

17 Bayesian neural networks

This chapter is coauthored with Andrew Wilson.

17.1 Introduction

Modern DNNs are usually trained using a (penalized) maximum likelihood objective to find a single setting of parameters. However, large flexible models like neural networks can represent many functions, corresponding to different parameter settings, which fit the training data well, yet generalize in different ways. Considering all of these different models together can lead to improved accuracy and uncertainty representation.

Bayesian inference provides a compelling mechanism to combine these different models together. To use a **Bayesian neural network (BNN)**, we start by specifying a prior distribution over model parameters $p(\boldsymbol{\theta})$, which induces a prior distribution over neural network functions. We then infer a posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$, from which we can compute the posterior predictive distribution using Bayesian model averaging:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \quad (17.1)$$

Early work in this space, much of which was pioneered by David MacKay and Radford Neal in the 1990s [Mac92b; Mac95; Nea95], focused on small models. In this chapter we focus on techniques that scale to newer, larger models. The resulting field is sometimes called **Bayesian deep learning**, to emphasize that we are applying Bayesian inference to “deep” models with many learnable layers. For more details, on this topic, see e.g., [PS17; Wil20; WI20; Jos+22; Kha20].

17.2 Priors for BNNs

To perform Bayesian inference for the parameters of a DNN, we need to specify a prior $p(\boldsymbol{\theta})$. [Nal18; WI20; For21] discusses the issue of prior selection at length. Here we briefly discuss common approaches, and considerations for prior specification in Bayesian deep learning.

In the case of an MLP, we will assume L hidden layers and a linear output:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}_L(\cdots \varphi(\mathbf{W}_2\varphi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_L \quad (17.2)$$

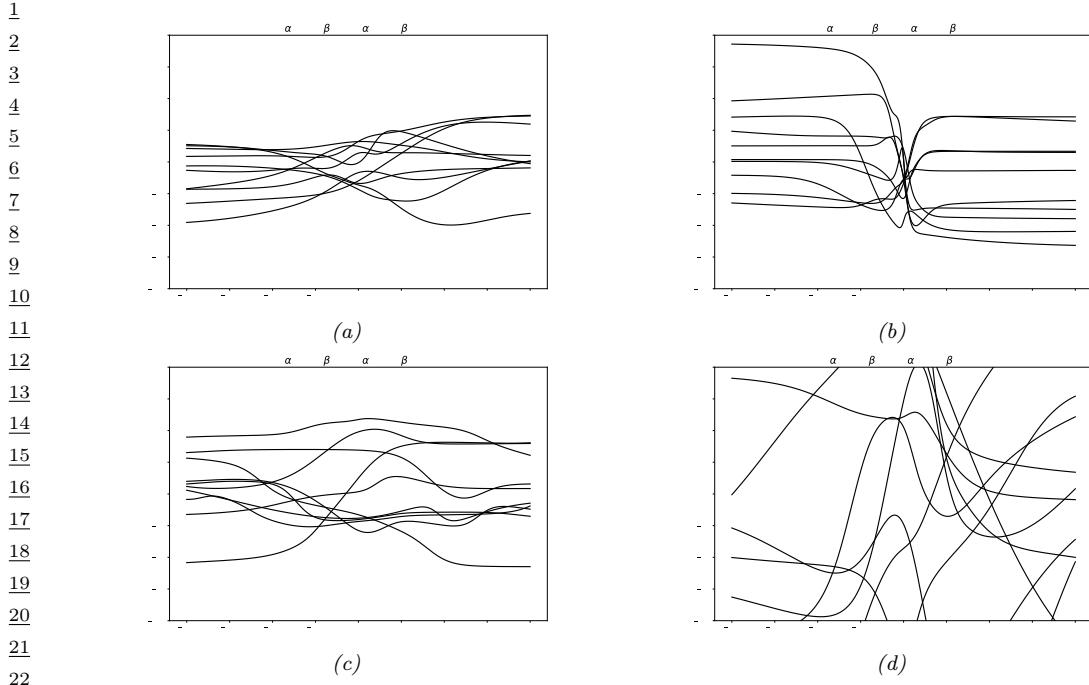


Figure 17.1: The effects of changing the hyperparameters on an MLP with one hidden layer. (a) Random functions sampled from a Gaussian prior with hyperparameters $\alpha_1 = 5$, $\beta_1 = 1$, $\alpha_2 = 1$, $\beta_2 = 1$. (b) Increasing α_1 by factor of 5. (c) Increasing β_1 by factor of 5. (d) Increasing α_2 by factor of 5. Generated by `mlpPriorsDemo2.py`.

27

28

17.2.1 Gaussian priors

30 The most common choice is to use a factored Gaussian prior:

$$32 \quad \mathbf{W}_\ell \sim \mathcal{N}(\mathbf{0}, \alpha_\ell^2 \mathbf{I}), \mathbf{b}_\ell \sim \mathcal{N}(\mathbf{0}, \beta_\ell^2 \mathbf{I}) \quad (17.3)$$

34 The **Xavier initialization** or **Glorot initialization**, named after the first author of [GB10], is to
35 set
36

$$37 \quad \alpha_\ell^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}} \quad (17.4)$$

40 where n_{in} is the fan-in of a node in level ℓ (number of weights coming into a neuron), and n_{out} is
41 the fan-out (number of weights going out of a neuron). **LeCun initialization**, named after Yann
42 LeCun, corresponds to using

$$43 \quad \alpha_\ell^2 = \frac{1}{n_{\text{in}}} \quad (17.5)$$

46

47

We can get a better understanding of these priors by considering the effect they have on the corresponding distribution over functions that they define. To help understand this correspondence, let us reparameterize the model as follows:

$$\mathbf{W}_\ell = \alpha_\ell \boldsymbol{\eta}_\ell, \quad \boldsymbol{\eta}_\ell \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad \mathbf{b}_\ell = \beta_\ell \boldsymbol{\epsilon}_\ell, \quad \boldsymbol{\epsilon}_\ell \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (17.6)$$

Hence every setting of the prior hyperparameters specifies the following random function:

$$f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \alpha_L \boldsymbol{\eta}_L (\cdots \varphi(\alpha_1 \boldsymbol{\eta}_1 \mathbf{x} + \beta_1 \boldsymbol{\epsilon}_1)) + \beta_L \boldsymbol{\epsilon}_L \quad (17.7)$$

To get a feeling for the effect of these hyperparameters, we can sample MLP parameters from this prior and plot the resulting random functions. We use a sigmoid nonlinearity, so $\varphi(a) = \sigma(a)$. We consider $L = 2$ layers, so \mathbf{W}_1 are the input-to-hidden weights, and \mathbf{W}_2 are the hidden-to-output weights. We assume the input and output are scalars, so we are generating random nonlinear mappings $f : \mathbb{R} \rightarrow \mathbb{R}$.

Figure 17.1(a) shows some sampled functions where $\alpha_1 = 5$, $\beta_1 = 1$, $\alpha_2 = 1$, $\beta_2 = 1$. In Figure 17.1(b) we increase α_1 ; this allows the first layer weights to get bigger, making the sigmoid-like shape of the functions steeper. In Figure 17.1(c), we increase β_1 ; this allows the first layer biases to get bigger, which allows the center of the sigmoid to shift left and right more, away from the origin. In Figure 17.1(d), we increase α_2 ; this allows the second layer linear weights to get bigger, making the functions more “wiggly” (greater sensitivity to change in the input, and hence larger dynamic range).

The above results are specific to the case of sigmoidal activation functions. ReLU units can behave differently. For example, [WI20, App. E] show that for MLPs with ReLU units, if we set $\beta_\ell = 0$, so the bias terms are all zero, the effect of changing α_ℓ is just to rescale the output. To see this, note that Equation (17.7) simplifies to

$$f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\beta} = \mathbf{0}) = \alpha_L \boldsymbol{\eta}_L (\cdots \varphi(\alpha_1 \boldsymbol{\eta}_1 \mathbf{x})) = \alpha_L \cdots \alpha_1 \boldsymbol{\eta}_L (\cdots \varphi(\boldsymbol{\eta}_1 \mathbf{x})) \quad (17.8)$$

$$= \alpha_L \cdots \alpha_1 f(\mathbf{x}; (\boldsymbol{\alpha} = \mathbf{1}, \boldsymbol{\beta} = \mathbf{0})) \quad (17.9)$$

where we used the fact that for ReLU, $\varphi(\alpha z) = \alpha \varphi(z)$ for any positive α , and $\varphi(\alpha z) = 0$ for any negative α (since the pre-activation $z \geq 0$). In general, it is the ratio of α and β that matters for determining what happens to input signals as they propagate forwards and backwards through a randomly initialized model; for details, see e.g., [Bah+20].

We see that initializing the model’s parameters at a particular random value is like sampling a point from this prior over functions. In the limit of infinitely wide neural networks, we can derive this prior distribution analytically: this is known as a **neural network Gaussian process**, and is explained in Section 18.7.

17.2.2 Sparsity-promoting priors

Although Gaussian priors are simple and widely used, they are not the only option. For some applications, it is useful to use **sparsity promoting priors**, such as the Laplace, which encourage most of the weights (or channels in a CNN) to be zero (c.f., Section 15.2.5). For details, see [Hoe+21].

17.2.3 Learning the prior

We have seen how different priors for the parameters correspond to different priors over functions. We could in principle set the hyperparameters (e.g., the α and β parameters of the Gaussian prior using grid search to optimize cross-validation loss. However, cross-validation can be slow, particularly if we allow different priors for each layer of the network, as our grid search will grow exponentially with the number of hyperparameters we wish to determine.

An alternative is to use gradient based methods to optimize the marginal likelihood

$$\log p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\beta}) = \int \log p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\alpha}, \boldsymbol{\beta})d\boldsymbol{\theta} \quad (17.10)$$

This approach is known as empirical Bayes (Section 3.6) or **evidence maximization**, since $\log p(\mathcal{D}|\boldsymbol{\alpha}, \boldsymbol{\beta})$ is also called the evidence [Mac92a; WS93; Mac99b]. This can give rise to sparse models, as we discussed in the context of automatic relevancy determination (Section 15.2.7). Unfortunately, computing the marginal likelihood is computationally difficult for large neural networks.

17.2.4 Priors in function space

Typically, the relationship between the prior distribution over parameters and the functions preferred by the prior is not transparent. In some cases, it can be possible to pick more informative priors, based on principles such as desired invariances that we want the function to satisfy (see e.g., [Nal18]). [FBW21] introduces *residual pathway priors*, providing a mechanism for encoding high level concepts into prior distributions, such as locality, independencies, and symmetries, without constraining model flexibility. A different approach to encoding interpretable priors over functions leverages kernel methods such as Gaussian processes (e.g., [Sun+19a]), as we discuss in Section 18.1.

17.2.5 Architectural priors

Ultimately, the prior that impacts generalization is the prior induced in *function space*, which arises from the combination of a prior over parameters $p(\boldsymbol{\theta})$ with the functional form of the model $f(\mathbf{x}; \boldsymbol{\theta})$. As argued in Wilson and Izmailov [WI20], there is strong evidence to suggest that the prior over functions implied by even generic $\mathcal{N}(\boldsymbol{\theta}|0, \alpha^2 I)$ priors over parameters in combination with a neural architecture, has many desirable statistical properties, especially with structured DNNs such as CNNs (Section 16.3.2).

For example, Ulyanov, Vedaldi, and Lempitsky [UVL18] showed that an untrained CNN with random parameters (sampled from a Gaussian) often works very well for low-level image processing tasks, such as image denoising, super-resolution and image inpainting. The resulting prior over functions has been called the **deep image prior**. Similarly, Pinto and Cox [PC12] showed that untrained CNNs with the right structure can do well at face recognition. Moreover, Zhang et al. [Zha+17] show that randomly initialized CNNs can process data to provide features that greatly improve the performance of other models, such as kernel methods. Wilson and Izmailov [WI20] additionally show that the prior over functions implied by a generic Gaussian distribution over parameters can induce a reasonable correlation function over images.

Moreover, Izmailov et al. [Izm+21b] show that using a high variance α^2 of a conventional Gaussian prior $\mathcal{N}(\boldsymbol{\theta}|0, \alpha^2 I)$ leads to good performance, and that the variance scale, or changing to different heavy-tailed logistic or mixture of Gaussian priors, has only a minor effect on the predictive

1 distribution. These results highlight the relative importance of architecture over parameter priors in
2 specifying a useful prior over functions.
3

4 Indeed, the architecture of a neural network encodes useful prior knowledge. A CNN architecture
5 encodes prior knowledge about translation invariance, due to its use of convolution, and hierarchical
6 structure, due to its use of multiple layers. Other forms of inductive bias are induced by different
7 architectures, such as RNNs. Thus we can think of the field of **neural architecture search**
8 (reviewed in [EMH19]) as a form of structural prior learning.
9

10 17.3 Likelihoods for BNNS

11 In this section, we discuss the likelihood model $p(y|\mathbf{x}, \boldsymbol{\theta})$ used by common Bayesian neural networks.
12 This is usually taken to be the same as any other classification or regression model. For example, we
13 may use
14

$$\small{15} \quad p(y|\mathbf{x}, \boldsymbol{\theta}) = \text{Cat}(y|\mathcal{S}(f(\mathbf{x}; \boldsymbol{\theta}))) \quad (17.11)$$

16 where $f(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^C$ returns the logits over the C class labels. This is the same as in multinomial
17 logistic regression (Section 15.3.2); the only difference is that f is a nonlinear function of $\boldsymbol{\theta}$.
18

19 However, in practice, it is often found (see e.g., [Zha+18a; Wen+20b; LST21]) that BNNS give
20 better predictive accuracy if the likelihood function is scaled by some power α . That is, instead of
21 targeting the posterior $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta})$, these methods target the **tempered posterior**,
22 $p_{\text{tempered}}(\boldsymbol{\theta}) \propto p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})^\alpha p(\boldsymbol{\theta})$. In log space, we have
23

$$\small{24} \quad \log p_{\text{tempered}}(\boldsymbol{\theta}) = \alpha \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{const} \quad (17.12)$$

25 This is also called an **α -posterior** or **power posterior** [Med+21].
26

27 Another common method is to target the **cold posterior**, $p_{\text{cold}}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})^{1/T}$, or, in log
28 space,
29

$$\small{30} \quad \log p_{\text{cold}}(\boldsymbol{\theta}) = \frac{1}{T} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) + \frac{1}{T} \log p(\boldsymbol{\theta}) + \text{const} \quad (17.13)$$

31 If $T < 1$, we say that the posterior is “cold”. Note that the only difference between cold and tempered
32 posteriors is that in the tempering case, the prior is not scaled. However, in the case of a Gaussian
33 prior, using the cold prior $p(\boldsymbol{\theta})^{1/T} = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \sigma^2 \mathbf{I})^{1/T}$ is the same as using the standard prior with a
34 different hyperparameter, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \tilde{\sigma}^2 \mathbf{I})$, where $\tilde{\sigma} = \sigma/T$. Thus both methods are effectively
35 the same, and just reweight the likelihood.
36

37 In BNNS for image classification problems, it has been found (see e.g., [Zha+18a; Wen+20b])
38 that using $\alpha > 1$ (or equivalently, $T < 1$) results in better predictive accuracy. There are several
39 explanations for this that have been proposed. In [Ait21], it is argued that the likelihood in
40 Equation (17.11) does not reflect the true probability of seeing a label when working with highly
41 curated datasets such as CIFAR or ImageNet. A more realistic of the data generating process
42 takes into account that there is selection bias, since examples are only included in the dataset if
43 multiple human annotators agreed on the label (c.f., [ASS20]). Thus these examples are likely
44 to be far from the decision boundary, which effectively upweights the likelihood, making it more
45 informative about the parameters. This naturally gives rise to a power likelihood term with $\alpha > 1$
46

¹ (see [Ait21] for details). In [Izm+21b], they argue that tempering is only needed when one uses data
² augmentation, presumably because data augmentation repeats the same example multiple times,
³ making it “untypically” informative in its likelihood.
⁴

⁵ However, in [GO17; KJD21; Med+21; ZN20], they prove that it is better to use $\alpha < 1$ in the case of
⁶ **model misspecification.** (See also ??.) From a pragmatic point of view, we can decide whether we
⁷ should use $\alpha < 1$ or $\alpha > 1$ or just $\alpha = 1$ using any model selection method, such as cross validation.
⁸

⁹ 17.4 Posteriors for BNNs

¹⁰ There are a large number of different approximate inference schemes that have been applied to
¹¹ Bayesian neural networks, with different strengths and limitations. In the sections below, we briefly
¹² describe some of these.
¹³

¹⁴ 17.4.1 Laplace approximation

¹⁵ In Section 7.4.3, we introduced the Laplace approximation, which computes a Gaussian approximation
¹⁶ to the posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, centered at the MAP estimate, $\boldsymbol{\theta}^*$, and whose precision is equal to the
¹⁷ Hessian of the negative log joint computed at the mode. The benefits of this approach are that it
¹⁸ is simple, and it can be used to derive a Bayesian estimate from a pretrained model. A detailed
¹⁹ explanation of the method in the context of DNNs can be found in [Dax+21]; here we just give a
²⁰ summary.
²¹

²² Let $\mathbf{f}(\mathbf{x}_n, \boldsymbol{\theta}) \in \mathbb{R}^C$ be the prediction function with C outputs, and $\boldsymbol{\theta} \in \mathbb{R}^P$ is the parameter vector.
²³ Let $\mathbf{r}(\mathbf{y}; \mathbf{f}) = \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})$ be the residual¹, and $\Lambda(\mathbf{y}; \mathbf{f}) = -\nabla_{\mathbf{f}}^2 \log p(\mathbf{y}|\mathbf{f})$ be the per-input noise
²⁴ term. In addition, let $\mathbf{J} \in \mathbb{R}^{C \times P}$ be the Jacobian, $[\mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x})]_{ci} = \frac{\partial f_c(\mathbf{x}, \boldsymbol{\theta})}{\partial \theta_i}$, and $\mathbf{H} \in \mathbb{R}^{C \times P \times P}$ be the
²⁵ Hessian, $[\mathbf{H}_{\boldsymbol{\theta}}(\mathbf{x})]_{cij} = \frac{\partial^2 f_c(\mathbf{x}, \boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}$. Then the gradient and Hessian of the log likelihood are given by the
²⁶ following [IKB21]:
²⁷

$$\nabla_{\boldsymbol{\theta}} \log p(\mathbf{y}|\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})) = \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x})^T \mathbf{r}(\mathbf{y}; \mathbf{f}) \quad (17.14)$$

$$\nabla_{\boldsymbol{\theta}}^2 \log p(\mathbf{y}|\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})) = \mathbf{H}_{\boldsymbol{\theta}}(\mathbf{x})^T \mathbf{r}(\mathbf{y}; \mathbf{f}) - \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x})^T \Lambda(\mathbf{y}; \mathbf{f}) \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{x}) \quad (17.15)$$

²⁸ Since the network Hessian \mathbf{H} is usually intractable to compute, it is usually dropped, leaving only the
²⁹ Jacobian term. This is called the **generalized Gauss–Newton** or **GGN** approximation [Sch02;
³⁰ Mar20]. The GGN approximation is guaranteed to be positive definite, whereas the original Hessian
³¹ in Equation (17.15) (since the objective is not convex). Furthermore, computing the Jacobian term
³² only takes $O(PC)$ time and space, where P is the number of parameters.
³³

³⁴ Putting it all together, for a Gaussian prior, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}_0, \mathbf{S}_0)$, the Laplace approximation
³⁵ becomes $p(\boldsymbol{\theta}|\mathcal{D}) \approx (\mathcal{N}|\boldsymbol{\theta}^*, \Sigma_{\text{GGN}})$, where
³⁶

$$\Sigma_{\text{GGN}}^{-1} = \sum_{n=1}^N \mathbf{J}_{\boldsymbol{\theta}^*}(\mathbf{x}_n)^T \Lambda(\mathbf{y}_n; \mathbf{f}_n) \mathbf{J}_{\boldsymbol{\theta}^*}(\mathbf{x}_n) + \mathbf{S}_0^{-1} \quad (17.16)$$

³⁷ Unfortunately inverting this matrix takes $O(P^3)$ time, so for models with many parameters, further
³⁸ approximations are usually used. The simplest is to use a diagonal approximation, which takes $O(P)$
³⁹

⁴⁰ 1. In the Gaussian case, this term becomes $\nabla_{\mathbf{f}} \|\mathbf{y} - \mathbf{f}\|^2 = 2\|\mathbf{y} - \mathbf{f}\|$, so can be interpreted as a residual error.
⁴¹

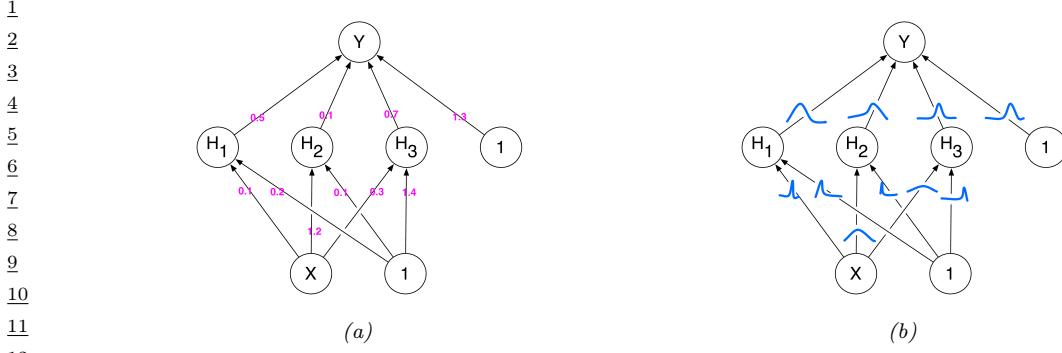


Figure 17.2: Illustration of an MLP with (a) point estimate for each weight, (b) a marginal distribution for each weight, corresponding to a fully factored posterior approximation. From Figure 1 of [Blu+15]. Used with kind permission of Charles Blundell.

time and space. A more sophisticated approach is presented in [RBB18a], which leverages the **KFAC** (Kronecker FActored Curvature) approximation of [MG15], which approximates the covariance of each layer using a kronecker product. See Section 6.4.4 for details.

A limitation of the Laplace approximation is that the posterior covariance is derived from the Hessian evaluated at the MAP parameters. This means Laplace forms a highly *local* approximation: even if the non-Gaussian posterior could be well-described by a Gaussian distribution, the Gaussian distribution *formed using Laplace* only captures the local characteristics of the posterior at the MAP parameters — and may therefore suffer badly from local optima, providing overly compact or diffuse representations. In addition, the curvature information is only used after the model has been estimated, and not during the model optimization process. By contrast, variational inference (Section 17.4.2) can provide more accurate approximations for comparable cost.

17.4.2 Variational inference

In fixed-form variational inference (Section 10.3), we choose a distribution for the posterior approximation $q(\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, and minimize $D_{\text{KL}}(q||p)$, with respect to $\boldsymbol{\theta}$. We often choose a Gaussian approximate posterior, $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, which lets us use the reparameterization trick to create a low variance estimator of the gradient of the ELBO (see Section 10.3.3). Despite the use of a Gaussian, the parameters that minimize the KL objective are often different what we would find with the Laplace approximation (Section 17.4.1).

Variational methods for neural networks date back to at least Hinton and Camp [HC93]. In deep learning, [Gra11] revisited variational methods, using a Gaussian approximation with a diagonal covariance matrix. This approximates the distribution of every parameter in the model by a univariate Gaussian, where the mean is the point estimate, and the variance captures the uncertainty, as shown in Figure 17.2. This approach was improved further in [Blu+15], who used the reparameterization trick to compute lower variance estimates of the ELBO; they called their method **Bayes by backprop** (**BBB**).

In [Blu+15], they used a diagonal Gaussian posterior and vanilla SGD. In [Osa+19a], they used the **variational online Gauss-Newton (VOGN)** method of [Kha+18], for improved scalability.

¹ VOGN is a noisy version of natural gradient descent, where the extra noise emulates the effect
² of variational inference. In [Mis+18], they replaced the diagonal approximation with a low-rank
³ plus diagonal approximation, and used VOGN for fitting. In [Tra+20b], they use a rank-one plus
⁴ diagonal approximation known as **NAGVAC** (see Section 10.3.4.2). In this case, there are only 3
⁵ times as many parameters as when computing a point estimate (for the variational mean, variance,
⁶ and rank-one vector), making the approach very scalable. In addition, in this case it is possible to
⁷ analytically compute the natural gradient, which speeds up model fitting. Many other variational
⁸ methods have also been proposed (see e.g., [LW16; Zha+18a; Wu+19a; HHK19]).

⁹ Note that VI often results in unnecessary weights being set to their prior value, in order to avoid
¹⁰ the KL penalty. This can be useful for inducing **sparsity** in the model [LUW17; MAV17]. However,
¹¹ it can also result in overconfidence, due to the **variational over-pruning** effect (Section 22.4), as
¹² discussed in [TT17].

¹³

¹⁴ 17.4.3 Expectation propagation

¹⁵ Expectation propagation is similar to variational inference, except it locally optimizes $D_{\text{KL}}(p\|q)$
¹⁶ instead of $D_{\text{KL}}(q\|p)$, where p is the exact posterior and q is the approximate posterior. For details,
¹⁷ see Section 10.7.

¹⁸ In [HLA15b], they show how to apply EP to fitting BNNs; they called their method **probabilistic**
¹⁹ **backpropagation** or **PBP**. They approximate every parameter in the model by a Gaussian factor,
²⁰ as in Figure 17.2. This work was extended to the classification setting in [BPK18]. One of the major
²¹ technical challenges is analytically propagating Gaussian densities through various nonlinear layers,
²² such as ReLU and softmax, without resorting to sampling. We discuss this in Section 17.6.2.

²³ In [HL+16a], they discussed black-box α -divergence minimization, which generalizes both VI
²⁴ ($\alpha = 0$) and EP ($\alpha = 1$). However, black-box techniques tend to be less efficient.

²⁵

²⁶ 17.4.4 Last layer methods

²⁷ Another scalable approximation is to only “be Bayesian” about the weights in the final layer, and to
²⁸ use MAP estimates for all the other parameters. This is called the **neural-linear** approximation
²⁹ (see Section 17.4.4). In [KHH20] they show this can reduce overconfidence in predictions for inputs
³⁰ that are far from the training data. However, this approach ignores uncertainty introduced by the
³¹ earlier feature extraction layers, where most of the parameters reside.

³² Earlier work on deep kernel learning [Wil+16b; Wil+16a] replaces the final linear layer with
³³ a Gaussian process. Building on this work, the **SNGP** (spectrally normalized Gaussian process)
³⁴ method of [Liu+20d] additionally constrains the feature extraction layers to be “distance preserving”,
³⁵ so that two inputs that are far apart in input space remain far apart after many layers of feature
³⁶ extraction. (This constraint is enforced using spectral normalization of the weights to bound the
³⁷ Lipschitz constant of the feature extractor.) The overall approach ensures that information that is
³⁸ relevant for computing the confidence of a prediction, but which might be irrelevant to computing
³⁹ the label of a prediction, is not lost. This can help performance in tasks such as out-of-distribution
⁴⁰ detection (Section 20.4.2).

⁴¹

⁴² 17.4.5 Dropout

⁴³ ⁴⁴ **Monte Carlo dropout** [GG16; KG17] is a very simple, and therefore popular, method for approx-
⁴⁵

imating the Bayesian predictive distribution. The idea is to add dropout layers to the model, as described in Section 16.2.6, and then train in the usual way.

At test-time, we drop out each hidden unit by sampling from a Bernoulli(p) distribution; we repeat this procedure S times, to create S distinct models. We then create an equally weighted average of the predictive distributions for each of these models. Although it has been argued that this process approximates variational inference [GG16], this is only true under a degenerate posterior approximation, corresponding to a mixture of two delta functions, one at 0 (for dropped out nodes) and one at the MLE. This posterior will not converge to the true posterior (which is a delta function at the MLE) even as the training set size goes to infinity, since we are always dropping out hidden nodes with a constant probability p . Thus the procedure is not “truly Bayesian” [Osb16; HGMG18; NHLS19; LF+21]. Fortunately this pathology can be fixed if the noise rate is optimized [GHK17].

17.4.6 MCMC methods

Markov-chain Monte Carlo methods (Section 12.2) are particularly suitable for exploring the sophisticated multi-modal posteriors in Bayesian neural networks. Without the unimodal or strong parametric constraints of standard deterministic approximations, such as variational inference, Radford Neal’s early work [Nea96] established Hamiltonian Monte Carlo (Section 12.5) as a gold standard for Bayesian inference in neural networks [Nea+11; Izm+21b; CJ21].

However, a significant limitation of standard MCMC procedures, including HMC, is that they require access to the full training set at each step. Stochastic gradient MCMC methods operate instead using mini-batches of data, offering a scalable alternative [Wel11; CFG14b; Zha+20d]. See Section 12.7 for details.

17.4.7 Methods based on the SGD trajectory

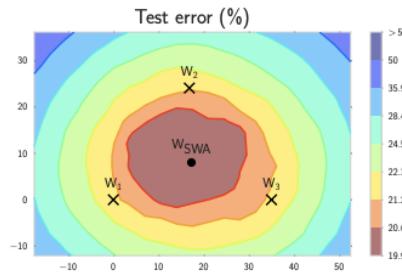
In [MHB17], it was shown that, under some assumptions, the iterates produced by stochastic gradient descent, when run at a fixed learning rate, correspond to samples from a Gaussian approximation to the posterior centered at a local mode, $p(\boldsymbol{\theta}|\mathcal{D}) \approx \mathcal{N}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}, \Sigma)$. We can therefore use SGD to generate approximate posterior samples, similarly to SG-MCMC methods, except without explicit gradient noise, and the learning rate is held constant.

In [Izm+18], they noted that these SGD solutions (with fixed learning rate) surround the periphery of points of good generalization, as shown in Figure 17.3. This is in part because SGD does not converge to a local optimum unless the learning rate is annealed to 0. They therefore proposed to compute the average of several SGD samples, each one collected after a certain interval (e.g., one epoch of training), to get $\bar{\boldsymbol{\theta}} = \frac{1}{S} \sum_{s=1}^S \boldsymbol{\theta}_s$. They call this **stochastic weight averaging (SWA)**. They showed that the resulting point tends to correspond to a broader local minimum than the SGD solutions (c.f., Figure 17.7), resulting in better generalization performance.

The SWA approach is related to Polyak-Ruppert averaging, which is often used in convex optimization. The difference is that Polyak-Ruppert typically assumes the learning rate decays to zero, and uses an exponential moving average (EMA) of iterates, rather than an equal average; Polyak-Ruppert averaging is mainly used to reduce variance in the SGD estimate, rather than as a method to find points of better generalization.

The SWA approach is also related to **snapshot ensembles** [Hua+17a], and **fast geometric ensembles** [Gar+18c]; these methods save the parameters $\boldsymbol{\theta}_s$ after increasing and decreasing

1
2
3
4
5
6
7
8
9
10



11 *Figure 17.3: Illustration of stochastic weight averaging (SWA). The three crosses represent different SGD*
 12 *solutions. The star in the middle is the average of these parameter values. From Figure 1 of [Izm+18]. Used*
 13 *with kind permission of Andrew Wilson.*

14

15

16 the learning rate multiple times in a cyclical fashion, and then average the predictions using
 17 $p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_s)$, rather than averaging the parameters and predicting with a single
 18 model (which is faster). Moreover, by finding a flat region, representing a “center or mass” in the
 19 posterior, SWA can be seen as approximating the Bayesian model average in Equation 17.1 with a
 20 single model.

21 In [Mad+19], they proposed to fit a Gaussian distribution to the set of samples produced by SGD
 22 near a local mode. They use the SWA solution as the mean of the Gaussian. For the covariance
 23 matrix, they use a low-rank plus diagonal approximation of the form $p(\boldsymbol{\theta}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\theta}|\bar{\boldsymbol{\theta}}, \boldsymbol{\Sigma})$, where
 24 $\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_{\text{diag}} + \boldsymbol{\Sigma}_{\text{lr}})/2$, $\boldsymbol{\Sigma}_{\text{diag}} = \text{diag}(\bar{\boldsymbol{\theta}}^2 - (\bar{\boldsymbol{\theta}})^2)$, $\bar{\boldsymbol{\theta}} = \frac{1}{S} \sum_{s=1}^S \boldsymbol{\theta}_s$, $\bar{\boldsymbol{\theta}}^2 = \frac{1}{S} \sum_{s=1}^S \boldsymbol{\theta}_s^2$, and $\boldsymbol{\Sigma}_{\text{lr}} = \frac{1}{S} \boldsymbol{\Delta} \boldsymbol{\Delta}^\top$
 25 is the sample covariance matrix of the last K samples of $\boldsymbol{\Delta}_i = (\boldsymbol{\theta}_i - \bar{\boldsymbol{\theta}}_i)$, where $\bar{\boldsymbol{\theta}}_i$ is the running
 26 average of the parameters from the first i samples. They call this method **SWAG**, which stands for
 27 “stochastic weight averaging with Gaussian posterior”. This can be used to generate an arbitrary
 28 number of posterior samples at prediction time. They show that SWAG scales to large residual
 29 networks with millions of parameters, and large datasets such as ImageNet, with improved accuracy
 30 and calibration over conventional SGD training, and no additional training overhead.

31

32 17.4.8 Deep ensembles

33

34 Many conventional approximate inference methods focus on approximating the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ in a
 35 local neighborhood around one of the posterior modes. While this is often not a major limitation in
 36 classical machine learning, modern deep neural networks have highly multi-modal posteriors, with
 37 parameters in different modes giving rise to very different functions. On the other hand, the functions
 38 in a neighborhood of a single mode may make fairly similar predictions. So using such a local
 39 approximation to compute the posterior predictive will underestimate uncertainty and generalize
 40 more poorly.

41 A simple alternative method is to train multiple models, and then to approximate the posterior
 42 using an equally weighted mixture of delta functions,

43

$$44 \quad p(\boldsymbol{\theta}|\mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^M \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m) \quad (17.17)$$

45

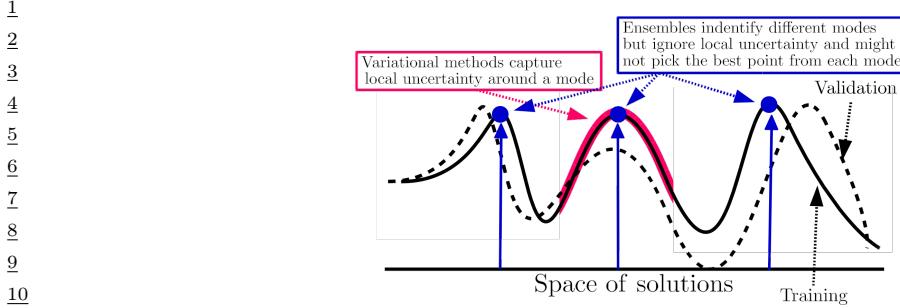


Figure 17.4: Cartoon illustration of the NLL as it varies across the parameter space. Subspace methods (red) model the local neighborhood around a local mode, whereas ensemble methods (blue) approximate the posterior using a set of distinct modes. From Figure 1 of [FHL19]. Used with kind permission of Balaji Lakshminarayanan.

where M is the number of models, and $\hat{\theta}_m$ is the MAP estimate for model m . See Figure 17.4 for a sketch. This approach is called **deep ensembles** [LPB17; FHL19].

The models can differ in terms of their random seed used for initialization [LPB17], or hyperparameters [Wen+20c], or architecture [Zai+20], or all of the above. Each local optima corresponds to a distinct prediction function, so combining these is more effective than combining multiple samples from the same basin of attraction, especially in the presence of dataset shift [Ova+19].

We can further improve on this approach by fitting a Gaussian to each local mode using the SWAG method from Section 17.4.7 to get a mixture of Gaussians approximation:

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^M \mathcal{N}(\boldsymbol{\theta}|\hat{\theta}_m, \Sigma_m) \quad (17.18)$$

This approach is known as **MultiSWAG** [WI20]. MultiSWAG performs a Bayesian model average both across multiple basins of attraction, like deep ensembles, but also within each basin, and provides an easy way to generate an arbitrary number of posterior samples, $S > M$, in an any-time fashion.

17.4.8.1 Deep ensembles as approximate Bayesian inference

The posterior predictive distribution for a Bayesian neural network cannot be expressed in closed form. Therefore all Bayesian inference approaches in deep learning are approximate. In this context, all approximate inference procedures fall onto a spectrum, representing how closely they approximate the true posterior predictive distribution. On this spectrum, deep ensembles are often closer to the Bayesian ideal than many canonical approximate Bayesian inference procedures, such as the Laplace approximation, in deep learning. By also marginalizing within basins, MultiSWAG moves deep ensembles further towards the Bayesian predictive distribution.

In short, deep ensembles can provide better approximations to a Bayesian model average than a single basin marginalization approach, because point masses from different basins of attraction represent greater functional diversity than standard Bayesian approaches which sample within a single basin. This result is illustrated in Figure 17.5.

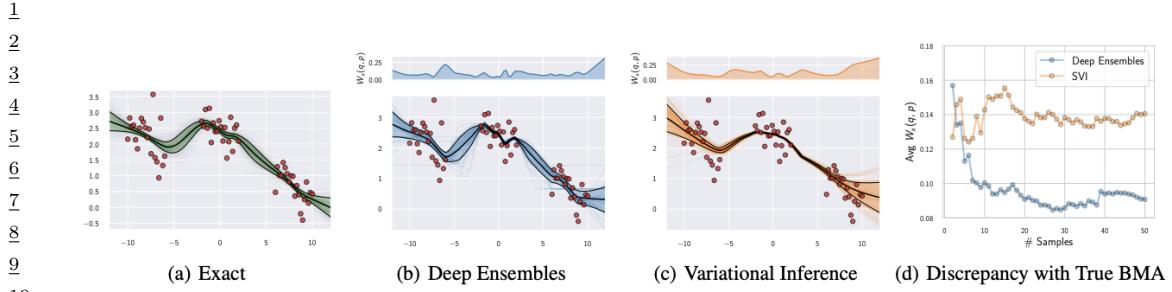


Figure 17.5: Comparison of approximate Bayesian inference methods for a 1d regression problem. (a) The “true” predictive distribution obtained by combining 200 HMC chains. (b) Deep ensembles predictive distribution using 50 independently trained networks. (c) Predictive distribution for factorized variational inference (VI). (d) Convergence of the predictive distributions for deep ensembles and variational inference as a function of the number of samples in terms of the average Wasserstein distance between the marginals in the range of input positions. The multi-basin deep ensembles approach provides a better approximation of the Bayesian predictive distribution than the conventional single-basin VI approach, which is overconfident between data clusters. The top panels show the Wasserstein distance between the true predictive distribution and the deep ensemble and VI approximations, as a function of inputs x . From Figure 4 of [WI20]. Used with kind permission of Andrew Wilson.

Note that deep ensembles is slightly different to a classical ensemble method, such as bagging and random forests, which obtains diversity of its predictors by training them on different subsets of the data (created using bootstrap resampling), or on different features. This data perturbation is necessary to get diversity when the base learner is a convex problem (such as a linear model, or shallow decision tree). In the deep ensemble approach, every model is trained on the same data, and the same features. The diversity arises due to different starting parameters, different random seeds, and SGD noise, which induces different solutions due to the nonconvex loss.

If we use weighted combinations of the models, $p(\boldsymbol{\theta}|\mathcal{D}) = \sum_{m=1}^M p(m|\mathcal{D})p(\boldsymbol{\theta}|m, \mathcal{D})$, where $p(m|\mathcal{D})$ is the marginal likelihood of model m , then, in the large sample limit, this mixture will concentrate on the MAP model, so only one component will be selected. This is because Bayes model averaging is not the same as model ensembling [Min00b], which fixes or optimizes the mixing weights; this can enlarge the expressive power of the posterior predictive distribution compared to BMA [OCM21].

We can also make the mixing weights be conditional on the inputs:

$$p(y|\boldsymbol{x}, \mathcal{D}) = \sum_m w_m(\boldsymbol{x})p(y|\boldsymbol{x}, \boldsymbol{\theta}_m) \quad (17.19)$$

If we constrain the weights to be non-zero and sum to one, this is called a **mixture of experts**. However, if we allow a general positive weighted combination, the approach is called **stacking** [Wol92; Bre96; Yao+18a; CAII20]. In stacking, the weights $w_m(\boldsymbol{x})$ are usually estimated on hold-out data, to make the method more robust to model misspecification.

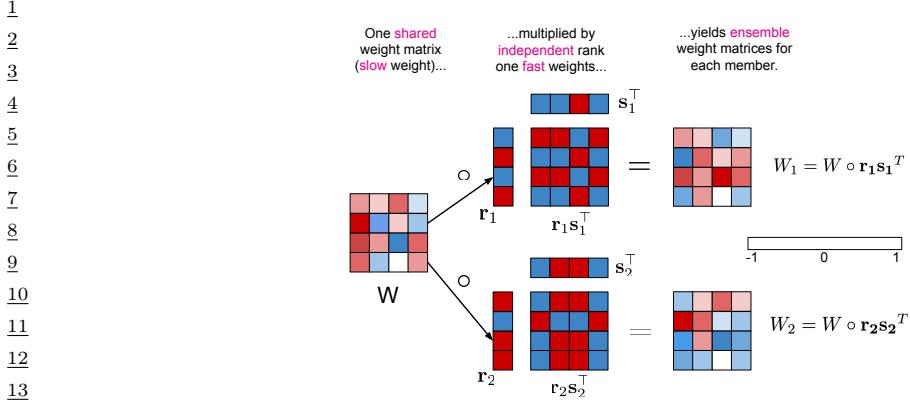


Figure 17.6: Illustration of batch ensemble with 2 ensemble members. From Figure 2 of [WTB20]. Used with kind permission of Paul Vicol.

17.4.8.2 Batch ensemble

Deep ensembles require M times more memory and time than a single model. One way to reduce the memory cost is to share most of the parameters — which we call **slow weights**, \mathbf{W} — and then let each ensemble member m estimate its own local perturbation, which we will call **fast weights**, \mathbf{F}_m . We then define $\mathbf{W}_m = \mathbf{W} \odot \mathbf{F}_m$. For efficiency, we can define \mathbf{F}_m to be a rank-one matrix, $\mathbf{F}_m = \mathbf{s}_m \mathbf{r}_m^T$, as illustrated in Figure 17.6. This is called **batch ensemble** [WTB20].

It is clear that the memory overhead is very small compared to naive ensembles, since we just need to store $2M$ vectors (\mathbf{s}_m^l and \mathbf{r}_m^l) for every layer, which is negligible compared to the quadratic cost of storing the shared weight matrix \mathbf{W}^l .

In addition to memory savings, batch ensemble can reduce the inference time by a constant factor by leveraging within-device parallelism. To see this, consider the output of one layer using ensemble m on example n :

$$y_n^m = \varphi(\mathbf{W}_m^T \mathbf{x}_n) = \varphi((\mathbf{W} \odot \mathbf{s}_m \mathbf{r}_m^T)^T \mathbf{x}_n) = \varphi((\mathbf{W}^T (\mathbf{x}_n \odot \mathbf{s}_m) \odot \mathbf{r}_m)) \quad (17.20)$$

We can vectorize this for a minibatch of inputs \mathbf{X} by repeating \mathbf{r}_m and \mathbf{s}_m into matrices, to get

$$\mathbf{Y}_m = \varphi(((\mathbf{X} \odot \mathbf{S}_m) \mathbf{W}) \odot \mathbf{R}_m) \quad (17.21)$$

This applies the same ensemble parameters m to every example in the minibatch of size B . To achieve diversity during training, we can divide the minibatch into M sub-batches, and use sub-batch m to train \mathbf{W}_m . (Note that this reduces the batch size for training each ensemble to B/M .) At test time, when we want to average over M models, we can replicate each input M times, leading to a batch size of BM .

In [WTB20], they show that this method outperforms MC dropout at negligible extra memory cost. However, the best combination was to combine batch ensemble with MC dropout; in some cases, this approached the performance of naive ensembles.

17.4.9 Approximating the posterior predictive distribution

Once we have approximated the parameter posterior, $q(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|\mathcal{D})$, we can use it to approximate the posterior predictive distribution:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int q(\boldsymbol{\theta}) p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) d\boldsymbol{\theta} \quad (17.22)$$

We usually approximate this integral using Monte Carlo:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}|\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}^s)) \quad (17.23)$$

where $\boldsymbol{\theta}^s \sim q(\boldsymbol{\theta})$. We discuss some extensions of this approach below.

17.4.9.1 A linearized approximation

In [IKB21] they point out that samples from an approximate posterior, $q(\boldsymbol{\theta})$, can result in bad predictions when plugged into the model if the posterior puts probability density “in the wrong places”. This is because $\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$ is a highly nonlinear function of $\boldsymbol{\theta}$ that might behave quite differently when $\boldsymbol{\theta}$ is far from the MAP estimate on which $q(\boldsymbol{\theta})$ is centered. To avoid this problem, they propose to replace $\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$ with a linear approximation centered at the MAP estimate $\boldsymbol{\theta}^*$:

$$\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta}^*) + \mathbf{J}_{\boldsymbol{\theta}^*}(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \quad (17.24)$$

Such a model is well behaved around $\boldsymbol{\theta}^*$, and so the approximation

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(\mathbf{y}|\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta}^s)) \quad (17.25)$$

often works better than Equation (17.23).

Note that $\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta})$ is a linear function of the parameters $\boldsymbol{\theta}$, but a nonlinear function of the inputs \mathbf{x} . Thus $p(\mathbf{y}|\mathbf{f}_{\text{lin}}^{\boldsymbol{\theta}^*}(\mathbf{x}, \boldsymbol{\theta}))$ is a generalized linear model (Section 15.1), so [IKB21] call this approximation the **GLM predictive distribution**.

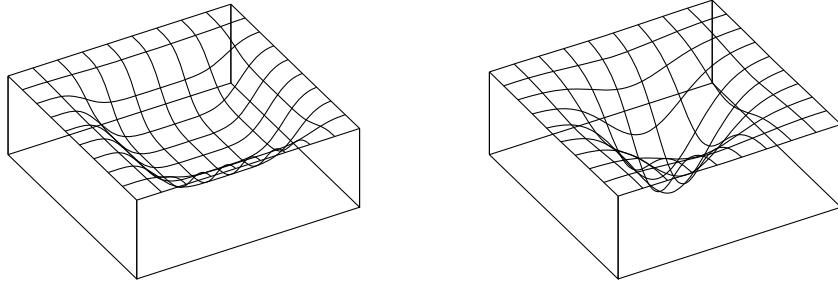
17.4.9.2 Distillation

The MC approximation to the posterior predictive is S times slower than a standard, deterministic plug-in approximation. One way to speed this up is to use **distillation** to approximate the semi-parametric “teacher” model p_t from Equation (17.23) by a parametric “student” model p_s by minimizing $\mathbb{E}[D_{\text{KL}}(p_t(\mathbf{y}|\mathbf{x}) \| p_s(\mathbf{y}|\mathbf{x}))]$ wrt p_s . This approach was first proposed in [HVD14], who called the technique “**dark knowledge**”, because the teacher has “hidden” information in its predictive probabilities (logits) than is not apparent in the raw one-hot labels.

In [Kor+15], this idea was used to distill the predictions from a teacher whose parameter posterior was computed using HMC; this is called “**Bayesian dark knowledge**”. A similar idea was used in [BPK16], who distilled the predictive distribution derived from MC dropout (Section 17.4.5).

Since the parametric student is typically less flexible than the semi-parametric teacher, it may be overconfident, and lack diversity in its predictions. To avoid this overconfidence, it is safer to make the student be a mixture distribution c.f., [SG05]. See also [Tra+20a].

1
2
3
4
5
6
7
8
9
10



11
12
13

Figure 17.7: Flat vs sharp minima. From Figures 1 and 2 of [HS97]. Used with kind permission of Jürgen Schmidhuber.

14
15

16 17.5 Generalization in Bayesian deep learning

17
18
19
20

In this section, we discuss why “being Bayesian” can improve predictive accuracy and generalization performance.

21
22

17.5.1 Sharp vs flat minima

23
24
25
26
27
28
29
30
31
32
33
34

Some optimization methods (in particular, second-order batch methods) are able to find “needles in haystacks”, corresponding to narrow but deep “holes” in the loss landscape, corresponding to parameter settings with very low loss. These are known as **sharp minima**, see Figure 17.7(right). From the point of view of minimizing the empirical loss, the optimizer has done a good job. However, such solutions generally correspond to a model that has overfit the data. It is better to find points that correspond to **flat minima**, as shown in Figure 17.7(left); such solutions are more robust and generalize better. To see why, note that flat minima correspond to regions in parameter space where there is a lot of posterior uncertainty, and hence samples from this region are less able to precisely memorize irrelevant details about the training set [AS17]. Put another way, the description length for sharp minima is large, meaning you need to use many bits of precision to specify the exact location in parameter space to avoid incurring large loss, whereas the description length for flat minima is less, resulting in better generalization [Mac03].

35
36
37
38
39
40
41
42
43
44

SGD often finds such flat minima by virtue of the addition of noise, which prevents it from “entering” narrow regions of the loss landscape (see Section 12.5.7). In addition, in higher dimensional spaces, flat regions occupy a much greater volume, and are thus much more easily discoverable by optimization procedures. More precisely, the analysis in [SL18] shows that the probability of entering any given basin of attraction \mathcal{A} around a minimum is given by $p_{SGD}(\boldsymbol{\theta} \in \mathcal{A}) \propto \int_{\mathcal{A}} e^{-\mathcal{L}(\boldsymbol{\theta})} d\boldsymbol{\theta}$. Note that this is integrating over the volume of space corresponding to \mathcal{A} , and hence is proportional to the model evidence (marginal likelihood) for that region, as explained in Section 3.7.1. Since the evidence is parameterization invariant (since we marginalize out the parameters), this means that SGD will avoid regions that have low evidence (corresponding to sharp minima) regardless of how we parameterize the model (contrary to the claims in [Din+17]).

45
46
47

In fact, several papers have shown that we can view SGD as approximately sampling from the Bayesian posterior (see e.g., [MHB17; SL18; CS18]). The SWA [Izm+18] method can be seen as

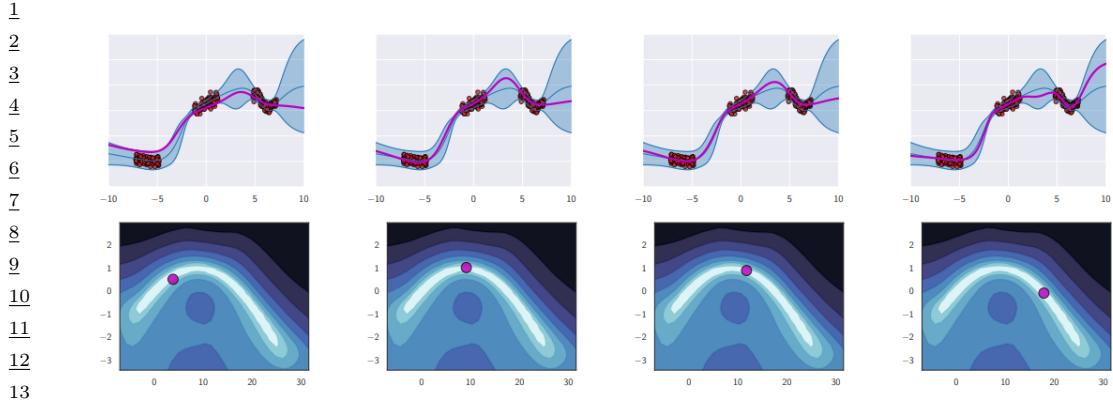


Figure 17.8: Diversity of high performing functions sampled from the posterior. Top row: we show predictions on the 1d input domain for 4 different functions. We see that they extrapolate in different ways outside of the support of the data. Bottom row: we show a 2d subspace spanning two distinct modes (MAP estimates), and connected by a low-loss curved path computed as in [Gar+18c]. From Figure 8 of [WI20]. Used with kind permission of Andrew Wilson.

19

20

21

22 finding a center of mass in the posterior, finding flatter solutions than SGD that generalize better
23 than SGD.

24 If we must use a single solution, a flat one will help us better approximate the Bayesian model
25 average in the integral of Equation (17.1). However, by attempting to perform a more complete
26 Bayesian model average, we will select for flatness without having to deal with the messiness of
27 having to worry about flatness definitions, or the effects of reparametrization, or unknown implicit
28 regularization, as the model average will automatically weight regions with the greatest volume.

29 Moreover, we can do much better than a *single* flat solution. Indeed, there are many low-loss
30 solutions, which provide complementary explanations of the data. In [Gar+18c], they showed that
31 two independently trained SGD solutions can be connected by a curve in a subspace, along which the
32 training loss remains near-zero, known as **mode connectivity**. Despite having the same training
33 loss, these different parameter settings give rise to very different functions, as illustrated in Figure 17.8,
34 where we show predictions on a 1d regression problem coming from different points in parameter
35 space obtained by interpolating along a mode connecting curve between two distinct MAP estimates.
36 Using a Bayesian model average, we can combine these functions together to provide much better
37 performance over a single flat solution [Izm+19].

38 Recently, it has been discovered [Ben+21] that there are in fact large multidimensional simplexes
39 of low loss solutions, which can be combined together for significantly improved performance. These
40 results further motivate the Bayesian approach (Equation (17.1)), where we perform a posterior
41 weighted model average.

42

43 17.5.2 Effective dimensionality of a model

44

45 Modern DNNs have millions of parameters, but these parameters are often not well-determined
46 by the data, i.e., there can be a lot of posterior uncertainty. By averaging over the posterior, we
47

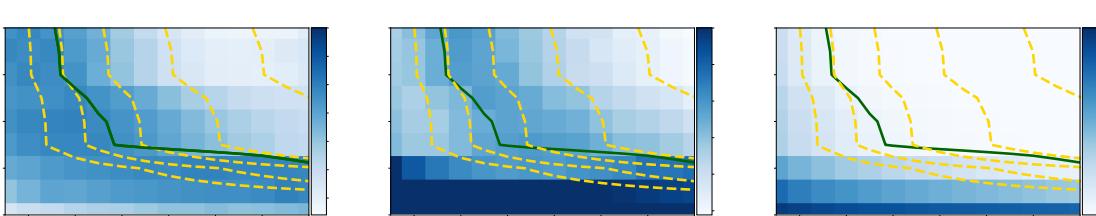


Figure 17.9: Left: Effective dimensionality as a function of model width and depth for a CNN on CIFAR-100. Center: Test loss as a function of model width and depth. Right: Train loss as a function of model width and depth. Yellow level curves represent equal parameter counts ($1e5, 2e5, 4e5, 1.6e6$). The green curve separates models with near-zero training loss. Effective dimensionality serves as a good proxy for generalization for models with low train loss. We see wide but shallow models overfit, providing low train loss, but high test loss and high effective dimensionality. For models with the same train loss, lower effective dimensionality can be viewed as a better compression of the data at the same fidelity. Thus depth provides a mechanism for compression, which leads to better generalization. From Figure 2 of [MBW20]. Used with kind permission of Andrew Wilson.

reduce the chance of overfitting, because we do not use “degrees of freedom” that are not needed or warranted.

In [MBW20], they revisit the **effective dimensionality** [Mac92b] of a model, shedding light on overparametrization, double descent, and width-depth trade-offs. The effective dimensionality is defined as

$$N_{\text{eff}}(\mathbf{H}, c) = \sum_{i=1}^k \frac{\lambda_i}{\lambda_i + c}, \quad (17.26)$$

where λ_i are the eigenvalues of the Hessian matrix \mathbf{H} computed at a local mode, and $c > 0$ is a regularization parameter. Intuitively, the effective dimension counts the number of well-determined parameters. A “flat minimum” will have many directions in parameter space that are not well-determined, and hence will have low effective dimensionality. This means that we can perform Bayesian inference in a low dimensional subspace [Izm+19]: Since there is functional homogeneity in all directions but those defining the effective dimension, neural networks can be significantly compressed.

This compression perspective can also be used to understand why the effective dimension can be a good proxy for generalization. If two models have similar training loss, but one has lower effective dimension, then it is providing a better compression for the data at the same fidelity. In Figure 17.9 we show that for CNNs with low training loss (above the green partition), the effective dimensionality closely tracks generalization performance. We also see that the number of parameters alone is not a strong determinant of generalization. Indeed, models with more parameters can have a lower number of effective parameters. We also see that wide but shallow models overfit, while depth helps provide lower effective dimensionality, leading to a better compression of the data. It is depth that makes modern neural networks distinctive, providing hierarchical inductive biases making it possible to discover more regularity in the data.

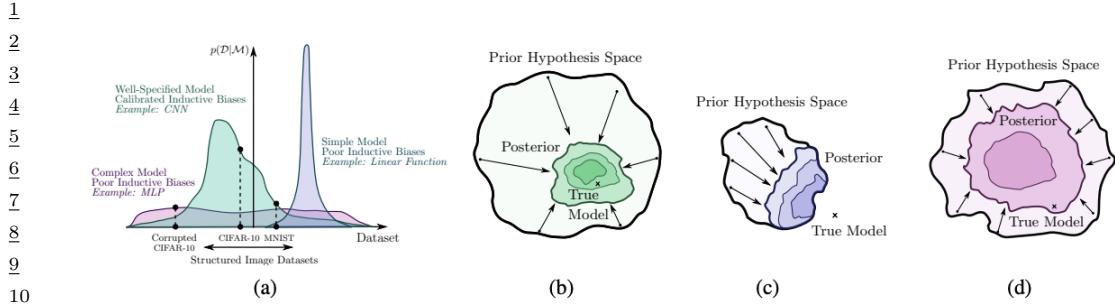


Figure 17.10: Illustration of the behavior of different kinds of model families and the prior distribution they induce over datasets. (a) The purple model is a simple linear model that has small support, and can only represent a few kinds of datasets. The pink model is an unstructured MLP: this has support over a large range of datasets but with a fairly uninformative (broad) prior. Finally the green model is a CNN; this has support over a large range of datasets but with a fairly uninformative (broad) prior. (b) The posterior for the green model (CNN) rapidly collapses to the true model. (c) The posterior for the purple model (linear) also rapidly collapses, but to a solution which cannot represent the true model. (d) The posterior for the pink model (MLP) collapses very slowly (as a function of dataset size). From Figure 2 of [WI20]. Used with kind permission of Andrew Wilson.

20
21
22
23

17.5.3 The hypothesis space of DNNs

Zhang et al. [Zha+17] showed that CNNs can fit CIFAR-10 images with random labels with zero training error, but can still generalize well on the noise-free test set. It has been claimed that this result contradicts a classical understanding of generalization, because it shows that neural networks are capable of significantly overfitting the data, but can still generalize well on structured inputs.

We can resolve this paradox by taking a Bayesian perspective. In particular, we know that modern CNNs are very flexible, so they can fit almost pattern (since they are in fact universal approximators). However, their architecture encodes a prior over what kinds of patterns they expect to see in the data (see Section 17.2.5). Image datasets with random labels *can* be represented by this function class, but such solutions receive very low marginal likelihood, since they strongly violate the prior assumptions [WI20]. By contrast, image datasets where the output labels are consistent with patterns in the input get much higher marginal likelihood.

This phenomenon is not unique to DNNs. For example, it also occurs with Gaussian processes (Chapter 18). Such models are also universal approximators, but they allocate most of their probability mass to a small range of solutions (depending on the chosen kernel). They can also fit image datasets with random labels, but such data receives a low marginal likelihood [WI20].

In general, we can distinguish the support of a model, i.e., the set of functions it can represent, from the distribution over that support, i.e., the inductive bias which leads it to prefer some functions over others. We would like to use models where the support is large, so we can capture the complexity of real-world data, but also where the inductive bias places probability mass on the kinds of functions we expect to see. If we succeed at this, the posterior will quickly converge on the true function after seeing a small amount of data. This idea is sketched in Figure 17.10.

47

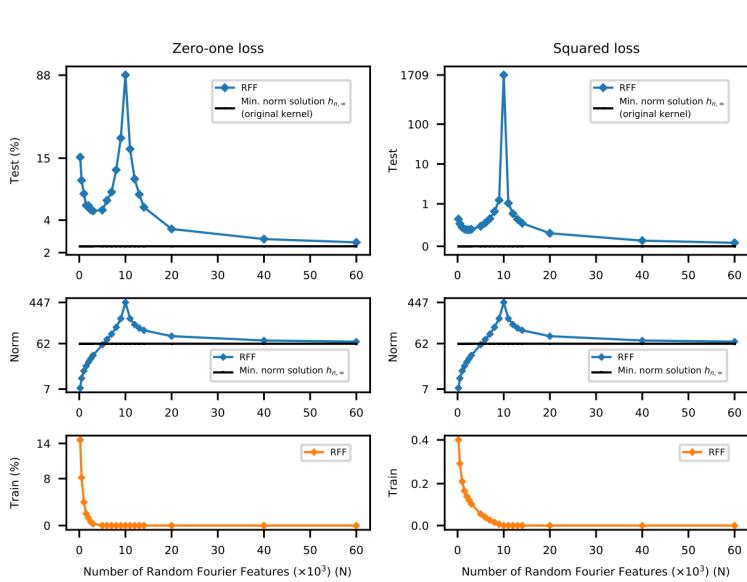


Figure 17.11: Illustration of the double descent risk curve for RFF model on a subset of MNIST. There are $N = 10^4$ examples, so interpolation occurs when the x -axis (which is number of features times 1000) equals 10. Top row: test error. Middle row: norm of \mathbf{w}^* . Bottom row: train error. From Figure 2 of [Bel+19b]. Used with kind permission of Mikhail Belkin.

17.5.4 Double descent

Practitioners often use very wide and deep models, with many more parameters than training points. These are called **over-parameterized models**. Naively one might think such models would overfit. However, they often exhibit very good generalization performance.

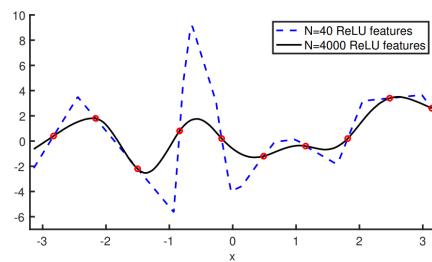
For example, consider a linear function of the form $f(\mathbf{x}) = \sum_{k=1}^K w_k \phi_k(\mathbf{x})$, where the $\phi_k(\mathbf{x})$ are random Fourier features (Section 18.2.3.1). If $K > N$, then there are more parameters than data points. In this setting, there are multiple solutions that can achieve zero training error. Call this set of interpolators $\mathcal{W} = \{\mathbf{w} : \text{RSS}(\mathbf{w}) = 0\}$. We pick the “simplest” solution in this set, which we define to be the one with least norm: $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathcal{W}} \|\mathbf{w}\|_2$.²

In Figure 17.11, we show the results of this method when applied to a subset of MNIST.³ As K increases, the test error goes down as underfitting is eliminated. As K approaches N , test error rises dramatically, as overfitting kicks in. However, when K exceeds N , test error goes down again, until it reaches the asymptotic performance. This is known as the **double descent risk curve**. We also

2. For linear models (with fixed basis functions ϕ), we can compute the minimum norm solution, by using the psuedo inverse, as explained in [Mur22, Sec 7.7.2]. For nonlinear models, it can be shown that gradient descent often converges to the minimum norm solution when started from $\mathbf{w} = \mathbf{0}$ (see e.g., [Nac+19c]).

3. For simplicity, we treat this as a linear regression problem with C outputs (corresponding to the one-hot class labels), rather than a logistic regression problem. This is known as “**least squares classification**”, and is known to work well if we are only interested in predicting the most probable class, (i.e., in minimizing zero-one loss), rather than computing a probability distribution [RK04].

1
2
3
4
5
6
7
8
9
10



11 *Figure 17.12: Two shallow MLPs fit to $N = 10$ data points (shown in red) using one layer of $K = 40$
12 or $K = 4000$ random ReLU features, i.e., the model has the form $h(x) = w_0 + \sum_{k=1}^K w_k \phi(x; \mathbf{v}_k)$, where
13 $\phi(x; \mathbf{v}) = \max(v_0 + v_1 x, 0)$, \mathbf{v}_k are chosen randomly and \mathbf{w} is optimized. The piecewise linear nature of the
14 fitted function is visually apparent when $K = 40$, but the function is smoother with larger K . The scaled
15 norm of the parameter vectors, $\frac{\|\mathbf{w}\|_2}{\sqrt{K}}$, is 695 and 159 for $K = 40$ and $K = 4000$, reflecting the fact that the
16 latter model is simpler. (Similar results hold when we optimize both \mathbf{V} and \mathbf{w} , but it is harder to implement
17 the minimum norm solution in this case.) From Figure 3 of [Bel+19b]. Used with kind permission of Mikhail
18 Belkin.*

19
20

21 plot the norm of the optimal vector as we increase K , and we see that the over-parameterized models
22 are simpler.

23 The “spike” at the interpolation threshold, when number of parameters is equal to the number of
24 datapoints, is due to the condition number of the design matrix \mathbf{X} going to infinity. If we add some
25 ℓ_2 regularization (or increase the “label noise”) this spike can be eliminated (see [Adl] for a detailed
26 analysis). Nevertheless performance still continues to improve as the (regularized) model becomes
27 more overparameterized, even when there is no such spike, due to the increasing simplicity of the
28 model.

29 Note that this phenomenon is not specific to this kind of model or data — it occurs with many
30 kinds of models and datasets [Bel+19b; Nak+19]. For example, Figure 17.12 compares two one-layer
31 MLPs fit to $N = 10$ points from a 1d regression problem. One model has $K = 40$ hidden units,
32 and one has $K = 4000$ hidden units. Both perfectly interpolate the training data, but the latter is
33 smoother.

34 A cartoon summarization of the situation is shown in Figure 17.13. On the left we show the classic
35 U-shaped curve, where increasing the model capacity too much results in overfitting. On the right, we
36 show the situation observed above, where with overparameterized models, adding more parameters
37 actually helps generalization performance.

38 A theoretical explanation for this phenomenon is given in [BS21]. Their result, roughly speaking,
39 is that if you want to interpolate the training data of N data points each of D dimensions using a
40 *smooth* function, then you need at least ND parameters. So by over-parameterizing the solution, we
41 can not just interpolate, but interpolate smoothly.

42

43 17.5.5 A Bayesian Resolution to Double Descent

44
45 We have argued that we wish to build models that have large support — and thus great flexibility —
46 but with a reasonable prior (largely induced by the neural architecture). From a Bayesian perspective,
47

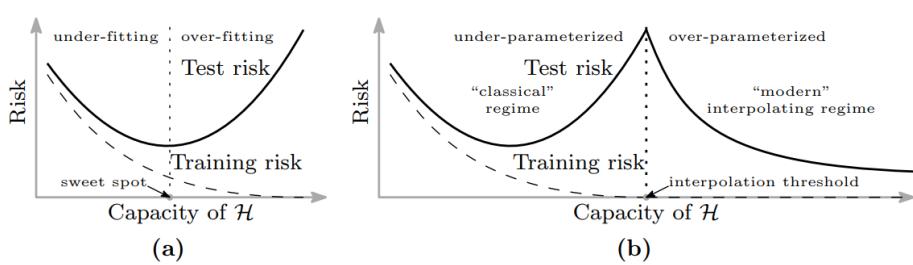


Figure 17.13: Curves for training risk (dashed line) and test risk (solid line) as a function of the size of the hypothesis space \mathcal{H} for the unknown function f . (a) Classical U-shaped curve in the under-parameterized regime. (b) Illustration of the double descent risk curve that arises when we consider models that may be over-parameterized. From Figure 1 of [Bel+19b]. Used with kind permission of Mikhail Belkin.

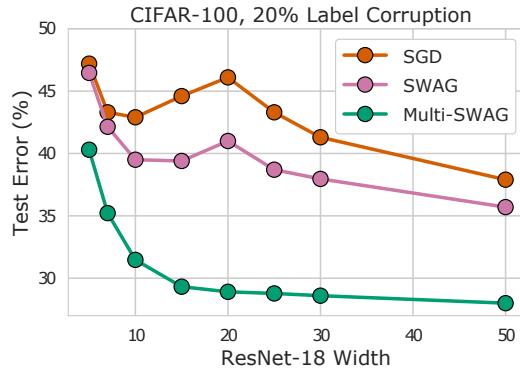


Figure 17.14: A ResNet-18 with layers of varying width trained on CIFAR-100 with 20% label corruption. Standard SGD training leads to significant double descent behavior. SWAG [Mad+19], which performs a single basin Bayesian model average, helps mitigate double descent. Multi-SWAG [WI20], which performs exhaustive multi-basin marginalization, entirely alleviates double descent. There is also a dramatic performance improvement, even for accuracy of point predictions, between Multi-SWAG and classical SGD training. From Figure 8 of [WI20]. Used with kind permission of Andrew Wilson.

one would expect that, with exhaustive Bayesian model averaging, performance should improve monotonically with model flexibility and we wouldn't observe double descent.

We indeed observe in Figure 17.14 that exhaustive Bayesian model averaging from Multi-SWAG (Section 17.4.8), which forms a mixture of Gaussians approximation to the posterior, entirely alleviates double descent. Multi-SWAG also leads to significant performance improvements over classical SGD training, even in terms of accuracy of point predictions. For example, for the ResNet-18 with layers of width 20 on CIFAR-100 with 20% label corruption, classical training achieves over 45% error, while Multi-SWAG has under 30% error.

We can also gain insights into why double descent occurs in classical training from a Bayesian perspective. In Section 17.5 we discussed how the effective dimensionality of the Hessian is proportional

1
2
3
4
5
6
7
8
9
10
11
12
13
14



15 *Figure 17.15: A resolution of double descent.* We replicate the double descent behaviour of deep neural
16 networks using a ResNet-18 on CIFAR-100, where train loss decreases to zero with sufficiently wide model while
17 test loss decreases, then increases, and then decreases again. Unlike model width, the effective dimensionality
18 computed from the eigenspectrum of the Hessian of the loss on training data alone follows the test loss in the
19 overparameterized regime, acting as a much better proxy for generalization than naive parameter counting.
20 From Figure 1 of [MBW20]. Used with kind permission of Andrew Wilson.

21

22 to posterior contraction in a Bayesian neural network. In Figure 17.15, the models with no training
23 loss can all be viewed as providing lossless compressions of the data. In this regime, we see that
24 effective dimensionality closely tracks double descent, and that as model size increases, the effective
25 number of parameters in fact decreases. In other words, the models with more parameters are
26 providing a better compression of the data, corresponding to flatter regions of the loss surface. Better
27 compressions can be found by discovering greater regularity explaining the data, and are thus more
28 likely to generalize. From a Bayesian perspective of model selection, flatter solutions also incur less
29 of an Occam factor penalty.

30 But why is SGD finding simpler solutions as we increase model size? As indicated by the effective
31 dimensionality, these simpler solutions correspond to flatter regions of the loss surface. In higher
32 dimensional spaces, flat regions occupy a much greater volume, and are thus much more easily
33 discoverable by optimization procedures. Since flat solutions are associated with better generalization,
34 we might call this behavior a “blessing of dimensionality” [Hua+19a; Izm+18].
35

36

37 17.5.6 PAC-Bayes

38 **PAC-Bayes** [McA99; LC02; Gue19; Alq21; GSZ21] provides a promising mechanism to derive
39 non-vacuous generalization bounds for large *stochastic networks* [Ney+17; NBS18; DR17], with
40 parameters sampled from a probability distribution. In particular, the difference between the train
41 error and the generalization error can be expressed as
42

$$43 \quad \sqrt{\frac{\text{KL}QP + c}{2(n-1)}}, \quad (17.27)$$

44
45

46 where c is a constant and n is the number of training points. P is the prior distribution over the
47

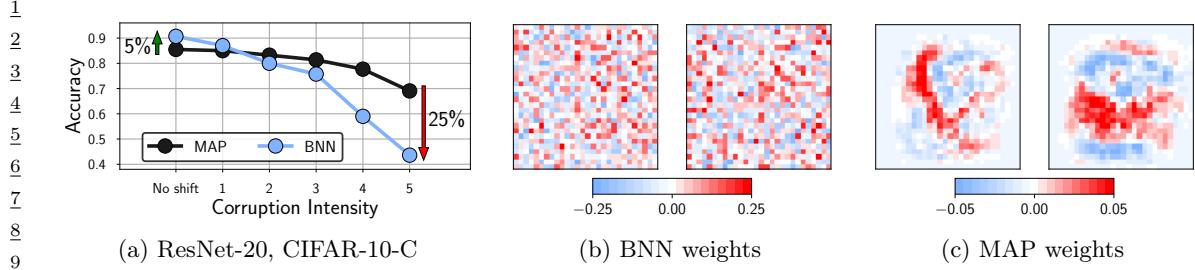


Figure 17.16: **Bayesian neural networks under covariate shift.** (a): Performance of a ResNet-20 on the pixelate corruption in CIFAR-10-C. For the highest degree of corruption, a Bayesian model average underperforms a MAP solution by 25% (44% against 69%) accuracy. See Izmailov et al. [Izm+21b] for details. (b): Visualization of the weights in the first layer of a Bayesian fully-connected network on MNIST sampled via HMC. (c): The corresponding MAP weights. We visualize the weights connecting the input pixels to a neuron in the hidden layer as a 28×28 image, where each weight is shown in the location of the input pixel it interacts with. This figure is adapted from Figure 1 of Izmailov et al. [Izm+21a].

parameters and Q is an arbitrary distribution, which can be chosen to optimize the bound.

The perspective in this chapter is largely complementary, and in some ways orthogonal, to the PAC-Bayes literature. Our focus has been on Bayesian marginalization, particularly multi-modal marginalization, and a prescriptive approach to model construction. In contrast, PAC-Bayes bounds are about bounding the empirical risk of a single sample, rather than marginalization, and are not currently prescriptive: what we would do to improve the bounds, such as reducing the number of model parameters, or using highly compact priors, does not typically improve generalization. Moreover, while we have seen Bayesian model averaging over multimodal posteriors has a significant effect on generalization, it has a minimal logarithmic effect on PAC-Bayes bounds. In general, because the bounds are loose, albeit non-vacuous in some cases, there is often room to make modeling choices that improve PAC-Bayes bounds without improving generalization, making it hard to derive a prescription for model construction from the bounds.

17.5.7 Out-of-Distribution Generalization for BNNs

Bayesian methods are often considered a robust alternative to classical training, because they represent many possible solutions to a given problem, corresponding to epistemic uncertainty. For this reason, Bayesian methods are often applied in out-of-distribution settings [Ova+19; KG17; Cha+19b; WI20; Dus+20; Ben+21]. There are two common objectives in this context: (1) to simplify detect a point as out-of-distribution and refuse to make a prediction; (2) to provide a reasonable predictive distribution for these points.

In many real-world applications we are in the second setting, where the training and test distributions are not exactly the same, but we still want to make a useful prediction: medical images taken from different devices, autonomous vehicles operating in different cities, recognizing typewritten characters from handwritten examples, and so on. Many standard out-of-distribution benchmarks, such as MNIST-C, CIFAR-10-C, ImageNet-C, and MNIST to SVHN, also represent this setting, with the test points still clearly recognizable from the training points, through noise corruptions, or mild domain shift. Approximate inference procedures have been providing increasingly good generalization

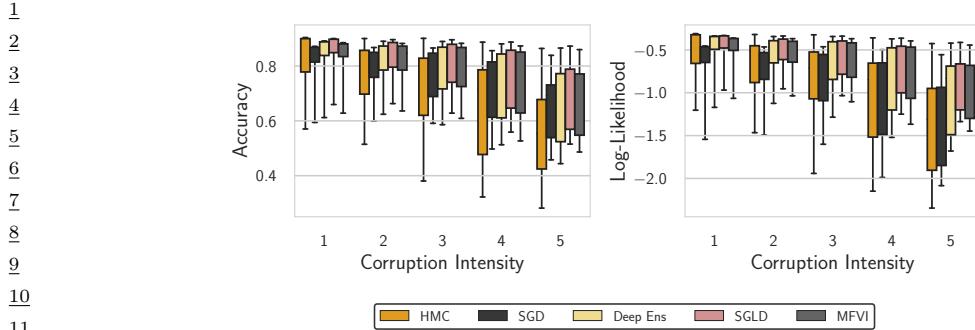
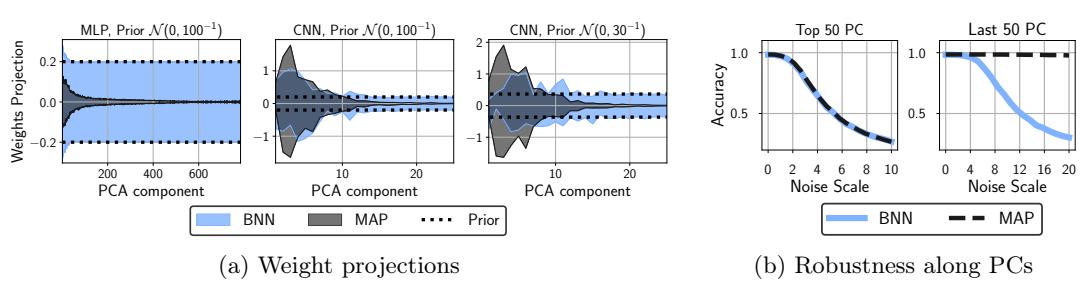


Figure 17.17: Evaluation on CIFAR-10-C. Accuracy and log-likelihood of HMC, SGD, deep ensembles, SGLD and MFVI on a distribution shift task, where the CIFAR-10 test set is corrupted in 16 different ways at various intensities on the scale of 1 to 5. We use the ResNet-20-FRN architecture. Boxes capture the quartiles of performance over each corruption, with the whiskers indicating the minimum and maximum. HMC is surprisingly the worst of the considered methods: even a single SGD solution provides better OOD robustness. This figure is adapted from Figure 7 of Izmailov et al. [Izm+21b].

accuracy on such benchmarks [Mil+21a]. However, high-fidelity approximate inference, through HMC, provides very poor generalization accuracy on out-of-distribution data, despite providing significantly better in-distribution accuracy on the analogous task, as shown in Figure 17.16 and Figure 17.17. These results stand in contrast to the good performance of many approximate inference procedures on these tasks [WI20; Dus+20; Ben+21], as well as work showing a strong correlation between in and out-of-distribution generalization accuracy on related tasks [Mil+21a], and the good performance of deep ensembles for both out-of-distribution detection and out-of-distribution generalization accuracy [Ova+19; Izm+21b].

Rather than an idiosyncracy of HMC, Izmailov et al. [Izm+21a] show this lack of robustness is a foundational issue of Bayesian model averaging under covariate shift, caused by degeneracies in the training data. As an illustrative special case, consider MNIST digits, which always have black corner pixels. The corresponding first layer weights are always multiplied by zero and have no effect on the likelihood. Consequently, these weights are simply sampled from the prior. If at test time the corner pixels are not black, e.g., due to corruption, these pixel values will be multiplied by random weights sampled from the prior, and propagated to the next layer, significantly degrading performance. On the other hand, classical MAP training or deep ensembles of MAP solutions drive the unrestricted parameters towards zero due to weight decay induced by any generic Gaussian prior, and will not be similarly affected by noise at test time. Here we indeed see a big difference between optimizing a posterior for MAP in comparison to a posterior weighted model average.

Figure 17.16(b, c) visualizes this example, where we see the weights in the first layer of a fully-connected network for a sample from the BNN posterior and the MAP solution on the MNIST dataset. The MAP solution weights are highly structured, while the BNN sample appears extremely noisy, similar to a draw from the Gaussian prior. In particular the weights corresponding to *dead pixels* (i.e. pixel positions that are black for all the MNIST images) near the boundary of the input image are set near zero (shown in white) by the MAP solution, but sampled randomly by the BNN. If at test time the data is corrupted, e.g. by Gaussian noise, and the pixels near the boundary of the image are activated, the MAP solution will ignore these pixels, while the predictions of the BNN will



(a) Weight projections

(b) Robustness along PCs

Figure 17.18: Bayesian inference samples weights along low-variance principal components from the prior, while MAP sets these weights to zero. (a): The distribution ($\text{mean} \pm 2 \text{ std}$) of projections of the weights of the first layer on the directions corresponding to the PCA components of the data for BNN samples and MAP solution using MLP and CNN architectures with different prior scales. In each case, MAP sets the weights along low-variance components to zero, while BNN samples them from the prior. (b): Accuracy of BNN and MAP solutions on the MNIST test set with Gaussian noise applied along the 50 highest and 50 lowest variance PCA components of the train data (left and right respectively). MAP is very robust to noise along low-variance PCA directions, while BMA is not; the two methods are similarly robust along the highest-variance PCA components. This figure is adapted from Figure 4 of Izmailov et al. [Izm+21a].

be significantly affected.

Izmailov et al. [Izm+21a] prove that this problem more generally occurs whenever there are linear dependencies in the input features of the data, for both fully-connected and convolutional networks. Intuitively, if there exists a direction in the input space such that all of the training data points have a constant projection on this direction (i.e. the data lies in a hyperplane), then posterior coincides with the prior in this direction. Hence, the BMA predictions are highly susceptible to perturbations that move the test inputs in a direction orthogonal to the hyperplane. The MAP solution on the other hand is completely robust to such perturbations.

We can gain some empirical intuitions into these theoretical results in Figure 17.18, by considering a fully-connected BNN on MNIST. The MNIST training dataset has linearly dependent features. In Figure 17.18(a), we see that the distribution of projections of the first layer weights onto the principal components of the data almost exactly coincides with the prior for PCA directions that are nearly constant in the data. The MAP solution, on the other hand, sets the weights along these PCA directions close to zero. In Figure 17.18(b), we see the MAP solution is very robust to noise along the low-variance directions, while BMA is not.

By introducing a prior over parameters which is aligned with the principal components of the training inputs, we can substantially improve the generalization accuracy of Bayesian neural networks in out-of-distribution settings. Izmailov et al. [Izm+21a] propose the following *EmpCov* prior: $p(w^1) = \mathcal{N}(0, \alpha\Sigma + \epsilon I)$, where w^1 are the first layer weights, $\Sigma = \frac{1}{n-1} \sum_{i=1}^n x_i x_i^T$ is the empirical covariance of the training input features x_i , $\alpha > 0$ determines the scale of the prior, and ϵ is a small positive constant to ensure the covariance matrix is positive definite.

¹ ² 17.6 Online inference

³ In Section 17.4, we have focused on batch or offline inference. However, an important application
⁴ of Bayesian inference is in sequential settings, where the data arrives in a continuous stream, and
⁵ the model has to “keep up”. This is called **sequential Bayesian inference**, and is one approach to
⁶ **online learning** (see Section 20.7.5). In this section, we discuss some algorithmic approaches to
⁷ this problem in the context of DNNs. These methods are widely used for continual learning, which
⁸ we discuss Section 20.7.

⁹ ¹⁰ 17.6.1 Extended Kalman Filtering for DNNs

¹¹ In Section 8.4.2, we showed how Kalman filtering can be used to incrementally compute the
¹² exact posterior for the weights of a linear regression model with known variance, i.e., we compute
¹³ $p(\boldsymbol{\theta}|\mathcal{D}_{1:t}, \sigma^2)$, where $\mathcal{D}_{1:t} = \{(\mathbf{u}_i, y_i) : i = 1 : t\}$ is the data seen so far, and

$$\frac{16}{17} p(y_t|\mathbf{u}_t, \boldsymbol{\theta}, \sigma^2) = \mathcal{N}(y_t|\boldsymbol{\theta}^\top \mathbf{u}_t, \sigma^2) \quad (17.28)$$

¹⁸ is the linear regression likelihood. The application of KF to this model is known as recursive least
¹⁹ squares.

²⁰ Now consider the case of nonlinear regression:

$$\frac{22}{23} p(y_t|\mathbf{u}_t, \boldsymbol{\theta}, \sigma^2) = \mathcal{N}(y_t|f(\boldsymbol{\theta}, \mathbf{u}_t), \sigma^2) \quad (17.29)$$

²⁴ where $f(\boldsymbol{\theta}, \mathbf{u}_t)$ is some nonlinear function, such as an MLP. We can use the extended Kalman filter
²⁵ (Section 8.5.2) to approximately compute $p(\boldsymbol{\theta}_t|\mathcal{D}_{1:t}, \sigma^2)$, where $\boldsymbol{\theta}_t$ is the hidden state (see e.g., [SW89;
²⁶ PF03]). To see this, note that we can set the dynamics model to the identity function, $f(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t$, so
²⁷ the parameters are propagated through unchanged, and the observation model to the input-dependent
²⁸ function $f(\boldsymbol{\theta}_t) = f(\boldsymbol{\theta}_t, \mathbf{u}_t)$. We set the observation noise to $\mathbf{R}_t = \sigma^2$, and the dynamics noise to
²⁹ $\mathbf{Q}_t = q\mathbf{I}$, where q is a small constant, to allow the parameters to slowly drift) according to artificial
³⁰ **process noise**. (In practice it can be useful to anneal q from a large initial value to something near
³¹ 0.)

³² In more detail, let \mathbf{H}_t be the Jacobian of the observation model at time t , which can be computed
³³ using automatic differentiation. \mathbf{H}_t is a matrix of N_o column vectors, each of size N_w , where N_o is
³⁴ the number of outputs of the MLP, and N_w is the number of weights (parameters). (Note that \mathbf{H}_t
³⁵ plays the role of the observation matrix \mathbf{C}_t in the KF.) The dynamics matrix is $\mathbf{F}_t = \mathbf{I}$. The EKF
³⁶ equations from Section 8.5.2 can then be written as follows:

$$\frac{38}{39} \mathbf{S}_t = (\mathbf{R}_t + \mathbf{H}_t^\top \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t) \quad (17.30)$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t \mathbf{S}_t^{-1} \quad (17.31)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t (y_t - \hat{y}_t) \quad (17.32)$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \quad (17.33)$$

$$\boldsymbol{\mu}_{t+1|t} = \boldsymbol{\mu}_t \quad (17.34)$$

$$\boldsymbol{\Sigma}_{t+1|t} = \boldsymbol{\Sigma}_t + \mathbf{Q}_t \quad (17.35)$$

⁴⁶ where $\hat{y}_t = f(\mathbf{u}_t; \boldsymbol{\mu}_{t-1})$ is the predicted output.

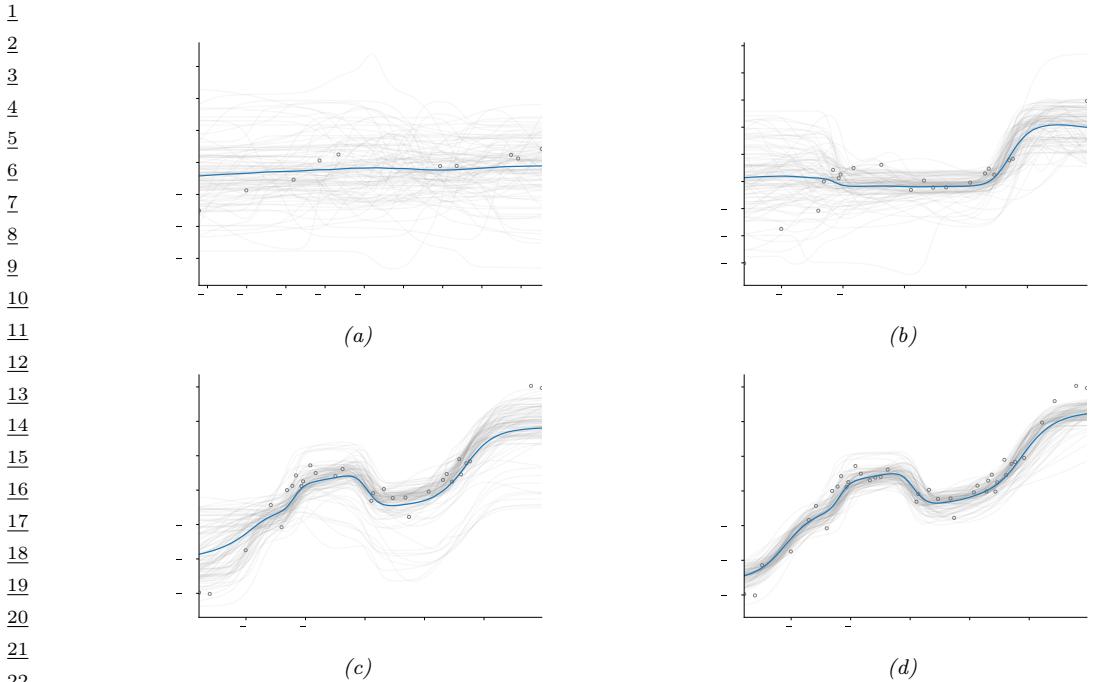


Figure 17.19: Sequential Bayesian inference for the parameters of an MLP applied to a nonlinear regression problem using the extended Kalman filter. We show results after seeing the first 10, 20, 30 and 40 observations. (For a video of this, see <https://bit.ly/3wXnWaM>.) Generated by `ekf_vs_ukf_mlp_demo.py`.

17.6.1.1 Example

We now give an example of this process in action. We sample a synthetic dataset from the true function

$$h^*(u) = x - 10 \cos(u) \sin(u) + u^3 \quad (17.36)$$

and add Gaussian noise with $\sigma = 3$. We then fit this with an MLP with one hidden layer with H hidden units, so the model has the form

$$f(\boldsymbol{\theta}, \mathbf{u}) = \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{u} + \mathbf{b}_1) + \mathbf{b}_2 \quad (17.37)$$

where $\mathbf{W}_1 \in \mathbb{R}^{H \times 1}$, $\mathbf{b}_1 \in \mathbb{R}^H$, $\mathbf{W}_2 \in \mathbb{R}^{1 \times H}$, $\mathbf{b}_2 \in \mathbb{R}^1$. We set $H = 6$, so there are $D = 19$ parameters in total.

Given the data, we sequentially compute the posterior, starting from a vague Gaussian prior, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{0}, \boldsymbol{\Sigma}_0)$, where $\boldsymbol{\Sigma}_0 = 100\mathbf{I}$. (In practice we cannot start from the prior mean, which is $\boldsymbol{\theta}_0 = \mathbf{0}$, since linearizing the model around this point results in a zero gradient, so we use an initial random sample for $\boldsymbol{\theta}_0$.) The results are shown in Figure 17.19. We can see that the model adapts to the data, without having to specify any learning rate. In addition, we see that the predictions become gradually more confident, as the posterior concentrates on the MLE.

1 **17.6.1.2 Setting the variance terms**

2 In the above example, we set the variance terms by hand. In general we need to estimate the noise
3 variance σ , which determines \mathbf{R}_t and hence the learning rate, as well as the strength of the prior Σ_0 ,
4 which controls the amount of regularization. Some methods for doing this are discussed in [FNG00].
5

6 **17.6.1.3 Reducing the computational complexity**

7 The naive EKF method described above takes $O(D^3)$ time, which is prohibitive for large neural
8 networks. A simple approximation, known as the **decoupled EKF**, was proposed in [PF91; SPD92]
9 (see [PF03] for a review). This partitions the weights into G groups or blocks, and estimates the
10 relevant matrices for each group g independently. More precisely, the update becomes
11

12
$$\mathbf{A}_t = \left(\mathbf{R}_t + \sum_{g=1}^G (\mathbf{H}_t^g)^\top \mathbf{P}_t^g \mathbf{H}_t^g \right)^{-1} \quad (17.38)$$

13
$$\mathbf{K}_t^g = \mathbf{P}_t^g \mathbf{H}_t^g \mathbf{A}_t \quad (17.39)$$

14
$$\mathbf{w}_{t+1}^g = \mathbf{w}_t^g + \mathbf{K}_t^g (\mathbf{y}_t - \hat{\mathbf{y}}_t) \quad (17.40)$$

15
$$\mathbf{P}_{t+1}^g = \mathbf{P}_t^g - \mathbf{K}_t^g (\mathbf{H}_t^g)^\top \mathbf{P}_t^g + \mathbf{Q}_t^g \quad (17.41)$$

16 If there are N_g weights in block g , $N_w = \sum_{g=1}^G N_g$ weights in total, and N_o outputs from the model,
17 the computation time is $O(N_o^2 N_w + N_o \sum_{g=1}^G N_g^2)$ and the space becomes $O(\sum_{g=1}^G N_g^2)$. If $G = 1$,
18 this reduces the standard global EKF. If we put each weight into its own group, we get a fully
19 diagonal approximation. In practice this does not work any better than SGD (possibly with diagonal
20 scaling), since it ignores correlations between the parameters. A useful compromise is to put all the
21 weights corresponding to each neuron into its own group; this is called “node decoupled EKF”.
22

23 **17.6.1.4 Beyond squared error**

24 The EKF assumes Gaussian observation noise, and thus minimizes the sum of squared errors at each
25 step between the prediction $\hat{\mathbf{y}}_t$ and the target vector \mathbf{y}_t . In Section 8.5.4, we discuss how to extend
26 the EKF so it can be applied to classification problems,
27

28 **17.6.2 Assumed Density Filtering for DNNs**

29 In Section 8.8.4, we discussed how to use assumed density filtering (ADF) to perform online (binary)
30 logistic regression. In this section, we generalize this to nonlinear predictive models, such as DNNs.
31 The key is to perform Gaussian moment matching of the hidden activations at each layer of the model.
32 This provides an alternative to the EKF approach in Section 17.6.1, which is based on linearization
33 of the network.
34

35 We will assume the following likelihood:

36
$$p(\mathbf{y}_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Expfam}(\mathbf{y}_t | \ell^{-1}(f(\mathbf{y}_t; \mathbf{w}_t))) \quad (17.42)$$

37 where $f(\mathbf{x}; \mathbf{w})$ is the DNN, ℓ^{-1} is the inverse link function, and $\text{Expfam}()$ is some exponential family
38 distribution. For example, if f is linear and we are solving a binary classification problem, we can
39

1
2 write

$$\underline{3} \quad p(y_t | \mathbf{y}_t, \mathbf{w}_t) = \text{Ber}(y_t | \boldsymbol{\sigma}(\mathbf{y}_t^\top \mathbf{w}_t)) \quad (17.43)$$

5 We discussed using ADF to fit this model in Section 8.8.4.

6 In [HLA15a], they propose **Probabilistic backpropagation (PBP)**, which is an instance of
7 ADF applied to MLPs. The basic idea is to approximate the posterior over the weights in each layer
8 using a fully factorized distribution

$$\underline{9} \quad p(\mathbf{w}_t | \mathcal{D}_{1:t}) \approx p_t(\mathbf{w}_t) = \prod_{l=1}^L \prod_{i=1}^{D_l} \prod_{j=1}^{D_{l-1}+1} \mathcal{N}(w_{ijl} | \mu_{ijl}^t, \tau_{ijl}^t) \quad (17.44)$$

13 where L is the number of layers, and D_l is the number of neurons in layer l . (The **expectation**
14 **backpropagation** algorithm of [SHM14] is a special case of this, where the variances are fixed to
15 $v = 1$.)

16 Suppose the parameters are static, so $\mathbf{w}_t = \mathbf{w}_{t-1}$. Then the new posterior, after conditioning on
17 the t 'th observation, is given by

$$\underline{18} \quad \hat{p}_t(\mathbf{w}) = \frac{1}{Z_t} p(\mathbf{y}_t | \mathbf{y}_t, \mathbf{w}) \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}^{t-1}, \boldsymbol{\Sigma}^{t-1}) \quad (17.45)$$

21 where $\boldsymbol{\Sigma}^{t-1} = \text{diag}(\boldsymbol{\tau}^{t-1})$. We then project $\hat{p}_t(\mathbf{w})$ instead the space of factored Gaussians to compute
22 the new (approximate) posterior, $p_t(\mathbf{w})$. This can be done by computing the following means and
23 variances [Min01a]:

$$\underline{24} \quad \mu_{ijl}^t = \mu_{ijl}^{t-1} + \tau_{ijl}^{t-1} \frac{\partial \ln Z_t}{\partial \mu_{ijl}^{t-1}} \quad (17.46)$$

$$\underline{27} \quad \tau_{ijl}^t = \tau_{ijl}^{t-1} - (\tau_{ijl}^{t-1})^2 \left[\left(\frac{\partial \ln Z_t}{\partial \mu_{ijl}^{t-1}} \right)^2 - 2 \frac{\partial \ln Z_t}{\partial \tau_{ijl}^{t-1}} \right] \quad (17.47)$$

30 In the forwards pass, we compute Z_t by propagating the input \mathbf{y}_t through the model. Since we have
31 a Gaussian distribution over the weights, instead of a point estimate, this induces an (approximately)
32 Gaussian distribution over the values of the hidden units. For certain kinds of activation functions
33 (such as ReLU), the relevant integrals (to compute the means and variances) can be solved analytically,
34 as in GP-neural networks (Section 18.7). The result is that we get a Gaussian distribution over the
35 final layer of the form $\mathcal{N}(\boldsymbol{\eta}_t | \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\eta}_t = f(\mathbf{y}_t; \mathbf{w}_t)$ is the output of the neural network before
36 the GLM link function induced by $p_t(\mathbf{w}_t)$. Hence we can approximate the partition function using
37

$$\underline{38} \quad Z_t \approx \int p(\mathbf{y}_t | \boldsymbol{\eta}_t) \mathcal{N}(\boldsymbol{\eta}_t | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{\eta}_t \quad (17.48)$$

40 In the backwards pass, we take derivatives of this, and update the posteriors, \boldsymbol{m}^t and \boldsymbol{v}^t .

41 It remains to compute the marginal likelihood in Equation (17.48). In the case of probit classification,
42 with $y \in \{-1, +1\}$, we have $p(y | \mathbf{x}, \mathbf{w}) = \Phi(y\eta)$, where Φ is the cdf of the standard normal. We can
43 then use the following analytical result

$$\underline{45} \quad \int \Phi(y\eta) \mathcal{N}(h | \mu, \sigma) d\eta = \Phi \left(\frac{y\mu}{\sqrt{1+\sigma}} \right) \quad (17.49)$$

¹ In the case of logistic classification, with $y \in \{0, 1\}$, we have $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\eta))$; in this case,
² we can use the probit approximation from Section 15.3.5. For the multiclass case, where $\mathbf{y} \in \{0, 1\}^C$
³ (one-hot encoding), we have $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \text{Cat}(\mathbf{y}|\mathcal{S}(\boldsymbol{\eta}))$. A variational lower bound to $\log Z_t$ for this
⁴ case is given in [GDFY16]. They also give an approximation to Z_t when using the Poisson likelihood,
⁵ where $y \in \{0, 1, 2, \dots\}$, and $p(y|\mathbf{x}, \mathbf{w}) = \text{Poi}(y|e^\eta)$.

⁶

⁷ 17.6.3 Sequential Laplace for DNNs

⁸ In [RBB18b], they extended the Laplace method of Section 17.4.1 to the sequential setting. Specifically,
⁹ let $p(\boldsymbol{\theta}|\mathcal{D}_{1:t-1}) \approx \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Lambda}_{t-1}^{-1})$ be the approximate posterior from the previous step; we assume
¹⁰ the precision matrix is Kronecker factored. We now compute the new mean by solving the MAP
¹¹ problem

$$\boldsymbol{\mu}_t = \operatorname{argmax} \log p(\mathcal{D}_t|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}|\mathcal{D}_{t-1}) \quad (17.50)$$

$$= \operatorname{argmax} \log p(\mathcal{D}_t|\boldsymbol{\theta}) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_{t-1})\boldsymbol{\Lambda}_{t-1}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{t-1}) \quad (17.51)$$

¹² Once we have computed $\boldsymbol{\mu}_t$, we compute the approximate Hessian at this point, and get the new
¹³ posterior precision

$$\boldsymbol{\Lambda}_t = \lambda \mathbf{H}(\boldsymbol{\mu}_t) + \boldsymbol{\Lambda}_{t-1} \quad (17.52)$$

¹⁴ where $\lambda \geq 0$ is a weighting factor that trades off how much the model pays attention to the new data
¹⁵ vs old data.

¹⁶ Now suppose we use a diagonal approximation to the posterior prediction matrix. From Equa-
¹⁷ tion (17.51), we see that this amounts to adding a quadratic penalty to each new MAP estimate, to
¹⁸ encourage it to remain close to the parameters from previous tasks. This approach is called **elastic**
¹⁹ **weight consolidation (EWC)** [Kir+17].

²⁰

²¹ 17.6.4 Variational methods

²²

²³ A natural approach to online Bayesian inference is to use variational inference, where the prior is the
²⁴ posterior from the previous time step. That is, we optimize

$$\mathcal{L}(\boldsymbol{\psi}_t) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_t)} [\log p(\mathcal{D}_t|\boldsymbol{\theta})] - D_{\text{KL}}(q(\boldsymbol{\theta}|\boldsymbol{\psi}_t) \| q(\boldsymbol{\theta}|\boldsymbol{\psi}_{t-1})) \quad (17.53)$$

²⁵ This is called **variational continual learning** or **VCL** [Ngu+18b], as we discussed in Section 10.3.9.
²⁶ Unfortunately this method often does not work in practice, in part because of the variational
²⁷ overpruning effect discussed in Section 17.4.2, in which hidden units from earlier tasks are “turned
²⁸ off” (have their weights set to 0), which is an effect that cannot be easily undone using gradient
²⁹ based learning. Some generalizations of VCL that ameliorate this problem are discussed in [LST21].

³⁰

³¹ 17.7 Hierarchical Bayesian neural networks

³²

³³ In some problems, we have multiple related datasets, such as a set of medical images from different
³⁴ hospitals. Some aspects of the data (e.g., the shape of cells as a function of their state) is generally
³⁵

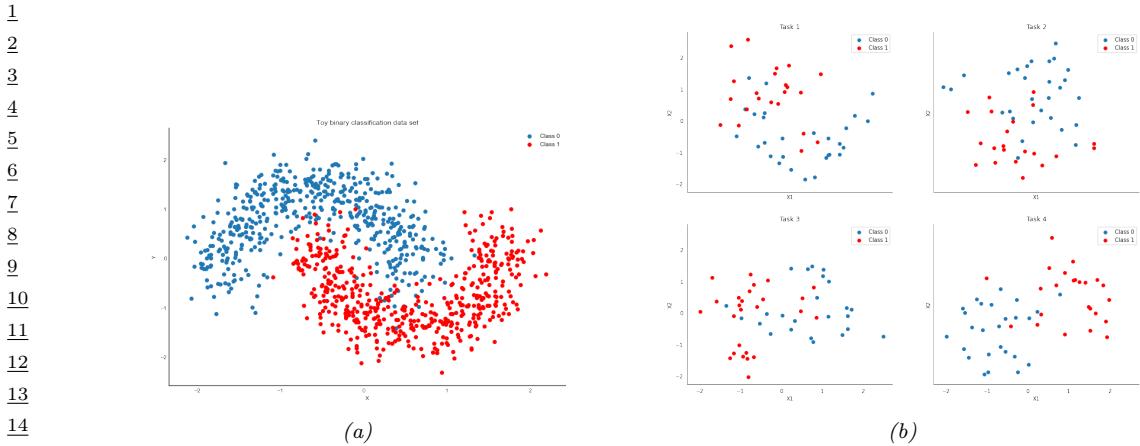


Figure 17.20: (a) Two moons synthetic dataset. (b) Multi-task version, where we rotate the data to create 18 related tasks (groups). Each dataset has 50 training and 50 test points. Here we show the first 4 tasks. Generated by [bnn_hierarchical_blackjax.ipynb](#).

the same across datasets, but other aspects may be unique or idiosyncratic (e.g., each hospital may use a different colored die for staining). To model this, we can use a hierarchical Bayesian model, in which we allow the parameters for each dataset to be different (to capture random effects), while coming from a common prior (to capture shared effects). This is a neural net version of the mixed effects models we studied in Section 15.5.1. We give an example below.

17.7.1 Solving multiple related classification problems

In this section, we consider an example⁴ where we want to solve multiple related nonlinear binary classification problems. We assume that each subproblem or “task” t only has a small number N_t of examples, which share the same shaped decision boundary, but modified in a way that is unique to each task. (This is an example of multi-task learning, which we discuss in more detail in Section 20.5.5, and is related to domain generalization, which we discuss in more detail in Section 20.5.6.)

We first create some synthetic 2d data for the $T = 18$ tasks. We start with the “two-moons” dataset, illustrated in Figure 17.20a. Each task is obtained by rotating the 2d inputs by a different amount, to create 18 related classification problems (see Figure 17.20b). Since we have multiple versions of the two-moons dataset, we call it the “multi-moons” dataset. See Figure 17.20b for the training data for 4 tasks.

To handle the nonlinear decision boundary, we use a multilayer perceptron. Since the dataset is low-dimensional (2d input), we use a shallow model with just 2 hidden layers, each with 5 neurons. We could fit a separate MLP to each task, but since we have limited data per task ($N_t = 50$ examples for training), this works poorly, as we show below. We could also pool all the data and fit a single model, but this does even worse, since the datasets come from different underlying distributions. Instead we adopt a hierarchical Bayesian approach, as illustrated in Figure 17.21. In particular, we

4. This example is from https://twiecki.io/blog/2018/08/13/hierarchical_bayesian_neural_network/.

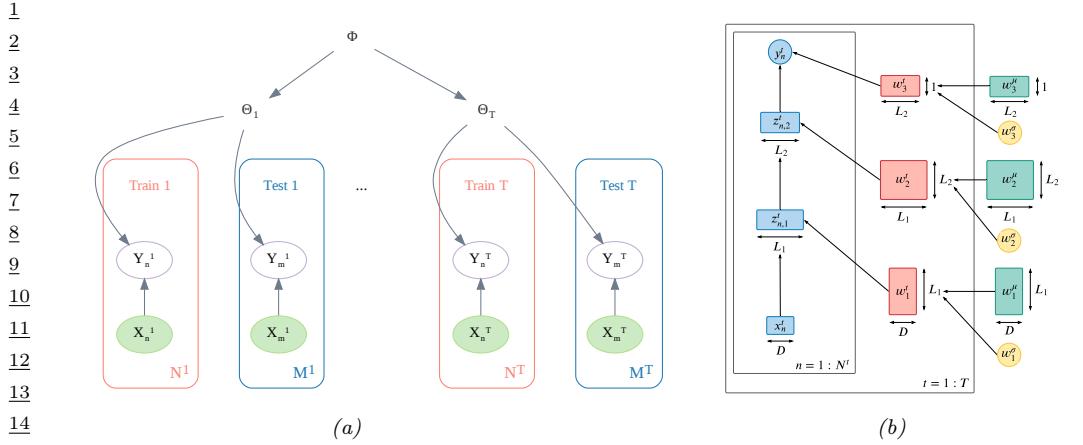


Figure 17.21: (a) Illustration of a hierarchical Bayesian discriminative model with T groups or tasks. Generated by [hbayes_figures2.ipynb](#). (b) Zoom-in on the dependency structure, for the case where $p(y|\mathbf{x}, \theta_t)$ is an MLP with 2 hidden layers, with sizes L_1 and L_2 . The input is D -dimensional, and output is the scalar logit for the binary output. Used with kind permission of Aleyna Kara.

20

21

assume the weight from unit i to unit j in layer l for task t , denoted $w_{i,j,l}^t$, come from a common (task independent) prior $\mathcal{N}(\mu_{i,j,l}, \sigma_l^2)$. We use the non-centered parameterization from Section 12.6.4 to write

$$w_{i,j,l}^t = \mu_{i,j,l} + \tilde{w}_{i,j,l}^t \times \sigma_l \quad (17.54)$$

where $\tilde{w}_{i,j,l}^t \sim \mathcal{N}(0, 1)$. For the hyper-parameters, we put $\mathcal{N}(0, 1)$ priors on $\mu_{i,j,l}$, and $\mathcal{N}_+(1)$ priors on σ_l . By allowing a different σ_l per layer, we let the model control the degree of shrinkage to the prior for each layer separately.

We compute the posterior $p(\tilde{\mathbf{w}}^{1:T}, \boldsymbol{\mu}, \boldsymbol{\sigma} | \mathcal{D})$ using HMC (Section 12.5). The average classification accuracy on the train and test sets for the non-hierarchical model (one MLP per group, fit separately) is 86% and 83%. For the hierarchical model, this improves to 91% and 89% respectively.

To see why the hierarchical model works better, we will plot the posterior predictive distribution in 2d. Figure 17.22 shows the results for the non-hierarchical models; we see that the method fails to learn the common underlying Z-shaped decision boundary. By contrast, Figure 17.23 shows that the method has correctly recovered the common pattern, while still allowing group variation.

Of course, multi-task learning does not always help (see e.g., [WZR20]), because sometimes there can be “**task interference**” or “**negative transfer**”. In Bayesian terms, this just means the modeling assumptions about a shared unimodal prior being useful for all tasks is inappropriate. This can be solved by using richer priors, such as mixture distributions, or sparse priors.

42

43

44

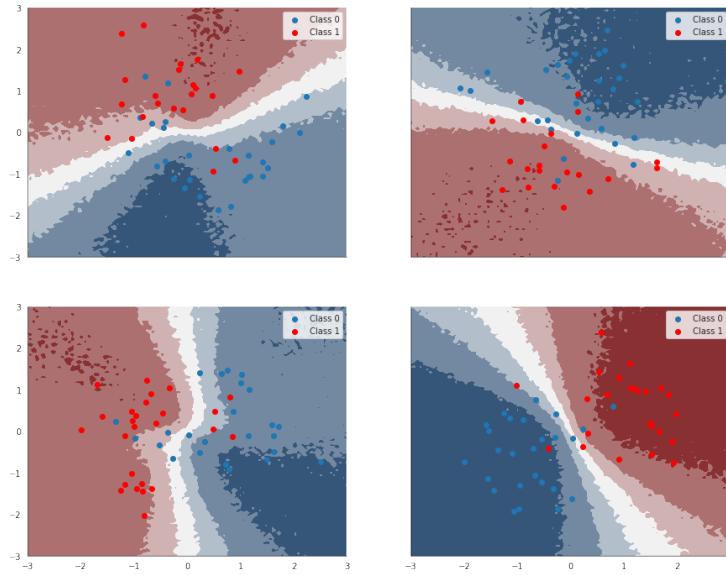
45

46

47

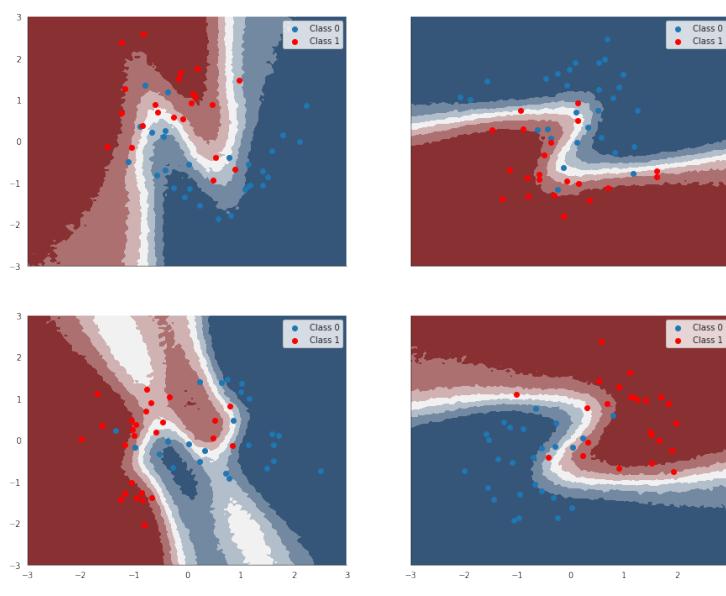
17.7. HIERARCHICAL BAYESIAN NEURAL NETWORKS

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20



21 *Figure 17.22: Results of fitting separate MLPs on each dataset. Generated by [bnn_hierarchical_blackjax.ipynb](#).*

22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43



44 *Figure 17.23: Results of fitting hierarchical MLP on all datasets jointly. Generated by [bnn_hierarchical_blackjax.ipynb](#).*

45
46
47

18 Gaussian processes

This chapter is co-authored with Andrew Wilson.

18.1 Introduction

Deep neural networks are a family of flexible function approximators of the form $f(\mathbf{x}; \boldsymbol{\theta})$, where the dimensionality of $\boldsymbol{\theta}$ (i.e., the number of parameters) is fixed, and independent of the size N of the training set. However, such parametric models can overfit when N is small, and can underfit when N is large, due to their fixed capacity. In order to create models whose capacity automatically adapts to the amount of data, we turn to **nonparametric models**.

There are many approaches to building nonparametric models for classification and regression (see e.g., [Was06]). In this chapter, we consider a Bayesian approach in which we represent uncertainty about the input-output mapping f by defining a prior distribution over functions, and then updating it given data. This is an example of a Bayesian nonparametric model — see Chapter 33 for other examples.

To explain this procedure in more detail, recall that a Gaussian random vector of length N , $\mathbf{f} = [f_1, \dots, f_N]$, is defined by its mean $\boldsymbol{\mu} = \mathbb{E}[\mathbf{f}]$ and its covariance $\boldsymbol{\Sigma} = \text{Cov}[\mathbf{f}]$. Now consider a function $f : \mathcal{X} \rightarrow \mathbb{R}$ evaluated at a set of inputs, $\mathbf{X} = \{\mathbf{x}_n \in \mathcal{X}\}_{n=1}^N$. Let $\mathbf{f}_X = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)]$ be the set of unknown function values at these points. If \mathbf{f}_X is jointly Gaussian for any set of $N \geq 1$ points, then we say that $f : \mathcal{X} \rightarrow \mathbb{R}$ is a **Gaussian process**. Such a process is defined by its **mean function** $m(\mathbf{x}) \in \mathbb{R}$ and a **covariance function**, $\mathcal{K}(\mathbf{x}, \mathbf{x}') \geq 0$, which is any positive definite **Mercer kernel** (see Section 18.2). For example, we might use an RBF kernel of the form $\mathcal{K}(\mathbf{x}, \mathbf{x}') \propto \exp(-\|\mathbf{x} - \mathbf{x}'\|^2)$ (see Section 18.2 for details).

We denote the corresponding GP by

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \mathcal{K}(\mathbf{x}, \mathbf{x}')) \quad (18.1)$$

where

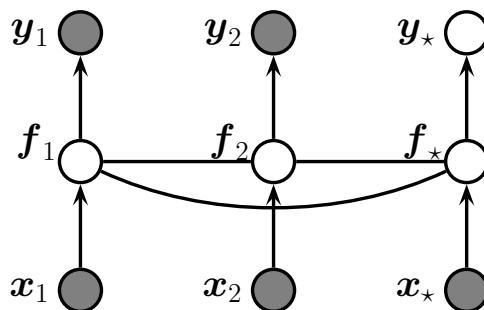
$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \quad (18.2)$$

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^\top] \quad (18.3)$$

This means that, for any finite set of points $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, we have

$$p(\mathbf{f}_X | \mathbf{X}) = \mathcal{N}(\mathbf{f}_X | \boldsymbol{\mu}_X, \mathbf{K}_{X,X}) \quad (18.4)$$

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 18.1: A Gaussian process for 2 training points, \mathbf{x}_1 and \mathbf{x}_2 , and 1 testing point, \mathbf{x}_* , represented as*
 15 *a graphical model representing $p(\mathbf{y}, \mathbf{f}_X | \mathbf{X}) = \mathcal{N}(\mathbf{f}_X | m(\mathbf{X}), \mathcal{K}(\mathbf{X})) \prod_i p(y_i | f_i)$. The hidden nodes $f_i = f(\mathbf{x}_i)$*
 16 *represent the value of the function at each of the data points. These hidden nodes are fully interconnected*
 17 *by undirected edges, forming a Gaussian graphical model; the edge strengths represent the covariance terms*
 18 *$\Sigma_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$. If the test point \mathbf{x}_* is similar to the training points \mathbf{x}_1 and \mathbf{x}_2 , then the value of the hidden*
 19 *function f_* will be similar to f_1 and f_2 , and hence the predicted output y_* will be similar to the training*
 20 *values y_1 and y_2 .*

21
22
23

24 where $\mu_X = (m(\mathbf{x}_1), \dots, m(\mathbf{x}_N))$ and $\mathbf{K}_{X,X}(i,j) \triangleq \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$.

25 A GP can be used to define a prior over functions. We can evaluate this prior at any set of points
 26 we choose. However, to learn about the function from data, we have to update this prior with a
 27 likelihood function. We typically assume we have a set of N iid observations $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1 : N\}$,
 28 where $y_i \sim p(y|f(\mathbf{x}_i))$, as shown in Figure 18.1. If we use a Gaussian likelihood, we can compute the
 29 posterior $p(f|\mathcal{D})$ in closed form, as we discuss in Section 18.3. For other kinds of likelihoods, we will
 30 need to use approximate inference, as we discuss in Section 18.4. In many cases f is not directly
 31 observed, and instead forms part of a latent variable model, both in supervised and unsupervised
 32 settings such as in Section 29.3.6.

33 The generalization properties of a Gaussian process are controlled by its covariance function
 34 (kernel), which we describe in Section 18.2. These kernels live in a reproducing kernel Hilbert space
 35 (RKHS), described in Section 18.3.7.1.

36 GPs were originally designed for spatial data analysis, where the input is 2d. This special case
 37 is called **kriging**. However, they can be applied to higher dimensional inputs. In addition, while
 38 they have been traditionally limited to small datasets, it is now possible to apply GPs to problems
 39 with millions of points, with essentially exact inference. We discuss these scalability advances in
 40 Section 18.5.

41 Moreover, while Gaussian processes have historically been considered smoothing interpolators, GPs
 42 now routinely perform representation learning, through covariance function learning, and multilayer
 43 models. These advances have clearly illustrated that GPs are neural networks are not competing,
 44 but complementary, and can be combined for better performance than would be achieved by deep
 45 learning alone. We describe GPs for representation learning in Section 18.6.

46 The connections between Gaussian processes and neural networks can also be further understood
 47

1 by considering infinite limits of neural networks that converge to Gaussian processes with particular
 2 covariance functions, which we describe in Section 18.7.
 3

4 So Gaussian processes are non-parametric models which can scale and do representation learning.
 5 But why, in the age of deep learning, should we want to use a Gaussian process? There are several
 6 compelling reasons to prefer a GP, including:

- 7 Gaussian processes typically provide well-calibrated predictive distributions, with a good characteriza-
 8 tion of epistemic (model) uncertainty — uncertainty arising from not knowing which of
 9 many solutions is correct. For example, as we move away from the data, there are a greater variety
 10 of consistent solutions, and so we expect greater uncertainty.
- 11
- 12 Gaussian processes are often state-of-the-art for continuous regression problems, especially spa-
 13 tiotemporal problems, such as weather interpolation and forecasting. In regression, Gaussian
 14 process inference can also typically be performed in closed form.
- 15
- 16 The marginal likelihood of a Gaussian process provides a powerful mechanism for flexible kernel
 17 learning. Kernel learning enables us to provide long-range extrapolations, but also tells us
 18 interpretable properties of the data that we didn't know before, towards scientific discovery.
- 19
- 20 Gaussian processes are often used as a probabilistic surrogate for objectives in optimization, in a
 21 procedure known as **Bayesian optimization** (Section 6.9). To maximize an objective, we wish
 22 to move where there is a high expected value, but also to explore where we have large uncertainty.
 23 The ability for a Gaussian process to provide closed form inference in regression, in conjunction
 24 with high quality uncertainty representations, make them particularly impactful in this setting.
 25 Bayesian optimization has a wide range of applications, including A/B testing, experimental
 26 design, protein engineering, hyperparameter tuning, and AutoML. See Section 6.9 for details.

27 18.2 Mercer kernels

29 The generalization properties of Gaussian processes boil down to how we encode prior knowledge
 30 about the similarity of two input vectors. If we know that \mathbf{x}_i is similar to \mathbf{x}_j , then we can encourage
 31 the model to make the predicted output at both locations (i.e., $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$) to be similar.

32 To define similarity, we introduce the notion of a **kernel function**. The word “kernel” has many
 33 different meanings in mathematics; here we consider a **Mercer kernel**, also called a **positive**
 34 **definite kernel**. This is any symmetric function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ such that

$$36 \quad \sum_{i=1}^N \sum_{j=1}^N \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0 \quad (18.5)$$

39 for any set of N (unique) points $\mathbf{x}_i \in \mathcal{X}$, and any choice of numbers $c_i \in \mathbb{R}$. We assume $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) > 0$,
 40 so that we can only achieve equality in the above equation if $c_i = 0$ for all i .

41 Another way to understand this condition is the following. Given a set of N datapoints, let us
 42 define the **Gram matrix** as the following $N \times N$ similarity matrix:

$$44 \quad \mathbf{K} = \begin{pmatrix} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \mathcal{K}(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \mathcal{K}(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} \quad (18.6)$$

¹ We say that \mathcal{K} is a Mercer kernel iff the Gram matrix is positive definite for any set of (distinct)
² inputs $\{\mathbf{x}_i\}_{i=1}^N$.

³ The most widely used kernel for real-valued inputs is the **radial basis function kernel** (RBF)
⁴ kernel), also called the **exponentiated quadratic kernel**, **Gaussian kernel**, **squared exponential**
⁵ **(SE) kernel**. This kernel is defined as
⁶

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right) \quad (18.7)$$

⁷ Here ℓ corresponds to the length-scale of the kernel, i.e., the distance over which we expect differences
⁸ to matter. This is also sometimes known as the **bandwidth** parameter. The RBF kernel measures
⁹ similarity between two vectors in \mathbb{R}^D using (scaled) Euclidean distance. In Section 18.2.1, we will
¹⁰ discuss several other kinds of kernel.

¹¹

¹² 18.2.1 Some popular Mercer kernels

¹³ In the sections below, we describe some popular Mercer kernels. More details can be found at [Wil14]
¹⁴ and <https://www.cs.toronto.edu/~duvenaud/cookbook/>. See also Section 18.6 where we discuss
¹⁵ how to learn kernels from data.

¹⁶

¹⁷ 18.2.1.1 Stationary kernels for real-valued vectors

¹⁸ For real-valued inputs, $\mathcal{X} = \mathbb{R}^D$, it is common to use **stationary kernels**, which are functions of
¹⁹ the form $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\mathbf{r})$, where $\mathbf{r} = \mathbf{x} - \mathbf{x}'$; thus the output only depends on the relative difference
²⁰ between the inputs. Furthermore, in many cases, it is only the magnitude of the difference, $r = \|\mathbf{r}\|_2$,
²¹ that matters. We give some examples below.

²²

²³ Squared exponential kernel

²⁴ The **squared exponential** (SE) kernel is defined as

$$\mathcal{K}(r; \ell) = \exp\left(-\frac{r^2}{2\ell^2}\right) \quad (18.8)$$

²⁵ Here ℓ corresponds to the length-scale of the kernel, i.e., the distance over which we expect differences
²⁶ to matter.

²⁷

²⁸ ARD kernel

²⁹ We can generalize the RBF kernel by replacing Euclidean distance with Mahalanobis distance, as
³⁰ follows:

$$\mathcal{K}(\mathbf{r}; \Sigma, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2} \mathbf{r}^\top \Sigma^{-1} \mathbf{r}\right) \quad (18.9)$$

³¹

³² where $\mathbf{r} = \mathbf{x} - \mathbf{x}'$. If Σ is diagonal, this can be written as

$$\mathcal{K}(\mathbf{r}; \ell, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2} \sum_{d=1}^D \frac{1}{\ell_d^2} r_d^2\right) = \prod_{d=1}^D \mathcal{K}(r_d; \ell_d, \sigma^{2/d}) \quad (18.10)$$

³³

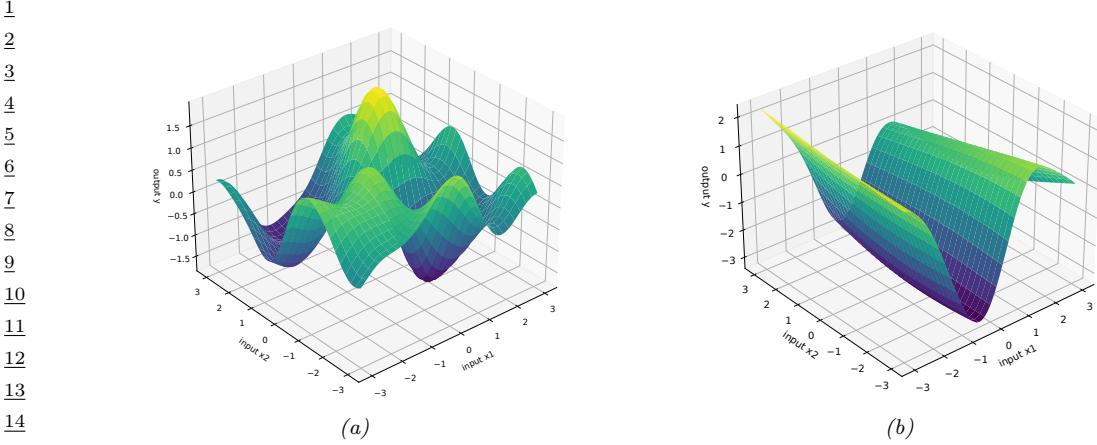


Figure 18.2: Function samples from a GP with an ARD kernel. (a) $\ell_1 = \ell_2 = 1$. Both dimensions contribute to the response. (b) $\ell_1 = 1, \ell_2 = 5$. The second dimension is essentially ignored. Adapted from Figure 5.1 of [RW06]. Generated by [gprDemoArd.py](#).

where

$$\mathcal{K}(r; \ell, \sigma^2) = \sigma^2 \exp\left(-\frac{1}{2} \frac{1}{\ell^2} r^2\right) \quad (18.11)$$

We can interpret σ^2 as the overall variance, and ℓ_d as defining the **characteristic length scale** of dimension d . If d is an irrelevant input dimension, we can set $\ell_d = \infty$, so the corresponding dimension will be ignored. This is known as **Automatic Relevance Determination** or **ARD** (Section 15.2.7). Hence the corresponding kernel is called the **ARD kernel**. See Figure 18.2 for an illustration of some 2d functions sampled from a GP using this prior.

Matern kernels

The SE kernel gives rise to functions that are infinitely differentiable, and therefore are very smooth. For many applications, it is better to use the **Matern kernel**, which gives rise to “rougher” functions, which can better model local “wiggles” without having to make the overall length scale very small.

The Matern kernel has the following form:

$$\mathcal{K}(r; \nu, \ell) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}r}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}r}{\ell} \right) \quad (18.12)$$

where K_ν is a modified Bessel function and ℓ is the length scale. Functions sampled from this GP are k -times differentiable iff $\nu > k$. As $\nu \rightarrow \infty$, this approaches the SE kernel.

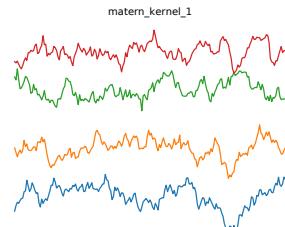
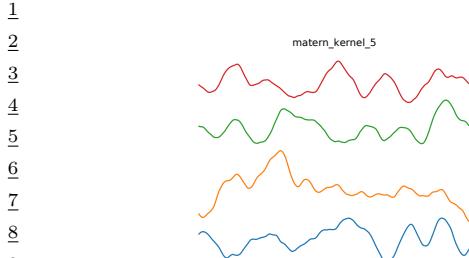


Figure 18.3: Functions sampled from a GP with a Matern kernel. (a) $\nu = 5/2$. (b) $\nu = 1/2$. Generated by `gpKernelPlot.py`.

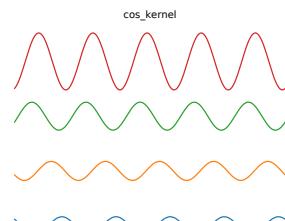
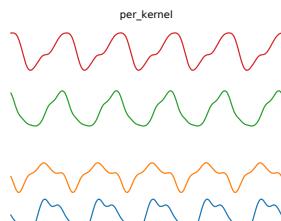


Figure 18.4: Functions sampled from a GP using various stationary periodic kernels. Generated by `gpKernelPlot.py`.

For values $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$, the function simplifies as follows:

$$\mathcal{K}(r; \frac{1}{2}, \ell) = \exp\left(-\frac{r}{\ell}\right) \quad (18.13)$$

$$\mathcal{K}(r; \frac{3}{2}, \ell) = \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right) \quad (18.14)$$

$$\mathcal{K}(r; \frac{5}{2}, \ell) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right) \quad (18.15)$$

The value $\nu = \frac{1}{2}$ corresponds to the **Ornstein-Uhlenbeck process**, which describes the velocity of a particle undergoing Brownian motion. The corresponding function is continuous but not differentiable, and hence is very “jagged”. See Figure 18.3b for an illustration.

2 Periodic kernels

3 One way to create a periodic 1d random function is to map x to the 2d space $\mathbf{u}(x) = (\cos(x), \sin(x))$,
4 and then use an SE kernel in \mathbf{u} -space:

$$5 \quad \mathcal{K}(x, x') = \exp\left(-\frac{2 \sin^2((x - x')/2)}{\ell^2}\right) \quad (18.16)$$

6 which follows since $(\cos(x) - \cos(x'))^2 + (\sin(x) - \sin(x'))^2 = 4 \sin^2((x - x')/2)$. We can generalize this
7 by specifying the period p to get the **periodic kernel**, also called the **exp-sine-squared kernel**:

$$8 \quad \mathcal{K}_{\text{per}}(r; \ell, p) = \exp\left(-\frac{2}{\ell^2} \sin^2\left(\pi \frac{r}{p}\right)\right) \quad (18.17)$$

9 where p is the period and ℓ is the length scale. See Figure 18.4a for an illustration.

10 A related kernel is the **cosine kernel**:

$$11 \quad \mathcal{K}(r; p) = \cos\left(2\pi \frac{r}{p}\right) \quad (18.18)$$

12 See Figure 18.4b for an illustration.

22 Rational quadratic kernel

23 We define the **rational quadratic** kernel to be

$$24 \quad \mathcal{K}_{RQ}(r; \ell, \alpha) = \left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha} \quad (18.19)$$

25 We recognize this is proportional to a Student T density. Hence it can be interpreted as a scale
26 mixture of SE kernels of different characteristic lengths.

31 18.2.1.2 Making new kernels from old

32 Given two valid kernels $\mathcal{K}_1(\mathbf{x}, \mathbf{x}')$ and $\mathcal{K}_2(\mathbf{x}, \mathbf{x}')$, we can create a new kernel using any of the following
33 methods:

$$35 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = c\mathcal{K}_1(\mathbf{x}, \mathbf{x}'), \text{ for any constant } c > 0 \quad (18.20)$$

$$36 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\mathcal{K}_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}'), \text{ for any function } f \quad (18.21)$$

$$37 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = q(\mathcal{K}_1(\mathbf{x}, \mathbf{x}')) \text{ for any function polynomial } q \text{ with nonneg. coef.} \quad (18.22)$$

$$38 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp(\mathcal{K}_1(\mathbf{x}, \mathbf{x}')) \quad (18.23)$$

$$39 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{A} \mathbf{x}', \text{ for any psd matrix } \mathbf{A} \quad (18.24)$$

40 For example, suppose we start with the linear kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x}\mathbf{x}'$. We know this is a valid
41 Mercer kernel, since the corresponding Gram matrix is just the (scaled) covariance matrix of the
42 data. From the above rules, we can see that the **polynomial kernel**

$$43 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^M \quad (18.25)$$

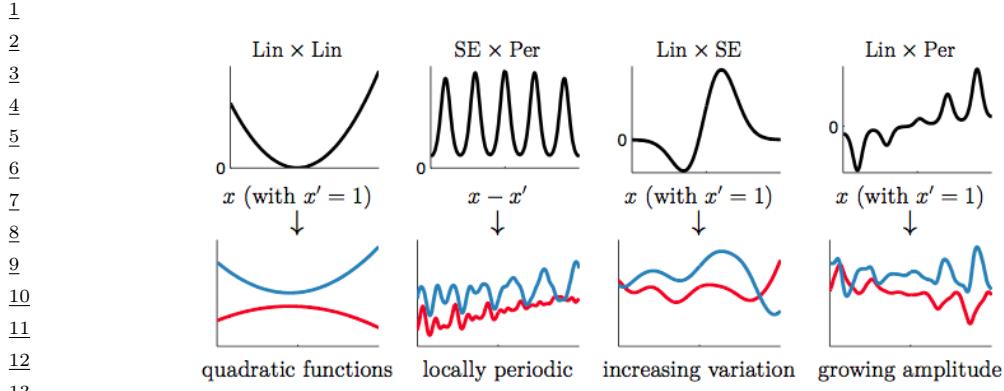


Figure 18.5: Examples of 1d structures obtained by multiplying elementary kernels. Top row shows $\mathcal{K}(x, x' = 1)$. Bottom row shows some functions sampled from $GP(f|0, \mathcal{K})$. From Figure 2.2 of [Duv14]. Used with kind permission of David Duvenaud.

is a valid Mercer kernel. This contains all monomials of order M . For example, if $M = 2$ and the inputs are 2d, we have

$$(22) \quad (\mathbf{x}^\top \mathbf{x}')^2 = (x_1 x'_1 + x_2 x'_2)^2 = (x_1 x'_1)^2 + (x_2 x'_2)^2 + (x_1 x'_1)(x_2 x'_2) \quad (18.26)$$

We can generalize this to contain all terms up to degree M by using the kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^M$. For example, if $M = 2$ and the inputs are 2d, we have

$$(26) \quad \begin{aligned} (\mathbf{x}^\top \mathbf{x}' + 1)^2 &= (x_1 x'_1)^2 + (x_1 x'_1)(x_2 x'_2) + (x_1 x'_1) \\ &\quad + (x_2 x'_2)(x_1 x'_1) + (x_2 x'_2)^2 + (x_2 x'_2) \\ &\quad + (x_1 x'_1) + (x_2 x'_2) + 1 \end{aligned} \quad (18.27)$$

We can also use the above rules to establish that the Gaussian kernel is a valid kernel. To see this, note that

$$(34) \quad \|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^\top \mathbf{x} + (\mathbf{x}')^\top \mathbf{x}' - 2\mathbf{x}^\top \mathbf{x}' \quad (18.28)$$

and hence

$$(37) \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2) = \exp(-\mathbf{x}^\top \mathbf{x} / 2\sigma^2) \exp(\mathbf{x}'^\top \mathbf{x}' / \sigma^2) \exp(-(\mathbf{x}')^\top \mathbf{x}' / 2\sigma^2) \quad (18.29)$$

is a valid kernel.

40

41 18.2.1.3 Combining kernels by addition and multiplication

42 We can also combine kernels using addition or multiplication:

$$(44) \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_1(\mathbf{x}, \mathbf{x}') + \mathcal{K}_2(\mathbf{x}, \mathbf{x}') \quad (18.30)$$

$$(45) \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_1(\mathbf{x}, \mathbf{x}') \times \mathcal{K}_2(\mathbf{x}, \mathbf{x}') \quad (18.31)$$

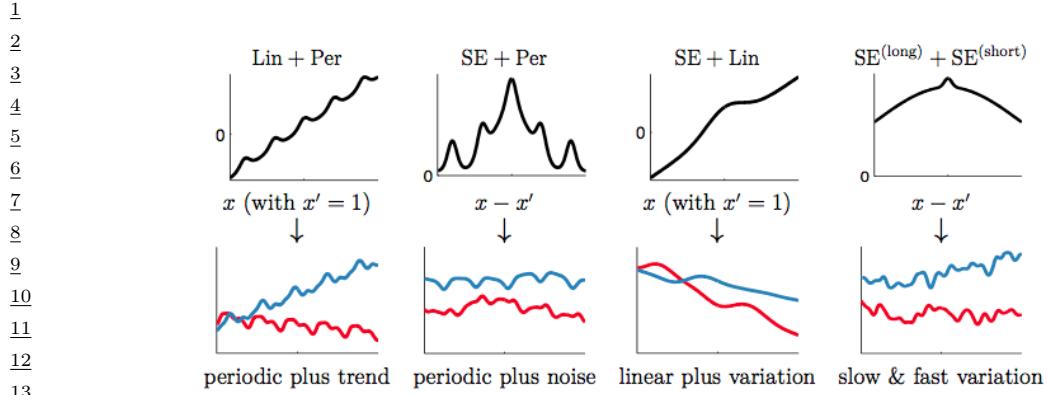


Figure 18.6: Examples of 1d structures obtained by adding elementary kernels. Here $\text{SE}^{(\text{short})}$ and $\text{SE}^{(\text{long})}$ are two SE kernels with different length scales. From Figure 2.4 of [Duv14]. Used with kind permission of David Duvenaud.

Multiplying two positive-definite kernels together always results in another positive definite kernel. This is a way to get a conjunction of the individual properties of each kernel, as illustrated in Figure 18.5.

In addition, adding two positive-definite kernels together always results in another positive definite kernel. This is a way to get a disjunction of the individual properties of each kernel, as illustrated in Figure 18.6.

For an example of combining kernels to forecast some timeseries data, see Section 19.3.3.1.

18.2.1.4 Kernels for structured inputs

Kernels are particularly useful when the inputs are structured objects, such as strings and graphs, since it is often hard to “featurize” variable-sized inputs. For example, we can define a **string kernel** which compares strings in terms of the number of n-grams they have in common [Lod+02; BC17].

We can also define kernels on graphs [KJM19]. For example, the **random walk kernel** conceptually performs random walks on two graphs simultaneously, and then counts the number of paths that were produced by both walks. This can be computed efficiently as discussed in [Vis+10]. For more details on graph kernels, see [KJM19].

For a review of kernels on structured objects, see e.g., [Gär03].

18.2.2 Mercer’s theorem

Recall that any positive definite matrix \mathbf{K} can be represented using an eigendecomposition of the form $\mathbf{K} = \mathbf{U}^T \boldsymbol{\Lambda} \mathbf{U}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues $\lambda_i > 0$, and \mathbf{U} is a matrix containing the eigenvectors. Now consider element (i, j) of \mathbf{K} :

$$k_{ij} = (\boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:i})^T (\boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:j}) \quad (18.32)$$

¹ where $\mathbf{U}_{:,i}$ is the i 'th column of \mathbf{U} . If we define $\phi(\mathbf{x}_i) = \Lambda^{\frac{1}{2}}\mathbf{U}_{:,i}$, then we can write

$$\begin{aligned} k_{ij} &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \sum_{m=1}^M \phi_m(\mathbf{x}_i) \phi_m(\mathbf{x}_j) \end{aligned} \quad (18.33)$$

⁶ where M is the rank of the kernel matrix. Thus we see that the entries in the kernel matrix can be
⁷ computed by performing an inner product of some feature vectors that are implicitly defined by the
⁸ eigenvectors of the kernel matrix.
⁹

¹⁰ This idea can be generalized to apply to kernel functions, not just kernel matrices, as we now show.
¹¹ First, we define an **eigenfunction** $\phi()$ of a kernel \mathcal{K} with eigenvalue λ wrt measure μ as a function
¹² that satisfies

$$\int \mathcal{K}(\mathbf{x}, \mathbf{x}') \phi(\mathbf{x}) d\mu(\mathbf{x}) = \lambda \phi(\mathbf{x}') \quad (18.34)$$

¹⁵ We usually sort the eigenfunctions in order of decreasing eigenvalue, $\lambda_1 \geq \lambda_2 \geq \dots$. The eigenfunctions
¹⁶ are orthogonal wrt μ :
¹⁷

$$\int \phi_i(\mathbf{x}) \phi_j(\mathbf{x}) d\mu(\mathbf{x}) = \delta_{ij} \quad (18.35)$$

²⁰ where δ_{ij} is the Kronecker delta. With this definition in hand, we can state **Mercer's theorem**.
²¹ Informally, it says that any positive definite kernel function can be represented as the following
²² infinite sum:
²³

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{m=1}^{\infty} \lambda_m \phi_m(\mathbf{x}) \phi_m(\mathbf{x}') \quad (18.36)$$

²⁶ where ϕ_m are eigenfunctions of the kernel, and λ_m are the corresponding eigenvalues. This is the
²⁷ functional analog of Equation (18.33).

²⁹ A **degenerate kernel** has only a finite number of non-zero eigenvalues. In this case, we can
³⁰ rewrite the kernel function as an inner product between two finite-length vectors. For example,
³¹ consider the **quadratic kernel** $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^2$. In 2d, we have

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = (x_1 x'_1 + x_2 x'_2)^2 = x_1^2(x'_1)^2 + 2x_1 x_2 x'_1 x'_2 + x_2^2(x'_2)^2 \quad (18.37)$$

³⁴ If we define $\phi(x_1, x_2) = [x_1^2, \sqrt{2}x_1 x_2, x_2^2] \in \mathbb{R}^3$, then we can write this as $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$.
³⁵ Thus we see that this kernel is degenerate.

³⁶ Now consider the RBF kernel. In this case, the corresponding feature representation is infinite
³⁷ dimensional (see Section 18.2.3.1 for details). However, by working with kernel functions, we can
³⁸ avoid having to deal with infinite dimensional vectors.

³⁹ From the above, we see that we can replace inner product operations in an explicit (possibly infinite
⁴⁰ dimensional) feature space with a call to a kernel function, i.e., we replace $\phi(\mathbf{x})^T \phi(\mathbf{x})$ with $\mathcal{K}(\mathbf{x}, \mathbf{x}')$.
⁴¹ This is called the **kernel trick**.

⁴²

⁴³ 18.2.3 Kernels from Spectral Densities

⁴⁴ Consider the case of a **shift-invariant kernel**, also known as a *stationary kernel*, which satisfies
⁴⁵ $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\boldsymbol{\delta})$, where $\boldsymbol{\delta} = \mathbf{x} - \mathbf{x}'$, for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$. Let us further assume that $\mathcal{K}(\boldsymbol{\delta})$ is positive definite.
⁴⁶

1 In this case, **Bochner's theorem** tells us that we can represent $\mathcal{K}(\delta)$ by its Fourier transform:

$$\mathcal{K}(\delta) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^\top \delta} d\omega \quad (18.38)$$

7 where $j = \sqrt{-1}$, $e^{j\theta} = \cos(\theta) + j \sin(\theta)$, and $p(\omega)$, the **spectral density**, is a distribution over
8 frequencies.

9 We can easily derive and gain intuitions into several kernels from spectral densities. If we take the
10 Fourier transform of an RBF kernel we find the spectral density $p(\omega) = \sqrt{2\pi\ell^2} \exp(-2\pi^2\omega^2\ell^2)$. Thus
11 the spectral density is also Gaussian, but with a bandwidth *inversely* proportional to the length-scale
12 hyperparameter ℓ . That is, as ℓ becomes large, the spectral density collapses onto a point mass.
13 This result is intuitive: as we increase the length-scale our model treats points as correlated over
14 large distances, and becomes very smooth and slowly varying, and thus low-frequency. In general,
15 since the Gaussian distribution has relatively light tails, we can see that RBF kernels won't generally
16 support high frequency solutions.

17 We can instead use a Student- t spectral density, which has heavy tails that will provide greater
18 support for higher frequencies. Taking the inverse Fourier transform of this spectral density, we
19 recover the Matern kernel, with degrees of freedom ν corresponding to the degrees of freedom in
20 the spectral density. Indeed, the smaller we make ν the less smooth and higher frequency are the
21 associated fits to data using a Matern kernel.

22 We can also derive **spectral mixture kernels** by modelling the spectral density as a scale-location
23 mixture of Gaussians and taking the inverse Fourier transform [WA13]. Since scale-location mixtures
24 of Gaussians are dense in the set of distributions, and can therefore approximate any spectral density,
25 this kernel can approximate any stationary kernel to arbitrary precision. The spectral mixture kernel
26 thus forms a powerful approach to kernel learning, which we discuss further in Section 18.6.5.

29 18.2.3.1 Random feature kernels

31 Although the power of kernels resides in the ability to avoid working with featurized representations
32 of the inputs, such kernelized methods can take $O(N^3)$ time, in order to invert the Gram matrix \mathbf{K} ,
33 This can make it difficult to use such methods on large scale data. Fortunately, we can approximate
34 the feature map for many kernels using a randomly chosen finite set of M basis functions, thus
35 reducing the cost to $O(NM + M^3)$.

36 We will show how to do this for shift-invariant kernels by returning to Bochner's theorem in
37 Eq. (18.38):

$$\mathcal{K}(\delta) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^\top \delta} d\omega \quad (18.39)$$

43 where $j = \sqrt{-1}$, $e^{j\theta} = \cos(\theta) + j \sin(\theta)$, and the spectral density $p(\omega)$ is a distribution over frequencies.

44 In the case of a Gaussian RBF kernel, we have seen that the spectral density is a Gaussian
45 distribution. Hence we can easily compute a Monte Carlo approximation to this integral by sampling
46 random Gaussian vectors. This yields the following approximation: $\mathcal{K}(\mathbf{x}, \mathbf{x}') \approx \phi(\mathbf{x})^\top \phi(\mathbf{x}')$, where

¹
² the (real-valued) feature vector is given by

³
⁴ $\phi(\mathbf{x}) = \sqrt{\frac{1}{D}} [\sin(\mathbf{z}_1^\top \mathbf{x}), \dots, \sin(\mathbf{z}_D^\top \mathbf{x}), \cos(\mathbf{z}_1^\top \mathbf{x}), \dots, \cos(\mathbf{z}_D^\top \mathbf{x})]$ (18.40)

⁵
⁶ $= \sqrt{\frac{1}{D}} [\sin(\mathbf{Z}^\top \mathbf{x}), \cos(\mathbf{Z}^\top \mathbf{x})]$ (18.41)

⁷
⁸ Here $\mathbf{Z} = (1/\sigma)\mathbf{G}$, and $\mathbf{G} \in \mathbb{R}^{d \times D}$ is a random Gaussian matrix, where the entries are sampled
⁹ iid from $\mathcal{N}(0, 1)$. The representation in Equation (18.41) are called **random Fourier features**
¹⁰ (**RFF**) [RR08] or “weighted sums of random kitchen sinks” [RR09]. (One can obtain an even better
¹¹ approximation by ensuring that the rows of \mathbf{Z} are random but orthogonal; this is called **orthogonal**
¹² **random features** [Yu+16].)

¹³
¹⁴ One can create similar random feature representations for other kinds of kernels. We can then
¹⁵ use such features for supervised learning by defining $f(\mathbf{x}; \theta) = \mathbf{W}\varphi(\mathbf{Z}\mathbf{x}) + \mathbf{b}$, where \mathbf{Z} is a random
¹⁶ Gaussian matrix, and the form of φ depends on the chosen kernel. This is equivalent to a one layer
¹⁷ MLP with random input-to-hidden weights; since we only optimize the hidden-to-output weights
¹⁸ $\theta = (\mathbf{W}, \mathbf{b})$, the model is equivalent to a linear model with fixed random features. If we use enough
¹⁹ random features, we can approximate the performance of a kernelized prediction model, but the
²⁰ computational cost is now $O(N)$ rather than $O(N^2)$.

²¹ ²² 18.3 GPs with Gaussian likelihoods

²³
²⁴ In this section, we discuss GPs for regression, using a Gaussian likelihood. In this case, all the
²⁵ computations can be performed in closed form, using standard linear algebra methods. We extend
²⁶ this framework to non-Gaussian likelihoods later in the chapter.

²⁷

²⁸ ²⁹ 18.3.1 Predictions using noise-free observations

³⁰

³¹ Suppose we observe a training set $\mathcal{D} = \{(\mathbf{x}_n, y_n) : n = 1 : N\}$, where $y_n = f(\mathbf{x}_n)$ is the noise-free
³² observation of the function evaluated at \mathbf{x}_n . If we ask the GP to predict $f(\mathbf{x})$ for a value of \mathbf{x} that it
³³ has already seen, we want the GP to return the answer $f(\mathbf{x})$ with no uncertainty. In other words, it
³⁴ should act as an **interpolator** of the training data. Here we assume the observed function values
³⁵ are noiseless. We will consider the case of noisy observations shortly.

³⁶ Now we consider the case of predicting the outputs for new inputs that may not be in \mathcal{D} . Specifically,
³⁷ given a test set \mathbf{X}_* of size $N_* \times D$, we want to predict the function outputs $\mathbf{f}_* = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_{N_*})]$.
³⁸ By definition of the GP, the joint distribution $p(\mathbf{f}_X, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$ has the following form

³⁹
⁴⁰
$$\begin{pmatrix} \mathbf{f}_X \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\top & \mathbf{K}_{*,*} \end{pmatrix} \right)$$
 (18.42)

⁴¹

⁴² where $\boldsymbol{\mu}_X = (m(\mathbf{x}_1), \dots, m(\mathbf{x}_N))$, $\boldsymbol{\mu}_* = (m(\mathbf{x}_1^*), \dots, m(\mathbf{x}_{N_*}^*))$, $\mathbf{K}_{X,X} = \mathcal{K}(\mathbf{X}, \mathbf{X})$ is $N \times N$, $\mathbf{K}_{X,*} =$
⁴³ $\mathcal{K}(\mathbf{X}, \mathbf{X}_*)$ is $N \times N_*$, and $\mathbf{K}_{*,*} = \mathcal{K}(\mathbf{X}_*, \mathbf{X}_*)$ is $N_* \times N_*$. See Figure 18.1 for a static illustration,
⁴⁴ and <http://www.infinitecuriosity.org/vizgp/> for an interactive visualization.

⁴⁵
⁴⁶ By the standard rules for conditioning Gaussians (Section 2.3.3), the posterior has the following
⁴⁷

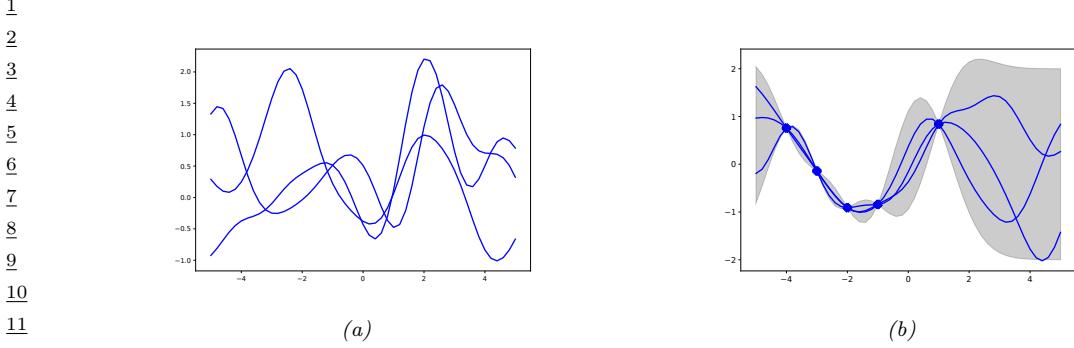


Figure 18.7: Left: some functions sampled from a GP prior with RBF kernel. Right: some samples from a GP posterior, after conditioning on 5 noise-free observations. The shaded area represents $\mathbb{E}[f(\mathbf{x})] \pm 2\text{std}[f(\mathbf{x})]$. Adapted from Figure 2.2 of [RW06]. Generated by `gprDemoNoiseFree.py`.

form

$$p(\mathbf{f}_* | \mathbf{X}_*, \mathcal{D}) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}) \quad (18.43)$$

$$\boldsymbol{\mu}_{*|X} = \boldsymbol{\mu}_X + \mathbf{K}_{X,*}^T \mathbf{K}_{X,X}^{-1} (\mathbf{f}_X - \boldsymbol{\mu}_*) \quad (18.44)$$

$$\boldsymbol{\Sigma}_{*|X} = \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^T \mathbf{K}_{X,X}^{-1} \mathbf{K}_{X,*} \quad (18.45)$$

This process is illustrated in Figure 18.7. On the left we show some samples from the prior, $p(f)$, where we use an RBF kernel (Section 18.2) and a zero mean function. On the right, we show samples from the posterior, $p(f|\mathcal{D})$. We see that the model perfectly interpolates the training data, and that the predictive uncertainty increases as we move further away from the observed data.

Note that the cost of the above method for sampling N_* points is $O(N_*^3)$. This can be reduced to $O(N_*)$ time using the methods in [Ple+18; Wil+20a].

18.3.2 Predictions using noisy observations

In Section 18.3.1, we showed how to do GP regression when the training data was noiseless. Now let us consider the case where what we observe is a noisy version of the underlying function, $y_n = f(\mathbf{x}_n) + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, \sigma_y^2)$. In this case, the model is not required to interpolate the data, but it must come “close” to the observed data. The covariance of the observed noisy responses is

$$\text{Cov}[y_i, y_j] = \text{Cov}[f_i, f_j] + \text{Cov}[\epsilon_i, \epsilon_j] = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) + \sigma_y^2 \delta_{ij} \quad (18.46)$$

where $\delta_{ij} = \mathbb{I}(i = j)$. In other words

$$\text{Cov}[\mathbf{y} | \mathbf{X}] = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I}_N \quad (18.47)$$

The joint density of the observed data and the latent, noise-free function on the test points is given by

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I} & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^T & \mathbf{K}_{*,*} \end{pmatrix}\right) \quad (18.48)$$

1 Hence the posterior predictive density at a set of test points \mathbf{X}_* is
2

$$\underline{3} \quad p(\mathbf{f}_* | \mathcal{D}, \mathbf{X}_*) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}) \quad (18.49)$$

$$\underline{4} \quad \boldsymbol{\mu}_{*|X} = \boldsymbol{\mu}_* + \mathbf{K}_{X,*}^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\mu}_X) \quad (18.50)$$

$$\underline{5} \quad \boldsymbol{\Sigma}_{*|X} = \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{K}_{X,*} \quad (18.51)$$

6 In the case of a single test input, this simplifies as follows
7

$$\underline{8} \quad p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | m_* + \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\mu}_X), k_{**} - \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{k}_*) \quad (18.52)$$

9 where $\mathbf{k}_* = [\mathcal{K}(\mathbf{x}_*, \mathbf{x}_1), \dots, \mathcal{K}(\mathbf{x}_*, \mathbf{x}_N)]$ and $k_{**} = \mathcal{K}(\mathbf{x}_*, \mathbf{x}_*)$. If the mean function is zero, we can
10 write the posterior mean as follows:

$$\underline{11} \quad \boldsymbol{\mu}_{*|X} = \mathbf{k}_*^\top \underbrace{\mathbf{K}_\sigma^{-1} \mathbf{y}}_{\boldsymbol{\alpha}} = \sum_{n=1}^N \mathcal{K}(\mathbf{x}_*, \mathbf{x}_n) \alpha_n \quad (18.53)$$

12 where
13

$$\underline{14} \quad \mathbf{K}_\sigma = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I} \quad (18.54)$$

$$\underline{15} \quad \boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1} \mathbf{y} \quad (18.55)$$

16 Fitting this model amounts to computing $\boldsymbol{\alpha}$ in Equation (18.55). This is usually done by computing
17 the Cholesky decomposition of \mathbf{K}_σ , as described in Section 18.3.6. Once we have computed $\boldsymbol{\alpha}$, we
18 can compute predictions for each test point in $O(N)$ time for the mean, and $O(N^2)$ time for the
19 variance.
20

21 18.3.3 Weight space vs function space

22 In this section, we show how Bayesian linear regression is a special case of a GP.

23 Consider the linear regression model $y = f(\mathbf{x}) + \epsilon$, where $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$ and $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$. If we
24 use a Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \boldsymbol{\Sigma}_w)$, then the posterior is as follows (see Section 15.2.1 for the
25 derivation):

$$\underline{26} \quad p(\mathbf{w} | \mathcal{D}) = \mathcal{N}(\mathbf{w} | \frac{1}{\sigma_y^2} \mathbf{A}^{-1} \boldsymbol{\Phi}^\top \mathbf{y}, \mathbf{A}^{-1}) \quad (18.56)$$

27 where $\boldsymbol{\Phi}$ is the $N \times D$ design matrix, and
28

$$\underline{29} \quad \mathbf{A} = \sigma_y^{-2} \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \boldsymbol{\Sigma}_w^{-1} \quad (18.57)$$

30 The posterior predictive distribution for $f_* = f(\mathbf{x}_*)$ is therefore
31

$$\underline{32} \quad p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | \frac{1}{\sigma_y^2} \boldsymbol{\phi}_*^\top \mathbf{A}^{-1} \boldsymbol{\Phi}^\top \mathbf{y}, \boldsymbol{\phi}_*^\top \mathbf{A}^{-1} \boldsymbol{\phi}_*) \quad (18.58)$$

33 where $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$. This views the problem of inference and prediction in **weight space**.

34

We now show that this is equivalent to the predictions made by a GP using a kernel of the form $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \Sigma_w \phi(\mathbf{x}')$. To see this, let $\mathbf{K} = \Phi \Sigma_w \Phi^\top$, $\mathbf{k}_* = \Phi \Sigma_w \phi_*$, and $k_{**} = \phi_*^\top \Sigma_w \phi_*$. Using this notation, and the matrix inversion lemma, we can rewrite Equation (18.58) as follows

$$p(f_* | \mathcal{D}, \mathbf{x}_*) = \mathcal{N}(f_* | \boldsymbol{\mu}_{*|X}, \boldsymbol{\Sigma}_{*|X}) \quad (18.59)$$

$$\boldsymbol{\mu}_{*|X} = \phi_*^\top \Sigma_w \Phi^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} = \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} \quad (18.60)$$

$$\boldsymbol{\Sigma}_{*|X} = \phi_*^\top \Sigma_w \phi_* - \phi_*^\top \Sigma_w \Phi^\top (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \Phi \Sigma_w \phi_* = k_{**} - \mathbf{k}_*^\top (\mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{k}_* \quad (18.61)$$

which matches the results in Equation (18.52), assuming $m(\mathbf{x}) = 0$. A non-zero mean can be captured by adding a constant feature with value 1 to $\phi(\mathbf{x})$.

Thus we can derive a GP from Bayesian linear regression. Note, however, that linear regression assumes $\phi(\mathbf{x})$ is a finite length vector, whereas a GP allows us to work directly in terms of kernels, which may correspond to infinite length feature vectors (see Section 18.2.2). That is, a GP works in **function space**.

18.3.4 Semi-parametric GPs

So far, we have mostly assumed the mean of the GP is 0, and have relied on its interpolation abilities to model the mean function. Sometimes it is useful to fit a global linear model for the mean, and use the GP to model the residual errors, as follows:

$$g(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\beta}^\top \phi(\mathbf{x}) \quad (18.62)$$

where $f(\mathbf{x}) \sim \text{GP}(0, \mathcal{K}(\mathbf{x}, \mathbf{x}'))$, and $\boldsymbol{\beta}$ are some fixed basis functions. This combines a parametric and a non-parametric model, and is known as a **semi-parametric model**.

If we assume $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{b}, \mathbf{B})$, we can integrate these parameters out to get a new GP [O'H78]:

$$g(\mathbf{x}) \sim \text{GP}(\phi(\mathbf{x})^\top \mathbf{b}, \mathcal{K}(\mathbf{x}, \mathbf{x}') + \phi(\mathbf{x})^\top \mathbf{B} \phi(\mathbf{x}')) \quad (18.63)$$

Let $\mathbf{H}_X = \phi(\mathbf{X})^\top$ be the $D \times N$ matrix of training examples, and $\mathbf{H}_* = \phi(\mathbf{X}_*)^\top$ be the $D \times N_*$ matrix of test examples. The corresponding predictive distribution for test inputs \mathbf{X}_* has the following form [RW06, p28]:

$$\mathbb{E}[g(\mathbf{X}_*) | \mathcal{D}] = \mathbf{H}_*^\top \bar{\boldsymbol{\beta}} + \mathbf{K}_{X,*}^\top \mathbf{K}_\sigma^{-1} (\mathbf{y} - \mathbf{H}_X^\top \bar{\boldsymbol{\beta}}) = \mathbb{E}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top \bar{\boldsymbol{\beta}} \quad (18.64)$$

$$\text{Cov}[g(\mathbf{X}_*) | \mathcal{D}] = \text{Cov}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top (\mathbf{B}^{-1} + \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} \mathbf{R} \quad (18.65)$$

$$\bar{\boldsymbol{\beta}} = (\mathbf{B}^{-1} + \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} (\mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{y} + \mathbf{B}^{-1} \mathbf{b}) \quad (18.66)$$

$$\mathbf{R} = \mathbf{H}_* - \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{K}_{X,*} \quad (18.67)$$

These results can be interpreted as follows: the mean is the usual mean from the GP, plus a global offset from the linear model, using $\bar{\boldsymbol{\beta}}$; and the covariance is the usual covariance from the GP, plus an additional positive term due to the uncertainty in $\boldsymbol{\beta}$.

In the limit of an uninformative prior for the regression parameters, as $\mathbf{B} \rightarrow \infty \mathbf{I}$, this simplifies to

$$\mathbb{E}[g(\mathbf{X}_*) | \mathcal{D}] = \mathbb{E}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top (\mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} \mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{y} \quad (18.68)$$

$$\text{Cov}[g(\mathbf{X}_*) | \mathcal{D}] = \text{Cov}[f(\mathbf{X}_*) | \mathcal{D}] + \mathbf{R}^\top (\mathbf{H}_X \mathbf{K}_\sigma^{-1} \mathbf{H}_X^\top)^{-1} \mathbf{R} \quad (18.69)$$

18.3.5 Marginal likelihood

3 Most kernels have some free parameters. For example, the RBF-ARD kernel (Section 18.2.1.1) has
4 the form

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp \left(-\frac{1}{2} \sum_{d=1}^D \frac{1}{\ell_d^2} (x_d - x'_d)^2 \right) = \prod_{d=1}^D \mathcal{K}_{\ell_d}(x_d, x'_d) \quad (18.70)$$

9 where each ℓ_d is a length scale for feature dimension d . Let these (and the observation noise variance
10 σ_y^2 , if present) be denoted by $\boldsymbol{\theta}$. We can compute the likelihood of these parameters as follows:

$$\mathbf{p}(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{f}_X, \boldsymbol{\theta}) p(\mathbf{f}_X|\mathbf{X}, \boldsymbol{\theta}) d\mathbf{f}_X \quad (18.71)$$

15 Since we are integrating out the function f , we often call $\boldsymbol{\theta}$ hyperparameters, and the quantity $p(\mathcal{D}|\boldsymbol{\theta})$
16 the marginal likelihood.

17 Since f is a GP, we can compute the above integral using the marginal likelihood for the corre-
18 sponding Gaussian. This gives

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = -\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}_X)^\top \mathbf{K}_\sigma^{-1} (\mathbf{y} - \boldsymbol{\mu}_X) - \frac{1}{2} \log |\mathbf{K}_\sigma| - \frac{N}{2} \log(2\pi) \quad (18.72)$$

23 The first term term is the square of the Mahalanobis distance between the observations and the
24 predicted values: better fits will have smaller distance. The second term is the log determinant
25 of the covariance matrix, which measures model complexity: smoother functions will have smaller
26 determinants, so $-\log |\mathbf{K}_\sigma|$ will be larger (less negative) for simpler functions. The marginal likelihood
27 measures the tradeoff between fit and complexity.

28 In Section 18.6.1, we discuss how to learn the kernel parameters from data by maximizing the
29 marginal likelihood wrt $\boldsymbol{\theta}$.

30

18.3.6 Computational and numerical issues

33 In this section, we discuss computational and numerical issues which arise when implementing the
34 above equations. For notational simplicity, we assume the prior mean is zero, $m(\mathbf{x}) = 0$.

35 The posterior predictive mean is given by $\boldsymbol{\mu}_* = \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1} \mathbf{y}$. For reasons of numerical stability, it is
36 unwise to directly invert \mathbf{K}_σ . A more robust alternative is to compute a Cholesky decomposition,
37 $\mathbf{K}_\sigma = \mathbf{L}\mathbf{L}^\top$, which takes $O(N^3)$ time. Given this, we can compute

$$\boldsymbol{\mu}_* = \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1} \mathbf{y} = \mathbf{k}_*^\top \mathbf{L}^{-\top} (\mathbf{L}^{-1} \mathbf{y}) = \mathbf{k}_*^\top \boldsymbol{\alpha} \quad (18.73)$$

40

41 Here $\boldsymbol{\alpha} = \mathbf{L}^\top \setminus (\mathbf{L} \setminus \mathbf{y})$, where we have used the backslash operator to represent backsubstitution.

42 We can compute the variance in $O(N^2)$ time for each test case using

$$\sigma_*^2 = k_{**} - \mathbf{k}_*^\top \mathbf{L}^{-T} \mathbf{L}^{-1} \mathbf{k}_* = k_{**} - \mathbf{v}^\top \mathbf{v} \quad (18.74)$$

43

44 where $\mathbf{v} = \mathbf{L} \setminus \mathbf{k}_*$.

45

1 Finally, the log marginal likelihood (needed for kernel learning, Section 18.6) can be computed
2 using
3

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_{n=1}^N \log L_{nn} - \frac{N}{2} \log(2\pi) \quad (18.75)$$

4
5 We see that overall cost is dominated by $O(N^3)$. We discuss faster, but approximate, methods in
6 Section 18.5.
7

8 18.3.7 Kernel ridge regression

9 The term **ridge regression** refers to linear regression with an ℓ_2 penalty on the regression weights:
10

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^N (y_n - f(\mathbf{x}_n; \mathbf{w}))^2 + \lambda \|\mathbf{w}\|_2^2 \quad (18.76)$$

11 where $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$. The solution for this is
12

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} = \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top + \lambda \mathbf{I} \right)^{-1} \left(\sum_{n=1}^N \mathbf{x}_n y_n \right) \quad (18.77)$$

13 In this section, we consider a function space version of this:
14

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2 + \lambda \|f\|^2 \quad (18.78)$$

15 For this to make sense, we have to define the function space \mathcal{F} and the norm $\|f\|$. If we use a
16 function space derived from a positive definite kernel function \mathcal{K} , the resulting method is called
17 **kernel ridge regression** (KRR). We will see that the resulting estimate $f^*(\mathbf{x}_*)$ is equivalent to
18 the posterior mean of a GP. We give the details below.
19

20 18.3.7.1 Reproducing kernel Hilbert spaces

21 In this section, we briefly introduce the relevant mathematical ‘‘machinery’’ needed to explain KRR.
22

23 Let $\mathcal{F} = \{f : \mathcal{X} \rightarrow \mathbb{R}\}$ be a space of real-valued functions. Elements of this space (i.e., functions)
24 can be added and scalar multiplied as if they were vectors. That is, if $f \in \mathcal{F}$ and $g \in \mathcal{F}$, then
25 $\alpha f + \beta g \in \mathcal{F}$ for $\alpha, \beta \in \mathbb{R}$. We can also define an **inner product** for \mathcal{F} , which is a mapping $\langle f, g \rangle \in \mathbb{R}$
26 which satisfies the following:
27

$$\langle \alpha f_1 + \beta f_2, g \rangle = \alpha \langle f_1, g \rangle + \beta \langle f_2, g \rangle \quad (18.79)$$

$$\langle f, g \rangle = \langle g, f \rangle \quad (18.80)$$

$$\langle f, f \rangle \geq 0 \quad (18.81)$$

$$\langle f, f \rangle = 0 \text{ iff } f(x) = 0 \text{ for all } x \in \mathcal{X} \quad (18.82)$$

1 We define the norm of a function using
2

$$\underline{3} \quad \underline{4} \quad \|f\| \triangleq \sqrt{\langle f, f \rangle} \quad (18.83)$$

5 A function space \mathcal{H} with an inner product operator is called a **Hilbert space**. (We also require
6 that the function space be complete, which means that every Cauchy sequence of functions $f_i \in \mathcal{H}$
7 has a limit that is also in \mathcal{H} .)

8 The most common Hilbert space is the space known as L^2 . To define this, we need to specify a
9 **measure** μ on the input space \mathcal{X} ; this is a function that assigns any (suitable) subset A of \mathcal{X} to a
10 positive number, such as its volume. This can be defined in terms of the density function $w : \mathcal{X} \rightarrow \mathbb{R}$,
11 as follows:

$$\underline{12} \quad \underline{13} \quad \mu(A) = \int_A w(x)dx \quad (18.84)$$

15 Thus we have $\mu(dx) = w(x)dx$. We can now define $L^2(\mathcal{X}, \mu)$ to be the space of functions $f : \mathcal{X} \rightarrow \mathbb{R}$
16 that satisfy
17

$$\underline{18} \quad \underline{19} \quad \int_{\mathcal{X}} f(x)^2 w(x)dx < \infty \quad (18.85)$$

20 This is known as the set of **square-integrable functions**. This space has an inner product defined
21 by
22

$$\underline{23} \quad \underline{24} \quad \langle f, g \rangle = \int_{\mathcal{X}} f(x)g(x)w(x)dx \quad (18.86)$$

26 We define a **Reproducing Kernel Hilbert Space** or **RKHS** as follows. Let \mathcal{H} be a Hilbert
27 space of functions $f : \mathcal{X} \rightarrow \mathbb{R}$. We say that \mathcal{H} is an RKHS endowed with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ if there
28 exists a (symmetric) **kernel function** $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ with the following properties:

- 29 • For every $\mathbf{x} \in \mathcal{X}$, $\mathcal{K}(\mathbf{x}, \cdot) \in \mathcal{H}$.
- 30 • \mathcal{K} satisfies the **reproducing property**:

$$\underline{31} \quad \underline{32} \quad \langle f(\cdot), \mathcal{K}(\cdot, \mathbf{x}') \rangle = f(\mathbf{x}') \quad (18.87)$$

33 The reason for the term “reproducing property” is as follows. Let $f(\cdot) = \mathcal{K}(\mathbf{x}, \cdot)$. Then we have that
34

$$\underline{35} \quad \langle \mathcal{K}(\mathbf{x}, \cdot), \mathcal{K}(\cdot, \mathbf{x}') \rangle = \mathcal{K}(\mathbf{x}, \mathbf{x}') \quad (18.88)$$

37 18.3.7.2 Complexity of a function in an RKHS

39 The main utility of RKHS from the point of view of machine learning is that it allows us to define a
40 notion of a function’s “smoothness” or “complexity” in terms of its norm, as we now discuss.

41 Suppose we have a positive definite kernel function \mathcal{K} . From Mercer’s theorem we have $\mathcal{K}(\mathbf{x}, \mathbf{x}') =$
42 $\sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$. Now consider a Hilbert space \mathcal{H} defined by functions of the form $f(\mathbf{x}) =$
43 $\sum_{i=1}^{\infty} f_i \phi_i(\mathbf{x})$, with $\sum_{i=1}^{\infty} f_i^2 / \lambda_i < \infty$. The inner product of two functions in this space is

$$\underline{44} \quad \underline{45} \quad \langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{f_i g_i}{\lambda_i} \quad (18.89)$$

1 Hence the (squared) norm is given by
2

$$\frac{4}{5} \|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{f_i^2}{\lambda_i} \quad (18.90)$$

7 This is analogous to the quadratic form $\mathbf{f}^T \mathbf{K}^{-1} \mathbf{f}$ which occurs in some GP objectives (see Equa-
8 tion (18.99)). Thus the smoothness of the function is controlled by the properties of the corresponding
9 kernel.

11 18.3.7.3 Representer theorem

12 In this section, we consider the problem of (regularized) empirical risk minimization in function space.
13 In particular, consider the following problem:

$$\frac{16}{17} f^* = \underset{f \in \mathcal{H}_{\mathcal{K}}}{\operatorname{argmin}} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 \quad (18.91)$$

19 where $\mathcal{H}_{\mathcal{K}}$ is an RKHS with kernel \mathcal{K} and $\ell(y, \hat{y}) \in \mathbb{R}$ is a loss function. Then one can show [KW70;
20 SHS01] the following result:

$$\frac{22}{23} f^*(x) = \sum_{n=1}^N \alpha_n \mathcal{K}(x, \mathbf{x}_n) \quad (18.92)$$

25 where $\alpha_n \in \mathbb{R}$ are some coefficients that depend on the training data. This is called the **representer
26 theorem**.

27 Now consider the special case where the loss function is squared loss, and $\lambda = \sigma_y^2$. We want to
28 minimize

$$\frac{30}{31} \mathcal{L}(f) = \frac{1}{2\sigma_y^2} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2 + \frac{1}{2} \|f\|_{\mathcal{H}}^2 \quad (18.93)$$

33 Substituting in Equation (18.92), and using the fact that $\langle \mathcal{K}(\cdot, \mathbf{x}_i), \mathcal{K}(\cdot, \mathbf{x}_j) \rangle = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$, we obtain
34

$$\frac{35}{36} \mathcal{L}(f) = \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} + \frac{1}{2\sigma_y^2} \|\mathbf{y} - \mathbf{K} \boldsymbol{\alpha}\|^2 \quad (18.94)$$

$$\frac{37}{38} = \frac{1}{2} \boldsymbol{\alpha}^T (\mathbf{K} + \frac{1}{\sigma_y^2} \mathbf{K}^2) \boldsymbol{\alpha} - \frac{1}{\sigma_y^2} \mathbf{y}^T \mathbf{K} \boldsymbol{\alpha} + \frac{1}{2\sigma_y^2} \mathbf{y}^T \mathbf{y} \quad (18.95)$$

40 Minimizing this wrt $\boldsymbol{\alpha}$ gives $\hat{\boldsymbol{\alpha}} = (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y}$, which is the same as Equation (18.55). Furthermore,
41 the prediction for a test point is

$$\frac{43}{44} \hat{f}(\mathbf{x}_*) = \mathbf{k}_*^T \boldsymbol{\alpha} = \mathbf{k}_*^T (\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y} \quad (18.96)$$

45 This is known as **kernel ridge regression** [Vov13]. We see that the result matches the posterior
46 predictive mean of a GP in Equation (18.53).

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

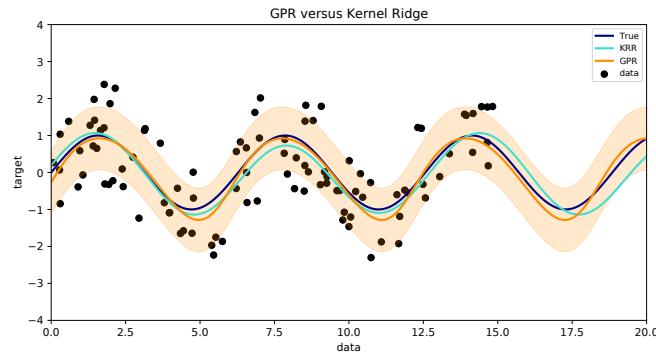


Figure 18.8: Kernel ridge regression (KRR) compared to Gaussian process regression (GPR) using the same kernel. Generated by `krr_vs_gpr.py`.

18.3.7.4 Example of KRR vs GPR

In this section, we compare KRR with GP regression on a simple 1d problem. Since the underlying function is believed to be periodic, we use the periodic kernel from Equation (18.17). To capture the fact that the observations are noisy, we add to this a **white noise kernel**

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sigma_y^2 \delta(\mathbf{x} - \mathbf{x}') \quad (18.97)$$

as in Equation (18.46). Thus there are 3 GP hyper-parameters: the kernel length scale ℓ , the kernel periodicity p , and the noise level σ_y^2 . We can optimize these by maximizing the marginal likelihood using gradient descent (see Section 18.6.1). For KRR, we also have 3 hyper-parameters (ℓ , p and $\lambda = \sigma_y^2$); we optimize these using grid search combined with cross validation (which in general is slower than gradient based optimization). The resulting model fits are shown in Figure 18.8, and are very similar, as is to be expected.

18.4 GPs with non-Gaussian likelihoods

So far, we have focused on GPs for regression using Gaussian likelihoods. In this case, the posterior is also a GP, and all computation can be performed analytically. However, if the likelihood is non-Gaussian, we can no longer compute the posterior exactly. In the sections below, we briefly discuss some approximate inference methods.

18.4.1 Binary classification

In this section, we consider binary classification using GPs. If we use the sigmoid link function, we have $p(y_n = 1|\mathbf{x}_n) = \sigma(y_n f(\mathbf{x}_n))$. If we assume $y_n \in \{-1, +1\}$, then we have $p(y_n|\mathbf{x}_n) = \sigma(y_n f_n)$, since $\sigma(-z) = 1 - \sigma(z)$. If we use the probit link, we have $p(y_n = 1|\mathbf{x}_n) = \Phi(y_n f(\mathbf{x}_n))$, where $\Phi(z)$ is the cdf of the standard normal. More generally, let $p(y_n|\mathbf{x}_n) = \text{Ber}(y_n|\varphi(f_n))$. The overall log

$\frac{1}{1}$	$\log p(y_i f_i)$	$\frac{\partial}{\partial f_i} \log p(y_i f_i)$	$\frac{\partial^2}{\partial f_i^2} \log p(y_i f_i)$
$\frac{2}{2}$	$\log \sigma(y_i f_i)$	$t_i - \pi_i$	$-\pi_i(1 - \pi_i)$
$\frac{3}{3}$	$\log \Phi(y_i f_i)$	$\frac{y_i \phi(f_i)}{\Phi(y_i f_i)}$	$-\frac{\phi_i^2}{\Phi(y_i f_i)^2} - \frac{y_i f_i \phi(f_i)}{\Phi(y_i f_i)}$

$\frac{6}{6}$ Table 18.1: Likelihood, gradient and Hessian for binary logistic/ probit GP regression. We assume $y_i \in$
 $\frac{7}{7}$ $\{-1, +1\}$ and define $t_i = (y_i + 1)/2 \in \{0, 1\}$ and $\pi_i = \sigma(f_i)$ for logistic regression, and $\pi_i = \Phi(f_i)$ for probit
 $\frac{8}{8}$ regression. Also, ϕ and Φ are the pdf and cdf of $\mathcal{N}(0, 1)$. From [RW06, p43].

$\frac{10}{10}$ joint has the form

$$\frac{13}{13} \mathcal{L}(\mathbf{f}_X) = \log p(\mathbf{y}|\mathbf{f}_X) + \log p(\mathbf{f}_X|\mathbf{X}) \quad (18.98)$$

$$\frac{15}{15} = \log p(\mathbf{y}|\mathbf{f}_X) - \frac{1}{2} \mathbf{f}_X^\top \mathbf{K}_{X,X}^{-1} \mathbf{f}_X - \frac{1}{2} \log |\mathbf{K}_{X,X}| - \frac{N}{2} \log 2\pi \quad (18.99)$$

$\frac{17}{17}$ The simplest approach to approximate inference is to use a Laplace approximation (Section 7.4.3).
 $\frac{18}{18}$ The gradient and Hessian of the log joint are given by

$$\frac{20}{20} \nabla \mathcal{L} = \nabla \log p(\mathbf{y}|\mathbf{f}_X) - \mathbf{K}_{X,X}^{-1} \mathbf{f}_X \quad (18.100)$$

$$\frac{22}{22} \nabla^2 \mathcal{L} = \nabla^2 \log p(\mathbf{y}|\mathbf{f}_X) - \mathbf{K}_{X,X}^{-1} = -\boldsymbol{\Lambda} - \mathbf{K}_{X,X}^{-1} \quad (18.101)$$

$\frac{24}{24}$ where $\boldsymbol{\Lambda} \triangleq -\nabla^2 \log p(\mathbf{y}|\mathbf{f}_X)$ is a diagonal matrix, since the likelihood factorizes across examples.
 $\frac{25}{25}$ Expressions for the gradient and Hessian of the log likelihood for the logit and probit case are shown
 $\frac{26}{26}$ in Table 18.1. At convergence, the Laplace approximation of the posterior takes the following form:

$$\frac{28}{28} p(\mathbf{f}_X|\mathcal{D}) \approx q(\mathbf{f}_X) = \mathcal{N}(\hat{\mathbf{f}}, (\mathbf{K}_{X,X}^{-1} + \boldsymbol{\Lambda})^{-1}) \quad (18.102)$$

$\frac{30}{30}$ where $\hat{\mathbf{f}}$ is the MAP estimate. See [RW06, Sec 3.4] for further details.

$\frac{31}{31}$ For improved accuracy, we can use variational inference, in which we assume $q(\mathbf{f}_X) = \mathcal{N}(\mathbf{f}_X|\mathbf{m}, \mathbf{S})$;
 $\frac{32}{32}$ we then optimize \mathbf{m} and \mathbf{S} using (stochastic) gradient descent, rather than assuming \mathbf{S} is the Hessian
 $\frac{33}{33}$ at the mode. See Section 18.5.4 for the details.

$\frac{34}{34}$ Once we have a Gaussian posterior $q(\mathbf{f}_X|\mathcal{D})$, we can then use standard GP prediction to compute
 $\frac{35}{35}$ $q(f_*|\mathbf{x}_*, \mathcal{D})$. Finally, we can approximate the posterior predictive distribution over binary labels
 $\frac{36}{36}$ using

$$\frac{38}{38} \pi_* = p(y_* = 1|\mathbf{x}_*, \mathcal{D}) = \int p(y_* = 1|f_*) q(f_*|\mathbf{x}_*, \mathcal{D}) df_* \quad (18.103)$$

$\frac{41}{41}$ This 1d integral can be computed using the probit approximation from Section 15.3.5. In this case
 $\frac{42}{42}$ we have $\pi_* \approx \sigma(\kappa(v)\mathbb{E}[f_*])$, where $v = \mathbb{V}[f_*]$ and $\kappa^2(v) = (1 + \pi v/8)^{-1}$.

$\frac{43}{43}$ In Figure 18.9, we show a synthetic binary classification problem in 2d. We use an SE kernel.
 $\frac{44}{44}$ On the left, we show predictions using hyper-parameters set by hand; we use a short length scale,
 $\frac{45}{45}$ hence the very sharp turns in the decision boundary. On the right, we show the predictions using the
 $\frac{46}{46}$ learned hyper-parameters; the model favors more parsimonious explanation of the data.

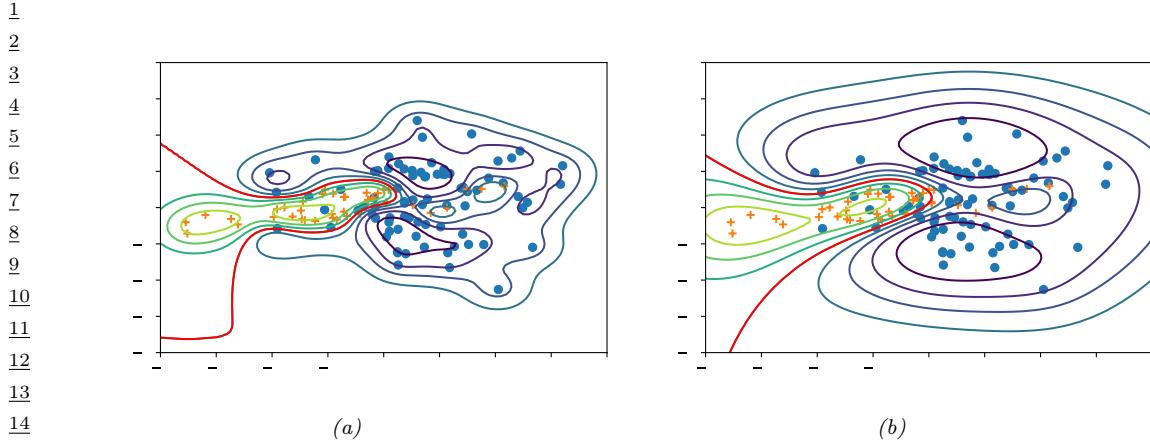


Figure 18.9: Contours of the posterior predictive probability for a binary classifier generated by a GP with an SE kernel. (a) Manual kernel parameters: short length scale, $\ell = 0.5$, variance $3.16^2 \approx 9.98$. (b) Learned kernel parameters: long length scale, $\ell = 1.19$, variance $4.79^2 \approx 22.9$. Generated by [gpc_demo_2d_sklearn.py](#).

19

20

21 18.4.2 Multi-class classification

22

23 The multi-class case is somewhat harder, since the function now needs to return a vector of C
24 logits to get $p(y_n|\mathbf{x}_n) = \text{Cat}(y_n|\mathcal{S}(\mathbf{f}_n))$, where $\mathbf{f}_n = (f_n^1, \dots, f_n^C)$. It is standard to assume that
25 $f^c \sim \text{GP}(0, \mathcal{K}_c)$. Thus we have one latent function per class, which are a priori independent, and
26 which may use different kernels.

27 We can derive a Laplace approximation for this model as discussed in [RW06, Sec 3.5]. Alternatively,
28 we can use a variational approach, using the local variational bound to the multinomial softmax in
29 [Cha12]. An alternative variational method, based on data augmentation with auxiliary variables, is
30 described in [Wen+19b; Liu+19a; GFWO20].

31

32 18.4.3 GPs for Poisson regression (Cox process)

33

34 In this section, we illustrate Poisson regression where the underlying log rate function is modeled by
35 a GP. This is known as a **Cox process**. We can perform approximate posterior inference in this
36 model using Laplace, MCMC or SVI (stochastic variational inference). In Figure 18.10 we give a 1d
37 example, where we use a Matern $\frac{5}{2}$ kernel. We apply MCMC and SVI. In the VI case, we additionally
38 have to specify the form of the posterior; we use a Gaussian approximation for the variational GP
39 posterior $p(\mathbf{f}|\mathbf{X}, \mathbf{y})$, and a point estimate for the kernel parameters.

40 An interesting application of this is to spatial **disease mapping**. For example, [VPV10] discuss
41 the problem of modeling the relative risk of heart attack in different regions in Finland. The data
42 consists of the heart attacks in Finland from 1996-2000 aggregated into 20km x 20km lattice cells.
43 The likelihood has the following form: $y_n \sim \text{Poi}(e_n r_n)$, where e_n is the known expected number of
44 deaths (related to the population of cell n and the overall death rate), and r_n is the **relative risk** of
45 cell n which we want to infer. Since the data counts are small, we regularize the problem by sharing
46 information with spatial neighbors. Hence we assume $f \triangleq \log(r) \sim \text{GP}(0, \mathcal{K})$. We use a Matern
47

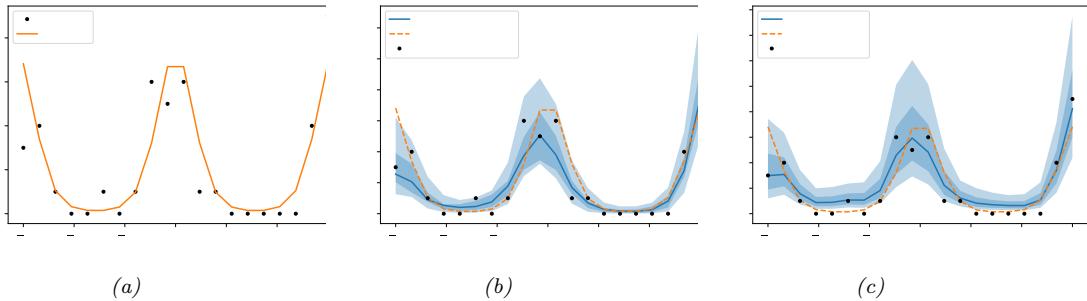


Figure 18.10: Poisson regression with a GP. (a) Observed data (black dots) and true log rate function (yellow line). (b) Posterior predictive distribution (shading shows 1 and 2 σ bands) from MCMC. (c) Posterior predictive distribution from SVI. Generated by [gp_poisson_1d.ipynb](#).

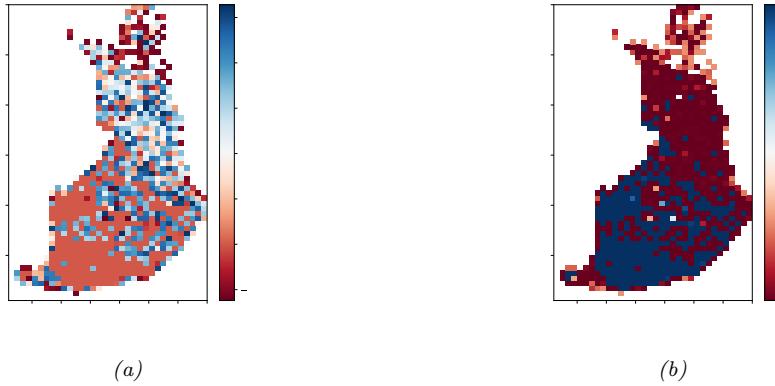


Figure 18.11: We show the relative risk of heart disease in Finland using a Poisson GP fit to 911 data points. Left: posterior mean. Right: posterior variance. Generated by [gp_spatial_demo.py](#).

kernel (Section 18.2.1.1) with $\nu = 3/2$, and a length scale and magnitude that are estimated from data.

Figure 18.11 gives an example of this method in action (using Laplace approximation). On the left we plot the posterior mean relative risk (RR), and on the right, the posterior variance. We see that the RR is higher in Eastern Finland, which is consistent with other studies. We also see that the variance in the North is higher, since there are fewer people living there.

18.5 Scaling GP inference to large datasets

In Section 18.3.6, we saw that the best way to perform GP inference and training is to compute a Cholesky decomposition of the $N \times N$ Gram matrix. Unfortunately, this takes $O(N^3)$ time. In this

Method	Cost	Section
Cholesky	$O(N^3)$	Section 18.3.6
Conj. Grad.	$O(CN^2)$	Section 18.5.5
Inducing	$O(NM^2 + M^3 + DNM)$	Section 18.5.3
Variational	$O(NM^2 + M^3 + DNM)$	Section 18.5.4
SVGP	$O(BM^2 + M^3 + DNM)$	Section 18.5.4.3
KISS-GP	$O(CN + CDM^D \log M)$	Section 18.5.5.4
SKIP	$O(DLN + DLM \log M + L^3 N \log D + CL^2 N)$	Section 18.5.5.4

Table 18.2: Summary of time to compute the log marginal likelihood of a GP regression model. Notation: N is number of training examples, M is number of inducing points, B is size of minibatch, D is dimensionality of input vectors (assuming $\mathcal{X} = \mathbb{R}^D$), C is number of conjugate gradient iterations. L is number of Lanczos iterations. Based on Table 2 of [Gar+18a].

section, we discuss methods to scale up GPs to handle large N . See Table 18.2 for a summary, and [Liu+20c] for more details.¹

18.5.1 Subset of data

The simplest approach to speeding up GP inference is to throw away some of the data. Suppose we keep a subset of M examples. In this case, exact inference will take $O(M^3)$ time. This is called the **subset-of-data** approach.

The key question is: how should we choose the subset? The simplest approach is to pick random examples (this method was recently analysed in [HIY19]). However, intuitively it makes more sense to try to pick a subset that in some sense “covers” the original data, so it contains approximately the same information (up to some tolerance) without the redundancy. Clustering algorithms are one heuristic approach, but we can also use coresets methods, which can provably find such an information-preserving subset (see e.g., [Hug+19] for an application of this idea to GPs).

18.5.1.1 Informative vector machine

Clustering and coresets methods are unsupervised, in that they only look at the features \mathbf{x}_i and not the labels y_i , which can be suboptimal. The **informative vector machine** [HLS03] uses a greedy strategy to iteratively add the labeled example (\mathbf{x}_j, y_j) that maximally reduces the entropy of the function’s posterior, $\Delta_j = \mathbb{H}(p(f_j)) - \mathbb{H}(p^{\text{new}}(f_j))$, where $p^{\text{new}}(f_j)$ is the posterior of f at \mathbf{x}_j after conditioning on y_j . (This is very similar to active learning.) To compute Δ_j , let $p(f_j) = \mathcal{N}(\mu_j, v_j)$, and $p(f_j|y_j) \propto p(f_j)\mathcal{N}(y_j|f_j, \sigma^2) = \mathcal{N}(f_j|\mu_j^{\text{new}}, v_j^{\text{new}})$, where $(v_j^{\text{new}})^{-1} = v_j^{-1} + \sigma^{-2}$. Since $\mathbb{H}(\mathcal{N}(\mu, v)) = \log(2\pi ev)/2$, we have $\Delta_j = 0.5 \log(1 + v_j/\sigma^2)$. Since this is a monotonic function of v_j , we can maximize it by choosing the site with the largest variance. (In fact, entropy is a submodular function, so we can use submodular optimization algorithms to improve on the IVM, as shown in [Kra+08].)

¹ We focus on efficient methods for evaluating the marginal likelihood and the posterior predictive distribution. For an efficient method for sampling a function from the posterior, see [Wil+20a].

1
2 **18.5.1.2 Discussion**

3 The main problem with the subset of data approach is that it ignores some of the data, which can
4 reduce predictive accuracy and increase uncertainty about the true function. Fortunately there
5 are other scalable methods that avoid this problem, essentially by approximately representing (or
6 compressing) the training data, as we discuss below.
7

8
9 **18.5.2 Nyström approximation**

10 Suppose we had a rank M approximation to the $N \times N$ matrix gram matrix of the following form:

11 $\mathbf{K}_{X,X} \approx \mathbf{U}\Lambda\mathbf{U}^\top$ (18.104)

13 where Λ is a diagonal matrix of the M leading eigenvalues, and \mathbf{U} is the matrix of the corresponding
14 M eigenvectors, each of size N . In this case, we can use the matrix inversion lemma to write
15

16 $\mathbf{K}_\sigma^{-1} = (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \approx \sigma^{-2} \mathbf{I}_N + \sigma^{-2} \mathbf{U}(\sigma^2 \Lambda^{-1} + \mathbf{U}^\top \mathbf{U})^{-1} \mathbf{U}^\top$ (18.105)

17 which takes $O(NM^2)$ time. Similarly, one can show (using the Sylvester determinant lemma) that
18

19 $|\mathbf{K}_\sigma| \approx |\Lambda| |\sigma^2 \Lambda^{-1} + \mathbf{U}^\top \mathbf{U}|$ (18.106)

20 which also takes $O(NM^2)$ time.

22 Unfortunately, directly computing such an eigendecomposition takes $O(N^3)$ time, which does not
23 help. However, suppose we pick a subset Z of $M < N$ points. We can partition the Gram matrix as
24 follows (where we assume the chosen points come first, and then the remaining points):

25 $\mathbf{K}_{X,X} = \begin{pmatrix} \mathbf{K}_{Z,Z} & \mathbf{K}_{Z,X-Z} \\ \mathbf{K}_{X-Z,Z} & \mathbf{K}_{X-Z,X-Z} \end{pmatrix}$ (18.107)

28 Let $\mathbf{K}_{Z,X}$ denote the top $M \times N$ block, and $\mathbf{K}_{X,Z}$ denote its transpose. We now compute an
29 eigendecomposition of $\mathbf{K}_{Z,Z}$ to get the eigenvalues $\{\lambda_i\}_{i=1}^M$ and eigenvectors $\{\mathbf{u}_i\}_{i=1}^M$. We now use
30 these to approximate the full matrix as shown below, where the scaling constants are chosen so that
31 $\|\tilde{\mathbf{u}}_i\| \approx 1$:

32 $\tilde{\lambda}_i \triangleq \frac{N}{M} \lambda_i$ (18.108)

35 $\tilde{\mathbf{u}} \triangleq \sqrt{\frac{M}{N}} \frac{1}{\lambda_i} \mathbf{K}_{X,Z} \mathbf{u}_i$ (18.109)

37 $\mathbf{K}_{X,X} \approx \sum_{i=1}^M \tilde{\lambda}_i \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top$ (18.110)

40 $= \sum_{i=1}^M \frac{N}{M} \lambda_i \sqrt{\frac{M}{N}} \frac{1}{\lambda_i} \mathbf{K}_{X,Z} \mathbf{u}_i \sqrt{\frac{M}{N}} \frac{1}{\lambda_i} \mathbf{u}_i^\top \mathbf{K}_{X,Z}^\top$ (18.111)

43 $= \mathbf{K}_{X,Z} \left(\sum_{i=1}^M \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^\top \right) \mathbf{K}_{Z,X}$ (18.112)

46 $= \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X}$ (18.113)

¹ This is known as the **Nyström approximation** [WS01]. If we define
²

$$\mathbf{Q}_{A,B} \triangleq \mathbf{K}_{A,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,B} \quad (18.114)$$

³ then we can write the approximate Gram matrix as $\mathbf{Q}_{X,X}$. We can then replace \mathbf{K}_σ with $\hat{\mathbf{Q}}_{X,X} =$
⁴ $\mathbf{Q}_{X,X} + \sigma^2 \mathbf{I}_N$. Computing the eigendecomposition takes $O(M^3)$ time, and computing $\hat{\mathbf{Q}}_{X,X}^{-1}$ takes
⁵ $O(NM^2)$ time. Thus complexity is now linear in N instead of cubic.
⁶

⁷ If we are approximating *only* $\hat{\mathbf{K}}_{X,X}$ in $\mu_{*|X}$ in Equation (18.50) and $\Sigma_{*|X}$ in Equation (18.51)
⁸ $\hat{\mathbf{Q}}_{X,X}$, then this is inconsistent with the other un-approximated kernel function evaluations in these
⁹ formulae, and can result in the predictive variance being negative. One solution to this is to use the
¹⁰ same \mathbf{Q} approximation for all terms.
¹¹

¹²

¹³ 18.5.3 Inducing point methods

¹⁴ In this section, we discuss an approximation method based on **inducing points**, also called **pseudo**
¹⁵ **inputs**, which are like a learned summary of the training data that we can condition on, rather than
¹⁶ conditioning on all of it.
¹⁷

¹⁸ Let \mathbf{X} be the observed inputs, and $\mathbf{f}_X = f(\mathbf{X})$ be the unknown vector of function values (for which
¹⁹ we have noisy observations \mathbf{y}). Let \mathbf{f}_* be the unknown function values at one or more test points
²⁰ \mathbf{X}_* . Finally, let us assume we have M additional inputs, \mathbf{Z} , with unknown function values \mathbf{f}_Z (often
²¹ denoted by \mathbf{u}). The exact joint prior has the form
²²

$$\begin{aligned} p(\mathbf{f}_X, \mathbf{f}_*) &= \int p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) d\mathbf{f}_Z = \int p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) d\mathbf{f}_Z = \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix}\right) \end{aligned} \quad (18.115)$$

²³ (We write $p(\mathbf{f}_X, \mathbf{f}_*)$ instead of $p(\mathbf{f}_X, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$, since the inputs can be thought of as just indices
²⁴ into the random function f .)

²⁵ We will choose \mathbf{f}_Z in such a way that it acts as a sufficient statistic for the data, so that we can
²⁶ predict \mathbf{f}_* just using \mathbf{f}_Z instead of \mathbf{f}_X , i.e., we assume $\mathbf{f}_* \perp \mathbf{f}_X | \mathbf{f}_Z$. Thus we approximate the prior
²⁷ as follows:

$$p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) = p(\mathbf{f}_* | \mathbf{f}_X, \mathbf{f}_Z) p(\mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) \approx p(\mathbf{f}_* | \mathbf{f}_Z) p(\mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) \quad (18.116)$$

²⁸ See Figure 18.12 for an illustration of this assumption, and Section 18.5.3.4 for details on how to
²⁹ choose the inducing set \mathbf{Z} . (Note that this method is often called a “**sparse GP**”, because it makes
³⁰ predictions for \mathbf{f}_* using a subset of the training data, namely \mathbf{f}_Z , instead of all of it, \mathbf{f}_X .)

³¹ From this, we can derive the following train and test conditionals

$$p(\mathbf{f}_X | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_X | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) \quad (18.117)$$

$$p(\mathbf{f}_* | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_* | \mathbf{K}_{*,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}) \quad (18.118)$$

³² The above equations can be seen as exact inference on noise-free observations \mathbf{f}_Z . To gain
³³ computational speedups, we will make further approximations of the form $\tilde{\mathbf{Q}}_{X,X} \approx \mathbf{K}_{X,X} - \mathbf{Q}_{X,X}$
³⁴ and $\tilde{\mathbf{Q}}_{*,*} \approx \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$. We consider various choices for these values below. All of these choices
³⁵ result in an initial training cost of $O(M^3 + NM^2)$, and then take $O(M)$ time for the predictive mean
³⁶ for each test case, and $O(M^2)$ time for the predictive variance. (Compare this to $O(N^3)$ training
³⁷ time and $O(N)$ and $O(N^2)$ testing time for exact inference.)
³⁸

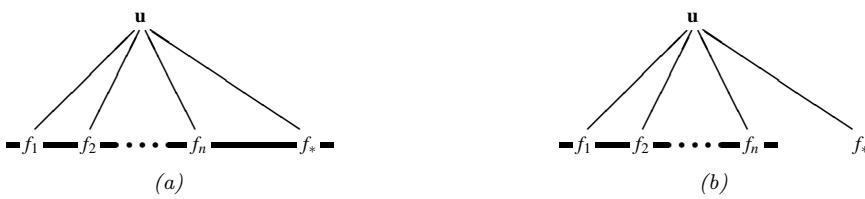


Figure 18.12: Illustration of the graphical model for a GP on n observations, $\mathbf{f}_{1:n}$, and one test case, f_* , with inducing variables \mathbf{u} . The thick lines indicate that all variables are fully interconnected. The observations y_i (not shown) are locally connected to each f_i . (a) no approximations are made. (b) we assume f_* is conditionally independent of \mathbf{f}_X given \mathbf{u} . From Figure 1 of [QCR05]. Used with kind permission of Joaquin Quinonero-Candela.

18.5.3.1 SOR/ DIC

Suppose we assume $\tilde{\mathbf{Q}}_{X,X} = \mathbf{0}$ and $\tilde{\mathbf{Q}}_{*,*} = \mathbf{0}$, so the conditionals are deterministic. This is called the **deterministic inducing conditional (DIC)** approximation [QCR05], or the **subset of regressors (SOR)** approximation [Sil85; SB01]. The corresponding joint prior has the form

$$q_{\text{SOR}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N}(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{Q}_{*,*} \end{pmatrix}) \quad (18.119)$$

Consequently the predictive distribution is

$$q_{\text{SOR}}(\mathbf{f}_* | \mathbf{y}) = \mathcal{N}(\mathbf{f}_* | \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{y}, \mathbf{Q}_{*,*} - \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{Q}_{X,*}) \quad (18.120)$$

$$= \mathcal{N}(\mathbf{f}_* | \sigma^{-2} \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,X} \mathbf{y}, \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,*}) \quad (18.121)$$

where we have defined $\hat{\mathbf{Q}}_{X,X} = \mathbf{Q}_{X,X} + \sigma^2 \mathbf{I}_N$, and $\Sigma = (\sigma^{-2} \mathbf{K}_{Z,X} \mathbf{K}_{X,Z} + \mathbf{K}_{Z,Z})^{-1}$.

This predictive distribution is equivalent to the usual one for GPs except we have replaced $\mathbf{K}_{X,X}$ by $\mathbf{Q}_{X,X}$. This is equivalent to performing GP inference with the following kernel function

$$\mathcal{K}_{\text{SOR}}(\mathbf{x}_i, \mathbf{x}_j) = \mathcal{K}(\mathbf{x}_i, \mathbf{Z}) \mathbf{K}_{Z,Z}^{-1} \mathcal{K}(\mathbf{Z}, \mathbf{x}_j) \quad (18.122)$$

The kernel matrix has rank M , so the GP is degenerate. Furthermore, the kernel will be near 0 when \mathbf{x}_i or \mathbf{x}_j is far from one of the chosen points \mathbf{Z} , which can result in an underestimate of the predictive variance.

18.5.3.2 DTC

One way to overcome the overconfidence of DIC is to only assume $\tilde{\mathbf{Q}}_{X,X} = \mathbf{0}$, but let $\tilde{\mathbf{Q}}_{*,*} = \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$ be exact. This is called the **deterministic training conditional** or **DTC** method [SWL03].

The corresponding joint prior has the form

$$q_{\text{dtc}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N}(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix}) \quad (18.123)$$

¹ Hence the predictive distribution becomes
²

$$q_{\text{dtc}}(\mathbf{f}_* | \mathbf{y}) = \mathcal{N}(\mathbf{f}_* | \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{y}, \mathbf{K}_{*,*} - \mathbf{Q}_{*,X} \hat{\mathbf{Q}}_{X,X}^{-1} \mathbf{Q}_{X,*}) \quad (18.124)$$

$$= \mathcal{N}(\mathbf{f}_* | \sigma^{-2} \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,X} \mathbf{y}, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*} + \mathbf{K}_{*,Z} \Sigma \mathbf{K}_{Z,*}) \quad (18.125)$$

⁶ The predictive mean is the same as in SOR, but the variance is larger (since $\mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$ is positive
⁷ definite) due to the uncertainty of \mathbf{f}_* given \mathbf{f}_Z .
⁸

⁹ 18.5.3.3 FITC

¹⁰ A widely used approximation assumes $q(\mathbf{f}_X | \mathbf{f}_Z)$ is fully factorized, i.e,

$$q(\mathbf{f}_X | \mathbf{f}_Z) = \prod_{n=1}^N p(f_n | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_X | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \text{diag}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X})) \quad (18.126)$$

¹⁵ This is called the **fully independent training conditional** or **FITC** assumption, and was first
¹⁶ proposed in [SG06a]. This throws away less uncertainty than the SOR and DTC methods, since it
¹⁷ does not make any deterministic assumptions about the relationship between \mathbf{f}_X and \mathbf{f}_Z .
¹⁸

¹⁹ The joint prior has the form

$$q_{\text{fitc}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N}(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} - \text{diag}(\mathbf{Q}_{X,X} - \mathbf{K}_{X,X}) & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix}) \quad (18.127)$$

²² The predictive distribution for a single test case is given by

$$q_{\text{fitc}}(f_* | \mathbf{y}) = \mathcal{N}(f_* | \mathbf{k}_{*,Z} \Sigma \mathbf{K}_{Z,X} \Lambda^{-1} \mathbf{y}, k_{**} - q_{**} + \mathbf{k}_{*,Z} \Sigma \mathbf{k}_{Z,*}) \quad (18.128)$$

²⁵ where $\Lambda \triangleq \text{diag}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X} + \sigma^2 \mathbf{I}_N)$, and $\Sigma \triangleq (\mathbf{K}_{Z,Z} + \mathbf{K}_{Z,X} \Lambda^{-1} \mathbf{K}_{X,Z})^{-1}$. If we have a batch
²⁶ of test cases, we can assume they are conditionally independent (an approach known as **fully
²⁷ independent conditional** or **FIC**), and multiply the above equation.
²⁸

²⁹ The computational cost is the same as for SOR and DTC, but the approach avoids some of the
³⁰ pathologies due to a non-degenerate kernel. In particular, one can show that the FIC method is
³¹ equivalent to exact GP inference with the following non-degenerate kernel:
³²

$$\mathcal{K}_{\text{fic}}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) & \text{if } i = j \\ \mathcal{K}_{\text{SOR}}(\mathbf{x}_i, \mathbf{x}_j) & \text{if } i \neq j \end{cases} \quad (18.129)$$

³⁵ 18.5.3.4 Learning the inducing points

³⁷ So far, we have not specified how to choose the inducing points or pseudo inputs \mathbf{Z} . We can treat
³⁸ these like kernel hyperparameters, and choose them so as to maximize the log marginal likelihood,
³⁹ given by

$$\log q(\mathbf{y} | \mathbf{X}, \mathbf{Z}) = \log \int \int p(\mathbf{y} | \mathbf{f}_X) q(\mathbf{f}_X | \mathbf{X}, \mathbf{f}_Z) p(\mathbf{f}_Z | \mathbf{Z}) d\mathbf{f}_Z d\mathbf{f} \quad (18.130)$$

$$= \log \int p(\mathbf{y} | \mathbf{f}_X) q(\mathbf{f}_X | \mathbf{X}, \mathbf{Z}) d\mathbf{f}_X \quad (18.131)$$

$$= -\frac{1}{2} \log |\mathbf{Q}_{X,X} + \Lambda| - \frac{1}{2} \mathbf{y}^\top (\mathbf{Q}_{X,X} + \Lambda)^{-1} \mathbf{y} - \frac{n}{2} \log(2\pi) \quad (18.132)$$

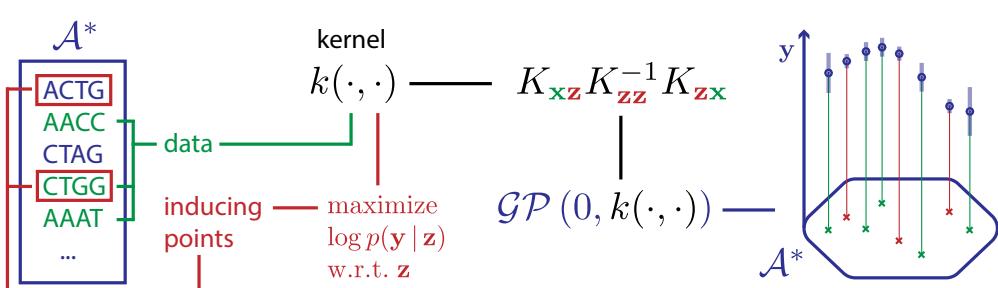


Figure 18.13: Illustration of how to choose inducing points from a discrete input domain (here DNA sequences of length 4) to maximize the log marginal likelihood. From Figure 1 of [For+18a]. Used with kind permission of Vincent Fortuin.

where the definition of Λ depends on the method, namely $\Lambda_{\text{SOR}} = \Lambda_{\text{dtc}} = \sigma^2 \mathbf{I}_N$, and $\Lambda_{\text{fitc}} = \text{diag}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) + \sigma^2 \mathbf{I}_N$.

If the input domain is \mathbb{R}^d , we can optimize $\mathbf{Z} \in \mathbb{R}^{Md}$ using gradient methods. However, one of the appeals of kernel methods is that they can handle structured inputs, such as strings and graphs (see Section 18.2.1.4). In this case, we cannot use gradient methods to select the inducing points. A simple approach is to select the inducing points from the training set, as in the subset of data approach in Section 18.5.1, or using the efficient selection mechanism in [Cao+15]. However, we can also use discrete optimization methods, such as simulated annealing (Section 12.2.5), as discussed in [For+18a]. See Figure 18.13 for an illustration.

18.5.4 Sparse variational methods

In this section, we discuss a variational approach to GP inference that is similar to the inducing point methods in Section 18.5.3, but which generalizes it to also handle non-conjugate likelihoods. This is called the **sparse variational GP** or **SVGP** approximation. For more details, see [Lei+20]. See also [WKS21] for connections between SVGP and the Nyström method.)

To explain the idea behind SVGP, let us assume, for simplicity, that the function f is defined over a finite set \mathcal{X} of possible inputs, which we partition into three subsets: the training set \mathbf{X} , a set of inducing points \mathbf{Z} , and all other points (which we can think of as the test set), \mathbf{X}_* . (We assume these sets are disjoint.) Let \mathbf{f}_X , \mathbf{f}_Z and \mathbf{f}_* represent the corresponding unknown function values on these points, and let $\mathbf{f} = [\mathbf{f}_X, \mathbf{f}_Z, \mathbf{f}_*]$ be all the unknowns. (Here we work with a fixed-length vector \mathbf{f} , but the result generalizes to Gaussian processes, as explained in [Mat+16].) We assume the function is sampled from a GP, so $p(\mathbf{f}) = \mathcal{N}(\mathbf{m}(\mathcal{X}), \mathbf{K}(\mathcal{X}, \mathcal{X}))$.

The inducing point methods in Section 18.5.3 approximates the GP prior by assuming $p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) \approx p(\mathbf{f}_* | \mathbf{f}_Z)p(\mathbf{f}_X | \mathbf{f}_Z)p(\mathbf{f}_Z)$. The inducing points \mathbf{f}_Z are chosen to maximize the likelihood of the observed data. We then perform exact inference in this approximate model. By contrast, in this section, we will keep the model unchanged, but we will instead approximate the posterior $p(\mathbf{f} | \mathbf{y})$ using variational inference. This approach is known as the **variational free energy (VFE)** method for GPs [Tit09; Mat+16]; it is also called SVGP.

In the VFE view, the inducing points \mathbf{Z} and inducing variables \mathbf{f}_Z are variational parameters,

¹ rather than model parameters, which avoids the risk of overfitting. Furthermore, one can show that
² as the number of inducing points m increases, the quality of the posterior consistently improves,
³ eventually recovering exact inference. By contrast, in the classical inducing point method, increasing
⁴ m does not always result in better performance [BWR16].

⁵ In more detail, the VFE approach tries to find an approximate posterior $q(\mathbf{f})$ to minimize
⁶ $D_{\text{KL}}(q(\mathbf{f})\|p(\mathbf{f}|\mathbf{y}))$. The key assumption is that $q(\mathbf{f}) = q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) = p(\mathbf{f}_*, \mathbf{f}_X|\mathbf{f}_Z)q(\mathbf{f}_Z)$, where
⁷ $p(\mathbf{f}_*, \mathbf{f}_X|\mathbf{f}_Z)$ is computed exactly using the GP prior, and $q(\mathbf{f}_Z)$ is learned, by minimizing $\mathcal{K}(q) =$
⁸ $D_{\text{KL}}(q(\mathbf{f})\|p(\mathbf{f}|\mathbf{y}))$.² Intuitively, $q(\mathbf{f}_Z)$ acts as a “bottleneck” which “absorbs” all the observations
⁹ from \mathbf{y} ; posterior predictions for elements of \mathbf{f}_X or \mathbf{f}_* are then made via their dependence on \mathbf{f}_Z ,
¹⁰ rather than their dependence on each other.

¹¹ We can derive the form of the loss, which is used to compute the posterior $q(\mathbf{f}_Z)$, as follows:

$$\mathcal{K}(q) = D_{\text{KL}}(q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z)\|p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z|\mathbf{y})) \quad (18.133)$$

$$= \int q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) \log \frac{q(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z)}{p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z|\mathbf{y})} d\mathbf{f}_* d\mathbf{f}_X d\mathbf{f}_Z \quad (18.134)$$

$$= \int p(\mathbf{f}_*, \mathbf{f}_X|\mathbf{f}_Z)q(\mathbf{f}_Z) \log \frac{p(\mathbf{f}_*|\mathbf{f}_X, \mathbf{f}_Z)p(\mathbf{f}_X|\mathbf{f}_Z)q(\mathbf{f}_Z)p(\mathbf{y})}{p(\mathbf{f}_*|\mathbf{f}_X, \mathbf{f}_Z)p(\mathbf{f}_X|\mathbf{f}_Z)p(\mathbf{f}_Z)p(\mathbf{y}|\mathbf{f}_X)} d\mathbf{f}_* d\mathbf{f}_X d\mathbf{f}_Z \quad (18.135)$$

$$= \int p(\mathbf{f}_*, \mathbf{f}_X|\mathbf{f}_Z)q(\mathbf{f}_Z) \log \frac{q(\mathbf{f}_Z)p(\mathbf{y})}{p(\mathbf{f}_Z)p(\mathbf{y}|\mathbf{f}_X)} d\mathbf{f}_* d\mathbf{f}_X d\mathbf{f}_Z \quad (18.136)$$

$$= \int q(\mathbf{f}_Z) \log \frac{q(\mathbf{f}_Z)}{p(\mathbf{f}_Z)} d\mathbf{f}_Z - \int p(\mathbf{f}_X|\mathbf{f}_Z)q(\mathbf{f}_Z) \log p(\mathbf{y}|\mathbf{f}_X) d\mathbf{f}_X d\mathbf{f}_Z + C \quad (18.137)$$

$$= D_{\text{KL}}(q(\mathbf{f}_Z)\|p(\mathbf{f}_Z)) - \mathbb{E}_{q(\mathbf{f}_X)}[\log p(\mathbf{y}|\mathbf{f}_X)] + C \quad (18.138)$$

²⁶ where $C = \log p(\mathbf{y})$ is an irrelevant constant.

²⁷ We can alternatively write the objective as an evidence lower bound that we want to maximize:

$$\log p(\mathbf{y}) = \mathcal{K}(q) + \mathbb{E}_{q(\mathbf{f}_X)}[\log p(\mathbf{y}|\mathbf{f}_X)] - D_{\text{KL}}(q(\mathbf{f}_Z)\|p(\mathbf{f}_Z)) \quad (18.139)$$

$$\geq \mathbb{E}_{q(\mathbf{f}_X)}[\log p(\mathbf{y}|\mathbf{f}_X)] - D_{\text{KL}}(q(\mathbf{f}_Z)\|p(\mathbf{f}_Z)) \triangleq \mathcal{L}(q) \quad (18.140)$$

³² Now suppose we choose a Gaussian posterior approximation, $q(\mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_Z|\mathbf{m}, \mathbf{S})$. Since $p(\mathbf{f}_Z) =$
³³ $\mathcal{N}(\mathbf{f}_Z|\mathbf{0}, \mathcal{K}(\mathbf{Z}, \mathbf{Z}))$, we can compute the KL term in closed form using the formula for KL divergence
³⁴ between Gaussians (Equation (5.66)).

³⁵ As for the expected log-likelihood term, we need to compute $q(\mathbf{f}_X)$. Since $q(\mathbf{f}_Z)$ is Gaussian, this
³⁶ can be done in closed form as follows:

$$q(\mathbf{f}_X|\mathbf{m}, \mathbf{S}) = \int p(\mathbf{f}_X|\mathbf{f}_Z, \mathbf{X}, \mathbf{Z})q(\mathbf{f}_Z|\mathbf{m}, \mathbf{S})d\mathbf{f}_Z = \mathcal{N}(\mathbf{f}_X|\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) \quad (18.141)$$

$$\tilde{\boldsymbol{\mu}}_i = \mathbf{m}(\mathbf{x}_i) + \boldsymbol{\alpha}(\mathbf{x}_i)^T(\mathbf{m} - \mathbf{m}(\mathbf{Z})) \quad (18.142)$$

$$\tilde{\boldsymbol{\Sigma}}_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) - \boldsymbol{\alpha}(\mathbf{x}_i)^T(\mathcal{K}(\mathbf{Z}, \mathbf{Z}) - \mathbf{S})\boldsymbol{\alpha}(\mathbf{x}_j) \quad (18.143)$$

$$\boldsymbol{\alpha}(\mathbf{x}_i) = \mathcal{K}(\mathbf{Z}, \mathbf{Z})^{-1}\mathcal{K}(\mathbf{Z}, \mathbf{x}_i) \quad (18.144)$$

⁴⁵ ⁴⁶ 2. One can show that $D_{\text{KL}}(q(\mathbf{f})\|p(\mathbf{f}|\mathbf{y})) = D_{\text{KL}}(q(\mathbf{f}_X, \mathbf{f}_Z)\|p(\mathbf{f}_X, \mathbf{f}_Z|\mathbf{y}))$, which is the original objective from [Tit09].
⁴⁷

1 Hence $q(f_n) = \mathcal{N}(f_n | \tilde{\mu}_n, \tilde{\Sigma}_{nn})$, which we can use to compute the expected log likelihood:

$$\mathbb{E}_{q(\mathbf{f}_X)} [\log p(\mathbf{y} | \mathbf{f}_X)] = \sum_{n=1}^N \mathbb{E}_{q(f_n)} [\log p(y_n | f_n)] \quad (18.145)$$

7 We discuss how to compute the expected loglikelihood below.

9 18.5.4.1 Gaussian likelihood

10 If we have a Gaussian observation model, we can compute the expected log likelihood in closed form.
11 In particular, if we assume $m(\mathbf{x}) = \mathbf{0}$, we have

$$\mathbb{E}_{q(f_n)} [\log \mathcal{N}(y_n | f_n, \beta^{-1})] = \log \mathcal{N}(y_n | \mathbf{k}_n^\top \mathbf{K}_{Z,Z}^{-1} \mathbf{m}, \beta^{-1}) - \frac{1}{2} \beta \tilde{k}_{nn} - \frac{1}{2} \text{tr}(\mathbf{S} \boldsymbol{\Lambda}_n) \quad (18.146)$$

16 where $\tilde{k}_{nn} = k_{nn} - \mathbf{k}_n^\top \mathbf{K}_{Z,Z}^{-1} \mathbf{k}_n$, \mathbf{k}_n is the n 'th column of $\mathbf{K}_{Z,X}$ and $\boldsymbol{\Lambda}_n = \beta \mathbf{K}_{Z,Z}^{-1} \mathbf{k}_n \mathbf{k}_n^\top \mathbf{K}_{Z,Z}^{-1}$.

17 Hence the overall ELBO has the form

$$\mathcal{L}(q) = \log \mathcal{N}(\mathbf{y} | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{m}, \beta^{-1} \mathbf{I}_N) - \frac{1}{2} \beta \text{tr}(\mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{S} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X}) \quad (18.147)$$

$$- \frac{1}{2} \beta \text{tr}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) - D_{\text{KL}}(q(\mathbf{f}_Z) \| p(\mathbf{f}_Z)) \quad (18.148)$$

23 To compute the gradients of this, we leverage the following result [OA09]:

$$\frac{\partial}{\partial \mu} \mathbb{E}_{\mathcal{N}(x|\mu, \sigma^2)} [h(x)] = \mathbb{E}_{\mathcal{N}(x|\mu, \sigma^2)} \left[\frac{\partial}{\partial x} h(x) \right] \quad (18.149)$$

$$\frac{\partial}{\partial \sigma^2} \mathbb{E}_{\mathcal{N}(x|\mu, \sigma^2)} [h(x)] = \frac{1}{2} \mathbb{E}_{\mathcal{N}(x|\mu, \sigma^2)} \left[\frac{\partial^2}{\partial x^2} h(x) \right] \quad (18.150)$$

30 We then substitute $h(x)$ with $\log p(y_n | f_n)$. Using this, one can show

$$\nabla_{\mathbf{m}} \mathcal{L}(q) = \beta \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \mathbf{y} - \boldsymbol{\Lambda} \mathbf{m} \quad (18.151)$$

$$\nabla_{\mathbf{S}} \mathcal{L}(q) = \frac{1}{2} \mathbf{S}^{-1} - \frac{1}{2} \boldsymbol{\Lambda} \quad (18.152)$$

36 Setting the derivatives to zero gives the optimal solution:

$$\mathbf{S} = \boldsymbol{\Lambda}^{-1} \quad (18.153)$$

$$\boldsymbol{\Lambda} = \beta \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} + \mathbf{K}_{Z,Z}^{-1} \quad (18.154)$$

$$\mathbf{m} = \beta \boldsymbol{\Lambda}^{-1} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} \mathbf{y} \quad (18.155)$$

42 This is called **sparse GP regression** or **SGPR** [Tit09].

43 With these parameters, the lower bound on the log marginal likelihood is given by

$$\log p(\mathbf{y}) \geq \log \mathcal{N}(\mathbf{y} | \mathbf{0}, \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{K}_{Z,X} + \beta^{-1} \mathbf{I}) - \frac{1}{2} \beta \text{tr}(\mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) \quad (18.156)$$

¹ where $\mathbf{Q}_{X,X} = \mathbf{K}_{X,Z}\mathbf{K}_{Z,Z}^{-1}\mathbf{K}_{Z,X}$. (This is called the “collapsed” lower bound, since we have marginalized out \mathbf{f}_Z .) If $Z = X$, then $\mathbf{K}_{Z,Z} = \mathbf{K}_{Z,X} = \mathbf{K}_{X,X}$, so the bound becomes tight, and we have $\log p(\mathbf{y}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_{X,X} + \beta^{-1}\mathbf{I})$.

⁵ Equation (18.156) is almost the same as the log marginal likelihood for the DTC model in
⁶ Equation (18.132), except for the trace term; it is this latter term that prevents overfitting, due to
⁷ the fact that we treat \mathbf{f}_Z as variational parameters of the posterior rather than model parameters of
⁸ the prior.

⁹ 18.5.4.2 Non-Gaussian likelihood

¹¹ In this section, we briefly consider the case of non-Gaussian likelihoods, which arise when using
¹² GPs for classification or for count data (see Section 18.4). We can compute the gradients of the
¹³ expected log likelihood by defining $h(f_n) = \log p(y_n|f_n)$ and then using a Monte Carlo approximation
¹⁴ to Equation (18.149) and Equation (18.150). In the case of a binary classifier, we can use the results
¹⁵ in Table 18.1 to compute the inner $\frac{\partial}{\partial f_n} h(f_n)$ and $\frac{\partial^2}{\partial f_n^2} h(f_n)$ terms.

¹⁶ 18.5.4.3 Minibatch SVI

¹⁹ Computing the optimal variational solution in Section 18.5.4.1 requires solving a batch optimization
²⁰ problem, which takes $O(M^3 + NM^2)$ time. This may still be too slow if N is large, unless M is
²¹ small, which compromises accuracy.

²³ An alternative approach is to perform stochastic optimization of the VFE objective, instead of
²⁴ batch optimization. This is known as stochastic variational inference (see Section 10.3.2). The
²⁵ key observation is that the log likelihood in Equation (18.145) is a sum of N terms, which we can
²⁶ approximate with minibatch sampling to compute noisy estimates of the gradient, as proposed in
²⁷ [HFL13].

²⁸ In more detail, the objective becomes

$$\mathcal{L}(q) = \left[\frac{N}{B} \sum_{b=1}^B \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \mathbb{E}_{q(f_n)} [\log p(y_n|f_n)] \right] - D_{\text{KL}}(q(\mathbf{f}_Z) \parallel p(\mathbf{f}_Z)) \quad (18.157)$$

³³ where \mathcal{B}_b is the b 'th batch, and B is the number of batches. Since the GP model (with Gaussian
³⁴ likelihoods) is in the exponential family, we can efficiently compute the natural gradient (Section 6.4)
³⁵ of Equation (18.157) wrt the canonical parameters of $q(\mathbf{f}_Z)$; this converges much faster than following
³⁶ the standard gradient. See [HFL13] for details.

³⁷ 18.5.5 Exploiting parallelization and structure via kernel matrix multiplies

³⁹ It takes $O(N^3)$ time to compute the Cholesky decomposition of $\mathbf{K}_{X,X}$, which is needed to solve the
⁴⁰ linear system $\mathbf{K}_\sigma \boldsymbol{\alpha} = \mathbf{y}$ and to compute $|\mathbf{K}_{X,X}|$. An alternative to Cholesky decomposition is to
⁴¹ use linear algebra methods, often called **Krylov subspace methods** based just on **matrix vector**
⁴² **multiplication** or **MVM**. These approaches are often much faster.

⁴⁴ In short, if the kernel matrix $\mathbf{K}_{X,X}$ has special algebraic structure, which is often the case through
⁴⁵ either the choice of kernel or the structure of the inputs, then it is typically easier to exploit this
⁴⁶ structure in performing fast matrix multiplies. Moreover, even if the kernel matrix **does not** have
⁴⁷

1 special structure, matrix multiplies are trivial to parallelize, and can thus be greatly accelerated by
 2 GPUs, unlike Cholesky based methods which are largely sequential. Algorithms based on matrix
 3 multiplies are in harmony with modern hardware advances, which enable significant parallelization.
 4

5 18.5.5.1 Using conjugate gradient and Lanczos methods

6 We can solve the linear system $\mathbf{K}_\sigma \boldsymbol{\alpha} = \mathbf{y}$ using conjugate gradients (CG). The key computational
 7 step in CG is the ability to perform MVMs. Let $\tau(\mathbf{K}_\sigma)$ be the time complexity of a single MVM
 8 with \mathbf{K}_σ . For a dense $n \times n$ matrix, we have $\tau(\mathbf{K}_\sigma) = n^2$; however, we can speed this up if \mathbf{K}_σ is
 9 sparse or structured, as we discuss below.

10 Even if \mathbf{K}_σ is dense, we may still be able to save time by solving the linear system approximately.
 11 In particular, if we perform C iterations, CG will take $O(C\tau(\mathbf{K}_\sigma))$ time. If we run for $C = n$, and
 12 $\tau(\mathbf{K}_\sigma) = n^2$, it gives the exact solution in $O(n^3)$ time. However, often we can use fewer iterations
 13 and still get good accuracy, depending on the condition number of \mathbf{K}_σ .

14 We can compute the log determinant of a matrix using the MVM primitive with a similar iterative
 15 method known as **stochastic Lanczos quadrature** [UCS17; Don+17a]. This takes $O(L\tau(\mathbf{K}_\sigma))$
 16 time for L iterations.

17 These methods have been used in the **blackbox matrix-matrix multiplication (BBMM)**
 18 inference procedure of [Gar+18a], which formulates a batch approach to CG that can be effectively
 19 parallelized on GPUs. Using 8 GPUs, this enabled the authors of [Wan+19a] to perform exact
 20 inference for a GP regression model on $N \sim 10^4$ datapoints in seconds, $N \sim 10^5$ datapoints in
 21 minutes, and $N \sim 10^6$ datapoints in hours.

22 Interestingly, Figure 18.14 shows that exact GP inference on a subset of the data can often
 23 outperform approximate inference on the full data. We also see that performance of exact GPs
 24 continues to significantly improve as we increase the size of the data, suggesting that GPs are not only
 25 useful in the small-sample setting. In particular, the BBMM is an exact method, and so will preserve
 26 the non-parametric representation of a GP with a non-degenerate kernel. By contrast, standard
 27 scalable approximations typically operate by replacing the exact kernel with an approximation that
 28 corresponds to a parametric model. The non-parametric GPs are able to grow their capacity with
 29 more data, benefiting more significantly from the structure present in large datasets.

30 18.5.5.2 Kernels with compact support

31 Suppose we use a kernel with **compact support**, where $\mathcal{K}(\mathbf{x}, \mathbf{x}') = 0$ if $\|\mathbf{x} - \mathbf{x}'\| > \epsilon$ for some
 32 threshold ϵ (see e.g., [MR09]), then \mathbf{K}_σ will be sparse, so $\tau(\mathbf{K}_\sigma)$ will be $O(N)$. We can also induce
 33 sparsity and structure in other ways, as we discuss in Section 18.5.5.4.

34 18.5.5.3 Gaussian state space models

35 Consider a function defined on a 1d scalar input, such as a time index. For many kernels, the
 36 corresponding GP can be modeled using a stochastic differential equation. This induces a block
 37 tri-diagonal precision matrix for the posterior $p(\mathbf{f}_{1:T} | \mathbf{y}_{1:T})$. We can therefore convert this to a
 38 linear-Gaussian state space model (Section 31.1), and perform exact inference in $O(T)$ time using
 39 Kalman smoothing, as explained in [SSH13; Ada+20]. This conversion can be done exactly for
 40 Matern kernels and approximately for Gaussian (RBF) kernels (see [SS19, Ch. 12]). In [SGF21], they
 41

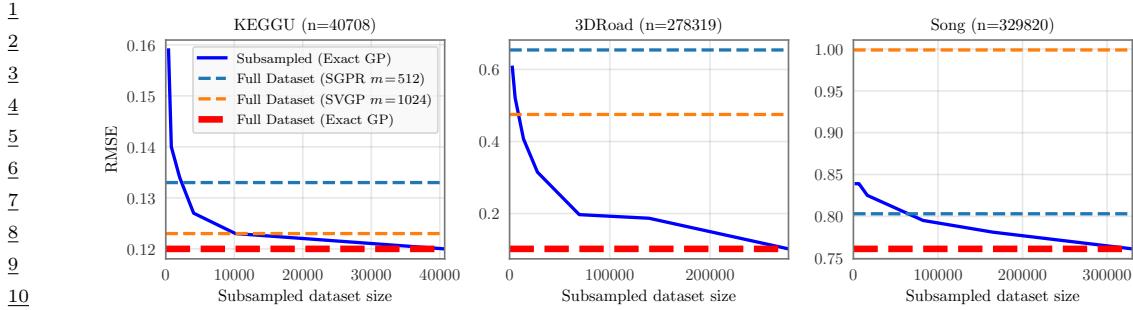


Figure 18.14: RMSE on test set as a function of training set size using a GP with Matern 3/2 kernel with shared lengthscale across all dimensions. Solid lines: exact inference. Dashed blue: SGPR method (closed-form batch solution to the Gaussian variational approximation) of Section 18.5.4.1 with $M = 512$ inducing points. Dashed orange: SVGP method (SGD on Gaussian variational approximation) of Section 18.5.4.3 with $M = 1024$ inducing points. Number of input dimensions: KEGGU $D = 27$, 3DRoad $D = 3$, Song $D = 90$. From Figure 4 of [Wan+19a]. Used with kind permission of Andrew Wilson.

17

18

19 describe how to reduce the linear dependence on T to $\log(T)$ time using a **parallel prefix scan**
20 operator, that can be run efficiently on GPUs.
21

22

18.5.5.4 KISS

24 One way to ensure that MVMs are fast is to force the kernel matrix to have structure. The **structured**
25 **kernel interpolation (SKI)** method of [WN15] does this as follows. First it assumes we have a set
26 of inducing points, with Gram matrix $\mathbf{K}_{Z,Z}$. It then interpolates these values to predict the entries
27 of the full kernel matrix using
28

$$29 \quad \mathbf{K}_{X,X} \approx \mathbf{W}_X \mathbf{K}_{Z,Z} \mathbf{W}_X^T \quad (18.158)$$

30

31 where \mathbf{W}_X is a sparse matrix containing interpolation weights. If we use cubic interpolation, each
32 row only has 4 nonzeros. Thus we can compute $(\mathbf{W}_X \mathbf{K}_{Z,Z} \mathbf{W}_X^T)\mathbf{v}$ for any vector \mathbf{v} in $O(N + M^2)$
33 time.

34 Note that the **SKI** approach generalizes all inducing point methods. For example, we can recover the
35 subset of regressors method (SOR) method by setting the interpolation weights to $\mathbf{W} = \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1}$.
36 We can identify this procedure as performing a global Gaussian process interpolation strategy on the
37 user specified kernel. See [WN15] and [WDN15] for more details.

38 In 1d, we can further reduce the running time by choosing the inducing points to be on a regular
39 grid, so that $\mathbf{K}_{Z,Z}$ is a Toeplitz matrix. In higher dimensions, we need to use a multidimensional grid
40 of points, resulting in $\mathbf{K}_{Z,Z}$ being a Kronecker product of Toeplitz matrices. This enables matrix
41 vector multiplication in $O(N + M \log M)$ time and $O(N + M)$ space. The resulting method is called
42 **KISS-GP** [WN15], which stands for ‘‘kernel interpolation for scalable, structured GPs’’.

43 Unfortunately, the KISS method can take exponential time in the input dimensions D when
44 exploiting Kronecker structure in $\mathbf{K}_{Z,Z}$, due to the need to create a fully connected multidimensional
45 lattice. In [Gar+18b], they propose a method called **SKIP**, which stands for ‘‘SKI for products’’.
46 The idea is to leverage the fact that many kernels (including ARD) can be written as a product of 1d
47

1 kernels: $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^D \mathcal{K}^d(\mathbf{x}, \mathbf{x}')$. This can be combined with the 1d SKI method to enable fast
 2 MVMs. The overall running time to compute the log marginal likelihood (which is the bottleneck
 3 for kernel learning) using C iterations of CG and a Lanczos decomposition of rank L , becomes
 4 $O(DL(N + M \log M) + L^3 N \log D + CL^2 N)$. Typical values are $L \sim 10^1$ and $C \sim 10^2$.

7 18.5.5.5 Tensor train methods

8 Consider the Gaussian VFE approach in Section 18.5.4. We have to estimate the covariance \mathbf{S} and
 9 the mean \mathbf{m} . We can represent \mathbf{S} efficiently using Kronecker structure, as used by KISS. Additionally,
 10 we can represent \mathbf{m} efficiently using the **tensor train decomposition** [Ose11] in combination with
 11 **SKI** [WN15]. The resulting **TT-GP** method can scale efficiently to billions of inducing points, as
 12 explained in [INK18].

14 18.6 Learning the kernel

15 In [Mac98], David MacKay asked: “How can Gaussian processes replace neural networks? Have we
 16 thrown the baby out with the bathwater?” This remark was made in the late 1990s, at the end of
 17 the second wave of neural networks. Researchers and practitioners had grown weary of the design
 18 decisions associated with neural networks — such as activation functions, optimization procedures,
 19 architecture design — and the lack of a principled framework to make these decisions. Gaussian
 20 processes, by contrast, were perceived as flexible and principled probabilistic models, which naturally
 21 followed from Radford Neal’s results on infinite neural networks [Nea96], which we discuss in more
 22 depth in Section 18.7.

23 However, MacKay [Mac98] noted that neural networks could discover rich representations of data
 24 through adaptive hidden basis functions, while Gaussian processes with standard kernel functions,
 25 such as the RBF kernel, are essentially just smoothing devices. Indeed, the generalization properties
 26 of Gaussian processes hinge on the suitability of the kernel function. *Learning* the kernel is how we
 27 do **representation learning** with Gaussian processes, and in many cases will be crucial for good
 28 performance — especially when we wish to perform **extrapolation**, making predictions far away
 29 from the data [WA13; Wil+14].

30 As we will see, learning a kernel is in many ways analogous to training a neural network. Moreover,
 31 neural networks and Gaussian processes can be synergistically combined through approaches such as
 32 deep kernel learning (see Section 18.6.6) and deep GPs (Section 18.7.3).

36 18.6.1 Empirical Bayes for the kernel parameters

37 Suppose, as in Section 18.3.2, we are performing 1d regression using a GP with an RBF kernel. Since
 38 the data has observation noise, the kernel has the following form:

$$\mathcal{K}_y(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(x_p - x_q)^2\right) + \sigma_y^2 \delta_{pq} \quad (18.159)$$

40 Here ℓ is the horizontal scale over which the function changes, σ_f^2 controls the vertical scale of
 41 the function, and σ_y^2 is the noise variance. Figure 18.15 illustrates the effects of changing these
 42 parameters. We sampled 20 noisy data points from the SE kernel using $(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$,
 43 and then made predictions various parameters, conditional on the data. In Figure 18.15(a), we use
 44

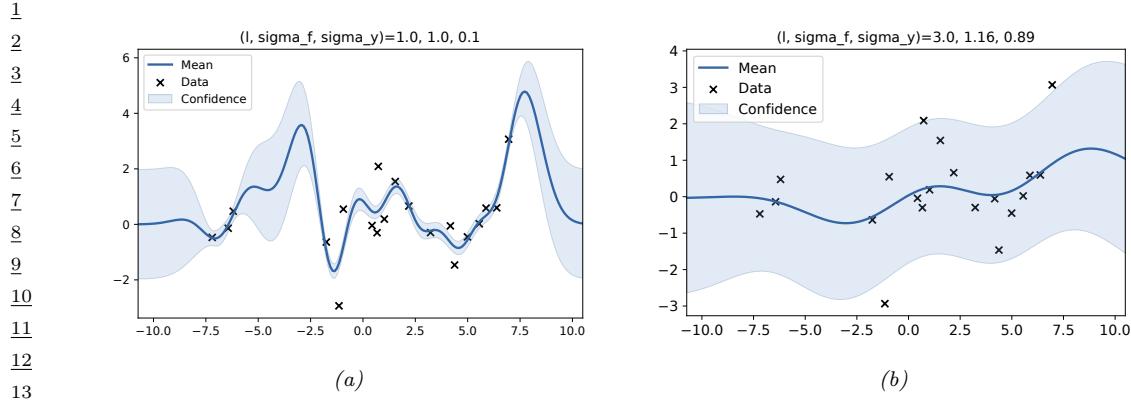


Figure 18.15: Some 1d GPs with RBF kernels but different hyper-parameters fit to 20 noisy observations. The hyper-parameters $(\ell, \sigma_f, \sigma_y)$ are as follows: (a) $(1, 1, 0.1)$ (b) $(3.0, 1.16, 0.89)$. Adapted from Figure 2.5 of [RW06]. Generated by [gprDemoChangeHparams.py](#).

$(\ell, \sigma_f, \sigma_y) = (1, 1, 0.1)$, and the result is a good fit. In Figure 18.15(b), we increase the length scale to $\ell = 3$; now the function looks smoother, but we are arguably underfitting.

To estimate the kernel parameters θ (sometimes called hyperparameters), we could use exhaustive search over a discrete grid of values, with validation loss as an objective, but this can be quite slow. (This is the approach used by nonprobabilistic methods, such as SVMs, to tune kernels.) Here we consider an empirical Bayes approach, which will allow us to use continuous optimization methods, which are much faster. In particular, we will maximize the marginal likelihood

$$p(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X})p(\mathbf{f}|\mathbf{X}, \theta)d\mathbf{f} \quad (18.160)$$

(The reason it is called the marginal likelihood, rather than just likelihood, is because we have marginalized out the latent Gaussian vector \mathbf{f} .) Since $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$, and $p(\mathbf{y}|\mathbf{f}) = \prod_{n=1}^N \mathcal{N}(y_n|f_n, \sigma_y^2)$, the marginal likelihood is given by

$$\log p(\mathbf{y}|\mathbf{X}, \theta) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_\sigma) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}_\sigma^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_\sigma| - \frac{N}{2} \log(2\pi) \quad (18.161)$$

where the dependence of \mathbf{K}_σ on θ is implicit. The first term is a data fit term, the second term is a model complexity term, and the third term is just a constant. To understand the tradeoff between the first two terms, consider a SE kernel in 1D, as we vary the length scale ℓ and hold σ_y^2 fixed. Let $J(\ell) = -\log p(\mathbf{y}|\mathbf{X}, \ell)$. For short length scales, the fit will be good, so $\mathbf{y}^\top \mathbf{K}_\sigma^{-1} \mathbf{y}$ will be small. However, the model complexity will be high: \mathbf{K} will be almost diagonal, since most points will not be considered “near” any others, so the $\log |\mathbf{K}_\sigma|$ will be large. For long length scales, the fit will be poor but the model complexity will be low: \mathbf{K} will be almost all 1’s, so $\log |\mathbf{K}_\sigma|$ will be small.

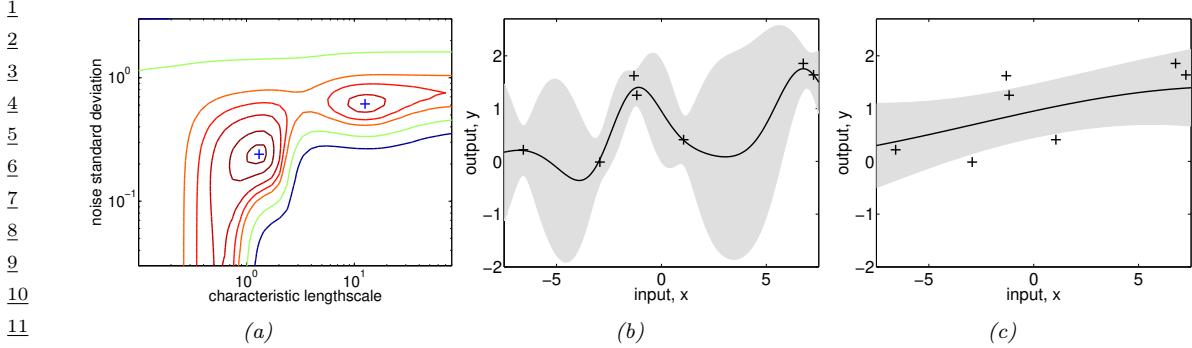


Figure 18.16: Illustration of local minima in the marginal likelihood surface. (a) We plot the log marginal likelihood vs σ_y^2 and ℓ , for fixed $\sigma_f^2 = 1$, using the 7 data points shown in panels b and c. (b) The function corresponding to the lower left local minimum, $(\ell, \sigma_n^2) \approx (1, 0.2)$. This is quite “wiggly” and has low noise. (c) The function corresponding to the top right local minimum, $(\ell, \sigma_n^2) \approx (10, 0.8)$. This is quite smooth and has high noise. The data was generated using $(\ell, \sigma_n^2) = (1, 0.1)$. Adapted from Figure 5.5 of [RW06]. Generated by [gpr_demo_marglik.py](#).

We now discuss how to maximize the marginal likelihood. One can show that

$$\frac{\partial}{\partial \theta_j} \log p(\mathbf{y} | \mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2} \mathbf{y}^\top \mathbf{K}_\sigma^{-1} \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j} \mathbf{K}_\sigma^{-1} \mathbf{y} - \frac{1}{2} \text{tr}(\mathbf{K}_\sigma^{-1} \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j}) \quad (18.162)$$

$$= \frac{1}{2} \text{tr} \left((\boldsymbol{\alpha} \boldsymbol{\alpha}^\top - \mathbf{K}_\sigma^{-1}) \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j} \right) \quad (18.163)$$

where $\boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1} \mathbf{y}$. It takes $O(N^3)$ time to compute \mathbf{K}_σ^{-1} , and then $O(N^2)$ time per hyper-parameter to compute the gradient.

The form of $\frac{\partial \mathbf{K}_\sigma}{\partial \theta_j}$ depends on the form of the kernel, and which parameter we are taking derivatives with respect to. Often we have constraints on the hyper-parameters, such as $\sigma_y^2 \geq 0$. In this case, we can define $\theta = \log(\sigma_y^2)$, and then use the chain rule.

Given an expression for the log marginal likelihood and its derivative, we can estimate the kernel parameters using any standard gradient-based optimizer. However, since the objective is not convex, local minima can be a problem, as we illustrate below, so we may need to use multiple restarts.

18.6.1.1 Example

Consider Figure 18.16. We use the SE kernel in Equation (18.159) with $\sigma_f^2 = 1$, and plot $\log p(\mathbf{y} | \mathbf{X}, \ell, \sigma_y^2)$ (where \mathbf{X} and \mathbf{y} are the 7 data points shown in panels b and c as we vary ℓ and σ_y^2). The two local optima are indicated by '+' in panel (a). The bottom left optimum corresponds to a low-noise, short-length scale solution (shown in panel b). The top right optimum corresponds to a high-noise, long-length scale solution (shown in panel c). With only 7 data points, there is not enough evidence to confidently decide which is more reasonable, although the more complex model (panel b) has a marginal likelihood that is about 60% higher than the simpler model (panel c). With more data, the more complex model would become even more preferred.

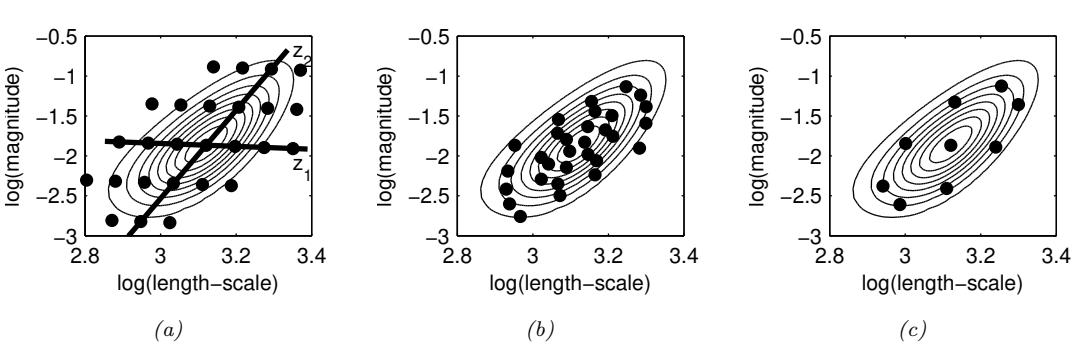


Figure 18.17: Three different approximations to the posterior over hyper-parameters: grid-based, Monte Carlo, and central composite design. From Figure 3.2 of [Van10]. Used with kind permission of Jarno Vanhatalo.

Figure 18.16 illustrates some other interesting (and typical) features. The region where $\sigma_y^2 \approx 1$ (top of panel a) corresponds to the case where the noise is very high; in this regime, the marginal likelihood is insensitive to the length scale (indicated by the horizontal contours), since all the data is explained as noise. The region where $\ell \approx 0.5$ (left hand side of panel a) corresponds to the case where the length scale is very short; in this regime, the marginal likelihood is insensitive to the noise level (indicated by the vertical contours), since the data is perfectly interpolated. Neither of these regions would be chosen by a good optimizer.

18.6.2 Bayesian inference for the kernel parameters

When we have a small number of datapoints (e.g., when using GPs for blackbox optimization, as we discuss in Section 6.9), using a point estimate of the kernel parameters can give poor results [Bul11; WF14]. As a simple example, if the function values that have been observed so far are all very similar, then we may estimate $\hat{\sigma} \approx 0$, which will result in overly confident predictions.³

To overcome such overconfidence, we can compute a posterior over the kernel parameters. If the dimensionality of θ is small, we can compute a discrete grid of possible values, centered on the MAP estimate $\hat{\theta}$ (computed as above). We can then approximate the posterior using

$$p(\mathbf{f}|\mathcal{D}) = \sum_{s=1}^S p(\mathbf{f}|\mathcal{D}, \theta_s) p(\theta_s|\mathcal{D}) w_s \quad (18.164)$$

where w_s denotes the weight for grid point s .

In higher dimensions, a regular grid suffers from the curse of dimensionality. One alternative is to place grid points at the mode, and at a distance $\pm 1\text{sd}$ from the mode along each dimension, for a total of $2|\theta| + 1$ points. This is called a **central composite design** [RMC09]. See Figure 18.17 for an illustration.

In higher dimensions, we can use Monte Carlo inference for the kernel parameters when computing

³ In [WSN00; BBV11b], they show how we can put a conjugate prior on σ^2 and integrate it out, to generate a Student version of the GP, which is more robust.

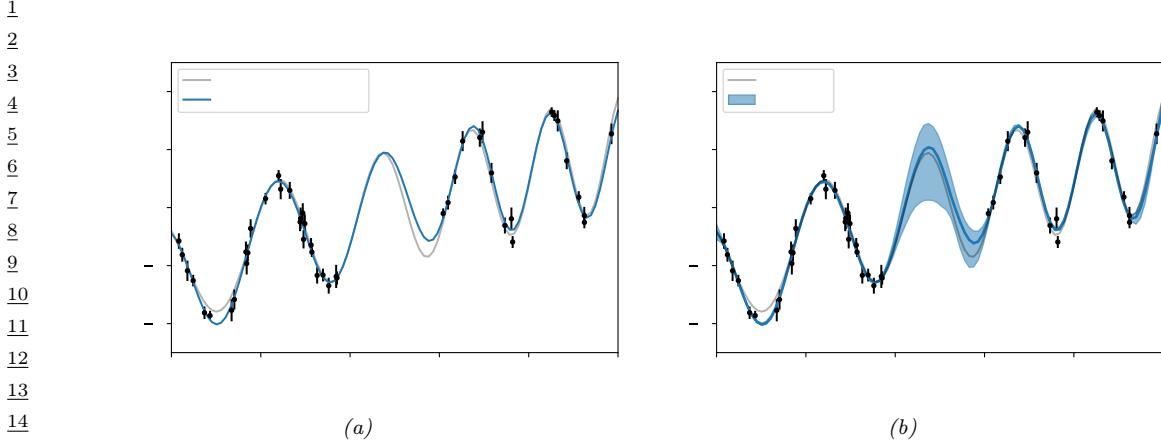


Figure 18.18: Difference between estimation and inference for kernel hyper-parameters. (a) Empirical Bayes approach based on optimization. We plot the posterior predicted mean given a plug-in estimate, $\mathbb{E}[f(x)|\mathcal{D}, \hat{\theta}]$. (b) Bayesian approach based on HMC. We plot the posterior predicted mean, marginalizing over hyper-parameters, $\mathbb{E}[f(x)|\mathcal{D}]$. Generated by `gp_kernel_opt.ipynb`.

Equation (18.164). For example, [MA10] shows how to use slice sampling (Section 12.4.1) for this task, [Hen+15] shows how to use HMC (Section 12.5), and [BBV11a] shows how to use SMC (Chapter 13).

In Figure 18.18, we illustrate the difference between kernel optimization vs kernel inference. We fit a 1d dataset using a kernel of the form

$$\mathcal{K}(r) = \sigma_1^2 \mathcal{K}_{\text{SE}}(r; \tau) \mathcal{K}_{\cos}(r; \rho_1) + \sigma_2^2 \mathcal{K}_{32}(r; \rho_2) \quad (18.165)$$

where $\mathcal{K}_{\text{SE}}(r; \ell)$ is the squared exponential kernel (Equation (18.11)), $\mathcal{K}_{\cos}(r; \rho_1)$ is the cosine kernel (Equation (18.18)), and $\mathcal{K}_{32}(r; \rho_2)$ is the Matern $\frac{3}{2}$ kernel (Equation (18.14)). We then compute a point-estimate of the kernel parameters using empirical Bayes, and posterior samples using HMC. We then predicting the posterior mean of f on a 1d test set by plugging in the MLE or averaging over samples. We see that the latter captures more uncertainty (beyond the uncertainty captured by the Gaussian itself).

18.6.3 Multiple kernel learning for additive kernels

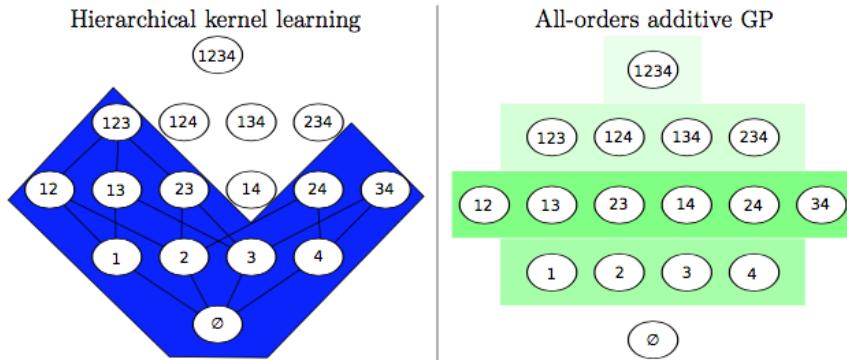
A special case of kernel learning arises when the kernel is a sum of B base kernels

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{b=1}^B w_b \mathcal{K}_b(\mathbf{x}, \mathbf{x}') \quad (18.166)$$

Optimizing the weights $w_b > 0$ using structural risk minimization is known as **multiple kernel learning**; see e.g., [Rak+08] for details.

Now suppose we constrain the base kernels to depend on a subset of the variables. Furthermore, suppose we enforce a hierarchical inclusion property (e.g., including the kernel k_{123} means we must

1
2
3
4
5
6
7
8
9
10
11
12
13



14
15 *Figure 18.19: Comparison of different additive model classes for a 4d function. Circles represent different*
16 *interaction terms, ranging from first-order to fourth-order. Color shades represent different weighting terms.*
17 *Left: hierarchical kernel learning uses a nested hierarchy of terms. Right: additive GPs use a weighted sum*
18 *of additive kernels of different orders. From Figure 6.2 of [Duv14]. Used with kind permission of David*
19 *Duvenaud.*

20

21

22

23 also include k_{12} , k_{13} and k_{23}), as illustrated in Figure 18.19(left). This is called **hierarchical kernel**
24 **learning**. We can find a good subset from this model class using convex optimization [Bac09];
25 however, this requires the use of cross validation to estimate the weights. A more efficient approach
26 is to use the empirical Bayes approach described in [DNR11].

27 In many cases, it is common to restrict attention to first order additive kernels, i.e., $\mathcal{K}(\mathbf{x}, \mathbf{x}') =$
28 $\sum_{d=1}^D \mathcal{K}_d(x_d, x'_d)$. The resulting function has the form

29

$$30 \quad f(\mathbf{x}) = f_1(x_1) + \dots + f_D(x_D) \quad (18.167)$$

31

32 This is called a **generalized additive model** or **GAM**.

33 Figure 18.20 shows an example of this, where each base kernel has the form $\mathcal{K}_d(x_d, x'_d) =$
34 $\sigma_d^2 \text{SE}(x_d, x'_d | \ell_d)$. In Figure 18.20, we see that the σ_d^2 terms for the coarse and fine features are
35 set to zero, indicating that these inputs have no impact on the response variable.

36 [DBW20] considers additive kernels operating on different linear projections of the inputs:

37

$$38 \quad 39 \quad \mathcal{K}(\mathbf{x}, \mathbf{x}') = \sum_{b=1}^B w_b \mathcal{K}_b(\mathbf{P}_b \mathbf{x}, \mathbf{P}_b \mathbf{x}') \quad (18.168)$$

40

41 Surprisingly, they show that these models can match or exceed the performance of kernels operating
42 on the original space, even when the projections are into a **single dimension**, and not learned. In
43 other words, it is possible to reduce many regression problems to a single dimension without loss
44 in performance. This finding is particularly promising for scalable inference, such as KISS (see
45 Section 18.5.5.4), and active learning, which are greatly simplified in a low dimensional setting.
46

47

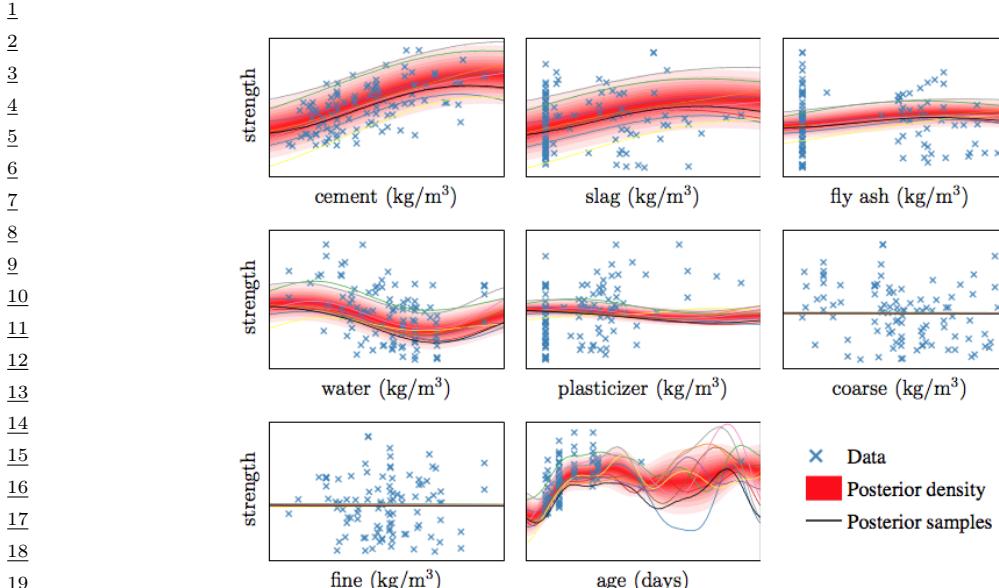


Figure 18.20: Predictive distribution of each term in a GP-GAM model applied to a dataset with 8 continuous inputs and 1 continuous output, representing the strength of some concrete. From Figure 2.7 of [Duv14]. Used with kind permission of David Duvenaud.

18.6.4 Automatic search for compositional kernels

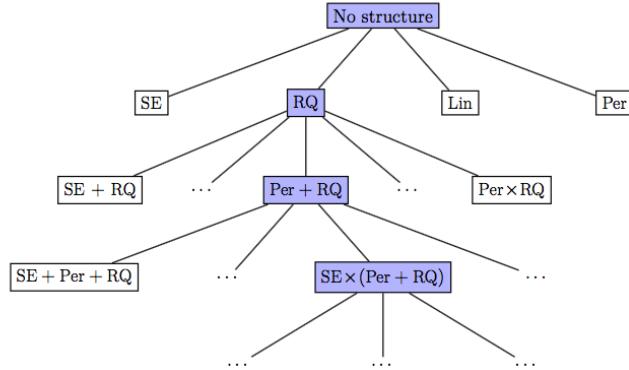
Although the above methods can estimate the hyperparameters of a specified set of kernels, they do not choose the kernels themselves (other than the special case of selecting a subset of kernels from a set). In this section, we describe a method, based on [Duv+13], for sequentially searching through the space of increasingly complex GP models so as to find a parsimonious description of the data.

We start with a simple kernel, such as the white noise kernel, and then consider replacing it with a set of possible alternative kernels, such as an SE kernel, RQ kernel, etc. We use the BIC score (Section 3.7.5.2) to evaluate each candidate model (choice of kernel) m . This has the form $BIC(m) = \log p(\mathcal{D}|m) - \frac{1}{2}|m|\log N$, where $p(\mathcal{D}|m)$ is the marginal likelihood, and $|m|$ is the number of parameters. The first term measures fit to the data, and the second term is a complexity penalty. We can also consider replacing a kernel by the addition of two kernels, $k \rightarrow (k + k')$, or the multiplication of two kernels, $k \rightarrow (k \times k')$. See Figure 18.21 for an illustration of the search space.

Searching through this space is similar to what a human expert would do. In particular, if we find structure in the residuals, such as periodicity, we can propose a certain “move” through the space. We can also start with some structure that is assumed to hold globally, such as linearity, but if we find this only holds locally, we can multiply the kernel by an SE kernel. We can also add input dimensions incrementally, to capture higher order interactions.

Figure 18.22 shows the output of this process applied to a dataset of monthly totals of international airline passengers. The input to the GP is the set of time stamps, $\mathbf{x} = 1 : t$; there are no other features.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



16 *Figure 18.21: Example of a search tree over kernel expressions. From Figure 3.2 of [Duv14]. Used with kind
17 permission of David Duvenaud.*

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

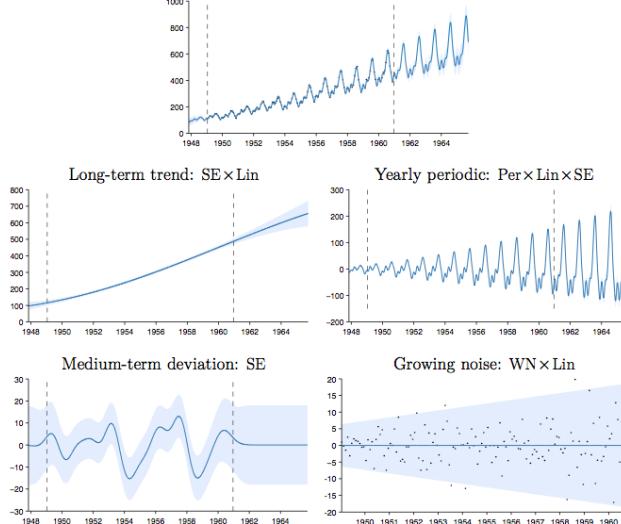
38

39

40

41

Complete Model: $SE \times Lin + Per \times Lin \times SE + Lin \times SE + WN \times Lin$



42 *Figure 18.22: Top row: airline dataset and posterior distribution of the model discovered after a search of
43 depth 10. Subsequent rows: predictions of the individual components. From Figure 3.5 of [Duv14], based on
44 [Llo+14]. Used with kind permission of David Duvenaud.*

45

46

47

The observed data lies in between the dotted vertical lines; curves outside of this region are extrapolations. We see that the system has discovered a fairly interpretable set of patterns in the data. Indeed, it is possible to devise an algorithm to automatically convert the output of this search process to a natural language summary, as shown in [Llo+14]. In this example, it summarizes the data as being generated by the addition of 4 underlying trends: a linearly increasing function; an approximately periodic function with a period of 1.0 years, and with linearly increasing amplitude; a smooth function; and uncorrelated noise with linearly increasing standard deviation.

Recently, [Sun+18] showed how to create a DNN which learns the kernel given two input vectors. The hidden units are defined as sums and products of elementary kernels, as in the above search based approach. However, the DNN can be trained in a differentiable way, so is much faster.

18.6.5 Spectral mixture kernel learning

Any shift-invariant (stationary) kernel can be converted via the Fourier transform to its dual form, known as its **spectral density**. This means that learning the spectral density is equivalent to learning any shift-invariant kernel. For example, if we take the Fourier transform of an RBF kernel, we get a Gaussian spectral density centered at the origin. If we take the Fourier transform of a Matern kernel, we get a Student- t spectral density centred at the origin. Thus standard approaches to multiple kernel learning, which typically involve additive compositions of RBF and Matern kernels with different length-scale parameters, amount to density estimation with a scale mixture of Gaussian or Student- t distributions at the origin. Such models are very inflexible for density estimation, and thus also very limited in being able to perform kernel learning.

On the other hand, *scale-location* mixture of Gaussians can model any density to arbitrary precision. Moreover, with even a small number of components these mixtures of Gaussians are highly flexible. Thus a spectral density corresponding to a scale-location mixture of Gaussians forms an expressive basis for all shift-invariant kernels. One can evaluate the inverse Fourier transform for a Gaussian mixture analytically, to derive the **spectral mixture kernel** [WA13], which we can express for one-dimensional inputs x as:

$$\mathcal{K}(x, x') = \sum_i w_i \cos((x - x')(2\pi\mu_i)) \exp(-2\pi^2(x - x')^2 v_i) \quad (18.169)$$

The mixture weights w_i , as well as the means μ_i and variances v_i of the Gaussians in the spectral density, can be learned by empirical Bayes optimization (Section 18.6.1) or in a fully-Bayesian procedure (Section 18.6.2) [Jan+17]. We illustrate the former approach in Figure 18.23.

By learning the parameters of the spectral mixture kernel, we can discover representations that enable extrapolation — to make reasonable predictions far away from the data. For example, in Section 19.3.3.1, compositions of kernels are carefully hand-crafted to extrapolate CO₂ concentrations. But in this instance, the human statistician is doing all of the interesting representation learning. Figure Figure 18.24 shows Gaussian processes with learned spectral mixture kernels instead automatically extrapolating on CO₂ and airline passenger problems.

These kernels can also be used to extrapolate higher dimensional large-scale spatio-temporal patterns. Large datasets can provide relatively more information for expressive kernel learning. However, scaling an expressive kernel learning approach poses different challenges than scaling a standard Gaussian process model. One faces additional computational constraints, and the need to retain significant model structure for expressing the rich information available in a large dataset.

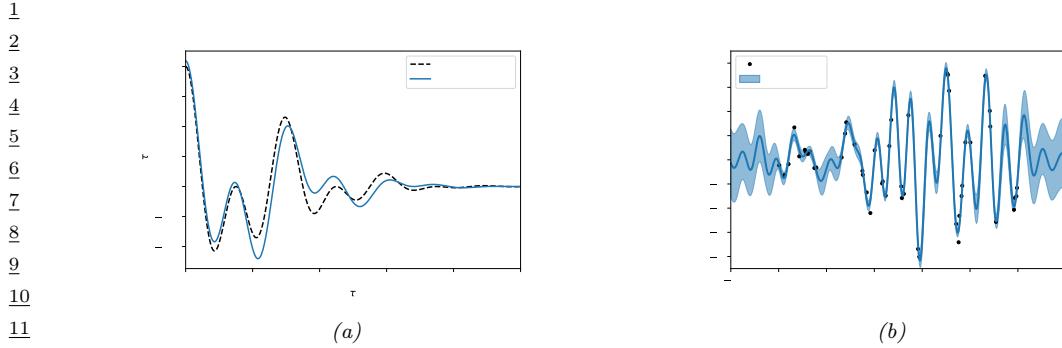


Figure 18.23: Illustration of a GP with a spectral mixture kernel in 1d. (a) Learned vs true kernel. (b) Predictions using learned kernel. Generated by `gp_spectral_mixture.ipynb`.

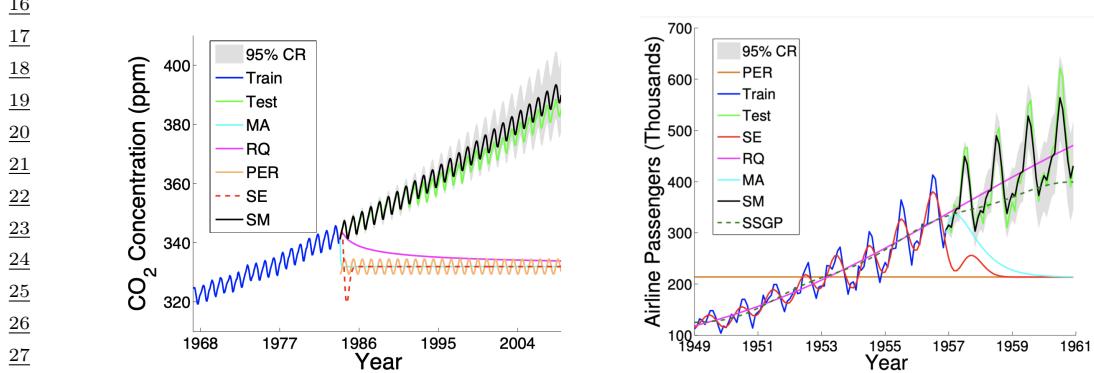


Figure 18.24: Extrapolations (point predictions and 95% credible set) on CO₂ and airline datasets using Gaussian processes with Matern, rational quadratic, periodic, RBF (SE), and spectral mixture kernels, each with hyperparameters learned using empirical Bayes. From Figure from [Wil14].

Indeed, in Figure 18.24 we can separately understand the effects of the kernel learning approach and scalable inference procedure, in being able to discover structure necessary to extrapolate textures. An expressive kernel model and a scalable inference approach that preserves a *non-parametric* representation are needed for good performance.

Structure exploiting inference procedures, such as Kronecker methods, as well as KISS-GP and conjugate gradient based approaches, are appropriate for these tasks — since they generally preserve or exploit existing structure, rather than introducing approximations that corrupt the structure. Spectral mixture kernels combined with these scalable inference techniques have been used to great effect for spatiotemporal extrapolation problems, including land-surface temperature forecasting, epidemiological modeling, and policy-relevant applications.

47

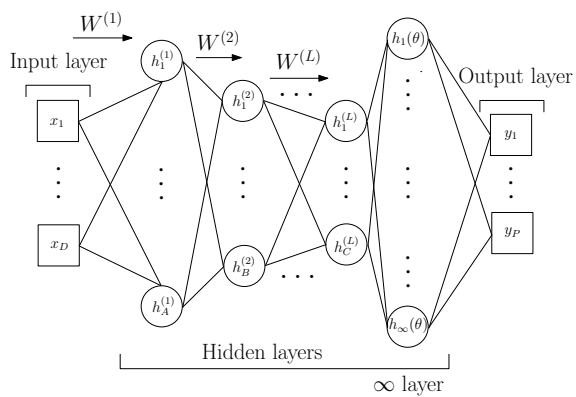


Figure 18.25: Deep Kernel Learning: A Gaussian process with a deep kernel maps D dimensional inputs \mathbf{x} through L parametric hidden layers followed by a hidden layer with an infinite number of basis functions, with base kernel hyperparameters θ . Overall, a Gaussian process with a deep kernel produces a probabilistic mapping with an infinite number of adaptive basis functions parametrized by $\gamma = \{\mathbf{w}, \theta\}$. All parameters γ are learned through the marginal likelihood of the Gaussian process. From Figure 1 of [Wil+16b].

18.6.6 Deep kernel learning

Deep kernel learning [SH07; Wil+16b] combines the structural properties of neural networks with the non-parametric flexibility and uncertainty representation provided by Gaussian processes. For example, we can define a “deep RBF kernel” as follows:

$$\mathcal{K}_\theta(\mathbf{x}, \mathbf{x}') = \exp \left[-\frac{1}{2\sigma^2} \|\mathbf{h}_\theta^L(\mathbf{x}) - \mathbf{h}_\theta^L(\mathbf{x}')\|^2 \right] \quad (18.170)$$

where $\mathbf{h}_\theta^L(\mathbf{x})$ are the outputs of layer L from a DNN. We can then learn the parameters θ by maximizing the marginal likelihood of the Gaussian processes.

This framework is illustrated in Figure 18.25. We can understand the neural network features as inputs into a base kernel. The neural network can either be (i) pre-trained, (ii) learned jointly with the base kernel parameters, or (iii) pre-trained and then fine-tuned through the marginal likelihood. This approach can be viewed as a “last-layer” Bayesian model, where a Gaussian process is applied to the final layer of a neural network. The base kernel often provides a good measure of distance in feature space, desirably encouraging predictions to have high uncertainty as we move far away from the data.

We can use deep kernel learning to help the GP learn discontinuous functions, as illustrated in Figure 18.26. On the left we show the results of a GP with a standard Matern $\frac{3}{2}$ kernel. It is clear that the out-of-sample predictions are poor. On the right we show the results of the same model where we first transform the input through a learned 2 layer MLP (with 15 and 10 hidden units). It is clear that the model is working much better.

As a more complex example, we consider a regression problem where we wish to map faces (vectors of pixel intensities) to a continuous valued orientation angle. In Figure 18.27, we evaluate the deep kernel matrix (with RBF and spectral mixture base kernels, discussed in Section 18.6.5) on data ordered by orientation angle. We can see that the learned deep kernels, in the left two panels, have a

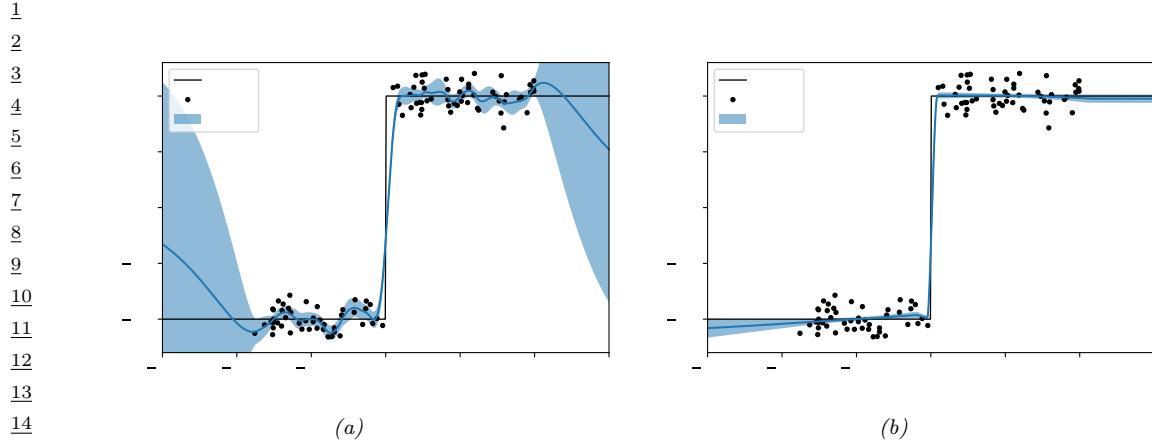


Figure 18.26: Modeling a discontinuous function with (a) a GP with a “shallow” Matern $\frac{3}{2}$ kernel, and (b) a GP with a “deep” MLP + Matern kernel. Generated by [gp_deep_kernel_learning.py](#).

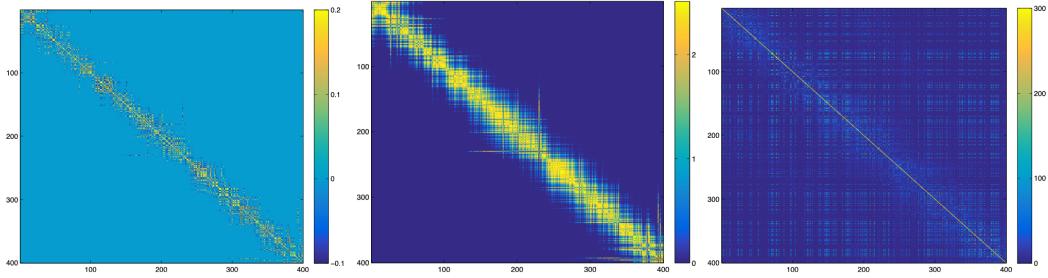


Figure 18.27: **Left:** The learned covariance matrix of a deep kernel with spectral mixture base kernel on a set of test cases for the **Olivetti faces dataset**, where the test samples are ordered according to the orientations of the input faces. **Middle:** The respective covariance matrix using a deep kernel with RBF base kernel. **Right:** The respective covariance matrix using a standard RBF kernel. From Figure 5 of [Wil+16b].

pronounced diagonal band, meaning that they have *discovered* that faces with similar orientation angles are correlated. On the other hand, in the right panel we see that the entries even for a learned RBF kernel are highly diffuse. Since the RBF kernel essentially uses Euclidean distance as a metric for similarity, it is unable to learn a representation that effectively solves this problem. In this case, one must do highly non-Euclidean metric learning.

40

41 18.6.7 Functional kernel learning

42

43 Gaussian processes naturally give rise to a function-space view of modelling, whereby we place a prior
 44 distribution directly over functions that could model our data. Since the kernel crucially influences
 45 the generalization properties of the Gaussian process, it is natural to take this perspective one step
 46 further in a hierarchical model — so that we can represent the prior belief that the kernel does not
 47

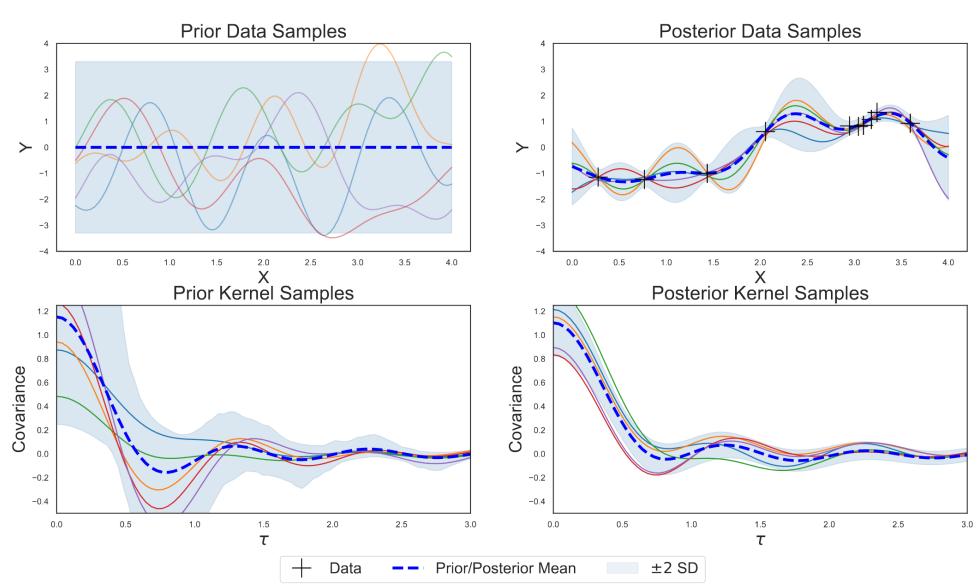


Figure 18.28: In the above two panels we show the conventional function-space approach to modeling data with GPs, with prior and posterior draws. In the bottom two rows we apply this perspective to kernels, with priors and posterior draws from a distribution over kernels, in a functional kernel learning model. From Figure 1 of [Ben+19].

have a simple functional form and that we have uncertainty over its values. This approach is referred to as **functional kernel learning** (FKL) [Ben+19].

FKL provides a function-space distribution over kernels by modeling a spectral density with a transformed Gaussian process. It is then possible to specify the prior expected kernel to be whatever one would have used otherwise, such as an RBF kernel, while enabling a non-parametric relaxation around its values. We can now imagine drawing prior kernel functions, observing data, and inferring posterior kernels. When we wish to form predictions, we can then marginalize this posterior over kernel functions.

This approach is shown in [Ben+19] to provide strong extrapolation performance on a number of spatiotemporal problems, without requiring much manual intervention.

18.7 GPs and DNNs

In this section, we discuss various connections between Gaussian processes (GPs) and deep neural networks (DNNs). Note that this is a rapidly changing field, so we just present a few highlights, rather than going into depth.

1 **18.7.1 Kernels derived from random DNNs (NN-GP)**

2
3
4 In Section 17.2.1, we showed that the prior over parameters of the DNN indirectly induces a prior
5 over functions. If we use infinitely wide MLPs with randomly initialized weights (drawn from a
6 Gaussian), the resulting distribution over functions is the same as a GP with a certain kernel. This
7 result was first shown for one layer MLPs in [Nea96; Wil98], and was later extended to deep MLPs
8 in [DFS16; Lee+18]. The resulting kernel is called the **NNGP** kernel [Lee+18] (also called the
9 **compositional kernel** [DFS16]).

10 A similar result holds for CNNs, where we let the number of channels per layer go to infinity, as
11 shown in [Nov+19]. In fact, one can show this phenomenon for a large class of neural networks, such
12 as RNNs and models with attention [Yan19]. Below we derive the result for the MLP case.

14151617181920

21 **18.7.1.1 Result for MLP with one hidden layer**

22

23
24 Let us start by considering an MLP with a single hidden layer with inputs $\mathbf{x} \in \mathbb{R}^{D_0}$ and outputs
25 $f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{z}^2 \in \mathbb{R}^{D_2}$, defined as follows:

26272829

$$\begin{aligned} \text{30} \quad z_i^2(\mathbf{x}) &= b_i^2 + \sum_{j=1}^{D_1} W_{ij}^2 h_j^1(\mathbf{x}), \quad h_j^1(\mathbf{x}) = \varphi \left(b_j^1 + \sum_{k=1}^{D_0} W_{jk}^1 x_k \right) \end{aligned} \tag{18.171}$$

3132333435

We call z_i^ℓ the pre-activation for unit i at layer ℓ , and h_i^ℓ the post-activation.

We assume $W_{ij}^\ell \sim \mathcal{N}(0, \sigma_w^2 / D_\ell)$ and $b_i^\ell \sim \mathcal{N}(0, \sigma_b^2)$. Because the weight and bias parameters are iid, the post-activations $h_i^1, h_{i'}^1$ are independent for different units, $i \neq i'$. Therefore we can consider a single unit i in our analysis.

Since the post-activations $z_i^2(\mathbf{x})$ is a sum of iid terms, it follows from the central limit theorem (CLT) that, as $D_1 \rightarrow \infty$, the distribution of $z_i^2(\mathbf{x})$ will tend towards a Gaussian. Furthermore, from the multidimensional CLT, any finite collection $\{z_i^2(\mathbf{x}_1), \dots, z_i^2(\mathbf{x}_N)\}$ will tend towards a joint Gaussian, which is the definition of a GP. We have one GP for each output dimension i , but the parameters will be the same for each i .

The mean of this process is given by $\mu(\mathbf{x}) = \mathbb{E}[f_i(\mathbf{x})] = \mathbb{E}[z_i^2(\mathbf{x})] = 0$, since the parameters have zero mean. Using the independence of the parameters, we can derive the covariance of this process

47

as follows:⁴

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f_i(\mathbf{x})f_i(\mathbf{x}')] = \mathbb{E}[z_i^2(\mathbf{x})z_i^2(\mathbf{x}')] \quad (18.172)$$

$$= \mathbb{E}\left[\left(b_i^2 + \sum_{j=1}^{D_1} W_{ij}^2 h_j^1(\mathbf{x})\right)\left(b_i^2 + \sum_{j=1}^{D_1} W_{ij}^2 h_j^1(\mathbf{x}')\right)\right] \quad (18.173)$$

$$= \mathbb{E}[(b_i^2)(b_i^2)] + \mathbb{E}\left[\sum_{j=1}^{D_1} (W_{ij}^2)(W_{ij}^2)h_j^1(\mathbf{x})h_j^1(\mathbf{x}')\right] \quad (18.174)$$

$$= \sigma_b^2 + \sum_{j=1}^{D_1} \frac{\sigma_w^2}{D_1} \mathbb{E}[h_j^1(\mathbf{x})h_j^1(\mathbf{x}')] \quad (18.175)$$

$$= \sigma_b^2 + \sigma_w^2 C(\mathbf{x}, \mathbf{x}') \quad (18.176)$$

where we have defined

$$C(\mathbf{x}, \mathbf{x}') \triangleq \mathbb{E}[\varphi(b + \mathbf{w}^\top \mathbf{x})\varphi(b + \mathbf{w}^\top \mathbf{x}')] \quad (18.177)$$

where expectations are wrt b, \mathbf{w} .

In some cases we can compute the above Gaussian integral analytically. For example, for the erf activation function, [Wil98] derive the following result:

$$C(\mathbf{x}, \mathbf{x}') = \frac{2}{\pi} \sin^{-1} \left(\frac{2\mathbf{x} \cdot \mathbf{x}'}{\sqrt{(1 + 2\mathbf{x} \cdot \mathbf{x})(1 + 2\mathbf{x}' \cdot \mathbf{x}')}} \right) \quad (18.178)$$

This is sometimes called the **neural net kernel**. Note that this is a **nonstationary kernel** (i.e., not a function of $\|\mathbf{x} - \mathbf{x}'\|$), and sample paths from it are nearly discontinuous and tend to constant values for large positive or negative inputs. See Figure 18.29 for an illustration.

[CS09] extend the above result to the case of ReLU activations. However, in general we need to use Monte Carlo approximation, or numerical integration, to compute the kernel function.

18.7.1.2 Result for deep MLPs

We can extend the above result by induction to show that the activations in every layer are distributed as a GP, if we let each of the hidden layer widths go to infinity. More precisely, let us assume that the output from the previous layer, $\{z_j^{\ell-1}(\mathbf{x})\}$, is a GP for each j . Now consider layer ℓ :

$$z_i^\ell(\mathbf{x}) = b_i^\ell + \sum_{j=1}^{D_{\ell-1}} W_{ij}^\ell h_j^{\ell-1}(\mathbf{x}), \quad h_j^{\ell-1}(\mathbf{x}) = \varphi(z_j^{\ell-1}(\mathbf{x})) \quad (18.179)$$

The $\{h_j^{\ell-1}(\mathbf{x})\}_j$ are iid random variables, since the inputs $\{z_j^{\ell-1}(\mathbf{x})\}_j$ are also iid (being jointly Gaussian but with zero covariance between units j). Thus as $D_\ell \rightarrow \infty$, we have that $z_i^\ell \sim \text{GP}(0, \mathcal{K}^\ell)$,

⁴ We are using the fact that $u \sim \mathcal{N}(0, \sigma^2)$ implies $\mathbb{E}[u^2] = \mathbb{V}[u] = \sigma^2$.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

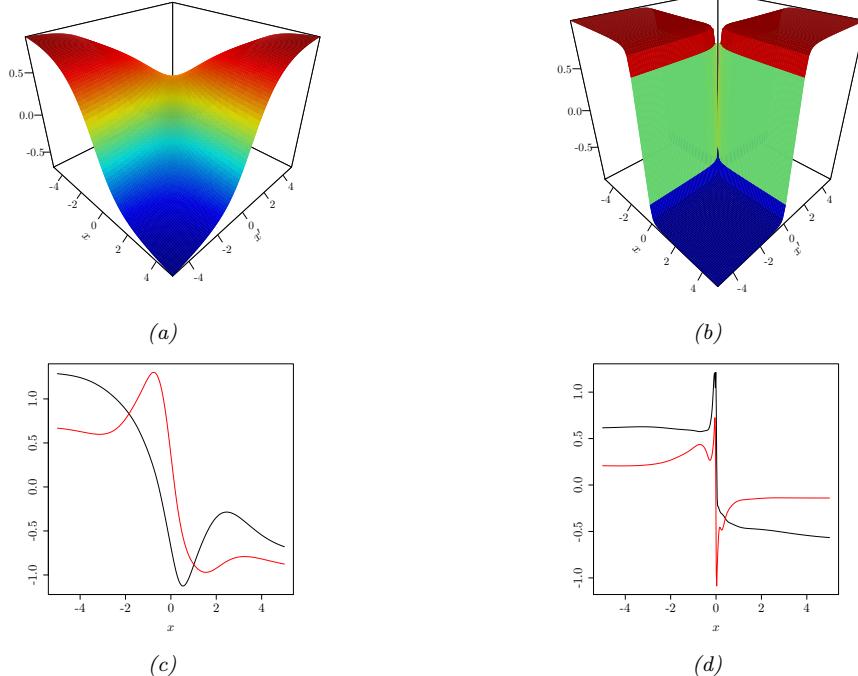


Figure 18.29: Illustration of the neural net kernel, corresponding to $\mathcal{K}(x, x') = \mathbb{E}[\text{erf}((w_0 + w_1 x)\text{erf}(w_0 + w_1 x'))]$, where $w_0 \sim \mathcal{N}(0, \sigma_0^2)$ and $w_1 \sim \mathcal{N}(0, \sigma_1^2)$. (a) We plot $\mathcal{K}(x, x')$ for $\sigma_1 = 1$ (b) $\sigma_1 = 50$. In both cases, $\sigma_0 = 1$. (c-d). We plot samples from $f \sim \text{GP}(0, \mathcal{K})$. From Figure 2 of [Moh+19b]. Used with kind permission of Hossein Mohammadi.

where

$$\mathcal{K}^\ell(\mathbf{x}, \mathbf{x}') = \mathbb{E}[z_i^\ell(\mathbf{x})z_i^\ell(\mathbf{x}')] \quad (18.180)$$

$$= \sigma_b^2 + \sigma_w^2 \mathbb{E}_{z_{\ell-1} \sim \text{GP}(0, \mathcal{K}^{\ell-1})} [\varphi(z_i^{\ell-1}(\mathbf{x}))\varphi(z_i^{\ell-1}(\mathbf{x}'))] \quad (18.181)$$

$$= \sigma_b^2 + \sigma_w^2 \mathcal{C} \left(\begin{bmatrix} \mathcal{K}^{\ell-1}(\mathbf{x}, \mathbf{x}) & \mathcal{K}^{\ell-1}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{\ell-1}(\mathbf{x}', \mathbf{x}) & \mathcal{K}^{\ell-1}(\mathbf{x}', \mathbf{x}') \end{bmatrix} \right) \quad (18.182)$$

$$\mathcal{C}(\Sigma) \triangleq \mathbb{E}[\varphi(u)\varphi(v)], \quad (u, v) \sim \mathcal{N}(\mathbf{0}, \Sigma) \quad (18.183)$$

For the base case \mathcal{K}^1 , we have

$$\mathcal{K}^1(\mathbf{x}, \mathbf{x}') = \mathbb{E}[z_i^1(\mathbf{x})z_i^1(\mathbf{x}')] = \mathbb{E}[(b + \mathbf{w}^\top \mathbf{x})(b + \mathbf{w}^\top \mathbf{x}')] \quad (18.184)$$

$$= \mathbb{E}[b^2] + \mathbb{E}[(\mathbf{x}^\top \mathbf{w})(\mathbf{w}^\top \mathbf{x}')] = \sigma_b^2 + \frac{\sigma_w^2}{D_1} \mathbf{x}^\top \mathbf{x}' \quad (18.185)$$

See Figure 18.30 for an illustration of how the $N \times N$ covariance matrix between examples in a minibatch evolves across layers, and how this compares to the $D_\ell \times N$ matrix of activations produced by a parametric model.

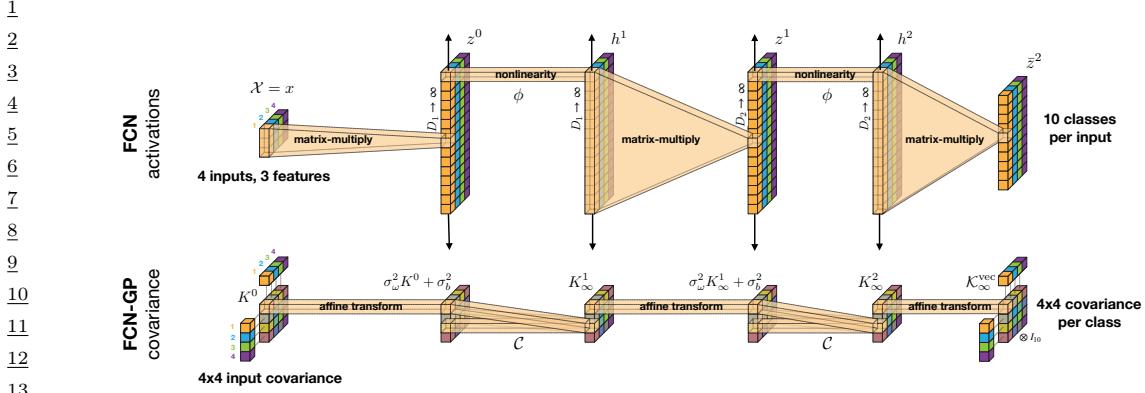


Figure 18.30: Comparison of a parametric fully connected network (FCN) and its GP equivalent. Here there are $N = 4$ input examples, each of which are $D_0 = 3$ dimensional. From Figure 3 of [Nov+19]. Used with kind permission of Roman Novak.

18.7.2 Kernels derived from trained DNNs (neural tangent kernel)

Although it is useful to convert a random, untrained DNN to a GP in order to gain insight into the corresponding implicit prior over functions, it is even more useful to derive the posterior over functions that result from training a DNN to minimize $\mathcal{L}(\theta; \mathcal{D})$.

Suppose we just train the final layer weights to compute $\hat{\theta}^L$; let the resulting parameters be $\theta = (\theta^1, \dots, \theta^{L-1}, \hat{\theta}^L)$ where all are random except the last. The resulting prediction $f(\mathbf{x}; \hat{\theta})$ will be equivalent to exact GP inference $\mathbb{E}[f(\mathbf{x})|\mathcal{D}]$ using the NN-GP kernel.

Now suppose we train all the weights. Let $\mathbf{f} = [f(\mathbf{x}_n; \theta)]_{n=1}^N$ be the $N \times 1$ prediction vector, let $\nabla_f \mathcal{L} = [\frac{\partial \mathcal{L}}{\partial f(\mathbf{x}_n)}]_{n=1}^N$ be the $N \times 1$ loss gradient vector, let $\theta = [\theta_p]_{p=1}^P$ be the $P \times 1$ vector of parameters, and let $\nabla_\theta \mathbf{f} = [\frac{\partial f(\mathbf{x}_n)}{\partial \theta_p}]$ be the $P \times N$ matrix of partials. Suppose we perform continuous time gradient descent with fixed learning rate η . The parameters evolve over time as follows:

$$\partial_t \theta_t = -\eta \nabla_\theta \mathcal{L}(\mathbf{f}_t) = -\eta \nabla_\theta \mathbf{f}_t \cdot \nabla_f \mathcal{L}(\mathbf{f}_t) \quad (18.186)$$

Thus the function evolves over time as follows:

$$\partial_t \mathbf{f}_t = \nabla_\theta \mathbf{f}_t^\top \partial_t \theta_t = -\eta \nabla_\theta \mathbf{f}_t^\top \nabla_\theta \mathbf{f}_t \cdot \nabla_f \mathcal{L}(\mathbf{f}_t) = -\eta \mathcal{T}_t \cdot \nabla_f \mathcal{L}(\mathbf{f}_t) \quad (18.187)$$

where \mathcal{T}_t is the $N \times N$ kernel matrix

$$\mathcal{T}_t(\mathbf{x}, \mathbf{x}') \triangleq \nabla_\theta f_t(\mathbf{x}) \cdot \nabla_\theta f_t(\mathbf{x}') = \sum_{p=1}^P \frac{\partial f(\mathbf{x}; \theta)}{\partial \theta_p} \Big|_{\theta_t} \frac{\partial f(\mathbf{x}'; \theta)}{\partial \theta_p} \Big|_{\theta_t} \quad (18.188)$$

If we let the learning rate η become infinitesimally small, and the widths go to infinity, one can show that this kernel converges to a constant matrix, this is known as the **neural tangent kernel** or **NTK** [JGH18]:

$$\mathcal{T}(\mathbf{x}, \mathbf{x}') \triangleq \nabla_\theta f(\mathbf{x}; \theta_\infty) \cdot \nabla_\theta f(\mathbf{x}'; \theta_\infty) \quad (18.189)$$

¹ **18.7.2.1 Computing the NTK**

³ We can compute the NTK in a recursive fashion, similar to the computation of the NN-GP kernel in
⁴ Section 18.7.1.1. In particular, let us define
⁵

$$\dot{\mathcal{C}}(\Sigma) \triangleq \mathbb{E}[\varphi'(u)\varphi'(v)], \quad (u, v) \sim \mathcal{N}(\mathbf{0}, \Sigma) \quad (18.190)$$

⁷ Then one can show, by induction, that
⁸

$$\mathcal{T}^l(\mathbf{x}, \mathbf{x}') = \mathcal{K}^l(\mathbf{x}, \mathbf{x}') + \sigma_w^2 \mathcal{T}^{l-1}(\mathbf{x}, \mathbf{x}') \dot{\mathcal{C}} \begin{pmatrix} \mathcal{K}^{l-1}(\mathbf{x}, \mathbf{x}) & \mathcal{K}^{l-1}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{l-1}(\mathbf{x}', \mathbf{x}) & \mathcal{K}^{l-1}(\mathbf{x}', \mathbf{x}') \end{pmatrix} \quad (18.191)$$

¹² where \mathcal{K} is the NN-GP kernel.

¹³ Details on how to compute the $\dot{\mathcal{C}}$ function for various models, such as CNNs, graph neural nets,
¹⁴ and general neural nets, can be found in [Aro+19; Du+19; Yan19]. A software library to compute the
¹⁵ NNGP and NTK is available in [Ano19].

¹⁶ **18.7.2.2 Connections with GPs**

¹⁸ Suppose \mathcal{L} is the square loss, and \mathbf{y} is the true prediction targets, so
¹⁹

$$\nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) = \frac{1}{2N} \nabla_{\mathbf{f}} \|\mathbf{f}_t - \mathbf{y}\|_2^2 = \frac{1}{N} (\mathbf{f}_t - \mathbf{y}) \quad (18.192)$$

²² From Equation (18.187), and since the NTK kernel converges to a constant matrix, the function
²³ evolves linearly over time as follows:
²⁴

$$\partial_t \mathbf{f}_t = -\eta \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{f}_t) = -\eta \mathcal{T}(\mathbf{f}_t - \mathbf{y}) \quad (18.193)$$

²⁷ We can solve this differential equation to get $f_t(\mathbf{x}) = \mathcal{T}(\mathbf{x}, \mathbf{X}) \mathcal{T}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{H}_t \mathbf{y}$, where $\mathbf{H}_t \triangleq \mathbf{I} - e^{-\eta t \mathcal{T}(\mathbf{X}, \mathbf{X})}$. Hence this converges to

$$f_\infty(\mathbf{x}) = \mathcal{T}(\mathbf{x}, \mathbf{X}) \mathcal{T}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y} = \mathcal{T}(\mathbf{x}, \mathbf{X})^\top \boldsymbol{\alpha} \quad (18.194)$$

³¹ where $\boldsymbol{\alpha} = \mathcal{T}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y}$. Thus we see that this is a linear model, $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\alpha}$, using a fixed set of
³² basis functions, $\boldsymbol{\phi}(\mathbf{x}) = [\mathcal{T}(\mathbf{x}, \mathbf{x}_1), \dots, \mathcal{T}(\mathbf{x}, \mathbf{x}_N)]$. This has the same form as kernel ridge regression
³³ (Section 18.3.7) without a ridge penalty term. This is known as “ridgeless regression” [LR18], and
³⁴ corresponds to a model that perfectly interpolates the training data. This also matches the mean of
³⁵ a GP with no observation noise. However, the variance of $f_\infty(\mathbf{x})$ is different from a GP.
³⁶

³⁷ **18.7.2.3 Discussion**

³⁸ The assumptions behind the NTK results in the parameters barely changing from their initial values
³⁹ (which is why a linear approximation around the starting parameters is valid). This can still lead
⁴⁰ to a change in the final predictions (and zero final training error), because the final layer weights
⁴¹ can learn to use the random features just like in kernel regression. However, this phenomenon —
⁴² which has been called “lazy training” [COB18] — is not representative of DNN behavior in practice
⁴³ [Woo+19], where parameters often change a lot.

⁴⁵ A theoretical analysis that more closely matches practice is currently being developed by various
⁴⁶ authors, but is beyond the scope of this chapter (see e.g., [Fan+19] for one of many recent works).

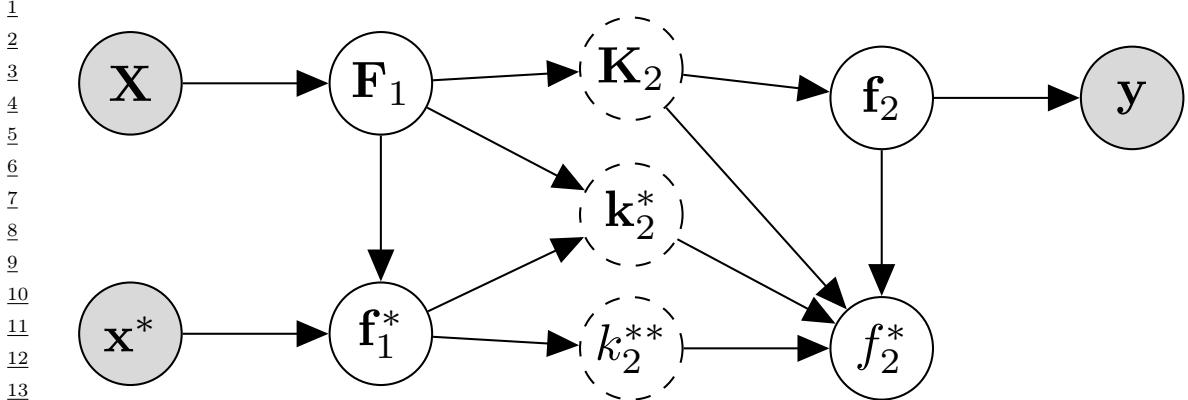


Figure 18.31: Graphical model corresponding to a deep GP with 2 layers. The dashed nodes are deterministic functions of their parents, and represent kernel matrices. The shaded nodes are observed, the unshaded nodes are hidden. From Figure 5 of [PC21]. Used with kind permission of Geoff Pleis.

18.7.3 Deep GPs

A **deep Gaussian process** or **DGP** is a composition of GPs [DL13]. (See [Jak21] for a recent survey.) More formally, a DGP of L layers is a hierarchical model of the form

$$\text{DGP}(\mathbf{x}) = f_L \circ \dots \circ f_1(\mathbf{x}), \quad f_i(\cdot) = [f_i^{(1)}(\cdot), \dots, f_i^{(H_i)}(\cdot)], \quad f_i^{(j)} \sim \text{GP}(0, \mathcal{K}_i(\cdot, \cdot)) \quad (18.195)$$

This is similar to a deep neural network, except the hidden nodes are now hidden functions.

A natural question is: what is gained by this approach compared to a standard GP? Although conventional single-layer GPs are nonparametric, and can model any function (assuming the use of a non-degenerate kernel) with enough data, in practice their performance is limited by the choice of kernel. This can be partially overcome by using a DGP, as we show in Section 18.7.3.2. Unfortunately, posterior inference in DGPs is challenging, as we discuss in Section 18.7.3.3.

In Section 18.7.3.4, we discuss the expressive power of infinitely wide DGPs, and in Section 18.7.3.5 we discuss connections with DNNs.

18.7.3.1 Construction of a deep GP

In this section we give an example of a 2 layer DGP, following the presentation in [PC21]. Let $f_1^{(j)} \sim \text{GP}(0, \mathcal{K}_1)$ for $j = 1 : H_1$, where H_1 is the number of hidden units, and $f_2 \sim \text{GP}(0, \mathcal{K}_2)$. Assume we have labeled training data $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ and $\mathbf{y} = (y_1, \dots, y_N)$. Define $\mathbf{F}_1 = [f_1(\mathbf{x}_1), \dots, f_1(\mathbf{x}_N)]$ and $\mathbf{f}_2 = [f_2(f_1(\mathbf{x}_1)), \dots, f_2(f_1(\mathbf{x}_N))]$. Let \mathbf{x}^* be a test input and define $\mathbf{f}_1^* = f_1(\mathbf{x}^*)$ and $\mathbf{f}_2^* = f_2(\mathbf{f}_1^*)$. The corresponding joint distribution over all the random variables is given by

$$p(\mathbf{f}_2^*, \mathbf{f}_2, \mathbf{F}_1, \mathbf{f}_1, \mathbf{y}) = p(\mathbf{f}_2^* | \mathbf{f}_2, \mathbf{F}_1, \mathbf{f}_1) p(\mathbf{f}_2 | \mathbf{F}_1, \mathbf{f}_1^*) p(\mathbf{f}_1^*, \mathbf{F}_1) p(\mathbf{y} | \mathbf{f}_2) \quad (18.196)$$

where we drop the dependence on \mathbf{X} and \mathbf{x}^* for brevity. This is illustrated by the graphical model in Figure 18.31, where we define $\mathbf{K}_2 = \mathcal{K}_2(\mathbf{F}_1, \mathbf{F}_1)$, $\mathbf{k}_{2*} = \mathcal{K}_2(\mathbf{F}_1, \mathbf{f}_1^*)$, and $\mathbf{k}_{2**} = \mathcal{K}_2(\mathbf{f}_1^*, \mathbf{f}_1^*)$.

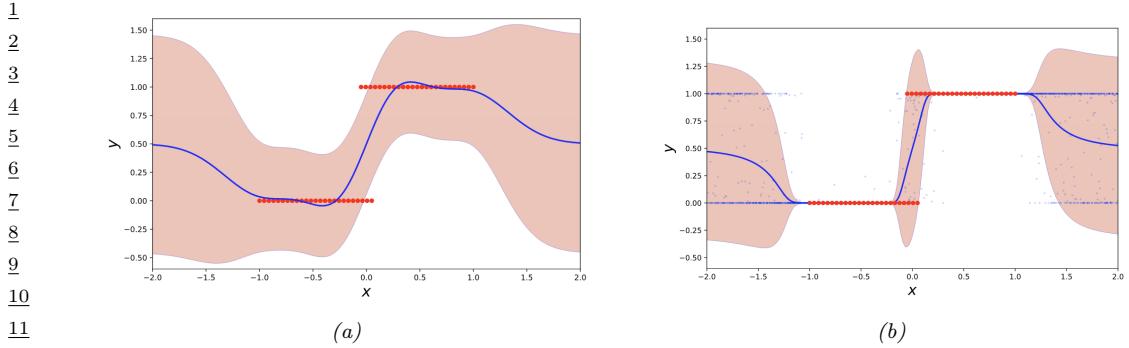


Figure 18.32: Some data (red points) sampled from a step function fit with (a) a standard GP with RBF kernel and (b) a deep GP with 4 layers of RBF kernels. The solid blue line is the posterior mean. The pink shaded area represents the posterior variance ($\mu(x) \pm 2\sigma(x)$). The thin blue dots in (b) represent posterior samples. From [Law19]. Used with kind permission of Neil Lawrence.

18.7.3.2 Example: 1d step function

Suppose we have data from a piecewise constant function. (This can often happen when modeling certain physical processes, which can exhibit saturation effects.) Figure 18.32a shows what happens if we fit data from such a step function using a standard GP with an RBF (Gaussian) kernel. Obviously this method oversmooths the function and does not “pick up on” the underlying discontinuity. It is possible to learn kernels that can capture such discontinuous (non-stationary) behavior by learning to warp the input with a neural net before passing into the RBF kernel (see Figure 18.26).

Another approach is to learn a sequence of smooth mappings which together capture the overall complex behavior, analogous to the approach in deep learning. Suppose we fit a 4 layer DGP with a single hidden unit at each layer; we will use an RBF kernel. Thus the kernel at level 1 is $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D))$, the kernel at level 2 is $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}')) = \exp(-\|\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_1(\mathbf{x}')\|^2/(2H_1))$, etc.

We can perform posterior inference in this model to compute $p(\mathbf{f}_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y})$ for a set of test points \mathbf{x}_* (see Section 18.7.3.3 for the details). Figure 18.32b shows the resulting posterior predictive distribution. We see that the predictions away from the data capture two plausible modes: either the signal continues at the level $y = 0$ or at $y = 1$. (The posterior mean, shown by the solid blue line, is a poor summary of the predictive distribution in this case, since it lies between these two modes.) This is an example of non-trivial extrapolation behavior outside of the support of the data.

Figure 18.33 shows the individual functions learned at each layer (these are all maps from 1d to 1d). We see that the functions are individually smooth (since they are derived from an RBF kernel), but collectively they define non-smooth behavior.

47

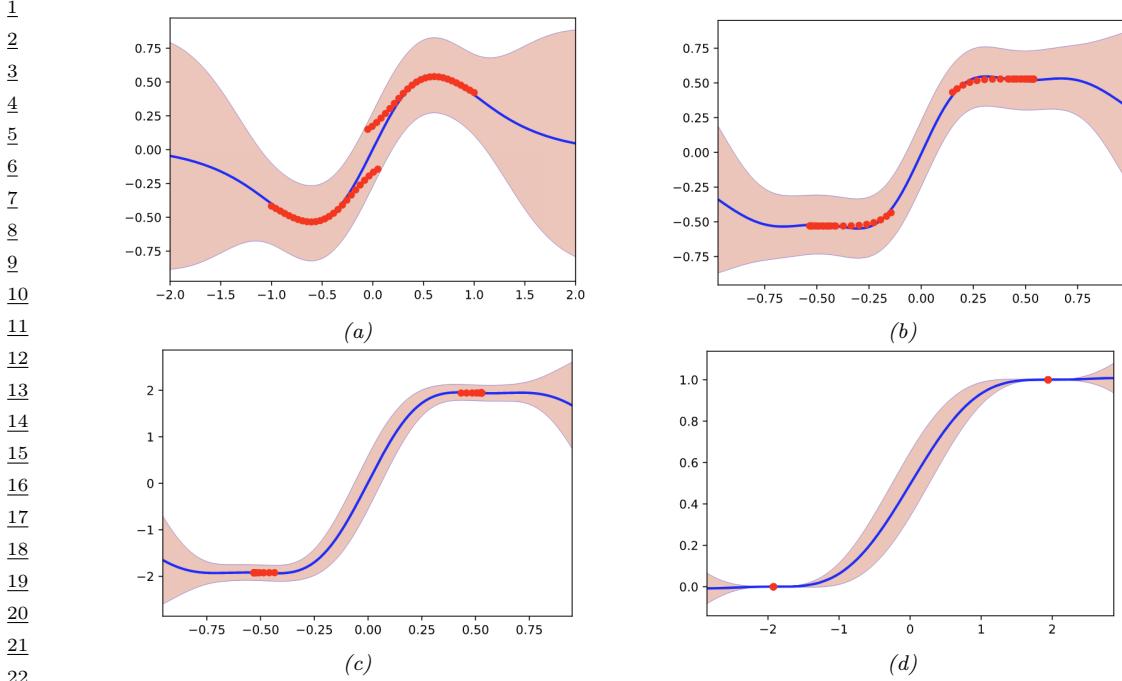


Figure 18.33: Illustration of the functions learned at each layer of the DGP. (a) Input to layer 1. (b) Layer 2 to layer 1. (c) Layer 2 to layer 3. (d) Layer 3 to output. From [Law19]. Used with kind permission of Neil Lawrence.

18.7.3.3 Posterior inference

In Equation (18.196), we defined the joint distribution defined by a (2 layer) DGP. We can condition on \mathbf{y} to convert this into a joint posterior, as follows:

$$p(f_2^*, \mathbf{f}_2, \mathbf{F}_1, f_1 | \mathbf{y}) = p(f_2^* | \mathbf{f}_2, \mathbf{f}_1^*, \mathbf{F}_1, \mathbf{y}) p(\mathbf{f}_2 | \mathbf{F}_1, \mathbf{f}_1^*, \mathbf{y}) p(\mathbf{f}_1^*, \mathbf{F}_1 | \mathbf{y}) \quad (18.197)$$

$$= p(f_2^* | \mathbf{f}_2, \mathbf{f}_1^*, \mathbf{F}_1) p(\mathbf{f}_2 | \mathbf{F}_1, \mathbf{y}) p(\mathbf{f}_1^*, \mathbf{F}_1 | \mathbf{y}) \quad (18.198)$$

where the simplifications in the second line follow from the conditional independencies encoded in Figure 18.31. Note that \mathbf{f}_2 and f_2^* depend on \mathbf{F}_1 and \mathbf{f}_1^* only through \mathbf{K}_2 , \mathbf{k}_2^* and k_2^{**} , where

$$p(\mathbf{f}_2 | \mathbf{K}_2) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_2), \quad p(f_2^* | k_2^{**}, \mathbf{k}_2^*, \mathbf{K}_2, \mathbf{f}_2) \sim \mathcal{N}((\mathbf{k}_2^*)^\top \mathbf{K}_2^{-1} \mathbf{f}_2, k_2^{**} - (\mathbf{k}_2^*)^\top \mathbf{K}_2^{-1} \mathbf{k}_2^*) \quad (18.199)$$

Hence

$$p(f_2^*, \mathbf{f}_2, \mathbf{F}_1, f_1 | \mathbf{y}) = p(f_2^* | \mathbf{f}_2, \mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}) p(\mathbf{f}_2 | \mathbf{K}_2, \mathbf{y}) p(\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**} | \mathbf{y}) \quad (18.200)$$

For prediction we only care about f_2^* , so we marginalize out the other variables. The posterior

1 mean is given by
2

$$\mathbb{E}_{f_2^*|\mathbf{y}}[f_2^*] = \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}|\mathbf{y}} [\mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [\mathbb{E}_{f_2^*|f_2, \mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}} [f_2^*]]] \quad (18.201)$$

$$= \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*, k_2^{**}|\mathbf{y}} [\mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [(k_2^*)^\top \mathbf{K}_2^{-1} f_2]] \quad (18.202)$$

$$= \mathbb{E}_{\mathbf{K}_2, \mathbf{k}_2^*|\mathbf{y}} \left[\underbrace{\mathbf{k}_2^* \mathbf{K}_2^{-1} \mathbb{E}_{f_2|\mathbf{K}_2, \mathbf{y}} [f_2]}_{\alpha} \right] \quad (18.203)$$

10 Since \mathbf{K}_2 and \mathbf{k}_2^* are deterministic transformations of $\mathbf{f}_1(\mathbf{x}^*), \mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)$, we can rewrite this
11 as
12

$$\mathbb{E}_{f_2^*|\mathbf{y}}[f_2^*] = \mathbb{E}_{\mathbf{f}_1(\mathbf{x}^*), \mathbf{f}_1(\mathbf{x}_1), \dots, \mathbf{f}_1(\mathbf{x}_N)|\mathbf{y}} \left[\sum_{i=1}^N \alpha_i \mathcal{K}_2(\mathbf{f}_1(\mathbf{x}_i), \mathbf{f}_1(\mathbf{x}^*)) \right] \quad (18.204)$$

16 We see from the above that inference in a DGP is, in general, very expensive, due to the need to
17 marginalize over a lot of variables, corresponding to all the hidden function values at each layer at
18 each data point. In [SD17], they propose an approach to approximate inference in DGPs based on
19 the sparse variational method of Section 10.1.1. The key assumption is that each layer has a set of
20 inducing points, along with corresponding inducing values, that simplifies the dependence between
21 unknown function values within each layer. However, the dependence between layers is modeled
22 exactly. In [Dut+21] they show that the posterior mean of such a sparse variational approximation
23 can be computed by performing a forwards pass through a ReLU DNN.
24

25 18.7.3.4 Behavior in the limit of infinite width 26

27 Consider the case of a DGP where the depth is 2. The posterior mean of the predicted output
28 at a test point is given by Equation (18.204). We see that this is a mixture of data-dependent
29 kernel functions, since both \mathbf{K}_2 and \mathbf{k}_2 depend on the data \mathbf{y} . This is what makes deep GPs more
30 expressive than single layer GPs, where the kernel is fixed. However, [PC21] show that, in the limit
31 $H_1 \rightarrow \infty$, the posterior over the kernels for the layer 2 features becomes independent of the data,
32 i.e., $p(\mathbf{K}_2, \mathbf{k}_2^*|\mathbf{y}) = \delta(\mathbf{K}_2 - \mathbf{K}_{\lim})\delta(\mathbf{k}_2^* - \mathbf{k}_{\lim}^*)$, where $\mathbf{K}_{\lim} = \mathbb{E}[f_2 f_2^\top]$ and $\mathbf{k}_{\lim}^* = \mathbb{E}[f_2^* f_2]$, where the
33 expectations depend on \mathbf{X} but not \mathbf{y} . Consequently the posterior predictive mean reduces to
34

$$\lim_{H_1 \rightarrow \infty} \mathbb{E}_{f_2^*|\mathbf{y}}[f_2^*] = \sum_{i=1}^N \alpha_i \mathcal{K}_{\lim}(\mathbf{x}_i, \mathbf{x}_*) \quad (18.205)$$

38 which is the same form as a single layer GP.

39 As a concrete example, consider a 2 layer DGP with an RBF kernel at each layer. Thus the kernel
40 at level 1 is $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2D))$, and the kernel at level 2 is $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}')) =$
41 $\exp(-\|\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_1(\mathbf{x}')\|^2/(2H_1))$. Let us fit this model to a noisy step function. In Figure 18.34 we
42 show the results as we increase the width of the hidden layer. When the width is 1, we see that
43 the covariance of the resulting DGP, $\mathcal{K}_2(\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1(\mathbf{x}'))$, is nonstationary. In particular, there are
44 long-range correlations near $\mathbf{x} = \pm 1$ (since the function is constant in this region), but short range
45 correlations near $\mathbf{x} = 0$ (since the function is changing rapidly in this region). However, as the width
46 increases, we lose this nonstationarity, as shown by the constant diagonals of the kernel matrix. Indeed,
47

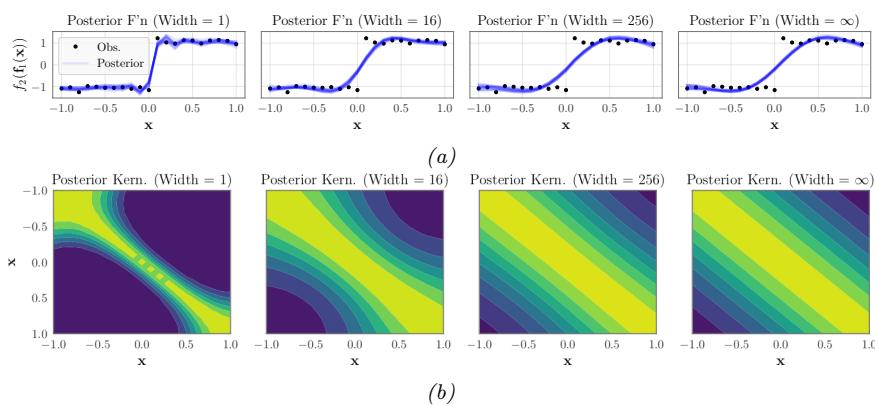


Figure 18.34: (a) Posterior of 2-layer RBF deep GP fit to a noisy step function. Columns represent width of 1, 16, 256 and infinity. (b) Average posterior covariance of the DGP, given by $\mathbb{E}_{f_1(\mathbf{x}), f_1(\mathbf{x}') \mid \mathbf{y}} [\mathcal{K}_2(f_1(\mathbf{x}), f_1(\mathbf{x}'))]$. As the width increases, the covariance becomes stationary, as shown by the kernel’s constant diagonals. From Figure 1 of [PC21]. Used with kind permission of Geoff Pleis.

in [PC21, App. G] they prove that the limiting kernel is $\mathcal{K}_{\lim}(\mathbf{x}, \mathbf{x}') = \exp(\exp(-||\mathbf{x} - \mathbf{x}'||^2/(2D)) - 1)$, which is stationary.

In [PC21], they also show that increasing the width makes the marginals more Gaussian, due to central-limit like behavior. However, increasing the depth makes the marginals less Gaussian, and causes them to have sharper peaks and heavier tails. Thus one often gets best results with a deep GP if it is deep but narrow.

18.7.3.5 Connection with Bayesian neural networks

A Bayesian neural network (BNN) is a DNN in which we place priors over the parameters (see Section 17.1). One can show (see e.g., [OA21]) that BNNs are a degenerate form of deep GPs. For example, consider a 2 layer MLP, $f_2(f_1(\mathbf{x}))$, with $f_1 : \mathbb{R}^D \rightarrow \mathbb{R}^{H_1}$ and $f_2 : \mathbb{R}^{H_1} \rightarrow \mathbb{R}$, defined by

$$f_1^{(i)}(\mathbf{x}) = (\mathbf{w}_1^{(i)})^\top \mathbf{x} + \beta \mathbf{b}_1, \quad f_2(\mathbf{z}) = \frac{1}{\sqrt{H_1}} \mathbf{w}_2^\top \varphi(\mathbf{z}) + \beta b_2 \quad (18.206)$$

where $\beta > 0$ is a scaling constant, and $\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2, b_2$ are Gaussian. The first layer is a linear regression model, and hence (from the results in Section 18.3.3) corresponds to a GP with a linear kernel of the form $\mathcal{K}_1(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$. The second layer is also a linear regression model but applied to features $\varphi(\mathbf{z})$. Hence (from the results in Section 18.3.3) this corresponds to a GP with a linear kernel of the form $\mathcal{K}_2(\mathbf{z}, \mathbf{z}') = \varphi(\mathbf{z})^\top \varphi(\mathbf{z}')$. Thus each layer of the model corresponds to a (degenerate) GP, and hence the overall model is a (degenerate) DGP. (The term “degenerate” refers to the fact that the covariance matrices only have a finite number of non-zero eigenvalues, due to the use of a finite set of basis functions.) Consequently we can use the results from Section 18.7.3.4 to conclude that infinitely wide DNNs also reduce to a single layer GP, as we already established in Section 18.7.1.

In practice we use finite-width DNNs. The width should be wide enough to approximate a standard GP at each layer, but should not be too wide, otherwise the corresponding kernels of the resulting

1 deep GP will no longer be adapted to the data, i.e., there will not be any “feature learning”. See e.g.,
2 [Ait20; PC21; ZVP21] for details.
3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

19 Structured prediction

19.1 Introduction

In **structured prediction**, we want to learn a conditional model $p(\mathbf{y}|\mathbf{x})$, where the output $\mathbf{y} \in \mathcal{Y}$ lives in some structured space, such as the set of sequences, or the set of labels of nodes on a graph. In contrast to other kinds of conditional generative model, such as text-to-image generation, in structured prediction there are often constraints on the set of valid values of the output \mathbf{y} . For example, if we want to perform sentence parsing, the output set of tags should satisfy the rules of grammar. In some cases, the “constraints” are “soft”, rather than “hard”. For example, if we want to perform pixel labeling, we usually want to encourage the label at one location to be the same as its neighbors, unless the visual input strongly suggests a change in semantic content at this location (e.g., at the edge of an object). We can capture both of these scenarios by using conditional graphical models, as we discuss in Section 19.2.

Structured prediction can also be used for temporal prediction problems, where we condition on the past and generate the future, as we discuss in Section 19.3.

19.2 Conditional random fields (CRFs)

A **conditional random field** or **CRF** [LMP01] is a version of an MRF where all the clique potentials are conditioned on input features:

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \prod_c \psi_c(\mathbf{y}_c; \mathbf{x}, \boldsymbol{\theta}) \quad (19.1)$$

(Note how the partition function now depends on the inputs \mathbf{x} as well as the parameters $\boldsymbol{\theta}$.)

If the potential functions are log-linear and have the form

$$\psi_c(\mathbf{y}_c; \mathbf{x}, \boldsymbol{\theta}) = \exp(\boldsymbol{\theta}_c^\top \boldsymbol{\phi}_c(\mathbf{x}, \mathbf{y}_c)) \quad (19.2)$$

then the result model is called a **conditional maxent model**. This can be thought of as an extension of logistic regression where we model the correlation amongst the output labels, conditioned on the input features. Of course, we can also use nonlinear potential functions, such as DNNs.

19.2.1 1d CRFs

In this section, we focus on 1d CRFs defined on chain-structured graphical models.

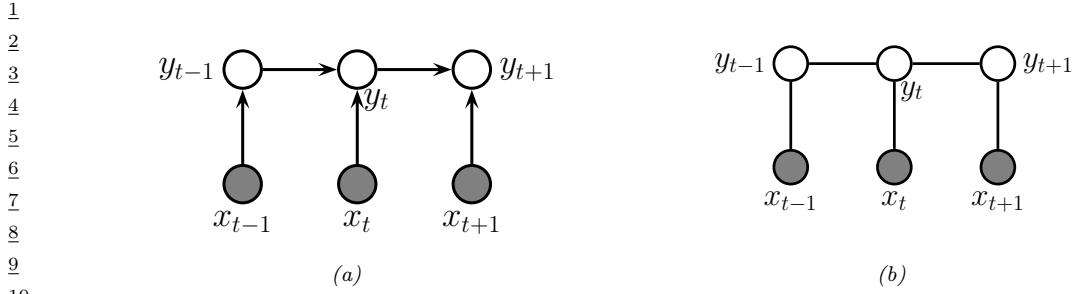


Figure 19.1: Two discriminative models for sequential data. (a) A directed model (MEMM). (b) An undirected model (CRF).

19.2.1.1 The label bias problem

We start by contrasting CRFs with **maximum entropy Markov models** (MEMMs), which is a directed Markov chain in which the state transition probabilities are conditioned on the input features, as shown in Figure 19.1(a). An MEMM seems like the natural generalization of classifiers to the structured-output setting, but it suffers from a subtle problem known (rather obscurely) as the **label bias** problem [LMP01]. The problem is that local features at time t do not influence states prior to time t . This follows by examining the DAG, which shows that \mathbf{x}_t is d-separated from \mathbf{y}_{t-1} (and all earlier time points) by the v-structure at \mathbf{y}_t , thus blocking the information flow backwards in time.

To understand what this means in practice, consider the **part of speech tagging** task, in which we must label every word in the sentence with a part of speech (POS), such as noun, verb, etc. Suppose we see the word “banks”; this could be a verb (as in “he banks at Chase”), or a noun (as in “the river banks were overflowing”). Locally the POS tag for the word is ambiguous. However, suppose that later in the sentence, we see the word “fishing”; this gives us enough context to infer that the sense of “banks” is “river banks”. However, in an MEMM the “fishing” evidence will not flow backwards, so we will not be able to infer the correct label for “banks”.

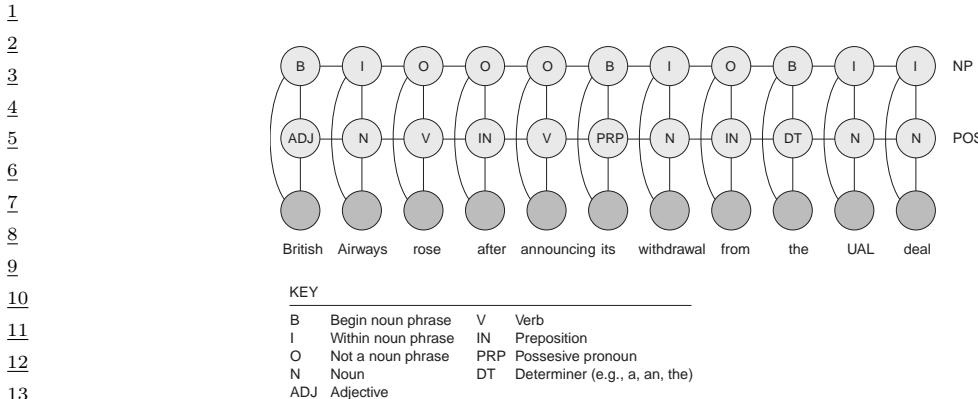
Now consider a chain-structured CRF. This model has the form

$$p(\mathbf{y}_{1:T} | \mathbf{x}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \prod_{t=1}^T \psi(y_t; \mathbf{x}, \boldsymbol{\theta}) \prod_{t=1}^{T-1} \psi(y_t, y_{t+1}; \mathbf{x}, \boldsymbol{\theta}) \quad (19.3)$$

From the graph in Figure 19.1(b) we see that the label bias problem no longer exists, since y_t does not block the information from x_t from reaching other $y_{t'}$ nodes, due to the lack of directed edges.

The label bias problem in MEMMs occurs because directed models are **locally normalized**, meaning each CPD sums to 1. By contrast, MRFs and CRFs are **globally normalized**, which means that local factors do not need to sum to 1, since the partition function Z , which sums over all joint configurations, will ensure the model defines a valid distribution.

However, this solution comes at a price: we do not get a valid probability distribution over $\mathbf{y}_{1:T}$ until we have seen the whole sentence, since only then can we normalize over all configurations. Consequently, CRFs are not as useful as PGM-D’s (whether discriminative or generative) for online or real-time inference. Furthermore, the fact that Z is a function of all the parameters makes CRFs less modular much slower to train than PGM-D’s, as we discuss in Section 19.2.3.



14 *Figure 19.2: A CRF for joint POS (part of speech) tagging and NP (noun phrase) segmentation. From Figure*
 15 *4.E.1 of [KF09a]. Used with kind permission of Daphne Koller.*

19.2.1.2 Applications to NLP

21 CRFs used to be widely used in the natural language processing (NLP) community, before the advent
 22 of RNNs and transformers. Fortunately, we can get the best of both worlds by combining CRFs
 23 with DNNs. In particular, instead of using local potentials defined in terms of linear classifiers on
 24 pre-defined featursm $\phi(y_t, \mathbf{x})$,

$$26 \quad \psi(y_t|\mathbf{x}) = \exp(\mathbf{w}^\top \phi(y_t, \mathbf{x})) \quad (19.4)$$

27 we can use DNN classifiers instead. This is called a **neural CRF** [DK15b]. The pairwise edge
 28 potentials $\psi(y_t, y_{t+1}; \mathbf{x}, \theta)$, can often be manually designed to encode prior knowledge or constraints,
 29 as we illustrate below.

19.2.1.3 Noun phrase chunking

34 One common NLP task is **noun phrase chunking**, which refers to the task of segmenting a sentence
 35 into its distinct noun phrases (NPs). This can be implemented as a seq2seq problem, in which each
 36 word is labeled with **BIO** tags (begin / inside / outside). A standard approach to this problem
 37 would first convert the string of words into a string of **POS** (part of speech tags), and then convert
 38 the POS tags to a string of BIOs. However, such a **pipeline** method can propagate errors. A more
 39 robust approach is to build a joint probabilistic model of the form $p(\text{NP}_{1:T}, \text{POS}_{1:T} | \text{words}_{1:T})$. One
 40 way to do this is to use the CRF in Figure 19.2. The connections between adjacent labels encode the
 41 probability of transitioning between the B, I and O states, and can enforce constraints such as the
 42 fact that B (begin) must preceed I (inside).

43 Inference in the model in Figure 19.2 is tractable, since it is essentially a “fat chain”, so we can
 44 use the forwards-backwards algorithm (Section 8.3.3) for exact inference in $O(T|\text{POS}|^2|\text{NP}|^2)$ time,
 45 where $|\text{POS}|$ is the number of POS tags, and $|\text{NP}|$ is the number of NP tags. However, the seemingly
 46 similar graph in Figure 19.3, to be explained below, is computationally intractable.

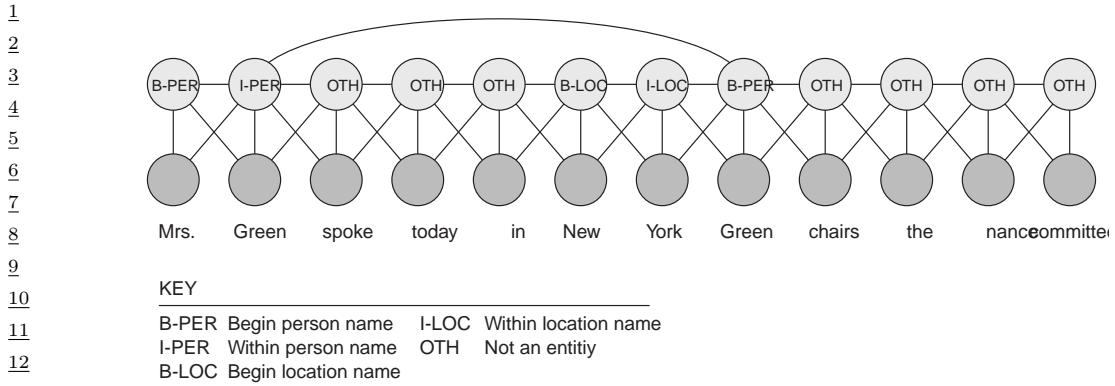


Figure 19.3: A skip-chain CRF for named entity recognition. From Figure 4.E.1 of [KF09a]. Used with kind permission of Daphne Koller.

19.2.1.4 Named entity recognition

20 A task that is related to NP chunking is **named entity extraction**. A simple approach to this is to
 21 extend the BIO notation to {B-Per, I-Per, B-Loc, I-Loc, B-Org, I-Org, Other }. We can then apply
 22 a linear CRF.

23 However, sometimes it is ambiguous whether a word is a person, location, or something else.
 24 (Proper nouns are particularly difficult to deal with because they belong to an **open class**, that is,
 25 there is an unbounded number of possible names, unlike the set of nouns and verbs, which is large
 26 but essentially fixed.) We can get better performance by considering long-range correlations between
 27 words. For example, we might add a link between all occurrences of the same word, and force the
 28 word to have the same tag in each occurrence. (The same technique can also be helpful for resolving
 29 the identity of pronouns.) This is known as a **skip-chain CRF**. See Figure 19.3 for an illustration,
 30 where we show that the word “Green” is interpreted as a person in both occurrences within the same
 31 sentence.

32 We see that the graph structure itself changes depending on the input, which is an additional
 33 advantage of CRFs over generative models. Unfortunately, inference in this model is generally
 34 more expensive than in a simple chain with local connections because of the larger treewidth (see
 35 Section 9.4.2).

19.2.1.5 Natural language parsing

39 A generalization of chain-structured models for language is to use probabilistic grammars. In
 40 particular, a probabilistic **context free grammar** or **PCFG** is a set of re-write or production rules
 41 of the form $\sigma \rightarrow \sigma' \sigma''$ or $\sigma \rightarrow x$, where $\sigma, \sigma', \sigma'' \in \Sigma$ are non-terminals (analogous to parts of speech),
 42 and $x \in \mathcal{X}$ are terminals, i.e., words. Each such rule has an associated probability. The resulting
 43 model defines a probability distribution over sequences of words. We can compute the probability of
 44 observing a particular sequence $\mathbf{x} = x_1 \dots x_T$ by summing over all trees that generate it. This can be
 45 done in $O(T^3)$ time using the **inside-outside algorithm**; see e.g., [JM08; MS99; Eis16] for details.

46 PCFGs are generative models. It is possible to make discriminative versions which encode
 47

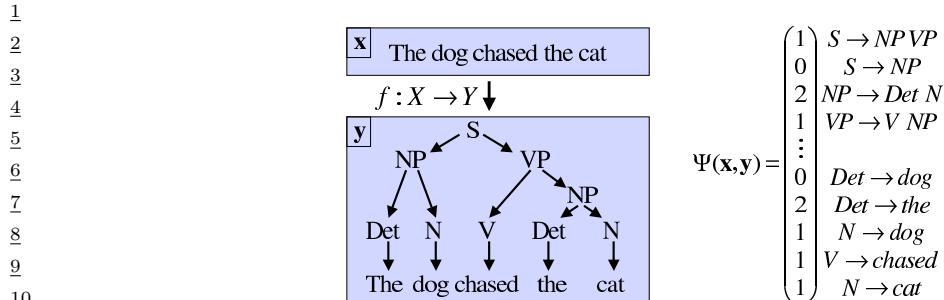


Figure 19.4: Illustration of a simple parse tree based on a context free grammar in Chomsky normal form. The feature vector $\Psi(\mathbf{x}, \mathbf{y})$ counts the number of times each production rule was used, and is used to define the energy of a particular tree structure, $E(\mathbf{y}|\mathbf{x}) = -\mathbf{w}^\top \Psi(\mathbf{x}, \mathbf{y})$. The probability distribution over trees is given by $p(\mathbf{y}|\mathbf{x}) \propto \exp(-E(\mathbf{y}|\mathbf{x}))$. From Figure 5.2 of [AHT07]. Used with kind permission of Yasemin Altun.

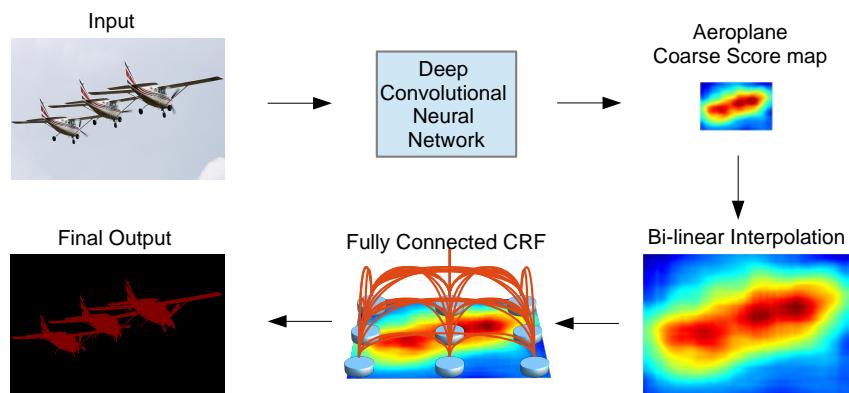


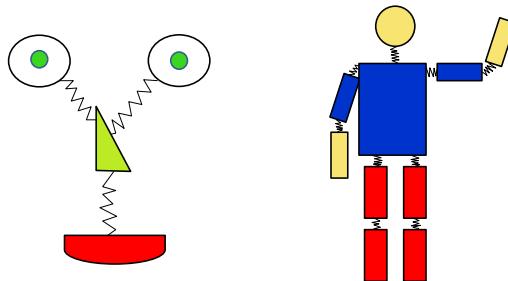
Figure 19.5: A fully connected CRF is added to the output of a CNN, in order to increase the sharpness of the segmentation boundaries. From Figure 3 of [Che+15]. Used with kind permission of Jay Chen.

the probability of a labeled tree, \mathbf{y} , given a sequence of words, \mathbf{x} , by using a CRF of the form $p(\mathbf{y}|\mathbf{x}) \propto \exp(\mathbf{w}^\top \Psi(\mathbf{x}, \mathbf{y}))$. For example, we might define $\Psi(\mathbf{x}, \mathbf{y})$ to count the number of times each production rule was used (which is analogous to the number of state transitions in a chain-structured model), as illustrated in Figure 19.4. We can also use a deep neural net to define the features, as in the **neural CRF parser** method of [DK15b].

19.2.2 2d CRFs

It is also possible to apply CRFs to image processing problems, which are usually defined on 2d grids. We give some examples below.

1
2
3
4
5
6
7
8
9
10



11 *Figure 19.6: Pictorial structures model for a face and body. Each body part corresponds to a node in the*
12 *CRF whose state space represents the location of that part. The edges (springs) represent pairwise spatial*
13 *constraints. The local evidence nodes are not shown. Adapted from a figure by Pedro Felzenszwalb.*

14

15

16 19.2.2.1 Semantic segmentation

17 The task of **semantic segmentation** is to assign a label to every pixel in an image. We can easily
18 solve this problem using a CNN with one softmax output node per pixel. However, this may fail to
19 capture long-range dependencies, since convolution is a local operation.

20 One way to get better results is to feed the output of the CNN into a CRF. Since the CNN already
21 uses convolution, its outputs will usually already be locally smooth, so the benefits from using a CRF
22 with a local grid structure may be quite small. However, we can sometimes get better results if we
23 use a **fully connected CRF**, which has connections between all the pixels. This can capture long
24 range connections which the grid-structured CRF cannot. See Figure 19.5 for an illustration, and
25 [Che+17a] for details.

26 Unfortunately, exact inference in a fully connected CRF is intractable, but in the case of Gaussian
27 potentials, it is possible to devise an efficient mean field algorithm, as described in [KK11]. Interest-
28 ingly, [Zhe+15] showed how the mean field update equations can be implemented using a recurrent
29 neural network (see Section 16.3.3), allowing end-to-end training. Alternatively, if we are willing
30 to use a finite number of iterations, we can just “unroll” the computation graph and treat it as a
31 fixed-sized feedforward circuit. The result is a graph-structured neural network, where the topology
32 of the GNN is derived from the graphical model (c.f., Section 9.3.7). The advantage of this compared
33 to standard CRF methods is that we can train this entire model end-to-end using standard gradient
34 descent methods; we no longer have to worry about the partition function (see Section 19.2.3), or
35 the lack of convergence that can arise when combining approximate inference with standard CRF
36 learning.

37

38 19.2.2.2 Deformable parts models

39

40 Consider the problem of **object detection**, i.e., finding the location(s) of an object of a given class
41 (e.g., a person or a car) in an image. One way to tackle this is to train a binary classifier that takes
42 as input an image patch and specifies if the patch contains the object or not. We can then apply this
43 to every image patch, and return the locations where the classifier has high confidence detections;
44 this is known as a **sliding window detector**, and works quite well for rigid objects such as cars
45 or frontal faces. Such an approach can be made efficient by using convolutional neural networks or
46

47

1 CNNs; see Section 16.3.2 for details.

2 However, such methods can work poorly when there is occlusion, or when the shape is deformable,
3 such as a person’s or animal’s body, because there is too much variation in the overall appearance.
4 A natural strategy to deal with such problems is break the object into parts, and then to detect
5 each part separately. But we still need to enforce spatial coherence of the parts. This can be done
6 using a pairwise CRF, where node y_i specifies the location of part i in the image (assuming it is
7 present), and where we connect adjacent parts by a potential function that encourages them to be
8 close together. For example, we can use a pairwise potential of the form $\psi(y_i, y_j) = \exp(-d(y_i, y_j))$,
9 where $y_i \in \{1, \dots, K\}$ is the location of part i (a discretization of the 2d image plane), and $d(y_i, y_j)$ is
10 the distance between parts i and j . In addition we will have a local evidence term of the form $p(y_i|\mathbf{x})$,
11 which can be any kind of discriminative classifier, such as a CNN, which predicts the distribution
12 over locations for part i given the image \mathbf{x} . The overall model has the form

$$\frac{1}{Z(\mathbf{x})} \left[\prod_i p(y_i|f(\mathbf{x})_i) \right] \left[\prod_{(i,j) \in E} \psi(y_i, y_j | \mathbf{x}) \right] \quad (19.5)$$

19 where E is the set of edges in the CRF, and $f(\mathbf{x})_i$ is the i ’th output of the CNN.

20 We can think of this CRF as a series of parts connected by springs, where the energy of the system
21 increases if the parts are moved too far from their expected relative distance. This is illustrated in
22 Figure 19.6. The resulting model is known as a **pictorial structure** [FE73], or **deformable parts**
23 **model** [Fel+10]. Furthermore, since this is a conditional model, we can make the spring strengths
24 be image dependent.

25 We can find the globally optimal joint configuration $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$ using brute force
26 enumeration in $O(K^T)$ time, where T is the number of nodes and K is the number of states (locations)
27 per node. While T is often small, (e.g., just 10 body parts in Figure 19.6), K is often very large,
28 since there are millions of possible locations in an image. By using tree-structured graphs, exact
29 inference can be done in $O(TK^2)$ time, as we explain in Section 9.2.1. Furthermore, by exploiting
30 the fact that the discrete states are ordinal, inference time can be further reduced to $O(TK)$, as
31 explained in [Fel+10].

32 Note that by “augmenting” standard deep neural network libraries with a dynamic programming
33 inference “module”, we can represent DPMs as a kind of CNN, as shown in [Gir+15]. The key
34 property is that we can backpropagate gradients through the inference algorithm.

37 19.2.3 Parameter estimation

38 In this section, we discuss how to perform maximum likelihood estimation for CRFs. This is a small
39 extension of the MRF case in Section 4.3.6.1.

42 19.2.3.1 Log-linear potentials

44 In this section we assume the log potential functions are linear in the parameters, i.e.,

$$\psi_c(\mathbf{y}_c; \mathbf{x}, \boldsymbol{\theta}) = \exp(\boldsymbol{\theta}_c^\top \boldsymbol{\phi}_c(\mathbf{x}, \mathbf{y}_c)) \quad (19.6)$$

1 Hence the log likelihood becomes
2

$$\ell(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_n \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \frac{1}{N} \sum_n \left[\sum_c \boldsymbol{\theta}_c^\top \boldsymbol{\phi}_c(\mathbf{y}_n, \mathbf{x}_n) - \log Z(\mathbf{x}_n; \boldsymbol{\theta}) \right] \quad (19.7)$$

3
4 where
5

$$Z(\mathbf{x}_n; \boldsymbol{\theta}) = \sum_{\mathbf{y}} \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{y}, \mathbf{x}_n)) \quad (19.8)$$

6 is the partition function for example n .
7

8 We know from Section 2.5.3 that the derivative of the log partition function yields the expected
9 sufficient statistics, so the gradient of the log likelihood can be written as follows:
10

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N} \sum_n \left[\boldsymbol{\phi}_c(\mathbf{y}_n, \mathbf{x}_n) - \frac{\partial}{\partial \boldsymbol{\theta}_c} \log Z(\mathbf{x}_n; \boldsymbol{\theta}) \right] \quad (19.9)$$

$$= \frac{1}{N} \sum_n [\boldsymbol{\phi}_c(\mathbf{y}_n, \mathbf{x}_n) - \mathbb{E}[\boldsymbol{\phi}_c(\mathbf{y}, \mathbf{x}_n)]] \quad (19.10)$$

11 Since the objective is convex, we can use a variety of solvers to find the MLE, such as the
12 stochastic meta descent method of [Vis+06], which is a variant of SGD where the stepsize is adapted
13 automatically.
14

15 19.2.3.2 General case

16 In the general case, a CRF can be written as follows:

$$p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta}) = \frac{\exp(f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}))}{Z(\mathbf{x}; \boldsymbol{\theta})} = \frac{\exp(f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}))}{\sum_{\mathbf{y}'} \exp(f(\mathbf{x}, \mathbf{y}'; \boldsymbol{\theta}))} \quad (19.11)$$

17 where $f(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ is a scoring (negative energy) function, where high scores correspond to probable
18 configurations. The gradient of the log likelihood is
19

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_n, \mathbf{y}_n; \boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \log Z(\mathbf{x}_n; \boldsymbol{\theta}) \quad (19.12)$$

20 Computing derivatives of the log partition function is tractable provided we can compute the
21 corresponding expectations, as we discuss in Section 4.3.6.2. Note, however, that we need to compute
22 these derivatives of for every training example, which is slower than the MRF case.
23

24 19.2.4 Other approaches

25 Many other approaches to structured prediction have been proposed, going beyond CRFs. Some of
26 these methods are discussed in [Now+14]. More recently, [BYM17] proposed **Structured Prediction**
27 **Energy Networks**, which are a form of energy based model (Chapter 25), where we predict using
28 an optimization procedure, $\hat{\mathbf{y}}(\mathbf{x}) = \operatorname{argmin} \mathcal{E}(\mathbf{x}, \mathbf{y})$. In addition, it is common to use graph neural
29 networks (Section 16.3.5) and sequence-to-sequence models such as transformers (Section 16.3.4) for
30 this task.
31

19.3 Time series forecasting

In this section, we discuss **time series forecasting**, which is the problem of computing the predictive distribution over future observations $h \geq 1$ steps into the future, given the data up until the present, i.e., computing $p(\mathbf{y}_{t+h}|\mathbf{y}_{1:t})$. (The model may optionally be conditioned on known inputs, to get $p(\mathbf{y}_{t+h}|\mathbf{y}_{1:t}, \mathbf{x}_{1:t+h})$.) There are many approaches to this problem (see e.g., [HA21]). In the sections below, we just focus on a few.

19.3.1 Structural time series models

In this section, we discuss a particular approach to time series forecasting known as the **structural time series (STS)** approach. The basic idea is to represent the observed data as a sum of C individual **components**:

$$15 \quad f(t) = f_1(t) + f_2(t) + \cdots + f_C(t) + \epsilon_t \quad (19.13)$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. For example, we might have a seasonal component that causes the observed values to oscillate up and down, and a growth component, that causes the observed values to get larger over time. Each latent process $f_c(t)$ is modeled by a linear Gaussian state-space model (**SSM**, Section 31.1), which (in this context) is also called a **dynamic linear model (DLM)**. Since these are linear, we can combine them altogether into a single LG-SSM. In particular, in the case of scalar observations, the model has the form

$$23 \quad p(\mathbf{z}_t|\mathbf{z}_{t-1}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_t|\mathbf{A}\mathbf{z}_{t-1}, \mathbf{Q}) \quad (19.14)$$

$$25 \quad p(y_t|\mathbf{z}_t, \boldsymbol{\theta}) = \mathcal{N}(y_t|\mathbf{C}\mathbf{z}_t + \boldsymbol{\beta}^\top \mathbf{x}_t, \sigma_y^2) \quad (19.15)$$

where \mathbf{A} and \mathbf{Q} are block structured matrices, with one block per component. The vector \mathbf{C} then adds up all the relevant pieces from each component to generate the overall mean. Note that the matrices \mathbf{A} and \mathbf{C} are fixed sparse matrices which can be derived from the form of the corresponding components of the model, as we discuss below. So the only model parameters are the variance terms, \mathbf{Q} and σ_y^2 , and the optional regression coefficients $\boldsymbol{\beta}$.¹

Once we have specified the model, we can infer the latent state distribution at each step, $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \boldsymbol{\theta})$, using the Kalman filter (Section 8.4.1). Given the current posterior, we can then “roll forward” in time to forecast future observations h steps ahead by computing $p(\mathbf{y}_{t+h}|\mathbf{y}_{1:t}, \boldsymbol{\theta})$, as in Section 8.4.1.6. To estimate the model parameters, $\boldsymbol{\theta} = (\mathbf{A}, \mathbf{C}, \sigma_y^2, \mathbf{Q})$, we can either use maximum likelihood estimation (see Section 31.2.5), or Bayesian inference (see Section 31.4.2). The latter approach is known as **Bayesian structural time series** or **BSTS** modeling and tends to give more reliable results (since small errors in estimating the noise variances can have a large impact on the forecast quality).

Many classical time series methods such as the **ARMA** (autoregressive moving average) method, can be represented as an STS model (see e.g., [Har90; Sim06; CK07; Fra08; DK12; Sar13; PFW21; Tri21]). However, the STS approach has much more flexibility. For example, we can replace the Gaussian likelihood with other kinds of distributions, and we can replace the linear observation and dynamics models with nonlinear versions.

¹ In the statistics community, the notation is often slightly different, and often follow [DK12]. In particular, the dynamics are written as $\boldsymbol{\alpha}_t = \mathbf{T}_t \boldsymbol{\alpha}_{t-1} + \mathbf{c}_t \mathbf{R}_t \boldsymbol{\eta}_t$ and the observations are written as $y_t = \mathbf{Z}_t \boldsymbol{\alpha}_t + \boldsymbol{\beta}^\top \mathbf{x}_t + H_t \epsilon_t$, where $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\epsilon_t \sim \mathcal{N}(0, 1)$.

Below we illustrate some of the basic STS components, and then give some examples. We focus on forecasting individual scalar times series, as opposed to vector valued series, or multiple sets of related series.

19.3.1.1 Local level model

The simplest latent dynamical process is known as the **local level model**. It assumes the observations $y_t \in \mathbb{R}$ are generated by a Gaussian with (latent) mean μ_t , which evolves over time according to a random walk:

$$y_t = \mu_t + \epsilon_{y,t} \quad \epsilon_{y,t} \sim \mathcal{N}(0, \sigma_y^2) \quad (19.16)$$

$$\mu_t = \mu_{t-1} + \epsilon_{\mu,t} \quad \epsilon_{\mu,t} \sim \mathcal{N}(0, \sigma_\mu^2) \quad (19.17)$$

Under this model, the latent mean has distribution $\mu_t \sim \mathcal{N}(0, t\sigma_\mu^2)$, so the variance grows with time. We can also use an autoregressive (AR) process, $\mu_t = \rho\mu_{t-1} + \epsilon_{\mu,t}$, where $|\rho| < 1$. This has the stationary distribution $\mu_\infty \sim \mathcal{N}(0, \frac{\sigma_\mu^2}{1-\rho^2})$, so the uncertainty grows to a finite asymptote instead of unboundedly.

19.3.1.2 Local linear model

Many time series exhibit linear trends upwards or downwards, at least locally. We can model this by letting the level μ_t change by an amount δ_{t-1} (representing the slope of the line over an interval $\Delta t = 1$) at each step

$$\mu_t = \mu_{t-1} + \delta_{t-1} + \epsilon_{\mu,t} \quad (19.18)$$

The slope itself also follows a random walk,

$$\delta_t = \delta_{t-1} + \epsilon_{\delta,t} \quad (19.19)$$

and $\epsilon_{\delta,t} \sim \mathcal{N}(0, \sigma_\delta^2)$. This is called a **local linear trend** model.

We can combine these two processes by defining the following dynamics model:

$$\underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{z_t} = \underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} \mu_{t-1} \\ \delta_{t-1} \end{pmatrix}}_{z_{t-1}} + \underbrace{\begin{pmatrix} \epsilon_{\mu,t} \\ \epsilon_{\delta,t} \end{pmatrix}}_{\epsilon_t} \quad (19.20)$$

For the emission model we have

$$y_t = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_C \underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{z_t} + \epsilon_{y,t} \quad (19.21)$$

We can also use an autoregressive model for the slope, i.e.,

$$\delta_t = D + \rho(\delta_{t-1} - D) + \epsilon_{\delta,t} \quad (19.22)$$

where D is the long run slope to which δ will revert. This is called a “**semilocal linear trend**” model, and is useful for longer term forecasts.

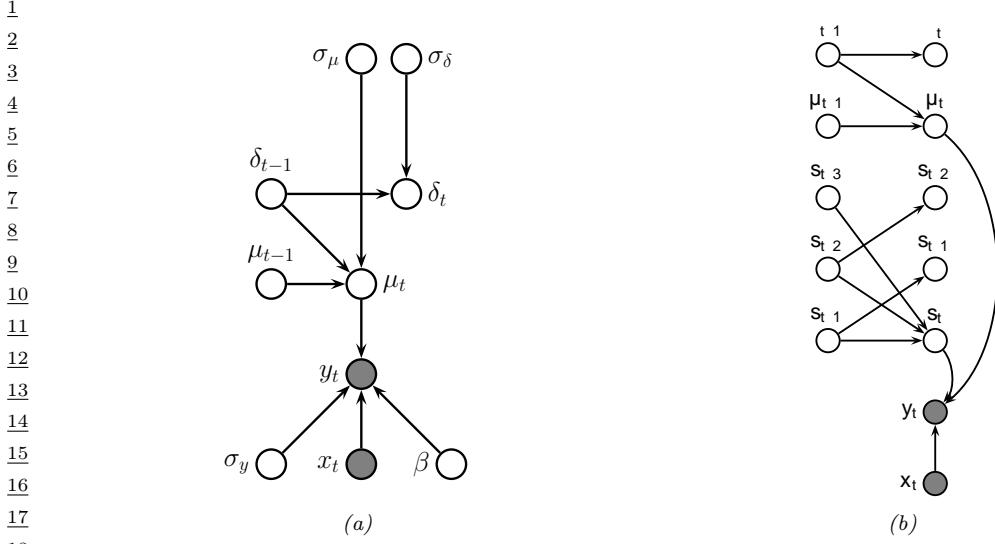


Figure 19.7: (a) A BSTS model with local linear trend and linear regression. The observed output is y_t . The latent state vector is defined by $z_t = (\mu_t, \delta_t)$. The (static) parameters are $\theta = (\sigma_y, \sigma_\mu, \sigma_\delta, \beta)$. The covariates are x_t . (b) Adding a latent seasonal process (with $S = 4$ seasons). Parameter nodes are omitted for clarity.

19.3.1.3 Adding covariates

We can also include covariates x_t into the model, to increase prediction accuracy. If we use a linear model, we have

$$y_t = \mu_t + \beta^T x_t + \epsilon_{y,t} \quad (19.23)$$

See Figure 19.7a for an illustration of the local level model with covariates. (Note that, when forecasting into the future, we will need some way to predict the input values of future x_{t+h} ; a simple approach is just to assume future inputs are the same as the present, $x_{t+h} = x_t$.)

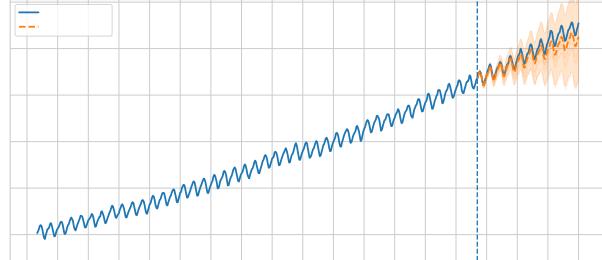
19.3.1.4 Modelling seasonality

Many time series also exhibit **seasonality**, i.e., they fluctuate periodically. This can be modeled by creating a latent process consisting of a series offset terms, s_t , which sums to zero (on average) over a complete cycle of S steps:

$$s_t = - \sum_{k=1}^{S-1} s_{t-k} + \epsilon_{s,t}, \quad \epsilon_{s,t} \sim \mathcal{N}(0, \sigma_s^2) \quad (19.24)$$

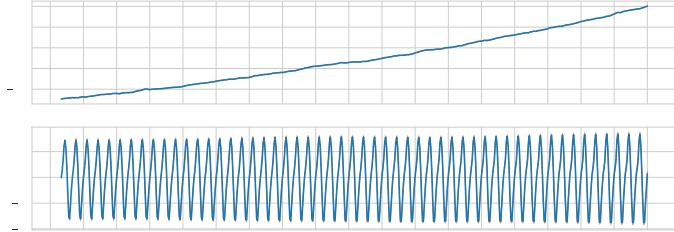
For example, for $S = 4$, we have $s_t = -(s_{t-1} + s_{t-2} + s_{t-3})$. We can convert this to a first-order model by stacking the last $S - 1$ seasons into the state vector, as shown in Figure 19.7b.

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 19.8: CO₂ levels from Mauna Loa. In orange plot we show predictions for the most recent 10 years.*
15 *Generated by [STS_TFP.ipynb](#).*

16
17
18
19
20
21
22
23
24
25
26



27 *Figure 19.9: Underlying components for the STS mode which was fit to Figure 19.8. Generated by*
28 *[STS_TFP.ipynb](#).*

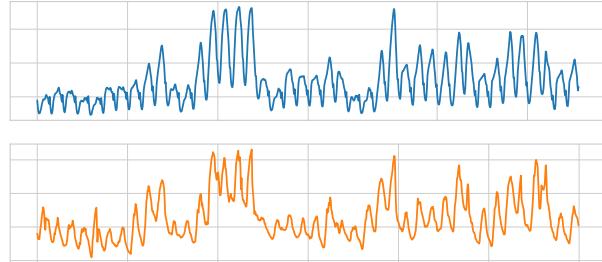
30
31
32
33
34

35 19.3.1.5 Adding it all up

36 We can combine the various latent processes (local level, linear trend, and seasonal cycles) into a
37 single linear-Gaussian SSM, because the sparse graph structure can be encoded by sparse matrices.
38 More precisely, the transition model becomes
39

$$40 \quad \underbrace{\begin{pmatrix} s_t \\ s_{t-1} \\ s_{t-2} \\ \mu_t \\ \delta_t \end{pmatrix}}_{z_t} = \underbrace{\begin{pmatrix} -1 & -1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} s_{t-1} \\ s_{t-2} \\ s_{t-3} \\ \mu_{t-1} \\ \delta_{t-1} \end{pmatrix}}_{z_{t-1}} + \mathcal{N}(\mathbf{0}, \text{diag}([\sigma_s^2, 0, 0, \sigma_\mu^2, \sigma_\delta^2])) \quad (19.25)$$

1
2
3
4
5
6
7
8
9
10
11
12
13
14



15 *Figure 19.10: Hourly temperature and electricity demand in Victoria, Australia in 2014. Generated by
16 STS_TFP.ipynb.*

19.3.1.6 Example: modeling CO₂ levels from Mauna Loa

20 In this section, we fit an STS model to the monthly atmospheric CO₂ readings from the Mauna
21 Loa observatory in Hawaii.² The data is from January 1966 to February 2019. We combine a local
22 linear trend model with a seasonal model, where we assume the periodicity is $S = 12$, since the data
23 is monthly (see Figure 19.8). We fit the model to all the data except for the last 10 years using
24 variational Bayes. The resulting posterior mean and standard deviations for the parameters are
25 $\sigma_y = 0.169 \pm 0.008$, $\sigma_\mu = 0.159 \pm 0.017$, $\sigma_\delta = 0.009 \pm 0.003$, $\sigma_s = 0.038 \pm 0.008$. We can sample 10
26 parameter vectors from the posterior and then plug them it to create a distribution over forecasts.
27 The results are shown in orange in Figure 19.8. Finally, in Figure 19.9, we plot the posterior mean
28 values of the two latent components (linear trend and current seasonal value) over time. We see how
29 the model has successfully decomposed the observed signal into a sum of two simpler signals. (See
30 also Section 19.3.3.1 where we model this data using a GP.)

32 19.3.1.7 Example: forecasting electricity demand

34 In this section, we consider a more complex example: forecasting electricity demand in Victoria,
35 Australia, as a function of the previous value and the external temperature. (Remember that January
36 is summer in Australia!) The hourly data from the first six weeks of 2014 is shown in Figure 19.10.³

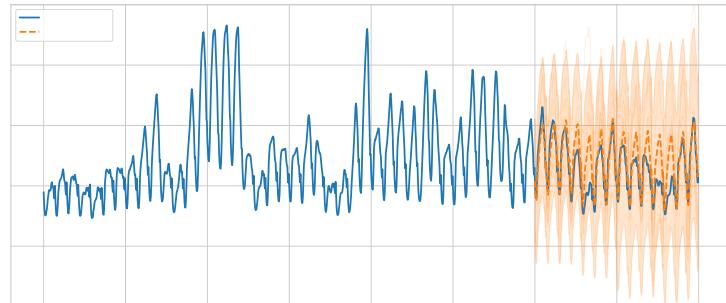
37 We fit an STS to this using 4 components: a seasonal hourly effect (period 24), a seasonal daily effect
38 (period 7, with 24 steps per season), a linear regression on the temperature, and an autoregressive
39 term on the observations themselves. We fit the model with variational inference. (This takes about
40 a minute on a GPU.) We then draw 10 posterior samples and show the posterior predictive forecasts
41 in Figure 19.11. We see that the results are reasonable, but there is also considerable uncertainty.

42 We plot the individual components in Figure 19.12. Note that they have different vertical scales,
43 reflecting their relative importance. We see that the regression on the external temperature is the

45 2. For details, see <https://blog.tensorflow.org/2019/03/structural-time-series-modeling-in.html>.

46 3. The data is from <https://github.com/robjhyndman/fpp2-package>.

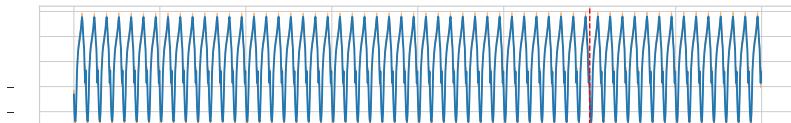
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



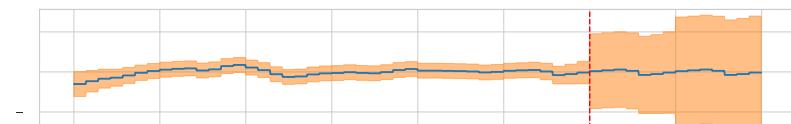
16
17

Figure 19.11: Electricity forecasts using an STS. Generated by [STS_TFP.ipynb](#).

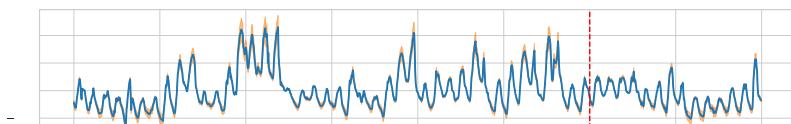
18
19
20
21
22



23
24
25
26



27



28



29

-

30

-

31

-

Figure 19.12: Components of the electricity forecasts. Generated by [STS_TFP.ipynb](#).

32

-

33

-

34

-

35

-

36

-

Figure 19.12: Components of the electricity forecasts. Generated by [STS_TFP.ipynb](#).

44

45

46

47

Draft of “Probabilistic Machine Learning: Advanced Topics” by Kevin Murphy. February 28, 2022

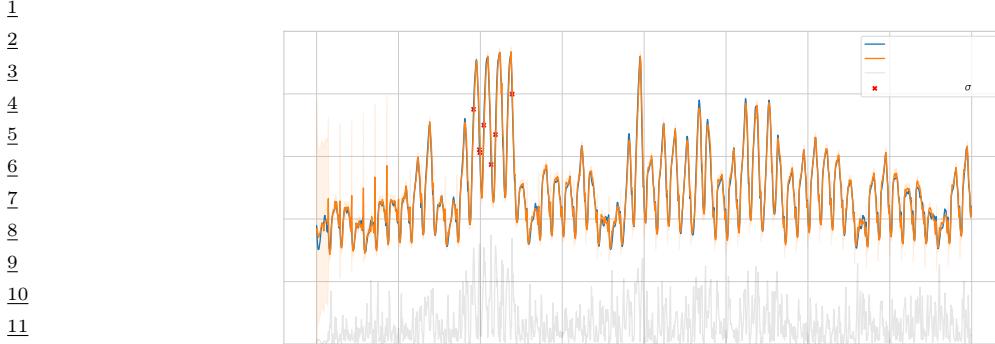


Figure 19.13: We plot the observed electricity data in blue and the predictions in orange. In gray, we plot the z-score at time t , given by $(y_t - \mu_t)/\sigma_t$, where $p(y_t|y_{1:t-1}, \mathbf{x}_{1:t}) = \mathcal{N}(\mu_t, \sigma_t^2)$. Anomalous observations are defined as points where $z_t > 3$ and are marked with red crosses. Generated by [STS_TFP.ipynb](#).

most important effect. However, the hour of day effect is also quite significant, even after accounting for external temperature. The autoregressive effect is the most uncertain one, since it is responsible for modeling all of the residual variation in the data beyond what is accounted for by the observation noise.

We can also use the model for **anomaly detection**. To do this, we compute the one-step-ahead predictive distributions, $p(y_t|y_{1:t-1}, \mathbf{x}_{1:t})$, for each timestep t , and then flag all timesteps where the observation is improbable. The results are shown in Figure 19.13.

19.3.2 Prophet

Prophet [TL18a] is a popular time series forecasting library from Facebook. It fits a generalized additive model of the form

$$y(t) = g(t) + s(t) + h(t) + \mathbf{w}^\top \mathbf{x}(t) + \epsilon_t \quad (19.26)$$

where $g(t)$ is a trend function, $s(t)$ is a seasonal fluctuation (modeled using linear regression applied to a sinusoidal basis set), $h(t)$ is an optional set of sparse “holiday effects”, $\mathbf{x}(t)$ are an optional set of (possibly lagged) covariates, \mathbf{w} are the regression coefficients, and $\epsilon(t)$ is the residual noise term, assumed to be iid Gaussian.

Prophet is a regression model, not an auto-regressive model, since it predicts the time series $\mathbf{y}_{1:T}$ given the time stamp t and the covariates $\mathbf{x}_{1:T}$, but without conditioning on past observations of y . To model the dependence on time, the trend function is assumed to be a piecewise linear trend with S changepoints, uniformly spaced in time. (See Section 30.5.2 for a discussion of changepoint detection.) That is, the model has the form

$$g(t) = (k + \mathbf{a}(t)^\top \boldsymbol{\delta})t + (m + \mathbf{a}(t)^\top \boldsymbol{\gamma}) \quad (19.27)$$

where k is the growth rate, m is the offset, $a_j(t) = \mathbb{I}(t \geq s_j)$, where s_j is the time of the j 'th changepoint, $\delta_t \sim \text{Laplace}(\tau)$ is the magnitude of the change, and $\gamma_j = -s_j \delta_j$ to make the function

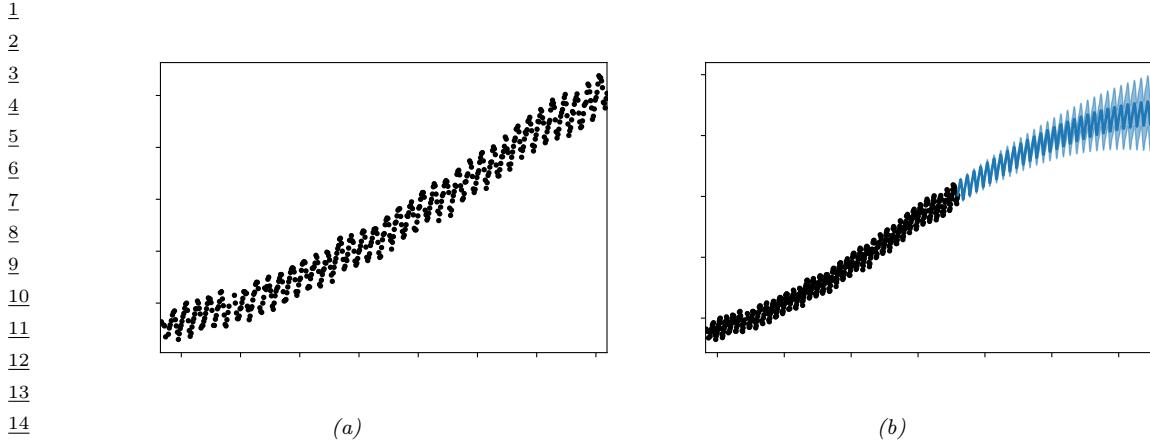


Figure 19.14: (a) The observed Mauna Loa CO₂ time series. (b) Forecasts from a GP. Generated by [gp_mauna_loa.ipynb](#).

continuous. The Laplace prior on δ ensures the MAP parameter estimate is sparse, so the difference across change point boundaries is usually 0.

For an interactive visualization of how prophet works, see <http://prophet.mbrouns.com/>.

19.3.3 Gaussian processes for timeseries forecasting

It is possible to use Gaussian processes (Chapter 18) to perform timeseries forecasting (see e.g., [Rob+13]). The basic idea is to model the unknown output as a function of time, $f(t)$, and to represent a prior about the form of f as a GP; we then update this prior given the observed evidence, and forecast into the future. Naively this would take $O(T^3)$ time. However, for certain stationary kernels, it is possible to reformulate the problem as a linear-Gaussian state space model, and then use the Kalman smoother to perform inference in $O(T)$ time, as explained in [SSH13; SS19; Ada+20]. This conversion can be done exactly for Matern kernels and approximately for Gaussian (RBF) kernels (see [SS19, Ch. 12]). In [SGF21], they describe how to reduce the linear dependence on T to $\log(T)$ time using a **parallel prefix scan** operator, that can be run efficiently on GPUs.

19.3.3.1 Example: Mauna Loa revisited

In this section, we revisit the Mauna Loa CO₂ dataset from Section 19.3.1.6. We show the raw data in Figure 19.14(a). We see that there is periodic (or quasi-periodic) signal with a year-long period superimposed on a long term trend. Following [RW06, Sec 5.4.3], we will model this with a composition of kernels:

$$k(r) = k_1(r) + k_2(r) + k_3(r) + k_4(r) \quad (19.28)$$

where $k(t, t') = k(t - t')$, and

To capture the long term smooth rising trend, we let k_1 be a squared exponential (SE) kernel,

1 where θ_0 is the amplitude and θ_1 is the length scale:
 2

$$3 \quad k_1(r) = \theta_0^2 \exp\left(-\frac{r^2}{2\theta_1^2}\right) \quad (19.29)$$

4

5 To model the periodicity, we can use periodic or exp-sine-squared kernel from Equation (18.17)
 6 with a period of 1 year. However, since it is not clear if the seasonal trends is exactly periodic, we
 7 multiply this periodic kernel with another SE kernel to allow for a decay away from periodicity; the
 8 result is k_2 , where θ_2 is the magnitude, θ_3 is the decay time for the periodic component, $\theta_4 = 1$ is
 9 the period, and θ_5 is the smoothness of the periodic component.
 10

$$11 \quad k_2(r) = \theta_2^2 \exp\left(-\frac{r^2}{2\theta_3^2} - \theta_5 \sin^2\left(\frac{\pi r}{\theta_4}\right)\right) \quad (19.30)$$

12

13 To model the (small) medium term irregularities, we use a rational quadratic kernel (Equation
 14 (18.19)):
 15

$$16 \quad k_3(r) = \theta_6^2 \left[1 + \frac{r^2}{2\theta_7^2\theta_8}\right]^{-\theta_8} \quad (19.31)$$

17

18 where θ_6 is the magnitude, θ_7 is the typical length scale, and θ_8 is the shape parameter.
 19

20 The magnitude of the independent noise can be incorporated into the observation noise of the
 21 likelihood function. For the correlated noise, we use another SE kernel:
 22

$$23 \quad k_4(r) = \theta_9^2 \exp\left(-\frac{r^2}{2\theta_{10}^2}\right) \quad (19.32)$$

24

25 where θ_9 is the magnitude of the correlated noise, and θ_{10} is is length scale. (Note that the combination
 26 of k_1 and k_4 is non-identifiable, but this does not affect predictions.)
 27

28 We can fit this model by optimizing the marginal likelihood wrt $\boldsymbol{\theta}$ (see Section 18.6.1). The
 29 resulting forecast is shown in Figure 19.14(b).
 30

31 19.3.4 Neural forecasting methods

32 Classical time series methods work well when there is little data (e.g., short sequences, or few
 33 covariates). However, in some cases, we have a lot of data. For example, we might have a single, but
 34 very long sequence, such as in anomaly detection from real-time sensors [Ahm+17]. Or we may have
 35 multiple, related sequences, such as sales of related products [Sal+19b]. In both cases, larger data
 36 means we can afford to fit more complex parametric models. Neural networks are a natural choice,
 37 because of their flexibility. Until recently, their performance in forecasting tasks was not competitive
 38 with classical methods, but this has recently started to change, as described in [Ben+20; LZ20].
 39

40 A common benchmark in the univariate time series forecasting literature is the **M4 forecasting**
 41 **competition** [MSA18], which requires participants to make forecasts on many different kinds of
 42 (univariate) time series (without covariates). This was recently won by a neural method [Smy20].
 43 More precisely, the winner of the 2019 M4 competition was a *hybrid* RNN-classical method called
 44 **ES-RNN** [Smy20]. The exponential smoothing (ES) part allows data-efficient adaptation to the
 45 observed past of the current time series; the recurrent neural network (RNN) part allows for learning
 46

¹ of nonlinear components from multiple related timeseries. (This is known as a **local+global** model,
² since the ES part is “trained” just on the local timeseries, whereas the RNN is trained on a global
³ dataset of related time series.)
⁴

⁵ In [Ran+18] they adopt a different approach for combining RNNs and classical methods, called
⁶ **DeepSSM**. In particular, they train a single RNN to predict the parameters of a state-space model
⁷ (see Section 31.1). In more detail, let $\mathbf{x}_{1:T}^n$ represent the n ’th time series, and let $\boldsymbol{\theta}_t^n$ represent the
⁸ non-stationary parameters of a linear-trend SSM model (see Section 19.3.1). We train an RNN to
⁹ compute $\boldsymbol{\theta}_t^n = f(\mathbf{c}_{1:T}^n; \boldsymbol{\phi})$, where $\boldsymbol{\phi}$ are the RNN parameters shared across all sequences. We can use
¹⁰ the predicted parameters to compute the log likelihood of the sequence, $L_n = \log p(\mathbf{x}_{1:T}^n | \mathbf{c}_{1:T}^n, \boldsymbol{\theta}_{1:T}^n)$,
¹¹ using the Kalman filter. These two modules can be combined to allow for end-to-end training of $\boldsymbol{\phi}$
¹² to maximize $\sum_{n=1}^N L_n$.

¹³ In [Wan+19c], they propose a different hybrid model known as **Deep Factors**. The idea is to
¹⁴ represent each time series (or its latent function, for non-Gaussian data) as a weighted sum of a
¹⁵ global time series, coming from a neural model, and a stochastic local model, such as an SSM or GP.
¹⁶ The **DeepGLO** (global-local) approach of [SYD19] proposes a related hybrid method, where the
¹⁷ global model uses matrix factorization to learn shared factors. This is then combined with temporal
¹⁸ convolutional networks.

¹⁹ It is also possible to train a purely neural model, without resorting to classical methods. For
²⁰ example, the **N-BEATS** model of [Ore+20] trains a residual network to predict the weights of a set
²¹ of basis functions, corresponding to a polynomial trend and a periodic signal. The weights for the
²² basis functions are predicted for each window of input using the neural network. Another approach
²³ is the **DeepAR** model of [Sal+19b], which fits a single RNN to a large number of time series. The
²⁴ original paper used integer (count) time series, modeled with a negative binomial likelihood function.
²⁵ This is a unimodal distribution, which may not be suitable for all tasks. More flexible forms, such as
²⁶ mixtures of Gaussians, have also been proposed [Muk+18]. A popular alternative is to use **quantile**
²⁷ **regression** [Koe05], in which the model is trained to predict quantiles of the distribution. For
²⁸ example, [Gas+19] proposed **SQF-RNN**, which uses splines to represent the quantile function.
²⁹ They used **CRPS** or **continuous-ranked probability score** as the loss function. This is a proper
³⁰ scoring rule, but is less sensitive to outliers, and is more “distance aware”, than log loss.

³¹ The above methods all predict a single output (per time step). If there are multiple simultaneous
³² observations, it is best to try to model their interdependencies. In [Sal+19a], they use a (low-rank)
³³ **Gaussian copula** for this, and in [Tou+19], they use a **nonparametric copula**.

³⁴ In [Wen+17], they simultaneously predict quantiles for multiple steps ahead using dilated causal
³⁵ convolution (or an RNN). They call their method **MQ-CNN**. In [WT19], they extend this to predict
³⁶ the full quantile function, taking as input the desired quantile level α , rather than prespecifying a
³⁷ fixed set of levels. They also use a copula to learn the dependencies between multiple univariate
³⁸ marginals.

³⁹

⁴⁰ 19.3.5 Causal impact of a time series intervention

⁴¹

⁴² In this section, we discuss how to perform **counterfactual reasoning** about the effect on an
⁴³ intervention given some observational (non experimental) time series data. (We discuss counterfactuals
⁴⁴ in more detail in Section 4.6.5.) For example, suppose y_t is the click through rate (CTR) of the web
⁴⁵ page of some company at time t . The company launches an ad campaign at time n , and observes
⁴⁶ outcomes $\mathbf{y}_{1:n}$ before the intervention and $\mathbf{y}_{n+1:m}$ after the intervention. A natural question to ask
⁴⁷

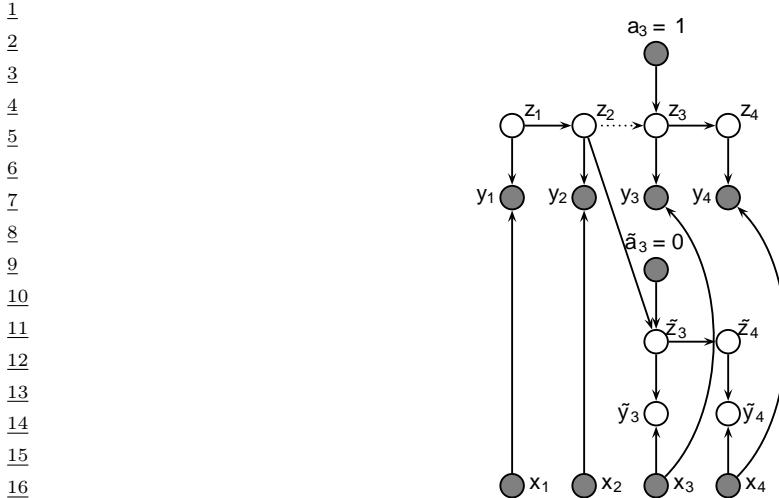


Figure 19.15: Twin network state space model for estimating causal impact of an intervention that occurs just after time step $n = 2$. We have $m = 4$ actual observations, denoted in top row. We cut the incoming arcs to z_3 since the causal mechanisms and distributions in the world after the intervention may have changed. However, in the counterfactual world, shown at the bottom of the figure (with tilde symbols), we assume the distributions are the same as in the past.

is: what would the CTR have been had the company not run the ad campaign?

To answer this question, we will use a structural time series (STS) model (see Section 19.3.1 for details). An STS model is a linear-Gaussian state-space model, where arrows have a natural causal interpretation in terms of the **arrow of time**; thus a STS is a kind of structural equation model, and hence a structural causal model (see Section 4.6). The use of an SCM allows us to infer the latent state of the noise variables given the observed data; we can then “roll back time” to the point of intervention, where we explore an alternative “fork in the road” from the one we actually took by “rolling forward in time” in a new version of the model, using the twin network approach to counterfactual inference (see Section 4.6.5). This approach is known as “**causal impact**” [Bro+15], and was developed by econometricians at Google.

19.3.5.1 Basic idea

To explain the idea in more detail, consider the twin network in Figure 19.15. The intervention occurs after time $n = 2$, and there are $m = 4$ observations in total. We observe 2 data points before the intervention, $\mathbf{y}_{1:2}$, and 2 data points afterwards, $\mathbf{y}_{3:4}$. We assume observations are generated by latent states $\mathbf{z}_{1:4}$, which evolve over time. The states are subject to exogenous noise terms, which can represent any set of unmodeled factors, such as the state of the economy. In addition, we have exogenous covariates, $\mathbf{x}_{1:m}$.

To predict what would have happened if we had not performed the intervention, (an event denoted by $\tilde{a} = 0$), we replicate the part of the model that occurs after the intervention, and use it to make forecasts. The goal is to compute the counterfactual distribution, $p(\tilde{\mathbf{y}}_{n+1:m} | \mathbf{y}_{1:n}, \mathbf{x}_{1:m})$, where $\tilde{\mathbf{y}}_t$

¹ represents counterfactual outcomes if the action had been $\tilde{a} = 0$. We can compute this counterfactual
² distribution as follows:

$$\begin{aligned} \text{⁴} p(\tilde{\mathbf{y}}_{n+1:m} | \mathbf{y}_{1:n}, \mathbf{x}_{1:m}) &= \int p(\tilde{\mathbf{y}}_{n+1:m} | \tilde{\mathbf{z}}_{n+1:m}, \mathbf{x}_{n+1:m}, \boldsymbol{\theta}) p(\tilde{\mathbf{z}}_{n+1:m} | \mathbf{z}_n, \boldsymbol{\theta}) \times \\ \text{⁵} \quad p(\mathbf{z}_n, \boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\boldsymbol{\theta} dz_n d\tilde{\mathbf{z}}_{n+1:m} \end{aligned} \quad (19.33)$$

$$\text{⁶} \quad p(\mathbf{z}_n, \boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\boldsymbol{\theta} dz_n d\tilde{\mathbf{z}}_{n+1:m} \quad (19.34)$$

⁷ where

$$\text{¹⁰} \quad p(\mathbf{z}_n, \boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) = p(\mathbf{z}_n | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}) \quad (19.35)$$

¹¹ For linear Gaussian SSMs, the term $p(\mathbf{z}_n | \mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \boldsymbol{\theta})$ can be computed using Kalman smoothing
¹² (Section 8.4.1), and the term $p(\boldsymbol{\theta} | \mathbf{y}_{1:n}, \mathbf{x}_{1:n})$, can be computed using MCMC or variational inference.

¹³ We can use samples from the above posterior predictive distribution to compute a Monte Carlo
¹⁴ approximation to the distribution of the **treatment effect** per time step, $\tau_t^i = y_t - \tilde{y}_t^i$, where the
¹⁵ i index refers to posterior samples. We can also approximate the distribution of the cumulative
¹⁶ causal impact using $\sigma_t^i = \sum_{s=n+1}^t \tau_s^i$. (There will be uncertainty in these quantities arising both from
¹⁷ epistemic uncertainty, about the true parameters controlling the model, and aleatoric uncertainty,
¹⁸ due to system and observation noise.)

¹⁹ The validity of the method is based on 3 assumptions: (1) Predictability: we assume that the
²⁰ outcome can be adequately predicted by our model given the data at hand. (We can check this by
²¹ using **backcasting**, in which we make predictions on part of the historical data.) (2) Unaffectedness:
²² we assume that the intervention does not change future covariates $\mathbf{x}_{n+1:m}$. (We can potentially check
²³ this by running the method with each of the covariates as an outcome variable.) (3) Stability: we
²⁴ assume that, had the intervention not taken place, the model for the outcome in the pre-treatment
²⁵ period would have continued in the post-treatment period. (We can check this by seeing if we predict
²⁶ an effect if the treatment is shifted earlier in time.)

²⁷

²⁹ 19.3.5.2 Example: local level model

³⁰ As a concrete example, let us assume we have a local level model and we use linear regression to
³¹ model the dependence on the covariates, as in Section 19.3.1.3. That is,

$$\text{³³} \quad y_t = \mu_t + \boldsymbol{\beta}^\top \mathbf{x}_t + \mathcal{N}(0, \sigma_y^2) \quad (19.36)$$

$$\text{³⁴} \quad \mu_t = \mu_{t-1} + \delta_{t-1} + \mathcal{N}(0, \sigma_\mu^2) \quad (19.37)$$

$$\text{³⁵} \quad \delta_t = \delta_{t-1} + \mathcal{N}(0, \sigma_\delta^2) \quad (19.38)$$

³⁶

³⁷ See the graphical model in Figure 19.16. The static parameters of the model are $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma_y^2, \sigma_\mu^2, \sigma_\delta^2)$,
³⁸ the other terms are state or observation variables. (Note that we are free to use any kind of STS
³⁹ model the local level model is just a simple default.)

⁴⁰ The use of a linear combination of other “**donor**” time series is similar in spirit to the concept of a
⁴¹ “**synthetic control**” [Aba; Shi+21]. However we do not restrict ourselves to a convex combination of
⁴² donors. Furthermore, when we have many covariates, we can use a spike-and-slab prior (Section 15.2.4)
⁴³ or horseshoe prior (Section 15.2.6) to select the relevant ones.

⁴⁴ We now give a simple example using synthetic data. We create 3 random sequences of covariates,
⁴⁵ $\mathbf{x}_t \in \mathbb{R}^3$, and define the observed output to be given by Equation (19.36), with regression weights
⁴⁶

⁴⁷

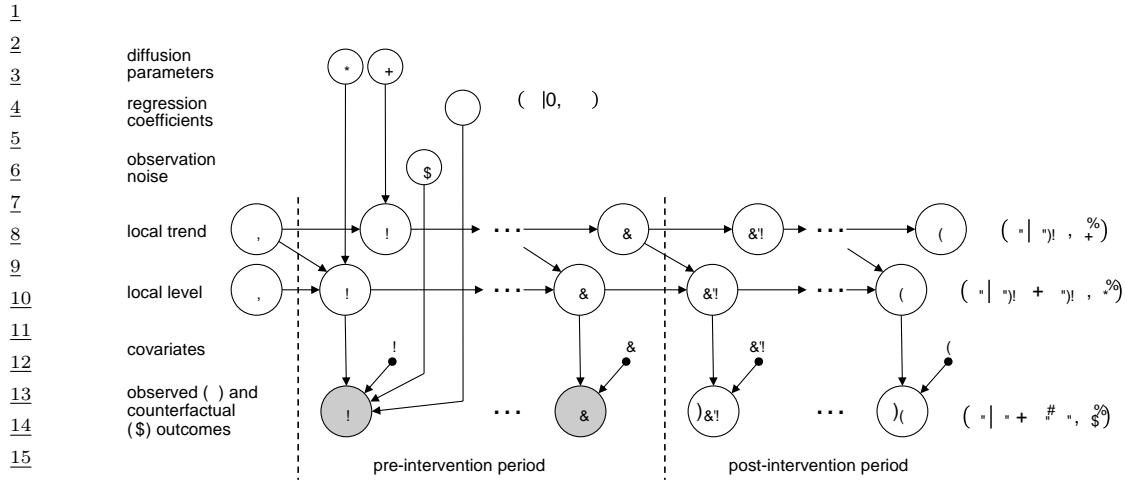
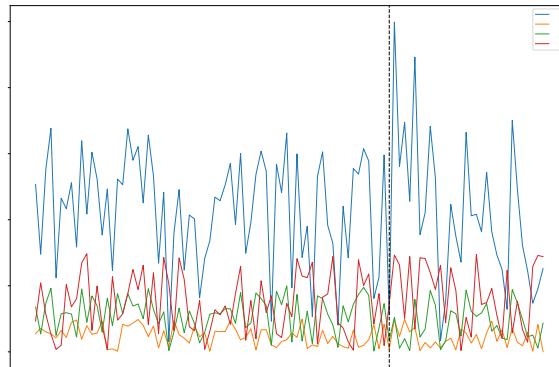


Figure 19.16: A graphical model representation of the local level causal impact model. The dotted line represents the time n at which an intervention occurs. Adapted from Figure 2 of [Bro+15]. Used with kind permission of Kay Brodersen.

$\beta = (2, 3, 0)$. At time step $n = 70$, we perform an artificial intervention by adding Δ_t to y_t , where Δ_t starts off at 5 and drops down to 0 over a period of 10 steps. This simulates the kind of transient lift one often sees when performing marketing campaigns. The data is shown in Figure 19.17(a). We see that there seems to be a small increase in the blue curve y at around time step 70, but it is hard to tell because of the noise.

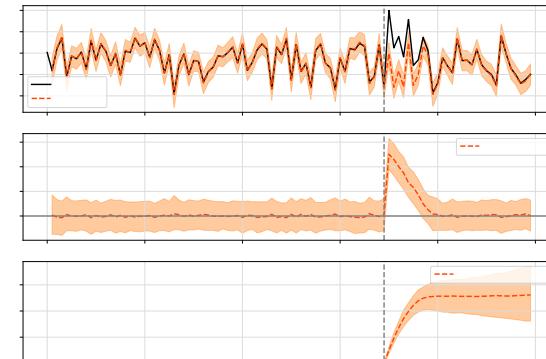
We fit a local level STS model using variational inference, and then make the counterfactual forecast shown in the top row of Figure 19.17(b). Now we see more clearly that, had the process continued without the intervention, the counterfactual outcome would have been smaller. We can therefore estimate the instantaneous and cumulative causal impact, shown in the middle and bottom rows of Figure 19.17(b). Furthermore, we find that the posterior mean estimate for the regression coefficient is $\bar{\beta} = (1.4927632, 1.945029, -0.10319362)$. We see that the third input variable is mostly ignored, as desired.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22



(a)

23
24
25
26
27
28
29
30
31
32
33
34
35
36
37



(b)

38
39 *Figure 19.17: (a) Some simulated time series data which we use to estimate the causal impact of some*
40 *intervention, which occurs at time $n = 70$, illustrated by the dotted line. The blue curve are the observed*
41 *outcomes y , the other curves are covariates (inputs). (b) Output of causal inference. Top row: observed vs*
42 *predicted outcomes. Middle row: Estimate of causal effect τ_t at each time step. Bottom row: Cumulative*
43 *causal effect, σ_t , up to each time step. Generated by [causal_impact_tfp.ipynb](#).*

44
45
46
47

20 Beyond the iid assumption

20.1 Introduction

The standard approach to supervised ML assumes the training and test sets both contain iid samples from the same distribution. However, there are many settings in which the test distribution may be different from the training distribution; this is known as **distribution shift**, as we discuss in Section 20.2. There are various techniques we can use to modify the training of the model to make it more robust to distribution shift, as we discuss in Section 20.3. Alternatively, we can wait until runtime, and hope that we can detect and/or adapt to the shifts as they occur, as we discuss in Section 20.4.

Another approach to handling distribution shift is to train using multiple related distributions; we discuss this in Section 20.5. We can also consider cases in which there is a single training distribution, but it changes over time; we discuss this in Section 20.7. Finally, in Section 20.8, we discuss settings in which the test distribution is chosen by an adversary to minimize performance of a pre-trained prediction system.

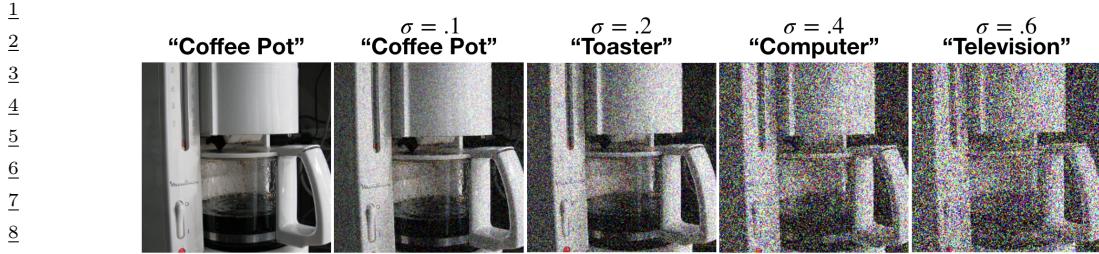
20.2 Distribution shift

If the test distribution encountered “at run time” is different from the training distribution, we say that there has been a **distribution shift** or **dataset shift** [QC+08]. More formally, let $p_{\text{tr}}(\mathbf{x}, \mathbf{y})$ represent the training or source distribution, and $p_{\text{te}}(\mathbf{x}, \mathbf{y})$ represent the testing or target distribution. Distribution shift refers to the case where $p_{\text{te}} \neq p_{\text{tr}}$. Distributions can differ in many ways, and the nature and degree of shift will have profound impact on the performance of the model on the test distribution (i.e., its **out-of-distribution generalization**, or its **external validity**), as we discuss below.

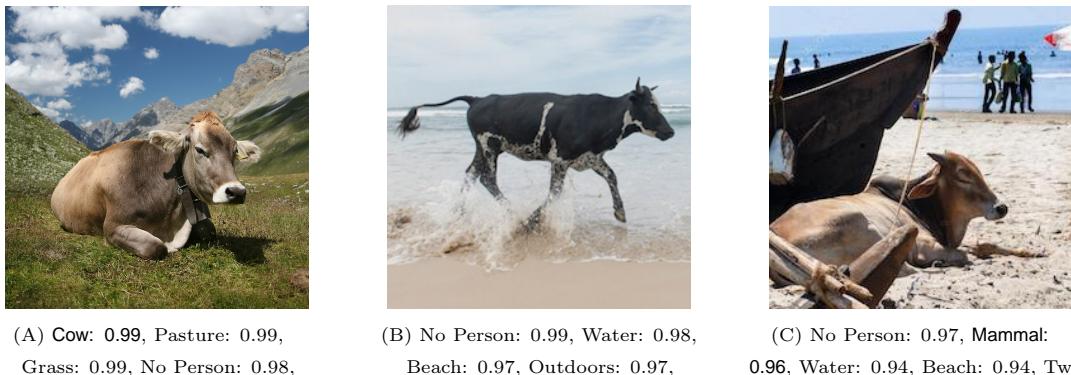
20.2.1 Motivating examples

Figure 20.1 shows how adding a small amount of Gaussian noise can hurt performance of an otherwise high accuracy image classifier. Similar effects occur with other kinds of common corruption, such as image blurring [HD19]. Analogous problems can also occur in the text domain [Ryc+19]. These examples illustrate that models can be very sensitive to small changes in distribution.

Performance can also drop on “clean” images, but which exhibit “semantic shift”. Figure 20.2 gives an amusing example of this. In particular, it illustrates how the performance of a CNN image classifier can be very accurate on **in-domain** data, but can be very inaccurate on **out-of-domain**



10 *Figure 20.1: Effect of Gaussian noise of increasing magnitude on an image classifier. The model is a*
 11 *ResNet-50 CNN trained on ImageNet with added Gaussian noise of standard deviation σ . From Figure 23 of*
 12 *[For+19]. Used with kind permission of Justin Gilmer.*



32
 33 data. Although misclassifying cows may not seem like a big deal, the same problem can plague
 34 medical image classification systems (see e.g., [DJL21]).

35 Analogous problems arise with other kinds of ML models, as well as other data modalities, such as
 36 text (e.g., changing “he” to “she” can flip the output of a sentiment analysis system) and audio (e.g.,
 37 adding background noise can easily confuse speech recognition systems). Furthermore, the changes
 38 to the input needed to change the output can often be imperceptible, as we discuss in the section on
 39 adversarial robustness (Section 20.8).

40 The root cause of many of these problems is the fact that discriminative models often leverage
 41 features that are predictive of the output *in the training set*, but which are not reliable in general.
 42 For example, in an image classification dataset, we may find that green grass in the background
 43 is very predictive of the class label “cow”, but this is not a feature that is stable across different
 44 distributions; this is called a **spurious correlation**. Unfortunately, such **shortcut features** are
 45 often easier for models to learn, for reasons explained in [Gei+20a; Xia+21; Sha+20]. We discuss
 46 some solutions to this problem later in this chapter.

Name	Source	Target	Causal
Covariate / domain shift	$p_{\text{tr}}(X)p_{\text{tr}}(Y X)$	$p_{\text{te}}(X)p_{\text{tr}}(Y X)$	Causal
Concept shift	$p_{\text{tr}}(X)p_{\text{tr}}(Y X)$	$p_{\text{tr}}(X)p_{\text{te}}(Y X)$	Causal
Label (prior) shift	$p_{\text{tr}}(Y)p_{\text{tr}}(X Y)$	$p_{\text{te}}(Y)p_{\text{tr}}(X Y)$	Anti-causal
Conditional shift	$p_{\text{tr}}(Y)p_{\text{tr}}(X Y)$	$p_{\text{tr}}(Y)p_{\text{te}}(X Y)$	Anti-causal

Table 20.1: The 4 main types of distribution shift.

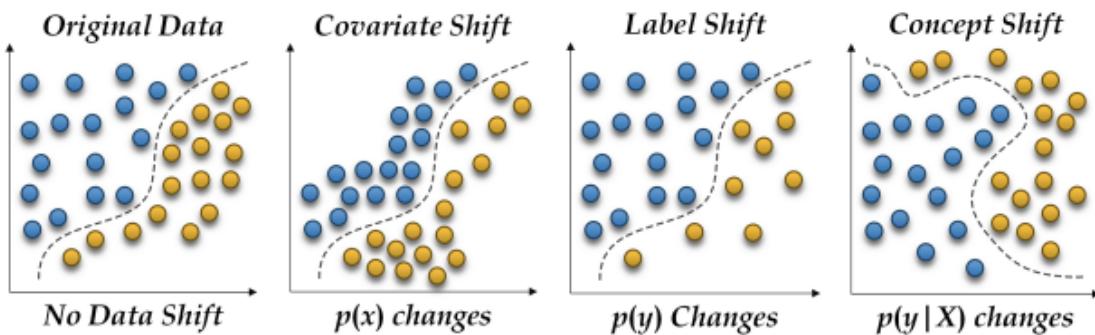


Figure 20.3: Illustration of distribution shift for a 2d binary classification problem. From Figure 1 of [PVP18]. Used with kind permission of Ali Pesaranghader.

20.2.2 A causal view of distribution shift

In the sections below, we briefly summarize some canonical kinds of distribution shift, and some strategies that can be adopted to ameliorate their impact. We adopt a causal view of the problem, as do many papers, e.g., [Sch+12a; Zha+13b; BP16; SS18b; Mei18a; CWG20]).¹ (See Section 4.6 for a brief discussion of causal DAGs, and Chapter 38 for details.)

We assume the inputs to the model (the covariates) are X and the targets to be predicted (the labels) are Y . If we believe that X causes Y , denoted $X \rightarrow Y$, we call it **causal prediction**. If we believe that Y causes X , denoted $Y \rightarrow X$, we call it **anti-causal prediction** [Sch+12a]. The decision about which kind of problem we are dealing with requires understanding the domain, and the nature of the **data generating process**. For example, suppose X is a medical image, and Y is an image segmentation created by a human expert or an algorithm. If we change the image, we will change the annotation, and hence $X \rightarrow Y$. Now suppose X is a medical image and Y is the ground truth disease state of the patient, as estimated by some other means (e.g., a lab test). In this case, we have $Y \rightarrow X$, since changing the disease state will change the appearance of the image. As another example, suppose X is a text review of a movie, and Y is a measure of how informative the review is. Clearly we have $X \rightarrow Y$. Now suppose Y is the star rating of the movie, representing the degree to which the user liked it; this will affect the words that they write, and hence $Y \rightarrow X$.

Based on the above discussion, we can identify 4 main types of distribution shift, as summarized

1. In the causality literature, the question of whether a model can generalize to a new distribution is called the question of **external validity**. If a model is externally valid, we say that it is **transportable** from one distribution to another [BP16].

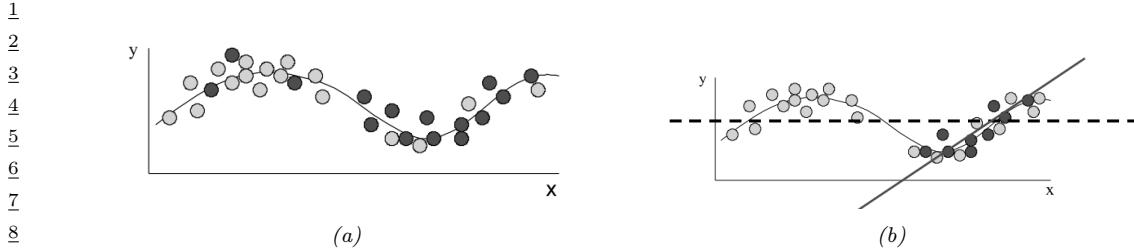


Figure 20.4: (a) Illustration of covariate shift. Light gray represents training distribution, dark gray represents test distribution. We see the test distribution has shifted to the right but the underlying input-output function is constant. (b) Dashed line: fitting a linear model across the full support of X . Solid black line: fitting the same model only on parts of input space that have high likelihood under the test distribution. From Figures 1–2 of [Sto09]. Used with kind permission of Amos Storkey.

15

16

17 in the different causal DAGs in Table 20.1. Note that “manifestation shift”, where $p_{\text{tr}}(Y)p_{\text{tr}}(X|Y)$
18 changes to $p_{\text{tr}}(Y)p_{\text{te}}(X|Y)$, is not widely considered. This leaves the 3 main types shown in Figure 20.3.
19 See the sections below for more details.

20

21 20.2.3 Covariate shift 22

23 In this section, we consider **covariate shift**, which refers to discriminative classifiers of the form
24 $p(X)p(Y|X)$ where $p(X)$ changes. For example, consider the 1d regression problem in Figure 20.4a:
25 the light colored points represent the source distribution, and the dark colored points represent
26 the target distribution. We see that the target distribution is shifted to the right of the source
27 distribution. An example of covariate shift in the medical imaging context can occur if the model is
28 trained on adults and tested on children, or any other kind of **population shift**.

29 For a discriminative model of the form $p(\mathbf{y}|\mathbf{x})$, it might seem that such a change in $p(\mathbf{x})$ will not
30 affect the predictions. If the predictor $p(y|x)$ is the correct model for all parts of the input space \mathbf{x} ,
31 then this conclusion is warranted. However, most models will only be accurate in certain parts of the
32 input space, and may “waste” capacity modeling the training distribution, instead of focusing on the
33 test distribution. In such cases, we might want to bias the model so that it is more accurate on the
34 test distribution, as illustrated in Figure 20.4b. We discuss a way to do this in Section 20.3.1.

35

36 20.2.4 Domain shift 37

38 We now discuss a special kind of covariate shift known as **domain shift**, also called **acquisition**
39 **shift**. Instead of using the model $X \rightarrow Y$, we introduce a latent variable Z which specifies the
40 underlying state of the world (e.g., the shape of an object, or the thought in someone’s head), and we
41 treat X as a noisy measurement of Z . Domain shift refers to the setting in which the sensor model
42 $p(X|Z)$ can change from source to target, even though the distribution over the world states, $p(Z)$,
43 remains the same. The change in $p(Z)$ induces a change in $p(X)$, but we treat this as distinct from
44 covariate shift because the solution techniques can be different (see Section 20.3.2).

45 Here are some examples of domain shift: the training distribution may be clean images of coffee
46 pots, and the test distribution may be images of coffee pots with Gaussian noise, as shown in
47

Figure 20.1; or the training distribution may be photos of objects in a catalog, with uncluttered white backgrounds, and the test distribution may be photos of the same kinds of objects collected “in the wild”; or the training data may be synthetically generated images, and the test distribution may be real images. In all of these cases, the distribution over scenes $p(Z)$ is the same, but the appearance of the scenes differs. Similar shifts can occur in the text domain; for example, the training distribution may be movie reviews written in English, and the test distribution may be translations of these reviews into Spanish.

20.2.5 Label / prior shift

In this section, we consider **label shift**, also called **prior shift** or **prevalence shift**, which refers to generative classifiers of the form $p(Y)p(X|Y)$ where the distribution over labels $p(Y)$ changes. For example, consider the medical imaging context, where $Y = 1$ if the patient has some disease and $Y = 0$ otherwise. If the training distribution is an urban hospital and the test distribution is a rural hospital, then the prevalence of the disease, represented by $p(Y = 1)$, might very well be different.

To tackle this, let us assume we fit the generative classifier $p_{\text{tr}}(\mathbf{y})p_{\text{tr}}(\mathbf{x}|\mathbf{y})$ to labeled data from the source, $\mathcal{D}_1^{XY} \sim p_{\text{tr}}^{XY}$. If we have labeled examples from the target distribution, $\mathcal{D}_2^{XY} \sim p_{\text{te}}^{XY}$, we can estimate $p_{\text{te}}(\mathbf{y})$ and then modify our predictor to be

$$p_{\text{te}}(\mathbf{y}|\mathbf{x}) \propto p_{\text{te}}(\mathbf{y})p_{\text{te}}(\mathbf{x}|\mathbf{y}) = p_{\text{te}}(\mathbf{y})p_{\text{tr}}(\mathbf{x}|\mathbf{y}) \quad (20.1)$$

However, if we don’t have labeled target data, we may still be able to estimate $p_{\text{te}}(\mathbf{y})$, as we discuss in Section 20.3.5.

20.2.6 Concept shift

In this section, we consider **concept shift**, also called **annotation shift**, which refers to discriminative classifiers of the form $p(X)p(Y|X)$ where distribution over labels $p(Y|X)$ changes. For example, consider the medical imaging context: the conventions for annotating images might be different between the training distribution and test distribution. Since this is a difference in what we “mean” by a label, it is hard to fix this problem without coming to some shared agreement. Another example of concept shift occurs when a new label can occur in the target distribution that was not part of the source distribution. This is related to open world recognition, discussed in Section 20.4.4.

20.2.7 Manifestation shift

Conditional shift [Zha+13b], also called **manifestation shift** [CWG20], refers to generative models of the form $p(Y)p(X|Y)$ where the distribution over $p(X|Y)$ changes. This is, in some sense, the inverse of concept shift, and is related to domain shift. For example, consider the medical imaging context: the way that the same disease Y manifests itself in the shape of a tumor X might be different between the training distribution and test distribution for various reasons (e.g., different age of the patients).

20.2.8 Selection bias

In some cases, we may induce a shift in the distribution just due to the way the data is collected. In particular, let $S = 1$ if a sample from the population is included in the training set, and $S = 0$

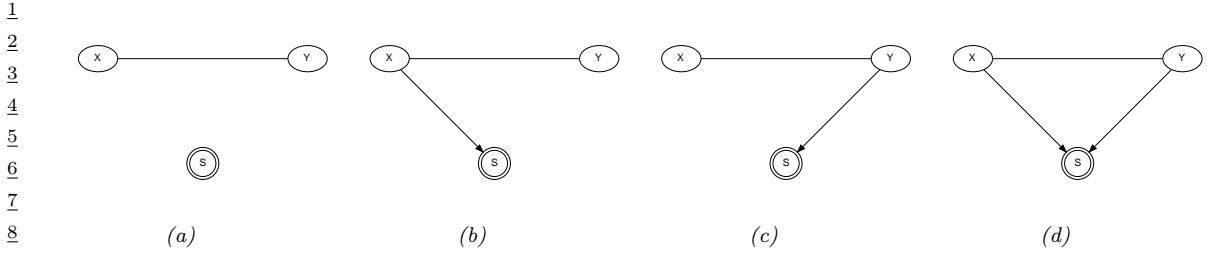


Figure 20.5: Causal diagrams for different kinds of dataset selection. (The double ringed S node is an indicator variable that is set to $S = 1$ iff its parents meet some criterion, and only samples where $S = 1$ are included.) (a) No selection. (b) Selection on X . (c) Selection on Y . (d) Selection on X and Y , which can cause selection bias.

otherwise. We may have $p_{\text{tr}}(X, Y) = p(X, Y|S = 1)$ but $p_{\text{te}}(X, Y) = p(X, Y)$.

In Figure 20.5 we visualize the four kinds of selection. For example, suppose we select based on X meeting certain criteria, e.g., images of a certain quality, or exhibiting a certain pattern; this can induce domain shift or covariate shift. Now suppose we select based on Y meeting certain criteria, e.g., we are more likely to select rare examples where $Y = 1$, in order to **balance the dataset** (for reasons of computational efficiency); this can induce label shift. Finally, suppose we select based on both X and Y ; this can induce non-causal dependencies between X and Y , a phenomenon known as **selection bias** (see Section 4.2.3.2 for details).

20.3 Training-time techniques for distribution shift

Our goal is to train a predictor $\hat{y} = f(\mathbf{x})$ that minimizes the expected loss or risk on the target distribution:

$$\mathcal{L} = \int \int \ell(f(\mathbf{x}), \mathbf{y}) p_{\text{te}}(\mathbf{y}|\mathbf{x}) p_{\text{te}}(\mathbf{x}) d\mathbf{x} d\mathbf{y} \quad (20.2)$$

If we have lots of labeled samples from the target distribution, we can use standard empirical risk minimization. If we have a few labeled samples from the target distribution, we can use a technique called transfer learning, which we discuss in Section 20.5.1. In this section, we assume we only have access to *unlabeled* samples from the target distribution at training time. This can enable us to tackle covariate and domain shift. We can also (with some assumptions) tackle label shift. However, we cannot tackle concept shift or manifestation shift, since this involves a change in the definition of what we mean by each label, and this cannot be learned without labeled examples.

20.3.1 Importance weighting for covariate shift

In this section, we discuss a common technique for minimizing the impact of covariate shift. Suppose for the moment that we know the target distribution, so we can compute the importance weights

$$w_n = \frac{p_{\text{te}}(\mathbf{x}_n)}{p_{\text{tr}}(\mathbf{x}_n)} \quad (20.3)$$

Then we can use weighted empirical risk minimization on $\mathcal{D}_{XY}^{\text{train}}$ to approximate the above risk, as proposed by [Shi00a; SKM07]. Thus our goal becomes

$$\min_f \frac{1}{N} \sum_{n=1}^N w_n \ell(f(\mathbf{x}_n), \mathbf{y}_n) \quad (20.4)$$

where the samples are from the source distribution, $(\mathbf{x}_n, \mathbf{y}_n) \sim p_{\text{tr}}$.

In most cases we do not know the target distribution. However, assume we have access to an unlabeled dataset from the target distribution, $\mathcal{D}_2^X \sim p_{\text{te}}^X$. Rather than trying to estimate the density $p_{\text{te}}(\mathbf{x})$, it suffices instead to estimate the density ratio $p_{\text{te}}(\mathbf{x})/p_{\text{tr}}(\mathbf{x})$. We can do this by fitting a binary classifier, as discussed in Section 2.9.5. In particular, suppose we have an equal number of samples from $p_{\text{tr}}(\mathbf{x})$ and $p_{\text{te}}(\mathbf{x})$. Let us label the first set with $c = -1$ and the second set with $c = 1$. Then the probability of this source / target label for a given sample is

$$p(c = 1|\mathbf{x}) = \frac{p_{\text{te}}(\mathbf{x})}{p_{\text{te}}(\mathbf{x}) + p_{\text{tr}}(\mathbf{x})} \quad (20.5)$$

and hence $\frac{p(c=1|\mathbf{x})}{p(c=-1|\mathbf{x})} = \frac{p_{\text{te}}(\mathbf{x})}{p_{\text{tr}}(\mathbf{x})}$. If the classifier has the form $f(\mathbf{x}) = p(c = 1|\mathbf{x}) = \sigma(h(\mathbf{x})) = \frac{1}{1+\exp(-h(\mathbf{x}))}$, where $h(\mathbf{x})$ is the prediction function that returns the logits, then the importance weights are given by

$$w_n = \frac{1/(1+\exp(-h(\mathbf{x}_n)))}{\exp(-h(\mathbf{x}_n))/(1+\exp(-h(\mathbf{x}_n)))} = \exp(h(\mathbf{x}_n)) \quad (20.6)$$

Of course this method requires that \mathbf{x} values that may occur in the test distribution should also be possible in the training distribution, i.e. $p_{\text{te}}(\mathbf{x}) > 0 \implies p_{\text{tr}}(\mathbf{x}) = 0$. Hence there are no guarantees about this method being able to interpolate beyond the training distribution.

20.3.1.1 Conformal prediction with covariate shift

It is possible to use this importance weighting method for conformal prediction (Section 14.3) in the presence of covariate shift; this is known as **weighted conformal prediction** [Tib+19]. Given a calibration set of N examples, we precompute the importance weights $w_i = \frac{p_{\text{te}}(\mathbf{x}_i)}{p_{\text{tr}}(\mathbf{x}_i)}$, as above. At test time, given new input \mathbf{x}_{N+1} , we compute the weights for $i = 1 : N + 1$:

$$p_i^w(\mathbf{x}) = \frac{w(\mathbf{x}_i)}{\sum_{j=1}^N w(\mathbf{x}_j) + w(\mathbf{x})}, \quad (20.7)$$

We then compute the $1 - \alpha$ quantile of the reweighted distribution of the conformity scores $s_i = s(\mathbf{x}_i, y_i)$:

$$\hat{q}(\mathbf{x}) = \min\{s_j : \sum_{i=1}^j p_i^w(\mathbf{x}) \mathbb{I}(s_i \leq s_j) \geq 1 - \alpha\} \quad (20.8)$$

Finally, we define the predictive set to be $\mathcal{T}(\mathbf{x}) = \{y : s(\mathbf{x}, y) \leq \hat{q}(\mathbf{x})\}$, as usual.

If we set $p_i^w(\mathbf{x}) = \frac{1}{N+1}$, we recover the standard method, which selects the largest $\lceil (N+1)(1-\alpha) \rceil$ 'th largest score for \hat{q} . However, by using weighting, we can adapt the width of the prediction interval based on the input \mathbf{x} : If the covariate shift makes easier values of \mathbf{x} more likely, we make the quantile smaller; if the shift makes harder examples more likely, we make the quantile larger.

Conformal prediction can also be extended to more general kinds of distribution shift. In [Cau+20], they propose a method to adapt the quantile threshold \hat{q} based on the estimated variability of the conformity scores on the calibration set, on the assumption that the test distribution's scores will be reasonably close in distribution (as measured by f -divergence). If the assumptions are met, then the prediction set derived from this “robustified” threshold is guaranteed to have the desired coverage even if the test distribution is different than the calibration distribution.

12

20.3.2 Domain adaptation

13

14

A common approach to minimizing the impact of domain shift is to use a technique called (unsupervised) **domain adaptation** (see e.g., [KL21a] for a review). In this setup, we have a labeled dataset from the source distribution, $\mathcal{D}_1^{XY} \sim p_{\text{tr}}^{XY}$, and an unlabeled dataset from the target distribution, $\mathcal{D}_2^X \sim p_{\text{te}}^X$. We then fit a model on $\mathcal{D}_1^{XY} \cup \mathcal{D}_2^X$, and then evaluate it on a labeled dataset from the target distribution, $\mathcal{D}_2^{XY} \sim p_2^{XY}$.

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

$$\min_{\phi} \max_{\theta} \frac{1}{N_1 + N_2} \sum_{n \in \mathcal{D}_1, \mathcal{D}_2} \ell(d_n, f_{\theta}(\mathbf{x}_n)) + \frac{1}{N_1} \sum_{n \in \mathcal{D}_1} \ell(y_n, g_{\phi}(f_{\theta}(\mathbf{x}_n))) \quad (20.9)$$

where $N_1 = |\mathcal{D}_1|$, $N_2 = |\mathcal{D}_2|$, f maps $\mathcal{X}_1 \cup \mathcal{X}_2 \rightarrow \mathcal{H}$, and g maps $\mathcal{H} \rightarrow \mathcal{Y}_t$. The objective in Equation (20.9) minimizes the loss on the desired task of classifying y , but *maximizes* the loss on the auxiliary task of classifying the domain label d . This can be implemented by the **gradient sign reversal** trick, and is related to GANs (Section 27.7.6).

See [KL21b] for a recent review of other approaches to domain adaptation, and [CB20] for a causal perspective.

20.3.3 Domain randomization

A special case of domain shift occurs when the training set is synthetically generated (e.g., from a computer graphics engine), and the test set is the real world. Such a setting is quite common in robotics. However, a model which is just trained on synthetic data may work poorly when deployed in the field due to the **sim2real gap**. [Tob+17] proposed a simple but effective solution to this known as **domain randomization**. The basic idea is to make the training data be as diverse as possible, by including random backgrounds, random lighting, etc. This will force the learned model to be robust to “semantically irrelevant” changes. Then it can treat the idiosyncrasies of the real world as just another source of noise.

20.3.4 Data augmentation

In settings in which we don't have access to samples from the target distribution, we may be able to simulate such samples by modifying the source data. This is called **data augmentation**, and is widely used in the deep learning community. For example, it is standard to apply small perturbations to images (e.g., shifting them or rotating them), while keeping the label the same (assuming that the label should be invariant to such changes); see e.g., [SK19; Hen+20] for details. Similarly, in NLP (natural language processing), it is standard to change words that should not affect the label (e.g., replacing "he" with "she" in a sentiment analysis system), or to use **back translation** (from a source language to a target language and back) to generate paraphrases; see e.g., [Fen+21] for a review of such techniques. For a causal perspective on data augmentation, see e.g., [Kau+21].

20.3.5 Unsupervised label shift estimation

In this section, we describe an approach known as **black box shift estimation**, due to [LWS18], which can be used to tackle the label shift problem in an unsupervised way. We assume we are using a generative classifier of the form $p(\mathbf{y}|\mathbf{x}) \propto p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$. We also assume the label shift assumption holds, so the only thing that changes in the target distribution is the label prior. In other words, if the source distribution is denoted by $p(\mathbf{x}, \mathbf{y})$ and target distribution is denoted by $q(\mathbf{x}, \mathbf{y})$, we assume $q(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y})q(\mathbf{y})$.

First note that, for any deterministic function $f : \mathcal{X} \rightarrow \mathcal{Y}$, we have

$$p(\mathbf{x}|y) = q(\mathbf{x}|y) \implies p(f(\mathbf{x})|y) = q(f(\mathbf{x})|y) \implies p(\hat{y}|y) = q(\hat{y}|y) \quad (20.10)$$

where $\hat{y} = f(\mathbf{x})$ is the predicted label. Let $\mu_i = q(\hat{y} = i)$ be the empirical fraction of times the model predicts class i on the test set, and let $q(y = i)$ be the true but unknown label distribution on the test set. Then we have

$$\mu(\hat{y}) = \sum_y q(\hat{y}|y)q(y) = \sum_y p(\hat{y}|y)q(y) = \sum_y p(\hat{y}, y) \frac{q(y)}{p(y)} \quad (20.11)$$

We can estimate μ_i from unlabeled test data. We can also estimate the class confusion matrix $C_{ij} = p(\hat{y} = i|y = j)$ on the labeled training (or validation) set. Hence we can solve $\mathbf{q} = \mathbf{C}^{-1}\boldsymbol{\mu}$, providing that \mathbf{C} is not singular. Once we know the new label distribution, $q(\mathbf{y})$, we can adjust our classifier to use $q(\mathbf{y}|\mathbf{x}) \propto q(\mathbf{y})p(\mathbf{x}|\mathbf{y})$.

The confusion matrix will be invertible if \mathbf{C} is strongly diagonal, i.e., the model predicts class y_i correctly more often than any other class y_j . We also require that for every $q(y) > 0$ we have $p(y) > 0$, which means we see every label at training time. Finally the label shift assumption must hold. If these three conditions hold, the above approach is a valid estimator. See [LWS18] for the details.

20.3.6 Distributionally robust optimization

We can make a discriminative model that is robust to (some forms of) covariate shift by modifying the importance weighted optimization problem in Equation (20.4) as follows:

$$\min_{f \in \mathcal{F}} \max_{\mathbf{w} \in \mathcal{W}} \frac{1}{N} \sum_{n=1}^N w_n \ell(f(\mathbf{x}_n), \mathbf{y}_n) \quad (20.12)$$

where the samples are from the source distribution, $(\mathbf{x}_n, \mathbf{y}_n) \sim p_{\text{tr}}$. This is an example of a **min-max optimization problem**, in which we want to minimize the worst case risk. The specification of the robustness set, \mathcal{W} , is a key factor that determines how well the method works, and how difficult the optimization problem is. For details, see e.g., [WYG14; Hu+18]. More generally, we can use methods from **distributionally robust optimization** (see e.g., [CP20a; LFG21]).

20.4 Test-time techniques for distribution shift

In general it will not be possible to make a model robust to all of the ways a distribution can shift at test time, nor will we always have access to test samples at training time. As an alternative, it may be sufficient for the model to *detect* that a shift has happened, and then to respond in the appropriate way. There are several ways of detecting distribution shift, some of which we summarize below. (See also Section 30.5.2, where we discuss changepoint detection in time series data.) The main distinction between methods is based on whether we have a set of samples from the target distribution, or just a single sample, and whether the test samples are labeled or unlabeled. We discuss these different scenarios below.

20.4.1 Detecting shifts using two-sample testing

Suppose we collect a set of samples from the source and target distribution. We can then use standard techniques for **two-sample testing** to estimate if the null hypothesis, $p_{\text{tr}}(\mathbf{x}, \mathbf{y}) = p_{\text{te}}(\mathbf{x}, \mathbf{y})$, is true or not. (If we have unlabeled samples, we just test if $p_{\text{tr}}(\mathbf{x}) = p_{\text{te}}(\mathbf{x})$.) For example, we can use MMD (Section 2.9.3) to measure the distance between the set of samples (see e.g., [Liu+20a]).

In some cases it may be possible to just test if the distribution of the labels $p(y)$ has changed, which is an easier problem than testing for changes in the distribution of inputs $p(\mathbf{x})$. In particular, if the label shift assumption (Section 20.2.5) holds (i.e., $p_{\text{te}}(\mathbf{x}|y) = p_{\text{tr}}(\mathbf{x}|y)$), plus some other assumptions, then we can use the blackbox shift estimation technique from Section 20.3.5 to estimate $p_{\text{te}}(y)$. If we find that $p_{\text{te}}(y) = p_{\text{tr}}(y)$, then we can conclude that $p_{\text{te}}(\mathbf{x}, y) = p_{\text{tr}}(\mathbf{x}, y)$. In [RGL19], they showed experimentally that this method worked well for detecting distribution shifts even when the label shift assumption does not hold.

It is also possible to use conformal prediction (Section 14.3) to develop “distribution free” methods for detecting covariate shift, given only access to a calibration set and some conformity scoring function [HL20].

20.4.2 Detecting single out-of-distribution (OOD) inputs

Now suppose we just have *one* unlabeled sample from the target distribution, $\mathbf{x} \sim p_{\text{te}}$, and we want to know if \mathbf{x} is in-distribution (**ID**) or out-of-distribution (**OOD**). We will call this problem **out-of-distribution detection**, although it is also called **anomaly detection**, and **novelty detection**.²

2. The task of **outlier detection** is somewhat different from anomaly or OOD detection, despite the similar name. In the outlier detection literature, the assumption is that there is a single unlabeled dataset, and the goal is to identify samples which are “untypical” compared to the majority. This is often used for **data cleaning**. (Note that this is a **transductive learning** task, where the model is trained and evaluated on the same data. We focus on inductive tasks, where we train a model on one dataset, and then test it on another.)

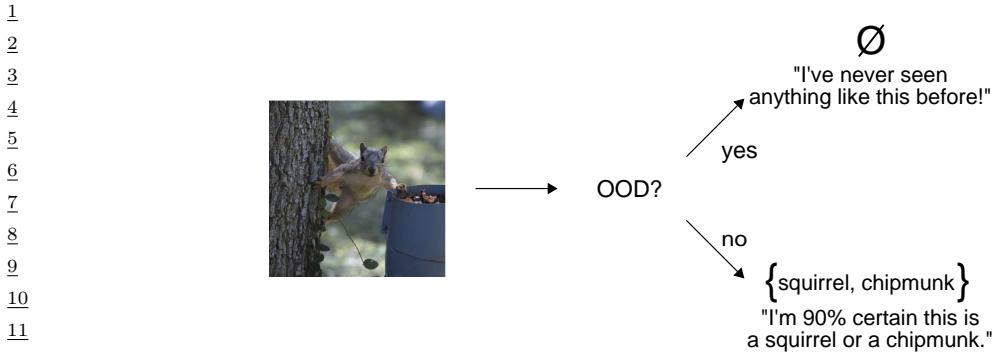


Figure 20.6: Illustration of a two-stage decision problem. First we must decide if the input image is out-of-distribution (OOD) or not. If it is not, we must return the set of class labels that have high probability. From Figure from [AB21]. Used with kind permission of Anastasios Angelopoulos.

The OOD detection problem requires making a binary decision about whether the test sample is ID or OOD. If it is ID, we may optionally require that we return its class label, as shown in Figure 20.6. In the sections below, we give a brief overview of techniques that have been proposed for tackling this problem, but for more details, see e.g., [Pan+21; Ruf+21; Bul+20; Yan+21; Sal+21; Hen+19b].

20.4.2.1 Supervised ID/OOD methods (outlier exposure)

The simplest method for OOD detection assumes we have access to labeled ID and OOD samples at training time. Then we just fit a binary classifier to distinguish the OOD or background class (called “known unknowns”) from the ID class (called “known knowns”). This technique is called **outlier exposure** (see e.g., [HMD19; Thu+21; Bit+21]) and can work well. However, in most cases we will not have enough examples of the OOD “class”, since the OOD set is basically the set of all possible inputs except for the ones of interest.

20.4.2.2 Classification confidence methods

Instead of trying to solve the binary ID/OOD classification problem, we can directly try to predict the class of the input. Let the probabilities over the C labels be $p_c = p(y = c|\mathbf{x})$, and let the logits be $\ell_c = \log p_c$. We can derive a **confidence score** or **uncertainty metric** in a variety of ways from these quantities, e.g., the max probability $s = \max_c p_c$, the margin $s = \max_c \ell_c - \max_c^2 \ell_c$ (where \max^2 means the second largest element), the entropy $s = \mathbb{H}(\mathbf{p}_{1:C})$ ³, the “**energy score**” $\sum_c \ell_c$ [Liu+21], etc. Several sophisticated methods have been proposed for uncertainty quantification, but [Mil+21b; Vaz+22] show that the simple max probability baseline performs very well in practice.

³ [Kir+21] argues against using entropy, since it confuses uncertainty about which of the C labels to use with uncertainty about whether any of the labels is suitable, compared to a “none-of-the-above” option.

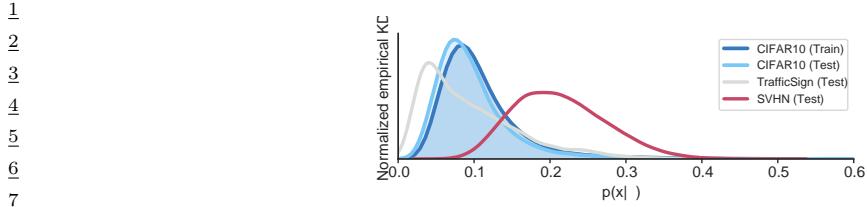


Figure 20.7: Likelihoods from a Glow normalizing flow model (Section 24.2.1) trained on CIFAR10 and evaluated on different test sets. The SVHN street sign dataset has lower visual complexity, and hence higher likelihood. Qualitatively similar results are obtained for other generative models and data set. From Figure 1 of [Ser+20]. Used with kind permission of Joan Serra.

12

13

14 20.4.2.3 Conformal prediction

15

16 It is possible to create a method for OOD detection and ID classification that has provably bounded
17 risk using conformal prediction (Section 14.3). The details are in [Ang+21], but we sketch the basic
18 idea here.

19 We want to solve the two-stage decision problems illustrated in Figure 20.6. We define the
20 prediction set as follows:

21

$$22 \quad \mathcal{T}_\lambda(\mathbf{x}) = \begin{cases} \emptyset & \text{if } \text{OOD}(\mathbf{x}) > \lambda_1 \\ 23 \quad \text{APS}(\mathbf{x}) & \text{otherwise} \end{cases} \quad (20.13)$$

24

25 where $\text{OOD}(\mathbf{x})$ is some heuristic OOD score, and $\text{APS}(\mathbf{x})$ is the adaptive prediction set method of
26 Section 14.3.1, which returns the set of the top K class labels, such that the sum of their probabilities
27 exceeds threshold λ_2 . (Formally, $\text{APS}(\mathbf{x}) = \{\pi_1, \dots, \pi_K\}$ where π sorts $f(\mathbf{x})_{1:C}$ in descending order,
28 and $K = \min\{K : \sum_{c=1}^K f(\mathbf{x})_c > \lambda_2\}$.)

29 We choose the thresholds λ_1 and λ_2 using a calibration set and a frequentist hypothesis testing
30 method (see [Ang+21]). The resulting thresholds will jointly minimize the following risks:

31

$$32 \quad R_1(\lambda) = P(\mathcal{T}_\lambda(\mathbf{x}) = \emptyset) \quad (20.14)$$

$$33 \quad R_2(\lambda) = P(\mathbf{y} \notin \mathcal{T}_\lambda(\mathbf{x}) | \mathcal{T}_\lambda(\mathbf{x}) \neq \emptyset) \quad (20.15)$$

34

35 where $P(\mathbf{x}, \mathbf{y})$ is the true but unknown distribution (of ID samples, no OOD samples required), R_1 is
36 the chance that an ID sample will be incorrectly rejected as OOD (type-I error), and R_2 is the chance
37 (conditional on the decision to classify) that the true label is not in the predicted set. The goal is
38 to set λ_1 as large as possible (so we can detect OOD examples when they arise) while controlling
39 the type-I error (e.g., we may want to ensure that we falsely flag (as OOD) no more than 10% of
40 in-distribution samples). We then set λ_2 in the usual way for the APS method in Section 14.3.1.

41

42 20.4.2.4 Unsupervised methods

43

44 If we don't have labeled examples, a natural approach to OOD detection is to fit an unconditional
45 density model (such as a VAE or AR model) to the ID samples, and then to evaluate the likelihood
46 $p_{\text{tr}}(\mathbf{x})$ and compare this to some threshold value. Unfortunately for many kinds of deep model
47

and datasets, we find that $p_{\text{tr}}(\mathbf{x})$ is lower for samples that are from the training distribution than from a novel test distribution. For example, if we train a pixel-CNN model (Section 23.3.2) or a normalizing-flow model (Chapter 24) on Fashion-MNIST and evaluate it on MNIST, we find it gives higher likelihood to the MNIST samples [Nal+19a; Ren+19; KIW20; ZGR21]. This phenomenon occurs for several other models and datasets (see Figure 20.7).

One solution to this is to use log a **likelihood ratio** relative to a baseline density model, $R(\mathbf{x}) = \log p(\mathbf{x})/q(\mathbf{x})$, as opposed to the raw log likelihood, $L(\mathbf{x}) = \log p(\mathbf{x})$. (This technique was explored in [Ren+19], amongst other papers.) An important advantage of this is that the ratio is invariant to transformations of the data. To see this, let $\mathbf{x}' = \phi(\mathbf{x})$ be some invertible, but possibly nonlinear, transformation. By the change of variables, we have $p(\mathbf{x}') = p(\mathbf{x})|\det \mathbf{J}_{\mathbf{x}}\phi^{-1}(\mathbf{x})|$. Thus $L(\mathbf{x}')$ will differ from $L(\mathbf{x})$ in a way that depends on the transformation. By contrast, we have $R(\mathbf{x}) = R(\mathbf{x}')$, regardless of ϕ , since

$$R(\mathbf{x}') = \log p(\mathbf{x}') - \log q(\mathbf{x}') = \log p(\mathbf{x}) + \log |\det \mathbf{J}_{\mathbf{x}}\phi^{-1}(\mathbf{x})| - \log q(\mathbf{x}) - \log |\det \mathbf{J}_{\mathbf{x}}\phi^{-1}(\mathbf{x})| \quad (20.16)$$

Various other strategies have been proposed, such as computing the log-likelihood adjusted by a measure of the complexity (coding length computed by a lossless compression algorithm) of the input [Ser+20], computing the likelihood of model features (such as the output probability itself) [Mor+21], etc.

A closely related technique relies on **reconstruction error**. The idea is to fit an autoencoder or VAE (Section 22.2) to the ID samples, and then measure the reconstruction error of the input: a sample that is OOD is likely to incur larger error (see e.g. [Pol+19]). However, this suffers from the same problems as density estimation methods.

An alternative to trying to estimate the likelihood, or reconstruct the output, is to use a GAN (Chapter 27) that is trained to discriminate “real” from “fake” data. This has been extended to the open set recognition setting in the **OpenGAN** method of [KR21b].

20.4.3 Selective prediction

In this section, we discuss some ways that a classifier can respond at run time if the input distribution has shifted.

Suppose the system has a confidence level of p that an input is OOD (see Section 20.4.4 for a discussion of some ways to compute such confidence scores). If p is below some threshold, the system may choose to **abstain** from classifying it with a specific label. By varying the threshold, we can control the tradeoff between accuracy and abstention rate. This is called **selective prediction**, and is useful for applications where an error can be more costly than asking a human expert for help (e.g., medical image classification).

20.4.3.1 Example: SGLD vs SGD for MLPs

One way to improve performance of OOD detection is to “be Bayesian” about the parameters of the model, so that the uncertainty in their values is reflected in the posterior predictive distribution. This can result in better performance in selective prediction tasks.

In this section, we give a simple example of this, where we fit a shallow MLP to the MNIST dataset using either standard SGD (specifically RMSprop) or preconditioned Stochastic Gradient Langevin Dynamics (see Section 12.7.1), which is a form of MCMC inference. We use 5,000 training

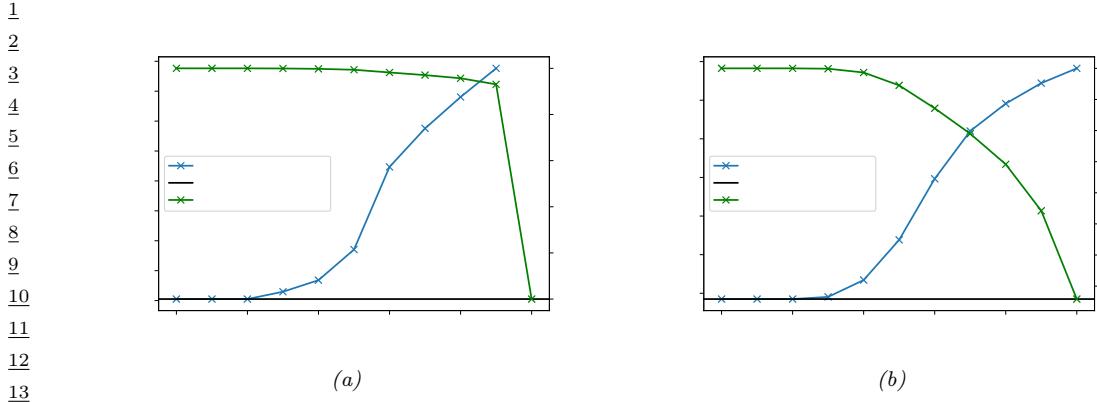


Figure 20.8: Accuracy vs confidence plots for an MLP fit to the MNIST training set, and then evaluated on one batch from the MNIST test set. (a) Plugin approach, computed using SGD. (b) Bayesian approach, computed using 10 samples from SGLD. Generated by [bnn_mnist_sgld_whitejax.ipynb](#).

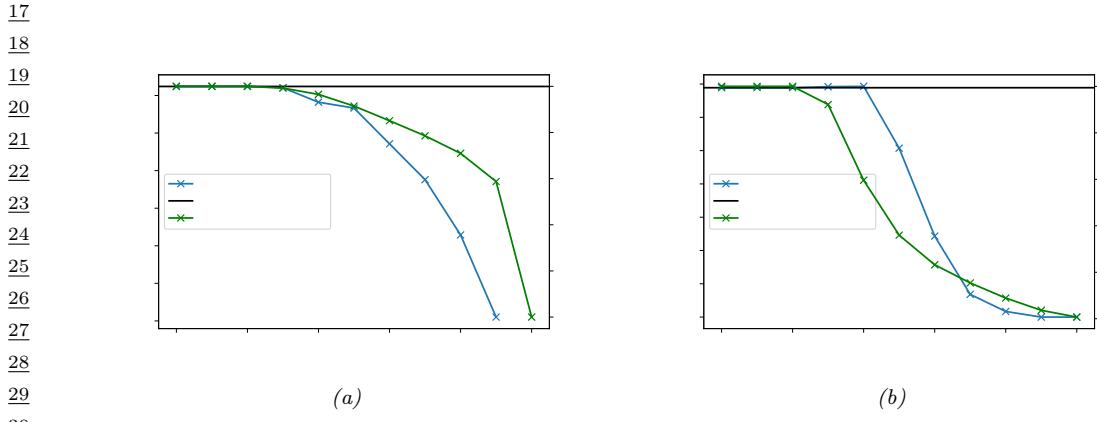


Figure 20.9: Similar to Figure 20.8, except that performance is evaluated on the Fashion MNIST dataset. Generated by [bnn_mnist_sgld.ipynb](#).

steps, where each step uses a minibatch of size 1,000. After fitting the model to the training set, we evaluate its predictions on the test set. To assess how well calibrated the model is, we select a subset of predictions whose confidence is above a threshold t . (The confidence value is just the probability assigned to the MAP class.) As we increase the threshold t from 0 to 1, we make predictions on fewer examples, but the accuracy should increase. This is shown in Figure 20.8 (green curve is the fraction of the test set for which we make a prediction, and blue curve is the accuracy). On the left we show SGD, and on the right we show SGLD. We see that SGD assigns nearly all of the predictions a very high confidence, whereas SGLD is more conservative. The prediction accuracy in both models is very high.⁴

⁴ 4. In this example, SLGD accuracy is slightly worse than SGD, but this can be fixed by suitable tweaking of the hyperparameters.

In Figure 20.9 we show what happens when we apply these models to OOD data, where the inputs are drawn from the FashionMNIST dataset. We see that SGD remains quite confident in many of its predictions: If we consider a confidence threshold of 0.6, the SGD approach predicts on about 80% of the examples, even though the accuracy is only about 6% on these (and this accuracy is at chance level, due to the completely different set of output labels). The SGLD method is more conservative, and only predicts on about 20% of the examples at this confidence threshold.

More details on the behavior of Bayesian neural networks under distribution shift can be found in Section 17.5.7.

20.4.4 Open world recognition

In Section 20.4.3, we discussed methods that “refuse to classify” if the input is suspected to be OOD, i.e., is not one of the existing classes. An alternative approach is to treat the input as an example from a new class. If the set of classes is allowed to grow over time in this way, the problem is called **open world classification** [BB15a].⁵ Note that open world classification is most naturally tackled in the context of a continual learning system, which we discuss in Section 20.7.3.

20.4.5 Online adaptation

In some settings, it is possible to continuously update the model parameters. This allows the model to adapt to changes in the input distribution. If the input stream is labeled, we can use continual learning methods, which we discuss in Section 20.7.

If the input stream is unlabeled, we can use any of the unsupervised adaptation techniques from Section 20.3. In [Sun+20] they proposed an approach called “**test-time training**”, in which a self-supervised proxy task is used to create pseudo-labels, which can then be used to adapt the model at run time. In more detail, suppose we create a Y-structured network, where we first perform feature extraction, $\mathbf{x} \rightarrow \mathbf{h}$, and then use \mathbf{h} to predict the output \mathbf{y} and some proxy output \mathbf{r} , such as the angle of rotation of the input image. The rotation angle is known if we use data augmentation. Hence we can apply this technique at test time, even if \mathbf{y} is unknown, and update the $\mathbf{x} \rightarrow \mathbf{h} \rightarrow \mathbf{r}$ part of the network.

In [ZLF21], they propose a method called “**MEMO**” (marginal entropy minimization with one test point) that can be used for any architecture. The idea is, once again, to apply data augmentation at test time to the input \mathbf{x} , to create a set of inputs, $\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_B$. Now we update the parameters so as to minimize the predictive entropy produced by the averaged distribution

$$\bar{p}(\mathbf{y}|\mathbf{x}, \theta) = \frac{1}{B} \sum_{b=1}^B p(\mathbf{y}|\tilde{\mathbf{x}}_b, \theta) \quad (20.17)$$

This ensures that the model gives the same predictions for each perturbation of the input, and that the predictions are confident (low entropy).

⁵ This is not to be confused with **open set recognition**, which refers to classification problems in which the system should label samples from unknown classes as OOD, rather than trying to incrementally learn about new classes. See e.g., [GHC20] for a review of open set recognition methods.

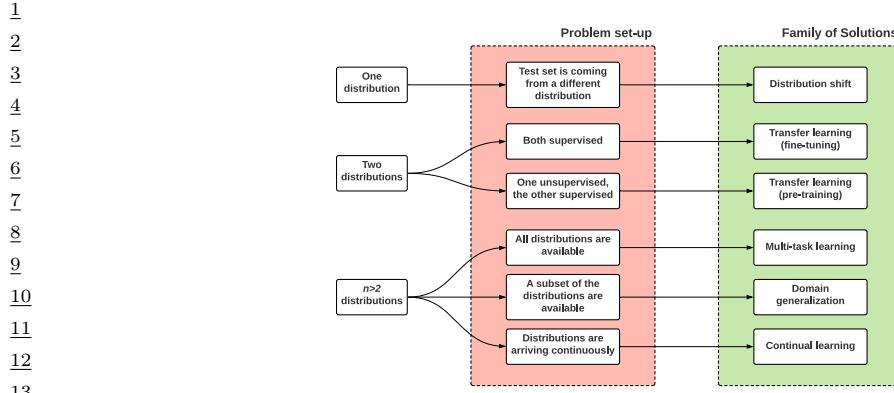


Figure 20.10: Schematic overview of techniques for learning from 1 or more different distributions. Adapted from slide 3 of [Sca21].

20.5 Learning from multiple distributions

In Section 20.2, we discussed distribution shift, in which a model is trained on a source distribution, and then evaluated on a distinct target distribution. In this section, we generalize this to a setting in which the model is trained on data from two or more training **source distributions**, before being tested on data from a **target distribution**. Our goal is to minimize the population risk on the test set

$$R(f, p_{\text{te}}) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p_{\text{te}}} [\ell(\mathbf{y}, f(\mathbf{x}))] \quad (20.18)$$

which we will do by minimizing some variant of the empirical risk on data \mathcal{D} drawn from the source distributions p_{tr} :

$$R(f, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_n, \mathbf{y}_n) \in \mathcal{D}} \ell(\mathbf{y}_n, f(\mathbf{x}_n)) \quad (20.19)$$

We summarize the different problem set-ups that we consider in Figure 20.10.

20.5.1 Transfer learning

Suppose we have labeled training data from a source distribution, $\mathcal{D}_1 \sim p_1$, and also some some labeled data from the target distribution, $\mathcal{D}_2 \sim p_2$, where $p_{\text{te}} = p_2$. Our goal is to minimize the risk on p_{te} , which we can approximate empirically using

$$f_2^* = \underset{f}{\operatorname{argmin}} R(f, \mathcal{D}_2) \quad (20.20)$$

If \mathcal{D}_2 is large enough, we can just ignore \mathcal{D}_1 , and then we have a standard supervised learning problem with a single distribution. However, if \mathcal{D}_2 is small, we might want to somehow use \mathcal{D}_1 to learn a model that works well on p_2 . This is called **transfer learning**, since we hope to “transfer knowledge” from p_1 to p_2 . There are many approaches to transfer learning (see e.g., [Zhu+21] for a review). We briefly mention a few below.

20.5.1.1 Pre-train and fine-tune

The simplest and most widely used approach to transfer learning is the **pre-train and fine-tune** approach. We first fit a model to the source distribution by computing $f_1^* = \operatorname{argmin}_f R(f, \mathcal{D}_1)$, and then we adapt it to work on the target distribution:

$$f_2^* = \operatorname{argmin}_f R(f, \mathcal{D}_2) + \lambda \|f - f_1^*\| \quad (20.21)$$

where $\|f - f_1^*\|$ is some distance between the functions, and $\lambda \geq 0$ controls the degree of regularization.

For example, suppose f_1 is a DNN classifier, and the target distribution has a different label distribution. We define the distance between functions in terms of the Euclidean distance in their parameter vectors. Then we can solve Equation (20.21) by “chopping off the head” from f_1 and replacing it with a new linear layer, to map to the new set of labels, and then just training this final layer. When tackling image classification problems, f_1 is often a large CNN which is pre-trained on ImageNet (with 1000 class labels) or some other large dataset. We can then fine-tune it on a smaller, specialized dataset, such as dogs vs cats or medical images, to create f_2 . Alternatively, f_1 might be a large unsupervised language model (e.g., GPT, Section 23.4.1) to which we add a linear (or nonlinear) classification layer, which we train on the target distribution. Since we assume that we have very few samples from the target distribution, we typically “freeze” most of the parameters of the source model. (This makes an implicit assumption that the features that are useful for the source distribution also work well for the target.)

20.5.2 Few-shot learning

People can learn to predict from very few labeled examples. This is called **few-shot learning** (see e.g., [Lu+20]). If we only have a single example of each class, this is called **one-shot learning**. We can think of the new classes as coming from a new distribution, p_2 , while the existing classes are from p_1 . We usually assume the labeled training set from p_2 is small. The most common ways to tackle FSL are to use transfer learning (Section 20.5.1) and meta learning (Section 20.6). See e.g., [Dum+21] for more details.

20.5.3 Prompt tuning

Recently another approach to FSL has been developed, that leverages large models, such as transformers (Section 23.4), which are trained on massive web datasets, usually in an unsupervised way, and then adapted to a small, task-specific target distribution. The interesting thing about this approach is the parameters of the original model, f_1^* , are not changed; instead, the model is simply “conditioned” on new training data, $\mathcal{D}_2 \sim p_2$, usually in the form of text **prompts**. That is, we compute

$$f_2(\mathbf{x}) = f_1^*(\mathbf{x} \cup \mathcal{D}_2) \quad (20.22)$$

This approach works because f_1 uses attention (Section 16.2.7) to “look at” all its inputs, and modifies its output in response (see Section 23.4.1 for details).

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

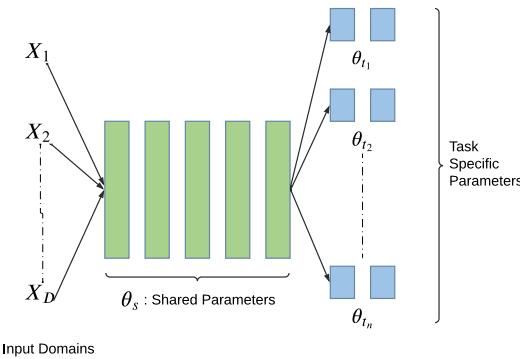


Figure 20.11: Illustration of multi-headed network for multi-task learning.

20.5.4 Zero-shot learning

An extreme case of FSL occurs when no labeled examples of the new class are given; this is called **zero-shot learning**. We cannot solve such problems without additional information, since it is obviously impossible to predict a new label if the label has not been defined, either by examples or by some other means.

There are several approaches to this problem (see e.g., [Pou+20] for a recent review). One method assumes that the class label y can be generated by a known mapping m from a set of attributes, $\mathbf{a} = f(\mathbf{x})$, so $\hat{y}(\mathbf{x}) = m(f(\mathbf{x}))$, where m is fixed. For example, in the context of animal classification, the attributes might be a list of binary features such as: $a_1=\text{hairy}$, $a_2=\text{four-legged}$, $a_3=\text{small}$, $a_4=\text{vegetarian}$, $a_5=\text{swims}$, $a_6=\text{has-tail}$, etc. We then define the label “dog” to be $(a_1 = 1, a_2 = 1, a_3 = ?, a_4 = 0, a_5 = 0, a_6 = 1)$, and the label “cow” to be $(a_1 = 0, a_2 = 1, a_3 = 0, a_4 = 1, a_5 = 0, a_6 = 1)$, etc. As long as the attribute predictor f is transportable from p_1 to p_2 , then we can predict new labels (e.g., “cow”) even if we have never seen them before.

An alternative solution is to train an embedder to map the input and label to a shared low-dimensional embedding space, $\mathbf{e}_x = f_x(\mathbf{x})$, $\mathbf{e}_y = f_y(y)$, and then predict the label using $\hat{y}(\mathbf{x}) = \text{argmax } S(\mathbf{e}_x, \mathbf{e}_y)$ using some similarity metric (e.g., cosine distance). We train the embedding functions f_x, f_y on data from p_1 . If this training distribution is diverse enough, then novel labels from p_2 can still be embedded reliably, and thus the similarity computation is transportable. This is the approach to ZSL used by CLIP (Section 34.1).

38

39

20.5.5 Multi-task learning

In **multi-task learning** [Car97], we have labeled data from T different distributions, $\mathcal{D}^t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : t = 1 : N_t\}$, and the goal is to learn a model that predicts well on all T of them simultaneously:

$$f^* = \underset{f}{\operatorname{argmin}} \mathbb{E}_{(\mathbf{x}, \mathbf{y}_{1:t}) \sim p_{\text{te}}(\mathbf{x}, \mathbf{y})} \left[\sum_{t=1}^T \ell_t(\mathbf{y}_t, f_t(\mathbf{x})) \right] \quad (20.23)$$

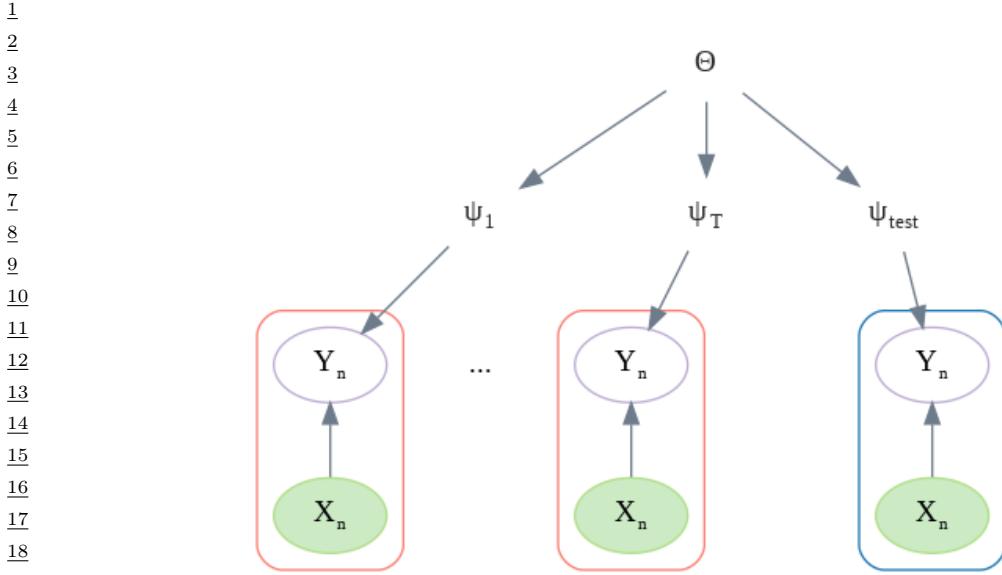


Figure 20.12: Hierarchical Bayesian model for learning from T domains, and then testing on a new distribution. We assume all the parameters of each distribution ϕ_t come from a common prior $p(\psi_t|\Theta)$, which lets us share information between them.

There are many approaches to solving MTL. The simplest is to fit a single model with multiple “output heads”, as illustrated in Figure 20.11. The main challenge is defining a suitable architecture (i.e., deciding which parts of the feature extractor to share across tasks), and how to balance the different losses from each task. See [BLS11] for a theoretical analysis of this problem, and [ZY21] for a more detailed review of neural approaches.

20.5.6 Domain generalization

The problem of **domain generalization** assumes we train on T different labeled source distributions and then test on some unseen labeled target distribution. The source distributions are often called **environments** or **domains**, and are assumed to be related in some way. In some cases the relationship is not specified, so each environment is just identified with a meaningless integer id; in this case, we can think of the set of T distributions as a mixture distribution, and we receive samples from each component, one at a time. In more realistic settings, each different distribution has associated **meta-data** or **context variables** that characterizes the environment in which the data was collected, such as the time, location, imaging device, etc. In both cases, the input to the training algorithm is a set of datasets, $\mathcal{D}_{\text{train}} = \{\mathcal{D}_1, \dots, \mathcal{D}_T\}$, where $\mathcal{D}_t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : n = 1 : N_t\}$, where $(\mathbf{x}_n^t, \mathbf{y}_n^t) \sim p_t$ for domain t . The goal is to learn a prediction function that will have low expected loss on some new distribution:

$$f^* = \underset{f}{\operatorname{argmin}} \mathbb{E}_{\mathcal{D}_{\text{test}} \sim p(\mathcal{D})} [R(f, \mathcal{D}_{\text{test}})] \quad (20.24)$$

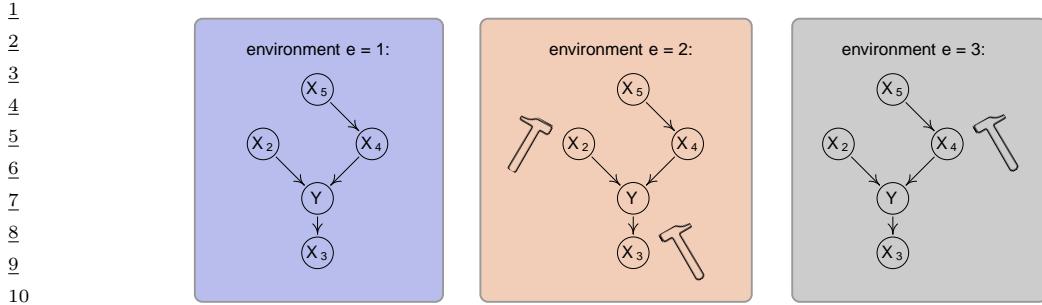


Figure 20.13: Illustration of invariant causal prediction. The hammer symbol represents variables whose distribution is perturbed in the given environment. An invariant predictor must use features $\{X_2, X_4\}$. Considering indirect causes instead of direct ones (e.g. $\{X_2, X_5\}$) or an incomplete set of direct causes (e.g. $\{X_4\}$) may not be sufficient to guarantee invariant prediction. From Figure 1 of [PBM16b]. Used with kind permission of Jonas Peters.

where $p(\mathcal{D})$ is a distribution over datasets or environments.

Since we don't have access to future test distributions, we need to make some assumptions about how the distributions are related, i.e., we need to learn or model $p(\mathcal{D})$. One approach is to adopt a hierarchical Bayesian approach, in which we assume each p_t has its own local parameters, ψ_t , which are all generated from a common prior with hyper-parameters θ . See Figure 20.12 for an illustration for the overall setup.⁶ Alternatively, we can use invariant risk minimization (Section 20.5.7). Many other techniques have been proposed. Note, however, that [GLP21] found that none of these methods (including IRM) worked consistently better than the baseline approach of performing empirical risk minimization across all the provided datasets. For more information, see e.g., [GLP21; She+21; Wan+21] for reviews of this topic, and [Chr+21] for a causal perspective.

20.5.7 Invariant risk minimization

As we discussed in Section 20.2.1, discriminative models are often prone to exploiting "spurious correlations" between the input features and the output label, because these seem to be easier (for current methods) to learn. However, such spurious features are not stable to changes in the distribution. We would like to encourage the model to use stable or causal features, that are invariant across changes in the distribution.

One approach to this problem is to assume that we have samples from multiple training environments, $\mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_T\}$, where $\mathcal{D}_t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) \sim p_t : n = 1 : N_t\}$ are samples from the t 'th environment. We want the learned predictor to work well in a new environment. (This is an example of domain generalization, which we discuss in Section 20.5.6.)

In [PBM16b], they propose a method called **invariant causal prediction**, that uses hypothesis testing methods to find the set of predictors (features) that directly cause the outcome in each environment, rather than features that are indirect causes, or are just correlated with the outcome. See Figure 20.13 for an illustration.

⁶ The difference from the hierarchical model in Figure 17.20a is that here we only care about performance on the new distribution, p_{T+1} , whereas in standard hierarchical modeling, we care about performance on all distributions, $p_{1:T}$.

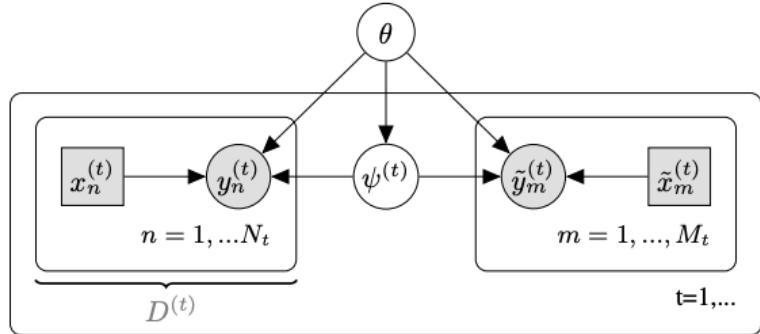


Figure 20.14: Hierarchical Bayesian model for meta-learning. There are T tasks, each of which has a training set $\mathcal{D}^t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : n = 1 : N_t\}$ and a test set $\mathcal{D}_{\text{test}}^t = \{(\tilde{\mathbf{x}}_m^t, \tilde{\mathbf{y}}_m^t) : m = 1 : M_t\}$. ψ^t are the task specific parameters, and θ are the shared parameters. From Figure 1 of [Gor+19]. Used with kind permission of Jonathan Gordon.

In [Arj+19], they proposed an extension of ICP to handle the case of high dimensional inputs, where the individual variables do not have any causal meaning (e.g., they correspond to pixels). Their approach is called **invariant risk minimization** or **IRM**, and corresponds to finding a predictor that works well on average, across all environments, while also being optimal for each individual environment. That is, we want to find

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{t \in \mathcal{E}} \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(\mathbf{y}_n^t, f(\mathbf{x}_n^t)) \quad (20.25)$$

$$\text{such that } f \in \underset{g \in \mathcal{F}}{\operatorname{argmin}} \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(\mathbf{y}_n^t, g(\mathbf{x}_n^t)) \text{ for all } t \in \mathcal{E} \quad (20.26)$$

The intuition behind this is as follows: there may be many functions that achieve low empirical loss on any given environment, since the problem may be underspecified, but if we pick the one that also works well on all environments, it is more likely to rely on causal features rather than spurious features.

Unfortunately, more recent work has shown that the IRM principle often does not work well for covariate shift, both in theory [RRR21] and practice [GLP21], although it can work well in some anti-causal problems [Ahu+21].

20.6 Meta-learning

The goal of **meta-learning** is to “learn the learning algorithm” [TP97]. A common way to do this is to provide the meta-learner with a set of datasets from different distributions, similar to domain generalization (Section 20.5.6). A general review can be found in [Hos+20]. Our presentation follows the unifying “meta-learning as probabilistic inference for prediction” or **ML-PIP** framework of [Gor+19].

¹ ² 20.6.1 Meta-learning as probabilistic inference for prediction

³ We assume there are T tasks (distributions), each of which has a training set $\mathcal{D}^t = \{(\mathbf{x}_n^t, \mathbf{y}_n^t) : n = 1 : N_t\}$ and a test set $\mathcal{D}_{\text{test}}^t = \{(\tilde{\mathbf{x}}_m^t, \tilde{\mathbf{y}}_m^t) : m = 1 : M_t\}$. In addition, ψ^t are the task specific parameters, and θ are the shared parameters. See Figure 20.14. We will learn a point estimate for θ (since it is shared across all datasets, and thus has little uncertainty), but compute an approximate posterior for ψ^t , since each task often has little data. We denote this posterior by $p(\psi^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta)$. From this, we can compute the posterior predictive distribution for each task:

$$\begin{aligned} \text{10} \quad p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta) &= \int p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi^t) p(\psi^t | \mathcal{D}^t, \theta) d\psi^t \\ \text{11} \end{aligned} \quad (20.27)$$

¹² where θ is estimated based on $\mathcal{D}^{1:T}$.

¹³ Since computing the posterior is in general intractable, we will learn an amortized approximation ¹⁴ (see Section 10.3.7) denoted $q_\phi(\psi^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta)$. We choose the parameters of the prior θ and the ¹⁵ inference network ϕ so as to maximize the expected accuracy of the *posterior predictive distribution* ¹⁶ for any given dataset:

$$\begin{aligned} \text{17} \quad \phi^* &= \operatorname{argmin}_\phi \mathbb{E}_{p(\mathcal{D}, \tilde{\mathbf{x}})} [D_{\text{KL}}(p(\tilde{\mathbf{y}} | \tilde{\mathbf{x}}, \mathcal{D}, \theta) \| q_\phi(\tilde{\mathbf{y}} | \tilde{\mathbf{x}}, \mathcal{D}, \theta))] \\ \text{18} \end{aligned} \quad (20.28)$$

$$\begin{aligned} \text{19} \quad &= \operatorname{argmin}_\phi \mathbb{E}_{p(\mathcal{D}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})} \left[\log \int p(\tilde{\mathbf{y}} | \tilde{\mathbf{x}}, \psi) q_\phi(\psi | \mathcal{D}, \theta) \right] \\ \text{20} \end{aligned} \quad (20.29)$$

²² We can make a Monte Carlo approximation to the outer expectation by sampling T tasks (distributions) ²³ from $p(\mathcal{D})$, each of which gets partitioned into a train and test set, $\{(\mathcal{D}^t, \mathcal{D}_{\text{test}}^t) \sim p(\mathcal{D}) : t = 1 : T\}$. ²⁴ We can make an MC approximation to the inner expectation (the integral) by drawing S samples ²⁵ from the task-specific parameter posterior $\psi_s^t \sim q_\phi(\psi^t | \mathcal{D}^t, \theta)$. The resulting objective has the form ²⁶ (where we assume each test set has M samples for notational simplicity):

$$\begin{aligned} \text{27} \quad \mathcal{L}_{\text{ML-PIP}}(\theta, \phi) &= \frac{1}{MT} \sum_{m=1}^M \sum_{t=1}^T \log \left(\frac{1}{S} \sum_{s=1}^S p(\tilde{\mathbf{y}}_m^t | \tilde{\mathbf{x}}_m^t, \psi_s^t) \right) \\ \text{28} \end{aligned} \quad (20.30)$$

³¹ Note that this is different from standard (amortized) variational inference, that focuses on ³² approximating the expected posterior accuracy *of the parameters* given all of the data for a task, ³³ $\mathcal{D}_{\text{all}}^t = \mathcal{D}^t \cup \mathcal{D}_{\text{test}}^t$, rather than focusing on predictive accuracy of a test set given a training set. ³⁴ Indeed, the standard objective has the form

$$\begin{aligned} \text{35} \quad \mathcal{L}_{\text{VI}}(\theta, \phi) &= \frac{1}{T} \sum_{t=1}^T \left(\sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{all}}^t} \left[\frac{1}{S} \sum_{s=1}^S \log p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi_s^t) \right] - D_{\text{KL}}(q_\phi(\psi^t | \mathcal{D}_{\text{all}}^t, \theta) \| p(\psi^t | \theta)) \right) \\ \text{36} \end{aligned} \quad (20.31)$$

³⁹ where $\psi_s^t \sim q_\phi(\psi^t | \mathcal{D}_{\text{all}}^t)$. We see that the standard formulation takes the average of a log, but the ⁴⁰ meta-learning formulation takes the log of an average. The latter can give provably better predictive ⁴¹ accuracy, as pointed out in [MAD20]. Another difference is that the meta-learning formulation ⁴² optimizes the forward KL, not reverse KL. Finally, in the meta-learning formulation, we do not need ⁴³ to specify a prior $p(\psi^t | \theta)$, instead we just need to specify the form of the posterior $q_\phi(\psi^t | \mathcal{D}^t, \theta)$.

⁴⁴ We now show how this ML-PIP framework includes several common approaches to meta-learning. ⁴⁵ Most approaches compute a point estimate for the task-specific parameters $q(\psi^t | \mathcal{D}^t, \theta) = \delta(\psi^t - \psi^*(\mathcal{D}^t, \theta))$, where ψ^* is some function that depends on the method.

20.6.2 Gradient-based meta-learning

In **gradient-based meta-learning**, we define the task specific inference procedure as follows:

$$\psi^*(\mathcal{D}^t, \theta) = \theta + \eta \nabla_{\psi} \log \sum_{n=1}^{N_t} p(\mathbf{y}_n^t | \mathbf{x}_n^t, \psi) |_{\theta} \quad (20.32)$$

That is, we compute the gradient starting at the prior value of θ , and then take one step in that direction, with step size η . This approach is called **model-agnostic meta-learning** or **MAML** [FAL17]. It is also possible to take multiple gradient steps, by feeding the gradient into an RNN [RL17].

20.6.3 Metric-based few-shot learning

Now suppose θ correspond to the parameters of a shared neural feature extractor, $h_{\theta}(\mathbf{x})$, and the task specific parameters are the weights and biases of the last linear layer of a classifier, $\psi^t = \{\mathbf{w}_c^t, b_c^t\}_{c=1}^C$. Let us compute the average of the feature vectors for each class in each task:

$$\mu_c^t = \frac{1}{|\mathcal{D}_c^t|} \sum_{\mathbf{x}_n^c \in \mathcal{D}_c^t} h_{\theta}(\mathbf{x}_n^c) \quad (20.33)$$

Now define the task specific inference procedure as follows:

$$\psi^*(\mathcal{D}^t, \theta) = \{\mu_c^t, -\|\mu_c^t\|^2/2\}_{c=1}^C \quad (20.34)$$

The predictive distribution becomes

$$p(\tilde{y}^t = c | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta) \propto \exp(-d(h_{\theta}(\tilde{\mathbf{x}}), \mu_c^t)) = \exp\left(h_{\theta}(\tilde{\mathbf{x}})^T \mu_c^t - \frac{1}{2} \|\mu_c^t\|^2\right) \quad (20.35)$$

where $d(\mathbf{u}, \mathbf{v})$ is the Euclidean distance. This is equivalent to the technique known as **prototypical networks** [SSZ17].

20.6.4 VERSA

We can also compute a posterior over the parameters of the last layer weights, rather than a point estimate, by using

$$q_{\phi}(\psi | \mathcal{D}, \theta) = \prod_{c=1}^C q_{\phi}(\psi_c | \mathcal{D}_c^t, \theta) \quad (20.36)$$

where $q_{\phi}(\psi_c | \mathcal{D}_c^t, \theta)$ is a network that takes the set of examples of class c in \mathcal{D}^t and returns a distribution over the parameters for class c . This is equivalent to the technique known as **VERSA** [Gor+19].

¹ **20.6.5 Neural processes**

³ In the special case that the task-specific inference network computes a point estimate, $q(\psi^t | \mathcal{D}^t, \theta) =$
⁴ $\delta(\psi^t - \psi^*(\mathcal{D}^t, \theta))$, the posterior predictive distribution becomes
⁵

⁶
$$q(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \mathcal{D}^t, \theta) = \int p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi^t) q(\psi^t | \mathcal{D}^t, \theta) d\psi^t = p(\tilde{\mathbf{y}}^t | \tilde{\mathbf{x}}^t, \psi^*(\mathcal{D}^t, \theta), \theta) \quad (20.37)$$

⁷

⁸ where $\psi^*(\mathcal{D}^t, \theta)$ is a function that takes in a set, and returns some parameters. We can directly
⁹ optimize this by maximum likelihood. This gives rise to a class of methods called **neural processes**
¹⁰ [Gar+18e; Gar+18d]. (See [DGF20] for a good tutorial.)
¹¹

¹² **20.7 Continual learning**

¹³ In this section, we discuss **continual learning** (see e.g., [Had+20; Del+21; Qu+21; LCR21]), also
¹⁴ called **life-long learning** (see e.g., [Thr98; CL18]), in which the system learns from a sequence of
¹⁵ different distributions, p_1, p_2, \dots . In particular, at each time step t , the model receives a batch of
¹⁶ labeled data,
¹⁷

¹⁸
$$\mathcal{D}_t = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N_t, \mathbf{x}_n \sim p_t(\mathbf{x}), \mathbf{y}_n \sim p(\mathbf{y} | f_t(\mathbf{x}_n))\} \quad (20.38)$$

¹⁹

²⁰ where $p_t(\mathbf{x})$ is the unknown input distribution, and $f_t : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ is the unknown prediction function.
²¹ (We typically assume the input space \mathcal{X}_t is the same at each time step (e.g., $\mathcal{X} = \mathbb{R}^D$), although the
²² support p_t over \mathcal{X} can change.) The learner is then expected to update its belief state about the
²³ true function f , and to use it to make predictions on a test set,
²⁴

²⁵
$$\mathcal{D}_t^{\text{test}} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N_t^{\text{test}}, \mathbf{x}_n \sim p_t^{\text{test}}(\mathbf{x}), \mathbf{y}_n \sim p(\mathbf{y} | f_t^{\text{test}}(\mathbf{x}_n))\} \quad (20.39)$$

²⁶

²⁷ Depending on how we assume $p_t(\mathbf{x})$ and f_t evolve over time, and how the test set is defined, we can
²⁸ create a variety of different CL scenarios, as we discuss below.
²⁹

³⁰ **20.7.1 Domain drift**

³¹ The problem of **domain drift** refers to the setting in which $p_t(\mathbf{x})$ changes over time (i.e., covariate
³² shift), but the functional mapping $f_t : \mathcal{X} \rightarrow \mathcal{Y}$ is constant. For example, the vision system of a self
³³ driving car may have to classify cars vs pedestrians under shifting lighting conditions, so we want
³⁴ our model to perform **online adaptation** of its parameters.

³⁵ To evaluate such a model, we can define $p_t^{\text{test}}(\mathbf{x})$ to be the current input distribution p_t (e.g., if it
³⁶ is currently night time, we want the detector to work well on dark images), or we can define it to be
³⁷ the union of all the input distributions, $p_t^{\text{test}} = \cup_{t=1}^T p_t$ (e.g., we want the detector to work well on
³⁸ dark and light images). This latter assumption is illustrated in Figure 20.15.
³⁹

⁴⁰ **20.7.2 Concept drift**

⁴¹ The problem of **concept drift** refers to the setting where the functional mapping $f_t : \mathcal{X} \rightarrow \mathcal{Y}$ changes
⁴² over time, but the input distribution $p_t(\mathbf{x})$ is constant [WK96]. For example, we can imagine a
⁴³ setting in which people engage in certain behaviors, and at step t some of these are classified as
⁴⁴

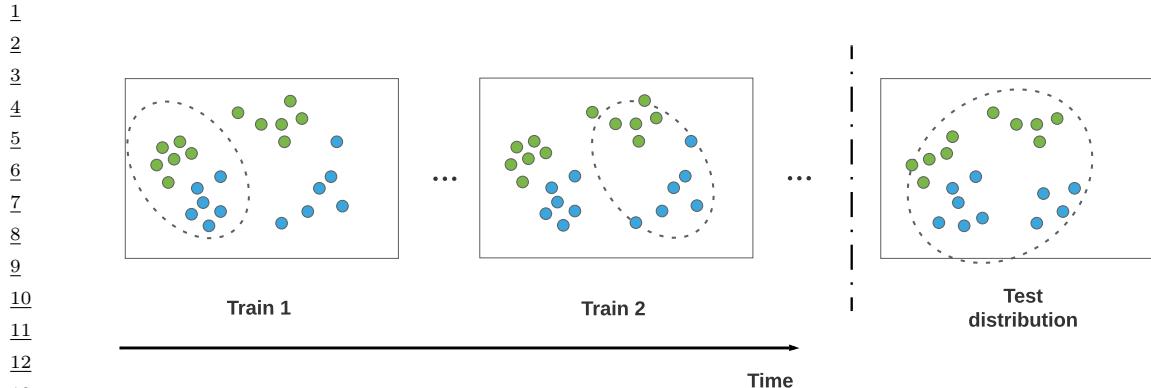


Figure 20.15: An illustration of domain drift.

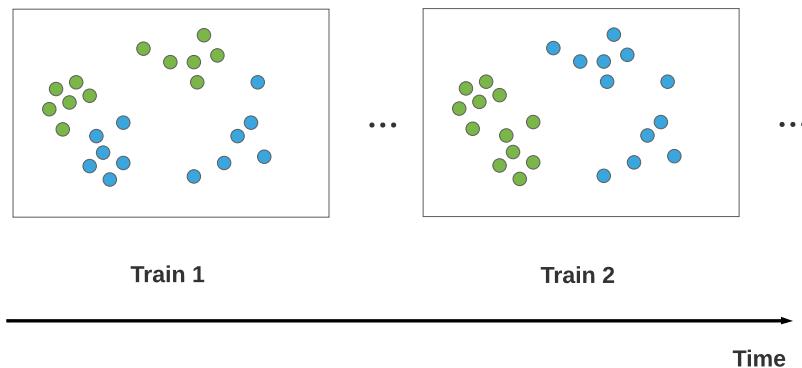


Figure 20.16: An illustration of concept drift.

illegal, and at step $t' > t$, the definition of what is legal changes, and hence the decision boundary changes. This is illustrated in Figure 20.16.

As another example, we might initially be faced with a sort-by-color task, where red objects go on the left and blue objects on the right, and then a sort-by-shape task, where square objects go on the left and circular objects go on the right.⁷ We can think of this as a problem where $p(y|\mathbf{x}, \text{task})$ is stationary, but the task is unobserved, so $p(y|\mathbf{x})$ changes.

In the concept drift scenario, we see that the prediction for the same underlying input point $\mathbf{x} \in \mathcal{X}$ will change depending on when the prediction is performed. This means that the test distribution also needs to change over time for meaningful identification. Alternatively, we can “tag” each input with the corresponding time stamp or task id.

⁴⁶ 7. This example is from Mike Mozer.

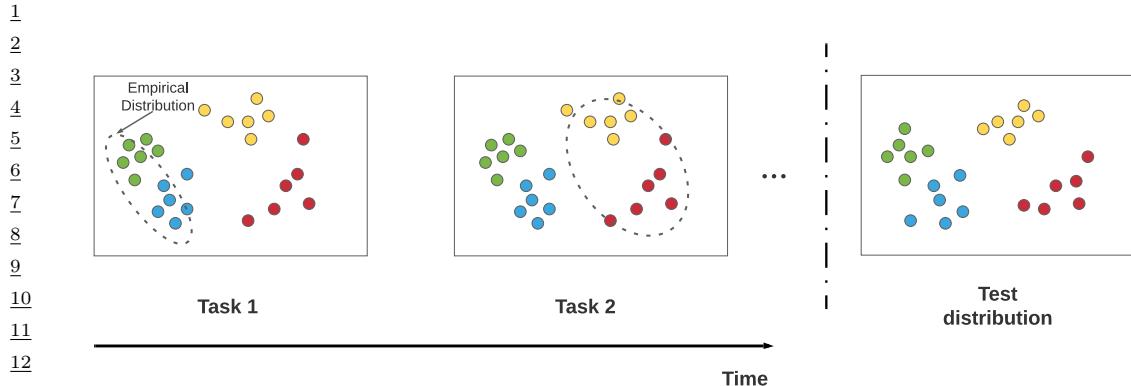


Figure 20.17: An illustration of class incremental learning. Adapted from Figure 1 of [LCR21].

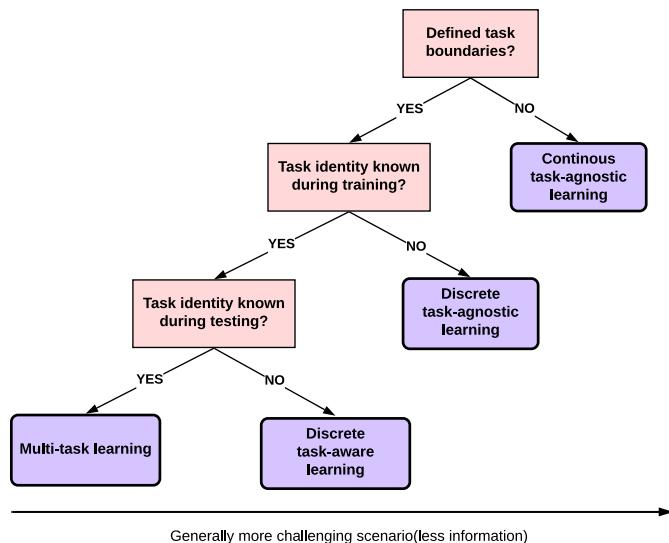


Figure 20.18: Different kinds of incremental learning. Adapted from Figure 1 of [Zen+18].

20.7.3 Task incremental learning

A very widely studied form of continual learning focuses on the setting in which new class labels are “revealed” over time. That is, there is assumed to be a true static prediction function $f : \mathcal{X} \rightarrow \mathcal{Y}$, but at step t , the learner only sees samples from $(\mathcal{X}, \mathcal{Y}_t)$, where $\mathcal{Y}_t \subset \mathcal{Y}$. For example, \mathcal{X} may be the space of images, and \mathcal{Y}_1 might be {cats, dogs}, and \mathcal{Y}_2 might be {cars, bikes, trucks}. Learning to classify with an increasing number of categories is called **class incremental learning** (see e.g., [Mas+20]). This is also called **task incremental learning**, since each distribution is considered

1 as a different **task**, also **data stream classification** (see e.g., [Din+21]). See Figure 20.17 for an
2 illustration.
3

4 The problem of class incremental learning has been studied under a variety of different assumptions,
5 as discussed in [Hsu+18; VT18; FG18; Del+21]. The most common scenarios are shown in Figure 20.18.
6 If we assume there are no well defined boundaries between tasks (as often occurs in distribution drift
7 or concept drift), we have **continuous task-agnostic learning**. If there are well defined boundaries
8 (i.e., discontinuous changes of the training distribution), then we can distinguish two subcases. If the
9 boundaries are not known during training (similar to detecting distribution shift), we have **discrete**
10 **task-agnostic learning**. Finally, if the boundaries are given to the training algorithm, we have a
11 **task-aware continual learning** problem.

12 A common experimental setup in the task-aware setting is to define each task to be a different
13 version of the MNIST dataset, e.g., with all 10 classes present but with the pixels randomly permuted
14 (this is called **permuted MNIST**) or with a subset of 2 classes present at each step (this is called
15 **split MNIST**).⁸ In the task-aware setting, the task label may or may not be known at test time.
16 If it is, the problem is essentially equivalent to multi-task learning (see Section 20.5.5). If it is not,
17 the model must predict the task and corresponding class label within that task (which is a standard
18 supervised problem with a hierarchical label space); this is commonly done by using a **multi-headed**
19 **DNN**, with CT outputs, where C is the number of classes, and T is the number of tasks.

20 In the multi-headed approach, the number of “heads” is usually specified as input to the algorithm,
21 because the softmax imposes a sum-to-one constraint that prevents incremental estimation of the
22 output weights in the open-class setting. An alternative approach is to wait until a new class label is
23 encountered for the first time, and then train the model with an enlarged output head. This requires
24 storing past data from each class, as well as data the new class (see e.g., [PTD20]). Alternatively, we
25 can use generative classifiers where we do not need to worry about “output heads”. If we use a “deep”
26 nearest neighbor classifier, with a shared feature extractor (embedding function), the main challenge
27 is to efficiently update the stored prototypes for past classes as the feature extractor parameters
28 change (see e.g., [DLT21]). If we fit a separate generative model per class (e.g., a VAE, as in [VLT21]),
29 online learning becomes easier, but the method may be less sample efficient.

30 At the time of writing, most of the CL literature focuses on the task-aware setting. However, from
31 a practical point of view, the assumption that task boundaries are provided at training or test time is
32 very unrealistic. For example, consider the problem of training a robot to perform various activities:
33 The data just streams in, and the robot must learn what to do, without anyone telling it that it is
34 now being given an example from a new task or distribution (see e.g., [Fon+21; Woł+21]). Thus
35 future research should focus on the task-agnostic setting, with either discrete or continuous changes.
36

37 20.7.4 Catastrophic forgetting

38 In the class incremental learning literature, it is common to train on a sequence of tasks, but to
39 test (at each step) on all tasks. In this scenario, there are two main possible failure modes. The
40 first possible problem is called “**catastrophic forgetting**” (see e.g., [Rob95b; Fre99; Kir+17]). This
41 refers to the phenomenon in which performance on a previous task drops when trained on a
42

43 44 8. In the split MNIST setup, for task 1, digits (0,1) get labeled as (0,1), but in task 2, digits (2,3) get labeled as (0,1).
45 So the “meaning” of the output label depends on what task we are solving. (It seems odd to “reuse” the same label for
46 perceptually different things, but there are some problems where words can be ambiguous, so one could argue this
47 problem setting is worth studying.)

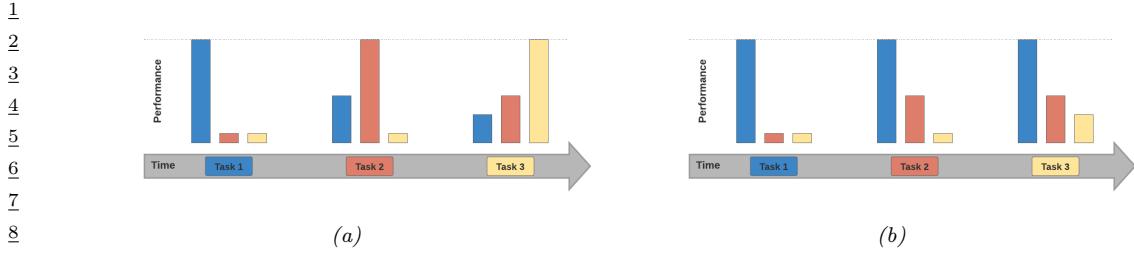


Figure 20.19: Some failure modes in class incremental learning. We train on task 1 (blue) and evaluate on tasks 1–3 (blue, orange, yellow); we then train on task 2 and evaluate on tasks 1–3; etc. (a) Catastrophic forgetting refers to the phenomenon in which performance on a previous task drops when trained on a new task. (b) Too little plasticity (e.g., due to too much regularization) refers to the phenomenon in which only the first task is learned. Adapted from Figure 2 of [Had+20].

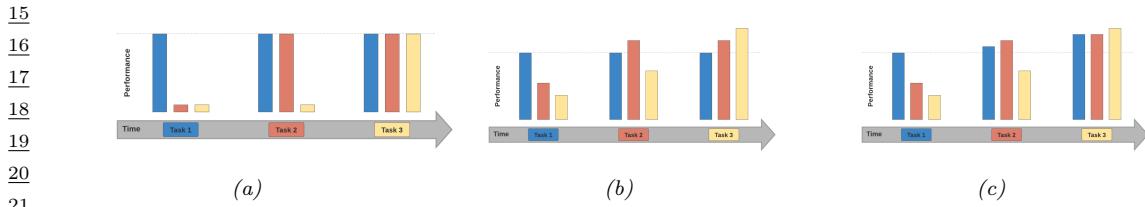


Figure 20.20: What success looks like for class incremental learning. We train on task 1 (blue) and evaluate on tasks 1–3 (blue, orange, yellow); we then train on task 2 and evaluate on tasks 1–3; etc. (a) No forgetting refers to the phenomenon in which performance on previous tasks does not degrade over time. (b) Forward transfer refers to the phenomenon in which training on past tasks improves performance on future tasks beyond what would have been obtained by training from scratch. (c) Backwards transfer refers to the phenomenon in which training on future tasks improves performance on past tasks beyond what would have been obtained by training from scratch. Adapted from Figure 2 of [Had+20].

new task (see Figure 20.19(a)). Another possible problem is that only the first task is learned, and the model does not adapt to new tasks (see Figure 20.19(b)).

If we avoid these problems, we should expect to see the performance profile in Figure 20.20(a), where performance of incremental training is equal to training on each task separately. However, we might hope to do better by virtue of the fact that we are training on multiple tasks, which are often assumed to be related. In particular, we might hope to see **forward transfer**, in which training on past tasks improves performance on future tasks beyond what would have been obtained by training from scratch (see Figure 20.20(b)). Additionally, we might hope to see **backwards transfer**, in which training on future tasks improves performance on past tasks (see Figure 20.20(c)). We can quantify the degree of transfer as follows, following [LPR17]. If R_{ij} is the performance on task j after it was trained on task i , R_j^{ind} is the performance on task j when trained just on j , and there are T tasks, then the amount of forward transfer is

$$\text{FWT} = \frac{1}{T} \sum_{j=1}^T R_{j,j} - R_j^{\text{ind}} \quad (20.40)$$

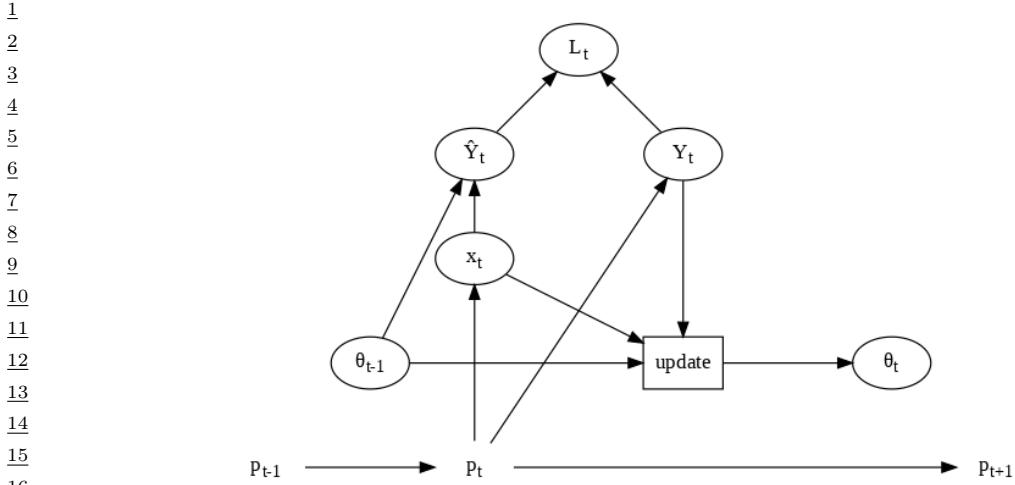


Figure 20.21: Online learning illustrated as a graphical model.

and the amount of backwards transfer is

$$\text{BWT} = \frac{1}{T} \sum_{j=1}^T R_{T,j} - R_{j,j} \quad (20.41)$$

There are many methods that have been devised to overcome the problem of catastrophic forgetting, but we can group them into three main types. The first is **regularization methods**, which add a loss to preserve information that is relevant to old tasks. (For example, online Bayesian inference is of this type, since the posterior for the parameters is derived from the new data and the past prior; see e.g., the **elastic weight consolidation** method discussed in Section 17.6.3, or the **variational continual learning** method discussed in Section 10.3.9). The second is **memory methods**, which rely on some kind of **experience replay** or **rehearsal** of past data (see e.g., [Hen+21]), or some kind of generative model of past data. The third is **architectural methods**, that add capacity to the network whenever a task boundary is encountered, such as a new class label (see e.g., [Rus+16]).

Of course, these techniques can be combined. For example, we can create a semi-parametric model, in which we store some past data (exemplars) while also learning parameters online in a Bayesian (regularized) way (see e.g., [Kur+20]). The “right” method depends, as usual, on what inductive bias you want to use, and want your computational budget is in terms of time and memory.

20.7.5 Online learning

The problem of **online learning** refers to the setting in which the input distribution $p_t(\mathbf{x})$ and the target function f_t can change at each step, but where we only evaluate performance on one-step-ahead predictions. That is, at each step, the algorithm obtains an unlabeled sample, $\mathbf{x}_t \sim p_t(\mathbf{x})$, it makes a prediction $\hat{\mathbf{y}}_{t|t-1} = \operatorname{argmax} p(\mathbf{y}|\mathbf{x}_t, \mathcal{D}_{1:t-1})$, and then incurs loss $\mathcal{L}_t = \ell(\hat{\mathbf{y}}_{t|t-1}, \mathbf{y}_t)$, where $\mathbf{y}_t = f_t(\mathbf{x}_t)$ is the true label. Finally, it updates its belief about the unknown prediction function, $p(\theta_t|\mathcal{D}_{1:t})$. See Figure 20.21 for an illustration. (This setup is also called **prequential inference** [DV99].)

1 In contrast to the continual learning scenarios studied above, the loss incurred at each step is what
2 matters, rather than loss on a fixed test set. That is, we want to minimize
3

$$\underline{4} \quad \mathcal{L} = \sum_{t=1}^T \mathcal{L}_t = \sum_{t=1}^T \ell(\hat{\mathbf{y}}_{t|t-1}, \mathbf{y}_t) \quad (20.42)$$

5 Since it is hard to interpret this number, it is common to compare it to the optimal value one could
6 have obtained in hindsight. This yields a quantity called the **regret**:

$$\underline{7} \quad \text{regret} = \sum_{t=1}^T [\ell(\hat{\mathbf{y}}_{t:t-1}, \mathbf{y}_t) - \ell(\hat{\mathbf{y}}_{t:T}, \mathbf{y}_t)] \quad (20.43)$$

8 where $\hat{\mathbf{y}}_{t|t-1} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}_t, \mathcal{D}_{1:t-1})$ is the online prediction, and $\hat{\mathbf{y}}_{t:T} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}_t, \mathcal{D}_{1:T})$ is
9 the optimal estimate at the end of training. It is possible to convert bounds on regret, which are
10 backwards looking, into bounds on risk (i.e., expected future loss), which is forwards looking. See
11 [HT15] for details.

12 Online learning is very useful for decision and control problems, such as multi-armed bandits
13 (Section 36.4) and reinforcement learning (see Chapter 37), where the agent “lives forever”, and
14 where there is no fixed training phase followed by a test phase. However, it is possible to include
15 the previous continual learning scenarios as special cases of online learning, by defining a suitable
16 sequence of distributions. We also need to distinguish between steps in which we train (i.e., perform
17 an update of our model) and steps in which we just test (with the model held fixed). Furthermore,
18 we allow the training steps and test steps to work with minibatches of samples, instead of individual
19 samples. We call this “**generalized online learning**”.

20 With this setup, we can capture class incremental learning as follows: at step 1, we train on
21 samples from p_1 , and at step 1’, we test on samples from $p_* = p_{1:T}$; at step 2, we train on samples
22 from p_2 , and at step 2’, we test on samples from p_* ; etc. This is a special case of generalized online
23 learning in which the sequence of distributions has the form $p_1, p_*, p_2, p_*, \dots$, and the training mode
24 bit sequence has the form 1, 0, 1, 0, … . This framing makes clear the assumption that we should only
25 evaluate performance on all tasks, p_* , if we think they will occur again in the future. By contrast,
26 if a task will not repeat again, it is okay to forget about it (something that commonly happens in
27 human learning, due to finite memory and compute abilities).

28 Similarly, we can frame distribution shift and concept shift as special cases of (generalized) online
29 learning. For distribution shift, we let $p_t(\mathbf{x})$ evolve, but keep f_t fixed, and we alternate between
30 training on $p_t(\mathbf{x}, \mathbf{y})$ and testing on $p_*(\mathbf{x}, \mathbf{y})$. For concept drift, we keep $p_t(\mathbf{x})$ fixed, but let f_t evolve,
31 and we alternate between training on $p_t(\mathbf{x}, \mathbf{y})$ and testing on $p_t(\mathbf{x}, \mathbf{y})$.

32 20.8 Adversarial examples

33 *This section is coauthored with Justin Gilmer.*

34 In Section 20.2, we discussed what happens to a predictive model when the input distribution
35 shifts for some reason. In this section, we consider the case where an adversary deliberately chooses
36 inputs to minimize the performance of a predictive model. That is, suppose an input \mathbf{x} is classified
37 as belonging to class c . We then choose a new input \mathbf{x}_{adv} which minimizes the probability of this
38

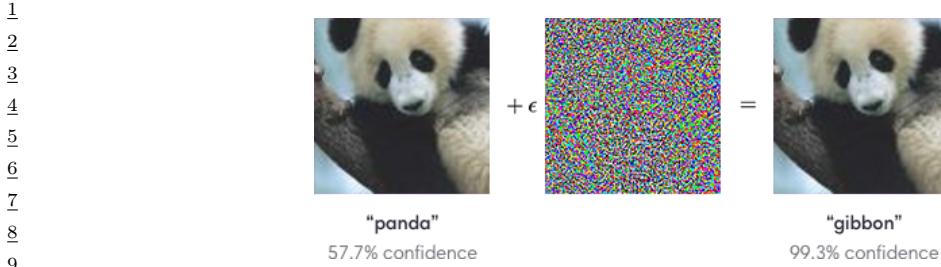


Figure 20.22: Example of an adversarial attack on an image classifier. Left column: original image which is correctly classified. Middle column: small amount of structured noise which is added to the input (magnitude of noise is magnified by $10\times$). Right column: new image, which is confidently misclassified as a “gibbon”, even though it looks just like the original “panda” image. Here $\epsilon = 0.007$. From Figure 1 of [GSS15]. Used with kind permission of Ian Goodfellow.

label, subject to the constraint that \mathbf{x}_{adv} is “perceptually similar” to the original input \mathbf{x} . This gives rise to the following objective:

$$\mathbf{x}_{\text{adv}} = \underset{\mathbf{x}' \in \Delta(\mathbf{x})}{\operatorname{argmin}} \log p(y = c | \mathbf{x}') \quad (20.44)$$

where $\Delta(\mathbf{x})$ is the set of images that are “similar” to \mathbf{x} (we discuss different notions of similarity below).

Equation (20.44) is an example of an **adversarial attack**. We illustrate this in Figure 20.22. The input image \mathbf{x} is on the left, and is predicted to be a panda with probability 57%. By adding a tiny amount of carefully chosen noise (shown in the middle) to the input, we generate the **adversarial image** \mathbf{x}_{adv} on the right: this “looks like” the input, but is now classified as a gibbon with probability 99%.

The ability to create adversarial images was first noted in [Sze+14]. It is surprisingly easy to create such examples, which seems paradoxical, given the fact that modern classifiers seem to work so well on normal inputs, and the perturbed images “look” the same to humans. We explain this paradox in Section 20.8.5.

The existence of adversarial images also raises security concerns. For example, [Sha+16] showed they could force a face recognition system to misclassify person A as person B , merely by asking person A to wear a pair of sunglasses with a special pattern on them, and [Eyk+18] show that is possible to attach small “**adverarial stickers**” to traffic signs to classify stop signs as speed limit signs.

Below we briefly discuss how to create adversarial attacks, why they occur, and how we can try to defend against them. We focus on the case of deep neural nets for images, although it is important to note that many other kinds of models (including logistic regression and generative models) can also suffer from adversarial attacks. Furthermore, this is not restricted to the image domain, but occurs with many kinds of high dimensional inputs. For example, [Li+19] contains an audio attack and [Dal+04; Jia+19] contains a text attack. More details on adversarial examples can be found in e.g., [Wiy+19; Yua+19].

1 **20.8.1 Whitebox (gradient-based) attacks**

3 To create an adversarial example, we must find a “small” perturbation δ to add to the input x to
4 create $x_{\text{adv}} = x + \delta$ so that $f(x_{\text{adv}}) = y'$, where $f()$ is the classifier, and y' is the label we want to
5 force the system to output. This is known as a **targeted attack**. Alternatively, we may just want
6 to find a perturbation that causes the current predicted label to change from its current value to any
7 other value, so that $f(x + \delta) \neq f(x)$, which is known as **untargeted attack**.

8 In general, we define the objective for the adversary as *maximizing* the following loss:

$$\underline{10} \quad x_{\text{adv}} = \underset{x' \in \Delta(x)}{\operatorname{argmax}} \mathcal{L}(x', y; \theta) \quad (20.45)$$

12 where y is the true label. For the untargeted case, we can define $\mathcal{L}(x', y; \theta) = -\log p(y|x')$, so we
13 minimize the probability of the true label; and for the targeted case, we can define $\mathcal{L}(x', y; \theta) =$
14 $\log p(y'|x')$, where we maximize the probability of the desired label $y' \neq y$.

15 To define what we mean by “small” perturbation, we impose the constraint that $x_{\text{adv}} \in \Delta(x)$,
16 which is the set of “perceptually similar” images to the input x . Most of the literature has focused
17 on a simplistic setting in which the adversary is restricted to making bounded l_p perturbations of a
18 clean input x , that is

$$\underline{20} \quad \Delta(x) = \{x' : \|x' - x\|_p < \epsilon\} \quad (20.46)$$

21 Typically people assume $p = 1$ or $p = 0$. We will discuss more realistic threat models in Section 20.8.3.
22 In this section, we assume that the attacker knows the model parameters θ ; this is called a
23 **whitebox attack**, and lets us use gradient based optimization methods. We relax this assumption
24 in Section 20.8.2.)

25 To solve the optimization problem in Equation (20.45), we can use any kind of constrained
26 optimization method. In [Sze+14] they used bound-constrained BFGS. [GSS15] proposed the more
27 efficient **fast gradient sign (FGS)** method, which performs iterative updates of the form

$$\underline{29} \quad x_{t+1} = x_t + \delta_t \quad (20.47)$$

$$\underline{30} \quad \delta_t = \epsilon \operatorname{sign}(\nabla_x \log p(y'|x, \theta)|_{x_t}) \quad (20.48)$$

32 where $\epsilon > 0$ is a small learning rate. (Note that this gradient is with respect to the input pixels, not
33 the model parameters.) Figure 20.22 gives an example of this process.

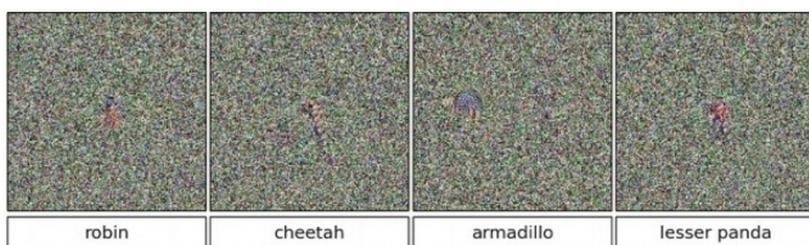
34 More recently, [Mad+18] proposed the more powerful **projected gradient descent (PGD)**
35 attack; this can be thought of as an iterated version of FGS. There is no “best” variant of PGD for
36 solving 20.45. Instead, what matters more is the implementation details, e.g. how many steps are
37 used, the step size, and the exact form of the loss. To avoid local minima, we may use random restarts,
38 choosing random points in the constraint space Δ to initialize the optimization. The algorithm
39 should be carefully tuned to the specific problem, and the loss should be monitored to check for
40 optimization issues. For best practices, see [Car+19].

41

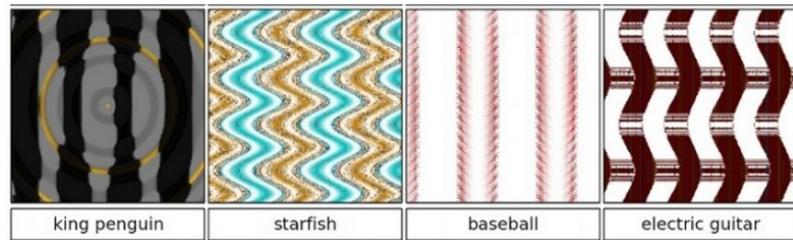
42 **20.8.2 Blackbox (gradient-free) attacks**

44 In this section, we no longer assume that the adversary knows the parameters θ of the predictive model
45 f . This is known as a **black box attack**. In such cases, we must use derivative-free optimization
46 methods (see Section 6.12).

47



10 *Figure 20.23: Images that look like random noise but which cause the CNN to confidently predict a specific*
 11 *class. From Figure 1 of [NYC15]. Used with kind permission of Jeff Clune.*



22 *Figure 20.24: Synthetic images that cause the CNN to confidently predict a specific class. From Figure 1 of*
 23 *[NYC15]. Used with kind permission of Jeff Clune.*

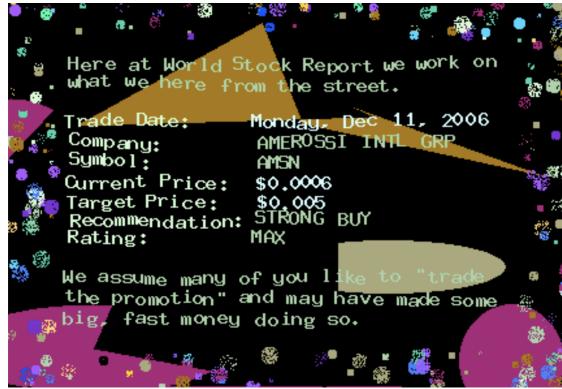
24
25
26
27
28 Evolutionary algorithms (EA) are one class of DFO solvers. These were used in [NYC15] to create
 29 blackbox attacks. Figure 20.23 shows some images that were generated by applying an EA to a
 30 random noise image. These are known as **fooling images**, as opposed to adversarial images, since
 31 they are not visually realistic. Figure 20.24 shows some fooling images that were generated by
 32 applying EA to the parameters of a compositional pattern-producing network [Sta07].⁹ By suitably
 33 perturbing the CPPN parameters, it is possible to generate structured images with high fitness
 34 (classifier score), but which do not look like natural images [Aue12].

35 In [SVK19], they used differential evolution to attack images by modifying a single pixel. This is
 36 equivalent to bounding the ℓ_0 norm of the perturbation, so that $\|\mathbf{x}_{\text{adv}} - \mathbf{x}\|_0 = 1$.

37 In [Pap+17], they learned a differentiable surrogate model of the blackbox, by just querying its
 38 predictions y for different inputs \mathbf{x} . They then used gradient-based methods to generate adversarial
 39 attacks on their surrogate model, and then showed that these attacks transferred to the real model.
 40 In this way, they were able to attack various the image classification APIs of various cloud service
 41 providers, including Google, Amazon and MetaMind.

42
43
44 9. A CPPN is a set of elementary functions (such as linear, sine, sigmoid, and Gaussian) which can be composed in
 45 order to specify the mapping from each coordinate to the desired color value. CPPN was originally developed as a way
 46 to encode abstract properties such as symmetry and repetition, which are often seen during biological development.
 47

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 20.25: An adversarially modified image to evade spam detectors. The image is constructed from*
15 *scratch, and does not involve applying a small perturbation to any given image. This is an illustrative example*
16 *of how large the space of possible adversarial inputs Δ can be when the attacker has full control over the input.*
17 *From [Big+11]. Used with kind permission of Battista Biggio.*

18
19

20 20.8.3 Real world adversarial attacks

22 Typically, the space of possible adversarial inputs Δ can be quite large, and will be difficult to exactly
23 define mathematically as it will depend on semantics of the input based on the attacker's goals
24 [BR18]. (The set of variations Δ that we want the model to be invariant to is called the **threat**
25 **model**.)

26 Consider for example of the content constrained threat model discussed in [Gil+18a]. One instance
27 of this threat model involves image spam, where the attacker wishes to upload an image attachment
28 in an email that will not be classified as spam by a detection model. In this case Δ is incredibly
29 large as it consists of all possible images which contain some semantic concept the attacker wishes to
30 upload (in this case an advertisement). To explore Δ , spammers can utilize different fonts, word
31 orientations or add random objects to the background as is the case of the adversarial example in
32 Figure 20.25 (see [Big+11] for more examples). Of course, optimization based methods may still be
33 used here to explore parts of Δ . However, in practice it may be preferable to design an adversarial
34 input by hand as this can be significantly easier to execute with only limited-query black-box access
35 to the underlying classifier.

36

37 20.8.4 Defenses based on robust optimization

38

39 As discussed in Section 20.8.3, securing a system against adversarial inputs in more general threat
40 models seems extraordinarily difficult, due to the vast space of possible adversarial inputs Δ . However,
41 there is a line of research focused on producing models which are invariant to perturbations within
42 a small constraint set $\Delta(\mathbf{x})$, with a focus on l_p -robustness where $\Delta(\mathbf{x}) = \{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\|_p < \epsilon\}$.
43 Although solving this toy threat model has little application to security settings, enforcing smoothness
44 priors have in some cases improved robustness to random image corruptions [SHS], lead to models
45 which transfer better [Sal+20], and can bias models towards different features in the data [Yin+19a].

46 Perhaps the most straightforward method for improving l_p -robustness is to directly optimize for
47

it through **robust optimization** [BTEGN09], also known as **adversarial training** [GSS15]. We define the **adversarial risk** to be

$$\min_{\theta} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p(\mathbf{x}, \mathbf{y})} \left[\max_{\mathbf{x}' \in \Delta(\mathbf{x})} L(\mathbf{x}', \mathbf{y}; \theta) \right] \quad (20.49)$$

The min max formulation in equation 20.49 poses unique challenges from an optimization perspective—it requires solving both the non-concave inner maximization and the non-convex outer minimization problems. Even worse, the inner max is NP-hard to solve in general [Kat+17]. However, in practice it may be sufficient to compute the gradient of the outer objective $\nabla_{\theta} L(\mathbf{x}_{\text{adv}}, \mathbf{y}; \theta)$ at an approximately maximal point in the inner problem $\mathbf{x}_{\text{adv}} \approx \text{argmax}_{\mathbf{x}} L(\mathbf{x}_{\text{adv}}, \mathbf{y}; \theta)$ [Mad+18]. Currently, best practice is to approximate the inner problem using a few steps of PGD.

Other methods seek to **certify** that a model is robust within a given region $\Delta(x)$. One method for certification uses randomized smoothing [CRK19]—a technique for converting a model robust to random noise into a model which is provably robust to bounded worst-case perturbations in the l_2 -metric. Another class of methods applies specifically for networks with ReLU activations, leveraging the property that the model is locally linear, and that certifying in region defined by linear constraints reduces to solving a series of linear programs, for which standard solvers can be applied [WK18].

20.8.5 Why models have adversarial examples

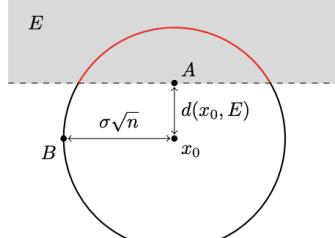
The existence of adversarial inputs is paradoxical, since modern classifiers seem to do so well on normal inputs. However, the existence of adversarial examples is a natural consequence of the general lack of robustness to distribution shift discussed in Section 20.2. To see this, suppose a model’s accuracy drops on some shifted distribution of inputs $p_{\text{te}}(\mathbf{x})$ that differs from the training distribution $p_{\text{tr}}(\mathbf{x})$; in this case, the model will necessarily be vulnerable to an adversarial attack: if errors exist, there must be a nearest such error. Furthermore, if the input distribution is high dimensional, then we should expect the nearest error to be significantly closer than errors which are sampled randomly from some out-of-distribution $p_{\text{te}}(\mathbf{x})$.

A cartoon illustration of what is going on is shown in Figure 20.26a, where \mathbf{x}_0 is the clean input image, B is an image corrupted by Gaussian noise, and A is an adversarial image. If we assume a linear decision boundary, then the error set E is a half space a certain distance from \mathbf{x}_0 . We can relate the distance to the decision boundary $d(\mathbf{x}_0, E)$ with the error rate in noise at some input \mathbf{x}_0 , denoted by $\mu = \mathbb{P}_{\delta \sim N(0, \sigma I)} [\mathbf{x}_0 + \delta \in E]$. With a linear decision boundary the relationship between these two quantities is determined by

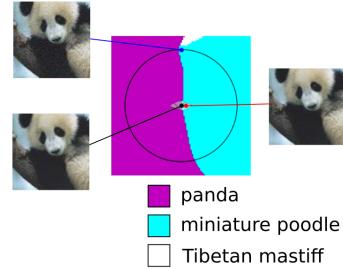
$$d(\mathbf{x}_0, E) = -\sigma \Phi^{-1}(\mu) \quad (20.50)$$

where Φ^{-1} denotes the inverse cdf of the gaussian distribution. When the input dimension is large, this distance will be significantly smaller than the distance to a randomly sampled noisy image $\mathbf{x}_0 + \delta$ for $\delta \sim N(0, \sigma I)$, as the noise term will with high probability have norm $\|\delta\|_2 \approx \sigma \sqrt{d}$. As a concrete example consider the ImageNet dataset, where $d = 224 \times 224 \times 3$ and suppose we set $\sigma = .2$. Then if the error rate in noise is just $\mu = .01$, equation 20.50 will imply that $d(\mathbf{x}_0, E) = .5$. Thus the distance to an adversarial example will be more than 100 times closer than the distance to a typical noisy images, which will be $\sigma \sqrt{d} \approx 77.6$. This phenomenon of small volume error sets being close

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47



(a)



(b)

Figure 20.26: (a) When the input dimension n is large and the decision boundary is locally linear, even a small error rate in random noise will imply the existence of small adversarial perturbations. Here, $d(\mathbf{x}_0, E)$ denotes the distance from a clean input \mathbf{x}_0 to an adversarial example (A) while the distance from \mathbf{x}_0 to a random sample $N(0; \sigma^2 I)$ (B) will be approximately $\sigma\sqrt{n}$. As $n \rightarrow \infty$ the ratio of $d(\mathbf{x}_0, A)$ to $d(\mathbf{x}_0, B)$ goes to 0. (b) A 2d slice of the InceptionV3 decision boundary through three points: a clean image (black), an adversarial example (red), and an error in random noise (blue). The adversarial example and the error in noise lie in the same region of the error set which is misclassified as ‘‘miniature poodle’’, which closely resembles a halfspace as in Figure (a). Used with kind permission of Justin Gilmer.

to most points in a data distribution $p(\mathbf{x})$ is called **concentration of measure**, and is a property common among many high dimensional data distributions [MDM19; Gil+18b].

In summary, although the existence of adversarial examples is often discussed as an unexpected phenomenon, there is nothing special about the existence of worst-case errors for ML classifiers—they will always exist as long as errors exist.

Part IV

Generation

21 Generative models: an overview

21.1 Introduction

A **generative model** is a joint probability distribution $p(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}$. In some cases, the model may be conditioned on inputs or covariates $\mathbf{c} \in \mathcal{C}$, which gives rise to a **conditional generative model** of the form $p(\mathbf{x}|\mathbf{c})$.

There are many kinds of generative model. We give a brief summary in Section 21.2, and go into more detail in subsequent chapters. See also [Tom22] for a recent book on this topic that goes into more depth.

21.2 Types of generative model

There are many kinds of generative model, some of which we list in Table 21.1. At a high level, we can distinguish between **deep generative models** (DGM) — which use deep neural network to learn a complex mapping from a single latent vector \mathbf{z} to the observed data \mathbf{x} — and more “classical” **probabilistic graphical models** (PGM), that map a set of interconnected latent variables $\mathbf{z}_1, \dots, \mathbf{z}_L$ to the observed variables $\mathbf{x}_1, \dots, \mathbf{x}_D$ using simpler, often linear, mappings. Of course, many hybrids are possible. For example, PGMs can use neural networks, and DGMs can use structured state spaces. We discuss PGMs in general terms in Chapter 4, and give examples in Chapter 29, Chapter 30, Chapter 31, Chapter 32. In this part of the book, we mostly focus on DGMs.

The main kinds of DGM are: **variational autoencoders (VAE)**, **autoregressive (AR)** models, **normalizing flows**, **diffusion models**, **energy based models (EBM)**, and **generative adversarial networks (GAN)**. We can categorize these models in terms of the following criteria (see Figure 21.1 for a visual summary):

- Density: does the model support pointwise evaluation of the probability density function $p(\mathbf{x})$, and if so, is this fast or slow, exact, approximate or a bound, etc? For **implicit models**, such as GANs, there is no well-defined density $p(\mathbf{x})$. For other models, we can only compute a lower bound on the density (VAEs), or an approximation to the density (EBMs, UGMs).
- Sampling: does the model support generating new samples, $\mathbf{x} \sim p(\mathbf{x})$, and if so, is this fast or slow, exact or approximate? Directed PGMs, VAEs and GANs all support fast sampling. However, undirected PGMs, EBMs, AR, diffusion and flows are slow for sampling.
- Training: what kind of method is used for parameter estimation? For some models (such as AR, flows and directed PGMs), we can perform exact maximum likelihood estimation (MLE), although

Model	Chapter	Density	Sampling	Training	Latents	Architecture
PGM-D	Chapter 4	Exact, fast	Fast	MLE	Optional	Sparse DAG
PGM-U	Chapter 4	SA, slow	Slow	MLE-A	Optional	Sparse graph
FA	Chapter 29	Exact, fast	Fast	MLE	\mathbb{R}^D	Linear
HMM	Chapter 30	Exact, fast	Fast	MLE	$\{1, \dots, K\}^T$	Chain
SSM-LG	Chapter 31	Exact, fast	Fast	MLE	$\mathbb{R}^{L \times T}$	Chain
SSM-NLG	Chapter 31	SA, fast	Fast	MLE-A	$\mathbb{R}^{L \times T}$	Chain
VAE	Chapter 22	LB, fast	Fast	MLE-LB	\mathbb{R}^L	Encoder-Decoder
AR	Chapter 23	Exact, fast	Slow	MLE	None	Sequential
Flows	Chapter 24	Exact, slow/fast	Slow	MLE	\mathbb{R}^D	Invertible
EBM	Chapter 25	SA, slow	Slow	MLE-A	Optional	Discriminative
Diffusion	Chapter 26	LB	Slow	MLE-LB	\mathbb{R}^D	Encoder-Decoder
GAN	Chapter 27	NA	Fast	Min-max	\mathbb{R}^L	Generator-Discriminator

Table 21.1: Characteristics of common kinds of generative model. Models above the line are “classical” probabilistic graphical models; models below the line are considered to be “deep” generative models (even though some of the models above the line can also use neural networks). Here D is the dimensionality of the observed \mathbf{x} (for time series data, we assume \mathbf{x} is $D \times T$ dimensional), and L is the dimensionality of the latent \mathbf{z} , if present. Abbreviations: MLE-A = MLE (Approximate), AR = autoregressive, EBM = Energy Based Model, FA = factor analysis, GAN = generative adversarial network, HMM = hidden Markov model, LB = lower bound, MLE = maximum likelihood estimation, MLE-LB = maximizing lower bound of the likelihood, PGM-D = directed probabilistic graphical model, PGM-U = undirected probabilistic graphical model, SA = stochastic approximation, SSM = state space model, SSM-LG = linear Gaussian SSM, SSM-NLG = non-linear and/or non-Gaussian SSM, VAE = variational autoencoder.

22

23

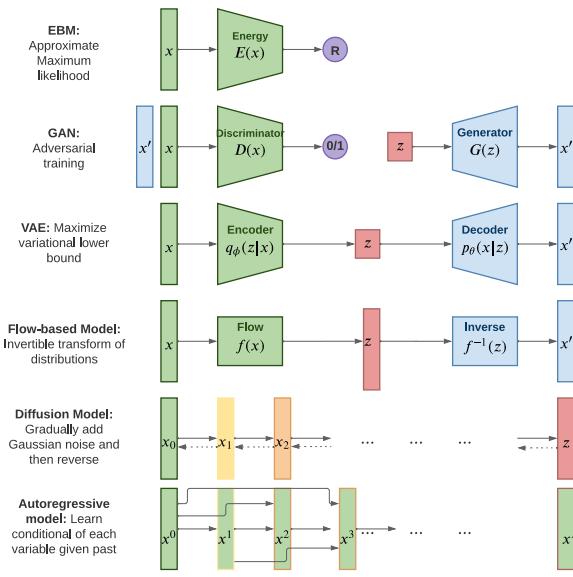


Figure 21.1: Summary of various kinds of deep generative model. Here \mathbf{x} is the observed data, \mathbf{z} is the latent code, and \mathbf{x}' is a sample from the model. AR models do not have a latent code \mathbf{z} . For diffusion models and flow models, the size of \mathbf{z} is the same as \mathbf{x} . For AR models, x^d is the d 'th dimension of \mathbf{x} . For diffusion models, \mathbf{x}_t is the t 'th “noised up version of \mathbf{x} , where $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{z} = \mathbf{x}_T$. Adapted from Figure 1 of [Wen21].

46

47



Figure 21.2: Synthetic faces from a score-based generative model (Section 25.3.5). From Figure 12 of [Son+21]. Used with kind permission of Yang Song.

the objective is usually non-convex, so we can only reach a local optimum. For other models, we cannot tractably compute the likelihood. In the case of VAEs, we maximize a lower bound on the likelihood; in the case of EBMs and UGMs, we maximize an approximation to the likelihood. For GANs we have to use min-max training, which can be unstable, and there is no clear objective function to monitor.

- Latents: does the model use a latent vector \mathbf{z} to generate \mathbf{x} or not, and if so, is it the same size as \mathbf{x} or is it a potentially compressed representation? For example, AR models do not use latents; flows and diffusion use latents, but they are not compressed. Graphical models, including EBMs, may or may not use latents.
- Architecture: what kind of neural network should we use, and are there restrictions? For flows, we are restricted to using invertible neural networks where each layer has a tractable Jacobian. For EBMs, we can use any model we like. The other models have different restrictions.

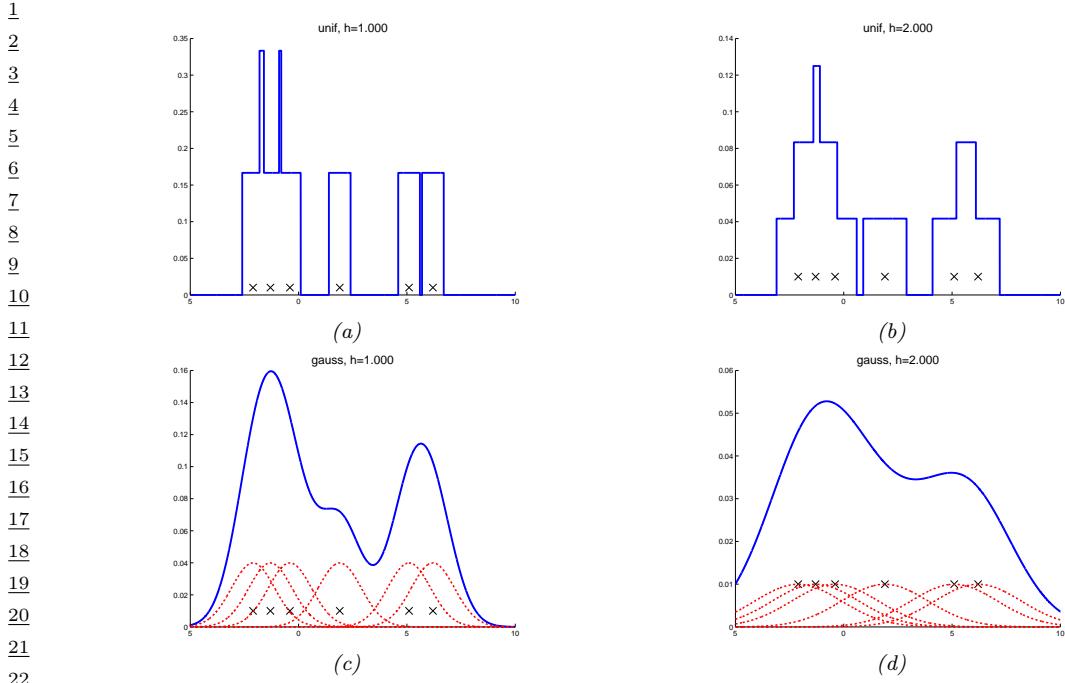
21.3 Goals of generative modeling

There are several different kinds of tasks that we can use generative models for, as we discuss below.

21.3.1 Generating data

One of the main goals of generative models is to generate (create) new data samples. For example, if we fit a model $p(\mathbf{x})$ to images of faces, we can sample new faces from it, as illustrated in Figure 21.2.¹ Similar methods can be used to create samples of text, audio, etc. When this technology is abused to make fake content, they are called **deep fakes**. For a review of this topic, see e.g., [Ngu+19].

¹ These images were made with a technique called score-based generative modeling (Section 25.3.5), although similar results can be obtained using many other techniques. See for example <https://this-person-does-not-exist.com/en> which shows results from a GAN model (Chapter 27).



23 *Figure 21.3: A nonparametric (Parzen) density estimator in 1d estimated from 6 data points, denoted by x .*
 24 *Top row: uniform kernel. Bottom row: Gaussian kernel. Left column: bandwidth parameter $h = 1$. Right*
 25 *column: bandwidth parameter $h = 2$. Adapted from http://en.wikipedia.org/wiki/Kernel_density_estimation. Generated by `parzen_window_demo2.py`.*

33 To control what is generated, it is useful to use a conditional generative model of the form $p(\mathbf{x}|\mathbf{c})$,
 34 For example, \mathbf{c} might be a text caption and \mathbf{x} might be an image, in which case $p(\mathbf{x}|\mathbf{c})$ is called a
 35 text-to-image model, which is useful for **image captioning**. Or \mathbf{c} might be a sequence of sounds
 36 and \mathbf{x} might a sequence of letters, in which case $p(\mathbf{x}|\mathbf{c})$ is called a speech-to-text model, which is
 37 useful for **automatic speech recognition**. Or \mathbf{c} might be a sequence of English words and \mathbf{x} might
 38 be a sequence of French words, in which case $p(\mathbf{x}|\mathbf{c})$ is called a sequence-to-sequence model, which is
 39 useful for **machine translation**.

40 Note that, in the conditional case, we often the inputs by \mathbf{x} and the outputs by \mathbf{y} . In this case
 41 the model has the familiar form $p(\mathbf{y}|\mathbf{x})$. In the special case that \mathbf{y} denotes a low dimensional
 42 quantity, such as a integer class label, $y \in \{1, \dots, C\}$, we get a predictive (discriminative) model.
 43 The main difference between a discriminative model and a conditional generative model is this: in a
 44 discriminative model, we assume there is one correct output, whereas in a conditional generative
 45 model, we assume there may be multiple correct outputs. This makes it harder to evaluate generative
 46 models, as we discuss in Section 21.4.

47

	Data sample	Variables			Missing values replaced by means							
		A	B	C	A	B	C					
1	1	6	6	NA	2	6	7.5					
2	2	NA	6	0	9	6	0					
3	3	NA	6	NA	9	6	7.5					
4	4	10	10	10	10	10	10					
5	5	10	10	10	10	10	10					
6	6	10	10	10	10	10	10					
7	Average		9	8	7.5	9	8					
8												
9												
10												
11												
12												
13												
14	Figure 21.4: Missing data imputation using the mean of each column.											
15												
16												
17												
18	21.3.2 Density estimation											
19	The task of density estimation refers to evaluating the probability of an observed data vector, $p = p(\mathbf{x})$. This can be useful for outlier detection (Section 20.4.2), data compression (Section 5.4), generative classifiers, model comparison, etc.											
20												
21												
22												
23	A simple approach to this problem, which works in low dimensions, is to use kernel density estimation or KDE , which has the form											
24												
25												
26	$p(\mathbf{x} \mathcal{D}) = \frac{1}{N} \sum_{n=1}^N \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n) \quad (21.1)$											
27												
28												
29	Here \mathcal{K}_h is a density kernel with bandwidth h , which is a function $\mathcal{K} : \mathbb{R} \rightarrow \mathbb{R}_+$ such that $\int \mathcal{K}(x)dx = 1$ and $\int x\mathcal{K}(x)dx = 0$. We give a 1d example of this in Figure 21.3: in the top row, we use a uniform (boxcar) kernel, and in the bottom row we use a Gaussian kernel.											
30												
31												
32	In higher dimensions, KDE suffers from the curse of dimensionality (see e.g., [AHK01]), and we need to use parametric density models $p_\theta(\mathbf{x})$ of some kind.											
33												
34												
35	21.3.3 Imputation											
36												
37	The task of imputation refers to “filling in” missing values of a data vector or data matrix. For example, suppose \mathbf{X} is an $N \times D$ matrix of data (think of a spreadsheet) in which some entries, call them \mathbf{X}_m , may be missing, while the rest, \mathbf{X}_o , are observed. A simple way to fill in the missing data is to use the mean value of each feature, $\mathbb{E}[x_d]$; this is called mean value imputation , and is illustrated in Figure 21.4. However, this ignores dependencies between the variables within each row, and does not return any measure of uncertainty.											
38												
39												
40												
41												
42												
43	We can generalize this by fitting a generative model to the observed data, $p(\mathbf{X}_o)$, and then computing samples from $p(\mathbf{X}_m \mathbf{X}_o)$. This is called multiple imputation . A generative model can be used to fill in more complex data types, such as in-painting occluded pixels in an image.											
44												
45												
46	See Section 22.3.5 for a more general discussion of missing data.											
47												

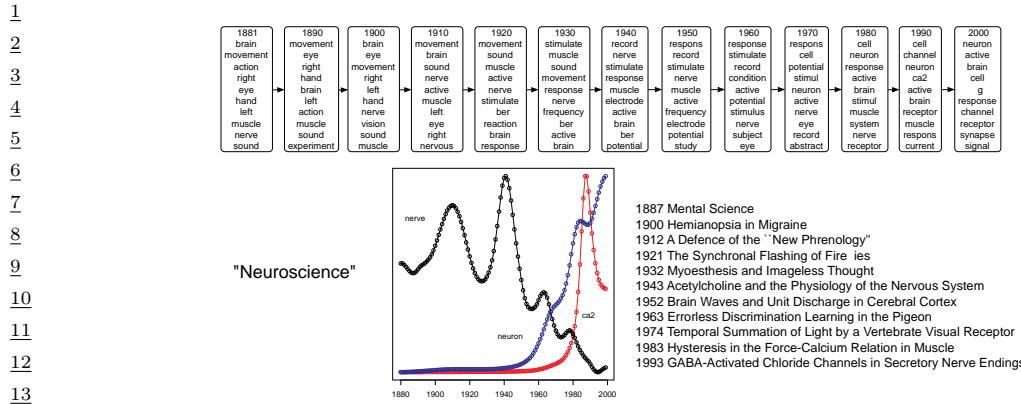


Figure 21.5: Part of the output of the dynamic topic model when applied to articles from Science. At the top, we show the top 10 words for the neuroscience topic over time. On the bottom left, we show the probability of three words within this topic over time. On the bottom right, we list paper titles from different years that contained this topic. From Figure 4 of [BL06]. Used with kind permission of David Blei.

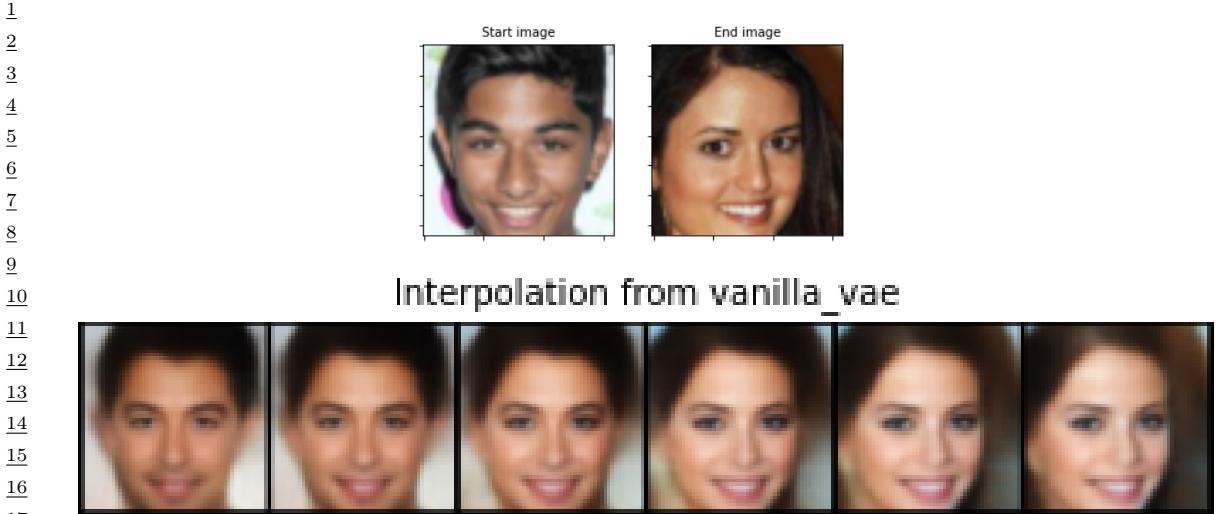
21.3.4 Structure discovery

Some kinds of generative models have latent variables \mathbf{z} , which are assumed to be the “causes” that generated the observed data \mathbf{x} . We can use Bayes rule to invert the model to compute $p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$. This can be useful for discovering latent, low-dimensional patterns in the data. We give an example of this in Figure 21.5, where we show the output of a **dynamic topic model** applied to 100 years of articles from *Science*. We see that the model has discovered various topics (groups of related words), and can track their usage over time. For details on topic models, see the supplementary material. For details on structural discovery using other kinds of latent variable models, see Part V.

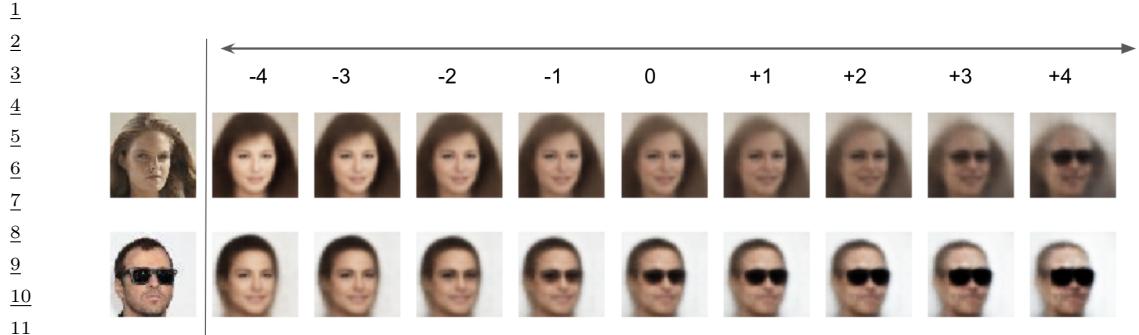
21.3.5 Latent space interpolation

One of the most interesting abilities of certain latent variable models is the ability to generate samples that have certain desired properties by interpolating between existing data points in latent space. To explain how this works, let \mathbf{x}_1 and \mathbf{x}_2 be two inputs (e.g. images), and let $\mathbf{z}_1 = e(\mathbf{x}_1)$ and $\mathbf{z}_2 = e(\mathbf{x}_2)$ be their latent encodings. (The method used for computing these will depend on the type of model; we discuss the details in later chapters.) We can regard \mathbf{z}_1 and \mathbf{z}_2 as two “anchors” in latent space. We can now generate new images that interpolate between these points by computing $\mathbf{z} = \lambda\mathbf{z}_1 + (1 - \lambda)\mathbf{z}_2$, where $0 \leq \lambda \leq 1$, and then decoding by computing $\mathbf{x}' = d(\mathbf{z})$, where $d()$ is the decoder. This is called **latent space interpolation**, and will generate data that combines semantic features from both \mathbf{x}_1 and \mathbf{x}_2 . (The justification for taking a linear interpolation is that the learned manifold often has approximately zero curvature, as shown in [SKTF18]. However, sometimes it is better to use nonlinear interpolation [MB21; Fad+20].) We give an example of this process in Figure 21.6, where we use a simple VAE (Chapter 22) fit to some face images. (Higher quality samples are possible by using other kinds of model.)

In some cases, we can go beyond interpolation, and can perform arithmetic in latent space, in



23 which we can increase or decrease the amount of a desired “semantic factor of variation”. This was
24 first shown in the **word2vec** model [Mik+13], but it also is possible in other latent variable models.
25 For example, consider our VAE model fit to **CelebA** dataset, which has faces of celebrities and some
26 corresponding attributes. Consider the attribute of wearing sunglasses. Let \mathbf{X}_i^+ be a set of images
27 which have attribute i , and \mathbf{X}_i^- be a set of images which do not have this attribute. Let \mathbf{z}_i^+ and \mathbf{z}_i^-
28 be the corresponding embeddings, and $\bar{\mathbf{z}}^+$ and $\bar{\mathbf{z}}^-$ be the average of these embeddings. We define the
29 offset vector as $\Delta = \bar{\mathbf{z}}^+ - \bar{\mathbf{z}}^-$. If we add some positive multiple of Δ to a new point \mathbf{z} , we increase
30 the amount of the sunglass factor; if we subtract some multiple of Δ , we decrease the amount of
31 the sunglass factor [Whi16]. We give an example of this in Figure 21.7. The j 'th reconstruction is
32 computed using $\hat{\mathbf{x}}_j = d(\mathbf{z} + s_j \Delta)$, where $\mathbf{z} = e(\mathbf{x})$ is the encoding of the original image, and s_j is a
33 scale factor. For the VAE, we see that we can remove sunglasses by setting $s = -4$, or make the
34 sunglasses bigger by setting $s = 4$.
35
36 **21.3.6 Representation learning**
37
38 **Representation learning** refers to learning (possibly uninterpretable) latent factors \mathbf{z} that generate
39 the observed data \mathbf{x} . The primary goal is for these features to be used in “**downstream**” supervised
40 tasks. This is discussed in Chapter 34.
41
42 **21.4 Evaluating generative models**
43
44 This section is coauthored with Mihaela Rosca, Shakir Mohamed and Balaji Lakshminarayanan.
45
46 Evaluating generative models requires metrics which capture
47



13 *Figure 21.7: Arithmetic in the latent space of a VAE . The first column is an input image, with embedding \mathbf{z} .*
 14 *Subsequent columns show the decoding of $\mathbf{z} + s\Delta$, where $s \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ and $\Delta = \bar{\mathbf{z}}^+ - \bar{\mathbf{z}}^-$*
 15 *is the difference in the average embeddings of images with or without a certain attribute (here, wearing*
 16 *sunglasses). Generated by [vae_celeba_lightning.ipynb](#).*

- 17
 18
 19 • **sample quality** - are samples generated by the model part of the data distribution?
 20
 21 • **sample diversity** - are samples from the model distribution capturing all modes of the data
 22 distribution?, and
 23
 24 • **generalization** - is the model generalizing beyond the training data?

25 There is no known metric which meets all these desiderata, but various metrics have been proposed
 26 to capture different aspects of the learned distribution, some of which we discuss below.
 27

28 21.4.1 Likelihood

29 A standard way to measure how close a model q is to a true distribution p is in terms of the KL
 30 divergence (Section 5.1):
 31

$$32 \quad D_{\text{KL}}(p\|q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} = -\mathbb{H}(p) + \mathbb{H}(p, q) \quad (21.2)$$

33 where $\mathbb{H}(p)$ is a constant, and $\mathbb{H}(p, q)$ is the cross entropy. If we approximate $p(\mathbf{x})$ by the empirical
 34 distribution, we can evaluate the cross entropy in terms of the empirical **negative log likelihood**
 35 on the dataset:

$$36 \quad \text{NLL} = -\frac{1}{N} \sum_{n=1}^N \log q(\mathbf{x}_n) \quad (21.3)$$

37 Usually we care about negative log likelihood on a held-out test set.²
 38

39
 40 2. In some applications, we report **bits per dimension**, which is the NLL using log base 2, divided by the dimensionality
 41 of \mathbf{x} . (To compute this metric, recall that $\log_2 L = \frac{\log_e L}{\log_2 e}$.)
 42

21.4.1.1 Evaluating log-likelihood

For models of discrete data, such as language models, it is easy to compute the (negative) log likelihood. However, it is common to measure performance using a quantity called **perplexity**, which is defined as 2^H , where $H = \text{NLL}$ is the cross entropy or negative log likelihood.

For image and audio models, one complication is that the model is usually a continuous distribution $p(\mathbf{x}) \geq 0$ but the data is usually discrete (e.g., $\mathbf{x} \in \{0, \dots, 255\}^D$ if we use one byte per pixel). Consequently the average log likelihood can be arbitrary large, since the pdf can be bigger than 1. To avoid this it is standard practice to use **uniform dequantization** [TOB16], in which we add uniform random noise to the discrete data, and then treat it as continuous-valued data. This gives a lower bound on the average log likelihood of the discrete model on the original data.

To see this, let \mathbf{z} be a continuous latent variable, and \mathbf{x} be a vector of binary observations computed by rounding, so $p(\mathbf{x}|\mathbf{z}) = \delta(\mathbf{x} - \text{round}(\mathbf{z}))$, computed elementwise. We have $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$. Let $q(\mathbf{z}|\mathbf{x})$ be a probabilistic inverse of \mathbf{x} , that is, it has support only on values where $p(\mathbf{x}|\mathbf{z}) = 1$. In this case, Jensen's inequality gives

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})] \quad (21.4)$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})] \quad (21.5)$$

Thus if we model the density of $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$, which is a dequantized version of \mathbf{x} , we will get a lower bound on $p(\mathbf{x})$.

21.4.1.2 Challenges with using likelihood

Unfortunately, there are several challenges with using likelihood to evaluate generative models, some of which we discuss below.

21.4.1.3 Likelihood can be hard to compute

For many models, computing the likelihood can be computationally expensive, since it requires knowing the normalization constant of the probability model. One solution is to use variational inference (Chapter 10), which provides a way to efficiently compute lower (and sometimes upper) bounds on the log likelihood. Another solution is to use annealed importance sampling (Section 11.5.4.1), which provides a way to estimate the log likelihood using Monte Carlo sampling. However, in the case of implicit generative models, such as GANs (Chapter 27), the likelihood is not even defined, so we need to find evaluation metrics that do not rely on likelihood.

21.4.1.4 Likelihood is not related to sample quality

A more subtle concern with likelihood is that it is often uncorrelated with the perceptual quality of the samples, at least for real-valued data, such as images and sound. In particular, a model can have great log-likelihood but create poor samples and vice versa.

To see why a model can have good likelihoods but create bad samples, consider the following argument from [TOB16]. Suppose q_0 is a density model for D -dimensional data \mathbf{x} which performs arbitrarily well as judged by average log-likelihood, and suppose q_1 is a bad model, such as white noise. Now consider samples generated from the mixture model

$$q_2(\mathbf{x}) = 0.01q_0(\mathbf{x}) + 0.99q_1(\mathbf{x}) \quad (21.6)$$

1 Clearly 99% of the samples will be poor. However, the log-likelihood per pixel will hardly change
2 between q_2 and q_0 if D is large, since
3

$$\log q_2(\mathbf{x}) = \log[0.01q_0(\mathbf{x}) + 0.99q_1(\mathbf{x})] \geq \log[0.01q_0(\mathbf{x})] = \log q_0(\mathbf{x}) - 100 \quad (21.7)$$

4
5 For high-dimensional data, $|\log q_0(\mathbf{x})| \sim D \gg 100$, so $\log q_2(\mathbf{x}) \approx \log q_0(\mathbf{x})$, and hence mixing in the
6 poor sampler does not significantly impact the log likelihood.
7

8 Now consider a case where the model has good samples but bad likelihoods. To achieve this,
9 suppose q is a GMM centered on the training images:
10

$$\begin{aligned} \underline{11} \\ \underline{12} \quad q(\mathbf{x}) &= \frac{1}{N} \sum_{n=1}^N \mathcal{N}(\mathbf{x} | \mathbf{x}_n, \epsilon^2 \mathbf{I}) \\ \underline{13} \\ \underline{14} \end{aligned} \quad (21.8)$$

15 If ϵ is small enough that the Gaussian noise is imperceptible, then samples from this model will look
16 good, since they correspond to the training set of real images. But this model will almost certainly
17 have poor likelihood on the test set due to overfitting. (In this case we say the model has effectively
18 just memorized the training set.)
19

20 21.4.2 Distances and divergences in feature space 21

22 Due to the challenges associated with comparing distributions in high dimensional spaces, and the
23 desire to compare distributions in a semantically meaningful way, it is common to use domain-specific
24 **perceptual distance metrics**, that measure how similar data vectors are to each other or to the
25 training data. However, most metrics used to evaluate generative models do not directly compare
26 raw data (e.g. pixels) but use a neural network to obtain features from the raw data and compare
27 the feature distribution obtained from model samples with the feature distribution obtained from
28 the dataset. The neural network used to obtain features can be trained solely for the purpose of
29 evaluation, or can be pretrained; a common choice is to use a pretrained classifier (see e.g., [Sal+16;
30 Heu+17b; Bin+18; Kyn+19; SSG18a]).

31 The **Inception Score** [Sal+16] measures the average KL divergence between the marginal distribution
32 of class labels obtained from the samples $p_{\theta}(y) = \int p(y|\mathbf{x})p_{\theta}(\mathbf{x})$ and the distribution $p(y|\mathbf{x})$
33 obtained from a sample $\mathbf{x} \sim p_{\theta}(\mathbf{x})$. This leads to the following score:
34

$$\begin{aligned} \underline{35} \quad \text{IS} &= \exp \left[\mathbf{E}_{p_{\theta}(\mathbf{x})} \mathbb{KL}(p(y|\mathbf{x}) || p_{\theta}(y)) \right] \\ \underline{36} \end{aligned} \quad (21.9)$$

37 If a model produces high quality samples from all classes in the dataset, then $p_{\theta}(y)$ should be close
38 to uniform, while $p(y|\mathbf{x})$ should be a sharp distribution corresponding to the class associated with \mathbf{x} ;
39 this leads to a high $\mathbb{KL}(p(y|\mathbf{x}) || p_{\theta}(y))$ and thus a high Inception Score score.

40 The Inception Score solely relies on class labels, and thus does not measure overfitting or sample
41 diversity outside the predefined dataset classes. For example, a model which generates one perfect
42 example per class would get a perfect Inception Score, despite not capturing the variety of examples
43 inside a class, as shown in Figure 21.8a. To address this drawback, the **Fréchet Inception Distance**
44 or **FID** score [Heu+17b] measures the Fréchet distance between two Gaussian distributions on sets
45 of features of a pre-trained classifier. One Gaussian is obtained by passing model samples through a
46 pretrained classifier, and the other by passing samples the dataset through the same classifier. If we
47

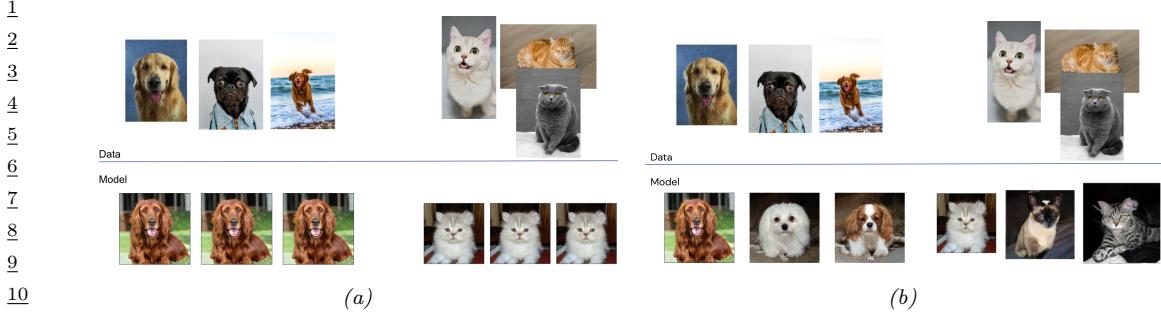


Figure 21.8: (a) Model samples with good (high) inception score are visually realistic. (b) Model samples with good (low) FID score are visually realistic and diverse.

assume that the mean and covariance obtained from model features are μ_m and Σ_m and those from the data are μ_d and Σ_d , then the FID is

$$\text{FID} = \|\mu_m - \mu_d\|_2^2 + \text{trace}\left(\Sigma_d + \Sigma_m - 2(\Sigma_d \Sigma_m)^{1/2}\right) \quad (21.10)$$

Since it uses features instead of class logits, the Fréchet distance captures more than modes captured by class labels, as shown in Figure 21.8b. Unlike the Inception score, a lower score is better since we want the two distributions to be as close as possible.

Unfortunately, the Fréchet distance has been shown to have a high bias, with results varying widely based on the number of samples used to compute the score. To mitigate this issue, the **Kernel Inception Distance** has been introduced [Bin+18], which measures the squared MMD (Section 2.9.3) between the features obtained from the data and features obtained from model samples.

21.4.3 Precision and recall metrics

Since the FID only measures the distance between the data and model distributions, it is difficult to use it as a diagnostic tool: a bad (high) FID can indicate that the model is not able to generate high quality data, or that it puts too much mass around the data distribution (e.g. in Figure 27.7a), or that the model only captures a subset of the data (e.g. in Figure 27.7d). Trying to disentangle between these two failure modes has been the motivation to seek individual precision (sample quality) and recall (sample diversity) metrics in the context of generative models [LPO17; Kyn+19]. (The diversity question is especially important in the context of GANs, where mode collapse (Section 27.3.3) can be an issue.)

A common approach taken is to use nearest neighbors in the feature space of a pretrained classifier to define precision and recall [Kyn+19]. To formalize this, let us define

$$f_k(\phi, \Phi) = \begin{cases} 1 & \text{if } \exists \phi' \in \Phi \text{ s.t. } \|\phi - \phi'\|_2^2 \leq \|\phi' - \text{NN}_k(\phi', \Phi)\|_2^2 \\ 0 & \text{otherwise} \end{cases} \quad (21.11)$$

where $\text{NN}_k(\phi', \Phi)$ is a function returning the k -th nearest neighbor of ϕ' in Φ , where Φ is a set of

¹ feature vectors. We now define precision and recall as follows.

$$\text{precision}(\Phi_{model}, \Phi_{data}) = \frac{1}{|\Phi_{model}|} \sum_{\phi \in \Phi_{model}} f_k(\phi, \Phi_{data}); \quad (21.12)$$

$$\text{recall}(\Phi_{model}, \Phi_{data}) = \frac{1}{|\Phi_{data}|} \sum_{\phi \in \Phi_{data}} f_k(\phi, \Phi_{model}); \quad (21.13)$$

Precision and recall are always between 0 and 1. Intuitively, the precision metric measures whether samples are as close to data as data is to other data examples, while recall measures whether data is as close to model samples as model samples are to other samples. The parameter k controls how lenient the metrics will be – the higher k , the higher both precision and recall will be. As in classification, precision and recall in generative models can be used to construct a trade-off curve between different models which allow practitioners to make an informed decision regarding which model they want to use.

21.4.4 Statistical tests

Statistical tests have been long used to determine whether two set of samples have been generated from the same distribution; these types of statistical tests are called **two sample tests**. Let us define the null hypothesis to represent that the set of samples are from the same distribution. We then compute a statistic from the data and compare it to a threshold, and based on this we decide whether to reject the null hypothesis. In the context of evaluating implicit generative models such as GANs, statistics based on classifiers [Saj+18] and the MMD [Liu+20b] have been used. To adapt to the high dimensional input spaces, which are ubiquitous in the era of deep learning, two sample tests have been adapted to use learned features instead of raw data.

Like all other evaluation metrics for generative models, statistical tests have their own advantages and disadvantages: while users can specify Type 1 error – the chance they allow that the null hypothesis is wrongly rejected – statistical tests tend to be computationally expensive and thus cannot be used to monitor progress in training, but rather ought just to be used to compare fully trained models.

21.4.5 Challenges with using pretrained classifiers

While popular and convenient, evaluation metrics that rely on pretrained classifiers (such as IS, FID, nearest neighbors in feature space, and statistical tests in feature space) have significant drawbacks. One might not have a pretrained classifier available for the dataset at hand, so classifiers trained on other datasets are used. Given the well known challenges with neural network generalization (see Section 17.5), the features of a classifier trained on images from one dataset might not be reliable enough to provide a fine grained signal of quality for samples obtained from a model trained on a different dataset. If the generative model is trained on the same dataset as the pre-trained classifier but the model is not capturing the data distribution perfectly, we are presenting the pre-trained classifier with out-of-distribution data and relying on its features to obtain score to evaluate our models. Far from being purely theoretical concerns, these issues have been studied extensively and have been shown to affect evaluation in practice [RV19; BS18].



Figure 21.9: Illustration of nearest neighbors in feature space: in the top left we have the query sample generated using BigGAN, and the rest of the images are its nearest neighbors from the dataset. The nearest neighbors search is done in the feature space of a pretrained classifier. From Figure 13 of [BDS18]. Used with kind permission of Andy Brock.

21.4.6 Using model samples to train classifiers

Instead of using pretrained classifiers to evaluate samples, one can *train* a classifier on samples from conditional generative models , and then see how good these classifiers are at classifying data. For example, does adding synthetic (sampled) data to the real data help? This is closer to a reliable evaluation of generative model samples, since ultimately, the performance of generative models is dependent on the downstream task they are trained for. If used for semi supervised learning, one should assess how much adding samples to a classifier dataset helps with test accuracy. If used for model based reinforcement learning, one should assess how much the generative model helps with agent performance. For examples of this approach, see e.g., [SSM18; SSA18; RV19; SS20].

21.4.7 Assessing overfitting

Many of the metrics discussed so far capture the sample quality and diversity, but do not capture overfitting to the training data. To capture overfitting, often a visual inspection is performed: a set of samples is generated from the model and for each sample its closest K nearest neighbors in the feature space of a pretrained classifier are obtained from the dataset. While this approach requires manually assessing samples, it is a simple way to test whether a model is simply memorizing the data. We show an example in Figure 21.9: since the model sample in the top left is quite different than its neighbors from the dataset (remaining images), we can conclude the sample is not simply memorised from the dataset. Similarly, sample diversity can be measured by approximating the support of the

1 learned distribution by looking for similar samples in a large sample pool — as in the pigeonhole
2 principle — but it is expensive and often requires manual human assessment[AZ17].

3 For likelihood-based models — such as variational autoencoders Chapter 22, autoregressive
4 models Chapter 23, and normalising flows Chapter 24 — we can assess memorisation by seeing how
5 much the log-likelihood of a model changes when a sample is included in the model’s training set or
6 not [BW21].

7

8 **21.4.8 Human evaluation**

9 One approach to evaluate generative models is to use human evaluation, by presenting samples
10 from the model along side samples from the data distribution, and ask human raters to compare
11 the quality of the samples [Zho+19b]. Human evaluation is a suitable metric if the model is used
12 to create art or other data for human display, or if reliable automated metrics are hard to obtain.
13 However, human evaluation can be difficult to standardize, hard to automate and can be expensive
14 or cumbersome to set up.

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

22 Variational autoencoders

22.1 Introduction

In this chapter, we discuss generative models of the form

$$\mathbf{z} \sim p_{\theta}(\mathbf{z}) \tag{22.1}$$

$$\mathbf{x}|\mathbf{z} \sim \text{Expfam}(\mathbf{x}|\eta) \tag{22.2}$$

where $p(\mathbf{z})$ is some kind of prior on the latent code \mathbf{z} , $d_{\theta}(\mathbf{z})$ is a deep neural network, known as the **decoder**, and $\text{Expfam}(\mathbf{x}|\eta)$ is an exponential family distribution, such as a Gaussian or product of Bernoullis. This is called a **deep latent variable model** or **DLVM**. When the prior is Gaussian (as is often the case), this model is called a **deep latent Gaussian model** or **DLGM**.

Posterior inference (i.e., computing $p_{\theta}(\mathbf{z}|\mathbf{x})$) is computationally intractable, as is computing the marginal likelihood

$$p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z}) d\mathbf{z} \tag{22.3}$$

Hence we need to resort to approximate inference. For most of this chapter, we will use **amortized inference**, which we discussed in Section 10.3.7. This trains another model, $q_{\phi}(\mathbf{z}|\mathbf{x})$, called the **recognition network** or **inference network**, simultaneously with the generative model to do approximate posterior inference. This combination is called a **variational autoencoder** or **VAE** [KW14; RMW14b; KW19a], since it can be thought of as a probabilistic version of a deterministic autoencoder.

In this chapter, we introduce the basic VAE, as well as some extensions. Note that the literature on VAE-like methods is vast¹, so we will only discuss a small subset of the ideas that have been explored.

22.2 VAE basics

In this section, we discuss the basics of variational autoencoders.

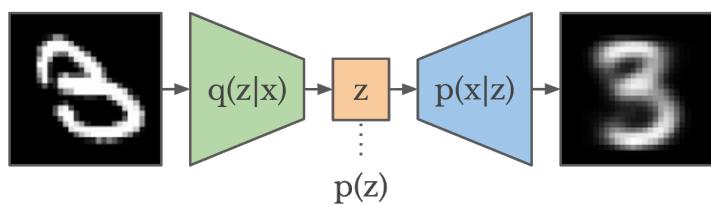


Figure 22.1: Schematic illustration of a VAE. From a figure in [Haf18]. Used with kind permission of Danijar Hafner.

22.2.1 Modeling assumptions

In the simplest setting, a VAE defines a generative model of the form

$$p_{\theta}(z, x) = p_{\theta}(z)p_{\theta}(x|z) \quad (22.4)$$

where $p_{\theta}(z)$ is usually a Gaussian, and $p_{\theta}(x|z)$ is usually a product of exponential family distributions (e.g., Gaussians or Bernoullis), with parameters computed by a neural network decoder, $d_{\theta}(z)$. For example, for binary observations, we can use

$$p_{\theta}(x|z) = \prod_{d=1}^D \text{Ber}(x_d | \sigma(d_{\theta}(z))) \quad (22.5)$$

In addition, a VAE fits a recognition model

$$q_{\phi}(z|x) = q(z|e_{\phi}(x)) \approx p_{\theta}(z|x) \quad (22.6)$$

to perform approximate posterior inference. Here $q_{\phi}(z|x)$ is usually a Gaussian, with parameters computed by a neural network encoder $e_{\phi}(x)$:

$$q_{\phi}(z|x) = \mathcal{N}(z|\mu, \text{diag}(\exp(\ell))) \quad (22.7)$$

$$(\mu, \ell) = e_{\phi}(x) \quad (22.8)$$

where $\ell = \log \sigma$. The model can be thought of as encoding the input x into a stochastic latent bottleneck z and then decoding it to approximately reconstruct the input, as shown in Figure 22.1.

The idea of training an inference network to “invert” a generative network, rather than running an optimization algorithm to infer the latent code, is called amortized inference, and is discussed in Section 10.3.7. This idea was first proposed in the **Helmholtz machine** [Day+95]. However, that paper did not present a single unified objective function for inference and generation, but instead used the wake sleep (Section 22.7) method for training. By contrast, the VAE optimizes a variational lower bound on the log-likelihood, which means that convergence to a locally optimal MLE of the parameters is guaranteed.

We can use other approaches to fitting the DLGM (see e.g., [Hof17; DF19]). However, learning an inference network to fit the DLGM is often faster and can have some regularization benefits (see e.g., [KP20]). Combining a generative model with an inference model in this way results in what Jacob Andreas, in his blog, called a **monference**, i.e., model-inference hybrid.²

⁴⁵ 1. For example, the website <https://github.com/matthewvowels1/Awesome-VAEs> lists over 900 papers.

⁴⁶ 2. See <http://blog.jacobandreas.net/monference.html>.

22.2.2 Evidence lower bound

When fitting the model, our goal is to maximize the marginal likelihood

$$p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|z)p_{\theta}(z) dz \quad (22.9)$$

Unfortunately, computing this quantity is intractable, because if we could compute it, we could then easily compute the posterior as follows:

$$p_{\theta}(z|x) = \frac{p_{\theta}(z, x)}{p_{\theta}(x)} \quad (22.10)$$

(Note that the numerator, $p_{\theta}(z, x)$, is always tractable in a directed graphical model.) For a DLVM, $p_{\theta}(z|x)$ is intractable. However, we can use the inference network to compute an approximate posterior, $q_{\phi}(z|x)$, and hence a lower bound to the marginal likelihood. This idea is discussed in Section 10.1.2, but we repeat the argument here, using slightly different notation.

First note that we have the following decomposition:

$$\log p_{\theta}(\mathbf{x}) = \mathbb{E}_{q(z|x)} [\log p_{\theta}(\mathbf{x})] \quad (22.11)$$

$$= \mathbb{E}_{q(z|x)} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, z)}{p_{\theta}(z|x)} \right) \right] \quad (22.12)$$

$$= \mathbb{E}_{q(z|x)} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, z)}{q_{\phi}(z|x)} \frac{q_{\phi}(z|x)}{p_{\theta}(z|x)} \right) \right] \quad (22.13)$$

$$= \underbrace{\mathbb{E}_{q(z|x)} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, z)}{q_{\phi}(z|x)} \right) \right]}_{\mathcal{L}_{\theta,\phi}(\mathbf{x})} + \underbrace{\mathbb{E}_{q(z|x)} \left[\log \left(\frac{q_{\phi}(z|x)}{p_{\theta}(z|x)} \right) \right]}_{D_{\text{KL}}(q(z|x) \| p_{\theta}(z|x))} \quad (22.14)$$

The second term in Equation (22.14) is non-negative, and hence

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) \leq \log p_{\theta}(\mathbf{x}) \quad (22.15)$$

The quantity $\log p_{\theta}(\mathbf{x})$ is the log marginal likelihood, also called the **evidence**. Hence $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ is called the **evidence lower bound** or **ELBO**. Our goal is to *maximize* this quantity. (Thus we use the symbol \mathcal{L} rather than \mathcal{L} , since the latter denotes a loss we want to minimize.)

We can rewrite the ELBO as follows:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q(z|x)} [\log p_{\theta}(\mathbf{x}, z) - q_{\phi}(z|x)] \quad (22.16)$$

$$= \mathbb{E}_{q(z|x)} [\log p_{\theta}(\mathbf{x}|z) + \log p_{\theta}(z) - q_{\phi}(z|x)] \quad (22.17)$$

$$= \mathbb{E}_{q(z|x)} [\log p_{\theta}(\mathbf{x}|z)] - D_{\text{KL}}(q_{\phi}(z|x) \| p_{\theta}(z)) \quad (22.18)$$

We can interpret this objective as the expected log likelihood plus a regularization term, that ensures the (per-sample) posterior is “well behaved” (does not deviate too far from the prior in terms of KL divergence).

The tightness of this lower bound is controlled by the **variational gap**, which is given by $D_{\text{KL}}(q_{\phi}(z|x) \| p_{\theta}(z|x))$. A better approximate posterior results in a tighter bound. When the KL goes to zero, the posterior is exact, so any improvements to the ELBO directly translate to improvements in the likelihood of the data, as in the EM algorithm (see Section 6.7.3).

22.2.3 Optimization

The ELBO for a single datapoint \mathbf{x} is given in Equation (22.18). The ELBO for the whole dataset is just the sum of these terms:

$$\mathbb{L}_{\theta, \phi}(\mathcal{D}) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} \mathbb{L}_{\theta, \phi}(\mathbf{x}) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))]] \quad (22.19)$$

where $N = |\mathcal{D}|$ is the number of examples. Our goal is to *maximize* this wrt θ and ϕ using stochastic gradient ascent. Alternatively, we can try to *minimize* the **negative ELBO**:

$$\mathcal{L}(\theta, \phi) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [-\log p_{\theta}(\mathbf{x}|\mathbf{z})] + D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))]] \quad (22.20)$$

We can create an unbiased minibatch approximation of this objective by sampling examples \mathbf{x} , and then computing the objective for a given \mathbf{x} . So now we focus on a fixed \mathbf{x} , for brevity.

If we assume $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ and $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$ we can use Equation (5.67) to compute the KL in closed form:

$$D_{\text{KL}}(q \| p) = -\frac{1}{2} \sum_{k=1}^K [\log \sigma_k^2 - \sigma_k^2 - \mu_k^2 + 1] \quad (22.21)$$

This just leaves us with the expected log likelihood term. We can approximate this by sampling $\mathbf{z}^s \sim q_{\phi}(\mathbf{z}|\mathbf{x})$, to get

$$\mathbb{L}_{\theta, \phi}(\mathbf{x}) = \log p_{\theta}(\mathbf{x}|\mathbf{z}^s) - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z})) \quad (22.22)$$

It is easy to take gradients of this wrt θ , using automatic differentiation. Unfortunately taking gradients wrt ϕ is harder, since we need to take into account that the sampling process itself depends on ϕ . We discuss a solution to this in Section 22.2.4.

22.2.4 The reparameterization trick

In this section, we discuss how to compute gradients of the (single sample) ELBO

$$\mathbb{L}_{\theta, \phi}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z}) + \log p_{\theta}(\mathbf{z}) - q_{\phi}(\mathbf{z}|\mathbf{x})] \quad (22.23)$$

(We write it this way, rather than in terms of a KL penalty, since for non-Gaussian distributions, the KL will be hard to compute.)

The gradient wrt the generative parameters θ is easy to compute, since we can push gradients inside the expectation, and use a single Monte Carlo sample:

$$\nabla_{\theta} \mathbb{L}_{\theta, \phi}(\mathbf{x}) = \nabla_{\theta} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}, \mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})] \quad (22.24)$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\nabla_{\theta} \{\log p_{\theta}(\mathbf{x}, \mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x})\}] \quad (22.25)$$

$$\approx \nabla_{\theta} \log p_{\theta}(\mathbf{x}, \mathbf{z}^s) \quad (22.26)$$

where $\mathbf{z}^s \sim q_{\phi}(\mathbf{z}|\mathbf{x})$. This is an unbiased estimate of the gradient, so can be used with SGD.

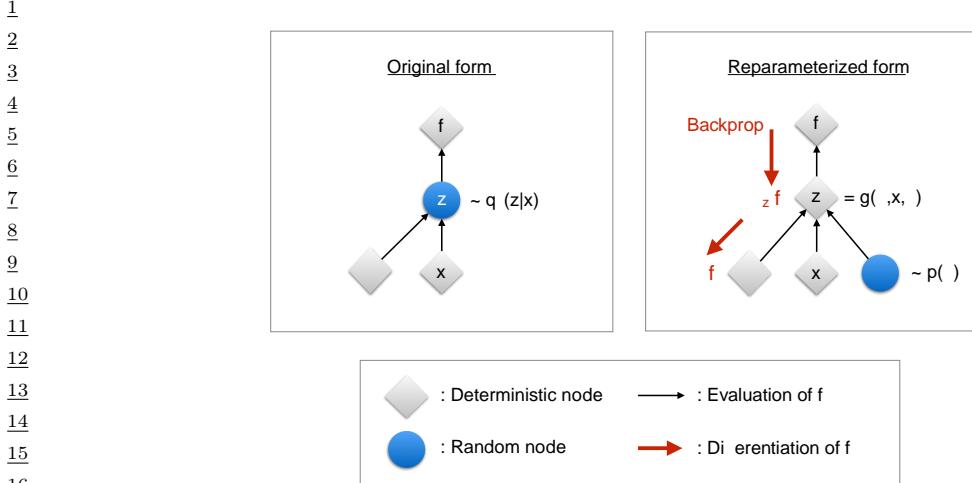


Figure 22.2: Illustration of the reparameterization trick. The objective f depends on the variational parameters ϕ , the observed data x , and the latent random variable $z \sim q_\phi(z|x)$. On the left, we show the standard form of the computation graph. On the right, we show a reparameterized form, in which we move the stochasticity into the noise source ϵ , and compute z deterministically, $z = f(\phi, x, \epsilon)$. The rest of the graph is deterministic, so we can backpropagate the gradient of the scalar f wrt ϕ through z and into ϕ . From Figure 2.3 of [KW19a]. Used with kind permission of Durk Kingma.

The gradient wrt the inference parameters ϕ is harder to compute since

$$\nabla_\phi \mathcal{L}_{\theta, \phi}(x) = \nabla_\phi \mathbb{E}_{q(z|x)} [\log p_\theta(x, z) - \log q_\phi(z|x)] \quad (22.27)$$

$$\neq \mathbb{E}_{q(z|x)} [\nabla_\phi \{\log p_\theta(x, z) - \log q_\phi(z|x)\}] \quad (22.28)$$

However, for continuous latent variables z , we can use the **reparameterization trick** to make the randomness independent of ϕ , which lets us pass gradients through. We explain this in detail in Section 6.6.4, but we summarize the basic idea here.

The key trick is to rewrite the random variable $z \sim q_\phi(z|x)$ as some differentiable (and invertible) transformation r of another random variable $\epsilon \sim p(\epsilon)$, which does not depend on ϕ , i.e., we assume we can write

$$z = r(\epsilon, \phi, x) \quad (22.29)$$

For example,

$$z \sim \mathcal{N}(\mu, \text{diag}(\sigma)) \iff z = \mu + \epsilon \odot \sigma, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (22.30)$$

Using this, we have

$$\mathbb{E}_{q(z|x)} [f(z)] = \mathbb{E}_{p(\epsilon)} [f(z)] \quad \text{s.t. } z = r(\epsilon, \phi, x) \quad (22.31)$$

where we define

$$f(z) = \log p_\theta(x, z) - \log q_\phi(z|x) \quad (22.32)$$

1 Hence

2

$$\nabla_{\phi} \mathbb{E}_{q(z|x)} [f(z)] = \nabla_{\phi} \mathbb{E}_{p(\epsilon)} [f(z)] = \mathbb{E}_{p(\epsilon)} [\nabla_{\phi} f(z)] \quad (22.33)$$

3

4 which we can approximate with a single Monte Carlo sample. This lets us propagate gradients back
5 through the f function and then into the DNN transformation function r that is used to compute
6 $z = r(\epsilon, \phi, x)$. See Figure 22.2 for an illustration.

7

22.2.5 Computing the reparameterized ELBO

8 Since we are now working with the random variable ϵ , we need to use the change of variables formula
9 to compute

10

$$\log q_{\phi}(z|x) = \log p(\epsilon) - \log \left| \det \left(\frac{\partial z}{\partial \epsilon} \right) \right| \quad (22.34)$$

11

12 where $\frac{\partial z}{\partial \epsilon}$ is the Jacobian:

13

$$\frac{\partial z}{\partial \epsilon} = \begin{pmatrix} \frac{\partial z_1}{\partial \epsilon_1} & \dots & \frac{\partial z_1}{\partial \epsilon_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial \epsilon_1} & \dots & \frac{\partial z_k}{\partial \epsilon_k} \end{pmatrix} \quad (22.35)$$

14

15 We design the transformation $z = r(\epsilon)$ such that this Jacobian is tractable to compute. We give
16 some examples below.

17

22.2.5.1 Fully factorized Gaussian

18 Suppose we have a fully factorized Gaussian posterior:

19

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (22.36)$$

20

21

$$z = \mu + \sigma \odot \epsilon \quad (22.37)$$

22

$$(\mu, \log \sigma) = e_{\phi}(x) \quad (22.38)$$

23 Then the Jacobian is

24

$$\frac{\partial z}{\partial \epsilon} = \text{diag}(\sigma) \quad (22.39)$$

25

26 so

27

$$\log q_{\phi}(z|x) = \sum_{k=1}^K \log \mathcal{N}(\epsilon_k | 0, 1) - \log \sigma_k = \sum_{k=1}^K -\frac{1}{2} \log(2\pi) - \frac{1}{2} \epsilon_k^2 - \log \sigma_k \quad (22.40)$$

28

29

22.2.5.2 Full covariance Gaussian

30 Now consider a full covariance Gaussian posterior:

31

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (22.41)$$

32

33

$$z = \mu + \mathbf{L}\epsilon \quad (22.42)$$

34

35

where \mathbf{L} is a lower triangular matrix with non-zero entries on the diagonal, which satisfies

$$\Sigma = \mathbf{L}\mathbf{L}^\top \quad (22.43)$$

The Jacobian of this affine transformation is

$$\frac{\partial \mathbf{z}}{\partial \epsilon} = \mathbf{L} \quad (22.44)$$

Since \mathbf{L} is a triangular matrix, its determinant is the product of its diagonals, so

$$\log |\det \frac{\partial \mathbf{z}}{\partial \epsilon}| = \sum_{k=1}^K \log |L_{kk}| \quad (22.45)$$

We need to make the parameters of the transformation r be a function of the inputs \mathbf{x} . One way to do this is to define

$$(\mu, \log \sigma, \mathbf{L}') = e_\phi(\mathbf{x}) \quad (22.46)$$

$$\mathbf{L} = \mathbf{M} \odot \mathbf{L}' + \text{diag}(\sigma) \quad (22.47)$$

where \mathbf{M} is a masking matrix with 0s on and above the diagonal, and 1s below the diagonal. With this construction, the diagonal entries of \mathbf{L} are given by σ , so

$$\log |\det \frac{\partial \mathbf{z}}{\partial \epsilon}| = \sum_{k=1}^K \log |L_{kk}| = \sum_{k=1}^K \log \sigma_k \quad (22.48)$$

See Algorithm 26 for the corresponding pseudo code for computing the reparameterized ELBO.

Algorithm 26: Computing a single sample unbiased estimate of the reparameterized ELBO for a VAE with full covariance Gaussian posterior, and factorized Bernoulli likelihood. Based on Algorithm 2 of [KW19a].

```

1  $(\mu, \log \sigma, \mathbf{L}') = e_\phi(\mathbf{x})$  ;
2  $\mathbf{M} = \text{np.triu}(\text{np.ones}(K), -1)$  ;
3  $\mathbf{L} = \mathbf{M} \odot \mathbf{L}' + \text{diag}(\sigma)$  ;
4  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  ;
5  $\mathbf{z} = \mathbf{L}\epsilon + \mu$  ;
6  $\mathbf{p} = d_\theta(\mathbf{z})$  ;
7  $\mathcal{L}_{\log qz} = -\sum_{k=1}^K [\frac{1}{2}\epsilon_k^2 + \frac{1}{2}\log(2\pi) + \log \sigma_k]$  // from  $q_\phi(\mathbf{z}|\mathbf{x})$  ;
8  $\mathcal{L}_{\log p_z} = -\sum_{k=1}^K [\frac{1}{2}z_k^2 + \frac{1}{2}\log(2\pi)]$  // from  $p_\theta(\mathbf{z})$  ;
9  $\mathcal{L}_{\log p_x} = -\sum_{d=1}^D [x_d \log p_d + (1-x_d) \log(1-p_d)]$  // from  $p_\theta(\mathbf{x}|\mathbf{z})$  ;
10  $\mathcal{L} = \mathcal{L}_{\log p_x} + \mathcal{L}_{\log p_z} - \mathcal{L}_{\log qz}$ 
```

22.2.5.3 Inverse autoregressive flows

In Section 10.4.3, we discuss how to use inverse autoregressive flows to learn more expressive posteriors $q_\phi(\mathbf{z}|\mathbf{x})$, leveraging the tractability of the Jacobian of this nonlinear transformation.

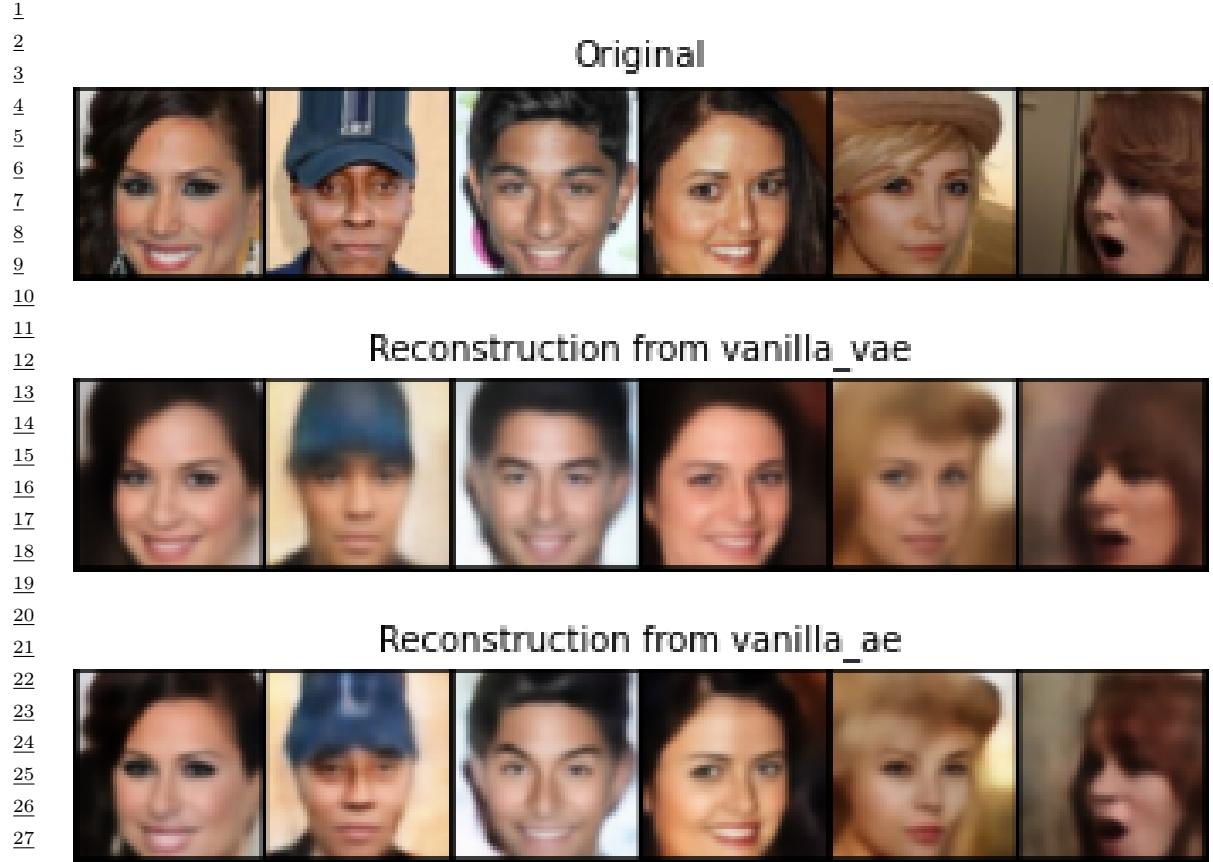


Figure 22.3: Illustration of image reconstruction. (a) Original images. (b) Variational autoencoder. (c) Deterministic autoencoder. Both models have the same convolutional structure with 256 latent dimensions. Both are trained for 30 epochs with a batch size of 256. Generated by [vae_compare_results.ipynb](#).

22.2.6 Comparison of VAEs and autoencoders

VAEs are very similar to deterministic autoencoders (DAE). In particular, the generative model, $p_{\theta}(\mathbf{x}|\mathbf{z})$ acts like the decoder, and the inference network, $q_{\phi}(\mathbf{z}|\mathbf{x})$, acts like the encoder. To illustrate this, we fit a a convolutional VAE and DAE to the **CelebA** dataset [Liu+15].³ In Figure 22.3, we see that both DAE and VAE can reconstruct the input images reasonably well, although the VAE reconstructions are somewhat more blurry, for reasons we discuss in Section 22.3.1.

The main advantage of a VAE over a deterministic autoencoder is that it defines a proper generative model, that can create sensible-looking novel images by decoding prior samples $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. By contrast, an autoencoder only knows how to decode latent codes derived from the training set, not

³ 3. CelebA contains about 200k images of famous celebrities. The images are also annotated with 40 attributes. We reduce the resolution of the images to 64x64, as is conventional.

12**Samples from vanilla_vae**3456789101112**Samples from vanilla_ae**13141516171819202122

Figure 22.4: Illustration of image sampling. (a) Variational autoencoder. (b) Deterministic autoencoder. Generated by `vae_compare_results.ipynb`.

232425

novel samples. This is illustrated in Figure 22.4.

2627

22.2.7 VAEs optimize in an augmented space

2829

In this section, we derive several alternative expressions for the ELBO which shed light on how VAEs work.

30

First, let us define the joint generative distribution

313233

$$p_{\theta}(\mathbf{x}, \mathbf{z}) = p_{\theta}(\mathbf{z})p_{\theta}(\mathbf{x}|\mathbf{z}) \quad (22.49)$$

3435

and the joint inference distribution

3637

$$q_{\phi}(\mathbf{z}, \mathbf{x}) = p_{\mathcal{D}}(\mathbf{x})q_{\phi}(\mathbf{z}|\mathbf{x}) \quad (22.50)$$

3839

where

404142

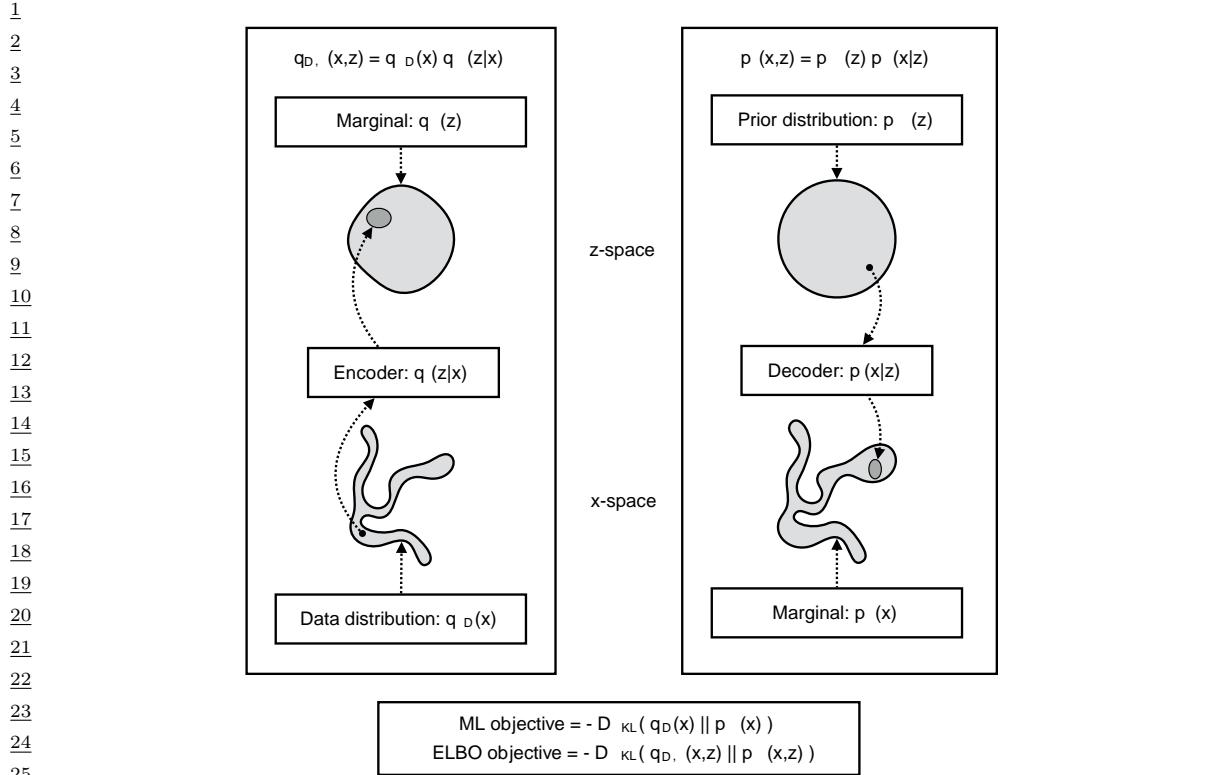
$$p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \delta(\mathbf{x}_n - \mathbf{x}) \quad (22.51)$$

4344

is the empirical distribution. Let us also define the data marginal

454647

$$p_{\theta}(\mathbf{x}) = \int_{\mathbf{z}} p_{\theta}(\mathbf{x}, \mathbf{z}) d\mathbf{z} \quad (22.52)$$



26 *Figure 22.5: The maximum likelihood (ML) objective can be viewed as the minimization of*
27 $D_{KL}(p_D(\mathbf{x}) \parallel p_\theta(\mathbf{x}))$, while the ELBO objective is minimization of $D_{KL}(q_{D,\phi}(\mathbf{x}, \mathbf{z}) \parallel p_\theta(\mathbf{x}, \mathbf{z}))$, which upper
28 bounds $D_{KL}(q_D(\mathbf{x}) \parallel p_\theta(\mathbf{x}))$. From Figure 2.4 of [KW19a]. Used with kind permission of Durk Kingma.

29

30

31 and the inference marginal, also called the **aggregated posterior**:

32

$$q_\phi(\mathbf{z}) = \int_{\mathbf{x}} q_\phi(\mathbf{x}, \mathbf{z}) d\mathbf{x} \quad (22.53)$$

33

34 Finally, we define the conditionals $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}, \mathbf{z})/p_\theta(\mathbf{x})$ and $q_\phi(\mathbf{x}|\mathbf{z}) = q_\phi(\mathbf{x}, \mathbf{z})/q_\phi(\mathbf{z})$. See
35 Figure 22.5 for a visual illustration.

36 Having defined our terms, we can now derive various alternative versions of the ELBO, following
37 [ZSE19]. First note that

$$\mathcal{L} = \mathbb{E}_{p_D(\mathbf{x})} [\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]] - \mathbb{E}_{p_D(\mathbf{x})} [D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}))] \quad (22.54)$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \quad (22.55)$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{x}, \mathbf{z})} + \log p_D(\mathbf{x}) \right] \quad (22.56)$$

$$= -D_{KL}(q_\phi(\mathbf{x}, \mathbf{z}) \parallel p_\theta(\mathbf{x}, \mathbf{z})) + \mathbb{E}_{p_D(\mathbf{x})} [\log p_D(\mathbf{x})] \quad (22.57)$$

46

47

If we define $\stackrel{c}{=}$ to mean equal up to additive constants, we can rewrite the above as

$$\mathcal{L} \stackrel{c}{=} -D_{\text{KL}}(q_{\phi}(\mathbf{x}, \mathbf{z}) \| p_{\theta}(\mathbf{x}, \mathbf{z})) \quad (22.58)$$

Now note that, from the chain rule for KL divergence,

$$D_{\text{KL}}(q_{\mathcal{D}, \phi}(\mathbf{x}, \mathbf{z}) \| p_{\theta}(\mathbf{x}, \mathbf{z})) = D_{\text{KL}}(q_{\mathcal{D}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) + \mathbb{E}_{q_{\mathcal{D}}(\mathbf{x})}[D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}|\mathbf{x}))] \quad (22.59)$$

Hence

$$\mathcal{L} \stackrel{c}{=} -D_{\text{KL}}(p_{\mathcal{D}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}[D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}|\mathbf{x}))] \quad (22.60)$$

Thus maximizing the ELBO requires minimizing the two KL terms. The first KL term is minimized by MLE, and the second KL term is minimized by fitting the true posterior. Thus if the posterior family is limited, there may be a conflict between these objectives.

Finally, we note that the ELBO can also be written as

$$\mathcal{L} \stackrel{c}{=} -D_{\text{KL}}(q_{\phi}(\mathbf{z}) \| p_{\theta}(\mathbf{z})) - \mathbb{E}_{q(\mathbf{z})}[D_{\text{KL}}(q_{\phi}(\mathbf{x}|\mathbf{z}) \| p_{\theta}(\mathbf{x}|\mathbf{z}))] \quad (22.61)$$

We see from Equation (22.61) that VAEs are trying to minimize both the aggregated posterior $D_{\text{KL}}(q_{\phi}(\mathbf{z}) \| p_{\theta}(\mathbf{z}))$ and the expected likelihood $D_{\text{KL}}(q_{\phi}(\mathbf{x}|\mathbf{z}) \| p_{\theta}(\mathbf{x}|\mathbf{z}))$. Since \mathbf{x} is typically of much higher dimensionality than \mathbf{z} , the latter term usually dominates. Consequently, if there is a conflict between these two objectives (e.g., due to limited modeling power), the VAE will favor reconstruction accuracy over posterior inference. Thus the learned posterior may not be a very good approximation to the true posterior (see [ZSE19] for further discussion).

22.3 VAE generalizations

In this section, we discuss some variants of the basic VAE model.

22.3.1 σ -VAE

It is often the case that VAEs generate somewhat blurry images, as illustrated in Figure 22.3, Figure 22.4 and Figure 21.6. This is not the case for models that optimize the exact likelihood, such as pixelCNNs (Section 23.3.2) and flow models (Chapter 24). To see why VAEs are different, consider the VAE objective:

$$\mathcal{L} = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}[\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p_{\theta}(\mathbf{x}|\mathbf{z})]] - R(q_{\phi}) = \mathbb{E}_{q(\mathbf{z})}[\mathbb{E}_{q(\mathbf{x}|\mathbf{z})}[\log p_{\theta}(\mathbf{x}|\mathbf{z})]] - R(q_{\phi}) \quad (22.62)$$

where $R()$ is the KL regularizer. Now consider the common case where the decoder is a Gaussian with fixed variance, so

$$\log p_{\theta}(\mathbf{x}|\mathbf{z}) = -\frac{1}{2\sigma^2} \|\mathbf{x} - d_{\theta}(\mathbf{z})\|_2^2 \quad (22.63)$$

For a fixed inference network, the optimal setting of the generator parameters is to ensure $d_{\theta}(\mathbf{z}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\mathbf{x}]$ [ZSE17] for any given \mathbf{z} . Thus the decoder should predict the average of all inputs \mathbf{x} that map to that \mathbf{z} . So unless the encoder is lossless, the outputs will be blurry.

¹ We can solve this problem by increasing the expressive power of the posterior approximation
² (avoiding the merging of distinct inputs into the same latent code), or of the generator (by adding
³ back information that is missing from the latent code), or both. However, an even simpler solution is
⁴ to optimize the noise variance σ^2 , which controls the degree of blurring.

⁵ To do this, consider a VAE with a Gaussian decoder with mean $\hat{\mathbf{x}} = d_{\theta}(\mathbf{z})$ and spherical covariance
⁶ $\sigma^2 \mathbf{I}$. The corresponding negative log likelihood has the form

$$\frac{8}{9} -\log p(\mathbf{x}|\mathbf{z}) = \frac{D}{2\sigma^2} \text{mse}(d_{\theta}(\mathbf{z}), \mathbf{x}) + D \log \sqrt{2\pi} \quad (22.64)$$

¹⁰ where

$$\frac{12}{13} \text{mse}(\hat{\mathbf{x}}, \mathbf{x}) = \frac{1}{D} \sum_{d=1}^D (\hat{x}_d - x_d)^2 \quad (22.65)$$

¹⁵ is the mean squared error. Hence the negative ELBO is given by

$$\frac{17}{18} \mathcal{L}(\theta, \phi, \sigma) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} \left[\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\frac{D}{2\sigma^2} \text{mse}(d_{\theta}(\mathbf{z}), \mathbf{x}) \right] + D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z})) \right] \quad (22.66)$$

¹⁹ We can learn the parameter σ just like we learn the other parameters, by minimizing the above
²⁰ objective. However, we can sometimes get better results, and faster training, if we first fit θ and ϕ ,
²¹ and then estimate the optimal σ based on the mean of the residual squared errors, just like we do for
²² linear regression, i.e.,

$$\frac{24}{25} \sigma^* = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\text{mse}(d_{\theta}(\mathbf{z}), \mathbf{x})]] \quad (22.67)$$

²⁶ (In practice, the inner expectation is approximated with a single latent sample, and the outer
²⁷ expectation is approximated using the current minibatch sample.) This method is called the σ -
²⁸ **VAE** [RDL21], and can (sometimes) result in improved sample quality (compare top two rows of
²⁹ Figure 22.6).

³⁰ We can also use this approach to estimate a diagonal covariance matrix, which associates one
³¹ variance term per pixel (and per color channel). However, this can easily overfit. To see why, consider
³² an image with mostly black or white pixels; we can set the corresponding variance terms to 0 and
³³ drive the log likelihood to infinity, since we are modeling a constant value with a delta function. This
³⁴ pathology can be avoided by tying the variance parameter across output dimensions.

³⁵ 22.3.2 β -VAE

³⁷ The negative ELBO loss (which we want to minimize) can be written as follows:

$$\frac{39}{40} \mathcal{L}(\theta, \phi | \mathbf{x}) = \underbrace{-\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})]}_{\mathcal{L}_E} + \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\mathcal{L}_R} \quad (22.68)$$

⁴¹ where \mathcal{L}_E is the reconstruction error (negative log likelihood), and \mathcal{L}_R is the KL regularizer.

⁴³ It is natural to consider a generalization, in which we weight the KL term by an adjustable constant,
⁴⁴ $\beta > 0$. This gives the β -VAE objective [Hig+17]:

$$\frac{45}{46} \mathcal{L}_{\beta} = \mathcal{L}_E + \beta \mathcal{L}_R \quad (22.69)$$

⁴⁷

1			
2			
3	Optimal -VAE		
4			
5	Gaussian VAE		
6	(=1)		
7	-VAE, =10^-1		
8			
9	-VAE, =10^-2		
10			
11	-VAE, =10^-3		
12			
13	-VAE, =10^-4		
14			
15	-VAE, =10^-5		
16			
17			
18			CelebA
19			

Figure 22.6: Comparison of σ -VAE (top row), standard VAE (second row), and β -VAE (remaining rows). The model is a convolutional hierarchical VAE (Section 22.5) with Gaussian prior and Gaussian likelihood. From Figure 2 of [RDL21]. Used with kind permission of Oleh Rybkin.

If we set $\beta = 1$, we recover the objective used in standard VAEs. However, by reducing β , we can increase the mutual information (MI) between the latent code \mathbf{z} and the input \mathbf{x} . In the limit, if we set $\beta = 0$, we recover the objective used in standard autoencoders, which maximizes the MI, since there is no KL penalty. This ensures that the latent variable \mathbf{z} captures as much information as possible about the input \mathbf{x} .

Unfortunately, using $\beta \neq 1$ loses the property that the ELBO is a lower bound on the log likelihood. In addition, if the KL penalty is too weak, the posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is not forced to be close to the prior, $p_\theta(\mathbf{z})$, so generative samples will be low quality when β is small, as is apparent from Figure 22.6.

22.3.2.1 Connection with σ -VAE

The β -VAE is usually combined with a Gaussian decoder with unit variance. The loss function in this case becomes

$$\mathcal{L}_\beta = \frac{D}{2} \text{mse}(\hat{\mathbf{x}}, \mathbf{x}) + \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \quad (22.70)$$

We see that this is a special case of the σ -VAE from Section 22.3.1 where $\sigma^2 = \beta/2$. It is common to tune the β parameter in order to get good-looking samples, but obviously we could just tune σ^2 instead, which often works better, as shown in Figure 22.6.

1 **22.3.2.2 Disentangled representations**

3 One advantage of using $\beta > 1$ is that it encourages the learning of a latent representation that is
4 “**disentangled**”. Intuitively this means that each latent dimension represents a different **factor of**
5 **variation** in the input. This is often formalized in terms of the total correlation (Section 5.3.5.1),
6 which is defined as follows:

7

$$\text{TC}(\mathbf{z}) = \sum_k \mathbb{H}(z_k) - \mathbb{H}(\mathbf{z}) = D_{\text{KL}} \left(p(\mathbf{z}) \middle\| \prod_k p_k(z_k) \right) \quad (22.71)$$

8

9 This is zero iff the components of \mathbf{z} are all mutually independent, and hence disentangled. In [AS18],
10 they prove that using $\beta > 1$ will decrease the TC.

11 Unfortunately, in [Loc+18] they prove that nonlinear latent variable models are unidentifiable, and
12 therefore for any disentangled representation, there is an equivalent fully entangled representation
13 with exactly the same likelihood. Thus it is not possible to recover the correct latent representation
14 without choosing the appropriate inductive bias, via the encoder, decoder, prior, dataset, or learning
15 algorithm, i.e., merely adjusting β is not sufficient.

16

17 **22.3.2.3 Connection with information bottleneck**

18

19 In this section, we show that the β -VAE is unsupervised version of the information bottleneck (IB)
20 objective from Section 5.6. If the input is \mathbf{x} , the hidden bottleneck is \mathbf{z} , and the target outputs are
21 $\tilde{\mathbf{x}}$, then the unsupervised IB objective becomes

22

$$\mathcal{L}_{\text{UIB}} = \beta \mathbb{I}(\mathbf{z}; \mathbf{x}) - \mathbb{I}(\mathbf{z}; \tilde{\mathbf{x}}) \quad (22.72)$$

23

$$= \beta \mathbb{E}_{p(\mathbf{x}, \mathbf{z})} \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})p(\mathbf{z})} \right] - \mathbb{E}_{p(\mathbf{z}, \tilde{\mathbf{x}})} \left[\log \frac{p(\mathbf{z}, \tilde{\mathbf{x}})}{p(\mathbf{z})p(\tilde{\mathbf{x}})} \right] \quad (22.73)$$

24 where

25

$$p(\mathbf{x}, \mathbf{z}) = p_{\mathcal{D}}(\mathbf{x})p(\mathbf{z}|\mathbf{x}) \quad (22.74)$$

26

$$p(\mathbf{z}, \tilde{\mathbf{x}}) = \int p_{\mathcal{D}}(\mathbf{x})p(\mathbf{z}|\mathbf{x})p(\tilde{\mathbf{x}}|\mathbf{z})d\mathbf{x} \quad (22.75)$$

27 Intuitively, the objective in Equation (22.72) means we should pick a representation \mathbf{z} that can
28 predict $\tilde{\mathbf{x}}$ reliably, while not memorizing too much information about the input \mathbf{x} . The tradeoff
29 parameter is controlled by β .

30 From Equation (5.163), we have the following variational upper bound on this unsupervised
31 objective:

32

$$\mathcal{L}_{\text{UVIB}} = -\mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})] + \beta \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))] \quad (22.76)$$

33 which matches Equation (22.69) when averaged over \mathbf{x} .

34

35 **22.3.3 InfoVAE**

36 In Section 22.2.7, we discussed some drawbacks of the standard ELBO objective for training VAEs,
37 namely the tendency to ignore the latent code when the decoder is powerful (Section 22.4), and the
38

tendency to learn a poor posterior approximation due to the mismatch between the KL terms in data space and latent space (Section 22.2.7). We can fix these problems to some degree by using a generalized objective of the following form:

$$\mathcal{L}(\theta, \phi | \mathbf{x}) = -\lambda D_{\text{KL}}(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{q(\mathbf{z})} [D_{\text{KL}}(q_\phi(\mathbf{x}|\mathbf{z}) \| p_\theta(\mathbf{x}|\mathbf{z}))] + \alpha \mathbb{I}_q(\mathbf{x}; \mathbf{z}) \quad (22.77)$$

where $\alpha \geq 0$ controls much we weight the mutual information $\mathbb{I}_q(\mathbf{x}; \mathbf{z})$ between \mathbf{x} and \mathbf{z} , and $\lambda \geq 0$ controls the tradeoff between \mathbf{z} -space KL and \mathbf{x} -space KL. This is called the **InfoVAE** objective [ZSE19]. If we set $\alpha = 0$ and $\lambda = 1$, we recover the standard ELBO, as shown in Equation (22.61).

Unfortunately, the objective in Equation (22.77) cannot be computed as written, because of the intractable MI term:

$$\mathbb{I}_q(\mathbf{x}; \mathbf{z}) = \mathbb{E}_{q(\mathbf{x}, \mathbf{z})} \left[\log \frac{q_\phi(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{x})q_\phi(\mathbf{z})} \right] = \mathbb{E}_{q(\mathbf{x}, \mathbf{z})} \left[\log \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \quad (22.78)$$

However, using the fact that $q_\phi(\mathbf{x}|\mathbf{z}) = p_{\mathcal{D}}(\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})/q_\phi(\mathbf{z})$, we can rewrite the objective as follows:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{x}, \mathbf{z})} \left[-\lambda \log \frac{q_\phi(\mathbf{z})}{p_\theta(\mathbf{z})} - \log \frac{q_\phi(\mathbf{x}|\mathbf{z})}{p_\theta(\mathbf{x}|\mathbf{z})} - \alpha \log \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \quad (22.79)$$

$$= \mathbb{E}_{q(\mathbf{x}, \mathbf{z})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log \frac{q_\phi(\mathbf{z})^{\lambda+\alpha-1} p_{\mathcal{D}}(\mathbf{x})}{p_\theta(\mathbf{z})^\lambda q_\phi(\mathbf{z}|\mathbf{x})^{\alpha-1}} \right] \quad (22.80)$$

$$= \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]] - (1-\alpha) \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}))]$$

$$- (\alpha + \lambda - 1) D_{\text{KL}}(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\log p_{\mathcal{D}}(\mathbf{x})] \quad (22.81)$$

where the last term is a constant we can ignore. The first two terms can be optimized using the reparameterization trick. Unfortunately, the last term requires computing $q_\phi(\mathbf{z}) = \int_{\mathbf{x}} q_\phi(\mathbf{x}, \mathbf{z}) d\mathbf{x}$, which is intractable. Fortunately, we can easily sample from this distribution, by sampling $\mathbf{x} \sim p_{\mathcal{D}}(\mathbf{x})$ and $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Thus $q_\phi(\mathbf{z})$ is an **implicit probability model**, similar to a GAN (see Chapter 27).

As long as we use a strict divergence, meaning $D(q, p) = 0$ iff $q = p$, then one can show that this does not affect the optimality of the procedure. In particular, proposition 2 of [ZSE19] tells us the following:

Theorem 1. Let \mathcal{X} and \mathcal{Z} be continuous spaces, and $\alpha < 1$ (to bound the MI) and $\lambda > 0$. For any fixed value of $\mathbb{I}_q(\mathbf{x}; \mathbf{z})$, the approximate InfoVAE loss, with any strict divergence $D(q_\phi(\mathbf{z}), p_\theta(\mathbf{z}))$, is globally optimized if $p_\theta(\mathbf{x}) = p_{\mathcal{D}}(\mathbf{x})$ and $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$.

22.3.3.1 Connection with MMD VAE

If we set $\alpha = 1$, the InfoVAE objective simplifies to

$$\mathcal{L} \stackrel{c}{=} \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]] - \lambda D_{\text{KL}}(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) \quad (22.82)$$

The **MMD VAE**⁴ replaces the KL divergence in the above term with the (squared) maximum mean discrepancy or **MMD** divergence defined in Section 2.9.3. (This is valid based on the above theorem.)

4. Proposed in <https://ermongroup.github.io/blog/a-tutorial-on-mmd-variational-autoencoders/>.

¹ The advantage of this approach over standard InfoVAE is that the resulting objective is tractable. In
² particular, if we set $\lambda = 1$ and swap the sign we get
³

$$\mathcal{L} = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [-\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]] + \text{MMD}(q_{\boldsymbol{\phi}}(\mathbf{z}), p_{\boldsymbol{\theta}}(\mathbf{z})) \quad (22.83)$$

⁴ As we discuss in Section 2.9.3, we can compute the MMD as follows:
⁵

$$\text{MMD}(p, q) = \mathbb{E}_{p(\mathbf{z}), p(\mathbf{z}')} [\mathcal{K}(\mathbf{z}, \mathbf{z}')] + \mathbb{E}_{q(\mathbf{z}), q(\mathbf{z}')} [\mathcal{K}(\mathbf{z}, \mathbf{z}')] - 2\mathbb{E}_{p(\mathbf{z}), q(\mathbf{z}')} [\mathcal{K}(\mathbf{z}, \mathbf{z}')] \quad (22.84)$$

⁶ where $\mathcal{K}()$ is some kernel function, such as the RBF kernel, $\mathcal{K}(\mathbf{z}, \mathbf{z}') = \exp(-\frac{1}{2\sigma^2} \|\mathbf{z} - \mathbf{z}'\|_2^2)$. Intuitively
⁷ the MMD measures the similarity (in latent space) between samples from the prior and samples from
⁸ the aggregated posterior.
⁹

¹⁰ In practice, we can implement the MMD objective by using the posterior predicted mean $\mathbf{z}_n =$
¹¹ $e_{\boldsymbol{\phi}}(\mathbf{x}_n)$ for all B samples in the current minibatch, and comparing this to B random samples from
¹² the $\mathcal{N}(\mathbf{0}, \mathbf{I})$ prior.
¹³

¹⁴ If we use a Gaussian decoder with fixed variance, the negative log likelihood is just a squared error
¹⁵ term:
¹⁶

$$-\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \|\mathbf{x}_n - d_{\boldsymbol{\theta}}(\mathbf{z}_n)\|_2^2 \quad (22.85)$$

¹⁷ Thus the entire model is deterministic, and just predicts the means in latent space and visible space.
¹⁸

¹⁹ 22.3.3.2 Connection with β -VAEs

²⁰ If we set $\alpha = 0$ and $\lambda = 1$, we get back the original ELBO. If $\lambda > 0$ is freely chosen, but we use
²¹ $\alpha = 1 - \lambda$, we get the β -VAE.
²²

²³ 22.3.3.3 Connection with adversarial autoencoders

²⁴ If we set $\alpha = 1$ and $\lambda = 1$, and D is chosen to be the Jensen Shannon divergence (which can be
²⁵ minimized by training a binary discriminator, as explained in Section 27.2.2), then we get a model
²⁶ known as an **adversarial autoencoder** [Mak+15a].
²⁷

²⁸

²⁹ 22.3.3.4 Example

³⁰ In this section, we give a simple example of InfoVAE (MMD version) applied to MNIST. We use
³¹ just $L = 2$ latent dimensions, so we can plot the latent space. We use the squared MMD divergence
³² (Section 2.9.3) as an alternative to $D_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z})\|p_{\boldsymbol{\theta}}(\mathbf{z}))$. We implement MMD using an RBF kernel,
³³ using the bandwidth parameter $\sigma^2 = 2/L$.⁵ The neural architecture is an MLP with one hidden
³⁴ layer. In Figure 22.7(a), we show the result of fitting this model using the standard ELBO. In
³⁵ Figure 22.7(b), we show the result of fitting this model using the InfoVAE loss, with $\alpha = 1$ and
³⁶ $\alpha + \lambda - 1 = 1$. We see that the latent space for the InfoVAE model shows better class separation,
³⁷ suggesting that it has captured more of the “semantics” of the data.
³⁸

³⁹ 22.3.4 Multi-modal VAEs

⁴⁰ It is possible to extend VAEs to create joint distributions over different kinds of variables, such as
⁴¹ images and text. This is sometimes called a **multimodal VAE** or **MVAE**. Let us assume there are
⁴²

⁴³ 5. This is the hyperparameter recommended in the MMD blog post.
⁴⁴

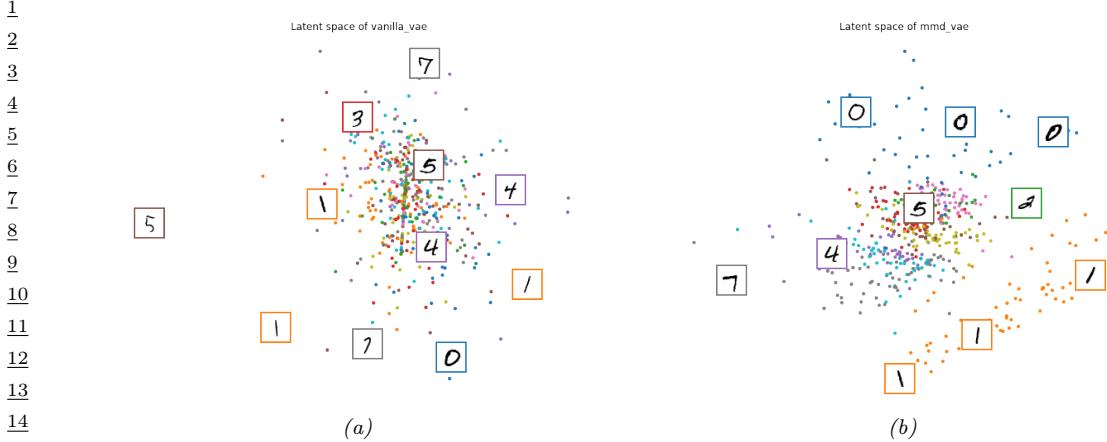


Figure 22.7: Illustration of 2d embedding space for (a) VAE and (b) MMD-VAE fit to MNIST. Generated by `vae_latent_space.ipynb`.

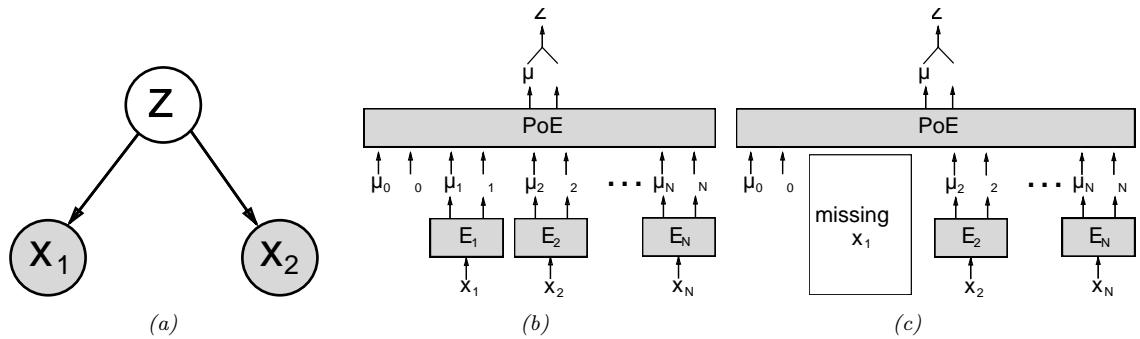


Figure 22.8: Illustration of multi-modal VAE. (a) The generative model with $N = 2$ modalities. (b) The product of experts (PoE) inference network is derived from N individual Gaussian experts E_i . μ_0 and σ_0 are parameters of the prior. (c) If a modality is missing, we omit its contribution to the posterior. From Figure 1 of [WG18]. Used with kind permission of Mike Wu.

M modalities. We assume they are conditionally independent given the latent code, and hence the generative model has the form

$$p_{\theta}(\mathbf{x}_1, \dots, \mathbf{x}_M, \mathbf{z}) = p(\mathbf{z}) \prod_{m=1}^M p_{\theta}(\mathbf{x}_m | \mathbf{z}) \quad (22.86)$$

where we treat $p(\mathbf{z})$ as a fixed prior. See Figure 22.8(a) for an illustration.

The standard ELBO is given by

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{X})} \left[\sum_m \log p_{\theta}(\mathbf{x}_m | \mathbf{z}) \right] - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{X}) \| p(\mathbf{z})) \quad (22.87)$$

where $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_M)$ is the observed data. However, the different likelihood terms $p(\mathbf{x}_m|\mathbf{z})$ may have different dynamic ranges (e.g., Gaussian pdf for pixels, and categorical pmf for text), so we introduce weight terms $\lambda_m \geq 0$ for each likelihood. In addition, let $\beta \geq 0$ control the amount of KL regularization. This gives us a weighted version of the ELBO, as follows:

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{X})} \left[\sum_m \lambda_m \log p_{\theta}(\mathbf{x}_m|\mathbf{z}) \right] - \beta D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{X}) \| p(\mathbf{z})) \quad (22.88)$$

Often we don't have a lot of paired (aligned) data from all M modalities. For example, we may have a lot of images (modality 1), and a lot of text (modality 2), but very few (image, text) pairs. So it is useful to generalize the loss so it fits the marginal distributions of subsets of the features. Let $O_m = 1$ if modality m is observed (i.e., \mathbf{x}_m is known), and let $O_m = 0$ if it is missing or unobserved. Let $\mathbf{X} = \{\mathbf{x} : O_m = 1\}$ be the visible features. We now use the following objective:

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{X})} \left[\sum_{m:O_m=1} \lambda_m \log p_{\theta}(\mathbf{x}_m|\mathbf{z}) \right] - \beta D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{X}) \| p(\mathbf{z})) \quad (22.89)$$

The key problem is how to compute the posterior $q_{\phi}(\mathbf{z}|\mathbf{X})$ given different subsets of features. In general this can be hard, since the inference network is a discriminative model that assumes all inputs are available. For example, if it is trained on (image, text) pairs, $q_{\phi}(\mathbf{z}|\mathbf{x}_1, \mathbf{x}_2)$, how can we compute the posterior just given an image, $q_{\phi}(\mathbf{z}|\mathbf{x}_1)$, or just given text, $q_{\phi}(\mathbf{z}|\mathbf{x}_2)$? (This issue arises in general with VAE when we have missing inputs; we discuss the general case in Section 22.3.5.)

Fortunately, based on our conditional independence assumption between the modalities, we can compute the optimal form for $q_{\phi}(\mathbf{z}|\mathbf{X})$ given set of inputs by computing the exact posterior under the model, which is given by

$$p(\mathbf{z}|\mathbf{X}) = \frac{p(\mathbf{z})p(\mathbf{x}_1, \dots, \mathbf{x}_M|\mathbf{z})}{p(\mathbf{x}_1, \dots, \mathbf{x}_M)} = \frac{p(\mathbf{z})}{p(\mathbf{x}_1, \dots, \mathbf{x}_M)} \prod_{m=1}^M p(\mathbf{x}_m|\mathbf{z}) \quad (22.90)$$

$$= \frac{p(\mathbf{z})}{p(\mathbf{x}_1, \dots, \mathbf{x}_M)} \prod_{m=1}^M \frac{p(\mathbf{z}|\mathbf{x}_m)p(\mathbf{x}_m)}{p(\mathbf{z})} \quad (22.91)$$

$$\propto p(\mathbf{z}) \prod_{m=1}^M \frac{p(\mathbf{z}|\mathbf{x}_m)}{p(\mathbf{z})} \approx p(\mathbf{z}) \prod_{m=1}^M \tilde{q}(\mathbf{z}|\mathbf{x}_m) \quad (22.92)$$

This can be viewed as a product of experts (Section 25.1.1), where each $\tilde{q}(\mathbf{z}|\mathbf{x}_m)$ is an “expert” for the m ’th modality, and $p(\mathbf{z})$ is the prior. We can compute the above posterior for any subset of modalities for which we have data by modifying the product over m . If we use Gaussian distributions for the prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0^{-1})$ and marginal posterior ratio $\tilde{q}(\mathbf{z}|\mathbf{x}_m) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m^{-1})$, then we can compute the product of Gaussians using the result from Equation (2.94):

$$\prod_{m=0}^M \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m^{-1}) \propto \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} = (\sum_m \boldsymbol{\Lambda}_m)^{-1}, \quad \boldsymbol{\mu} = \boldsymbol{\Sigma}(\sum_m \boldsymbol{\Lambda}_m \boldsymbol{\mu}_m) \quad (22.93)$$

Thus the overall posterior precision is the sum of individual expert posterior precisions, and the overall posterior mean is the precision weighted average of the individual expert posterior means.

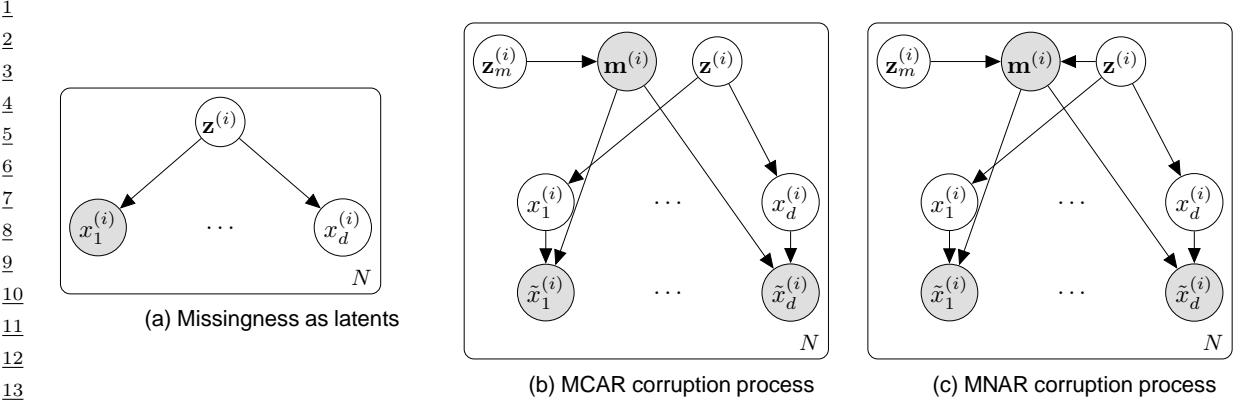


Figure 22.9: Illustration of different VAE variants for handling missing data. From Figure 1 of [CNW20]. Used with kind permission of Mark Collier.

See Figure 22.8(b) for an illustration. For a linear Gaussian (factor analysis) model, we can ensure $q(\mathbf{z}|\mathbf{x}_m) = p(\mathbf{z}|\mathbf{x}_m)$, in which case the above solution is the exact posterior [WN18], but in general it will be an approximation.

We need to train the individual expert recognition models $q(\mathbf{z}|\mathbf{x}_m)$ as well as the joint model $q(\mathbf{z}|\mathbf{X})$, so the model knows what to do with fully observed as well as partially observed inputs at test time. In [Ved+18], they propose a somewhat complex “triple ELBO” objective. In [WG18], they propose the simpler approach of optimizing the ELBO for the fully observed feature vector, all the marginals, and a set of J randomly chosen joint modalities:

$$\mathcal{L}_{\theta, \phi}(\mathbf{X}) = \mathcal{L}_{\theta, \phi}(\mathbf{x}_1, \dots, \mathbf{x}_M) + \sum_{m=1}^M \mathcal{L}_{\theta, \phi}(\mathbf{x}_m) + \sum_{j=1}^J \mathcal{L}_{\theta, \phi}(\mathbf{X}_j) \quad (22.94)$$

This generalizes nicely to the semi-supervised setting, in which we only have a few aligned (“labeled”) examples from the joint, but have many unaligned (“unlabeled”) examples from the individual marginals. See Figure 22.8(c) for an illustration.

Note that the above scheme can only handle the case of a fixed number of missingness patterns; we generalize to allow for arbitrary missingness in Section 22.3.5.

22.3.5 VAEs with missing data

Sometimes we may have **missing data**, in which parts of the data vector $\mathbf{x} \in \mathbb{R}^D$ may be unknown. In Section 22.3.4 we saw a special case of this when we discussed multimodal VAEs. In this section we allow for arbitrary patterns of missingness.

To model the missing data, let $\mathbf{m} \in \{0, 1\}^D$ be a binary vector where $m_j = 1$ is x_j is missing, and $m_j = 0$ otherwise. Let $\mathbf{X} = \{\mathbf{x}^{(n)}\}$ and $\mathbf{M} = \{\mathbf{m}^{(n)}\}$ be $N \times D$ matrices. Furthermore, let \mathbf{X}_o be the observed parts of \mathbf{X} and \mathbf{X}_h be the hidden parts. If we assume $p(\mathbf{M}|\mathbf{X}_o, \mathbf{X}_h) = p(\mathbf{M})$, we say the data is **missing completely at random** or **MCAR**, since the missingness does not depend on the hidden or observed features. If we assume $p(\mathbf{M}|\mathbf{X}_o, \mathbf{X}_h) = p(\mathbf{M}|\mathbf{X}_o)$, we say the data is **missing at**

1 random or MAR, since the missingness does not depend on the hidden features, but may depend
2 on the visible features. If neither of these assumptions hold, we say the data is **not missing at**
3 **random** or **NMAR**.

4 In the MCAR and MAR cases, we can ignore the missingness mechanism, since it tells us nothing
5 about the hidden features. However, in the NMAR case, we need to model the **missing data**
6 **mechanism**, since the lack of information may be informative. For example, the fact that someone
7 did not fill out an answer to a sensitive question on a survey (e.g., “Do you have COVID?”) could be
8 informative about the underlying value. See e.g., [LR87; Mar08] for more information on missing
9 data models.

10 In the context of VAEs, we can model the MCAR scenario by treating the missing values as latent
11 variables. This is illustrated in Figure 22.9(a). Since missing leaf nodes in a directed graphical model
12 do not affect their parents, we can simply ignore them when computing the posterior $p(\mathbf{z}^{(i)}|\mathbf{x}_o^{(i)})$,
13 where $\mathbf{x}_o^{(i)}$ are the observed parts of example i . However, when using an amortized inference network,
14 it can be difficult to handle missing inputs, since the model is usually trained to compute $p(\mathbf{z}^{(i)}|\mathbf{x}_{1:d}^{(i)})$.
15 One solution to this is to use the product of experts approach discussed in the context of multi-modal
16 VAEs in Section 22.3.4. However, this is designed for the case where whole blocks (corresponding to
17 different modalities) are missing, and will not work well if there are arbitrary missing patterns (e.g.,
18 pixels that get dropped out due to occlusion or scratches on the lens). In addition, this method will
19 not work for the MNAR case.

20 An alternative approach, proposed in [CNW20], is to explicitly include the missingness indicators
21 into the model, as shown in Figure 22.9(b). We assume the model always generates each \mathbf{x}_j for
22 $j = 1 : d$, but we only get to see the “corrupted” versions $\tilde{\mathbf{x}}_j$. If $m_j = 0$ then $\tilde{\mathbf{x}}_j = \mathbf{x}_j$, but if $m_j = 1$,
23 then $\tilde{\mathbf{x}}_j$ is a special value, such as 0, unrelated to \mathbf{x}_j . We can model any correlation between the
24 missingness elements (components of \mathbf{m}) by using another latent variable \mathbf{z}_m . This model can easily
25 be extended to the MNAR case by letting \mathbf{m} depend on the latent factors for the observed data, \mathbf{z} ,
26 as well as the usual missingness latent factors \mathbf{z}_m , as shown in Figure 22.9(c).

27 We modify the VAE to be conditional on the missingness pattern, so the VAE decoder has the
28 form $p(\mathbf{x}_o|\mathbf{z}, \mathbf{m})$, and the encoder has the form $q(\mathbf{z}|\mathbf{x}_o, \mathbf{m})$. However, we assume the prior is $p(\mathbf{z})$
29 as usual, independent of \mathbf{m} . We can compute a lower bound on the log marginal likelihood of the
30 observed data, given the missingness, as follows:

31

32

33

$$\log p(\mathbf{x}_o|\mathbf{m}) = \log \int \int p(\mathbf{x}_o, \mathbf{x}_m|\mathbf{z}, \mathbf{m}) p(\mathbf{z}) d\mathbf{x}_m d\mathbf{z} \quad (22.95)$$

34

35

$$= \log \int p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) p(\mathbf{z}) d\mathbf{z} \quad (22.96)$$

36

37

$$= \log \int p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) p(\mathbf{z}) \frac{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})}{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} d\mathbf{z} \quad (22.97)$$

38

39

$$= \log \mathbb{E}_{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} \left[p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} \right] \quad (22.98)$$

40

41

$$\geq \mathbb{E}_{q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})} [\log p(\mathbf{x}_o|\mathbf{z}, \mathbf{m}) - D_{\text{KL}}(q(\mathbf{z}|\tilde{\mathbf{x}}, \mathbf{m})||p(\mathbf{z}))] \quad (22.99)$$

42

43

44

45

46

47

We can fit this model in the usual way. See Figure 22.10 for an example.

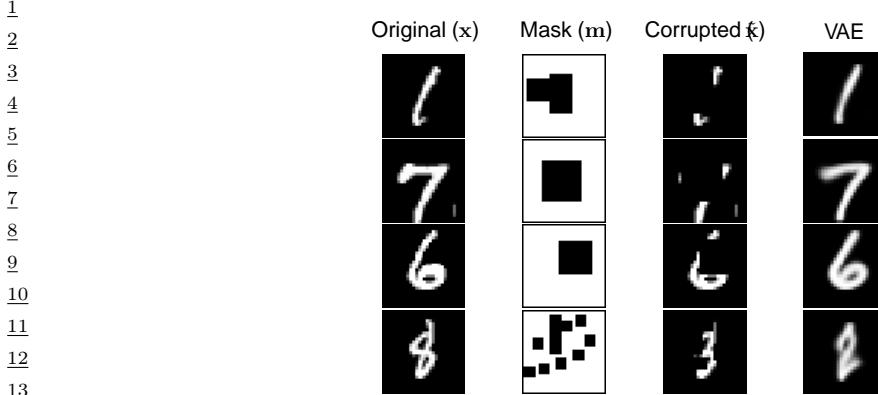


Figure 22.10: Imputing missing pixels given a masked out image using a VAE using a MCAR assumption. From Figure 2 of [CNW20]. Used with kind permission of Mark Collier.

22.3.6 Semi-supervised VAEs

In this section, we discuss how to extend VAEs to the **semi-supervised learning** setting in which we have both labeled data, $\mathcal{D}_L = \{(\mathbf{x}_n, y_n)\}$, and unlabeled data, $\mathcal{D}_U = \{(\mathbf{x}_n)\}$. We focus on the **M2** model, proposed in [Kin+14].

The generative model has the following form:

$$p_{\theta}(\mathbf{x}, y) = p_{\theta}(y)p_{\theta}(\mathbf{x}|y) = p_{\theta}(y) \int p_{\theta}(\mathbf{x}|y, \mathbf{z})p_{\theta}(\mathbf{z})d\mathbf{z} \quad (22.100)$$

where \mathbf{z} is a latent variable, $p_{\theta}(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ is the latent prior, $p_{\theta}(y) = \text{Cat}(y|\boldsymbol{\pi})$ the label prior, and $p_{\theta}(\mathbf{x}|y, \mathbf{z}) = p(\mathbf{x}|f_{\theta}(y, \mathbf{z}))$ is the likelihood, such as a Gaussian, with parameters computed by f (a deep neural network). The main innovation of this approach is to assume that data is generated according to both a latent class variable y as well as the continuous latent variable \mathbf{z} . The class variable y is observed for labeled data and unobserved for unlabeled data.

To compute the likelihood for the *labeled data*, $p_{\theta}(\mathbf{x}, y)$, we need to marginalize over \mathbf{z} , which we can do by using an inference network of the form

$$q_{\phi}(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(y, \mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x}))) \quad (22.101)$$

We then use the following variational lower bound

$$\log p_{\theta}(\mathbf{x}, y) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, y)} [\log p_{\theta}(\mathbf{x}|y, \mathbf{z}) + \log p_{\theta}(y) + \log p_{\theta}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|\mathbf{x}, y)] = -\mathcal{L}(\mathbf{x}, y) \quad (22.102)$$

as is standard for VAEs (see Section 22.2). The only difference is that we observe two kinds of data: \mathbf{x} and y .

To compute the likelihood for the *unlabeled data*, $p_{\theta}(\mathbf{x})$, we need to marginalize over \mathbf{z} and y , which we can do by using an inference network of the form

$$q_{\phi}(\mathbf{z}, y|\mathbf{x}) = q_{\phi}(\mathbf{z}|\mathbf{x})q_{\phi}(y|\mathbf{x}) \quad (22.103)$$

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(\mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x}))) \quad (22.104)$$

$$q_{\phi}(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}_{\phi}(\mathbf{x})) \quad (22.105)$$

¹ Note that $q_\phi(y|\mathbf{x})$ acts like a discriminative classifier, that imputes the missing labels. We then use
² the following variational lower bound:
³

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}, y|\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|y, \mathbf{z}) + \log p_{\boldsymbol{\theta}}(y) + \log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\phi}(\mathbf{z}, y|\mathbf{x})] \quad (22.106)$$

$$= - \sum_y q_{\phi}(y|\mathbf{x}) \mathcal{L}(\mathbf{x}, y) + \mathbb{H}(q_{\phi}(y|\mathbf{x})) = -\mathcal{U}(\mathbf{x}) \quad (22.107)$$

⁹ Note that the discriminative classifier $q_{\phi}(y|\mathbf{x})$ is only used to compute the log-likelihood of the
¹⁰ unlabeled data, which is undesirable. We can therefore add an extra classification loss on the
¹¹ supervised data, to get the following overall objective function:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}_L} [\mathcal{L}(\mathbf{x}, y)] + \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_U} [\mathcal{U}(\mathbf{x})] + \alpha \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}_L} [-\log q_{\phi}(y|\mathbf{x})] \quad (22.108)$$

¹² where α is a hyperparameter that controls the relative weight of generative and discriminative
¹³ learning.

¹⁴

¹⁵ 22.3.7 VAEs with sequential encoders/decoders

¹⁶ In this section, we discuss VAEs for sequential data, such as text and biosequences, in which the
¹⁷ data \mathbf{x} is a variable-length sequence, but we have a fixed-sized latent variable $\mathbf{z} \in \mathbb{R}^K$. (We consider
¹⁸ the more general case in which \mathbf{z} is also a variable-length sequence of latents in Section 31.5.) All we
¹⁹ have to do is modify the decoder $p(\mathbf{x}|\mathbf{z})$ and encoder $q(\mathbf{z}|\mathbf{x})$ to work with sequences.

²⁰

²¹ 22.3.7.1 Models

²²

²³ If we use an RNN for the encoder and decoder of a VAE, we get a model which is called a **VAE-RNN**. This was first proposed in [Bow+16a]. In more detail, the generative model is $p(\mathbf{z}, \mathbf{x}_{1:T}) =$
²⁴ $p(\mathbf{z}) \text{RNN}(\mathbf{x}_{1:T}|\mathbf{z})$, where \mathbf{z} can be injected as the initial state of the RNN, or as an input to every
²⁵ time step. The inference model is $q(\mathbf{z}|\mathbf{x}_{1:T}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}(\mathbf{h}), \boldsymbol{\Sigma}(\mathbf{h}))$, where $\mathbf{h} = [\mathbf{h}_T^\rightarrow, \mathbf{h}_1^\leftarrow]$ is the output
²⁶ of a bidirectional RNN applied to $\mathbf{x}_{1:T}$. See Figure 22.11 for an illustration.

²⁷ More recently, people have tried to combine transformers with VAEs. For example, in the **Optimus**
²⁸ model of [Li+20a], they use a BERT model for the encoder. In more detail, the encoder $q(\mathbf{z}|\mathbf{x})$ is
²⁹ derived from the embedding vector associated with a dummy token corresponding to the “class label”
³⁰ which is appended to the input sequence \mathbf{x} . The decoder is a standard autoregressive model (similar
³¹ to GPT), with one additional input, namely the latent vector \mathbf{z} . They consider two ways of injecting
³² the latent vector. The simplest approach is to add \mathbf{z} to the embedding layer of every token in the
³³ decoding step, by defining $\mathbf{h}'_i = \mathbf{h}_i + \mathbf{W}\mathbf{z}$, where $\mathbf{h}_i \in \mathbb{R}^H$ is the original embedding for the i 'th
³⁴ token, and $\mathbf{W} \in \mathbb{R}^{H \times K}$ is a decoding matrix, where K is the size of the latent vector. However, they
³⁵ get better results in their experiments by letting all the layers of the decoder attend to the latent
³⁶ code \mathbf{z} . An easy way to do this is to define the memory vector $\mathbf{h}_m = \mathbf{W}\mathbf{z}$, where $\mathbf{W} \in \mathbb{R}^{LH \times K}$,
³⁷ where L is the number of layers in the decoder, and then to append $\mathbf{h}_m \in \mathbb{R}^{L \times H}$ to all the other
³⁸ embeddings at each layer.

³⁹ An alternative approach, known as **transformer VAE**, was proposed in [Gre20]. This model uses
⁴⁰ a **funnel transformer** [Dai+20b] as the encoder, and the **T5** [Raf+19] conditional transformer for
⁴¹ the decoder. In addition, it uses an MMD VAE (Section 22.3.3.1) to avoid posterior collapse.

⁴²

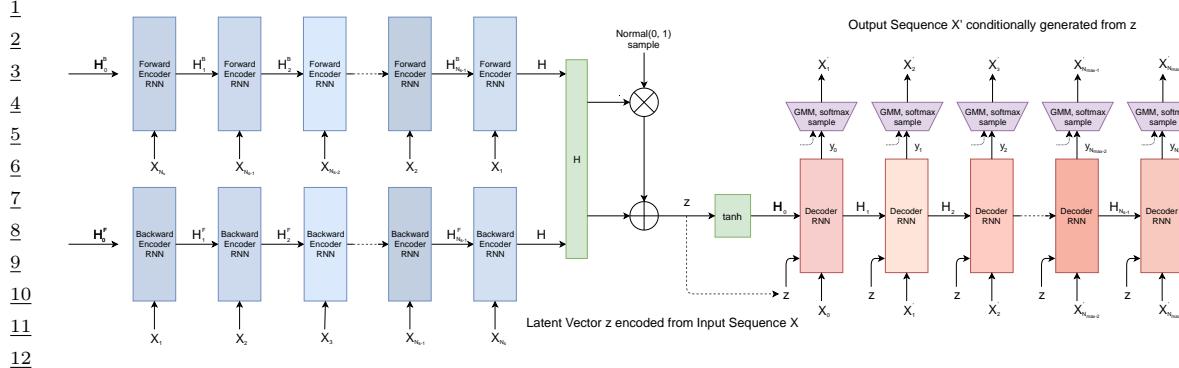


Figure 22.11: Illustration of a VAE with a bidirectional RNN encoder and a unidirectional RNN decoder. The output generator can use a GMM and/or softmax distribution. From Figure 2 of [HE18]. Used with kind permission of David Ha.

he was silent for a long moment .
 he was silent for a moment .
 it was quiet for a moment .
 it was dark and cold .
 there was a pause .
 it was my turn .

i went to the store to buy some groceries .
 i store to buy some groceries .
 i were to buy any groceries .
 horses are to buy any groceries .
 horses are to buy any animal .
 horses the favorite any animal .
 horses the favorite favorite animal .
 horses are my favorite animal .

(a)

(b)

Figure 22.12: (a) Samples from the latent space of a VAE text model, as we interpolate between two sentences (on first and last line). Note that the intermediate sentences are grammatical, and semantically related to their neighbors. From Table 8 of [Bow+16b]. (b) Same as (a), but now using a deterministic autoencoder (with the same RNN encoder and decoder). From Table 1 of [Bow+16b]. Used with kind permission of Sam Bowman.

22.3.7.2 Applications

In this section, we discuss some applications of VAEs to sequence data.

Text

In [Bow+16b], they apply the VAE-RNN model to natural language sentences. (See also [MB16; SSB17] for related work.) Although this does not improve performance in terms of the standard perplexity measures (predicting the next word given the previous words), it does provide a way to infer a semantic representation of the sentence. This can then be used for latent space interpolation, as discussed in Section 21.3.5. The results of doing this with the VAE-RNN are illustrated in Figure 22.12a. (Similar results are shown in [Li+20a], using a VAE-transformer.) By contrast, if we use a standard deterministic autoencoder, with the same RNN encoder and decoder networks, we learn a much less meaningful space, as illustrated in Figure 22.12b. The reason is that the deterministic autoencoder has “holes” in its latent space, which get decoded to nonsensical outputs.

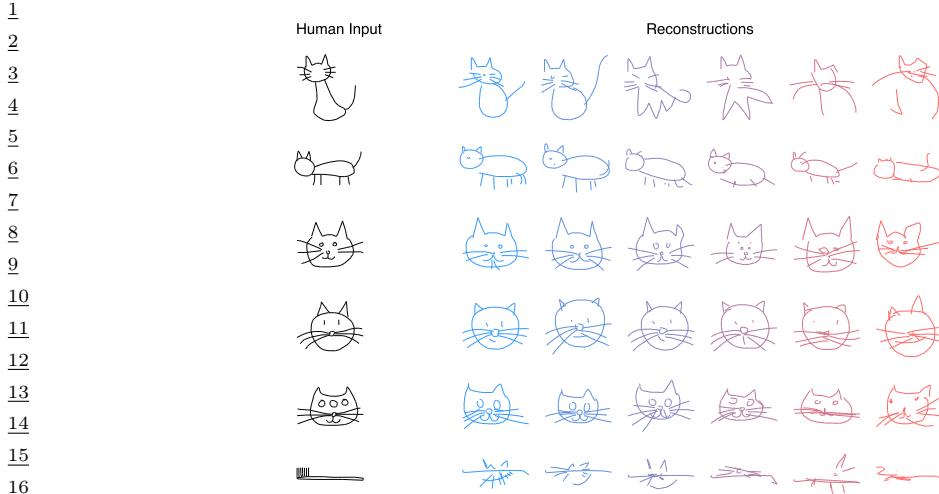


Figure 22.13: Conditional generation of cats from sketch-RNN model. We increase the temperature parameter from left to right. From Figure 5 of [HE18]. Used with kind permission of David Ha.

However, because RNNs (and transformers) are powerful decoders, we need to address the problem of posterior collapse, which we discuss in Section 22.4. One common way to avoid this problem is to use KL annealing, but a more effective method is to use the InfoVAE method of Section 22.3.3, which includes adversarial autoencoders (used in [She+20] with an RNN decoder) and MMD autoencoders (used in [Gre20] with a transformer decoder).

Sketches

In [HE18], they apply the VAE-RNN model to generate sketches (line drawings) of various animals and hand-written characters. They call their model **sketch-rnn**. The training data records the sequence of (x, y) pen positions, as well as whether the pen was touching the paper or not. The emission model used a GMM for the real-valued location offsets, and a categorical softmax distribution for the discrete state.

Figure 22.13 shows some samples from various class-conditional models. We vary the temperature parameter τ of the emission model to control the stochasticity of the generator. (More precisely, we multiply the GMM variances by τ , and divide the discrete probabilities by τ before renormalizing.) When the temperature is low, the model tries to reconstruct the input as closely as possible. However, when the input is untypical of the training set (e.g., a cat with three eyes, or a toothbrush), the reconstruction is “regularized” towards a canonical cat with two eyes, while still keeping some features of the input.

47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

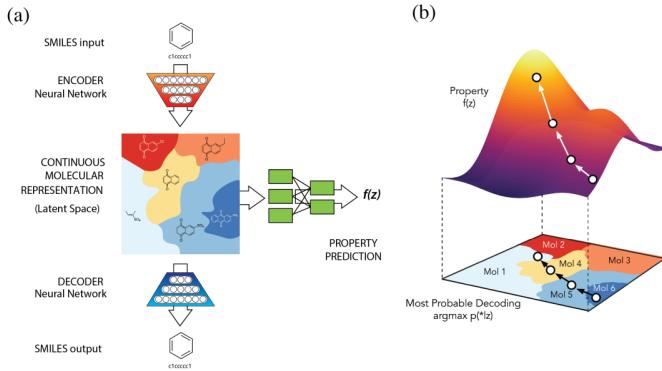


Figure 22.14: Application of VAE-RNN to molecule design. (a) The VAE-RNN model is trained on a sequence representation of molecules known as SMILES. We can fit an MLP to map from the latent space to properties of the molecule, such as its “fitness” $f(\mathbf{z})$. (b) We can perform gradient ascent in $f(\mathbf{z})$ space, and then decode the result to a new molecule with high fitness. From Figure 1 of [GB+18]. Used with kind permission of Rafael Gomez-Bombarelli.

Molecular design

In [GB+18], they use VAE-RNNs to model molecular graph structure, represented as a string using the SMILES representation.⁶ It is also possible to learn a mapping from the latent space to some scalar quantity of interest, such as the solubility or drug efficacy of a molecule. We can then perform gradient-based optimization in the continuous latent space to try to generate new graphs which maximize this quantity. See Figure 22.14 for a sketch of this approach.

The main problem is to ensure that points in latent space decode to valid strings/ molecules. There are various solutions to this, including using a **grammar VAE**, where the RNN decoder is replaced by a stochastic context free grammar. See [KPHL17] for details.

22.4 Avoiding posterior collapse

If the decoder $p_{\theta}(\mathbf{x}|\mathbf{z})$ is sufficiently powerful (e.g., a pixel CNN, or an RNN for text), then the VAE does not need to use the latent code \mathbf{z} for anything. This is called **posterior collapse** or **variational overpruning** (see e.g., [Che+17b; Ale+18; Hus17a; Phu+18; TT17; Yeu+17; Luc+19; DWW19; WBC21]). To see why this happens, consider Equation (22.60). If there exists a parameter setting for the generator θ^* such that $p_{\theta^*}(\mathbf{x}|\mathbf{z}) = p_{\mathcal{D}}(\mathbf{x})$ for every \mathbf{z} , then we can make $D_{\text{KL}}(p_{\mathcal{D}}(\mathbf{x})||p_{\theta}(\mathbf{x})) = 0$. Since the generator is independent of the latent code, we have $p_{\theta}(\mathbf{z}|\mathbf{x}) = p_{\theta}(\mathbf{z})$. The prior $p_{\theta}(\mathbf{z})$ is usually a simple distribution, such as a Gaussian, so we can find a setting of the inference parameters so that $q_{\phi^*}(\mathbf{z}|\mathbf{x}) = p_{\theta}(\mathbf{z})$, which ensures $D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\theta}(\mathbf{z}|\mathbf{x})) = 0$. Thus we have successfully minimized the ELBO, but we have not learned any useful latent representation of the data, which is

⁶ See https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system.

1 one of the goals of latent variable modeling.⁷

2 We discuss some solutions to posterior collapse below.

3

4 **22.4.1 KL annealing**

5 A common approach to solving this problem, proposed in [Bow+16a], is to use **KL annealing**, in
6 which the KL penalty term in the ELBO is scaled by β , which is increased from 0.0 (corresponding
7 to an autoencoder) to 1.0 (which corresponds to standard MLE training). (Note that, by contrast,
8 the β -VAE model in Section 22.3.2 uses $\beta > 1$.)

9 KL annealing can work well, but requires tuning the schedule for β . A standard practice [Fu+19]
10 is to use **cyclical annealing**, which repeats the process of increasing β multiple times. This ensures
11 the progressive learning of more meaningful latent codes, by leveraging good representations learned
12 in a previous cycle as a way to warmstart the optimization.

13

14

15 **22.4.2 Lower bounding the rate**

16 An alternative approach is to stick with the original unmodified ELBO objective, but to prevent
17 the rate (i.e., the $D_{\text{KL}}(q\|p)$ term) from collapsing to 0, by limiting the flexibility of q . For example,
18 [XD18; Dav+18] use a von Mises-Fisher (Section 2.4.2) prior and posterior, instead of a Gaussian,
19 and they constrain the posterior to have a fixed concentration, $q(\mathbf{z}|\mathbf{x}) = \text{vMF}(\mathbf{z}|\boldsymbol{\mu}(\mathbf{x}), \kappa)$. Here
20 the parameter κ controls the rate of the code. The **δ -VAE** method [Oor+19] uses a Gaussian
21 autoregressive prior and a diagonal Gaussian posterior. We can ensure the rate is at least δ by
22 adjusting the regression parameter of the AR prior.

23

24 **22.4.3 Free bits**

25 In this section, we discuss the method of **free bits** [Kin+16], which is another way of lower bounding
26 the rate. To explain this, consider a fully factorized posterior in which the KL penalty has the form

$$\mathcal{L}_R = \sum_i D_{\text{KL}}(q_\phi(z_i|\mathbf{x})\|p_\theta(z_i)) \quad (22.109)$$

31

32 where z_i is the i 'th dimension of \mathbf{z} . We can replace this with a hinge loss, that will give up driving
33 down the KL for dimensions that are already beneath a target compression rate λ :

34

$$\mathcal{L}'_R = \sum_i \max(\lambda, D_{\text{KL}}(q_\phi(z_i|\mathbf{x})\|p_\theta(z_i))) \quad (22.110)$$

35

36 Thus the bits where the KL is sufficiently small “are free”, since the model does not have to “pay” to
37 encode them according to the prior.

38

39 7. Note that [Luc+19; DWW20] show that posterior collapse can also happen in linear VAE models, where the ELBO
40 corresponds to the exact marginal likelihood, so the problem is not only due to powerful (nonlinear) decoders, but is
41 also related to spurious local maxima in the objective.

42

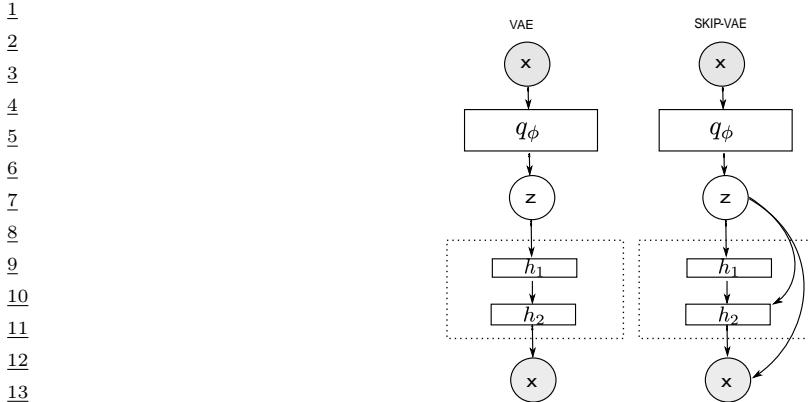


Figure 22.15: (a) VAE. (b) Skip-VAE. From Figure 1 of [Die+19a]. Used with kind permission of Adji Dieng.

22.4.4 Adding skip connections

One reason for latent variable collapse is that the latent variables z are not sufficiently “connected to” the observed data x . One simple solution is to modify the architecture of the generative model by adding **skip connections**, similar to a residual network (Section 16.2.4), as shown in Figure 22.15. This is called a **skip-VAE** [Die+19a].

22.4.5 Improved variational inference

The posterior collapse problem is caused in part by the poor approximation to the posterior. In [He+19], they proposed to keep the the model and VAE objective unchanged, but to more aggressively update the inference network before each step of generative model fitting. This enables the inference network to capture the current true posterior more faithfully, which will encourage the generator to use the latent codes when it is useful to do so.

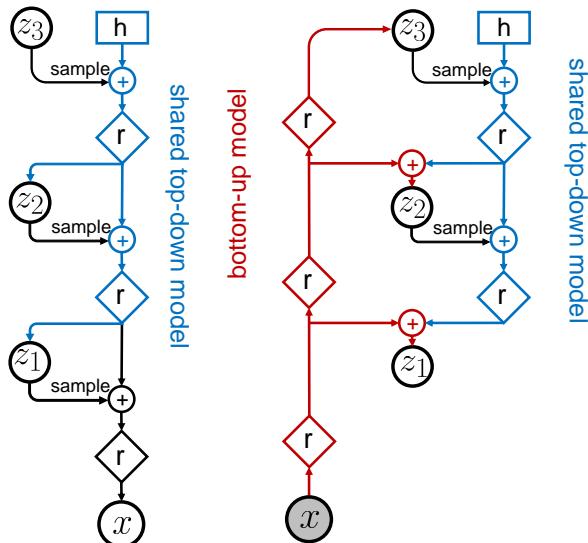
However, this only addresses the part of posterior collapse that is due to the amortization gap [CLD18], rather than the more fundamental problem of variational pruning, in which the KL term penalizes the model if its posterior deviates too far from the prior, which is often to simple to match the aggregated posterior.

Another way to ameliorate variational pruning is to use lower bounds than are tighter than the vanilla ELBO (Section 10.5), or more accurate posterior approximations (Section 10.4), or more accurate (hierarchical) generative models (Section 22.5).

22.4.6 Alternative objectives

An alternative to the above methods is to replace the ELBO objective with other objectives, such as the InfoVAE objective discussed in Section 22.3.3, which includes adversarial autoencoders and MMD autoencoders as special cases. The InfoVAE objective includes a term to explicit enforce non-zero mutual information between x and z , which effectively solves the problem of posterior collapse.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19



20 *Figure 22.16: Hierarchical VAEs with 3 stochastic layers. Left: Generative model. Right: Inference network.*
 21 *Diamond is a residual network, \oplus is feature combination (e.g., concatenation), and h is a trainable parameter.*
 22 *We first do bottom-up inference, by propagating x up to z_3 to compute $z_3^s \sim q_\phi(z_3|x)$, and then we perform*
 23 *top-down inference by computing $z_2^s \sim q_\phi(z_2|z_3, z_3^s)$ and then $z_1^s \sim q_\phi(z_1|z_1, z_{2:3}^s)$. From Figure 2 of [VK20].*
 24 *Used with kind permission of Arash Vahdat.*

25

26

27

28

29

30

31

32

33 22.4.7 Enforcing identifiability

34

35 In [WBC21], they show that posterior collapse happens whenever the latent variables are not uniquely
 36 identifiable given the data. Thus it can happen even in simple models, like a GMM, and even using
 37 exact inference methods. To see why, recall that non-identifiability of the latent variable means for
 38 each z , and all θ , there is some $z' \neq z$ such that $p(x|z, \theta) = p(x|z', \theta) = p(x|\theta)$, so the likelihood is
 39 invariant to the latent code.⁸ Consequently the posterior collapses to the prior:

40

41

$$42 \quad p(z|x, \theta) \propto p(z)p(x|z, \theta) = p(z)p(x|\theta) \propto p(z) \quad (22.111)$$

43

44

45 To enforce identifiability of z , the authors propose to use monotone transport maps, and input-
 46 convex neural networks. Unfortunately this makes inference slower. See the paper for details.

47

22.5 VAEs with hierarchical structure

We define a **hierarchical VAE**, with L stochastic layers, to be the following generative model:⁹

$$p_{\theta}(\mathbf{x}, \mathbf{z}_{1:L}) = p_{\theta}(\mathbf{z}_L) \left[\prod_{l=L-1}^1 p_{\theta}(\mathbf{z}_l | \mathbf{z}_{l+1}) \right] p_{\theta}(\mathbf{x} | \mathbf{z}_1) \quad (22.112)$$

We can improve on the above model by making it non-Markovian, i.e., letting each \mathbf{z}_l depend on all the higher level stochastic variables, $\mathbf{z}_{l+1:L}$, not just the preceding level, i.e.,

$$p_{\theta}(\mathbf{x}, \mathbf{z}) = p_{\theta}(\mathbf{z}_L) \left[\prod_{l=L-1}^1 p_{\theta}(\mathbf{z}_l | \mathbf{z}_{l+1:L}) \right] p_{\theta}(\mathbf{x} | \mathbf{z}_{1:L}) \quad (22.113)$$

Note that the likelihood is now $p_{\theta}(\mathbf{x} | \mathbf{z}_{1:L})$ instead of just $p_{\theta}(\mathbf{x} | \mathbf{z}_1)$. This is analogous to adding skip connections from all preceding variables to all their children. It is easy to implement this by using a deterministic “backbone” of residual connections, that accumulates all stochastic decisions, and propagates them down the chain, as illustrated in Figure 22.16(left). We discuss how to perform inference and learning in such models below.

22.5.1 Bottom-up vs top-down inference

To perform inference in a hierarchical VAE, we could use a **bottom-up inference model** of the form

$$q_{\phi}(\mathbf{z} | \mathbf{x}) = q_{\phi}(\mathbf{z}_1 | \mathbf{x}) \prod_{l=2}^L q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{1:l-1}) \quad (22.114)$$

However, a better approach is to use a **top-down inference model** of the form

$$q_{\phi}(\mathbf{z} | \mathbf{x}) = q_{\phi}(\mathbf{z}_L | \mathbf{x}) \prod_{l=L-1}^1 q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{l+1:L}) \quad (22.115)$$

Inference for \mathbf{z}_l combines bottom-up information from \mathbf{x} with top-down information from higher layers, $\mathbf{z}_{>l} = \mathbf{z}_{l+1:L}$. See Figure 22.16(right) for an illustration.¹⁰

With the above model, the ELBO can be written as follows:

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) = \mathbb{E}_{q_{\phi}(\mathbf{z} | \mathbf{x})} [\log p_{\theta}(\mathbf{x} | \mathbf{z})] - D_{\text{KL}}(q_{\phi}(\mathbf{z}_L | \mathbf{x}) \| p_{\theta}(\mathbf{z}_L)) \quad (22.116)$$

$$- \sum_{l=L-1}^1 \mathbb{E}_{q_{\phi}(\mathbf{z}_{>l} | \mathbf{x})} [D_{\text{KL}}(q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{>l}) \| p_{\theta}(\mathbf{z}_l | \mathbf{z}_{>l}))] \quad (22.117)$$

8. Note that identifiability of \mathbf{z} is a weaker requirement compared to the whole model being identifiable, including the parameters.

9. There is a split in the literature about whether to label the top level as \mathbf{z}_L or \mathbf{z}_1 . We adopt the former convention, since we view lower numbered layers, such as \mathbf{z}_1 , as being “closer to the data”, and higher numbered layers, such as \mathbf{z}_L , as being “more abstract”.

10. Note that it is also possible to have a stochastic bottom-up encoder and a stochastic top-down encoder, as discussed in the **BIVA** paper [Maa+19]. (BIVA stands for “Bidirectional-Inference Variational Autoencoder.”)

1 where

2

$$\underline{q}_{\phi}(\mathbf{z}_{>l|\mathbf{x}}) = \prod_{i=l+1}^L q_{\phi}(\mathbf{z}_i|\mathbf{x}, \mathbf{z}_{>i}) \quad (22.118)$$

3 is the approximate posterior above layer l (i.e., the parents of \mathbf{z}_l).

4 The reason the top-down inference model is better is that it more closely approximates the true
5 posterior of a given layer, which is given by

6

$$p_{\theta}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L}) \propto p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L}) p_{\theta}(\mathbf{x}|\mathbf{z}_l, \mathbf{z}_{l+1:L}) \quad (22.119)$$

7 Thus the posterior combines the top-down prior term $p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L})$ with the bottom-up likelihood
8 term $p_{\theta}(\mathbf{x}|\mathbf{z}_l, \mathbf{z}_{l+1:L})$. We can approximate this posterior by defining

9

$$q_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L}) \propto p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L}) \tilde{q}_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L}) \quad (22.120)$$

10 where $\tilde{q}_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L})$ is a learned Gaussian approximation to the bottom-up likelihood. If both prior
11 and likelihood are Gaussian, we can compute this product in closed form, as proposed in the **ladder**
12 **network** paper [Sn+16; Søn+16].¹¹ A more flexible approach is to let $q_{\phi}(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{l+1:L})$ be learned,
13 but to force it to share some of its parameters with the learned prior $p_{\theta}(\mathbf{z}_l|\mathbf{z}_{l+1:L})$, as proposed in
14 [Kin+16]. This reduces the number of parameters in the model, and ensures that the posterior and
15 prior remain somewhat close.

24 22.5.2 Example: Very deep VAE

25 There have been many papers exploring different kinds of HVAE models (see e.g., [Kin+16; Sn+16;
26 Chi21; VK20; Maa+19]), and we do not have space to discuss them all. Here we focus on the “**very**
27 **deep VAE**” or **VD-VAE** model of [Chi21], since it is simple but yields state of the art results (at
28 the time of writing).

29 The architecture is a simple convolutional VAE with bidirectional inference, as shown in Figure 22.17.
30 For each layer, the prior and posterior are diagonal Gaussians. The author found that nearest-neighbor
31 upsampling (in the decoder) worked much better than transposed convolution, and avoided posterior
32 collapse. This enabled training with the vanilla VAE objective, without needing any of the tricks
33 discussed in Section 22.5.4.

34 The low-resolution latents (at the top of the hierarchy) capture a lot of the global structure of
35 each image; the remaining high-resolution latents are just used to fill in details, that make the image
36 look more realistic, and improve the likelihood. This suggests the model could be useful for lossy
37 compression, since a lot of the low-level details can be drawn from the prior (i.e., “hallucinated”),
38 rather than having to be sent by the encoder. We illustrate this in Figure 22.18 using a small model
39 trained on CIFAR-10, which is a dataset of 60,000 32x32 color images from 10 classes. If we just use
40 the top level code (of size $1 \times 1 \times C$, where $C = 384$ channels), we cannot reliably encode the input
41 image. (For example, the jet fighter (airplane) gets reconstructed as a car.) However, we start to get
42 “plausible hallucinations” after using just the top 3 levels (of size $8 \times 8 \times C$). (Similar results are
43 shown in [VK20].)

44 ¹¹ The term “ladder network” arises from the horizontal “rungs” in Figure 22.16(right). Note that a similar idea was
45 independently proposed in [Sal16].

46

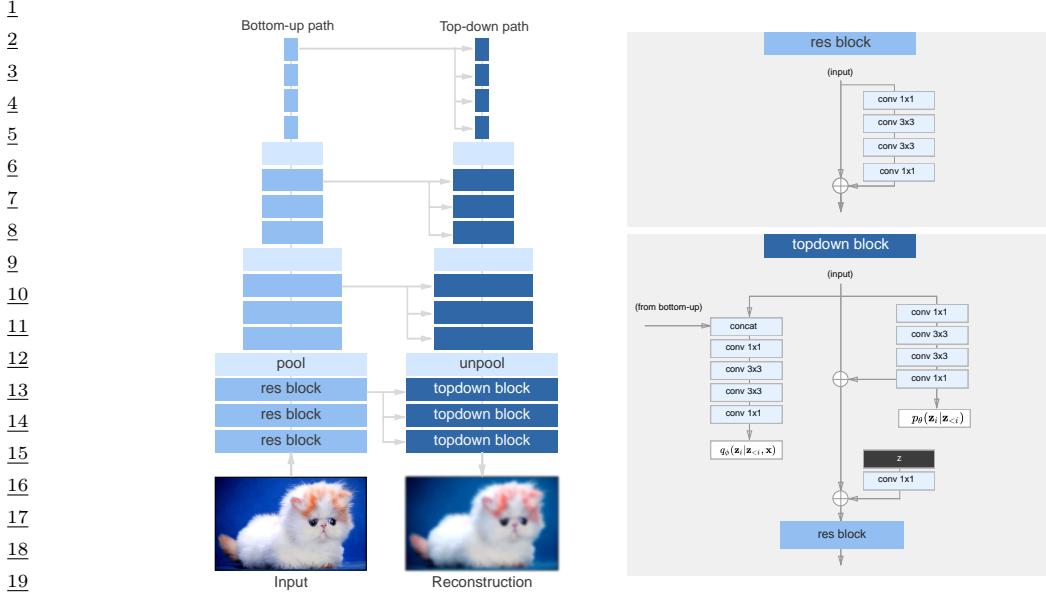


Figure 22.17: The top-down encoder used by the hierarchical VAE in [Chi21]. Each convolution is preceded by the GELU nonlinearity. The model uses average pooling and nearest-neighbor upsampling for the pool and unpool layers. The posterior q_ϕ and prior p_θ are diagonal Gaussians. From Figure 3 of [Chi21]. Used with kind permission of Ronan Child.

We can also use the model for unconditional sampling at multiple resolutions. This is illustrated in Figure 22.19, using a model with 78 stochastic layers trained on the FFHQ-256 dataset.¹²

22.5.3 Connection with autoregressive models

Until recently, most hierarchical VAEs only had a small number of stochastic layers. Consequently the images they generated have not looked as good, or had as high likelihoods, as images produced by other models, such as the autoregressive PixelCNN model (see Section 23.3.2). However, by endowing VAEs with many more stochastic layers, it is possible to outperform AR models in terms of likelihood and sample quality, while using fewer parameters and much less computing power [Chi21; VK20; Maa+19].

To see why this is possible, note that we can represent any AR model as a degenerate VAE, as shown in Figure 22.20(left). The idea is simple: the encoder copies the input into latent space by setting $\mathbf{z}_{1:D} = \mathbf{x}_{1:D}$ (so $q_\phi(z_i = x_i | \mathbf{z}_{>i}, \mathbf{x}) = 1$), then the model learns an autoregressive prior $p_\theta(\mathbf{z}_{1:D}) = \prod_d p(z_d | \mathbf{z}_{1:d-1})$, and finally the likelihood function just copies the latent vector to output space, so $p_\theta(x_i = z_i | \mathbf{z}) = 1$. Since the encoder computes the exact (albeit degenerate) posterior, we

12. This is a 256^2 version of the Flickr-Faces High Quality dataset from <https://github.com/NVlabs/ffhq-dataset>, which has 80k images at 1024^2 resolution.

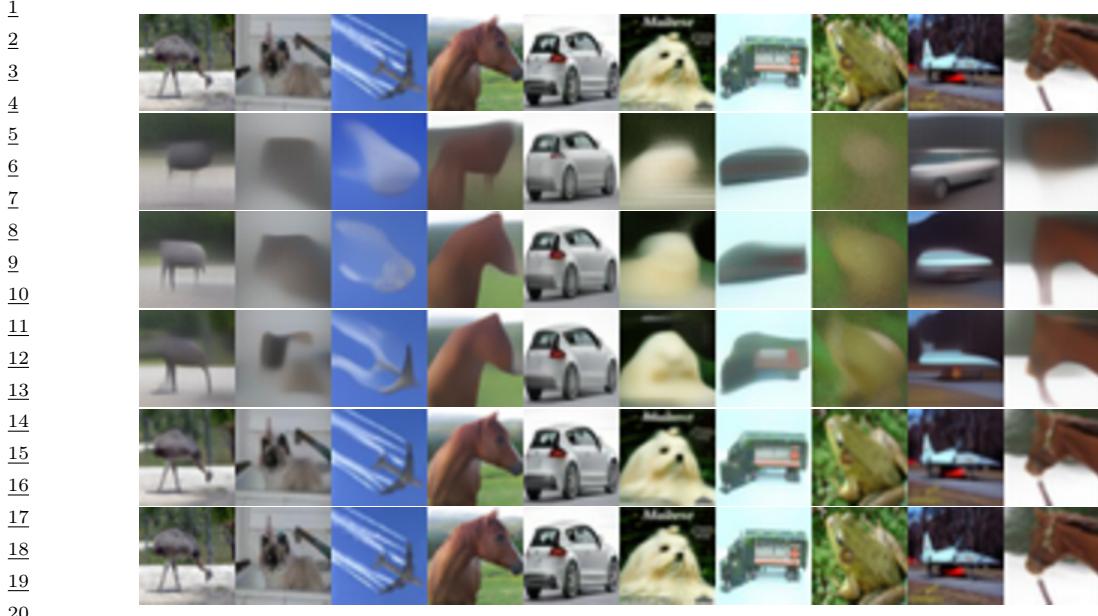


Figure 22.18: Reconstruction from the VD-VAE model trained on CIFAR10 using different numbers of latent layers. The latent code is a sample from the posterior for the first $L_1 < L$ layers, and then is sampled from the prior (ignoring higher levels) at a low temperature; this emulates models of different stochastic depth. Top row: input images. Subsequent rows: reconstructing down to resolutions 1, 4, 8, 16, 32. Generated by `vdvae_flax_demo_cifar.ipynb`.



Figure 22.19: Samples from a VDVAE model (trained on FFHQ dataset) from different levels of the hierarchy. From Figure 1 of [Chi21]. Used with kind permission of Ronan Child.

have $q_{\phi}(\mathbf{z}|\mathbf{x}) = p_{\theta}(\mathbf{z}|\mathbf{x})$, so the ELBO is tight and reduces to the log likelihood,

$$\log p_{\theta}(\mathbf{x}) = \log p_{\theta}(\mathbf{z}) = \sum_d \log p_{\theta}(x_d | \mathbf{x}_{<d}) \quad (22.121)$$

Thus we can emulate any AR model with a VAE providing it has at least D stochastic layers, where D is the dimensionality of the observed data.

In practice data usually lives in a lower-dimensional manifold (see e.g., [DW19]), which can allow for a much more compact latent code. For example, Figure 22.20(right) shows a hierarchical code in which the latent factors at the lower level are conditionally independent given the higher level,

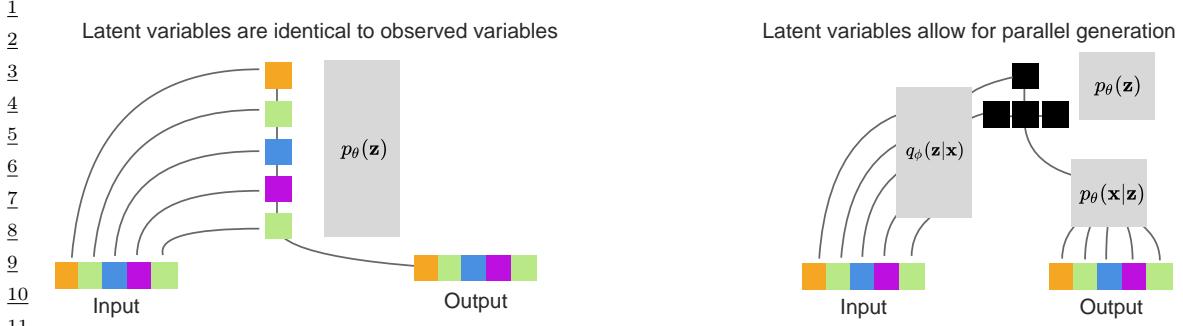


Figure 22.20: Left: a hierarchical VAE which emulates an autoregressive model using an identity encoder, autoregressive prior, and identity decoder. Right: a hierarchical VAE with a 2 layer hierarchical latent code. The bottom hidden nodes (black) are conditionally independent given the top layer. From Figure 2 of [Chi21]. Used with kind permission of Rewon Child.

and hence can be generated in parallel. Such a tree-like structure can enable sample generation in $O(\log D)$ time, whereas an autoregressive model always takes $O(D)$ time. (Recall that for an image D is the number of pixels, so it grows quadratically with image resolution. For example, even a tiny 32x32 image has $D = 3072$.)

In addition to speed, hierarchical models also require many fewer parameters than “flat” models. The typical architecture used for generating images is a **multi-scale** approach: the model starts from a small, spatially arranged set of latent variables, and at each subsequent layer, the spatial resolution is increased (usually by a factor of 2). This allows the high level to capture global, long-range correlations (e.g., the symmetry of a face, or overall skin tone), while letting lower levels capture fine-grained details.

22.5.4 Variational pruning

A common problem with hierarchical VAEs is that the higher level latent layers are often ignored, so the model does not learn interesting high level semantics. This is caused by **variational pruning**. This problem is analogous to the issue of latent variable collapse, which we discussed in Section 22.4 in the context of a single layer of latents with expressive decoders, such as RNNs; in the context of HVAEs, the “expressive decoder” corresponds to lower layers of the hierarchy.

A common heuristic to mitigate this problem is to use KL balancing coefficients [Che+17b], to ensure that an equal amount of information is encoded in each layer. That is, we use the following penalty:

$$\sum_{l=1}^L \gamma_l \mathbb{E}_{q(\mathbf{z}_{>l}|\mathbf{x})} [D_{\text{KL}}(q_\phi(\mathbf{z}_l|\mathbf{x}, \mathbf{z}_{>l}) \| p_\theta(\mathbf{z}_l|\mathbf{z}_{>l}))] \quad (22.122)$$

The balancing term γ_l is set to a small value when the KL penalty is small (on the current minibatch), to encourage use of that layer, and is set to a large value when the KL term is large. (This is only done during the “warmup period”.) Concretely, [VK20] proposes to set the coefficients γ_l to be

¹ proportional to the size of the layer, s_l , and the average KL loss:

$$\gamma_l \propto s_l \mathbb{E}_{\mathbf{x} \sim \mathcal{B}} [\mathbb{E}_{q(\mathbf{z}_{>l} | \mathbf{x})} [D_{\text{KL}}(q_{\phi}(\mathbf{z}_l | \mathbf{x}, \mathbf{z}_{>l}) \| p_{\theta}(\mathbf{z}_l | \mathbf{z}_{>l}))]] \quad (22.123)$$

⁵ where \mathcal{B} is the current minibatch.

⁷ 22.5.5 Other optimization difficulties

⁹ A common problem when training (hierarchical) VAEs is that the loss can become unstable. The
¹⁰ main reason for this is that the KL term is unbounded (can become infinitely large). In [Chi21], they
¹¹ tackle the problem in two ways. First, ensure the initial random weights of the final convolutional
¹² layer in each residual bottleneck block get scaled by $1/\sqrt{L}$. Second, skip an update step if the norm
¹³ of the gradient of the loss exceeds some threshold.

¹⁴ In the **Nouveau VAE** method of [VK20], they use some more complicated measures to ensure
¹⁵ stability. First they use batch normalization, but with various tweaks. Second they use spectral
¹⁶ regularization for the encoder. Specifically they add the penalty $\beta \sum_i \lambda_i$, where λ_i is the largest
¹⁷ singular value of the i 'th convolutional layer (estimated using a single power iteration step), and
¹⁸ $\beta \geq 0$ is a tuning parameter. Third, they use inverse autoregressive flows (Section 24.2.4.3) in each
¹⁹ layer, instead of a diagonal Gaussian approximation. Fourth, they represent the posterior using a
²⁰ residual representation. In particular, let us assume the prior for the i 'th variable in layer l is

$$p_{\theta}(z_l^i | \mathbf{z}_{>l}) = \mathcal{N}(z_l^i | \mu_i(\mathbf{z}_{>l}), \sigma_i(\mathbf{z}_{>l})) \quad (22.124)$$

²³ They propose the following posterior approximation:

$$q_{\phi}(z_l^i | \mathbf{x}, \mathbf{z}_{>l}) = \mathcal{N}(z_l^i | \mu_i(\mathbf{z}_{>l}) + \Delta\mu_i(\mathbf{z}_{>l}, \mathbf{x}), \sigma_i(\mathbf{z}_{>l}) \cdot \Delta\sigma_i(\mathbf{z}_{>l}, \mathbf{x})) \quad (22.125)$$

²⁷ where the Δ terms are the relative changes computed by the encoder. The corresponding KL penalty
²⁸ reduces to the following (dropping the l subscript for brevity):

$$D_{\text{KL}}(q_{\phi}(z^i | \mathbf{x}, \mathbf{z}_{>l}) \| p_{\theta}(z^i | \mathbf{z}_{>l})) = \frac{1}{2} \left(\frac{\Delta\mu_i^2}{\sigma_i^2} + \Delta\sigma_i^2 - \log \Delta\sigma_i^2 - 1 \right) \quad (22.126)$$

³³ So as long as σ_i is bounded from below, the KL term can be easily controlled just by adjusting the
³⁴ encoder parameters.

³⁶ 22.6 Vector quantization VAE

³⁸ In this section, we describe **VQ-VAE**, which stands for “vector quantized VAE” [OVK17; ROV19].
³⁹ This is like a standard VAE except it uses a set of discrete latent variables.

⁴⁰

⁴¹ 22.6.1 Autoencoder with binary code

⁴³ The simplest approach to the problem is to construct a standard VAE, but to add a discretization
⁴⁴ layer at the end of the encoder, $\mathbf{z}_e(\mathbf{x})$. For example, we can binarize the latent vector using a ReLU
⁴⁵ function, so the code becomes $\mathbf{z} = \text{ReLU}(\mathbf{z}_e(\mathbf{x})) \in \{0, 1\}^K$. This can be useful for data compression
⁴⁶ (see e.g., [BLS17]).

⁴⁷



Figure 22.21: Autoencoder for MNIST using 256 binary latents. Top row: input images. Middle row: reconstruction. Bottom row: latent code, reshaped to a 16×16 image. Generated by [quantized_autoencoder_mnist.ipynb](#).

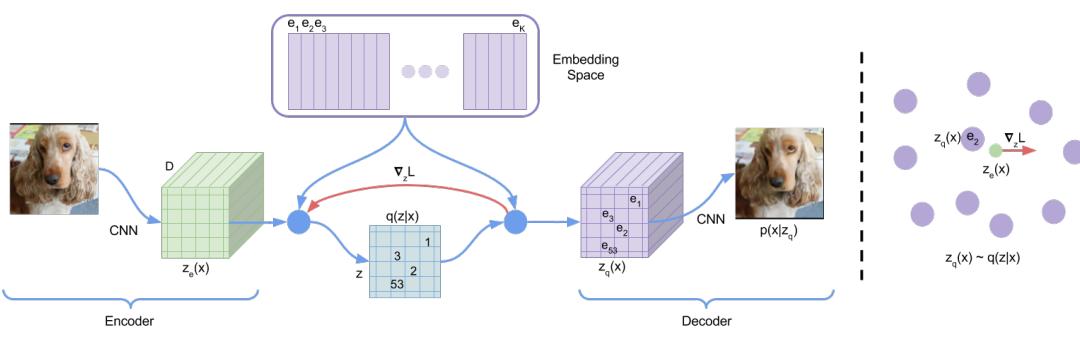


Figure 22.22: VQ-VAE architecture. From Figure 1 of [OVK17]. Used with kind permission of Aaron van den Oord.

Suppose we assume the prior over the latent codes is uniform. Since the encoder is deterministic, the KL divergence reduces to a constant, equal to $\log K$. This avoids the problem with posterior collapse (Section 22.4). Unfortunately, the discreteness of the encoder prohibits the direct use of gradient based optimization. The solution proposed in [OVK17] is to use the straight-through estimator, which we discuss in Section 6.6.8. We show a simple example of this approach in Figure 22.21, where we use a Gaussian likelihood, so the loss function has the form

$$\mathcal{L} = \|\mathbf{x} - D(E(\mathbf{x}))\|_2^2 \quad (22.127)$$

where $E(\mathbf{x}) \in \{0, 1\}^K$ is the encoder, and $D(\mathbf{z}) \in \mathbb{R}^{28 \times 28}$ is the decoder.

22.6.2 VQ-VAE model

We can get a more expressive model by using a 3d tensor of discrete latents, $\mathbf{z} \in \mathbb{R}^{H \times W \times K}$, where K is the number of discrete values per latent variable. Rather than just binarizing the continuous

¹ vector $\mathbf{z}_e(\mathbf{x})_{ij}$, we compare it to a **codebook** of embedding vectors, $\{\mathbf{e}_k : k = 1 : K, \mathbf{e}_k \in \mathbb{R}^L\}$, and
² then set \mathbf{z}_{ij} to the index of the nearest codebook entry:
³

$$\begin{aligned} q(\mathbf{z}_{ij} = k | \mathbf{x}) &= \begin{cases} 1 & \text{if } k = \operatorname{argmin}_{k'} \|\mathbf{z}_e(\mathbf{x})_{i,j,:} - \mathbf{e}_{k'}\|_2 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (22.128)$$

⁴ When reconstructing the input we replace each discrete code index by the corresponding real-valued
⁵ codebook vector:
⁶

$$\begin{aligned} \mathbf{z}_q &= \mathbf{e}_k \text{ where } \mathbf{z}_{ij} = k \end{aligned} \quad (22.129)$$

⁷ These values are then passed to the decoder, $p(\mathbf{x} | \mathbf{z}_q)$, as usual. See Figure 22.22 for an illustration of
⁸ the overall architecture. Note that although \mathbf{z}_q is generated from a discrete combination of codebook
⁹ vectors, the use of a distributed code makes the model very expressive. For example, if we use a
¹⁰ grid of 32×32 , with $K = 512$, then we can generate $512^{32 \times 32} = 2^{9216}$ distinct images, which is
¹¹ astronomically large.

¹² To fit this model, we can minimize the negative log likelihood (reconstruction error) using the
¹³ straight-through estimator, as before. This amounts to passing the gradients from the decoder input
¹⁴ $\mathbf{z}_q(\mathbf{x})$ to the encoder output $\mathbf{z}_e(\mathbf{x})$, bypassing Equation (22.128), as shown by the red arrow in
¹⁵ Figure 22.22. Unfortunately this means that the codebook entries will not get any learning signal.
¹⁶ To solve this, the authors proposed to add an extra term to the loss, known as the **codebook loss**,
¹⁷ that encourages the codebook entries \mathbf{e} to match the output of the encoder. We treat the encoder
¹⁸ $\mathbf{z}_e(\mathbf{x})$ as a fixed target, by adding a **stop gradient** operator to it; this ensures \mathbf{z}_e is treated normally
¹⁹ in the forwards pass, but has zero gradient in the backwards pass. The modified loss (dropping the
²⁰ spatial indices i, j) becomes

$$\begin{aligned} \mathcal{L} &= -\log p(\mathbf{x} | \mathbf{z}_q(\mathbf{x})) + \|\mathbf{sg}(\mathbf{z}_e(\mathbf{x})) - \mathbf{e}\|_2^2 \end{aligned} \quad (22.130)$$

²¹ where \mathbf{e} refers to the codebook vector assigned to $\mathbf{z}_e(\mathbf{x})$.

²² An alternative way to update the codebook vectors is to use moving averages. To see how this
²³ works, first consider the batch setting. Let $\{\mathbf{z}_{i,1}, \dots, \mathbf{z}_{i,n_i}\}$ be the set of n_i outputs from the encoder
²⁴ that are closest to the dictionary item \mathbf{e}_i . We can update \mathbf{e}_i to minimize the MSE

$$\sum_{j=1}^{n_i} \|\mathbf{z}_{i,j} - \mathbf{e}_i\|_2^2 \quad (22.131)$$

²⁵ which has the closed form update

$$\mathbf{e}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{z}_{i,j} \quad (22.132)$$

²⁶ This is like the M step of the EM algorithm when fitting the mean vectors of a GMM. In the minibatch
²⁷ setting, we replace the above operations with an exponentially moving average, as follows:

$$N_i^t = \gamma N_i^{t-1} + (1 - \gamma) n_i^t \quad (22.133)$$

$$\mathbf{m}_i^t = \gamma \mathbf{m}_i^{t-1} + (1 - \gamma) \sum_j \mathbf{z}_{i,j}^t \quad (22.134)$$

$$\mathbf{e}_i^t = \frac{\mathbf{m}_i^t}{N_i^t} \quad (22.135)$$

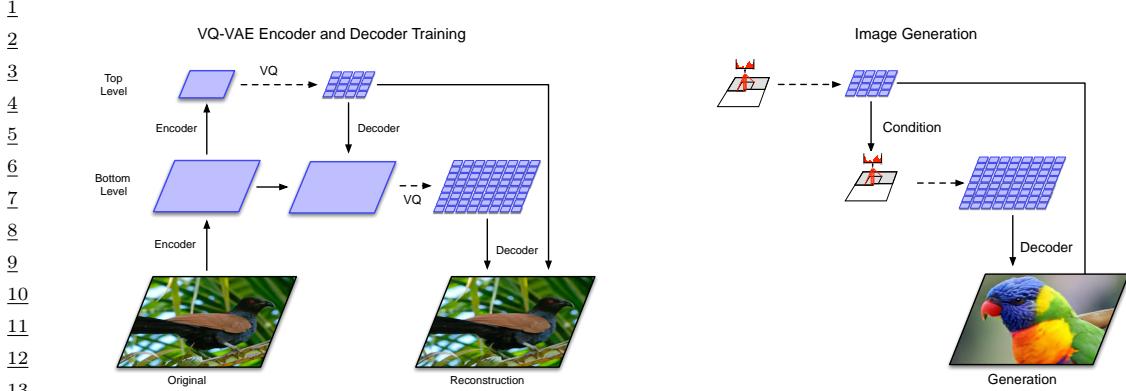


Figure 22.23: Hierarchical extension of VQ-VAE. (a) Encoder and decoder architecture. (b) Combining a Pixel-CNN prior with the decoder. From Figure 2 of [ROV19]. Used with kind permission of Aaron van den Oord.

The authors found $\gamma = 0.9$ to work well.

The above procedure will learn to update the codebook vectors so it match the output of the encoder. However, it is also important to ensure the encoder does not “change its mind” too often about what codebook value to use. To prevent this, the authors propose to add a third term to the loss, known as the **commitment loss**, that encourages the encoder output to be close to the codebook values. Thus we get the final loss:

$$\mathcal{L} = -\log p(\mathbf{x}|\mathbf{z}_q(\mathbf{x})) + \|\text{sg}(\mathbf{z}_e(\mathbf{x})) - \mathbf{e}\|_2^2 + \beta \|\mathbf{z}_e(\mathbf{x}) - \text{sg}(\mathbf{e})\|_2^2 \quad (22.136)$$

The authors found $\beta = 0.25$ to work well, although of course the value depends on the scale of the reconstruction loss (NLL) term. (A probabilistic interpretation of this loss can be found in [Hen+18].) Overall, the decoder optimizes the first term only, the encoder optimizes the first and last term, and the embeddings optimize the middle term.

22.6.3 Learning the prior

After training the VQ-VAE model, it is possible to learn a better prior, to match the aggregated posterior. To do this, we just apply the encoder to a set of data, $\{\mathbf{x}_n\}$, thus converting them to discrete sequences, $\{\mathbf{z}_n\}$. We can then learn a joint distribution $p(\mathbf{z})$ using any kind of sequence model. In the original VQ-VAE paper [OVK17], they used the causal convolutional PixelCNN model (Section 23.3.2). More recent work has used transformer decoders (Section 23.4). Samples from this prior can then be decoded using the decoder part of the VQ-VAE model. We give some examples of this in the sections below.

22.6.4 Hierarchical extension (VQ-VAE-2)

In [ROV19], they extend the original VQ-VAE model by using a hierarchical latent code. The model is illustrated in Figure 22.23. They applied this to images of size $256 \times 256 \times 3$. The first latent layer



14 *Figure 22.24: Some images generated by a class conditional VQ-VAE-2 model trained on Imagenet. Row 1:*
15 *Label = “Pekinese”.* Row 1: *Label = “Papillon”.* Row 2: *Label = “Drake”* Row 3: *Label = “spotted salamander”.*
16 *From Figure 4 of [ROV19]. Used with kind permission of Aaron van den Oord.*

171819

20 maps this to a quantized representation of size 64×64 , and the second latent layer maps this to a
21 quantized representation of size 32×32 . This hierarchical scheme allows the top level to focus on
22 high level semantics of the image, leaving fine visual details, such as texture, to the lower level. (See
23 Section 22.5 for more discussion of hierarchical VAEs.)

24 After fitting the VQ-VAE, they learn a prior over the top level code using a PixelCNN model
25 augmented with self-attention (Section 16.2.7), to capture long-range dependencies. (This hybrid
26 model is known as **PixelSNAIL** [Che+17c].) For the lower level prior, they just use standard
27 PixelCNN, since attention would be too expensive. Samples from the model can then be decoded
28 using the VQ-VAE decoder, as shown in Figure 22.23.

29 Some samples from a class-conditional version of this model, after training on ImageNet at 256×256
30 resolution, are shown in Figure 22.24. They show good visual quality and diversity.

3132

33 22.6.5 Discrete VAE

34 In VQ-VAE, we use a one-hot encoding for the latents, $q(z = k|\mathbf{x}) = 1$ iff $k = \text{argmin}_k \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_k\|_2$,
35 and then set $\mathbf{z}_q = \mathbf{e}_k$. This does not capture any uncertainty in the latent code, and requires the use
36 of the straight-through estimator for training.

37 Various other approaches to fitting VAEs with discrete latent codes have been investigated. In the
38 DALL-E paper (Section 23.4.3), they use a fairly simple method, based on using the Gumbel-Softmax
39 relaxation for the discrete variables (see Section 6.6.6). In brief, let $q(z = k|\mathbf{x})$ be the probability
40 that the input \mathbf{x} is assigned to codebook entry i . We can exactly sample $w_k \sim q(z = k|\mathbf{x})$ from this
41 by computing $w_k = \text{argmax}_i g_k + \log q(z = k|\mathbf{x})$, where each g_k is from a Gumbel distribution. We
42 can now “relax” this by using a softmax with temperature $\tau > 0$ and computing

$$\begin{aligned} w_k &= \frac{\exp(\frac{g_k + \log q(z=k|\mathbf{x})}{\tau})}{\sum_{j=1}^K \exp(\frac{g_j + \log q(z=j|\mathbf{x})}{\tau})} \end{aligned} \tag{22.137}$$

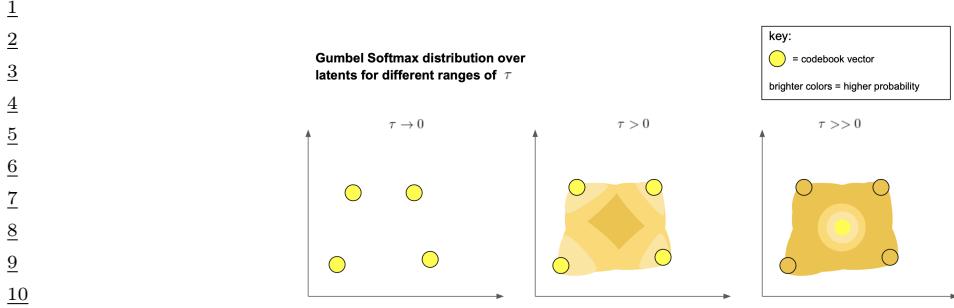


Figure 22.25: Illustration of the Gumbel Softmax trick applied to $K = 4$ codebook vectors in $L = 2$ dimensions. From <https://ml.berkeley.edu/blog/posts/dalle2/>. Used with kind permission of Charlie Snell.

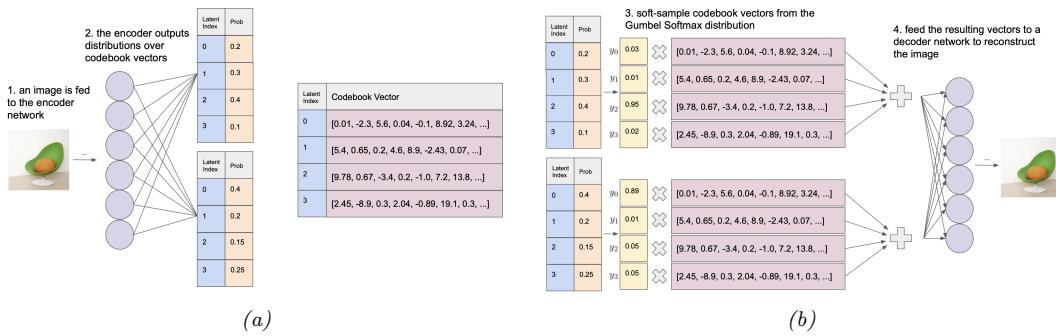


Figure 22.26: Illustration of the dVAE model (a) Encoder. (b) Decoder. From <https://ml.berkeley.edu/blog/posts/dalle2/>. Used with kind permission of Charlie Snell.

We now set the latent code to be a weighted sum of the codebook vectors:

$$\mathbf{z}_q = \sum_{k=1}^K w_k \mathbf{e}_k \quad (22.138)$$

In the limit that $\tau \rightarrow 0$, the distribution over weights \mathbf{w} converges to a one-hot distribution, in which case \mathbf{z} becomes equal to one of the codebook entries. But for finite τ , we “fill in” the space between the vectors, as illustrated in Figure 22.25.

This allows us to express the ELBO in the usual differentiable way:

$$\mathcal{L} = -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z})] + \beta D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \quad (22.139)$$

where $\beta > 0$ controls the amount of regularization. (Unlike VQ-VAE, the KL term is not a constant, because the encoder is stochastic.) Furthermore, since the Gumbel noise variables are sampled from a distribution that is independent of the encoder parameters, we can use the reparameterization trick (Section 6.6.4) to optimize this.

The overall architecture is illustrated in Figure 22.26. (The “avocado chair” image is discussed in more detail on the section on DALL-E in Section 23.4.3.)

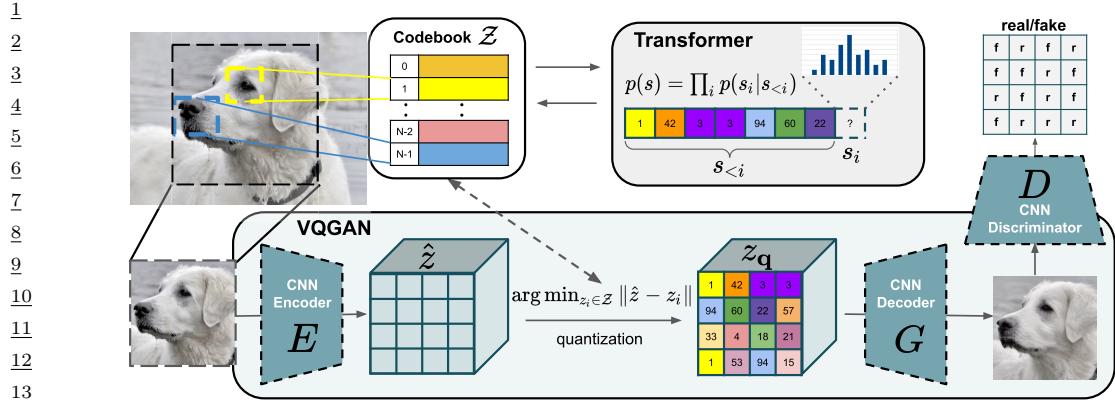


Figure 22.27: Illustration of the VQ-GAN. From Figure 2 of [ERO21]. Used with kind permission of Patrick Esser.

22.6.6 VQ-GAN

One drawback of VQ-VAE is that it uses mean squared error in its reconstruction loss, which can result in blurry samples. In the **VQ-GAN** paper [ERO21], they replace this with a (patch-wise) GAN loss (see Chapter 27), together with a perceptual loss; this results in much higher visual fidelity. In addition, they use a transform (see Section 16.3.4) to model the prior on the latent codes. See Figure 22.27 for a visualization of the overall model. In [Yu+21], they replace the CNN encoder and decoder of the VQ-GAN model with transformers, yielding improved results; they call this **VIM** (Vector-quantized Image Modeling).

28

22.7 Wake-sleep algorithm

30

So far in this chapter we have focused on fitting latent variable models by maximizing the ELBO. This has two main drawbacks. First, it does not work well when we have discrete latent variables, because in such cases we cannot use the reparameterization trick; thus we have to use higher variance estimators, such as REINFORCE (see Section 10.3.1). Second, even in the case where we can use the reparameterization trick, the lower bound may not be very tight. We can improve the tightness by using the IWAE multi-sample bound (Section 10.5.1), but paradoxically this may not result in learning a better model, for reasons discussed in Section 10.5.1.1.

In this section, we discuss a different way to jointly train generative and inference models, which avoids some of the problems with ELBO maximization. The method is known as the **wake-sleep algorithm** [Hin+95; BB15b; Le+19; FT19], because it alternates between two steps: in the wake phase, we optimize the generative model parameters θ to maximize the marginal likelihood of the observed data (we approximate $\log p_\theta(\mathbf{x})$ by drawing importance samples from the inference network); and in the sleep phase, we optimize the inference model parameters ϕ to learn to invert the generative model by training the inference network on labeled (\mathbf{x}, \mathbf{z}) pairs, where \mathbf{x} are samples generated by the current model parameters. This can be viewed as a form of **adaptive importance sampling**, which iteratively improves its proposal, while simultaneously optimizing the model. We give further

47

1 details below.

4 22.7.1 Wake phase

5 In the **wake phase**, we minimize the KL divergence from the empirical distribution to the model's
6 distribution:

$$\mathcal{L}(\boldsymbol{\theta}) = D_{\text{KL}}(p_{\mathcal{D}}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [-\log p_{\boldsymbol{\theta}}(\mathbf{x})] + \text{const} \quad (22.140)$$

10 where $p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z}$. This is equivalent to maximizing the likelihood of the observed
11 data:

$$LL(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x})] \quad (22.141)$$

14 Since the log marginal likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ cannot be computed exactly, we will approximate it. In
15 the original wake sleep paper, they proposed to use the ELBO lower bound. In the **reweighted wake**
16 **sleep** (RWS) algorithm of [BB15b; Le+19], they propose to use the IWAE bound from Section 10.5.1
17 instead. In particular, if we draw S samples from the inference network, $\mathbf{z}_s \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, we get the
18 following estimator:

$$LL(\boldsymbol{\theta}|\boldsymbol{\phi}, \mathbf{x}) = \log \left(\frac{1}{S} \sum_{s=1}^S w_s \right) \quad (22.142)$$

23 where $w_s = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s)}{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x})}$.

24 We now discuss how to compute the gradient of this objective. Using the log-derivative trick, we
25 have that

$$\nabla_{\boldsymbol{\theta}} \log w_s = \frac{1}{w_s} \nabla_{\boldsymbol{\theta}} w_s = \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_s) \quad (22.143)$$

29 Hence

$$\nabla_{\boldsymbol{\theta}} LL(\boldsymbol{\theta}|\boldsymbol{\phi}, \mathbf{x}) = \frac{1}{\frac{1}{S} \sum_{s=1}^S w_s} \left(\frac{1}{S} \sum_{s=1}^S \nabla_{\boldsymbol{\theta}} w_s \right) \quad (22.144)$$

$$= \frac{1}{\sum_{s=1}^S w_s} \left(\sum_{s=1}^S \frac{p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x})} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x}) \right) \quad (22.145)$$

$$= \sum_{s=1}^S \bar{w}_s \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}_s, \mathbf{x}) \quad (22.146)$$

39 where $\bar{w}_s = w_s / (\sum_{s'=1}^S w_{s'})$.

42 22.7.2 Sleep phase

43 In the **sleep phase**, we try to minimize the KL divergence between the true posterior (under the
44 current model) and the inference network's approximation to that posterior:

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{p(\mathbf{x})} [D_{\text{KL}}(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \| q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))] = \mathbb{E}_{p(\mathbf{z}, \mathbf{x})} [-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})] + \text{const} \quad (22.147)$$

¹ Equivalently, we can maximize the following loglikelihood objective:
²

$$\frac{3}{4} \quad LL(\phi|\theta) = \mathbb{E}_{(z,x) \sim p(z,x)} [\log q_\phi(z|x)] \quad (22.148)$$

⁵ where $p_\theta(z, x) = p_\theta(z)p_\theta(x|z)$. We see that the sleep phase amounts to maximum likelihood training
⁶ of the inference network based on samples from the generative model. These “fantasy samples”,
⁷ created while the network “dreams”, can be easily generated using ancestral sampling (Section 4.2.4).
⁸ If we use S such samples, the objective becomes

$$\frac{9}{10} \quad LL(\phi|\theta) = \frac{1}{S} \sum_{s=1}^S \log q_\phi(z'_s|x'_s) \quad (22.149)$$

¹² where $(z'_s, x'_s) \sim p_\theta(z, x)$. The gradient of this is given by
¹³

$$\frac{14}{15} \quad \nabla_\phi LL(\phi|\theta) = \frac{1}{S} \sum_{s=1}^S \nabla_\phi \log q_\phi(z'_s|x'_s) \quad (22.150)$$

¹⁷ We do not require $q_\phi(z'|x)$ to be reparameterizable, since the samples are drawn from a distribution
¹⁸ that is independent of ϕ . This means it is easy to apply this method to models with discrete latent
¹⁹ variables.

²⁰ Note that the technique of training an inference network to invert a known generative model
²¹ using supervised learning on generated samples is known as **inference compilation** [LBW17] or
²² **amortized inference** [GG14].
²³

²⁴ 22.7.3 Daydream phase

²⁵ The disadvantage of the sleep phase is that the inference network, $q_\phi(z|x)$, is trying to follow a
²⁶ moving target, $p_\theta(z|x)$. Furthermore, it is only being trained on synthetic data from the model,
²⁷ not on real data. The reweighted wake sleep algorithm of [BB15b] proposed to learn the inference
²⁸ network by using real data from the empirical distribution, in addition to fantasy data. They call
²⁹ the case where you use real data the “**wake-phase q update**”, but we will call it the “**daydream**
³⁰ **phase**”, since, unlike sleeping, the system uses real data x to update the inference model, instead of
³¹ fantasies.¹³ [Le+19] went further, and proposed to only use the wake and daydream phases, and to
³² skip the sleep phase entirely.
³³

³⁴ In more detail, the new objective which we want to minimize becomes

$$\frac{35}{36} \quad \mathcal{L}(\phi|\theta) = \mathbb{E}_{p_D(x)} [D_{\text{KL}}(p_\theta(z|x)\|q_\phi(z|x))] \quad (22.151)$$

³⁷ We can compute a single sample approximation to the negative of the above expression as follows:

$$\frac{38}{39} \quad LL(\phi|\theta, x) = \mathbb{E}_{p(z|x)} [\log q_\phi(z|x)] \quad (22.152)$$

⁴⁰ where $x \sim p_D$. We can approximate this expectation using importance sampling, with q_ϕ as the
⁴¹ proposal. This results in the following estimator of the gradient for each datapoint:
⁴²

$$\frac{43}{44} \quad \nabla_\phi LL(\phi|\theta, x) = \int p_\theta(z|x) \nabla_\phi \log q_\phi(z|x) dz \approx \sum_{s=1}^S \bar{w}_s \nabla_\phi \log q_\phi(z_s|x) \quad (22.153)$$

⁴⁵ ⁴⁶ 13. We thank Rif Sauros for suggesting this term.
⁴⁷

1 where $\mathbf{z}_s \sim q_\phi(\mathbf{z}_s | \mathbf{x})$ and \bar{w}_s are the normalized weights.
2

3 We see that Equation (22.153) is very similar to Equation (22.150). The key difference is that in
4 the daydream phase, we sample from $(\mathbf{x}, \mathbf{z}_s) \sim p_{\mathcal{D}}(\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})$, where \mathbf{x} is a real data point, whereas
5 in the sleep phase, we sample from $(\mathbf{x}'_s, \mathbf{z}'_s) \sim p_\theta(\mathbf{z}, \mathbf{x})$, where \mathbf{x}'_s is generated data point.

7 22.7.4 Summary of algorithm

10 **Algorithm 27:** One SGD update using wake-sleep algorithm.

- 11 1 Sample \mathbf{x}_n from dataset ;
 - 12 2 Draw S samples from inference network: $\mathbf{z}_s \sim q(\mathbf{z}|\mathbf{x}_n)$;
 - 13 3 Compute unnormalized weights: $w_s = \frac{p(\mathbf{x}_n, \mathbf{z}_s)}{q(\mathbf{z}_s | \mathbf{x}_n)}$;
 - 14 4 Compute normalized weights: $\bar{w}_s = \frac{w_s}{\sum_{s'=1}^S w_{s'}}$;
 - 15 5 Optional: Compute estimate of log likelihood: $\log p(\mathbf{x}_n) = \log(\frac{1}{S} \sum_{s=1}^S w_s)$;
 - 16 6 Wake phase: Update θ using $\sum_{s=1}^S \bar{w}_s \nabla_\theta \log p_\theta(\mathbf{z}_s, \mathbf{x}_n)$;
 - 17 7 Daydream phase: Update ϕ using $\sum_{s=1}^S \bar{w}_s \nabla_\phi \log q_\phi(\mathbf{z}_s | \mathbf{x}_n)$;
 - 18 8 Optional sleep phase: Draw S samples from model, $(\mathbf{x}'_s, \mathbf{z}'_s) \sim p_\theta(\mathbf{x}, \mathbf{z})$ and update ϕ using
19 $\frac{1}{S} \sum_{s=1}^S \nabla_\phi \log q_\phi(\mathbf{z}'_s | \mathbf{x}'_s)$
-

22 We summarize the RWS algorithm in Algorithm 27. The disadvantage of the RWS algorithm is
23 that it does not optimize a single well-defined objective, so it is not clear if the method will converge,
24 in contrast to ELBO maximization. On the other hand, the method is fairly simple, since it consists
25 of two alternating weighted maximum likelihood problems. It can also be shown to “sandwich” a
26 lower and upper bound of the log marginal likelihood. We can think of this in terms of the two joint
27 distributions $p_\theta(\mathbf{x}, \mathbf{z})$ and $q_{\mathcal{D}, \phi}(\mathbf{x}, \mathbf{z}) = q_{\mathcal{D}}(\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})$:

30 wake phase $\min_{\theta} D_{\text{KL}}(q_{\mathcal{D}, \phi}(\mathbf{x}, \mathbf{z}) \| p_\theta(\mathbf{x}, \mathbf{z}))$ (22.154)

31 daydream phase $\min_{\phi} D_{\text{KL}}(p_\theta(\mathbf{x}, \mathbf{z}) \| q_{\mathcal{D}, \phi}(\mathbf{x}, \mathbf{z}))$ (22.155)

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

23 Auto-regressive models

23.1 Introduction

By the chain rule of probability, we can write any joint distribution over T variables as follows:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2, \mathbf{x}_1)p(\mathbf{x}_4|\mathbf{x}_3, \mathbf{x}_2, \mathbf{x}_1)\dots = \prod_{t=1}^T p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) \quad (23.1)$$

where $\mathbf{x}_t \in \mathcal{X}$ is the t 'th observation, and we define $p(\mathbf{x}_1|\mathbf{x}_{1:0}) = p(x_1)$ as the initial state distribution. This is called an **auto-regressive model**. This corresponds to a fully connected DAG, in which each node depends on all its predecessors in the ordering, as shown in Figure 23.1. The models can also be conditioned on arbitrary inputs or context \mathbf{c} , in order to define $p(\mathbf{x}|\mathbf{c})$, although we omit this for notational brevity.

We could of course also factorize the joint distribution “backwards” in time, using

$$p(\mathbf{x}_{1:T}) = \prod_{t=T}^1 p(\mathbf{x}_t|\mathbf{x}_{t+1:T}) \quad (23.2)$$

However, this “anti-causal” direction is often harder to learn (see e.g., [PJS17]).

Although the decomposition in Equation (23.1) is general, each term in this expression (i.e., each conditional distribution $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$) becomes more and more complex, since it depends on an increasing number of arguments, which makes the terms slow to compute, and makes estimating their parameters more data hungry (see Section 2.8.3.2).

One approach to solving this intractability is to make the (first-order) **Markov assumption**, which gives rise to a **Markov model** $p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1})$, which we discuss in Section 2.8. (This is also called an auto-regressive model of order 1.) Unfortunately, the Markov assumption is very limiting. One way to relax it, and to make \mathbf{x}_t depend on all the past $\mathbf{x}_{1:t-1}$ without explicitly regressing on them, is to assume the past can be compressed into a **hidden state** \mathbf{z}_t . If \mathbf{z}_t is a deterministic function of the past observations $\mathbf{x}_{1:t-1}$, the resulting model is known as a **recurrent neural network**, discussed in Section 16.3.3. If \mathbf{z}_t is a stochastic function of the past hidden state, \mathbf{z}_{t-1} , the resulting model is known as a **hidden Markov model**, which we discuss in Section 30.1.

Another approach is to stay with the general AR model of Equation (23.1), but to use a restricted functional form, such as some kind of neural network, for the conditionals $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$. Thus rather than making conditional independence assumptions, or explicitly compressing the past into a sufficient

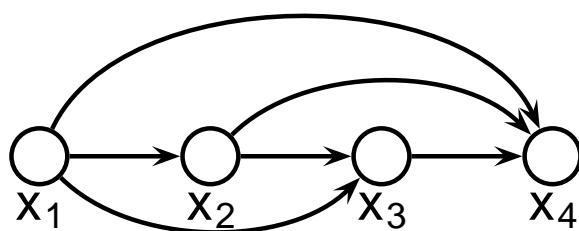


Figure 23.1: A fully-connected auto-regressive model.

statistic, we implicitly learn a compact mapping from the past to the future. In the sections below, we discuss different functional forms for these conditional distributions.

The main advantage of such AR models is that it is easy to compute, and optimize, the exact likelihood of each sequence (data vector). The main disadvantage is that generating samples is inherently sequential, which can be slow. In addition, the method does not learn a compact latent representation of the data.

19

20 23.2 Neural autoregressive density estimators (NADE) 21

22 A simple way to represent each conditional probability distribution $p(x_t | \mathbf{x}_{1:t-1})$ is to use a generalized 23 linear model, such as logistic regression, as proposed in [Fre98]. We can make the model be more 24 powerful by using a neural network. The resulting model is called the **neural auto-regressive** 25 **density estimator** or **NADE** model [LM11].

26 If we let $p(x_t | \mathbf{x}_{1:t-1})$ be a conditional mixture of Gaussians, we get a model known as **RNADE** 27 (“Real-valued Neural Autoregressive Density Estimator”) of [UML13]. More precisely, this has the 28 form

$$30 \quad p(x_t | \mathbf{x}_{1:t-1}) = \sum_{k=1}^K \pi_{t,k} \mathcal{N}(x_t | \mu_{t,k}, \sigma_{t,k}^2) \quad (23.3)$$

31 where the parameters are generated by a network, $(\mu_t, \sigma_t, \pi_t) = f_t(\mathbf{x}_{1:t-1}; \theta_t)$.

32 Rather than using separate neural networks, f_1, \dots, f_T , it is more efficient to create a single 33 network with T inputs and T outputs. This can be done using masking, resulting in a model called 34 the **MADE** (“Masked Autoencoder for Density Estimation”) model [Ger+15].

35 One disadvantage of NADE-type models is that they assume the variables have a natural linear 36 ordering. This makes sense for temporal or sequential data, but not for more general data types, 37 such as images or graphs. An orderless extension to NADE was proposed in [UML14; Uri+16].

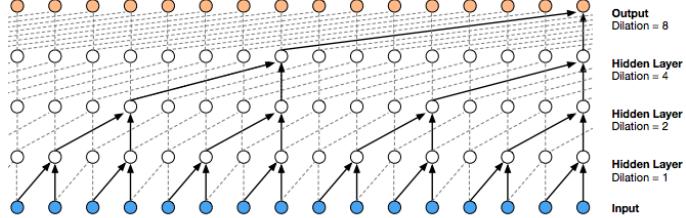
40

41 23.3 Causal CNNs

42

43 One approach to representing the distribution $p(x_t | \mathbf{x}_{1:t-1})$ is to try to identify patterns in the past 44 history that might be predictive of the value of x_t . If we assume these patterns can occur in any 45 location, it makes sense to use a **convolutional neural network** to detect them. However, we need 46 47

1
2
3
4
5
6
7
8
9
10



11 *Figure 23.2: Illustration of the wavenet model using dilated (atrous) convolutions, with dilation factors of 1,
12 2, 4 and 8. From Figure 3 of [oor+16]. Used with kind permission of Aaron van den Oord.*

13
14

15 to make sure we only apply the convolutional mask to past inputs, not future ones. This can be done
16 using **masked convolution**, also called **causal convolution**. We discuss this in more detail below.
17

18 23.3.1 1d causal CNN (Convolutional Markov models)

19 Consider the following **convolutional Markov model** for 1d discrete sequences:

20

$$21 \quad p(\mathbf{x}_{1:T}) = \prod_{t=1}^T p(x_t | \mathbf{x}_{1:t-1}; \theta) = \prod_{t=1}^T \text{Cat}(x_t | \mathcal{S}(\varphi(\sum_{\tau=1}^{t-k} \mathbf{w}^\top \mathbf{x}_{\tau:\tau+k}))) \quad (23.4)$$

22

23 where \mathbf{w} is the convolutional filter of size k , and we have assumed a single nonlinearity φ and
24 categorical output, for notational simplicity. This is like regular 1d convolution except we “mask out”
25 future inputs, so that x_t only depends on the past values. We can of course use deeper models, and
26 we can condition on input features \mathbf{c} .

27 In order to capture long-range dependencies, we can use **dilated convolution** (see [Mur22, Sec
28 14.4.1]). This model has been successfully used to create a state of the art **text to speech** (TTS)
29 synthesis system known as **wavenet** [oor+16]. See Figure 23.2 for an illustration.

30 The wavenet model is a conditional model, $p(\mathbf{x}|\mathbf{c})$, where \mathbf{c} is a set of linguistic features derived
31 from an input sequence of words, and \mathbf{x} is raw audio. The **tacotron** system [Wan+17c] is a fully
32 end-to-end approach, where the input is words rather than linguistic features.

33 Although wavenet produces high quality speech, it is too slow for use in production systems. How-
34 ever, it can be “distilled” into a parallel generative model [Oor+18], as we discuss in Section 24.2.4.3.
35

36 23.3.2 2d causal CNN (PixelCNN)

37 We can extend causal convolutions to 2d, to get an autoregressive model of the form

38

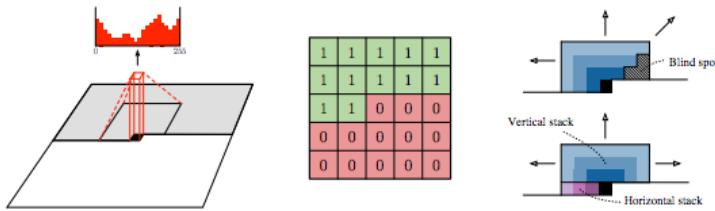
$$39 \quad p(\mathbf{x}|\theta) = \prod_{r=1}^R \prod_{c=1}^C p(x_{r,c} | f_\theta(\mathbf{x}_{1:r-1,1:C}, \mathbf{x}_{r,1:c-1})) \quad (23.5)$$

40

41 where R is the number of rows, C is the number of columns, and we condition on all previously
42 generated pixels in a **raster scan** order, as illustrated in Figure 23.3. This is called the **pixelCNN**
43

44

1
2
3
4
5
6
7
8
9



10 *Figure 23.3: Illustration of causal 2d convolution in the PixelCNN model. The red histogram shows the*
 11 *empirical distribution over discretized values for a single pixel of a single RGB channel. The red and green*
 12 *5 × 5 array shows the binary mask, which selects the top left context, in order to ensure the convolution is*
 13 *causal. The diagrams on the right illustrate how we can avoid blind spots by using a vertical context stack,*
 14 *that contains all previous rows, and a horizontal context stack, that just contains values from the current row.*
 15 *From Figure 1 of [Oor+16]. Used with kind permission of Aaron van den Oord.*

16
17

18 model [Oor+16]. Naive sampling (generation) from this model takes $O(N)$ time, where $N = RC$ is
 19 the number of pixels, but [Ree+17] shows how to use a multiscale approach to reduce the complexity
 20 to $O(\log N)$.

21 Various extensions of this model have been proposed. The **pixelCNN++** model of [Sal+17c]
 22 improved the quality by using a mixture of logistic distributions, to capture the multimodality
 23 of $p(x_i | \mathbf{x}_{1:i-1})$. The **pixelRNN** of [OKK16] combined masked convolution with an RNN to get
 24 even longer range contextual dependencies. The **Subscale Pixel Network** of [MK19] proposed to
 25 generate the pixels such the higher order bits are sampled before lower order bits, which allows high
 26 resolution details to be sampled conditioned on low resolution versions of the whole image, rather
 27 than just the top left corner.
 28

29

30 23.4 Transformer decoders

31

32 We introduced transformers in Section 16.3.4. They can be used for encoding sequences (as in BERT),
 33 or for decoding (generating) sequences. In this section, we focus on the latter case.

34 The basic idea is as follows. At each step t , the model applies masked (causal) self attention
 35 (Section 16.2.7) to the first t inputs, $\mathbf{y}_{1:t}$, to compute a set of attention weights, $\mathbf{a}_{1:t}$. From this it
 36 computes an activation vector $\mathbf{z}_t = \sum_{\tau=1}^t a_{t\tau} \mathbf{y}_\tau$. This is then passed through a feed-forward layer to
 37 compute $\mathbf{h}_t = \text{MLP}(\mathbf{z}_t)$. This process is repeated for each layer in the model. Finally the output
 38 is used to predict the next element in the sequence, $\mathbf{y}_{t+1} \sim \text{Cat}(\mathcal{S}(\mathbf{W}\mathbf{h}_t))$. (In the conditional
 39 generation setting, where we want to compute $p(\mathbf{y}|\mathbf{x})$, we can just treat the first \mathbf{x} tokens as part of
 40 the initial output sequence. There is no need to use an encoder block.)

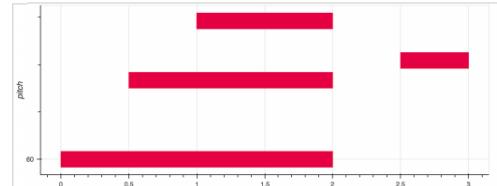
41 At training time, all predictions can happen in parallel, since the target generated sequence is
 42 already available. That is, the t 'th output \mathbf{y}_t can be predicted given inputs $\mathbf{y}_{1:t-1}$, and this can be
 43 done for all t simultaneously. However, at test time, the model must be applied sequentially, so
 44 the output generated at $t + 1$ is fed back into the model to predict $t + 2$, etc. Note that the running
 45 time of transformers is $O(T^2)$, although a variety of more efficient versions have been developed (see
 46 e.g., [Mur22, Sec 15.6] for details).

47

1 PROMPT:
2
3 In a shocking finding, scientist discovered a herd of unicorns living in a remote,
4 previously unexplored valley, in the Andes Mountains. Even more surprising to the
5 researchers was the fact that the unicorns spoke perfect English.
6
7 RESPONSE:
8 The scientist named the population, after their distinctive horn, Ovid's Unicorn.
9 These four-horned, silver-white unicorns were previously unknown to science.
10 Now, after almost two centuries, the mystery of what sparked this odd phenomenon
11 is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz,
12 and several companions, were exploring the Andes Mountains when they found a small valley,
13 with no other animals or humans....
14
15 *Figure 23.4: Sample text generated by GPT-2 in response to an input prompt. From <https://openai.com/blog/better-language-models/>.*
16
17
18 Transformers are the basis of many popular (conditional) generative models for sequences. We
19 give some examples below.
20
21 **23.4.1 Text generation (GPT)**
22
23 In [Rad+18], OpenAI proposed a model called **GPT**, which is short for “Generative Pre-training
24 Transformer”. This is decoder-only transformer model that uses causal (masked) attention. In
25 [Rad+19], they propose **GPT-2**, which is a larger version of GPT (1.5 billion parameters, or
26 6.5GB, for the XL version), trained on a large web corpus (8 million pages, or 40GB). They
27 also simplify the training objective, and just train it using maximum likelihood. The fluency of
28 text generated by GPT-2 is quite remarkable; see Figure 23.4 for an example. See also <https://demo.allennlp.org/next-token-1m>, which lets you interact with the (medium sized) model, and
29 generates the K most likely sequences (computed using beam search) given some input context.
30
31 More recently, OpenAI released **GPT-3** [Bro+20b], which is an even larger version of GPT-2
32 (175 billion parameters), trained on even more data (300 billion words), but based on the same
33 principles. (Training was estimated to take 355 GPU years and cost \$4.6M.) Due to the large size
34 of the data and model, GPT-3 shows even more remarkable abilities to generate novel text. In
35 particular, the output can be (partially) controlled by just changing the conditioning prompt. This
36 enables the model to perform tasks that it has never been trained on, just by giving it some examples
37 in the prompt. This is called “**in-context learning**”, and is an example of (gradient-free) “few-shot
38 learning” (see Section 20.5.2). See Figure 23.5 for an example, and <https://gpt3demo.com/apps/openai-gpt-3-playground> for an interactive demo.
39
40
41 **23.4.2 Music generation**
42
43 It is possible to modify transformer decoders so that they generate music instead of natural language,
44 as shown by the **music transformer** paper [Hua+18a]. The key “trick” is to note that the **midi**
45 **format** for music can be represented as a sequence of parameterized tokens, as shown in Figure 23.6.
46 To cope with the long sequence length, a relative attention mechanism was devised. See Figure 23.7
47

1
2
3 A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses
the word whatpu is:
4 We were traveling in Africa and we saw these very cute whatpus.
5
6 To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses
the word farduddle is:
7 One day when I was playing tag with my little sister, she got really excited and she
started doing these crazy farduddles.
8 A "yalubalu" is a type of vegetable that looks like a big pumpkin. An example of a sentence
that uses the word yalubalu is:
9 I was on a trip to Africa and I tried this yalubalu vegetable that was grown in a garden
there. It was delicious.
10
11 A "Burringo" is a car with very fast acceleration. An example of a sentence that uses the
word Burringo is:
12 In our garage we have a Burringo that my father drives to work every day.
13
14 A "Gigamuru" is a type of Japanese musical instrument. An example of a sentence that uses the
word Gigamuru is:
15 I have a Gigamuru that my uncle gave me as a gift. I love to play it at home.
16
17 To "screeg" something is to swing a sword at it. An example of a sentence that uses the word
screeg is:
18 We screeghed at each other for several minutes and then we went outside and ate ice cream.
19
20

Figure 23.5: Illustration of few shot learning with GPT-3. The model is asked to create an example sentence using a new word whose meaning is provided in the prompt. Boldface is GPT-3's completions, light gray is human input. From Figure 3.16 of [Bro+20b].



21 SET_VELOCITY<80>, NOTE_ON<60>
TIME_SHIFT<500>, NOTE_ON<64>
TIME_SHIFT<500>, NOTE_ON<67>
TIME_SHIFT<1000>, NOTE_OFF<60>, NOTE_OFF<64>,
NOTE_OFF<67>
TIME_SHIFT<500>, SET_VELOCITY<100>, NOTE_ON<65>
TIME_SHIFT<500>, NOTE_OFF<65>

22
23
24
25
26
27
28
29
30
31
32 Figure 23.6: A snippet of a piano performance visualized as a pianoroll (left) and encoded as performance
events (right, serialized from left to right and then down the rows). There are 128 discrete values for note
on/off, 32 values for velocity, and 100 for time shift, so the input is a sequence of one-hot vectors of length
33 388. From Figure 7 of [Hua+18a]. Used with kind permission of Anna Huang.

34 for a visualization. To best appreciate the quality of the generated output, please see the interactive
35 demo at <https://magenta.tensorflow.org/music-transformer>.

36

37 23.4.3 Text-to-image generation (DALL-E)

38 The **DALL-E** model¹ from OpenAI [Ram+21] can generate images of remarkable quality and
39 diversity given text prompts, as shown in Figure 23.8. The methodology is conceptually quite
40

41 1. The name is derived from the artist Salvador Dalí and Pixar's movie "WALL-E"
42

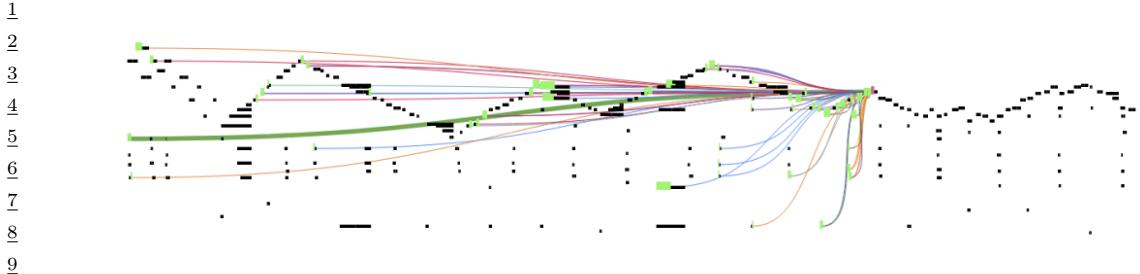


Figure 23.7: Illustration of attention in the music transformer. Different colored lines correspond to the 6 attention heads. Line thickness corresponds to attention weights. From Figure 8 of [Hua+18a]. Used with kind permission of Anna Huang.

13

14

15

16

17

18

19

20

21

22

23

24

25



Figure 23.8: Some images generated by the DALL-E model in response to a text prompt. (a) “An armchair in the shape of an avocado”. (b) “An illustration of a baby hedgehog in a christmas sweater walking a dog”. From <https://openai.com/blog/dall-e/>. Used with kind permission of Aditya Ramesh.

30

31

32 straightforward, and most of the effort went into data collection (they scrape the web for 250 million
 33 image-text pairs) and scaling up the training (they fit a model with 12 billion parameters). Here we
 34 just focus on the algorithmic methods.

35 The basic idea is to transform an image \mathbf{x} into a sequence of discrete tokens \mathbf{z} using a discrete
 36 VAE model (Section 22.6.5), which defines a model of the form $p(\mathbf{x}, \mathbf{z})$. We then fit a transformer to
 37 the concatenation of the image tokens \mathbf{z} and text tokens \mathbf{y} to get a model of the form $p(\mathbf{z}, \mathbf{y})$.

38 To sample an image \mathbf{x} given a text prompt \mathbf{y} , we sample a latent code $\mathbf{z} \sim p(\mathbf{z}|\mathbf{y})$, and then we
 39 feed \mathbf{z} into the VAE decoder to get $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$. Multiple images are generated for each prompt, and
 40 these are then ranked according to a pre-trained critic, which gives them scores depending on how
 41 well the generated image matches the input text: $s_n = \text{critic}(\mathbf{x}_n, \mathbf{y}_n)$. The critic they used was the
 42 contrastive CLIP model (see Section 34.1). This discriminative reranking significantly improves the
 43 results.

44 Some sample results are shown in Figure 23.8, and more can be found online at <https://openai.com/blog/dall-e/>. The image on the right of Figure 23.8 is particularly interesting, since the
 45 prompt — “An illustration of a baby hedgehog in a Christmas sweater walking a dog” — arguably
 46

1 requires that the model solve the “**variable binding problem**”. This refers to the fact that the
2 sentence implies the hedgehog should be wearing the sweater and not the dog. We see that the model
3 sometimes interprets this correctly, but not always: sometimes it draws both animals with Christmas
4 sweaters. In addition, sometimes it draws a hedgehog walking a smaller hedgehog. The quality of the
5 results can also be sensitive to the form of the prompt. Thus although impressive, this technology is
6 clearly not yet reliable.
7

8 Since being released in January 2021, many alternatives to DALL-E have been proposed. For
9 example, Google released **XMC-GAN** [Zha+21], which uses a GAN (Chapter 27) instead of a
10 VQ-transformer for the image generation. And in December 2021, OpenAI released **GLIDE** [Nic+21],
11 which uses a diffusion model (Chapter 26) instead of the VQ-transformer for the image generation.
12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

24 Normalizing Flows

This chapter was written by George Papamakarios and Balaji Lakshminarayanan.

24.1 Introduction

In this chapter we discuss **normalizing flows**, a class of flexible density models that can be easily sampled from and whose exact likelihood function is efficient to compute. Such models can be used for many tasks, such as density modeling, inference and generative modeling. We introduce the key principles of normalizing flows and refer to recent surveys by Papamakarios et al. [Pap+19] and Kobyzev, Prince, and Brubaker [KPB19] for readers interested in learning more. See also <https://github.com/janosh/awesome-normalizing-flows> for a list of papers and software packages.

24.1.1 Preliminaries

Normalizing flows create complex probability distributions $p(\mathbf{x})$ by passing random variables $\mathbf{u} \in \mathbb{R}^D$, drawn from a simple **base distribution** $p(\mathbf{u})$ through a nonlinear but *invertible* transformation $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$. That is, $p(\mathbf{x})$ is defined by the following process:

$$\mathbf{x} = \mathbf{f}(\mathbf{u}) \quad \text{where} \quad \mathbf{u} \sim p(\mathbf{u}). \tag{24.1}$$

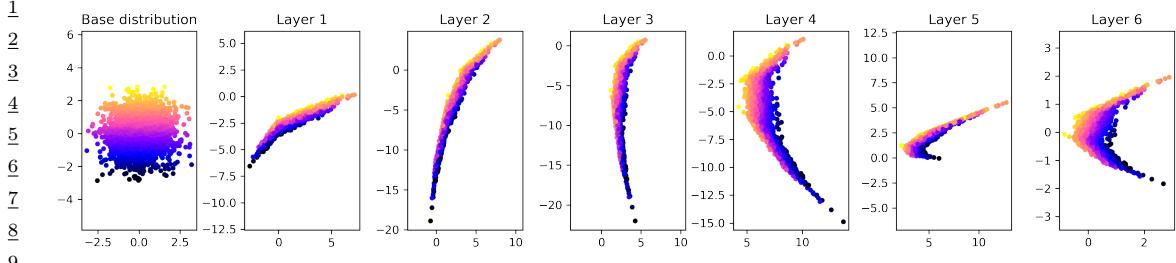
The base distribution is typically chosen to be simple, for example standard Gaussian or uniform, so that we can easily sample from it and compute the density $p(\mathbf{u})$. A flexible enough transformation \mathbf{f} can induce a complex distribution on the transformed variable \mathbf{x} even if the base distribution is simple, as illustrated in Figure 24.1.

Sampling from $p(\mathbf{x})$ is straightforward: we first sample \mathbf{u} from $p(\mathbf{u})$ and then compute $\mathbf{x} = \mathbf{f}(\mathbf{u})$. To compute the density $p(\mathbf{x})$, we rely on the fact that \mathbf{f} is invertible. Let $\mathbf{g}(\mathbf{x}) = \mathbf{f}^{-1}(\mathbf{x}) = \mathbf{u}$ be the inverse mapping, which “**normalizes**” the data distribution by mapping it back to the base distribution (which is often a Normal distribution). Using the change-of-variables formula for random variables from Equation (2.265), we have

$$p_x(\mathbf{x}) = p_u(\mathbf{g}(\mathbf{x})) |\det \mathbf{J}(\mathbf{g})(\mathbf{x})| = p_u(\mathbf{u}) |\det \mathbf{J}(\mathbf{f})(\mathbf{u})|^{-1}, \tag{24.2}$$

where $\mathbf{J}(\mathbf{f})(\mathbf{u}) = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}|_{\mathbf{u}}$ is the Jacobian matrix of \mathbf{f} evaluated at \mathbf{u} . Taking logs of both sides of Equation (24.2), we get

$$\log p_x(\mathbf{x}) = \log p_u(\mathbf{u}) - \log |\det \mathbf{J}(\mathbf{f})(\mathbf{u})|. \tag{24.3}$$



10 *Figure 24.1: Mapping a 2d standard Gaussian to a more complex distribution using an invertible MLP.*
11 Points are color-coded by their initial position along the y-axis. Adapted from [Jan18]. Generated by
12 `flow_2d_mlp.ipynb`.

13

14

15 As discussed above, $p(\mathbf{u})$ is typically easy to evaluate. So, if one can use flexible invertible transfor-
16 mations \mathbf{f} whose Jacobian determinant $\det \mathbf{J}(\mathbf{f})(\mathbf{u})$ can be computed efficiently, then one can construct
17 complex densities $p(\mathbf{x})$ that allow exact sampling and efficient exact likelihood computation. This is
18 in contrast to latent variable models, which require methods like variational inference to lower-bound
19 the likelihood.

20 One might wonder how flexible are the densities $p(\mathbf{x})$ obtained by transforming random variables
21 sampled from simple $p(\mathbf{u})$. It turns out that we can use this method to approximate any smooth
22 distribution. To see this, consider the scenario where the base distribution $p(\mathbf{u})$ is a one-dimensional
23 uniform distribution. Recall that inverse transform sampling (Section 11.3.1) samples random
24 variables from a uniform distribution and transforms them using the inverse Cumulative Distribution
25 Function (CDF) to generate samples from the desired density. We can use this method to sample
26 from any one-dimensional density as long as the transformation \mathbf{f} is powerful enough to model the
27 inverse CDF (which is a reasonable assumption for well-behaved densities whose CDF is invertible
28 and differentiable). We can further extend this argument to multiple dimensions by first expressing
29 the density $p(\mathbf{x})$ as a product of one-dimensional conditionals using the chain rule of probability,
30 and then applying inverse transform sampling to each one-dimensional conditional. The result is a
31 normalizing flow that transforms a product of uniform distributions into any desired distribution
32 $p(\mathbf{x})$. We refer to [Pap+19] for a more detailed proof.

33 How do we define flexible invertible mappings whose Jacobian determinant is easy to compute?
34 We discuss this topic in detail in Section 24.2, but in summary, there are two main ways. The first
35 approach is to define a set of simple transformations that are invertible by design, and whose Jacobian
36 determinant is easy to compute; for instance, if the Jacobian is a triangular matrix, its determinant
37 can be computed efficiently. The second approach is to exploit the fact that a composition of invertible
38 functions is also invertible, and the overall Jacobian determinant is just the product of the individual
39 Jacobian determinants. More precisely, if $\mathbf{f} = \mathbf{f}_N \circ \dots \circ \mathbf{f}_1$ where each \mathbf{f}_i is invertible, then \mathbf{f} is also
40 invertible, with inverse $\mathbf{g} = \mathbf{g}_1 \circ \dots \circ \mathbf{g}_N$ and log Jacobian determinant given by

$$\log |\det \mathbf{J}(\mathbf{g})(\mathbf{x})| = \sum_{i=1}^N \log |\det \mathbf{J}(\mathbf{g}_i)(\mathbf{u}_i)| \quad (24.4)$$

41 where $\mathbf{u}_i = \mathbf{f}_i \circ \dots \circ \mathbf{f}_1(\mathbf{u})$ is the i 'th intermediate output of the flow. This allows us to create
42 complex flows from simple components, just as graphical models allow us to create complex joint
43

1 distributions from simpler conditional distributions.

2 Finally, a note on terminology. An invertible transformation is also known as a **bijection**. A
3 bijection that is differentiable and has a differentiable inverse is known as a **diffeomorphism**. The
4 transformation \mathbf{f} of a flow model is a diffeomorphism, although in the rest of this chapter we will refer
5 to it as a “bijection” for simplicity, leaving the differentiability implicit. The density $p_x(\mathbf{x})$ of a flow
6 model is also known as the **pushforward** of the base distribution $p_u(\mathbf{u})$ through the transformation
7 \mathbf{f} , and is sometimes denoted as $p_x = \mathbf{f}_* p_u$. Finally, in mathematics the term “flow” refers to any
8 family of diffeomorphisms \mathbf{f}_t indexed by a real number t such that $t = 0$ indexes the identity function,
9 and $t_1 + t_2$ indexes $\mathbf{f}_{t_2} \circ \mathbf{f}_{t_1}$ (in physics, t often represents time). In machine learning we use the term
10 “flow” by analogy to the above meaning, to highlight the fact that we can create flexible invertible
11 transformations by composing simpler ones; in this sense, the index t is analogous to the number i of
12 transformations in $\mathbf{f}_i \circ \dots \circ \mathbf{f}_1$.

14 24.1.2 Example

15 In this section, we consider a simple 2d example of a normalizing flow, where $p(\mathbf{u})$ is a 2d standard
16 Gaussian, and \mathbf{f}_θ is an MLP. Each layer of the MLP corresponds to the following nonlinear
17 transformation:

$$\small{20} \quad \mathbf{z}_{l+1} = \varphi(\mathbf{A}_l \mathbf{z}_l + \mathbf{b}_l), \quad (24.5)$$

21 where φ is a nonlinear activation function applied elementwise and \mathbf{A}_l is an invertible square matrix.
22 We cannot use a ReLU activation function, since it is not invertible, so instead we will use a
23 parameterized ReLU function, which is like a leaky ReLU with a learnable slope $\lambda_j > 0$ for each
24 dimension:

$$\small{27} \quad \varphi(\mathbf{x})_j = x_j \mathbb{I}(x_j \geq 0) + \lambda_j x_j \mathbb{I}(x_j < 0). \quad (24.6)$$

28 We now discuss how to compute the log Jacobian determinant of layer l . Let $\mathbf{x} = \mathbf{z}_l$ be its input,
29 $\mathbf{a} = \mathbf{Ax} + \mathbf{b} = \mathbf{f}(\mathbf{x})$ be its linear transformation (we drop the l subscript from the parameters for
30 simplicity), and $\mathbf{y} = \varphi(\mathbf{a})$ be its output. From Equation (24.4), we have

$$\small{32} \quad \log |\det \mathbf{J}(\varphi \circ f)| = \log |\det \mathbf{J}(\varphi)| + \log |\det \mathbf{J}(f)|. \quad (24.7)$$

33 Since φ is applied elementwise, we have

$$\small{36} \quad \mathbf{J}(\varphi) = \text{diag} \left(\frac{\partial y_i}{\partial a_i} \right), \quad (24.8)$$

37 where

$$\small{40} \quad \frac{\partial y_i}{\partial a_i} = \mathbb{I}(a_i \geq 0) + \lambda_i \mathbb{I}(a_i < 0). \quad (24.9)$$

42 For the second term, we have

$$\small{45} \quad \mathbf{J}(f)_{ij} = \frac{\partial a_i}{\partial x_j} = \frac{\partial (\mathbf{A}_{i,:}^\top \mathbf{x} + b_i)}{\partial x_j} = A_{ij}. \quad (24.10)$$

¹ Hence $\mathbf{J}(f) = \mathbf{A}$. In general, computing the determinant of \mathbf{A} takes cubic time in the number of
² dimensions. In Section 24.2, we will discuss transformations which support linear time computation
³ of their Jacobian determinants.
⁴

⁵ We can compose multiple such layers to define a flexible density $p_{\theta}(\mathbf{x})$. This is illustrated in
⁶ Figure 24.1, where we map a 2d Gaussian to a more complex 2d distribution using a 6-layer MLP.
⁷ Note that we needed to use a deep model even for this simple 2d example because each layer is
⁸ limited in its expressive power. We consider more complex invertible transformations in Section 24.2.
⁹

¹⁰ 24.1.3 How to train a flow model

¹¹ There are two common applications of normalizing flows. The first one is density estimation of
¹² observed data, which is achieved by fitting $p_{\theta}(\mathbf{x})$ to the data and using it as an estimate of the
¹³ data density, potentially followed by generating new data from $p_{\theta}(\mathbf{x})$. The second one is variational
¹⁴ inference, which involves sampling from and evaluating a variational posterior $q_{\theta}(\mathbf{z}|\mathbf{x})$ parameterized
¹⁵ by the flow model. As we will see below, these applications optimize different objectives and impose
¹⁶ different computational constraints on the flow model.
¹⁷

¹⁸ 24.1.3.1 Density estimation

¹⁹ Density estimation requires maximizing the likelihood function in Equation (24.2). This requires that
²⁰ we can efficiently evaluate the inverse flow $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x})$ and its Jacobian determinant $\det \mathbf{J}(\mathbf{f}^{-1})(\mathbf{x})$
²¹ for any given \mathbf{x} . After optimizing the model, we can optionally use it to generate new data. To
²² sample new points, we require that the forward mapping \mathbf{f} be tractable.
²³

²⁴ 24.1.3.2 Variational inference

²⁵ Normalizing flows are commonly used for variational inference to parametrize the approximate
²⁶ posterior distribution in latent variable models, as discussed in Section 10.4.3. Consider a latent
²⁷ variable model with continuous latent variables \mathbf{z} and observable variables \mathbf{x} . For simplicity, we
²⁸ consider the model parameters to be fixed as we are interested in approximating the true posterior
²⁹ $p^*(\mathbf{z}|\mathbf{x})$ with a normalizing flow $q_{\theta}(\mathbf{z}|\mathbf{x})$.¹ As discussed in Section 10.1.2, the variational parameters
³⁰ are trained by maximizing the evidence lower bound (ELBO), given by
³¹

$$\frac{32}{33} L(\boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q_{\theta}(\mathbf{z}|\mathbf{x})] \quad (24.11)$$

³⁴ When viewing the ELBO as a function of $\boldsymbol{\theta}$, it can be simplified as follows (note we drop the
³⁵ dependency on \mathbf{x} for simplicity):
³⁶

$$\frac{37}{38} L(\boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z})} [\ell_{\boldsymbol{\theta}}(\mathbf{z})]. \quad (24.12)$$

³⁹ Let $q_{\theta}(\mathbf{z})$ denote a normalizing flow with base distribution $q(\mathbf{u})$ and transformation $\mathbf{z} = f_{\boldsymbol{\theta}}(\mathbf{u})$. Then
⁴⁰ the reparametrization trick (Section 6.6.4) allows us to optimize the parameters using stochastic
⁴¹ gradients. To achieve this, we first write the expectation with respect to the base distribution:
⁴²

$$\frac{43}{44} L(\boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z})} [\ell_{\boldsymbol{\theta}}(\mathbf{z})] = \mathbb{E}_{q(\mathbf{u})} [\ell_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\mathbf{u}))]. \quad (24.13)$$

⁴⁵ 1. We denote the parameters of the variational posterior by $\boldsymbol{\theta}$ here, which should not be confused with the model
⁴⁶ parameters which are also typically denoted by $\boldsymbol{\theta}$ elsewhere.

1 Then, since the base distribution does not depend on θ , we can obtain stochastic gradients as follows:

$$\nabla_{\theta} L(\theta) = \mathbb{E}_{q(\mathbf{u})} [\nabla_{\theta} \ell_{\theta}(f_{\theta}(\mathbf{u}))] \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \ell_{\theta}(f_{\theta}(\mathbf{u}_n)), \quad (24.14)$$

8 where $\{\mathbf{u}_n\}_{n=1}^N$ are samples from $q(\mathbf{u})$.

9 As we can see, in order to optimize this objective, we need to be able to efficiently sample
10 from $q_{\theta}(\mathbf{z}|\mathbf{x})$ and evaluate the probability density of these samples during optimization. (See
11 Section 24.2.4.3 for details on how to do this.) This is contrast to the MLE approach in Section 24.1.3.1,
12 which requires that we be able to compute efficiently the density of arbitrary training datapoints,
13 but it does not require samples during optimization.

15 24.2 Constructing Flows

17 In this section, we discuss how to compute various kinds of flows that are invertible by design and
18 have efficiently computable Jacobian determinants.

20 24.2.1 Affine flows

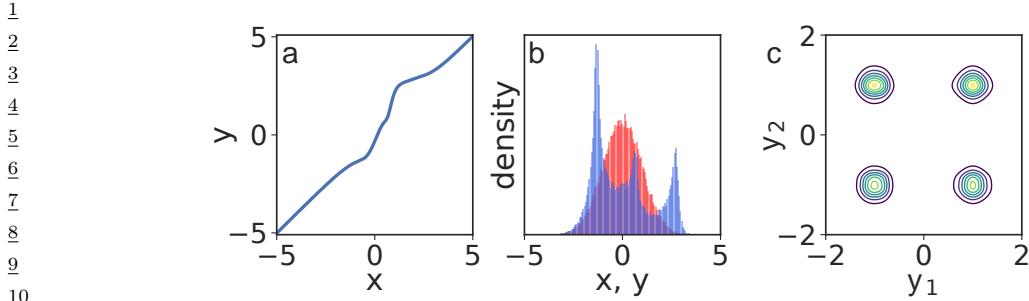
22 A simple choice is to use an affine transformation $\mathbf{x} = \mathbf{f}(\mathbf{u}) = \mathbf{A}\mathbf{u} + \mathbf{b}$. This is a bijection if and
23 only if \mathbf{A} is an invertible square matrix. The Jacobian determinant of \mathbf{f} is $\det \mathbf{A}$, and its inverse is
24 $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x} - \mathbf{b})$. A flow consisting of affine bijections is called an **affine flow**, or a **linear**
25 **flow** if we ignore \mathbf{b} .

26 On their own, affine flows are limited in their expressive power. For example, suppose the base
27 distribution is Gaussian, $p(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\boldsymbol{\mu}, \Sigma)$. Then the pushforward distribution after an affine
28 bijection is still Gaussian, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\Sigma\mathbf{A}^T)$. However, affine bijections are useful building
29 blocks when composed with the non-affine bijections we discuss later, as they encourage “mixing” of
30 dimensions through the flow.

31 For practical reasons, we need to ensure the Jacobian determinant and the inverse of the flow are
32 fast to compute. In general, computing $\det \mathbf{A}$ and \mathbf{A}^{-1} explicitly takes $O(D^3)$ time. To reduce the
33 cost, we can add structure to \mathbf{A} . If \mathbf{A} is diagonal, the cost becomes $O(D)$. If \mathbf{A} is triangular, the
34 Jacobian determinant is the product of the diagonal elements, so takes $O(D)$ time; inverting the
35 flow requires solving the triangular system $\mathbf{A}\mathbf{u} = \mathbf{x} - \mathbf{b}$, which can be done with backsubstitution in
36 $O(D^2)$ time.

37 The result of a triangular transformation depends on the ordering of the dimensions. To reduce
38 sensitivity to this, and to encourage “mixing” of dimensions, we can multiply \mathbf{A} with a permutation
39 matrix, which has an absolute determinant of 1. We often use a permutation that reverses the indices
40 at each layer or that randomly shuffles them. However, usually the permutation at each layer is fixed
41 rather than learned.

42 For spatially structured data (such as images), we can define \mathbf{A} to be a convolution matrix. For
43 example, GLOW [KD18b] uses 1×1 convolution; this is equivalent to pointwise linear transformation
44 across feature dimensions, but regular convolution across spatial dimensions. Two more general meth-
45 ods for modeling $d \times d$ convolutions are presented in [HBW19], one based on stacking autoregressive
46 convolutions, and the other on carrying out the convolution in the Fourier domain.



¹¹ *Figure 24.2: Non-linear squared flow (NLSq). Left: an invertible mapping consisting of 4 NLSq layers.*
¹² *Middle: red is the base distribution (Gaussian), blue is the distribution induced by the mapping on the left.*
¹³ *Right: density of a 5-layer autoregressive flow using NLSq transformations and a Gaussian base density,*
¹⁴ *trained on a mixture of 4 Gaussians. From Figure 5 of [ZR19b]. Used with kind permission of Zachary*
¹⁵ *Ziegler.*

¹⁶

¹⁷

¹⁸ 24.2.2 Elementwise flows

¹⁹

²⁰ Let $h : \mathbb{R} \rightarrow \mathbb{R}$ be a scalar-valued bijection. We can create a vector-valued bijection $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$
²¹ by applying h elementwise, that is, $\mathbf{f}(\mathbf{u}) = (h(u_1), \dots, h(u_D))$. The function \mathbf{f} is invertible, and
²² its Jacobian determinant is given by $\prod_{i=1}^D \frac{dh}{du_i}$. A flow composed of such bijections is known as an
²³ **elementwise flow**.

²⁴ On their own, elementwise flows are limited, since they do not model dependencies between the
²⁵ elements. However, they are useful building blocks for more complex flows, such as coupling flows
²⁶ (Section 24.2.3) and autoregressive flows (Section 24.2.4), as we will see later. In this section, we
²⁷ discuss techniques for constructing scalar-valued bijections $h : \mathbb{R} \rightarrow \mathbb{R}$ for use in elementwise flows.
²⁸

²⁹ 24.2.2.1 Affine scalar bijection

³⁰

³¹ An **affine scalar bijection** has the form $h(u; \boldsymbol{\theta}) = au + b$, where $\boldsymbol{\theta} = (a, b) \in \mathbb{R}^2$. (This is a scalar
³² version of an affine flow.) Its derivative $\frac{dh}{du}$ is equal to a . It is invertible if and only if $a \neq 0$. In
³³ practice, we often parameterize a to be positive, for example by making it the exponential or the
³⁴ softplus of an unconstrained parameter. When $a = 1$, $h(u; \boldsymbol{\theta}) = u + b$ is often called an **additive**
³⁵ **scalar bijection**.
³⁶

³⁷ 24.2.2.2 Higher-order perturbations

³⁸

³⁹ The affine scalar bijection is simple to use, but limited. We can make it more flexible by adding
⁴⁰ higher-order perturbations, under the constraint that invertibility is preserved. For example, Ziegler
⁴¹ and Rush [ZR19b] propose the following, which they term **non-linear squared flow**:

$$\begin{aligned} \text{42} \quad h(u; \boldsymbol{\theta}) &= au + b + \frac{c}{1 + (du + e)^2}, \\ \text{43} \end{aligned} \tag{24.15}$$

⁴⁴

⁴⁵ where $\boldsymbol{\theta} = (a, b, c, d, e) \in \mathbb{R}^5$. When $c = 0$, this reduces to the affine case. When $c \neq 0$, it adds an
⁴⁶ inverse-quadratic perturbation, which can induce multimodality as shown in Figure 24.2. Under the
⁴⁷

1 constraints $a > \frac{9}{8\sqrt{3}}cd$ and $d > 0$ the function becomes invertible, and its inverse can be computed
2 analytically by solving a quadratic polynomial.
3

4 24.2.2.3 Combinations of strictly monotonic scalar functions

5 A strictly monotonic scalar function is one that is always increasing (has positive derivative everywhere)
6 or always decreasing (has negative derivative everywhere). Such functions are invertible. Many
7 activation functions, such as the logistic sigmoid $\sigma(u) = 1/(1 + \exp(-u))$, are strictly monotonic.
8

9 Using such activation functions as a starting point, we can build more flexible monotonic functions
10 via **conical combination** (linear combination with positive coefficients) and function composition.
11 Suppose h_1, \dots, h_K are strictly increasing; then the following are also strictly increasing:
12

- 13 • $a_1 h_1 + \dots + a_K h_K + b$ with $a_k > 0$ (conical combination with a bias),
14
- 15 • $h_1 \circ \dots \circ h_K$ (function composition).

16 By repeating the above two constructions, we can build arbitrarily complex increasing functions. For
17 example, a composition of conical combinations of logistic sigmoids is just an MLP where all weights
18 are positive [Hua+18b].

19 The derivative of such a scalar bijection can be computed by repeatedly applying the chain rule,
20 and in practice can be done with automatic differentiation. However, the inverse is not typically
21 computable in closed form. In practice we can compute the inverse using bisection search, since the
22 function is monotonic.

23 24.2.2.4 Scalar bijections from integration

24 A simple way to ensure a scalar function is strictly monotonic is to constrain its derivative to be
25 positive. Let $h' = \frac{dh}{du}$ be this derivative. Wehenkel and Louppe [WL19] directly parameterize h' with
26 a neural network whose output is made positive via an ELU activation function shifted up by 1.
27 They then integrate the derivative numerically to get the bijection:
28

$$\small{30} \quad h(u) = \int_0^u h'(t)dt + b, \quad (24.16)$$

31 where b is a bias. They call this approach **unconstrained monotonic neural networks**.
32

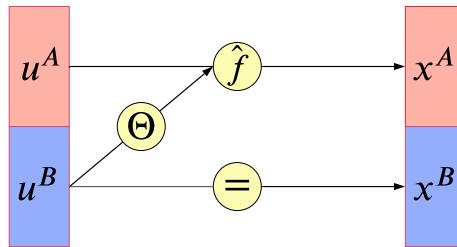
33 The above integral is generally not computable in closed form. It can be, however, if h' is
34 constrained appropriately. For example, Jaini, Selby, and Yu [JSY19] take h' to be a sum of K
35 squared polynomials of degree L :
36

$$\small{37} \quad h'(u) = \sum_{k=1}^K \left(\sum_{\ell=0}^L a_{k\ell} u^\ell \right)^2. \quad (24.17)$$

38 This makes h' a non-negative polynomial of degree $2L$. The integral is analytically tractable, and
39 makes h an increasing polynomial of degree $2L + 1$. For $L = 0$, h' is constant, so h reduces to an
40 affine scalar bijection.

41 In these approaches, the derivative of the bijection can just be read off. However, the inverse is not
42 analytically computable in general. In practice, we can use bisection search to compute the inverse
43 numerically.

1
2
3
4
5
6
7
8
9
10



11 *Figure 24.3: Illustration of a coupling layer $\mathbf{x} = f(\mathbf{u})$. A bijection, with parameters determined by \mathbf{u}^B , is*
 12 *applied to \mathbf{u}^A to generate \mathbf{x}^A ; meanwhile $\mathbf{x}^B = \mathbf{u}^B$ is passed through unchanged, so the mapping can be*
 13 *inverted. From Figure 3 of [KPB19]. Used with kind permission of Ivan Kobyzev.*

14
15
16
17

18 24.2.2.5 Splines

19

20 Another way to construct monotonic scalar functions is using **splines**. These are piecewise-polynomial
 21 or piecewise-rational functions, parameterized in terms of $K + 1$ **knots** (u_k, x_k) through which the
 22 spline passes. That is, we set $h(u_k) = x_k$, and define h on the interval (u_{k-1}, u_k) by interpolating
 23 from x_{k-1} to x_k with a polynomial or rational function (ratio of two polynomials). By increasing the
 24 number of knots we can create arbitrarily flexible monotonic functions.

25 Different ways to interpolate between knots give different types of spline. A simple choice is
 26 to interpolate linearly [Mül+19a], however this makes the derivative discontinuous at the knots.
 27 Interpolating with quadratic polynomials [Mül+19a] gives enough flexibility to make the derivative
 28 continuous. Interpolating with cubic polynomials [Dur+19], ratios of linear polynomials [DEL20],
 29 or ratios of quadratic polynomials [DBP19] allows the derivatives at the knots to be arbitrary
 30 parameters.

31 The spline is strictly increasing if we take $u_{k-1} < u_k$, $x_{k-1} < x_k$, and make sure the interpolation
 32 between knots is itself increasing. Depending on the flexibility on the interpolating function, more
 33 than one interpolations may exist; in practice we chose one that is guaranteed to be always increasing
 34 (see references above for details).

35 An advantage of splines is that they can be inverted analytically if the interpolating functions
 36 only contain low-degree polynomials. In this case, we compute $u = h^{-1}(x)$ as follows: first, we use
 37 binary search to locate the interval (x_{k-1}, x_k) in which x lies; then, we analytically solve the resulting
 38 low-degree polynomial for u .

39
40

41 24.2.3 Coupling flows

42

43 In this section we describe coupling flows, which allow us to model dependencies between dimensions
 44 using arbitrary non-linear functions (such as deep neural networks). Consider a partition of the input
 45 $\mathbf{u} \in \mathbb{R}^D$ into two subspaces, $(\mathbf{u}^A, \mathbf{u}^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$, where d is an integer between 1 and $D - 1$.
 46 Assume a bijection $\hat{f}(\cdot; \theta) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ parameterized by θ and acting on the subspace \mathbb{R}^d . We define
 47

the function $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ given by $\mathbf{x} = \mathbf{f}(\mathbf{u})$ as follows:

$$\mathbf{x}^A = \hat{\mathbf{f}}(\mathbf{u}^A; \Theta(\mathbf{u}^B)) \quad (24.18)$$

$$\mathbf{x}^B = \mathbf{u}^B. \quad (24.19)$$

See Figure 24.3 for an illustration. The function \mathbf{f} is called a **coupling layer** [DKB15; DSDB17], because it “couples” \mathbf{u}^A and \mathbf{u}^B together through $\hat{\mathbf{f}}$ and Θ . We refer to flows consisting of coupling layers as **coupling flows**.

The parameters of $\hat{\mathbf{f}}$ are computed by $\boldsymbol{\theta} = \Theta(\mathbf{u}^B)$, where Θ is an *arbitrary* function called the **conditioner**. Unlike affine flows, which mix dimensions linearly, and elementwise flows, which do not mix dimensions at all, coupling flows can mix dimensions with a flexible non-linear conditioner Θ . In practice we often implement Θ as a deep neural network; any architecture can be used, including MLPs, CNNs, ResNets, etc.

The coupling layer \mathbf{f} is *invertible*, and its inverse is given by $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x})$, where

$$\mathbf{u}^A = \hat{\mathbf{f}}^{-1}(\mathbf{x}^A; \Theta(\mathbf{x}^B)) \quad (24.20)$$

$$\mathbf{u}^B = \mathbf{x}^B. \quad (24.21)$$

That is, \mathbf{f}^{-1} is given by simply replacing $\hat{\mathbf{f}}$ with $\hat{\mathbf{f}}^{-1}$. Because \mathbf{x}^B does not depend on \mathbf{u}^A , the Jacobian of \mathbf{f} is block triangular:

$$\mathbf{J}(\mathbf{f}) = \begin{pmatrix} \partial \mathbf{x}^A / \partial \mathbf{u}^A & \partial \mathbf{x}^A / \partial \mathbf{u}^B \\ \partial \mathbf{x}^B / \partial \mathbf{u}^A & \partial \mathbf{x}^B / \partial \mathbf{u}^B \end{pmatrix} = \begin{pmatrix} \mathbf{J}(\hat{\mathbf{f}}) & \partial \mathbf{x}^A / \partial \mathbf{u}^B \\ \mathbf{0} & \mathbf{I} \end{pmatrix}. \quad (24.22)$$

Thus, $\det \mathbf{J}(\mathbf{f})$ is equal to $\det \mathbf{J}(\hat{\mathbf{f}})$.

We often define $\hat{\mathbf{f}}$ to be an elementwise bijection, so that $\hat{\mathbf{f}}^{-1}$ and $\det \mathbf{J}(\hat{\mathbf{f}})$ are easy to compute. That is, we define:

$$\hat{\mathbf{f}}(\mathbf{u}^A; \boldsymbol{\theta}) = (h(u_1^A; \boldsymbol{\theta}_1), \dots, h(u_d^A; \boldsymbol{\theta}_d)), \quad (24.23)$$

where $h(\cdot; \boldsymbol{\theta}_i)$ is a scalar bijection parameterized by $\boldsymbol{\theta}_i$. Any of the scalar bijections described in Section 24.2.2 can be used here. For example, $h(\cdot; \boldsymbol{\theta}_i)$ can be an affine bijection with $\boldsymbol{\theta}_i$ its scale and shift parameters (Section 24.2.2.1); or it can be a monotonic MLP with $\boldsymbol{\theta}_i$ its weights and biases (Section 24.2.2.3); or it can be a monotonic spline with $\boldsymbol{\theta}_i$ its knot coordinates (Section 24.2.2.5).

There are many ways to define the partition of \mathbf{u} into $(\mathbf{u}^A, \mathbf{u}^B)$. A simple way is just to partition \mathbf{u} into two halves. We can also exploit spatial structure in the partitioning. For example, if \mathbf{u} is an image, we can partition its pixels using a “checkerboard” pattern, where pixels in “black squares” are in \mathbf{u}^A and pixels in “white squares” are in \mathbf{u}^B [DSDB17]. Since only part of the input is transformed by each coupling layer, in practice we typically employ different partitions along a coupling flow, to ensure all variables get transformed and are given the opportunity to interact.

Finally, if $\hat{\mathbf{f}}$ is an elementwise bijection, we can implement arbitrary partitions easily using a binary mask \mathbf{b} as follows:

$$\mathbf{x} = \mathbf{b} \odot \mathbf{u} + (1 - \mathbf{b}) \odot \hat{\mathbf{f}}(\mathbf{u}; \Theta(\mathbf{b} \odot \mathbf{u})), \quad (24.24)$$

where \odot denotes elementwise multiplication. A value of 0 in \mathbf{b} indicates that the corresponding element in \mathbf{u} is transformed (belongs to \mathbf{u}^A); a value of 1 indicates that it remains unchanged (belongs to \mathbf{u}^B).

¹ ² 24.2.4 Autoregressive flows

³ In this section we discuss **autoregressive flows**, which are flows composed of autoregressive bijections.
⁴ Like coupling flows, autoregressive flows allow us to model dependencies between variables with
⁵ arbitrary non-linear functions, such as deep neural networks.

⁶ Suppose the input \mathbf{u} contains D scalar elements, that is, $\mathbf{u} = (u_1, \dots, u_D) \in \mathbb{R}^D$. We define an
⁷ **autoregressive bijection** $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$, its output denoted by $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$, as follows:
⁸

$$\underline{10} \quad x_i = h(u_i; \Theta_i(\mathbf{x}_{1:i-1})), \quad i = 1, \dots, D. \quad (24.25)$$

¹¹ Each output x_i depends on the corresponding input u_i and all previous outputs $\mathbf{x}_{1:i-1} = (x_1, \dots, x_{i-1})$.
¹² The function $h(\cdot; \boldsymbol{\theta}) : \mathbb{R} \rightarrow \mathbb{R}$ is a scalar bijection (for example, one of those described in Section 24.2.2),
¹³ and is parameterized by $\boldsymbol{\theta}$. The function Θ_i is a conditioner that outputs the parameters $\boldsymbol{\theta}_i$ that
¹⁴ yield x_i , given all previous outputs $\mathbf{x}_{1:i-1}$. Like in coupling flows, Θ_i can be an arbitrary non-linear
¹⁵ function, and is often parameterized as a deep neural network.
¹⁶

¹⁷ Because h is invertible, \mathbf{f} is also invertible, and its inverse is given by:

$$\underline{19} \quad u_i = h^{-1}(x_i; \Theta_i(\mathbf{x}_{1:i-1})), \quad i = 1, \dots, D. \quad (24.26)$$

²⁰ An important property of \mathbf{f} is that each output x_i depends on $\mathbf{u}_{1:i} = (u_1, \dots, u_i)$, but not on
²¹ $\mathbf{u}_{i+1:D} = (u_{i+1}, \dots, u_D)$; as a result, the partial derivative $\partial x_i / \partial u_j$ is identically zero whenever $j > i$.
²² Therefore, the Jacobian matrix $\mathbf{J}(\mathbf{f})$ is triangular, and its determinant is simply the product of its
²³ diagonal entries:
²⁴

$$\underline{26} \quad \det \mathbf{J}(\mathbf{f}) = \prod_{i=1}^D \frac{\partial x_i}{\partial u_i} = \prod_{i=1}^D \frac{dh}{du_i}. \quad (24.27)$$

²⁹ In other words, the autoregressive structure of \mathbf{f} leads to a Jacobian determinant that can be
³⁰ computed efficiently in $O(D)$ time.

³¹ Although invertible, autoregressive bijections are computationally asymmetric: evaluating \mathbf{f} is
³² inherently sequential, whereas evaluating \mathbf{f}^{-1} is inherently parallel. That is because we need $\mathbf{x}_{1:i-1}$ to
³³ compute x_i ; therefore, computing the components of \mathbf{x} must be done sequentially, by first computing
³⁴ x_1 , then using it to compute x_2 , then using x_1 and x_2 to compute x_3 , and so on. On the other hand,
³⁵ computing the inverse can be done in parallel for each u_i , since \mathbf{u} does not appear on the right-hand
³⁶ side of Equation (24.26). Hence, in practice it is often faster to compute \mathbf{f}^{-1} than to compute \mathbf{f} ,
³⁷ assuming h and h^{-1} have similar computational cost.
³⁸

³⁹ 24.2.4.1 Affine autoregressive flows

⁴⁰ For a concrete example, we can take h to be an affine scalar bijection (Section 24.2.2.1) parameterized
⁴¹ by a log scale α and a bias μ . Such autoregressive flows are known as **affine autoregressive flows**.
⁴² The parameters of the i 'th component, α_i and μ_i , are functions of $\mathbf{x}_{1:i-1}$, so \mathbf{f} takes the following
⁴³ form:
⁴⁴

$$\underline{46} \quad x_i = u_i \exp(\alpha_i(\mathbf{x}_{1:i-1})) + \mu_i(\mathbf{x}_{1:i-1}). \quad (24.28)$$

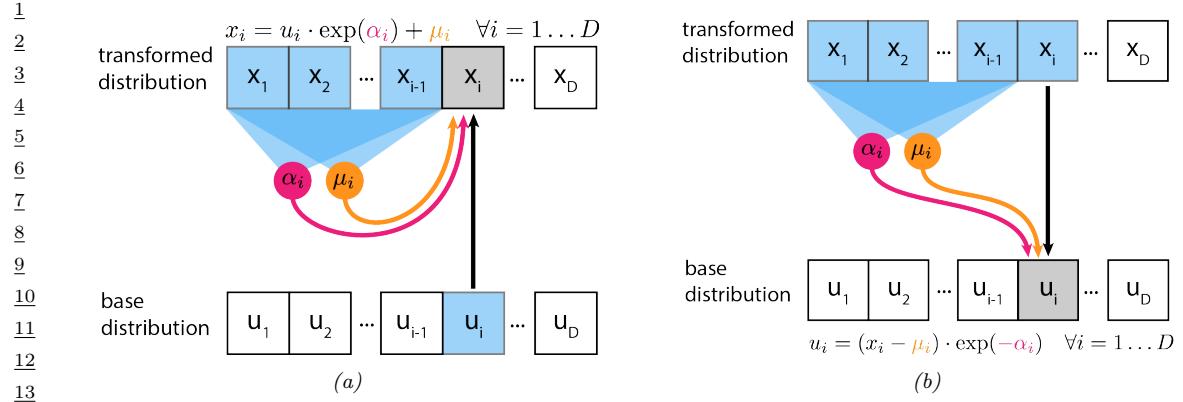


Figure 24.4: (a) Affine autoregressive flow with one layer. In this figure, \mathbf{u} is the input to the flow (sample from the base distribution) and \mathbf{x} is its output (sample from the transformed distribution). (b) Inverse of the above. From [Jan18]. Used with kind permission of Eric Jang.

This is illustrated in Figure 24.4(a). We can invert this by

$$u_i = (x_i - \mu_i(\mathbf{x}_{1:i-1})) \exp(-\alpha_i(\mathbf{x}_{1:i-1})). \quad (24.29)$$

This is illustrated in Figure 24.4(b). Finally, we can calculate the log absolute Jacobian determinant by

$$\log |\det \mathbf{J}(\mathbf{f})| = \log \left| \prod_{i=1}^D \exp(\alpha_i(\mathbf{x}_{1:i-1})) \right| = \sum_{i=1}^D \alpha_i(\mathbf{x}_{1:i-1}). \quad (24.30)$$

Let us look at an example of an affine autoregressive flow on a 2d density estimation problem. Consider an affine autoregressive flow $\mathbf{x} = (x_1, x_2) = \mathbf{f}(\mathbf{u})$, where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and \mathbf{f} is a single autoregressive bijection. Since x_1 is an affine transformation of $u_1 \sim \mathcal{N}(0, 1)$, it is Gaussian with mean μ_1 and standard deviation $\sigma_1 = \exp \alpha_1$. Similarly, if we consider x_1 fixed, x_2 is an affine transformation of $u_2 \sim \mathcal{N}(0, 1)$, so it is *conditionally* Gaussian with mean $\mu_2(x_1)$ and standard deviation $\sigma_2(x_1) = \exp \alpha_2(x_1)$. Thus, a single affine autoregressive bijection will always produce a distribution with Gaussian conditionals, that is, a distribution of the following form:

$$p(x_1, x_2) = p(x_1) p(x_2|x_1) = \mathcal{N}(x_1|\mu_1, \sigma_1^2) \mathcal{N}(x_2|\mu_2(x_1), \sigma_2(x_1)^2) \quad (24.31)$$

This result generalizes to an arbitrary number of dimensions D .

A single affine bijection is not very powerful, regardless of how flexible the functions $\alpha_2(x_1)$ and $\mu_2(x_1)$ are. For example, suppose we want to fit the cross-shaped density shown in Figure 24.5(a) with such a flow. The resulting maximum-likelihood fit is shown in Figure 24.5(b). The red contours show the predictive distribution, $\hat{p}(\mathbf{x})$, which clearly fails to capture the true distribution. The green dots show transformed versions of the data samples, $p(\mathbf{u})$; we see that this is far from the Gaussian base distribution.

Fortunately, we can obtain a better fit by composing multiple autoregressive bijections (layers), and reversing the order of the variables after each layer. For example, Figure 24.5(c) shows the

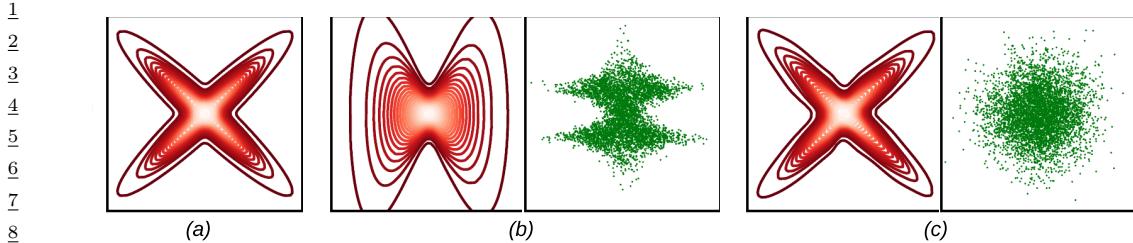


Figure 24.5: Density estimation with affine autoregressive flows, using a Gaussian base distribution. (a) True density. (b) Estimated density using a single autoregressive layer with ordering (x_1, x_2) . On the left (contour plot) we show $p(\mathbf{x})$. On the right (green dots) we show samples of $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x})$, where \mathbf{x} is sampled from the true density. (c) Same as (b), but using 5 autoregressive layers and reversing the variable ordering after each layer. Adapted from Figure 1 of [PPM17]. Used with kind permission of Iain Murray.

results of an affine autoregressive flow with 5 layers applied to the same problem. The red contours show that we have matched the empirical distribution, and the green dots show we have matched the Gaussian base distribution.

Note that another way to obtain a better fit is to replace the affine bijection h with a more flexible one, such as a monotonic MLP (Section 24.2.2.3) or a monotonic spline (Section 24.2.2.5).

24.2.4.2 Masked autoregressive flows

As we have seen, the conditioners Θ_i can be arbitrary non-linear functions. The most straightforward way to parameterize them is separately for each i , for example by using D separate neural networks. However, this can be parameter-inefficient for large D .

In practice, we often share parameters between conditioners by combining them into a single model Θ that takes in \mathbf{x} and outputs $(\theta_1, \dots, \theta_D)$. For the bijection to remain autoregressive, we must constrain Θ so that θ_i depends only on $\mathbf{x}_{1:i-1}$ and not on $\mathbf{x}_{i:D}$. One way to achieve this is to start with an arbitrary neural network (an MLP, a CNN, a ResNet, etc.), and drop connections (for example, by zeroing out weights) until θ_i is only a function of $\mathbf{x}_{1:i-1}$.

An example of this approach is the **masked autoregressive flow (MAF)** model of [PPM17]. This model is an affine autoregressive flow combined with permutation layers, as we described in Section 24.2.4.1. MAF implements the combined conditioner Θ as follows: it starts with an MLP, and then multiplies (elementwise) the weight matrix of each layer with a binary mask of the same size (different masks are used for different layers). The masks are constructed using the method of [Ger+15]. This ensures that all computational paths from x_j to θ_i are zeroed out whenever $j \geq i$, effectively making θ_i only a function of $\mathbf{x}_{1:i-1}$. Still, evaluating the masked conditioner Θ has the same computational cost as evaluating the original (unmasked) MLP.

The key advantage of MAF (and of related models) is that, given \mathbf{x} , all parameters $(\theta_1, \dots, \theta_D)$ can be computed efficiently with one neural-network evaluation, so the computation of the inverse \mathbf{f}^{-1} is fast. Thus, we can efficiently evaluate the probability density of the flow model for arbitrary datapoints. However, in order to compute \mathbf{f} , the conditioner Θ must be called a total of D times, since not all entries of \mathbf{x} are available to start with. Thus, generating new samples from the flow is D times more expensive than evaluating its probability density function. This makes MAF suitable

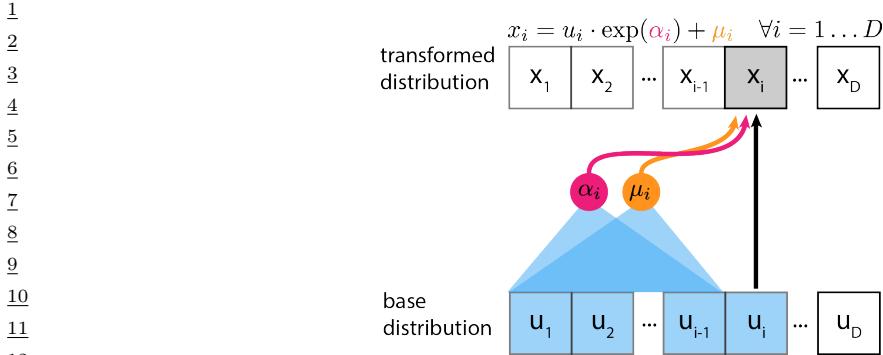


Figure 24.6: Inverse autoregressive flow that uses affine scalar bijections. In this figure, \mathbf{u} is the input to the flow (sample from the base distribution) and \mathbf{x} is its output (sample from the transformed distribution). From [Jan18]. Used with kind permission of Eric Jang.

for density estimation, but less so for data generation.

24.2.4.3 Inverse autoregressive flows

As we have seen, the parameters θ_i that yield the i 'th output x_i are functions of the previous outputs $\mathbf{x}_{1:i-1}$. This ensures that the Jacobian $\mathbf{J}(\mathbf{f})$ is triangular, and so its determinant is efficient to compute.

However, there is another possibility: we can make θ_i a function of the previous *inputs* instead, that is, a function of $\mathbf{u}_{1:i-1}$. This leads to the following bijection, which is known as **inverse autoregressive**:

$$x_i = h(u_i; \Theta_i(\mathbf{u}_{1:i-1})), \quad i = 1, \dots, D. \quad (24.32)$$

Like its autoregressive counterpart, this bijection has a triangular Jacobian whose determinant is also given by $\det \mathbf{J}(\mathbf{f}) = \prod_{i=1}^D \frac{dh}{du_i}$. Figure 24.6 illustrates an inverse autoregressive flow, for the case where h is affine.

To see why this bijection is called “inverse autoregressive”, compare Equation (24.32) with Equation (24.26). The two formulas differ only notationally: we can get from one to the other by swapping \mathbf{u} with \mathbf{x} and h with h^{-1} . In other words, the inverse autoregressive bijection corresponds to a direct parameterization of the inverse of an autoregressive bijection.

Since inverse autoregressive bijections swap the forward and inverse directions of their autoregressive counterparts, they also swap their computational properties. This means that the forward direction \mathbf{f} of an inverse autoregressive flow is inherently parallel and therefore fast, whereas its inverse direction \mathbf{f}^{-1} is inherently sequential and therefore slow.

An example of an inverse autoregressive flow is their namesake **IAF** model of [Kin+16]. IAF uses affine scalar bijections, masked conditioners and permutation layers, so it is precisely the inverse of the MAF model described in Section 24.2.4.2. Using IAF, we can generate \mathbf{u} in parallel from the base distribution (using, for example, a diagonal Gaussian), and then sample each element of \mathbf{x} in parallel. However, evaluating $p(\mathbf{x})$ for an arbitrary datapoint \mathbf{x} is slow, because we have to evaluate each element of \mathbf{u} sequentially. Fortunately, evaluating the likelihood of samples generated from IAF

¹ (as opposed to externally provided samples) incurs no additional cost, since in this case the u_i terms
² will already have been computed.
³

⁴ Although not so suitable for density estimation or maximum-likelihood training, IAFs are well-
⁵ suited for parameterizing variational posteriors in variational inference. This is because in order to
⁶ estimate the variational lower bound (ELBO), we only need samples from the variational posterior
⁷ and their associated probability densities, both of which are efficient to obtain. See Section 24.1.3.2
⁸ for details.

⁹ Another useful application of IAFs is training them to mimic models whose probability density is
¹⁰ fast to evaluate but which are slow to sample from. A notable example is the **parallel wavenet**
¹¹ model of [Oor+18]. This model is an IAF p_s that it trained to mimic a pretrained wavenet model p_t
¹² by minimizing the KL divergence $D_{\text{KL}}(p_s \| p_t)$. This KL can be easily estimated by first sampling
¹³ from p_s and then evaluating $\log p_s$ and $\log p_t$ at those samples, operations which are all efficient for
¹⁴ these models. After training, we obtain an IAF that can generate audio of similar quality as the
¹⁵ original wavenet, but can do so much faster.

¹⁶

¹⁷ 24.2.4.4 Connection with autoregressive models

¹⁸ Autoregressive flows can be thought of as generalizing autoregressive models of continuous random
¹⁹ variables, discussed in Section 23.1. Specifically, any continuous autoregressive model can be
²⁰ reparameterized as a one-layer autoregressive flow, as we describe below.

²¹ Consider a general autoregressive model over a continuous random variable $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$
²² written as

$$\begin{aligned} \mathbf{x} &= \prod_{i=1}^D p_i(x_i | \boldsymbol{\theta}_i) \quad \text{where } \boldsymbol{\theta}_i = \Theta_i(\mathbf{x}_{1:i-1}). \end{aligned} \tag{24.33}$$

²⁷ In the above expression, $p_i(x_i | \boldsymbol{\theta}_i)$ is the i -th conditional distribution of the autoregressive model,
²⁸ whose parameters $\boldsymbol{\theta}_i$ are arbitrary functions of the previous variables $\mathbf{x}_{1:i-1}$. For example, $p_i(x_i | \boldsymbol{\theta}_i)$
²⁹ can be a mixture of one-dimensional Gaussian distributions, with $\boldsymbol{\theta}_i$ representing the collection of its
³⁰ means, variances and mixing coefficients.

³¹ Now consider sampling a vector \mathbf{x} from the autoregressive model, which can be done by sampling
³² one element at a time as follows:

$$x_i \sim p_i(x_i | \Theta_i(\mathbf{x}_{1:i-1})) \quad \text{for } i = 1, \dots, D. \tag{24.34}$$

³⁵ Each conditional can be sampled from using inverse transform sampling (Section 11.3.1). Let $U(0, 1)$
³⁶ be the uniform distribution on the interval $[0, 1]$, and let $\text{CDF}_i(x_i | \boldsymbol{\theta}_i)$ be the cumulative distribution
³⁷ function of the i -th conditional. Sampling can be written as:

$$x_i = \text{CDF}_i^{-1}(u_i | \boldsymbol{\theta}_i(\mathbf{x}_{1:i-1})) \quad \text{where } u_i \sim U(0, 1). \tag{24.35}$$

⁴¹ Comparing the above expression with the definition of an autoregressive bijection in Equation (24.25),
⁴² we see that the autoregressive model has been expressed as a one-layer autoregressive flow whose base
⁴³ distribution is uniform on $[0, 1]^D$ and whose scalar bijections correspond to the inverse conditional
⁴⁴ CDFs. Viewing autoregressive models as flows this way has an important advantage, namely that it
⁴⁵ allows us to increase the flexibility of an autoregressive model by composing multiple instances of it
⁴⁶ in a flow, without sacrificing the overall tractability.

⁴⁷

24.2.5 Residual flows

A residual network is a composition of **residual connections**, which are functions of the form $\mathbf{f}(\mathbf{u}) = \mathbf{u} + \mathbf{F}(\mathbf{u})$. The function $\mathbf{F} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ is called the **residual block**, and it computes the difference between the output and the input, $\mathbf{f}(\mathbf{u}) - \mathbf{u}$.

Under certain conditions on \mathbf{F} , the residual connection \mathbf{f} becomes invertible. We will refer to flows composed of invertible residual connections as **residual flows**. In the following, we describe two ways the residual block \mathbf{F} can be constrained so that the residual connection \mathbf{f} is invertible.

24.2.5.1 Contractive residual blocks

One way to ensure the residual connection is invertible is to choose the residual block to be a contraction. A contraction is a function \mathbf{F} whose Lipschitz constant is less than 1; that is, there exists $0 \leq L < 1$ such that for all \mathbf{u}_1 and \mathbf{u}_2 we have:

$$\|\mathbf{F}(\mathbf{u}_1) - \mathbf{F}(\mathbf{u}_2)\| \leq L\|\mathbf{u}_1 - \mathbf{u}_2\|. \quad (24.36)$$

The invertibility of $\mathbf{f}(\mathbf{u}) = \mathbf{u} + \mathbf{F}(\mathbf{u})$ can be shown as follows. Consider the mapping $\mathbf{g}(\mathbf{u}) = \mathbf{x} - \mathbf{F}(\mathbf{u})$. Because \mathbf{F} is a contraction, \mathbf{g} is also a contraction. So, by Banach's fixed-point theorem, \mathbf{g} has a unique fixed point \mathbf{u}_* . Hence we have

$$\mathbf{u}_* = \mathbf{x} - \mathbf{F}(\mathbf{u}_*) \quad (24.37)$$

$$\Rightarrow \mathbf{u}_* + \mathbf{F}(\mathbf{u}_*) = \mathbf{x} \quad (24.38)$$

$$\Rightarrow \mathbf{f}(\mathbf{u}_*) = \mathbf{x}. \quad (24.39)$$

Because \mathbf{u}_* is unique, it follows that $\mathbf{u}_* = \mathbf{f}^{-1}(\mathbf{x})$.

An example of a residual flow with contractive residual blocks is the **iResNet** model of [Beh+19]. The residual blocks of iResNet are convolutional neural networks, that is, compositions of convolutional layers with non-linear activation functions. Because the Lipschitz constant of a composition is less or equal to the product of the Lipschitz constants of the individual functions, it is enough to ensure the convolutions are contractive, and to use increasing activation functions with slope less or equal to 1. The iResNet model ensures the convolutions are contractive by applying spectral normalization to their weights [Miy+18a].

In general, there is no analytical expression for the inverse \mathbf{f}^{-1} . However, we can approximate $\mathbf{f}^{-1}(\mathbf{x})$ using the following iterative procedure:

$$\mathbf{u}_n = \mathbf{g}(\mathbf{u}_{n-1}) = \mathbf{x} - \mathbf{F}(\mathbf{u}_{n-1}). \quad (24.40)$$

Banach's fixed-point theorem guarantees that the sequence $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \dots$ will converge to $\mathbf{u}_* = \mathbf{f}^{-1}(\mathbf{x})$ for any choice of \mathbf{u}_0 , and it will do so at a rate of $O(L^n)$, where L is the Lipschitz constant of \mathbf{g} (which is the same as the Lipschitz constant of \mathbf{F}). In practice, it is convenient to choose $\mathbf{u}_0 = \mathbf{x}$.

In addition, there is no analytical expression for the Jacobian determinant, whose exact computation costs $O(D^3)$. However, there is a computationally efficient stochastic estimator of the log Jacobian determinant. The idea is to express the log Jacobian determinant as a power series. Using the fact that $\mathbf{f}(\mathbf{x}) = \mathbf{x} + \mathbf{F}(\mathbf{x})$, we have

$$\log |\det \mathbf{J}(\mathbf{f})| = \log |\det(\mathbf{I} + \mathbf{J}(\mathbf{F}))| = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \text{tr}[\mathbf{J}(\mathbf{F})^k]. \quad (24.41)$$

¹ This power series converges when the matrix norm of $\mathbf{J}(\mathbf{F})$ is less than 1, which here is guaranteed
² exactly because \mathbf{F} is a contraction. The trace of $\mathbf{J}(\mathbf{F})^k$ can be efficiently approximated using
³ Jacobian-vector products via the **Hutchinson trace estimator** [Ski89; Hut89; Mey+21]:
⁴

$$\text{tr}[\mathbf{J}(\mathbf{F})^k] \approx \mathbf{v}^\top \mathbf{J}(\mathbf{F})^k \mathbf{v}, \quad (24.42)$$

⁵ where \mathbf{v} is a sample from a distribution with zero mean and unit covariance, such as $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
⁶ Finally, the infinite series can be approximated by a finite one either by truncation [Beh+19], which
⁷ unfortunately yields a biased estimator, or by employing the **Russian-roulette estimator** [Che+19],
⁸ which is unbiased.
⁹

¹⁰

¹¹ 24.2.5.2 Residual blocks with low-rank Jacobian

¹² There is an efficient way of computing the determinant of a matrix which is a low-rank perturbation
¹³ of an identity matrix. Suppose \mathbf{A} and \mathbf{B} are matrices, where \mathbf{A} is $D \times M$ and \mathbf{B} is $M \times D$. The
¹⁴ following formula is known as the **Weinstein–Aronszajn identity**², and is a special case of the
¹⁵ more general **matrix determinant lemma**:
¹⁶

$$\det(\mathbf{I}_D + \mathbf{AB}) = \det(\mathbf{I}_M + \mathbf{BA}). \quad (24.43)$$

¹⁷ We write \mathbf{I}_D and \mathbf{I}_M for the $D \times D$ and $M \times M$ identity matrices respectively. The significance of
¹⁸ this formula is that it turns a $D \times D$ determinant that costs $O(D^3)$ into an $M \times M$ determinant
¹⁹ that costs $O(M^3)$. If M is smaller than D , this saves computation.
²⁰

²¹ With some restrictions on the residual block $\mathbf{F} : \mathbb{R}^D \rightarrow \mathbb{R}^D$, we can apply this formula to compute
²² the determinant of a residual connection efficiently. The trick is to create a bottleneck inside \mathbf{F} . We
²³ do that by defining $\mathbf{F} = \mathbf{F}_2 \circ \mathbf{F}_1$, where $\mathbf{F}_1 : \mathbb{R}^D \rightarrow \mathbb{R}^M$, $\mathbf{F}_2 : \mathbb{R}^M \rightarrow \mathbb{R}^D$ and $M \ll D$. The chain
²⁴ rule gives $\mathbf{J}(\mathbf{F}) = \mathbf{J}(\mathbf{F}_2)\mathbf{J}(\mathbf{F}_1)$, where $\mathbf{J}(\mathbf{F}_2)$ is $D \times M$ and $\mathbf{J}(\mathbf{F}_1)$ is $M \times D$. Now we can apply our
²⁵ determinant formula as follows:
²⁶

$$\det \mathbf{J}(\mathbf{f}) = \det(\mathbf{I}_D + \mathbf{J}(\mathbf{F})) = \det(\mathbf{I}_D + \mathbf{J}(\mathbf{F}_2)\mathbf{J}(\mathbf{F}_1)) = \det(\mathbf{I}_M + \mathbf{J}(\mathbf{F}_1)\mathbf{J}(\mathbf{F}_2)). \quad (24.44)$$

²⁷ Since the final determinant costs $O(M^3)$, we can make the Jacobian determinant efficient by reducing
²⁸ M , that is, by narrowing the bottleneck.
²⁹

³⁰ An example of the above is the **planar flow** of [RM15b]. In this model, each residual block is an
³¹ MLP with one hidden layer and one hidden unit. That is,
³²

$$\mathbf{f}(\mathbf{u}) = \mathbf{u} + \mathbf{v}\sigma(\mathbf{w}^\top \mathbf{u} + b), \quad (24.45)$$

³³ where $\mathbf{v} \in \mathbb{R}^D$, $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the parameters, and σ is the activation function. The residual
³⁴ block is the composition of $\mathbf{F}_1(\mathbf{u}) = \mathbf{w}^\top \mathbf{u} + b$ and $\mathbf{F}_2(z) = \mathbf{v}\sigma(z)$, so $M = 1$. Their Jacobians
³⁵ are $\mathbf{J}(\mathbf{F}_1)(\mathbf{u}) = \mathbf{w}^\top$ and $\mathbf{J}(\mathbf{F}_2)(z) = \mathbf{v}\sigma'(z)$. Substituting these in the formula for the Jacobian
³⁶ determinant we obtain:
³⁷

$$\det \mathbf{J}(\mathbf{f})(\mathbf{u}) = 1 + \mathbf{w}^\top \mathbf{v}\sigma'(\mathbf{w}^\top \mathbf{u} + b), \quad (24.46)$$

³⁸ which can be computed efficiently in $O(D)$. Other examples include the **circular flow** of [RM15b]
³⁹ and the **Sylvester flow** of [Ber+18].
⁴⁰

⁴¹ 2. See https://en.wikipedia.org/wiki/Weinstein–Aronszajn_identity.

This technique gives an efficient way of computing determinants of residual connections with bottlenecks, but in general there is no guarantee that such functions are invertible. This means that invertibility must be satisfied on a case-by-case basis. For example, the planar flow is invertible when σ is the hyperbolic tangent and $\mathbf{w}^\top \mathbf{v} > -1$, but otherwise it may not be.

24.2.6 Continuous-time flows

So far we have discussed flows that consist of a sequence of bijections $\mathbf{f}_1, \dots, \mathbf{f}_N$. Starting from some input $\mathbf{x}_0 = \mathbf{u}$, this creates a sequence of outputs $\mathbf{x}_1, \dots, \mathbf{x}_N$ where $\mathbf{x}_n = \mathbf{f}_n(\mathbf{x}_{n-1})$. However, we can also have flows where the input is transformed into the final output in a continuous way. That is, we start from $\mathbf{x}_0 = \mathbf{x}(0)$, create a continuously-indexed sequence $\mathbf{x}(t)$ for $t \in [0, T]$ with some fixed T , and take $\mathbf{x}(T)$ to be the final output. Thinking of t as analogous to time, we refer to these as **continuous-time flows**.

The sequence $\mathbf{x}(t)$ is defined as the solution to a first-order ordinary differential equation (ODE) of the form:

$$\frac{d\mathbf{x}}{dt}(t) = \mathbf{F}(\mathbf{x}(t), t). \quad (24.47)$$

The function $\mathbf{F} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$ is a time-dependent vector field that parameterizes the ODE. If we think of $\mathbf{x}(t)$ as the position of a particle in D dimensions, the vector $\mathbf{F}(\mathbf{x}(t), t)$ determines the particle's velocity at time t .

The flow (for time T) is a function $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ that takes in an input \mathbf{x}_0 , solves the ODE with initial condition $\mathbf{x}(0) = \mathbf{x}_0$, and returns $\mathbf{x}(T)$. The function \mathbf{f} is a well-defined bijection if the solution to the ODE exists for all $t \in [0, T]$ and is unique. These conditions are not generally satisfied for arbitrary \mathbf{F} , but they are if $\mathbf{F}(\cdot, t)$ is Lipschitz continuous with a Lipschitz constant that does not depend on t . That is, \mathbf{f} is a well-defined bijection if there exists a constant L such that for all $\mathbf{x}_1, \mathbf{x}_2$ and $t \in [0, T]$ we have:

$$\|\mathbf{F}(\mathbf{x}_1, t) - \mathbf{F}(\mathbf{x}_2, t)\| \leq L \|\mathbf{x}_1 - \mathbf{x}_2\|. \quad (24.48)$$

This result is a consequence of the **Picard–Lindelöf theorem** for ODEs.³ In practice, we can parameterize \mathbf{F} using any choice of model, provided the Lipschitz condition is met.

Usually the ODE cannot be solved analytically, but we can solve it approximately by discretizing it. A simple example is **Euler's method**, which corresponds to the following discretization for some small step size $\epsilon > 0$:

$$\mathbf{x}(t + \epsilon) = \mathbf{x}(t) + \epsilon \mathbf{F}(\mathbf{x}(t), t). \quad (24.49)$$

This is equivalent to a residual connection with residual block $\epsilon \mathbf{F}(\cdot, t)$, so the ODE solver can be thought of as a deep residual network with $O(T/\epsilon)$ layers. A smaller step size leads to a more accurate solution, but also to more computation. There are several other solution methods varying in accuracy and sophistication, such as those in the broader Runge–Kutta family, some of which use adaptive step sizes.

The inverse of \mathbf{f} can be easily computed by solving the ODE in reverse. That is, to compute $\mathbf{f}^{-1}(\mathbf{x}_T)$ we solve the ODE with initial condition $\mathbf{x}(T) = \mathbf{x}_T$ and return $\mathbf{x}(0)$. Unlike some other

³. See https://en.wikipedia.org/wiki/Picard-Lindelöf_theorem

¹ flows (such as autoregressive flows) which are more expensive to compute in one direction than in
² the other, continuous-time flows require the same amount of computation in either direction.
³

⁴ In general, there is no analytical expression for the Jacobian determinant of \mathbf{f} . However, we can
⁵ express it as the solution to a separate ODE, which we can then solve numerically. First, we define
⁶ $\mathbf{f}_t : \mathbb{R}^D \rightarrow \mathbb{R}^D$ to be the flow for time t , that is, the function that takes \mathbf{x}_0 , solves the ODE with
⁷ initial condition $\mathbf{x}(0) = \mathbf{x}_0$ and returns $\mathbf{x}(t)$. Clearly, \mathbf{f}_0 is the identity function and $\mathbf{f}_T = \mathbf{f}$. Let us
⁸ define $L(t) = \log |\det \mathbf{J}(\mathbf{f}_t)(\mathbf{x}_0)|$. Because \mathbf{f}_0 is the identity function, $L(0) = 0$, and because $\mathbf{f}_T = \mathbf{f}$,
⁹ $L(T)$ gives the Jacobian determinant of \mathbf{f} that we are interested in. It can be shown that L satisfies
¹⁰ the following ODE:

$$\frac{dL}{dt}(t) = \text{tr}[\mathbf{J}(\mathbf{F}(\cdot, t))(\mathbf{x}(t))]. \quad (24.50)$$

¹¹ That is, the rate of change of L at time t is equal to the Jacobian trace of $\mathbf{F}(\cdot, t)$ evaluated at $\mathbf{x}(t)$. So
¹² we can compute $L(T)$ by solving the above ODE with initial condition $L(0) = 0$. Moreover, we can
¹³ compute $\mathbf{x}(T)$ and $L(T)$ simultaneously, by combining their two ODEs into a single ODE operating
¹⁴ on the extended space (\mathbf{x}, L) .

¹⁵ An example of a continuous-time flow is the **Neural ODE** model of [Che+18c], which uses a
¹⁶ neural network to parameterize \mathbf{F} . To avoid backpropagating gradients through the ODE solver,
¹⁷ which can be computationally demanding, they use the **adjoint sensitivity method** to express the
¹⁸ time evolution of the gradient with respect to $\mathbf{x}(t)$ as a separate ODE. Solving this ODE gives the
¹⁹ required gradients, and can be thought of as the continuous-time analogue of backpropagation.

²⁰ Another example is the **FFJORD** model of [Gra+18]. This is similar to the Neural ODE model,
²¹ except that it uses the Hutchinson trace estimator to approximate the Jacobian trace of $\mathbf{F}(\cdot, t)$.
²² This usage of the Hutchinson trace estimator is analogous to that in contractive residual flows
²³ (Section 24.2.5.1), and it speeds up computation in exchange for a stochastic (but unbiased) estimate.
²⁴

²⁵ 24.3 Applications

²⁶ In this section, we highlight some applications of flows for canonical probabilistic machine learning
²⁷ tasks.
²⁸

²⁹ 24.3.1 Density estimation

³⁰ Flow models allow exact density computation and can be used to fit multi-modal densities to observed
³¹ data. An early example is Gaussianization [CG00] who applied this idea to fit low-dimensional
³² densities. Tabak and Vanden-Eijnden [TVE10] and Tabak and Turner [TT13] introduced the modern
³³ idea of flows (including the term ‘normalizing flows’), describing a flow as a composition of simpler
³⁴ maps. Deep density models [RA13] was one of the first to use neural networks for flows to parametrize
³⁵ high-dimensional densities. There has been a rich line of follow-up work including **NICE** [DKB15]
³⁶ and **Real NVP** [DSDB17]. (NVP stands for “non-volume preserving”, which refers to the fact that
³⁷ the Jacobian of the transform is not unity.) Masked autoregressive flows (Section 24.2.4.2) further
³⁸ improved performance on unconditional and conditional density estimation tasks.
³⁹

⁴⁰ Flows can be used for *hybrid models* which model the joint density of inputs and targets $p(\mathbf{x}, y)$, as
⁴¹ opposed to discriminative classification models which just model the conditional $p(y|\mathbf{x})$ and density
⁴² models which just model the marginal $p(\mathbf{x})$. Nalisnick et al. [Nal+19b] proposed a flow-based hybrid
⁴³
⁴⁴

1 model using invertible mappings for representation learning and showed that the joint density $p(\mathbf{x}, y)$
2 can be computed efficiently, which can be useful for downstream tasks such as anomaly detection,
3 semi-supervised learning and selective classification. Flow-based hybrid models are memory-efficient
4 since most of the parameters are in the invertible representation which are shared between the
5 discriminative and generative models; furthermore, the density $p(\mathbf{x}, y)$ can be computed in a single
6 forward pass leading to computational savings. Residual flows [Che+19] use invertible residual
7 mappings [Beh+19] for hybrid modeling which further improves performance. Flows have also been
8 used to fit densities to embeddings [Zha+20b; CZG20] for anomaly detection tasks.
9

10 24.3.2 Generative Modeling

11 Another task is generation, which involves generating novel samples from a fitted model $p^*(\mathbf{x})$.
12 Generation is a popular downstream task for normalizing flows, which have been applied for different
13 data modalities including images, video, audio, text and structured objects such as graphs and point
14 clouds. Images are arguably the most popular modality for deep generative models: GLOW [KD18b]
15 was one of the first flow-based models to generate compelling high-dimensional images, and has been
16 extended to video to produce RGB frames [Kum+19b]; residual flows [Che+19] have also been shown
17 to produce sharp images.
18

19 Oord et al. [Oor+18] used flows for audio synthesis by distilling WaveNet into an IAF (Section 24.2.4.3), which enables faster sampling than WaveNet. Other flow models for audio include
20 WaveFLOW [PVC19] and FlowWaveNet [Kim+19], which directly speed up WaveNet using coupling
21 layers.
22

23 Flows have been also used for text. Tran et al. [Tra+19] define a discrete flow over a vocabulary
24 for language-modeling tasks. Another popular approach is to define a latent variable model with
25 discrete observation space but a continuous latent space. For example, Ziegler and Rush [ZR19a] use
26 normalizing flows in latent space for language modeling.
27

28 24.3.3 Inference

29 Normalizing flows have been used for probabilistic inference. Rezende and Mohamed [RM15b]
30 popularized normalizing flows in machine learning, and showed how they can be used for modeling
31 variational posterior distributions in latent variable models. Various extensions such as Householder
32 flows [TW16], inverse autoregressive flows [Kin+16], multiplicative normalizing flows [LW17] and
33 Sylsvester flows [Ber+18] have been proposed for modeling the variational posterior for latent variable
34 models as well as posteriors for Bayesian neural networks.
35

36 Flows have been used as complex proposal distributions for importance sampling; examples include
37 neural importance sampling [Mül+19b] and Boltzmann generators [Noé+19]. Hoffman et al. [Hof+19]
38 used flows to improve the performance of Hamiltonian Monte Carlo (Section 12.5) by defining bijective
39 transformations to transform random variables to simpler distributions and performing HMC in that
40 space instead.
41

42 Finally, flows can be used in the context of simulation-based inference, where the likelihood function
43 of the parameters is not available, but simulating data from the model is possible. The main idea is
44 to train a flow on data simulated from the model in order to approximate the posterior distribution
45 or the likelihood function. The flow model can also be used to guide simulations in order to make
46 inference more efficient [PSM19; GNM19]. This approach has been used for inference of simulation
47

1 models in cosmology [Als+19] and computational neuroscience [Gon+20].
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

25 Energy-based models

This chapter was co-authored with Yang Song and Durk Kingma, and is an extension of [SK21].

25.1 Introduction

Probabilistic models with a tractable likelihood, such as auto-regressive models (Chapter 23) and normalizing flows (Chapter 24), are a double-edged sword. On the one hand, a tractable likelihood allows for straightforward comparison between models, and straightforward optimization of the model parameters w.r.t. the log-likelihood of the data. On the other hand, the set of models with a tractable likelihood is constrained. For example, in the case of autoregressive models, the model distribution is factorized as a product of conditional distributions, and in flow-based generative models the data is modeled as an invertible transformation of a base distribution. Variational autoencoders (Chapter 22) offer more freedom, at the cost of optimizing a lower bound on the log-likelihood, as opposed to the exact likelihood.

All of the above models can be formulated in terms of directed graphical models (Chapter 4), where we generate the data one step at a time, using locally normalized distributions. In some cases, it is easier to specify a distribution in terms of a set of constraints that valid samples must satisfy, rather than a generative process. This can be done using an undirected graphical model (Chapter 4).

Energy-based models or **EBM** are similar to undirected graphical models, but they do not necessarily make any Markov assumptions. Thus they can be written as a Gibbs distribution, as follows:

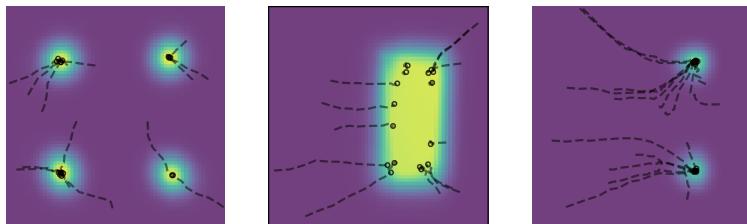
$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}} \quad (25.1)$$

where $E_{\theta}(\mathbf{x}) \geq 0$ is known as the **energy function** with parameters θ , and Z_{θ} is the **partition function**:

$$Z_{\theta} = \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.2)$$

This is constant w.r.t. \mathbf{x} but is a function of θ . In general, evaluating this integral is intractable, although we can use approximate methods, such as annealed importance sampling, discussed in Section 11.5.4.1.

Since the energy function does not need to integrate to one, and just needs to output a single scalar, it can be implemented using a variety of neural network architectures. As such, EBMs have



11 *Figure 25.1: Combining two energy functions in 2d by summation, which is equivalent to multiplying the*
12 *corresponding probability densities. We also illustrate some sampled trajectories towards high probability (low*
13 *energy) regions. From Figure 14 of [DM19a]. Used with kind permission of Yilun Du.*

14

15 found wide applications in many fields of machine learning, including image generation [Ngi+11;
16 Xie+16; DM19b], discriminative learning [Gra+20b], natural processing [Mik+13; Den+20], density
17 estimation [Wen+19a; Son+19] and reinforcement learning [Haa+17; Haa+18a], to list a few.
18

20 25.1.1 Example: Products of experts (PoE)

21

22 As an example of why energy based models are useful, suppose we want to create a generative model
23 of proteins that are thermally stable at room temperature, and which bind to the COVID-19 spike
24 receptor. Suppose $p_1(\mathbf{x})$ can generate stable proteins and $p_2(\mathbf{x})$ can generate proteins that bind.
25 (For example, both of these models could be autoregressive sequence models, trained on different
26 datasets.) We can view each of these models as “experts” about a particular aspect of the data.
27 On their own, they are not an adequate model of the data that we have (or want to have), but we
28 can then combine them, to represent the **conjunction of features**, by computing a **product of**
29 **experts (PoE)** [Hin02]:

30

$$\underline{31} \quad p_{12}(\mathbf{x}) = \frac{1}{Z_{12}} p_1(\mathbf{x}) p_2(\mathbf{x}) \quad (25.3)$$

32

33 This will assign high probability to proteins that are stable and which bind, and low probability to
34 all others. By contrast, a **mixture of experts** would either generate from p_1 or from p_2 , but would
35 not combine features from both.

36 If the experts are represented as energy based models (EBM), then the PoE model is also an EBM,
37 with an energy given by

38

$$\underline{39} \quad \mathcal{E}_{12}(\mathbf{x}) = \mathcal{E}_1(\mathbf{x}) + \mathcal{E}_2(\mathbf{x}) \quad (25.4)$$

40 Intuitively, we can think of each component of energy as a “**soft constraint**” on the data. This idea
41 is illustrated in Figure 25.1.

42

43 25.1.2 Computational difficulties

44

45 Although the flexibility of EBMs can provide significant modeling advantages, computation of the
46 likelihood and drawing samples from the model are generally intractable. In this chapter, we will
47

1 discuss a variety of approximate methods to solve these problems.
2

4 25.2 Maximum Likelihood Training 5

6 The *de facto* standard for learning probabilistic models from i.i.d. data is maximum likelihood
7 estimation (MLE). Let $p_{\theta}(\mathbf{x})$ be a probabilistic model parameterized by θ , and $p_{\text{data}}(\mathbf{x})$ be the
8 underlying data distribution of a dataset. We can fit $p_{\theta}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by maximizing the expected
9 log-likelihood function over the data distribution, defined by

$$\ell(\theta) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log p_{\theta}(\mathbf{x})] \quad (25.5)$$

10 as a function of θ . Here the expectation can be easily estimated with samples from the dataset.
11 Maximizing likelihood is equivalent to minimizing the KL divergence between $p_{\text{data}}(\mathbf{x})$ and $p_{\theta}(\mathbf{x})$,
12 because

$$\ell(\theta) = D_{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log p_{\text{data}}(\mathbf{x})] \quad (25.6)$$

$$= D_{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) - \text{constant}, \quad (25.7)$$

13 where the second equality holds because $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log p_{\text{data}}(\mathbf{x})]$ does not depend on θ .
14

15 We cannot usually compute the likelihood of an EBM because the normalizing constant Z_{θ}
16 is often intractable. Nevertheless, we can still estimate the gradient of the log-likelihood with
17 MCMC approaches, allowing for likelihood maximization with stochastic gradient ascent [You99]. In
18 particular, the gradient of the log-probability of an EBM decomposes as a sum of two terms:

$$\nabla_{\theta} \log p_{\theta}(\mathbf{x}) = -\nabla_{\theta} E_{\theta}(\mathbf{x}) - \nabla_{\theta} \log Z_{\theta}. \quad (25.8)$$

19 The first gradient term, $-\nabla_{\theta} E_{\theta}(\mathbf{x})$, is straightforward to evaluate with automatic differentiation. The
20 challenge is in approximating the second gradient term, $\nabla_{\theta} \log Z_{\theta}$, which is intractable to compute
21 exactly. This gradient term can be rewritten as the following expectation:
22

$$\nabla_{\theta} \log Z_{\theta} = \nabla_{\theta} \log \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.9)$$

$$\stackrel{(i)}{=} \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \nabla_{\theta} \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.10)$$

$$= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \nabla_{\theta} \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.11)$$

$$\stackrel{(ii)}{=} \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.12)$$

$$= \int \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.13)$$

$$\stackrel{(iii)}{=} \int \exp(-E_{\theta}(\mathbf{x}) - Z_{\theta}) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.14)$$

$$\stackrel{(iv)}{=} \int p_{\theta}(\mathbf{x}) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \quad (25.15)$$

$$= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [-\nabla_{\theta} E_{\theta}(\mathbf{x})], \quad (25.16)$$

where steps (i) and (ii) are due to the chain rule of gradients, and (iii) and (iv) are from definitions in Equations (25.1) and (25.2). Thus, we can obtain an unbiased one sample Monte Carlo estimate of the log-likelihood gradient by using

$$\nabla_{\theta} \log Z_{\theta} \simeq -\frac{1}{S} \sum_{s=1}^S \nabla_{\theta} E_{\theta}(\tilde{\mathbf{x}}_s), \quad (25.17)$$

where $\tilde{\mathbf{x}}_s \sim p_{\theta}(\mathbf{x})$, i.e., a random sample from the distribution over \mathbf{x} given by the EBM. Therefore, as long as we can draw random samples from the model, we have access to an unbiased Monte Carlo estimate of the log-likelihood gradient, allowing us to optimize the parameters with stochastic gradient ascent.

Much of the literature has focused on methods for efficient MCMC sampling from EBMs. We discuss some of these methods below.

25.2.1 Gradient-based MCMC methods

Some efficient MCMC methods, such as **Langevin MCMC** (Section 12.5.6) or Hamiltonian Monte Carlo (Section 12.5), make use of the fact that the gradient of the log-probability w.r.t. \mathbf{x} (known as the **score function**) is equal to the (negative) gradient of the energy, and is therefore easy to calculate:

$$\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_{\theta}}_{=0} = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}). \quad (25.18)$$

For example, when using Langevin MCMC to sample from $p_{\theta}(\mathbf{x})$, we first draw an initial sample \mathbf{x}^0 from a simple prior distribution, and then simulate an overdamped Langevin diffusion process for K steps with step size $\epsilon > 0$:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \frac{\epsilon^2}{2} \underbrace{\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}^k)}_{=-\nabla_{\mathbf{x}} E(\mathbf{x})} + \epsilon \mathbf{z}^k, \quad k = 0, 1, \dots, K-1. \quad (25.19)$$

where $\mathbf{z}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a Gaussian noise term. We show an example of this process in Figure 25.3d. When $\epsilon \rightarrow 0$ and $K \rightarrow \infty$, \mathbf{x}^K is guaranteed to distribute as $p_{\theta}(\mathbf{x})$ under some regularity conditions. In practice we have to use a small finite ϵ , but the discretization error is typically negligible, or can be corrected with a Metropolis-Hastings step (Section 12.2), leading to the Metropolis-Adjusted Langevin Algorithm (Section 12.5.6).

25.2.2 Contrastive divergence

Running MCMC till convergence to obtain a sample $\mathbf{x} \sim p_{\theta}(\mathbf{x})$ can be computationally expensive. Therefore we typically need approximations to make MCMC-based learning of EBMs practical. One popular method for doing so is **contrastive divergence** (CD) [Hin02]. In CD, one initializes the MCMC chain from the datapoint \mathbf{x} , and proceeds to perform MCMC for a fixed number of steps. One can show that T steps of CD minimizes the following objective:

$$\text{CD}_T = D_{\text{KL}}(p_0 \| p_{\infty}) - D_{\text{KL}}(p_T \| p_{\infty}) \quad (25.20)$$

where p_t is the distribution over \mathbf{x} after t MCMC updates, and p_0 is the data distribution. Typically we can get good results with a small value of T , sometimes just $T = 1$. We give the details below.

25.2.2.1 Fitting RBMs with CD

CD was initially developed to fit a special kind of latent variable EBM known as a restricted Boltzmann machine (Section 4.3.2.4). This model was specially designed to support fast block Gibbs sampling, which is required by CD (and can also be exploited by standard MCMC-based learning methods [AHS85].)

For simplicity, we will assume the hidden and visible nodes are binary, and we use 1-step contrastive divergence. As discussed in the supplementary material, the binary RBM has the following energy function:

$$\mathcal{E}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) = \sum_{d=1}^D \sum_{k=1}^K x_d z_k W_{dk} + \sum_{d=1}^D x_d b_d + \sum_{k=1}^K z_k c_k \quad (25.21)$$

(Henceforth we will drop the unary (bias) terms, which can be emulated by clamping $z_k = 1$ or $x_d = 1$.) This is a loglinear model where we have one binary feature per edge. Thus from Equation (4.115) the gradient of the log-likelihood is given by the clamped expectations minus the unclamped expectations:

$$\frac{\partial \ell}{\partial w_{dk}} = \frac{1}{N} \sum_{n=1}^N \mathbb{E}[x_d z_k | \mathbf{x}_n, \boldsymbol{\theta}] - \mathbb{E}[x_d z_k | \boldsymbol{\theta}] \quad (25.22)$$

We can write rewrite the above gradient in matrix-vector form as follows:

$$\nabla_{\mathbf{w}} \ell = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})p(\mathbf{z}|\boldsymbol{\theta})} [\mathbf{x}\mathbf{z}^T] - \mathbb{E}_{p(\mathbf{z}|\boldsymbol{\theta})} [\mathbf{x}\mathbf{z}^T] \quad (25.23)$$

(We can derive a similar expression for the gradient of the bias terms by setting $x_d = 1$ or $z_k = 1$.)

The first term in the expression for the gradient in Equation (25.22), when \mathbf{x} is fixed to a data case, is sometimes called the **clamped phase**, and the second term, when \mathbf{x} is free, is sometimes called the **unclamped phase**. When the model expectations match the empirical expectations, the two terms cancel out, the gradient becomes zero and learning stops.

We can also make a connection to the principle of **Hebbian learning** in neuroscience. In particular, Hebb's rule says that the strength of connection between two neurons that are simultaneously active should be increased. (This theory is often summarized as "Cells that fire together wire together".¹) The first term in Equation (25.22) is therefore consider a Hebbian term, and the second an anti-Hebbian term, due to the sign change.

We can leverage the Markov structure of the bipartite graph to approximate the expectations as follows:

$$\mathbf{z}_n \sim p(\mathbf{z} | \mathbf{x}_n, \boldsymbol{\theta}) \quad (25.24)$$

$$\mathbf{x}'_n \sim p(\mathbf{x} | \mathbf{z}_n, \boldsymbol{\theta}) \quad (25.25)$$

$$\mathbf{z}'_n \sim p(\mathbf{z} | \mathbf{x}'_n, \boldsymbol{\theta}) \quad (25.26)$$

¹. See https://en.wikipedia.org/wiki/Hebbian_theory.

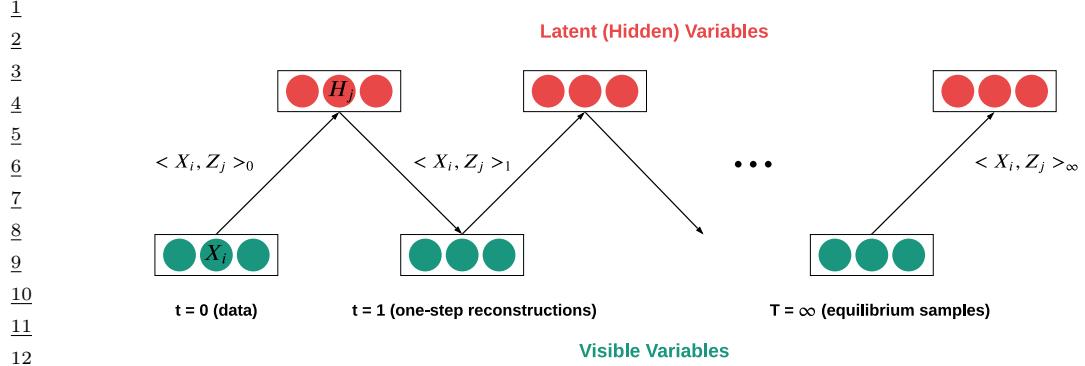


Figure 25.2: Illustration of contrastive divergence sampling for an RBM. The visible nodes are initialized at a datapoint, then we sample a hidden vector, then another visible vector, etc. Eventually (at “infinity”) we will be producing samples from the joint distribution $p(\mathbf{x}, \mathbf{z}|\theta)$.

We can think of \mathbf{x}'_n as the model’s best attempt at reconstructing \mathbf{x}_n after being encoded and then decoded by the model. Such samples are sometimes called **fantasy data**. See Figure 25.2 for an illustration. Given these samples, we then make the approximation

$$\mathbb{E}_{p(\cdot|\theta)} [\mathbf{x}\mathbf{z}^\top] \approx \mathbf{x}_n(\mathbf{z}'_n)^\top \quad (25.27)$$

In practice, it is common to use $\mathbb{E}[\mathbf{z}|\mathbf{x}'_n]$ instead of a sampled value \mathbf{z}'_n in the above expression, since this reduces the variance. However, it is not valid to use $\mathbb{E}[\mathbf{z}|\mathbf{x}_n]$ instead of sampling $\mathbf{z}_n \sim p(\mathbf{z}|\mathbf{x}_n)$ in Equation (25.24), because then each hidden unit would be able to pass more than 1 bit of information, so it would not act as much of a bottleneck.

The whole procedure is summarized in Algorithm 28. For more details, see [Hin10; Swe+10].

Algorithm 28: CD-1 training for an RBM with binary hidden and visible units

```

32 1 Initialize weights  $\mathbf{W} \in \mathbb{R}^{D \times K}$  randomly;
33 2 for  $t = 1, 2, \dots$  do
34 3   for each minibatch of size  $B$  do
35 4     Set minibatch gradient to zero,  $\mathbf{g} := \mathbf{0}$  ;
36 5     for each case  $\mathbf{x}_n$  in the minibatch do
37 6       Compute  $\boldsymbol{\mu}_n = \mathbb{E}[\mathbf{z}|\mathbf{x}_n, \mathbf{W}]$ ;
38 7       Sample  $\mathbf{z}_n \sim p(\mathbf{z}|\mathbf{x}_n, \mathbf{W})$ ;
39 8       Sample  $\mathbf{x}'_n \sim p(\mathbf{x}|\mathbf{z}_n, \mathbf{W})$ ;
40 9       Compute  $\boldsymbol{\mu}'_n = \mathbb{E}[\mathbf{z}|\mathbf{x}'_n, \mathbf{W}]$ ;
41 10      Compute gradient  $\nabla_{\mathbf{W}} = (\mathbf{x}_n)(\boldsymbol{\mu}_n)^\top - (\mathbf{x}'_n)(\boldsymbol{\mu}'_n)^\top$  ;
42 11      Accumulate  $\mathbf{g} := \mathbf{g} + \nabla_{\mathbf{w}}$ ;
43 12    Update parameters  $\mathbf{W} := \mathbf{W} + \eta_t \frac{1}{B} \mathbf{g}$ 
44

```

25.2.2.2 Persistent CD

One variant of CD that sometimes performs better is **persistent contrastive divergence** (PCD) [Tie08]. In this approach, a single MCMC chain with a persistent state is employed to sample from the EBM. In PCD, we do not restart the MCMC chain when training on a new datapoint; rather, we carry over the state of the previous MCMC chain and use it to initialize a new MCMC chain for the next training step. See Line 12 for some pseudocode. Hence there are two dynamical processes running at different time scales: the states \mathbf{x} change quickly, and the parameters $\boldsymbol{\theta}$ change slowly.

Algorithm 29: Persistent MCMC-SGD for fitting an EBM

```

1 Initialize parameters  $\boldsymbol{\theta}$  randomly;
2 Initialize chains  $\tilde{\mathbf{x}}_{1:S}$  randomly ;
3 Initialize learning rate  $\eta$ ;
4 for  $t = 1, 2, \dots$  do
5   for  $\mathbf{x}_b$  in minibatch of size  $B$  do
6      $\mathbf{g}_b = \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}_b)$ 
7     for sample  $s = 1 : S$  do
8       Sample  $\tilde{\mathbf{x}}_s \sim \text{MCMC}(\text{target} = p(\cdot | \boldsymbol{\theta}), \text{init} = \tilde{\mathbf{x}}_s, \text{nsteps} = N)$  ;
9        $\tilde{\mathbf{g}}_s = \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_s)$  ;
10       $\mathbf{g}_t = -(\frac{1}{B} \sum_{b=1}^B \mathbf{g}_b) - (\frac{1}{S} \sum_{s=1}^S \tilde{\mathbf{g}}_s)$  ;
11       $\boldsymbol{\theta} := \boldsymbol{\theta} + \eta \mathbf{g}_t$ ;
12      Decrease step size  $\eta$ ;

```

A theoretical justification for this was given in [You89], who showed that we can start the MCMC chain at its previous value, and just take a few steps, because $p(\mathbf{x}|\boldsymbol{\theta}_t)$ is likely to be close to $p(\mathbf{x}|\boldsymbol{\theta}_{t-1})$, since we only changed the parameters by a small amount in the intervening SGD step.

25.2.2.3 Other methods

PCD can be further improved by keeping multiple historical states of the MCMC chain in a replay buffer and initialize new MCMC chains by randomly sampling from it [DM19b]. Other variants of CD include mean field CD [WH02], and multi-grid CD [Gao+18].

EBMs trained with CD may not capture the data distribution faithfully, since truncated MCMC can lead to biased gradient updates that hurt the learning dynamics [SMB10; FI10; Nij+19]. There are several methods that focus on removing this bias for improved MCMC training. For example, one line of work proposes unbiased estimators of the gradient through coupled MCMC [JOA17; QZW19]; and Du et al. [Du+20] propose to reduce the bias by differentiating through the MCMC sampling algorithm and estimating an entropy correction term.

25.3 Score Matching (SM)

If two continuously differentiable real-valued functions $f(\mathbf{x})$ and $g(\mathbf{x})$ have equal first derivatives everywhere, then $f(\mathbf{x}) \equiv g(\mathbf{x}) + \text{constant}$. When $f(\mathbf{x})$ and $g(\mathbf{x})$ are log probability density functions

(PDFs) with equal first derivatives, the normalization requirement (Equation (25.1)) implies that $\int \exp(f(\mathbf{x}))d\mathbf{x} = \int \exp(g(\mathbf{x}))d\mathbf{x} = 1$, and therefore $f(\mathbf{x}) \equiv g(\mathbf{x})$. As a result, one can learn an EBM by (approximately) matching the first derivatives of its log-PDF to the first derivatives of the log-PDF of the data distribution. If they match, then the EBM captures the data distribution exactly. The first-order gradient function of a log-PDF is also called the **score** of that distribution. For training EBMs, it is useful to transform the equivalence of distributions to the equivalence of scores, because the score of an EBM can be easily obtained as follows:

$$\mathbf{s}_{\theta}(\mathbf{x}) \triangleq \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}) \quad (25.28)$$

We see that this does not involve the typically intractable normalizing constant Z_{θ} .

Let $p_{\text{data}}(\mathbf{x})$ be the underlying data distribution, from which we have a finite number of i.i.d. samples but do not know its PDF. The **score matching** objective [Hyv05] minimizes a discrepancy between two distributions called the **Fisher divergence**:

$$D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})\|^2 \right]. \quad (25.29)$$

The expectation w.r.t. $p_{\text{data}}(\mathbf{x})$, in this objective and its variants below, admits a trivial unbiased Monte Carlo estimator using the empirical mean of samples $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$. However, the second term of Equation (25.29), $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$, is generally impractical to calculate since it requires knowing the PDF of $p_{\text{data}}(\mathbf{x})$. We discuss a solution to this below.

25.3.1 Basic score matching

Hyvärinen [Hyv05] shows that, under certain regularity conditions, the Fisher divergence can be rewritten using integration by parts, with second derivatives of $E_{\theta}(\mathbf{x})$ replacing the unknown first derivatives of $p_{\text{data}}(\mathbf{x})$:

$$D_F(p_{\text{data}}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} \right)^2 + \frac{\partial^2 E_{\theta}(\mathbf{x})}{(\partial x_i)^2} \right] + \text{constant} \quad (25.30)$$

$$= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \|\mathbf{s}_{\theta}(\mathbf{x})\|^2 + \text{tr}(\mathbf{J}_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x})) \right] + \text{constant} \quad (25.31)$$

where d is the dimensionality of \mathbf{x} , and $\mathbf{J}_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x})$ is the Jacobian of the score function. The constant does not affect optimization and thus can be dropped for training. It is shown by [Hyv05] that estimators based on Score Matching are consistent under some regularity conditions, meaning that the parameter estimator obtained by minimizing Equation (25.29) converges to the true parameters in the limit of infinite data. See Figure 25.3 for an example.

An important downside of the objective Equation (25.31) is that it takes $O(d^2)$ time to compute the trace of the Jacobian. For this reason, the implicit SM formulation of Equation (25.31) has only been applied to relatively simple energy functions where computation of the second derivatives is tractable.

Score Matching assumes a continuous data distribution with positive density over the space, but it can be generalized to discrete or bounded data distributions [Hyv07b; Lyu12]. It is also possible to consider higher-order gradients of log-PDFs beyond first derivatives [PDL+12].

25.3.2 Denoising Score Matching (DSM)

The Score Matching objective in Equation (25.31) requires several regularity conditions for $\log p_{\text{data}}(\mathbf{x})$, *e.g.*, it should be continuously differentiable and finite everywhere. However, these conditions may not always hold in practice. For example, a distribution of digital images is typically discrete and bounded, because the values of pixels are restricted to the range $\{0, 1, \dots, 255\}$. Therefore, $\log p_{\text{data}}(\mathbf{x})$ in this case is discontinuous and is negative infinity outside the range, and thus SM is not directly applicable.

To alleviate this, one can add a bit of noise to each datapoint: $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$. As long as the noise distribution $p(\epsilon)$ is smooth, the resulting noisy data distribution $q(\tilde{\mathbf{x}}) = \int q(\tilde{\mathbf{x}} | \mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}$ is also smooth, and thus the Fisher divergence $D_F(q(\tilde{\mathbf{x}}) \| p_{\theta}(\tilde{\mathbf{x}}))$ is a proper objective. [KL10] showed that the objective with noisy data can be approximated by the noiseless Score Matching objective of Equation (25.31) plus a regularization term; this regularization makes Score Matching applicable to a wider range of data distributions, but still requires expensive second-order derivatives.

[Vin11] proposed an elegant and scalable solution to the above difficulty, by showing that:

$$D_F(q(\tilde{\mathbf{x}}) \| p_{\theta}(\tilde{\mathbf{x}})) = \mathbb{E}_{q(\tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}) - \nabla_{\mathbf{x}} \log p_{\theta}(\tilde{\mathbf{x}})\|_2^2 \right] \quad (25.32)$$

$$= \mathbb{E}_{q(\mathbf{x}, \tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}} | \mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\theta}(\tilde{\mathbf{x}})\|_2^2 \right] + \text{constant}, \quad (25.33)$$

where the expectation is again approximated by the empirical average of samples, thus completely avoiding both the unknown term $p_{\text{data}}(\mathbf{x})$ and computationally expensive second-order derivatives. The constant term does not affect optimization and can be ignored without changing the optimal solution. This estimation method is called **Denoising Score Matching** (DSM) by [Vin11]. Similar formulations were also explored by Raphan and Simoncelli [RS07; RS11] and can be traced back to Tweedie's formula [Efr11] and Stein's Unbiased Risk Estimation [Ste81].

25.3.2.1 Difficulties

The major drawback of adding noise to data arises when $p_{\text{data}}(\mathbf{x})$ is already a well-behaved distribution that satisfies the regularity conditions required by Score Matching. In this case, $D_F(q(\tilde{\mathbf{x}}) \| p_{\theta}(\tilde{\mathbf{x}})) \neq D_F(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x}))$, and DSM is not a consistent objective because the optimal EBM matches the noisy distribution $q(\tilde{\mathbf{x}})$, not $p_{\text{data}}(\mathbf{x})$. This inconsistency becomes non-negligible when $q(\tilde{\mathbf{x}})$ significantly differs from $p_{\text{data}}(\mathbf{x})$.

One way to attenuate the inconsistency of DSM is to choose $q \approx p_{\text{data}}$, *i.e.*, use a small noise perturbation. However, this often significantly increases the variance of objective values and hinders optimization. As an example, suppose $q(\tilde{\mathbf{x}} | \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}} | \mathbf{x}, \sigma^2 I)$ and $\sigma \approx 0$. The corresponding DSM objective is

$$\begin{aligned} D_F(q(\tilde{\mathbf{x}}) \| p_{\theta}(\tilde{\mathbf{x}})) &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, I)} \left[\frac{1}{2} \left\| \frac{\mathbf{z}}{\sigma} + \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x} + \sigma \mathbf{z}) \right\|_2^2 \right] \\ &\simeq \frac{1}{2N} \sum_{i=1}^N \left\| \frac{\mathbf{z}^{(i)}}{\sigma} + \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}^{(i)} + \sigma \mathbf{z}^{(i)}) \right\|_2^2, \end{aligned} \quad (25.34)$$

¹
² where $\{\mathbf{x}^{(i)}\}_{i=1}^N \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$, and $\{\mathbf{z}^{(i)}\}_{i=1}^N \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I})$. When $\sigma \rightarrow 0$, we can leverage Taylor series
³ expansion to rewrite the Monte Carlo estimator in Equation (25.34) to

⁴
⁵
$$\frac{1}{2N} \sum_{i=1}^N \left[\frac{2}{\sigma} (\mathbf{z}^{(i)})^\top \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + \frac{\|\mathbf{z}^{(i)}\|_2^2}{\sigma^2} \right] + \text{constant}. \quad (25.35)$$

⁶

⁷
⁸ When estimating the above expectation with samples, the variances of $(\mathbf{z}^{(i)})^\top \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})/\sigma$ and
⁹ $\|\mathbf{z}^{(i)}\|_2^2/\sigma^2$ will both grow unbounded as $\sigma \rightarrow 0$ due to division by σ and σ^2 . This enlarges the
¹⁰ variance of DSM and makes optimization challenging. Various methods have been proposed to reduce
¹¹ this variance (see e.g., [Wan+20c]).
¹²

¹³ ¹⁴ 25.3.3 Sliced Score Matching (SSM)

¹⁵ By adding noise to data, DSM avoids the expensive computation of second-order derivatives. However,
¹⁶ as mentioned before, the optimal EBM that minimizes the DSM objective corresponds to the
¹⁷ distribution of noise-perturbed data $q(\tilde{\mathbf{x}})$, not the original noise-free data distribution $p_{\text{data}}(\mathbf{x})$. In
¹⁸ other words, DSM does not give a consistent estimator of the data distribution, *i.e.*, one cannot
¹⁹ directly obtain an EBM that exactly matches the data distribution even with unlimited data.
²⁰

²¹ **Sliced Score Matching (SSM)** [Son+19] is one alternative to Denoising Score Matching that is
²² both consistent and computationally efficient. Instead of minimizing the Fisher divergence between
²³ two vector-valued scores, SSM randomly samples a projection vector \mathbf{v} , takes the inner product
²⁴ between \mathbf{v} and the two scores, and then compares the resulting two scalars. More specifically, Sliced
²⁵ Score Matching minimizes the following divergence called the **sliced Fisher divergence**:

²⁶
²⁷
$$D_{SF}(p_{\text{data}}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{v})} \left[\frac{1}{2} (\mathbf{v}^\top \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - \mathbf{v}^\top \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}))^2 \right], \quad (25.36)$$

²⁸

²⁹ where $p(\mathbf{v})$ denotes a projection distribution such that $\mathbb{E}_{p(\mathbf{v})}[\mathbf{v}\mathbf{v}^\top]$ is positive definite. Similar to
³⁰ Fisher divergence, sliced Fisher divergence has an implicit form that does not involve the unknown
³¹ $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$, which is given by
³²

³³
³⁴
$$D_{SF}(p_{\text{data}}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{v})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} v_i \right)^2 + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] + C. \quad (25.37)$$

³⁵
³⁶

³⁷ All expectations in the above objective can be estimated with empirical means, and again the
³⁸ constant term C can be removed without affecting training. The second term involves second-order
³⁹ derivatives of $E_{\boldsymbol{\theta}}(\mathbf{x})$, but contrary to SM, it can be computed efficiently with a cost linear in the
⁴⁰ dimensionality d . This is because
⁴¹

⁴²
⁴³
$$\sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j = \sum_{i=1}^d \frac{\partial}{\partial x_i} \left(\underbrace{\sum_{j=1}^d \frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_j} v_j}_{:= f(\mathbf{x})} \right) v_i, \quad (25.38)$$

⁴⁴
⁴⁵
⁴⁶

where $f(\mathbf{x})$ is the same for different values of i . Therefore, we only need to compute it once with $O(d)$ computation, *plus* another $O(d)$ computation for the outer sum to evaluate Equation (25.38), whereas the original SM objective requires $O(d^2)$ computation.

For many choices of $p(\mathbf{v})$, part of the SSM objective (Equation (25.37)) can be evaluated in closed form, potentially leading to lower variance. For example, when $p(\mathbf{v}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, we have

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{v})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} v_i \right)^2 \right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} \right)^2 \right] \quad (25.39)$$

and as a result,

$$D_{SF}(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} \right)^2 + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\theta}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] + C. \quad (25.40)$$

$$= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{1}{2} (\mathbf{v}^T \mathbf{s}_{\theta}(\mathbf{x}))^2 + \mathbf{v}^T [\mathbf{J}\mathbf{v}] \right] \quad (25.41)$$

where $\mathbf{J} = \mathbf{J}_x \mathbf{s}_{\theta}(\mathbf{x})$. (Note that $\mathbf{J}\mathbf{v}$ can be computed using a Jacobian vector product operation.)

The above objective Equation (25.40) can also be obtained by approximating the sum of second-order gradients in the standard SM objective (Equation (25.31)) with the Hutchinson trace estimator [Ski89; Hut89; Mey+21]. It often (but not always) has lower variance than Equation (25.37), and can perform better in some applications [Son+19].

25.3.4 Connection to Contrastive Divergence

Though Score Matching and Contrastive Divergence (Section 25.2.2) are seemingly very different approaches, they are closely connected to each other. In fact, Score Matching can be viewed as a special instance of Contrastive Divergence in the limit of a particular MCMC sampler [Hyy07a]. Moreover, the Fisher divergence optimized by Score Matching is related to the derivative of KL divergence [Cov99], which is the underlying objective of Contrastive Divergence.

Contrastive Divergence requires sampling from the Energy-Based Model $E_{\theta}(\mathbf{x})$, and one popular method for doing so is Langevin MCMC. Recall from Section 25.2.1 that given any initial data point \mathbf{x}^0 , the Langevin MCMC method executes the following

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \frac{\epsilon}{2} \nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}^k) + \sqrt{\epsilon} \mathbf{z}^k, \quad (25.42)$$

iteratively for $k = 0, 1, \dots, K-1$, where $\mathbf{z}^k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\epsilon > 0$ is the step size.

Suppose we only run one-step Langevin MCMC for Contrastive Divergence. In this case, the gradient of the log-likelihood is given by

$$\begin{aligned} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\theta} \log p_{\theta}(\mathbf{x})] &= -\mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\theta} E_{\theta}(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\theta} E_{\theta}(\mathbf{x})] \\ &\simeq -\mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\theta} E_{\theta}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\nabla_{\theta} E_{\theta} \left(\mathbf{x} - \frac{\epsilon^2}{2} \nabla_{\mathbf{x}} E_{\theta'}(\mathbf{x}) + \epsilon \mathbf{z} \right) \Big|_{\theta'=\theta} \right]. \end{aligned} \quad (25.43)$$

¹ After Taylor series expansion with respect to ϵ followed by some algebraic manipulations, the above
² equation can be transformed to the following (see Hyvarinen [Hyv07a])
³

⁴

$$\frac{\epsilon^2}{2} \nabla_{\theta} D_F(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) + o(\epsilon^2). \quad (25.44)$$

⁵

⁶ When ϵ is sufficiently small, it corresponds to the re-scaled gradient of the Score Matching objective.
⁷ In general, Score Matching minimizes the Fisher divergence $D_F(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x}))$, whereas
⁸ Contrastive Divergence minimizes an objective related to the KL divergence $D_{KL}(p_{\text{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x}))$,
⁹ as shown in Equation (25.20). The above connection of Score Matching and Contrastive Divergence
¹⁰ is a natural consequence of the connection between those two statistical divergences, as characterized
¹¹ by *de Bruijin's identity* [Cov99; Lyu12]:
¹²

¹³

$$\frac{d}{dt} D_{KL}(q_t(\tilde{\mathbf{x}}) \| p_{\theta,t}(\tilde{\mathbf{x}})) = -\frac{1}{2} D_F(q_t(\tilde{\mathbf{x}}) \| p_{\theta,t}(\tilde{\mathbf{x}})).$$

¹⁴

¹⁵ Here $q_t(\tilde{\mathbf{x}})$ and $p_{\theta,t}(\tilde{\mathbf{x}})$ denote smoothed versions of $p_{\text{data}}(\mathbf{x})$ and $p_{\theta}(\mathbf{x})$, resulting from adding
¹⁶ Gaussian noise to \mathbf{x} with variance t ; i.e., $\tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, t\mathbf{I})$.
¹⁷

¹⁸ 25.3.5 Score-Based Generative Models

¹⁹ One typical application of EBMs is creating new samples that are similar to the training data. Towards
²⁰ this end, we can first train an EBM with Score Matching, and then sample from it with MCMC
²¹ approaches. Many efficient sampling methods for EBMs, such as Langevin MCMC, rely on just the
²² score of the EBM (see Equation (25.19)). In addition, Score Matching objectives (Equations (25.31),
²³ (25.33) and (25.37)) depend solely on the scores of EBMs. Therefore, we only need a model for
²⁴ the score when training with Score Matching and sampling with score-based MCMC, and do not
²⁵ have to model the energy explicitly. (Note that this loses the fact that the score is derived from the
²⁶ derivative of the scalar energy, which can be a useful constraint.) By building such a score model, we
²⁷ save the gradient computation of EBMs and can make training and sampling more efficient. These
²⁸ kind of models are named **score-based generative models** [SE19; SE20b; Son+21]. (See also
²⁹ Section 26.4.1 for a discussion of the related approach to denoising diffusion probabilistic models.)
³⁰

³¹ We can optimize the score function $s_{\theta}(\mathbf{x})$ using score matching, sliced score matching, or denoising
³² score matching. Figure 25.3 gives a simple example in 2d. In Figure 25.3a, we show the **swiss roll**
³³ dataset. We estimate the score function by fitting an MLP with 2 hidden layers, each with 128
³⁴ hidden units. In Figure 25.3b, we showed the output of the network after training for 10,000 steps of
³⁵ SGD. We see that there are no major false negatives (since wherever there is high data the gradient
³⁶ field is zero, as it should be), but there are some false positives (since some regions of zero gradient
³⁷ do not correspond to data regions). The comparison of the predicted outputs with the empirical
³⁸ data density is shown more clearly in Figure 25.3c. In Figure 25.3d, we show some samples from the
³⁹ learned model.

⁴⁰

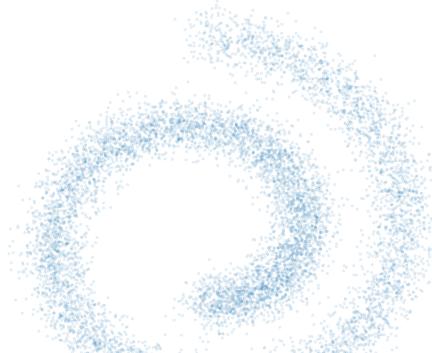
⁴¹ 25.3.5.1 Adding noise at multiple scales

⁴²

⁴³ In general, score matching can have difficulty when there are regions of low data density. To see this,
⁴⁴ suppose $p_{\text{data}}(\mathbf{x}) = \pi p_0(\mathbf{x}) + (1 - \pi)p_1(\mathbf{x})$. Let $\mathcal{S}_0 := \{\mathbf{x} \mid p_0(\mathbf{x}) > 0\}$ and $\mathcal{S}_1 := \{\mathbf{x} \mid p_1(\mathbf{x}) > 0\}$ be
⁴⁵

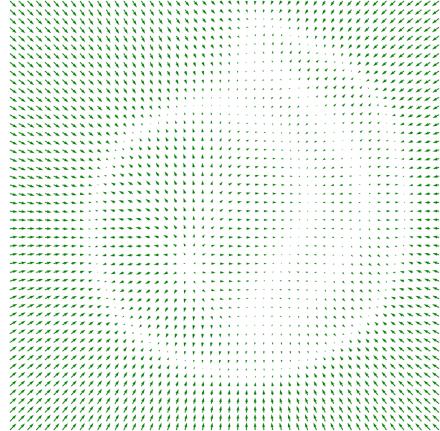
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

(a)



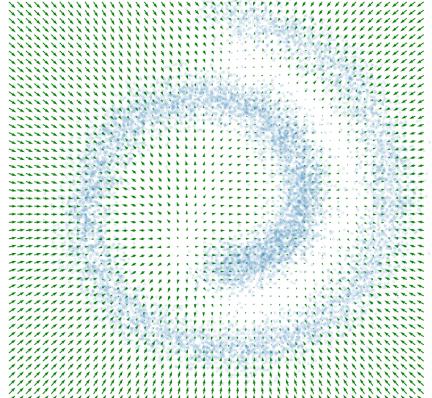
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

(b)



37
38

(c)



39
40
41
42
43
44
45
46
47

(d)



Figure 25.3: Fitting a score-based generative model to the 2d swiss roll dataset. (a) Training set. (b) Learned score function trained using the basic score matching. (c) Superposition of learned score function and empirical density. (d) Langevin sampling applied to the learned model. We show 3 different trajectories, each of length 25. Generated by [score_matching_swiss_roll.ipynb](#).

¹ the supports of $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$ respectively. When they are disjoint from each other, the score of
² $p_{\text{data}}(\mathbf{x})$ is given by
³

⁴

$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \begin{cases} \nabla_{\mathbf{x}} \log p_0(\mathbf{x}), & \mathbf{x} \in \mathcal{S}_0 \\ \nabla_{\mathbf{x}} \log p_1(\mathbf{x}), & \mathbf{x} \in \mathcal{S}_1, \end{cases} \quad (25.45)$$

⁵

⁶ which does not depend on the weight π . Hence score matching cannot correctly recover the true
⁷ distribution. Furthermore, Langevin sampling will have difficulty traversing between modes. (In
⁸ practice this will happen even when the different modes only have approximately disjoint supports.)
⁹ Song and Ermon [SE19; SE20b] and Song et al. [Son+21] overcome this difficulty by perturbing
¹⁰ training data with different scales of noise. Specifically, they use
¹¹

¹²

$$q_{\sigma}(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}|\mathbf{x}, \sigma^2 \mathbf{I}) \quad (25.46)$$

¹³

¹⁴

$$q_{\sigma}(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) q_{\sigma}(\tilde{\mathbf{x}}|\mathbf{x}) d\mathbf{x} \quad (25.47)$$

¹⁵

¹⁶ For a large noise perturbation, different modes are connected due to added noise, and the estimated
¹⁷ weights between them are therefore accurate. For a small noise perturbation, different modes are
¹⁸ more disconnected, but the noise-perturbed distribution is closer to the original unperturbed data
¹⁹ distribution. Using a sampling method such as annealed Langevin dynamics [SE19; SE20b; Son+21]
²⁰ or diffusion sampling [SD+15a; HJA20; Son+21], we can sample from the most noise-perturbed
²¹ distribution first, then smoothly reduce the magnitude of noise scales until reaching the smallest one.
²² This procedure helps combine information from all noise scales, and maintains the correct estimation
²³ of weights from higher noise perturbations when sampling from smaller ones.

²⁴ In practice, all score models share weights and are implemented with a single neural network
²⁵ conditioned on the noise scale; this is called a **Noise Conditional Score Network**, and has the
²⁶ form $s_{\theta}(\mathbf{x}, \sigma)$. Scores of different scales are estimated by training a mixture of Score Matching
²⁷ objectives, one per noise scale. If we use the denoising score matching objective in Equation (25.33),
²⁸ we get

²⁹

$$\mathcal{L}(\boldsymbol{\theta}; \sigma) = \mathbb{E}_{q(\mathbf{x}, \tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_{\theta}(\tilde{\mathbf{x}}, \sigma) - \nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2 \right] \quad (25.48)$$

³⁰

³¹

$$= \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \sigma^2 \mathbf{I})} \left\{ \left\| s_{\theta}(\tilde{\mathbf{x}}, \sigma) + \frac{(\tilde{\mathbf{x}} - \mathbf{x})}{\sigma^2} \right\|_2^2 \right\} \quad (25.49)$$

³²

³³ These losses are combined in a weighted fashion using

³⁴

³⁵

$$\mathcal{L}(\boldsymbol{\theta}; \sigma_{1:L}) = \frac{1}{L} \sum_{i=1}^L \lambda(\sigma_i) \mathcal{L}(\boldsymbol{\theta}; \sigma_i) \quad (25.50)$$

³⁶

³⁷ where we choose $\sigma_1 > \sigma_2 > \dots > \sigma_L$, and the weighting term satisfies $\lambda(\sigma) > 0$.

³⁸ Empirically Song and Ermon [SE19] found that setting $\lambda(\sigma_i) = \sigma_i$ works well. To set the σ_o ,
³⁹ we choose σ_1 small enough such that $p_{\sigma_1}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$, and we choose σ_L large enough that
⁴⁰ $p_{\sigma_L}(\mathbf{x}) \approx \mathcal{N}(\mathbf{x}|\mathbf{0}, \sigma_L^2 \mathbf{I})$. (See [SE20b] for further discussion.)

⁴¹ In [Son+21], this technique is extended to an infinite number of noise levels, by working with
⁴² stochastic differential equations in continuous time. At the time of writing (May 2021), this method
⁴³

is amongst the best generative modeling approaches for high-resolution image generation (see samples in Figure 21.2) and audio generation [Che+20b].

25.4 Noise Contrastive Estimation

Another principle for learning the parameters of EBMs is **Noise Contrastive Estimation** (NCE), introduced by [GH10]. It is based on the idea that we can learn an Energy-Based Model by contrasting it with another distribution with known density.

Let $p_{\text{data}}(\mathbf{x})$ be our data distribution, and let $p_n(\mathbf{x})$ be a chosen distribution with known density, called a noise distribution. This noise distribution is usually simple and has a tractable PDF, like $\mathcal{N}(\mathbf{0}, \mathbf{I})$, such that we can compute the PDF and generate samples from it efficiently. Strategies exist to learn the noise distribution, as referenced below. Furthermore, let y be a binary variable with Bernoulli distribution, which we use to define a mixture distribution of noise and data: $p_{n,\text{data}}(\mathbf{x}) = p(y=0)p_n(\mathbf{x}) + p(y=1)p_{\text{data}}(\mathbf{x})$. According to the Bayes' rule, given a sample \mathbf{x} from this mixture, the posterior probability of $y=0$ is

$$p_{n,\text{data}}(y=0 \mid \mathbf{x}) = \frac{p_{n,\text{data}}(\mathbf{x} \mid y=0)p(y=0)}{p_{n,\text{data}}(\mathbf{x})} = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\text{data}}(\mathbf{x})} \quad (25.51)$$

where $\nu = p(y=1)/p(y=0)$.

Let our Energy-Based Model $p_{\boldsymbol{\theta}}(\mathbf{x})$ be defined as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))/Z_{\boldsymbol{\theta}} \quad (25.52)$$

Contrary to most other EBMs, $Z_{\boldsymbol{\theta}}$ is treated as a learnable (scalar) parameter in NCE. Given this model, similar to the mixture of noise and data above, we can define a mixture of noise and the model distribution: $p_{n,\boldsymbol{\theta}}(\mathbf{x}) = p(y=0)p_n(\mathbf{x}) + p(y=1)p_{\boldsymbol{\theta}}(\mathbf{x})$. The posterior probability of $y=0$ given this noise/model mixture is:

$$p_{n,\boldsymbol{\theta}}(y=0 \mid \mathbf{x}) = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\boldsymbol{\theta}}(\mathbf{x})} \quad (25.53)$$

In NCE, we indirectly fit $p_{\boldsymbol{\theta}}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by fitting $p_{n,\boldsymbol{\theta}}(y \mid \mathbf{x})$ to $p_{n,\text{data}}(y \mid \mathbf{x})$ through a standard conditional maximum likelihood objective:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathbb{E}_{p_{n,\text{data}}(\mathbf{x})}[D_{KL}(p_{n,\text{data}}(y \mid \mathbf{x}) \parallel p_{n,\boldsymbol{\theta}}(y \mid \mathbf{x}))] \quad (25.54)$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathbb{E}_{p_{n,\text{data}}(\mathbf{x}, y)}[\log p_{n,\boldsymbol{\theta}}(y \mid \mathbf{x})], \quad (25.55)$$

which can be solved using stochastic gradient ascent. Just like any other deep classifier, when the model is sufficiently powerful, $p_{n,\boldsymbol{\theta}^*}(y \mid \mathbf{x})$ will match $p_{n,\text{data}}(y \mid \mathbf{x})$ at the optimum. In that case:

$$p_{n,\boldsymbol{\theta}^*}(y=0 \mid \mathbf{x}) \equiv p_{n,\text{data}}(y=0 \mid \mathbf{x}) \quad (25.56)$$

$$\iff \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\boldsymbol{\theta}^*}(\mathbf{x})} = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \nu p_{\text{data}}(\mathbf{x})} \quad (25.57)$$

$$\iff p_{\boldsymbol{\theta}^*}(\mathbf{x}) \equiv p_{\text{data}}(\mathbf{x}) \quad (25.58)$$

¹ Consequently, $E_{\theta^*}(\mathbf{x})$ is an unnormalized energy function that matches the data distribution $p_{\text{data}}(\mathbf{x})$,
² and Z_{θ^*} is the corresponding normalizing constant.
³

⁴ As one unique feature that Contrastive Divergence and Score Matching do not have, NCE provides
⁵ the normalizing constant of an Energy-Based Model as a by-product of its training procedure. When
⁶ the EBM is very expressive, *e.g.*, a deep neural network with many parameters, we can assume
⁷ it is able to approximate a normalized probability density and absorb Z_{θ} into the parameters
⁸ of $E_{\theta}(\mathbf{x})$ [MT12], or equivalently, fixing $Z_{\theta} = 1$. The resulting EBM trained with NCE will be
⁹ self-normalized, *i.e.*, having a normalizing constant close to 1.

¹⁰ In practice, choosing the right noise distribution $p_n(\mathbf{x})$ is critical to the success of NCE, especially
¹¹ for structured and high-dimensional data. As argued in Gutmann and Hirayama [GH12b], NCE
¹² works the best when the noise distribution is close to the data distribution (but not exactly the
¹³ same). Many methods have been proposed to automatically tune the noise distribution, such
¹⁴ as Adversarial Contrastive Estimation [BLC18], Conditional NCE [CG18] and Flow Contrastive
¹⁵ Estimation [Gao+20]. NCE can be further generalized using Bregman divergences (Section 6.5.1),
¹⁶ where the formulation introduced here reduces to a special case.

¹⁷

¹⁸ 25.4.1 Connection to Score Matching

¹⁹ Noise Contrastive Estimation provides a family of objectives that vary for different $p_n(\mathbf{x})$ and ν . This
²⁰ flexibility may allow adaptation to special properties of a task with hand-tuned $p_n(\mathbf{x})$ and ν , and
²¹ may also give a unified perspective for different approaches. In particular, when using an appropriate
²² $p_n(\mathbf{x})$ and a slightly different parameterization of $p_{n,\theta}(y | \mathbf{x})$, we can recover Score Matching from
²³ NCE [GH12b].

²⁴ Specifically, we choose the noise distribution $p_n(\mathbf{x})$ to be a perturbed data distribution: given a
²⁵ small (deterministic) vector \mathbf{v} , let $p_n(\mathbf{x}) = p_{\text{data}}(\mathbf{x} - \mathbf{v})$. It is efficient to sample from this $p_n(\mathbf{x})$, since
²⁶ we can first draw any datapoint $\mathbf{x}' \sim p_{\text{data}}(\mathbf{x}')$ and then compute $\mathbf{x} = \mathbf{x}' + \mathbf{v}$. It is, however, difficult
²⁷ to evaluate the density of $p_n(\mathbf{x})$ because $p_{\text{data}}(\mathbf{x})$ is unknown. Since the original parameterization of
²⁸ $p_{n,\theta}(y | \mathbf{x})$ in NCE (Equation (25.53)) depends on the PDF of $p_n(\mathbf{x})$, we cannot directly apply the
²⁹ standard NCE objective. Instead, we replace $p_n(\mathbf{x})$ with $p_{\theta}(\mathbf{x} - \mathbf{v})$ and parameterize $p_{n,\theta}(y = 0 | \mathbf{x})$
³⁰ with the following form

$$\begin{aligned} \frac{32}{33} \quad p_{n,\theta}(y = 0 | \mathbf{x}) &:= \frac{p_{\theta}(\mathbf{x} - \mathbf{v})}{p_{\theta}(\mathbf{x}) + p_{\theta}(\mathbf{x} - \mathbf{v})} \end{aligned} \tag{25.59}$$

³⁴ In this case, the NCE objective (Equation (25.55)) reduces to:

$$\begin{aligned} \frac{35}{36} \quad \theta^* &= \operatorname{argmin}_{\theta} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\log(1 + \exp(E_{\theta}(\mathbf{x}) - E_{\theta}(\mathbf{x} - \mathbf{v})) + \log(1 + \exp(E_{\theta}(\mathbf{x}) - E_{\theta}(\mathbf{x} + \mathbf{v})))] \\ \frac{37}{38} \quad & \end{aligned} \tag{25.60}$$

³⁹ At θ^* , we have a solution where:

$$\begin{aligned} \frac{40}{41} \quad p_{n,\theta^*}(y = 0 | \mathbf{x}) &\equiv p_{n,\text{data}}(y = 0 | \mathbf{x}) \end{aligned} \tag{25.61}$$

$$\begin{aligned} \frac{42}{43} \quad \Rightarrow \quad \frac{p_{\theta^*}(\mathbf{x} - \mathbf{v})}{p_{\theta^*}(\mathbf{x}) + p_{\theta^*}(\mathbf{x} - \mathbf{v})} &\equiv \frac{p_{\text{data}}(\mathbf{x} - \mathbf{v})}{p_{\text{data}}(\mathbf{x}) + p_{\text{data}}(\mathbf{x} - \mathbf{v})} \end{aligned} \tag{25.62}$$

⁴⁴ which implies that $p_{\theta^*}(\mathbf{x}) \equiv p_{\text{data}}(\mathbf{x})$, *i.e.*, our model matches the data distribution.

⁴⁵

As noted in Gutmann and Hirayama [GH12b] and Song et al. [Son+19], when $\|\mathbf{v}\|_2 \approx 0$, the NCE objective Equation (25.55) has the following equivalent form by Taylor expansion

$$\operatorname{argmin}_{\boldsymbol{\theta}} \frac{1}{4} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^d \left(\frac{\partial E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i} v_i \right)^2 + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2 E_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] + 2 \log 2 + o(\|\mathbf{v}\|_2^2). \quad (25.63)$$

Comparing against Equation (25.37), we immediately see that the above objective equals that of SSM, if we ignore small additional terms hidden in $o(\|\mathbf{v}\|_2^2)$ and take the expectation with respect to \mathbf{v} over a user-specified distribution $p(\mathbf{v})$.

25.5 Other Methods

Aside from MCMC-based training, Score Matching and Noise Contrastive Estimation, there are also other methods for learning EBMs. Below we briefly survey some examples of them. Interested readers can learn more details from references therein.

25.5.1 Minimizing Differences/Derivatives of KL Divergences

The overarching strategy for learning probabilistic models from data is to minimize the KL divergence between data and model distributions. However, because the normalizing constants of EBMs are typically intractable, it is hard to directly evaluate the KL divergence when the model is an EBM (see the discussion in Section 25.2.1). One generic idea that has frequently circumvented this difficulty is to consider differences/derivatives of KL divergences. It turns out that the unknown partition functions of EBMs are often cancelled out after taking the difference of two closely related KL divergences, or computing the derivatives.

Typical examples of this strategy include minimum velocity learning [Mov08; Wan+20c], minimum probability flow [SDBD11] and minimum KL contraction [Lyu11], to name a few. In minimum velocity learning and minimum probability flow, a Markov chain is designed such that it starts from the data distribution $p_{\text{data}}(\mathbf{x})$ and converges to the EBM distribution $p_{\boldsymbol{\theta}}(\mathbf{x}) = e^{-E(\mathbf{x})}/Z_{\boldsymbol{\theta}}$. Specifically, the Markov chain satisfies $p_0(\mathbf{x}) \equiv p_{\text{data}}(\mathbf{x})$ and $p_{\infty}(\mathbf{x}) \equiv p_{\boldsymbol{\theta}}(\mathbf{x})$, where we denote by $p_t(\mathbf{x})$ the state distribution at time $t \geq 0$.

This Markov chain will evolve towards $p_{\boldsymbol{\theta}}(\mathbf{x})$ unless $p_{\text{data}}(\mathbf{x}) \equiv p_{\boldsymbol{\theta}}(\mathbf{x})$. Therefore, we can fit the EBM distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ to $p_{\text{data}}(\mathbf{x})$ by minimizing the modulus of the “velocity” of this evolution, defined by

$$\frac{d}{dt} \mathbb{KL}(p_t(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) \Big|_{t=0} \quad \text{or} \quad \frac{d}{dt} \mathbb{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_t(\mathbf{x})) \Big|_{t=0} \quad (25.64)$$

in minimum velocity learning and minimum probability flow respectively. These objectives typically do not require computing the normalizing constant $Z_{\boldsymbol{\theta}}$.

In minimum KL contraction [Lyu11], a distribution transformation Φ is chosen such that $\mathbb{KL}(p(\mathbf{x}) \parallel q(\mathbf{x})) \geq \mathbb{KL}(\Phi\{p(\mathbf{x})\} \parallel \Phi\{q(\mathbf{x})\})$, with equality if and only if $p(\mathbf{x}) = q(\mathbf{x})$. We can leverage this Φ to train an EBM, by minimizing

$$\mathbb{KL}(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})) - \mathbb{KL}(\Phi\{p_{\text{data}}(\mathbf{x})\} \parallel \Phi\{p_{\boldsymbol{\theta}}(\mathbf{x})\}). \quad (25.65)$$

¹ This objective does not require computing the partition function Z_θ whenever Φ is linear.

² Minimum velocity learning, minimum probability flow, and minimum KL contraction can all be
³ viewed as generalizations to Score Matching and Noise Contrastive Estimation [Mov08; SDBD11;
⁴ Lyu11].

⁷ 25.5.2 Minimizing the Stein Discrepancy

⁸ We can train EBMs by minimizing the Stein discrepancy, defined by

$$\underline{11} \quad D_{\text{Stein}}(p_{\text{data}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) := \sup_{\mathbf{f} \in \mathcal{F}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) + \text{trace}(\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}))], \quad (25.66)$$

¹⁴ where \mathcal{F} is a family of vector-valued functions, and $\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x})$ denotes the Jacobian of $\mathbf{f}(\mathbf{x})$. With
¹⁵ some regularity conditions [GM15; LLJ16], we have $D_S(p_{\text{data}}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) \geq 0$, where the equality
¹⁶ holds if and only if $p_{\text{data}}(\mathbf{x}) \equiv p_\theta(\mathbf{x})$. Similar to Score Matching (Equation (25.31)), the objective
¹⁷ Equation (25.66) only involves the score function of $p_\theta(\mathbf{x})$, and does not require computing the EBM's
¹⁸ partition function. Still, the trace term in Equation (25.66) may demand expensive computation,
¹⁹ and does not scale well to high dimensional data.

²⁰ There are two common methods that sidestep this difficulty. Gorham and Mackey [GM15] and
²¹ Liu, Lee, and Jordan [LLJ16] discovered that when \mathcal{F} is a unit ball in a Reproducing Kernel Hilbert
²² Space (RKHS) with a fixed kernel, the Stein discrepancy becomes kernelized Stein discrepancy, where
²³ the trace term is a constant and does not affect optimization. Otherwise, $\text{trace}(\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}))$ can be
²⁴ approximated with the Skilling-Hutchinson trace estimator [Ski89; Hut89; Gra+20c].

²⁶ 25.5.3 Adversarial Training

²⁸ Recall from Section 25.2.1 that when training EBMs with maximum likelihood estimation (MLE),
²⁹ we need to sample from the EBM per training iteration. However, sampling using multiple MCMC
³⁰ steps is expensive and requires careful tuning of the Markov chain. One way to avoid this difficulty is
³¹ to use non-MLE methods that do not need sampling, such as Score Matching and Noise Contrastive
³² Estimation. Here we introduce another family of methods that sidestep costly MCMC sampling by
³³ learning an auxiliary model through adversarial training, which allows fast sampling.

³⁴ Using the definition of EBMs, we can rewrite the maximum likelihood objective by introducing a
³⁵ variational distribution $q_\phi(\mathbf{x})$ parameterized by ϕ :

$$\begin{aligned} \underline{37} \quad \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\log p_\theta(\mathbf{x})] &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \log Z_\theta \\ \underline{38} \quad &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \log \int e^{-E(\mathbf{x})} d\mathbf{x} \\ \underline{39} \quad &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \log \int q_\phi(\mathbf{x}) \frac{e^{-E(\mathbf{x})}}{q_\phi(\mathbf{x})} d\mathbf{x} \\ \underline{40} \quad &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \int q_\phi(\mathbf{x}) \log \frac{e^{-E(\mathbf{x})}}{q_\phi(\mathbf{x})} d\mathbf{x} \\ \underline{41} \quad &\stackrel{(i)}{\leq} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \int q_\phi(\mathbf{x}) \log \frac{e^{-E(\mathbf{x})}}{q_\phi(\mathbf{x})} d\mathbf{x} \\ \underline{42} \quad &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} [-E_\theta(\mathbf{x})] - \mathbb{E}_{q(\mathbf{x})} [-E_\theta(\mathbf{x})] - H(q_\phi(\mathbf{x})), \end{aligned} \quad (25.67)$$

where $H(q_\phi(\mathbf{x}))$ denotes the entropy of $q_\phi(\mathbf{x})$. Step (i) is due to Jensen’s inequality. Equation (25.67) provides an upper bound to the expected log-likelihood. For EBM training, we can first minimize the upper bound Equation (25.67) with respect to $q_\phi(\mathbf{x})$ so that it is closer to the likelihood objective, and then maximize Equation (25.67) with respect to $E_\theta(\mathbf{x})$ as a surrogate for maximizing likelihood. This amounts to using the following maximin objective

$$\max_{\theta} \min_{\phi} \mathbb{E}_{q_\phi(\mathbf{x})}[E_\theta(\mathbf{x})] - \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[E_\theta(\mathbf{x})] - H(q_\phi(\mathbf{x})). \quad (25.68)$$

Optimizing the above objective is similar to training Generative Adversarial Networks (GANs) [Goo+14a] and can be achieved by adversarial training. The variational distribution $q_\phi(\mathbf{x})$ should allow both fast sampling and efficient entropy evaluation to make Equation (25.68) tractable. This limits the model family of $q_\phi(\mathbf{x})$, and usually restricts our choice to invertible probabilistic models, such as inverse autoregressive flow (Section 24.2.4.3). See Dai et al. [Dai+19b] for an example on designing $q_\phi(\mathbf{x})$ and training EBMs with Equation (25.68).

Kim and Bengio [KB16] and Zhai et al. [Zha+16] propose to represent $q_\phi(\mathbf{x})$ with neural samplers, like the generator of GANs. A neural sampler is a deterministic mapping g_ϕ that maps a random Gaussian noise $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ directly to a sample $\mathbf{x} = g_\phi(\mathbf{z})$. When using a neural sampler as $q_\phi(\mathbf{x})$, it is efficient to draw samples through the deterministic mapping, but $H(q_\phi(\mathbf{x}))$ is intractable since the density of $q_\phi(\mathbf{x})$ is unknown. Kim and Bengio [KB16] and Zhai et al. [Zha+16] propose several heuristics to approximate this entropy function. Kumar et al. [Kum+19c] propose to estimate the entropy through its connection to mutual information: $H(q_\phi(\mathbf{z})) = I(g_\phi(\mathbf{z}), \mathbf{z})$, which can be estimated from samples with variational lower bounds [NWJ10b; NCT16b]. Dai et al. [Dai+19a] noticed that when defining $p_\theta(\mathbf{x}) = p_0(\mathbf{x})e^{-E(\mathbf{x})}/Z_\theta$, with $p_0(\mathbf{x})$ being a fixed base distribution, the entropy term $-H(q_\phi(\mathbf{x}))$ in Equation (25.68) equates $\text{KL}(q_\phi(\mathbf{x}) \parallel p_0(\mathbf{x}))$, which can also be approximated with variational lower bounds using samples from $q_\phi(\mathbf{x})$ and $p_0(\mathbf{x})$, without requiring the density of $q_\phi(\mathbf{x})$.

Grathwohl et al. [Gra+20a] represent $q_\phi(\mathbf{x})$ as a noisy neural sampler, where samples are obtained via $g_\phi(\mathbf{z}) + \sigma\epsilon$, assuming $\mathbf{z}, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. With a noisy neural sampler, $\nabla_\phi H(q_\phi(\mathbf{x}))$ becomes particularly easy to estimate, which allows gradient-based optimization for the minimax objective in Equation (25.67). A related approach is proposed in Xie et al. [Xie+18], where authors train a noisy neural sampler with samples obtained from MCMC, and initialize new MCMC chains with samples generated from the neural sampler. This cooperative sampling scheme improves the convergence of MCMC, but may still require multiple MCMC steps for sample generation. It does not optimize the objective in Equation (25.67).

When using both adversarial training and MCMC sampling, Yu et al. [Yu+20] noticed that EBMs can be trained with an arbitrary f -divergence, including KL, reverse KL, total variation, Hellinger, etc.. The method proposed by Yu et al. [Yu+20] allows us to explore the trade-offs and inductive bias of different statistical divergences for more flexible EBM training.

26 Denoising diffusion models

In this section, we consider a class of models called **denoising diffusion probabilistic models** or **DDPM**. This model was first introduced in [SD+15b] and has been extended in several subsequent papers (e.g., [HJA20; Son+21; DN21]).

The basic insight is the following: it can be hard to convert noise into structured data, but it is easy to convert structured data into noise. In particular, we can use a **forwards process** or **diffusion process** to gradually convert the observed data $\mathbf{x} = \mathbf{x}_0$ into a noisy version $\mathbf{z} = \mathbf{x}_T$ with no structure by passing the data through T steps of a stochastic encoder $q(\mathbf{x}_t | \mathbf{x}_{t-1})$. We then learn a **reverse process** to undo this, by passing the noise through T steps of a decoder $p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)$ until we generate \mathbf{x}_0 . See Figure 26.1 for an overall sketch of the approach, and Section 26.1 for the details. (Note that diffusion models is a very active field of research, so this section is just a brief summary. For some JAX tutorials, see https://github.com/acids-ircam/diffusion_models.)

26.1 Model definition

In this section, we describe DDPMs in more detail. The diffusion process is designed to add a small amount of noise to the data at each step. For real-valued data, we can use

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t | \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}) \quad (26.1)$$

where β_t is a small positive term that controls the variance of the noise. We can sample from this by first sampling $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and then setting

$$\mathbf{x}_t = \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \beta_t \boldsymbol{\epsilon}_t \quad (26.2)$$

By first scaling down \mathbf{x}_{t-1} before adding noise, the overall variance of each intermediate version of the data remains bounded. (We assume the observed data is already standardized.)

For binary data, we can use the following “noisy channel” model:

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \text{Ber}(\mathbf{x}_t | (1 - \beta_t) \mathbf{x}_{t-1} + 0.5 \beta_t) \quad (26.3)$$

where we use the following shorthand for a factored Bernoulli distribution:

$$\text{Ber}(\mathbf{x} | \boldsymbol{\mu}) = \prod_{d=1}^D \text{Ber}(x_d | \mu_d) \quad (26.4)$$

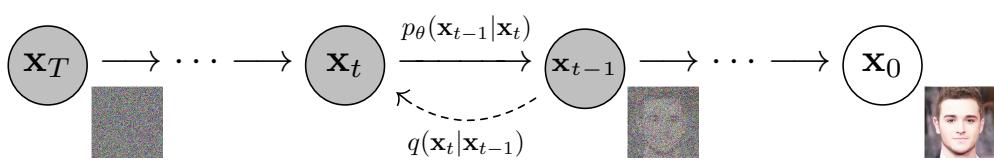


Figure 26.1: Denoising diffusion probabilistic model. The forwards diffusion process, $q(\mathbf{x}_t|\mathbf{x}_{t-1})$, implements the (non-learned) inference network; this just adds Gaussian noise at each step. The reverse diffusion process, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, implements the decoder; this is a learned Gaussian model. From Figure 1 of [HJA20]. Used with kind permission of Jonathan Ho.

If we keep sampling from the diffusion process, we eventually get to the stationary distribution, defined by

$$\pi(\mathbf{x}) = \int d\mathbf{x}' q(\mathbf{x}|\mathbf{x}') \quad (26.5)$$

We usually choose $\pi(\mathbf{x})$ to be a simple maximum entropy distribution, such as an isotropic Gaussian (for real data), or product of uniform Bernoullis (for binary data). We choose the noise schedule $\beta_{1:T}$ so that $q(\mathbf{x}_T|\mathbf{x}_0) \approx \pi(\mathbf{x}_T)$. That is, after T steps, we have “destroyed” all the information in the data. The overall encoder can be represented as follows:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad (26.6)$$

To generate from this model, we first sample a latent noise vector $\mathbf{x}_T \sim \pi(\mathbf{x}_T)$, and then gradually “denoise” it by sampling from the reverse process, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, until we get the final output \mathbf{x}_0 . (Note that each latent sample has the same size as the observed data, so there is no dimensionality reduction.)

The reverse conditionals $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are designed to “undo” the effects of noise added by the diffusion process. For small enough β_t , one can show that the reverse conditionals will have the same form as the forward conditionals. Thus for real-valued data we can use

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}|f_\mu(\mathbf{x}_t, t; \boldsymbol{\theta}), f_\Sigma(\mathbf{x}_t, t; \boldsymbol{\theta})) \quad (26.7)$$

and for binary data, we can use

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \text{Ber}(\mathbf{x}_{t-1}|f(\mathbf{x}_t, t; \boldsymbol{\theta})) \quad (26.8)$$

The overall decoder can be represented as follows:

$$p_\theta(\mathbf{x}_{0:T}) = \pi(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad (26.9)$$

The marginal probability of the observed data (the output of the model) is given by

$$p_\theta(\mathbf{x}_0) = \int d\mathbf{x}_{1:T} p(\mathbf{x}_{0:T}) \quad (26.10)$$

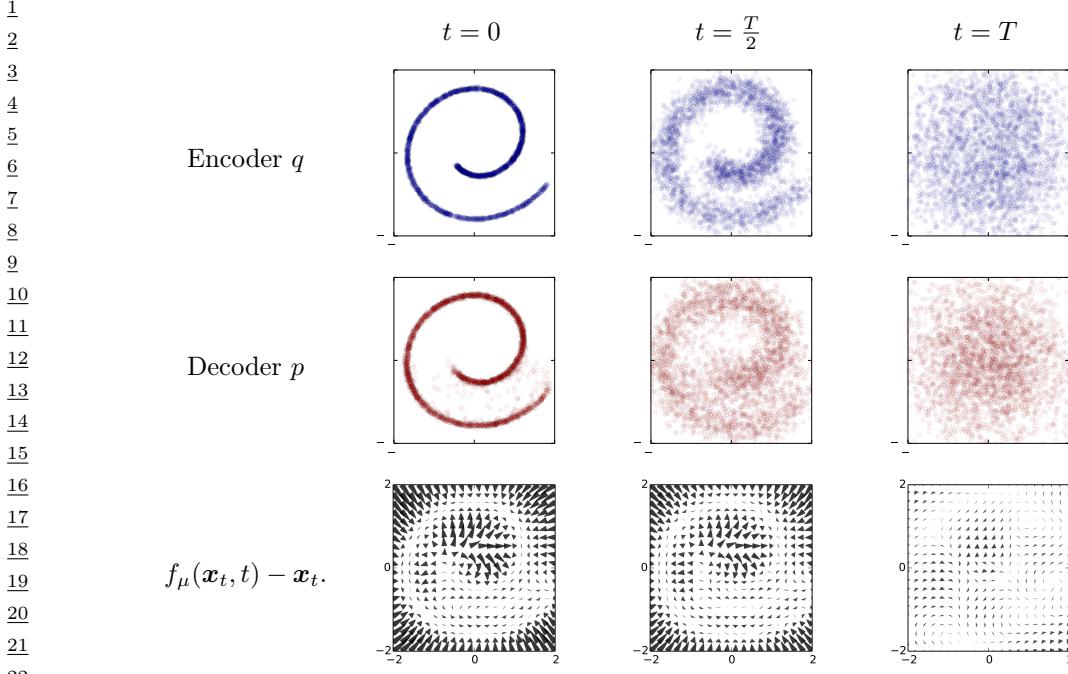


Figure 26.2: Diffusion on 2d swiss roll. Top row: samples from the encoder $q(\mathbf{x}_{0:T})$ at 3 different time slices. (Here $q(\mathbf{x}_0)$ is the input data.) Middle row: samples from the generator $p_{\theta}(\mathbf{x}_{0:T})$ at 3 different time slices. Bottom row: Visualization of the residual or drift term, $f_{\mu}(\mathbf{x}_t, t) - \mathbf{x}_t$. We see that this resembles the score function, in that it points towards regions of high data density (see Section 26.4.1). From Figure 1 of [SD+15b]. Used with kind permission of Jascha Sohl-Dickstein.

This is intractable to compute. However, we can use a method similar to annealed importance sampling (Section 11.5.4) to rewrite this as follows:

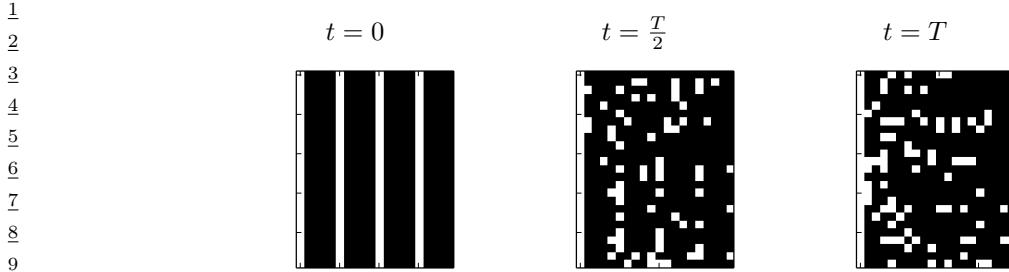
$$p(\mathbf{x}_0) = \int d\mathbf{x}_{1:T} p_{\theta}(\mathbf{x}_{0:T}) \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} = \int d\mathbf{x}_{1:T} q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p_{\theta}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \quad (26.11)$$

$$= \int d\mathbf{x}_{1:T} q(\mathbf{x}_{1:T}|\mathbf{x}_0) \pi(\mathbf{x}_T) \prod_{t=1}^T \frac{p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \quad (26.12)$$

which we can approximate by Monte Carlo sampling. We will leverage this result below.

26.2 Examples

In Figure 26.2, we show an example of a Gaussian DDPM fit to the 2d swiss roll dataset. The model uses $T = 40$ steps. The generator network $f(\mathbf{x}_t, t; \boldsymbol{\theta})$ was defined to be a radial basis function network with 16 hidden units. It has two output heads, one for $\boldsymbol{\mu}$, and one for the diagonal $\boldsymbol{\Sigma}$. (The latter uses the sigmoid nonlinearity to ensure the variances are between 0 and 1.) A separate output head



12 *Figure 26.3: Diffusion on binary data. Each row contains an independent sample from $p(\mathbf{x}_{0:T})$ at 3 different
13 time steps. From Figure 2 of [SD+15b]. Used with kind permission of Jascha Sohl-Dickstein.*

14

15

16 was learned for each time step t , but all the other parameters were shared across heads and across
17 time.

18 In Figure 26.3, we show an example of a binary DDPM fit to a synthetic distribution consisting of
19 bit vectors of length 20, in which every 5th time step is on. The model uses $T = 2000$ time steps and
20 a prior of the form $\pi(\mathbf{x}_T) = \text{Ber}(\mathbf{x}_T|0.2)$. The generator network $f(\mathbf{x}_t, t; \theta)$ was defined to be an
21 MLP with sigmoid nonlinearities, 20 input units and 3 hidden layers, each with 50 units. The final
22 sigmoid later was learned separately for each time step t , but all the other parameters were shared
23 across time.

24 In more recent papers (e.g., [HJA20; Son+21; DN21]), DDPMs have been scaled up to much more
25 complex image datasets, such as CIFAR, CelebA and Imagenet. The resulting generated images have
26 a visual realism which matches or exceeds other methods, such as GANs (Chapter 27). In addition,
27 the samples are more diverse, and the log likelihoods are higher. These results rely on more complex
28 neural network architectures, borrowing pieces from U-net, PixelCNN, transformers, etc.

29

30 26.3 Model training

31

32 To fit the generator p , we minimize the cross entropy between the empirical distribution $p_{\mathcal{D}}$ and the
33 model p , or equivalently we maximize

34

$$\begin{aligned} \text{35} \quad L &= \int d\mathbf{x}_0 p_{\mathcal{D}}(\mathbf{x}_0) \log p_{\theta}(\mathbf{x}_0) \end{aligned} \tag{26.13}$$

37 To simplify notation, let us define $p_{\mathcal{D}}(\mathbf{x}) = q(\mathbf{x}_0)$. Then we have

38

$$\begin{aligned} \text{39} \quad L &= \int d\mathbf{x}_{0:T} q(\mathbf{x}_0) q(\mathbf{x}_{1:T}|\mathbf{x}_0) \pi(\mathbf{x}_T) \prod_{t=1}^T \frac{p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \end{aligned} \tag{26.14}$$

40

41 By Jensen's inequality we find that $L \geq \mathbb{L}$, where \mathbb{L} is the ELBO:

42

$$\begin{aligned} \text{43} \quad \mathbb{L} &= \mathbb{E}_q \left[\log \pi(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \end{aligned} \tag{26.15}$$

44

We now proceed to rewrite the lower bound in Equation (26.15) in a simplified form that is more amenable to optimization. First note that

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) \quad (26.16)$$

since the diffusion process is Markov. Hence by Bayes rule, we have

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)q(\mathbf{x}_t | \mathbf{x}_0)}{q(\mathbf{x}_{t-1} | \mathbf{x}_0)} \quad (26.17)$$

Thus

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \pi(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} + \log \frac{p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \quad (26.18)$$

$$= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \pi(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} + \underbrace{\sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)}}_{*} + \log \frac{p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \quad (26.19)$$

The term marked * is a telescoping sum, and can be simplified as follows:

$$* = \log q(\mathbf{x}_{T-1} | \mathbf{x}_0) + \log q(\mathbf{x}_{T-2} | \mathbf{x}_0) + \dots + \log q(\mathbf{x}_1 | \mathbf{x}_0) \quad (26.20)$$

$$- \log q(\mathbf{x}_T | \mathbf{x}_0) - \log q(\mathbf{x}_{T-1} | \mathbf{x}_0) - \dots - \log q(\mathbf{x}_2 | \mathbf{x}_0) \quad (26.21)$$

$$= \log q(\mathbf{x}_1 | \mathbf{x}_0) - \log q(\mathbf{x}_T | \mathbf{x}_0) \quad (26.22)$$

Hence

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \frac{\pi(\mathbf{x}_T)}{q(\mathbf{x}_T | \mathbf{x}_0)} + \sum_{t=2}^T \log \frac{p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} + \log p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1) \right] \quad (26.23)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T | \mathbf{x}_0) \| \pi(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{L_{t-1}} - \underbrace{\log p_{\theta}(\mathbf{x}_0 | \mathbf{x}_1)}_{L_0} \right] \quad (26.24)$$

The L_T term is a constant, since q has no free parameters. The L_0 term can be optimized using the reparameterization trick. Finally, each L_t term can be computed analytically, as we discuss below. Furthermore, we can compute a stochastic approximation to the overall objective terms by sampling a single timestep at random, and just optimizing a single L_t term. This allows us to efficiently train very deep (large T) models.

To compute the L_t terms, we will focus on the Gaussian case. First note that $q(\mathbf{x}_t | \mathbf{x}_0)$ corresponds to t steps through a linear-Gaussian Markov chain, and so the result is a Gaussian. To derive its moments, let us define $\alpha_t = 1 - \beta$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$. Then we have

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t | \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}) \quad (26.25)$$

- 1 To compute $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, we use Bayes rule for Gaussians to get another Gaussian:
- 2
- $$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \propto q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_0) \quad (26.26)$$
- 3
- $$= \mathcal{N}(\mathbf{x}_{t-1}|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}) \quad (26.27)$$
- 4
- $$\tilde{\mu}_t = \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad (26.28)$$
- 5
- $$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \quad (26.29)$$
- 6
- 7
- 8
- 9
- 10

11 Finally, recall from Equation (5.66) that the KL divergence between two Gaussian is

12

$$\begin{aligned} D_{\text{KL}}(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\|\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) \\ = \frac{1}{2} \left[\text{tr}(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - D + \log \left(\frac{\det(\boldsymbol{\Sigma}_2)}{\det(\boldsymbol{\Sigma}_1)} \right) \right] \end{aligned} \quad (26.30)$$

13

14

15

16

17 Thus every L_t terms can be tractably computed.

18 To finish specifying the model, we have to define the noise schedule for β_t in the encoder q . It is
19 possible to learn β_t , but it is more common to set it using a deterministic schedule. In [SD+15b],
20 they define $\beta_t = 1/(T-t+1)$, which erases a fraction $1/T$ of the information per step. In [HJA20],
21 they set β_t to a constant.

22

23 26.4 Connections with other generative models

24

25 DDPMs are closely related to several other model types, as summarized in Figure 21.1. We summarize
26 these connections below.

27

28 26.4.1 Connection with score matching

29

30 Suppose we use a Gaussian DDPM model of the form $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}|\mu_\theta(\mathbf{x}_t, t), \sigma^2 \mathbf{I})$. In this
31 case, there is a very close connection between DDPM and score matching (Section 25.3.5).

32 To see this, note that we can rewrite the L_{t-1} term in the ELBO in Equation (26.24) as follows:

33

$$L_{t-1} = \mathbb{E}_{q(\mathbf{x}_0, \mathbf{x}_t)} \left[\frac{1}{2\sigma_t^2} \|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2 \right] + C \quad (26.31)$$

34

35

36

37 where C is a constant that does not depend on θ , and $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$ is the predicted mean value of \mathbf{x}_{t-1} .
38 Since $\mathbf{x}_t = \mathbf{x}_{t-1} + \sigma^2 \epsilon_t$, the network $\mu_\theta(\mathbf{x}_t, t)$ needs to predict the noise term $\epsilon_t = \frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\sigma^2}$ if it is to
39 reliably predict \mathbf{x}_{t-1} . But predicting the noise term is essentially what a denoising score matching
40 method does (see Section 25.3.2 for details). In particular, if we use a noise distribution of the form
41 $p_\sigma(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\mathbf{x}_{t-1}, \sigma^2 \mathbf{I})$, then the score function is given by

42

$$\nabla_{\mathbf{x}_t} \log p_\sigma(\mathbf{x}_t|\mathbf{x}_{t-1}) = -\frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\sigma^2} \quad (26.32)$$

43

44

45 Thus we see that score networks and DDPM models are trained to predict the same targets. (For a
46 more detailed discussion of the similarities and differences between these two models, see [Son+21].)
47

1
2

26.4.2 Connection with VAEs

3
4
5
6
7
8
9
10
11
12
13
14

There is a close connection between DDPMs and variational autoencoders (Chapter 22). In particular, we can view a DDPM as a degenerate VAE with many latent Gaussian layers, all of which have the same size as the input, and where the encoder q has no free parameters, but just adds a small amount of Gaussian noise at each step. The decoder (generator) learns to map the noise back to data. The main advantage of DDPMs compared to other VAEs is that the model is easier to train. In particular, we will see that we can sample a layer at random, and then optimize a least squares loss for that layer. Furthermore, we do not need to worry about posterior collapse (Section 22.4). The disadvantage is that there is no dimensionality reduction, so there is no compact latent representation (although there is some work on combining VAEs and DDPMs to create a low dimensional encoding, see e.g., [Pan+22]). Furthermore, sampling can be slow, since the model often needs many time steps of the reverse diffusion process (although there is recent work to speed things up, see e.g., [SH22]).

15
16

26.4.3 Connection with flow models

17
18
19
20
21

DDPMs are also similar in spirit to normalizing flow models (Chapter 24), since they transform noise into data vectors with the same size through a series of small steps. The main advantage of DDPMs compared to flows is that the architecture is unconstrained, and does not need to be invertible. The main disadvantage is that we can only compute a lower bound on the likelihood, rather than the exact likelihood.

22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

27 Generative adversarial networks

This chapter is written by Mihaela Rosca, Shakir Mohamed and Balaji Lakshminarayanan.

27.1 Introduction

In this chapter, we focus on **implicit generative models**, which are a kind of probabilistic model without an explicit likelihood function [ML16]. This includes the family of **Generative Adversarial Networks** or **GANs** [Goo16]. In this chapter, we provide an introduction to this topic, focusing on a probabilistic perspective.

To develop a probabilistic formulation for GANs, it is useful to first distinguish between two types of probabilistic models: “**prescribed probabilistic models**” and “**implicit probabilistic models**” [DG84]. Prescribed probabilistic models, which we will call **explicit probabilistic models**, provide an explicit parametric specification of the distribution of an observed random variable \mathbf{x} , specifying a log-likelihood function $\log q_\theta(\mathbf{x})$ with parameters θ . Most models we encountered in this book thus far are of this form, whether they be state-of-the-art classifiers, large-vocabulary sequence models, or fine-grained spatio-temporal models. Alternatively, we can specify an **implicit probabilistic model** that defines a stochastic procedure to directly generate data. Such models are the natural approach for problems in climate and weather, population genetics, and ecology, since the mechanistic understanding of such systems can be used to directly describe the generative model [citations](#). We illustrate the difference between implicit and explicit models in Figure 27.1.

The form of implicit generative models we focus on in this chapter can be expressed as a probabilistic latent variable model, similar to VAEs (Chapter 22). Implicit generative models use a latent variable \mathbf{z} and transform it using a deterministic function G_θ that maps from $\mathbb{R}^m \rightarrow \mathbb{R}^d$ using parameters θ . Implicit generative models do not include a likelihood function or observation model. Instead, the generating procedure defines a valid density on the output space that forms an effective likelihood function:

$$\mathbf{x} = G_\theta(\mathbf{z}'); \quad \mathbf{z}' \sim q(\mathbf{z}) \tag{27.1}$$

$$q_\theta(\mathbf{x}) = \frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_d} \int_{\{G(\mathbf{z}) \leq \mathbf{x}\}} q(\mathbf{z}) d\mathbf{z}, \tag{27.2}$$

where $q(\mathbf{z})$ is a distribution over latent variables that provides the external source of randomness. Equation (27.2) is the definition of the transformed density $q_\theta(\mathbf{x})$ defined as the derivative of a cumulative distribution function, and hence integrates the distribution $q(\mathbf{z})$ over all events defined

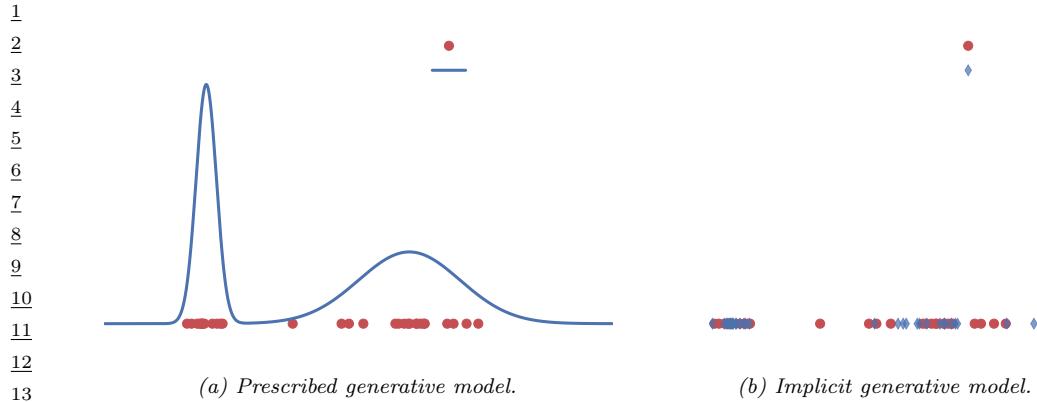


Figure 27.1: Visualizing the difference between prescribed and implicit generative models. Prescribed models provide direct access to the learned density (sometimes unnormalized). Implicit models only provide access to a simulator which can be used to generate samples from an implied density. Generated by genmo_types_implicit_explicit.ipynb

by the set $\{G_{\theta}(\mathbf{z}) \leq \mathbf{x}\}$. When the latent and data dimension are equal ($m = d$) and the function function $G_{\theta}(\mathbf{z})$ is invertible or has easily characterized roots, we recover the rule for transformations of probability distributions. This transformation of variables property is also used in normalizing flows (Chapter 24). In diffusion models (Chapter 26), we also transform noise into data and vice versa, but the transformation is not strictly invertible.

We can develop more general and flexible implicit generative models where the function G is a non-linear function with $d > m$, e.g., specified using by deep networks. When using deep networks, implicit generative models have also been referred to as generator networks, generative neural samplers, or as (differentiable) simulator-models. The integral (27.2) is intractable in the general case: we will be unable to determine the set $\{G_{\theta}(\mathbf{z}) \leq \mathbf{x}\}$, the integral is often unknown and the derivative is high-dimensional and difficult to compute. As we covered in other chapters on deep latent variable models (Chapter 22), intractability is also a challenge for explicit latent variable models, but the lack of a likelihood term significantly reduces the approaches available for learning, and learning methods are often referred to as **likelihood-free inference** or **simulation-based inference**.

Likelihood-free inference also forms the basis of the field known as **Approximate Bayesian Computation** or **ABC**, which we briefly discuss in Section 13.6.5. ABC and GANs give us two different algorithmic frameworks for learning in implicit generative models. Both approaches rely on a learning principle based on *comparing real and simulated data*. This type of learning by comparison instantiates a core principle of likelihood-free inference, and expanding on this idea is the focus of the next section. The subsequent sections will then focus on GANs specifically, to develop a more detailed foundation and practical considerations.

27.2 Learning by Comparison

In most of this book, we rely on the principle of maximum likelihood for learning. By maximizing the likelihood we effectively minimize the KL divergence between the model q_{θ} (with parameters θ)



Figure 27.2: The aim of implicit generative modelling objectives: to measure distances between distributions only from samples, in order to distinguish between distributions which are further apart (left) compared to those which are closer (right).

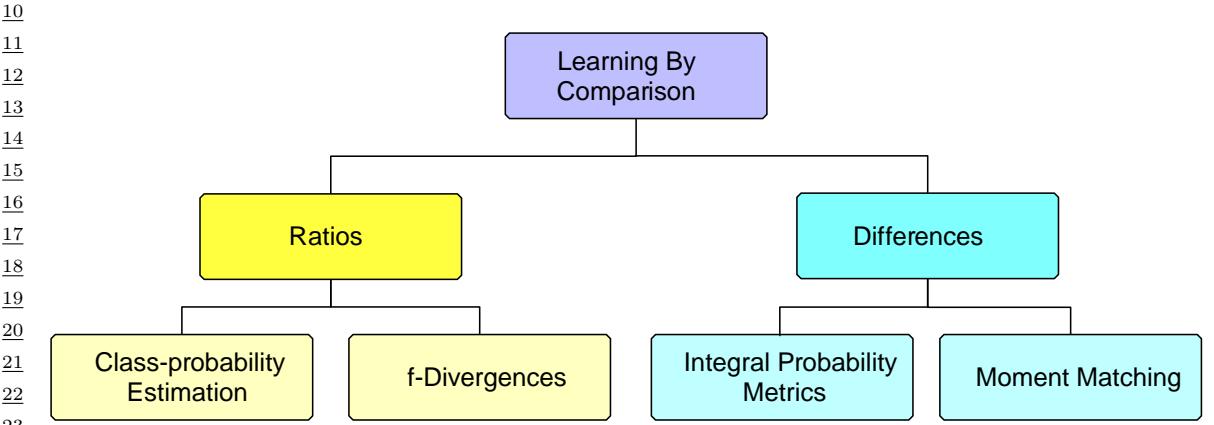


Figure 27.3: Overview of approaches for learning in implicit generative models

and the unknown true data distribution p^* . Recalling equation (27.2), in implicit models we cannot evaluate $q_\theta(\mathbf{x})$, and thus cannot use maximum likelihood training. As implicit models provide a sampling procedure, we instead are searching for learning principles that only use *samples from the model*.

Figure 27.2 shows that the task of learning in implicit models is to determine, from two sets of samples whether their distributions are close to each other and to quantify the distance between them. We can think of this as a ‘two sample’ or likelihood-free approach to learning by comparison. There are many ways of doing this, including using distributional divergences or distances through binary classification, the method of moments, and other approaches. Figure 27.3 shows an overview of different approaches for learning by comparison.

In this chapter, we will not explore a Bayesian approach to learning the model parameters (but see Section 13.6.5). The general approach we take is to find a way to quantify the distance between model and data distributions, use those distances to specify learning objectives, and then train the implicit model parameters θ to minimize this measure of distributional divergence.

27.2.1 Guiding principles

We are looking for objectives $\mathcal{D}(p^*, q)$ that satisfy the following desiderata:

1. Provide guarantees about learning the data distribution: $\operatorname{argmin}_q \mathcal{D}(p^*, q) = p^*$.

- ¹
- ² 2. Can be evaluated only using samples from the data and model distribution.
 - ³ 3. Are computationally cheap to evaluate.
- ⁴

⁵ Many distributional distances and divergences satisfy the first requirement, since by definition they
⁶ satisfy the following:

⁷

$$\mathcal{D}(p^*, q) \geq 0; \quad \mathcal{D}(p^*, q) = 0 \iff p^* = q \quad (27.3)$$

⁸

⁹ Many distributional distances and divergences, however, fail to satisfy the other two requirements:
¹⁰ they cannot be evaluated only using samples — such as the KL divergence, or are computationally
¹¹ intractable — such as the Wasserstein distance. The main approach to overcome these challenges is
¹² to *approximate the desired quantity through optimization* by introducing a comparison model—often
¹³ called a **discriminator** or a **critic** D , such that:

¹⁴

¹⁵

$$\mathcal{D}(p^*, q) = \underset{D}{\operatorname{argmax}} \mathcal{F}(D, p^*, q) \quad (27.4)$$

¹⁶

¹⁷ where \mathcal{F} is a functional that depends on p^* and q only through samples. For the cases we discuss, both
¹⁸ the model and the critic are parametric with parameters θ and ϕ respectively; instead of optimizing
¹⁹ over distributions or functions, we optimize with respect to parameters. For the critic, this results in
²⁰ the optimization problem $\operatorname{argmax}_{\phi} \mathcal{F}(D_{\phi}, p^*, q_{\theta})$. For the model parameters θ , the exact objective
²¹ $\mathcal{D}(p^*, q_{\theta})$ is replaced with the tractable approximation provided through the use of D_{ϕ} .

²² A convenient approach to ensure that $\mathcal{F}(D_{\phi}, p^*, q_{\theta})$ can be estimated using only samples from the
²³ model and the unknown data distribution is to depend on the two distributions only in expectation:

²⁴

$$\mathcal{F}(D_{\phi}, p^*, q_{\theta}) = \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}, \phi) + \mathbb{E}_{q_{\theta}(\mathbf{x})} g(\mathbf{x}, \phi) \quad (27.5)$$

²⁵

²⁶ where f and g are real valued functions whose choice will define \mathcal{F} . In the case of implicit generative
²⁷ models, this can be rewritten to use the sampling path $\mathbf{x} = G_{\theta}(\mathbf{z})$, $\mathbf{z} \sim q(\mathbf{z})$:

²⁸

²⁹

$$\mathcal{F}(D_{\phi}, p^*, q_{\theta}) = \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}, \phi) + \mathbb{E}_{q(\mathbf{z})} g(G_{\theta}(\mathbf{z}), \phi) \quad (27.6)$$

³⁰

³¹ which can be estimated using Monte Carlo estimation

³²

$$\mathcal{F}(D_{\phi}, p^*, q_{\theta}) \approx \frac{1}{N} \sum_{i=1}^N f(\hat{\mathbf{x}}_i, \phi) + \frac{1}{M} \sum_{i=1}^M g(G_{\theta}(\hat{\mathbf{z}}_i), \phi); \quad \hat{\mathbf{x}}_i \sim p^*(\mathbf{x}); \quad \hat{\mathbf{z}}_i \sim q(\mathbf{z}) \quad (27.7)$$

³³

³⁴ Next, we will see how to instantiate these guiding principles in order to find the functions f
³⁵ and g and thus the objective \mathcal{F} which can be used to train implicit models: class probability
³⁶ estimation (Section 27.2.2), bounds on f -divergences (Section 27.2.3), Integral Probability Metrics
³⁷ (Section 27.2.4) and moment matching (Section 27.2.5).

³⁸

³⁹

27.2.2 Class probability estimation

⁴⁰

⁴¹ One way to compare two distributions p^* and q_{θ} is to compute their density ratio $r(\mathbf{x}) = \frac{p^*(\mathbf{x})}{q_{\theta}(\mathbf{x})}$. The
⁴² distributions are the same if and only if the ratio is 1 everywhere in space and density ratios like this
⁴³ are common in probabilistic learning. Since we cannot evaluate we cannot evaluate the densities
⁴⁴

Loss	Objective Function ($D := D(\mathbf{x}; \phi) \in [0, 1]$)
Bernoulli loss	$\mathbb{E}_{p^*(\mathbf{x})}[\log D] + \mathbb{E}_{q_\theta(\mathbf{x})}[\log(1 - D)]$
Brier score	$\mathbb{E}_{p^*(\mathbf{x})}[-(1 - D)^2] + \mathbb{E}_{q_\theta(\mathbf{x})}[-D^2]$
Exponential loss	$\mathbb{E}_{p^*(\mathbf{x})} \left[\left(-\frac{1-D}{D}\right)^{\frac{1}{2}} \right] + \mathbb{E}_{q_\theta(\mathbf{x})} \left[\left(-\frac{D}{1-D}\right)^{\frac{1}{2}} \right]$
Misclassification	$\mathbb{E}_{p^*(\mathbf{x})}[-\mathbb{I}[D \leq 0.5]] + \mathbb{E}_{q_\theta(\mathbf{x})}[-\mathbb{I}[D > 0.5]]$
Hinge loss	$\mathbb{E}_{p^*(\mathbf{x})} \left[-\max \left(0, 1 - \log \frac{D}{1-D} \right) \right] + \mathbb{E}_{q_\theta(\mathbf{x})} \left[-\max \left(0, 1 + \log \frac{D}{1-D} \right) \right]$
Spherical	$\mathbb{E}_{p^*(\mathbf{x})}[\alpha D] + \mathbb{E}_{q_\theta(\mathbf{x})}[\alpha(1 - D)]; \quad \alpha = (1 - 2D + 2D^2)^{-\frac{1}{2}}$

Table 27.1: Proper scoring rules that can be maximized in class probability-based learning of implicit generative models. Based on [ML16].

of implicit models, we must instead develop techniques to compute the density ratio from samples alone, following the guiding principles established above.

To compute the density ratio of the data and implicit model distributions, we assign the label $y = 1$ to samples from the data distribution $\mathbf{x}^* \sim p^*$, and $y = 0$ to the samples from the model distribution $\mathbf{x} \sim q_\theta$. By this construction, $p^*(\mathbf{x}) = p(\mathbf{x}|y=1)$ and $q_\theta(\mathbf{x}) = p(\mathbf{x}|y=0)$. Using this insight along with Bayes rule, we can rewrite the density ratio as:

$$\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})} = \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} = \frac{p(y=1|\mathbf{x})p(\mathbf{x})}{p(y=0)} \Big/ \frac{p(y=0|\mathbf{x})p(\mathbf{x})}{p(y=0)} = \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}. \quad (27.8)$$

This derivation shows one can recast the problem of ratio estimation as binary classification. For simplicity, we have assumed that the same number of data and model samples are used to learn the classifier, so $p(y=0) = p(y=1) = \frac{1}{2}$. Since we only need to classify samples to the positive class, we denote the classifier $p(y=1|\mathbf{x})$ as $D(\mathbf{x})$, the discriminator or critic.

For parametric classification, we can learn discriminators $D_\phi(\mathbf{x}) \in [0, 1]$ with parameters ϕ . Using knowledge and insight about probabilistic classification, we can learn the parameters by minimizing any proper scoring rule [GR07b] (see also Section 14.2.1). For the familiar Bernoulli log-loss (or binary cross entropy loss), we obtain the objective:

$$\begin{aligned} V(q_\theta, p^*) &= \arg \max_{\phi} \mathbb{E}_{p(\mathbf{x}|y)p(y)}[y \log D_\phi(\mathbf{x}) + (1 - y) \log(1 - D_\phi(\mathbf{x}))] \\ &= \arg \max_{\phi} \mathbb{E}_{p(\mathbf{x}|y=1)p(y=1)} \log D_\phi(\mathbf{x}) + \mathbb{E}_{p(\mathbf{x}|y=0)p(y=0)} \log(1 - D_\phi(\mathbf{x})) \end{aligned} \quad (27.9)$$

$$= \arg \max_{\phi} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} \log D_\phi(\mathbf{x}) + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} \log(1 - D_\phi(\mathbf{x})). \quad (27.10)$$

The same procedure can be extended beyond the Bernoulli log-loss to other proper scoring rules used for binary classification, such as those presented in Table 27.1, adapted from [ML16]. The optimal discriminator D is $\frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}$, since:

$$\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})} = \frac{D^*(\mathbf{x})}{1 - D^*(\mathbf{x})} \implies D^*(\mathbf{x}) = \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})} \quad (27.11)$$

By substituting the optimal discriminator into the scoring rule (27.10), we can show that the objective

1 V can also be interpreted as the minimization of the Jensen-Shannon divergence.
2

$$\frac{3}{4} \quad V^*(q_\theta, p^*) = \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(1 - \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})})] \quad (27.12)$$

$$\frac{5}{6} \quad = \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log \frac{p^*(\mathbf{x})}{\frac{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}{2}}] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(\frac{q_\theta(\mathbf{x})}{\frac{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}{2}})] - \log 2 \quad (27.13)$$

$$\frac{7}{8} \quad = \frac{1}{2} D_{\text{KL}} \left(p^* \middle\| \frac{p^* + q_\theta}{2} \right) + \frac{1}{2} D_{\text{KL}} \left(q_\theta \middle\| \frac{p^* + q_\theta}{2} \right) - \log 2 \quad (27.14)$$

$$\frac{10}{11} \quad = JSD(p^*, q_\theta) - \log 2 \quad (27.15)$$

12 where JSD denotes the Jensen-Shannon divergence:
13

$$\frac{14}{15} \quad JSD(p^*, q_\theta) = \frac{1}{2} D_{\text{KL}} \left(p^* \middle\| \frac{p^* + q_\theta}{2} \right) + \frac{1}{2} D_{\text{KL}} \left(q_\theta \middle\| \frac{p^* + q_\theta}{2} \right) \quad (27.16)$$

17 This establishes a connection between *optimal* binary classification and distributional divergences.
18 By using binary classification, we were able to compute the distributional divergence using only
19 samples, which is the important property needed for learning implicit generative models; as expressed
20 in the guiding principles (Section 27.2.1), we have turned an intractable estimation problem — how
21 to estimate the JSD divergence, into an optimization problem — how to learn a classifier which can
22 be used to approximate that divergence.

23 We would like to train the parameters θ of generative model to minimise the divergence:

$$\frac{24}{25} \quad \min_{\theta} JSD(p^*, q_\theta) = \min_{\theta} V^*(q_\theta, p^*) + \log 2 \quad (27.17)$$

$$\frac{26}{27} \quad = \min_{\theta} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} \log D^*(\mathbf{x}) + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} \log(1 - D^*(\mathbf{x})) + \log 2 \quad (27.18)$$

29 Since we do not have access to the optimal classifier D^* but only to the neural approximation D_ϕ
30 obtained using the optimization in (27.10), this results in a min-max optimization problem:

$$\frac{31}{32} \quad \min_{\theta} \max_{\phi} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log D_\phi(\mathbf{x})] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(1 - D_\phi(\mathbf{x}))] \quad (27.19)$$

34 By replacing the generating procedure (27.1) in (27.19) we obtain the objective in terms of the
35 latent variables \mathbf{z} of the implicit generative model:
36

$$\frac{37}{38} \quad \min_{\theta} \max_{\phi} \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log D_\phi(\mathbf{x})] + \frac{1}{2} \mathbb{E}_{q(\mathbf{z})} [\log(1 - D_\phi(G_\theta(\mathbf{z})))], \quad (27.20)$$

40 which recovers the definition proposed in the original Generative Adversarial Network (GAN)[Goo+14b].
41 The core principle behind GANs is to train a discriminator, in this case a binary classifier, to ap-
42 proximate a distance or divergence between the model and data distributions, and to then train the
43 generative model to minimize this approximation of the divergence or distance.

44 Beyond the use of the Bernoulli scoring rule used above, other scoring rules have been used to
45 train generative models via min-max optimization. The Brier scoring rule, which under discriminator
46 optimality conditions can be shown to correspond to minimizing the Pearson χ^2 divergence via
47

1 similar arguments as the ones shown above has lead to LS-GAN [Mao+17]. The hinge scoring rule
2 has become popular [Miy+18b; BDS18], and under discriminator optimality conditions corresponds
3 to minimizing the total variational distance [NWJ+09].

4 The connection between proper scoring rules and distributional divergences allows the construction
5 of converge guarantees for the learning criteria above, under infinite capacity of the discriminator and
6 generator: since the minimizer of distributional divergence is the true data distribution (Equation 27.3),
7 if the discriminator is optimal and the generator has enough capacity, it will learn the data distribution.
8 In practice however, this assumption will not hold, as discriminators are rarely optimal; we will
9 discuss this at length in Section 27.3.

11 27.2.3 Bounds on f -divergences

12 As we saw with the appearance of the Jensen-Shannon divergence in the previous section, we can
13 consider directly using a measure of distributional divergence to derive methods for learning in
14 implicit models. One general class of divergences are the f -divergences [LV06; CS04], defined as:

$$\mathcal{D}_f [p^*(\mathbf{x}) \| q_\theta(\mathbf{x})] = \int q_\theta(\mathbf{x}) f\left(\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}\right) d\mathbf{x} \quad (27.21)$$

15 where f is a convex function such that $f(1) = 0$. For different choices of f , we can recover known
16 distributional divergences such as the KL, reverse KL, and Jensen Shannon divergence. We discuss
17 such connections in Section 2.9.1, and provide a summary in Table 27.2.

18 To evaluate Equation (27.21) we will need to evaluate the density of the data $p^*(\mathbf{x})$ and the model
19 $q_\theta(\mathbf{x})$, neither of which are available. In the previous section we overcame the challenge of evaluating
20 the density ratio by transforming it into a problem of binary classification. In this section, we will
21 instead look towards the role of lower bounds on f -divergences, which is an approach for tractability
22 that is also used for variational inference (Chapter 10).

23 f -divergences have a widely-developed theory in convex analysis and information theory. Since the
24 function f in Equation (27.21) is convex, we know that we can find a tangent that bounds it from
25 below. The variational formulation of the f -divergence is [NWJ10b; NCT16c]:

$$\mathcal{D}_f [p^*(\mathbf{x}) \| q_\theta(\mathbf{x})] = \int q_\theta(\mathbf{x}) f\left(\frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}\right) d\mathbf{x} \quad (27.22)$$

$$= \int q_\theta(\mathbf{x}) \sup_{t: \mathcal{X} \rightarrow \mathbb{R}} \left[t(\mathbf{x}) \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})} - f^\dagger(t(\mathbf{x})) \right] d\mathbf{x} \quad (27.23)$$

$$= \int \sup_{t: \mathcal{X} \rightarrow \mathbb{R}} p^*(\mathbf{x}) t(\mathbf{x}) - q_\theta(\mathbf{x}) f^\dagger(t(\mathbf{x})) d\mathbf{x} \quad (27.24)$$

$$\geq \sup_{t \in \mathcal{T}} \mathbb{E}_{p^*(\mathbf{x})}[t(\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{x})}[f^\dagger(t(\mathbf{x}))]. \quad (27.25)$$

42 In the second line we use the result from convex analysis, discussed in the supplementary material,
43 that re-expresses the convex function f using $f(u) = \sup_t ut - f^\dagger(t)$, where f^\dagger is the convex conjugate
44 of the function f , and t is a parameter we optimize over. Since we apply f at $u = \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}$ for all $\mathbf{x} \in \mathcal{X}$,
45 we make the parameter t be a function $t(\mathbf{x})$. The final inequality comes from replacing the supremum
46 over all functions from the data domain \mathcal{X} to \mathbb{R} with the supremum over a family of functions \mathcal{T}

Divergence	f	f^\dagger	Optimal Critic
KL	$u \log u$	e^{u-1}	$1 + \log r(\mathbf{x})$
Reverse KL	$-\log u$	$-1 - \log(-u)$	$-1/r(\mathbf{x})$
JSD	$u \log u - (u+1) \log \frac{u+1}{2}$	$-\log(2 - e^u)$	$\frac{2}{1+1/r(\mathbf{x})}$
Pearson χ^2	$(u-1)^2$	$\frac{1}{4}u^2 + u$	$\left(\sqrt{r(\mathbf{x})} - 1\right) \sqrt{1/r(\mathbf{x})}$

8 Table 27.2: Standard divergences as f divergences for various choices of f . The optimal critic is written as a
9 function of the density ratio $r(\mathbf{x}) = \frac{p^*(\mathbf{x})}{q_\theta(\mathbf{x})}$.

11 (such as the family of functions expressible by a neural network architecture), which might not be
12 able to capture the true supremum. The function t takes the role of the discriminator or critic.

13 The final expression in Equation (27.25) follows the general desired form of Equation 27.5: it is the
14 difference of two expectations, and these expectations can be computed by Monte Carlo estimation
15 using only samples, as in Equation (27.7); despite starting with an objective (Equation 27.21) which
16 contravened the desired principles for training implicit generative models, variational bounds have
17 allowed us to construct an approximation which satisfies all desiderata.

18 Using bounds on the f -divergence, we obtain an objective (27.25) that allows learning both the
19 generator and critic parameters. We use a critic D with parameters ϕ to estimate the bound, and
20 then optimize the parameters θ of the generator to minimise the approximation of the f -divergence
21 provided by the critic (we replace t above with D_ϕ , to retain standard GAN notation):

$$23 \quad \min_{\theta} \mathcal{D}_f(p^*, q_\theta) \geq \min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})}[D_\phi(\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{x})}[f^\dagger(D_\phi(\mathbf{x}))] \quad (27.26)$$

$$25 \quad = \min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})}[D_\phi(\mathbf{x})] - \mathbb{E}_{q(\mathbf{z})}[f^\dagger(D_\phi(G_\theta(\mathbf{z})))] \quad (27.27)$$

27 This approach to train an implicit generative model leads to f -GANs [NCT16c]. It is worth noting
28 that there exists an equivalence between the scoring rules in the previous section and bounds on
29 f -divergences [RW11]: for each scoring rule we can find an f -divergence that leads to the same
30 training criteria and the same min-max game of Equation 27.27. An intuitive way to grasp the
31 connection between f -divergences and proper scoring rules is through their use of density ratios:
32 in both cases the optimal critic approximates a quantity directly related to the density ratio (see
33 Table 27.2 for f -divergences and Equation (27.11) for scoring rules).

34

35 27.2.4 Integral probability metrics

37 Instead of comparing distributions by using their ratio as we did in the previous two sections, we
38 can instead study their difference. A general class of measure of difference is given by the Integral
39 Probability Metrics (IPMs) defined as:

$$40 \quad I_{\mathcal{F}}(p^*(\mathbf{x}), q_\theta) = \sup_{f \in \mathcal{F}} |\mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x})|. \quad (27.28)$$

43 The function f is a test or witness function that will take the role of the discriminator or critic. To
44 use IPMs we must define the class of real valued, measurable functions \mathcal{F} over which the supremum is
45 taken, and this choice will lead to different distances, just as choosing different convex functions
46 f leads to different f -divergences. Integral probability metrics are distributional distances: beyond
47

satisfying the conditions for distributional divergences $\mathcal{D}(p^*, q) \geq 0$; $\mathcal{D}(p^*, q) = 0 \iff p^* = q$ (Equation (27.3)), they are also symmetric $\mathcal{D}(p, q) = \mathcal{D}(q, p)$ and satisfy the triangle inequality $\mathcal{D}(p, q) \leq \mathcal{D}(p, r) + \mathcal{D}(r, q)$.

Not all function families satisfy these conditions of create a valid distance $I_{\mathcal{F}}$. To see why consider the case where $\mathcal{F} = \{z\}$ where z is the function $z(\mathbf{x}) = 0$. This choice of \mathcal{F} entails that regardless of the two distributions chosen, the value in Equation 27.28 would be 0, violating the requirement that distance between two distributions be 0 only if the two distributions are the same. A popular choice of \mathcal{F} for which $I_{\mathcal{F}}$ satisfies the conditions of a valid distributional distance is the set of 1-Lipschitz functions, which leads to the Wasserstein distance [Vil08]:

$$W_1(p^*(\mathbf{x}), q_\theta(\mathbf{x})) = \sup_{f: \|f\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x}) \quad (27.29)$$

We show an example of a Wasserstein critic in Figure 27.4a. The supremum over the set of 1-Lipschitz functions is intractable for most cases, which again suggests the introduction of a learned critic:

$$W_1(p^*(\mathbf{x}), q_\theta(\mathbf{x})) = \sup_{f: \|f\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x}) \quad (27.30)$$

$$\geq \max_{\phi: \|D_\phi\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} D_\phi(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} D_\phi(\mathbf{x}), \quad (27.31)$$

where the critic D_ϕ has to be regularized to be 1-Lipschitz (various techniques for Lipschitz regularization via gradient penalties or spectral normalization methods have been used [ACB17; Gul+17]). As was the case with f -divergences, we replace an intractable quantity which requires a supremum over a class of functions with a bound obtained using a subset of this function class, a subset which can be modeled using neural networks.

To train a generative model, we again introduce a min max game:

$$\min_{\theta} W_1(p^*(\mathbf{x}), q_\theta(\mathbf{x})) \geq \min_{\theta} \max_{\phi: \|D_\phi\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} D_\phi(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} D_\phi(\mathbf{x}) \quad (27.32)$$

$$= \min_{\theta} \max_{\phi: \|D_\phi\|_{\text{Lip}} \leq 1} \mathbb{E}_{p^*(\mathbf{x})} D_\phi(\mathbf{x}) - \mathbb{E}_{q(\mathbf{z})} D_\phi(G_\theta(\mathbf{z})) \quad (27.33)$$

This leads to the popular WassersteinGAN [ACB17].

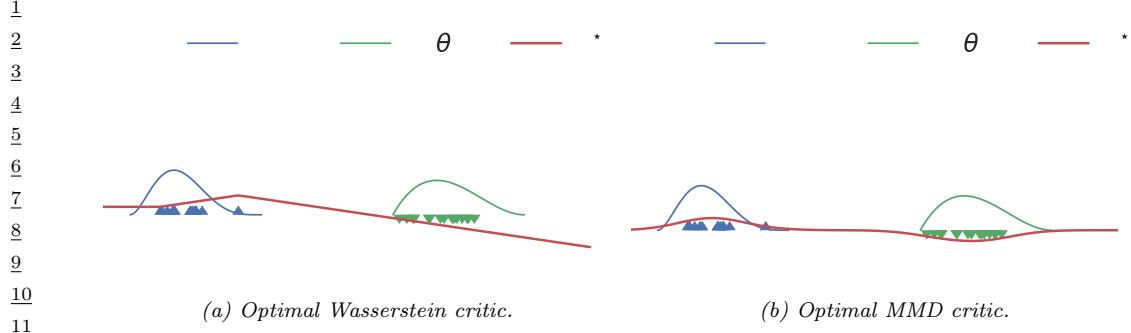
If we replace the choice of function family \mathcal{F} to that of functions in an RKHS (Section 18.3.7.1) with norm one, we obtained the maximum mean discrepancy MMD discussed in Section 2.9.3:

$$\text{MMD}(p^*(\mathbf{x}), q_\theta(\mathbf{x})) = \sup_{f: \|f\|_{\text{RKHS}} = 1} \mathbb{E}_{p^*(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x}). \quad (27.34)$$

We show an example of an MMD critic in Figure 27.4b. It is often more convenient to use the square MMD loss [LSZ15; DRG15], which can be evaluated using the kernel \mathcal{K} (Section 18.3.7.1):

$$\text{MMD}^2(p^*, q_\theta) = \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{p^*(\mathbf{x}')} \mathcal{K}(\mathbf{x}, \mathbf{x}') - 2 \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{q_\theta(\mathbf{y})} \mathcal{K}(\mathbf{x}, \mathbf{y}) + \mathbb{E}_{q_\theta(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{y}')} \mathcal{K}(\mathbf{y}, \mathbf{y}') \quad (27.35)$$

$$= \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{p^*(\mathbf{x}')} \mathcal{K}(\mathbf{x}, \mathbf{x}') - 2 \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{q(\mathbf{z})} \mathcal{K}(\mathbf{x}, G_\theta(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z})} \mathbb{E}_{q(\mathbf{z}')} \mathcal{K}(G_\theta(\mathbf{z}), G_\theta(\mathbf{z}')) \quad (27.36)$$



[Figure 27.4: Optimal critics in Integral Probability Metrics \(IPMs\). Generated by `IPM_divergences.ipynb`](#)

The MMD can be directly used to learn a generative model, often called a generative matching network [LSZ15]:

$$\min_{\theta} \text{MMD}^2(p^*, q_\theta) \quad (27.37)$$

The choice of kernel is important. Using a fixed or predefined kernel such as a radial basis function (RBF) kernel might not be appropriate for all data modalities, such as high dimensional images. Thus we are looking for a way to learn a feature function ζ such that we can $\mathcal{K}(\zeta(\mathbf{x}), \zeta(\mathbf{x}'))$ is a valid kernel; luckily, we can use that for any characteristic kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}')$ and injective function ζ , $\mathcal{K}(\zeta(\mathbf{x}), \zeta(\mathbf{x}'))$ is also a characteristic kernel. While this tells us that we can use feature functions in the MMD objective, it does not tell us how to learn the features. In order to ensure that the learned features are sensitive to differences between the data distribution $p^*(\mathbf{x})$ and the model distribution $q_\theta(\mathbf{x})$, the kernel parameters are trained to *maximize* the square MMD. This again casts the problem into a familiar min max objective by learning the projection ζ with parameters ϕ [Li+17a]:

$$\min_{\theta} \text{minMMD}_{\zeta}^2(p_{\mathcal{D}}, q_\theta) \quad (27.38)$$

$$\begin{aligned} &= \min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{p^*(\mathbf{x}')} \mathcal{K}(\zeta_\phi(\mathbf{x}), \zeta_\phi(\mathbf{x}')) \\ &\quad - 2 \mathbb{E}_{p^*(\mathbf{x})} \mathbb{E}_{q_\theta(\mathbf{y})} \mathcal{K}(\zeta_\phi(\mathbf{x}), \zeta_\phi(\mathbf{y})) \\ &\quad + \mathbb{E}_{q_\theta(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{y}') \mathcal{K}(\zeta_\phi(\mathbf{y}), \zeta_\phi(\mathbf{y}'))} \end{aligned} \quad (27.39)$$

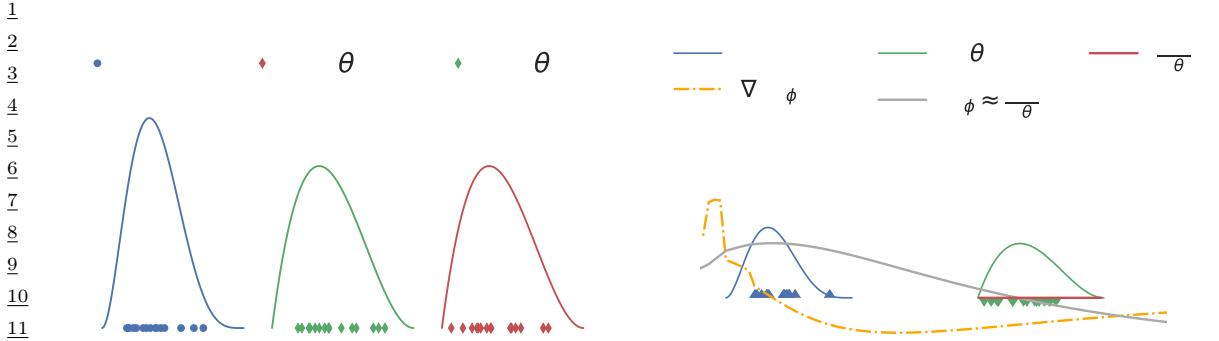
where ζ_ϕ regularized to be injective, though this is sometimes relaxed [Bin+18]. Unlike the Wasserstein distance and f -divergences, Equation (27.39) can be estimated using Monte Carlo estimation, without requiring a lower bound on the original objective.

40

41 27.2.5 Moment matching

42 More broadly than distances defined by integral probability metrics, for a set of test statistics s , one
43 can define a **moment matching** criteria [Pea36], also known as the method of moments:

$$\min_{\theta} \left\| \mathbb{E}_{p^*(\mathbf{x})} s(\mathbf{x}) - \mathbb{E}_{q_\theta(\mathbf{x})} s(\mathbf{x}) \right\|_2^2 \quad (27.40)$$



(a) Failure of the KL divergence to distinguish between distributions with non-overlapping support: $D_{\text{KL}}(p^* \| q_{\theta_1}) = D_{\text{KL}}(p^* \| q_{\theta_2}) = \infty$, despite q_{θ_2} being closer to p^* than q_{θ_1} .

(b) The density ratio $\frac{p^*}{q_{\theta}}$ used by the KL divergence and a smooth estimate given by an MLP, together with the gradient it provides with respect to the input variable.

Figure 27.5: The KL divergence cannot provide learning signal for distributions without overlapping support (left), while the smooth approximation given by a learned decision surface like an MLP can (right). Generated by [IPM_divergences.ipynb](#)

where $m(\boldsymbol{\theta}) = \mathbb{E}_{q_{\theta}(\mathbf{x})} s(\mathbf{x})$ is the *moment function*. The choice of statistic $s(\mathbf{x})$ is crucial, since as with distributional divergences and distances, we would like to ensure that if the objective is minimized and reaches the minimal value 0, the two distributions are the same $p^*(\mathbf{x}) = q_{\theta}(\mathbf{x})$. To see that not all functions s satisfy this requirement consider the function $s(\mathbf{x}) = \mathbf{x}$: simply matching the means of two distributions is not sufficient to match higher moments (such as variance). For likelihood based models the score function $s(\mathbf{x}) = \log q_{\theta}(\mathbf{x})$ satisfies the above requirement and leads to a consistent estimator [Vaa00], but this choice of s is not available for implicit generative models.

This motivates the search for other approaches of integrating the method of moments for implicit models. The MMD can be seen as a moment matching criteria, by matching the means of the two distributions after lifting the data into the feature space of a Reproducing Kernel Hilbert Space. But moment matching can go beyond integral probability metrics: Ravuri et al. [Rav+18] show that one can *learn* useful moments by using s as the set of features containing the gradients of a trained discriminator classifier D_{ϕ} together with the feature of the learned critic: $s_{\phi}(\mathbf{x}) = [\nabla_{\phi} D_{\phi}(\mathbf{x}), h_1(\mathbf{x}), \dots, h_n(\mathbf{x})]$ where $h_1(\mathbf{x}), \dots, h_n(\mathbf{x})$ are the hidden activations of the learned critic. Both features and gradients are needed: the gradients $\nabla_{\phi} D_{\phi}(\mathbf{x})$ are required to ensure the estimator for the parameters $\boldsymbol{\theta}$ is consistent, since the number of moments $s(\mathbf{x})$ needs to be larger than the number of parameters $\boldsymbol{\theta}$, which will be true if the critic will have more parameters than the model; the features $h_i(\mathbf{x})$ are added since they have been shown empirically to improve performance, thus showcasing the importance of the choice of test statistics s used to train implicit models.

27.2.6 On density ratios and differences

We have seen how density ratios (Sections 27.2.2 and 27.2.3) and density differences (Section 27.2.4) can be used to define training objectives for implicit generative models. We now explore some of the

¹ distinctions between using ratios and differences for learning by comparison, as well as explore the
² effects of using approximations to these objectives using function classes such as neural networks has
³ on these distinctions.
⁴

⁵ One often stated downside of using divergences that rely on density ratios (such as f -divergences)
⁶ is their poor behavior when the distributions p^* and q_θ do not have overlapping support. For
⁷ non-overlapping support, the density ratio $\frac{p^*}{q_\theta}$ will be ∞ in the parts of the space where $p^*(\mathbf{x}) > 0$ but
⁸ $q_\theta(\mathbf{x}) = 0$, and 0 otherwise. In that case, the $D_{\text{KL}}(p^*||q_\theta) = \infty$ and the $JSD(p^*, q_\theta) = \log 2$, regardless
⁹ of the value of θ . Thus *f -divergences cannot distinguish between different model distributions when*
¹⁰ *they do not have overlapping support with the data distribution*, as visualized in Figure 27.5a. This is
¹¹ in contrast with difference based methods such as IPMs such as the Wasserstein distance and the
¹² MMD, which have smoothness requirements built in the *definition* of the method, by constraining
¹³ the norm of the critic (Equations (27.29) and (27.34)). We can see the effect of these constraints
¹⁴ in Figure 27.4: both the Wasserstein distance and the MMD provide useful signal in the case of
¹⁵ distributions with non-overlapping support.

¹⁶ While the *definition* of f -divergences relies on density ratios (Equation (27.21)), we have seen that
¹⁷ to train implicit generative models we use approximations to those divergences obtained using a
¹⁸ parametric critic D_ϕ . If the function family of the critic used to approximate the divergence (via the
¹⁹ bound or class probability estimation) contains only smooth functions, it will not be able to model
²⁰ the sharp true density ratio which jumps from 0 to ∞ , but instead provide a smooth approximation.
²¹ We show an example in Figure 27.5b, where we show the density ratio for two distributions without
²² overlapping support and an approximation provided by an MLP trained to approximate the KL
²³ divergence using Equation 27.25. Here, the smooth decision surface provided by the MLP can be
²⁴ used to train a generative model while the underlying KL divergence cannot be; the learned MLP
²⁵ provides the gradient signal on how to move distribution mass to areas with more density under
²⁶ the data distribution, while the KL divergence provides a zero gradient almost everywhere in the
²⁷ space. This ability of approximations to f -divergences to overcome non-overlapping support issues is
²⁸ a desirable property of generative modeling training criteria, as it allows models to learn the data
²⁹ distribution regardless of initialization [Fed+18]. Thus while the case of non-overlapping support
³⁰ provides an important theoretical difference between IPMs and f -divergences, it is less significant in
³¹ practice since bounds on f -divergences or class probability estimation are used with smooth critics
³² to approximate the underlying divergence.

³³ Some density ratio and density difference based approaches also share commonalities: bounds are
³⁴ used both for f -divergences (variational bounds in Equation 27.25) and for the Wasserstein distance
³⁵ (Equation (27.31)). These bounds to distributional divergence and distances have their own set of
³⁶ challenges: since the generator minimizes a lower bound of the underlying divergence or distance,
³⁷ minimizing this objective provides no guarantees that the divergence will decrease in training. To see
³⁸ this, we can look at Equation 27.26: its RHS can get arbitrarily low without decreasing the LHS,
³⁹ the divergence we are interested in minimizing; this is unlike variational *upper* bound on the KL
⁴⁰ divergence used to train Variational Autoencoders Chapter 22.

⁴¹

⁴²

⁴³ 27.3 Generative Adversarial Networks

⁴⁴

⁴⁵ We have looked at different learning principles that do not require the use of explicit likelihoods, and
⁴⁶ thus can be used to train implicit models. These learning principles specify training criteria, but do
⁴⁷

not tell us how to *train* models or parametrize models. To answer these questions, we now look at algorithms for training implicit models, where the models (both the discriminator and generator) are deep neural networks; this leads us to Generative Adversarial Networks (GANs). We cover how to turn learning principles into loss functions for training GANs (Section 27.3.1); how to train models using gradient descent (Section 27.3.2); how to improve GAN optimization (Section 27.3.4) and how to assess GAN convergence (Section 27.3.5).

27.3.1 From learning principles to loss functions

In Section 27.2 we discussed learning principles for implicit generative models: class probability estimation, bounds on f -divergences, Integral Probability Metrics and moment matching. These principles can be used to formulate loss functions to train the model parameters θ and the critic parameters ϕ . Many of these objectives follow use **zero-sum losses** via a **min-max** formulation: the generator’s goal is to minimize the same function the discriminator is maximizing. We can formalize this as:

$$\min \max V(\phi, \theta) \quad (27.41)$$

As an example, we recover the original GAN with the Bernoulli log-loss (Equation (27.19)) when

$$V(\phi, \theta) = \frac{1}{2} \mathbb{E}_{p^*(\mathbf{x})} [\log D_\phi(\mathbf{x})] + \frac{1}{2} \mathbb{E}_{q_\theta(\mathbf{x})} [\log(1 - D_\phi(\mathbf{x}))]. \quad (27.42)$$

The reason most of the learning principles we have discussed lead to zero-sum losses is due to their underlying structure: the critic maximizes a quantity in order to approximate a divergence or distance — such as an f -divergence or Integral Probability Metric — and the model minimizes this approximation to the divergence or distance. That need not be the case, however. Intuitively, the discriminator training criteria needs to ensure that the discriminator can distinguish between data and model samples, while the generator loss function needs to ensure that model samples are indistinguishable from data according to the discriminator.

To construct a GAN that is not zero-sum, consider the zero-sum criteria in the original GAN (Equation 27.42), induced by the Bernoulli scoring rule. The discriminator tries to distinguish between data and model samples by classifying the data as real (label 1) and samples as fake (label 0), while the goal of the generator is to minimize the probability that the discriminator classifies its samples as fake: $\min_{\theta} \mathbb{E}_{q_\theta(\mathbf{x})} \log(1 - D_\phi(\mathbf{x}))$. An equally intuitive goal for the generator is to maximize the probability that the discriminator classifies its samples as real. While the difference might seem subtle, this loss, known as the “non-saturating loss” [Goo+14b], defined as $\mathbb{E}_{q_\theta(\mathbf{x})} - \log D_\phi(\mathbf{x})$, enjoys better gradient properties early in training, as shown in Figure 27.6: the non-saturating loss provides a stronger learning signal (via the gradient) when the generator is performing poorly, and the discriminator can easily distinguish its samples from data, i.e. $D(G(\mathbf{z}))$ is low; more on the gradients properties the saturating and non-saturating losses can be found in [AB17; Fed+18].

There exist many other GAN losses which are not zero-sum, including formulations of LS-GAN [Mao+17], GANs trained using the hinge loss [LY17] and RelativisticGANs [JM18]. We can thus generally write a GAN formulation as follows:

$$\min_{\phi} L_D(\phi, \theta); \quad \min_{\theta} L_G(\phi, \theta). \quad (27.43)$$

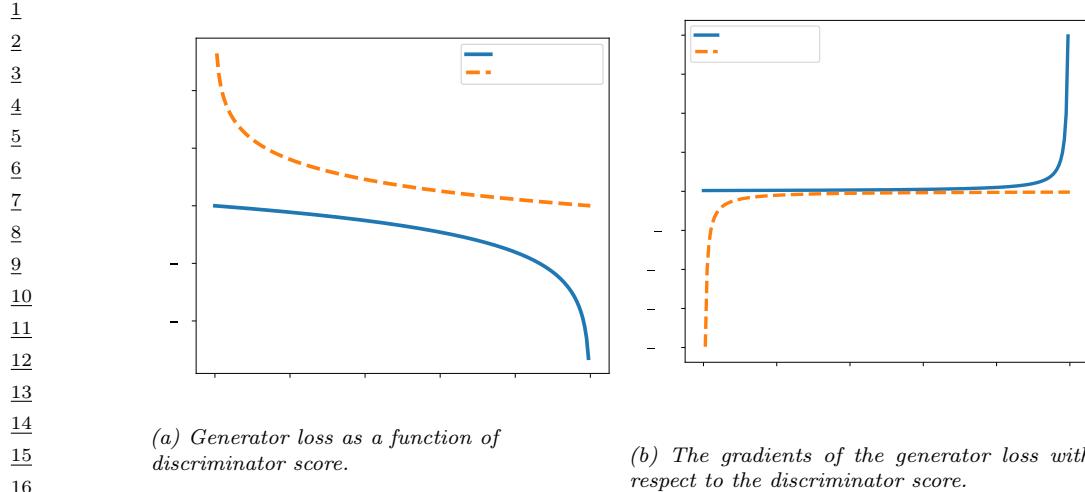


Figure 27.6: Saturating $\log(1 - D(G(z))$) vs non-saturating $-\log D(G(z))$ loss functions. The non-saturating loss provides stronger gradients when the discriminator is easily detecting that generated samples are fake.

Generated by GAN_loss_types.ipynb

We recover the zero-sum formulations if $-L_D(\phi, \theta) = L_G(\phi, \theta) = V(\phi, \theta)$. Despite departing from the zero-sum structure, the nested form of the optimization remains in the general formulation, as we will discuss in Section 27.3.2.

The loss functions for the discriminator and generator, L_D and L_G respectively, follow the general form in Equation 27.5, which allows them to be used to efficiently train implicit generative models. The majority of loss functions considered here can thus be written as follows:

$$L_D(\phi, \theta) = \mathbb{E}_{p^*(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{x})} h(D_\phi(\mathbf{x})) = \mathbb{E}_{p^*(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q(\mathbf{z})} h(D_\phi(G_\theta(\mathbf{z}))) \quad (27.44)$$

$$L_G(\phi, \theta) = \mathbb{E}_{q_\theta(\mathbf{x})} l(D_\phi(\mathbf{x})) = \mathbb{E}_{q(\mathbf{z})} l(D_\phi(G_\theta(\mathbf{z}))) \quad (27.45)$$

where $g, h, l : \mathbb{R} \rightarrow \mathbb{R}$. We recover the original GAN for $g(t) = -\log t$, $h(t) = -\log(1-t)$ and $l(t) = \log(1-t)$; the non-saturating loss for $g(t) = -\log t$, $h(t) = -\log(1-t)$ and $l(t) = -\log(t)$; the Wasserstein distance formulation for $g(t) = t$, $h(t) = -t$ and $l(t) = t$; for f -divergences $g(t) = t$, $h(t) = -f^\dagger(t)$ and $l(t) = f^\dagger(t)$.

27.3.2 Gradient Descent

GANs employ the learning principles discussed above in conjunction with gradient based learning for the parameters of the discriminator and generator. We assume a general formulation with a discriminator loss function $L_D(\phi, \theta)$ and a generator loss function $L_G(\phi, \theta)$. Since the discriminator is often introduced to approximate a distance or divergence $D(p^*, q_\theta)$ (Section 27.2), for the generator to minimize a good approximation of that divergence one should solve the discriminator optimization fully for each generator update. That would entail that for each generator update one would first find the optimal discriminator parameters $\phi^* = \arg \min_{\phi} L_D(\phi, \theta)$ in order to perform a gradient update given by $\nabla_{\theta} L_G(\phi^*, \theta)$. Fully solving the inner optimization problem $\phi^* = \arg \min_{\phi} L_D(\phi, \theta)$

for each optimization step of the generator is computationally prohibitive, which motivates the use of alternating updates: performing a few gradient steps to update the discriminator parameters, followed by a generator update. Note that when updating the discriminator, we keep the generator parameters fixed, and when updating the generator, we keep the discriminator parameters fixed. We show a general algorithm for these alternative updates in Algorithm 30.

Algorithm 30: General GAN training algorithm with alternating updates

```

1 Initialize  $\phi, \theta$ ;
2 for each training iteration do
3   for  $K$  steps do
4     Update the discriminator parameters  $\phi$  using the gradient  $\nabla_\phi L_D(\phi, \theta)$ ;
5     Update the generator parameters  $\theta$  using the gradient  $\nabla_\theta L_G(\phi, \theta)$  ;
6 Return  $\phi, \theta$ 

```

We are thus interested in computing $\nabla_\phi L_D(\phi, \theta)$ and $\nabla_\theta L_G(\phi, \theta)$. Given the choice of loss functions follows the general form in Equations 27.44 and 27.45 both for the discriminator and generator, we can compute the gradients that can be used for training. To compute the discriminator gradients, we write:

$$\nabla_\phi L_D(\phi, \theta) = \nabla_\phi [\mathbb{E}_{p^*(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{x})} h(D_\phi(\mathbf{x}))] \quad (27.46)$$

$$= \mathbb{E}_{p^*(\mathbf{x})} \nabla_\phi g(D_\phi(\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{x})} \nabla_\phi h(D_\phi(\mathbf{x})) \quad (27.47)$$

where $\nabla_\phi g(D_\phi(\mathbf{x}))$ and $\nabla_\phi h(D_\phi(\mathbf{x}))$ can be computed via backpropagation, and each expectation can be estimated using Monte Carlo estimation. For the generator, we would like to compute the gradient:

$$L_G(\phi, \theta) = \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{x})} l(D_\phi(\mathbf{x})) \quad (27.48)$$

Here we cannot change the order of differentiation and integration since the distribution under the integral depends on the differentiation parameter θ . Instead, we will use that $q_\theta(\mathbf{x})$ is the distribution induced by an implicit generative model (also known as the “reparametrization trick”, see Section 6.6.4):

$$\nabla_\theta L_G(\phi, \theta) = \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{x})} l(D_\phi(\mathbf{x})) = \nabla_\theta \mathbb{E}_{q(\mathbf{z})} l(D_\phi(G_\theta(\mathbf{z}))) = \mathbb{E}_{q(\mathbf{z})} \nabla_\theta l(D_\phi(G_\theta(\mathbf{z}))) \quad (27.49)$$

and again use Monte Carlo estimation to approximate the gradient using samples from the prior $q(\mathbf{z})$. Replacing the choice of loss functions and Monte Carlo estimation in Algorithm 30 leads to Algorithm 31, which is often used to train GANs. See <https://github.com/probml/pyprobml/tree/master/gan> for some sample (PyTorch) code which trains various kinds of (convolutional) GANs on the CelebA face dataset.

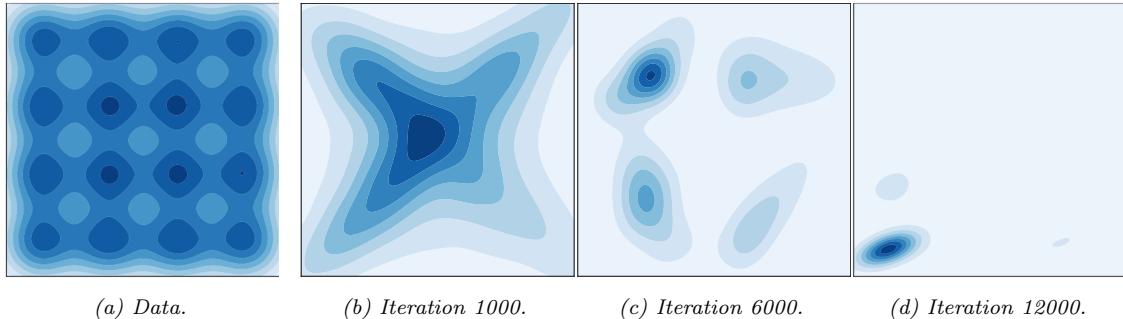
27.3.3 Challenges with GAN training

Due to the adversarial game nature of GANs the optimizing dynamics of GANs are both hard to study in theory, and to stabilize in practice. GANs are known to suffer from **mode collapse**, a

```

1
2 Algorithm 31: GAN training algorithm
3 1 Initialize  $\phi, \theta$ ;
4 2 for each training iteration do
5   3   for  $K$  steps do
6     4     Sample minibatch of  $M$  noise vectors  $\mathbf{z}_m \sim q(\mathbf{z})$ ;
7     5     Sample minibatch of  $M$  examples  $\mathbf{x}_m \sim p^*(\mathbf{x})$ ;
8     6     Update the discriminator by performing stochastic gradient descent using this gradient:
9       7        $\nabla_{\phi} \frac{1}{M} \sum_{m=1}^M [g(D_{\phi}(\mathbf{x}_m)) + \nabla_{\phi} h(D_{\phi}(G_{\theta}(\mathbf{z}_m)))]$ . ;
10      8       Sample minibatch of  $M$  noise vectors  $\mathbf{z}_m \sim q(\mathbf{z})$ ;
11      9       Update the generator by performing stochastic gradient descent using this gradient:
12        10        $\nabla_{\theta} \frac{1}{M} \sum_{m=1}^M l(D_{\phi}(G_{\theta}(\mathbf{z}_m)))$ . ;
13
14 9 Return  $\phi, \theta$ 
15

```



26 *Figure 27.7: Mode hopping and collapse behavior in GAN training for a grid-structured mixture of Gaussians*
27 *dataset. Left: the dataset, a mixture of 16 Gaussians in 2 dimensions. The other panels depict the progression*
28 *of samples of a GAN trained on this dataset, showing how mass accumulates on one mode out of the 16.*
29 *Generated by [gan_mog_mode_hopping.ipynb](#).*

30
31
32
33
34 phenomenon where the generator converges to a distribution which does not cover not all the modes
35 (peaks) of the data distribution, thus the model underfits the distribution. We show an example
36 in Figure 27.7: while the data is a mixture of Gaussians with 16 modes (Figure 27.7a), the model
37 converges only to one mode (Figure 27.7d). Alternatively, another problematic behavior is **mode**
38 **hopping**, where the generator “hops” between generating different modes of the data distribution.
39 An intuitive explanation for this behavior is as follows: if the generator becomes good at generating
40 data from one mode, it will generate more from that mode. If the discriminator cannot learn to
41 distinguish between real and generated data in this mode, the generator has no incentive to expand
42 its support and generate data from other modes. On the other hand, if the discriminator eventually
43 learns to distinguish between the real and generated data inside this mode, the generator can simply
44 move (hop) to a new mode, and this game of cat and mouse can continue.

45 While mode collapse and mode hopping are often associated with GANs, many improvements have
46 made GAN training more stable, and these behaviors more rare. These improvements include using
47

1 large batch sizes, increasing the discriminator neural capacity, using discriminator and generator
2 regularization, as well as more complex optimization methods.
3

4 27.3.4 Improving GAN optimization

5 Hyperparameter choices such as the choice of momentum can be crucial when training GANs, with
6 lower momentum values being preferred compared to the usual high momentum used in supervised
7 learning. Algorithms such as Adam[KB14a] provide a great boost in performance [RMC16]. Many
8 other optimization methods have been successfully applied to GANs, such as those which target
9 variance reduction [Cha+19c]; those which backpropagate through gradient steps, thus ensuring not
10 that generator that does well not against the current discriminator, but to the discriminator *after it*
11 *has been updated* [Met+16]; or using a local bilinear approximation of the two player game [SA19].
12 While promising, these advanced optimization methods tend to have a higher computational cost,
13 making them harder to scale to large models or large datasets compared to less efficient optimization
14 methods.
15

16 27.3.5 Convergence of GAN training

17 The challenges with GAN optimization make it hard to quantify when convergence has occurred. In
18 Section 27.2 we saw how global convergence guarantees can be provided under optimality conditions for
19 multiple objectives constructed starting with different distributional divergences and distances: if the
20 discriminator is optimal, the generator is minimising a distributional divergence or distance between
21 the data and model distribution, and thus under infinite capacity and perfect optimization can learn
22 the data distribution. This type of argument has been used since the original GAN paper [Goo+14b]
23 to connect GANs to standard objectives in generative models, and obtain the associated theoretical
24 guarantees. From a game theory perspective, this type of convergence guarantee provides an existence
25 proof of a global Nash equilibrium for the GAN game, though under strong assumptions. A Nash
26 equilibrium is achieved when both players (the discriminator and generator) would incur a loss if
27 they decide to act by changing their parameters. Consider the original GAN defined by the objective
28 in Equation 27.19; then $q_\theta = p^*$ and $D_\phi(\mathbf{x}) = \frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})} = \frac{1}{2}$ is a global Nash equilibrium, since
29 for a given q_θ , the ratio $\frac{p^*(\mathbf{x})}{p^*(\mathbf{x}) + q_\theta(\mathbf{x})}$ is the optimal discriminator (Equation 27.11), and given an
30 optimal discriminator, the data distribution is the optimal generator as it is the minimizer of the
31 Jensen-Shannon divergence (Equation 27.15).
32

33 While these global theoretical guarantees provide useful insights about the GAN game, they do
34 not account for optimization challenges that arise with accounting for the optimization trajectories
35 of the two players, or for neural network parametrization since they assume infinite capacity both for
36 the discriminator and generator. In practice GANs do not decrease a distance or divergence at every
37 optimization step [Fed+18] and global guarantees are difficult to obtain when using optimization
38 methods such as gradient descent. Instead, the focus shifts towards local convergence guarantees,
39 such as reaching a local Nash equilibrium. A local Nash equilibrium requires that both players are at
40 a local, not global minimum: a local Nash equilibrium is a stationary point (the gradients of the two
41 loss functions are zero, i.e $\nabla_\phi L_D(\phi, \theta) = \mathbf{0}$ and $\nabla_\theta L_G(\phi, \theta) = \mathbf{0}$), and the eigenvalues of the Hessian
42 of each player ($\nabla_\phi \nabla_\phi L_D(\phi, \theta) L_D$ and $\nabla_\theta \nabla_\theta L_G(\phi, \theta)$) are non-negative; for a longer discussion on
43 Nash equilibria in continuous games see [RBS16]. For the general GAN game, it is not guaranteed
44 that a local Nash equilibrium always exists [FO20], and weaker conditions such as stationarity or
45

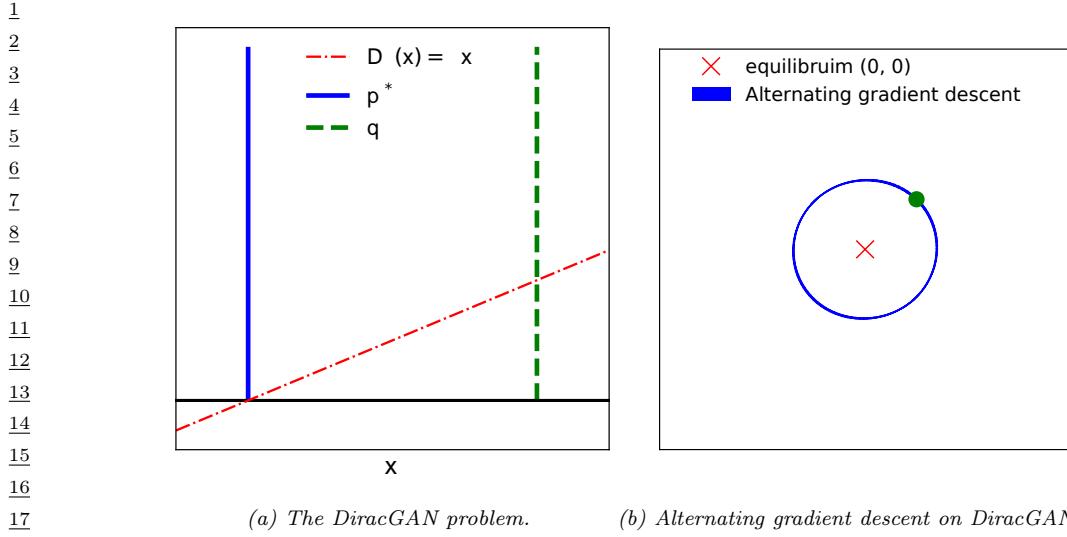


Figure 27.8: Visualizing divergence using a simple GAN: DiracGAN. Generated by [DiracGAN.ipynb](#)

locally stable stationarity have been studied [Ber+19]; other equilibrium definitions inspired by game theory have also been used [JNJ20; HLC19].

To motivate why convergence analysis is important in the case of GANs, we visualize an example of a GAN that does not converge trained with gradient descent. In DiracGAN [MGN18a] the data distribution $p^*(\mathbf{x})$ is Dirac delta distribution with mass at zero. The generator is modeling a Dirac delta distribution with parameter θ : $G_\theta(z) = \theta$ and the discriminator is a linear function of the input with learned parameter ϕ : $D_\phi(x) = \phi x$. We also assume a GAN formulation where $g = h = -l$ in the general loss functions L_D and L_G defined above, see Equations (27.44) and (27.45). This results in the zero-sum game given by:

$$L_D = \mathbb{E}_{p^*(x)} - l(D_\phi(x)) + \mathbb{E}_{q_\theta(x)} - l(D_\phi(x)) = -l(0) - l(\theta\phi) \quad (27.50)$$

$$L_G = \mathbb{E}_{p^*(x)} l(D_\phi(x)) + \mathbb{E}_{q_\theta(x)} l(D_\phi(x)) = +l(0) + l(\theta\phi) \quad (27.51)$$

where l depends on the GAN formulation used ($l(z) = -\log(1 + e^{-z})$ for instance). The unique equilibrium point is $\theta = \phi = 0$. We visualize the DiracGAN problem in Figure 27.8 and show that DiracGANs with alternating gradient descent (Algorithm 30) do not reach the equilibrium point, but instead takes a circular trajectory around the equilibrium.

There are two main theoretical approaches taken to understand GAN convergence behavior around an equilibrium: by analyzing either the discrete dynamics of gradient descent, or the underlying continuous dynamics of the game using approaches such as stability analysis. To understand the difference between the two approaches, consider the discrete dynamics defined by gradient descent with learning rates αh and λh , either via alternating updates (as we have seen in Algorithm 30):

$$\phi_t = \phi_{t-1} - \alpha h \nabla_\phi L_D(\phi_{t-1}, \theta_{t-1}), \quad (27.52)$$

$$\theta_t = \theta_{t-1} - \lambda h \nabla_\theta L_G(\phi_t, \theta_{t-1}) \quad (27.53)$$

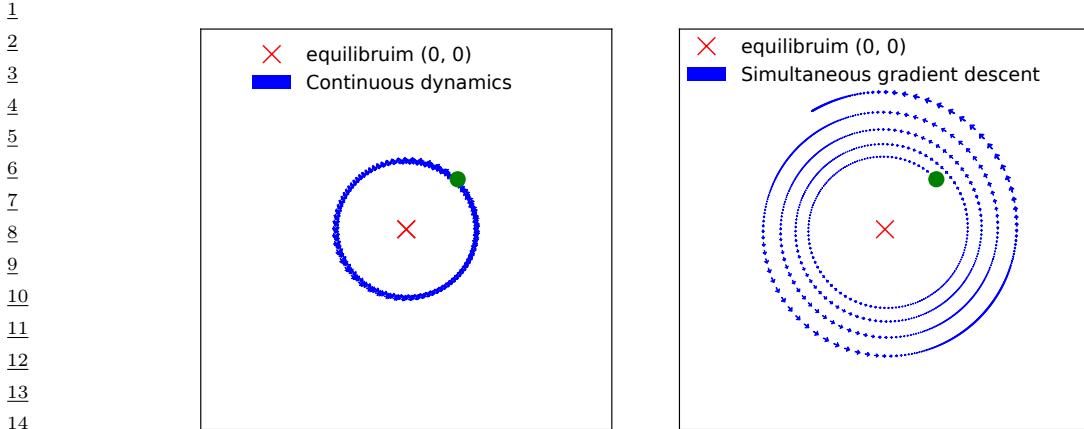


Figure 27.9: Continuous (left) and discrete dynamics (right) take different trajectories in DiracGAN.
Generated by [DiracGAN.ipynb](#)

or simultaneous updates, where instead of alternating the gradient updates between the two players, they are both updated simultaneously:

$$\phi_t = \phi_{t-1} - \alpha h \nabla_\phi L_D(\phi_{t-1}, \theta_{t-1}), \quad (27.54)$$

$$\theta_t = \theta_{t-1} - \lambda h \nabla_\theta L_G(\phi_{t-1}, \theta_{t-1}) \quad (27.55)$$

The above dynamics of gradient descent are obtained using Euler numerical integration from the ODEs that describes the game dynamics of the two players:

$$\dot{\phi} = -\nabla_\phi L_D(\phi, \theta), \quad (27.56)$$

$$\dot{\theta} = -\nabla_\theta L_G(\phi, \theta) \quad (27.57)$$

One approach to understand the behavior of GANs is to study these underlying ODEs which when discretize results in the gradient descent updates above, rather than directly study the discrete updates. These ODEs can be used for stability analysis to study the behavior around an equilibrium entails finding the eigenvalues of the Jacobian of the game

$$J = \begin{bmatrix} -\nabla_\phi \nabla_\phi L_D(\phi, \theta) & -\nabla_\theta \nabla_\phi L_D(\phi, \theta) \\ -\nabla_\phi \nabla_\theta L_G(\phi, \theta) & -\nabla_\theta \nabla_\theta L_G(\phi, \theta) \end{bmatrix} \quad (27.58)$$

evaluated at a stationary point (i.e. where $\nabla_\phi L_D(\phi, \theta) = 0$ and $\nabla_\theta L_G(\phi, \theta) = 0$). If the eigenvalues of the Jacobian all have negative real parts, then the system is asymptotically stable around the equilibrium; if at least one eigenvalue has positive real part, the system is unstable around the equilibrium. For the DiracGAN, the Jacobian evaluated at the equilibrium $\theta = \phi = 0$ is:

$$J = \begin{bmatrix} \nabla_\phi \nabla_\phi l(\theta\phi) + l'(0) & \nabla_\theta \nabla_\phi l(\theta\phi) + l'(0) \\ -\nabla_\phi \nabla_\theta l(\theta\phi) + l'(0) & -\nabla_\theta \nabla_\theta l(\theta\phi) + l'(0) \end{bmatrix} = \begin{bmatrix} 0 & l'(0) \\ -l'(0) & 0 \end{bmatrix} \quad (27.59)$$

where eigenvalues of this Jacobian are $\lambda_{\pm} = \pm il'(0)$. This is interesting, as the real parts of the eigenvalues are both 0; this result tells us that there is no asymptotic convergence to an equilibrium, but linear convergence could still occur. In this simple case we can reach the conclusion that convergence does not occur as we observe that there is a preserved quantity in this system, as $\theta^2 + \phi^2$ does not change in time (Figure 27.9, left):

$$\frac{d(\theta^2 + \phi^2)}{dt} = 2\theta \frac{d\theta}{dt} + 2\phi \frac{d\phi}{dt} = -2\theta l'(\theta\phi)\phi + 2\phi l'(\theta\phi)\theta = 0.$$

Using stability analysis to understand the underlying continuous dynamics of GANs around an equilibrium has been used to show that explicit regularization can help convergence [NK17; Bal+18]. Alternatively, one can directly study the updates of simultaneous gradient descent shown in Equations 27.54 and 27.55. Under certain conditions [MNG17b] prove that GANs trained with simultaneous gradient descent reach a local Nash equilibrium [MNG17b]. Their approach relies on assessing the convergence of series of the form $F^k(\mathbf{x})$ resulting from the repeated application of gradient descent update of the form $F(\mathbf{x}) = \mathbf{x} + hG(\mathbf{x})$, where h is the learning rate. Since the function F depends on the learning rate h , their convergence results depend on the size of the learning rate, which is not the case for continuous time approaches.

Both continuous and discrete approaches have been useful in understanding and improving GAN training; however, both approaches still leave a gap between our theoretical understanding and the most commonly used algorithms to train GANs in practice, such as alternating gradient descent or more complex optimizers used in practice, like Adam. Far from only providing different proof techniques, these approaches can reach different conclusions about the convergence of a GAN: we show an example in Figure 27.9, where we see that simultaneous gradient descent and the continuous dynamics behave differently when a large enough learning rate is used. In this case, the discretization error — the difference between the behavior of the continuous dynamics in Equations 27.56 and 27.57 and the gradient descent dynamics in Equations 27.54 and 27.55 — makes the analysis of gradient descent using continuous dynamics reach the wrong conclusion about DiracGAN [Ros+21]. This difference in behavior has been a motivator to train GANs with higher order numerical integrators such as RungeKutta4, which to more closely follow the underlying continuous system compared to gradient descent [Qin+20].

While optimization convergence analysis is an indispensable step in understanding GAN training and has led to significant practical improvements, it is worth noting that ensuring converge to an equilibrium does not ensure the model has learned a good fit of the data distribution. The loss landscape determined by the choice of L_D and L_G , as well as the parametrization of the discriminator and generator can lead to equilibria which do not capture the data distribution. The lack of distributional guarantees provided by game equilibria showcases the need to complement convergence analysis with work looking at the effect of gradient based learning in this game setting on the learned distribution.

40

41

42 27.4 Conditional GANs

43

44 We have thus far discussed how to use implicit generative models to learn a true unconditional
 45 distribution $p^*(\mathbf{x})$ from which we only have samples. It is often useful, however, to be able to learn
 46 *conditional distributions* of the from $p^*(\mathbf{x}|\mathbf{y})$. This requires having **paired data**, where each input
 47

\underline{x}_n is paired with a corresponding set of covariates \mathbf{y}_n , such as a class label, or a set of attributes or words, so $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1 : N\}$, as in standard supervised learning. The conditioning variable can be discrete - like a class label - or continuous - such as an embedding encoding information about past experience. **Conditional generative models** are appealing since we can specify that we want the generated sample to be associated with conditioning information y , making them very amenable to real world applications - see Section 27.7.

To be able to learn implicit conditional distributions $q_\theta(\mathbf{x}|\mathbf{y})$, we require datasets that specify the conditioning information associated with data as well as adapt model architectures and loss functions to learn conditional distributions. In the GAN case, changing the loss function for the generative model can be done by changing the critic, since the critic is part of the loss function of the generator; it is important for the critic to provide learning signal accounting for conditioning information, by penalizing a generator which provides realistic samples but which ignore the provided conditioning.

If we do not change the form of the min-max game, but provide the conditioning information to the two players, a **conditional GAN** can be created from the original GAN game [MO14]:

$$\min_{\theta} \max_{\phi} \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{p^*(\mathbf{x}|\mathbf{y})} [\log D_\phi(\mathbf{x}, \mathbf{y})] + \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{x}|\mathbf{y})} [\log(1 - D_\phi(\mathbf{x}, \mathbf{y}))] \quad (27.60)$$

In the case of implicit latent variable models, the embedding information becomes an additional input to the generator, together with the latent variable \mathbf{z} :

$$\min_{\theta} \max_{\phi} \mathcal{L}(\theta, \phi) = \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{p^*(\mathbf{x}|\mathbf{y})} [\log D_\phi(\mathbf{x}, \mathbf{y})] + \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{x}|\mathbf{y})} [\log(1 - D_\phi(\mathcal{G}_\theta(\mathbf{z}, \mathbf{y}), \mathbf{y}))] \quad (27.61)$$

For discrete conditioning information such as labels, one can also add a new loss function, by training a critic which does not only learn to distinguish between real and fake data, but learns to classify both data and generated samples as pertaining to one of the K classes provided in the dataset [OOS17]:

$$\mathcal{L}_c(\theta, \phi) = \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{p^*(\mathbf{x}|\mathbf{y})} [\log D_\phi(\mathbf{y}|\mathbf{x})] + \frac{1}{2} \mathbb{E}_{p(\mathbf{y})} \mathbb{E}_{q_\theta(\mathbf{x}|\mathbf{y})} [\log(D_\phi(\mathbf{y}|\mathbf{x}))] \quad (27.62)$$

Note that while we could have two critics, one unsupervised critic and one supervised which maximizes the equation above, in practice the same critic is used, to aid shaping the features used in both decision surfaces. Unlike the adversarial nature of the unsupervised game, it is in the interest of both players to maximize the classification loss \mathcal{L}_c . Thus together with the adversarial dynamics provided by \mathcal{L} , the two players are trained as follows:

$$\max_{\phi} \mathcal{L}(\theta, \phi) + \mathcal{L}_c(\theta, \phi) \quad \min_{\theta} \mathcal{L}(\theta, \phi) - \mathcal{L}_c(\theta, \phi) \quad (27.63)$$

In the case of conditional latent variable models, the latent variable controls the sample variability *inside* the mode specified by the conditioning information. In early conditional GANs, the conditioning information was provided as additional input to the discriminator and generator, for example by concatenating the conditioning information to the latent variable \mathbf{z} in the case of the generator; it has been since observed that it is important to provide the conditioning information at various layers of the model, both for the generator and the discriminator [DV+17; DSK16] or use a projection discriminator [MK18].

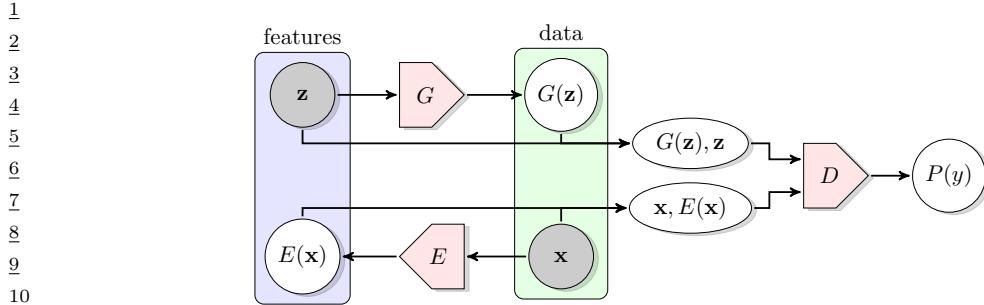


Figure 27.10: Learning an implicit posterior using an adversarial approach, as done in BiGAN. From Figure 1 of [DKD16]. Used with kind permission of Jeff Donahue.

27.5 Inference with GANs

Unlike other latent variable models such as Variational Autoencoders, GANs do not define an inference procedure associated with the generative model. To deploy the principles behind GANs to find a posterior distribution $p(\mathbf{z}|\mathbf{x})$, multiple approaches have been taken, from combining GANs and Variational Autoencoders via hybrid methods [MNG17a; Sri+17; Lar+16; Mak+15b] to constructing inference methods catered to implicit variable models [Dum+16; DKD16; DS19]. An overview of these methods can be found in [Hus17b].

GAN based methods which perform inference and learn **implicit posterior distribution** $p(\mathbf{z}|\mathbf{x})$ introduce changes to the GAN algorithm to do so. An example of such a method is **BiGAN** (bidirectional GAN) [DKD16] or **ALI** (adversarialy learned inference) [Dum+16], which trains an implicit parametrized encoder E_ζ to map input \mathbf{x} to latent variables \mathbf{z} . To ensure consistency between the encoder E_ζ and the generator G_θ , an adversarial approach is introduced with a discriminator D_ϕ learning to distinguish between pairs of data and latent samples: D_ϕ learns to consider pairs $(\mathbf{x}, E_\zeta(\mathbf{x}))$ with $\mathbf{x} \sim p^*$ as real, while $(G_\theta(\mathbf{z}), \mathbf{z})$ with $\mathbf{z} \sim q(\mathbf{z})$ is considered fake. This approach, shown in Figure 27.10, ensures that the joint distributions are matched, and thus the marginal distribution $q_\theta(\mathbf{x})$ given by G_θ should learn $p^*(\mathbf{x})$, while the conditional distribution $p_\zeta(\mathbf{z}|\mathbf{x})$ given by E_ζ should learn $q_\theta(\mathbf{z}|\mathbf{x}) = \frac{q(\mathbf{x}, \mathbf{z})}{q(\mathbf{x})} \propto q_\theta(\mathbf{x}|\mathbf{z})q(\mathbf{z})$. This joint GAN loss can be used both to train the generator G_θ and the encoder E_ζ , without requiring a reconstruction loss common in other inference methods. While not using a reconstruction loss, this objective retains the property that under global optimality conditions the encoder and decoder are inverses of each other: $E_\zeta(G_\theta(\mathbf{z})) = \mathbf{z}$ and $G_\zeta(E_\zeta(\mathbf{x})) = \mathbf{x}$. (See also Section 22.2.7 for a discussion of how VAEs learn to ensure $p^*(\mathbf{x})p_\zeta(\mathbf{z}|\mathbf{x})$ matches $p(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ using an explicit model of the data.)

27.6 Neural architectures in GANs

We have so far discussed the learning principles, algorithms, and optimization methods that can be used to train implicit generative models parametrized by deep neural networks. We have not discussed, however, the importance of the choice of neural network architectures for the model and the critic, choices which have fueled the progress in GAN generation since their conception. We will look at a few case studies which show the importance of information about data modalities into

the critic and the generator (Section 27.6.1), employing the right inductive biases (Section 27.6.2), incorporating attention in GAN models (Section 27.6.3), progressive generation (Section 27.6.4), regularization (Section 27.6.5) and using large scale architectures (Section 27.6.6).

27.6.1 The importance of discriminator architectures

Since the discriminator or critic is rarely optimal – either due to the use of alternating gradient descent or the lack of capacity of the neural discriminator – GANs do not perform distance or divergence minimization in practice. Instead, the critic acts as part of a **learned loss function** for the model (the generator). Every time the critic is updated, the loss function for the generative model changes; this is in stark contrast with divergence minimization such maximum likelihood estimation, where the loss function stays the same throughout the training of the model. Just as learning features of data instead of handcrafting them is a reason for the success of deep learning methods, learning loss functions advanced the state of the art of generative modeling. Critics that take data modalities into account — such as convolutional critics for images and recurrent critics for sequential data such as text or audio — become part of data modality dependent loss functions. This in turn provides modality specific learning signal to the model, for example by penalizing blurry images and encouraging sharp edges, which is achieved due to the convolutional parametrization of the critic. Even within the same data modality changes to critic architectures and regularisation have been one of the main drivers in obtaining better GANs, since they affect the generator’s loss function, and thus also the *gradients of the generator* and have a strong effect on optimization.

27.6.2 Architectural inductive biases

While the original GAN paper used convolutions only sparingly, Deep Convolutional GAN (**DCGAN**) [RMC15] performed an extensive study on what architectures are most useful for GAN training, resulting in a set of useful guidelines that led to a substantial boost in performance. Without changing the learning principles behind GANs, DCGAN was able to obtain better results on image data by using convolutional generators (Figure 27.11) and critics, using BatchNormalization for both the generator and critic, replacing pooling layers with strided convolutions, using ReLU activation networks in the generator and LeakyReLU activations in the discriminator. Many of these principles are still in use today, for larger architectures and with various loss functions. Since DCGAN, residual convolutional layers have become a key staple of both models and critics for image data [Gul+17], and recurrent architectures are used for sequence data such as text [SSG18b; Md+19].

27.6.3 Attention in GANs

Attention mechanisms are explained in detail in Section 16.2.7. In this section, we discuss how to use them for both the GAN generator and discriminator; this is called the Self Attention GAN or **SAGAN** model [Zha+19b]. The advantage of self attention is that it ensures that both discriminator and generator have access to a *global* view of other units of the same layer, unlike convolutional layers. This is illustrated in Figure 27.12, which visualizes the global span of attention: query points can attend to various other areas in the image.

The self-attention mechanism for convolutional features reshaped to $\mathbf{h} \in \mathbb{R}^{C \times N}$ is defined by $\mathbf{f} = W_f \mathbf{h}$, $\mathbf{g} = W_g \mathbf{h}$, $\mathbf{S} = \mathbf{f}^T \mathbf{g}$, where $W_f \in \mathbb{R}^{C' \times C}$, $W_g \in \mathbb{R}^{C' \times C}$, where $C' \leq C$ is a hyperparameter.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

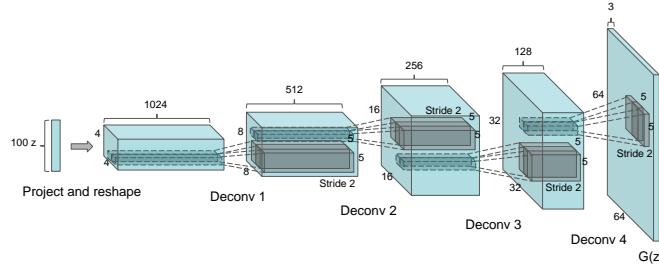


Figure 27.11: DCGAN convolutional generator. From Figure 1 of [RMC15]. Used with kind permission of Alec Radford.

From $\mathbf{S} \in \mathbb{R}^{N \times N}$, a probability row matrix β is obtained by applying the softmax operator for each row, which is then used to attend to a linear transformation of the features $\mathbf{o} = W_o(W_h \mathbf{h})\beta^T \in R^{C \times N}$, using learned operators $W_h \in \mathbb{R}^{C' \times C}$, $W_o \in \mathbb{R}^{C \times C'}$. An output is then created by $\mathbf{y} = \gamma \mathbf{o} + \mathbf{h}$, where $\gamma \in \mathbb{R}$ is a learned parameter.

Beyond providing global signal to the players, it is worth noting the flexibility of the self attention mechanism. The learned parameter γ ensures that the model can decide not to use the attention layer, and thus adding self attention does not restrict the set of possible models an architecture can learn. Moreover, self attention significantly increases the number of parameters of the model (each attention layer introduced 4 learned matrices $\mathbf{W}_f, \mathbf{W}_g, \mathbf{W}_h, \mathbf{W}_o$), an approach that has been observed as a fruitful way to improve GAN training.

27.6.4 Progressive generation

One of the first successful approaches to generating higher resolution, color images from a GAN is via an iterative process, by first generating a lower dimensional sample, and then using that as conditioning information to generate a higher dimensional sample, and repeating the process until the desired resolution is reached. **LapGAN** [DCF+15] uses a Laplacian pyramid as the iterative building block, by first upsampling the lower dimensional samples using a simple upsampling operation, such as smoothed upsampling, and then using a conditional generator to produce a residual to be added to the upsampled version to produce the higher resolution sample. In turn, this higher resolution sample can then be provided to another LapGAN layer to produce another, even higher resolution sample, and so on - this process is shown in Figure 27.13. In LapGAN, a different generator and critic are trained for each iterative block of the model; in ProgressiveGAN [Kar+18] the lower resolution generator and critic are “grown”, by becoming part of the generator and critic used to learn to generate higher resolution samples. The higher resolution generator is obtained by adding new layers on top of the last layer of the lower resolution generator. A residual connection between an upscaled version of the lower dimensional sample and the output of the newly created higher resolution generator is added, which is annealed from 0 to 1 in training - transitioning from using the



Figure 27.12: Attention queries used by a SAGAN model, showcasing the global span of attention. Each row first shows the input image and a set of color coded query locations in the image. The subsequent images show the attention maps corresponding to each query location in the first image, with the query color coded location being shown, and arrows from it to the attention map are used to highlight the most attended regions. From Figure 1 of [Zha+19b]. Used with kind permission of Han Zhang.

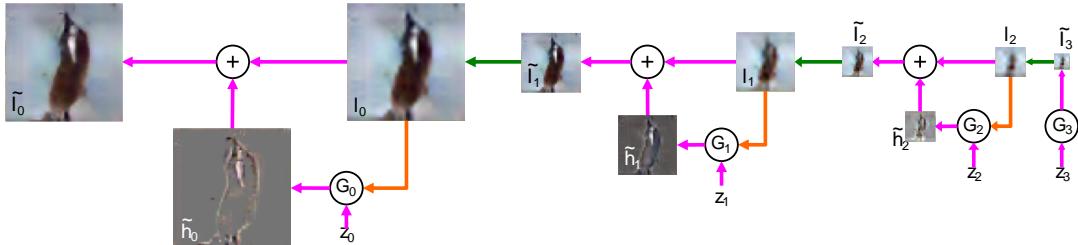


Figure 27.13: LapGAN generation algorithm: the generation process starts with a low dimension sample, which gets upsampled and residually added to the output of a generator at a higher resolution. The process gets repeated multiple times. From Figure 1 of [DCF+15]. Used with kind permission of Emily Denton.

upscaled version of the lower dimensional sample early in training, to only using the sample of the higher resolution generator at the end of training. A similar change is done to the discriminator, but the new layers are added before the layers of the higher of the lower level discriminator. Figure 27.14 shows the growing generator and discriminators in ProgressiveGAN training.

27.6.5 Regularization

Regularizing both the discriminator and the generator has by now a long tradition in GAN training. Regularizing GANs can be justified from multiple perspectives: theoretically, as it has been shown to be tied to convergence analysis [MGN18b]; empirically, as it has been shown to help performance and stability in practice [RMC15; Miy+18c; Zha+19b; BDS18]; and intuitively, as it can be used to avoid overfitting in the discriminator and generator. Regularization approaches include adding noise to the

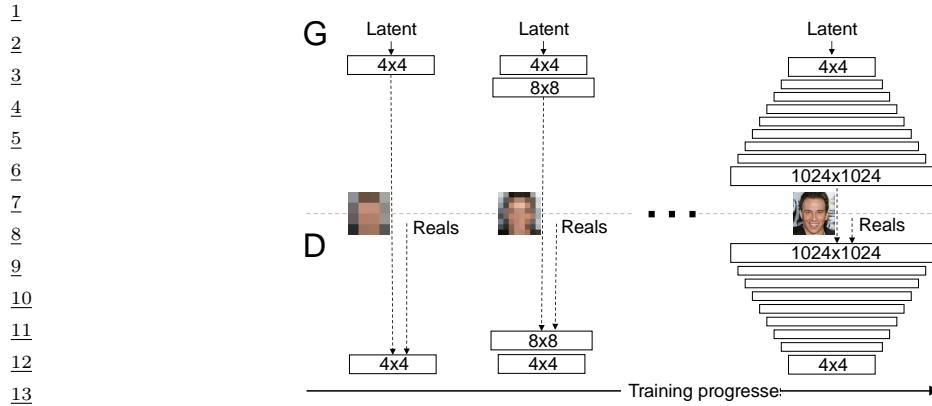


Figure 27.14: ProgressiveGAN training algorithm. The input to the discriminator at the bottom of the figure is either a generated image, or a real image (denotes as ‘Reals’ in the figure) at the corresponding resolution. From Figure 1 of [Kar+18]. Used with kind permission of Tero Karras.

discriminator input [AB17], adding noise to the discriminator and generator hidden features [ZML16], using BatchNorm for the two players [RMC15], adding dropout in the discriminator [RMC15], Spectral Normalization [Miy+18c; Zha+19b; BDS18], gradient penalties — penalizing the norm of the discriminator gradient with respect to its input $\|\nabla_x D_\phi(x)\|^2$ by adding a regularization term to the loss function [Arb+18; Fed+18; ACB17; Gul+17]. Often regularization methods help training regardless of the type of loss function used, and have been shown to have effects both on training performance as well as a stabilizer of the GAN game. However, improving stability and improving performance in GAN training can be at odds with each other, since too much regularization can make the models very stable, but reduce performance [BDS18].

27.6.6 Scaling up GAN models

By combining many of the architectural tricks discussed thus far — very large residual networks, self attention, spectral normalization both in the discriminator and the generator, BatchNormalization in the generator — one can train GANs to generating diverse, high quality data, as done with BigGAN [BDS18], StyleGAN [Kar+20c], and Alias-Free GAN [Kar+21]. Beyond combining carefully chosen architectures and regularization, creating large scale GANs also require changes in optimization, with large batch sizes being a key component. This furthers the view that the key components of the GAN game — the losses, the parametrization of the models, and optimization have to be viewed collectively rather than in isolation.

27.7 Applications

The ability to generate new plausible data enables a wide range of applications for GANs. This section will look at a set of applications that aim to demonstrate the breadth of GANs across different data modalities: images (Section 27.7.1), video (Section 27.7.2), audio (Section 27.7.3) and text (Section 27.7.4), and include applications such as imitation learning (Section 27.7.5), domain



Figure 27.15: Increasingly realistic synthetic faces generated by different kinds of GAN, specifically (from left to right): original GAN [Goo+14b], DCGAN [RMC15], CoupledGAN [LT16], ProgressiveGAN [Kar+18], StyleGAN [KLA19]. Used with kind permission of Ian Goodfellow. An online demo, which randomly generates face images using StyleGAN, can be found at <https://thispersondoesnotexist.com>.

adaption (Section 27.7.6) and art (Section 27.7.7).

27.7.1 GANs for image generation

The most widely studied application area is in image generation. Image generation can take various forms, of which we cover the translation of one image to another using either paired or unpaired data sets. There are many other topics related to image GANs that we do not cover, and a more complete overview can be found in other sources, such as [Goo16] for the theory and [Bro19] for the practice. We show the progression of quality in sample generation of faces using GANs in Figure 27.15. There is also increasing need to consider the generation of images with regards to the potential risks they can have when used in other domains, which involve discussions of synthetic media and **deep fakes**, and sources for discussion include [Bru+18; Wit].

27.7.1.1 Conditional image generation

Class-conditional image generation using GANs has become a very fruitful endeavor. BigGAN [BDS18] carries out class-conditional generation of ImageNet samples across a variety of categories, from dogs to cats and volcanoes and hamburgers. StyleGAN [KLA19] is able to generate high quality images of faces at high resolution by learning a conditioning style vector and the ProgressiveGAN architecture discussed in Section 27.6.4. By learning the conditioning vector they are able to generate samples which are interpolating between the styles of other samples, for example by preserving coarser style elements such as pose or face shape from one sample, and smaller scale style elements such as hair style from another; this provides fine grained control over the style of the generated images.

27.7.1.2 Paired image-to-image generation

We have discussed in Section 27.4 how using paired data of the form $(\mathbf{x}_n, \mathbf{y}_n)$ can be used to build conditional generative models of $p(\mathbf{x}|\mathbf{y})$. In some cases, the conditioning variable \mathbf{y} has the same size and shape as the output variable \mathbf{x} . The resulting model $p_{\theta}(\mathbf{x}|\mathbf{y})$ can then be used to perform **image to image translation**, as illustrated in Figure 27.16, where \mathbf{y} is drawn from the **source**

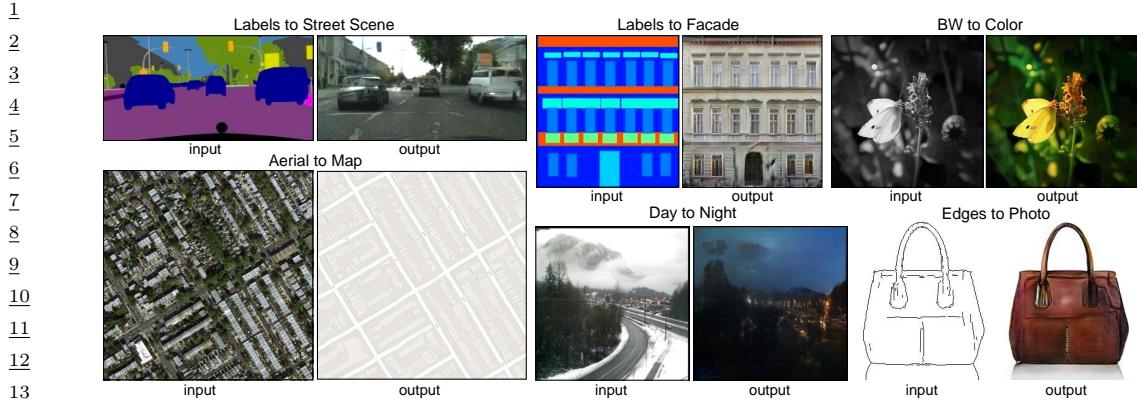


Figure 27.16: Example results on several image-to-image translation problems as generated by the pix2pix conditional GAN. From Figure 1 of [Iso+17]. Used with kind permission of Philip Isola.

domain, and \mathbf{x} from the **target domain**. Collecting paired data of this form can be expensive, but in some cases, we can acquire it automatically. One such example is image colorization, where a paired dataset can easily be obtained by processing color images into grayscale images (see e.g., [Jas]).

A conditional GAN used for paired image-to-image translation was proposed in [Iso+17], and is known as the **pix2pix** model. It uses a U-net style architecture for the generator, as used for semantic segmentation tasks. However, they replace the batch normalization layers with instance normalization, as in neural style transfer.

For the discriminator, pix2pix uses a **patchGAN** model, that tries to classify local patches as being real or fake (as opposed to classifying the whole image). Since the patches are local, the discriminator is forced to focus on the style of the generated patches, and ensure they match the statistics of the target domain. A patch-level discriminator is also faster to train than a whole-image discriminator, and gives a denser feedback signal. This can produce results similar to Figure 27.16 (depending on the dataset).

33

27.7.1.3 Unpaired image-to-image generation

36 A major drawback of conditional GANs is the need to collect paired data. It is often much easier 37 to collect **unpaired data** of the form $\mathcal{D}_x = \{\mathbf{x}_n : n = 1 : N_x\}$ and $\mathcal{D}_y = \{\mathbf{y}_n : n = 1 : N_y\}$. For 38 example, \mathcal{D}_x might be a set of daytime images, and \mathcal{D}_y a set of night-time images; it would be 39 impossible to collect a paired dataset in which exactly the same scene is recorded during the day and 40 night (except using a computer graphics engine, but then we wouldn't need to learn a generator).

41 We assume that the datasets \mathcal{D}_x and \mathcal{D}_y come from the marginal distributions $p(\mathbf{x})$ and $p(\mathbf{y})$ 42 respectively. We would then like to fit a joint model of the form $p(\mathbf{x}, \mathbf{y})$, so that we can compute 43 conditionals $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y}|\mathbf{x})$ and thus translate from one domain to another. This is called 44 **unsupervised domain translation**.

45 In general, this is an ill-posed problem, since there are an infinite number of different joint 46 distributions that are consistent with a set of marginals $p(\mathbf{x})$ and $p(\mathbf{y})$. We can try, however, to learn 47

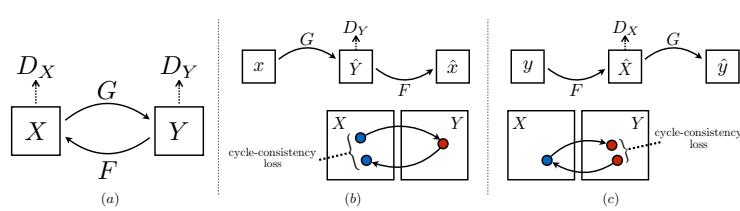


Figure 27.17: Illustration of the CycleGAN training scheme. (a) Illustration of the 4 functions that are trained. (b) Forward cycle consistency from X back to X . (c) Backwards cycle consistency from Y back to Y . From Figure 3 of [Zhu+17]. Used with kind permission of Jun-Yan Zhu.

a joint distribution such that samples from it satisfy additional constraints. For example, if G is a conditional generator that maps a sample from \mathcal{X} to \mathcal{Y} , and F maps a sample from \mathcal{Y} to \mathcal{X} , it is reasonable to require that these be inverses of each other, i.e., $F(G(\mathbf{x})) = \mathbf{x}$ and $G(F(\mathbf{y})) = \mathbf{y}$. This is called a **cycle consistency** loss [Zhu+17]. We can encourage G and F to satisfy this constraint by using a penalty term on the difference between the starting image and the image we get after going through this cycle:

$$\mathcal{L}_{\text{cycle}} = \mathbb{E}_{p(\mathbf{x})} \|F(G(\mathbf{x})) - \mathbf{x}\|_1 + \mathbb{E}_{p(\mathbf{y})} \|G(F(\mathbf{y})) - \mathbf{y}\|_1 \quad (27.64)$$

To ensure that the outputs of G are samples from $p(\mathbf{y})$ and those of F are samples from $p(\mathbf{x})$, we use a standard GAN approach, introducing discriminators D_X and D_Y , which can be done using any choice of GAN loss \mathcal{L}_{GAN} , as visualized in Figure 27.17. Finally, we can optionally check that applying the conditional generator to images from its own domain does not change them:

$$\mathcal{L}_{\text{identity}} = \mathbb{E}_{p(\mathbf{x})} \|\mathbf{x} - F(\mathbf{x})\|_1 + \mathbb{E}_{p(\mathbf{y})} \|\mathbf{y} - G(\mathbf{y})\|_1 \quad (27.65)$$

We can combine all three of these consistency losses to train the translation mappings F and G , using hyperparameters λ_1 and λ_2 :

$$\mathcal{L} = \mathcal{L}_{\text{GAN}} + \lambda_1 \mathcal{L}_{\text{cycle}} + \lambda_2 \mathcal{L}_{\text{identity}} \quad (27.66)$$

CycleGAN results on various datasets are shown in Figure 27.18. The bottom row shows how CycleGAN can be used for **style transfer**.

27.7.2 Video generation

The GAN framework can be expanded from individual images (frames) to videos; the techniques used to generate realistic images can also be applied to generate videos, with additional techniques required to ensure *spatio-temporal consistency*. Spatio-temporal consistency is obtained by ensuring that the discriminator has access to the real data and generated sequences in order, thus penalizing the generator when generating realistic individual frames without respecting temporal order [SMS17; Sai+20; CDS19; Tul+18]. Another discriminator can be employed to additionally ensure each frame is realistic [Tul+18; CDS19]. The generator itself needs to have a temporal element, which is often implemented through a recurrent component. As with images, the generation framework can be expanded to video-to-video translation [Ban+18; Wan+18], encompassing applications such as motion transfer [Cha+19a].

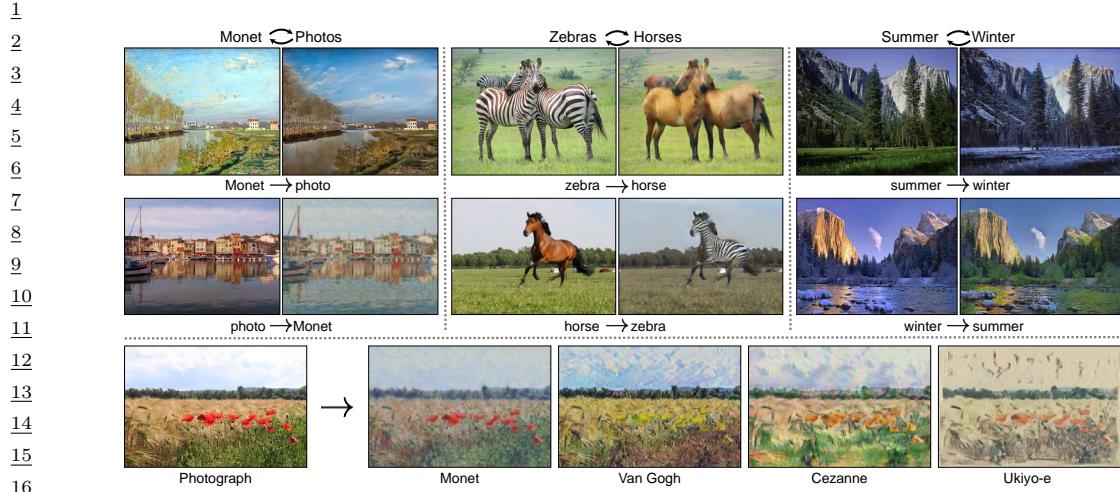


Figure 27.18: Some examples of unpaired image-to-image translation generated by the CycleGAN model. From Figure 1 of [Zhu+17]. Used with kind permission of Jun-Yan Zhu.

20

27.7.3 Audio generation

23 Generative models have been demonstrated in the tasks of generating audio waveforms, as well
 24 as for the task of text-to-speech (TTS) generation. Other types of generative models, such as
 25 autoregressive models, such as WaveNet [oor+16] and WaveRNN [Kal+18b] have been developed for
 26 these applications, although autoregressive models are difficult to parallelize over time since they
 27 predict each time step of the audio sequentially and can be computationally expensive and too slow
 28 to be used in practice. GANs provide an alternative approach for these tasks and other paths for
 29 addressing these concerns.

30 Many different GAN architectures have been developed for audio-only generation, including
 31 generation of single note recordings from instruments by GANSynth, a vocoder model that uses
 32 GANs to generate magnitude spectrograms from mel-spectrograms [Eng+18], in voice conversion
 33 using a modified CycleGAN discussed above [Kan+20], and the direct generation of raw audio in
 34 WaveGAN [DMP18].

35 Initial work on GANs for TTS was developed [Yan+17] whose approach is similar to conditional
 36 GANs for image generation (see Section 27.7.1.2), but uses 1d convolution instead of 2d. More
 37 recent GANs such as GAN-TTS [Biñ+19] use more advanced architectures and discriminators that
 38 operate at multiple frequency scales that have performance that now matches the best performing
 39 autoregressive models when assessed using mean opinion scores. In both the direct-audio generation,
 40 the ability of GANs to allow faster generation and different types of context is the advantage that
 41 makes them advantageous compared to other models.

42

27.7.4 Text generation

45 Similar to image and audio domains, there are several tasks for text data for which GAN-based
 46 approaches have been developed, including conditional text generation and text-style transfer. Text
 47

1 data are often represented as discrete values, at either the character level or the word-level, indicating
2 membership within a set of a particular vocabulary size (alphabet size, or number of words). Due to
3 the discrete nature of text, GAN models trained on text are *explicit*, since they explicitly model the
4 probability distribution of the output, rather than modeling the sampling path. This is unlike most
5 GAN models of continuous data such as images that we have discussed in the chapter so far, though
6 explicit GANs of continuous data do exist [Die+19b].

7 The discrete nature of text is why maximum likelihood is one of the most common methods of
8 learning generative models of text. However, models trained with maximum likelihood are often
9 limited to autoregressive models, while like in the audio case, GANs make it possible to generate
10 text in a non-autoregressive manner, making other tasks possible, such as one-shot feedforward
11 generation [Gul+17].

12 The difficulty of generating discrete data such as text using GANs can be seen looking at their
13 loss function - examples in Equations (27.19), (27.21) and (27.28). GAN losses contain terms of
14 the form $\mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x})$, which we not only need to evaluate, but also backpropagate through, by
15 computing $\nabla_\theta \mathbb{E}_{q_\theta(\mathbf{x})} f(\mathbf{x})$. In the case of implicit distributions given by latent variable models, we
16 used the reparametrization trick to compute this gradient (Equation 27.49). In the discrete case, the
17 reparametrization trick is not available and we have to look for other ways to estimate the desired
18 gradient. One approach is to use the score function estimator, discussed in Section 6.6.3. However,
19 the score function estimator exhibits high gradient variance, which can destabilize training. One
20 common approach to avoid this issue is to pre-train the language model generator using maximum
21 likelihood, and then to fine-tune with a GAN loss which gets backpropagated into the generator
22 using the score-function estimator, as done by Sequence GAN [Yu+17], MaliGAN [Che+17], and
23 RankGAN [Lin+17]. While these methods spearheaded the use of GANs for text, they do not
24 address the inherent instabilities of score function estimation and thus have to limit the amount of
25 adversarial fine tuning to a small number of epochs and often use a small learning rate, keeping their
26 performance close to that of the maximum-likelihood solution [SSG18a; Cac+18].

27 An alternative to maximum likelihood pretraining is to use other approaches to stabilize the score
28 function estimator or to use continuous relaxations for backpropagation. ScratchGAN is a word-level
29 model that uses large batch sizes and discriminator regularization to stabilize score function training
30 (these techniques are the same that we have seen as stabilizers for training image GANs) [Md+19].
31 [Pre+17] completely avoid the score function estimator and develop a character level model without
32 pre-training, by using continuous relaxations and curriculum learning. These training approaches
33 can also benefit from other architectural advances, e.g., [NNP19] showed that language GANs can
34 benefit from complex architectures such as Relation Networks [San+17].

35 Finally, unsupervised text style transfer, mimicking image style transfer, have been proposed by
36 [She+17; Fu+17] using adversarial classifiers to decode to a different style/language, or like [Pra+18]
37 who trains different encoders, one per style, by combining the encoder of a pre-trained NMT and
38 style classifiers, among other approaches.

40 27.7.5 Imitation Learning

41 Imitation learning takes advantage of observations of expert demonstrations to learn action policies
42 and reward functions of unknown environments by minimizing some form of discrepancy between
43 learned and the expert behaviors. There are many approaches available, including behavioral
44 cloning [PPG91] that treats this problem as one of supervised learning, and inverse reinforcement
45

1 learning [NR00b]. GANs are appealing for imitation learning since they provide a way to avoid the
2 difficulty of designing good discrepancy functions for behaviors, and instead learn these discrepancy
3 functions using a discriminator between trajectories generated by a learned agent and observed
4 demonstrations.
5

6 This approach, known as Generative Adversarial Imitation Learning (GAIL) [HE16a] demonstrates
7 the ability to use GANs for complex behaviors in high-dimensional environments. GAIL jointly
8 learns a generator, which forms a stochastic policy, along with a discriminator that acts as a reward
9 signal. Like we saw in the probabilistic development of GANs in the earlier sections, GAIL can
10 also be generalized to multiple f -divergences, rather than the standard Jensen-Shannon divergence
11 used as the standard loss in GANs. This has lead to a family of other GAIL variants that use
12 other f -divergences [Ke+19a; Fin+16; Bro+20a], including f -GAIL that aims to also learn the
13 best f -divergence to use [Zha+20e], as well as new analytical insight into the computation and
14 generalization of such approaches [Che+20a].
15

16 27.7.6 Domain Adaptation

17 An important task in machine learning is to correct for shifts in the data distribution over time,
18 minimizing some measure of domain shift, as we discuss in Section 20.3.2. Like with the other
19 applications, GANs are popular as ways of avoiding the choice of distance or degree of shift.
20 Both the supervised and unsupervised approaches for image generation we reviewed earlier looked
21 at pixel-level domain adaptation models that perform distribution alignment in raw pixel space,
22 translating source data to the style of a target domain, as with pix2pix and CycleGAN. Extensions
23 of these approaches for the general problem of domain adaptation seek to do this not only in the
24 observed data space (e.g., with pixels), but also at the feature level. One general approach is
25 domain-adversarial training of neural networks [Gan+16b] or adversarial discriminative domain
26 adaptation (ADDA) [Tze+17]; The CyCADA approach of [Hof+18] extends CycleGAN by enforcing
27 both structural and semantic consistency during adaptation using a cycle-consistency loss and
28 semantics losses based on a particular visual recognition task. There are also many extensions that
29 include class conditional information [Tsa+18; Lon+18] or adaptation when the modes to be matched
30 have different frequencies in the source and target domains [BHC19].
31

32 27.7.7 Design, Art and Creativity

33 Generative models, particularly of images, have added to approaches in the more general area of
34 algorithmic art. The applications in image and audio generation with transfer, can also be considered
35 aspects of artistic image generation. In these cases, the goal of training is not generalization, but to
36 create appealing images across different types of visual aesthetics [Sar18]. One example takes style
37 transfer GANs to create visual experiences, in which objects placed under a video are re-rendered
38 using other visual styles in real time [AFG19]. The generation ability has been used to explore
39 alternative designs and fabrics in fashion [Kat+19], and have now also become part of major drawing
40 software to provide new tools to support designers [Ado]. And beyond images, creative and artistic
41 expression using GANs include areas in music, voice, dance, and typography [AI 19].
42

43
44
45
46
47

Part V

Discovery

28 Discovery methods: an overview

28.1 Introduction

We have seen in Part III how to create probabilistic models that can make predictions about outputs given inputs, using supervised learning methods (conditional likelihood maximization). And we have seen in Part IV how to create probabilistic models that can generate outputs unconditionally, using unsupervised learning methods (unconditional likelihood maximization). However, in some settings, our goal is to try to *understand* a given dataset. That is, we want to *discover* something “interesting”, and possibly “actionable”. Prediction and generation are useful subroutines for discovery, but are not sufficient on their own. In particular, although neural networks often implicitly learn useful features from data, they are often hard to interpret, and the results can be unstable and sensitive to arbitrary details of the training protocol (e.g., SGD learning rates, or random seeds).

In this part of the book, we focus on learning models that create an interpretable representation of high dimensional data. A common approach is to use a **latent variable model**, in which we make the assumption that the observed data \mathbf{x} was caused by, or generated by, some underlying (often low dimensional) **latent factors** \mathbf{z} , which represents the “true” state of the world. Crucially, these latent variables are assumed to be meaningful to the end user of the model. (Thus evaluating such models will generally require domain expertise.)

For example, suppose we want to interpret an image \mathbf{x} in terms of an underlying 3d scene, \mathbf{z} , which is represented in terms of objects and surfaces. The **forwards mapping** from \mathbf{z} to \mathbf{x} is often many-to-one, i.e., different latent values, say \mathbf{z} and \mathbf{z}' , may give rise to the same observation \mathbf{x} , due to limitations of the sensor. (This is called **perceptual aliasing**.) Consequently the inverse mapping, from \mathbf{x} to \mathbf{z} , is ill-posed. In such cases, we need to impose a prior, $p(\mathbf{z})$, to make our estimate well-defined. In simple settings, we can use a point estimate, such as the MAP estimate

$$\hat{\mathbf{z}}(\mathbf{x}) = \underset{\mathbf{z}}{\operatorname{argmax}} p(\mathbf{z}|\mathbf{x}) = \underset{\mathbf{z}}{\operatorname{argmax}} \log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z}) \quad (28.1)$$

In the context of computer vision, this approach is known as **vision as inverse graphics** or **analysis by synthesis** [KMY04; YK06; Doy+07; MC19]. See Figure 28.1 for an illustration.

This approach to inverse modeling is widely used in science and engineering, where \mathbf{z} represents the underlying state of the world which we want to estimate, and \mathbf{x} is just a noisy or partial manifestation of this true state. In some cases, we know both the prior $p(\mathbf{z}|\boldsymbol{\theta})$ and the likelihood $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$, and we just need to solve the inference problem for \mathbf{z} . But more commonly, the model parameters $\boldsymbol{\theta}$ are also (partially) unknown, and need to be inferred from observable samples $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$. In some cases, the structure of the model itself is unknown and needs to be learned.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

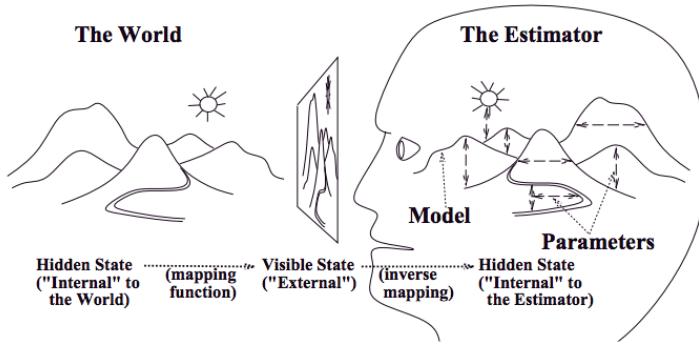


Figure 28.1: Vision as inverse graphics. The agent (here represented by a human head) has to infer the scene \mathbf{z} given the image \mathbf{x} using an estimator. From Figure 1 of [Rao99]. Used with kind permission of Rajesh Rao.

28.2 Overview of Part V

In Chapter 29, we discuss simple latent variable models where typically the observed data is a fixed-dimensional vector such as $\mathbf{x} \in \mathbb{R}^D$. In Chapter 30 and Chapter 31 we extend these models to work with sequences of correlated vectors, $\mathbf{x} = \mathbf{x}_{1:T}$, such as speech, video, genomics data, etc. It is straightforward to make parts of these model be nonlinear (“deep”), as we discuss. These models can also be extended to the spatio-temporal setting.

The models in Chapters 29 to 31 can all be interpreted as probabilistic graphical models with different kinds of CPDs. In Chapter 32, we discuss how to learn the structure of PGMs from data. In Chapter 33, we discuss non-parametric Bayesian models, which allow us to represent uncertainty about many aspects of a model, such as the number of hidden states, the structure of the model, the form of a functional dependency, etc. Thus the complexity of the learned representation can grow dynamically, depending on the quantity and quality (informativeness) of the data. This is important when performing discovery tasks, and helps us maintain flexibility while still retaining interpretability.

In Chapter 34, we discuss representation learning using neural networks. This can be tackled using latent variable modeling, but there are also a variety of other estimation methods one can use. Finally, in Chapter 35, we discuss how to interpret the behavior of a learned (prediction) model (typically a neural network).

29 Latent variable models

29.1 Introduction

A **latent variable model (LVM)** is any probabilistic model in which some variables are always latent or hidden. A simple example is a mixture model (Section 29.2), which has the form $p(\mathbf{x}) = \sum_k p(\mathbf{x}|z=k)p(z=k)$, where z is an indicator variable that specifies which mixture component to use for generating \mathbf{x} . However, we can also use continuous latent variables, or a mixture of discrete and continuous. And we can also have multiple latent variables, which are interconnected in complex ways.

In this chapter, we discuss a very simple kind of LVM that has the following form:

$$\mathbf{z} \sim p(\mathbf{z}) \tag{29.1}$$

$$\mathbf{x}|\mathbf{z} \sim \text{Expfam}(\mathbf{x}|f(\mathbf{z})) \tag{29.2}$$

where $f(\mathbf{z})$ is known as the **decoder**, and $p(\mathbf{z})$ is some kind of prior. We assume that \mathbf{z} is a single “layer” of hidden random variables, corresponding to a set of “latent factors”. We will also mostly assume that the decoder f is a simple linear model. Thus the overall model is similar to a GLM (Section 15.1), except the input to the model is hidden.

We can create a large variety of different “classical” models by changing the form of the prior $p(\mathbf{z})$ and/or the likelihood $p(\mathbf{x}|\mathbf{z})$, as we show in Table 29.1. We will give the details in the following sections. (Note that, although we are discussing generative models, our focus is on posterior inference of meaningful latents (discovery), rather than generating realistic samples of data.)

29.2 Mixture models

One way to create more complex probability models is to take a convex combination of simple distributions. This is called a **mixture model**. This has the form

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K \pi_k p_k(\mathbf{x}) \tag{29.3}$$

where p_k is the k 'th mixture component, and π_k are the mixture weights which satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^K \pi_k = 1$.

We can re-express this model as a hierarchical model, in which we introduce the discrete **latent variable** $z \in \{1, \dots, K\}$, which specifies which distribution to use for generating the output \mathbf{x} . The

Model	$p(\mathbf{z})$	$p(\mathbf{x} \mathbf{z})$	Section
FA/PCA	$\mathcal{N}(\mathbf{z} \mathbf{0}, \mathbf{I})$	$\mathcal{N}(\mathbf{x} \mathbf{W}\mathbf{z}, \boldsymbol{\Psi})$	Section 29.3.1
GMM	$\sum_c \text{Cat}(c \boldsymbol{\pi})\mathcal{N}(\mathbf{z} \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$	$\mathcal{N}(\mathbf{x} \mathbf{W}\mathbf{z}, \boldsymbol{\Psi})$	Section 29.2.4
MixFA	$\text{Cat}(c \boldsymbol{\pi})\mathcal{N}(\mathbf{z} \mathbf{0}, \mathbf{I})$	$\mathcal{N}(\mathbf{x} \mathbf{W}_c\mathbf{z} + \boldsymbol{\mu}_c, \boldsymbol{\Psi}_c)$	Section 29.4.3
NMF	$\prod_k \text{Ga}(z_k \alpha_k, \beta_k)$	$\prod_d \text{Poi}(x_d \exp(\mathbf{w}_d^\top \mathbf{z}))$	Section 29.5.1
Simplex FA (mPCA)	$\text{Dir}(\mathbf{z} \boldsymbol{\alpha})$	$\prod_d \text{Cat}(x_d \mathbf{W}_d\mathbf{z})$	Section 29.5.2
LDA	$\text{Dir}(\mathbf{z} \boldsymbol{\alpha})$	$\prod_d \text{Cat}(x_d \mathbf{W}\mathbf{z})$	Supplementary
ICA	$\prod_d \text{Lap}(z_d \lambda)$	$\prod_d \delta(x_d - \mathbf{w}_d^\top \mathbf{z})$	Section 29.6
Sparse coding	$\prod_k \text{Lap}(z_k \lambda)$	$\prod_d \mathcal{N}(x_d \mathbf{w}_d^\top \mathbf{z}, \sigma^2)$	Section 29.6.5

Table 29.1: Some popular “shallow” latent factor models. Abbreviations: FA = factor analysis, PCA = principal components analysis, GMM = Gaussian mixture model, NMF = non-negative matrix factorization, mPCA = multinomial PCA, LDA = latent Dirichlet allocation, ICA = independent components analysis. $k = 1 : L$ ranges over latent dimensions, $d = 1 : D$ ranges over observed dimensions. (For ICA, we have the constraint that $L = D$.)

16

17

18 prior on this latent variable is $p(z = k) = \pi_k$, and the conditional is $p(\mathbf{x}|z = k) = p_k(\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta}_k)$.
19 That is, we define the following joint model:

20

$$21 \quad p(z|\boldsymbol{\theta}) = \text{Cat}(z|\boldsymbol{\pi}) \quad (29.4)$$

$$22 \quad p(\mathbf{x}|z = k, \boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta}_k) \quad (29.5)$$

23

24 The “generative story” for the data is that we first sample a specific component z , and then we
25 generate the observations \mathbf{x} using the parameters chosen according to the value of z . By marginalizing
26 out z , we recover Equation (29.3):

27

$$28 \quad p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K p(z = k|\boldsymbol{\theta})p(\mathbf{x}|z = k, \boldsymbol{\theta}) = \sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\theta}_k) \quad (29.6)$$

30
31 We can create different kinds of mixture model by varying the base distribution p_k , as we illustrate
32 below.

33

34 29.2.1 Gaussian mixture models (GMMs) 35

36 A **Gaussian mixture model** or **GMM**, also called a **mixture of Gaussians (MoG)**, is defined
37 as follows:

38

$$39 \quad p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (29.7)$$

40

41 In Figure 29.1 we show the density defined by a mixture of 3 Gaussians in 2d. Each mixture
42 component is represented by a different set of elliptical contours. If we let the number of mixture
43 components grow sufficiently large, a GMM can approximate any smooth distribution over \mathbb{R}^D .

44 GMMs are often used for unsupervised **clustering** of real-valued data samples $\mathbf{x}_n \in \mathbb{R}^D$. This
45 works in two stages. First we fit the model e.g., by computing the MLE $\hat{\boldsymbol{\theta}} = \text{argmax } \log p(\mathcal{D}|\boldsymbol{\theta})$, where
46

47

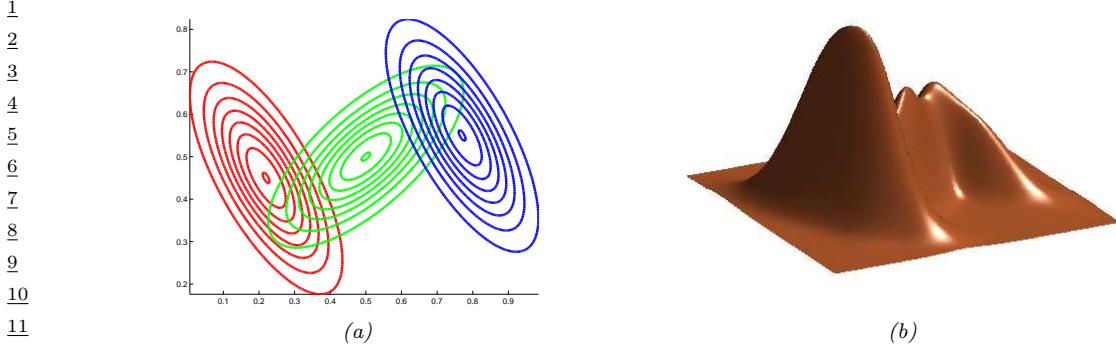


Figure 29.1: A mixture of 3 Gaussians in 2d. (a) We show the contours of constant probability for each component in the mixture. (b) A surface plot of the overall density. Adapted from Figure 2.23 of [Bis06]. Generated by [gmm_plot_demo.py](#).

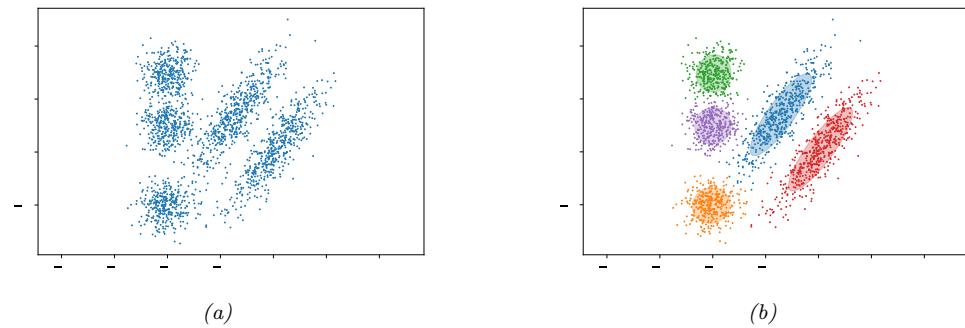


Figure 29.2: (a) Some data in 2d. (b) A possible clustering using $K = 3$ clusters computed using a GMM. Generated by [gmm_2d.py](#).

$\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$ (e.g., using EM or SGD). Then we associate each data point \mathbf{x}_n with a discrete latent or hidden variable $z_n \in \{1, \dots, K\}$ which specifies the identity of the mixture component or cluster which was used to generate \mathbf{x}_n . These latent identities are unknown, but we can compute a posterior over them using Bayes rule:

$$r_{nk} \triangleq p(z_n = k | \mathbf{x}_n, \boldsymbol{\theta}) = \frac{p(z_n = k | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta})}{\sum_{k'=1}^K p(z_n = k' | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k', \boldsymbol{\theta})} \quad (29.8)$$

The quantity r_{nk} is called the **responsibility** of cluster k for data point n . Given the responsibilities, we can compute the most probable cluster assignment as follows:

$$\hat{z}_n = \arg \max_k r_{nk} = \arg \max_k [\log p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta}) + \log p(z_n = k | \boldsymbol{\theta})] \quad (29.9)$$

This is known as **hard clustering**. (If we use the responsibilities to fractionally assign each data point to different clusters, it is called **soft clustering**.) See Figure 29.2 for an example.

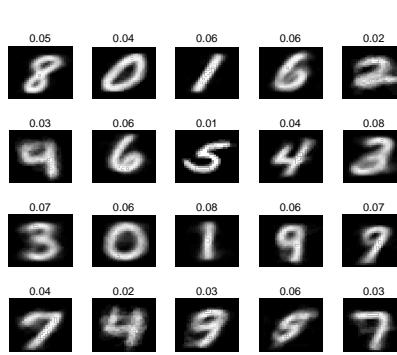


Figure 29.3: We fit a mixture of 20 Bernoullis to the binarized MNIST digit data. We visualize the estimated cluster means $\hat{\mu}_k$. The numbers on top of each image represent the estimated mixing weights $\hat{\pi}_k$. No labels were used when training the model. Generated by `mix_ber_em_mnist.py`.

If we have a uniform prior over z_n , and we use spherical Gaussians with $\Sigma_k = \mathbf{I}$, the hard clustering problem reduces to

$$z_n = \operatorname{argmin}_k \|\mathbf{x}_n - \hat{\mu}_k\|_2^2 \quad (29.10)$$

In other words, we assign each data point to its closest centroid, as measured by Euclidean distance. This is the basis of the K-means clustering algorithm (see the prequel to this book).

29.2.2 Bernoulli mixture models

If the data is binary valued, we can use a **Bernoulli mixture model** (BMM), also called a **mixture of Bernoullis**, where each mixture component has the following form:

$$p(\mathbf{x}|z=k, \boldsymbol{\theta}) = \prod_{d=1}^D \text{Ber}(y_d|\mu_{dk}) = \prod_{d=1}^D \mu_{dk}^{y_d} (1-\mu_{dk})^{1-y_d} \quad (29.11)$$

Here μ_{dk} is the probability that bit d turns on in cluster k .

For example, consider fitting a mixture of Bernoullis using $K = 20$ components to the MNIST dataset. The resulting parameters for each mixture component (i.e., μ_k and π_k) are shown in Figure 29.3. We see that the model has “discovered” a representation of each type of digit. (Some digits are represented multiple times, since the model does not know the “true” number of classes. See Section 3.7.1 for information on how to choose the number K of mixture components.)

29.2.3 Gaussian scale mixtures

A **Gaussian scale mixture** of **GSM** [AM74; Wes87] is like an “infinite” mixture of Gaussians, each with a different scale (variance). More precisely, let $x = \epsilon z$, where $z \sim \mathcal{N}(0, \sigma_0^2)$ and $\epsilon \sim p(\epsilon)$. We can

1 think of this as **multiplicative noise** being applied to the Gaussian rv z . We have $x|\epsilon \sim \mathcal{N}(0, \epsilon^2 \sigma_0^2)$.
2 Marginalizing out the scale ϵ gives
3

4

$$\underline{p(x) = \int \mathcal{N}(x|0, \sigma_0^2 \epsilon^2) p(\epsilon^2) d\epsilon} \quad (29.12)$$

5

6 By changing the prior $p(\epsilon)$, we can create various interesting distributions. We give some examples
7 below.
8

9 The main advantage of this approach is that it is often computationally more convenient to
10 work with the **expanded parameterization**, in which we explicitly include the scale term ϵ , since,
11 conditional on that, the distribution is Gaussian. We use this formulation in Section 6.7.5, where we
12 discuss robust regression.
13

14 29.2.3.1 Student t distribution as GSM

15

16 We can represent the Student distribution as a GSM as follows:

17

$$\underline{\mathcal{T}(y|0, \sigma^2, \nu) = \int_0^\infty \mathcal{N}(y|0, z\sigma^2) \text{IG}(z|\frac{\nu}{2}, \frac{\nu}{2}) dz = \int_0^\infty \mathcal{N}(y|0, z\sigma^2) \chi^{-2}(z|\nu, 1) dz} \quad (29.13)$$

18

19 where IG is the inverse Gamma distribution (Section 2.2.8). Thus we can think of the Student as an
20 infinite superposition of Gaussians of different widths; marginalizing this out induces a distribution
21 with wider tails than a Gaussian with the same variance. This result also explains why the Student
22 distribution approaches a Gaussian as the dof gets large, since when $\nu = \infty$, the inverse Gamma
23 distribution becomes a delta function.
24

25 29.2.3.2 Laplace distribution as GSM

26

27 Similarly one can show that the Laplace distribution is an infinite weighted sum of Gaussians, where
28 the precision comes from a Gamma distribution:
29

30

$$\underline{\text{Lap}(w|0, \lambda) = \int \mathcal{N}(w|0, \tau^2) \text{Ga}(\tau^2|1, \frac{\lambda^2}{2}) d\tau^2} \quad (29.14)$$

31

32 29.2.3.3 Spike and slab distribution

33

34 Suppose $\epsilon \sim \text{Ber}(\pi)$. Note that $p(\epsilon^2) = p(\epsilon)$, since $\epsilon \in \{0, 1\}$. In this case we have
35

36

$$\underline{x = \sum_{\epsilon \in \{0,1\}} \mathcal{N}(x|0, \sigma_0^2 \epsilon) p(\epsilon) = \pi \mathcal{N}(x|0, \sigma_0^2) + (1 - \pi) \delta_0(x)} \quad (29.15)$$

37

38 This is known as the **spike and slab** distribution, since the $\delta_0(x)$ is a “spike” at 0, and the $\mathcal{N}(x|0, \sigma_0^2)$
39 acts like a uniform “slab” for large enough σ_0 . This distribution is useful in sparse modeling.
40

41 29.2.3.4 Horseshoe distribution

42

43 Suppose $\epsilon \sim \mathcal{C}_+(1)$, which is the half-Cauchy distribution (see Section 2.2.4). Then the induced
44 distribution $p(x)$ is called the **horseshoe distribution** [CPS10]. This has a spike at 0, like the
45

1

(a)

2

(b)

3

(c)

4

(d)

5

(e)

6

(f)

7891011121314151617181920212223242526272829

Figure 29.4: Example of recovering a clean image (right) from a corrupted version (left) using MAP estimation with a GMM patch prior and Gaussian likelihood. First row: image denoising. Second row: image deblurring. Third row: image inpainting. From [RW15] and [ZW11]. Used with kind permission of Dan Rosenbaum and Daniel Zoran.

303132333435

Student and Laplace distributions, but has heavy tails that do not asymptote to zero. This makes it useful as a sparsity promoting prior, that “kills off” small parameters, but does not overregularize large parameters.

36

In this section, we consider using GMMs as a blackbox density model to regularize the inversion of a many-to-one mapping. Specifically, we consider the problem of inferring a “clean” image \mathbf{x} from a corrupted version \mathbf{y} . We use a linear-Gaussian forwards model of the form

37

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{W}\mathbf{x}, \sigma^2 \mathbf{I}) \quad (29.16)$$

38

where σ^2 is the variance of the measurement noise. The form of the matrix \mathbf{W} depends on the nature of the corruption, which we assume is known, for simplicity. Here are some common examples of different kinds of corruption we can model in our approach:

- If the corruption is due to additive noise (as in Figure 29.4a), we can set $\mathbf{W} = \mathbf{I}$. The resulting MAP estimate can be used for **image denoising**, as in Figure 29.4b.
- If the corruption is due to blurring (as in Figure 29.4c), we can set \mathbf{W} to be a fixed convolutional kernel [KF09b]. The resulting MAP estimate can be used for **image deblurring**, as in Figure 29.4d.
- If the corruption is due to occlusion (as in Figure 29.4e), we can set \mathbf{W} to be a diagonal matrix, with 0s in the locations corresponding to the occluders. The resulting MAP estimate can be used for **image inpainting**, as in Figure 29.4f.
- If the corruption is due to downsampling, we can set \mathbf{W} to a convolutional kernel. The resulting MAP estimate can be used for **image super-resolution**.

Thus we see that the linear-Gaussian likelihood model is surprisingly flexible. Given the model, our goal is to invert it, by computing the MAP estimate $\hat{\mathbf{x}} = \text{argmax } p(\mathbf{x}|\mathbf{y})$. However, the problem of inverting this model is ill-posed, since there are many possible latent images \mathbf{x} that map to the same observed image \mathbf{y} . Therefore we need to use a prior to regularize the inversion process.

In [ZW11], they propose to partition the image into patches, and to use a GMM prior of the form $p(\mathbf{x}_i) = \sum_k p(c_i = k) \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ for each patch i . They use $K = 200$ mixture components, and they fit the GMM on a dataset of 2M clean image patches.

To compute the MAP mixture component, c_i^* , we can marginalize out \mathbf{x}_i and use Equation (2.62) to compute the marginal likelihood

$$c_i^* = \underset{c}{\text{argmax}} p(c) p(\mathbf{y}_i | c) = \underset{c}{\text{argmax}} p(c) \mathcal{N}(\mathbf{y}_i | \mathbf{W}\boldsymbol{\mu}_c, \sigma^2 \mathbf{I} + \mathbf{W}\boldsymbol{\Sigma}_c\mathbf{W}^\top) \quad (29.17)$$

We can then approximate the MAP for the latent patch \mathbf{x}_i by using the approximation

$$p(\mathbf{x}_i | \mathbf{y}_i) \approx p(\mathbf{x}_i | \mathbf{y}_i, c_i^*) \propto \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_{c_i^*}, \boldsymbol{\Sigma}_{c_i^*}) \mathcal{N}(\mathbf{y}_i | \mathbf{W}\mathbf{x}_i, \sigma^2 \mathbf{I}) \quad (29.18)$$

If we know c_i^* , we can compute the above using Bayes rule for Gaussians in Equation (2.59).

To apply this method to full images, [ZW11] optimize the following objective

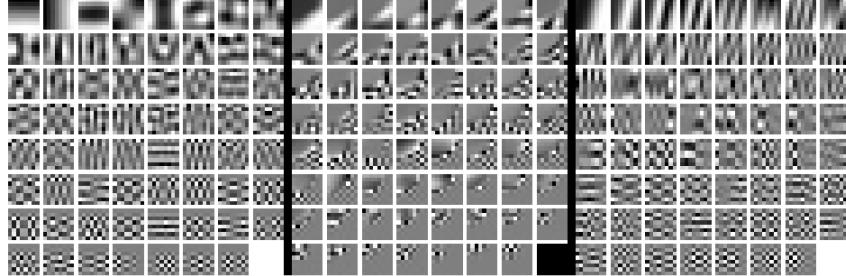
$$E(\mathbf{x} | \mathbf{y}) = \frac{1}{2\sigma^2} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|^2 - \text{EPPL}(\mathbf{x}) \quad (29.19)$$

where **EPPL** is the “expected patch log likelihood”, given by

$$\text{EPPL}(\mathbf{x}) = \sum_i \log p(\mathbf{P}_i \mathbf{x}) \quad (29.20)$$

where $\mathbf{x}_i = \mathbf{P}_i \mathbf{x}$ is the i 'th patch computed by projection matrix \mathbf{P}_i . Since these patches overlap, this is not a valid likelihood, since it overcounts the pixels. Nevertheless, optimizing this objective (using a method called “half quadratic splitting”) works well empirically. See Figure 29.4 for some examples of this process in action.

1
2
3
4
5
6
7
8
9
10



11 *Figure 29.5: Illustration of the parameters learned by a GMM applied to image patches. Each of the 3 panels*
 12 *corresponds to a different mixture component k . Within each panel, we show the eigenvectors (reshaped as*
 13 *images) of the covariance matrix Σ_k in decreasing order of eigenvalue. We see various kinds of patterns,*
 14 *including ones that look like the ones learned from PCA (see Figure 29.25), but also ones that look like edges*
 15 *and texture. From Figure 6 of [ZW11]. Used with kind permission of Daniel Zoran.*

16
17

18 A more principled solution to the overlapping patch problem is to use a multiscale model, as
 19 proposed in [PE16]. Another approach, proposed in [FW21], uses Gibbs sampling to combine samples
 20 from overlapping patches. This approach has the additional advantage of computing posterior samples
 21 from $p(\mathbf{x}|\mathbf{y})$, which can look much better than the posterior mean or mode computed by optimization
 22 methods. (For example, if the corruption process removes the color from the latent image \mathbf{x} to
 23 create a gray scale \mathbf{y} , then the posterior MAP estimate of \mathbf{x} will also be a gray scale image, whereas
 24 posterior samples will be color images.) See also Section 29.4.3 where we show how to extend the
 25 GMM model to a mixture of low rank Gaussians, which lets us directly model images instead of
 26 image patches.
 27

28 **29.2.4.1 Why does the method work?**
 29

30 To understand why such a simple model of image patches works so well, note that the log prior for a
 31 single latent image patch \mathbf{x}_i using mixture component k can be written as follows:
 32

$$33 \quad \log p(\mathbf{x}_i|c_n = k) = \log \mathcal{N}(\mathbf{x}_i|\mathbf{0}, \Sigma_k) = -\mathbf{x}_i^\top \Sigma_k^{-1} \mathbf{x}_i + a_k \quad (29.21)$$

34 where a_k is a constant that depends on k but is independent of \mathbf{x}_i . Let $\Sigma_k = \mathbf{V} \Lambda_k \mathbf{V}^\top$ be an
 35 eigendecomposition of Σ_k , where $\lambda_{k,d}$ is the d 'th eigenvalue of Σ_k , and $\mathbf{v}_{k,d}$ is the d 'th eigenvector.
 36 Then we can rewrite the above as follows:
 37

$$38 \quad \log p(\mathbf{x}_i|c_n = k) = -\sum_{d=1}^D \frac{1}{\lambda_{k,d}} (\mathbf{v}_{k,d}^\top \mathbf{x}_i)^2 + a_k \quad (29.22)$$

41 Thus we see that the eigenvectors are acting like templates. Each mixture component has a different
 42 set of templates, each with their own weight (eigenvalue), as illustrated in Figure 29.5. By mixing
 43 these together, we get a powerful model for the statistics of natural image patches. (See [ZW12] for
 44 more analysis of why this simple model works so well, based on the “dead leaves” model of image
 45 formation.)
 46

47

1 **29.2.4.2 Speeding up inference using discriminative models**

2 Although simple and effective, computing $f(\mathbf{y}) = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$ for each image patch can be slow
3 if the image is large. However, every time we solve this problem, we can store the result, and build
4 up a dataset of $(\mathbf{y}, f(\mathbf{y}))$ pairs. We can then train an amortized inference network (Section 10.3.7)
5 to learn this $\mathbf{y} \rightarrow f(\mathbf{y})$ mapping, to speed up future inferences, as proposed in [RW15]. (See also
6 [Par+19] for further speedup tricks.)
7

8 An alternative approach is to dispense with the generative model, and to train on an artificially
9 created dataset of the form (\mathbf{y}, \mathbf{x}) , where \mathbf{x} is a clean natural image, and $\mathbf{y} = C(\mathbf{x})$ is an artificial
10 corruption of it. We can then train a discriminative model $\hat{f}(\mathbf{y})$ directly from (\mathbf{y}, \mathbf{x}) pairs. This
11 technique works very well (see e.g., [Luc+18]), but is limited by the form of corruptions C it is trained
12 on. This means we need to train a different network for every linear operator \mathbf{W} , and sometimes
13 even for every different noise level σ^2 .
14

15 **29.2.4.3 Blind inverse problems**

16 In the discussion above, we assumed the forward model had the form $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}|\mathbf{W}\mathbf{x}, \sigma^2\mathbf{I})$,
17 where \mathbf{W} is known. If \mathbf{W} is not known, then computing $p(\mathbf{x}|\mathbf{y})$ is known as a **blind inverse**
18 **problem.**

19 Such problems are much harder to solve. One approach is to estimate the parameters of the
20 forwards model, \mathbf{W} , and the latent image, \mathbf{x} , using an EM-like method from a set of images coming
21 from the same likelihood function. That is, we alternate between estimating $\hat{\mathbf{x}} = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}, \hat{\mathbf{W}})$
22 in the E step, and estimating $\hat{\mathbf{W}} = \operatorname{argmax}_{\mathbf{W}} p(\mathbf{y}|\hat{\mathbf{x}}, \mathbf{W})$ in the M step. Some encouraging results of
23 this approach are shown in [Ani+18]. (They use a GAN prior for $p(\mathbf{x})$ rather than a GMM.)
24

25 In cases where we get two independent noisy samples, \mathbf{y}_1 and \mathbf{y}_2 , generated from the same
26 underlying image \mathbf{x} , then we can avoid having to explicitly learn an image prior $p(\mathbf{x})$, and can instead
27 directly learn an estimator for the posterior mode, $f(\mathbf{y}) = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$, without needing access
28 to the latent image \mathbf{x} , by exploiting a form of cycle consistency; see [XC19] for details.
29

30 **29.3 Factor analysis**

31 In this section, we discuss a simple latent factor model in which the prior $p(\mathbf{z})$ is Gaussian, and the
32 likelihood $p(\mathbf{x}|\mathbf{z})$ is also Gaussian, using a linear decoder $f(\mathbf{z})$ for the mean. This family includes
33 many important special cases, such as PCA, as we discuss below.
34

35 **29.3.1 Vanilla factor analysis**

36 Factor analysis corresponds to the following linear-Gaussian latent variable generative model:
37

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \tag{29.23}$$

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \tag{29.24}$$

38 where \mathbf{W} is a $D \times L$ matrix, known as the **factor loading matrix**, and $\boldsymbol{\Psi}$ is a diagonal $D \times D$
39 covariance matrix.
40

1 **29.3.1.1 FA as a Gaussian with low-rank plus diagonal covariance**

3 FA can be thought of as a low-rank version of a Gaussian distribution. To see this, note that the
4 induced marginal distribution $p(\mathbf{x}|\boldsymbol{\theta})$ is a Gaussian (see Equation (2.62) for the derivation):
5

$$\underline{6} \quad p(\mathbf{x}|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) d\mathbf{z} \quad (29.25)$$

$$\underline{7} \quad = \mathcal{N}(\mathbf{x}|\mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}, \boldsymbol{\Psi} + \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T) \quad (29.26)$$

9 The first and second moments can be derived as follows:
10

$$\underline{11} \quad \mathbb{E}[\mathbf{x}] = \mathbf{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu} \quad (29.27)$$

$$\underline{12} \quad \text{Cov}[\mathbf{x}] = \mathbf{W}\text{Cov}[\mathbf{z}]\mathbf{W}^T + \boldsymbol{\Psi} = \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T + \boldsymbol{\Psi}$$

13 From this, we see that we can set $\boldsymbol{\mu}_0 = \mathbf{0}$ without loss of generality, since we can always absorb
14 $\mathbf{W}\boldsymbol{\mu}_0$ into $\boldsymbol{\mu}$. Similarly, we can set $\boldsymbol{\Sigma}_0 = \mathbf{I}$ without loss of generality, since we can always absorb a
15 correlated prior by using a new weight matrix, $\tilde{\mathbf{W}} = \mathbf{W}\boldsymbol{\Sigma}_0^{-\frac{1}{2}}$, since then
16

$$\underline{17} \quad \text{Cov}[\mathbf{x}] = \mathbf{W}\boldsymbol{\Sigma}_0\mathbf{W}^T + \boldsymbol{\Psi} = \tilde{\mathbf{W}}\tilde{\mathbf{W}}^T + \boldsymbol{\Psi} \quad (29.28)$$

18 Finally, we see that we should restrict $\boldsymbol{\Psi}$ to be diagonal, otherwise we could set $\tilde{\mathbf{W}} = \mathbf{0}$, thus ignoring
19 the latent factors, while still being able to model any covariance. After these simplifications we have
20 the final model:
21

$$\underline{22} \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}) \quad (29.29)$$

$$\underline{23} \quad p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \quad (29.30)$$

$$\underline{25} \quad p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}) \quad (29.31)$$

26 For example, suppose where $L = 1$, $D = 2$ and $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$. We illustrate the generative process in
27 this case in Figure 29.6. We can think of this as taking an isotropic Gaussian “spray can”, representing
28 the likelihood $p(\mathbf{x}|\mathbf{z})$, and “sliding it along” the 1d line defined by $\mathbf{w}\mathbf{z} + \boldsymbol{\mu}$ as we vary the 1d latent
29 prior \mathbf{z} . This induces an elongated (and hence correlated) Gaussian in 2d. That is, the induced
30 distribution has the form $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{w}\mathbf{w}^T + \sigma^2\mathbf{I})$.
31

32 In general, FA approximates the covariance matrix of the visible vector using a low-rank decompo-
33 sition:
34

$$\mathbf{C} = \text{Cov}[\mathbf{x}] = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi} \quad (29.32)$$

35 This only uses $O(LD)$ parameters, which allows a flexible compromise between a full covariance
36 Gaussian, with $O(D^2)$ parameters, and a diagonal covariance, with $O(D)$ parameters.
37

38 **29.3.1.2 Computing the posterior**

39 We can compute the posterior over the latent codes, $p(\mathbf{z}|\mathbf{x})$, using Bayes rule for Gaussians. In
40 particular, from Equation (2.59), we have
41

$$\underline{43} \quad p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x}) \quad (29.33)$$

$$\underline{44} \quad \boldsymbol{\Sigma}_{z|x}^{-1} = \mathbf{I} + \mathbf{W}^T \boldsymbol{\Psi}^{-1} \mathbf{W} \quad (29.34)$$

$$\underline{45} \quad \boldsymbol{\mu}_{z|x} = \boldsymbol{\Sigma}_{z|x} [\mathbf{W}^T \boldsymbol{\Psi}^{-1} (\mathbf{x} - \boldsymbol{\mu})] \quad (29.35)$$

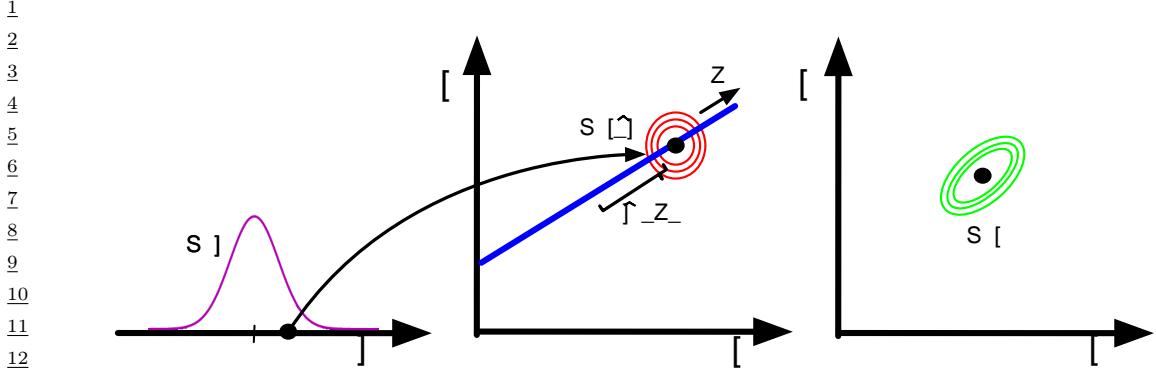


Figure 29.6: Illustration of the FA generative process, where we have $L = 1$ latent dimension generating $D = 2$ observed dimensions; we assume $\Psi = \sigma^2 \mathbf{I}$. The latent factor has value $z \in \mathbb{R}$, sampled from $p(z)$; this gets mapped to a 2d offset $\delta = zw$, where $w \in \mathbb{R}^2$, which gets added to μ to define a Gaussian $p(x|z) = \mathcal{N}(x|\mu + \delta, \sigma^2 \mathbf{I})$. By integrating over z , we “slide” this circular Gaussian “spray can” along the principal component axis w , which induces elliptical Gaussian contours in x space centered on μ . Adapted from Figure 12.9 of [Bis06].

This can also be extended to handle missing data (if we make the missing at random assumption — see Section 22.3.5 for discussion). In particular, let us partition $x = (x_1, x_2)$, $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2]$, and $\mu = [\mu_1, \mu_2]$, and suppose x_2 is missing (unknown). From Section 2.3.6, we have

$$p(z|x_1) = \mathcal{N}(z|\mu_{z|1}, \Sigma_{z|1}) \quad (29.36)$$

$$\Sigma_{z|1}^{-1} = \mathbf{I} + \mathbf{W}_1^\top \Sigma_{11}^{-1} \mathbf{W}_1 \quad (29.37)$$

$$\mu_{z|1} = \Sigma_{z|1} [\mathbf{W}_1^\top \Sigma_{11}^{-1} (x_1 - \mu_1)] \quad (29.38)$$

where Σ_{11} is the top left block of Ψ .

29.3.1.3 Computing the likelihood

In this section, we discuss how to efficiently compute the log (marginal) likelihood, which is given by

$$\log p(x|\mu, \mathbf{C}) = -\frac{1}{2} [D \log(2\pi) + \log \det(\mathbf{C}) + \tilde{x}^\top \mathbf{C}^{-1} \tilde{x}] \quad (29.39)$$

where $\tilde{x} = x - \mu$, and $\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \Psi$. We can avoid inverting the $D \times D$ matrix \mathbf{C} by using the matrix inversion lemma:

$$\mathbf{C}^{-1} = (\mathbf{W}\mathbf{W}^\top + \Psi)^{-1} \quad (29.40)$$

$$= \Psi^{-1} - \Psi^{-1} \mathbf{W} (\mathbf{I} + \mathbf{W}^\top \Psi^{-1} \mathbf{W})^{-1} \mathbf{W}^\top \Psi^{-1} \quad (29.41)$$

$$= \Psi^{-1} - \Psi^{-1} \mathbf{W} \mathbf{L}^{-1} \mathbf{W}^\top \Psi^{-1} \quad (29.42)$$

where $\mathbf{L} = \mathbf{I} + \mathbf{W}^\top \Psi^{-1} \mathbf{W}$ is $L \times L$. The Mahalanobis distance is then given by

$$\tilde{x}^\top \mathbf{C}^{-1} \tilde{x} = \tilde{x}^\top [\Psi^{-1} \tilde{x} - \Psi^{-1} \mathbf{W} \mathbf{L}^{-1} (\mathbf{W}^\top \Psi^{-1} \tilde{x})] \quad (29.43)$$

¹ which takes $O(L^3 + LD)$ to compute. From the matrix determinant lemma, the log determinant is
² given by
³

$$\log \det(\mathbf{C}) = \log \det(\mathbf{L}) + \log \det(\boldsymbol{\Psi}) \quad (29.44)$$

⁴ which takes $O(L^3 + D)$ to compute.
⁵

⁸ 29.3.1.4 Unidentifiability of the parameters

⁹ The parameters of a FA model are unidentifiable. To see this, consider a model with weights \mathbf{W} and
¹⁰ observation covariance $\boldsymbol{\Psi}$. We have

$$\text{Cov}[\mathbf{x}] = \mathbf{W}\mathbb{E}[zz^\top]\mathbf{W}^\top + \mathbb{E}[\epsilon\epsilon^\top] = \mathbf{WW}^\top + \boldsymbol{\Psi} \quad (29.45)$$

¹⁴ Now consider a different moel with weights $\tilde{\mathbf{W}} = \mathbf{WR}$, where \mathbf{R} is an arbitrary orthogonal rotation
¹⁵ matrix, satisfying $\mathbf{RR}^\top = \mathbf{I}$. This has the same likelihood, since
¹⁶

$$\text{Cov}[\mathbf{x}] = \tilde{\mathbf{W}}\mathbb{E}[zz^\top]\tilde{\mathbf{W}}^\top + \mathbb{E}[\epsilon\epsilon^\top] = \mathbf{WRR}^\top\mathbf{W}^\top + \boldsymbol{\Psi} = \mathbf{WW}^\top + \boldsymbol{\Psi} \quad (29.46)$$

¹⁸ Geometrically, multiplying \mathbf{W} by an orthogonal matrix is like rotating \mathbf{z} before generating \mathbf{x} ; but
¹⁹ since \mathbf{z} is drawn from an isotropic Gaussian, this makes no difference to the likelihood. Consequently,
²⁰ we cannot uniquely identify \mathbf{W} , and therefore cannot uniquely identify the latent factors, either.
²¹ This is called the “**factor rotations problem**” (see e.g., [Dar80]).

²² To break this symmetry, several solutions can be used, as we discuss below.

- ²⁴ • **Forcing \mathbf{W} to have orthonormal columns.** Perhaps the simplest solution to the identifiability
²⁵ problem is to force \mathbf{W} to have orthonormal columns. This is the approach adopted by PCA.
²⁶ The resulting posterior estimate will then be unique, up to permutation of the latent dimensions.
²⁷ (In PCA, this ordering ambiguity is resolved by sorting the dimensions in order of decreasing
²⁸ eigenvalues of \mathbf{W} .)
- ²⁹ • **Forcing \mathbf{W} to be lower triangular.** One way to resolve permutation unidentifiability, which
³⁰ is popular in the Bayesian community (e.g., [LW04]), is to ensure that the first visible feature is
³¹ only generated by the first latent factor, the second visible feature is only generated by the first
³² two latent factors, and so on. For example, if $L = 3$ and $D = 4$, the correspond factor loading
³³ matrix is given by

$$\mathbf{W} = \begin{pmatrix} w_{11} & 0 & 0 \\ w_{21} & w_{22} & 0 \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} \quad (29.47)$$

³⁹ We also require that $w_{kk} > 0$ for $k = 1 : L$. The total number of parameters in this constrained
⁴⁰ matrix is $D + DL - L(L - 1)/2$, which is equal to the number of uniquely identifiable parameters in
⁴¹ FA.¹ The disadvantage of this method is that the first L visible variables, known as the **founder**
⁴² **variables**, affect the interpretation of the latent factors, and so must be chosen carefully.

⁴³ 1. We get D parameters for $\boldsymbol{\Psi}$ and DL for \mathbf{W} , but we need to remove $L(L - 1)/2$ degrees of freedom coming from \mathbf{R} ,
⁴⁴ since that is the dimensionality of the space of orthogonal matrices of size $L \times L$. To see this, note that there are $L - 1$
⁴⁵ free parameters in \mathbf{R} in the first column (since the column vector must be normalized to unit length), there are $L - 2$
⁴⁶ free parameters in the second column (which must be orthogonal to the first), and so on.

- **Sparsity promoting priors on the weights.** Instead of pre-specifying which entries in \mathbf{W} are zero, we can encourage the entries to be zero, using ℓ_1 regularization [ZHT06], ARD [Bis99; AB08], or spike-and-slab priors [Rat+09]. This is called sparse factor analysis. This does not necessarily ensure a unique MAP estimate, but it does encourage interpretable solutions.
- **Choosing an informative rotation matrix.** There are a variety of heuristic methods that try to find rotation matrices \mathbf{R} which can be used to modify \mathbf{W} (and hence the latent factors) so as to try to increase the interpretability, typically by encouraging them to be (approximately) sparse. One popular method is known as varimax [Kai58].
- **Use of non-Gaussian priors for the latent factors.** If we replace the prior on the latent variables, $p(\mathbf{z})$, with a non-Gaussian distribution, we can sometimes uniquely identify \mathbf{W} , as well as the latent factors. See e.g., [KKH20] for details.

29.3.2 Probabilistic PCA

In this section, we consider a special case of the factor analysis model in which \mathbf{W} has orthonormal columns, $\Psi = \sigma^2\mathbf{I}$ and $\mu = 0$. This model is called **probabilistic principal components analysis (PPCA)** [TB99], or **sensible PCA** [Row97]. The marginal distribution on the visible variables has the form

$$p(\mathbf{x}|\theta) = \int \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})d\mathbf{z} = \mathcal{N}(\mathbf{x}|\mu, \mathbf{C}) \quad (29.48)$$

where

$$\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I} \quad (29.49)$$

29.3.2.1 Connection to PCA

The log likelihood for PPCA is given by

$$\log p(\mathbf{X}|\mu, \mathbf{W}, \sigma^2) = -\frac{ND}{2} \log(2\pi) - \frac{N}{2} \log |\mathbf{C}| - \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n - \mu)^\top \mathbf{C}^{-1} (\mathbf{x}_n - \mu) \quad (29.50)$$

The MLE for μ is $\bar{\mathbf{x}}$. Plugging in gives

$$\log p(\mathbf{X}|\mu, \mathbf{W}, \sigma^2) = -\frac{N}{2} [D \log(2\pi) + \log |\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} \mathbf{S})] \quad (29.51)$$

where $\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^\top$ is the empirical covariance matrix.

In [TB99; Row97] they show that the maximum of this objective must satisfy

$$\mathbf{W} = \mathbf{U}_L (\Lambda_L - \sigma^2 \mathbf{I})^{\frac{1}{2}} \mathbf{R} \quad (29.52)$$

where \mathbf{U}_L is a $D \times L$ matrix whose columns are given by the L eigenvectors of \mathbf{S} with largest eigenvalues, Λ_L is the $L \times L$ diagonal matrix of corresponding eigenvalues, and \mathbf{R} is an arbitrary $L \times L$ orthogonal matrix, which (WLOG) we can take to be $\mathbf{R} = \mathbf{I}$.

If we plug in the MLE for \mathbf{W} , we find the covariance for the predictive distribution to be

$$\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I} = \mathbf{U}_L (\Lambda_L - \sigma^2 \mathbf{I}) \mathbf{U}_L^\top + \sigma^2 \mathbf{I} \quad (29.53)$$

1 In the noise-free limit, where $\sigma^2 = 0$, we see that $\mathbf{W}_{\text{mle}} = \mathbf{U}_L \Lambda_L^{\frac{1}{2}}$ and $\mathbf{C} = \mathbf{WW}^\top$, as in PCA.
2 The MLE for the observation variance is
3

4

$$\sigma^2 = \frac{1}{D - L} \sum_{i=L+1}^D \lambda_i \quad (29.54)$$

5

6 which is the average distortion associated with the discarded dimensions. If $L = D$, then the
7 estimated noise is 0, since the model collapses to $\mathbf{z} = \mathbf{x}$.
8

9 **29.3.2.2 Computing the posterior**

10 To use PPCA as an alternative to PCA, we need to compute the posterior mean $\mathbb{E}[\mathbf{z}|\mathbf{x}]$, which is the
11 equivalent of the PCA encoder model. Using the factor analysis results from Section 29.3.1.2, we
12 have
13

14

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\sigma^{-2}\mathbf{WW}^\top(\mathbf{x} - \boldsymbol{\mu}), \Sigma) \quad (29.55)$$

15

16 where
17

18

$$\Sigma^{-1} = \mathbf{I} + \sigma^{-2}\mathbf{WW}^\top = \frac{1}{\sigma^2}(\sigma^2\mathbf{I} + \mathbf{WW}^\top) = \frac{1}{\sigma^2}\mathbf{M} \quad (29.56)$$

19

20 To compute this efficiently, we can use the matrix inversion lemma to write
21

22

$$(\mathbf{WW}^\top + \sigma^2\mathbf{I})^{-1} = (\mathbf{I} - \mathbf{W}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top)/\sigma^2 \quad (29.57)$$

23

24 If we define
25

26

$$\mathbf{M} = (\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}) \quad (29.58)$$

27

28 the posterior covariance simplifies to
29

30

$$\Sigma = \sigma^2\mathbf{M} \quad (29.59)$$

31

32 Hence
33

34

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{M}^{-1}\mathbf{WW}^\top(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1}) \quad (29.60)$$

35

36 In the $\sigma^2 = 0$ limit, we have $\mathbf{M} = \mathbf{WW}^\top$ and so
37

38

$$\mathbb{E}[\mathbf{z}|\mathbf{x}] = (\mathbf{WW}^\top)^{-1}\mathbf{WW}^\top(\mathbf{x} - \bar{\mathbf{x}}) \quad (29.61)$$

39

40 This is the orthogonal projection of the data into the latent space, as in standard PCA.
41

29.3.2.3 Computing the likelihood

42 To compute $p(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C})$ efficiently, we need to evaluate \mathbf{C}^{-1} and $\log |\mathbf{C}|$ efficiently. To do this, we can
43 use the matrix inversion lemma to write
44

45

$$\mathbf{C}^{-1} = (\mathbf{WW}^\top + \sigma^2\mathbf{I})^{-1} = (\mathbf{I} - \mathbf{W}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top)/\sigma^2 = \sigma^{-2} [\mathbf{I} - \mathbf{WM}^{-1}\mathbf{W}^\top] \quad (29.62)$$

46

47

where \mathbf{M} is defined in Equation (29.58). When we plug in the MLE for \mathbf{W} from Equation (29.52) (using $\mathbf{R} = \mathbf{I}$) we find

$$\mathbf{C}^{-1} = \sigma^{-2} [\mathbf{I} - \mathbf{U}_L(\Lambda_L - \sigma^2 \mathbf{I})\Lambda_L^{-1}\mathbf{U}_L^\top] = \sigma^{-2}(\mathbf{I} - \mathbf{U}_L\mathbf{K}\mathbf{U}_L^\top) \quad (29.63)$$

where $\mathbf{K} = \text{diag}(k_d)$ and $k_d = 1 - \frac{\sigma_d^2}{\lambda_d}$. Similarly one can show

$$\log |\mathbf{C}| = \log |\mathbf{W}\mathbf{W}^\top + \sigma^2 \mathbf{I}| = (D - L) \log \sigma^2 + \sum_{j=1}^L \log \lambda_j \quad (29.64)$$

$$= D \log \sigma^2 - \sum_{d=1}^L \log(1 - k_d) \quad (29.65)$$

29.3.2.4 EM for (P)PCA

In Section 29.3.2.1, we showed how to fit the (P)PCA model using eigenvector method. we can also use EM, which has some advantages which we discuss in Section 29.3.2.5. This relies on the probabilistic formulation of PCA. However the algorithm continues to work in the zero noise limit, $\sigma^2 = 0$, as shown by [Row97].

In particular, let $\tilde{\mathbf{Z}} = \mathbf{Z}^\top$ be a $L \times N$ matrix storing the posterior means (low-dimensional representations) along its columns. Similarly, let $\tilde{\mathbf{X}} = \mathbf{X}^\top$ store the original data along its columns. From Equation (29.60), when $\sigma^2 = 0$, we have

$$\tilde{\mathbf{Z}} = (\mathbf{W}^\top \mathbf{W})^{-1} \mathbf{W}^\top \tilde{\mathbf{X}} \quad (29.66)$$

This constitutes the E step. Notice that this is just an orthogonal projection of the data.

From Equation 29.110, the M step is given by

$$\hat{\mathbf{W}} = \left[\sum_n \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^T \right] \left[\sum_n \mathbb{E}[\mathbf{z}_n] \mathbb{E}[\mathbf{z}_n]^T \right]^{-1} \quad (29.67)$$

where we exploited the fact that $\Sigma = \text{Cov}[\mathbf{z}|\mathbf{x}, \theta] = 0\mathbf{I}$ when $\sigma^2 = 0$.

It is worth comparing this expression to the MLE for multi-output linear regression, which has the form $\mathbf{W} = (\sum_i \mathbf{y}_i \mathbf{x}_i^T)(\sum_i \mathbf{x}_i \mathbf{x}_i^T)^{-1}$. Thus we see that the M step is like linear regression where we replace the observed inputs by the expected values of the latent variables.

In summary, here is the entire algorithm:

$$\tilde{\mathbf{Z}} = (\mathbf{W}^\top \mathbf{W})^{-1} \mathbf{W}^\top \tilde{\mathbf{X}} \quad (\text{E step}) \quad (29.68)$$

$$\mathbf{W} = \tilde{\mathbf{X}} \tilde{\mathbf{Z}}^T (\tilde{\mathbf{Z}} \tilde{\mathbf{Z}}^T)^{-1} \quad (\text{M step}) \quad (29.69)$$

[TB99] showed that the only stable fixed point of the EM algorithm is the globally optimal solution. That is, the EM algorithm converges to a solution where \mathbf{W} spans the same linear subspace as that defined by the first L eigenvectors. However, if we want \mathbf{W} to be orthogonal, and to contain the eigenvectors in descending order of eigenvalue, we have to orthogonalize the resulting matrix (which can be done quite cheaply). Alternatively, we can modify EM to give the principal basis directly [AO03].

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

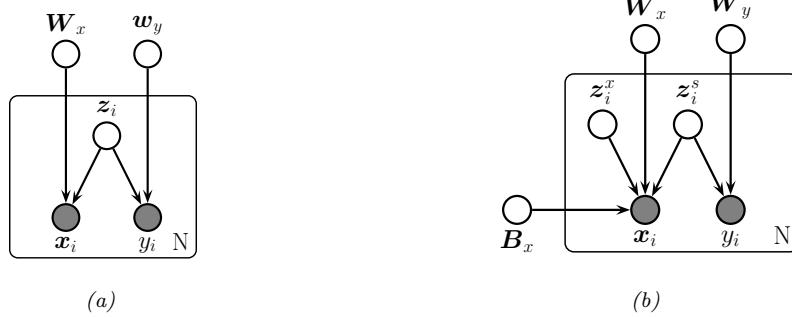


Figure 29.7: Gaussian latent factor models for paired data. (a) Supervised PCA. (b) Partial least squares.

29.3.2.5 EM vs eigenvector methods

- EM for PCA has the following advantages over eigenvector methods:
- EM can be faster. In particular, assuming $N, D \gg L$, the dominant cost of EM is the projection operation in the E step, so the overall time is $O(TLND)$, where T is the number of iterations. [Row97] showed experimentally that the number of iterations is usually very small (the mean was 3.6), regardless of N or D . (This results depends on the ratio of eigenvalues of the empirical covariance matrix.) This is much faster than the $O(\min(ND^2, DN^2))$ time required by straightforward eigenvector methods, although more sophisticated eigenvector methods, such as the Lanczos algorithm, have running times comparable to EM.
 - EM can be implemented in an online fashion, i.e., we can update our estimate of \mathbf{W} as the data streams in.
 - EM can handle missing data in a simple way (see e.g., [IR10; DJ15]). (See Section 22.3.5 for more discussion of missing data.)
 - EM can be extended to handle mixtures of PPCA/ FA models (see Section 29.4).
 - EM can be modified to variational EM or to variational Bayes EM to fit more complex models (see e.g., Section 29.3.4).

29.3.3 Factor analysis models for paired data

In this section, we discuss linear-Gaussian factor analysis models when we have two kinds of observed variables, $\mathbf{x} \in \mathbb{R}^{D_x}$ and $\mathbf{y} \in \mathbb{R}^{D_y}$, which are paired. These often correspond to different sensors or modalities (e.g., images and sound). We follow the presentation of [Vir10].

29.3.3.1 Supervised PCA

If we have two observed signals, we can model the joint $p(\mathbf{x}, \mathbf{y})$ using a shared low-dimensional representation using the following linear Gaussian model:

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n | \mathbf{0}, \mathbf{I}_L) \quad (29.70)$$

$$p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \mathbf{W}_x \mathbf{z}_n, \sigma_x^2 \mathbf{I}_{D_x}) \quad (29.71)$$

$$p(\mathbf{y}_n | \mathbf{z}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_n | \mathbf{W}_y \mathbf{z}_n, \sigma_y^2 \mathbf{I}_{D_y}) \quad (29.72)$$

This is illustrated as a graphical model in Figure 29.7a. The intuition is that \mathbf{z}_n is a shared latent subspace, that captures features that \mathbf{x}_n and \mathbf{y}_n have in common. The variance terms σ_x and σ_y control how much emphasis the model puts on the two different signals.

The above model is called **supervised PCA** [Yu+06]. If we put a prior on the parameters $\boldsymbol{\theta} = (\mathbf{W}_x, \mathbf{W}_y, \sigma_x, \sigma_y)$, it is called **Bayesian factor regression** model of [Wes03].

We can marginalize out \mathbf{z}_n to get $p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta})$. If \mathbf{y}_n is a scalar, this becomes

$$p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) = \mathcal{N}(y_n | \mathbf{x}_n^\top \mathbf{v}, \mathbf{w}_y^\top \mathbf{C} \mathbf{w}_y + \sigma_y^2) \quad (29.73)$$

$$\mathbf{C} = (\mathbf{I} + \sigma_x^{-2} \mathbf{W}_x^\top \mathbf{W}_x)^{-1} \quad (29.74)$$

$$\mathbf{v} = \sigma_x^{-2} \mathbf{W}_x \mathbf{C} \mathbf{w}_y \quad (29.75)$$

To apply this to the classification setting, we can replace the Gaussian $p(\mathbf{y} | \mathbf{z})$ with a logistic regression model:

$$p(y_n | \mathbf{z}_n, \boldsymbol{\theta}) = \text{Ber}(y_n | \boldsymbol{\sigma}(\mathbf{w}_y^\top \mathbf{z}_n)) \quad (29.76)$$

In this case, we can no longer compute the marginal posterior predictive $p(y_n | \mathbf{x}_n, \boldsymbol{\theta})$ in closed form, but we use can techniques similar to exponential family PCA (see [Guo09] for details).

The above model is completely symmetric in \mathbf{x} and \mathbf{y} . If our goal is to predict \mathbf{y} from \mathbf{x} via the latent bottleneck \mathbf{z} , then we might want to upweight the likelihood term for \mathbf{y} , as proposed in [Ris+08]. This gives

$$p(\mathbf{X}, \mathbf{Y}, \mathbf{Z} | \boldsymbol{\theta}) = p(\mathbf{Y} | \mathbf{Z}, \mathbf{W}_y) p(\mathbf{X} | \mathbf{Z}, \mathbf{W}_x)^\alpha p(\mathbf{Z}) \quad (29.77)$$

where $\alpha \leq 1$ controls the relative importance of modeling the two sources. The value of α can be chosen by cross-validation.

29.3.3.2 Partial least squares

We now consider an asymmetric or more “discriminative” form of supervised PCA. The key idea is to allow some of the (co)variance in the input features to be explained by its own subspace, \mathbf{z}_i^x , and to let the rest of the subspace, \mathbf{z}_i^s , be shared between input and output. The model has the form

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i^s | \mathbf{0}, \mathbf{I}_{L_s}) \mathcal{N}(\mathbf{z}_i^x | \mathbf{0}, \mathbf{I}_{L_x}) \quad (29.78)$$

$$p(\mathbf{y}_i | \mathbf{z}_i) = \mathcal{N}(\mathbf{W}_y \mathbf{z}_i^s + \boldsymbol{\mu}_y, \sigma^2 \mathbf{I}_{D_y}) \quad (29.79)$$

$$p(\mathbf{x}_i | \mathbf{z}_i) = \mathcal{N}(\mathbf{W}_x \mathbf{z}_i^s + \mathbf{B}_x \mathbf{z}_i^x + \boldsymbol{\mu}_x, \sigma^2 \mathbf{I}_{D_x}) \quad (29.80)$$

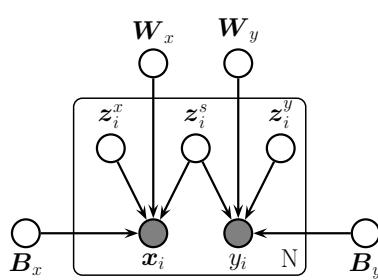


Figure 29.8: Canonical correlation analysis as a PGM.

See Figure 29.7b. The corresponding induced distribution on the visible variables has the form

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{v}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I})d\mathbf{z}_i = \mathcal{N}(\mathbf{v}_i|\boldsymbol{\mu}, \mathbf{WW}^T + \sigma^2\mathbf{I}) \quad (29.81)$$

where $\mathbf{v}_i = (\mathbf{x}_i; \mathbf{y}_i)$, $\boldsymbol{\mu} = (\boldsymbol{\mu}_y; \boldsymbol{\mu}_x)$ and

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_y & \mathbf{0} \\ \mathbf{W}_x & \mathbf{B}_x \end{pmatrix} \quad (29.82)$$

$$\mathbf{WW}^T = \begin{pmatrix} \mathbf{W}_y \mathbf{W}_y^T & \mathbf{W}_x \mathbf{W}_x^T \\ \mathbf{W}_x \mathbf{W}_x^T & \mathbf{W}_x \mathbf{W}_x^T + \mathbf{B}_x \mathbf{B}_x^T \end{pmatrix} \quad (29.83)$$

We should choose L large enough so that the shared subspace does not capture covariate-specific variation.

MLE in this model is equivalent to the technique of **partial least squares (PLS)** [Gus01; Nou+02; Sun+09]. This model can be also be generalized to discrete data using the exponential family [Vir10].

29.3.3.3 Canonical correlation analysis

We now consider a symmetric unsupervised version of PLS, in which we allow each view to have its own ‘‘private’’ subspace, but there is also a shared subspace. If we have two observed variables, \mathbf{x}_i and \mathbf{y}_i , then we have three latent variables, $\mathbf{z}_i^s \in \mathbb{R}^{L_s}$ which is shared, $\mathbf{z}_i^x \in \mathbb{R}^{L_x}$ and $\mathbf{z}_i^y \in \mathbb{R}^{L_y}$ which are private. We can write the model as follows [BJ05]:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i^s|\mathbf{0}, \mathbf{I}_{L_s})\mathcal{N}(\mathbf{z}_i^x|\mathbf{0}, \mathbf{I}_{L_x})\mathcal{N}(\mathbf{z}_i^y|\mathbf{0}, \mathbf{I}_{L_y}) \quad (29.84)$$

$$p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|\mathbf{B}_x \mathbf{z}_i^x + \mathbf{W}_x \mathbf{z}_i^s + \boldsymbol{\mu}_x, \sigma^2 \mathbf{I}_{D_x}) \quad (29.85)$$

$$p(\mathbf{y}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{y}_i|\mathbf{B}_y \mathbf{z}_i^y + \mathbf{W}_y \mathbf{z}_i^s + \boldsymbol{\mu}_y, \sigma^2 \mathbf{I}_{D_y}) \quad (29.86)$$

See Figure 29.8 The corresponding joint distribution has the form

$$p(\mathbf{v}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{v}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I})d\mathbf{z}_i = \mathcal{N}(\mathbf{v}_i|\boldsymbol{\mu}, \mathbf{WW}^T + \sigma^2\mathbf{I}_D) \quad (29.87)$$

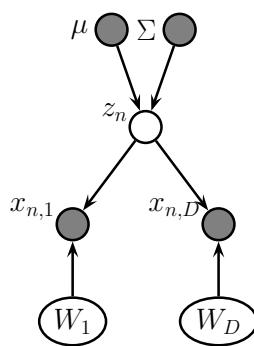


Figure 29.9: Exponential family PCA model as a PGM-D.

where

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_x & \mathbf{B}_x & \mathbf{0} \\ \mathbf{W}_y & \mathbf{0} & \mathbf{B}_y \end{pmatrix} \quad (29.88)$$

$$\mathbf{WW}^T = \begin{pmatrix} \mathbf{W}_x \mathbf{W}_x^T + \mathbf{B}_x \mathbf{B}_x^T & \mathbf{W}_x \mathbf{W}_y^T \\ \mathbf{W}_y \mathbf{W}_y^T & \mathbf{W}_y \mathbf{W}_y^T + \mathbf{B}_y \mathbf{B}_y^T \end{pmatrix} \quad (29.89)$$

[BJ05] showed that MLE for this model is equivalent to a classical statistical method known as **canonical correlation analysis** or **CCA** [Hot36]. However, the PGM perspective allows us to easily generalize to multiple kinds of observations (this is known as **generalized CCA** [Hor61]) or to nonlinear models (this is known as **deep CCA** [WLL16; SNM16]), or exponential family CCA [KVK10]. See [Uur+17] for further discussion of CCA and its extensions.

29.3.4 Factor analysis with exponential family likelihoods

So far we have assumed the observed data is real-valued, so $\mathbf{x}_n \in \mathbb{R}^D$. If we want to model other kinds of data (e.g., binary or categorical), we can simply replace the Gaussian output distribution with a suitable member of the exponential family, where the natural parameters are given by a linear function of \mathbf{z}_n . That is, we use

$$p(\mathbf{x}_n | \mathbf{z}_n) = \exp(\mathcal{T}(\mathbf{x})^\top \boldsymbol{\theta} + h(\mathbf{x}) - g(\boldsymbol{\theta})) \quad (29.90)$$

where the $N \times D$ matrix of natural parameters is assumed to be given by the low rank decomposition $\boldsymbol{\Theta} = \mathbf{Z}\mathbf{W}$, where \mathbf{Z} is $N \times L$ and \mathbf{W} is $L \times D$. The resulting model is called **exponential family factor analysis**

Unlike the linear-Gaussian FA, we cannot compute the exact posterior $p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{W})$ due to the lack of conjugacy between the expfam likelihood and the Gaussian prior. Furthermore, we cannot compute the exact marginal likelihood either, which prevents us from finding the optimal MLE.

[CDS02] proposed a coordinate ascent method for a deterministic variant of this model, known as **exponential family PCA**. This alternates between computing a point estimate of \mathbf{z}_n and \mathbf{W} . This

¹ can be regarded as a degenerate version of variational EM, where the E step uses a delta function
² posterior for \mathbf{z}_n . [GS08] present an improved algorithm that finds the global optimum, and [Ude+16]
³ presents an extension called **generalized low rank models**, that covers many different kinds of
⁴ loss function.
⁵

⁶ However, it is often preferable to use a probabilistic version of the model, rather than computing
⁷ point estimates of the latent factors. In this case, we must represent the posterior use a non-degenerate
⁸ distribution to avoid overfitting, since the number of latent variables is proportional to the number
⁹ of datacases [WCS08]. Fortunately, we can use a non-degenerate posterior, such as a Gaussian, by
¹⁰ optimizing the variational lower bound. We give some examples of this below.

¹¹ **29.3.4.1 Example: binary PCA**

¹² Consider a factored Bernoulli likelihood:
¹³

$$\begin{aligned} \sup_{14} \quad p(\mathbf{x}|\mathbf{z}) &= \prod_d \text{Ber}(x_d|\boldsymbol{\sigma}(\mathbf{w}_d^\top \mathbf{z})) \end{aligned} \tag{29.91}$$

¹⁵ Suppose we observe $N = 150$ bit vectors of length $D = 6$. Each example is generated by choosing one
¹⁶ of three binary prototype vectors, and then by flipping bits at random. See Figure 29.10(a) for the
¹⁷ data. We can fit this using the variational EM algorithm (see [Tip98] for details). We use $L = 2$ latent
¹⁸ dimensions to allow us to visualize the latent space. In Figure 29.10(b), we plot $\mathbb{E}[\mathbf{z}_n|\mathbf{x}_n, \hat{\mathbf{W}}]$. We see
¹⁹ that the projected points group into three distinct clusters, as is to be expected. In Figure 29.10(c),
²⁰ we plot the reconstructed version of the data, which is computed as follows:
²¹

$$\begin{aligned} \sup_{22} \quad p(\hat{x}_{nd} = 1|\mathbf{x}_n) &= \int d\mathbf{z}_n p(\mathbf{z}_n|\mathbf{x}_n)p(\hat{x}_{nd}|\mathbf{z}_n) \end{aligned} \tag{29.92}$$

²³ If we threshold these probabilities at 0.5 (corresponding to a MAP estimate), we get the “denoised”
²⁴ version of the data in Figure 29.10(d).
²⁵

²⁶ **29.3.4.2 Example: categorical PCA**

²⁷ We can generalize the model in Section 29.3.4.1 to handle categorical data by using the following
²⁸ likelihood:
²⁹

$$\begin{aligned} \sup_{30} \quad p(\mathbf{x}|\mathbf{z}) &= \prod_d \text{Cat}(x_d|\mathcal{S}(\mathbf{W}_d \mathbf{z})) \end{aligned} \tag{29.93}$$

³¹ We call this **categorical PCA (CatPCA)**. A variational EM algorithm for fitting this is described
³² in [Kha+10].
³³

³⁴ **29.3.5 Factor analysis with DNN likelihoods**

³⁵ The FA model assumes the observed data can be modeled as arising from a linear mapping from a
³⁶ low-dimensional set of Gaussian factors. One way to relax this assumption is to let the mapping
³⁷ from \mathbf{z} to \mathbf{x} be a nonlinear model, such as a neural network. That is, the likelihood becomes
³⁸

$$\begin{aligned} \sup_{39} \quad p(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\mathbf{x}|f(\mathbf{w}; \boldsymbol{\theta}), \sigma^2 \mathbf{I}) \end{aligned} \tag{29.94}$$

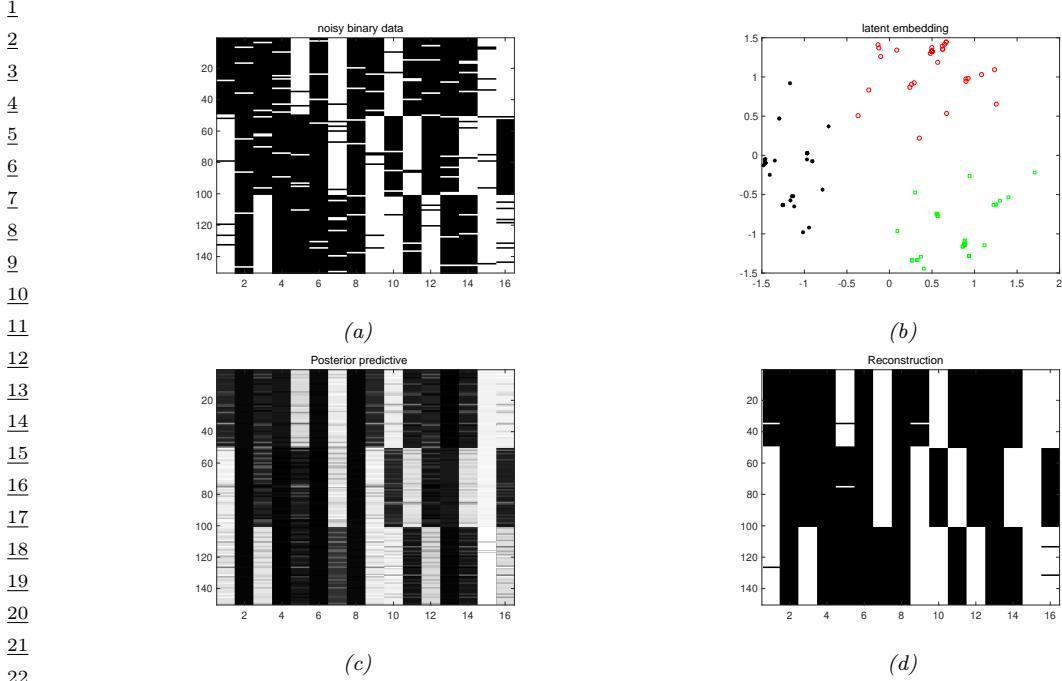


Figure 29.10: (a) 150 synthetic 16 dimensional bit vectors. (b) The 2d embedding learned by binary PCA, fit using variational EM. We have color coded points by the identity of the true “prototype” that generated them. (c) Predicted probability of being on. (d) Thresholded predictions. Generated by [binary_fa_demo.py](#).

We call this “**DNN factor analysis**”. (We can of course replace the Gaussian likelihood with other distributions, such as categorical.) Unfortunately we can no longer compute the posterior or the MLE exactly, so we need to use approximate methods. In Chapter 22, we discuss variational autoencoders, which fits this model using amortized variational inference. However, it is also possible to fit the same model using other inference methods, such as MCMC (see e.g., [Hof17]).

29.3.6 Factor analysis with GP likelihoods (GP-LVM)

Another way to make a nonlinear version of FA is to replace f with a Gaussian process (Chapter 18). This is known as a **GP-LVM**, which stands for “Gaussian process latent variable model” [Law05]. (This is closely related to an approach known as **kernel PCA**, discussed in [Mur22, Sec 20.4.6].)

To explain the method in more detail, we start with PPCA (Section 29.3.2). Recall that the PPCA model is as follows:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i | \mathbf{0}, \mathbf{I}) \quad (29.95)$$

$$p(\mathbf{x}_i | \mathbf{z}_i, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_i | \mathbf{W}\mathbf{z}_i, \sigma^2 \mathbf{I}) \quad (29.96)$$

We can fit this model by maximum likelihood, by integrating out the \mathbf{z}_i and maximizing \mathbf{W} (and

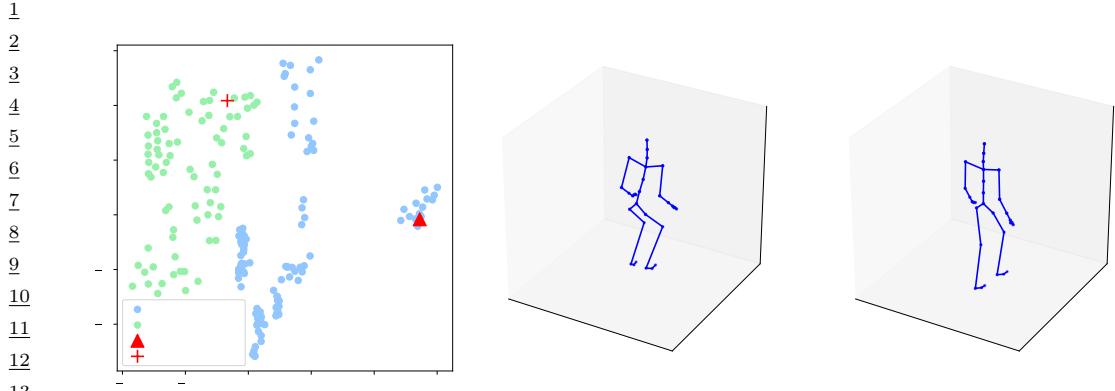


Figure 29.11: Illustration of a 2d embedding of human motion-capture data using a GP-LVM. We show two poses and their corresponding embeddings. Generated by [gplvm_mocap.ipynb](#). Used with kind permission of Aditya Ravuri.

17

18

19

20

21 σ^2). The objective is given by

22

$$23 \quad p(\mathbf{X}|\mathbf{W}, \sigma^2) = (2\pi)^{-DN/2} |\mathbf{C}|^{-N/2} \exp\left(-\frac{1}{2} \text{tr}(\mathbf{C}^{-1} \mathbf{X}^\top \mathbf{X})\right) \quad (29.97)$$

25

26 where $\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2 \mathbf{I}$. As we showed in Section 29.3.2, the MLE for \mathbf{W} can be computed in terms
27 of the eigenvectors of $\mathbf{X}^\top \mathbf{X}$.

28 Now we consider the dual problem, whereby we maximize \mathbf{Z} and integrate out \mathbf{W} . We will use a
29 prior of the form $p(\mathbf{W}) = \prod_j \mathcal{N}(\mathbf{w}_j | \mathbf{0}, \mathbf{I})$. The corresponding likelihood becomes
30

31

$$32 \quad p(\mathbf{X}|\mathbf{Z}, \sigma^2) = \prod_{d=1}^D \mathcal{N}(\mathbf{X}_{:,d} | \mathbf{0}, \mathbf{Z}\mathbf{Z}^\top + \sigma^2 \mathbf{I}) \quad (29.98)$$

34

$$35 \quad = (2\pi)^{-DN/2} |\mathbf{K}_z|^{-D/2} \exp\left(-\frac{1}{2} \text{tr}(\mathbf{K}_z^{-1} \mathbf{X} \mathbf{X}^\top)\right) \quad (29.99)$$

37

38 where $\mathbf{K}_z = \mathbf{K} + \sigma^2 \mathbf{I}$, and $\mathbf{K} = \mathbf{Z}\mathbf{Z}^\top$. The MLE for \mathbf{Z} can be computed in terms of the eigenvectors
39 of \mathbf{K}_z , and gives the same results as PPCA (see [Law05] for the details).

40 The advantage of the dual formulation is that we can use a more general kernel for \mathbf{K} instead of
41 $\mathbf{K} = \mathbf{Z}\mathbf{Z}^\top$. The MLE for \mathbf{Z} is no longer be available via eigenvalue methods, but can be computed
42 using gradient-based optimization.

43 In Figure 29.11, we illustrate the model (with an ARD kernel) applied to some **motion capture**
44 data, from the CMU mocap database at <http://mocap.cs.cmu.edu/>. Each person has 41 markers,
45 whose motion in 3d is tracked using 12 infrared cameras. Each data point corresponds to a different
46 body pose. When projected to 2d, we see that similar poses are clustered nearby.

47

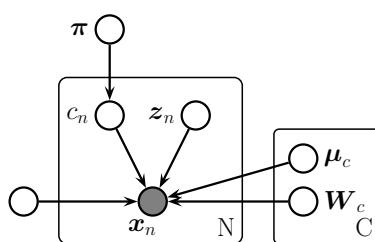


Figure 29.12: Mixture of factor analysers as a PGM.

29.4 Mixture of factor analysers

The factor analysis model (Section 29.3.1) assumes the observed data can be modeled as arising from a linear mapping from a low-dimensional set of Gaussian factors. One way to relax this assumption is to assume the model is only locally linear, so the overall model becomes a (weighted) combination of FA models; this is called a **mixture of factor analysers**. The overall model for the data is a mixture of linear manifolds, which can be used to approximate an overall curved manifold. (Another way to think of this model is a mixture of Gaussians, where each mixture component has a low-rank covariance matrix.)

29.4.1 Model definition

More precisely, let latent indicator $c_n \in \{1, \dots, K\}$, specifying which subspace (cluster) we should use to generate the data. If $c_n = k$, we sample \mathbf{z}_n from a Gaussian prior and pass it through the \mathbf{W}_k matrix and add noise, where \mathbf{W}_k maps from the L -dimensional subspace to the D -dimensional visible space.² More precisely, the model is as follows:

$$p(\mathbf{x}_n | \mathbf{z}_n, c_n = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k + \mathbf{W}_k \mathbf{z}_n, \boldsymbol{\Psi}_k) \quad (29.100)$$

$$p(\mathbf{z}_n | \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n | \mathbf{0}, \mathbf{I}) \quad (29.101)$$

$$p(c_n | \boldsymbol{\theta}) = \text{Cat}(c_n | \boldsymbol{\pi}) \quad (29.102)$$

This is called a **mixture of factor analysers** (MFA) [GH96b]. The corresponding distribution in the visible space is given by

$$p(\mathbf{x} | \boldsymbol{\theta}) = \sum_k p(c = k) \int p(\mathbf{z} | c) p(\mathbf{x} | \mathbf{z}, c) d\mathbf{z} = \sum_k \pi_k \int \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}_k, \mathbf{I}) \mathcal{N}(\mathbf{x} | \mathbf{W}_k \mathbf{z}, \boldsymbol{\Psi}_k) d\mathbf{z} \quad (29.103)$$

In the special case that $\boldsymbol{\Psi}_k = \sigma^2 \mathbf{I}$, we get a mixture of PPCA models (although it is difficult to ensure orthogonality of the \mathbf{W}_k in this case). See Figure 29.13 for an example of the method applied to some 2d data.

² If we allow \mathbf{z}_n to depend on c_n , we can let each subspace have a different dimensionality, as suggested in [KS15].

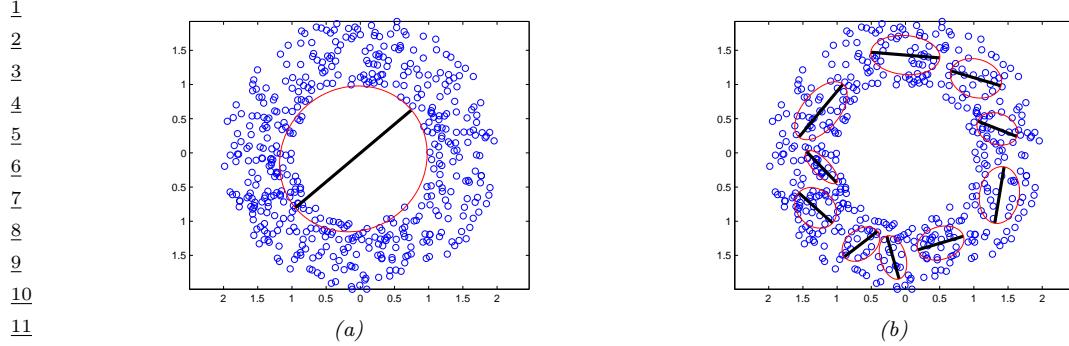


Figure 29.13: Mixture of PPCA models fit to a 2d dataset, using $L = 1$ latent dimensions. (a) $K = 1$ mixture components. (b) $K = 10$ mixture components. Generated by `mixPpcaDemo.py`.

We can think of this as a low-rank version of a mixture of Gaussians. In particular, this model needs $O(KLD)$ parameters instead of the $O(KD^2)$ parameters needed for a mixture of full covariance Gaussians. This can reduce overfitting.

29.4.2 Model fitting

There are several ways to fit an MFA model, some of which we discuss below.

29.4.2.1 Model fitting using EM

We can fit this model using EM (see [GH96b] for the derivation). In the E step, we compute the posterior responsibility of cluster c for data point n using

$$r_{nc} \triangleq p(c_n = c | \mathbf{x}_n, \boldsymbol{\theta}) \propto \pi_c \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_c, \mathbf{W}_c \mathbf{W}_c^\top + \boldsymbol{\Psi}) \quad (29.104)$$

The conditional posterior for \mathbf{z}_n is given by

$$p(\mathbf{z}_n | \mathbf{x}_n, c_n = c, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n | \mathbf{m}_{nc}, \boldsymbol{\Sigma}_{nc}) \quad (29.105)$$

$$\boldsymbol{\Sigma}_{nc} \triangleq (\mathbf{I} + \mathbf{W}_c^\top \boldsymbol{\Psi}_c^{-1} \mathbf{W}_c)^{-1} \quad (29.106)$$

$$\mathbf{m}_{nc} \triangleq \boldsymbol{\Sigma}_{nc} (\mathbf{W}_c^\top \boldsymbol{\Psi}_c^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_c)) \quad (29.107)$$

For the M step, we define $\tilde{\mathbf{W}}_c = (\mathbf{W}_c, \boldsymbol{\mu}_c)$, and $\tilde{\mathbf{z}} = (\mathbf{z}, 1)$. Also, define

$$\mathbf{b}_{nc} \triangleq \mathbb{E} [\tilde{\mathbf{z}} | \mathbf{x}_n, c_n = c] = [\mathbf{m}_{nc}; 1] \quad (29.108)$$

$$\boldsymbol{\Omega}_{nc} \triangleq \mathbb{E} [\tilde{\mathbf{z}} \tilde{\mathbf{z}}^\top | \mathbf{x}_n, c_n = c] = \begin{pmatrix} \mathbb{E} [\mathbf{z} \mathbf{z}^\top | \mathbf{x}_n, c_n = c] & \mathbb{E} [\mathbf{z} | \mathbf{x}_n, c_n = c] \\ \mathbb{E} [\mathbf{z} | \mathbf{x}_n, c_n = c]^\top & 1 \end{pmatrix} \quad (29.109)$$

1
2 Then the M step is as follows:

3
4 $\hat{\mathbf{W}}_c = \left[\sum_n r_{nc} \mathbf{x}_n \mathbf{b}_c^\top \right] \left[\sum_n r_{nc} \boldsymbol{\Omega}_{nc} \right]^{-1}$ (29.110)
5

6
7 $\hat{\Psi} = \frac{1}{N} \text{diag} \left\{ \sum_{nc} r_{nc} (\mathbf{x}_n - \hat{\mathbf{W}}_c \mathbf{b}_{nc}) \mathbf{x}_n^\top \right\}$ (29.111)
8

9
10 $\hat{\pi}_c = \frac{1}{N} \sum_n r_{nc}$ (29.112)
11

12 29.4.2.2 Model fitting using SGD

14 We can also fit mixture models using SGD, as shown in [RW18]. This idea can be combined with
15 an inference network (see Section 10.3.7) to efficiently approximate the posterior over the latent
16 variables. [Zon+18] use this approach to jointly learn a GMM applied to a deep autoencoder to
17 provide a nonlinear extension of MFA; they show good results on anomaly detection.s
18

19 29.4.2.3 Model selection

21 To choose the number of mixture components C , and the number of latent dimensions L , we can
22 use discrete search combined with objectives such as the marginal likelihood or validation likelihood.
23 However, we can also use numerical optimization methods to optimize L , which can be faster. We
24 initially assume that C is known. To estimate L , we set the model to its maximal size, and then use
25 a technique called automatic relevance determination or ARD to automatically prune out irrelevant
26 weights (see Section 15.2.7). This can be implemented using variational Bayes EM (Section 10.2.5);
27 for details, see [Bis99; GB00].

28 Figure 29.14 illustrates this approach applied to a mixture of FA models fit to a small synthetic
29 dataset. The figures visualize the weight matrices for each cluster, using **Hinton diagrams**, where
30 where the size of the square is proportional to the value of the entry in the matrix. We see that
31 many of them are sparse. Figure 29.15 shows that the degree of sparsity depends on the amount of
32 training data, in accord with the Bayesian Occam’s razor. In particular, when the sample size is
33 small, the method automatically prefers simpler models, but as the sample size gets sufficiently large,
34 the method converges on the “correct” solution, which is one with 6 subspaces of dimensionality 1, 2,
35 2, 3, 4 and 7.

36 Although the ARD method can estimate the number of latent dimensions L , it still needs to
37 perform discrete search over the number of mixture components C . This is done using “birth” and
38 “death” moves [GB00]. An alternative approach is to perform stochastic sampling in the space of
39 models. Traditional approaches, such as [LW04], are based on reversible jump MCMC, and also use
40 birth and death moves. However, this can be slow and difficult to implement. More recent approaches
41 use non-parametric priors, combined with Gibbs sampling, see e.g., [PC09].
42

43 29.4.3 MixFA for image generation

45 In this section, we use the MFA model as a generative model for images, following [RW18]. This is
46 equivalent to using a mixture of Gaussians, where each mixture component has a low-rank covariance
47

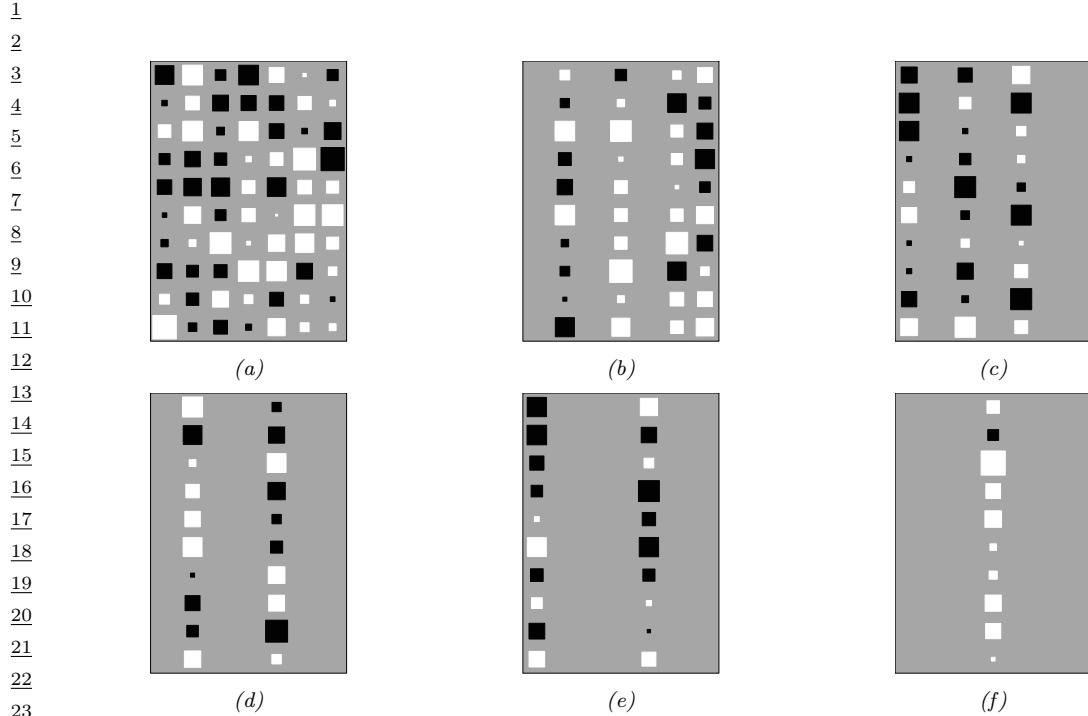


Figure 29.14: Illustration of estimating the effective dimensionalities in a mixture of factor analysers using variational Bayes EM with an ARD prior. Black are negative values, white are positive, gray is 0. The blank columns have been forced to 0 via the ARD mechanism, reducing the effective dimensionality. The data was generated from 6 clusters with intrinsic dimensionalities of 7, 4, 3, 2, 2, 1, which the method has successfully estimated. From Figure 4.4 of [Bea03]. Used with kind permission of Matt Beal.

29

30

31

number of points per cluster	intrinsic dimensionalities						
	1	7	4	3	2	2	
8		2		1			
8	1		2				
16	1		4			2	
32	1	6	3	3	2	2	
64	1	7	4	3	2	2	
128	1	7	4	3	2	2	

Figure 29.15: We show the estimated number of clusters, and their estimated dimensionalities, as a function of sample size. The ARD algorithm found two different solutions when $N = 8$. Note that more clusters, with larger effective dimensionalities, are discovered as the sample sizes increases. From Table 4.1 of [Bea03]. Used with kind permission of Matt Beal.

41

42

43

44

45

46

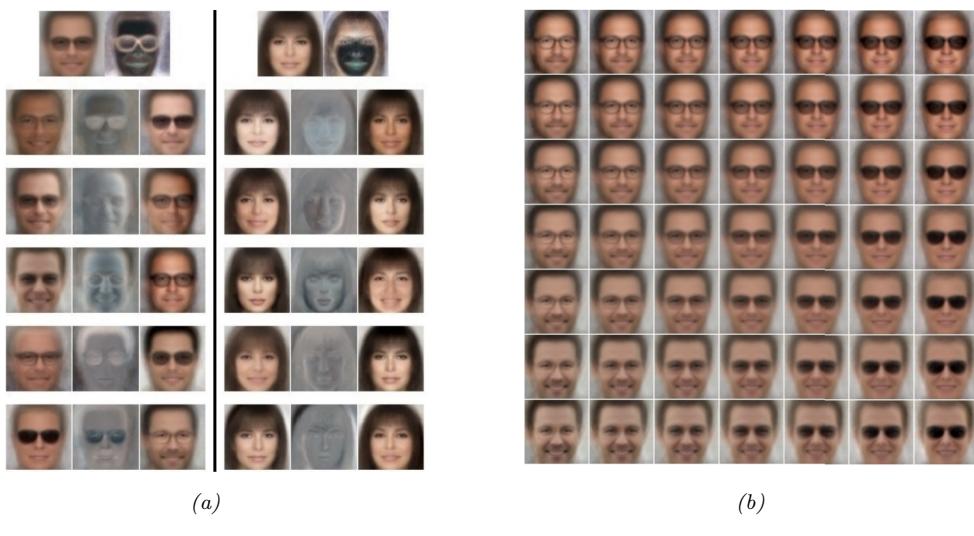
47

1
2
3
4
5
6
7
8
9



10 *Figure 29.16: Random samples from the MixFA model fit to CelebA. Generated by mix_PPCCA_celeba.ipynb.*
 11 *Adapted from Figure 4 of [RW18]. Used with kind permission of Yair Weiss*

12



30 *Figure 29.17: (a) Visualization of the parameters learned by the MFA model. The top row shows the mean*
 31 *μ_k and noise variance Ψ_k , reshaped from 12,288-dimensional vectors to $64 \times 64 \times 3$ images, for two mixture*
 32 *components k . The next 5 rows show the first 5 (of 10) basis functions (columns of \mathbf{W}_k) as images. On*
 33 *row i , left column, we show $\mu_k - \mathbf{W}_k[:, i]$; in the middle, we show $0.5 + \mathbf{W}_k[:, i]$, and on the right we show*
 34 *$\mu_k + \mathbf{W}_k[:, i]$. (b) Images generated by computing $\mu_k + z_1 \mathbf{W}_k[:, i] + z_2 \mathbf{W}_k[:, j]$, for some component k and*
 35 *dimensions i, j , where (z_1, z_2) are drawn from the grid $[-1 : 1, -1 : 1]$, so the central image is just μ_k . From*
 36 *Figure 6 of [RW18]. Used with kind permission of Yair Weiss*

37

38

39 matrix. Surprisingly, the results are competitive with deep generative models such as those in Part IV,
 40 despite the fact that no neural networks are used in the model.

41 In [RW18], they fit the MFA model to the CelebA dataset, which is a dataset of faces of celebrities
 42 (movie stars). They use $K = 300$ components, each of latent dimension $L = 10$; the observed data
 43 has dimension $D = 64 \times 64 \times 3 = 12,288$. They fit the model using SGD, using the methods from
 44 Section 29.3.1.3 to efficiently compute the log likelihood, despite the high dimensionality. The μ_k
 45 parameters are initialized using K-means clustering, and the \mathbf{W}_k parameters are initialized using
 46 factor analysis for each component separately. Then the model is fine-tuned end-to-end.

47

1
2
3
4
5
6
7
8
9
10
11
12
13



14 *Figure 29.18: Samples from the 100 CelebA images with lowest likelihood under the MFA model. Generated*
15 *by mix_PPCA_celeba.ipynb. Adapted from Figure 7a of [RW18]. Used with kind permission of Yair Weiss*

16
17
18
19
20
21
22
23
24
25
26
27
28
29



30 *Figure 29.19: Illustration of image imputation using an MFA. Left column shows 4 original images. Subsequent*
31 *pairs of columns show an occluded input, and a predicted output. Generated by mix_PPCA_celeba.ipynb.*
32 *Adapted from Figure 7b of [RW18]. Used with kind permission of Yair Weiss*

33
34
35

36 Figure 29.16 shows some images generated from the fitted model. The results are surprisingly good
37 for such a simple locally linear model. The reason the method works is similar to the discussion
38 in Section 29.2.4.1: essentially the \mathbf{W}_k matrix learns a set of L -dimensional basis functions for the
39 subset of face images that get mapped to cluster k . See Figure 29.17 for an illustration.

40 There are several advantages to this model compared to VAEs and GANs. First, [RW18], showed
41 that this MixFA model captures more of the modes of the data distribution than more sophisticated
42 generative models, such as VAEs (Section 22.2) and GANs (Chapter 27). Second, we can compute
43 the exact likelihood $p(\mathbf{x})$, so we can compute outliers or unusual images. This is illustrated in
44 Figure 29.18.

45 Third, we can perform image imputation from partially observed images given arbitrary missingness
46 patterns. To see this, let us partition $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, where \mathbf{x}_1 (of size D_1) is observed and \mathbf{x}_2 (of size
47

$D_2 = D - D_1$) is missing. We can compute the most probable cluster using

$$k^* = \operatorname{argmax}_{k=1}^K p(c=k)p(\mathbf{x}_1|c=k) \quad (29.113)$$

where

$$\log p(\mathbf{x}_1|\boldsymbol{\mu}_k, \mathbf{C}_k) = -\frac{1}{2} \left[D_1 \log(2\pi) + \log \det(\mathbf{C}_{k,11}) + \tilde{\mathbf{x}}_1^\top \mathbf{C}_{k,11}^{-1} \tilde{\mathbf{x}}_1 \right] \quad (29.114)$$

where $\mathbf{C}_{k,11}$ is the top left $D_1 \times D_1$ block of $\mathbf{W}_k \mathbf{W}_k^\top + \boldsymbol{\Psi}_k$, and $\tilde{\mathbf{x}}_1 = \mathbf{x}_1 - \boldsymbol{\mu}_k[1:D_1]$. Once we know which discrete mixture component to use, we can compute the Gaussian posterior $p(\mathbf{z}|\mathbf{x}_1, k^*)$ using Equation (29.36). Let $\hat{\mathbf{z}} = \mathbb{E}[\mathbf{z}|\mathbf{x}_1, k^*]$. Given this, we can compute the predicted output for the full image:

$$\hat{\mathbf{x}} = \mathbf{W}_{k^*} \hat{\mathbf{z}} + \boldsymbol{\mu}_{k^*} \quad (29.115)$$

We then use the estimate $\mathbf{x}' = [\mathbf{x}_1, \hat{\mathbf{x}}_2]$, so the observed pixels are not changed. This is an example of **image imputation**, and is illustrated in Figure 29.19. Note that we can condition on an arbitrary subset of pixels, and fill in the rest, whereas some other models (e.g., autoregressive models) can only predict the bottom right given the top left (since they assume a generative model which works in raster-scan order).

29.5 LVMs with non-Gaussian priors

In this section, we discuss (linear) latent factor models with non-Gaussian priors. See Table 29.1 for a summary of the models we will discuss.

29.5.1 Non-negative matrix factorization (NMF)

Suppose that we use a gamma distribution for the latents: $p(\mathbf{z}) = \prod_k \text{Ga}(z_{1,k}|\alpha_k, \beta_k)$. This results in a sparse, non-negative hidden representation, which can help interpretability. This is particularly useful when the data is also sparse and non-negative, such as word counts. In this case, it makes sense to use a Poisson likelihood: $p(\mathbf{x}|\mathbf{z}) = \prod_{d=1}^D \text{Poi}(x_d|\mathbf{w}_d^\top \mathbf{z})$. The overall model has the form

$$p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \left[\prod_k \text{Ga}(z_{1,k}|\alpha_k, \beta_k) \right] \left[\prod_{d=1}^D \text{Poi}(x_d|\mathbf{w}_d^\top \mathbf{z}) \right] \quad (29.116)$$

The resulting model is called the **GaP** (Gamma-Poisson) model [Can04]. See Figure 29.20a for the graphical model.

The parameters α_k and β_k control the sparsity of the latent representation \mathbf{z}_n . If we set $\alpha_k = \beta_k = 0$, and compute the MLE for \mathbf{W} , we recover **non-negative matrix factorization (NMF)** [PT94; LS99; LS01], as shown in [BJ06].

Figure 29.21 illustrates the result of applying NMF to a dataset of image patches of faces, where the data correspond to non-negative pixel intensities. We see that the learned basis functions are small localized **parts** of faces. Also, the coefficient vector \mathbf{z} is sparse and positive. For PCA, the coefficient vector has negative values, and the resulting basis functions are global, not local. For

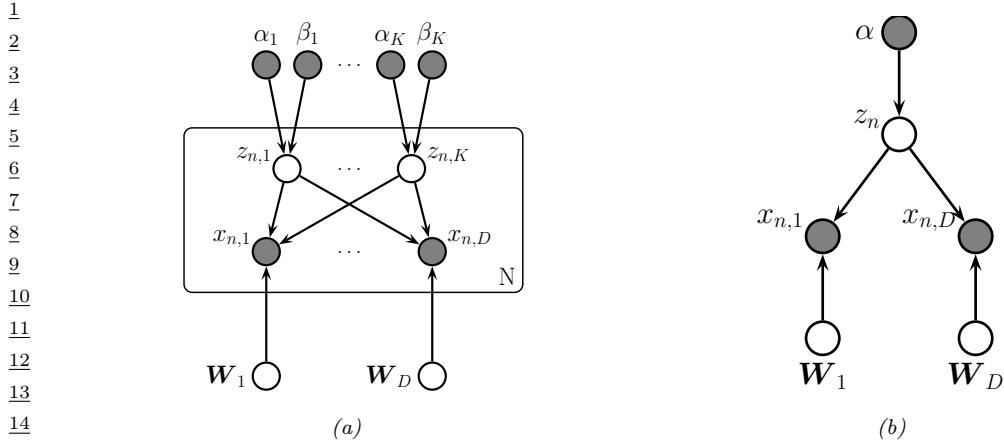


Figure 29.20: (a) Gaussian-Poisson (GAP) model as a PGM-D. Here $z_{nk} \in \mathbb{R}^+$ and $x_{n,d} \in \mathbb{Z}_{\geq 0}$. (b) Simplex FA model as a PGM-D. Here $\mathbf{z}_n \in \mathbb{S}_K$ and $x_{n,d} \in \{1, \dots, V\}$.

vector quantization (i.e., GMM model), \mathbf{z} is a one-hot vector, with a single mixture component turned on; the resulting weight vectors correspond to entire image prototypes. The reconstruction quality is similar in each case, but the nature of the learned latent representation is quite different.

29.5.2 Multinomial PCA

Suppose we use a Dirichlet prior for the latents, $p(\mathbf{z}) = \text{Dir}(\mathbf{z}|\boldsymbol{\alpha})$, so $\mathbf{z} \in \mathbb{S}_K$, which is the K -dimensional probability simplex. As in Section 29.5.1, the vector \mathbf{z} will be sparse and non-negative, but in addition it will satisfy the constraint $\sum_{k=1}^K z_k = 1$, so the components are not independent. Now suppose our data is categorical, $x_d \in \{1, \dots, V\}$, so our likelihood has the form $p(\mathbf{x}|\mathbf{z}) = \prod_d \text{Cat}(x_d|\mathbf{W}_d \mathbf{z})$. The overall model is therefore

$$p(\mathbf{z}, \mathbf{x}) = \text{Dir}(\mathbf{z}|\boldsymbol{\alpha}) \prod_{d=1}^D \text{Cat}(x_d|\mathbf{W}_d \mathbf{z}) \quad (29.117)$$

See Figure 29.20b for the PGM-D. This model (or small variants of it) has multiple names: **user rating profile model** [Mar03], **admixture model** [PSD00], **mixed membership model** [EFL04], **multinomial PCA (mPCA)** [BJ06], or **simplex factor analysis (sFA)** [BD11].

29.5.2.1 Example: roll call data

Let us consider the example from [BJ06], who applied this model to analyze some **roll call** data from the US Senate in 2003. Specifically, the data has the form $x_{n,d} \in \{+1, -1, 0\}$ for $n = 1 : 100$ and $d = 1 : 459$, where x_{nd} is the vote of the n 'th senator on the d 'th bill, where +1 means in favor, -1 means against, and 0 means not voting. In addition, we have the overall outcome, which we denote by $x_{101,d} \in \{+1, -1\}$, where +1 means the bill was passed, and -1 means it was rejected.

We fit the mPCA model to this data using 5 latent factors using variational EM. Figure 29.22 plots $\mathbb{E}[z_{nk}|\mathbf{x}_n] \in [0, 1]$, which is the degree to which senator n belongs to latent component or “bloc”

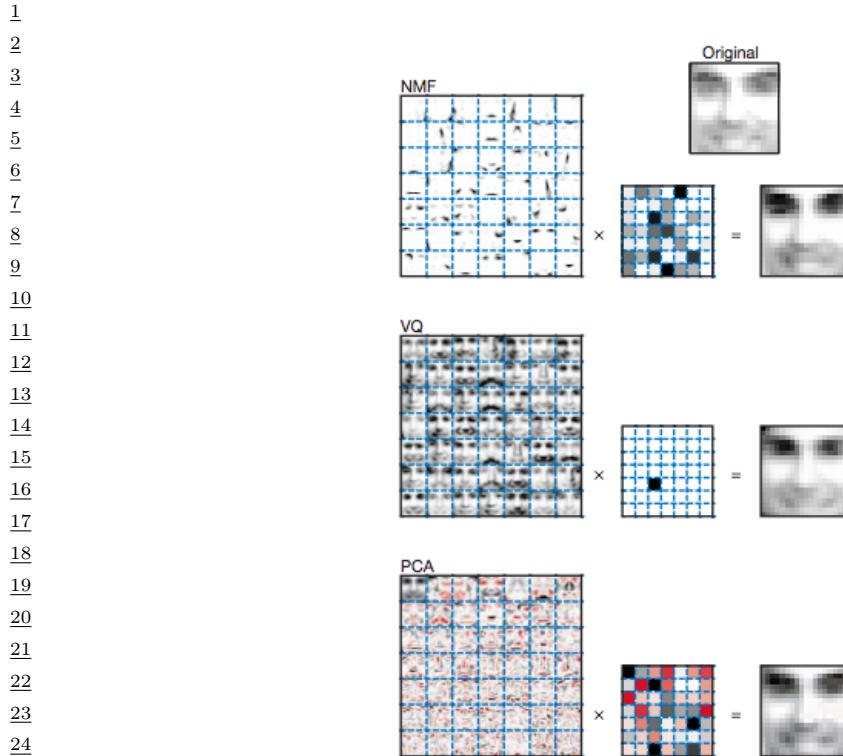


Figure 29.21: Illustrating the difference between Non-negative Matrix Factorization (NMF), Vector Quantization (VQ), and Principal Components Analysis (PCA). Left column: Filters (columns of \mathbf{W}) learned from a set of 2429 faces images, each of size 19×19 . There are 49 basis functions in total, shown in a 7×7 montage; each filter is reshaped to a 19×19 image for display purposes. (For PCA, negative weights are red, positive weights are black.) Middle column: The 49 latent factors \mathbf{z} when the model is applied to the original face image shown at the top. Right column: reconstructed face image. From Figure 1 of [LS99].

k. We see that component 5 is the Democratic majority, and block 2 is the Republican majority. See [BJ06] for further details.

29.5.2.2 Advantage of Dirichlet prior over Gaussian prior

The main advantage of using a Dirichlet prior compared to a Gaussian prior is that the latent factors are more interpretable. To see this, note that the mean parameters for d 'th output distribution have the form $\boldsymbol{\mu}_{nd} = \mathbf{W}^d \mathbf{z}_n$, and hence

$$p(x_{nd} = v | \mathbf{z}_n) = \sum_k z_{nk} w_{kv}^d \quad (29.118)$$

Thus the latent variables define the mean parameters. By contrast, the CatPCA model in Section 29.3.4.2 uses a Gaussian prior, so $\mathbf{W}^d \mathbf{z}_n$ can be negative; consequently it must pass this vector

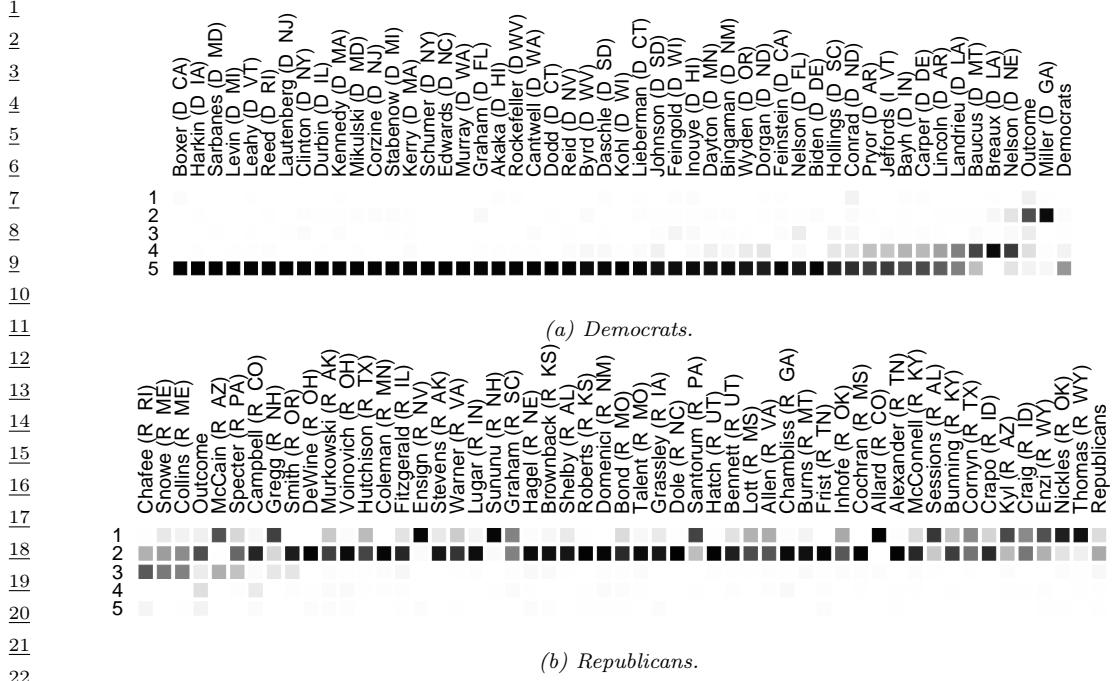


Figure 29.22: The simplex factor analysis model applied to some roll call data from the US Senate collected in 2003. The senators have been sorted from left to right using the binary PCA method of [Lee06]. See text for details. From Figures 8–9 of [BJ06]. Used with kind permission of Wray Buntine.

through a softmax, to convert from natural parameters to mean parameters; this makes \mathbf{z}_n harder to interpret.

29.5.2.3 Connection to mixture models

If \mathbf{z}_n were a one-hot vector, the mPCA model would be equivalent to selecting a single column from \mathbf{W}_d corresponding to the discrete hidden state:

$$p(x_{nd} = v | z_n) = \sum_{k=1}^K \mathbb{I}(z_n = k) w_{kv}^d \quad (29.119)$$

This is equivalent to a finite mixture of categorical distributions (c.f., Section 29.2.2), and corresponds to the assumption that \mathbf{x} is generated by a single cluster. However, the mPCA model does not require that \mathbf{z}_n be one-hot, and instead allows \mathbf{x}_n to partially belong to multiple clusters. For this reason, this model is also known as an **admixture mixture** or **mixed membership model** [EFL04].

29.5.3 Latent Dirichlet Allocation (LDA)

In this section, we discuss a simple extension of the multinomial PCA model of Section 29.5.2 in which \mathbf{x} is a variable-length sequence of tokens, and the weights are tied across “time”. Thus the

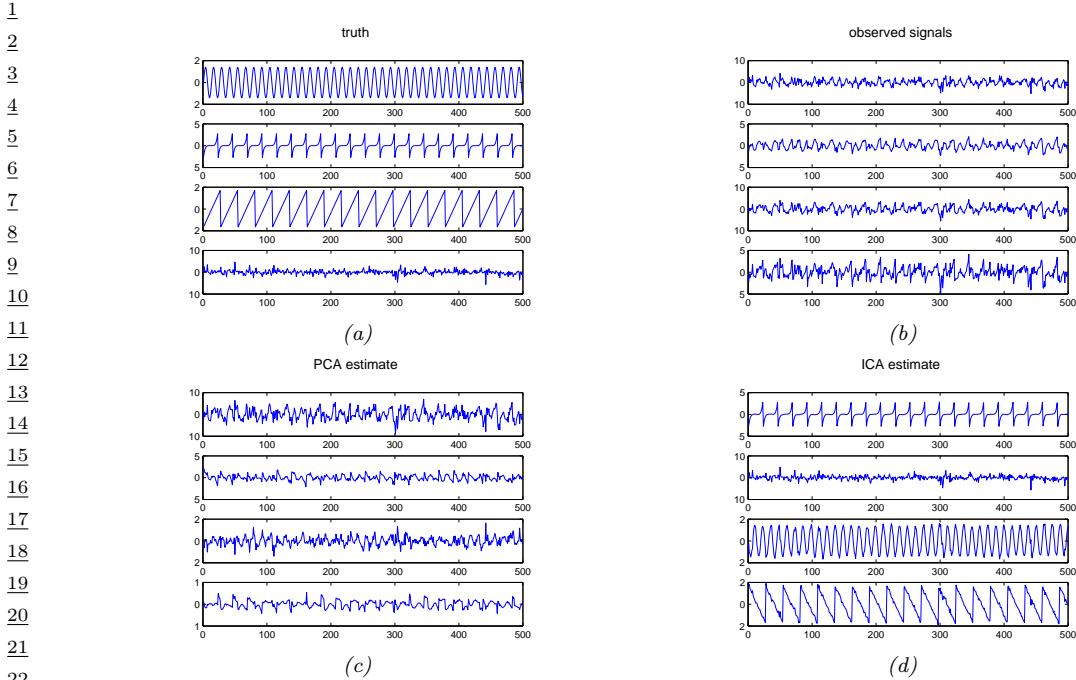


Figure 29.23: Illustration of ICA applied to 500 iid samples of a 4d source signal. (a) Latent signals. (b) Observations. (c) PCA estimate. (d) ICA estimate. This matches the true sources, up to permutation of the dimension indices. Generated by `ica_demo.py`.

model can be defined as follows:

$$p(\mathbf{z}, \mathbf{x}) = \text{Dir}(\mathbf{z}|\boldsymbol{\alpha}) \prod_{t=1}^T \text{Cat}(x_t|\mathbf{W}\mathbf{z}) \quad (29.120)$$

This is known as **latent Dirichlet allocation** or **LDA** [BNJ03a; Ble12; BGHM17]. Due to lack of space, we discuss this model in more detail in the supplementary material.

29.6 Independent components analysis (ICA)

Consider the following situation. You are in a crowded room and many people are speaking. Your ears essentially act as two microphones, which are listening to a linear combination of the different speech signals in the room. Your goal is to deconvolve the mixed signals into their constituent parts. This is known as the **cocktail party problem**, or the **blind source separation (BSS)** problem, where “blind” means we know “nothing” about the source of the signals. Besides the obvious applications to acoustic signal processing, this problem also arises when analysing EEG and MEG signals, financial data, and any other dataset (not necessarily temporal) where latent sources or factors get mixed together in a linear way. See Figure 29.23 for an example.

1 **29.6.1 Noiseless ICA model**

3 We can formalize the problem as follows. Let $\mathbf{x}_n \in \mathbb{R}^D$ be the vector of observed responses, at “time”
4 n , where D is the number of sensors / microphones. Let $\mathbf{z}_n \in \mathbb{R}^D$ be the hidden vector of source
5 signals at time n , of the same dimensionality as the observed signal. We assume that
6

7
$$\mathbf{x}_n = \mathbf{A}\mathbf{z}_n \tag{29.121}$$

8 where \mathbf{A} is an invertible $D \times D$ matrix known as the **mixing matrix** or the **generative weights**.
9 The prior has the form $p(\mathbf{z}_n) = \prod_{j=1}^D p_j(z_j)$. Typically we assume this is a sparse prior, so only a
10 subset of the signals are active at any one time (see Section 29.6.2 for further discussion of priors for
11 this model). This model is called **independent components analysis** or **ICA**, since we assume
12 that each observation \mathbf{x}_n is a linear combination of independent components represented by sources
13 \mathbf{z}_n , i.e,

14
$$x_{nj} = \sum_i A_{ij} z_{nj} \tag{29.122}$$

15 Our goal is to infer the source signals, $p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{A})$. Since the model is noiseless, we have

16
$$p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{A}) = \delta(\mathbf{z}_n - \mathbf{B}\mathbf{x}_n) \tag{29.123}$$

17 where $\mathbf{B} = \mathbf{A}^{-1}$ are the **recognition weights**. (We discuss how to estimate these weights in
18 Section 29.6.3.)

19 **29.6.2 The need for non-Gaussian priors**

20 Since $\mathbf{x} = \mathbf{A}\mathbf{z}$, we have $\mathbb{E}[\mathbf{x}] = \mathbf{A}\mathbb{E}[\mathbf{z}]$ and $\text{Cov}[\mathbf{x}] = \text{Cov}[\mathbf{A}\mathbf{z}] = \mathbf{A}\text{Cov}[\mathbf{z}]\mathbf{A}^\top$. Without loss of
21 generality, we can assume $\mathbb{E}[\mathbf{z}] = \mathbf{0}$, since we can always center the data. Similarly, we can assume
22 $\text{Cov}[\mathbf{z}] = \mathbf{I}$, since $\mathbf{A}\mathbf{A}^\top$ can capture any correlation in \mathbf{x} . Thus \mathbf{z} is a set of D unit variance,
23 uncorrelated variables, as in factor analysis (Section 29.3.1).

24 However, this is not sufficient to uniquely identify \mathbf{A} and hence \mathbf{z} , as we explained in Section 29.3.1.4.
25 So we need to go beyond an uncorrelated prior and enforce an independent, and non-Gaussian, prior.
26 To illustrate this, suppose we have two independent sources with uniform distributions, as shown
27 in Figure 29.24(a). Now suppose we have the following mixing matrix

28
$$\mathbf{A} = \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \tag{29.124}$$

29 Then we observe the data shown in Figure 29.24(b) (assuming no noise). The full-rank PCA model
30 (where $K = D$) is equivalent to ICA, except it uses a factored Gaussian prior for \mathbf{z} . The result of
31 using PCA is shown in Figure 29.24(c). This corresponds to a **whitening** or **sphering** of the data,
32 in which $\text{Cov}[\mathbf{z}] = \mathbf{I}$. To uniquely recover the sources, we need to perform an additional rotation.
33 The trouble is, there is no information in the symmetric Gaussian posterior to tell us which angle to
34 rotate by. In a sense, PCA solves “half” of the problem, since it identifies the linear subspace; all
35 that ICA has to do is then to identify the appropriate rotation. To do this, ICA uses an independent,
36 but non-Gaussian, prior. The result is shown in Figure 29.24(d). This shows that ICA can recover
37 the source variables, up to a permutation of the indices and possible sign change.

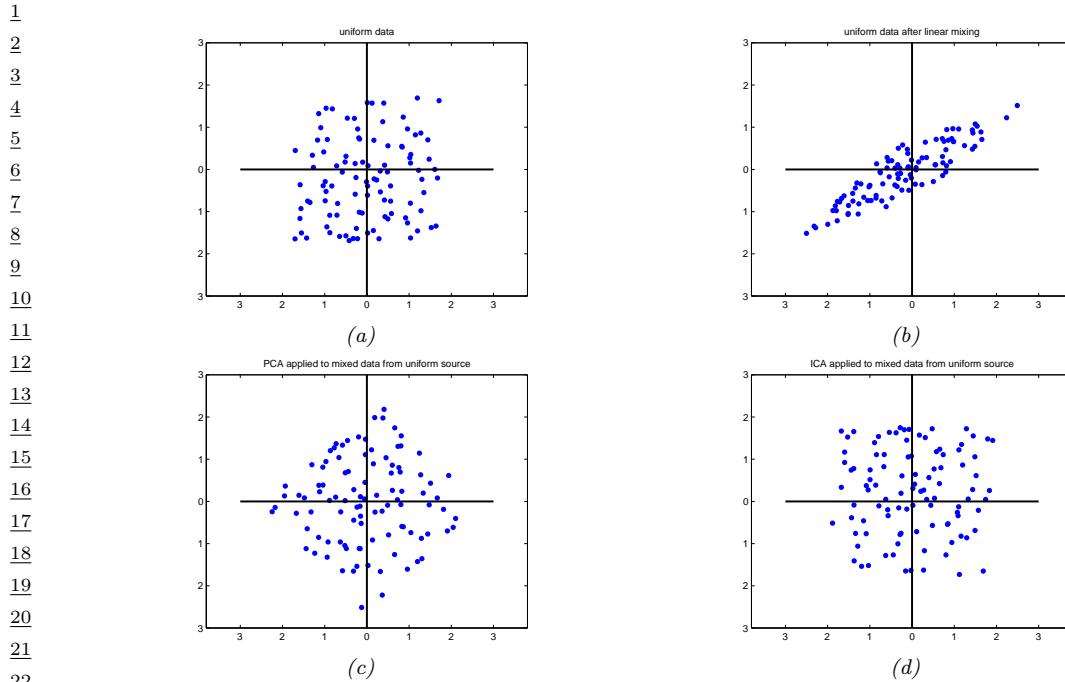


Figure 29.24: Illustration of ICA and PCA applied to 100 iid samples of a 2d source signal with a uniform distribution. (a) Latent signals. (b) Observations. (c) PCA estimate. (d) ICA estimate. Generated by `ica_demo_uniform.py`.

We typically use a prior which is a super-Gaussian distribution, meaning it has heavy tails; this helps with identifiability. One option is to use a Laplace prior. For mean zero and variance 1, this has a log pdf given by

$$\log p(z) = -\sqrt{2}|z| - \log(\sqrt{2}) \quad (29.125)$$

However, since the Laplace prior is not differentiable at the origin, in ICA it is more common to use the logistic distribution, discussed in Section 15.4.1. The corresponding log pdf, for the case where the mean is zero and the variance is 1, is given by the following:

$$\log p(z) = -2 \log \cosh\left(\frac{\pi}{2\sqrt{3}}z\right) - \log \frac{4\sqrt{3}}{\pi} \quad (29.126)$$

29.6.3 Maximum likelihood estimation

Since $\mathbf{x} = \mathbf{A}\mathbf{z}$, from the change of variables formula, the density of the observed data is given by

$$p_x(\mathbf{x}) = p_z(\mathbf{z}) |\det(\mathbf{A}^{-1})| = p_z(\mathbf{B}\mathbf{x}) |\det(\mathbf{B})| \quad (29.127)$$

¹ where $\mathbf{B} = \mathbf{A}^{-1}$. To maximize this, we can simplify the problem as follows. Let $\Sigma = \text{Cov}[\mathbf{x}]$, and
² define some nonsingular matrix \mathbf{V} such that $\mathbf{B} = \mathbf{A}^{-1} = \mathbf{V}\Sigma^{-\frac{1}{2}}$. Then
³

$$\underline{z} = \mathbf{B}\mathbf{x} = \mathbf{V}\Sigma^{-\frac{1}{2}}\mathbf{x} \quad (29.128)$$

⁷ Since we assumed $\text{Cov}[\mathbf{z}] = \mathbf{I}$, we have
⁸

$$\underline{9} \quad \text{Cov}[\mathbf{z}] = \mathbf{V}\Sigma^{-\frac{1}{2}}\Sigma\Sigma^{\frac{1}{2}}\mathbf{V}^T = \mathbf{V}\mathbf{V}^T = \mathbf{I} \quad (29.129)$$

¹¹ Hence \mathbf{V} is orthogonal. So if we sphere the data, by computing $\tilde{\mathbf{x}} = \Sigma^{-1/2}\mathbf{x}$, we will have $\mathbf{B} = \mathbf{V}$.
¹² So now our goal simplifies to estimating an orthogonal \mathbf{V} from the whitened data.³

¹³ With this new notation, we can write the likelihood as
¹⁴

$$\underline{15} \quad p_x(\mathbf{x}) = p_z(\mathbf{V}\mathbf{x})|\det(\mathbf{V})| \quad (29.130)$$

¹⁷ Thus the average negative log likelihood is given by
¹⁸

$$\underline{19} \quad \text{NLL}(\mathbf{V}) = -\frac{1}{N} \log p(\mathbf{X}|\mathbf{V}) = -\log |\det(\mathbf{V})| - \frac{1}{N} \sum_{j=1}^L \sum_{n=1}^N \log p_j(\mathbf{v}_j^T \mathbf{x}_n) \quad (29.131)$$

²³ where \mathbf{v}_j is the j 'th row of \mathbf{V} . Since we are constraining \mathbf{V} to be orthogonal, the $\log |\det(\mathbf{V})|$ term
²⁴ is a constant, so we can drop it. We can also replace the sum over n with an expectation wrt the
²⁵ empirical distribution to get the following objective
²⁶

$$\underline{27} \quad \text{NLL}(\mathbf{V}) = \sum_j \mathbb{E}[G_j(z_j)] \quad (29.132)$$

³⁰ where $z_j = \mathbf{v}_j^T \mathbf{x}$ and $G_j(z) \triangleq -\log p_j(z)$. We want to minimize this (nonconvex) objective subject to
³¹ the constraint that \mathbf{V} is an orthonormal matrix.
³²

³³ It is straightforward to derive a (projected) gradient descent algorithm to fit this model; however,
³⁴ it is rather slow. One can also derive a faster algorithm that follows the natural gradient; see e.g.,
³⁵ [Mac03, ch 34] for details. However, the most popular method is to use an approximate Newton
³⁶ method, known as **fast ICA** [HO00]. This was used to produce Figure 29.23.

³⁷

³⁸ 29.6.4 Alternatives to MLE

³⁹

⁴⁰ In this section, we discuss various alternatives estimators for ICA that have been proposed over the
⁴¹ years. We will show that they are equivalent to MLE. However, they bring interesting perspectives
⁴² to the problem.

⁴³

⁴⁴ 3. Traditionally in the ICA literature the stated goal is to estimate the orthogonal matrix \mathbf{W} , but this notation
⁴⁵ conflicts with our use of \mathbf{W} as generative weights in the factor analysis model of Section 29.3.1. So we use the letter \mathbf{V}
⁴⁶ instead.

⁴⁷

1 **29.6.4.1 Maximizing non-Gaussianity**

3 An early approach to ICA was to find a matrix \mathbf{V} such that the distribution $\mathbf{z} = \mathbf{V}\mathbf{x}$ is as far from
4 Gaussian as possible. (There is a related approach in statistics called **projection pursuit** [FT74].)
5 One measure of non-Gaussianity is kurtosis, but this can be sensitive to outliers. Another measure is
6 the **negentropy**, defined as
7

$$\text{negentropy}(z) \triangleq \mathbb{H}(\mathcal{N}(\mu, \sigma^2)) - \mathbb{H}(z) \quad (29.133)$$

8 where $\mu = \mathbb{E}[z]$ and $\sigma^2 = \mathbb{V}[z]$. Since the Gaussian is the maximum entropy distribution, this
9 measure is always non-negative and becomes large for distributions that are highly non-Gaussian.

10 We can define our objective as maximizing

$$J(\mathbf{V}) = \sum_j \text{negentropy}(z_j) = \sum_j \mathbb{H}(\mathcal{N}(\mu_j, \sigma_j^2)) - \mathbb{H}(z_j) \quad (29.134)$$

11 where $\mathbf{z} = \mathbf{V}\mathbf{x}$. Since we assume $\mathbb{E}[\mathbf{z}] = \mathbf{0}$ and $\text{Cov}[\mathbf{z}] = \mathbf{I}$, the first term is a constant. Hence

$$J(\mathbf{V}) = \sum_j -\mathbb{H}(z_j) + \text{const} = \sum_j \mathbb{E}[\log p(z_j)] + \text{const} \quad (29.135)$$

12 which we see is equal (up to a sign change, and irrelevant constants) to the log-likelihood in
13 Equation (29.132).

14 **29.6.4.2 Minimizing total correlation**

15 In Section 5.3.5.1, we show that the total correlation of \mathbf{z} is given by

$$\text{TC}(\mathbf{z}) = \sum_j \mathbb{H}(z_j) - \mathbb{H}(\mathbf{z}) = D_{\text{KL}} \left(p(\mathbf{z}) \middle\| \prod_j p_k(z_j) \right) \quad (29.136)$$

16 This is zero iff the components of \mathbf{z} are all mutually independent. In Section 22.3.2.2, we show that
17 minimizing this results in a representation that is **disentangled**.

18 Now since $\mathbf{z} = \mathbf{V}\mathbf{x}$, we have

$$\text{TC}(\mathbf{z}) = \sum_j \mathbb{H}(z_j) - \mathbb{H}(\mathbf{V}\mathbf{x}) \quad (29.137)$$

20 Since we constrain \mathbf{V} to be orthogonal, we can drop the last term, since $\mathbb{H}(\mathbf{V}\mathbf{x}) = \mathbb{H}(\mathbf{x}) = \text{const}$, since
21 multiplying by \mathbf{V} does not change the shape of the distribution. Hence we have $\text{TC}(\mathbf{z}) = \sum_k \mathbb{H}(z_k)$.
22 Minimizing this is equivalent to maximizing the negentropy, which is equivalent to maximum
23 likelihood.

24 **29.6.4.3 Maximizing mutual information (InfoMax)**

25 Let $z_j = \phi(v_j^\top \mathbf{x}) + \epsilon$ be the noisy output of an encoder, where ϕ is some nonlinear scalar function,
26 and $\epsilon \sim \mathcal{N}(0, 1)$. It seems reasonable to try to maximize the information flow through this system, a
27

¹ principle known as **infomax** [Lin88b; BS95]. That is, we want to maximize the mutual information
² between \mathbf{z} (the internal neural representation) and \mathbf{x} (the observed input signal). We have $I(\mathbf{x}; \mathbf{z}) =$
³ $H(\mathbf{z}) - H(\mathbf{z}|\mathbf{x})$, where the latter term is constant if we assume the noise has constant variance. One
⁴ can show that we can approximate the former term as follows
⁵

$$\frac{6}{7} \quad H(\mathbf{z}) = \sum_j \mathbb{E} [\log \phi'(\mathbf{v}_j^\top \mathbf{x})] + \log |\det(\mathbf{V})| \quad (29.138)$$

$$\frac{8}{}$$

⁹ where, as usual, we can drop the last term if \mathbf{V} is orthogonal. If we define $\phi(z)$ to be a cdf, then
¹⁰ $\phi'(z)$ is its pdf, and the above expression is equivalent to the log likelihood. In particular, if we
¹¹ use a logistic nonlinearity, $\phi(z) = \sigma(z)$, then the corresponding pdf is the logistic distribution, and
¹² $\log \phi'(z) = \log \cosh(z)$, which matches Equation (29.126) (ignoring irrelevant constants). Thus we
¹³ see that infomax is equivalent to maximum likelihood.
¹⁴

¹⁵ 29.6.5 Sparse coding

¹⁶

¹⁷ In this section, we consider an extension of ICA to the case where we allow for observation noise
¹⁸ (using a Gaussian likelihood), and we allow for a non-square mixing matrix \mathbf{W} . We also use a Laplace
¹⁹ prior for \mathbf{z} . The resulting model is as follows:

$$\frac{20}{21} \quad p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \left[\prod_k \text{Lap}(z_k|\lambda) \right] \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z}, \sigma^2 \mathbf{I}) \quad (29.139)$$

$$\frac{22}{23}$$

²⁴ Thus each observation \mathbf{x} is approximated by a sparse combination of columns of \mathbf{W} , known as **basis**
²⁵ **functions**; the sparse vector of weights is given by \mathbf{z} . (This can be thought of as a form of sparse
²⁶ factor analysis, except the sparsity is in the latent code \mathbf{z} , not the weight matrix \mathbf{W} .)

²⁷ Not all basis functions will be active for any given observation, due to the sparsity penalty.
²⁸ Hence we can allow for more latent factors K than observations D . This is called **overcomplete**
²⁹ **representation**.

³⁰ If we have a batch of N examples, stored in the rows of \mathbf{X} , the log joint becomes

$$\frac{31}{32} \quad \log p(\mathbf{X}, \mathbf{Z}|\mathbf{W}) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{W}\mathbf{z}_n\|_2^2 + \lambda \|\mathbf{z}_n\|_1 = \frac{1}{2} \|\mathbf{X} - \mathbf{W}\mathbf{Z}\|_F^2 + \lambda \|\mathbf{Z}\|_{1,1} \quad (29.140)$$

$$\frac{33}{34}$$

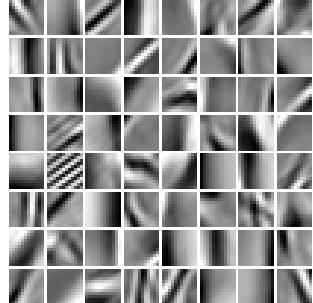
³⁵ The MAP inference problem consists of estimating \mathbf{Z} for a fixed \mathbf{W} ; this is known as **sparse coding**,
³⁶ and can be solved using standard algorithms for sparse linear regression (see Section 15.2.5).⁴
³⁷ The learning problem consists of estimating \mathbf{W} , marginalizing out \mathbf{Z} . This is called **dictionary**
³⁸ **learning**. Since this is computationally difficult, it is common to jointly optimize \mathbf{W} and \mathbf{Z} (thus
³⁹ “maxing out” \mathbf{Z} instead of marginalizing it out). We can do this by applying alternating optimization
⁴⁰ to Equation (29.140): estimating \mathbf{Z} given \mathbf{W} is a sparse linear regression problem, and estimating \mathbf{W}
⁴¹ given \mathbf{Z} is a simple least squares problem. (For faster algorithms, see [Mai+10].)

⁴² Figure 29.25(a) illustrates the results of dictionary learning when applied to a dataset of natural
⁴³ image patches. (Each patch is first centered and normalized to unit norm.) We see that the method

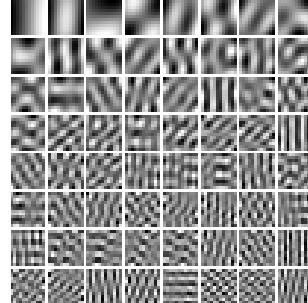
⁴⁴ 4. Solving an ℓ_1 optimization problem for each data example can be slow. However, it is possible to train a neural
⁴⁵ network to approximate the outcome of this process; this is known as **predictive sparse decomposition** [KRL08;
⁴⁶ GL10].

⁴⁷

1
2
3
4
5
6
7
8
9
10
11
12



(a)



(b)

13
14
15

Figure 29.25: Illustration of the filters learned by various methods when applied to natural image patches. (a) Sparse coding. (b) PCA. Generated by [sparse_dict_demo.ipynb](#).

16

17 has learned bar and edge detectors that are similar to the simple cells in the primary visual cortex
18 of the mammalian brain [OF96]. By contrast, PCA results in sinusoidal gratings, as shown in
19 Figure 29.25(b).⁵
20

21
22

29.6.6 Nonlinear ICA

23
24
25

There are various ways to extend ICA to the nonlinear case. The resulting methods are similar to variational autoencoders (Chapter 22). For details, see e.g., [KKH20].

26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

5. The reason PCA discovers sinusoidal grating patterns is because it is trying to model the covariance of the data, which, in the case of image patches, is translation invariant. This means $\text{Cov}[I(x, y), I(x', y')] = f[(x - x')^2 + (y - y')^2]$ for some function f , where $I(x, y)$ is the image intensity at location (x, y) . One can show (see e.g., [HHH09, p125]) that the eigenvectors of a matrix of this kind are always sinusoids of different phases, i.e., PCA discovers a **Fourier basis**.

30 Hidden Markov models

30.1 Introduction

In Section 2.8, we discussed Markov models. However, the assumption that we only need to know the current observation, \mathbf{y}_t , to predict the future, $\mathbf{y}_{t+\tau}$, without needing to know the past, $\mathbf{y}_{1:t-1}$, is a very limiting assumption. To allow the model to have “infinite” memory, we need to use a **hidden variable** \mathbf{z}_t , which summarizes *all* the past history, $\mathbf{y}_{1:t}$.

We now discuss one model of this kind, known as a **hidden Markov model** or **HMM**. This corresponds to a joint distribution of the following form:

$$p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}) = \left[p(z_1) \prod_{t=2}^T p(z_t | z_{t-1}) \right] \left[\prod_{t=1}^T p(\mathbf{y}_t | z_t) \right] \quad (30.1)$$

We see that $\mathbf{z}_{1:T}$ corresponds to a first-order Markov chain in a latent state space, and at each time step, the model generates an observation \mathbf{y}_t . Thus we can think of an HMM as a **partially observed Markov process**. This is an example of a more general family of models known as state-space models, discussed in Chapter 31.

Figure 30.1a shows an HMM as a graphical model, where we unroll the model for 3 time steps. Figure 30.12 shows the corresponding plate version, where the parameter nodes are shown explicitly. The plates can capture how the parameters are shared across samples n (representing parameter tying), but it cannot capture the repetitive structure of the temporal “backbone” of the model.

30.2 HMMs: parameterization

30.2.1 Transition model

In an HMM, the hidden state z_t is discrete, so we can define the **transition model** as follows:

$$p(z_t = j | z_{t-1} = i) = A_{ij} \quad (30.2)$$

Here the i ’th row corresponds to the outgoing distribution from state i . This is a **row stochastic matrix**, meaning each row sums to one. We can visualize the non-zero entries in the transition matrix by creating a state transition diagram, as shown in Figure 2.16.

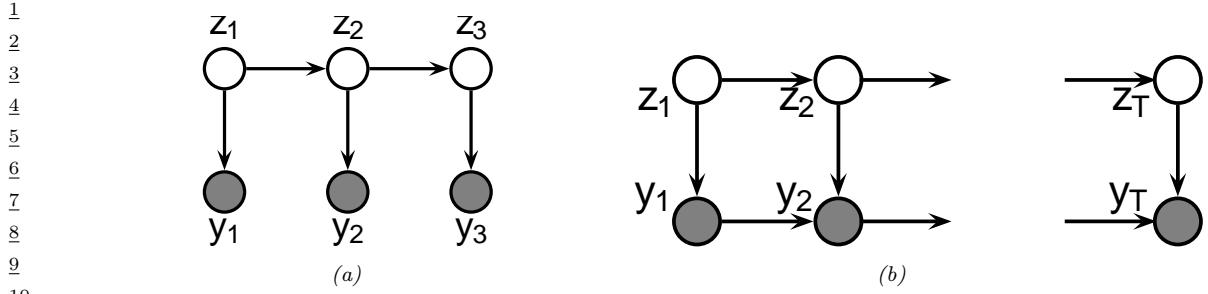


Figure 30.1: (a) An HMM represented as a PGM-D unrolled for 3 time steps. (b) An auto-regressive HMM of order 1.

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

30.2.2 Observation model

31

32 The term $p(\mathbf{y}_t|z_t = j)$ is the **observation model**. The form of this distribution depends on the
33 type of data, for example, whether it is discrete or continuous. We give some examples below.

34

30.2.2.1 Categorical likelihood

35

36 If y_t is discrete, it is common to represent $p(y_t|z_t)$ by a set of categorical distributions, with one
37 distribution per hidden state. In particular, we can define

$$38 \quad p(y_t = k|z_t = j) = B_{jk} \quad (30.3)$$

39

40 which is the probability of emitting symbol k from state j . Here \mathbf{B} is a row stochastic matrix, similar
41 to \mathbf{A} . See Section 8.1.2 for an example.

42 If we have D discrete observations per time step, we can use a factorial model of the form

$$43 \quad p(\mathbf{y}_t|z_t = j) = \prod_{d=1}^D \text{Cat}(y_{td}|\mathbf{B}_{d,j,:}) \quad (30.4)$$

44

45

46

47

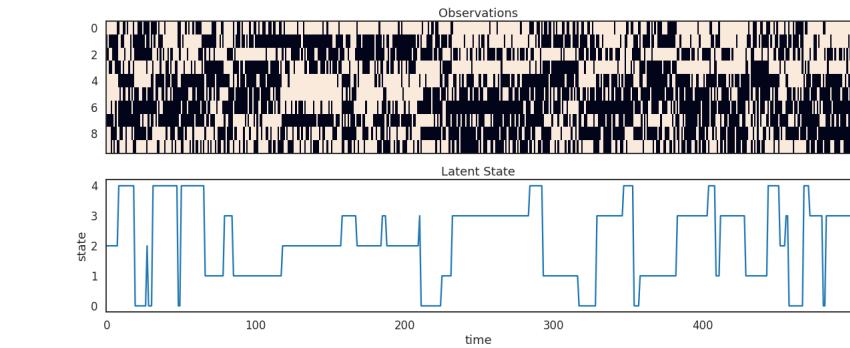


Figure 30.2: Some samples from an HMM with 5 Bernoulli observables. Generated by
beroulli_hmm_example.py.

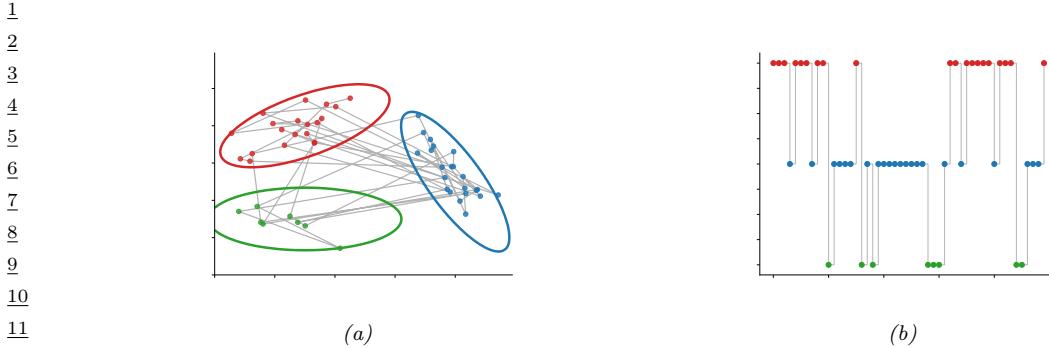


Figure 30.3: (a) Some 2d data sampled from a 3 state HMM. Each state emits from a 2d Gaussian. (b) The hidden state sequence. Adapted from Figure 13.8 of [Bis06]. Generated by `hmm_lillypad_demo.py`.

In the special case of binary observations, this becomes

$$p(\mathbf{y}_t | z_t = j) = \prod_{d=1}^D \text{Ber}(y_{td} | B_{d,j}) \quad (30.5)$$

In Figure 30.2, we give an example of an HMM with a 5d factored Bernoulli likelihood.

30.2.2.2 Poisson likelihood

In Section 30.3.1, we give a worked example of an HMM which models a timeseries of integer counts using a Poisson observation model.

30.2.2.3 Gaussian and GMM likelihood

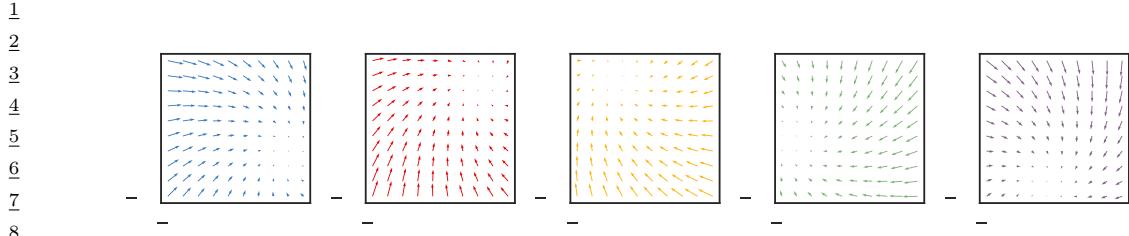
If \mathbf{y}_t is continuous, it is common to use a Gaussian observation model:

$$p(\mathbf{y}_t | z_t = j) = \mathcal{N}(\mathbf{y}_t | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (30.6)$$

As a simple example, suppose we have an HMM with 3 hidden states, each of which generates a 2d Gaussian. We can represent these Gaussian distributions as 2d ellipses, as shown in Figure 30.3(a). We call these “lilly pads”, because of their shape. We can imagine a frog hopping from one lilly pad to another. (This analogy is due to the late Sam Roweis.) It will stay on a pad for a while (corresponding to remaining in the same discrete state z_t), and then jump to a new pad (corresponding to a transition to a new state). See Figure 30.3(b). The data we see are just the 2d points (e.g., water droplets) coming from near the pad that the frog is currently on. Thus this model is like a Gaussian mixture model (Section 29.2.1), in that it generates clusters of observations, except now there is temporal correlation between the data points.

We can also use more flexible observation models. For example, if we use a K -component GMM, then we have

$$p(\mathbf{y}_t | z_t = j) = \sum_{k=1}^K \pi_{jk} \mathcal{N}(\mathbf{y}_t | \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}) \quad (30.7)$$



10 *Figure 30.4: Illustration of the observation dynamics for each of the 5 hidden states. The attractor*
11 *point corresponds to the steady state solution for the corresponding autoregressive process. Generated*
12 *by [arhmm_example.py](#).*

13

14

15 30.2.2.4 Autoregressive likelihoods

16 The standard HMM assumes the observations are conditionally independent given the hidden state.
17 In practice this is often not the case. However, it is straightforward to have direct arcs from y_{t-1} to
18 y_t as well as from z_t to y_t , as in Figure 30.1b. This is known as an **auto-regressive HMM**.

19 For continuous data, we can use an observation model of the form

$$\underline{21} \quad p(\mathbf{y}_t | \mathbf{y}_{t-1}, z_t = j, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_t | \mathbf{W}_j \mathbf{y}_{t-1} + \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (30.8)$$

22 This is a linear regression model, where the parameters are chosen according to the current hidden
23 state. (We could also use a nonlinear model, such as a neural network.) Such models are widely
24 used in econometrics, where they are called **regime switching Markov model** [Ham90]. Similar
25 models can be defined for discrete observations.

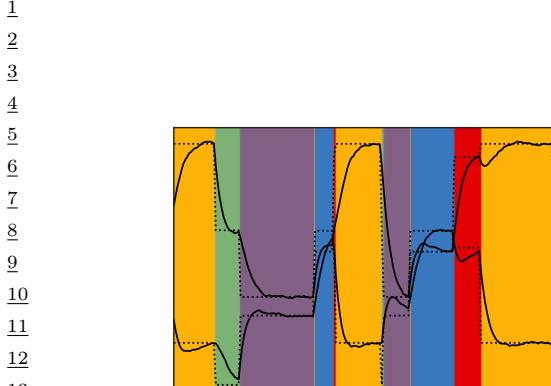
26 We can also consider higher-order extensions, where we condition on the last L observations:

$$\underline{27} \quad p(\mathbf{y}_t | \mathbf{y}_{t-L:t-1}, z_t = j, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_t | \sum_{\ell=1}^L \mathbf{W}_{j,\ell} \mathbf{y}_{t-\ell} + \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (30.9)$$

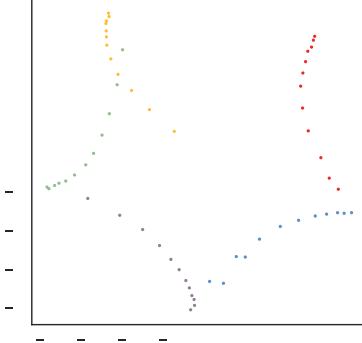
28 The AR-HMM essentially combines two Markov chains, one on the hidden variables, to capture long
29 range dependencies, and one on the observed variables, to capture short range dependencies [Ber99].
30 Since all the visible nodes are observed, adding connections between them just changes the likelihood,
31 but does not complicate the task of posterior inference (see Section 8.3.3).

32 Let us now consider a 2d example of this, due to Scott Linderman. We use a left-to-right transition
33 matrix with 5 states. In addition, the final state returns to first state, so we just cycle through the
34 states. Let $\mathbf{y}_t \in \mathbb{R}^2$, and suppose we set \mathbf{W}_j to a rotation matrix with a small angle of 7 degrees, and
35 we set each $\boldsymbol{\mu}_j$ to 72 degrees, so each state rotates 1/5 of the way around the circle. If the model stays
36 in the same state j for a long time, the observed dynamics will converge to the steady state $\mathbf{y}_{*,j}$, which
37 satisfies $\mathbf{y}_{*,j} = \mathbf{W}_j \mathbf{y}_{*,j} + \boldsymbol{\mu}_j$; we can solve for the steady state vector using $\mathbf{y}_{*,j} = (\mathbf{I} - \mathbf{W}_j)^{-1} \boldsymbol{\mu}_j$.
38 We can visualize the induced 2d flow for each of the 5 states as shown in Figure 30.4.

39 In Figure 30.5(a), we show a trajectory sampled from this model. We see that the two components
40 of the observation vector undergo different dynamics, depending on the underlying hidden state. In
41 Figure 30.5(b), we show the same data in a 2d scatter plot. The first observation is the yellow dot
42 (from state 2) at $(-0.8, 0.5)$. The dynamics converge to the stationary value of $\mathbf{y}_{*,2} = (-2.0, 3.8)$.
43



(a)



(b)

Figure 30.5: Samples from the 2d AR-HMM. (a) Time series plot of $y_{t,1}$ and $y_{t,2}$. (The latter are shifted up vertically by 4.7) The background color is the generating state. The dotted lines represent the stationary value for that component of the observation. (b) Scatter plot of observations. Colors denote the generating state. We show the first 12 samples from each state.

Then the system jumps to the green state (state 3), so it adds an offset of \mathbf{b}_3 to the last observation, and then converges to the staionary value of $\mathbf{y}_{*,3} = (-4.3, -0.8)$. And so on.

30.2.2.5 “Deep” likelihoods

We can easily use (conditional) deep generative models, such as VAEs (Section 22.2) or normalizing flows (Chapter 24), as the likelihood model. For example, [HNBK18] shows how to perform unsupervised learning of grammatical structure from sequences of word embeddings, as opposed to sequences of words, using a mixture of normalizing flows as the observation model.

30.3 HMMS: Applications

30.3.1 Segmentation of time series data

In this section, we give a variant of the casino example from Section 8.1.2, where our goal is to segment a time series into different regimes, each of which corresponds to a different statistical distribution. In Figure 30.6a we show the data, corresponding to counts generated from some process (e.g., visits to a web site, or number of infections). We see that the count rate seems to be roughly constant for a while, and then changes at certain points. We would like to segment this data stream into K different regimes or states, each of which is associated with a Poisson observation model with rate λ_k :

$$p(y_t|z_t = k) = \text{Poi}(y_t|\lambda_k) \quad (30.10)$$

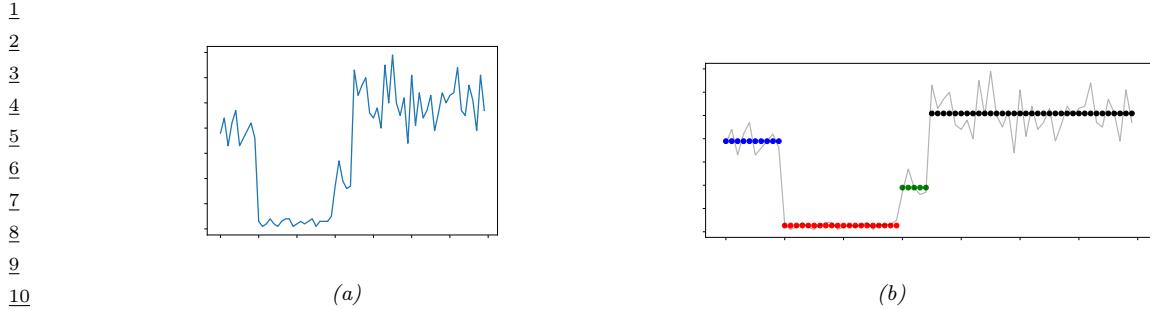


Figure 30.6: (a) A sample time series dataset of counts. (b) A segmentation of this data using a 4 state HMM. Generated by `hmm_poisson_changepoint_jax.ipynb`.

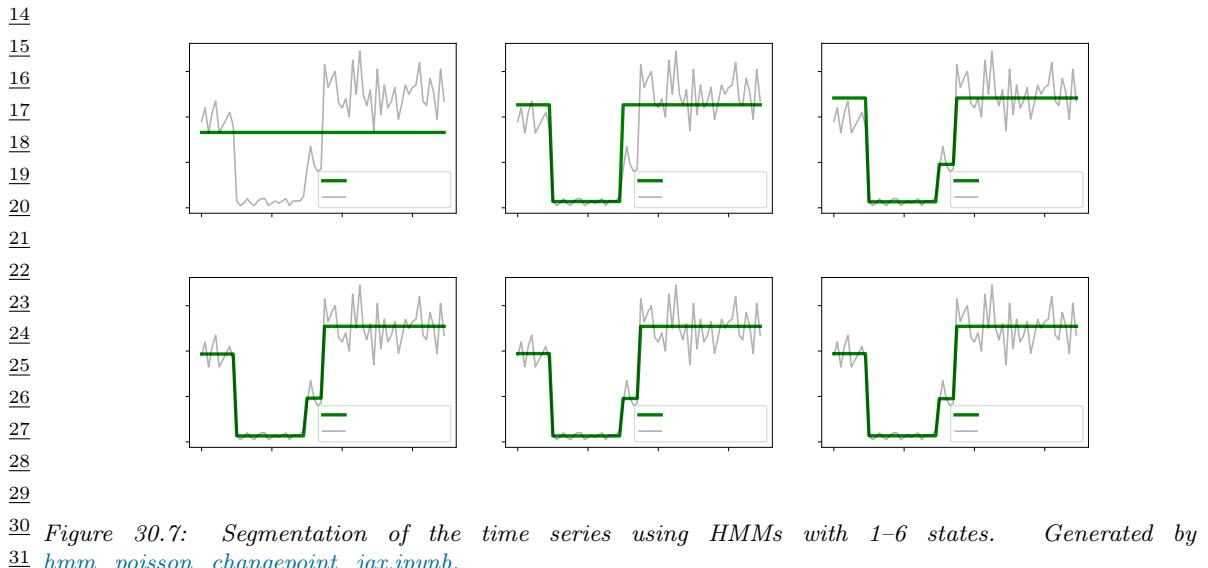


Figure 30.7: Segmentation of the time series using HMMs with 1–6 states. Generated by `hmm_poisson_changepoint_jax.ipynb`.

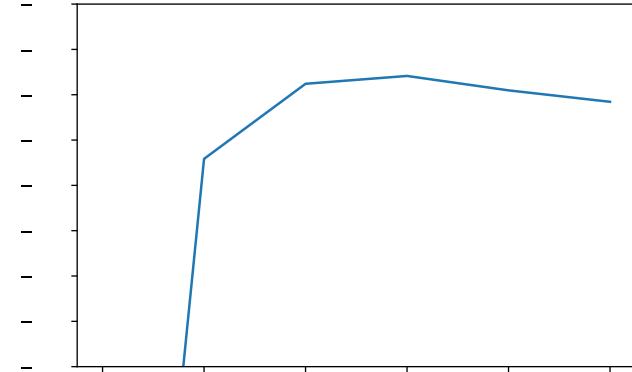
We use a uniform prior over the initial states. For the transition matrix, we the Markov chain stays in the same state with probability $p = 0.95$, and otherwise transitions to one of the other $K - 1$ states uniformly at random:

$$z_1 \sim \text{Categorical} \left(\left\{ \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} \right\} \right) \quad (30.11)$$

$$z_t | z_{t-1} \sim \text{Categorical} \left(\left\{ \begin{array}{ll} p & \text{if } z_t = z_{t-1} \\ \frac{1-p}{4-1} & \text{otherwise} \end{array} \right\} \right) \quad (30.12)$$

We compute a MAP estimate for the parameters $\lambda_{1:K}$ using a log-Normal(5,5) prior. We optimize the log of the Poisson rates using gradient descent, initializing the parameters at a random value centered on the log of the overall count means. We show the results in Figure 30.6b. See the method has successfully partitioned the data into 4 regimes, which is in fact how it was generated. (We

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17



18 Figure 30.8: Marginal likelihood vs number of states K in the Poisson HMM. Generated by
19 hmm_poisson_changepoint_jax.ipynb.

20

21 generating rates are $\lambda = (40, 3, 20, 50)$, with the changepoints happening at times $(10, 30, 35)$.)

22 In general we don't know the optimal number of states K . To solve this, we can fit many different
23 models, as shown in Figure 30.7, for $K = 1 : 6$. We see that after $K \geq 3$, the model fits are very
24 similar, since multiple states get associated to the same regime. We can pick the “best” K to be the
25 one with the highest marginal likelihood. Rather than summing over both discrete latent states and
26 integrating over the unknown parameters λ , we just maximize over the parameters (empirical Bayes
27 approximation):

28
$$p(\mathbf{y}_{1:T}|K) \approx \max_{\lambda} \sum_z p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}|\lambda, K) \quad (30.13)$$

31 We show this plot in Figure 30.8. We see the peak is at $K = 3$ or $K = 4$; after that it starts to go
32 down, due to the Bayesian Occam's razor effect.

33

34 30.3.2 Spelling correction

35 In this section, we illustrate how to use an HMM for **spelling correction**. The goal is to infer the
36 sequence of words $\mathbf{z}_{1:T}$ that the user meant to type, given observations of what they actually did
37 type, $\mathbf{y}_{1:T}$.

38

39 30.3.2.1 Baseline model

40 We start by using a simple unigram language model, so $p(\mathbf{z}_{1:T}) = \prod_{1:T} p(z_t)$, where $p(z_t = k)$ is the
41 prior probability of word k being used. These probabilities can be estimated by simply normalizing
42 word frequency counts from a large training corpus. We ignore any Markov structure.

43 Now we turn to the observation model, $p(y_t = v|z_t = k)$, which is the probability the user types
44 word v when they meant to type word k . For this, we use a **noisy channel model**, in which the
45

¹ “message” z_t gets corrupted by one of four kinds of error: substitution error, where we swap one
² letter for another (e.g., “government” mistyped as “govermmnt”); transposition errors, where we
³ swap the order of two adjacent letters (e.g., “government” mistyped as “governmnet”); deletion errors,
⁴ where we omit one letter (e.g., “government” mistyped as “goverment”); and insertion errors, where
⁵ we add an extra latter (e.g., “government” mistyped as “governmennt”). If y differs from z by d such
⁶ errors, we say that y and z have an **edit distance** of d . Let $\mathcal{D}(y, d)$ be the set of words that are edit
⁷ distance d away from y . We can then define the following likelihood function:
⁸

$$\begin{aligned} \underline{10} \quad p(y|z) &= \begin{cases} p_1 & y = z \\ \underline{11} \quad p_2 & y \in \mathcal{D}(z, 1) \\ \underline{12} \quad p_3 & y \in \mathcal{D}(z, 2) \\ \underline{13} \quad p_4 & \text{otherwise} \end{cases} \end{aligned} \tag{30.14}$$

¹⁵ where $p_1 > p_2 > p_3 > p_4$.

¹⁶ We can combine the likelihood with the prior to get the overall score for each hypothesis (i.e.,
¹⁷ candidate correction). This simple model, which was proposed by Peter Norvig¹, can work quite
¹⁸ well. However, it also has some flaws. For example, the error model assumes that the smaller the edit
¹⁹ distance, the more likely the word, but this is not always valid. For example, “reciet” gets corrected
²⁰ to “recite” instead of “receipt”, and “adres” gets corrected to “acres” not “address”. We can fix this
²¹ problem by learning the parameters of the noise model based on a labeled corpus of (z, x) pairs
²² derived from actual spelling errors. One possible way to get such a corpus is to look at web search
²³ behavior: if a user types query q_1 and then quickly changes it to q_2 followed by a click on a link, it
²⁴ suggests that q_2 is a manual correction for q_1 , so we can set $(z = q_2, y = q_1)$. This heuristic has been
²⁵ used in the Etsy search engine.² It is also possible to manually collect such data (see e.g., [Hag+17]),
²⁶ or to algorithmically create (z, y) pairs, where y is an automatically generated misspelling of z (see
²⁷ e.g., [ECM18]).
²⁸

²⁹ 30.3.2.2 HMM model

³¹ The baseline model can work well, but has room for improvement. In particular, many errors will
³² be hard to correct without context. For example, suppose the user typed “advice”: did they mean
³³ “advice” or “advise”? It depends on whether they intended to use a noun or a verb, which is hard
³⁴ to tell without looking at the sequence of words. To do this, we will “upgrade” our model to an
³⁵ HMM. We just have to replace our independence prior $p(z_{1:T}) = \prod_t p(z_t)$ by a standard first-order
³⁶ language model on words, $p(z_{1:T}) = \prod_t p(z_t|z_{t-1})$. The parameters of this model can be estimated
³⁷ by counting bigrams in a large corpus of “clean” text (see Section 2.8.3.1). The observation model
³⁸ $p(y_t|z_t)$ can remain unchanged.

³⁹ Given this model, we can compute the top N most likely hidden sequences in $O(NTK^2)$ time,
⁴⁰ where K is the number of hidden states, and T is the length of the sequence, as explained in
⁴¹ Section 8.3.6.5. In a naive implementation, the number of hidden states K is the number of words in
⁴² the vocabulary, which would make the method very slow. However, we can exploit sparsity of the
⁴³

⁴⁴ 1. See his excellent tutorial at <http://norvig.com/spell-correct.html>.
⁴⁵ 2. See this blogpost by Mohit Nayyar for details: <https://codeascraft.com/2017/05/01/modeling-spelling-correction-for-search-at-etsy/>.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

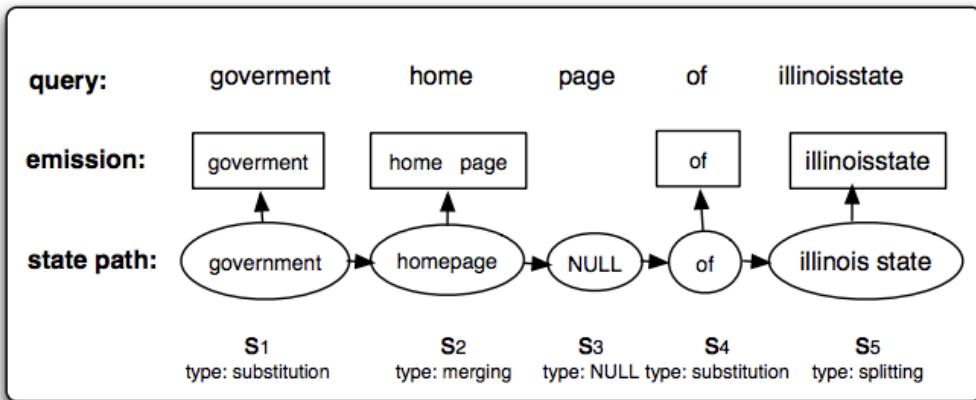


Figure 30.9: Illustration of an HMM applied to spelling correction. The top row, labeled “query”, represents the search query $y_{1:T}$ typed by the user, namely “goverment home page of illnoisstate”. The bottom row, labeled “state path”, represents the most probable assignment to the hidden states, $z_{1:T}$, namely “government homepage of illinois state”. (The NULL state is a silent state, that is needed to handle the generation of two tokens from a single hidden state.) The middle row, labeled “emission”, represents the words emitted by each state, which match the observed data. From Figure 1 of [LDZ11].

likelihood function (i.e., the fact that $p(y|z)$ is 0 for most values of z) to generate small candidate lists of hidden states for each location in the sequence. This gives us a sparse belief state vector α_t .

30.3.2.3 Extended HMM model

We can extend the HMM model to handle higher level errors, in addition to misspellings of individual words. In particular, [LDZ11; LDZ12] proposed modeling the following kinds of errors:

- Two words merged into one, e.g., “home page” → “homepage”.
- One word split into two, e.g., “illnoisstate” → “illinois state”.
- Within-word errors, such as substitution, transposition, insertion and deletion of letters, as we discussed in Section 30.3.2.2.

We can model this with an HMM, where we augment the state space with a **silent state**, that does not emit any symbols. Figure 30.9 illustrates how this model can “denoise” the observed query “goverment home page of illnoisstate” into the correctly formulated query “government homepage of illnois state”.

An alternative to using HMMs is to use supervised learning to fit a sequence-to-sequence translation model, using RNNs or transformers. This can work very well, but often needs much more training data, which can be problematic for **low-resource languages** [ECM18].

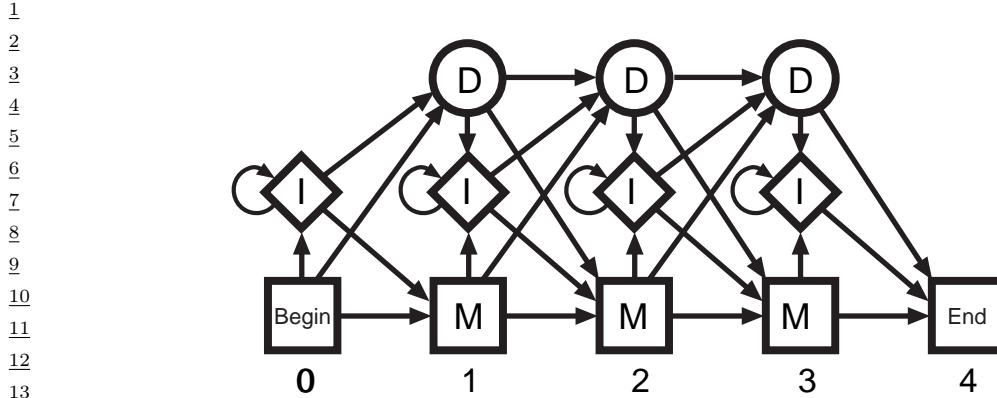


Figure 30.10: State transition diagram for a profile HMM. From Figure 5.7 of [Dur+98]. Used with kind permission of Richard Durbin.

Q5E940_BOVIN	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_HUMAN	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_MOUSE	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_RAT	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_CHICK	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_RAMS	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--S-A-L-E-	76
QTZUG3_BRAVE	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_BIRD	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_DRONE	-M-P-E-D-E-T-W-S-H-Y-E-L-K-E-I-L-L-D-D-P-C-F-T-I-V-A-D-N-Y-G-C-O-M-O-I-M-S-L-R-Q-K-A-V-V-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-A-L-E-	76
RLAO_DICDI	-M-S-E-A-G-S-E-K-B-K-L-F-E-K-T-K-L-F-T-T-D-K-M-V-A-C-A-F-Y-G-S-H-U-K-K-I-E-S-K-E-G-I-C-A-V-Y-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-E-L-D-	75
Q5A1D6_DICDI	-M-S-E-A-G-S-E-K-B-K-L-F-E-K-T-K-L-F-T-T-D-K-M-V-A-C-A-F-Y-G-S-H-U-K-K-I-E-S-K-E-G-I-C-A-V-Y-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-E-L-D-	75
RLAO_MU	-M-S-E-A-G-S-E-K-B-K-L-F-E-K-T-K-L-F-T-T-D-K-M-V-A-C-A-F-Y-G-S-H-U-K-K-I-E-S-K-E-G-I-C-A-V-Y-L-G-K-E-T-M-M-R-A-I-G-H-E-L-L-N-N--P-E-L-D-	75
RLAO_SULAC	-M-I-L-A-V-T-T-T-K-K-L-A-K-E-B-D-E-V-A-E-L-T-B-L-E-K-T-Q-K-T-T-L-A-M-I-E-G-F-A-D-K-L-H-E-T-K-K-K-L-R-Q-K-A-D-E-Y-V-E-T-M-I-F-Y-L-E-K-N-E--P-E-L-D-	79
RLAO_SULTO	-M-I-H-A-V-T-Y-Q-E-R-K-T-A-C-K-H-E-V-E-K-E-L-B-K-L-E-H-T-T-L-A-M-I-E-G-F-A-D-K-L-H-E-T-K-K-K-L-R-Q-K-A-D-E-Y-V-E-T-M-I-F-Y-L-E-K-N-E--P-E-L-D-	80
RLAO_ALB	-M-I-H-A-V-T-Y-Q-E-R-K-T-A-C-K-H-E-V-E-K-E-L-B-K-L-E-H-T-T-L-A-M-I-E-G-F-A-D-K-L-H-E-T-K-K-K-L-R-Q-K-A-D-E-Y-V-E-T-M-I-F-Y-L-E-K-N-E--P-E-L-D-	80
RLAO_AERPE	-M-S-V-E-T-E-B-O-M-Y-R-E-K-E-T-P-E-S-E-T-P-E-T-L-M-R-E-L-E-B-L-F-S-K-W-S-V-E-L-O-U-G-S-E-B-P-V-W-D-E-V-E-K-K-K-K-W-K-P-D-M-W-A-K-B-L-L-I-B-A-M-S-A-B-L-L-E--L-O-D-N-	86
RLAO_PYRRO	-M-H-L-A-G-K-R-R-Y-V-R-T-Q-D-P-A-R-V-K-V-I-S-E-T-T-L-L-O-L-O-V-Y-V-E-L-F-O-D-U-S-G-E-R-I-L-H-E-Y-V-E-Y-L-H-Y-Y-S-V-T-K-I-R-O-L-E-F-K-T-A-F-T-K-Y-V-E-G--I-P-A-S-	85
RLAO_METAC	-M-A-E-E-R-H-I-T-E-H-P-O-W-K-D-E-I-E-N-Y-E-L-L-O-Q-K-W-V-O-V-E-T-E-G-L-A-T-K-H-O-I-E-S-E-D-K-Q-V-A-V-V-E-R-E-T-E-L-E-A-N-Q-O-L--E-E-T-H-	78
RLAO_METC	-M-A-E-E-R-H-I-T-E-H-P-O-W-K-D-E-I-E-N-Y-E-L-L-O-Q-K-W-V-O-V-E-T-E-G-L-A-T-K-H-O-I-E-S-E-D-K-Q-V-A-V-V-E-R-E-T-E-L-E-A-N-Q-O-L--E-E-T-H-	78
RLAO_ARCFU	-M-A-V-Y-E-S-S-E-P-E-Y-E-B-V-A-E-E-V-E-B-E-K-H-M-S-E-P-V-A-L-Y-S-E-P-E-R-F-E-N-C-O-M-O-I-S-E-R-F-Q-K-A-E-L-E-V-V-E-T-E-L-L-E-A-R-D-A-L-E--G-D-V-E-L-	75
RLAO_METKA	-M-A-V-Y-E-S-S-E-P-E-Y-E-B-V-A-E-E-V-E-B-E-K-H-M-S-E-P-V-A-L-Y-S-E-P-E-R-F-E-N-C-O-M-O-I-S-E-R-F-Q-K-A-E-L-E-V-V-E-T-E-L-L-E-A-R-D-A-L-E--G-D-V-E-L-	88
RLAO_METTA	-M-A-V-Y-E-S-S-E-P-E-Y-E-B-V-A-E-E-V-E-B-E-K-H-M-S-E-P-V-A-L-Y-S-E-P-E-R-F-E-N-C-O-M-O-I-S-E-R-F-Q-K-A-E-L-E-V-V-E-T-E-L-L-E-A-R-D-A-L-E--G-D-V-E-L-	74
RLAO_METEL	-M-I-T-A-S-E-H-K-T-A-P-E-L-E-V-V-A-E-L-E-K-L-L-K-S-A-N-T-A-L-L-O-M-M-E-Y-P-A-V-O-L-E-T-E-K-O-K-I-R-D-O-M-E-L-K-M-S-B-E-L-L-I-K-R-A-V-E-F-V-A-B-E-T-G-M-P-F-A-	82
RLAO_METVA	-M-I-T-A-S-E-H-K-T-A-P-E-L-E-V-V-A-E-L-E-K-L-L-K-S-A-N-T-A-L-L-O-M-M-E-Y-P-A-V-O-L-E-T-E-K-O-K-I-R-D-O-M-E-L-K-M-S-B-E-L-L-I-K-R-A-V-E-F-V-A-B-E-T-G-M-P-F-A-	82
RLAO_METJA	-M-E-T-Y-K-V-A-H-Y-A-V-P-E-L-E-V-V-A-E-L-E-K-L-L-K-S-B-V-A-L-Y-V-O-M-M-D-F-A-E-L-E-K-O-K-I-R-D-O-K-Y-V-E-L-M-S-B-E-L-L-I-K-R-A-V-E-F-V-A-B-E-T-G-M-P-F-A-	81
RLAO_METL	-M-E-T-Y-K-V-A-H-Y-A-V-P-E-L-E-V-V-A-E-L-E-K-L-L-K-S-B-V-A-L-Y-V-O-M-M-D-F-A-E-L-E-K-O-K-I-R-D-O-K-Y-V-E-L-M-S-B-E-L-L-I-K-R-A-V-E-F-V-A-B-E-T-G-M-P-F-A-	77
RLAO_PYRRO	-M-A-H-V-A-E-W-K-K-K-E-V-E-E-L-A-L-K-L-K-B-V-T-A-L-V-V-O-S-H-M-A-Y-V-T-S-O-H-E-R-L-E-B-E-N-G-E-L-L-R-V-B-E-T-T-L-L-A-L-K-K-A-K-E-L-O-G-K-E-L-E-	77
RLAO_PYRFU	-M-A-H-V-A-E-W-K-K-K-E-V-E-E-L-A-L-K-L-K-B-V-T-A-L-V-V-O-S-H-M-A-Y-V-T-S-O-H-E-R-L-E-B-E-N-G-E-L-L-R-V-B-E-T-T-L-L-A-L-K-K-A-K-E-L-O-G-K-E-L-E-	77
RLAO_PZERO	-M-E-D-E-R-K-T-E-T-I-P-E-V-E-B-E-K-L-M-S-E-P-V-A-L-Y-S-E-P-E-R-F-E-N-C-O-M-O-I-S-E-R-F-Q-K-A-E-L-E-V-V-E-T-E-L-L-E-A-R-D-A-L-E--G-D-V-E-L-	76
RLAO_MALV	-M-E-D-E-R-K-T-E-T-I-P-E-V-E-B-E-K-L-M-S-E-P-V-A-L-Y-S-E-P-E-R-F-E-N-C-O-M-O-I-S-E-R-F-Q-K-A-E-L-E-V-V-E-T-E-L-L-E-A-R-D-A-L-E--G-D-V-E-L-	79
RLAO_HALVO	-M-S-E-S-V-Q-T-E-V-P-O-W-P-E-R-E-V-E-D-F-D-E-S-E-S-V-C-V-V-E-V-A-G-P-E-D-O-S-H-E-R-E-L-H-S-Q-A-A-V-R-M-B-E-T-L-V-N-A-L-D-O-F-T--D-E-L-D-	79
RLAO_HALSA	-M-S-E-S-V-Q-T-E-V-P-O-W-P-E-R-E-V-E-D-F-D-E-S-E-S-V-C-V-V-E-V-A-G-P-E-D-O-S-H-E-R-E-L-H-S-Q-A-A-L-S-M-B-E-T-L-V-N-A-L-D-O-F-T--D-E-L-D-	79
RLAO_HEDM	-M-S-E-S-V-Q-T-E-V-P-O-W-P-E-R-E-V-E-D-F-D-E-S-E-S-V-C-V-V-E-V-A-G-P-E-D-O-S-H-E-R-E-L-H-S-Q-A-A-L-S-M-B-E-T-L-V-N-A-L-D-O-F-T--D-E-L-D-	72
RLAO_THEVO	-M-R-K-I-N-D-K-E-I-V-S-E-L-A-D-E-T-S-K-S-A-V-V-T-Y-T-G-V-E-U-S-E-R-O-G-T-I-S-E-K-H-D-K-O-K-T-K-E-V-E-S-E-L-L-E-A-R-D-S-E-N-D---E-K-E-T	72
RLAO_PICTO	-M-T-E-P-A-O-W-I-D-Y-V-N-E-L-E-M-I-S-S-K-R-V-A-L-Y-S-E-K-G-E-R-N-H-E-T-C-K-T-S-I-S-T-H-O-K-A-R-K-Y-V-E-A-R-L-L-A-L-E-H-E-K---N-N-E-V	72

Figure 30.11: Example of multiple sequence alignment. We show the first 90 positions of the acidic ribosomal protein P0 from several organisms. Colors represent functional properties of the corresponding amino acid. Dashes represent insertions or deletions. Symbols at the top represent degree of agreement across sequences at each location (a measure of how well conserved that location is). From https://en.wikipedia.org/wiki/Multiple_sequence_alignment. Used with kind permission of Wikipedia author Miguel Andrade.

30.3.3 Protein sequence alignment

An important application of HMMs is to the problem of **protein sequence alignment** [Dur+98]. Here the goal is to determine if a test sequence $y_{1:T}$ belongs to a protein family or not, and if so, how it aligns with the canonical representation of that family. (Similar methods can be used to align DNA and RNA sequences.) The technique we will use is similar to the spelling correction model in Section 30.3.2.2, but can be trained in an unsupervised way from raw sequences, without having to manually create labeled (z, y) pairs.

To solve the alignment problem, let us initially assume we have a set of aligned sequences from a protein family, from which we can generate a **consensus sequence**. This defines a probability

1 distribution over symbols at each location t in the string; denote each **position-specific scoring**
2 **matrix** by $\theta_t(v) = p(y_t = v)$. These parameters can be estimated by counting.
3

4 Now we turn the PSSM into an HMM with 3 hidden states, representing the events that the
5 location t matches the consensus sequence, $z_t = M$, or inserts its own unique symbol, $z_t = I$, or
6 deletes (skips) the corresponding consensus symbol, $z_t = D$. We define the observation models for
7 these 3 events as follows. For matches, we use the PSSM $p(y_t = v | z_t = M) = \theta_t(v)$. For insertions
8 we use the uniform distribution $p(y_t = v | z_t = I) = 1/V$, where V is the size of the vocabulary. For
9 deletions, we use $p(y_t = - | z_t = D)$, where “-” is a special deletion symbol used to pad the generated
10 sequence to the correct length. The corresponding state transition matrix is shown in Figure 30.10:
11 we see that matches and deletions advance one location along the consensus sequence, but insertions
12 stay in the same location (represented by the self-transition from I to I). This model is known as a
13 **profile HMM**.

14 Given a profile HMM with consensus parameters θ , we can compute $p(\mathbf{y}_{1:T} | \theta)$ in $O(T)$ time using
15 the forwards algorithm, as described in Section 8.3.1. This can be used to decide if the sequence
16 belongs to this family or not, by thresholding the log-odds score, $L(\mathbf{y}) = \log p(\mathbf{y} | \theta) / p(\mathbf{y} | \mathcal{M}_0)$, where
17 \mathcal{M}_0 is a baseline model, such as the uniform distribution. If the string matches, we can compute an
18 alignment to the consensus using the Viterbi algorithm, as described in Section 8.3.6. See Figure 30.11
19 for an illustration of such a **multiple sequence alignment**. If we don’t have an initial set of aligned
20 sequences from which to compute the consensus sequence θ , we can use the Baum-Welch algorithm
21 (Section 30.4.1) to compute the MLE for the parameters θ from a set of unaligned sequences. For
22 details, see e.g., [Dur+98, Ch.6].
23

24 30.4 HMMS: parameter learning

26 In this section, we discuss how to compute a point estimate or the full posterior over the model
27 parameters of an HMM given a set of partially observed sequences.

29 30.4.1 The Baum-Welch (EM) algorithm

31 In this section, we discuss how to compute an approximate MLE for the parameters of an HMM
32 using the EM algorithm (Section 6.7.3). (We can easily extend this to compute a MAP estimate, by
33 adding a log prior term.) The resulting method is known as the **Baum-Welch** algorithm [Bau+70].
34

35 30.4.1.1 Log likelihood

36 The joint probability of a single sequence is given by

$$\begin{aligned} \text{38} \quad p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \theta) &= \left[p(z_1 | \pi) \prod_{t=2}^T p(z_t | z_{t-1}, \mathbf{A}) \right] \left[\prod_{t=1}^T p(\mathbf{y}_t | z_t, \mathbf{B}) \right] \end{aligned} \quad (30.15)$$

41 where $\theta = (\pi, \mathbf{A}, \mathbf{B})$. Of course, we cannot compute this objective, since $\mathbf{z}_{1:T}$ is hidden. So instead
42 we will optimize the expected complete data log likelihood,
43

$$\begin{aligned} \text{44} \quad Q(\theta, \theta^{\text{old}}) &= \mathbb{E}_{p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}, \theta^{\text{old}})} [\log p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \theta)] \end{aligned} \quad (30.16)$$

46 This can be easily summed over N sequences. See Figure 30.12 for the graphical model.
47

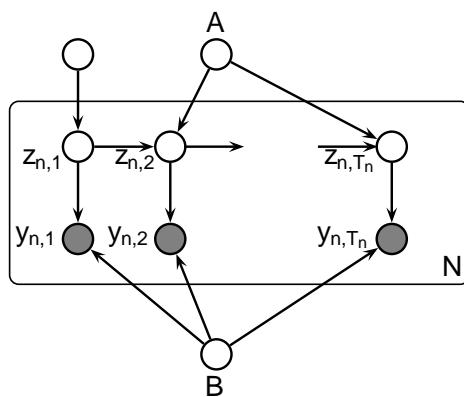


Figure 30.12: HMM with plate notation. A are the parameters for the state transition matrix $p(z_t|z_{t-1})$ and B are the parameters for the discrete observation model $p(x_t|z_t)$. T_n is the length of the n'th sequence.

The above objective is a lower bound on the observed data log likelihood, $\log p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$, so the entire procedure is a bound optimization method that is guaranteed to converge to a local optimum. (In fact, in the case of HMMs, it can be shown to converge to (close to) one of the global optima [YBW15].)

30.4.1.2 E step

Let $A_{jk} = p(z_t = k|z_{t-1} = j)$ be the $K \times K$ transition matrix. For the first time slice, let $\pi_k = p(z_1 = k)$ be the initial state distribution. Let $\boldsymbol{\theta}_k$ represent the parameters of the observation model for state k .

One can show that the expected complete data log likelihood is given by

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{k=1}^K \mathbb{E}[N_k^1] \log \pi_k + \sum_{j=1}^K \sum_{k=1}^K \mathbb{E}[N_{jk}] \log A_{jk} \quad (30.17)$$

$$+ \sum_{n=1}^N \sum_{t=1}^{T_n} \sum_{k=1}^K p(z_t = k | \mathbf{y}_n, \boldsymbol{\theta}^{\text{old}}) \log p(\mathbf{y}_{n,t} | \boldsymbol{\theta}_k) \quad (30.18)$$

where T_n is the length of sequence n . To compute the expected counts, we first run the forwards-backwards algorithm on each sequence (see Section 8.3.3). This returns the following node and edge marginals:

$$\gamma_{n,t}(j) \triangleq p(z_t = j | \mathbf{y}_{n,1:T_n}, \boldsymbol{\theta}^{\text{old}}) \quad (30.19)$$

$$\xi_{n,t}(j, k) \triangleq p(z_{t-1} = j, z_t = k | \mathbf{y}_{n,1:T_n}, \boldsymbol{\theta}^{\text{old}}) \quad (30.20)$$

We can then derive the expected counts as follows (note that we pool the sufficient statistics across time, since the parameters are tied, as well as across sequences):

$$\mathbb{E}[N_k^1] = \sum_{n=1}^N \gamma_{n,1}(k), \quad \mathbb{E}[N_k] = \sum_{n=1}^M \sum_{t=2}^{T_n} \gamma_{n,t}(k), \quad \mathbb{E}[N_{jk}] = \sum_{n=1}^N \sum_{t=2}^{T_n} \xi_{n,t}(j, k) \quad (30.21)$$

30.4.1.3 M step

We can estimate the transition matrix and initial state probabilities by maximizing the objective subject to the sum to one constraint. The result is just a normalized version of the expected counts:

$$\hat{A}_{jk} = \frac{\mathbb{E}[N_{jk}]}{\sum_{k'} \mathbb{E}[N_{jk'}]}, \quad \hat{\pi}_k = \frac{\mathbb{E}[N_k^1]}{N} \quad (30.22)$$

This result is quite intuitive: we simply add up the expected number of transitions from j to k , and divide by the expected number of times we transition from j to anything else.

For a categorical observation model, the expected sufficient statistics are

$$\mathbb{E}[M_{kv}] = \sum_{n=1}^N \sum_{t=1}^{T_n} \gamma_{n,t}(k) \mathbb{I}(y_{n,t} = v) = \sum_{n=1}^N \sum_{t:y_{n,t}=v} \gamma_{n,t}(k) \quad (30.23)$$

The M step has the form

$$\hat{B}_{kv} = \frac{\mathbb{E}[M_{kv}]}{\mathbb{E}[N_k]} \quad (30.24)$$

This result is quite intuitive: we simply add up the expected number of times we are in state k and we see a symbol v , and divide by the expected number of times we are in state k . See Algorithm 11 for the pseudocode.

For a Gaussian observation model, the expected sufficient statistics are given by

$$\mathbb{E}[\bar{\mathbf{y}}_k] = \sum_{n=1}^N \sum_{t=1}^{T_n} \gamma_{n,t}(k) \mathbf{y}_{n,t}, \quad \mathbb{E}[(\bar{\mathbf{y}}\bar{\mathbf{y}})^T_k] = \sum_{n=1}^N \sum_{t=1}^{T_n} \gamma_{n,t}(k) \mathbf{y}_{n,t} \mathbf{y}_{n,t}^T \quad (30.25)$$

The M step becomes

$$\hat{\boldsymbol{\mu}}_k = \frac{\mathbb{E}[\bar{\mathbf{y}}_k]}{\mathbb{E}[N_k]} \quad (30.26)$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{\mathbb{E}[(\bar{\mathbf{y}}\bar{\mathbf{y}})^T_k] - \mathbb{E}[N_k] \hat{\boldsymbol{\mu}}_k \hat{\boldsymbol{\mu}}_k^T}{\mathbb{E}[N_k]} \quad (30.27)$$

This can (and should) be regularized in the same way we regularize GMMs.

30.4.1.4 Initialization

As usual with EM, we must take care to ensure that we initialize the parameters carefully, to minimize the chance of getting stuck in poor local optima. There are several ways to do this, such as

- Use some fully labeled data to initialize the parameters.
- Initially ignore the Markov dependencies, and estimate the observation parameters using the standard mixture model estimation methods, such as K-means or EM.
- Randomly initialize the parameters, use multiple restarts, and pick the best solution.

1

2 **Algorithm 32:** Baum Welch algorithm for (discrete observation) HMMs

3 1 Initialize parameters θ ;
 4 2 **for** each iteration **do**
 5 3 // E step ;
 6 4 Initialize ESS: $\mathbb{E}[N_k] = 0$, $\mathbb{E}[N_{jk}] = 0$, $\mathbb{E}[M_{kv}] = 0$;
 7 5 **for** each datacase n **do**
 8 6 Use forwards-backwards algorithm on y_n to compute $\gamma_{n,t}$ and $\xi_{n,t}$
 9 7 (Equations 30.19–30.20) ;
 10 8 $\mathbb{E}[N_k] := \mathbb{E}[N_k] + \sum_{t=2}^{T_n} \gamma_{n,t}(k)$;
 11 9 $\mathbb{E}[N_{jk}] := \mathbb{E}[N_{jk}] + \sum_{t=2}^{T_n} \xi_{n,t}(j, k)$;
 12 9 $\mathbb{E}[M_{kv}] := \mathbb{E}[M_{kv}] + \sum_{t:x_{n,t}=v} \gamma_{n,t}(k)$
 13 10 // M step ;
 14 11 Compute new parameters $\theta = (\mathbf{A}, \mathbf{B}, \pi)$ using Equations 30.22

15

16

17

18

19

20

21

22

23

24

25

26

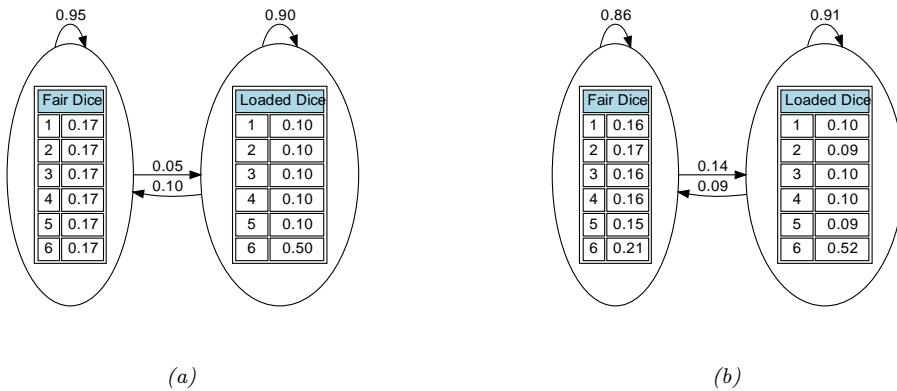
27

28

29

30

31



32 *Figure 30.13: Illustration of the casino HMM. (a) True parameters used to generate the data. (b) Estimated
 33 parameters, using EM for 20 iterations. Generated by `hmm_casino_em_train.py`.*

34

35

36 Techniques such as deterministic annealing [UN98; RR01a] can help mitigate the effect of local
 37 minima. Also, just as K-means is often used to initialize EM for GMMs, so it is common to initialize
 38 EM for HMMs using Viterbi training. The Viterbi algorithm is explained in Section 8.3.6, but
 39 basically it is an algorithm to compute the single most probable path. As an approximation to
 40 the E step, we can replace the sum over paths with the statistics computed using this single path.
 41 Sometimes this can give better results [AG11].

42

43 30.4.1.5 Example: casino HMM

44

45 In this section, we fit the casino HMM from Section 8.1.2. The true generative model is shown in
 46 Figure 30.13a. We used this to generate 5 sequences of varying length (max 3000), totalling 5670
 47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

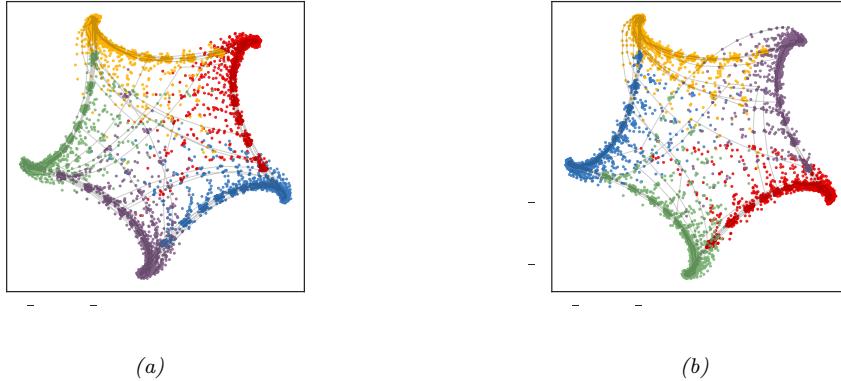


Figure 30.14: (a) Samples from the true AR-HMM. (b) Samples from the learned AR-HMM.

observations. We initialized the model with random parameters, and then ran EM for 20 iterations. We got the results in Figure 30.13b.

30.4.1.6 Example: AR-HMM

In this section, we revisit the 2d AR-HMM example from Section 30.2.2.4. We generate a single sequence of length 10,000, and fit the model using EM. In Figure 30.14(a) we show samples from the true model, and in Figure 30.14(b) we show samples from the learned model. We can see that the results are very similar, modulo the change in color due to label switching.

To avoid the label switching problem, we can find the optimal permutation between the learned states and the true states by solving a **linear assignment problem**, also called a **minimum weight matching** in a bipartite graph. This finds the binary matrix \mathbf{X} which minimizes

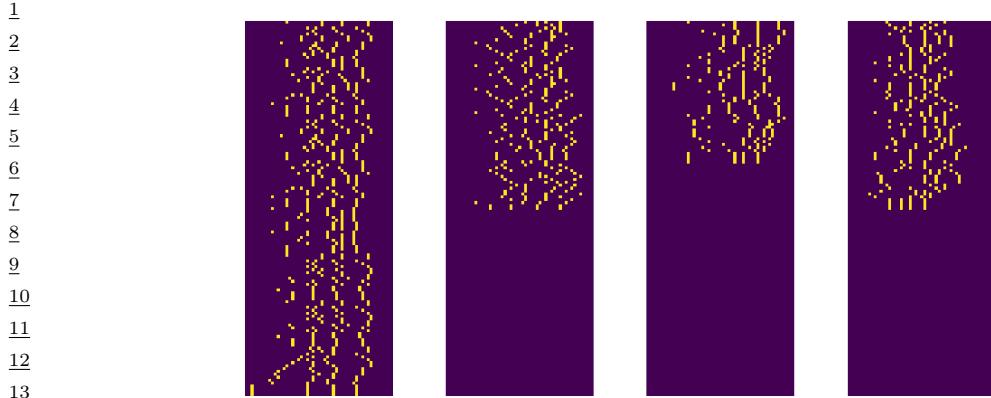
$$\mathcal{L}(\mathbf{X}) = \sum_i \sum_j C_{ij} X_{ij} \quad (30.28)$$

where $X_{ij} = 1$ if row i is assigned to column j , and C_{ij} is the cost. This can be computed using the **Hungarian algorithm**. In the context of label switching, we first define N_{ij} as the number of time steps where the true label is state i and the predicted state is j , and then set $C_{ij} = -N_{ij}$ (since we want to maximize agreement).

30.4.2 Parameter estimation using SGD

Although the EM algorithm is the “traditional” way to fit HMMs, it is inherently a batch algorithm, so it does not scale well to large datasets (with many sequences). Although it is possible to extend bound optimization to the online case (see e.g., [Mai15]), this can take a lot of memory.

A simple alternative is to optimize $\log p(\mathbf{y}_{1:T} | \boldsymbol{\theta})$ using SGD. We can compute this objective using



14 *Figure 30.15: First 4 training sequences from the Bach Chorales dataset. Vertical axis represents time (start*
15 *of music at top), horizontal axis represents the 51 notes. Generated by `hmm_bach_chorales.ipynb`.*

16

17

18 the forwards algorithm, as shown in Equation (8.37):

19

$$\log p(\mathbf{y}_{1:T} | \boldsymbol{\theta}) = \sum_{t=1}^T \log p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, \boldsymbol{\theta}) = \sum_{t=1}^T \log Z_t \quad (30.29)$$

20

21 where the normalization constant for each time step is given by

22

$$Z_t \triangleq p(\mathbf{y}_t | \mathbf{y}_{1:t-1}) = \sum_{j=1}^K p(z_t = j | \mathbf{y}_{1:t-1}) p(\mathbf{y}_t | z_t = j) \quad (30.30)$$

23

24 Of course, we need to ensure the transition matrix remains a valid row stochastic matrix, i.e., that
25 $0 \leq A_{ij} \leq 1$ and $\sum_j A_{ij} = 1$. Similarly, if we have categorical observations, we need to ensure B_{jk}
26 is a valid row stochastic matrix, and if we have Gaussian observations, we need to ensure Σ_k is a
27 valid psd matrix. These constraints are automatically taken care of in EM. When using SGD, we can
28 reparameterize to an unconstrained form, as proposed in [BC94].

29

30 30.4.2.1 Example: Bach Chorales

31 In this section, we give an example of fitting an HMM with SGD to some music data from [BLBV12].
32 The training set consists of 229 sequences of Chorales composed by J. S. Bach. Each sequence has a
33 maximum length of 129, and contains 51 notes.³ Thus the data matrix is a boolean tensor of size
34 $N \times T_{\max} \times D = 229 \times 129 \times 51$, although the actual sequence lengths are variable, so the data is
35 stored in a ragged array. See Figure 30.15 for a visualization of some of the data.

36 We model the observation at each time step using a factored observation distribution of the form

37

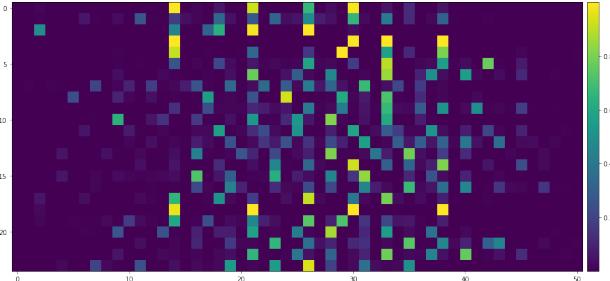
$$p(\mathbf{y}_t | z_t = k) = \prod_{d=1}^D \text{Ber}(y_{td} | B_d(k)) \quad (30.31)$$

38

39 3. We follow the same preprocessing as used in <https://pyro.ai/examples/hmm.html>, where they drop 37 notes that
40 are never played.

41

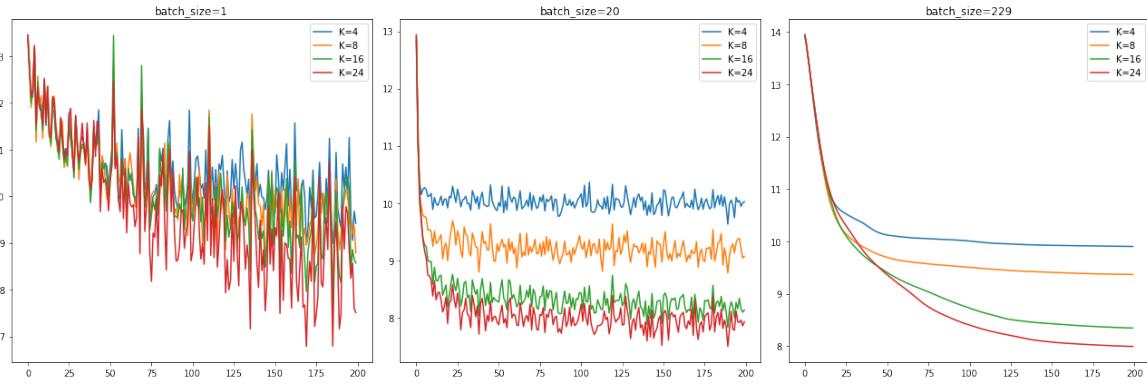
1
2
3
4
5
6
7
8
9
10



11
12
13

Figure 30.16: Learned observation distributions for the 51 notes from a 24 state HMM. Generated by `hmm_bach_chorales.ipynb`.

14
15
16
17
18
19
20
21
22
23
24
25
26



27
28
29
30
31

Figure 30.17: Negative log likelihood vs iterations on the Bach Chorales dataset for different number of hidden states K , and batch sizes of $B = 1$ (left), $B = 20$ (middle) and $B = N = 229$ (right). Generated by `hmm_bach_chorales.ipynb`.

32
33
34
35
36
37
38
39
40
41
42

where $B_d(k) = p(y_{td} = 1 | z_t = k)$. We then fit an HMM with $K \in \{4, 8, 16, 24\}$ states to this using SGD. In Figure 30.16 we show the learned observation model for $K = 24$ (which was the model with the best test set loglikelihood). We see that some states generate multiple notes at once, presumably corresponding to chords.

In Figure 30.17, we show how the negative log likelihood on the training set is reduced across SGD iterations. Obviously the training loss is lower for larger K . In addition, the loss curve is noisier for small batch sizes. The test set negative log likelihood using $K = 24$ is 8.05 nats per time step, which is comparable to the value of 8.28 reported in Table 1 of [Obe+19] using an HMM with $K = 16$.

30.4.3 Parameter estimation using spectral methods

43
44
45
46
47

Fitting HMMs using maximum likelihood is difficult, because the log likelihood is not convex. Thus there are many local optima, and EM and SGD can give poor results. An alternative approach is to marginalize out the hidden variables, and work instead with predictive distributions in the visible space. For discrete observation HMMs, with observation matrix $B_{jk} = p(y_t = k | z_t = j)$, such a

1 distribution has the form
2

$$\underline{3} \quad [\mathbf{y}_t]_k \triangleq p(y_t = k | \mathbf{y}_{1:t-1}) \quad (30.32)$$

4 This is called a **predictive state representation** [SJR04].
5

6 Suppose there are m possible hidden states, and n possible visible symbols, where $n \geq m$. One can
7 show [HKZ12; Joh12] that the PSR vectors lie in a subspace in \mathbb{R}^n with a dimensionality of $m \leq n$.
8 Intuitively this is because the linear operator \mathbf{A} defining the hidden state update in Equation (8.36),
9 combined with the mapping to observables via \mathbf{B} , induces low rank structure in the output space.
10 Furthermore, we can estimate a basis for this low rank subspace using SVD applied to the observable
11 matrix of co-occurrence counts:
12

$$\underline{13} \quad [\mathbf{P}_2]_{ij} = p(y_t = i, y_{t-1} = j) \quad (30.33)$$

14 We also need to estimate the third order statistics
15

$$\underline{16} \quad [\mathbf{P}_3]_{ijk} = p(y_t = i, y_{t-1} = j, y_{t-2} = k) \quad (30.34)$$

17 Using these quantities, it possible to perform recursive updating of our predictions while working
18 entirely in visible space. This is called **spectral estimation**, or **tensor decomposition** [HKZ12;
19 AHK12; Rod14; Ana+14; RSG17].
20

21 We can use spectral methods to get a good initial estimate of the parameters for the latent variable
22 model, which can then be refined using EM (see e.g., [Smi+00]). Alternatively, we can use them “as is”,
23 without needing EM at all. See [Mat14] for a comparison of these methods. See also Section 31.2.5.3
24 where we discuss spectral methods for fitting linear dynamical systems.
25

26 30.4.4 Bayesian parameter inference

27 MLE methods can easily overfit, and can suffer from numerical problems, especially when sample
28 sizes are small. In [Fot+14], they show how to apply stochastic variational inference (Section 10.3.2)
29 to HMMs, by leveraging the conjugate structure. An alternative approach is to marginalize out the
30 discrete latents (using the forwards algorithm), and then to use SVI to target the log posterior
31

$$\underline{32} \quad \log p(\boldsymbol{\theta}, \mathcal{D}) = \log p(\boldsymbol{\theta}) + \sum_{n=1}^N \log p(\mathbf{y}_{1:T,n} | \boldsymbol{\theta}) \quad (30.35)$$

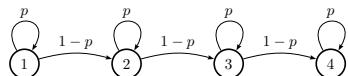
33 This can be thought of as “collapsed SVI”, and was proposed in [Obe+19].
34

35 We can also apply HMC (Section 12.5) to the same log joint. This is a form of “collapsed MCMC”.
36 This is generally faster than the older technique of block Gibbs sampling [Sco02], that alternates
37 between sampling latent sequences $\mathbf{z}_{1:T,1:N}^s$ using the forwards filtering backwards sampling algorithm
38 (Section 8.3.7) and sampling the parameters from their full conditionals, $p(\boldsymbol{\theta} | \mathbf{y}_{1:T}, \mathbf{z}_{1:T,1:N}^s)$, since the
39 high correlation between \mathbf{z} and $\boldsymbol{\theta}$ makes this coordinate-wise approach rather slow.
40

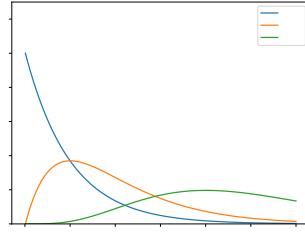
41 30.5 HMMs: Generalizations

42 In this section, we discuss various extensions of the vanilla HMM introduced in Section 30.1.
43

1
2
3
4
5
6
7
8
9
10



(a)



(b)

11
12 Figure 30.18: (a) A Markov chain with $n = 4$ repeated states and self loops. (b) The resulting distribution
13 over sequence lengths, for $p = 0.99$ and various n . Generated by `hmm_self_loop_dist.py`.

14
15
16

30.5.1 Hidden semi-Markov model (HSMM)

17 In a standard HMM (Section 30.1), the probability we remain in state i for exactly d steps is
18

$$19 \quad p(d_i = d) = (1 - A_{ii})A_{ii}^d \propto \exp(d \log A_{ii}) \quad (30.36)$$

20 where A_{ii} is the self-loop probability. This is called the **geometric distribution**. However, this
21 kind of exponentially decaying function of d is sometimes unrealistic.

22 A simple way to model non-geometric waiting times is to replace each state with n new states,
23 each with the same emission probabilities as the original state. For example, consider the model in
24 Figure 30.18(a). Obviously the smallest sequence this can generate is of length $n = 4$. Any path
25 of length d through the model has probability $p^{d-n}(1-p)^n$; multiplying by the number of possible
26 paths we find that the total probability of a path of length d is
27

$$28 \quad p(d) = \binom{d-1}{n-1} p^{d-n}(1-p)^n \quad (30.37)$$

29 This is equivalent to the negative binomial distribution. By adjusting n and the self-loop probabilities
30 p of each state, we can model a wide range of waiting times: see Figure 30.18(b).

31 A more general solution is to use a **semi-Markov model**, in which the next state not only depends
32 on the previous state, but also on how long we've been in that state. When the state space is
33 not observed directly, the result is called a **hidden semi-Markov model (HSMM)**, a **variable**
34 **duration HMM**, or an **explicit duration HMM** [Yu10].

35 One way to represent a HSMM is to use the graphical model shown in Figure 30.19(a). The
36 $d_t \in \{1, \dots, D\}$ node is a state duration counter, where D is the maximum duration of any state.
37 When we first enter state j , we sample d_t from the duration distribution for that state, $d_t \sim p_j(\cdot)$.
38 Thereafter, d_t deterministically counts down until $d_t = 1$. More precisely, we define the following
39 CPD:
40

$$41 \quad p(d_t = d' | d_{t-1} = d, z_t = j) = \begin{cases} D_j(d') & \text{if } d = 1 \\ 1 & \text{if } d' = d - 1 \text{ and } d > 0 \\ 0 & \text{otherwise} \end{cases} \quad (30.38)$$

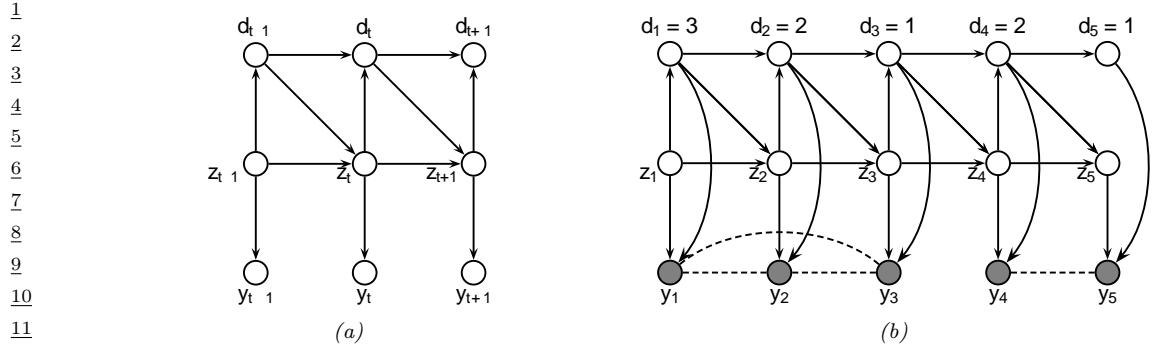


Figure 30.19: Encoding a hidden semi-Markov model as a PGM-D. (a) d_t is a deterministic down counter (duration variable). Each observation is generated independently. (b) Similar to (a), except now we generate the observations within each segment as a block. In this figure, we represent the non-Markovian dependencies between the observations within each segment by using undirected edges. We represent the conditional independence between the observations across different segments by disconnecting $y_{1:3}$ from $y_{4:5}$; this can be enforced by ‘breaking the link’ whenever $d_t = 1$ (representing the end of a segment).

Note that $D_j(d)$ could be represented as a table (a non-parametric approach) or as some kind of parametric distribution, such as a Gamma distribution. If $D_j(d)$ is a geometric distribution, this emulates a standard HMM.

While $d_t > 1$, the state z_t is not allowed to change. When $d_t = 1$, we make a stochastic transition to a new state. More precisely, we define the state CPD as follows:

$$p(z_t = k | z_{t-1} = j, d_{t-1} = d) = \begin{cases} 1 & \text{if } d > 0 \text{ and } j = k \\ A_{jk} & \text{if } d = 1 \\ 0 & \text{otherwise} \end{cases} \quad (30.39)$$

This ensures that the model stays in the same state for the entire duration of the segment. At each step within this segment, an observation is generated.

HSMMs are useful not only because they can model the duration of each state explicitly, but also because they can model the distribution of a whole subsequence of observations at once, instead of assuming all observations are generated independently at each time step. That is, they can use likelihood models of the form $p(\mathbf{y}_{t:t+l} | z_t = k, d_t = l)$, which generate l correlated observations if the duration in state k is for l time steps. This approach, known as a **segmental HMM**, is useful for modeling data that is piecewise linear, or shows other local trends [ODK96]. We can also use an RNN to model each segment, resulting in an **RNN-HSMM** model [Dai+17].

More precisely, we can define a segmental HMM as follows:

$$p(\mathbf{y}, \mathbf{z}, \mathbf{d}) = \left[p(z_1) p(d_1 | z_1) \prod_{t=2}^T p(z_t | z_{t-1}, d_{t-1}) p(d_t | z_t, d_{t-1}) \right] p(\mathbf{y} | \mathbf{z}, \mathbf{d}) \quad (30.40)$$

In a standard HSMM, we assume

$$p(\mathbf{y} | \mathbf{z}, \mathbf{d}) = \prod_{t=1}^T p(y_t | z_t) \quad (30.41)$$

so the duration variables only determine the hidden state dynamics. To define $p(\mathbf{y}|\mathbf{z}, \mathbf{d})$ for a segmental HMM, let us use s_i and e_i to denote the start and end times of segment i . This sequence can be computed deterministically from \mathbf{d} using $s_1 = 1$, $s_i = s_{i-1} + d_{s_{i-1}}$, and $e_i = s_i + d_{s_i} - 1$. We now define the observation model as follows:

$$p(\mathbf{y}|\mathbf{z}, \mathbf{d}) = \prod_{i=1}^{|s|} p(\mathbf{y}_{s_i:e_i} | z_{s_i}, d_{s_i}) \quad (30.42)$$

See Figure 30.19(b) for the PGM-D.

If we use an RNN for each segment, we have

$$p(\mathbf{y}_{s_i:e_i} | z_{s_i}, d_{s_i}) = \prod_{t=s_i}^{e_i} p(y_t | \mathbf{y}_{s_i:t-1}, z_{s_i}) = \prod_{t=s_i}^{e_i} p(y_t | h_t, z_{s_i}) \quad (30.43)$$

where h_t is the hidden state that is deterministically updated given the previous observations in this sequence.

As shown in [Chi14], it is possible to compute $p(z_t, d_t | \mathbf{y}_{1:T})$ in $O(TK^2 + TKD)$ time, where T is the sequence length, K is the number of states, and D is the maximum duration of any segment. In [Dai+17], they show how to train an approximate inference algorithm, based on a mean field approximation $q(\mathbf{z}, \mathbf{d} | \mathbf{y}) = \prod_t q(z_t | \mathbf{y})q(d_t | \mathbf{y})$, to compute the posterior in $O(TK + TD)$ time.

30.5.2 HSMMs for changepoint detection

In this section, we discuss how to use HSMM-like models for the problem of **changepoint detection**, which is the task of detecting when there are “abrupt” changes in the distribution of the observed values in a time series. For a review of offline methods to this problem, see e.g., [AC17; TOV18]. (See also [BW20] for a recent empirical evaluation of various methods, focused on the 1d time series case.)

In this section, we focus on the online case. The methods we discuss can (in principle) be used for high-dimensional time series segmentation. Our starting point is the hidden semi-Markov models (HSMM) discussed in Section 30.5.1. This is like an HMM in which we explicitly model the duration spent in each state. This is done by augmenting the latent state z_t with a duration variable d_t which is initialized according to a duration distribution, $d_t \sim D_{z_t}(\cdot)$, and which then *counts down* to 0. An alternative approach is to add a variable $r_t \{0, 1, \dots\}$ which encodes the **run length** for the current state; this starts at 0 whenever a new segment is created, and then *counts up* by one at each step. The transition dynamics is specified by

$$p(r_t | r_{t-1}) = \begin{cases} H(r_{t-1} + 1) & \text{if } r_t = 0 \\ 1 - H(r_{t-1} + 1) & \text{if } r_t = r_{t-1} + 1 \\ 0 & \text{otherwise} \end{cases} \quad (30.44)$$

where $H(\tau)$ is a **hazard function**:

$$H(\tau) = \frac{p_g(\tau)}{\sum_{t=\tau}^{\infty} p_g(t)} \quad (30.45)$$

where $p_g(t)$ is the probability of a gap of length t . See Figure 30.20 for an illustration. If we set p_g to be a geometric distribution with parameter λ , then the hazard function is the constant $H(\tau) = 1/\lambda$.

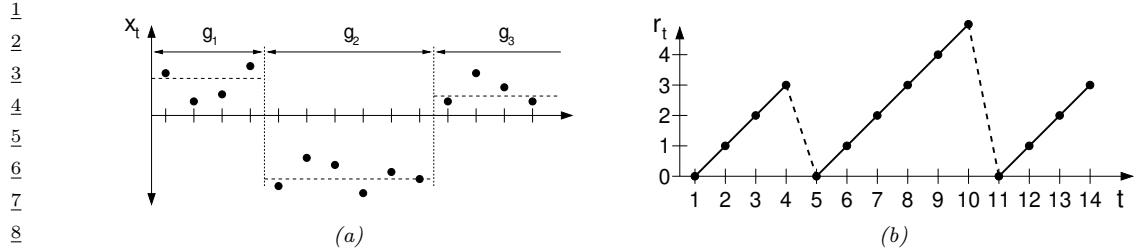


Figure 30.20: Illustration of Bayesian online changepoint detection (BOCPD). (a) Hypothetical segmentation of a univariate time series divided by changepoints on the mean into three segments of lengths $g_1 = 4$, $g_2 = 6$, and an undetermined length for g_3 (since it the third segment has not yet ended). From Figure 1 of [AM07]. Used with kind permission of Ryan Adams.

The advantage of the run-length representation is that we can define the observation model for a segment in a causal way (that only depends on past data):

$$p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, r_t = r, z_t = k) = p(\mathbf{y}_t | \mathbf{y}_{t-r:t-1}, z_t = k) = \int p(\mathbf{y}_t | \boldsymbol{\eta}) p(\boldsymbol{\eta} | \mathbf{y}_{t-r:t-1}, z_t = k) d\boldsymbol{\eta} \quad (30.46)$$

where $\boldsymbol{\eta}$ are the parameters that are “local” to this segment. This called the **underlying predictive model** or **UPM** for the segment. The posterior over the UPM parameters is given by

$$p(\boldsymbol{\eta} | \mathbf{y}_{t-r:t-1}, z_t = k) \propto p(\boldsymbol{\eta} | z_t = k) \prod_{i=t-r}^{t-1} p(\mathbf{y}_i | \boldsymbol{\eta}) \quad (30.47)$$

where we initialize the prior for $\boldsymbol{\eta}$ using hyper-parameters chosen by state k . If the model is conjugate exponential, we can compute this marginal likelihood in closed form, and we have

$$\pi_t^{r,k} = p(\mathbf{y}_t | \mathbf{y}_{t-r:t-1}, z_t = k) = p(\mathbf{y}_t | \psi_t^{r,k}) \quad (30.48)$$

where $\psi_t^{r,k}$ are the parameters of the posterior predictive distribution at time t based on the last r observations (and using a prior from state k).

In the special case in which we have $K = 1$ hidden states, then each segment is modeled independently, and we get a **product partition model** [BH92]:

$$p(\mathbf{y} | \mathbf{r}) = p(\mathbf{y}_{s_1:e_1}) \dots p(\mathbf{y}_{s_N:e_N}) \quad (30.49)$$

where s_i and e_i are the start and end of segment i , which can be computed from the run lengths \mathbf{r} . (We initialize with $r_0 = 0$.) Thus there is no information sharing between segments. This can be useful for timeseries in which there are abrupt changes, and where the new parameters are unrelated to the old ones.

Detecting the locations of these changes is called **changepoint detection**. An exact online algorithm for solving this task was proposed in [FL07] and independently in [AM07]; in the latter paper, they call the method **Bayesian online changepoint detection** or **BOCPD**. We can compute a posterior over the current run length recursively as follows:

$$p(r_t | \mathbf{y}_{1:t}) \propto p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, r_t) p(r_t | \mathbf{y}_{1:t-1}) \quad (30.50)$$

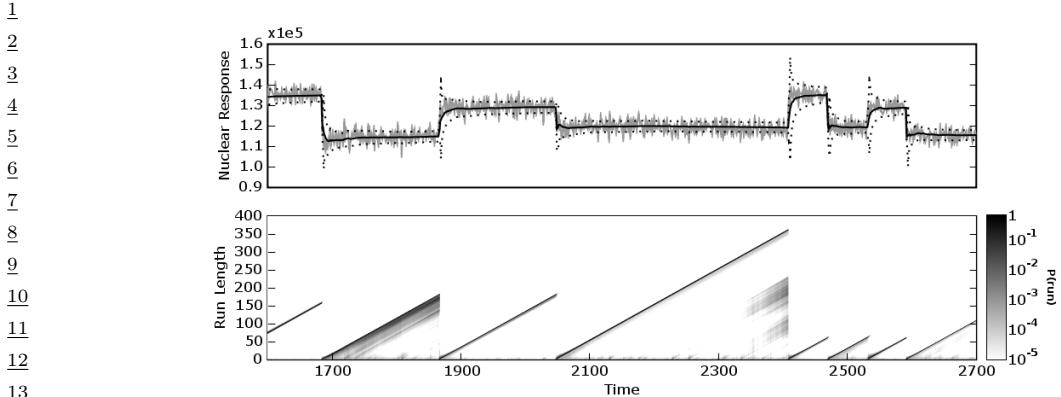


Figure 30.21: Illustration of BOCPD applied to the well-log dataset. Top row: Nuclear magnetic response during the drilling of a well. The data are plotted in light gray, the dark line shows the posterior predictive mean, and the dashed lines the 1σ error bars. Bottom row: Posterior probability over run lengths, $p(r_t|y_{1:t})$, using a logarithmic scale. Darker colors indicate higher probability. From Figure 2 of [AM07]. Used with kind permission of Ryan Adams.

where we initialize with $p(r_0 = 0) = 1$. The marginal likelihood $p(y_t|y_{1:t-1}, r_t)$ is given by Equation (30.46) (with $z_t = 1$ dropped, since there is just one state). The prior predictive is given by

$$p(r_t|y_{1:t-1}) = \sum_{r_{t-1}} p(r_t|r_{t-1})p(r_{t-1}|y_{1:t-1}) \quad (30.51)$$

The one step ahead predictive distribution is given by

$$p(y_{t+1}|y_{1:t}) = \sum_{r_t} p(y_{t+1}|y_{1:t}, r_t)p(r_t|y_{1:t}) \quad (30.52)$$

30.5.2.1 Example

See Figure 30.21 for an illustration of this method applied to a 1d well-log dataset from [RF96; FC03]. The likelihood is a univariate Gaussian, $p(y_t|\mu) = \mathcal{N}(y_t|\mu, \sigma^2)$, where $\sigma^2 = 4000^2$ is fixed, and μ is inferred using a Gaussian prior $p(\mu) = \mathcal{N}(\mu|\mu_0 = 1.15 \times 10^5, \sigma_0^2 = 1 \times 10^8)$. The hazard function is set to a geometric distribution, $H(r_t) = 1/\lambda$ where $\lambda = 250$. We see that sudden changes in the mean are detected, and then the run length resets to 0. The posterior predictive distribution at the start of each segment is broad, but rapidly concentrates as more data is collected. Despite the simplicity of this method, it was shown to be one of the best performing approaches to changepoint detection in an extensive recent benchmark [BW20].

30.5.2.2 Extensions

Although the above method is exact, each update step takes $O(t)$ time, so the total cost of the algorithm is $O(T^2)$. We can reduce this by pruning out states with low probability. In particular, we

¹ can use particle filtering (Section 13.2) with N particles, together with a stratified optimal resampling
² method, to reduce the cost to $O(TN)$. See [FL07] for details.
³

⁴ In addition, the above method relies on a conjugate exponential model in order to compute the
⁵ marginal likelihood, and update the posterior parameters for each r_t in $O(1)$ time. For more complex
⁶ models, we need to use approximations. In [TBS13], they use variational Bayes (Section 10.2.3), and
⁷ in [Mav16], they use particle filtering (Section 13.2), which is more general, but much slower.

⁸ It is possible to extend the model in various other ways. In [FL11], they allow for Markov
⁹ dependence between the parameters of neighboring segments. In [STR10], they use a Gaussian
¹⁰ process (Chapter 18) to represent the UPM, which captures correlations between observations within
¹¹ the same segment. In [KJD18], they use generalized Bayesian inference (Section 14.1.3) to create a
¹² method that is more robust to model misspecification.

¹³ In [Gol+17], they extend the model by modeling the probability of a sequence of observations,
¹⁴ rather than having to make the decision about whether to insert a changepoint or not based on just
¹⁵ on the likelihood ratio of a single time step.

¹⁶ In [AE+20], they extend the model by allowing for multiple discrete states, as in an HSMM. In
¹⁷ addition, they add both the run length r_t and the duration d_t to the state space. This allows the
¹⁸ method to specify not just when the current segment started, but also when it is expected to end.
¹⁹ In addition, it allows the UPM to depend on the duration of the segment, and not just on past
²⁰ observations. For example, we can use

$$\begin{aligned} \text{21} \quad p(y_t | r_t, d_t, \eta) &= \mathcal{N}(y_t | \phi(r_t/d_t)^\top \boldsymbol{\eta}, \sigma^2) \\ \text{22} \end{aligned} \tag{30.53}$$

²³ where $0 \leq r_t/d_t \leq 1$, and $\phi()$ is a set of learned basis functions. This allows observation sequences
²⁴ for the same hidden state to have a common functional shape, even if the time spent in each state is
²⁵ different.

²⁶

²⁷ 30.5.3 Hierarchical HMMs

²⁸

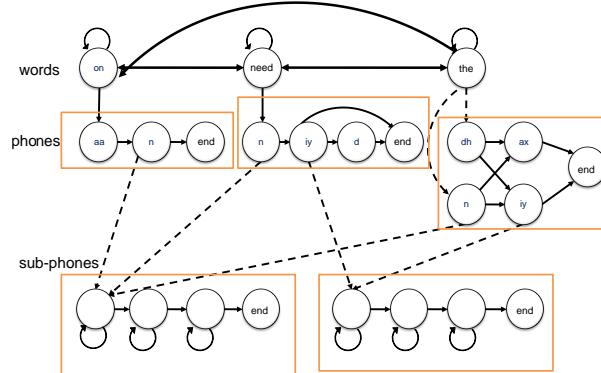
²⁹ A **hierarchical HMM** (HHMM) [FST98] is an extension of the HMM that is designed to model
³⁰ domains with hierarchical structure. Figure 30.22 gives an example of an HHMM used in automatic
³¹ speech recognition, where words are composed of phones which are composed of subphones. We
³² can always “flatten” an HHMM to a regular HMM, but a factored representation is often easier to
³³ interpret, and allows for more efficient inference and model fitting.

³⁴ HHMMs have been used in many application domains, e.g., speech recognition [Bil01], gene finding
³⁵ [Hu+00], plan recognition [BVW02], monitoring transportation patterns [Lia+07], indoor robot
³⁶ localization [TMK04], etc. HHMMs are less expressive than stochastic context free grammars (SCFGs)
³⁷ since they only allow hierarchies of bounded depth, but they support more efficient inference. In
³⁸ particular, inference in SCFGs (using the inside outside algorithm, [JM08]) takes $O(T^3)$ whereas
³⁹ inference in an HHMM takes $O(T)$ time [MP01; WM12].

⁴⁰ We can represent an HHMM as a directed graphical model as shown in Figure 30.23. Q_t^ℓ represents
⁴¹ the state at time t and level ℓ . A state transition at level ℓ is only “allowed” if the chain at the level
⁴² below has “finished”, as determined by the $F_t^{\ell-1}$ node. (The chain below finishes when it chooses to
⁴³ enter its end state.) This mechanism ensures that higher level chains evolve more slowly than lower
⁴⁴ level chains, i.e., lower levels are nested within higher levels.

⁴⁵ A variable duration HMM can be thought of as a special case of an HHMM, where the top level is
⁴⁶ a deterministic counter, and the bottom level is a regular HMM, which can only change states once
⁴⁷

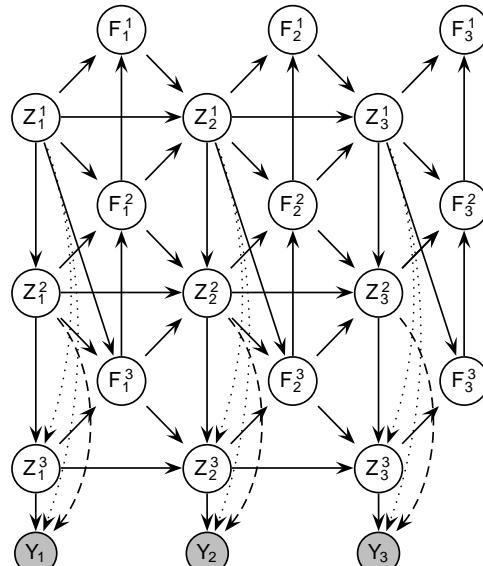
1
2
3
4
5
6
7
8
9
10
11
12
13
14



15
16
17
18
19
20
21
22

Figure 30.22: An example of an HHMM for an ASR system which can recognize 3 words. The top level represents bigram word probabilities. The middle level represents the phonetic spelling of each word. The bottom level represents the subphones of each phone. (It is traditional to represent a phone as a 3 state HMM, representing the beginning, middle and end; these are known as subphones.) Adapted from Figure 7.5 of [JM00].

23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41



42
43
44
45
46
47

Figure 30.23: An HHMM represented as a PGM-D. Z_t^ℓ is the state at time t , level ℓ ; $F_t^\ell = 1$ if the HMM at level ℓ has finished (entered its exit state), otherwise $F_t^\ell = 0$. Shaded nodes are observed; the remaining nodes are hidden. We may optionally clamp $F_T^\ell = 1$, where T is the length of the observation sequence, to ensure all models have finished by the end of the sequence. From Figure 2 of [MP01].

¹ the counter has “timed out”. See [MP01] for further details.
²

³ 30.5.4 Factorial HMMs

⁴ An HMM represents the hidden state using a single discrete random variable $z_t \in \{1, \dots, K\}$. To
⁵ represent 10 bits of information would require $K = 2^{10} = 1024$ states. By contrast, consider a
⁶ **distributed representation** of the hidden state, where each $z_{t,m} \in \{0, 1\}$ represents the m 'th bit
⁷ of the t 'th hidden state. Now we can represent 10 bits using just 10 binary variables. This model is
⁸ called a **factorial HMM** [GJ97].
⁹

¹⁰ More precisely, the model is defined as follows:
¹¹

$$\begin{aligned} \text{12} \quad p(\mathbf{z}, \mathbf{y}) &= \prod_t \left[\prod_m p(z_{tm} | z_{t-1,m}) \right] p(\mathbf{y}_t | \mathbf{z}_t) \\ \text{13} \end{aligned} \tag{30.54}$$

¹⁴ where $p(z_{tm} = k | z_{t-1,m} = j) = A_{mj_k}$ is an entry in the transition matrix for chain m , $p(z_{1m} =$
¹⁵ $k | z_{0m}) = p(z_{1m} = k) = \pi_{mk}$, is the initial state distribution for chain m , and

$$\begin{aligned} \text{16} \quad p(\mathbf{y}_t | \mathbf{z}_t) &= \mathcal{N} \left(\mathbf{y}_t | \sum_{m=1}^M \mathbf{W}_m \mathbf{z}_{tm}, \boldsymbol{\Sigma} \right) \\ \text{17} \end{aligned} \tag{30.55}$$

¹⁸ is the observation model, where \mathbf{z}_{tm} is a 1-of- K encoding of z_{tm} and \mathbf{W}_m is a $D \times K$ matrix (assuming
¹⁹ $\mathbf{y}_t \in \mathbb{R}^D$). Figure 30.24(a) illustrates the model for the case where $M = 3$.

²⁰ An interesting application of FHMMs is to the problem of **energy disaggregation** [KJ12a].
²¹ In this problem, we observe the total energy usage of a house at each moment in time, i.e., the
²² observation model has the form

$$\begin{aligned} \text{23} \quad p(y_t | \mathbf{z}_t) &= \mathcal{N}(y_t | \sum_{m=1}^M w_m z_{tm}, \sigma^2) \\ \text{24} \end{aligned} \tag{30.56}$$

²⁵ where w_m is the amount of energy used by device m , and $z_{tm} = 1$ if device m is being used at time t
²⁶ and $z_{tm} = 0$ otherwise. The transition model is assumed to be

$$\begin{aligned} \text{27} \quad p(z_{t,m} = 1 | z_{t-1,m}) &= \begin{cases} A_{01} & \text{if } z_{t-1,m} = 0 \\ A_{11} & \text{if } z_{t-1,m} = 1 \end{cases} \\ \text{28} \end{aligned} \tag{30.57}$$

²⁹ We do not know which devices are turned on at each time step (i.e., the z_{tm} are hidden), but by
³⁰ applying inference in the FHMM over time, we can separate the total energy into its parts, and
³¹ thereby determine which devices are using the most electricity.

³² 30.5.4.1 Structured mean field for FHMMs

³³ Even though each chain in an FHMM is a priori independent, they become coupled in the posterior due
³⁴ to having an observed common child, \mathbf{y}_t . Exact inference for this model therefore takes $O(TM K^{M+1})$
³⁵ time. In this section, we derive a structured mean field algorithm, from [GJ97], that takes $O(TMK^2 I)$
³⁶ time, where I is the number of mean field iterations (typically $I \sim 10$ suffices for good performance).
³⁷

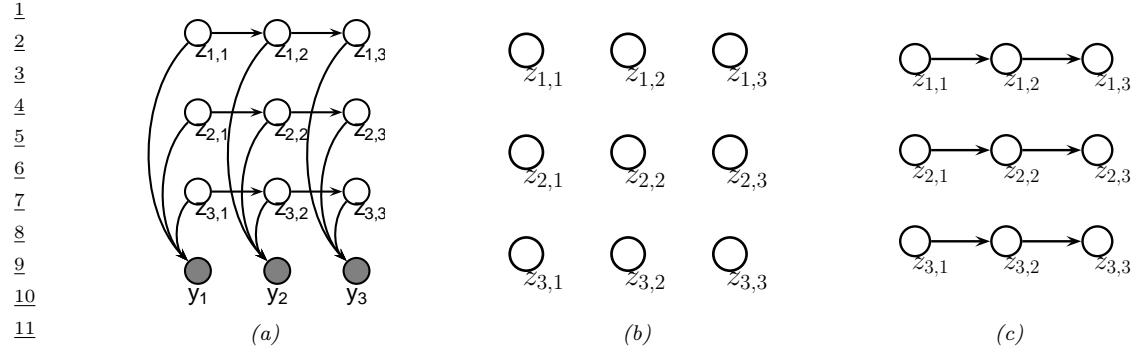


Figure 30.24: (a) A factorial HMM with 3 chains. (b) A fully factorized approximation. (c) A product-of-chains approximation. Adapted from Figure 2 of [GJ97].

We can write the exact posterior in the following form:

$$p(\mathbf{z}|\mathbf{y}) = \frac{1}{Z(\mathbf{y})} \exp(-\mathcal{E}(\mathbf{z}, \mathbf{y})) \quad (30.58)$$

$$\begin{aligned} \mathcal{E}(\mathbf{z}, \mathbf{y}) &= \frac{1}{2} \sum_{t=1}^T \left(\mathbf{y}_t - \sum_m \mathbf{W}_m \mathbf{z}_{tm} \right)^T \Sigma^{-1} \left(\mathbf{y}_t - \sum_m \mathbf{W}_m \mathbf{z}_{tm} \right) \\ &\quad - \sum_{m=1}^M \mathbf{z}_{1m}^\top \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^T \sum_{m=1}^M \mathbf{z}_{t-1,m}^\top \tilde{\mathbf{A}}_m \mathbf{z}_{t,m} \end{aligned} \quad (30.59)$$

where $\tilde{\mathbf{A}}_m \triangleq \log \mathbf{A}_m$ and $\tilde{\boldsymbol{\pi}}_m \triangleq \log \boldsymbol{\pi}_m$, where the log is applied elementwise.

We can approximate the posterior as a product of marginals, as in Figure 30.24(b), but a better approximation is to use a product of chains, as in Figure 30.24(c). Each chain can be tractably updated individually, using the forwards-backwards algorithm (Section 8.3.3). More precisely, we assume

$$q(\mathbf{z}|\mathbf{y}) = \frac{1}{Z_q} \prod_{m=1}^M q(z_{1m}|\boldsymbol{\psi}_{1m}) \prod_{t=2}^T q(z_{tm}|z_{t-1,m}, \boldsymbol{\psi}_{tm}) \quad (30.60)$$

$$q(z_{1m}|\boldsymbol{\psi}_{1m}) = \prod_{k=1}^K (\xi_{1mk} \pi_{mk})^{z_{1mk}} \quad (30.61)$$

$$q(z_{tm}|z_{t-1,m}, \boldsymbol{\psi}_{tm}) = \prod_{k=1}^K \left(\xi_{tmk} \prod_{j=1}^K (A_{mjk})^{z_{t-1,m,j}} \right)^{z_{tmk}} \quad (30.62)$$

Here the variational parameter ξ_{tmk} plays the role of an approximate local evidence, averaging out the effects of the other chains. This is in contrast to the exact local evidence, which couples all the chains together.

By separating out the approximate local evidence terms, we can rewrite the above as $q(\mathbf{z}|\mathbf{y}) =$

1 $\frac{1}{Z_q(\mathbf{y})} \exp(-\mathcal{E}_q(\mathbf{z}, \mathbf{y}))$, where

2

$$\mathcal{E}_q(\mathbf{z}, \mathbf{y}) = -\sum_{t=1}^T \sum_{m=1}^M \mathbf{z}_{tm}^\top \tilde{\boldsymbol{\psi}}_{tm} - \sum_{m=1}^M \mathbf{z}_{1m}^\top \tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^T \sum_{m=1}^M \mathbf{z}_{t-1,m}^\top \tilde{\mathbf{A}}_m \mathbf{z}_{t,m} \quad (30.63)$$

3

4 where $\tilde{\boldsymbol{\psi}}_{tm} = \log \boldsymbol{\psi}_{tm}$. We see that this has the same temporal factors as the exact log joint in
5 Equation (30.59), but the local evidence terms are different: the dependence on the visible data \mathbf{y}
6 has been replaced by dependence on “virtual data” $\boldsymbol{\psi}$.

7 The objective function is given by

8

$$D_{\text{KL}}(q \parallel \tilde{p}) = \mathbb{E}_q [\log q - \log \tilde{p}] \quad (30.64)$$

9

$$= -\mathbb{E}_q [\mathcal{E}_q(\mathbf{z}, \mathbf{y})] - \log Z_q(\mathbf{y}) + \mathbb{E}_q [\mathcal{E}(\mathbf{z}, \mathbf{y})] + \log Z(\mathbf{y}) \quad (30.65)$$

10 where $q = q(\mathbf{z} | \mathbf{y})$ and $\tilde{p} = p(\mathbf{z} | \mathbf{y})$. One can show that we can optimize this using coordinate descent,
11 where each update step is given by

12

$$\boldsymbol{\psi}_{tm} = \exp \left(\mathbf{W}_m^\top \boldsymbol{\Sigma}^{-1} \tilde{\mathbf{y}}_{tm} - \frac{1}{2} \boldsymbol{\delta}_m \right) \quad (30.66)$$

13

$$\boldsymbol{\delta}_m \triangleq \text{diag}(\mathbf{W}_m^\top \boldsymbol{\Sigma}^{-1} \mathbf{W}_m) \quad (30.67)$$

14

$$\tilde{\mathbf{y}}_{tm} \triangleq \mathbf{y}_t - \sum_{\ell \neq m}^M \mathbf{W}_\ell \mathbb{E}[\mathbf{z}_{t,\ell}] \quad (30.68)$$

15 The intuitive interpretation of $\tilde{\mathbf{y}}_{tm}$ is that it is the observation \mathbf{y}_t minus the predicted effect from
16 all the other chains apart from m . This is then used to compute the approximate local evidence,
17 $\boldsymbol{\psi}_{tm}$. Having computed the $\boldsymbol{\psi}_{tm}$ terms for each chain, we can perform forwards-backwards in parallel,
18 using these approximate local evidence terms to compute $q(\mathbf{z}_{t,m} | \mathbf{y}_{1:T})$ for each m and t .

19 The update cost is $O(TM^2K^2)$ for a full “sweep” over all the variational parameters, since we have
20 to run forwards-backwards M times, for each chain independently. This is the same cost as a fully
21 factorized approximation, but is much more accurate.

22 30.5.5 Coupled HMMs

23 If we have multiple related data streams, we can use a **coupled HMM** [Bra96]. This is a series of
24 HMMs where the state transitions depend on the states of neighboring chains. That is, we represent
25 the conditional distribution for each time slice as

26

$$p(\mathbf{z}_t, \mathbf{y}_t | \mathbf{z}_{t-1}) = \prod_m p(\mathbf{y}_{tm} | \mathbf{z}_{tm}) p(\mathbf{z}_{tm} | \mathbf{z}_{t-1, m-1:m+1}) \quad (30.69)$$

27 with boundary conditions defined in the obvious way. See Figure 30.25 for an illustration with $M = 3$
28 chains.

29 Coupled HMMs have been used for various tasks, such as **audio-visual speech recognition**
30 [Nef+02], modeling freeway traffic flows [KM00], and modeling conversational interactions between
31 people [Bas+01].

32

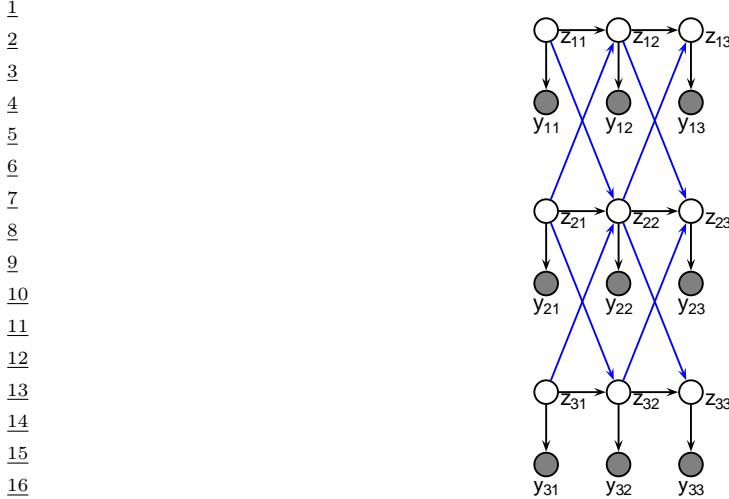


Figure 30.25: A coupled HMM with 3 chains.

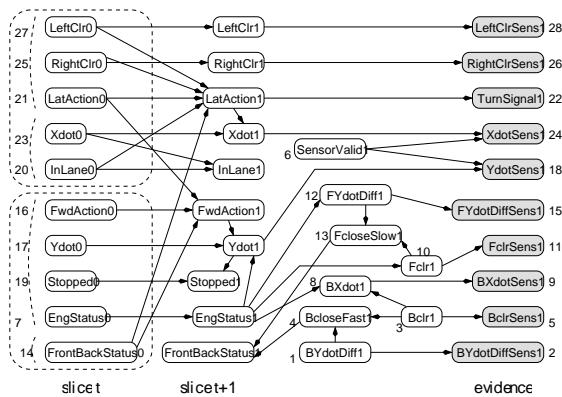


Figure 30.26: The BATnet DBN. The transient nodes are only shown for the second slice, to minimize clutter. The dotted lines are used to group related variables. Used with kind permission of Daphne Koller.

However, there are two drawbacks to this model. First, exact inference takes $O(T(K^M)^2)$, as in an factorial HMM; however, in practice this is not usually a problem, since M is often small. Second, the model requires $O(MK^4)$ parameters to specify, if there are M chains with K states per chain, because each state depends on its own past plus the past of its two neighbors. There is a closely related model, known as the **influence model** [Asa00], which uses fewer parameters, by computing a convex combination of pairwise transition matrices.

1 **30.5.6 Dynamic Bayes nets (DBN)**

2 A **dynamic Bayesian network (DBN)** is a way to represent a stochastic process using a directed
3 graphical model [Mur02]. (Note that the network is not dynamic (the structure and parameters are
4 fixed), rather it is a network representation of a dynamical system.) A DBN can be considered as a
5 natural generalization of an HMM.

6 An example is shown in Figure 30.26, which is a DBN designed to monitor the state of a simulated
7 autonomous car known as the “Bayesian Automated Taxi”, or “BATmobile” [For+95]. To define
8 the model, you just need to specify the structure of the first time-slice, the structure between two
9 time-slices, and the form of the CPDs. For details, see [KF09a].

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

31 State-space models

31.1 Introduction

A **state space model (SSM)** is a generalization of an HMM in which the hidden state vector, which evolves over time, is real-valued, $\mathbf{z}_t \in \mathbb{R}^L$. This generates some noisy observations, often real-valued, $\mathbf{y}_t \in \mathbb{R}^D$. The goal is to model the relationship between the hidden state trajectory, $\mathbf{z}_{1:T}$, and the observed data sequence, $\mathbf{y}_{1:T}$. We then use an inference algorithm to infer the hidden state, $p(\mathbf{z}_{1:t} | \mathbf{y}_{1:t})$, and/or predict future visible states, $p(\mathbf{y}_{t+1:T} | \mathbf{y}_{1:t})$. (We may also have optional observed inputs, such as **controls** or **actions**, denoted $\mathbf{u}_{1:T}$.)

We will focus on discrete-time SSMs, which can be written as follows:

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t, \boldsymbol{\epsilon}_t) \tag{31.1}$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t, \boldsymbol{\eta}_t) \tag{31.2}$$

where f is the dynamics model, h is the observation model, and $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\eta}_t$ are the dynamics and observation noise. This defines the following joint distribution:

$$p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T} | \mathbf{u}_{1:T}) = \left[p(\mathbf{z}_1 | \mathbf{u}_1) \prod_{t=2}^T p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) \right] \left[\prod_{t=1}^T p(\mathbf{y}_t | \mathbf{z}_t) \right] \tag{31.3}$$

where $p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t)$ is the transition model induced by $f_z(\mathbf{z}_{t-1}, \mathbf{u}_t, \boldsymbol{\epsilon}_t)$ and $p(\mathbf{y}_t | \mathbf{z}_t)$ is the observation model induced by $f_x(\mathbf{z}_t, \boldsymbol{\eta}_t)$. See Figure 31.1 for an illustration of the corresponding graphical model.

There are a variety of algorithms for performing inference in this model, depending on the assumptions we make about the forms of these distributions. For example, if everything is Gaussian, we can use Kalman filtering and its variants, as discussed in Chapter 8. For more general models, we can use variational inference (Chapter 10) or particle filtering (Section 13.2). We will illustrate some of these algorithms in the sections below, in the context of specific models. For more details, see e.g., [Sim06; Fra08; DK12; Sar13; PFW21; Tri21].

31.2 Linear dynamical systems

Consider the state space model in Equation (31.3). If we assume **additive Gaussian noise**, the model becomes

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \boldsymbol{\epsilon}_t \tag{31.4}$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t) + \boldsymbol{\eta}_t \tag{31.5}$$

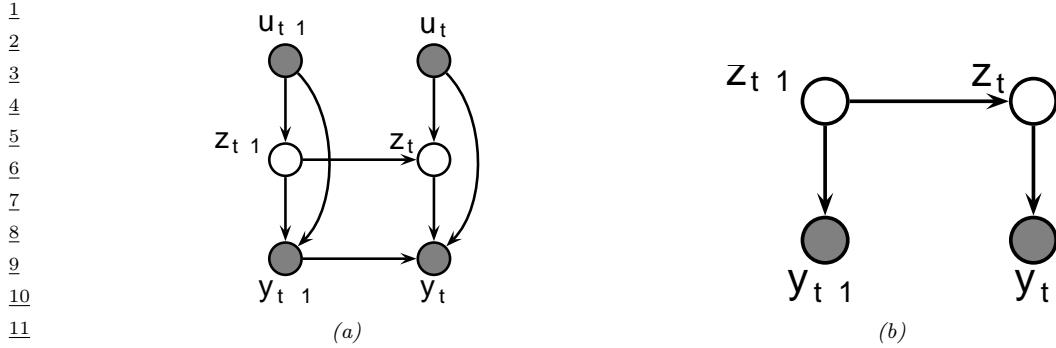


Figure 31.1: State-space model represented as a graphical model. (a) Generic form, with inputs u_t , hidden state z_t , and observations y_t . We assume the observation likelihood is first-order auto-regressive. (b) Simplified form, with no inputs, and Markovian observations.

where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$ and $\eta_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$. We will call these **Gaussian SSMs**.

An important special case of a Gaussian SSM arises when f and h are linear functions. This is known as a **linear-Gaussian state space model (LG-SSM)** or **linear dynamical system (LDS)**. In this case, we have

$$z_t = \mathbf{A}_t z_{t-1} + \mathbf{B}_t u_t + \epsilon_t \quad (31.6)$$

$$y_t = \mathbf{C}_t z_t + \mathbf{D}_t u_t + \eta_t \quad (31.7)$$

Typically we assume the parameters $\theta_t = (\mathbf{A}_t, \mathbf{C}_t, \mathbf{B}_t, \mathbf{D}_t, \mathbf{Q}_t, \mathbf{R}_t)$ are independent of time, so the model is stationary. (We discuss how to estimate the parameters in Section 31.2.5.) Given the parameters, we discuss how to perform online posterior inference of the latent states using the Kalman filter in Section 8.4.1, and offline inference using the Kalman smoother in Section 8.4.4. But first, we give some examples.

31.2.1 Example: Noiseless 1d spring-mass system

In this section, we consider an example from Wikipedia¹ of a **spring mass system** operating in 1d. Like many physical systems, this is best modeled in **continuous time**, although we will later discretize it.

Let $x(t)$ be the position of an object which is attached by a spring to a wall, and let $\dot{x}(t)$ and $\ddot{x}(t)$ be its velocity and acceleration. By **Newton's laws of motion**, we have the following **ordinary differential equation**:

$$m\ddot{x}(t) = u(t) - b\dot{x}(t) - kx(t) \quad (31.8)$$

where $u(t)$ is an externally applied force (e.g., someone tugging on the object), b is the viscous friction coefficient, k is the spring constant, and m is the mass of the object. See Figure 31.3 for the setup. We assume that we only observe the position, and not the velocity.

¹ 1. https://en.wikipedia.org/wiki/State-space_representation#Moving_object_example

We now proceed to represent this as a first order Markov system. For simplicity, we ignore the noise ($\mathbf{Q}_t = \mathbf{R}_t = \mathbf{0}$). We define the state space to contain the position and velocity, $\mathbf{z}(t) = [x(t), \dot{x}(t)]$. Thus the model becomes

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}(t) \quad (31.9)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{z}(t) + \mathbf{D}\mathbf{u}(t) \quad (31.10)$$

where

$$\begin{pmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix} u(t) \quad (31.11)$$

$$y(t) = (1 \ 0) \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} \quad (31.12)$$

To simulate from this system, we need to evaluate the state at a set of discrete time intervals, $t_k = k\Delta$, where Δ is the sampling rate or step size. There are many ways to discretize an ODE.² Here we discuss the **generalized bilinear transform** [ZCC07]. In this approach, we specify a step size Δ and compute

$$\mathbf{z}_{k+1} = \underbrace{(\mathbf{I} - \alpha\Delta\mathbf{A})^{-1}(\mathbf{I} + (1 - \alpha)\Delta\mathbf{A})}_{\bar{\mathbf{A}}} \mathbf{z}_k + \underbrace{\Delta(\mathbf{I} - \alpha\Delta\mathbf{A})^{-1}\mathbf{B}}_{\bar{\mathbf{B}}} \mathbf{u}_k \quad (31.13)$$

If we set $\alpha = 0$, we recover **Euler's method**, which simplifies to

$$\mathbf{z}_{k+1} = \underbrace{(\mathbf{I} + \Delta\mathbf{A})}_{\bar{\mathbf{A}}} \mathbf{z}_k + \underbrace{\Delta\mathbf{B}}_{\bar{\mathbf{B}}} \mathbf{u}_k \quad (31.14)$$

If we set $\alpha = 1$, we recover the **backward Euler method**. If we set $\alpha = \frac{1}{2}$ we get the **bilinear method**, which preserves the stability of the system [ZCC07]; we will use this in Section 31.5.5. Regardless of how we do the discretization, the resulting discrete time SSM becomes

$$\mathbf{z}_k = \bar{\mathbf{A}}\mathbf{z}_{k-1} + \bar{\mathbf{B}}\mathbf{u}_k \quad (31.15)$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{z}_k + \mathbf{D}\mathbf{u}_k \quad (31.16)$$

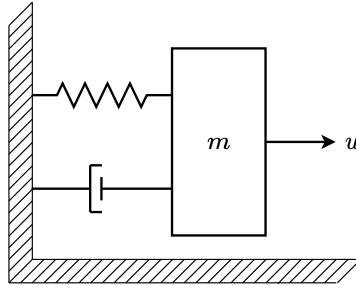
Now consider simulating a system where we periodically “tug” on the object, so the force increases and then decreases for a short period, as shown in the top row of Figure 31.3. We can discretize the dynamics and compute the corresponding state and observation at integer time points. The result is shown in the bottom row of Figure 31.3. We see that the object’s location changes smoothly, since it integrates the force over time.

31.2.2 Example: Noisy 2d tracking problem

We now consider a 2d example, such as a particle moving in \mathbb{R}^2 . We follow the approach in Section 31.2.1 to convert the dynamics to discrete time. For notational simplicity, we revert to

² See discussion at <https://en.wikipedia.org/wiki/Discretization>.

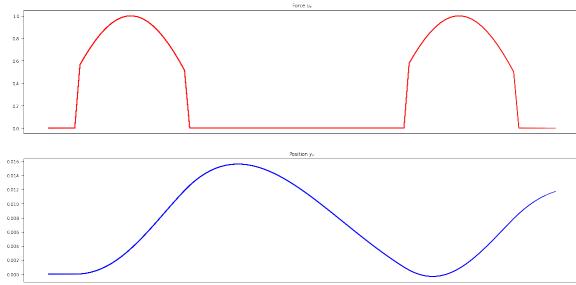
1
2
3
4
5
6
7
8
9
10



11
12

Figure 31.2: Illustration of the spring mass system.

13
14
15
16
17
18
19
20
21
22



23
24 Figure 31.3: Signals generated by the spring mass system. Top row shows the input force. Bottom row
25 shows the observed location of the end-effector. Adapted from A figure by Sasha Rush. Generated by
26 `ssm_spring_demo.ipynb`.

27
28
29
30

31 indexing (discrete) time with t rather than k , and we drop the overline symbol from the matrices.
32 Furthermore, we ignore any input or control signals.

33 Let the state be the position and velocity of the object, $\mathbf{z}_t = (u_t \quad \dot{u}_t \quad v_t \quad \dot{v}_t)$. (We use u and v
34 for the two coordinates, to avoid confusion with the state and observation variables.) If we use Euler
35 discretization, the dynamics become

36

$$\begin{aligned} 37 \quad \underbrace{\begin{pmatrix} u_t \\ \dot{u}_t \\ v_t \\ \dot{v}_t \end{pmatrix}}_{\mathbf{z}_t} &= \underbrace{\begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} u_{t-1} \\ \dot{u}_{t-1} \\ v_{t-1} \\ \dot{v}_{t-1} \end{pmatrix}}_{\mathbf{z}_{t-1}} + \epsilon_t \end{aligned} \tag{31.17}$$

43

44 where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the **process noise**.

45 Let us assume that the process noise is a **white noise process** added to the velocity components
46 of the state, but not to the location. (This is known as a **random accelerations model**.) One can
47

show [Sar13, p60] that the covariance matrix of the discrete time noise process is given by

$$\mathbf{Q} = \begin{pmatrix} \frac{q_1^c \Delta^3}{3} & 0 & \frac{q_1^c \Delta^2}{2} & 0 \\ 0 & \frac{q_2^c \Delta^3}{3} & 0 & \frac{q_2^c \Delta^2}{2} \\ \frac{q_1^c \Delta^2}{2} & 0 & q_1^c \Delta^2 & 0 \\ 0 & \frac{q_2^c \Delta^2}{2} & 0 & q_2^c \Delta^2 \end{pmatrix} \quad (31.18)$$

where q_i^c is the **spectral density** (continuous time variance) of the process noise for the i 'th component of the velocity. For simplicity, we often take $q_1^c = q_2^c = q$ and $\Delta = 1$, in which case this simplifies to

$$\mathbf{Q} = \begin{pmatrix} q/3 & 0 & q/2 & 0 \\ 0 & q/3 & 0 & q/2 \\ q/2 & 0 & q & 0 \\ 0 & q/2 & 0 & q \end{pmatrix} \quad (31.19)$$

We sometimes further approximate this by $\mathbf{Q} = q\mathbf{I}$.

Now suppose that at each discrete time point we observe the location, corrupted by Gaussian noise. Thus the observation model becomes

$$\underbrace{\begin{pmatrix} y_{1,t} \\ y_{2,t} \end{pmatrix}}_{\mathbf{y}_t} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{C}} \underbrace{\begin{pmatrix} u_t \\ \dot{u}_t \\ v_t \\ \dot{v}_t \end{pmatrix}}_{\mathbf{z}_t} + \boldsymbol{\eta}_t \quad (31.20)$$

where $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the **observation noise**. We see that the observation matrix \mathbf{C} simply “extracts” the relevant parts of the state vector.

Suppose we sample a trajectory and corresponding set of noisy observations from this model, $(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) \sim p(\mathbf{z}, \mathbf{y} | \boldsymbol{\theta})$. (We use diagonal observation noise, $\mathbf{R} = \text{diag}(\sigma_1^2, \sigma_2^2)$.) The results are shown in Figure 31.4(a). We can use the **Kalman filter** (Section 8.4.1) to compute $p(\mathbf{z}_t | \mathbf{y}_{1:t}, \boldsymbol{\theta})$ for each t . (We initialize the filter with a vague prior, namely $p(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0 | \mathbf{0}, 10^5 \mathbf{I})$.) The results are shown in Figure 31.4(b). We see that the posterior mean (red line) is close to the ground truth, but there is considerable uncertainty (shown by the confidence ellipses). To improve results, we can use the **Kalman smoother** (Section 8.4.4) to compute $p(\mathbf{z}_t | \mathbf{y}_{1:T}, \boldsymbol{\theta})$, where we condition on all the data, past and future. The results are shown in Figure 31.4(c). Now we see that the resulting estimate is smoother, and the uncertainty is reduced. (The uncertainty is larger at the edges because there is less information in the neighbors to condition on.)

The dynamics of the object in Figure 31.4 are rather simple, since we assumed a linear model with no external inputs, except for Gaussian process noise. However, suppose we change the (discrete time) linear transition matrix to the following:

$$\mathbf{A} = \begin{pmatrix} 0.1 & 1.1 & \Delta & 0 \\ -1 & 1 & 0 & \Delta \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{pmatrix} \quad (31.21)$$

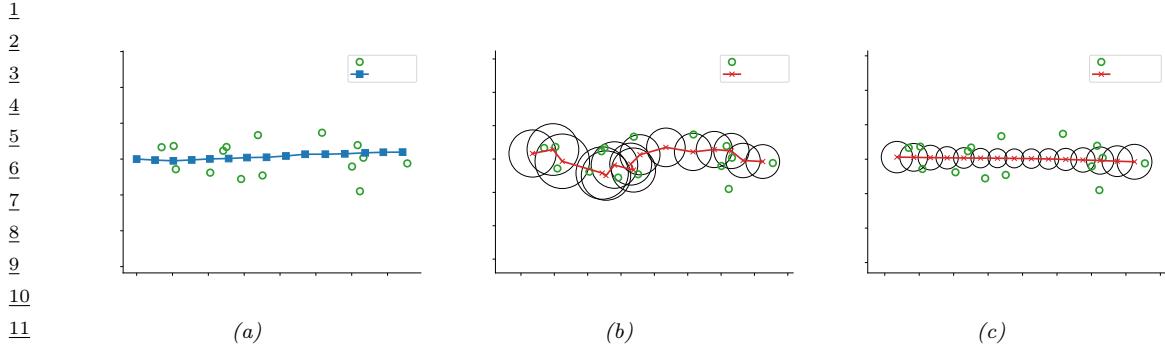


Figure 31.4: Illustration of Kalman filtering and smoothing for a linear dynamical system. (Repeated from Figure 8.5.) (a) Observations (green circles) are generated by an object moving to the right (true location denoted by blue squares). (b) Results of online Kalman filtering. Red cross is the posterior mean, circles are 95% confidence ellipses derived from the posterior covariance. (c) Same as (b), but using offline Kalman smoothing. The MSE in the trajectory for filtering is 3.13, and for smoothing is 1.71. Generated by kf_tracking_demo.py.

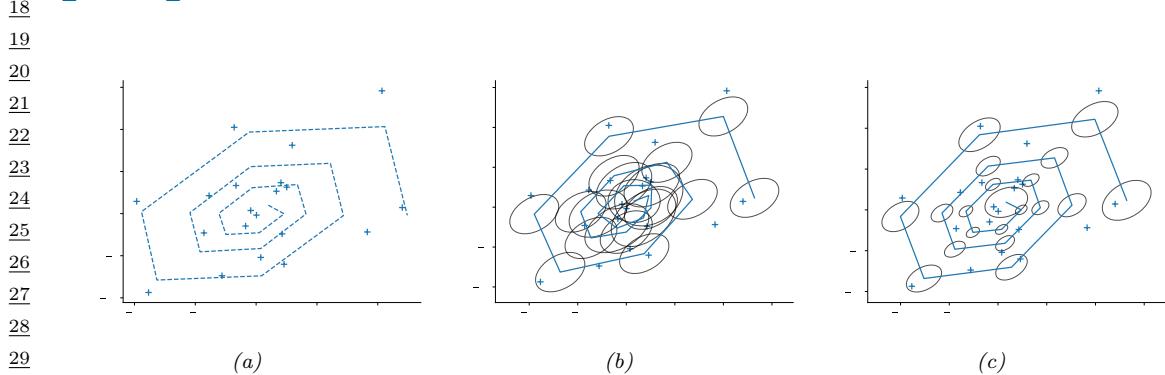


Figure 31.5: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observed data. (b) Filtering. (c) Smoothing. Generated by kf_spiral_demo.py.

The eigenvectors of the top left block of this transition matrix are complex, resulting in cyclical behavior, as explained in [Str15]. Furthermore, since the velocities are shrinking at each step by a factor of 0.1, the cycling behavior becomes a spiral inwards, as illustrated by the solid line in Figure 31.5(a). The crosses correspond to noisy measurements of the location, as before. In Figure 31.5(b-c), we show the results of Kalman filtering and smoothing.

31.2.3 Example: Online linear regression

In Section 15.2.1, we discuss how to compute $p(\mathbf{w}|\sigma^2, \mathcal{D})$ for a linear regression model in batch mode, using a Gaussian prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In this section, we discuss how to compute

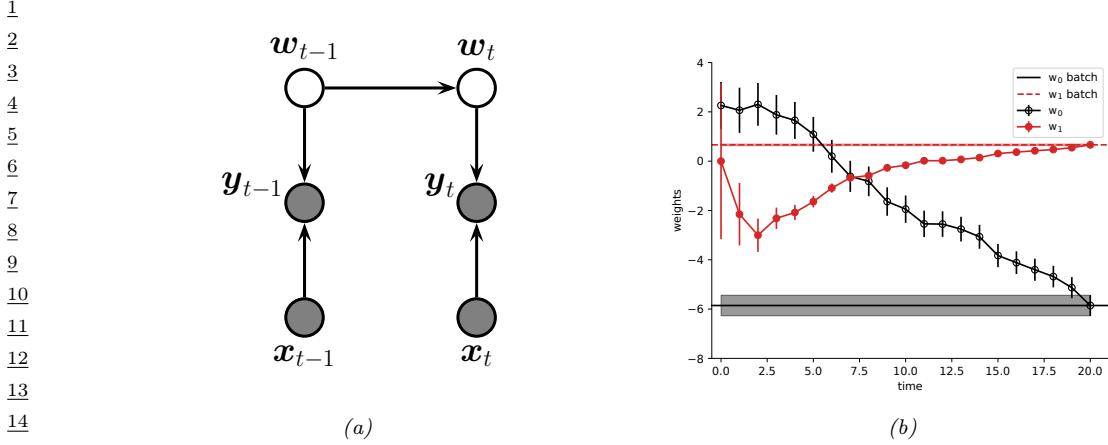


Figure 31.6: (a) A dynamic generalization of linear regression. (Repeated from Figure 8.9.) (b) Illustration of the recursive least squares algorithm applied to the model $p(y|x, \mathbf{w}) = \mathcal{N}(y|w_0 + w_1 x, \sigma^2)$. We plot the marginal posterior of w_0 and w_1 vs number of data points. (Error bars represent $\mathbb{E}[w_j|y_{1:t}, \mathbf{x}_{1:t}] \pm \sqrt{\mathbb{V}[w_j|y_{1:t}, \mathbf{x}_{1:t}]}$.) After seeing all the data, we converge to the offline (batch) Bayes solution, represented by the horizontal lines. (Shading represents the marginal posterior variance.) Generated by [linreg_kf_demo.py](#).

this posterior online, by repeatedly performing the following update:

$$p(\mathbf{w}|\mathcal{D}_{1:t}) \propto p(\mathcal{D}_t|\mathbf{w})p(\mathbf{w}|\mathcal{D}_{1:t-1}) \quad (31.22)$$

$$\propto p(\mathcal{D}_t|\mathbf{w})p(\mathcal{D}_{t-1}|\mathbf{w}) \dots p(\mathcal{D}_1|\mathbf{w})p(\mathbf{w}) \quad (31.23)$$

where $\mathcal{D}_t = (\mathbf{x}_t, y_t)$ is the t 'th labeled example, and $\mathcal{D}_{1:t-1}$ are the first $t-1$ examples. (For brevity, we drop the conditioning on σ^2 .) We see that the previous posterior, $p(\mathbf{w}|\mathcal{D}_{1:t-1})$, becomes the current prior, which gets updated by \mathcal{D}_t to become the new posterior, $p(\mathbf{w}|\mathcal{D}_{1:t})$. This is an example of sequential Bayesian updating or online Bayesian inference.

We can implement this method by using a linear Gaussian state space model. The basic idea is to let the hidden state represent the regression parameters, and to let the (time-varying) observation model represent the current data vector. If we assume the regression parameters do not change, the dynamics model becomes

$$p(\mathbf{w}_t|\mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t|\mathbf{w}_{t-1}, 0) = \delta(\mathbf{w}_t - \mathbf{w}_{t-1}) \quad (31.24)$$

*If we do let the parameters change over time, by adding non-zero **process noise** $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, we get a so-called **dynamic linear model** [Har90; WH97; PPC09].) The (non-stationary) observation model has the form

$$p(y_t|\mathbf{w}_t, \mathbf{x}_t) = \mathcal{N}(y_t|\mathbf{C}_t \mathbf{z}_t, \mathbf{R}_t) = \mathcal{N}(y_t|\mathbf{x}_t^\top \mathbf{w}_t, \sigma^2) \quad (31.25)$$

See Figure 31.6a for the model.

If we apply the Kalman filter to this model, we recover an algorithm known as **recursive least squares** or **RLS**; see Section 8.4.2 for the details. In Figure 8.9b, we show that this converges to the optimal offline Bayes posterior after a single pass over the data.

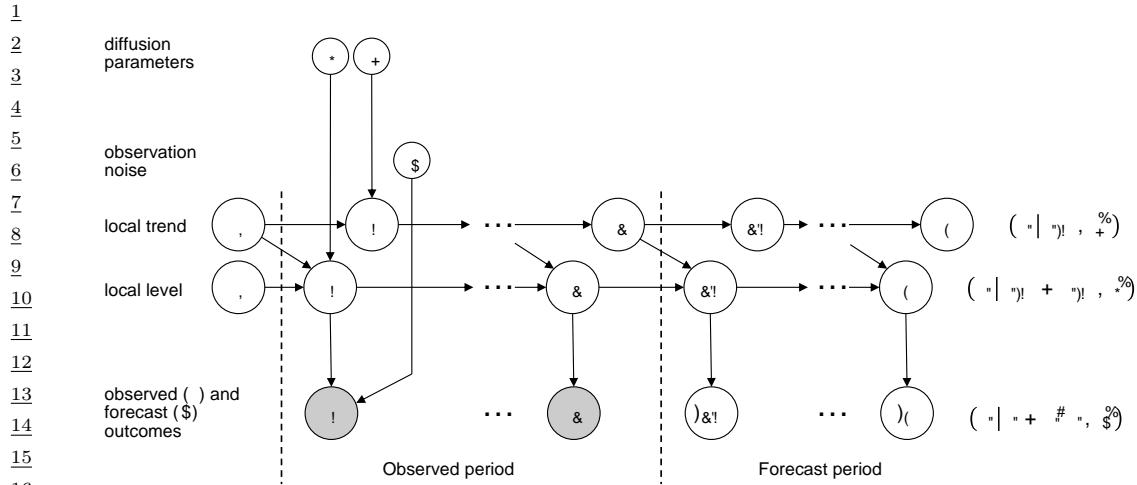


Figure 31.7: A graphical model representation of the local level STS model. Adapted from Figure 2 of [Bro+15]. Used with kind permission of Kay Brodersen.

In Section 8.8.4, we extend this approach to perform online parameter estimation for logistic regression, and in Section 17.6.1, we extend this approach to perform nline parameter estimation for MLPs.

31.2.4 Example: structural time series forecasting

In Section 19.3.1, we discuss how to use LDS models for time series forecasting using a **structural time series** model. There are many kinds of STS model, but the simplest is the **local linear model**, to capture linear trends. (We do not add external covariates, \mathbf{x}_t , since for forecasting into the future, these will be unknown.) This model can be written as follows:

$$\underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{\mathbf{z}_t} = \underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} \mu_{t-1} \\ \delta_{t-1} \end{pmatrix}}_{\mathbf{z}_{t-1}} + \underbrace{\begin{pmatrix} \epsilon_{\mu,t} \\ \epsilon_{\delta,t} \end{pmatrix}}_{\boldsymbol{\epsilon}_t} \quad (31.26)$$

$$y_t = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{\mathbf{C}} \underbrace{\begin{pmatrix} \mu_t \\ \delta_t \end{pmatrix}}_{\mathbf{z}_t} + \epsilon_{y,t} \quad (31.27)$$

See Figure 31.7 for the graphical model. We can use the Kalman filter to compute $p(\mathbf{z}_t | \mathbf{y}_{1:t})$, and then make predictions forwards in time. See Section 19.3.1 for details.

42

31.2.5 Parameter estimation

In the examples in Section 31.2.1, Section 31.2.2 and Section 31.2.3, the transition and observation matrices, \mathbf{A} and \mathbf{C} , were specified by hand, and the only unknown parameters were the noise terms

\mathbf{Q} and \mathbf{R} . This is commonly the case when the hidden state has some well-specified physical meaning. However, there are also applications where the hidden state is just a latent vector we introduce to compress the data, similar to PCA; in this case we also need to estimate \mathbf{A} and \mathbf{C} .

There are many approaches for estimating the parameters of state space models. (In the control theory community, this is known as **systems identification** [Lju87].) In the case of linear dynamical systems, many of the methods are similar to techniques used to fit HMMs, discussed in Section 30.4. For example, we can use EM, SGD, or spectral methods, as we discuss below. (We can also use Bayesian inference, see Section 31.4.2.)

31.2.5.1 Training with fully observed data

If we observe the hidden state sequences, we can fit the model by computing the MLEs for the parameters by solving a multivariate linear regression problem for $\mathbf{z}_{t-1} \rightarrow \mathbf{z}_t$ and for $\mathbf{z}_t \rightarrow \mathbf{y}_t$. That is, we can estimate \mathbf{A} by solving the least squares problem $\mathcal{L}(\mathbf{A}) = \sum_{t=1}^T (\mathbf{z}_t - \mathbf{A}\mathbf{z}_{t-1})^2$, Similarly, we can estimate \mathbf{C} by minimizing $\mathcal{L}(\mathbf{C}) = \sum_{t=1}^T (\mathbf{C}\mathbf{z}_t - \mathbf{y}_t)^2$.

We can estimate the system noise covariance \mathbf{Q} from the residuals in predicting \mathbf{z}_t from \mathbf{z}_{t-1} , and estimate the observation noise covariance \mathbf{R} from the residuals in predicting \mathbf{y}_t from \mathbf{z}_t . However, we can set $\mathbf{Q} = \mathbf{I}$ without loss of generality, since an arbitrary noise covariance can be modeled by appropriately modifying \mathbf{A} . Also, by analogy with factor analysis, we can require \mathbf{R} to be diagonal without loss of generality. Doing this reduces the number of free parameters and improves numerical stability.

31.2.5.2 EM for LG-SSM

If we only observe the output sequence, we can compute ML or MAP estimates of the parameters using EM. The method is conceptually quite similar to the Baum-Welch algorithm for HMMs (Section 30.4.1), except we use Kalman smoothing instead of forwards-backwards in the E step, and use different calculations in the M step. In particular, we need to compute the following expected sufficient statistics:

$$\sum_{t=1}^T \mathbb{E}[\mathbf{z}_t \mathbf{z}_t^\top], \sum_{t=1}^T \mathbb{E}[\mathbf{z}_t \mathbf{z}_{t+1}^\top], \sum_{t=1}^T \mathbb{E}[\mathbf{z}_t \mathbf{y}_t^\top] \quad (31.28)$$

where all expectations are wrt $p(\mathbf{z}_t | \mathbf{y}_{1:T}, \boldsymbol{\theta})$. We can maximize the expected complete data log likelihood by using a weighted least squares method. The details can be found in [GH96a].

Note that computing the expected sufficient statistics in Equation (31.28) in the inner loop of EM takes $O(T)$ time, which can be expensive for long sequences. In [Mar10b], a faster method, known as **ASOS** (approximate second order statistics), is proposed. In this approach, various statistics are precomputed in a single pass over the sequence, and from then on, all iterations take constant time (independent of T).

31.2.5.3 Subspace identification methods

EM does not always give satisfactory results, because it is sensitive to the initial parameter estimates. One way to avoid this is to use a different approach known as a **subspace identification (SSID)** [OM96; Kat05].

¹ To understand this approach, let us initially assume there is no observation noise and no system
² noise. In this case, we have $\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1}$ and $\mathbf{y}_t = \mathbf{C}\mathbf{z}_t$, and hence $\mathbf{y}_t = \mathbf{C}\mathbf{A}^{t-1}\mathbf{z}_1$. Consequently all
³ the observations must be generated from a $\dim(\mathbf{z}_t)$ -dimensional linear manifold or subspace. We can
⁴ identify this subspace using PCA. Once we have an estimate of the \mathbf{z}_t 's, we can fit the model as if it
⁵ were fully observed. We can either use these estimates in their own right, or use them to initialize
⁶ EM. Several papers (e.g., [Smi+00; BK15]) have shown that initializing EM this way gives much
⁷ better results than initializing EM at random, or just using SSID without EM.
⁸

⁹ Although the theory only works for noise-free data, we can try to estimate the system noise
¹⁰ covariance \mathbf{Q} from the residuals in predicting \mathbf{z}_t from \mathbf{z}_{t-1} , and to estimate the observation noise
¹¹ covariance \mathbf{R} from the residuals in predicting \mathbf{y}_t from \mathbf{z}_t . We can either use these estimates in their
¹² own right, or use them to initialize EM. Because this method relies on taking an SVD, it is called a
¹³ **spectral estimation method**. Similar methods can also be used for HMMs (see Section 30.4.3).
¹⁴

¹⁵ 31.2.5.4 Numerical stability

¹⁶ When estimating the dynamics matrix \mathbf{A} , it is very useful to impose a constraint on its eigenvalues.
¹⁷ To see why this is important, consider the case of no system noise. In this case, the hidden state at
¹⁸ time t is given by
¹⁹

$$\mathbf{z}_t = \mathbf{A}^t \mathbf{z}_1 = \mathbf{U} \Lambda^t \mathbf{U}^{-1} \mathbf{z}_1 \quad (31.29)$$

²⁰

²¹ where \mathbf{U} is the matrix of eigenvectors for \mathbf{A} , and $\Lambda = \text{diag}(\lambda_i)$ contains the eigenvalues. If any
²² $\lambda_i > 1$, then for large t , \mathbf{z}_t will blow up in magnitude. Consequently, to ensure stability, it is useful
²³ to require that all the eigenvalues are less than 1 [SBG07]. Of course, if all the eigenvalues are less
²⁴ than 1, then $\mathbb{E}[\mathbf{z}_t] = \mathbf{0}$ for large t , so the state will return to the origin. Fortunately, when we add
²⁵ noise, the state becomes non-zero, so the model does not degenerate.
²⁶

²⁷ 31.3 Non-linear dynamical systems

²⁸ In this section, we consider a **nonlinear dynamical system (NLDS)** with additive Gaussian noise.
²⁹ The corresponding generative model is as follows:
³⁰

$$\mathbf{z}_t = f(\mathbf{z}_{t-1}, \mathbf{u}_t) + \boldsymbol{\epsilon}_t \quad (31.30)$$

³¹

$$\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (31.31)$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t) + \boldsymbol{\eta}_t \quad (31.32)$$

$$\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \quad (31.33)$$

³² Henceforth we will ignore the inputs \mathbf{u}_t for brevity.
³³ Inferring the states of an NLDS model is in general computationally difficult. Fortunately, there
³⁴ are a variety of approximate inference schemes that can be used, such as the extended Kalman filter
³⁵ (Section 8.5.2), the unscented Kalman filter (Section 8.6.2), the particle filtering (Section 13.2), etc.
³⁶

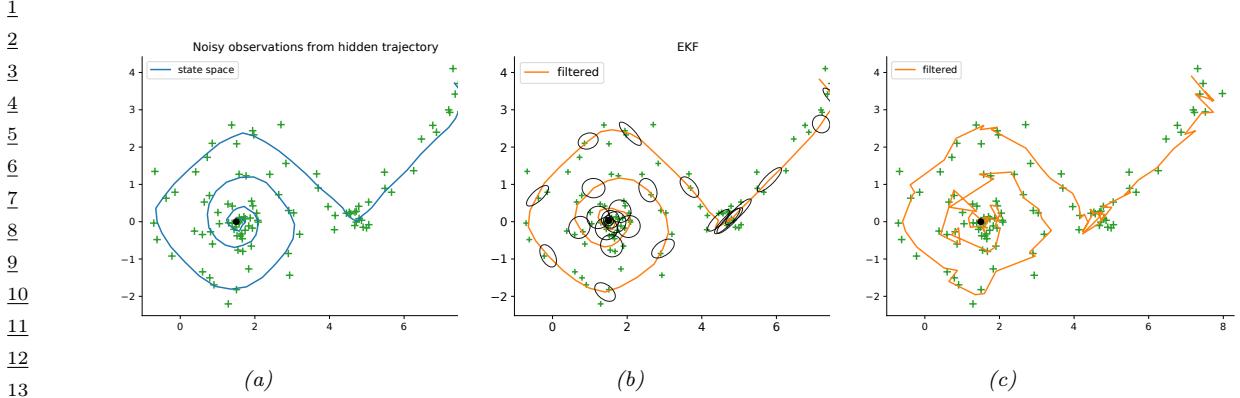


Figure 31.8: Illustration of filtering applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) Extended Kalman filter estimate Generated by `ekf_vs_ukf_demo.py`. (c) Particle filter estimate using 2000 samples. Generated by `bootstrap_filter_demo.py`.

31.3.1 Example: nonlinear 2d tracking problem

Consider the following nonlinear dynamical system in 2d:

$$\mathbf{f}(\mathbf{z}) = (z_1 + \Delta \sin(z_2), z_2 + \Delta \cos(z_1)) \quad (31.34)$$

$$\mathbf{y} = \mathbf{h}(\mathbf{z}) = (z_1, z_2) \quad (31.35)$$

where Δ is the step size. We assume Gaussian noise is added to the state transitions and observations. In Figure 31.8b, we show the results of EKF, and in Figure 31.8c, we show the results of PF (using the dynamics model as a proposal); we see that both methods work quite well in this simple problem.

31.3.2 Example: Simultaneous localization and mapping (SLAM)

Consider a robot moving around a 2d surface. It needs to learn a map of the environment, and keep track of its location (pose) within that map. This problem is known as **simultaneous localization and mapping**, or **SLAM** for short. SLAM is widely used in mobile robotics (see e.g., [SC86; CN01; TBF06] for details). It is also useful in augmented reality, where the task is to recursively estimate the 3d pose of a handheld camera with respect to a set of 2d visual landmarks (this is known as **visual SLAM**). See e.g., [TUI17; SMT18; Cza+20] for details.

Let us assume we can represent the map as the 2d locations of a set of K landmarks, denote them by $\mathbf{l}^1, \dots, \mathbf{l}^K$ (each is a vector in \mathbb{R}^2). (We can use data association to figure out which landmark generated each observation, as discussed in Section 31.3.4.) Let \mathbf{r}_t represent the unknown location of the robot at time t . Let $\mathbf{z}_t = (\mathbf{r}_t, \mathbf{l}_t^{1:K})$ be the combined state space.

The motion model is define as

$$p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = p(\mathbf{r}_t | \mathbf{r}_{t-1}, \mathbf{l}_{t-1}^{1:K}, \mathbf{u}_t) \prod_{k=1}^K p(\mathbf{l}_t^k | \mathbf{l}_{t-1}^k) \quad (31.36)$$

where $p(\mathbf{r}_t | \mathbf{r}_{t-1}, \mathbf{l}_{t-1}^{1:K}, \mathbf{u}_t)$ specifies how the robot moves given the control signal \mathbf{u}_t and the location of the obstacles $\mathbf{l}_{t-1}^{1:K}$. (Note that in this section, we assume that a human is joysticking the robot

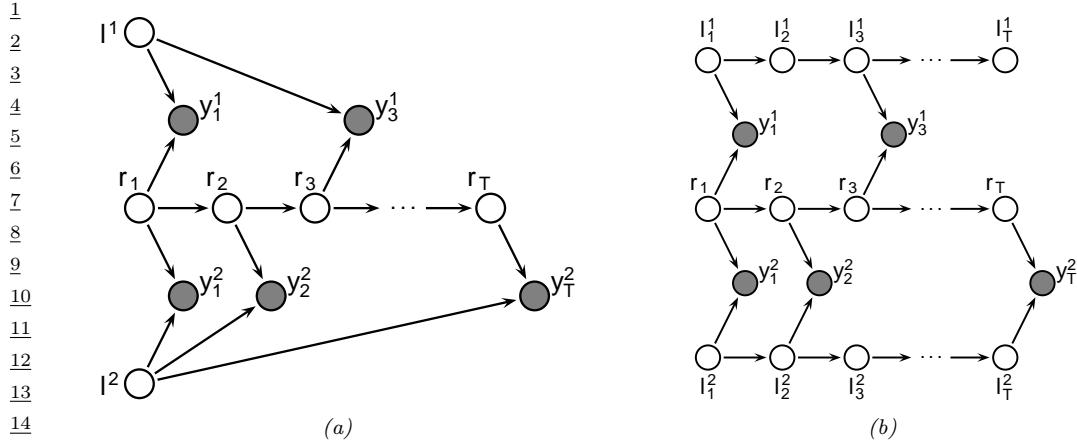


Figure 31.9: Illustration of graphical model underlying SLAM. \mathbf{l}_t^k is the location of landmark k at time t , and \mathbf{y}_t is the observation vector. In the model on the left, the landmarks are static, on the right, their location can change over time. The robot's observations are based on the distance to the nearest landmarks from the current state, denoted $f(\mathbf{r}_t, \mathbf{l}_t^k)$. In this example, the robot gets the following observation trace: $\mathbf{y}_1 = [f(\mathbf{r}_1, \mathbf{l}_1^1), f(\mathbf{r}_1, \mathbf{l}_1^2)]$, $\mathbf{y}_2 = f(\mathbf{r}_2, \mathbf{l}_2^2)$ up to $\mathbf{y}_T = f(\mathbf{r}_T, \mathbf{l}_T^2)$. Thus we see that the number of observations per time step is variable. Adapted from Figure 15.A.3 of [KF09a].

through the environment, so $\mathbf{u}_{1:t}$ is given as input, i.e., we do not address the decision-theoretic issue of choosing where to move.)

If the obstacles (landmarks) are static, we can define $p(\mathbf{l}_t^k | \mathbf{l}_{t-1}^k) = \delta(\mathbf{l}_t^k - \mathbf{l}_{t-1}^k)$, which is equivalent to treating the map as unknown parameter that is shared globally across all time steps. More generally, we can let the landmark locations evolve over time [Mur00].

The observations \mathbf{y}_t measure the distance from \mathbf{r}_t to the set of closest landmarks. Figure 31.9 shows the corresponding graphical model for the case where $K = 2$, and where on the first step it sees landmarks 1 and 2, then just landmark 2, then just landmark 1, etc. We can then perform online inference so that the robot can update its estimate of its own location, and the landmark locations.

If all the CPDs are linear-Gaussian, then we can use a Kalman filter to maintain our belief state about the location of the robot and the location of the landmarks, $p(\mathbf{z}_t | \mathbf{y}_{1:t}, \mathbf{u}_{1:t})$. In the more general case of a nonlinear model, we can use the EKF (Section 8.5.2) or UKF (Section 8.6.2).

Over time, the uncertainty in the robot's location will increase, due to wheel slippage etc., but when the robot returns to a familiar location, its uncertainty will decrease again. This is called **closing the loop**, and is illustrated in Figure 31.10(a), where we see the uncertainty ellipses, representing $\text{Cov}[\mathbf{z}_t | \mathbf{y}_{1:t}, \mathbf{u}_{1:t}]$, grow and then shrink.

In addition to visualizing the uncertainty of the robot's location, we can visualize the uncertainty about the map. To do this, consider the posterior precision matrix, $\mathbf{\Lambda}_t = \mathbf{\Sigma}_t^{-1}$. Zeros in the precision matrix correspond to absent edges in the corresponding undirected Gaussian graphical model (see Section 4.3.2.8). Initially all the landmarks are uncorrelated (by assumption), so the GGM is a disconnected graph, and $\mathbf{\Lambda}_t$ is diagonal. However, as the robot moves about, it will induce correlation between nearby landmarks. Intuitively this is because the robot is estimating its position based on distance to the landmarks, but the landmarks' locations are being estimated based on the robot's

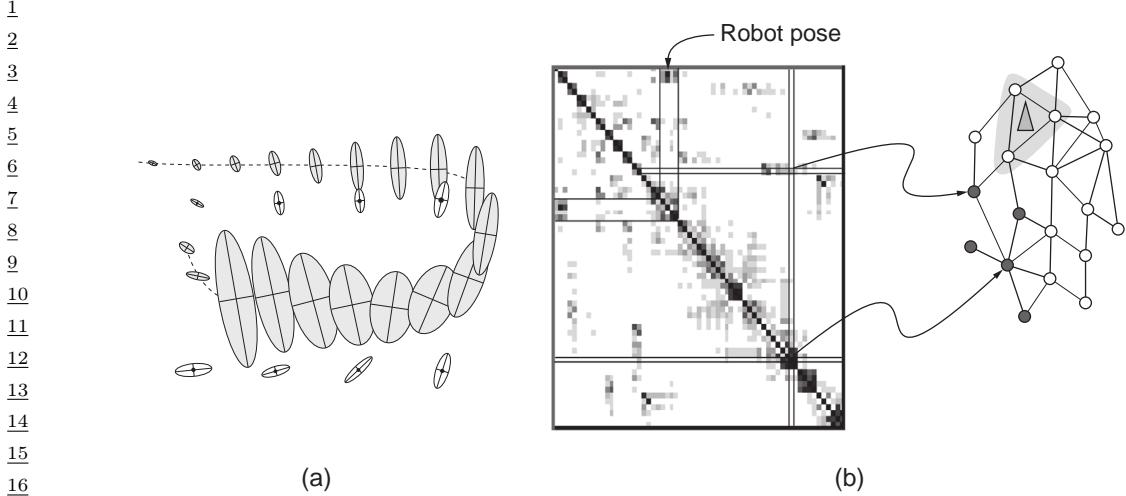


Figure 31.10: Illustration of the SLAM problem. (a) A robot starts at the top left and moves clockwise in a circle back to where it started. We see how the posterior uncertainty about the robot's location increases and then decreases as it returns to a familiar location, closing the loop. If we performed smoothing, this new information would propagate backwards in time to disambiguate the entire trajectory. (b) We show the precision matrix, representing sparse correlations between the landmarks, and between the landmarks and the robot's position (pose). This sparse precision matrix can be visualized as a Gaussian graphical model, as shown on the right. From Figure 15.A.3 of [KF09a]. Used with kind permission of Daphne Koller.

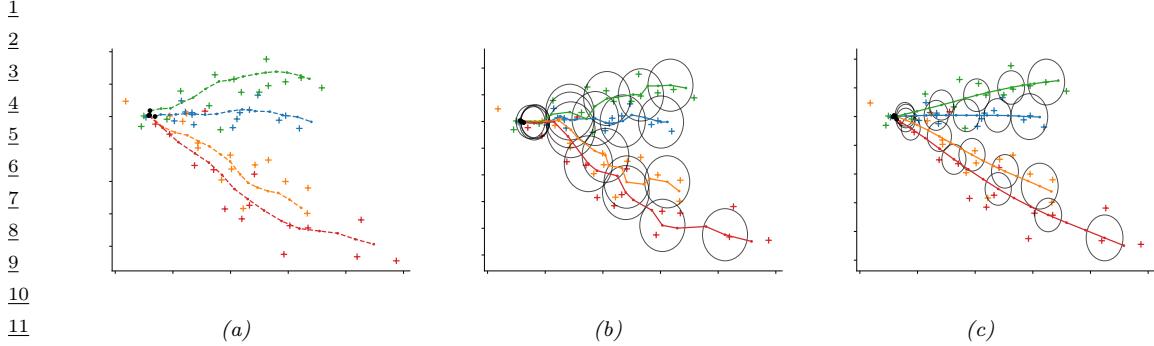
position, so they all become inter-dependent. This can be seen more clearly from the graphical model in Figure 31.9: it is clear that l^1 and l^2 are not d-separated by $\mathbf{y}_{1:t}$, because there is a path between them via the unknown sequence of $r_{1:t}$ nodes. Consequently, the precision matrix becomes denser over time. As a consequence of the precision matrix becoming denser, and each inference step takes $O(K^3)$ time. This prevents the method from being applied to large maps.

One approach to the $O(K^3)$ complexity problem is based on the observation that the correlation pattern moves along with the location of the robot (see Figure 31.10(b)). The remaining correlations become weaker over time. Consequently we can dynamically “prune out” weak edges from the GGM using a technique called the thin junction tree filter [Pas03] (junction trees are explained in Section 9.5).

A second approach is to notice that, conditional on knowing the robot's path, $\mathbf{r}_{1:t}$, the landmark locations are independent, i.e., $p(\mathbf{l}_t | \mathbf{r}_{1:t}, \mathbf{y}_{1:t}) = \prod_{k=1}^K p(l_t^k | \mathbf{r}_{1:t}, \mathbf{y}_{1:t})$. This can be seen by looking at the DGM in Figure 31.9. We can therefore sample the trajectory using particle filtering, and apply Kalman filtering to each landmark independently. See Section 13.5.2 for details.

31.3.3 Example: stochastic volatility models

In finance, it is common to model the log-returns, $y_t = \log(p_t/p_{t-1})$, where p_t is the price of some asset at time t . A common model for this problem, known as a **stochastic volatility model**,



13 *Figure 31.11: Illustration of Kalman filtering and smoothing for tracking multiple moving objects. Generated*
 14 *by kf_parallel_demo.py.*

15

16

17 is the following:

$$18 \quad p(y_t|z_t) = \sigma_y^2 \exp(z_t) \mathcal{N}(0, 1) \quad (31.37)$$

$$20 \quad p(z_t|z_{t-1}) = \mathcal{N}(\mu + \rho(z_{t-1} - \mu), \sigma_z^2) \quad (31.38)$$

22 (For identifiability, we must either pick $\sigma_y = 1$ or $\mu = 0$, with the latter being preferred for
 23 computational reasons [KSC98].) We see that the dynamical model is a first-order autoregressive
 24 process. We typically require that $|\rho| < 1$, to ensure the system is stationary. The observation model
 25 is Gaussian, but can be replaced by a heavy-tailed distribution such as a Student.

26 We can capture longer range temporal correlation by using a higher order auto-regressive process.
 27 To do this, we just expand the state space to contain the past K values. For example, if $K = 2$ we
 28 have

$$29 \quad \begin{pmatrix} z_t - \mu \\ z_{t-1} - \mu \end{pmatrix} = \begin{pmatrix} \rho_1 & \rho_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} z_{t-1} - \mu \\ z_{t-2} - \mu \end{pmatrix} + \begin{pmatrix} q_t \\ 0 \end{pmatrix} \quad (31.39)$$

32 where $q_t \sim \mathcal{N}(0, \sigma_z^2)$. Thus we have

$$34 \quad z_t = \mu + \rho_1(z_{t-1} - \mu) + \rho_2(z_{t-2} - \mu) + q_t \quad (31.40)$$

36 For more details, see [KSC98].

37

38 31.3.4 Example: Multi-target tracking

40 The problem of **multi-target tracking** frequently arises in engineering applications (especially in
 41 aerospace and defence), and can be tackled by inference in various kinds of SSMs. This is a very
 42 large topic (see e.g. [Vo+15] for details), but we summarize the basic ideas below.

43 In the simplest setting, we know there are N objects we want to track, and each one generates its
 44 own uniquely identified observation. If we assume the objects are independent, we can apply Kalman
 45 filtering and smoothing in parallel, as shown in Figure 31.11. (In this example, each object follows a
 46 linear dynamical model with different initial random velocities, as in Section 31.2.2.)

47

More generally, at each step we may observe M measurements e.g., “blips” on a radar screen. We can have $M < N$ due to occlusion or missed detections. We can have $M > N$ due to clutter or false alarms. Or we can have $M = N$. In any case, we need to figure out the **correspondence** between the M detections \mathbf{x}_t^m and the N objects \mathbf{z}_t^i . This is called the problem of **data association**, and it arises in many application domains. See Figure 4.40(d) for an illustration, where we have $M = 2$ observations per time step, but where there is uncertainty about how many objects are actually present.

We can model this problem by augmenting the state space with discrete variables s_t that represent the association matrix between the observations, $\mathbf{y}_{t,1:M}$, and the sources, $\mathbf{z}_{t,1:N}$. As we mentioned in Section 8.8.3.2, inference in such hybrid (discrete-continuous) models is intractable, due to the exponential number of posterior modes. In the sections below, we briefly mention a few approximate inference methods.

31.3.4.1 Nearest neighbor approximation using Hungarian algorithm

A common way to perform approximate inference in this model is to compute an $N \times M$ weight matrix, where W_{im} measures the “compatibility” between object i and measurement m , typically based on how close m is to where the model thinks i is (the so-called **nearest neighbor data association** heuristic).

We can make this into a square matrix by adding dummy background objects, which can explain all the false alarms, and adding dummy observations, which can explain all the missed detections. We can then compute the maximal weight bipartite matching using the **Hungarian algorithm**, which takes $O(\max(N, M)^3)$ time (see e.g., [BDM09]).

Conditional on knowing the assignments of measurements to tracks, we can perform the usual Bayesian state update procedure (e.g., based on Kalman filtering). Note that objects that are assigned to dummy observations do not perform a measurement update, so their state estimate is just based on forwards prediction from the dynamics model.

31.3.4.2 Other approximate inference techniques

The Hungarian algorithm can be slow (since it is cubic in the number of measurements), and can give poor results since it relies on hard assignment. Better performance can be obtained by using loopy belief propagation (Section 9.3). The basic idea is to approximately marginalize out the unknown assignment variables, rather than perform a MAP estimate. This is known as the **SPADA** method (sum-product algorithm for data association) [WL14; Mey+18].

The cost of each iteration of the iterative procedure is $O(NM)$. Furthermore, [WL14] proved this will always converge in a finite number of steps, and [Von13] showed that the corresponding solution will in fact be the global optimum. The SPADA method is more efficient, and more accurate, than earlier heuristic methods, such as **JPDA** (joint probabilistic data association) [BSWT11; Vo+15].

It is also possible to use sequential Monte Carlo methods to solve data association and tracking. See Section 13.2 for a general discussion of SMC, and [RAG04; Wan+17b] for a review of specific techniques for this model family.

Name	Distribution	η_t	ϕ	γ_t
Gaussian	$\mathcal{N}(y \eta_t, \phi)$	Mean	Variance	-
Poisson	$\text{Poi}(y \gamma_t \exp(\eta_t))$	Log Rate	-	Exposure time
Binomial	$\text{Bin}(y \gamma_t, \sigma(\eta_t))$	Logit	-	Num. trials
Gamma	$\text{Ga}(y \exp(\eta_t), \phi)$	Log shape	Rate	-
Stochastic volatility	$y_t = \exp(\eta_t/2)\mathcal{N}(0, 1)$	Log volatility	-	-

8 Table 31.1: Some exponential family likelihoods. List is from [HV21]. In the stochastic volatility model
9 (Section 31.3.3), the dynamics is also constrained to have the form in Equation (31.38).

10

11

12

13 31.3.4.3 Handling an unknown number of targets

14

15 In general, we do not know the true number of targets N , so we have to deal with variable-sized
16 state space. This is an example of an **open world** model [Rus15; LB19], which differs from the
17 standard **closed world assumption** where we know how many objects of interest there are. (See
18 Section 4.5.3.)

19 A common approximate solution to this is to create new objects whenever an observation cannot be
20 “explained” (i.e., generated with high likelihood) by any existing objects, and to prune out old objects
21 that have not been detected in a while (in order to keep the computational cost bounded). Sets
22 whose size and content are both random are called **random finite sets**. An elegant mathematical
23 framework for dealing with such objects is described in [Mah07; Mah13; Vo+15].

24

25 31.4 Other kinds of SSM

26

27 31.4.1 Exponential family SSM

28

29 It is natural to generalize Gaussian SSMs to the setting where the likelihood function is from the
30 exponential family. This is called an **exponential family state space model** (see e.g., [Vid99;
31 Hel17]). We will assume the observation model is as follows:

32

$$\text{33} \quad p(y_t | \mathbf{z}_t, \mathbf{u}_t) = p_t(y_t | \mathbf{C}_t \mathbf{z}_t, \phi, \gamma_t) \quad (31.41)$$

34

35 where $p_t(y_t | \eta_t, \phi, \gamma_t)$ is an exponential family with natural parameters η_t , and where ϕ and γ_t are
36 optional extra parameters that depend on the model family. See Table 31.1 for some examples. (For
37 multivariate outputs, we assume the likelihood factorizes, so $p(\mathbf{y}_t | \mathbf{z}_t, \mathbf{u}_t) = \prod_{d=1}^D p(y_{td} | \mathbf{z}_t, \mathbf{u}_t)$.)

38

39 31.4.1.1 Laplace EM method

40

41 In this section we discuss how to fit an SSM with linear-Gaussian latent dynamics and a GLM
42 likelihood using the **Laplace-EM** algorithm. As we discussed in Section 31.2.5.2, we need to compute
43 the expected sufficient statistics in the E step, and then we can easily maximize the expected complete
44 data log likelihood in the M step. We focus here on the E (inference) step.

45 First we compute the maximum $\mathbf{z}_{1:T}^* = \text{argmax}_{\mathbf{z}_{1:T}} \log p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})$ using some gradient-based
46 optimizer. We then we compute the Hessian at this point, leveraging the fact that this is block

47

1 tridiagonal. In particular, let

$$\underline{3} \quad \underline{4} \quad \mathbf{H}_{t,y} = -\nabla_{\mathbf{z}_t}^2 \log p(\mathbf{y}_t | \mathbf{z}_t), \mathbf{H}_0 = \nabla \log p(\mathbf{z}_0) \quad (31.42)$$

$$\underline{5} \quad \underline{6} \quad \mathbf{H}_{t,11} = -\nabla_{\mathbf{z}_t}^2 \log p(\mathbf{z}_{t+1} | \mathbf{z}_t), \mathbf{H}_{t,22} = -\nabla_{\mathbf{z}_t}^2 \log p(\mathbf{z}_t | \mathbf{z}_{t-1}), \mathbf{H}_{t,12} = -\nabla_{\mathbf{z}_t} \nabla_{\mathbf{z}_{t+1}} \log p(\mathbf{z}_{t+1} | \mathbf{z}_t) \quad (31.43)$$

7 The Gaussian approximation to the posterior has the form $p(\mathbf{z} | \mathbf{y}) \propto \mathcal{N}(\mathbf{z} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = \mathbf{z}_{1:T}^*$,
8 and the precision matrix is given by

$$\underline{10} \quad \underline{11} \quad \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \mathbf{J}_{0,0} & \mathbf{J}_{0,1} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{J}_{1,0} & \mathbf{J}_{1,1} & \mathbf{J}_{1,2} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{J}_{2,1} & \mathbf{J}_{2,2} & \mathbf{J}_{1,2} & \cdots \end{pmatrix} \quad (31.44)$$

14 where $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_t + \mathbf{H}_{t,11}$ for $t = 0$, $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_{t,11} + \mathbf{H}_{t,22}$ for $t = 1 : T - 1$, and
15 $\mathbf{J}_{t,t} = \mathbf{H}_{t,y} + \mathbf{H}_{t,22}$ for $t = T$.

16 Using this, the log joint has the following form, where $\mathbf{J} = \boldsymbol{\Sigma}^{-1}$ and $\mathbf{h} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$:

$$\underline{18} \quad \underline{19} \quad \log p(\mathbf{z}, \mathbf{y}) = \exp\left[-\frac{1}{2}\mathbf{z}^\top \mathbf{J} \mathbf{z} + \mathbf{z}^\top \mathbf{h} - \log Z(\mathbf{J}, \mathbf{h})\right] \quad (31.45)$$

$$\underline{20} \quad \underline{21} \quad = \exp\left[-\frac{1}{2} \sum_{t=1}^T \mathbf{z}_t^\top \mathbf{J}_{t,t} \mathbf{z}_t - \sum_{t=1}^{T-1} \mathbf{z}_{t+1}^\top \mathbf{J}_{t+1,t} \mathbf{z}_t + \sum_{t=1}^T \mathbf{z}_t^\top \mathbf{h}_t - \log Z(\mathbf{J}, \mathbf{h})\right] \quad (31.46)$$

23 From this, we can compute the expected sufficient statistics needed for the E step using $\mathbb{E}[\mathbf{z}_t | \mathbf{y}] = \nabla_{\mathbf{h}_t} \log Z(\mathbf{J}, \mathbf{h})$, $\mathbb{E}[\mathbf{z}_t \mathbf{z}_t^\top | \mathbf{y}] = -2\nabla_{\mathbf{J}_{t,t}} \log Z(\mathbf{J}, \mathbf{h})$, and $\mathbb{E}[\mathbf{z}_{t+1} \mathbf{z}_t^\top | \mathbf{y}] = -\nabla_{\mathbf{J}_{t+1,t}} \log Z(\mathbf{J}, \mathbf{h})$,

26 To derive the MLE for the dynamics model from this, we can use a weighted least squares method
27 [GH96a]. For the observation model, we can maximize the expected complete data log likelihood by
28 sampling from the posterior.

30 31.4.1.2 Example: Poisson likelihood

31 In this section we discuss consider an SSM with linear-Gaussian latent dynamics and a Poisson
32 likelihood. Such models are widely used in neuroscience (see e.g., [Pan+10; Mac+11]). We create a
33 model with 2 continuous latent variables, and we set the dynamics matrix \mathbf{A} to a random rotation
34 matrix. The observation model has the form $p(\mathbf{y}_t | \mathbf{z}_t) = \prod_{d=1}^D \text{Poi}(y_{td} | \exp(\mathbf{w}_d^\top \mathbf{z}_t))$, where \mathbf{w}_d is a
35 random vector, and we use $D = 5$ observations per time step. Some samples from this model are
36 shown in Figure 31.12.

37 We fit this model using the Laplace-EM algorithm from Section 31.4.1.1. We then perform posterior
38 inference. We show the result of these two steps in Figure 31.13, where we compare the parameters
39 \mathbf{A} and the posterior trajectory $\mathbb{E}[\mathbf{z}_t | \mathbf{y}_{1:T}]$ using the true model and the estimated model. We see
40 good agreement.

42 31.4.1.3 Example: demand forecasting

44 In this section, we consider the problem of **demand forecasting**, in which we want to predict how
45 many of an item will be sold each day. Let the latent demand (for some item) be z_t , and the observed
46 sales volume be y_t . Sometimes we get $y_t = 0$ due to out-of-stock situations; if we do not model this
47

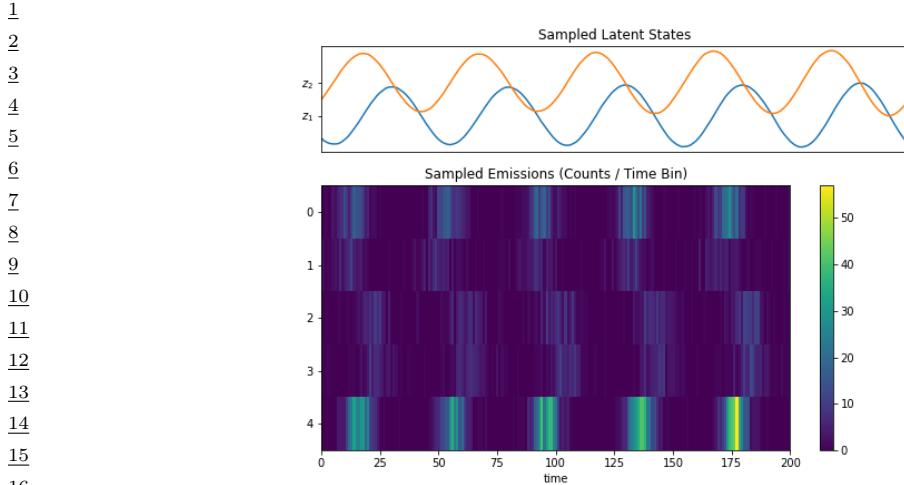


Figure 31.12: Samples from a 2d LDS with 5 Poisson likelihood terms. Generated by [poisson_lds_example.ipynb](#).

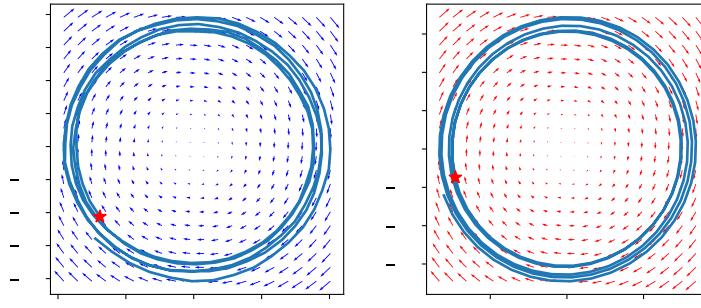


Figure 31.13: Latent state trajectory (blue lines) and dynamics matrix \mathbf{A} (arrows) for (left) true model and (right) estimated model. The star marks the start of the trajectory. Generated by [poisson_lds_example.ipynb](#).

properly, we may incorrectly infer that $z_t = 0$, thus underestimating demand. This may result in not ordering enough stock for the future, further compounding the error.

To avoid this, [SSF16] propose a model which combines SSMs with GLMs, as we discussed in Section 31.4.1. In particular, they consider a likelihood of the form $y_t \sim \text{Poi}(y_t | g(d_t^y))$, where $d_t = z_t + \mathbf{u}_t^\top \mathbf{w}$ is the instantaneous latent demand, $g(d) = e^d$ or $\log(1 + e^d)$ is the transfer function, and $z_t = z_{t-1} + \alpha \mathcal{N}(0, 1)$ is a local random walk term for this latent demand (to capture serial correlation in the data). The covariates \mathbf{u}_t can encode seasonal indicators, such as distance from holidays, but can also encode out-of-stock signals. If the model knows that the item is unavailable, then $y_t = 0$ is “explained away” (Section 4.2.3.2), and so we don’t need to update the latent state z_t .

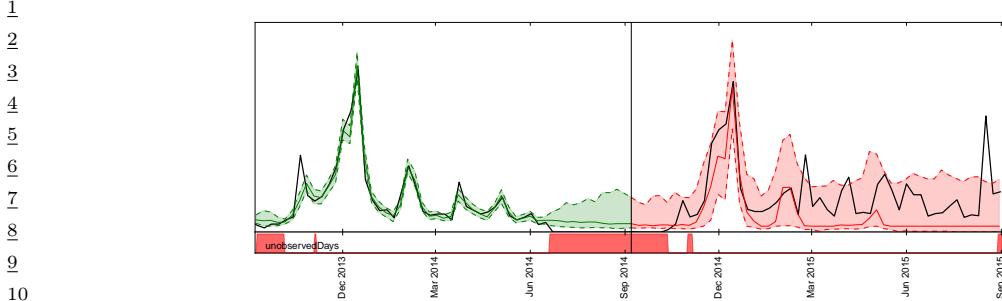


Figure 31.14: Visualization of a probabilistic demand forecast. The black line denotes the actual demand, while the green and red lines denote quantiles of the forecasted demand distribution (10th percentile, median and 90th percentile). Green lines denote the model samples in the training range, while the red lines show the actual probabilistic forecast on data unseen by the model. Note that the demand can be partially unobserved (e.g., due to out-of-stock situations), as indicated by the red bars at the bottom. From Figure 1 of [Bös+17]. Used with kind permission of Tim Januschowski.

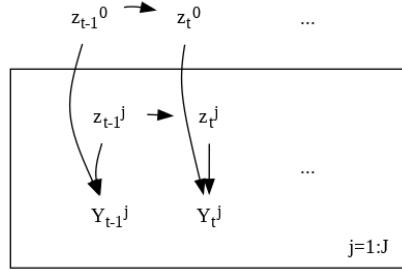


Figure 31.15: Illustration of hierarchical state-space model.

This avoids underestimating the demand, and thus ensures the supply chain is adequately stocked. See Figure 31.14 for an illustration.

The Poisson likelihood is not a good fit for datasets which have many zeros, such as inventory data, due to the out-of-stock problem. One solution is to use a **zero-inflated Poisson (ZIP)** model [Lam92] for the likelihood. This is a mixture model of the form $p(y_t|d_t) = p_0 \mathbb{I}(y_t = 0) + (1 - p_0) \text{Poi}(y_t|e^{d_t})$, where p_0 is the probability of the first mixture component. It is also common to use a (possibly zero-inflated) negative binomial model (Section 2.2.1.4) as the likelihood. This is used in [Cha14; Sal+19b] for the demand forecasting problem.

The disadvantage of these likelihoods is that they are not log-concave for $d_t = 0$, which complicates posterior inference. In particular, the Laplace approximation is a poor choice, since it may find a saddle point. In [SSF16], they tackle this using a log-concave **multi-stage likelihood**.

31.4.1.4 Example: modeling electoral panel data

Suppose we perform a survey for the US presidential elections. Let N_t^j be the number of people who vote at time t in state j , and let Y_t^j be the number of those people who vote Democrat. (We assume

¹ $N_t^j - Y_t^j$ vote Republican.) It is natural to want to model the dependencies in this data both across
² time (longitudinally) and across space (this is an example of **panel data**).
³

⁴ We can do this using a hierarchical SSM, as illustrated in Figure 31.15. The top level Markov
⁵ chain, z_t^0 , models national-level trends, and the state-specific chains, z_t^j , model local “random effects”.
⁶ In practice we would usually also include covariates at the national level, \mathbf{u}_t^0 and state level, \mathbf{u}_t^j .
⁷ Thus the model becomes

$$\underline{y}_t^j \sim \text{Bin}(y_t^j | \pi_t^j, N_t^j) \quad (31.47)$$

$$\underline{\pi}_t^j = \boldsymbol{\sigma} \left[(\underline{z}_t^0)^\top \mathbf{u}_t^0 + (\underline{z}_t^j)^\top \mathbf{u}_t^j \right] \quad (31.48)$$

$$\underline{z}_t^0 = \underline{z}_{t-1}^0 + \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \quad (31.49)$$

$$\underline{z}_t^j = \underline{z}_{t-1}^j + \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}) \quad (31.50)$$

¹⁵ For more details, see [Lin13b].
¹⁶

¹⁸ 31.4.2 Bayesian SSM

¹⁹ SSMs can be quite sensitive to their parameter values, which is a particular concern when they are
²⁰ used for forecasting applications (see Section 31.2.4), or when the latent states or parameters are
²¹ interpreted for scientific purposes (see e.g., [AM+16]). In such cases, it is wise to represent our
²² uncertainty about the parameters by using Bayesian inference.
²³

²⁴ There are various algorithms we can use to perform this task. For linear-Gaussian SSMs, it is
²⁵ possible to use variational Bayes EM [Bea03; BC07] (see Section 10.2.5), or blocked Gibbs sampling
²⁶ [CK94b; CMR05; FS07] (see Section 12.3.7). In the latter case, we alternate between sampling from
²⁷ $p(\mathbf{z}_{1:T} | \mathbf{y}_{1:T}, \boldsymbol{\theta})$ using the forwards-filter backwards-sampling algorithm (Section 8.3.7), and sampling
²⁸ from $p(\boldsymbol{\theta} | \mathbf{z}_{1:T}, \mathbf{y}_{1:T})$, which is easy to do if we use conjugate priors. (Note, however, that $\boldsymbol{\theta}$ and \mathbf{z} are
²⁹ highly correlated, so this method can be slow.)

³⁰ For non-linear and/or non-Gaussian models, things get more complex. One approach is to perform
³¹ joint inference of $p(\boldsymbol{\theta}, \mathbf{z}_{1:T} | \mathbf{y}_{1:T})$ using HMC. However, this can be very slow, since the size of $\mathbf{z}_{1:T}$
³² is often very large.

³³ Another approach is to use particle MCMC methods (Section 13.7). In this approach, we use
³⁴ Metropolis Hastings (e.g., with an adaptive Gaussian proposal) for $p(\boldsymbol{\theta} | \mathbf{y})$, and then we approximate
³⁵ the marginal likelihood $p(\mathbf{y} | \boldsymbol{\theta}) = \int p(\mathbf{z}, \mathbf{y} | \boldsymbol{\theta}) d\mathbf{z}$ using a Monte Carlo method, such as SMC. (This is
³⁶ called a **pseudo marginal** method.) To make this efficient, we should use data-driven proposals, as
³⁷ we discussed in Section 13.4. For example, for exponential family likelihoods with linear Gaussian
³⁸ dynamics, we can use a Laplace approximation to “Gaussianize” each likelihood (see Section 13.4.2).
³⁹ For nonlinear models, we can use the EKF to linearize the model (see Section 13.4.3).

⁴⁰

⁴¹ 31.4.3 GP-SSM

⁴²

⁴³ In Section 31.3 we discussed parametric nonlinear SSMs. We can also represent nonlinear dynamics
⁴⁴ and/or observation models using a non-parametric Gaussian process (Chapter 18). This is known as
⁴⁵ an **GP-SSM**, which stands for “Gaussian process state-space model”. For details, see e.g., [WHF06;
⁴⁶ UFF06; TDR10; FCR14; Ele+17; SS17b].
⁴⁷

31.5 Deep state space models

Traditional state-space models assume linear dynamics and linear observation models, both with additive Gaussian noise. This is obviously very limiting. In this section, we allow the dynamics and/or observation model to be modeled by nonlinear and/or non-Markovian deep neural networks; we call these **deep state space models**, also known as **dynamical variational autoencoders**. [Gir+20]. To be consistent with the literature on VAEs, we denote the observations by \mathbf{x}_t instead of \mathbf{y}_t .

31.5.1 Deep Markov models

If we use a deep neural network for the dynamics or observation models, the result is called a **deep Markov model** [KSS17] or **stochastic RNN** [Fra+16]. (This is not quite the same as a variational RNN, which we explain in Section 31.5.4.)

We can fit a DMM using SVI (Section 10.3.2). The key is to infer the posterior over the latents. From the first-order Markov properties, the exact posterior is given by

$$p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = p(\mathbf{z}_0) \prod_{t=1}^T p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:T}) = p(\mathbf{z}_0) \prod_{t=1}^T p(\mathbf{z}_t|\mathbf{z}_{t-1}, \underline{\mathbf{x}}_{1:t-1}, \mathbf{x}_{t:T}) \quad (31.51)$$

where the cancellation follows since $\mathbf{z}_t \perp \mathbf{x}_{1:t-1} | \mathbf{z}_{t-1}$, as pointed out in [KSS17].

In general, it is intractable to compute $p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$, so we approximate it with an inference network. There are many choices for q . A simple one is a fully factorized model, $q(\mathbf{z}_{1:T}) = \prod_t q(\mathbf{z}_t|\mathbf{x}_{1:t})$. This is illustrated in Figure 31.16a. Since \mathbf{z}_t only depends on past data, $\mathbf{x}_{1:t}$ (which is accumulated in the RNN hidden state \mathbf{h}_t), we can use this inference network at run time for online inference. However, for training the model offline, we can use a more accurate posterior by using

$$q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^T q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^T q(\mathbf{z}_t|\mathbf{z}_{t-1}, \underline{\mathbf{x}}_{1:t-1}, \mathbf{x}_{t:T}) \quad (31.52)$$

Note that the dependence on past observation $\mathbf{x}_{1:t-1}$ is already captured by \mathbf{z}_{t-1} , as in Equation (31.51). The dependencies on future observations, $\mathbf{x}_{t:T}$, can be summarized by a backwards RNN, as shown in Figure 31.16b. Thus

$$q(\mathbf{z}_{1:T}, \mathbf{h}_{1:T}|\mathbf{x}_{1:T}) = q(\mathbf{z}_0) \prod_{t=T}^1 \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t+1}, \mathbf{x}_t)) \prod_{t=1}^T q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{h}_t) \quad (31.53)$$

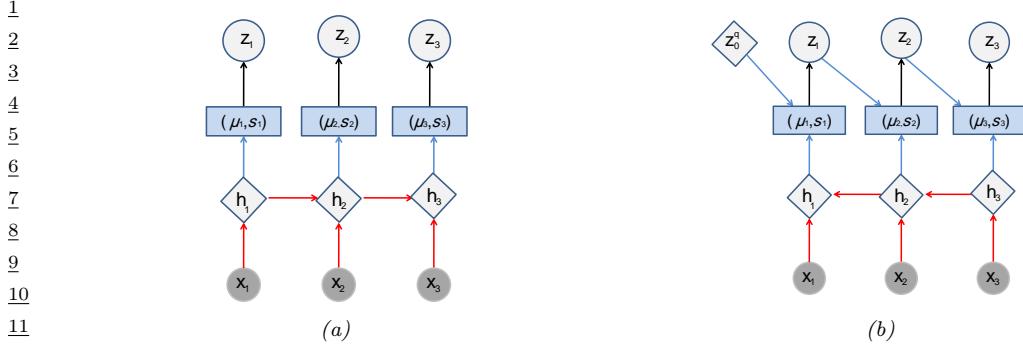


Figure 31.16: Inference networks for deep Markov model. (a) Fully factorized causal posterior $q(\mathbf{z}_{1:T} | \mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t | \mathbf{x}_{1:t})$. The past observations $\mathbf{x}_{1:t}$ are stored in the RNN hidden state \mathbf{h}_t . (b) Markovian posterior $q(\mathbf{z}_{1:T} | \mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{x}_{t:T})$. The future observations $\mathbf{x}_{t:T}$ are stored in the RNN hidden state \mathbf{h}_t .

Given a fully factored $q(\mathbf{z}_{1:T})$, we can compute the ELBO as follows.

$$\log p(\mathbf{x}_{1:T}) = \log \left[\sum_{\mathbf{z}_{1:T}} p(\mathbf{x}_{1:T} | \mathbf{z}_{1:T}) p(\mathbf{z}_{1:T}) \right] \quad (31.54)$$

$$= \log \mathbb{E}_{q(\mathbf{z}_{1:T})} \left[p(\mathbf{x}_{1:T} | \mathbf{z}_{1:T}) \frac{p(\mathbf{z}_{1:T})}{q(\mathbf{z}_{1:T})} \right] \quad (31.55)$$

$$= \log \mathbb{E}_{q(\mathbf{z}_{1:T})} \left[\prod_{t=1}^T \frac{p(\mathbf{x}_t | \mathbf{z}_t) p(\mathbf{z}_t | \mathbf{z}_{t-1})}{q(\mathbf{z}_t)} \right] \quad (31.56)$$

$$\geq \mathbb{E}_{q(\mathbf{z}_{1:T})} \left[\sum_{t=1}^T \log p(\mathbf{x}_t | \mathbf{z}_t) + \log p(\mathbf{z}_t | \mathbf{z}_{t-1}) - \log q(\mathbf{z}_t) \right] \quad (31.57)$$

$$= \sum_{t=1}^T \mathbb{E}_{q(\mathbf{z}_t)} [\log p(\mathbf{x}_t | \mathbf{z}_t)] - \mathbb{E}_{q(\mathbf{z}_{t-1})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t | \mathbf{z}_{t-1}))] \quad (31.58)$$

If we assume that the variational posteriors are jointly Gaussian, we can use the reparameterization trick to use posterior samples to compute stochastic gradients of the ELBO. Furthermore, since we assumed a Gaussian prior, the KL term can be computed analytically.

31.5.2 Recurrent SSM

In a DMM, the observation model $p(\mathbf{x}_t | \mathbf{z}_t)$ is first-order Markov, as is the dynamics model $p(\mathbf{z}_t | \mathbf{z}_{t-1})$. We can modify the model so that it captures long-range dependencies by adding deterministic hidden states as well. We can make the observation model depend on $\mathbf{z}_{1:t}$ instead of just \mathbf{z}_t by using $p(\mathbf{x}_t | \mathbf{h}_t)$, where $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{z}_t)$, so \mathbf{h}_t records all the stochastic choices. This is illustrated in Figure 31.17a. We can also make the dynamical prior depend on $\mathbf{z}_{1:t-1}$ by replacing $p(\mathbf{z}_t | \mathbf{z}_{t-1})$ with $p(\mathbf{z}_t | \mathbf{h}_{t-1})$, as is illustrated in Figure 31.17b. This is known as a **recurrent SSM**.

We can derive an inference network for an RSSM similar to the one we used for DMMs, except now we use a standard forwards RNN to compute $q(\mathbf{z}_t | \mathbf{x}_{1:t-1}, \mathbf{x}_{1:t})$.

47

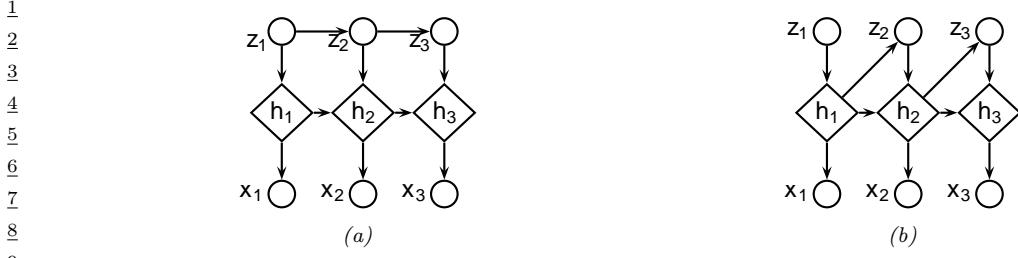


Figure 31.17: Recurrent state space models. (a) Prior is first-order Markov, $p(z_t|z_{t-1})$, but observation model is not Markovian, $p(x_t|h_t) = p(x_t|z_{1:t})$, where h_t summarizes $z_{1:t}$. (b) Prior model is no longer first-order Markov either, $p(z_t|h_{t-1}) = p(z_t|z_{1:t-1})$. Diamonds are deterministic nodes, circles are stochastic.

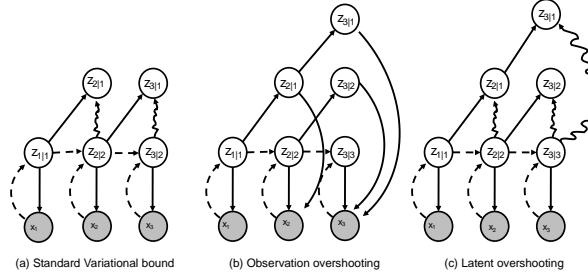


Figure 31.18: Unrolling schemes for SSMs. The labels $s_{i|j}$ is shorthand for $p(z_i|x_{1:j})$. Solid lines denote the generative process, dashed lines the inference process. Arrows pointing at shaded circles represent log-likelihood loss terms. Wavy arrows indicate KL divergence loss terms. (a) Standard 1 step reconstruction of the observations. (b) Observation overshooting tries to predict future observations by unrolling in latent space. (c) Latent overshooting predicts future latent states and penalizes their KL divergence, but does not need to care about future observations. Adapted from Figure 3 of [Haf+19].

31.5.3 Improving multi-step predictions

In Figure 31.18(a), we show the loss terms involved in the ELBO. In particular, the wavy edge $z_{t|t} \rightarrow z_{t|t-1}$ corresponds to $\mathbb{E}_{q(z_{t-1})} [D_{\text{KL}}(q(z_t) \| p(z_t|z_{t-1}))]$, and the solid edge $z_{t|t} \rightarrow z_t$ corresponds to $\mathbb{E}_{q(z_t)} [\log p(x_t|z_t)]$. We see that the dynamics model, $p(z_t|z_{t-1})$, is only ever penalized in terms of how it differs from the one-step-ahead posterior $q(z_t)$, which can hurt the ability of the model to make long-term predictions.

One solution to this is to make multi-step forward predictions using the dynamics model, and use these to reconstruct future observations, and add these errors as extra loss terms. This is called **observation overshooting** [Amo+18], and is illustrated in Figure 31.18(b).

A faster approach, proposed in [Haf+19], is to apply a similar idea but in latent space. More precisely, let us compute the multi-step prediction model, by repeatedly applying the transition model and integrating out the intermediate states to get $p(z_t|z_{t-d})$. We can then compute the ELBO

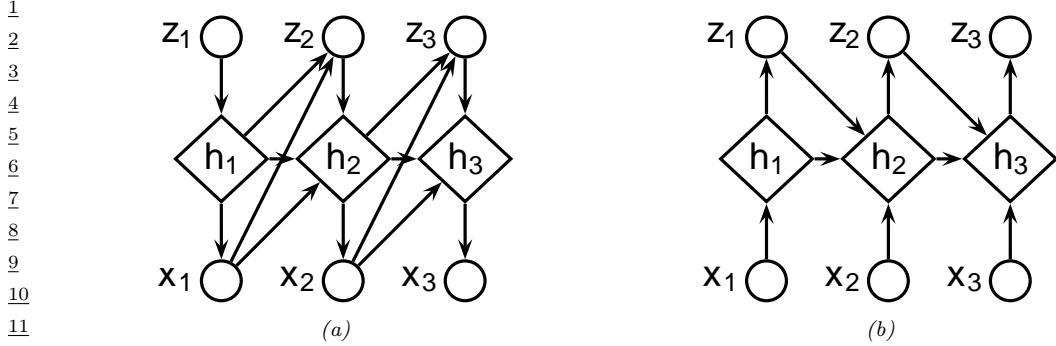


Figure 31.19: Variational RNN. (a) Generative model. (b) Inference model. The diamond-shaped nodes are deterministic.

for this as follows:

$$\log p_d(\mathbf{x}_{1:T}) \triangleq \log \int \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{z}_{t-d}) p(\mathbf{x}_t | \mathbf{z}_t) d\mathbf{z}_{1:T} \quad (31.59)$$

$$\geq \sum_{t=1}^T \mathbb{E}_{q(\mathbf{z}_t)} [\log p(\mathbf{x}_t | \mathbf{z}_t)] - \mathbb{E}_{p(\mathbf{z}_{t-1} | \mathbf{z}_{t-d}) q(\mathbf{z}_{t-d})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t | \mathbf{z}_{t-1}))] \quad (31.60)$$

To train the model so it is good at predicting at different future horizon depths d , we can average the above over all $1 \leq d \leq D$. However, for computational reasons, we can instead just average the KL terms, using weights β_d . This is called **latent overshooting** [Haf+19], and is illustrated in Figure 31.18(c). The new objective becomes

$$\frac{1}{D} \sum_{d=1}^D \log p_d(\mathbf{x}_{1:T}) \geq \sum_{t=1}^T \mathbb{E}_{q(\mathbf{z}_t)} [\log p(\mathbf{x}_t | \mathbf{z}_t)] \quad (31.61)$$

$$- \frac{1}{D} \sum_{d=1}^D \beta_d \mathbb{E}_{p(\mathbf{z}_{t-1} | \mathbf{z}_{t-d}) q(\mathbf{z}_{t-d})} [D_{\text{KL}}(q(\mathbf{z}_t) \| p(\mathbf{z}_t | \mathbf{z}_{t-1}))] \quad (31.62)$$

31.5.4 Variational RNNs

A **variational RNN** (VRNN) [Chu+15] is similar to a recurrent SSM except the hidden states are generated conditional on all past hidden states *and* all past observations, rather than just the past hidden states. This is a more expressive model, but is slower to use for forecasting, since unrolling into the future requires generating observations $\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \dots$ to “feed into” the hidden states, which controls the dynamics. This makes the model less useful for forecasting and model-based RL (see Section 37.4.5.2).

More precisely, the generative model is as follows:

$$p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}, \mathbf{h}_{1:T}) = \prod_{t=1}^T p(\mathbf{z}_t | \mathbf{h}_{t-1}, \mathbf{x}_{t-1}) \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, \mathbf{z}_t)) p(\mathbf{x}_t | \mathbf{h}_t) \quad (31.63)$$

where $p(\mathbf{z}_1|\mathbf{h}_0, \mathbf{x}_0) = p(\mathbf{z}_0)$ and $\mathbf{h}_1 = f(\mathbf{h}_0, \mathbf{x}_0, \mathbf{z}_1) = f(\mathbf{z}_1)$. Thus $\mathbf{h}_t = (\mathbf{z}_{1:t}, \mathbf{x}_{1:t-1})$ is a summary of the past observations and past and current stochastic latent samples. If we marginalize out these deterministic hidden nodes, we see that the dynamical prior on the stochastic latents is $p(\mathbf{z}_t|\mathbf{h}_{t-1}, \mathbf{x}_{t-1}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t-1})$, whereas in a DMM, it is $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, and in an RSSM, it is $p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$. See Figure 31.19a for an illustration.

We can train VRNNs using SVI. In [Chu+15], they use the following inference network:

$$q(\mathbf{z}_{1:T}, \mathbf{h}_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^T \mathbb{I}(\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{z}_{t-1}, \mathbf{x}_t)) q(\mathbf{z}_t|\mathbf{h}_t) \quad (31.64)$$

Thus $\mathbf{h}_t = (\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t})$. Marginalizing out these deterministic nodes, we see that the filtered posterior has the form $q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:t})$. See Figure 31.19b for an illustration. (We can also optionally replace \mathbf{x}_t with the output of a bidirectional RNN to get the smoothed posterior, $q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = \prod_t q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:T})$.)

This approach was used in [DF18] to generate simple videos of moving objects (e.g., a robot pushing a block); they call their method **stochastic video generation** or **SVG**. This was scaled up in [Vil+19], using simpler but larger architectures.

31.5.5 Structured State Space Sequence model (S4)

In this section, we briefly discuss a new sequence-to-sequence model known as the **Structured State Space Sequence model** or **S4** [GGR21a; GGR21b; Gu+20]. Our presentation of S4 is based in part on the excellent tutorial [Rus22].

An S4 model is basically a deep stack of (noiseless) linear SSMs (Section 31.2). In between each layer we add pointwise nonlinearities and a linear mapping. Because SSMs are recurrent first-order models, we can easily generate (sample) from them in $O(L)$ time, where L is the length of the sequence. This is much faster than the $O(L^2)$ required by standard transformers (Section 16.3.4). However, because these SSMs are linear, it turns out that we compute all the hidden representations given known inputs in parallel using convolution; this makes the models fast to train. Finally, since S4 models are derived from an underlying continuous time process, they can easily be applied to observations at different temporal frequencies. Empirically S4 has been found to be much better at modeling **long range dependencies** compared to transformers, which (at the time of writing, namely January 2022) are considered state of the art.

The basic building block, known as a **Linear State Space Layer (LSSL)**, is the following continuous time linear dynamical system that maps an input sequence $\mathbf{u}(t) \in \mathbb{R}$ to an output sequence $\mathbf{y}(t) \in \mathbb{R}^1$ via a sequence of hidden states $\mathbf{z}(t) \in \mathbb{R}^N$:

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}(t) \quad (31.65)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{z}(t) + \mathbf{D}\mathbf{u}(t) \quad (31.66)$$

Henceforth we will omit the skip connection corresponding to the \mathbf{D} term for brevity. We can convert this to a discrete time system using the generalized bilinear transform discussed in Section 31.2.1. If we set $\alpha = \frac{1}{2}$ in Equation (31.13) we get the **bilinear method**, which preserves the stability of the

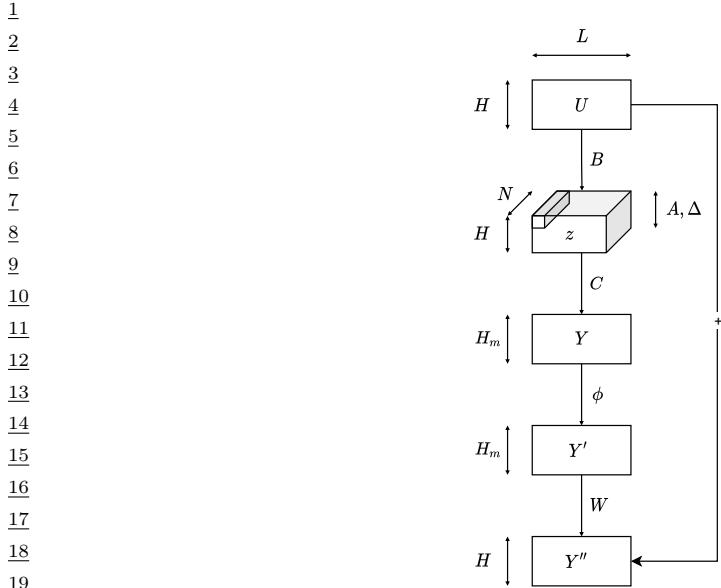


Figure 31.20: Illustration of one S4 block. The input \mathbf{X} has H sequences, each of length L . These get mapped (in parallel for each of the sequences) to the output \mathbf{Y} by a (noiseless) linear SSM with state size N and parameters $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$. The output \mathbf{Y} is mapped pointwise through a nonlinearity ϕ to create \mathbf{Y}' . The channels are then linearly combined by weight matrix \mathbf{W} and added to the input (via a skip connection) to get the final output \mathbf{Y}'' , which is another set of H sequences of length L .

25

26

27 system [ZCC07]. The result is
28

$$29 \quad \mathbf{z}_t = \bar{\mathbf{A}}\mathbf{z}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t \quad (31.67)$$

$$30 \quad \mathbf{y}_t = \bar{\mathbf{C}}\mathbf{z}_t \quad (31.68)$$

$$32 \quad \bar{\mathbf{A}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/2 \cdot \mathbf{A}) \in \mathbb{R}^{N \times N} \quad (31.69)$$

$$33 \quad \bar{\mathbf{B}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B} \in \mathbb{R}^{N \times N}, \bar{\mathbf{C}} = \mathbf{C} \in \mathbb{R}^{N \times 1} \quad (31.70)$$

35 We now discuss what is happening inside the LSSL layer. Let us assume the initial state is $\mathbf{z}_{-1} = \mathbf{0}$.
36 We can unroll the recursion to get

$$38 \quad \mathbf{z}_0 = \bar{\mathbf{B}}\mathbf{u}_0, \mathbf{z}_1 = \bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{B}}\mathbf{u}_1, \mathbf{z}_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_1 + \bar{\mathbf{B}}\mathbf{u}_2, \dots \quad (31.71)$$

$$40 \quad \mathbf{y}_0 = \bar{\mathbf{C}}\bar{\mathbf{B}}\mathbf{u}_0, \mathbf{y}_1 = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{B}}\mathbf{u}_1, \mathbf{y}_2 = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}}\mathbf{u}_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{u}_1 + \bar{\mathbf{C}}\bar{\mathbf{B}}\mathbf{u}_2, \dots \quad (31.72)$$

42 We see that \mathbf{z}_t is computing a weighted sum of all the past inputs, where the weights are controlled
43 by powers of the \mathbf{A} matrix. (See also the discussion of subspace identification techniques in
44 Section 31.2.5.3.) It turns out that we can define \mathbf{A} to have a special structure, known as **HiPPO**
45 (High-order Polynomial Projection Operator) [Gu+20], such that (1) it ensures \mathbf{z}_t embeds “relevant”
46 parts of the past history in a compact manner, (2) enables recursive computation of $\mathbf{z}_{1:L}$ in $O(N)$

47

1 time per step, instead of the naive $O(N^2)$ time for the matrix vector multiply; and (3) only has
2 $O(3N)$ (complex-valued) parameters to learn.
3

4 However, recursive computation still takes time linear in L . At training time, when all inputs to
5 each location are already available, we can further speed thing up by recognizing that the output
6 sequence can be computed in parallel using a convolution:

$$\underline{y}_k = \overline{\mathbf{C}}\overline{\mathbf{A}}^k\overline{\mathbf{B}}u_0 + \overline{\mathbf{C}}\overline{\mathbf{A}}^{k-1}\overline{\mathbf{B}}u_1 + \cdots + \overline{\mathbf{C}}\overline{\mathbf{B}}u_k \quad (31.73)$$

$$\underline{y} = \overline{\mathbf{K}} \odot \underline{u} \quad (31.74)$$

$$\overline{\mathbf{K}} = (\overline{\mathbf{C}}\overline{\mathbf{B}}, \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}, \dots, \overline{\mathbf{C}}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}) \quad (31.75)$$

12 We can compute the convolution kernel $\overline{\mathbf{K}}$ matrix in $O(N + L)$ time and space, using the S4
13 representation, and then use FFT to efficiently compute the output. Unfortunately the details of
14 how to do this are rather complicated, so we refer the reader to [GGR21a].
15

16 Once we have constructed an LSSL layer, we can process a stack of H sequences independently
17 in parallel by replicating the above process with different parameters, for $h = 1 : H$. If we let each
18 of the H \mathbf{C} matrices be of size $\mathbf{N} \times \mathbf{M}$, so they return a vector of channels instead of a scalar at
19 each location, the overall mapping is from $\underline{u}_{1:L} \in \mathbb{R}^{H \times L}$ to $\underline{y}_{1:L} \in \mathbb{R}^{HM \times L}$. We can add a pointwise
20 nonlinearity to the output and then apply a projection matrix $\mathbf{W} \in \mathbb{R}^{MH \times H}$ to linearly combine the
21 channels and map the result back to size $\underline{y}_{1:L} \in \mathbb{R}^{H \times L}$, as shown in Figure 31.20. This overall block
22 can then be repeated a desired number of times. The input to the whole model is an encoder matrix
23 which embeds each token, and the output is a decoder that creates the softmax layer at each location,
24 as in the transformer. We can now learn the \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{D} , and \mathbf{W} matrices (as well as the step size
25 Δ) for each layer using backpropagation, using whatever loss function we want on the output layer.

26272829303132333435363738394041424344454647

32 Graph learning

32.1 Introduction

Graphs are a very common way to represent data. In this chapter we discuss probability models for graphs. In Section 32.2, we assume the graph structure G is known, but we want to “explain” it in terms of a set of meaningful latent features; for this we use various kinds of latent variable models. In Section 32.3, we assume the graph structure G is unknown and needs to be inferred from correlated data, $\mathbf{x}_n \in \mathbb{R}^D$; for this, we will use probabilistic graphical models with unknown topology. (See also Section 16.3.5, where we discuss graph neural networks, for performing supervised learning using graph-structured data.)

32.2 Latent variable models for graphs

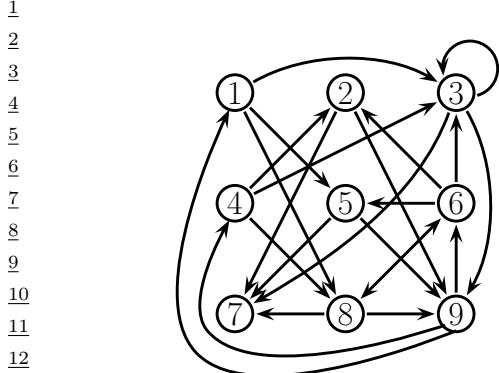
Graphs arise in many application areas, such as modeling social networks, protein-protein interaction networks, or patterns of disease transmission between people or animals. There are usually two primary goals when analysing such data: first, try to discover some “interesting structure” in the graph, such as clusters or communities; second, try to predict which links might occur in the future (e.g., who will make friends with whom). In this section, we focus on the former. More precisely, we will consider a variety of latent variable models for observed graphs.

32.2.1 Stochastic block model

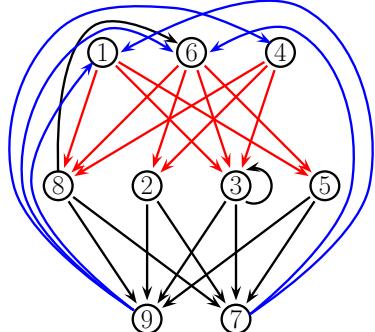
In Figure 32.1(a) we show a directed graph on 9 nodes. There is no apparent structure. However, if we look more deeply, we see it is possible to partition the nodes into three groups or blocks, $B_1 = \{1, 4, 6\}$, $B_2 = \{2, 3, 5, 8\}$, and $B_3 = \{7, 9\}$, such that most of the connections go from nodes in B_1 to B_2 , or from B_2 to B_3 , or from B_3 to B_1 . This is illustrated in Figure 32.1(b).

The problem is easier to understand if we plot the adjacency matrices. Figure 32.2(a) shows the matrix for the graph with the nodes in their original ordering. Figure 32.2(b) shows the matrix for the graph with the nodes in their permuted ordering. It is clear that there is block structure.

We can make a generative model of block structured graphs as follows. First, for every node, sample a latent block $q_i \sim \text{Cat}(\boldsymbol{\pi})$, where π_k is the probability of choosing block k , for $k = 1 : K$. Second, choose the probability of connecting group a to group b , for all pairs of groups; let us denote this probability by $\eta_{a,b}$. This can come from a beta prior. Finally, generate each edge R_{ij} using the

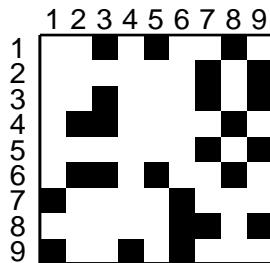


(a)

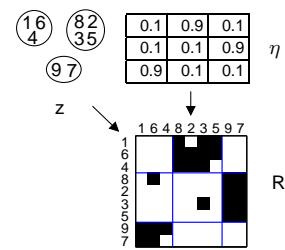


(b)

16 *Figure 32.1: (a) A directed graph. (b) The same graph, with the nodes partitioned into 3 groups, making the*
 17 *block structure more apparent.*



(a)



(b)

30 *Figure 32.2: (a) Adjacency matrix for the graph in Figure 32.1(a). (b) Rows and columns are shown permuted*
 31 *to show the block structure. We also sketch of how the stochastic block model can generate this graph. From*
 32 *Figure 1 of [Kem+06]. Used with kind permission of Charles Kemp.*

33
34 following model:
35

$$36 \quad p(R_{ij} = r | q_i = a, q_j = b, \eta) = \text{Ber}(r | \eta_{a,b}) \quad (32.1)$$

$$37$$

38 This is called the **stochastic block model** [NS01]. Figure 32.4(a) illustrates the model as a DGM,
 39 and Figure 32.2(c) illustrates how this model can be used to cluster the nodes in our example.

40 Note that this is quite different from a conventional clustering problem. For example, we see
 41 that all the nodes in block 3 are grouped together, even though there are no connections between
 42 them. What they share is the property that they “like to” connect to nodes in block 1, and to receive
 43 connections from nodes in block 2. Figure 32.3 illustrates the power of the model for generating many
 44 different kinds of graph structure. For example, some social networks have hierarchical structure,
 45 which can be modeled by clustering people into different social strata, whereas others consist of a set
 46 of cliques.

47

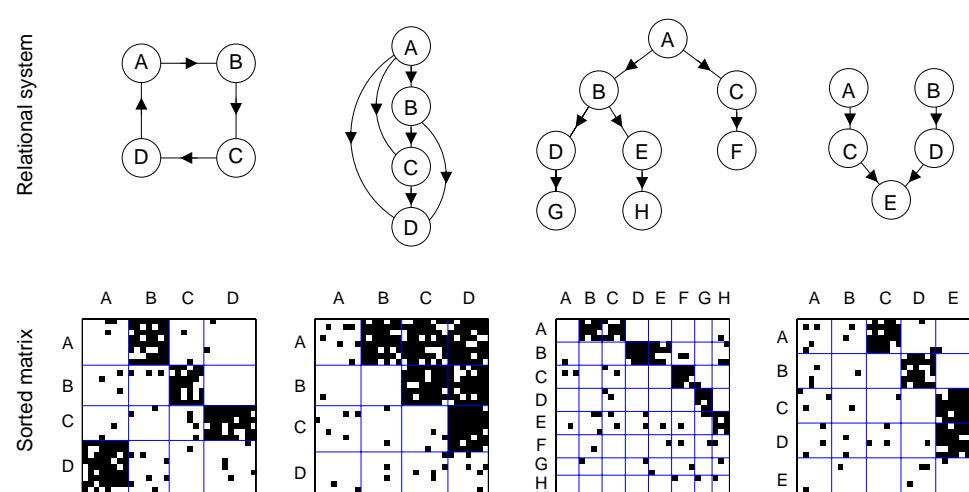


Figure 32.3: Some examples of graphs generated using the stochastic block model with different kinds of connectivity patterns between the blocks. The abstract graph (between blocks) represent a ring, a dominance hierarchy, a common-cause structure, and a common-effect structure. From Figure 4 of [Kem+10]. Used with kind permission of Charles Kemp.

Unlike a standard mixture model, it is not possible to fit this model using exact EM, because all the latent q_i variables become correlated. However, one can use variational EM [Air+08], collapsed Gibbs sampling [Kem+06], etc. We omit the details (which are similar to the LDA case).

In [Kem+06], they lifted the restriction that the number of blocks K be fixed, by replacing the Dirichlet prior on π by a Dirichlet process (see Section 33.2). This is known as the infinite relational model. See Section 32.2.3 for details.

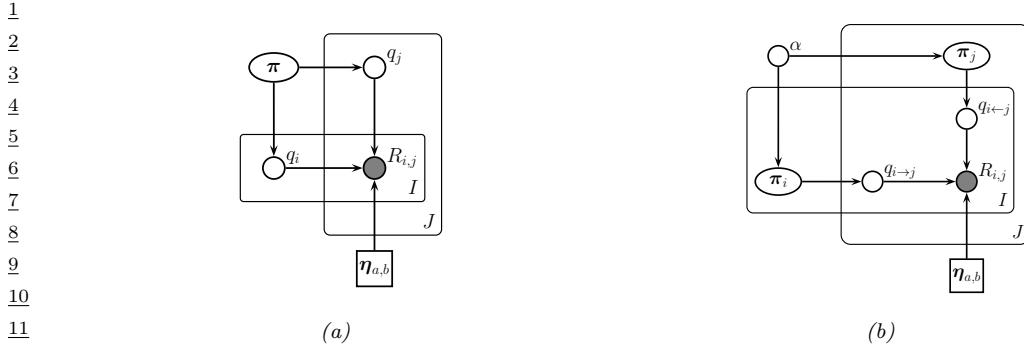
If we have features associated with each node, we can make a discriminative version of this model, for example by defining

$$p(R_{ij} = r | q_i = a, q_j = b, \mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\theta}) = \text{Ber}(r | \mathbf{w}_{a,b}^T f(\mathbf{x}_i, \mathbf{x}_j)) \quad (32.2)$$

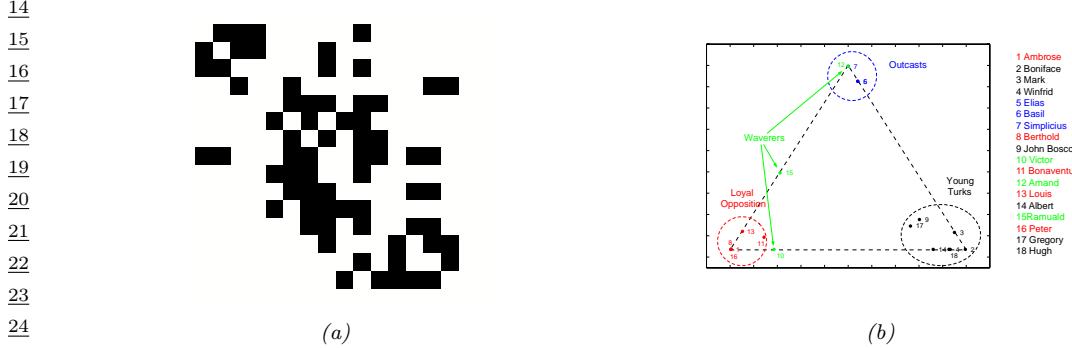
where $f(\mathbf{x}_i, \mathbf{x}_j)$ is some way of combining the feature vectors. For example, we could use concatenation, $[\mathbf{x}_i, \mathbf{x}_j]$, or elementwise product $\mathbf{x}_i \otimes \mathbf{x}_j$ as in supervised LDA. The overall model is like a relational extension of the mixture of experts model.

32.2.2 Mixed membership stochastic block model

In [Air+08], they lifted the restriction that each node only belong to one cluster. That is, they replaced $q_i \in \{1, \dots, K\}$ with $\pi_i \in S_K$. This is known as the **mixed membership stochastic block model**, and is similar in spirit to **fuzzy clustering** or **soft clustering**. Note that π_{ik} is not the same as $p(z_i = k | \mathcal{D})$; the former represents **ontological uncertainty** (to what degree does each object belong to a cluster) whereas the latter represents **epistemological uncertainty** (which cluster does an object belong to). If we want to combine epistemological and ontological uncertainty, we can compute $p(\pi_i | \mathcal{D})$.



13 *Figure 32.4: (a) Stochastic block model. (b) Mixed membership stochastic block model.*



24 *Figure 32.5: (a) Who-likes-whom graph for Sampson's monks. (b) Mixed membership of each monk in one of 25 three groups. From Figures 2-3 of [Air+08]. Used with kind permission of Edo Airoldi.*

30 In more detail, the generative process is as follows. First, each node picks a distribution over 31 blocks, $\pi_i \sim \text{Dir}(\boldsymbol{\alpha})$. Second, choose the probability of connecting group a to group b , for all pairs of 32 groups, $\eta_{a,b} \sim \beta(\alpha, \beta)$. Third, for each edge, sample two discrete variables, one for each direction:

$$34 \quad q_{i \rightarrow j} \sim \text{Cat}(\pi_i), \quad q_{i \leftarrow j} \sim \text{Cat}(\pi_j) \quad (32.3)$$

35 Finally, generate each edge R_{ij} using the following model:
 36

$$37 \quad p(R_{ij} = 1 | q_{i \rightarrow j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \eta_{a,b} \quad (32.4)$$

38 See Figure 32.4(b) for the DGM.

40 Unlike the regular stochastic block model, each node can play a different role, depending on who 41 it is connecting to. As an illustration of this, we will consider a data set that is widely used in 42 the social networks analysis literature. The data concerns who-likes-whom amongst of group of 18 43 monks. It was collected by hand in 1968 by Sampson [Sam68] over a period of months. (These 44 days, in the era of social media such as Facebook, a social network with only 18 people is trivially 45 small, but the methods we are discussing can be made to scale.) Figure 32.5(a) plots the raw data, 46 and Figure 32.5(b) plots $\mathbb{E}[\pi]$ for each monk, where $K = 3$. We see that most of the monk belong 47

to one of the three clusters, known as the “young turks”, the “outcasts” and the “loyal opposition”. However, some individuals, notably monk 15, belong to two clusters; Sampson called these monks the “waverers”. It is interesting to see that the model can recover the same kinds of insights as Sampson derived by hand.

One prevalent problem in social network analysis is missing data. For example, if $R_{ij} = 0$, it may be due to the fact that person i and j have not had an opportunity to interact, or that data is not available for that interaction, as opposed to the fact that these people don’t want to interact. In other words, *absence of evidence is not evidence of absence*. We can model this by modifying the observation model so that with probability ρ , we generate a 0 from the background model, and we only force the model to explain observed 0s with probability $1 - \rho$. In other words, we robustify the observation model to allow for outliers, as follows:

$$p(R_{ij} = r | q_{i \rightarrow j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \rho \delta_0(r) + (1 - \rho) \text{Ber}(r | \eta_{a,b}) \quad (32.5)$$

See [Air+08] for details.

32.2.3 Infinite relational model

It is straightforward to extend the stochastic block model to model **relational data**: we just associate a latent variable $q_i^t \in \{1, \dots, K_t\}$ with each entity i of each type t . We then define the probability of the relation holding between specific entities by looking up the probability of the relation holding between entities of that type. For example, if $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, we have

$$p(R(i, j, k) | q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\eta}) = \text{Ber}(\eta_{a,b,c}) \quad (32.6)$$

We can also have real-valued relations, where each edge has a weight. For example, we can write $p(R(i, j, k) | q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\mu}) = \mathcal{N}(\mu_{a,b,c} + \mu_i + \mu_j + \mu_k, \sigma^2)$, where $\mu_{a,b,c}$ captures the average response for that group of clusters, and μ_i , μ_j and μ_k are offsets for specific entities. (Allowing a different offset for every combination of i , j and k would require too many parameters.) This model was proposed in [BBM07], who fit the model using an alternating minimization procedure.

If we allow the number of clusters K_t for each type of entity to be unbounded, by using a Dirichlet process, the model is called the **infinite relational model** (IRM) [Kem+06], also known as an **infinite hidden relational model** (IHRM) [Xu+06]. We can fit this model with variational Bayes [Xu+06; Xu+07] or collapsed Gibbs sampling [Kem+06]. Rather than go into algorithmic detail, we just sketch some interesting applications.

32.2.3.1 Learning ontologies

An **ontology** refers to an organisation of knowledge. In AI, ontologies are often built by hand (see e.g., [RN10]), but it is interesting to try and learn them from data. In [Kem+06], they show how this can be done using the IRM.

The data comes from the Unified Medical Language System [McC03], which defines a semantic network with 135 concepts (such as “disease or syndrome”, “diagnostic procedure”, “animal”), and 49 binary predicates (such as “affects”, “prevents”). We can represent this as a ternary relation $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, where T^1 is the set of concepts and T^2 is the set of binary predicates. The result is a 3d cube. We can then apply the IRM to partition the cube into regions of roughly

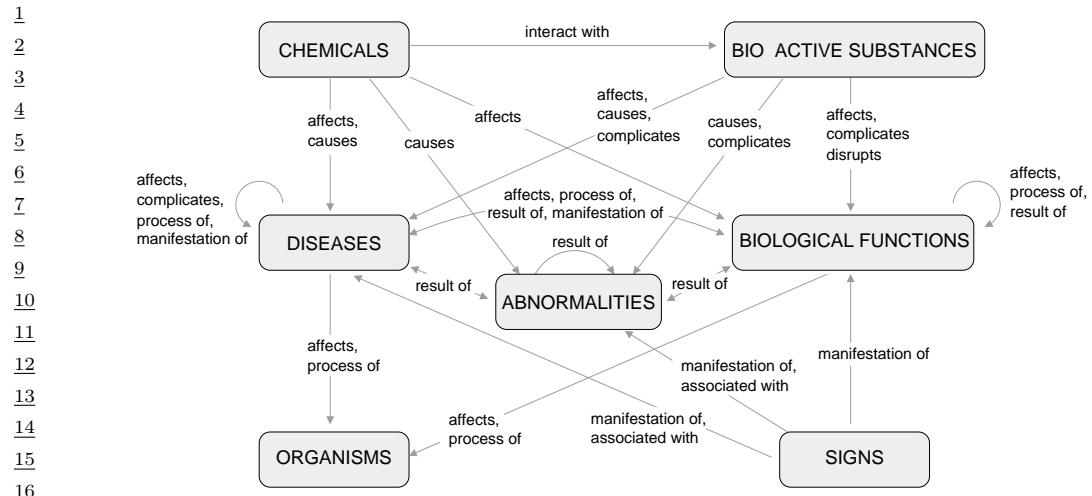


Figure 32.6: Illustration of an ontology learned by IRM applied to the Unified Medical Language System. The boxes represent 7 of the 14 concept clusters. Predicates that belong to the same cluster are grouped together, and associated with edges to which they pertain. All links with weight above 0.8 have been included. From Figure 9 of [Kem+10]. Used with kind permission of Charles Kemp.

homogeneous response. The system found 14 concept clusters and 21 predicate clusters. Some of these are shown in Figure 32.6. The system learns, for example, that biological functions affect organisms (since $\eta_{a,b,c} \approx 1$ where a represents the biological function cluster, b represents the organism cluster, and c represents the affects cluster).

32.2.3.2 Clustering based on relations and features

We can also use IRM to cluster objects based on their relations and their features. For example, consider a political dataset (from 1965) consisting of 14 countries, 54 binary predicates representing interaction types between countries (e.g., “sends tourists to”, “economic aid”), and 90 features (e.g., “communist”, “monarchy”). To create a binary dataset, real-valued features were thresholded at their mean, and categorical variables were dummy-encoded. The data has 3 types: T^1 represents countries, T^2 represents interactions, and T^3 represents features. We have two relations: $R^1 : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, and $R^2 : T^1 \times T^3 \rightarrow \{0, 1\}$. (This problem therefore combines aspects of both the biclustering model and the ontology discovery model.) When given multiple relations, the IRM treats them as conditionally independent. In this case, we have

$$p(\mathbf{R}^1, \mathbf{R}^2 | \mathbf{q}^1, \mathbf{q}^2, \mathbf{q}^3, \boldsymbol{\theta}) = p(\mathbf{R}^1 | \mathbf{q}^1, \mathbf{q}^2, \boldsymbol{\theta}) p(\mathbf{R}^2 | \mathbf{q}^1, \mathbf{q}^3, \boldsymbol{\theta}) \quad (32.7)$$

The results are shown in Figure 32.7. The IRM divides the 90 features into 5 clusters, the first of which contains “noncommunist”, which captures one of the most important aspects of this Cold-War era dataset. It also clusters the 14 countries into 5 clusters, reflecting natural geo-political groupings (e.g., US and UK, or the Communist Bloc), and the 54 predicates into 18 clusters, reflecting similar relationships (e.g., “negative behavior” and “accusations”).

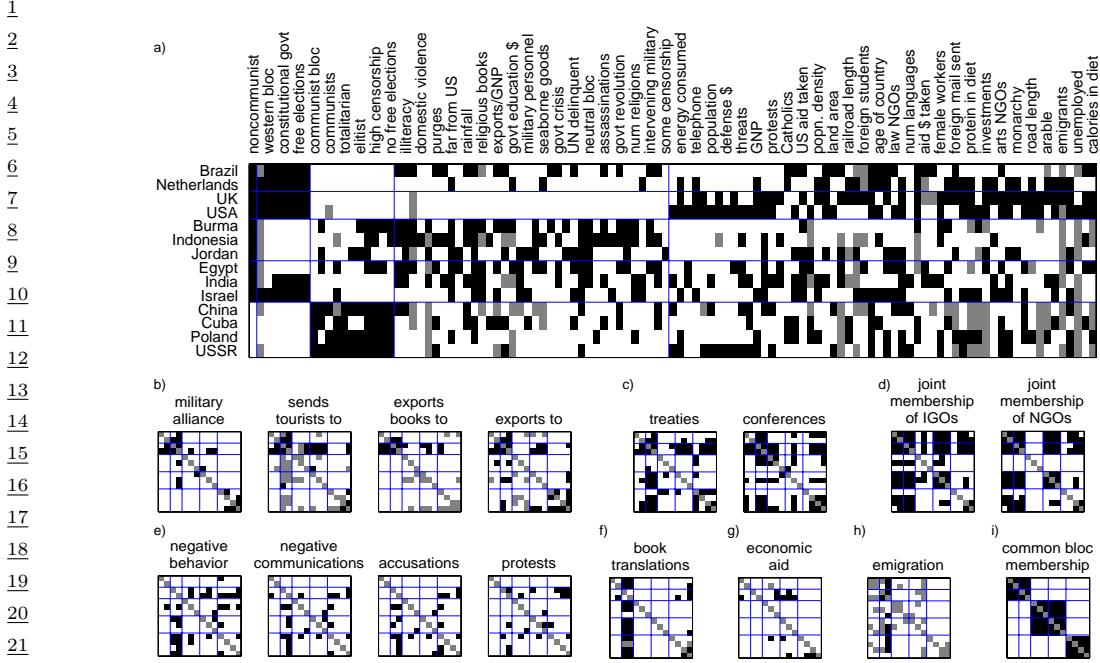


Figure 32.7: Illustration of IRM applied to some political data containing features and pairwise interactions. Top row (a): the partition of the countries into 5 clusters and the features into 5 clusters. Every second column is labelled with the name of the corresponding feature. Small squares at bottom (b-i): these are 8 of the 18 clusters of interaction types. From Figure 6 of [Kem+06]. Used with kind permission of Charles Kemp.

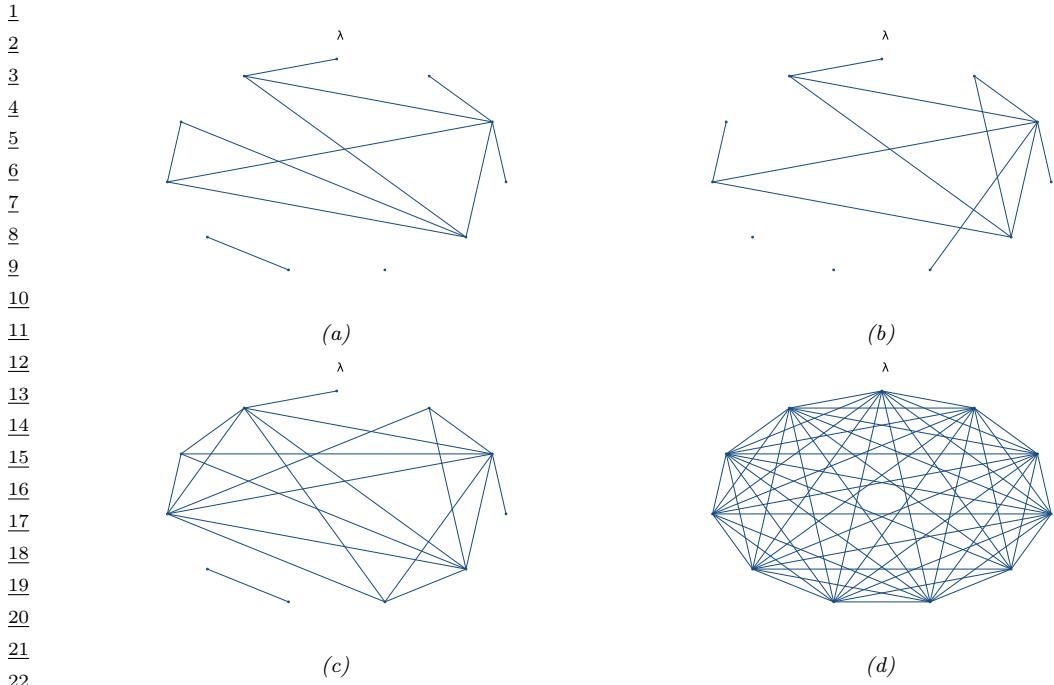
32.3 Graphical model structure learning

In this section, we discuss how to learn the structure of a probabilistic graphical model given sample observations of some or all of its nodes. That is, the input is an $N \times D$ data matrix, and the output is a graph G (directed or undirected) with V nodes. (Usually $V = D$, but we also consider the case where we learn extra latent nodes that are not present in the input.)

32.3.1 Applications

There are three main reasons to perform structure learning for PGMs: understanding, prediction, and causal inference (which involves both understanding and prediction), as we summarize below.

Learning sparse PGMs can be useful for gaining an understanding of multiple interacting variables. For example, consider a problem that arises in systems biology: we measure the phosphorylation status of some proteins in a cell [Sac+05] and want to infer how they interact. Figure 32.8 gives an example of a graph structure that was learned from this data, using a method called graphical lasso [FHT08; MH12], which is explained in the supplementary material. As another example, [Smi+06] showed that one can recover the neural “wiring diagram” of a certain kind of bird from multivariate time-series EEG data. The recovered structure closely matched the known functional connectivity of



23 *Figure 32.8: A sparse undirected Gaussian graphical model learned using graphical lasso applied to some flow
24 cytometry data (from [Sac+05]), which measures the phosphorylation status of 11 proteins. The sparsity level
25 is controlled by λ . (a) $\lambda = 36$. (b) $\lambda = 27$. (c) $\lambda = 7$. (d) $\lambda = 0$. Adapted from Figure 17.5 of [HTF09].
26 Generated by [ggm_lasso_demo.py](#).*

27

28

29

30

31 this part of the bird brain.

32 In some cases, we are not interested in interpreting the graph structure, we just want to use it to
33 make predictions. One example of this is in financial portfolio management, where accurate models of
34 the covariance between large numbers of different stocks is important. [CW07] show that by learning
35 a sparse graph, and then using this as the basis of a trading strategy, it is possible to outperform (i.e.,
36 make more money than) methods that do not exploit sparse graphs. Another example is predicting
37 traffic jams on the freeway. [Hor+05] describe a deployed system called JamBayes for predicting
38 traffic flow in the Seattle area, using a directed graphical model whose structure was learned from
39 data.

40 Structure learning is also an important pre-requisite for causal inference. In particular, to predict
41 the effects of interventions on a system, or to perform counterfactual reasoning, we need to know the
42 structural causal model (SCM), as we discuss in Section 4.6. An SCM is a kind of directed graphical
43 model where the relationships between nodes are deterministic (functional), except for stochastic
44 root (exogeneous) variables. Consequently one can use techniques for learning DAG structures as a
45 way to learn SCMs, if we make some assumptions about (lack of) confounders. This is called **causal**
46 **discovery**.

47

```

1
2
3
4
5
6
7
8
9
10
11
12

```

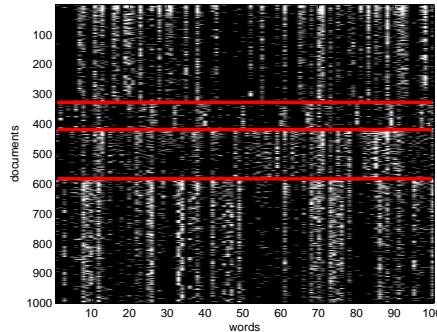


Figure 32.9: Subset of size 16242×100 of the 20-newsgroups data. We only show 1000 rows, for clarity. Each row is a document (represented as a bag-of-words bit vector), each column is a word. The red lines separate the 4 classes, which are (in descending order) comp, rec, sci, talk (these are the titles of USENET groups). We can see that there are subsets of words whose presence or absence is indicative of the class. The data is available from <http://cs.nyu.edu/~roweis/data.html>. Generated by `newsgroupsVisualize.py`.

```

18
19
20
21
22
23
24
25
26

```

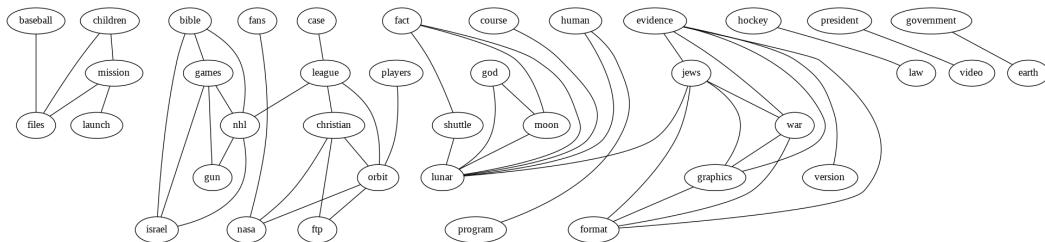


Figure 32.10: Part of a relevance network constructed from the 20 newsgroup data. data shown in Figure 32.9. We show edges whose mutual information is greater than or equal to 20% of the maximum pairwise MI. For clarity, the graph has been cropped, so we only show a subset of the nodes and edges. Generated by `relevance_network_newsgroup_demo.py`.

```

31
32

```

32.3.2 Relevance networks

If we are just interested in learning a graph structure that captures some properties of the data, rather than learning a full generative model, a simple approach is to compute a **relevance network**, in which we add an $i - j$ edge if the pairwise mutual information $\mathbb{I}(X_i; X_j)$ is above some threshold. In the Gaussian case, $\mathbb{I}(X_i; X_j) = -\frac{1}{2} \log(1 - \rho_{ij}^2)$, where ρ_{ij} is the correlation coefficient, and the resulting graph is called a **covariance graph** (Section 4.4.4.1). However, we can also apply it to discrete random variables.

Relevance networks are quite popular in systems biology [Mar+06], where they are used to visualize the interaction between genes. But they can also be applied to other kinds of datasets. For example, Figure 32.10 visualizes the MI between words in the 20 newsgroup dataset shown in Figure 32.9. The results seem intuitively reasonable.

However, relevance networks suffer from a major problem: the graphs are usually very dense, since most variables are dependent on most other variables, even after thresholding the MIs. For example,

1
2 suppose X_1 directly influences X_2 which directly influences X_3 (e.g., these form components of a
3 signalling cascade, $X_1 - X_2 - X_3$). Then X_1 has non-zero MI with X_3 (and vice versa), so there
4 will be a $1 - 3$ edge as well as the $1 - 2$ and $2 - 3$ edges; thus the graph may be fully connected,
5 depending on the threshold.

6 A solution to this is to learn a PGM, which represents conditional *independence*, rather than
7 *dependence*. In the chain example, there will not be a $1 - 3$ edge, since $X_1 \perp X_3 | X_2$. Consequently
8 graphical models are usually much sparser than relevance networks.

9

10 32.3.3 Learning sparse PGMs

11The details on algorithms for learning sparse PGM structure can be found in the supplementary
12 material, due to space constraints.
13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

33 Non-parametric Bayesian models

This chapter is written by Vinayak Rao.

33.1 Introduction

A **stochastic process** is a probability distribution over a potentially infinite set of random variables. A simple example is a **Markov process** (Section 2.8), in which the variables are time indexed. This defines the distribution $p(x_1, \dots, x_T)$ for any T , where $x_t \in \mathcal{X}$, using the form $p(\mathbf{x}) = p(x_1) \prod_{t=2}^T p(x_t|x_{t-1})$, where $\mathbf{x} = (x_1, \dots, x_T)$. Another example is a **Gaussian process**, specified by a mean function μ and a positive definite kernel function \mathcal{K} . This defines a probability distribution over an unknown function $f : \mathcal{X} \rightarrow \mathbb{R}$, with the joint distribution $p(f(\mathbf{x})) = \mathcal{N}(f(\mathbf{x})|\boldsymbol{\mu}(\mathbf{x}), \mathbf{K}(\mathbf{x}))$, where $f(\mathbf{x}) = [f(x_1), \dots, f(X_T)]$ evaluates this function at each input, $\boldsymbol{\mu}(\mathbf{x}) = [\mu(x_1), \dots, \mu(X_T)]$ is the mean, and $\mathbf{K}(\mathbf{x}, \mathbf{x}) = [\mathcal{K}(x_i, x_j)]$ is the Gram matrix. (See Chapter 18 for details.)

In this chapter, we discuss other kinds of stochastic processes commonly used in a subfield of Bayesian statistics called **Bayesian nonparametrics**. The defining characteristic of a parametric model is that the objects being modeled, whether regression or classification functions, probability densities, or something more modern like graphs or shapes, are indexed by a finite-dimensional parameter vector. For instance, linear regression restricts itself to linear functions, and for scalar inputs, indexes these functions with two parameters, a slope and an intercept. In a parametric Bayesian model, a prior probability distribution on these parameters is used to define a prior distribution on the objects of interest. Bayesian nonparametric models directly place prior distributions on objects of interest, typically via some stochastic process, realizations of which have ‘infinite complexity’. For instance, unlike a straight line, a function drawn from a Gaussian process typically cannot be summarized by a finite number of parameters. This allows the statistical complexity of inferences and predictions to grow with the size of the training datasets, avoiding underfitting. By taking a Bayesian approach, the danger of overfitting is minimized: one maintains a full posterior distribution over the infinite parameters, rather than trying to fit them using a finite dataset. Despite involving infinite-parameter objects, practitioners are often only interested in inferences on a finite training dataset and predictions on a finite test dataset. This often allows these models to be surprisingly tractable. Nonparametric Bayesian models thus present an elegant synthesis of probabilistic, statistical and computational ideas.

¹
² **33.2 Dirichlet process**

³ Recall from Chapter 18 that a Gaussian process is a nonparametric probability distribution over
⁴ functions f . We saw how in Bayesian settings, this can be used as a nonparametric prior for
⁵ supervised learning tasks like regression and classification. By contrast, a **Dirichlet process** (DP)
⁶ is a nonparametric probability distribution over probability distributions, and is useful as a flexible
⁷ prior for unsupervised learning tasks like clustering and density modeling [Fer73]. We give more
⁸ details below.
⁹

¹⁰
¹¹ **33.2.1 Definition**

¹² Let G be a probability distribution or a probability measure (we will use the latter terminology in
¹³ this chapter) on some space Θ . Recall that a probability measure is a function that assigns values
¹⁴ to subsets $T \subseteq \Theta$ satisfying the usual axioms of probability: $0 \leq G(T) \leq 1$, $G(\Theta) = \int_{\Theta} G(d\theta) = 1$,
¹⁵ and for disjoint subsets T_1, \dots, T_K of Θ , $G(T_1 \cup \dots \cup T_K) = \sum_{k=1}^K G(T_k)$. Bayesian unsupervised
¹⁶ learning now seeks to place a prior on the probability measure G .
¹⁷

¹⁸ We have already seen examples of *parametric* priors over probability measures. As a simple example,
¹⁹ consider a Gaussian distribution $\mathcal{N}(\theta|\mu, \sigma^2)$: this is a probability measure on Θ , and by placing
²⁰ priors on the parameters μ and σ^2 , we have a **parametric prior** on probability measures. Mixture
²¹ models form more flexible priors, allowing multimodality and asymmetry, and are parametrized
²² by the probabilities of the mixture components, as well as their parameters. DPs directly define a
²³ probability on probability measures G .

²⁴ A Dirichlet Process is specified by a positive real number α , called the **concentration parameter**,
²⁵ and a probability measure H , called the **base measure**. We write a random measure drawn from
²⁶ a DP as $G \sim \text{DP}(\alpha, H)$. H is typically a standard probability measure on Θ , and forms the mean
²⁷ of the Dirichlet process. That is, if $G \sim \text{DP}(\alpha, H)$, then for any subset T of Θ , $\mathbb{E}[G(T)] = H(T)$.
²⁸ The parameter α measures how concentrated the Dirichlet process is around H , with $\mathbb{V}[G(T)] =$
²⁹ $\frac{H(T)(1-H(T))}{1+\alpha}$. If Θ is \mathbb{R}^2 , then setting H to the bivariate normal $\mathcal{N}(0, I_2)$ and α to a large value
³⁰ implies a prior belief that G sampled from $\text{DP}(\alpha, H)$ is close to the normal, whereas a small α
³¹ represents a relatively uninformative prior.

³² We now define the Dirichlet process more precisely. Let (T_1, \dots, T_K) be a finite partition of
³³ Θ , that is, (T_1, \dots, T_K) are disjoint sets whose union is Θ . For a probability measure G , let
³⁴ $(G(T_1), \dots, G(T_K))$ be the vector of probabilities of the elements of this partition. Then $\text{DP}(\alpha, H)$ is
³⁵ a prior over probability measures G satisfying the following requirement: for any finite partition, the
³⁶ associated vector of probabilities has the following joint Dirichlet distribution:

$$\text{(33.1)} \quad (G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_1), \dots, \alpha H(T_K)).$$

³⁷ Just like the Gaussian process, the DP is defined implicitly through a set of finite-dimensional
³⁸ distributions, in this case through the distribution of G projected onto any finite partition. The
³⁹ finite-dimensional distributions are **consistent** in the following sense: if T_{11} and T_{12} form a partition
⁴⁰ of T_1 , then one can sample $G(T_1)$ in two ways: directly, by sampling
⁴¹

$$\text{(33.2)} \quad (G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_1), \dots, \alpha H(T_K))$$

⁴² or, indirectly, by sampling

$$\text{(33.3)} \quad (G(T_{11}), G(T_{12}), \dots, G(T_K)) \sim \text{Dir}(\alpha H(T_{11}), \alpha H(T_{12}), \dots, \alpha H(T_K))$$

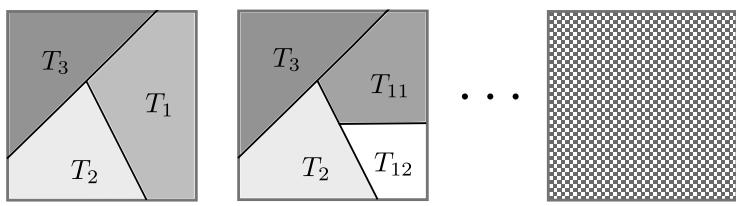


Figure 33.1: Partitions of the unit square. (left) One possible partition into $K = 3$ regions, and (center) A refined partition into $K = 4$ regions. In both figures, the shading of cell T_k is proportional $G(T_k)$, resulting from the same realization of a Dirichlet process. (right) An ‘infinite partition’ of the unit square. The Dirichlet process can informally be viewed as an infinite-dimensional Dirichlet distribution defined on this.

and then setting $G(T_1) = G(T_{11}) + G(T_{12})$. From the properties of the Dirichlet distribution, $G(T_1)$ sampled either way follows the same distribution. This consistency property implies, via Kolmogorov’s extension theorem [Kal06], that underlying all finite-dimensional probability vectors for different partitions is a single infinite-dimensional vector that we could informally write as

$$G(d\theta_1), \dots, G(\theta_\infty) \sim \text{Dir}(\alpha H(d\theta_1), \dots, \alpha H(d\theta_\infty)). \quad (33.4)$$

Very roughly, this ‘infinite-dimensional Dirichlet distribution’ is the Dirichlet Process. Figure 33.1 sketches this out.

Why is the Dirichlet process, defined in this indirect fashion, useful to practitioners? The answer has to do with conjugacy properties that it inherits from the Dirichlet distribution. One of the simplest unsupervised learning problems seeks to learn an unknown probability distribution G from i.i.d. samples $\{\bar{\theta}_1, \dots, \bar{\theta}_N\}$ drawn from it. Consider placing a DP prior on the unknown G . Then given the data, one is interested in the posterior distribution over G , representing the updated probability distribution over G . For a partition (T_1, \dots, T_K) of Θ , an observation falls into the cell z following a multinoulli distribution:

$$z \sim \text{Cat}(G(T_1), \dots, G(T_K)). \quad (33.5)$$

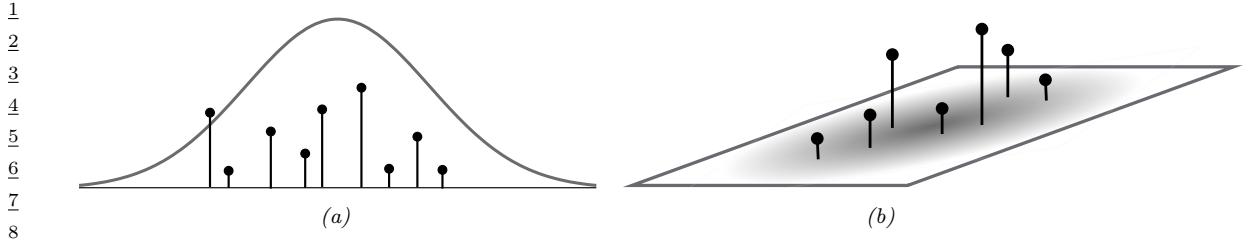
Under a DP prior on G , $(G(T_1), \dots, G(T_K))$ follows a Dirichlet distribution (equation (33.1)). From the Dirichlet-multinomial conjugacy, the posterior for $(G(T_1), \dots, G(T_K))$ given the observations is

$$(G(T_1), \dots, G(T_K)) | \{\bar{\theta}_1, \dots, \bar{\theta}_N\} \sim \text{Dir}(G(T_1) + \sum_{i=1}^N \mathbb{I}(\bar{\theta}_i \in T_1), \dots, G(T_K) + \sum_{i=1}^N \mathbb{I}(\bar{\theta}_i \in T_K)) \quad (33.6)$$

This is true for any finite partition, so that following our earlier definition, the posterior over G itself is a Dirichlet process, and it is easy to see that:

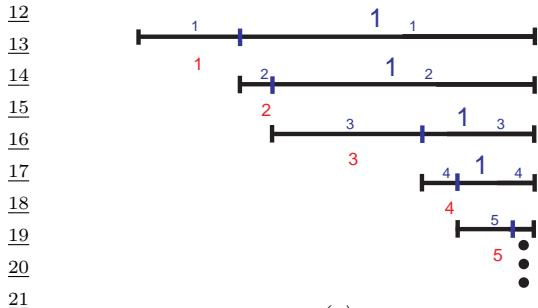
$$G | \bar{\theta}_1, \dots, \bar{\theta}_N, \alpha, H \sim \text{DP} \left(\alpha + N, \frac{1}{\alpha + N} \left(\alpha H + \sum_{i=1}^N \delta_{\theta_i} \right) \right). \quad (33.7)$$

Thus we see that the DP prior on G is a conjugate prior for i.i.d. observations from G , with the posterior distribution over G also a Dirichlet process with concentration parameter $\alpha + N$, and base

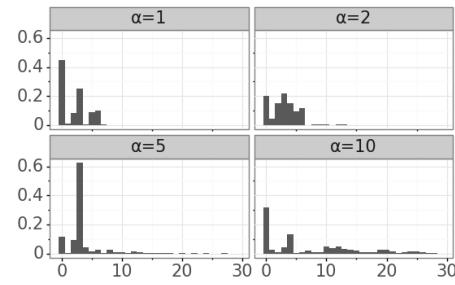


9 Figure 33.2: Realizations from a Dirichlet process when Θ is (a) the real line, and (b) the unit square. Also
10 shown are the base measures αH . In reality, the number of atoms is infinite for both cases.

11



(a)



(b)

23 Figure 33.3: Illustration of the stick breaking construction. (a) We have a unit length stick, which we break at
24 a random point β_1 ; the length of the piece we keep is called π_1 ; we then recursively break off pieces of the
25 remaining stick, to generate π_2, π_3, \dots . From Figure 2.22 of [Sud06]. Used with kind permission of Erik
26 Sudderth. (b) Samples of π_k from this process for different values of α . Generated by `stick_breaking_demo.py`.

27

28

29 measure a convex combination of the original base measure H and the empirical distribution of
30 the observations. Note that as N increases, the influence of the original base measure H starts to
31 wane, and the posterior base measure becomes closer and closer to the empirical distribution of the
32 observations. At the same time, the concentration parameter increases, suggesting that the posterior
33 distribution concentrates around the empirical distribution.

34

35 33.2.2 Stick breaking construction of the DP

36

37 Our discussion so far has been very abstract, with no indication of how to either sample the random
38 measure G or how to sample observations from G . We address the first question, giving a constructive
39 definition for the DP known as the **stick-breaking construction** [Set94].

40 We first mention that probability measures G sampled from a DP are **discrete with probability**
41 **one** (see Figure 33.2), taking the form

42

$$\frac{43}{44} G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}(\theta). \quad (33.8)$$

45

46 Thus G consists of an infinite number of **atoms**, the k th atom located at θ_k , and having weight π_k .

47

Informally, this follows from Equation (33.4), which represents the DP as an infinite-dimensional but infinitely-sparse Dirichlet distribution (recall that as its parameters become smaller, a Dirichlet distribution concentrates on sparse distributions that are dominated by a few components).

For a DP, the locations θ_k of the atoms are drawn independently from the base measure H , whereas the concentration parameter α controls the distribution of the weights π_k . Observe that the infinite sequence of weights (π_1, π_2, \dots) must add up to one, since G is a probability measure. The weights can be simulated by the following process sketched in Figure 33.3, and known as the **stick-breaking process**. Start with a stick of length 1 representing the total probability mass, and sequentially break off a random Beta($1, \alpha$) distributed fraction of the remaining stick. The k th break forms π_k . In equations, for $k = 1, 2, \dots$,

$$\beta_k \sim \text{Beta}(1, \alpha), \quad \theta_k \sim H, \quad (33.9)$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) = \beta_k \left(1 - \sum_{l=1}^{k-1} \pi_l\right) \quad (33.10)$$

Then, setting $G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}(\theta)$, one can show that $G \sim \text{DP}(\alpha, H)$. The distribution over the weights is often denoted by

$$\pi \sim \text{GEM}(\alpha), \quad (33.11)$$

where GEM stands for Griffiths, Engen and McCloskey (this term is due to [Ewe90]). Some samples from this process are shown in Figure 33.3.

We note that since the number of atoms is infinite, one cannot exactly simulate from a DP in finite time. However, the sequence of weights from the GEM distribution are **stochastically ordered**, having decreasing averages, and the truncation error resulting from terminating after a finite number of steps quickly becoming negligible [IJ01]. Nevertheless, we will see in the next section that it is possible to simulate samples and make predictions from a DP-distributed probability measure G without any truncation error. This exploits the conjugacy property of the DP.

33.2.3 The Chinese restaurant process (CRP)

Consider a single observation $\bar{\theta}_1$ from a DP-distributed probability measure G . The probability that $\bar{\theta}_1$ lies within a set $T \in \Theta$, marginalizing out the random G , is $\mathbb{E}[G(T)] = H(T)$, the equality following from the definition of the DP. This holds for arbitrary T , which implies that the first observation $\bar{\theta}_1$ is distributed as the base measure of the DP:

$$p(\bar{\theta}_1 = \theta | \alpha, H) = H(\theta). \quad (33.12)$$

Given N observations $\bar{\theta}_1, \dots, \bar{\theta}_N$, the updated distribution over G is still a DP, but now modified as in Equation (33.7). Repeating the same argument, it follows that the $(N+1)$ st observation is distributed as the base measure of the posterior DP, given by

$$p(\bar{\theta}_{N+1} = \theta | \bar{\theta}_{1:N}, \alpha, H) = \frac{1}{\alpha + N} \left(\alpha H(\theta) + \sum_{k=1}^N N_k \delta_{\bar{\theta}_k}(\theta) \right) \quad (33.13)$$

¹ where N_k is the number of observations equal to $\bar{\theta}_k$. The previous two equations form the basis of
² what is called the **Pólya urn** or **Blackwell-MacQueen** sampling scheme [BM+73]. This provides
³ a way to exactly produce samples from a DP-distributed random probability measure.
⁴

⁵ It is often more convenient to work with discrete variables (z_1, \dots, z_N) , with z_i specifying which
⁶ value of θ_k the i th sample takes. In particular, for the i th observation, $\bar{\theta}_i = \theta_{z_i}$. This allows us to
⁷ decouple the cluster or partition structure of the dataset (controlled by α) and the cluster parameters
⁸ (controlled by H). Let us assign the first observation to cluster 1, i.e. $z_1 = 1$. The second observation
⁹ can either belong to the same cluster as observation 1, or belong to a new cluster, which we call
¹⁰ cluster 2. In the former event, $z_2 = 1$, after which z_3 can equal 1 or 2. In the latter event, $z_2 = 2$,
¹¹ and z_3 can equal 1, 2 or 3. Based on the Equation (33.13), we have

¹²

$$\frac{13}{14} p(z_{N+1} = z | \mathbf{z}_{1:N}, \alpha) = \frac{1}{\alpha + N} \left(\alpha \mathbb{I}(z = K + 1) + \sum_{k=1}^K N_k \mathbb{I}(z = k) \right), \quad (33.14)$$

¹⁵

¹⁶

¹⁷ assuming the first N observations have been assigned to K clusters. This is called the **Chinese**
¹⁸ **restaurant process** or **CRP**, based on the following analogy: observations are customers in a
¹⁹ restaurant with an infinite number of tables, each corresponding to a different cluster. Each table
²⁰ has a dish, corresponding to the parameter θ of that cluster. When a customer enters the restaurant,
²¹ they may choose to join an existing table with probability proportional to the number of people
²² already sitting at this table (i.e. they join table k with probability proportional to N_k); otherwise,
²³ with probability proportional to α , they choose to sit at a new table, ordering a new dish by sampling
²⁴ from the base measure H .

²⁵ The sequence $Z = (z_1, \dots, z_N)$ of cluster assignments is **partition of the integers** 1 to N , and
²⁶ the CRP is a distribution over such partitions. The probability of creating a new table diminishes as
²⁷ the number of observations increases, but is always non-zero, and one can show that the number of
²⁸ occupied tables K approaches $\alpha \log(N)$ as $N \rightarrow \infty$ almost surely. The fact that currently occupied
²⁹ tables are more likely to get new customers is sometimes called a **rich get richer** phenomenon.
³⁰ It is important to recognize that despite being defined as a sequential process, the CRP is an
³¹ **exchangeable process**, with partition probabilities that are independent of the observation indices.
³² Indeed, it is easy to show that the probability of a partition of N integers into K clusters with
³³ sizes N_1, \dots, N_K is

³⁴

$$\frac{35}{36} p(n_1, \dots, n_k) = \frac{\alpha^{K-1}}{[\alpha + 1]_1^N} \prod_{k=1}^K N_k!. \quad (33.15)$$

³⁷

³⁸

³⁹ Here, $[\alpha + 1]_1^N = \prod_{i=0}^{N-1} (\alpha + 1 + i)$ is the rising factorial. Equation (33.15) depends only on the
⁴⁰ cluster sizes, and is called the Ewens sampling formula [Ewe72]. Exchangeability implies that the
⁴¹ probability that the first two customers sit at the same table is the same as the probability that the
⁴² first and last sit at the same table. Similarly all customers have the same probability of ending up in
⁴³ a cluster of size S . The fact that the first customer can only belong to cluster 1 (i.e. that $z_1 = 1$)
⁴⁴ does not contradict exchangeability and reflects the fact that the cluster indices are chosen arbitrarily.
⁴⁵ This disappears if we index clusters by their associated parameter θ_k .

⁴⁶

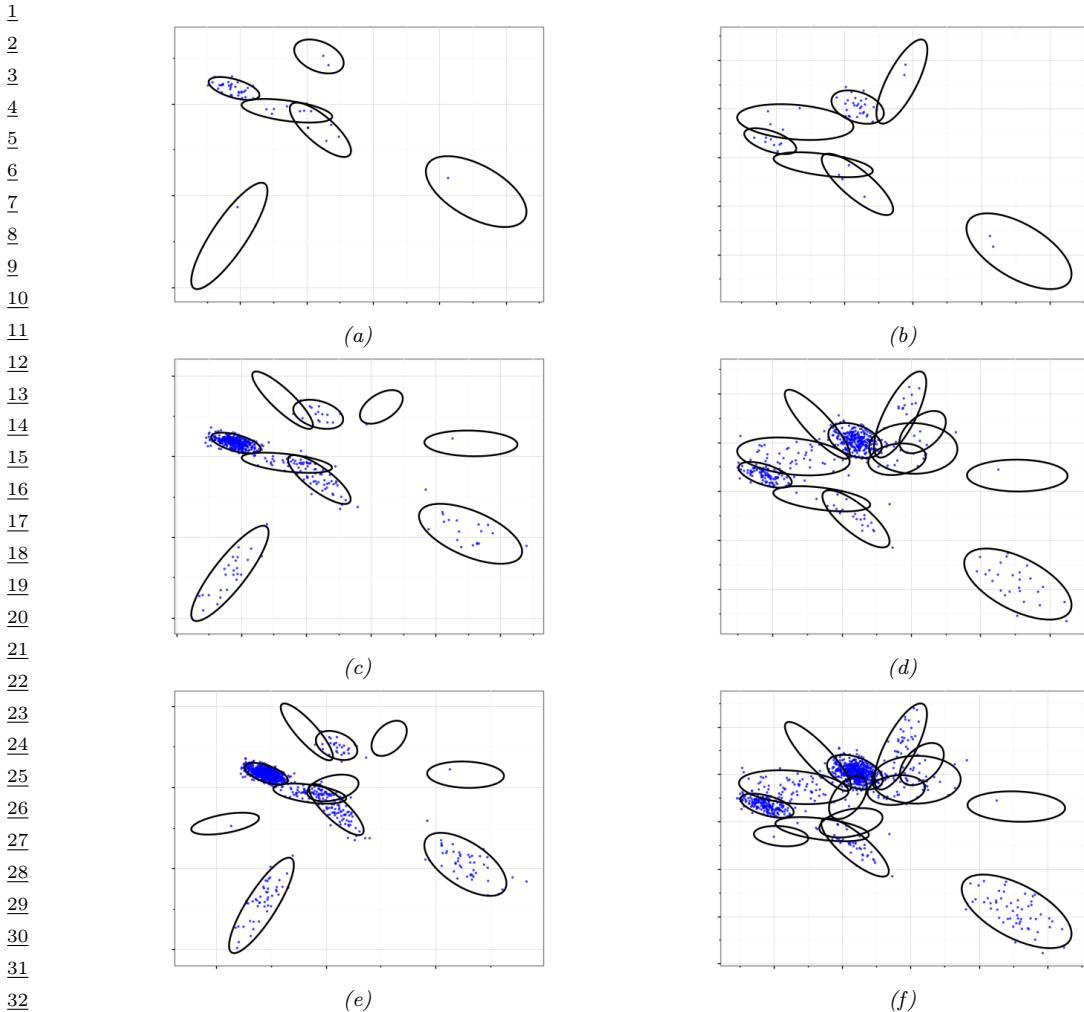


Figure 33.4: Some samples from a Dirichlet process mixture model of 2D Gaussians, with concentration parameter $\alpha = 1$ (left column) and $\alpha = 2$ (right column). From top to bottom, we show $N = 50$, $N = 500$ and $N = 1000$ samples. We also show the model parameters as ellipses, which are sampled from a NIW base distribution. Generated by [dpmSampleDemo.py](#).

33.2.4 Dirichlet process mixture models

Real-world datasets are often best modeled by continuous probability densities. By contrast, a sample G from a DP is discrete with probability one, and sampling observations from G will result in repeated values, making it inappropriate for many applications. However, the discrete structure of G is useful in clustering applications, as a prior for the latent variables underlying the observed datapoints. In particular, z_i and $\bar{\theta}_i$ can represent the cluster assignment and cluster parameter of

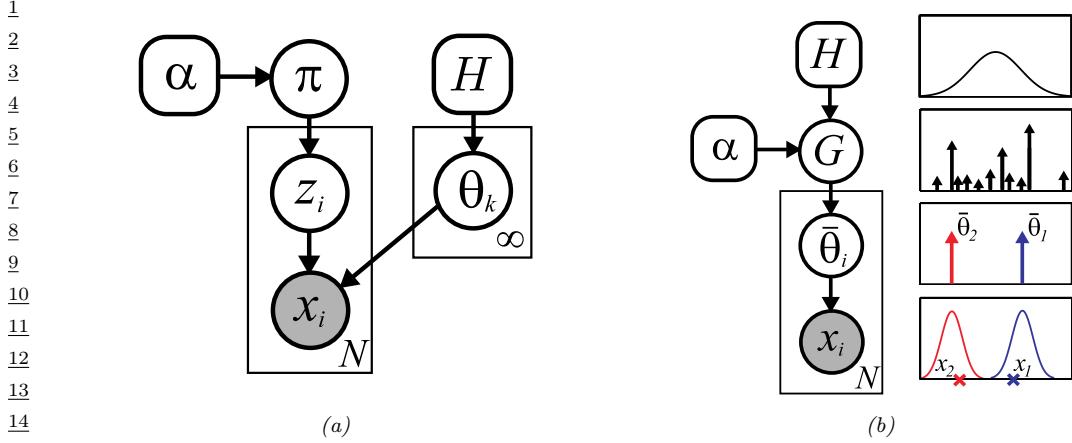


Figure 33.5: Two views of N observations sampled from a DP mixture model. Left: representation where cluster indicators are sampled from the GEM-distributed distribution π . Right: representation where parameters are samples from the DP-distributed RPM G . The rightmost picture illustrates the case where $N = 2$, θ is real-valued with a Gaussian prior $H(\cdot)$, and $F(x|\theta)$ is a Gaussian with mean θ and variance σ^2 . We generate two parameters, $\bar{\theta}_1$ and $\bar{\theta}_2$, from G , one per data point. Finally, we generate two data points, x_1 and x_2 , from $N(\bar{\theta}_1, \sigma^2)$ and $N(\bar{\theta}_2, \sigma^2)$. From Figure 2.24 of [Sud06]. Used with kind permission of Erik Sudderth.

the i th datapoint, whose value x_i is a draw from some parametric distribution $F(x|\theta)$ indexed by θ . The resulting model follows along the lines of a standard mixture model, but now is an **infinite mixture model**, consisting of an infinite number of components or clusters, one for each atom in G . We can write the model as follows:

$$\pi \sim GEM(\alpha), \quad \theta_k \sim H, \quad k = 1, 2, \dots \quad (33.16)$$

$$z_i \sim \pi, \quad x_i \sim F(\theta_{z_i}), \quad i = 1, \dots, N. \quad (33.17)$$

Equivalently, we can write this as

$$G \sim DP(\alpha, H) \quad (33.18)$$

$$\bar{\theta}_i \sim G, \quad x_i \sim F(\bar{\theta}_i), \quad i = 1, \dots, N. \quad (33.19)$$

G and F together define the infinite mixture model: $G_F(x) \sim \sum_{k=1}^{\infty} \pi_k F(x|\theta_k)$. If $F(x|\theta)$ is continuous, then so is $G_F(x)$, and the Dirichlet process mixture model serves as a nonparametric prior over continuous distributions or probability densities. A very common setting when $x_i \in \mathbb{R}^d$ is to set F to be the multivariate normal distribution, $\theta = (\mu, \Sigma)$, and H to be the Normal-Inverse-Wishart distribution. Then, each of the infinite clusters has an associated mean and covariance matrix, and to generate a new observation, one picks cluster k with probability π_k , and simulates from a normal with mean μ_k and covariance Σ_k . Figure 33.5 illustrates two graphical models that summarize this, corresponding to the two sets of equations above. The first generates the set of weights (π_1, π_2, \dots) from the GEM distribution, along with an infinite collection of cluster parameters $(\theta_1, \theta_2, \dots)$. It then generates observations by first sampling a cluster indicator z_i from π , indexing the associated cluster parameter θ_{z_i} and then simulating the observation x_i from $F(\theta_{z_i})$. The second graphical

model simulates a random measure G from the DP. It generates observations by directly simulating a parameter $\bar{\theta}_i$ from G , and then simulating x_i from $F(\bar{\theta}_i)$. The infinite mixture model can be viewed as the limit of a K -component finite mixture model with a $\text{Dir}(\alpha/K, \dots, \alpha/K)$ prior on the mixture weights (π_1, \dots, π_K) and with mixture parameters $\theta_1, \dots, \theta_K$, as $K \rightarrow \infty$ [Ras00; Nea00]. Producing exact samples (x_1, \dots, x_N) from this model involves one additional step to the Chinese restaurant process: after selecting a table (cluster) z_i with its associate dish (parameter) θ_{z_i} , the i th customer now samples an observation from the distribution $F(\theta_{z_i})$.

33.2.4.1 Fitting a DP mixture model

Given a dataset of observations, one is interested in the posterior distribution $p(G, z_1, \dots, z_N | x_1, \dots, x_N, \alpha, H)$, or equivalently, $p(\boldsymbol{\pi}, \theta_1, \theta_2, \dots, z_1, \dots, z_N | x_1, \dots, x_N, \alpha, H)$. The most common way to fit a DPMM is via Markov chain Monte Carlo (MCMC), producing samples by constructing a Markov chain that targets this posterior distribution. We describe a **collapsed Gibbs sampler** based on the Chinese restaurant process that marginalizes out the infinite-dimensional random measure G , and that targets the distribution $p(z_1, \dots, z_N | x_1, \dots, x_N, \alpha, H)$ summarizing all clustering information. It produces samples from this distribution by cycling through each observation x_i , and updating its assigned cluster z_i , conditioned on all other variables. Write \mathbf{x}_{-i} for all observations other than the i th observation, and \mathbf{z}_{-i} for their cluster assignments. Then we have

$$p(z_i = k | \mathbf{z}_{-i}, \mathbf{x}, \alpha, H) \propto p(z_i = k | \mathbf{z}_{-i}, \alpha) p(x_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, H) \quad (33.20)$$

By exchangeability, each observation can be treated as the last customer to enter the restaurant. Hence the first term is given by

$$p(z_i | \mathbf{z}_{-i}, \alpha) = \frac{1}{\alpha + N - 1} \left(\alpha \mathbb{I}(z_i = K_{-i} + 1) + \sum_{k=1}^{K_{-i}} N_{k,-i} \mathbb{I}(z_i = k) \right) \quad (33.21)$$

where $N_{k,-i}$ is the number of observations in cluster k , and K_{-i} is the number of clusters used by \mathbf{x}_{-i} , both obtained after removing observation i , eliminating empty clusters, and renumbering clusters.

To compute the second term, $p(x_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, H)$, let us partition the data \mathbf{x}_{-i} into clusters based on \mathbf{z}_{-i} . Let $\mathbf{x}_{-i,c} = \{x_j : z_j = c, j \neq i\}$ be the data points assigned to cluster c . If $z_i = k$, then x_i is conditionally independent of all the data points except those assigned to cluster k . Hence,

$$p(x_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k) = p(x_i | \mathbf{x}_{-i,k}) = \frac{p(x_i, \mathbf{x}_{-i,k})}{p(\mathbf{x}_{-i,k})}, \quad (33.22)$$

where

$$p(x_i, \mathbf{x}_{-i,k}) = \int p(x_i | \theta_k) \left[\prod_{j \neq i: z_j=k} p(x_j | \theta_k) \right] H(\theta_k) d\theta_k \quad (33.23)$$

is the marginal likelihood of all the data assigned to cluster k , including i , and $p(\mathbf{x}_{-i,k})$ is an analogous expression excluding i . Thus we see that the term $p(x_i | \mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k)$ is the posterior predictive distribution for cluster k evaluated at x_i .

1 If $z_i = k^*$, corresponding to a new cluster, we have
 2
 3 $p(x_i|\mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k^*) = p(x_i) = \int p(x_i|\theta)H(\theta)d\theta$ (33.24)
 4
 5

6 which is just the prior predictive distribution for a new cluster evaluated at x_i .
 7

8 The overall sampler is sometimes called ‘‘Algorithm 3’’ (from [Nea00]). Algorithm 33 provides the
 9 pseudocode. The algorithm is very similar to collapsed Gibbs for finite mixtures except that we
 10 have to consider the case $z_i = k^*$. Note that in order to evaluate the integrals in Equation (33.23)
 11 and Equation (33.24), we require the base measure H to be conjugate to the likelihood F . In
 12 such conjugate settings, it is then also straightforward to sample cluster parameters given cluster
 13 assignments (if needed). Extensions to the case of non-conjugate priors are discussed in [Nea00].
 14

Algorithm 33: Collapsed Gibbs sampler for DPMM given data $\mathcal{D} = \{x_1, \dots, x_N\}$

```

15 1 for each  $i = 1 : N$  in random order do
16 2   Remove  $\mathbf{x}_i$ ’s sufficient statistics from old cluster  $z_i$  ;
17 3   for each  $k = 1 : K$  do
18 4     Compute  $p_k(\mathbf{x}_i) = p(\mathbf{x}_i|\mathbf{x}_{-i}(k))$  ;
19 5     Set  $N_{k,-i} = \dim(\mathbf{x}_{-i}(k))$  ;
20 6     Compute  $p(z_i = k|\mathbf{z}_{-i}, \mathcal{D}) = \frac{N_{k,-i}}{\alpha+N-1}$  ;
21 7     Compute  $p(z_i = *|\mathbf{z}_{-i}, \mathcal{D}) = \frac{\alpha}{\alpha+N-1}p(\mathbf{x}_i)$  ;
22 8     Normalize  $p(z_i|\cdot)$  ;
23 9     Sample  $z_i \sim p(z_i|\cdot)$  ;
24 10    Add  $\mathbf{x}_i$ ’s sufficient statistics to new cluster  $z_i$  ;
25 11    If any cluster is empty, remove it and decrease  $K$ ;  

26
27
  
```

29 An example of this procedure in action is shown in Figure 33.6, where the sample clusterings,
 30 and the induced posterior over K , are reasonable looking. The MCMC algorithm tends to rapidly
 31 discover a good clustering. By contrast, Gibbs sampling (and EM) for a finite mixture model often
 32 gets stuck in poor local optima (not shown). This is because the DPMM is able to create extra
 33 redundant clusters early on, and to use them to escape local optima. The traceplots in Figure 33.7
 34 show that most of the time, the MCMC algorithm for the DPMM converges rapidly.
 35

36 An important issue is how to set the model hyper-parameters. These include the DP concentration
 37 parameter α , as well as any parameters λ of the base measure H . For the DP, the value of α does
 38 not have much impact on predictive accuracy, but has quite a strong affect the number of clusters.
 39 One approach is to put a Gamma prior for α , and then form its posterior, $p(\alpha|K, N, a, b)$ [EW95].
 40 Simulating α given the cluster assignments \mathbf{z} is quite straightforward, and can be incorporated into
 41 the earlier Gibbs sampler. The same is the case with the hyper-parameters λ [Ras00]. Alternatively,
 42 one can use empirical Bayes [MBJ06] to fit rather than sample these parameters.
 43

44 While Algorithm 33 is the simplest approach to posterior inference for DPMMs, a variety of
 45 other methods have been proposed as well. One popular class of MCMC samplers works with the
 46 stick-breaking representation of the DP instead of CRP, instantiating the random measure G [IJ01].
 47 The sampler then proceeds by sampling the cluster assignments \mathbf{z} given G , and then G given \mathbf{z} . An
 48 advantage of this is that the cluster assignments can be updated in parallel, unlike the CRP, where
 49

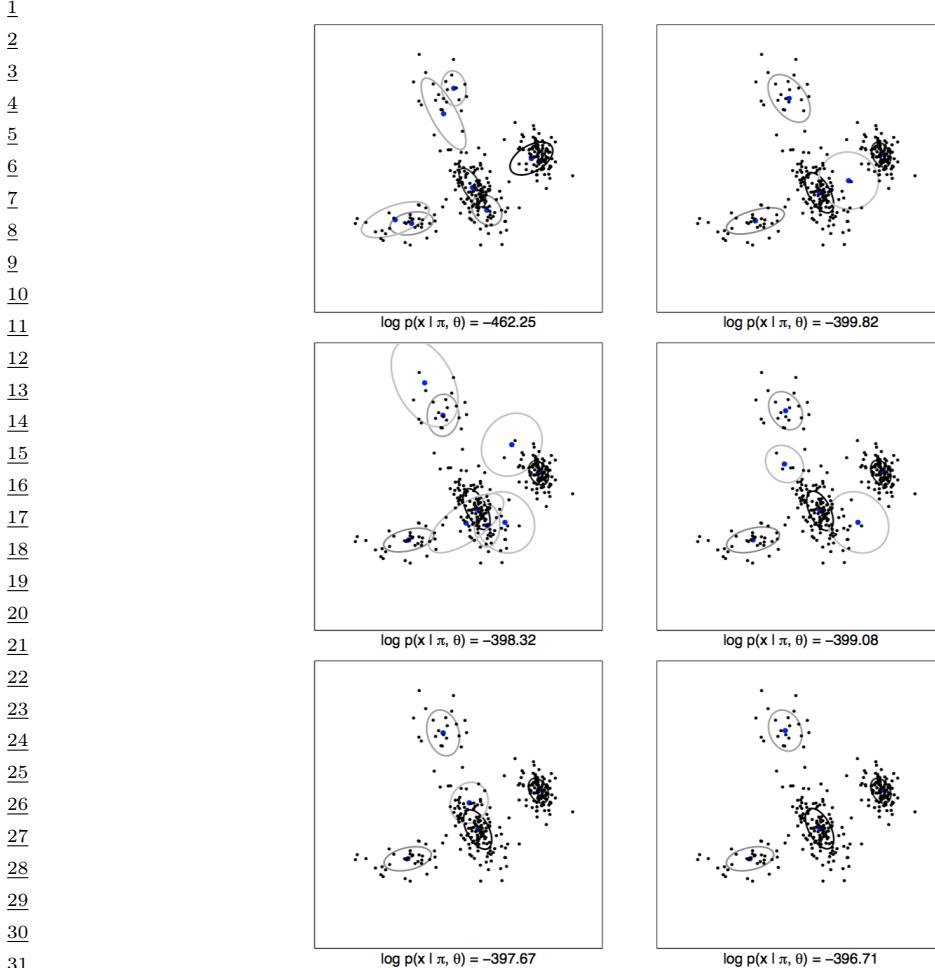
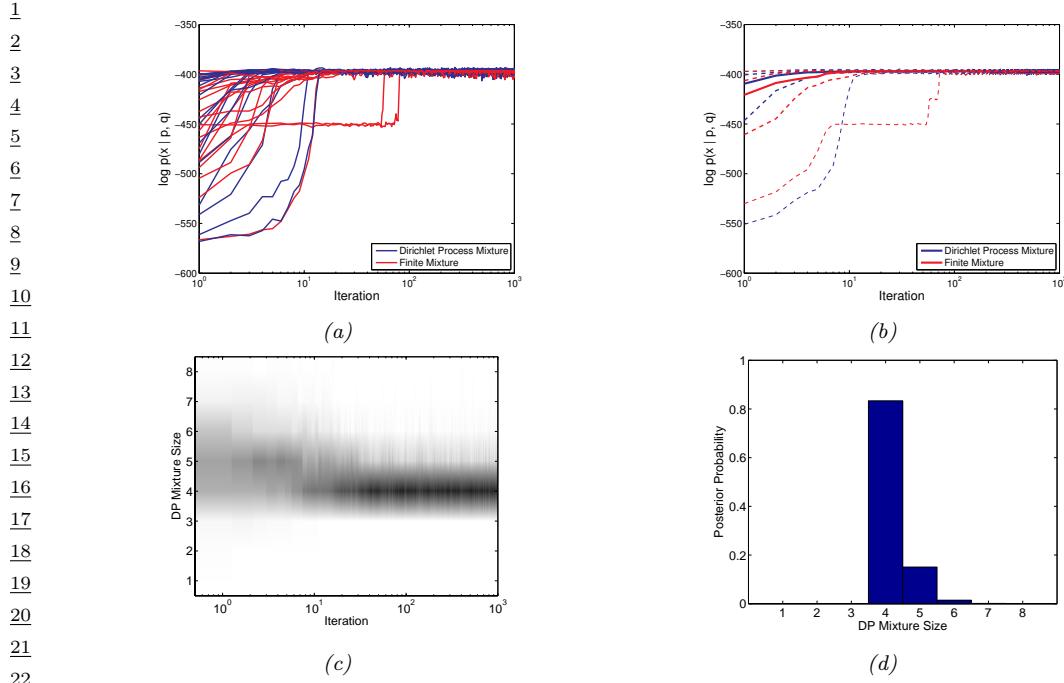


Figure 33.6: 300 data points in 2 dimensions are clustered using a DP mixture fit with collapsed Gibbs sampling. The two columns show different random initializations. The rows represent samples from the posterior over model parameters after $T = 2$ (top), $T = 10$ (middle) and $T = 50$ (bottom) iterations. From Figure 2.26 of [Sud06]. Used with kind permission of Erik Sudderth.

they are updated sequentially. To be feasible however, these methods require truncating G to a finite number of atoms, though the resulting approximation error can be quite small. The posterior approximation error can be eliminated altogether by slice-sampling methods [KGW11], that work with random truncation levels. Alternatives to MCMC also exist. [Dau07] shows how one can use A* search and beam search to quickly find an approximate MAP estimate. [Man+07] discusses how to fit a DPMM online using particle filtering, which is a like a stochastic version of beam search. This can be more efficient than Gibbs sampling, particularly for large datasets. A variety of variational approximation methods have been proposed as well, for example [BJ+06; KWW06; TKW08; Zob09;



23 *Figure 33.7: Comparison of collapsed Gibbs samplers for a DP mixture (dark blue)*
 24 *and a finite mixture (light red) with $K = 4$ applied to the $N = 300$ data points shown in Figure 33.6.* (a) Log probability $\log p(X|\pi, \theta)$ vs
 25 MCMC iteration for 20 different starting values. (b) Median (thick line) and quantiles (dashed lines) over
 26 100 different starting values. (c) Number of components with at least 2% of the probability mass, vs iteration.
 27 (Intensity is proportional to posterior probability.) (d) Posterior over K based on last 900 samples. From
 28 Figure 2.27 of [Sud06]. Used with kind permission of Erik Sudderth.

29
30

31 [WB12] among many others.

32

33.3 Generalizations of the Dirichlet process

34

35 Dirichlet process mixture models are flexible nonparametric models of continuous probability densities,
 36 and if set up with a little care, can possess important frequentist properties like **asymptotic**
 37 **consistency**: with more and more observations, the posterior distribution concentrates around the
 38 ‘true’ data generating distribution, with very little assumed about the this distribution. Nevertheless,
 39 DPs still represent very strong prior information, especially in clustering applications. We saw that the
 40 number of clusters in a dataset of size N a priori is around $\alpha \log N$. As indicated by Equation (33.15),
 41 not just the number of clusters, but also the distribution of their sizes is controlled by a single
 42 parameter α . The resulting clustering model is thus quite inflexible, and in many cases, inappropriate.
 43 Two examples from machine learning that highlight its limitations are applications involving text and
 44 image data. Here, it has been observed empirically that the number of unique words in a document,
 45 the frequency of word usage, the number of objects in an image, or the number of pixels in an object
 46

47

1 follow power-law distributions. Clusterings sampled from the CRP do not produce this property, and
2 the resulting model mismatch can result in poor predictive performance.
3

4 33.3.1 Pitman-Yor process

5 A popular generalization of the Dirichlet process is the Pitman-Yor process [PY97] (also called the
6 two-parameter Poisson-Dirichlet process). Written at PYP(α, d, H), the Pitman-Yor process includes
7 an additional **discount parameter** d which must be greater than 0. The concentration parameter
8 can now be negative, but must satisfy $\alpha \geq -d$. As with the DP, a sample G from a PYP is a
9 random probability measure that is discrete almost surely. It has a stick-breaking representation
10 that generalizes that of the DP: again, we start with a stick of length 1, but now at step k , a random
11 Beta($1 - d, \alpha + kd$) fraction of the remaining probability mass is broken off, so that G is written as
12

$$\text{13} G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}, \quad (33.25)$$

$$\text{14} \beta_k \sim \text{Beta}(1 - d, \alpha + kd), \quad \theta_k \sim H, \quad (33.26)$$

$$\text{15} \pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) = \beta_k \left(1 - \sum_{l=1}^{k-1} \pi_l\right). \quad (33.27)$$

16 Because G is discrete, once again observations $\bar{\theta}_1, \dots, \bar{\theta}_d$ sampled i.i.d. from G will possess a clustering
17 structure. These can directly be sampled following a sequential process that generalizes the CRP.
18 Now, when a new customer enters the restaurant, they join an existing table with N_k customers
19 with probability proportional to $(N_k - d)$, and create a new table with probability proportional to
20 $\alpha + Kd$, where K is the number of clusters.
21

22 Observe that the Dirichlet Process is a special instance of the Pitman-Yor process, corresponding
23 to d equal to 0. Non-zero settings of d counteract the rich-get-richer dynamics of the DP to some
24 extent, increasing the probability of creating new clusters. The more clusters there are present in a
25 dataset, the higher the probability of creating a new cluster (relative to the Dirichlet process). This
26 behavior means that a large number of clusters, as well as a few very large clusters are more likely
27 under the PYP than the DP.

28 An even more general class of probability measures are the so-called Gibbs-type priors [DB+13].
29 Under such a prior, given N observations, the probability these are clustered into K clusters, the k th
30 having n_k observations, is

$$\text{31} p(n_1, \dots, n_k) = V_{N,K} \prod_{k=1}^K (\sigma - 1)_{N_k - 1}, \quad (33.28)$$

32 where $(\sigma)_n = \sigma(\sigma + 1) \dots (\sigma + n - 1)$, and for non-negative weights $V_{N,K}$ satisfying $V_{N,K} =$
33 $(N - \sigma K)V_{N+1,K} + V_{N+1,K+1}$ and $V_{1,1} = 1$. This definition ensures that the probability over
34 partitions is consistent, exchangeable and tractable, and includes the DP and PYP as special cases.
35 Besides these two, Gibbs-type priors include settings where the number of components (or the number
36 of atoms in the random measures) are random but bounded. Recall that DP and PYP mixture
37 models are infinite mixture models, with the number of components growing with the number of
38 observations. A sometimes undesirable feature of these models is that if a dataset is generated from
39

¹ a finite number of clusters, these models will not this recover the true number of clusters [MH14].
² Instead, the estimated number of clusters will increases with the size of the dataset, resulting in
³ redundant clusters that are located very close to each other. Gibbs-type priors with almost surely
⁴ bounded number of components can learn the true number of clusters while still remaining reasonably
⁵ tractable: so long as one can calculate the $V_{N,K}$ s, MCMC for all these models can by carried out by
⁶ modifications of the CRP-based sampler described earlier.
⁷

⁸

⁹ 33.3.2 Dependent random probability measures

¹⁰

¹¹ Dirichlet processes, and more generally, random probability measures, have also been generalized
¹² from settings with a single set of observations to those involving grouped observations, or observations
¹³ indexed by covariates. Consider T sets of observations $\{\mathbf{X}^1, \dots, \mathbf{X}^T\}$, each perhaps corresponding to
¹⁴ a different country, a different year, or more abstractly, a different set of observed covariates. There
¹⁵ are two immediate (though inadequate) ways to model such data: 1) by pooling all datasets into a
¹⁶ single dataset, modeled as drawn from a DP or DPMM-distributed random probability measure G ,
¹⁷ or 2) by treating each group as independent, having its own random probability measure G^t . The
¹⁸ first approach fails to capture differences between the groups, while the second ignores similarities
¹⁹ between the groups (e.g. shared clusters). Dependent random probability measures seek a compromise
²⁰ between these extremes, allowing statistical information to be shared across different groups.

²¹ A seminal paper in the machine learning literature that addresses this problem is the **hierarchical**
²² **Dirichlet process** (HDP) [Teh+06]. The basic idea here is simple: each group has its own random
²³ measure drawn from a Dirichlet process $DP(\alpha, H)$. The twist now is that the base measure H itself
²⁴ is random, in fact it is itself drawn from a Dirichlet process. Thus, the overall generative process is

$$\underline{26} \quad H \sim DP(\alpha_0, H_0), \tag{33.29}$$

$$\underline{27} \quad G^t \sim DP(\alpha, H), \quad t \in 1, \dots, T \tag{33.30}$$

$$\underline{28} \quad \bar{\theta}_i^t \sim G^t, \quad i \in 1, \dots, N_t, \quad t \in 1, \dots, T. \tag{33.31}$$

³⁰ The $\bar{\theta}_i^t$ s might be the observations themselves, or the latent parameter underlying each observation,
³¹ with $\mathbf{x}_i^t \sim F(\bar{\theta}_i^t)$.

³³ Recall that if a probability measure G^1 is drawn from $DP(\alpha, H)$, its atoms are drawn independently
³⁴ from the base measure H . In the HDP, H , which is a draw from a DP, is discrete, so that some
³⁵ atoms of G^1 will sit on top of each other, becoming a single atom. More importantly, all measures
³⁶ G^t will share the same infinite set of locations: each atom of H will eventually be sampled to form a
³⁷ location of an atom of each G^t . This will allow the same clusters to appear in all groups, though they
³⁸ will have different weights. Moreover, a big cluster in one group is *a priori* likely to be a big cluster
³⁹ in another group, as underlying both is a large atom in H . Since the common probability measure
⁴⁰ H itself is random, its components (both weights as well as locations) will be learned from data.
⁴¹ Despite its apparent complexity, it is fairly easy to develop an analogue of the Chinese restaurant
⁴² process for the HDP, allowing us to sample observations directly without having to instantiate any of
⁴³ the infinite-dimensional measures. This is called the Chinese restaurant franchise, and essentially
⁴⁴ amounts to each group having its own Chinese restaurant with the following modification: whenever
⁴⁵ a customer sits at a new table and orders a dish, that dish itself is sampled from an upper CRP
⁴⁶ common to all restaurants. It is also possible to develop stick-breaking representations of the HDP.
⁴⁷

The HDP has found wide application in the machine learning literature. A common application is document modeling, where the location of each atom is a **topic**, corresponding to some distribution over words. Rather than bounding the number of topics, there are an infinite number of topics, with document d having its own distribution over topics (represented by a measure G^d). By tying all the G^d 's together through an HDP, different documents can share the same topics while emphasizing different topics. Another application involves infinite-state hidden Markov models. Recall a Markov chain is parametrized by a transition matrix, with row r giving the distribution over the next state if the current state is r . For an infinite-state HMM, this is an infinite-by-infinite matrix, with row r corresponding to a distribution G^r with an infinite number of atoms. The different G^r 's can be tied together by modeling them with an HDP, so that atoms from each correspond to the same states.

Hierarchical nonparametric models of this kind can be constructed with other measures such as the Pitman-Yor process. For certain parameter settings, the PYP possesses convenient marginalization properties that the DP does not. In particular, simulating an RPM in two steps as shown below

$$G_0|H_0 \sim \text{PYP}(\alpha d_0, d_0, H_0), \quad (33.32)$$

$$G_1|G_0 \sim \text{PYP}(\alpha, d_1, G_0) \quad (33.33)$$

is equivalent to directly simulating G_1 without instantiating G_0 as below:

$$G_1|H_0 \sim \text{PYP}(\alpha d_0, d_0 d_1, G_0). \quad (33.34)$$

This marginalization property (also called **coagulation**) allows deep hierarchies of dependent RPMs, with only a smaller, dataset-dependent subset of G 's having to be instantiated. In the **sequence memoizer** of [Woo+11], the authors model sequential data (e.g. text) with hierarchies that are *infinitely* deep, but with only a finite number of levels ever having to be instantiated. If needed, intermediate random measures can be instantiated by a dual **fragmentation** operator.

Deeper hierarchies like those described above allow more refined modeling of similarity between different groups. Under the original HDP, the groups themselves are exchangeable, with no subset of groups *a priori* more similar to each other than to others. For instance, suppose each of the measures G_1, \dots, G_T correspond to distributions over topics in different scientific journals. Modeling the G_t 's with an HDP allows statistical sharing across journals (through shared clusters and similar cluster probabilities), but does not regard some journals as *a priori* more similar than others. If one had further information, e.g. that some are physics journals and the rest are biology journals, then one might add another level to the hierarchy. Now rather than each journal having a probability measure G^t drawn from a DP(α, H), physics and biology journals have their own base measures H_p and H_b that allow statistical sharing among physics and biology journals respectively. Like the HDP, these are draws from a DP with base measure H . To allow sharing *across* disciplines, H is again random, drawn from a DP with base measure H_0 . Overall, if there are D disciplines, $1, 2, \dots, D$, the overall model is

$$H \sim \text{DP}(\alpha_0, H_0), \quad (33.35)$$

$$H^d \sim \text{DP}(\alpha_1, H), \quad d \in 1, \dots, D \quad (33.36)$$

$$G^{t,d} \sim \text{DP}(\alpha_2, H^d), \quad t \in 1, \dots, T_d \quad (33.37)$$

$$\bar{\theta}_i^{t,d} \sim G^{t,d} \quad d \in 1, \dots, T_d, \quad D, i \in 1, \dots, N_t. \quad (33.38)$$

¹ One might add further levels to the hierarchy given more information (e.g. if disciplines are
² grouped into physical sciences, social sciences and humanities).

³ Dependent random probability measures can also be indexed by covariates in some continuous
⁴ space, whether time, space, or some Euclidean space or manifold. This space is typically endowed
⁵ with some distance or similarity function, and one expects that measures with similar covariates
⁶ are *a priori* more similar to each other. Thus, G^t might represent a distribution over topics in
⁷ year t , and one might wish to model G^t evolving gradually over time. There is rich history of
⁸ dependent random probability measures in statistics literature, starting from [Mac99a]. A common
⁹ requirement in such models is that at any fixed time t , the marginal distribution of G^t follows some
¹⁰ well specified distribution, e.g. a Dirichlet process, or a PYP. Approaches exploit the stick-breaking
¹¹ representation, the CRP representation or something else like the Poisson structure underlying such
¹² models (see Section 33.6).

¹³ As a simple and early example, recall the stick-breaking construction from Section 33.2.2, where a
¹⁴ random probability measure is represented by an infinite collection of pairs (β_k, θ_k) . To construct
¹⁵ a family of RPMs indexed by some covariate t , we need a family of such sets (β_k^t, θ_k^t) for each of
¹⁶ the possibly infinite values of t . To ensure each G_t is marginally DP distributed with concentration
¹⁷ parameter α and base measure H , we need that for each t , the β_k^t 's are marginally i.i.d. draws from a
¹⁸ Beta($1, \alpha$), and the θ_k^t 's i.i.d. draws from H . Further, we do not want independence across t , rather,
¹⁹ for two times t_1 and t_2 , we want similarity to increase as $|t_1 - t_2|$ decreases. To achieve this, define
²⁰ an infinite sequence of independent Gaussian processes on T , $f_k(t)$, $k = 1, 2, \dots$, with mean 0 and
²¹ some covariance function. At any time t , $f_k(t)$ for all k are i.i.d. draws from a normal distribution.
²² By transforming these Gaussian processes through the cdf of a Gaussian density (to produce a i.i.d.
²³ uniform random variables at each t), and then through the inverse cdf of a Beta($1, \alpha$)), one has an
²⁴ infinite collection of i.i.d. Beta($1, \alpha$) random variables at any time t . The GP construction means
²⁵ that each β_k^t varies smoothly with t . A similar approach can be used to construct smoothly varying
²⁶ θ_k^t 's that are marginally H -distributed, and together, these a family of gradually varying RPMs G^t ,
²⁷ with

$$\sum_{i=1}^{\infty} \pi_k^t \delta_{\theta_k^t}, \quad \text{where } \pi_k^t = \beta_k^t \prod_{j=1}^{k-1} (1 - \beta_j^t) \quad (33.39)$$

²⁸ Of course, such a model comes with formidable computational challenges, however the marginal
²⁹ properties allows standard MCMC methods from the DP and other RPMs to be adapted to such
³⁰ settings.

³¹

³² 33.4 The Indian buffet process and the Beta process

³³

³⁴ The Dirichlet process is a Bayesian nonparametric model useful for clustering applications, where
³⁵ the number of clusters is allowed to grow with the size of the dataset. Under this generative model,
³⁶ each observation is assigned to a cluster through the variable z_i . Equivalently, z_i can be written as
³⁷ a one-hot binary vector, and the entire clustering structure can be written as a binary matrix Z
³⁸ consisting of N rows, each with exactly one non-zero element (Figure 33.8(a)).

³⁹ Clustering models that limit each observation to a single cluster can be overly restrictive, failing to
⁴⁰ capture the complexity of the real datasets. For instance, in computer vision applications, rather
⁴¹ than assign an image to a single cluster, it might be more appropriate to assign it a binary vector of
⁴²

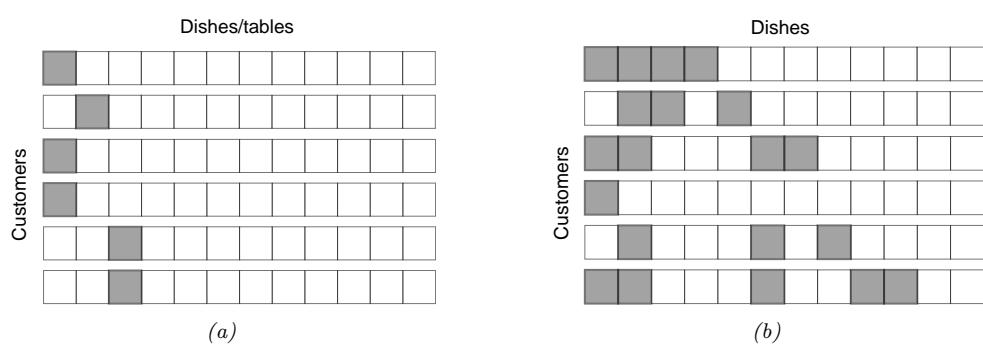


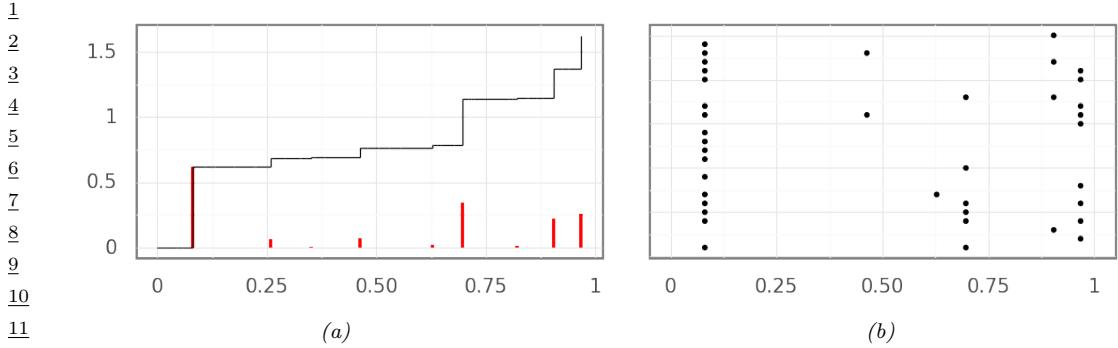
Figure 33.8: (a) A realization of the cluster matrix Z from the Chinese restaurant process (CRP) (b) A realization of the feature matrix Z from the Indian buffet process (IBP). Rows are customers and columns are dishes (for the CRP each table has its own dish). Both produce binary matrices, with the CRP assigning a customer to a single table (cluster), and the IBP assigning a set of dishes (features) to each customer.

features, indicating whether different object types are or are not present in the image. Similarly, in movie recommendation systems, rather than assign a movie to a single genre ('comedy' or 'romance' etc.), it is more realistic to assign to multiple genres ('comedy' AND 'romance'). Now, in contrast to all-or-nothing clustering (which would require a new genre 'romantic comedy'), different movies can have different but overlapping sets of features, allowing a partial sharing of statistical information.

Latent feature models generalize clustering models, allowing each observation to have multiple features. Nonparametric latent feature models allow the number of available features to be infinite (rather than fixed *a priori* to some finite number), with the number of active features in a dataset growing with a dataset size. Such models associate the dataset with an infinite binary matrix consisting of N rows, but now where each row can have multiple elements set to 1, corresponding to the active features. This is shown in Figure 33.8(b). As with the clustering models, each column is associated with a parameter drawn i.i.d. from some base measure H .

The Indian buffet process (IBP) in a Bayesian nonparametric analogue of the CRP for latent feature models. As with the CRP, the IBP is specified by a concentration parameter α , and a base measure H on some space Θ . The former controls the distribution over the binary feature matrix, whereas feature parameters θ_k are drawn i.i.d. from the latter. Under the IBP, individuals enter sequentially into a restaurant, now picking a set of dishes (instead of a single table). The first customer samples a Poisson(α)-distributed random number of dishes, and assigns each of them values drawn from H . When the i th customer enters the restaurant, they first make a pass through all dishes already chosen. Suppose N_d of the earlier customers have chosen dish d : then customer i selects this with probability N_d/i . This results in a rich-get-richer phenomenon, where popular dishes (common features) are more likely to be selected in the future. Additionally, the i customer samples a Poisson(α/i) number of new dishes. The results in a non-zero probability of new dishes, that nonetheless decreases with i .

A key property of the Indian buffet process is that, like the CRP, it is exchangeable. In other words, its statistical properties do not change if its rows are permuted. For instance, one can show that the number of dishes picked by any customer is marginally Poisson(α) distributed. Similarly, the distribution over the number of features shared by the first two customers is the same as the that for



13 *Figure 33.9: (a)* A realization of a Beta process on the real line. The atoms of the random measure are
 14 shown in red, while the Beta subordinator (whose value at time t sums all atoms up to t) is shown in black.
 15 *(b)* Samples from the Beta process

16
 17
 18 the first and last customer. We mention that like the CRP, the ordering of the dishes (or columns) is
 19 arbitrary and might appear to violate exchangeability. For instance, the first customer cannot pick
 20 the first and third dishes and not the second dish, while this is possible for the second customer.
 21 These artefacts disappear if we index columns by their associated parameters. Equivalently, after
 22 reordering rows, we can transform the feature matrix to be left-ordered (essentially, all new dishes
 23 selected by a customer must be adjacent, see [GG11]), and we can view the IBP as a prior on such
 24 left-ordered matrices.
 25

26 The exchangeability of the rows of the IBP implies via de Finetti's theorem that there exists
 27 an underlying random measure G , conditioned on which the rows are i.i.d. draws. Just as the
 28 Chinese restaurant process represents observations drawn from a Dirichlet process-distributed random
 29 probability measure, the dishes of each customer in the IBP represent observations drawn from a
 30 **Beta process**. Like the DP, the Beta process is an atomic measure taking the form

$$31 \quad G(\boldsymbol{\theta}) = \sum_{k=1}^{\infty} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}). \quad (33.40)$$

32
 33
 34

35 Each of the π_k 's lie between 0 and 1, but unlike the DP where they add up to 1, the π_k 's sum up
 36 to a finite random number. The Beta process is thus a **random measure**, rather than a random
 37 probability measure. One can imagine the k th atom as a coin located at $\boldsymbol{\theta}_k$, with probability of
 38 success equal to π_k . To simulate a row of the IBP, one flips each of the infinite coins, selecting the
 39 feature at $\boldsymbol{\theta}_k$ if the k coin comes up heads. One can show that if the π_k 's sum up to a finite number,
 40 the number of active features will be finite. Of the infinite atoms in G , a few will dominate the rest,
 41 these will be revealed through the rich-gets-richer dynamics of the IBP as features common to a large
 42 proportion of the observations.

43 The Beta process has a construction similar to the stick-breaking representation of the Dirichlet
 44 process. As with the DP, the locations of the atoms are independent draws from the base measure,
 45 while the sequence of weights π_1, π_2, \dots are constructed from an infinite sequence of Beta variables:
 46 now these are $\text{Beta}(\alpha, 1)$ distributed, rather than $\text{Beta}(1, \alpha)$ distributed. The overall representation
 47

1 of the Beta process is then

$$\beta_k \sim \text{Beta}(\alpha, 1), \quad \theta_k \sim H, \quad (33.41)$$

$$\pi_k = \beta_k \pi_{k1} = \prod_{l=1}^k \beta_l \quad (33.42)$$

9 It is not hard to see that under the IBP, the total number of dishes underlying a dataset of size
10 N follows a $\text{Poisson}(\alpha H_N)$ distribution, where $H_N = \sum_{i=1}^N 1/i$ is the N th harmonic number.
11 The Beta process and the IBP have been generalized to three-parameter versions allowing power-law
12 behavior in the total number of dishes (features) in a dataset of size N , as well as in the number
13 of customers trying each dish [TG09]. It has found application in tasks ranging from genetics,
14 collaborative filtering, and in models for graph and graphical model structures. Just as with the
15 DP, posterior inference can proceed via MCMC (exploiting either the IBP or the stick-breaking
16 representation), particle filtering or using variational methods.

18 33.5 Small-variance asymptotics

20 Nonparametric Bayesian methods can serve as a basis to develop new and efficient discrete optimization
21 algorithms that have the flavor of Lloyd's algorithm for the k -means objective function. The starting
22 point for this line of work is the view of the k -means algorithm as the *small-variance asymptotic*
23 limit of the EM algorithm for a mixture of Gaussians. Specifically, consider the EM algorithm to
24 estimate the unknown cluster means $\mu \equiv (\mu_1, \dots, \mu_k)$ of a mixture of k Gaussians, all of which have
25 the same known covariance $\sigma^2 I$. In the limit as $\sigma^2 \rightarrow 0$, the E-step, which computes the cluster
26 assignment probabilities of each observation given the current parameters $\mu^{(t)}$, now just assigns each
27 observation to the nearest cluster. The M-step recomputes a new set of parameters $\mu^{(t+1)}$ given the
28 cluster assignment probabilities, and when each observation is hard-assigned to a single cluster, the
29 cluster means are just the means of the assigned observations. The process of repeatedly assigning
30 observations to the nearest cluster, and recomputing cluster locations by averaging the assigned
31 observations is exactly Lloyd's algorithm for k -means clustering.

32 To avoid having to specify the number of clusters k , [KJ12b] considered a Dirichlet process mixture
33 of Gaussians, with all infinite components again having the same known variance σ^2 . The base
34 measure H from which the component means are drawn was set to a zero-mean Gaussian with
35 variance ρ^2 , with α the concentration parameter. The authors then considered a Gibbs sampler for
36 this model, very closely related to the sampler in Algorithm 33, except that instead of collapsing or
37 marginalizing out the cluster parameters, these are instantiated. Thus, an observation x_i as assigned
38 to a cluster c with mean μ_c with probability proportional to $N_{c,-i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2} \|x_i - \mu_c\|^2)$, while
39 it is assigned to a new cluster with probability proportional to $\alpha \frac{1}{\sqrt{2\pi(\sigma^2 + \rho^2)}} \exp(-\frac{1}{2(\sigma^2 + \rho^2)} \|x_i\|^2)$.
40 After cycling through all observations, one then resamples the parameters of each cluster. Following
41 the Gaussian base measure and Gaussian likelihood, this too follows a Gaussian distribution, whose
42 mean is a convex combination of the prior mean 0 and data-driven term, specifically the average of
43 the assigned observations. The weight of the prior term is proportional to the inverse of the prior
44 variance ρ^2 , while the weight of the likelihood term is proportional to the inverse of the likelihood
45 variance σ^2 .

¹ To derive a small-variance limit of the sampler above, we cannot just let σ^2 go to 0, as that
² would result in each observation being assigned to its own cluster. To prevent the likelihood from
³ dominating the prior in this way, one must also send the concentration parameter α to 0, so that the
⁴ DP prior enforces an increasingly strong penalty on a large number of clusters (recall that *a priori*,
⁵ the average number of clusters underlying N observations is $\alpha \log N$). [KJ12b] showed that with α
⁶ scaled as $\alpha = (1 + \rho^2/\sigma^2)^{d/2} \exp(-\frac{\lambda}{2\sigma^2})$ for some parameter λ , and taking the limit $\sigma^2 \rightarrow 0$ with ρ
⁷ fixed, we get the following modification of the k -means algorithm:
⁸

- ⁹ 1. Assign each observation to the nearest cluster, *unless the distance to the nearest cluster exceeds λ ,*
¹⁰ *in which case assign the observation to its own cluster.*
- ¹¹ 2. Set the cluster means equal to the average of the assigned observations.

¹²¹³¹⁴¹⁵ **Algorithm 34:** DP-means hard clustering algorithm for data $\mathcal{D} = \{x_1, \dots, x_N\}$ ¹⁶¹⁷ 1 Initialize the number of clusters $K = 1$, with cluster parameter μ_1 equal to the global mean:

¹⁸
$$\mu_1 = \frac{1}{N} \sum_{i=1}^N x_i;$$

¹⁹ 2 Initialize all cluster assignment indicators $z_i = 1$;²⁰ 3 **while** all z_i s have not converged **do**²¹ **for** each $i = 1 : N$ **do**²² Compute distance $d_{ik} = \|x_i - \mu_k\|^2$ to cluster k for $k = 1, \dots, K$;²³ **if** $\min_k d_{ik} > \lambda$ **then**²⁴ Increase the number of clusters K by 1: $K = K + 1$;²⁵ Assign observation i to this cluster: $z_i = K$ and $\mu_K = x_i$;²⁶ **else**²⁷ Set $z_i = \arg \min_c d_{ik}$;²⁸ **for** each $k = 1 : K$ **do**²⁹ Set \mathcal{D}_k be the observations assigned to cluster k . Compute cluster mean

³⁰
$$\mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x \in \mathcal{D}_k} x;$$

³¹³²³³

³⁴ Like k -means, this is a hard-clustering algorithm, except that instead of having to specify the
³⁵ number of clusters k , one just specifies a penalty λ for introducing a new cluster. The actual number
³⁶ of clusters is determined by the data, [KJ12b] refer to this algorithm as DP-means. One can show that
³⁷ the iterates above converge monotonically to a local maximum of the following objective function:
³⁸

$$\sum_{k=1}^K \sum_{x \in \mathcal{D}_k} \|x - \mu_k\|^2 + \lambda K, \quad \text{where } \mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x \in \mathcal{D}_k} x. \quad (33.43)$$

³⁹ The first term in the expression above is exactly the objective function of the k -means algorithm with
⁴⁰ K clusters. The second term is a penalty term that introduces a cost λ for each additional cluster.
⁴¹ Interestingly, the penalty term above corresponds to the so called Akaike Information Criterion
⁴² (AIC), a well studied approach to penalizing model complexity.

⁴³

It is also possible to derive hard-clustering algorithms for simultaneously clustering multiple datasets, while allowing these to share clusters. This is possible through the small-variance limit of a Gibbs sampler for the hierarchical Dirichlet process (HDP) and results in a clustering algorithm that now has two thresholding parameters, a local one λ_l and a global one λ_g . The algorithm proceeds by maintaining a set of global clusters, with the local clusters of each dataset assigned to a subset of the global clusters. It then repeats the following steps until convergence:

Assign observations to local clusters For x_{ij} , the i th datapoint in dataset j , compute the distance to all global clusters. For those global clusters not currently present in dataset j , add λ_l to their distance; this reflects the cost of introducing a new cluster into dataset j . Now, assign x_{ij} to the cluster with the smallest distance, unless the smallest distance exceeds $\lambda_g + \lambda_l$, in which case, create a new global cluster and assign x_{ij} to it. Observe that in the latter case, the distance of x_{ij} to the new cluster is 0, with $\lambda_g + \lambda_l$ reflecting the cost of introducing a new global and then local cluster.

Assign local clusters to global clusters For each local cluster l , compute the sum of the distances of all its assigned observations to the cluster mean. Call this d_l . Also compute the sum of the distances of the assigned observations to each global cluster. For global cluster p , call this $d_{l,p}$. Then assign local cluster l to the global cluster with the smallest $d_{l,p}$, unless $\min d_{l,p} > d_l + \lambda_g$ in which case we create a new global cluster.

Recompute global cluster means Set the global cluster means equal to the average of the assigned observations across all datasets.

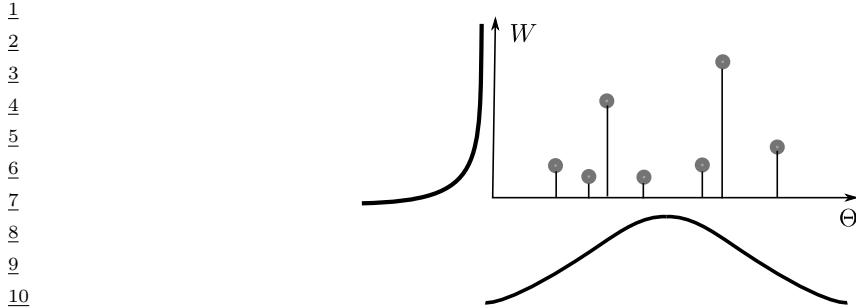
In [JKJ12], these ideas were extended from DP mixtures of Gaussians to DP mixtures of more general exponential family distributions. Briefly, the hard-clustering algorithms maintained the same structure, the only difference being that distance from clusters was measured using a **Bregman divergence** specific to that exponential family distribution. For Gaussians, the Bregman divergence reduces to the usual Euclidean distance.

In [Bkj13], the authors showed how such small-variance algorithms could be derived directly from a probabilistic model, and independent of any specific computational algorithm such as EM or Gibbs sampling. Their approach involves computing the MAP solution of the parameters of the model, and then taking the small-variance limit to obtain an objective function. This approach, called MAP-based Asymptotic Derivations from Bayes or MAD-Bayes allowed them to derive, among other things, an analog of the DP-means algorithm from feature-based models. They called this the BP-means algorithm, after the Beta process underlying the Indian Buffet process.

Write X for an $N \times D$ matrix of N D -dimensional observations, Z for an $N \times K$ matrix of binary feature assignments, and A for a $K \times D$ matrix of K D -dimensional features, with one seeking a pair (A, Z) such that ZA approximates X as well as possible. [Bkj13] showed in the small-variance limit, finding the MAP solution for the IBP is equivalent to the following problem:

$$\operatorname{argmin}_{K, Z, A} \text{trace}[(X - ZA)^t(X - ZA)] + \lambda K, \quad (33.44)$$

where again λ is a parameter of the algorithm, governing how the concentration parameter scales with the variance, and specifying a penalty on introducing new features. This objective function is very intuitive: the first term corresponds to the approximation error from K features, while the second term penalizes model complexity resulting from a large number of features K . This objective function can be optimized greedily by repeating three steps for each observation i until convergence:



¹¹ ¹² ¹³ ¹⁴ ¹⁵ ¹⁶ ¹⁷ ¹⁸ ¹⁹ ²⁰ ²¹ ²² ²³ ²⁴ ²⁵ ²⁶ ²⁷ ²⁸ ²⁹ ³⁰ ³¹ ³² ³³ ³⁴ ³⁵ ³⁶ ³⁷ ³⁸ ³⁹ ⁴⁰ ⁴¹ ⁴² ⁴³ ⁴⁴ ⁴⁵ ⁴⁶ ⁴⁷

Figure 33.10: Poisson process construction of a completely random measure $G = \sum_i w_i \delta_{\theta_i}$. The set of pairs $\{(w_1, \theta_1), (w_2, \theta_2), \dots\}$ is a realization from a Poisson process with intensity $\lambda(\theta, w) = H(\theta)\gamma(w)$.

1. Given A and K , compute the optimal value of the binary feature assignment vector z_i of observation i and update Z .
2. Given Z and A , introduce an additional feature vector equal to the residual for the i th observation, namely, $x_i - z_i A$. Call A' the updated feature matrix, and Z' the updated feature assignment matrix, where only observation i has been assigned this feature. If the configuration $(K+1, Z', A')$ results in a lower value of the objective function than (K, Z, A) , set the former as the new configuration. In words, if the benefit of introducing a new feature outweighs the penalty λ , then do so.
3. Update the feature vectors A given the feature assignment vectors Z .

[Bkj13] showed that this algorithm monotonically decreases the objective in Equation (33.44), converging eventually to a local optima. Subsequent works have considered the small-variance asymptotics of more structured models, such as topic models, hidden Markov models and even continuous-time stochastic process models.

33.6 Completely random measures

The Dirichlet process is a example of a random probability measure, a class of random measures which always integrate to 1. The Beta process, while not an RPM, belongs to another class of random measures: **completely random measures** (CRM). A completely random measure G satisfies the following property: for any two disjoint subsets T_1 and T_2 of Θ , the values $G(T_1)$ and $G(T_2)$ are independent random variables:

$$G(T_1) \perp\!\!\!\perp G(T_2) \quad \forall T_1, T_2 \subset \Theta \text{ s.t. } T_1 \cap T_2 = \emptyset. \quad (33.45)$$

Note that the Dirichlet process is not a CRM, since for disjoint sets T and its complement $T^c = \Theta \setminus T$, we have $G(T) = 1 - G(T^c)$, which is as far from independent as can be. More generally, under the DP, for disjoint sets T_1 and T_2 , the measures $G(T_1)$ and $G(T_2)$ are negatively correlated. We will see later though that the Dirichlet process is closely related to another CRM, the Gamma process. As a side point, beyond the sum-to-one constraint, $G(T_1)$ does not tell us anything about the distribution of probability within $G(T_1^c)$, making the DP what is known as a **neutral process**.

The simplest example of a CRM is the **Poisson process** (see Section 33.8). A Poisson process with intensity $\lambda(\theta)$ is a **point process** producing points or events in Θ , with the number of points in any set T following a $\text{Poisson}(\int_T \lambda(\theta)d\theta)$ -distribution, and with the counts in any two disjoint sets independent of each other. While it is common to think of this as a point process, one can also think of a realization from a Poisson process as an **integer-valued random measure**, where the measure of any set is the number of points falling within that set. It is clear then that the Poisson process is an example of a CRM.

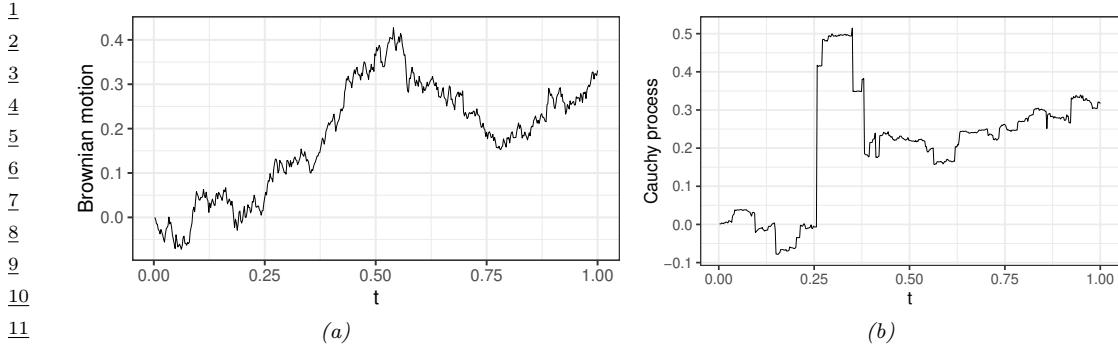
It turns out that the Poisson process underlies all completely random measures in a fundamental way. For some space Θ , and \mathbb{W} the positive real line, simulate a Poisson process on the product space $W \times \Theta$ with intensity $\lambda(\theta, w)$. Figure 33.10 shows a realization from this Poisson process, write it as $M = \{(\theta_1, w_1), \dots, (\theta_{|M|}, w_{|M|})\}$ where $|M|$ is the number of events. This can be used to construct an atomic measure $G = \sum_{i=1}^{|M|} w_i \delta_{\theta_i}$ as illustrated in Figure 33.10. From its Poisson construction, this is a completely random measure on Θ , with set $T \in \Theta$ having measure $\sum_i w_i \mathbb{I}(\theta_i \in T)$. Different settings of $\lambda(\theta, w)$ give rise to different CRMs, and in fact, other than CRMs with atoms at some fixed locations in Θ , this construction characterizes *all* CRMs.

For a CRM, the Poisson intensity is typically chosen to factor as $\lambda(\theta, w) = H(\theta)\gamma(w)$, with $\int_\Theta H(\theta)d\theta = 1$. Then, $H(\theta)$ is the base measure controlling the locations of the atoms in the CRM, while the measure $\gamma(w)$ controls the number of atoms, and the distribution of their weights. Setting $\gamma(w) = w^{-1}(1-w)^{\alpha-1}$ gives the Beta process with base measure $H(\theta)$. Other choices include the Gamma process ($\gamma(w) = \alpha w^{-1} \exp(-w)$), the stable process ($\gamma(w) = \frac{1}{\Gamma(1+\sigma)} \alpha w^{-1-\sigma}$), and the generalized Gamma process ($\gamma(w) = \frac{1}{\Gamma(1+\sigma)} \alpha w^{-1-\sigma} \exp(-\zeta w)$).

For all three processes described earlier, the $\gamma(w)$ integrates to infinity, so that $\int_\Theta \int_W \lambda(\theta, w)d\theta dw = \infty$. Consequently, the number of Poisson events, and thus the number of atoms in the CRMs are infinite with probability one. At the same time, mass of the $\gamma(w)$ function is mostly concentrated around 0 (see Figure 33.10), and for any $\epsilon > 0$, $\int_\epsilon^\infty \gamma(w)dw$ is finite. It is easy to show that the sum of the w 's is finite almost surely. Call this sum W , then the first condition ensures W greater than 0, while the second ensures it is finite. These two conditions make it sensible to divide a realization of a CRM by its sum, resulting in a random measure that integrates to 1: a random probability measure. Such RPMs are called **normalized completely random measures**, or sometimes just **normalized random measures (NRMs)**. The Dirichlet process we saw earlier is an example of an NRM: it is a normalized Gamma process. This result mirrors the situation with the finite Dirichlet distribution: one can simulate from a d -dimensional $\text{Dir}(\alpha_1, \dots, \alpha_d)$ distribution by first simulating d independent Gamma variables $g_i \sim \text{Ga}(\alpha_i, 1)$, $i = 1, \dots, d$, and then defining the probability vector $\frac{1}{\sum_{i=1}^d g_i} (g_1, \dots, g_d)$. The Pitman-Yor process is not an NRM except for special settings of its parameters: like we saw, $d = 0$ is a Dirichlet process or normalized Gamma process. $\alpha = 0$ is a **normalized stable process**. The normalized generalized Gamma process is an NRM that includes the DP and the normalized stable process as special cases.

33.7 Lévy processes

Completely random measures are also closely related to **Lévy processes** and **Lévy subordinators** [Ber96]. A Lévy process is a continuous-time stochastic process $\{L_t\}_{t \geq 0}$ taking values in some



13 *Figure 33.11: (a) A realization of a Brownian motion (b) A realization of Cauchy process (an α -stable process
14 with $\alpha = 1$).*

15

16

17 space (e.g. \mathbb{R}^d) that satisfies two properties¹:

18

19 **stationary increments:** for $t, \Delta > 0$, the random variable $L_{t+\Delta} - L_t$ does not depend on t

20

21 **independent increments:** for $t, \Delta > 0$, $L_{t+\Delta} - L_t$ is independent of values before t .

22 A Lévy subordinator is a real-valued, **nondecreasing** Lévy process. If we drop the stationarity
23 condition, it should be clear that the increments of a Lévy subordinator are exactly the atoms
24 of a completely random measure (see Figure 33.9). Common examples of Lévy subordinators are
25 Beta subordinators (or Beta processes), Gamma subordinators, stable subordinators and generalized
26 Gamma subordinators. CRMs generalize Lévy subordinators, by allowing them to be indexed by
27 some general space Θ (rather than by nonnegative reals), and by relaxing the stationarity condition
28 to allow the atoms to follow some general base measure H .

29 Unlike Lévy subordinators, a general Lévy process can have negative changes as well, with the
30 change $L_{t+\Delta} - L_t$ belonging to an **infinitely divisible distribution**, scaled by Δ . A random
31 variable X follows an infinitely divisible distribution if, for any positive integer N , there exists some
32 probability distribution such that the sum of N i.i.d. samples from that distribution has the same
33 distribution as X . Examples of infinitely divisible distributions include the Poisson distribution, the
34 Gamma distribution, the Cauchy distribution, the α -stable distribution, and the inverse-Gaussian
35 distribution (among many others), though by far the most well-known example is the Gaussian
36 distribution. It is easy to see why the change in a Lévy process $L_{t+\Delta} - L_t$ over some time interval
37 Δ follows an infinitely divisible distribution: just divide the interval into N equal-length segments.
38 From the properties of the Lévy process, the changes over these segments are independent and
39 identically distributed, and their sum equals $L_{t+\Delta} - L_t$. The **Lévy-Kintchine** formula shows that the
40 converse is also true: any infinitely divisible distribution represents the change of an associated Lévy
41 process. The Lévy process corresponding to the Gaussian distribution is the celebrated **Brownian**
42 **motion** (or **Weiner process**). Brownian motion is a fundamental and widely applied stochastic
43 process, whose mathematics was first studied by Louis Bachelier to model stock markets, and later,
44 famously, by Albert Einstein to argue about the existence of atoms. For a Brownian motion, the
45

46 1. There is also a technical continuity condition that we do not discuss here

47

increment $L_{t_1+\Delta} - L_{t_1}$ follows a normal $N(\mu\Delta, \sigma\Delta)$ distribution, where μ and σ are the *drift* and *diffusion* coefficients. Setting μ and σ to 0 and 1 gives **standard Brownian motion**. Paths sampled from the Weiner process are continuous with probability one, all other Lévy processes are jump processes. Figure 33.11 shows realizations from some processes. The jump processes have a Poisson construction related to the Lévy subordinators and completely random measures, and the **Lévy-Itô decomposition** shows that every Lévy process can be decomposed into Brownian and Poisson components. Levy processes have been widely applied in mathematical finance to model asset prices, insurance claims, stock prices and other financial assets.

33.8 Point processes with repulsion and reinforcement

In this section, we look more closely at the Poisson process, as well as other, more general point processes that allow inter-event interactions.

33.8.1 Poisson process

We have already briefly seen the Poisson process: this is a point process on some space Θ that is parametrized by an intensity function $\lambda(\theta) \geq 0$, and that produces a $\text{Poisson}(\int_T \lambda(\theta)d\theta)$ -distributed number of points of events in any set $T \in \Theta$, with the counts in any two disjoint sets independent of each other. Recall that if $N_i \sim \text{Poisson}(\lambda_i)$ are independent, then a well-known property of the Poisson distribution is that $N_1 + N_2 \sim \text{Poisson}(\lambda_1 + \lambda_2)$. This relates to the infinite divisibility of the Poisson distribution. It is clear that the average number of points is large in areas where $\lambda(\theta)$ is high, and small where $\lambda(\theta)$ is small. When the intensity $\lambda(\theta)$ is some constant λ , we have a **homogeneous Poisson process**, otherwise we have an **inhomogeneous Poisson process**.

Depending on whether $\int_\Theta \lambda(\theta)d\theta$ is infinite or finite, a Poisson process will either produce an infinite number of points (recall the Poisson process underlying the CRMs of Section 33.6) or a finite number of points. The latter is more common in applications, such as modeling phone calls or financial shocks (Θ is some finite time-interval), the locations of trees or forest fires (Θ is some subset of the Euclidean plane), the locations of cells or galaxies (Θ is a 3-dimensional space) or events in higher-dimensional spaces (for example, spatio-temporal activity). One way to simulate a finite Poisson process is to first simulate the number of total number of points N , which follows a $\text{Poisson}(\int_\Theta \lambda(\theta)d\theta)$ distribution. One can then simulate the locations of these points by sampling N times from the probability density $\frac{\lambda(\theta)}{\int_\Theta \lambda(\theta)d\theta}$. For a homogeneous Poisson process, the locations are uniformly distributed over Θ . One can also easily simulate a rate- λ homogeneous Poisson on the real line by exploiting the fact that inter-event times follow an exponential distribution with mean $1/\lambda$.

If the integral $\int_\Theta \lambda(\theta)d\theta$ is difficult to evaluate, one can also simulate a Poisson process using the **thinning theorem** [LS79]. Here, one needs to find a function $\gamma(\theta)$ such that $\gamma(\theta) \geq \lambda(\theta), \forall \theta$, and such that it is easy to simulate from a rate- $\gamma(\theta)$ Poisson process. Suppose the result is $\Psi = \{\psi_1, \dots, \psi_N\}$. Since $\gamma(\theta) \geq \lambda(\theta)$, Ψ is going to contain more events, and one *thins* Ψ by keeping each element ψ_i in Ψ with probability $\lambda(\psi_i)/\gamma(\psi_i)$ (otherwise one discards it). Once can show that the set of surviving points is then a realization of a Poisson process with rate $\lambda(\theta)$.

The defining feature of the Poisson process is the assumption of independence among events. In many settings, this is inappropriate and unrealistic, and the knowledge of an event at some location θ might suggest an reduced or elevated probability of events in neighboring areas. Point process

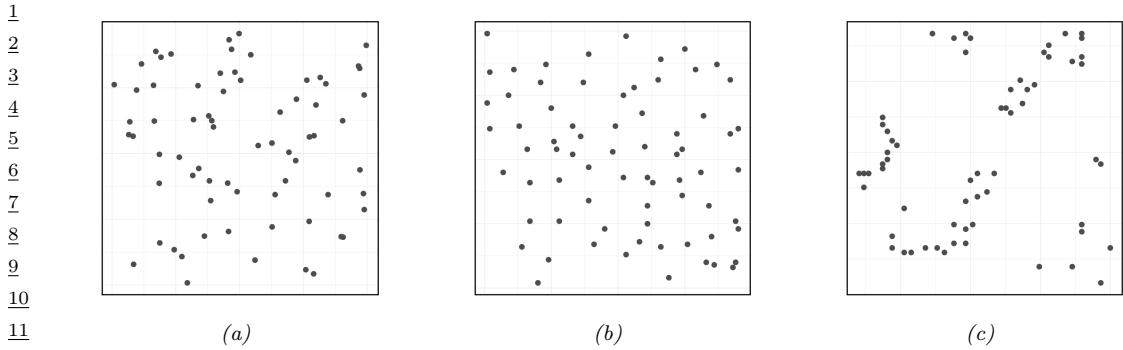


Figure 33.12: A realization of (a) a homogeneous Poisson process. (b) an underdispersed point process (Swedish pine sapling locations [Rip05]). (c) A realization of an overdispersed point process (California redwood tree locations [Rip77]).

satisfying the former property are called underdispersed (Figure 33.12(b)), while point processes satisfying the latter property are called overdispersed (Figure 33.12(c)). Examples of underdispersed point processes include the locations of trees or train stations, which tend to be more spread out than Poisson because of limited resources. An example of an overdispersed point process is earthquake locations, where aftershocks tend to occur in the vicinity of the main shock.

A simple approach to modeling overdispersed point processes is through hierarchical extensions of the Poisson process that allow the Poisson intensity function $\lambda(\cdot)$ to be a random variable. Such models are called doubly-stochastic Poisson processes or Cox processes [CI80], and a common approach is to model the intensity $\lambda(\cdot)$ via a Gaussian process. Note though that the Poisson process intensity function must be nonnegative, so that λ is often a transformed Gaussian process:

$$\lambda(\theta) = g(\ell(\theta)), \quad \ell(\theta) \sim \text{GP}. \quad (33.46)$$

Common examples for g include exponentiation, sigmoid transformation or just thresholding. Because of the smoothness of the unknown $\lambda(\cdot)$, observing an event at some location suggests events are likely in the neighborhood. Such models still do not capture direct interactions between point process events, rather, these are mediated through the unknown intensity functions, making them inappropriate for many applications. For instance, in neuroscience a neuron's spiking can be driven directly by past activity, and activity of other neurons in the network, rather than just through some shared stimulus $\lambda(t)$. Similarly, social media activity has a strong reciprocal component, where, for instance, emails sent out by a user might be in response to past activity, or activity of other users. The next few subsections show how one might explicitly model such interactions.

39

40 33.8.2 Renewal process

41 **Renewal processes** are one class of models of repulsion and reinforcement for point processes defined on the real line (typically regarded as time). Recall that for a homogeneous Poisson process, 44 inter-event times follow an exponential distribution. The exponential distribution has the property of 45 **memorylessness**, where the time until the next event is independent of how far in the past the last 46 event occurred. That is, if τ follows the exponential distribution, then for any δ and $\Delta > 0$, we have 47

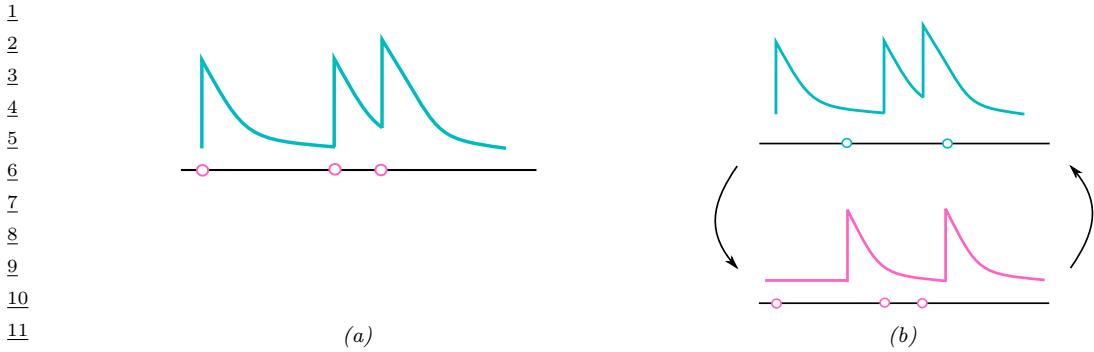


Figure 33.13: (a) A self-exciting Hawkes process. Each event causes a rise in the event rate, which can lead to burstiness. (b) A pair of mutually exciting Hawkes processes, events from each cause a rise in the event rate of the other.

$p(\tau > \Delta + \delta | \tau \geq \delta) = p(\tau > \Delta)$. Renewal processes incorporate memory by allowing the interevent times to follow some general distribution on the positive reals. Examples include Gamma renewal processes and Weibull renewal processes, where interevent times follow the Gamma and Weibull distribution respectively. Both these processes include parameter settings that recover the Poisson process, but also allow **burstiness** and **refractoriness**. Burstiness refers to the phenomenon where, after an event has just occurred, one is more likely to see more events than a longer time afterwards. This is useful for modeling email activity for instance. Refractoriness refers to the opposite situation, where an event occurring implies new events temporarily less likely to occur. This is useful to model neural spiking activity, for instance, where after spiking, a neuron is depleted of resources, and requires a recovery period before it can fire again.

33.8.3 Hawkes process

Hawkes processes [Haw71] are another class of reinforcing point processes that have attracted much recent attention. Hawkes processes provide an intuitive framework for modeling reinforcement in point processes, through self-excitation when a single point process is involved, and through mutual excitation (sometimes called reciprocity) when collections of point processes are under study. The former, shown in Figure 33.13(a), is relevant when modeling bursty phenomena like visits to a hospital by an individual, or purchases and sales of a particular stock. The latter, displayed in Figure 33.13(b) is useful to characterize activity on social or biological networks, such as email communications or neuronal spiking activity. In both examples, each event serves as a trigger for subsequent bursts of activity. This is achieved by letting $\lambda(t)$, the event rate at time t , be a function of past activity or *event history*. Write $\mathcal{H}(t) = \{t_k : t_k \leq t\}$ for the set of event times up to time t . Then the rate at time t , called the *conditional intensity function* at that time, is given by

$$\lambda(t|\mathcal{H}(t)) = \gamma + \sum_{t_k \in \mathcal{H}(t)} \phi(t - t_k), \quad (33.47)$$

where γ is called the **base-rate** and the function $\phi(\cdot)$ is called the **triggering kernel**. The latter characterizes the excitatory effect of a past event on the current event rate. Figure 33.13 shows

¹ the common situation where $\phi(\cdot)$ is an exponential kernel $\phi(\Delta) = \beta e^{-\Delta/\tau}$, $\Delta \geq 0$. Here, a new
² event causes a jump of magnitude β in the intensity $\lambda(t)$, with its excitatory influence decaying
³ exponentially back to γ , with τ the time-scale of the decay. For a multivariate Hawkes process, there
⁴ are m point processes $(N_1(t), N_2(t), \dots, N_m(t))$ associated with m users or nodes. Write $\mathcal{H}_i(t)$ for
⁵ the event history of the i th process at time t . Its conditional intensity can depend on all m event
⁶ histories, taking the form
⁷

$$\lambda_i(t|\{\mathcal{H}_j(t)\}_{j=1}^m) = \gamma_i + \sum_{j=1}^m \sum_{t_k \in H_j(t)} \phi_{ij}(t - t_k). \quad (33.48)$$

⁸
⁹ More typically, the conditional intensity of user i can depend only on the event histories of those
¹⁰ nodes ‘connected’ to it, with connectivity specified by a graph structure, and with $\phi_{ij}(\cdot) = 0$ if there
¹¹ is no edge linking i and j . Alternately, events can have marks indicating whom they are sent to (e.g.
¹² recipients of an email), with each event only updating the conditional intensities of its recipients.

¹³ Simulating from a Hawkes process is a fairly straightforward extension of simulating from an
¹⁴ inhomogeneous Poisson process. Consider the univariate Hawkes process, and suppose the last event
¹⁵ occurred at time t_i . Then, until the next event occurs, at any time $t > t_i$, we have that $\mathcal{H}(t) = \mathcal{H}(t_i)$,
¹⁶ so that the conditional intensity $\lambda(t|\mathcal{H}(t)) = \lambda(t|\mathcal{H}(t_i))$. The next event time, t_{i+1} , is then just the
¹⁷ time of the first event of a Poisson process with intensity $\lambda(t|\mathcal{H}(t_i))$ on the interval $[t_i, \infty)$. With most
¹⁸ choices of the kernel ϕ , t_{i+1} can easily be simulated, either by integrating $\lambda(t|\mathcal{H}(t_i))$, or by Poisson
¹⁹ thinning. At time t_{i+1} , the history is updated to incorporate the new event, and the conditional
²⁰ intensity experiences a jump. The updated intensity is used to simulate the next event at some time
²¹ $t_{i+2} > t_{i+1}$ from a rate- $\lambda(t|\mathcal{H}(t_{i+1}))$ Poisson process, and the process is repeated until the end of the
²² observation interval. For the case of multivariate Hawkes processes, one has a collection of competing
²³ intensities $\lambda_i(t|\{\mathcal{H}_j(t)\}_{j=1}^m)$. The next event is the first event among all events produced by these
²⁴ intensities, after which the intensities are updated and the process is repeated. A realization Ψ from
²⁵ a Hawkes process has log-likelihood
²⁶

$$\ell(\Psi) = \sum_{t^* \in \Psi} \log \lambda(t^*|\mathcal{H}(t^*)) - \int \lambda(t|\mathcal{H}(t)) dt. \quad (33.49)$$

²⁷
²⁸ This can typically be evaluated quite easily, so that maximum likelihood estimates of parameters like
²⁹ the base-rate as well as parameters of the excitation kernel can be obtained straightforwardly.

³⁰ The Hawkes process as described is a fairly simple model, and there have been a number of
³¹ extensions enriching its structure. An early example is [BBH12], where the authors considered the
³² multivariate Hawkes process, now with an underlying clustering structure. Instead of each individual
³³ point process having its own conditional intensity function, each cluster has an intensity function
³⁴ shared by all point process assigned to it. The interaction kernels are also defined at the cluster level,
³⁵ with an event in process i causing a jump $\phi_{c_i c}(\tau)$ in the intensity function of cluster c (where c_i is the
³⁶ cluster point process i belongs to). The cluster structure was modeled through a Dirichlet process,
³⁷ allowing the authors to learn the underlying clustering, as well as the inter-cluster interaction kernels
³⁸ from interaction data. In [Tan+16], the authors considered **marked** point processes, where event i at
³⁹ time t_i has some associated content y_i (for example, each event is a social media post, and y_i is the
⁴⁰ text associated with the post at time t_i). The authors allowed the jump in the conditional intensity
⁴¹ to depend on the associated mark, with the Hawkes kernel taking the form $\phi(\Delta) = f(y_i) \exp(-\Delta/\tau)$.
⁴²

In that work, the authors modeled the function f with a Gaussian process, though other approaches, such as ones based on neural networks, are possible.

The neural Hawkes process [ME17] is a more fleshed out approach to modeling point processes using neural networks. This models event intensities through the state of a continuous-time LSTM, a modification of the more standard discrete-time LSTMs from Section 16.3.3. Central to an LSTM is a memory cell \mathbf{c}_i to store long-term memory, summarizing the past until time step i . Continuous-time LSTMs include two long-term memory cells, \mathbf{c}_i and $\tilde{\mathbf{c}}_i$, summarizing the history until the i th Hawkes event. The first cell represents the starting value to which the intensity jumps after the i th event, and the second represents a baseline rate after the i event. These are both updated after each event, with $\mathbf{c}(t)$, the instantaneous rate at any intermediate time determined by \mathbf{c}_i decaying exponentially towards the baseline $\tilde{\mathbf{c}}_i$. This mechanism allows the intensity at any time to be influenced not just by the number of events in the past, but also the waiting times between them. It can be extended to marked point processes, where each event is also associated with a mark \mathbf{y} : now both a learned embedding of the mark as well as the time since the last event is used to update state and long-term memory. For more details, see also Du et al. [Du+16].

33.8.4 Gibbs point process

Gibbs point processes [MW07] from the statistical physics and spatial statistics literature provide a general framework for modeling interacting point processes on higher-dimensional spaces. Such spaces are more challenging than the real line, since there is now no ordering of points, and thus no natural notion of history affecting future activity. Instead, Gibbs point processes use an **energy function** E to quantify deviations from a Poisson process with rate 1. Specifically, under a Gibbs process, the probability density of any configuration Ψ with respect to a rate-1 Poisson process takes the form

$$P_\beta(\Psi) = \frac{1}{Z_\beta} \exp(-\beta E(\Psi)) \quad (33.50)$$

where β is the inverse-temperature parameter, and $Z_\beta = \mathbb{E}_\pi[\exp(-\beta E(\Psi))]$ is the normalization constant (the expectation is with respect to π , the unit-rate Poisson process). Under some conditions on the energy function E , the expectation Z_β is finite, and P_β is a well-defined density, whose integral with respect to π is 1. While the above equation resembles a Markov random field, the domain of Ψ is much more complicated, and evaluating Z_β now involves solving an infinite dimensional integral.

Equation (33.50) states that configurations Ψ for which $E(\Psi)$ is small are more likely than under a Poisson process, with β controlling how peaked this is. The most common energy functions are **pairwise potentials**, taking the form

$$E(\Psi) = \sum_{(s,s') \in \Psi} \phi(\|s - s'\|), \quad (33.51)$$

where the summation is over all pairs of events in Ψ , and $\phi : \mathbb{R}^+ \rightarrow \mathbb{R} \cup \infty$. The Strauss process is a specific example, with energy function specified by positive parameters a and R as

$$E(\Psi) = \sum_{s,s' \in \Psi} a \cdot \mathbb{I}(\|s - s'\| \leq R). \quad (33.52)$$

¹This is a repulsive process that penalizes configurations with events separated by distance less than
² R , and as $a \rightarrow \infty$ becomes a **hardcore repulsive process**, forbidding configurations with two
³points separated by less than R . More generally, the energy function can be piecewise-constant,
⁴parametrized by a collection of pairs $(a_1, R_1), \dots, (a_n, R_n)$:

$$\frac{6}{7} E(\Psi) = \sum_{s, s' \in \Psi} \sum_{i=1}^n a_i \cdot \mathbb{I}(\|s - s'\| \leq R_i). \quad (33.53)$$

⁹Another natural option is to use smooth functions like a squared exponential kernel. Gibbs point
¹⁰processes can also involve higher-order interactions, examples being Geyer's triplet point process
¹¹(which penalizes occurrences of 3 events that are all within some distance R), or area-interaction point
¹²processes (that center disks of radius R on each event, calculate the area of the union of these disks,
¹³and define E as some function of this area).

¹⁵While Gibbs point processes are flexible and interpretable point process models, the intractable
¹⁶normalization constant Z_β makes estimating parameters like β a formidable challenge. In practice,
¹⁷these models have to be fit using approximate approaches such as maximizing a pseudo-likelihood
¹⁸function (instead of Equation (33.50)).

¹⁹

²⁰²¹33.8.5 Determinantal point process

²²Determinantal point processes [Mac75; Bor; LMR15] or DPPs are another approach to modeling
²³repulsion, and have seen considerable popularity in the machine learning literature. Like any point
²⁴process, a DPP is a probability distribution over subsets of a fixed set \mathcal{S} , and in the DPP literature
²⁵this is often called the **ground set**. Point process applications typically have \mathcal{S} with uncountably
²⁶infinite cardinality (for example, \mathcal{S} could be the real line), although machine learning applications
²⁷of DPPs often focus on \mathcal{S} with a finite number of elements. For instance, \mathcal{S} could be a database of
²⁸images, a collection of news articles, or a group of individuals. A sample from a DPP is a random
²⁹subset of \mathcal{S} , produced, for instance, in response to a search query. The repulsive nature of DPPs
³⁰ensures diversity and parsimony in the returned subset. This could be useful, for example, to ensure
³¹responses to a search query are not minor variations of the same image. Another application is
³²clustering, where a DPP serves as a prior over the number of clusters and their locations, with the
³³repulsiveness discouraging redundant clusters that are very similar to each other.

³⁴DPPs are parametrized by a similarity kernel K , whose element K_{ij} gives the similarity between
³⁵elements i and j of the ground set. For finite ground sets, K is just a similarity matrix. DPPs require
³⁶ K to be positive definite, with largest eigenvalue less than 1, that is $0 \preceq K \preceq I$. For simplicity, K is
³⁷often assumed symmetric, though this is not necessary. Given a kernel K , the associated DPP is
³⁸defined as follows: if Y is the random subset of \mathcal{S} drawn from the DPP, then the probability that Y
³⁹contains any subset A of \mathcal{S} is given by

$$\frac{40}{41} p(A \subseteq Y) = \det(K_A). \quad (33.54)$$

⁴²Here K_A is the submatrix obtained by restricting K to rows and columns in A , and we define
⁴³ $\det(K_\phi) = 1$ for the empty set ϕ . Observe that this probability is specified exactly, and not just up
⁴⁴to a normalization constant. We immediately see that Y contains the empty set with probability one
⁴⁵(this is trivially true since the empty set is a subset of every set). We also see that the probability
⁴⁶

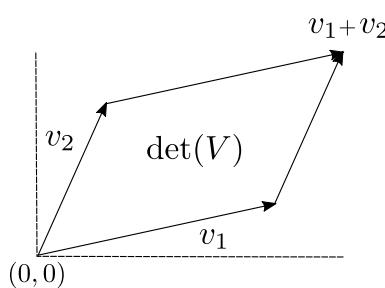


Figure 33.14: The determinant of a matrix V is the volume of the parallelogram spanned by the columns of V and the origin.

that element i is selected in Y is the i th diagonal element of K :

$$p(i \in Y) = \det(K_{ii}) = K_{ii}, \quad (33.55)$$

so that the diagonal of K gives the inclusion probabilities of the individual elements in \mathcal{S} . More interestingly, the probability a pair of elements $\{i, j\}$ are both contained in Y is given by

$$p(\{i, j\} \subseteq Y) = K_{ii}K_{jj} - K_{ij}^2. \quad (33.56)$$

The first term $K_{ii}K_{jj}$ is the probability of including i and j if they were independent, and this probability is adjusted by subtracting K_{ij}^2 , a measure of similarity between i and j . This is the source of repulsiveness or diversity in DPPs. More generally, the determinant of a set of vectors is the volume of the parallelogram spanned by them and the origin (see Figure 33.14), and by making the probability of inclusion proportional to the volume, DPPs encourage diversity. We mention for completeness that for uncountable ground sets like a Euclidean space, the determinant $\det(K_A)$ now gives the **product density function** of a realization A . This generalizes the intensity of a Poisson process to account for interactions between point process events, and we refer the interested reader to Lavancier, Møller, and Rubak [LMR15] for more details.

Equation (33.54) characterizes the marginal probabilities of a determinantal point process: what is the probability that a realization Y *contains* some subset of \mathcal{S} . More often, one is interested in the probability that the realization Y *equals* some subset of \mathcal{S} . The latter is of interest for simulation, inference and parameter learning with DPPs. For this, it is typical to work with a narrower class of DPPs called **L-ensembles** [Bor; KT+12]. Like general DPPs, such a process is characterized by a positive semidefinite matrix L , with the probability that Y equals some configuration A given by:

$$p(Y = A) \propto \det(L_A). \quad (33.57)$$

Note that this probability is specified only upto a normalization constant, and so we do not need to upperbound eigenvalues of L . In fact, it is not hard to show that the normalizer is given by $(I + \det(L))$, so that

$$p(Y = A) = \frac{\det(L_A)}{I + \det(L)}. \quad (33.58)$$

1 A similar calculation can be used to show that $p(A \subseteq Y) = \det(K)$, where $K = L(I + L)^{-1} =$
2 $I - (I + L)^{-1}$, showing that L-ensembles are indeed special kind of DPP. Equation (33.58) and
3 Equation (33.54) allow parameters of the DPP to estimated from realizations of a point process,
4 typically by gradient descent. Without additional structure, naively calculating determinants is
5 cubic in the cardinality of \mathcal{S} , and this represents a substantial saving when one considers the number
6 of possible subsets of \mathcal{S} . When even cubic scaling is too expensive, a number of approximation
7 approaches can be adopted, and these are often closely related to approaches to solve the cubic cost
8 of Gaussian processes.
9

10 So far, we have only discussed how to calculate the probability of samples from a DPP. Simulating
11 from a DPP, while straightforward, is a bit less intuitive, and we refer the reader to [LMR15; KT+12].
12 At a high level, these approaches use the eigenstructure of K to express a DPP as a mixture of
13 **determinantal projection point processes**. The latter are DPPs whose similarity kernel has
14 binary eigenvalues (either 0 or 1) and are easier to sample from. Observe that any eigenvalue λ_i of
15 the similarity kernel K must lie in the interval $[0, 1]$. This allows us to generate a random similarity
16 kernel \hat{K} with the same eigenvectors as K but with binary eigenvalues as follows: for each i , replace
17 eigenvalue λ_i of K with a binary variable $\hat{\lambda}_i$ simulated from a $\text{Bernoulli}(\lambda_i)$ distribution. One can
18 show that the DPP simulated from \hat{K} after simulating \hat{K} from K in this fashion is distributed
19 exactly as a DPP with kernel K . We refer the reader to [LMR15] for details on how to simulate a
20 determinantal projection point process with similarity kernel \hat{K} .

21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

34 Representation learning (Unfinished)

This chapter was written by Ben Poole and Simon Kornblith.

34.1 CLIP

35 Interpretability

This chapter was written by Been Kim and Finale Doshi-Velez.

35.1 Introduction

As machine learning models become increasingly commonplace, there exists increasing pressure to ensure that these models' behaviors align with our values and expectations. It is essential that models that automate even mundane tasks—from paperwork processing and serving information to flagging potential fraud—don't cause harms to their users or to society at large; for models that are used to inform potentially life-changing decisions, such as autonomous cars or health advice, the stakes are even higher.

However, while some harms—such as costs due to the model's false positive and false negative rate on training data—are easy to quantify—it is not easy to ensure that models are not dangerous, unfair, or too narrowly focused on a single objective in complex environments. Objectives can often be easily exploited (e.g., reward hacking). The value of any metric (whether it is the objective or a false negative rate) can be hard to estimate precisely in complex settings e.g., with limited data or a distribution shift. Modern machine learning methods—while they have accomplished much—still often learn shortcuts that do not generalize to new situations even as users start attributing notions like language or visual understanding to them [Gei+20b]. As a result, not only might unexpected, irreversible harms may occur (e.g., an incorrect medical procedure), but biases and more subtle safety issues may go unnoticed for long periods of time until sufficient reporting data accrues [Amo+16]. While it may not be possible to fully eliminate these harms, a broad, holistic approach to validation that includes expert inspection (via interpretability) and empirical analysis (e.g., more traditional statistical measures) will be critical in mitigating them.

Finally, even when we have satisfactory models for our tasks(e.g., models that make good predictions or compress the data well) we may be interested in understanding *why* they work to gain scientific and operational insights that go beyond the ML original task or to predict unseen failure cases. For example, one might gain insights in language acquisition and learning by asking why a language model performs so well; understanding why patient data cluster along particular axes may result in a better understanding of disease and the common treatment pathways. Ultimately, interpretation help humans to communicate better with machines to accomplish our tasks better in many metrics we care by communicating to or teaming up with machines.

In this chapter, we lay out the role and terminologies in interpretable ML before introducing methods, properties and evaluation of interpretability methods.

¹
² **35.1.1 The Role of Interpretability**

³

⁴ As noted above, ensuring that models have the behaviors that we want is a challenging task that
⁵ requires a holistic approach to design and validation. In some cases, the desired behavior can be
⁶ guaranteed by design, such as certain notions of privacy via differentially-private learning algorithms
⁷ or some chosen mathematical metric of fairness. In other cases, tracking various metrics, such
⁸ as adverse events or subgroup error rates, may be the appropriate and sufficient way to identify
⁹ concerns and demonstrate that they have been resolved. However, in many cases, the goal may be
¹⁰ fundamentally impossible to fully specify and thus formalize. In such cases, human inspection of the
¹¹ machine learning model may be necessary. Below we describe several examples.

¹²

¹³ **Blindspot Discovery.** Inspection may reveal **blindspots** in our modeling, objective, or data
¹⁴ [Bad+18; Zec+18; Gur+18] For example, suppose a company has trained a machine learning system
¹⁵ for credit scoring. The model was trained on a relatively affluent, middle-aged population, and now
¹⁶ the company is considering using it on a college population—a new population, on which they have
¹⁷ no data. Suppose that inspection of the model reveals that it relies heavily on the properties of the
¹⁸ applicant’s home and car. Not only might this suggest that the model might not transfer well to
¹⁹ the new college population, but it might encourage us to check for bias in the existing application
²⁰ because we know historical biases have prevented certain populations from achieving home ownership
²¹ (something that a purely quantitative definition of fairness may not be able to recognize). Indeed,
²² the most common application of interpretability in industry settings is for engineers to debug models
²³ and make deployment decisions [Pal].

²⁴

²⁵ **Novel Insights.** Inspection may catalyze **novel insights**. For example, suppose an algorithm
²⁶ determines that surgical procedures fall into three clusters, and the surgeries in one of the clusters
²⁷ of patients seem to consistently take longer than expected. A human inspecting these clusters may
²⁸ determine that a key factor in the cluster with the worst delays is that those surgeries happen in a
²⁹ more distant part of the hospital (a feature not in the original dataset), and then hypothesize that
³⁰ transit time for clinical staff may be affecting performance.

³¹

³² **Human+ML Teaming.** Inspectability may empower effective **human+ML interaction and**
³³ **teaming**. For example, suppose an anxiety treatment recommendation algorithm reveals that one of
³⁴ the key factors determining the recommendation was that the patient has insomnia. The patient
³⁵ reports that they no longer have trouble sleeping. Then they could re-run the algorithm with that
³⁶ input changed to get a better recommendation. More broadly, if a human can provide feedback to
³⁷ the model (e.g., correcting an incorrect input or assumption) or if they can incorporate information
³⁸ from the ML model into their own decision-making (e.g., having an accurate sense of the risk of a
³⁹ poor outcome, while trying to optimize for a good one) the human+ML team may be able to produce
⁴⁰ better combined performance than either alone (e.g. [Ame+19; Kam16]).

⁴¹

⁴² **Individual-Level Recourse.** A common setting under the law is that one needs to demonstrate
⁴³ that a specific harm or error happened in a specific context. A local explanation can help provide
⁴⁴ information needed for an individual to seek recourse. For example, if a loan applicant knows what
⁴⁵ features were used to deny them a loan, they have a starting point to argue that an error might have
⁴⁶ been made, or that the algorithm denied them unjustly. For this reason, inspectability is sometimes
⁴⁷

¹
² a legal requirement [Zer+19; GF17; Cou16].
³

⁴ As we look at the examples above, we see that one common element is that **interpretability**
⁵ **is needed when we need to combine human insights with the ML algorithm to get to**
⁶ **the ultimate goal.**¹ However, looking at the list above also emphasizes that beyond this very
⁷ basic commonality, **each application and task represents very different needs**. One would
⁸ not expect a scientist seeking to glean insights from a clustering on molecules (a case in which we
⁹ want to understand global patterns—such as all molecules with certain loop structures are more
¹⁰ stable—and are not under time pressure) to require the same kind of information as a clinician
¹¹ seeking to make a specific treatment decision for a specific patient (a case in which we need to make
¹² a local decision and are under time pressure). This brings us to our most important point: The best
¹³ form of explanation depends on the context; interpretability is a means to an end.
¹⁴

¹⁵ 35.1.2 Terminology and Framework

¹⁶ In broad strokes, “to interpret means to explain or present in understandable terms,”[Mer] [to a
¹⁷ human], and understanding involves an alignment of mental models. In interpretable machine
¹⁸ learning, that alignment is between what (perhaps part of) the machine learning model is doing and
¹⁹ what the user thinks the model is doing.
²⁰

²¹ Specifically, not only does the interpretable machine learning ecosystem include standard machine
²² learning (e.g., a prediction task), it also crucially includes what information is provided to the
²³ human user, in what context, and the user’s ultimate goal. The broader *socio-technical system*—the
²⁴ collection of interactions between human, social, organizational, and technical (hardware and software)
²⁵ factors—in which the machine learning system is situated cannot be ignored [Sel+19]. The goal
²⁶ of interpretable machine learning is to help a user do *their* task, with *their* cognitive strengths
²⁷ and weaknesses, with *their* focus and distractions [Mil19a]. Below we define the key terms of this
²⁸ expanded ecosystem and describe how they relate to each other. Before continuing, however, we note
²⁹ that the field of interpretable machine learning is relatively new, and a consensus around terminology
³⁰ is still evolving. Thus, it is always important to be precise about what one means in one’s usage.
³¹

³² Two key **social** or **human-factors** concepts in interpretable machine learning are the *context* and
the *end-task*.

³³ **Context.** We use the term *context* to describe the setting in which an interpretable machine
³⁴ learning system will be used. What is the setting? Who is the user? What information do they
³⁵ already have about the setting? What constraints are present on their time, cognition, or attention?
³⁶ We will use the terms *context* and *application* interchangeably [Sta].
³⁷

³⁸ **End-task.** We use the term *end-task* to refer to the user’s ultimate goal. What are they ultimately
³⁹ trying to achieve? Are they trying to help human experts to be more efficient or to do a recourse?
⁴⁰ We use *end-task* and *downstream task* interchangeably in this chapter.
⁴¹

⁴² Three key **technical** concepts in interpretable machine learning are the *method*, the *metrics*, and
⁴³ the *properties* of the methods.
⁴⁴

⁴⁵ 1. We emphasize that interpretability is different from manipulation or persuasion: interpretability assumes that the
⁴⁶ model will not intentionally deceive nor aim to convince users with a goal in mind.
⁴⁷

1

2

3 **Method.** What is the *method* used to provide interpretability? We use the term *explanation*
4 to mean whatever is provided by the machine learning system to the user—that is, *interpretable*
5 *machine learning* involves *providing explanations*. If the explanation is the model itself, we call the
6 method *inherently interpretable* or *interpretable by design*. In other cases, the model may be too
7 complex for a human to inspect in the required context: perhaps it is a large neural network that
8 no human could expect to understand; perhaps it is a medium-sized decision tree that could be
9 inspected if one had twenty minutes but not if one needs to make a decision in two. In such cases,
10 the explanation may be a *partial view* into the model, one that is ideally suited for performing the
11 end-task in the given context. We emphasize that even inherently interpretable models do not reveal
12 everything: one might be able to fully inspect the function (e.g. a two-node decision tree) but not
13 know what data it was trained on or which data points were most influential.

14

15 **Metrics.** How is the interpretability method evaluated? Evaluation is one of the most essential
16 and challenging aspects of interpretable machine learning, because we are interested in the **End-task**
17 performance of the *human*, when explanation is provided. We call this the *downstream performance*.
18 Just as different goals in ML require different metrics (e.g., positive predictive value, log likelihood,
19 AUC), different **contexts** and **end-tasks** will have different metrics. For instance, the model with
20 the best predictive performance (on some standard ML loss, such as log likelihood) may not be the
21 model that results in the best downstream performance.

22

23 **Properties.** What characteristics do the explanation have in relation to the model and the
24 end-tasks? For example, an explanation might have the property that it correctly describes of the
25 impact of modifying an input on the predictions, but only for models of low curvature (that is,
26 models that are relatively slowly varying). Different **contexts** and different **end-tasks** might require
27 different properties. For example, in a credit risk assessment scenario, the above property about
28 correctly describing the impact of modifying inputs might be critical for checking for effects of bias or
29 errors. Some other scenarios may require highly sparsity in explanations. In this way, properties serve
30 as a glue between interpretability methods and end-tasks: properties allow us to specify and quantify
31 aspects relevant to our ultimate end-task goals, and then we can make sure that our interpretability
32 method has those properties.

33

34 **How they all relate.** Formulating an interpretable machine learning problem generally starts by
35 specifying the context and the end-task. Together the context and the end-task imply what metrics
36 are appropriate to evaluate the downstream performance on the end-task and, ideally, suggest what
37 properties will be important in the explanation. Meanwhile, the context also determines the data
38 and training metric to train the ML model. The appropriate choice of explanation methods will
39 depend on the model and properties desired, and it will be evaluated with respect to the end-task
40 metric to determine the downstream performance. Figure 35.1 shows these relationships.

41 While there are many challenges in interpretable machine learning, including computing expla-
42 nations and optimizing interpretable models, creating explanations with certain properties, and
43 understanding the associated human factors, a grand challenge in interpretable machine learning is to
44 1) develop a general understanding of what properties are needed for different contexts and end-tasks,
45 and 2) identify and create interpretable machine learning methods that have those properties.

46

47

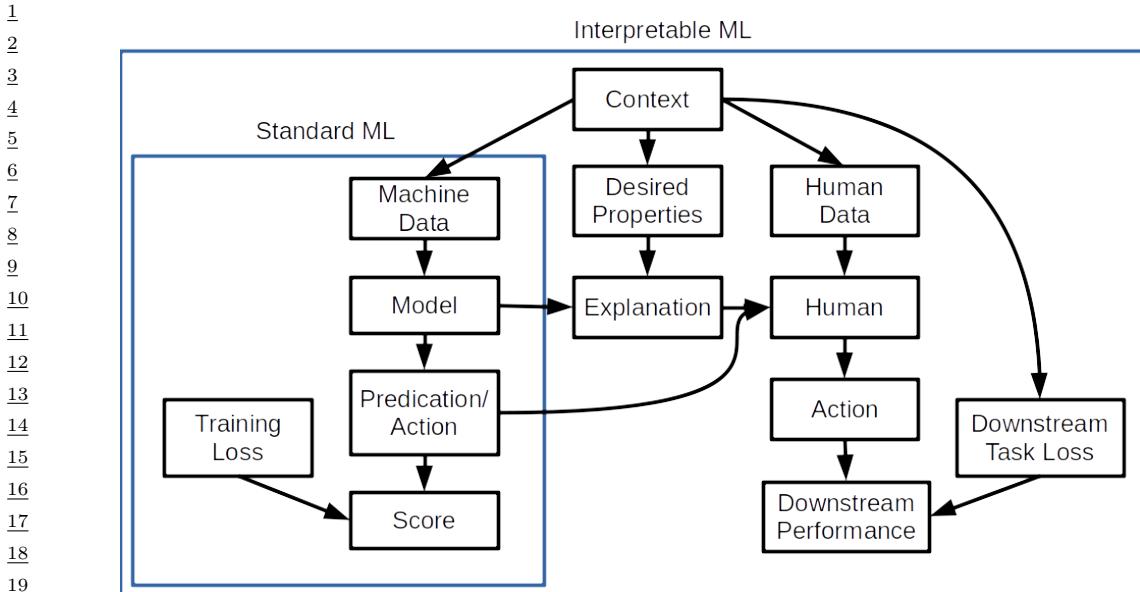


Figure 35.1: The Interpretable Machine Learning ecosystem. While standard machine learning can often abstract away elements of the context and consider only the the process of learning models given a data distribution and a loss, interpretable machine is inextricably tied to a socio-technical context.

A Simple Example. In the following sections, we will expand upon methods for interpretability, metrics for evaluation, and types of properties. First, however, we provide a simple example connecting all of the concepts we discussed above.

Suppose our context is that we have a lemonade stand, and our end-task is to understand when the stand is most successful in order to prioritize which days it is worth setting it up. (We have heard that sometimes machine learning models latch on to incorrect mechanisms and want to check before using the model to guide our lemonade stand business strategy.) Our metric for the downstream performance is whether we make the correct decision about whether to use this model to decide when to open our lemonade stand; this could be quantified as the amount of profit that we make by opening on busy days and being closed on quiet days.

To train our model, we collect data on two input features—the average temperature for the day (measured in degrees Fahrenheit) and the cleanliness of the sidewalk near our stand (measured as a proportion of the sidewalk that is free of litter, between 0 and 1)—and the output feature of whether the day was profitable. We realize that two models seem to fit the data approximately equally well:

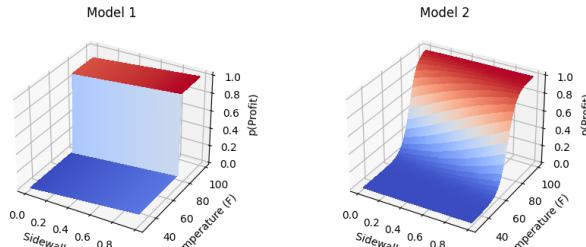
Model 1:

$$p(\text{profit}) = .9 * (\text{temperature} > 75) + .1(\text{howCleanSidewalk}) \quad (35.1)$$

Model 2:

$$p(\text{profit}) = \sigma(.9(\text{temperature} - 75)/\text{maxTemperature} + .1(\text{howCleanSidewalk} - .5)) \quad (35.2)$$

1
2
3
4
5
6
7
8
9
10



11 *Figure 35.2: Models described in the simple example. Both of these models have the same qualitative*
12 *characteristics, but different explanation methods will describe these models quite differently—and in ways*
13 *that could potentially confuse someone unfamiliar with the explanation is computed.*

14
15
16

17 These models are illustrated in Figure 35.2. Both of these models are inherently interpretable in
18 the sense that they are easy to inspect and understand. Both also rely mostly on the temperature,
19 which seems reasonable. Also note that these are not causal models; while causal model would work
20 for our end-task of determining our business strategy, we decided that we are not confident that
21 there will not be unknown confounding factors (common assumption in causal models).

22 For the sake of this example, suppose that the models above were blackboxes, and we could only
23 request partial views of it. We decide to ask the model for the most important features. Let us see
24 what happens when we consider two different ways of computing important features.²

25 Our first (feature-based) explanation method computes, for each training point, whether individually
26 changing each feature to its max or min changes the prediction. Important features are those that
27 change the prediction for many training points. One can think of this explanation method as a
28 variant of computing feature importance based on how important a feature is to the coalition that
29 produces the output (i.e., prediction). In this case, both models will report temperature to be the
30 dominating feature. If we used this explanation, we would correctly conclude that both models use
31 the features in a sensible way (and thus may be worth considering for deciding when to open our
32 lemonade stand).

33 Our second (feature-based) explanation method computes, for each training point, the magnitude
34 of the derivatives of the output with respect to the inputs. Important features are those that have a
35 large sum of absolute derivatives across the training set. One can think of this explanation method
36 as a variant of computing feature importances based on local geometry. In this case, Model 2 will
37 still report that temperature has higher derivatives. However, Model 1, which has very similar
38 behavior to Model 2, will report that sidewalk cleanliness is the dominating feature because the
39 derivative with respect to temperature is zero nearly everywhere. If we used this explanation to
40 vet our models, we would incorrectly conclude that Model 1 over-relied on a feature we believe is
41 relatively unimportant—and more generally, incorrectly conclude that Model 1 and Model 2 rely on
42 different features.

43 What happened? The key is that it's not about a method being right or wrong; it's about whether
44 the property of a chosen method is right or wrong for the context and the end-task. Here, the first
45

46 2. In the remainder of the chapter we will describe many other ways of creating and computing explanations.

47

1 explanation had the property of fidelity with respect to identifying features that, if changed, will affect
2 the prediction, whereas the second explanation had the property of correctly identifying features
3 have the most curvature. In this example, since our goal is to determine our business strategy, the
4 property that affects the prediction when feature is modified (first method) was a better fit than
5 measuring sensitivity of the feature (the second method). (Note: Other properties in addition to
6 fidelity may be important for this end-task. This example is just a simplest one.)
7

8 35.2 Methods for Interpretable Machine Learning

9 There exist many methods for interpretable machine learning. Each method has different properties
10 and the right choice will depend on context and end-tasks; as we noted in Section 35.1.2, the grand
11 challenge in interpretable machine learning is determining what kinds of properties are needed for
12 what contexts, and what explanation methods satisfy those properties. Thus, one should consider
13 this section a high-level snapshot of the rapidly changing options of methods that one may want to
14 choose for interpretable machine learning.
15

16 35.2.1 Inherently Interpretable Models: The Model is its Explanation

17 There are certain classes of models that we can consider inherently interpretable: a person can inspect
18 the full model and its internals in the finest granularity, and with reasonable effort, understand how
19 inputs—typically input features—become outputs.³ Specifically, we define inherently interpretable
20 models as those that require no additional process or proxies in order for them to be used as
21 explanation for the end-task. For example, a model using sparse rules for prediction may itself (i.e.,
22 rules) be the explanation without any extra processing. However, the "inherence"—the absence of any
23 additional process or proxies between the model and the user—alone does not guarantee improvement
24 of downstream tasks—inherently interpretable models must still be evaluated to determine whether
25 they help the user achieve their end-task.

26 Inherently interpretable models fall into two main categories: sparse (or otherwise compact) models
27 and logic-based models.

28 **Compact** or **sparse** feature-based models include various kinds of sparse regressions. Basic models
29 in this category include LASSO or other L1-regularized regressions. More advanced models in this
30 category include super-sparse linear integer models and other checklist models [DMV15; UTR14].
31 While simple functionally, sparsity has its drawbacks when it comes to inspection and interpretation:
32 for example, if features are correlated, then the model may be forced to pick one and not the other,
33 or even pick both with different signs. To handle these issues, as well as to express more complex
34 functions, some models in this category impose hierarchical or modular structures in which each
35 component is still relatively compact and can be inspected. Examples include topic models (e.g.
36 [BNJ03b]), (small) discrete time series models (e.g. [FHDV20]), and generalized additive models (e.g.
37 [HT17]).
38

39 3. There may be other questions, such as how training data influenced the model, which may still require additional
40 computation or information.

¹ **Logic-based** models use logical statements as basis; such as decision-trees [Bre+17], are often
2 most well-known interpretable models. Other logic-based models include decision lists [Riv87; WR15;
3 Let+15a; Ang+18; DMV15], decision tables, and decision sets [Hau+10; Wan+17a; LBL16; Mal+17;
4 Bén+21] and logic programming [MDR94]. A broader discussion, as well as a survey of user studies
5 on these methods, can be found in [Fre14]. Logic-based models have the advantage of being able to
6 easily model non-linear functions, but they often have trouble with handling continuously changing
7 values (e.g. expressing a linear function vs. a step-wise constant function). Like the compact models,
8 hierarchies and other forms of modularity can be used to extend the expressivity of the model while
9 keeping it human-inspectable. For example, one can define a new concept as a formula based on
10 some literals, and then use that concept in the next formula.

¹¹ ¹² When using inherently interpretable models, three key decisions need to be made: the choice of
the model class, how to manage uninterpretable input features, and the choice of optimization method.

¹³ ¹⁴ **Decision: Model Class.** Since the model is its own explanation, the decision on the model
class becomes the decision on the form of explanation. Thus, we need to consider both whether the
model class is a good choice for modeling the data as well as providing the necessary information
to the user in their context. For example, if one chooses to use a linear model to describe one's
data, then it is important that the intended users can understand or manipulate the linear model
in the desired context. A user may interact differently with, for example, a linear model versus
a decision tree. Moreover, if the fitting process produces a model that is too large to human-
inspectable, then it is no longer inherently interpretable, even if it belongs to one of the model
classes described above. Finally, a learnt model might be inherently interpretable for one context
and end-tasks but not another: For example, a decision tree with even a relatively “small” depth
of 10 nodes may be hard for a human to inspect, though perhaps they may be able to follow a sin-
gle path down it to try to understand whether a specific decision was made for an appropriate purpose.

¹⁵ ¹⁶ ¹⁷ ¹⁸ ¹⁹ ²⁰ ²¹ ²² ²³ ²⁴ ²⁵ ²⁶ ²⁷ ²⁸ ²⁹ ³⁰ ³¹ ³² ³³ ³⁴ ³⁵ ³⁶ ³⁷ ³⁸ ³⁹ **Decision: Optimization Methods for Training.** The kinds of model classes that are typically
inherently interpretable often require more advanced methods for optimization: compact, sparse, and
logic-based models all involve discrete parameters that must be determined. Fortunately, there is a
long and continuing history of research for optimizing such models, including directly via various
optimization programs, via various relaxation and rounding techniques, and various search-based
approaches. Another popular optimization approach is via distillation or mimics: one first trains
a complex model (e.g., neural network) and then use complex model's output to train a simpler
model (sometimes called student model); the simpler model is trained to mimic the behavior of
the complex model. Of course, in this case, only the simpler model is inherently interpretable; the
complex model is only used for producing training data for the simpler model before being discarded.
These optimization techniques are beyond the scope of this chapter but covered in other chapters
and optimization textbooks.

⁴⁰ ⁴¹ ⁴² ⁴³ ⁴⁴ **Decision: How to Manage Uninterpretable Input Features.** Sometimes the input features
themselves are not directly interpretable (e.g. pixels of an image or individual amplitudes spectrogram);
only collections of inputs have semantic meaning for human users. This situation challenges our
ability to create inherently interpretable models, but also explanations in general.

⁴⁵ ⁴⁶ To address this issue, more advanced methods attempt to add a “concept” layer that first converts
the uninterpretable raw input to a set of human-interpretable concept features, and then relate those
⁴⁷

1 concepts to a model’s decision (e.g., prediction)[[Kim+18a](#); [Bau+17](#)]; the latter stage can still be
2 inherently interpretable. A Concept can be built from a set of features or input patterns. For
3 example, one could map a pattern of spectrogram of semantically meaningful sound (e.g., people
4 chatting, cups clinking), and then from those sound to a sound classification (e.g., cafe). While
5 promising, one must ensure that the initial data-to-concept mapping truly maps the raw data to
6 concepts as the user understands them, no more and no less (e.g. sneaking in additional signals).
7 Creating and validating that machine-derived concepts to correspond to a semantically meaningful
8 human concepts is an open research challenge.
9

10 **When might we want to consider inherently interpretable models? When not?** Inher-
11 ently interpretable models have several advantages over other approaches. When the model is its
12 explanation, one need not worry about whether the explanation is faithful to the model or whether it
13 provides the right partial view of the model for the intended task. Thus, an inherently interpretable
14 model can likely be used for a greater variety of interpretation tasks. Relatedly, if a person vets the
15 model and finds nothing amiss, they might feel more confident about avoiding surprises. For all these
16 reasons, inherently interpretable models have been advocated for in high-stakes scenarios, as well as
17 generally being the first go-to to try[[Rud19](#)].
18

19 That said, these models do have their drawbacks. They typically require more specialized
20 optimization approaches. With appropriate optimization, inherently interpretable models can
21 often match the performance of more complex models, but there are domains—in particular, images
22 and text—in which deep models or other more complex models typically give significantly higher
23 performance. Trying to fit complex behavior with a too-simple function may result not only with high
24 bias in the trained model, but the use of features that people may still find stories to explain even if
25 they are irrelevant [[Lun+20](#)]. Finally, in an industry setting, one may not have the opportunity to
26 change an already-implemented model (e.g., a legacy, business critical model that has been tuned
27 over decades would run into some resistance).
28

29 Lastly, we note that just because a model is inherently interpretable, it does not guard against all
30 kinds of surprises: as noted in Section 35.1, interpretability is just one form of validation mechanism.
31 For example, if the data distribution shifts, then one may start observing model behavior that no
32 one expected.

33 35.2.2 Semi-Inherently Interpretable Models: Example-Based Methods

34 Example-based models uses examples (e.g., instances from training set) as basis for explanation. For
35 example, they classify a new input by finding similar instances or representative instances in the
36 training set with that decision, based on a chosen distance metric. They may then apply voting
37 scheme amongst those similar training instances. Like logic-based models, example-based models can
38 describe highly non-linear boundaries.
39

40 K-nearest neighbors is one of the best known model in this class, but there have been many exten-
41 sions, including methods to identify exemplars for predicted classes/clusters (e.g. [[KRS14](#); [KL17b](#);
42 [JL15a](#); [FD07b](#)][[RT16](#); [Arn+10](#)]), generating exemplars (e.g. [[Li+17c](#)]), sophisticated embedding
43 and distance metric methods to define similarity between instances (e.g.[[PM18a](#)]), and parts-based
44 approaches that first decompose an instance into parts and can find neighbors or exemplars between
45 the parts (e.g. [[Che+18b](#)]).
46

47 On one hand, individual decisions made by example-based methods seem fully inspectable: one

¹ can provide the user with exactly the training instances (including their labels) that were used to
² classify a particular input in a particular way. However, it may be difficult to convey a potentially
³ complex distance metric, and the user may extrapolate the reasons that the examples are rated as
⁴ similar incorrectly (e.g., what features made the examples similar). It is also often difficult to convey
⁵ the intuition behind the global decision boundary using examples.
⁶

⁷

⁸ 35.2.3 Post-hoc or Joint training: The Explanation gives a Partial View of the ⁹ Model

¹⁰

¹¹ While there exist many approaches for creating inherently interpretable models, inherently inter-
¹² pretabile models are clearly only a subset of all machine learning models. Various circumstances
¹³ may require working with a model that is not inherently interpretable: as noted above, large neural
¹⁴ models have demonstrated large performance benefits for certain kinds of data (e.g. images and text);
¹⁵ one might have to work with a legacy, business critical model that has been tuned for decades; one
¹⁶ might be trying to understand a system of interconnected models.

¹⁷ In these cases, explanations can still be extracted; however, the view that the explanation gives into
¹⁸ the model will necessarily be partial: the explanation may only be an approximation of the model or
¹⁹ be otherwise incomplete. Thus, more decisions have to be made about what the explanation should
²⁰ contain—and how it should be computed—so that it provides the information that is necessary for
²¹ the specific context and end-task. It is crucial that one understands the abilities and limitations of
²² these partial explanation methods [Sla+20; Yeh+19a; Kin+19; Ade+20a].

²³ Below, we split these (interconnected) decisions into two broad categories—what the explanation
²⁴ consists of (which also requires knowing whom it is for, the context) and how the explanation is
²⁵ computed given the trained model.

²⁶

²⁷ 35.2.3.1 What does the explanation consist of?

²⁸

²⁹ One set of decisions center around what the explanation consists of, including what properties it has
³⁰ given a context (includes whom it is for) and an end-task. One key decision is the form: Will the
³¹ explanation be a list of important features? A local model? Another is formalizing what properties
³² the explanation must have—for example, in what sense should that list of key features reflect the
³³ true model? Finally, one must decide the scope of the explanation: Global, local, or somewhere in
³⁴ between? We expand on each of these below; the right choice, as always, will depend on user—whom
³⁵ the explanation is for—and their end-task.

³⁶

³⁷ **Decision: Form of the Explanation.** In the case of inherently interpretable models, the choice
³⁸ of the model class used to fit the data was also the choice of the form of the explanation. Now, the
³⁹ model class and the explanation are two different entities. Thus, regardless of what model class
⁴⁰ one uses to fit the data, one also needs to determine the form of the explanation, that is, the way
⁴¹ in which the information will be delivered. For example, the model could be a deep network; the
⁴² explanation in the form of a decision tree.

⁴³ Works in interpretable machine learning have used a large variety of forms of explanations. The
⁴⁴ form could be a list of “important” input features [RSG16b; Lun+20; STY17; Smi+17; FV17] or
⁴⁵ “important” concepts [Kim+18a; Bau+20; Bau+18]. Or it could be a simpler model that approximates
⁴⁶ the complex model (e.g. a local linear approximation, an approximating rule set)[FH17; BKB17;
⁴⁷

[Aga+19b; Yin+19c]. Another choice could be a set of similar or prototypical examples [KRS14; AA18; Li+17c; JL15a; JL15b; Arn+10]. Finally, one can choose whether the explanation should include a contrast against an alternative (also sometimes described as a counterfactual explanation) [Goy+19; WMR18; Kar+20a] or include or influential examples [KL17b].

Different forms of explanations will facilitate different tasks in different contexts. For example, a contrastive explanation of why treatment A is better than treatment B may help a clinician determine whether to choose treatment A over treatment B—but that same contrast between treatments A and B may not help if the question is deciding between treatments A and C. Similarly, an explanation that provides the most important features for a prediction may not provide information about how those features interact. Given the large number of choices, observations or literature on what people communicate in the desired context may provide some guidance. For example, if the domain is one that involves making quick, high-stakes decisions, one might turn to the literature on recognition-primed decision making [Kle17] which discusses how trauma nurses and firefighters explain their decisions.

Decision: Determining what Properties the Context Needs. Each of the forms of explanations above have different levels of expressivity (e.g., consider the different expressivity between a local linear model and a local decision tree). For each form, there will also be many ways to compute an explanation of that form (more on this in Section 35.2.3.2). How do we choose amongst all of these different ways to compute the explanation? Rather than jump to particular forms of computation (e.g., gradients vs. local perturbations to determine feature importances) the first step should be to determine what properties are needed from the explanation; the properties provide the specification for the computation.

Specifying properties is especially important because different forms of explanation may not only have different intrinsic properties, but they may have different properties depending on the underlying model (that they are trying to explain). For example, if the function that represents the underlying model is relatively smooth, then a feature-based explanation relying on local gradients may be fairly faithful to the original model; however, if the function has strong spikes, the same feature-based explanation (with the same computation) may no longer be faithful to the model; producing conflicting results at best. Similarly, whether the data lie on a low-dimensional manifold or not will determine the properties of an approach that identifies key features based on local perturbations. Once the desired properties are determined, one can determine what kind of computation is necessary to achieve them. We will list commonly desirable properties in Section 35.3.

Decision: Scope of the Explanation: Global or Local. The final major decision regarding the parameters of the explanation is its scope: global or local.

Local explanation : In some cases, we may only need the explanation to interrogate an existing model about a specific decision: For example, why was this image predicted as a bird? Why was this patient predicted to have disease X? Local explanations are useful if one needs to dig into a particularly important decision that a model made, for example to see if an error was made or determine what could have been done differently to produce a different outcome (recourse).

Local explanations can take many forms. They may be a locally-fit simpler model in the neighborhood of the data of interest (e.g. LIME [RSG16b]). A family of methods called saliency maps or attribution maps [STY17; Smi+17; ZF14; Sel+17; Erh+09; Spr+14; Shr+16] use similar approach

¹ by estimating feature importances via first-order derivatives. A local explanation may also consist
² of representative examples, including identifying which data points e.g., from training set, were
³ most influential for a particular decision [KL17b] or identifying nearby data points with different
⁴ predictions (e.g., counterfactual examples) [MRW19; LHR20; Kar+20a]. With all local explanation
⁵ methods, one needs to be cautious not to overgeneralize beyond the scope of the explanation, as well
⁶ as not overfit user's mental model of the model based on a few local explanations; again, post-hoc
⁷ explanations are necessarily partial.
⁸

⁹
¹⁰ **Global explanation** : In other cases, we may desire insight into the model as a whole or for a
¹¹ collection of data points (e.g., all inputs predicted to one class). Such a broader scope may be needed,
¹² for example, if the end-task is to make a deployment decision; then it's not just the decision process
¹³ for a few inputs that matter, one wants to vet the model's overall decision process. That said, it is
¹⁴ important to remember that a global explanation is still a partial view of the underlying model; there
¹⁵ could still be qualities of the model—either highly local, or simply lost in the approximation—that
¹⁶ the global explanation misses.

¹⁷ Global explanations can take many forms. One choice to fit a simpler model (e.g. an inherently
¹⁸ interpretable model) that somehow approximate the original model (e.g. [HVD14]). One can identify
¹⁹ concepts or features that affect decision across many inputs (e.g. [Kim+18b]). Another approach
²⁰ is to provide a carefully set of representative examples[Yeh+18]. These examples might be chosen
²¹ to be somehow characteristic of, or providing coverage of, a class (e.g. [AA18]), to draw attention
²² to decision boundaries (e.g. [Zhu+18]), or to identify inputs particularly influential in training the
²³ model.

²⁴

²⁵ 35.2.3.2 How the explanation is computed

²⁶ Another set of decisions have to do with how the explanation is computed.

²⁸

²⁹ **Decision: Computation of Explanation.** Decisions that we discussed so far (form of explanations, properties, local vs. global) are our decisions/desiderata for our contexts and end-task. In
³⁰ order to produce the explanation, we now need to decide how to compute it. Together with other
³¹ decisions, this decision finalizes the definition of "explanation" in each method. Therefore, it is crucial
³² to iterate to optimize this decision for the context and end-task in mind.

³⁴ For example, suppose one is seeking to identify the most "important" input features that changes a
³⁵ prediction, the computation would differ depending on their definition of importance. One definition
³⁶ of importance (and therefore computation decision) might be the smallest region in an image when
³⁷ changed, prediction changes—a perturbation-based analysis. In case of local perturbations alone, we
³⁸ need to then decide how much to perturb (while remained within training distribution) and in what
³⁹ segments (e.g., perturb each pixels, a set of pixels at a time) [SVZ13; DG17; FV17; DSZ16; Adl+18;
⁴⁰ Bac+15]. Another way to compute "importance" is by measuring how often the input feature is
⁴¹ part of a "winning coalition" that drives the prediction, e.g. a Shapley or Banzaf score[LL17]. Now
⁴² suppose one is now seeking to identify the most "important" input features in terms of sensitivity
⁴³(e.g., largest gradients of the output with respect to the input feature). Even then, there are many
⁴⁴ other computational decisions one can make such as done in [STY17; Smi+17; Sel+17; Erh+09;
⁴⁵ Shr+16]. Each of these choices of how to compute the explanation will have different properties, as
⁴⁶ well as require different amounts of computation.

⁴⁷

Similar issues come up with other forms of explanations. For example, if an example-based form is chosen, then one has to determine what it needs to be ‘similar’ (e.g. cosine similarity between activations? a uniform L2 ball of a certain size between inputs?) or otherwise ‘representative.’ In another example, there are many different ways to obtain counterfactuals; by defining a distance function and loss terms to describe what “counterfactual example” is [WMR17; LHR20], or formulating it as a SAT problem[Kar+20a], or by following causal inference framework [Kus+18].

Decision: Joint Training vs. Post-hoc Application. So far, we have described our partial explanation techniques as extracting some information from an already-trained model. This approach is called deriving a post-hoc explanation. As noted above, post-hoc, partial explanations may have some limitations: for example, an explanation based on a local linear approximation may be great if the model is generally smooth, but provide little insight if the model has high curvature—this is not because the partial explanation is wrong, but because the view that local gradients provides isn’t sufficient for curvy true decision boundary.

Another approach to getting explanations to have the properties we desire is to train the model and the explanation jointly. For example, A regularizer that penalizes violations of desired properties can help steer the overall optimization process towards learning models that both perform well and are amenable to the kind of explanation that we desire [Plu+20]. It is often possible to find such a model because most complex model classes have multiple high-performing optima [Bre01].

The choice of regularization will depend on the desired properties, the form of the explanation, and its computation. For example, we may know of the kinds of features that people are likely to be using themselves for a task (e.g., lower frequency vs. higher frequency features in image classifiers [Wan+20a]) to make machine learned classification processes match those used by people. We may want the local explanation to use or not use certain input dimensions or to be sparse or otherwise compact (e.g. a small decision tree) while still being faithful to the underlying model [RHDV17; Shu+19; Vel+17; Nei+18; Wu+19b; Plu+20]; this category may also include attention models [JW19; WP19] depending on what properties are desired. We may also have constraints on the properties of concepts or other intermediate features [AMJ18b; Koh+20; Hen+16; BH20; CBR20; Don+17b].

When choosing between a post-hoc explanation or joint training, one key consideration is that joint training assumes that one can re-train the model or the system of interest. In many cases in practice, this may not be possible. Replacing a complex and well-validated system in deployment for a decade may not be possible or take prohibitively long time or not allowed. In that case, one can still extract approximated explanations using post-hoc methods. Finally, a joint optimization, even when it can be performed, is not a panacea: optimization for some properties may result in unexpected violations of other (unspecified but desired) properties. For this reason, explanations from jointly trained model is still often partial.

When might we want to consider post-hoc methods, and when not? The advantage of post-hoc interpretability methods is that they can be applied to any model. This family of methods is especially useful in real-world scenarios where one needs to work with a system that contains many models as its parts, where one cannot expect to replace the whole system with one model. These approaches can also provide at least some broader knowledge about the model to identify unexpected concerns.

That said, post-hoc explanations, as approximations of the true model, may not be fully faithful to

¹ the model nor do they cover the model completely. As such, an explanation method tailored for one
² context may not be transferable in another; even in the intended context, there may be blindspots
³ about the model that the explanation misses completely. For these reasons, in high stakes situations,
⁴ one should attempt to use an inherently interpretable model first if possible [Rud19]. In all situations
⁵ when post-hoc explanations are used, one must keep in mind that they are only one tool in a broader
⁶ accountability toolkit and warn users appropriately.
⁷

⁸

⁹ 35.2.4 Transparency and Visualization

¹⁰

¹¹ The scope of interpretable machine learning is usually centered around methods that expose the
¹² process by which a trained model makes a decision. However, the behavior of a model closely depends
¹³ on the training data, how the training data were collected and processed, and how the model was
¹⁴ trained and tested. Conveying to a human these other aspects of what goes into the creation of
¹⁵ a model can be as important as explaining the trained model itself. While a full discussion of
¹⁶ transparency and visualization is outside the scope of this chapter, we provide a brief discussion here
¹⁷ to describe these important adjacent concepts.

¹⁸

¹⁹ Transparency is an umbrella term for the many things that one could expose about the modeling
²⁰ process and its context. Interpreting models is one aspect. However, one could also be transparent
²¹ about other aspects, such as the data collection process or the training process (e.g. [Geb+21;
²² Mit+19; Dnp]). There are also situations in which a trained model is released (whether or not it is
²³ inherently interpretable), and thus the software can be inspected and run directly.

²⁴

²⁵ Visualization is one way to create transparency. One can visualize the data directly, various
²⁶ aspects of the model's process (e.g. [Str+17]), and create interactive/static visualizations that can
²⁷ convey more than text or code descriptions [ZF14; OMS17; MOT15; Ngu+16; Hoh+20]. Finally,
²⁸ in the specific context of interpretable machine learning, how the explanation is presented—the
²⁹ visualization—can make a large difference in how easily users can consume it. Even something as
³⁰ simple as a rule list has many choices of layout, highlighting, and other organization.

³¹

³² **When might we want to consider transparency and visualization? When not?** In many
³³ cases, the trouble with a model comes not from the model itself, but parts in its training pipeline. For
³⁴ example, the data it was trained on could cause problems—if policing data contain historical bias, then
³⁵ predictions of crime hot spots based on that data will be biased. Similarly, if clinicians only order
³⁶ tests when they are concerned about a patient's condition, then a model trained to predict risk based
³⁷ on tests ordered will only recapitulate what the clinicians already know. Transparency—knowing
³⁸ about the global properties of data, and how training and testing were performed can help identify
³⁹ these issues, which would cause problems regardless of which model was used.

⁴⁰ Of course, inspecting the data and the model generation process is something that takes time and
⁴¹ attention. Thus, visualizations and other descriptions to increase transparency are best-suited to
⁴² situations in which a human inspector wants to understand complex patterns and potential sources
⁴³ of trouble without much time pressure (e.g., prior to starting a project). These methods are not
⁴⁴ well-suited for situations in which a specific decision must be made in a relatively short amount of
⁴⁵ time, e.g. providing decision-support to a clinician at the bedside.

⁴⁶ Finally, transparency in the form of making code available can potentially assist in understanding
⁴⁷

1 how a model works, identifying bugs, and allow independent testing by third party (e.g, testing with
2 new set of inputs, evaluating counterfactuals in different testing distributions) However, if a model is
3 sufficiently complex, as many modern models are, then simply having access to the code may not be
4 enough for a human to gain sufficient understanding for their task.
5

6

7 35.3 Properties: The Abstraction Between Context and Method 8

9 Recall from the Terminology and Framework in Section 35.1.2 that the context and end-task determine
10 what properties of the explanation. For example, in a high-stakes setting—such as advising on
11 interventions for an unstable patient—it may be important that the explanation completely and
12 accurately reflects the model (fidelity). In contrast, in a discovery-oriented setting, it might be more
13 important for any explanation to allow for efficient iterative refinement, revealing different aspects
14 of the model in turn (interactivity). From these examples, it should be clear that not all contexts
15 and end-tasks need all properties, and the lack of a key property may result in poor downstream
16 performance.

17 While the research is still evolving, there exists a growing informal understanding about how
18 properties may work as an abstraction between methods and contexts. Many interpretability methods
19 from Section 35.2 share the same properties, and methods with the same properties may have similar
20 downstream performance in a specific end-task and context. If two contexts and end-tasks require
21 the same properties, then a method that works well for one may work well for the other. A method
22 with properties optimal for one downstream could miserably fail in another context.
23

24 **How to find desired properties?** Of course, identifying what properties are important for a
25 particular context and end-task is not trivial. Recall that identifying what properties are important
26 for what contexts, end-tasks, and downstream performance metrics is one facet of the grand challenge
27 of interpretable machine learning. For the present, the process of identifying the correct properties
28 will likely require iteration via user studies. However, iterating over properties is still a much smaller
29 space than iterating over methods. In particular, knowing what the currently focused properties are
30 can aid user study design. For example, if one wants to test whether the sparsity of the explanation
31 is key to good downstream performance, one could intentionally create explanations of varying levels
32 of sparsity to test that hypothesis. This is a much more precise knob than exhaustively trying out
33 different explanation methods across hyperparameter space.

34 Below, we first describe examples of properties that have been discussed in the interpretable
35 machine learning literature. Many of these properties are purely computational—that is, they can be
36 determined purely from the model and the explanation—while a few have some user-centric elements.
37 Next we list examples of properties of explanation from cognitive science (on human to human
38 explanations) and human-computer interaction (machine to human explanations). Some of these
39 properties have current analogs in the machine learning list, while others may serve as inspiration for
40 areas to formalize.
41

42 35.3.1 Properties of Explanations from Interpretable Machine Learning 43

44 Many lists of potentially-important properties of interpretable machine learning models have been
45 compiled, sometimes using different terms for similar concepts and sometimes using the similar terms
46 for different concepts. Below we list some commonly-described properties of explanations, knowing
47

¹
² that this list will evolve over time as the field advances.

³
⁴ **Compactness, Sparsity** (e.g. as described in [Lip18; Mur+19]). In general, an explanation
⁵ must be small enough such that the user can process it within the constraints of the task (e.g. how
⁶ quickly a decision must be made). Sparsity generally corresponds to some notion of smallness (a
⁷ few features, a few parameters, $L1$ norm etc.), whereas compactness generally carries an additional
⁸ notion of not including anything irrelevant (that is, even if the explanation is small enough, it could
⁹ be made smaller). Each must be formalized for the context.

¹⁰
¹¹ **Faithfulness, Fidelity** (e.g. as described in [JG20; JG21]). When the explanation is only a
¹² partial view of the model, how well does it match the model? There are many ways to make this
¹³ notion precise. For example, suppose a mimic (simple model) is used to provide a global explanation
¹⁴ of a more complex model. One possible measure of faithfulness could be whether the mimic gives the
¹⁵ same outputs as the original. Another could be whether the mimic has the same first derivatives
¹⁶ (local slope) as the original. In the context of a local explanation consisting of the key features for a
¹⁷ prediction, one could measure faithfulness by whether the prediction changes if the supposedly impor-
¹⁸ tant features are flipped. Another measure could check to make sure the prediction does not change if
¹⁹ a supposedly unimportant feature is flipped. The appropriate formalization will depend on the context.

²⁰
²¹ **Completeness** (e.g. as described in [Yeh+19b]). If the explanation is not the model, does it still
²² include all of the relevant elements? For example, if an explanation consists of important features
²³ for a prediction, does it include all of them, or leave some out? Moreover, if the explanation uses
²⁴ derived quantities that are not the raw input features—for example, some notion of higher-level
²⁵ concepts—are they expressive enough to explain all possible directions of variation that could change
²⁶ the prediction? Note that one can have faithful explanation but not complete (e.g., while flipping
²⁷ each unimportant features do not change prediction—faithful, the explanation may fail to include
²⁸ that flipping a selected set of them do change the prediction).

²⁹
³⁰ **Stability** (e.g. as described in [AMJ18a]) To what extent are the explanations similar for similar
³¹ inputs? Note that the underlying model will naturally affect whether the explanation can be stable
³² and faithful. For example, if the underlying model has high curvature and the explanation has limited
³³ expressiveness, then it may not be possible to have an explanation that is stable and faithful.

³⁴
³⁵ **Actionability** (e.g. as described in [Kar+20b; Poy+20]). Actionability implies filtering the
³⁶ content of the explanation to focus on only aspects of the model that the user might be able
³⁷ to intervene on. For example, if a patient is predicted to be at high risk of heart disease, an
³⁸ actionable explanation might only include mutable factors such as exercise, and not immutable
³⁹ factors such as age or genetics. The notion of recourse corresponds to actionability in a justice context.

⁴⁰
⁴¹ **Modularity** (e.g. as described in [Lip18; Mur+19]). Modularity implies that the explanation can
⁴² be broken down into understandable parts. While modularity does not guarantee that the user can
⁴³ explain the system as a whole, for more complex models, modular explanations—where the user can
⁴⁴ inspect each part—could be the most effective way to provide a reasonable level of insight into the
⁴⁵ model’s workings.

⁴⁶
⁴⁷

1 **Interactivity.** (e.g., [Ten+20]) Does the explanation allow the user to ask questions, such as how
2 the explanation changes for a related input, or how an output changes given a change in input? In
3 some contexts, providing everything that a user might want or need to know from the start might be
4 overwhelming, but it might be possible to provide a way for the user to navigate the information
5 about the model in a way that they choose.
6

7 **Translucence** (e.g. as described in [SF20; Lia+19]). Is the explanation clear about its limitations?
8 For example, if a linear model is locally fit around a particular input of interest and used to explain
9 a deep model, is there a mechanism that reports that this explanation may be limited if there are
10 strong feature interactions around that input? We emphasize that translucence is about exposing
11 limitations in the explanation, rather than the model. The goal of the explanation—and all our other
12 accountability methods—is to expose the limitations of the model.
13

14 **Simulability** (e.g. as described in [Lip18; Mur+19]). A model is simulable if a user can, given
15 the model and an input, compute the output (within any constraints of time and cognition). In the
16 context of explanations, the explanation must be a "simulable model". For example, a list of features
17 would not be simulable by itself, because a list of features alone does not imply a computation from
18 features to prediction. A user would have to make some assumptions (e.g. by assuming a linear
19 model) to predict the output of the model. However, an explanation in the form of a decision tree
20 does include a computation process—following the logic of the tree—and might be simulable as long
21 as the tree was not too deep. The latter examples also point out an important difference between
22 compactness and simulability: if an explanation is too large, it may not be simulable. However, just
23 because an explanation is compact—such as a short list of features—does not mean that a person
24 can compute the model's output with it.
25

26 It may seem that simulability is different from the properties we have listed so far because its defi-
27 nition involves human input. However, in practice, we may often know what kinds of explanations are
28 easy for people to simulate (e.g. decision trees with short path lengths, rule lists with small formulas,
29 etc.). This knowledge can then be turned into a purely computational training constraint where we
30 seek explanations that are known to be simulable. In this sense, computational formalizations of sim-
31 ulability are no different than computational formalizations of other properties such as compactness,
32 where there is a human-centric element that requires us to determine how compact is compact enough.
33

34 **Alignment to the User's Vocabulary and Mental Model.** (e.g., as described in [Kim+18a])
35 Is the content of the explanation designed for the user? For example, an explanation in terms of
36 parameter variances and influential points may be comprehensible to an engineer debugging a lending
37 model but not to a lay user trying to get a loan. Similarly, an explanation given in the semantics a
38 user knows—such as in terms of medical conditions vs. raw sensor readings to a clinician—help the
39 explanations to be placed in the context of user's expert knowledge and decision-making guidelines
40 (modified quote from [Clo+19]).

41 Like simulability, this property is more human-centric. However, just as before, we can imagine an
42 abstraction between eliciting vocabulary and mental models from users—that is, determining how
43 they define their terms and how to think—and ensuring that an explanation is provided in alignment
44 with whatever that elicited user vocabulary and mental model is.
45

46 Once desired properties are identified, we need to operationalize them. For example, if sparsity is a
47

¹ desired property, would using L1 norm enough? Or something more sophisticated loss term needs to
² be designed? This decision will necessarily be human-centric: how small an explanation needs to be,
³ or in what ways it needs to be small, is a decision that needs to consider how people will be using the
⁴ explanation. Once operationalized, most properties can be optimized for computationally. That said,
⁵ the properties should be evaluated with context and end-task, model and the chosen explanation
⁶ methods. Once evaluation returns, one may revisit the method and models decision depending on
⁷ the importance of the properties to the performance of the user on the end-task.
⁸

⁹ Finally, we emphasize that the ability to achieve a particular property will depend on the **intrinsic**
¹⁰ complexity of the model. For example, the behavior of a highly nonlinear model with interactions
¹¹ between the inputs will, in general, be harder to understand than a linear model. No matter how we
¹² try to explain it, if we are trying to explain something complicated, then users will have a harder
¹³ time understanding it.

¹⁴

¹⁵ 35.3.2 Properties of Explanations from Cognitive Science

¹⁶

¹⁷ While work in interpretable machine learning have developed lists of properties largely from a
¹⁸ computational perspective, in particular, the relationship between the model and the explanation, the
¹⁹ fields of cognitive science and human-computer interaction have examined what people consider good
²⁰ properties of an explanation for a long time, that is the relationship between the explanation and the
²¹ human. These more human-centered properties may be ones that researchers in machine learning
²² may be less aware of, yet essential for the explanation to do its job of communicating information to
²³ a human.

²⁴ Below we list several of these properties. Just as with the previous list, different properties may be
²⁵ important in different contexts—unsurprisingly, the literature on human explanation concurs that
²⁶ the explanation must fit the context [VF+80], therefore there is no one optimal explanation for all
²⁷ contexts. Finally, we emphasize that human explanations are also social constructs that happen as
²⁸ part of a conversation, and that they often include post-hoc rationalizations and other biases. In
²⁹ other words, one must carefully decide on desired properties for context and end-task in mind and
³⁰ weigh in potential human biases, rather than deciding "because humans sometimes do it". We focus
³¹ on properties of explanations that help users achieve their goals.

³²

³³ **Soundness** (e.g., as described in [Kul+13]). Explanations should contain nothing but the truth
³⁴ with respect to whatever they are describing. Soundness corresponds to notions of *compactness* and
³⁵ *faithfulness* above.

³⁶

³⁷ **Completeness** (e.g., as described in [Kul+13]). Explanations should contain the whole truth
³⁸ with respect to whatever they are describing. Completeness corresponds to notions of *completeness*
³⁹ and *faithfulness* above.

⁴⁰

⁴¹ **Contrastiveness** (e.g., as described in [Mil19a]). Contrastive explanations provide information of
⁴² how something differs from an alternate decision or prediction. For example, instead of providing a
⁴³ list of features for why a particular drug is recommended, it might provide a list of features that for
⁴⁴ why one drug is recommended over another. Contrastiveness relates to notions of *actionability* above,
⁴⁵ and more generally explanation types that include *counterfactuals*.

⁴⁶

⁴⁷

Generality. Overall, people understand that an explanation for one context may not apply in another. That said, there is an expectation that an explanation should reflect some underlying mechanism or principle and will thus apply to similar cases—for whatever notion of similar is in the person’s mental model. Explanations that do not generalize to similar cases may be misinterpreted.

Simplicity. All of the above being equal, simpler explanations are generally preferred. Simplicity relates to notions of *sparsity* and *complexity* above.

Finally, the cognitive science literature also notes that explanations are often goal directed. This matches notion of explanation in ML as information that helps a person use a prediction better, give understanding by providing appropriate features, or otherwise do well on their end-task. Different kinds of content or forms may help with different goals, and thus human explanations take many forms. Forms of explanations include deductive-nomological (i.e. a logical proofs) [HO48], providing some sense of underlying mechanism [BA05; Gle02; CO06], and conveying understanding [Kei06]. Knowing these forms can help us consider what options might be best among different sets of interpretable machine learning methods.

35.4 Evaluation of Interpretable Machine Learning Models

Recall that there is no one universal formalization of interpretability: interpretability will depend on the context, the end-task, and the downstream performance metric. If the explanation empowers the human to get better performance on their end-task, then it was useful and interpretable for that context. In addition, if another explanation results in even better downstream performance, then it is more interpretable than the first. In other words, there is no one universal formalization of interpretability without the context: interpretability will depend on the context, the end-task, and the metric [VF+80].

The fact that the optimal interpretability methods depends on the context proposes challenges in evaluation: How can we evaluate many methods, across different downstream contexts? Recall that one of the grand challenges in interpretable machine learning is to develop a general understanding of what properties are needed for different contexts and end-tasks.

That said, we also emphasize that grand challenges are grand challenges because they are difficult to solve, and likely will take an entire community to solve. Along the way, we must recognize the value of quantifying the impact of a method in its intended context and not expect that method to work across widely different contexts. And we must ensure that in that limited setting of a particular context, we do our evaluations rigorously [DVK17].

In this section, we describe two major categories for evaluating interpretable machine learning methods:

Computational evaluations of properties without people. The first involves computational evaluations of whether explanations have the desired properties. For example, one can computationally measure whether a particular explanation satisfies a definition of faithful (e.g., under different training and test data distributions), or whether the outputs of one explanation are more sparse than another, under a mathematical definition of sparsity. Such measures are valuable when one already knows that certain properties may be important for certain contexts. Computational evaluations

¹
² also serve as intermediate evaluations and sanity checks to identify undesirable explanation behavior
³ prior to a more expensive user study-based evaluation.

⁴
⁵ **User studies with people.** The second category of evaluation involves user studies with
⁶ humans. Rigorous, carefully-designed, quantitative and qualitative user studies measure how well
⁷ an interpretable machine learning method enables the user to complete their end-task in a given
⁸ context. We emphasize that performing a rigorous, well-designed user study in a real context is
⁹ significant work—much more so than computing a test likelihood on benchmark datasets. It requires
¹⁰ significant asks of not only the researchers but also the target users. Methods for different contexts
¹¹ will also have different evaluation challenges: while a system designed to assist with optimizing
¹² music recommendations might be testable on a wide population, a system designed to help a particle
¹³ physicist identify new kinds of interactions might only be tested with one or two physicists because
¹⁴ people with that expertise are hard to find. In all cases, the evaluation can be done rigorously with
¹⁵ proper attention to the experimental design.

¹⁶

¹⁷ **35.4.1 Computational Evaluation: Does the Method have Desired Properties?**

¹⁸
¹⁹ While the ultimate measure of interpretability is whether the method successfully empowers the
²⁰ user to perform their task, properties can serve as a valuable abstraction. For example, checking
²¹ whether an explanation has the right computational and desired properties can ensure that the
²² method works as expected (e.g., no obviously odd behaviors, no implementation errors) and iterate to
²³ make computational improvement (e.g., better optimization methods given quantitative metric of a
²⁴ property) before conducting expensive human experiments. Checking for specific properties can also
²⁵ help pinpoint in what way an explanation is falling short, which may be less clear from a user study
²⁶ due to confounding . For example, if we know from the context that the explanation must be faithful
²⁷ in a particular way, we can computationally check whether an explanation achieves that property,
²⁸ as well as whether one explanation scores better on that property than another. Computational
²⁹ checks can also to ensure whether properties that held for one model continue to hold when applied
³⁰ to another model.

³¹ There may also be cases where user studies are not required to demonstrate the value of a new
³² method in interpretable machine learning. For example, there may be situation in which an existing
³³ method has already been demonstrated to have certain properties—such as a faithful, compact
³⁴ decision set—such that they are useful in certain contexts. However, this method may have the
³⁵ computational challenge of computing (e.g., computational efficiency, or reliability across different
³⁶ datasets or model classes) the explanation. In this case, one can demonstrate methodological
³⁷ improvements purely computationally.

³⁸ In some cases, one might be able to mathematically prove that an explanation has certain prop-
³⁹erties, while in others the test must be empirical. For empirical testing, one strategy is to use a
⁴⁰ hypothesis-based sanity check; if we think a phenomenon X should never occur (hypothesis), we
⁴¹ can try to test whether we can create situations where X may occur. If it does, then the method
⁴² fails this sanity check. Another umbrella strategy is to create datasets with known characteristics or
⁴³ ground truth. These could be purely synthetic constructions (e.g. generated tables with intentionally
⁴⁴ correlated features), semi-synthetic approaches (e.g. intentionally changing the labels on an image
⁴⁵ dataset), or taking slices of a real dataset (e.g. introduce intentional bias by only selecting real
⁴⁶ image, label pairs that are of outdoor environments). Note that these tests only reflect lower-
⁴⁷

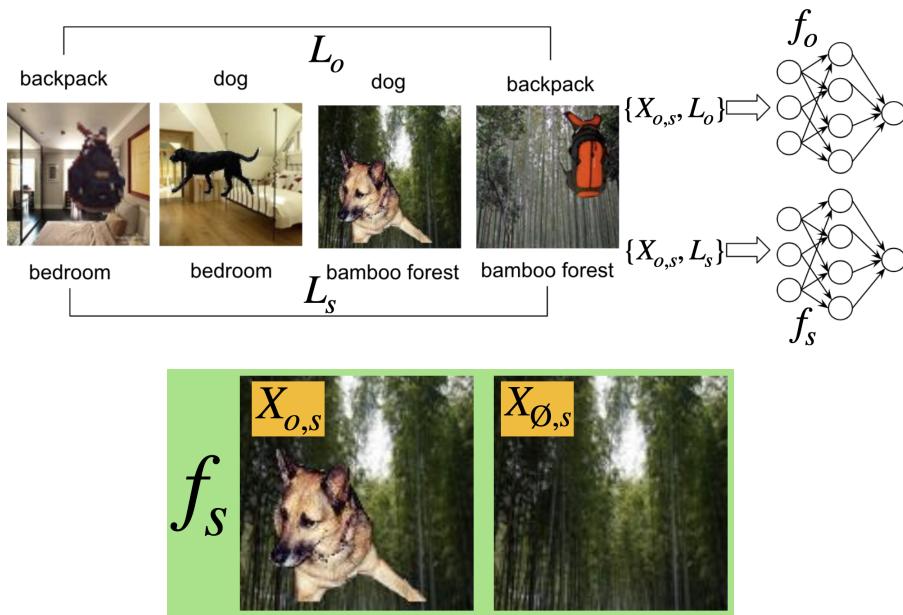


Figure 35.3: An example of computational evaluation using (semi-)synthetic datasets from [YK19]: foreground images (e.g. dogs, backpacks) are placed on different backgrounds (e.g. indoors, outdoors) to test what an explanation is looking for.

bounds: If a method finds what it should find in these constructed datasets or sanity checks, then there may still be missing blindspots, but if it fails in these settings, we know that something is wrong.

Examples of Sanity Checks. One strategy is to do hypothesis-based sanity check; if we think a phenomenon X should not occur (hypothesis), we can try to test whether we can create situations where X may occur. If it does, then the method fails this sanity check. This type of strategy often services as a lower-bound, bare minimum check by asking outside-of-the-box questions, and often to reveal some surprising failure modes of explanation methods.

For example, [Ade+20a] operates under the assumption that a faithful explanation should be function of a model's prediction. The hypothesis is that the explanation should significantly change when comparing a trained model to an untrained model (where prediction is random). They show that many existing methods fails to pass this sanity check (Figure 35.4).

In another example, [Kin+19] hypothesize that a faithful explanation should invariant to input transformations that do not affect model predictions or weights, such as constant shift of inputs (e.g., all inputs are added by 10). This hypothesis can be seen as testing both faithfulness and stability properties. This works shows that some methods fails this sanity check.

In a way, adversarial attacks[GAZ19] on explanation methods could be seen as a sanity check; one that would be harder for many methods to pass. [GAZ19] shows that two perceptively indistinguishable inputs with the same predicted label can be assigned very different interpretations.

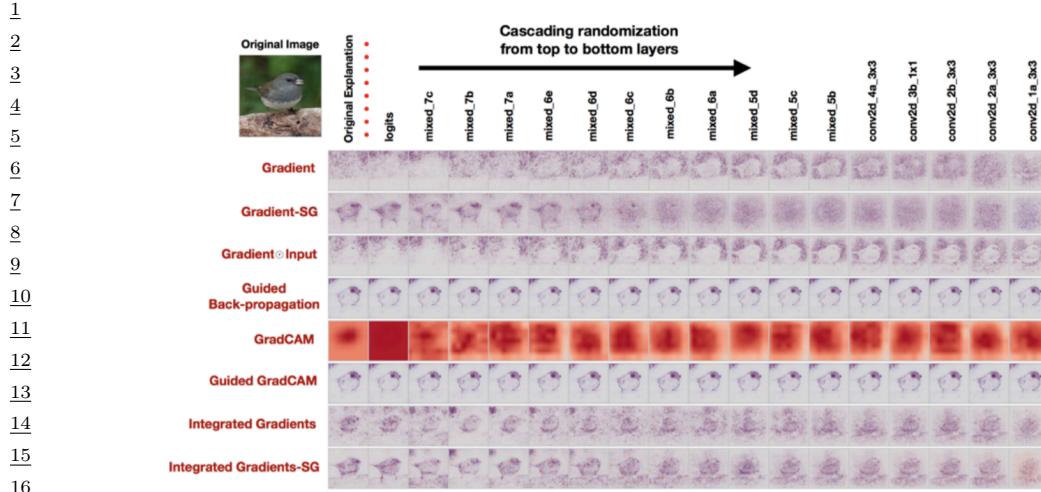


Figure 35.4: Interpretability methods (each row) and their explanations as we randomize layers starting from the logits, and cumulatively to the bottom layer (each column), in the context of image classification task. The rightmost column is showing a completely randomized network. Most methods output similar explanations for the first two columns; one predicts the bird, and the other predicts random. This sanity check tests the hypothesis that the explanation should significantly change (quantitatively and qualitatively) when comparing a trained model and an untrained model [Ade+20a].

Examples using (semi-)Synthetic Datasets. While sanity checks may often lead to binary conclusion, using (semi-)synthetic datasets for computational evaluation come put quantitative metrics that can be compared across methods.

We use the work of [YK19] as an example. Here, the authors were interested in explanations with the properties of compactness and faithfulness: it should not identify features as important if they are not (i.e., explanations should have low false positive). The "importance" is defined as a set of features that are correlated with prediction (not causation). To test fidelity under this definition, authors generate a set of image dataset with intentional correlation; an object often co-occur with particular background (an object is a image patch from a real picture, pasted onto another real background picture, see Figure 35.3). They generate a few sets of dataset with varying level of the correlation between objects and backgrounds. Note that each dataset comes with two labels per image: for an object and a background.

Note that correlation of features do not imply causal relationship—that those features 'caused' the prediction. What we do know with certainty is *relative* importance: for example, we can compare two models one trained to classify objects and one trained to classify backgrounds (left, Figure 35.3). If a model is trained to classify objects and they all happen to have the same background, the background should be less important than in a model trained to classify backgrounds (Model Contrast Score). We can also deduce similar relative importance given one model: a model is given two images with (image A) and without (image B) unimportant set of features (e.g., a model trained to predict backgrounds. All images contain a dog. Two images are given to the model one with and without a dog) (see right

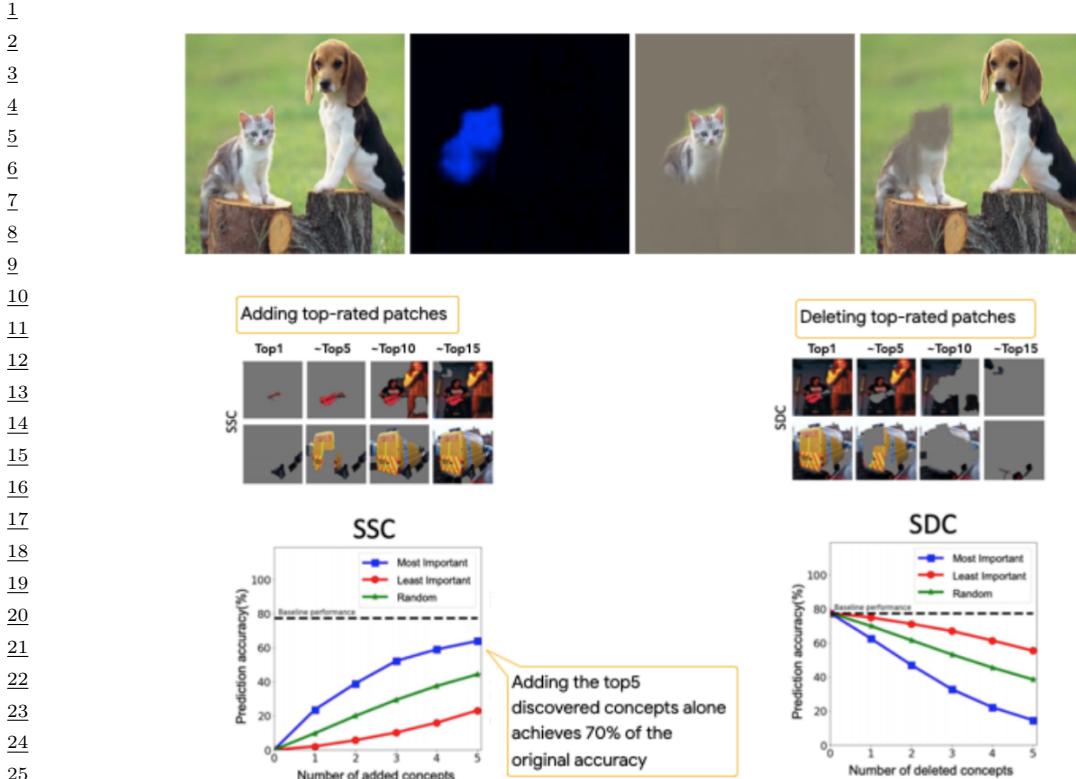


Figure 35.5: Examples of computational evaluation with real datasets from [DG17; Gho+19]. For example, one would expect that adding or deleting patches rated as most relevant for an image classification would have a large effect on the classification compared to patches not rated as important.

Figure 35.3). When comparing attributions in image A and B, a more faithful method must attribute less the unimportant part of the image (e.g., a dog part of the image) in image A than in image B. Similar idea was explored in [Kim+18b], except they used a surrogate metric (model's accuracy) as a source of ground truth explanations in combination with synthetic data.

Other works using similar strategies include [Wil+20b; Gha+21; PMT18; KPT21; Yeh+19b].

Examples with Real Datasets. While more difficult, it is possible to at least partially check for certain kinds of properties on real datasets that have no ground-truth.

For example, suppose a method returns a ranking of features that are most important to least, and we want to check its faithfulness to the underlying model. "Importance" of features here is defined as a set of minimum features triggering model's prediction. One can collect a model's correct prediction only with the top-1 most important feature (provided an interpretability method), then only the top-2 features, etc., then check if the delta in prediction accuracy between top- n and top- $n + 1$ decreases as n increases. Note that this evaluation assumes that there is no significant interacting between features; if features A, B, C are the top-3 features, but C is only important if feature set B

¹
² exists, this would over-estimate the importance of the feature set C .

³ Figure^{35.5} shows an example of this from [Gho+19]. The method aims to output a set of image
⁴ patches (e.g., a set of connected pixels) that correlates (if not causes) prediction. They adds top- n
⁵ image patches provided by an interpretability method, and observe the trend in accuracy backs
⁶ up their claim. Similar experiment in reverse direction (i.e., deleting top- n most important image
⁷ patches) provides additional evidence. Similar experiments are also conducted in [FV17; RSG16a].

⁸ Instead of using model's prediction accuracy as a proxy measure, one can define a faithfulness
⁹ metric that best fits their desired property. For example, in [DG17], authors define properties
¹⁰ in plain English first: Smallest sufficient region (smallest region of the image that alone allows a
¹¹ confident classification) and Smallest destroying region (smallest region of the image that when
¹² removed, prevents a confident classification), followed by careful operationalization of these properties
¹³ such that they become the objective for optimization. Then, separately, a evaluation metric of
¹⁴ saliency is defined to be " the tightest rectangular crop that contains the entire salient region and
¹⁵ to feed that rectangular region to the classifier to directly verify whether it is able to recognise the
¹⁶ requested class.". While the "rectangular" constraint may introduce artifacts, it is a neat trick to
¹⁷ make evaluation possible. By checking expected behavior as described above, authors confirm that
¹⁸ methods's behavior on the real data is aligned with the defined property compared to baselines.

¹⁹
²⁰ **Evaluating the Evaluations.** As we have seen so far, there are many ways to formalize a given
²¹ property and many empirical tests to determine whether a property is present. Each empirical test
²² will have different qualities. As an illustration, in [Tom+20], the authors ask whether popular saliency
²³ metrics give consistent results across literature. They tested whether different metrics for assessing
²⁴ the quality of a saliency-based explanations (that is, explanations that identify important pixels or
²⁵ regions in images) is evaluating similar properties. In other words, this work tests consistency and
²⁶ stability property of *metrics*. They show many metrics are statistically unreliable and inconsistent.
²⁷ While each metric may still have a particular use [Say+19], knowing this inconsistency between
²⁸ metrics helps us better understand the landscape and limitations of our evaluation approaches.
²⁹ Developing good evaluations for computational properties is an ongoing area of research.

³⁰

³¹ 35.4.2 User Study-based Evaluation: Does the Method Help a User Perform a ³² Task?

³³
³⁴ User study-based evaluations measure whether an interpretable machine learning method helps a
³⁵ human perform some task. This task could be the ultimate end-task of interest (e.g. does a method
³⁶ help a doctor make better treatment decisions) or a synthetic task that mirrors contexts of interest
³⁷(e.g. a simplified situation with artificial diseases and symptoms). In both cases, careful and rigorous
³⁸ experimental design is critical to ensuring that the experiment measures what we want it to measure.

³⁹

⁴⁰ 35.4.2.1 User Studies in Real Contexts.

⁴¹

⁴² The gold standard for testing whether an explanation is useful is to test it in the intended context:
⁴³ Do clinicians make better decisions with a certain kind of decision support? Do programmers debug
⁴⁴ code faster with a certain kind of explanation about model errors? Do developers improve fairer
⁴⁵ treatment of their customers?

⁴⁶ A complete guide on how to design and conduct user studies is out of scope for this chapter; below
⁴⁷

¹ we point out some very basic considerations and emphasize that understanding experimental design
² for user studies is essential for research in interpretable machine learning.
³

⁴
⁵ **35.4.2.2 Basic elements of user studies**
⁶

⁷ Performing a high-quality user study is a nuanced and non-trivial endeavor. There are many sources
⁸ of bias, some obvious (e.g. learning and fatigue effects during a study) and some less obvious (e.g.
⁹ participants willing to work with us are more optimistic about AI technology than those we could
¹⁰ not recruit, different participants may have different needs for cognition).
¹¹

¹² **Interface Design.** The explanation must be presented to the user, and as noted in Section 35.3.
¹³ Unlike the *intrinsic* difficulty of explaining a model (i.e., in general, complex models will necessarily
¹⁴ be harder to explain than simpler ones), the design of the interface is an *extrinsic* source of difficulty
¹⁵ (or ease) that can confound the experimental results. For example, it may be easier, in general, to
¹⁶ scan a list of important features if they are ordered by importance rather than ordered alphabetically.
¹⁷

¹⁸ When we perform an evaluation with respect to an end-task, intrinsic and extrinsic difficulties
¹⁹ can get conflated. Does one explanation type work better because it does a better job of explaining
²⁰ the complex system? Or does it work better simply because it was presented in way that was
²¹ easier for people to understand and use? Especially if the goal is to test the difference between one
²² explanation and another in the experiment, it is important that the interface for each is designed to
²³ tease out the effect from the explanations and their presentations. (Note that good presentations
²⁴ and visualization are an important but different object of study.) Moreover, if using the interface
²⁵ requires training, it is important to deliver the training in a way that is neutral in each testing
²⁶ conditions; in general, how the end-task and goals of the study are described and confirmed (e.g.
²⁷ with practice questions to test comprehension) will have a large impact on how users approach the task.
²⁸

²⁹ **Baselines.** Simply the presence of an explanation may change the way in which people interact
³⁰ with an ML system. Thus, depending on the experiment goals and setting, important baselines may
³¹ include: how a human performs with no ML system; an ML system with no explanation; an ML
³² system with a placebo explanation (an explanation that provides no information); an ML system with
³³ hand-crafted explanations (manually generated by humans who are presumably good communicators).
³⁴

³⁵ **Experimental Design and Hypothesis Testing.** Independent and dependent variables, hy-
³⁶ potheses, and inclusion and exclusion criteria must be clearly defined prior to the start of the study.
³⁷ For example, suppose that one hypothesizes that a particular explanation will help a developer debug
³⁸ an image classifier. In this case, the independent variable would be a form of assistance: various
³⁹ explanations, perhaps to be compared to no explanation at all. The dependent variable would be
⁴⁰ whether the developer can identify bugs. Inclusion and exclusion criteria might include a requirement
⁴¹ that the developer has sufficient experience training image classifiers (as determined by an initial
⁴² survey, or even a pre-test), demonstrates engagement (as measured by a base level of performance on
⁴³ some initial practice rounds), and does not have prior experience with the particular explanation
⁴⁴ types (as determined by an initial survey). Other exclusion criteria could be not using data from
⁴⁵ any participant that takes an unusually long or short time to perform the end-task (as proxies for
insufficient engagement).

⁴⁶ As noted in Section 35.2, there are many decisions that go into any interpretable machine learning
⁴⁷

¹ method, and the specific context may have many nuances. Studies of the form “Does explanation
² X (computed via some pipeline Y) help users in context Z compared to explanation X' ?” may not
³ provide much insight as to *why* that particular explanation is better or worse—making it harder not
⁴ only to iterate on a particular explanation but also to generalize to other explanations or contexts.
⁵ There are many, many sources of potential variation in the results, ranging from the properties of
⁶ the explanation and its presentation to the randomness and difficulty of the task.
⁷

⁸ To reduce this variance, and to get more useful and generalizable insights, one may want to, at
⁹ least in initial stages, manipulate some of these factors directly. For example, if the research question
¹⁰ is whether complete explanations are better than incomplete explanations in a particular context,
¹¹ then one might write out, by hand, explanations that are complete in what features they implicate,
¹² explanations in which one important feature is missing, and explanations in which several important
¹³ features are missing—being careful to choose the missing features either at random or in a specific
¹⁴ way. Doing so ensures even coverage of the different experimental regimes of interest, which may
¹⁵ not occur if the explanations were simply output from a pipeline. As another example, one might
¹⁶ intentionally create an image classifier with known bugs, or simply pretend to have an image classifier
¹⁷ that makes certain predictions, so one knows exactly what should be found and the classifier’s error
¹⁸ rate (as done in [Ade+20b]). These kinds of studies are called *wizard-of-oz* studies, and they can
¹⁹ help us more precisely uncover the science of why an explanation is useful (e.g. as done in [Jac+21]).
²⁰

²¹ Once the independent and dependent variables, hypotheses, and participant criteria (including
²² how the independent and dependent variables may be manipulated) are determined, the next step
²³ is setting up the study design itself. Broadly speaking, randomization—whether it is in assigning
²⁴ subjects to end-tasks or ordering end-tasks—marginalizes over various potential confounds (e.g.
²⁵ subject prior knowledge or learning effects). Matching (e.g., asking the same subject to perform the
²⁶ same or equivalent end-tasks with two different explanations) reduces variance. Repeated measures
²⁷ (e.g., asking the subject to perform the end-task for several different inputs) also reduces variance.

²⁸ In the kinds of studies that one is likely to be performing for user studies in interpretable machine
²⁹ learning, block randomized designs, or more generally, Latin square designs, can be used, for example,
³⁰ to randomize the order of explanation types while keeping tasks associated with each explanation type
³¹ grouped together, can be used to marginalize over the effects of learning and fatigue. Careful consid-
³² eration should be given to what is tested within subjects (a form of matching that produces lower
³³ variance, but adds potential bias from learning effects from the first task to the second) and across sub-
³⁴ jects (higher variance, potential bias from population shift during experiment recruitment). Finally,
³⁵ each of these study designs, as well as the choice of independent and dependent variables, will imply
³⁶ an appropriate significance test (e.g., ANOVA, bonferroni correction). It is essential to choose the
³⁷ right test and multiple hypothesis correction to avoid inflated significance values while retaining power.

³⁸ **Qualitative Studies.** So far, we have described the standard approach for the design of a
³⁹ quantitative user study—one in which the dependent variable is numerically measured (e.g. time
⁴⁰ taken to correctly identify a bug, % bugs detected). While quantitative studies provide value by
⁴¹ demonstrating that there is a consistent, quantifiable effect across many users, they usually do not
⁴² tell us *why* a certain explanation worked. In contrast, qualitative studies, often performed with a
⁴³ “think-aloud” or other discussion-based protocol in which users expose their thought process as they
⁴⁴ perform the experiment, can help identify why a particular form of explanation seems to be useful
⁴⁵ (or not), as the experimenter can gain insights by hearing how the user was using the information,
⁴⁶ and depending on the protocol, can ask for clarifications.

⁴⁷

1

2 For example, suppose one is interested in how people use an example-based explanation to
3 understand a video-game agent’s policy. The idea is to show a few video clips of an automated
4 agent in the video game, and then ask the user what the agent might do in novel situations. In a
5 think-aloud study, the user would perform this task while talking through how they are connecting
6 the videos they have seen to the new situation. By hearing these thoughts, a researcher might not
7 only gain deeper insight into how users make these connections—e.g. users might see the agent
8 collect coins in one video and presume that the agent will always go after coins (a goal directed
9 viewpoint, vs. the agent will go left or right in the presence of a coin)—to refine the explanation, but
10 they might also identify surprising bugs: for example, a user might see the agent fall into a pit and
11 attribute it to a one-off sloppy fingers, not internalizing that an automated agent might make that
12 mistake every time.

13

14 While a participant in a think-aloud study is typically more engaged in the study than the might
15 be otherwise (because they are describing their thinking), knowing their thoughts can provide insight
16 into the causal process between what information is being provided by the explanation and the
17 action that the human user takes, ultimately helping advance the science of how people interact with
18 machine-provided information.

19

20 **Pilot Studies:** The above descriptions are just a very high-level overview of the many factors
21 that must be designed properly for a high-quality evaluation. In practice, one do not typically get all
22 of these right the first time. Small scale pilot studies are essential to checking whether participants
23 attend to the provided information in unexpected ways (or not at all), whether instructions are
24 clear and well-designed, etc. Modifying the experiments after iterative small scale pilot studies can
25 save a lot of time and energy down the road. In these pilots, one should collect not only the usual
26 information about users and the dependent variables, but also discuss with the participants how they
27 approached the study tasks and whether elements were confusing. These discussions will lead to
28 insights and confidence that the study is testing what it is intended to test. The results from pilot
29 studies should be not included in the final results.

30

31 Finally, as the number of factors to test increases (e.g., baselines, independent variables), the study
32 design becomes more complex and may require more participants and longer participation times
33 to determine if the results are significant—which can in turn increase costs and effects of fatigue.
34 Pilots, think-aloud studies, and careful thinking about what aspects of the evaluation require user
35 studies and what can be completed computationally (e.g. if one explanation has significantly worse
36 properties than another) can all help distill down a user-based evaluation to the most important
37 factors.

38

39 35.4.2.3 User Studies in Synthetic Contexts

40

41 It is not always appropriate or possible to test an interpretable machine learning method in the real
42 context: for example, it would be unethical to test a prototype explanation system on patients each
43 time one has a new way to convey information about a treatment recommendation. In such cases, we
44 might want to run an experiment in which clinicians perform a task on made-up patients, or in some
45 analogous non-medical context where the participant pool is bigger and more affordable. Similarly,
46 one might create a relatively accessible image classification debugging context where one can control
47 the incorrect labels, distribution shifts, etc. (e.g. [Ade+20b]) and see what and if explanations help
users detect problems in this simpler setting. Using simpler setting could be a way to shed light on

1 what properties are important for debugging image classification more broadly. Synthetic contexts
2 may also allow us to study more general properties of an interpretable machine learning method. For
3 example, we may be interested in how different forms of explanation have different cognitive loads
4 or how a particular property affects performance (e.g., [Lag+19]). The same principles we outlined
5 above for user studies in real contexts continue to apply, but there are some important cautions.
6

7 **Cautions regarding synthetic contexts:** While user studies with synthetic contexts can be
8 valuable for identifying scientific principles, one must still be cautious when testing in situations that
9 would be difficult. For example, experimental subjects in a synthetic high-stakes context may not
10 treat the stakes of the problem as seriously, may be relatively unburdened with respect to distractions
11 or other demands on their time and attention (e.g. a quiet study environment vs. a chaotic hospital
12 floor), and ignore important factors of the task (e.g., a participant ignores some of the information
13 to complete the task as quickly as possible). Moreover, small differences in task definition can have
14 big effects: even the difference between asking users to simply perform a task with an explanation
15 available vs. asking users to answer some questions about the explanation first, may create very
16 different results as the latter forces the user to pay attention to the explanation and the former does
17 not. Priming users by giving them specific scenario where they can put themselves into a mindset
18 (e.g., imagine now you are an engineer at a company trying to sell a classifier. The deadline is
19 approaching and your boss asked for A) could also help.

20

21

22 35.5 Discussion: How to Think about Interpretable Machine Learning

23

24 Interpretable machine learning is a young, interdisciplinary field of study. As a result, consensus
25 on definitions, evaluation methods, and appropriate abstractions is still forming. The goal of this
26 section is to lay out a core set of principles about interpretable machine learning. While specifics in
27 the previous sections may change, the principles below will be more durable.

28

29 **There is no universal, mathematical definition of interpretability, and there never**
30 **will be. Defining a downstream performance metric (and justifying it) for each context**
31 **is a must.** The information that best communicates to the human what is needed to perform a
32 task will necessarily vary: for example, what a clinical expert needs to convince themselves that a
33 proposed treatment policy is worth trying on patients is very different than what a person whose
34 loan was just denied needs to determine what they need to change to get an approval. Similarly,
35 methods to communicate characteristics of models built on pixel data may not be appropriate for
36 communicating characteristics of models built on tabular data. We may hope to identify desired
37 properties in explanations to maximize downstream task performance for different classes of end
38 tasks—that is the grand challenge of interpretable machine learning—but ultimately, there will never
39 be one metric for all contexts.
40

41 While this lack of a universal metric may feel disappointing at first, this is a common in other areas
42 of machine learning when the techniques are used in real applications. For example, there exist many
43 metrics for fairness, and not only is it impossible to satisfy them all at the same time [KMR16], but
44 also in a particular situation, none of them may exactly match the desires of an algorithm designer
45 and stakeholders. Even in a standard classification setting, there are many metrics that correspond
46 to make the predicted and true labels as close as possible. Does one care about overall accuracy?

47

35.5. DISCUSSION: HOW TO THINK ABOUT INTERPRETABLE MACHINE LEARNING

¹ Sensitivity? Specificity? It is unlikely that one objective captures everything that is needed in one
² situation, much less across different contexts. Evaluation can still be rigorous as long as assumptions
³ and requirements are made precise.

⁴ What sets interpretable machine learning apart from other areas of machine learning, however, is
⁵ that a large class of evaluations require human input. As a necessarily interdisciplinary area, rigorous
⁶ work in interpretable machine learning requires not only knowledge how to approach computation
⁷ and statistics rigorously but also how approach experimental design and user studies rigorously.
⁸

⁹ **Interpretability is only a part of the solution for fairness, calibrated trust, accountability, causality and other important problems.** Learning models that are fair, safe, causal,
¹⁰ or engender calibrated trust are all goals, whereas interpretability is one means toward that goal.
¹¹

¹² In some cases, we don't need interpretability. For example, if the goal can be fully formalized in
¹³ mathematical terms (e.g., a regulatory requirement may mandate quotas or thresholds on specific
¹⁴ fairness metrics that a model must meet), we do not need any human input. If a model behaves
¹⁵ safely across an exhaustive set of pre-defined inputs, then it may be less important to understand
¹⁶ how it produced its outputs. Similarly, if a model performs well across a variety of regimes, that
¹⁷ might (appropriately) increase one's trust in it; if it makes errors, that might (appropriately) decrease
¹⁸ trust without an inspection of any of the system's internals.
¹⁹

²⁰ In other cases, human input is needed to achieve the end-task. For example, while there is much
²¹ excellent work in the identification of causal models, under many circumstances, it is not possible to
²² learn a model that is *guaranteed* to be causal from a dataset alone. Here, interpretability could be
²³ means for the end-task of causality by allowing a human to inspect the model's causal mechanism.
²⁴

²⁵ As another example, one could measure the safety of a clinical decision support system by tracking
²⁶ how often following its recommendations causes harm to patients—and stop using the system if it is
²⁷ causing too much harm. However, if we use this approach to safety, we will only discover that the
²⁸ system is unsafe *after* a significant number of patients have been harmed. Here, interpretability
²⁹ could support the end-task of safety by allowing clinical experts to understand if the model's decision
process has any red flags *prior* to deployment to minimize the potential harm.

³⁰ In general, complex contexts and end-tasks will require a constellation of methods (and people) to
³¹ achieve them. For example, formalizing a complex notion such as accountability will require a broad
³² collection of people—from policy makers and ethicists to corporations, engineers, and users—unifying
³³ vocabularies, exchanging domain knowledge, and identifying goals. Evaluating or monitoring it will
³⁴ involve various empirical measures of quality and insights from interpretability. While methods
³⁵ research in interpretable machine learning will often, by necessity, focus on a specific aspect of an
³⁶ interpretable machine learning approach, in many real settings, interpretability will be one part of
³⁷ responsible machine learning.
³⁸

³⁹ **Interpretability is distinct from full transparency into the model or knowing the model's code** Staring at the weights of every neuron in a large network is likely to be as effective
⁴⁰ as taking one's laptop apart to understand a bug in your code. There are many good reasons for
⁴¹ open source projects and models, but open source code itself is not interpretable: A typical user
⁴² will not be able to reason through 100K lines of parameters despite having all the pieces available.
⁴³ Interpretability is not just about the mechanism of the complex model itself, it's equally about the
⁴⁴ users who consume the explanations for their end-task.
⁴⁵

¹ **Interpretability is not about understanding everything about the model; it is about**
² **understanding enough to do the end-task.** The ultimate measure of an interpretable machine
³ learning method is whether it helps the user perform their end-task well. Suppose the end-task is
⁴ to fix an overheating laptop. If one knows the likely sources of the heat, the laptop is probably
⁵ interpretable enough to be fixed, even if one does not know the chemical properties of its components.
⁶ On the other hand, if the laptop keeps freezing up, knowing about the sources of heat may not be the
⁷ right information. Importantly, both end-tasks are have clear downstream performance metrics: we
⁸ can observe whether the information helped the user perform actions that make the laptop overheat
⁹ or freeze up less.

¹⁰ As another example, consider AlphaGo, Google DeepMind’s AI go player that beat the human
¹¹ world champion, Lee SeDol. The model is so complex that one cannot fully understand its decision
¹² process, including surprising moves like its famous move 37[Met16]. That said, partial probes (e.g.,
¹³ does AlphaGo believe the same move would have made a different impact if it was made earlier but
¹⁴ similar position in the game) might still help a Go expert gain insights on the rationale for the move
¹⁵ in the context of what they already know about the game.

¹⁶ **However, any partial view of a model is, necessarily, only a partial view; it does not**
¹⁷ **tell the full story.** While we just argued that many end-tasks do not require knowing everything
¹⁸ about a model, we also need to understand the flip-side of that statement for the sake of caution;
¹⁹ if the human user does not understand everything about the model, then that implies there are
²⁰ (potentially important) parts of the model that the user is not aware of. A set of features to change
²¹ in order to flip a loan decision is a partial view of a model, and it might be what a user needs to
²² know prior to resubmitting an application. However, one partial view might not provide enough
²³ information to vet if the model is discriminatory. Specifically, any probe will only return what it is
²⁴ designed to compute (e.g., an approximation of a complex function with a simpler one). Different
²⁵ probes may be able to reveal different properties at different levels of quality. Incorrectly believing
²⁶ the partial view is the full story could result in incorrect insights.

²⁷ **Partial Views can Lack Stability and Enable Attacks.** Relatedly, any explanation that
²⁸ reveals only certain parts of a model can lack stability (e.g. see [AMJ18a])and can be more
²⁹ easily attacked (e.g. see [Yeh+19a; GAZ19; Dom+19; Sla+20]). Especially when models are
³⁰ overparameterized such as neural networks, it is possible to learn models whose explanations say
³¹ one thing (e.g. a feature is not important, according to some formalization of feature importance)
³² while the model does another (e.g. uses the prohibited feature). Joint training only exacerbates the
³³ issue, as it allows the model to learn boundaries that pass some partial-view test while in reality
³⁴ violating the underlying constraint. Other adversarial approaches can work on the input, minimally
³⁵ perturbing it to change explanation while keeping the prediction constant or to change the prediction
³⁶ while keeping the explanation constant (again, for an explanation that is a specific partial view).

³⁷ These concerns highlight an important open area: We need to improve ways to endow explanations
³⁸ with the property of translucence, that is, explanations that communicate what they can and cannot
³⁹ say about the model. Translucence is important because misinterpreted explanations that happen to
⁴⁰ favor a user’s views create false basis for trust.

⁴¹ **Trade-offs between Inherently Interpretable Models and Performance often do not**
⁴² **exist; Partial Views can help when they do.**

⁴³

1 While some have claimed that there exists a trade-off between using an inherently-interpretable
2 model and performance (defined as a model's performance on some test data), this trade-off rarely
3 exists in practice for several reasons[Rud19].

4 First, in many cases, the data can be surprisingly well-fit by a fairly simple model (due to high
5 noise, for example) or a model that can be decomposed into interpretable parts. One can often find a
6 framing—an architecture, a regularizer, an optimizer—that produces inherently interpretable models
7 with performance comparable to, or sometimes even better than, blackbox approaches [Wan+17a;
8 LCG12; Car+15; Let+15b; UR16; FHDV20; KRS14]. In fact, interpretability and performance can
9 be synergistic: inherently interpretable models often encode a preference for simpler models (consider
10 an L1 regularizer that can enforce sparsity but initially developed to increase performance), which
11 can result in models that are also often more robust [RDV18].

12 Second, a narrow focus on the trade-off between using inherently interpretable models and a
13 predefined metric of performance, as usually measured on a validation set, overlooks a broader issue:
14 that predefined metric of performance may not tell the full story about the quality of the model. For
15 example, using an inherently interpretable model may enable a person realize that a prediction is
16 based on confounding, not causation—or other ways it might fail in deployment. In this way, one
17 might get better performance with an inherently interpretable model in practice even if a blackbox
18 appears to have better performance numbers in validation. An inherently interpretable model may
19 also enable better human+model teaming by allowing the human user to step in and override the
20 system appropriately.

21 **Human factors are essential.** All machine learning systems ultimately connect to broader socio-
22 technical contexts. However, in many cases, many aspects of model construction and optimization can
23 be performed in a purely computational setting: there are techniques to check for appropriate model
24 capacity, techniques for tuning a gradient descent or convex optimization. In contrast, interpretable
25 machine learning must consider human factors **from the beginning**: there is no point optimizing
26 an explanation to have various properties if it still fails to improve the user's performance on the
27 end-task—or even decreases their performance.

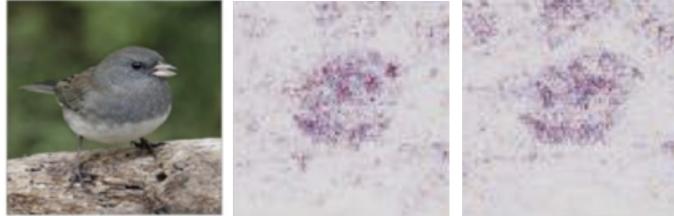
28 *Over-reliance.* Just because an explanation is present, does not mean that the user will analytically
29 and reasonably incorporate the information provided into their ultimate decision-making task. Indeed,
30 the presence of *any* explanation can increase a user's trust in the model (it even has a justification
31 for its recommendation!), exacerbating the general issue of over-trust in human+ML teams. Recent
32 studies have found that even data scientists often misuse interpretability tools because they do not
33 interpret the explanations in the way the tool intended, resulting inappropriately increased confidence.
34 The authors note that the participants' excitement about the tool led them to take it at face-value
35 rather than dig deeper. [LM20] reports a similar finding, noting that evocative presentations can
36 create a feeling of comprehension even if the information is not being presented accurately.

37 Over-reliance can be combated with explicit measures to force the user to engage analytically and
38 skeptically with the information in the explanation. For example, one could ask the user to submit
39 their decision first and only then show the recommendation and accompanying explanation to pique
40 their interest in why their choice and the recommendation might disagree (and prompting whether
41 they want to change their choice). Another option might be to ask the user some basic question
42 about the explanation prior to submitting their decision to force them to look at the explanation
43 carefully. Yet another might be to provide only the relevant information (the explanation) without
44 the recommendation, forcing the user to combine the additional information with their own. However,
45

46

1
2
3
4
5
6
7
8
9
10

Original Image



11
12 *Figure 35.6: (Potential) perception issues: an explanation from a trained network (left) is visually indistin-*
guishable to humans from one from an untrained network (right)—even if they are not exactly identical.
13

14
15

16 in all these cases, there is a delicate balance: users will often be amenable to expending additional
17 cognitive effort if they can see it achieves better results, but if they feel the effort is too much, they
18 may start ignoring the model entirely.

19

20 *Potential for Misuse.* A worse version of over-reliance is when explanations are used to manipulate
21 a user—such as increase trust—rather than facilitating an end-task of the user’s interest—such as
22 calibrating their trust. Furthermore, users may report that they *like* explanations are simple, require
23 little cognitive effort, etc. even when those explanations do not help them perform their end-task.
24 As creators of interpretable machine learning methods, one must be on alert to ensure that the
25 explanations help the user achieve what they want to (ideally in a way that they also like).

26

27 *Misunderstandings from a lack of understanding of machine learning.* Even when correctly engaged,
28 users in different contexts will have different levels of knowledge about machine learning. For example,
29 not everyone may understand what it means for factors to be additive or the implications of feature
30 importances selected based on Shapely values [Sha16]. Users may also attribute more understanding
31 to a model than it actually has. For example, if they see a set of pixels highlighted around a beak, or
32 a set of topic model terms about a disease, they may mistakenly believe that the machine learning
33 model has some notion of concepts that matches theirs, when the truth might be quite different.

34

35 *Perception issues.* In some cases, the explanation may not be truly understandable to the human.
36 For example, due to the inherent biases of our visual perception, humans may not perceive the
37 difference between different explanatory images. In Figure 35.6, two explanations (in terms of
38 important pixels in a bird image) seem to communicate similar message; for most people, both
39 explanations seem to suggest that the belly and cheek of the bird are the important parts for this
40 prediction. However, one of them is generated from a trained network (left), but the other one is
41 from a network that returns random predictions (right). While the two saliency maps aren’t identical
42 to machines, they look similar because humans don’t parse an image as pixel values, but as whole:
43 they see a bird in both pictures.

44 Another common issue with pixel-based explanations is that explanation creators often multiply
45 the original image with an importance “mask” (black and clear saliency mask, where black pixel
46 represents no importance and a clear pixel represents maximum importance), introducing the arbi-
47

35.5. DISCUSSION: HOW TO THINK ABOUT INTERPRETABLE MACHINE LEARNING

1 trary artifact that black objects never appear important. In addition, this binary mask is produced
2 by clipping important pixels in certain percentile (e.g., only taking 99-th percentile), which can
3 also introduce another artifact [Sun+19c]. The balancing act between artifacts introduced by visu-
4 alization for the ease of understanding and faithfully representing the explanation remains a challenge.
5

6 Together, all of these points on human factors emphasize what we said from the start: we cannot
7 divorce the study and practice of intepretable machine learning from its intended socio-technical
8 context.
9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

Part VI

Decision making

36 Multi-step decision problems

36.1 Introduction

We introduced the basics of **Bayesian decision theory** in Section 3.8. In this chapter, we consider applications and extensions of this to scenarios in which the agent must make a sequence of decisions.

36.2 Decision (influence) diagrams

When dealing with structured multi-stage decision problems, it is useful to use a graphical notation called an **influence diagram** [HM81; KM08], also called a **decision diagram**. This extends directed probabilistic graphical models (Chapter 4) by adding **decision nodes** (also called **action nodes**), represented by rectangles, and **utility nodes** (also called **value nodes**), represented by diamonds. The original random variables are called **chance nodes**, and are represented by ovals, as usual.

36.2.1 Example: oil wildcatter

As an example (from [Rai68]), consider creating a model for the decision problem faced by an oil “**wildcatter**”, which is a person who drills wildcat wells, which are exploration wells drilled in areas not known to be oil fields.

Suppose you have to decide whether to drill an oil well or not at a given location. You have two possible actions: $d = 1$ means drill, $d = 0$ means don’t drill. You assume there are 3 states of nature: $o = 0$ means the well is dry, $o = 1$ means it is wet (has some oil), and $o = 2$ means it is soaking (has a lot of oil). We can represent this as a decision diagram as shown in Figure 36.1(a).

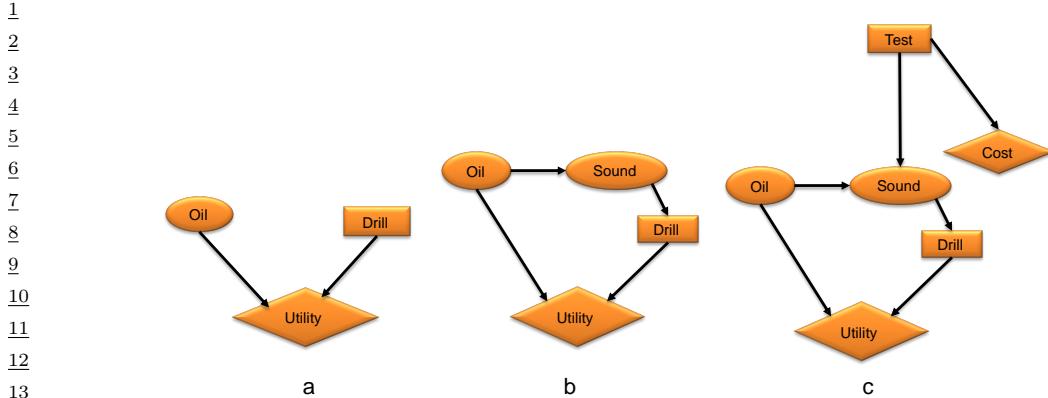
Suppose your prior beliefs are $p(o) = [0.5, 0.3, 0.2]$, and your utility function $U(d, o)$ is specified by the following table:

		$o = 0$	$o = 1$	$o = 2$
		0	0	0
$d = 0$	0	-70	50	200
	1	100	150	250

We see that if you don’t drill, you incur no costs, but also make no money. If you drill a dry well, you lose \$70; if you drill a wet well, you gain \$50; and if you drill a soaking well, you gain \$200.

What action should you take if you have no information beyond your prior knowledge? Your prior expected utility for taking action d is

$$\text{EU}(d) = \sum_{o=0}^2 p(o)U(d, o) \quad (36.1)$$



14 Figure 36.1: Influence diagrams for the oil wild catter problem. Ovals are random variables (chance nodes),
15 squares are decision (action) nodes, diamonds are utility (value) nodes. (a) Basic model. (b) An extension in
16 which we have an information arc from the Sound chance node to the Drill decision node. (c) An extension
17 in which we get to decide whether to perform a test or not, as well as whether to drill or not.

²⁰We find $\text{EU}(d = 0) = 0$ and $\text{EU}(d = 1) = 20$ and hence the maximum expected utility is

$$MEU = \max\{EU(d=0), EU(d=1)\} = \max\{0, 20\} = 20 \quad (36.2)$$

Thus the optimal action is to drill, $d^* = 1$.

26 36.2.2 Information arcs

²⁷ Now let us consider a slight extension to the model, in which you have access to a measurement
²⁸(called a “sounding”), which is a noisy indicator about the state of the oil well. Hence we add an
²⁹ $O \rightarrow S$ arc to the model. In addition, we assume that the outcome of the sounding test will be
³⁰available before we decide whether to drill or not; hence we add an **information arc** from S to D .
³¹This is illustrated in Figure 36.1(b). Note that the utility depends on the action and the true state
³²of the world, but not the measurement.

³³ We assume the sounding variable can be in one of 3 states: $s = 0$ is a diffuse reflection pattern,
³⁴ suggesting no oil; $s = 1$ is an open reflection pattern, suggesting some oil; and $s = 2$ is a closed
³⁵ reflection pattern, indicating lots of oil. Since S is caused by O , we add an $O \rightarrow S$ arc to our model.
³⁶ Let us model the reliability of our sensor using the following conditional distribution for $p(S|O)$:
³⁷

	$s = 0$	$s = 1$	$s = 2$
$o = 0$	0.6	0.3	0.1
$o = 1$	0.3	0.4	0.3
$o = 2$	0.1	0.4	0.5

⁴² Suppose the sounding observation is s . The posterior expected utility of performing action d is

$$\frac{44}{45} \quad \text{EU}(d|s) = \sum_{o=0}^2 p(o|s)U(o, d) \quad (36.3)$$

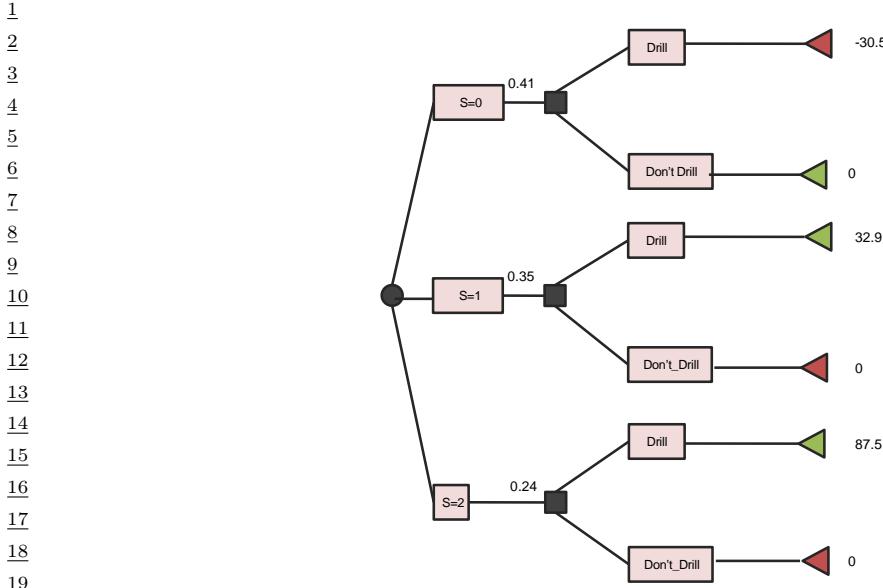


Figure 36.2: Decision tree for the oil wildcatter problem. Black circles are chance variables, black squares are decision nodes, diamonds are the resulting utilities. Green leaf nodes have higher utility than red leaf nodes.

We need to compute this for each possible observation, $s \in \{0, 1, 2\}$, and each possible action, $d \in \{0, 1\}$. If $s = 0$, we find the posterior over the oil state is $p(o|s = 0) = [0.732, 0.219, 0.049]$, and hence $\text{EU}(d = 0|s = 0) = 0$ and $\text{EU}(d = 1|s = 0) = -30.5$. If $s = 1$, we similarly find $\text{EU}(d = 0|s = 1) = 0$ and $\text{EU}(d = 1|s = 1) = 32.9$. If $s = 2$, we find $\text{EU}(d = 0|s = 2) = 0$ and $\text{EU}(d = 1|s = 2) = 87.5$. Hence the optimal policy $d^*(s)$ is as follows: if $s = 0$, choose $d = 0$ and get \$0; if $s = 1$, choose $d = 1$ and get \$32.9; and if $s = 2$, choose $d = 1$ and get \$87.5.

The maximum expected utility of the wildcatter, before seeing the experimental sounding, can be computed using

$$\text{MEU} = \sum_s p(s) \text{EU}(d^*(s)|s) \quad (36.4)$$

where prior marginal on the outcome of the test is $p(s) = \sum_o p(o)p(s|o) = [0.41, 0.35, 0.24]$. Hence the MEU is

$$\text{MEU} = 0.41 \times 0 + 0.35 \times 32.9 + 0.24 \times 87.5 = 32.2 \quad (36.5)$$

These numbers can be summarized in the **decision tree** shown in Figure 36.2.

36.2.3 Value of information

Now suppose you can choose whether to do the test or not. This can be modelled as shown in Figure 36.1(c), where we add a new test node T . If $T = 1$, we do the test, and S can enter states

¹ $\{0, 1, 2\}$, determined by O , exactly as above. If $T = 0$, we don't do the test, and S enters a special
² unknown state. There is also some cost associated with performing the test.
³

⁴ Is it worth doing the test? This depends on how much our MEU changes if we know the outcome
⁵ of the test (namely the state of S). If you don't do the test, we have $\text{MEU} = 20$ from Equation (36.2).
⁶ If you do the test, you have $\text{MEU} = 32.2$ from Equation (36.5). So the improvement in utility if
⁷ you do the test (and act optimally on its outcome) is \$12.2. This is called the **value of perfect**
⁸ **information** (VPI). So we should do the test as long as it costs less than \$12.2.

⁹ In terms of graphical models, the VPI of a variable S can be determined by computing the MEU
¹⁰ for the base influence diagram, \mathcal{G} , in Figure 36.1(b), and then computing the MEU for the same
¹¹ influence diagram where we add information arcs from S to the action node, and then computing the
¹² difference. In other words,

$$\frac{13}{14} \quad \text{VPI} = \text{MEU}(\mathcal{G} + S \rightarrow D) - \text{MEU}(\mathcal{G}) \quad (36.6)$$

¹⁵ where D is the decision node and S is the variable we are measuring. This will tell us whether it is
¹⁶ worth adding obtaining measurement S .
¹⁷

¹⁸ 36.2.4 Computing the optimal policy

²⁰ In general, given an influence diagram, we can compute the optimal policy automatically by modifying
²¹ the variable elimination algorithm (Section 9.4), as explained in [LN01; KM08]. The basic idea is to
²² work backwards from the final action, computing the optimal decision at each step, assuming all
²³ following actions are chosen optimally. When the influence diagram has a simple chain structure,
²⁴ as in a Markov decision process (Section 36.5), the result is equivalent to Bellman's equation
²⁵ (Section 36.5.5).
²⁶

²⁷

²⁸ 36.3 A/B testing

²⁹

³⁰ Suppose you are trying to decide which version of a product is likely to sell more, or which version of
³¹ a drug is likely to work better. Let us call the versions you are choosing between A and B; sometimes
³² version A is called the **control**, and version B is called the **treatment**. (Sometimes the different
³³ actions are called "**arms**".)

³⁴ A very common approach to such problems is to use an **A/B test**, in which you try both actions
³⁵ out for a while, by randomly assigning a different action to different subsets of the population, and
³⁶ then you measure the resulting accumulated **reward** from each action, and you pick the winner.
³⁷ (This is sometimes called a "**test and roll**" approach, since you test which method is best, and then
³⁸ roll it out for the rest of the population.)

³⁹ A key problem in A/B testing is to come up with a decision rule, or policy, for deciding which
⁴⁰ action is best, after obtaining potentially noisy results during the test phase. Another problem is
⁴¹ to choose how many people to assign to the treatment, n_1 , and how many to the control, n_0 . The
⁴² fundamental tradeoff is that using larger values of n_1 and n_0 will help you collect more data and
⁴³ hence be more confident in picking the best action, but this incurs an **opportunity cost**, because
⁴⁴ the testing phase involves performing actions that may not result in the highest reward. (This is
⁴⁵ an example of the exploration-exploitation tradeoff, which we discuss more in Section 36.4.3.) In
⁴⁶ this section, we give a simple Bayesian decision theoretic analysis of this problem, following the
⁴⁷

¹ presentation of [FB19].¹ More details on A/B testing can be found in [KTX20].

⁴ 36.3.1 A Bayesian approach

⁵ We assume the i 'th reward for action j is given by $Y_{ij} \sim \mathcal{N}(\mu_j, \sigma_j^2)$ for $i = 1 : n_j$ and $j = 0 : 1$, where
⁶ $j = 0$ corresponds to the control (action A), $j = 1$ corresponds to the treatment (action B), and n_j is
⁷ the number of samples you collect from group j . The parameters μ_j are the expected reward for
⁸ action j ; our goal is to estimate these parameters. (For simplicity, we assume the σ_j^2 are known.)

⁹ We will adopt a Bayesian approach, which is well suited to sequential decision problems. For
¹⁰ simplicity, we will use Gaussian priors for the unknowns, $\mu_j \sim \mathcal{N}(m_j, \tau_j^2)$, where m_j is the prior
¹¹ mean reward for action j , and τ_j is our confidence in this prior. We assume the prior parameters are
¹² known. (In practice we can use an empirical Bayes approach, as we discuss in Section 36.3.2.)

¹⁴ 36.3.1.1 Optimal policy

¹⁵ Initially we assume the sample size of the experiment (i.e. the values n_1 for the treatment and n_0 for
¹⁶ the control) are known. Our goal is to compute the optimal policy or decision rule $\pi(\mathbf{y}_1, \mathbf{y}_0)$, which
¹⁷ specifies which action to deploy, where $\mathbf{y}_j = (y_{1j}, \dots, y_{n_j j})$ is the data from action j .

¹⁸ The optimal policy is simple: choose the action with the greater expected posterior expected
¹⁹ reward:

$$\pi^*(\mathbf{y}_1, \mathbf{y}_0) = \begin{cases} 1 & \text{if } \mathbb{E}[\mu_1 | \mathbf{y}_1] \geq \mathbb{E}[\mu_0 | \mathbf{y}_0] \\ 0 & \text{if } \mathbb{E}[\mu_1 | \mathbf{y}_1] < \mathbb{E}[\mu_0 | \mathbf{y}_0] \end{cases} \quad (36.7)$$

²² All that remains is to compute the posterior over the unknown parameters, μ_j . Applying Bayes'
²³ rule for Gaussians (Equation (2.59)), we find that the corresponding posterior is given by

$$p(\mu_j | \mathbf{y}_j, n_j) = \mathcal{N}(\mu_j | m_j, \tau_j^2) \quad (36.8)$$

$$1/\tau_j = n_j/\sigma_j^2 + 1/\tau_j^2 \quad (36.9)$$

$$m_j / \tau_j = n_j \bar{y}_j / \sigma_j^2 + m_j / \tau_j^2 \quad (36.10)$$

³³ We see that the posterior precision (inverse variance) is a weighted sum of the prior precision plus n_j
³⁴ units of measurement precision. We also see that the posterior precision weighted mean is a sum of
³⁵ the prior precision weighted mean and the measurement precision weighted mean.

³⁶ Given the posterior, we can plug m_j into Equation (36.7). In the fully symmetric case, where
³⁷ $n_1 = n_0$, $m_1 = m_0 = m$, $\tau_1 = \tau_0 = \tau$, and $\sigma_1 = \sigma_0 = \sigma$, we find that the optimal policy is to simply
³⁸ "pick the winner", which is the arm with higher empirical performance:

$$\pi^*(\mathbf{y}_1, \mathbf{y}_0) = \mathbb{I}\left(\frac{m}{\tau^2} + \frac{\bar{y}_1}{\sigma^2} > \frac{m}{\tau^2} + \frac{\bar{y}_0}{\sigma^2}\right) = \mathbb{I}(\bar{y}_1 > \bar{y}_0) \quad (36.11)$$

⁴² However, when the problem is asymmetric, we need to take into account the different sample sizes
⁴³ and/or different prior beliefs.

⁴⁵ 1. For a similar set of results in the time-discounted setting, see <https://chris-said.io/2020/01/10/optimizing-sample-sizes-in-ab-testing-part-I>.

1
2 **36.3.1.2 Optimal sample size**

3 We now discuss how to compute the optimal sample size for each arm of the experiment, i.e., the
4 values n_0 and n_1 . We assume the total population size is N , and we cannot reuse people from the
5 testing phase,

6 The prior expected reward in the testing phase is given by

7

$$\mathbb{E}[R_{\text{test}}] = n_0 m_0 + n_1 m_1 \quad (36.12)$$

8 The expected reward in the roll phase depends on the decision rule $\pi(\mathbf{y}_1, \mathbf{y}_0)$ that we use:

9

$$\mathbb{E}_{\pi}[R_{\text{roll}}] = \int_{\mu_1} \int_{\mu_0} \int_{\mathbf{y}_1} \int_{\mathbf{y}_0} (N - n_1 - n_0) (\pi(\mathbf{y}_1, \mathbf{y}_0) \mu_1 + (1 - \pi(\mathbf{y}_1, \mathbf{y}_0)) \mu_0) \quad (36.13)$$

10

$$\times p(\mathbf{y}_0 | \mu_0) p(\mathbf{y}_1 | \mu_1) p(\mu_0) p(\mu_1) d\mathbf{y}_0 d\mathbf{y}_1 d\mu_0 d\mu_1 \quad (36.14)$$

11 For $\pi = \pi^*$ one can show that this equals

12

$$\mathbb{E}[R_{\text{roll}}] \triangleq \mathbb{E}_{\pi^*}[R_{\text{roll}}] = (N - n_1 - n_0) \left(m_1 + e\Phi\left(\frac{e}{v}\right) + v\phi\left(\frac{e}{v}\right) \right) \quad (36.15)$$

13 where ϕ is the Gaussian pdf, Φ is the Gaussian cdf, $e = m_0 - m_1$ and

14

$$v = \sqrt{\frac{\tau_1^4}{\tau_1^2 + \sigma_1^2/n_1} + \frac{\tau_0^4}{\tau_0^2 + \sigma_0^2/n_0}} \quad (36.16)$$

15 In the fully symmetric case, Equation (36.15) simplifies to

16

$$\mathbb{E}[R_{\text{roll}}] = \underbrace{(N - 2n)m}_{R_a} + \underbrace{(N - 2n) \frac{\sqrt{2}\tau^2}{\sqrt{\pi} \sqrt{2\tau^2 + \frac{2}{n}\sigma^2}}}_{R_b} \quad (36.17)$$

17 This has an intuitive interpretation. The first term, R_a , is the prior reward we expect to get before
18 we learn anything about the arms. The second term, R_b , is the reward we expect to see by virtue of
19 picking the optimal action to deploy.

20 Let us write $R_b = (N - 2n)R_i$, where R_i is the incremental gain. We see that the incremental
21 gain increases with n , because we are more likely to pick the correct action with a larger sample
22 size; however, this gain can only be accrued for a smaller number of people, as shown by the $N - 2n$
23 prefactor. (This is a consequence of the explore-exploit tradeoff.)

24 The total expected reward is given by adding Equation (36.12) and Equation (36.17):

25

$$\mathbb{E}[R] = \mathbb{E}[R_{\text{test}}] + \mathbb{E}[R_{\text{roll}}] = Nm + (N - 2n) \left(\frac{\sqrt{2}\tau^2}{\sqrt{\pi} \sqrt{2\tau^2 + \frac{2}{n}\sigma^2}} \right) \quad (36.18)$$

26 (The equation for the non-symmetric case is given in [FB19].)

We can maximize the expected reward in Equation (36.18) to find the optimal sample size for the testing phase, which (from symmetry) satisfies $n_1^* = n_2^* = n^*$, and from $\frac{d}{dn^*} \mathbb{E}[R] = 0$ satisfies

$$n^* = \sqrt{\frac{N}{4}u^2 + \left(\frac{3}{4}u^2\right)^2} - \frac{3}{4}u^2 \leq \sqrt{N}\frac{\sigma}{2\tau} \quad (36.19)$$

where $u^2 = \frac{\sigma^2}{\tau^2}$. Thus we see that the optimal sample size n^* increases as the observation noise σ increases, since we need to collect more data to be confident of the right decision. However, the optimal sample size decreases with τ , since a prior belief that the effect size $\delta = \mu_1 - \mu_0$ will be large implies we expect to need less data to reach a confident conclusion.

36.3.1.3 Regret

Given a policy, it is natural to wonder how good it is. We define the **regret** of a policy to be the difference between the expected reward given **perfect information** (PI) about the true best action and the expected reward due to our policy. Minimizing regret is equivalent to making the expected reward of our policy equal to the best possible reward (which may be high or low, depending on the problem).

An oracle with perfect information about which μ_j is bigger would pick the highest scoring action, and hence get an expected reward of $N\mathbb{E}[\max(\mu_1, \mu_2)]$. Since we assume $\mu_j \sim \mathcal{N}(m, \tau^2)$, we have

$$\mathbb{E}[R|PI] = N \left(m + \frac{\tau}{\sqrt{\pi}} \right) \quad (36.20)$$

Therefore the regret from the optimal policy is given by

$$\mathbb{E}[R|PI] - (\mathbb{E}[R_{\text{test}}|\pi^*] + \mathbb{E}[R_{\text{roll}}|\pi^*]) = N \frac{\tau}{\sqrt{\pi}} \left(1 - \frac{\tau}{\sqrt{\tau^2 + \frac{\sigma^2}{n^*}}} \right) + \frac{2n^*\tau^2}{\sqrt{\pi}\sqrt{\tau^2 + \frac{\sigma^2}{n^*}}} \quad (36.21)$$

One can show that the regret is $O(\sqrt{N})$, which is optimal for this problem when using a time horizon (population size) of N [AG13].

36.3.1.4 Expected error rate

Sometimes the goal is posed as **best arm identification**, which means identifying whether $\mu_1 > \mu_0$ or not. That is, if we define $\delta = \mu_1 - \mu_0$, we want to know if $\delta > 0$ or $\delta < 0$. This is naturally phrased as a **hypothesis test**. However, this is arguably the wrong objective, since it is usually not worth spending money on collecting a large sample size to be confident that $\delta > 0$ (say) if the magnitude of δ is small. Instead, it makes more sense to optimize total expected reward, using the method in Section 36.3.1.1.

Nevertheless, we may want to know the probability that we have picked the wrong arm if we use the policy from Section 36.3.1.1. In the symmetric case, this is given by the following:

$$\Pr(\pi(\mathbf{y}_1, \mathbf{y}_0) = 1 | \mu_1 < \mu_0) = \Pr(Y_1 - Y_0 > 0 | \mu_1 < \mu_0) = 1 - \Phi \left(\frac{\mu_1 - \mu_0}{\sigma \sqrt{\frac{1}{n_1} + \frac{1}{n_0}}} \right) \quad (36.22)$$

¹ The above expression assumed that μ_j are known. Since they are not known, we can compute the expected error rate using $\mathbb{E}[\Pr(\pi(\mathbf{y}_1, \mathbf{y}_0) = 1 | \mu_1 < \mu_0)]$. By symmetry, the quantity $\mathbb{E}[\Pr(\pi(\mathbf{y}_1, \mathbf{y}_0) = 0 | \mu_1 > \mu_0)]$ is the same. One can show that both quantities are given by

$$\text{Prob. error} = \frac{1}{4} - \frac{1}{2\pi} \arctan \left(\frac{\sqrt{2}\tau}{\sigma} \sqrt{\frac{n_1 n_0}{n_1 + n_0}} \right) \quad (36.23)$$

⁹ As expected, the error rate decreases with the sample size n_1 and n_0 , increases with observation noise ¹⁰ σ , and decreases with variance of the effect size τ . Thus a policy that minimizes the classification ¹¹ error will also maximize expected reward, but it may pick an overly large sample size, since it does ¹² not take into account the magnitude of δ .

¹³¹⁴

¹⁵ 36.3.2 Example

¹⁶ In this section, we give a simple example of the above framework. Suppose our goal is to do **website testing**, where have two different versions of a webpage that we want to compare in terms of their ¹⁹ **click through rate**. The observed data is now binary, $y_{ij} \sim \text{Ber}(\mu_j)$, so it is natural to use a Beta prior, $\mu_j \sim \text{Beta}(\alpha, \beta)$ (see Section 3.2.1). However, in this case the optimal sample size and decision rule is harder to compute (see [FB19; Sta+17] for details). As a simple approximation, we can assume ²² $\bar{y}_{ij} \sim \mathcal{N}(\mu_j, \sigma^2)$, where $\mu_j \sim \mathcal{N}(m, \tau^2)$, $m = \frac{\alpha}{\alpha+\beta}$, $\tau^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$, and $\sigma^2 = m(1-m)$.

²³ To set the Gaussian prior, [FB19] used empirical data from about 2000 prior A/B tests. For each ²⁴ test, they observed the number of times the page was served with each of the two variations, as well ²⁵ as the total number of times a user clicked on each version. Given this data, they used a hierarchical ²⁶ Bayesian model to infer $\mu_j \sim \mathcal{N}(m = 0.68, \tau = 0.03)$. This prior implies that the expected effect size ²⁷ is quite small, $\mathbb{E}[|\mu_1 - \mu_0|] = 0.023$. (This is consistent with the results in [Aze+20], who found that ²⁸ most changes made to the Microsoft Bing EXP platform had negligible effect, although there were ²⁹ occasionally some “big hits”.)

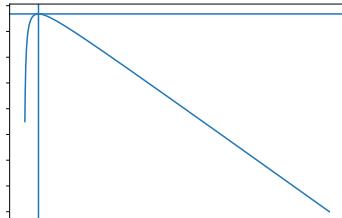
³⁰ With this prior, and assuming a population of $N = 100,000$, Equation (36.19) says that the optimal ³¹ number of trials to run is $n_1^* = n_0^* = 2284$. The expected reward (number of clicks or **conversions**) ³² in the testing phase is $\mathbb{E}[R_{\text{test}}] = 3106$, and in the deployment phase $\mathbb{E}[R_{\text{roll}}] = 66430$, for a total ³³ reward of 69536. The expected error rate is 10%.

³⁴ In Figure 36.3a, we plot the expected reward vs the size of the test phase n . We see that the ³⁵ reward increases sharply with n to the global maximum at $n^* = 2284$, and then drops off more slowly. ³⁶ This indicates that it is better to have a slightly larger test than one that is too small by the same ³⁷ amount. (However, when using a heavy tailed model, [Aze+20] finds that it is better to do lots of ³⁸ smaller tests.)

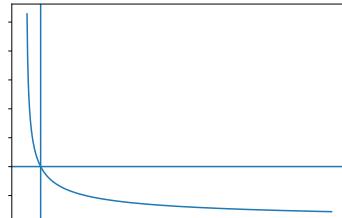
³⁹ In Figure 36.3b, we plot the probability of picking the wrong action vs n . We see that tests that ⁴⁰ are larger than optimal only reduce this error rate marginally. Consequently, if you want to make ⁴¹ the misclassification rate low, you may need a large sample size, particularly if $\mu_1 - \mu_0$ is small, since ⁴² then it will be hard to detect the true best action. However, it is also less important to identify ⁴³ the best action in this case, since both actions have very similar expected reward. This explains ⁴⁴ why classical methods for A/B testing based on frequentist statistics, which use hypothesis testing ⁴⁵ methods to determine if A is better than B, may often recommend sample sizes that are much larger ⁴⁶ than necessary. (See [FB19] and references therein for further discussion.)

⁴⁷

1
2
3
4
5
6
7
8
9
10
11
12



(a)



(b)

13 Figure 36.3: Total expected profit (a) and error rate (b) as a function of the sample size used for website
14 testing. Generated by `ab_test_demo.py`.

15
16

36.4 Contextual bandits

19 This section was co-authored with Lihong Li.

20 In Section 36.3, we discussed A/B testing, in which the decision maker tries two different actions,
21 a_0 and a_1 , a fixed number of times, n_1 and n_0 , measures the resulting sequence of rewards, y_1 and
22 y_0 , and then picks the best action to use for the rest of time (or the rest of the population) so as to
23 maximize expected reward.

24 We can obviously generalize this beyond two actions. More importantly, we can generalize this
25 beyond a one-stage decision problem. In particular, suppose we allow the decision maker to try an
26 action a_t , observe the reward r_t , and then decide what to do at time step $t + 1$, rather than waiting
27 until $n_1 + n_0$ experiments are finished. This immediate feedback allows for **adaptive policies**
28 that can result in much higher expected reward (lower regret). We have converted a one-stage
29 decision problem into a **sequential decision problem**. There are many kinds of sequential decision
30 problems, but in this section, we consider the simplest kind, known as a **bandit problem** (see e.g.,
31 [LS19; Sli19]).

32
33

36.4.1 Types of bandit

35 In a **multi-armed bandit** problem (MAB) there is an agent (decision maker) that can choose an
36 **action** from some **policy** $a_t \sim \pi_t$ at each step, after which it receives a **reward** sampled from the
37 **environment**, $r_t \sim p_R(a_t)$, with expected value $R(s, a) = \mathbb{E}[R|a]$.²

38 We can think of this in terms of an agent at a casino who is faced with multiple slot machines,
39 each of which pays out rewards at a different rate. A slot machine is sometimes called a **one-**
40 **armed bandit**, so a set of K such machines is called a **multi-armed bandit**; each different action
41 corresponds to pulling the arm of a different slot machine, $a_t \in \{1, \dots, K\}$. The goal is to quickly
42 figure out which machine pays out the most money, and then to keep playing that one until you

43
44 2. This is known as a **stochastic bandit**. It is also possible to allow the reward, and possibly the state, to be chosen
45 in an adversarial manner, where nature tries to minimize the reward of the agent. This is known as an **adversarial**
46 **bandit**.

47

¹
2 become as rich as possible.

³ We can extend this model by defining a **contextual bandit**, in which the input to the policy
⁴ at each step is a randomly chosen state or context $s_t \in \mathcal{S}$. The states evolve over time according
⁵ to some arbitrary process, $s_t \sim p(s_t|s_{1:t-1})$, independent of the actions of the agent. The policy
⁶ now has the form $a_t \sim \pi_t(a_t|s_t)$, and the reward function now has the form $r_t \sim p_R(r_t|s_t, a_t)$, with
⁷ expected value $R(s, a) = \mathbb{E}[R|s, a]$. At each step, the agent can use the observed data, $\mathcal{D}_{1:t}$ where
⁸ $\mathcal{D}_t = (s_t, a_t, r_t)$, to update its policy, to maximize expected reward.

⁹ In the **finite horizon** formulation of (contextual) bandits, the goal is to maximize the expected
¹⁰ cumulative reward:

$$\begin{aligned} \underline{11} \quad J &\triangleq \sum_{t=1}^T \mathbb{E}_{p_R(r_t|s_t, a_t)\pi_t(a_t|s_t)p(s_t|s_{1:t-1})} [r_t] = \sum_{t=1}^T \mathbb{E}[r_t] \end{aligned} \quad (36.24)$$

¹⁴(Note that the reward is accrued at each step, even while the agent updates its policy; this is
¹⁵sometimes called “**earning while learning**”.) In the **infinite horizon** formulation, where $T = \infty$,
¹⁶ the cumulative reward may be infinite. To prevent J from being unbounded, we introduce a **discount**
¹⁷ **factor** $0 < \gamma < 1$, so that
¹⁸

$$\begin{aligned} \underline{19} \quad J &\triangleq \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}[r_t] \end{aligned} \quad (36.25)$$

²²The quantity γ can be interpreted as the probability that the agent is terminated at any moment in
²³time (in which case it will cease to accumulate reward).

²⁴ Another way to write this is as follows:

$$\begin{aligned} \underline{26} \quad J &= \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}[r_t] = \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}\left[\sum_{a=1}^K R_a(s_t, a_t)\right] \end{aligned} \quad (36.26)$$

²⁹where we define

$$\begin{aligned} \underline{31} \quad R_a(s_t, a_t) &= \begin{cases} R(s_t, a) & \text{if } a_t = a \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (36.27)$$

³⁴Thus we conceptually evaluate the reward for all arms, but only the one that was actually chosen
³⁵(namely a_t) gives a non-zero value to the agent, namely r_t .

³⁶ There are many extensions of the basic bandit problem. A natural one is to allow the agent to
³⁷ perform **multiple plays**, choosing $M \leq K$ distinct arms at once. Let \mathbf{a}_t be the corresponding action
³⁸ vector which specifies the identity of the chosen arms. Then we define the reward to be

$$\begin{aligned} \underline{40} \quad r_t &= \sum_{a=1}^K R_a(s_t, \mathbf{a}_t) \end{aligned} \quad (36.28)$$

⁴²where

$$\begin{aligned} \underline{44} \quad R_a(s_t, \mathbf{a}_t) &= \begin{cases} R(s_t, a) & \text{if } a \in \mathbf{a}_t \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (36.29)$$

1 This is useful for modeling **resource allocation** problems.

2 Another variant is known as a **restless bandit** [Whi88]. This is the same as the multiple play
3 formulation, except we additionally assume that each arm has its own state vector s_t^a associated with
4 it, which evolves according to some stochastic process, regardless of whether arm a was chosen or
5 not. We then define

$$\begin{aligned} \text{10} \\ \text{11} \quad r_t &= \sum_{a=1}^K R_a(s_t^a, \mathbf{a}_t) \end{aligned} \tag{36.30}$$

12 where $s_t^a \sim p(s_t^a | s_{1:t-1}^a)$ is some arbitrary distribution, often assumed to be Markovian. (The fact
13 that the states associated with each arm evolve even if the arm is not picked is what gives rise to the
14 term “restless”.) This can be used to model serial dependence between the rewards given by each
15 arm.

16 36.4.2 Applications

17 Contextual bandits have many applications. For example, consider an **online advertising system**.
18 In this case, the state s_t represents features of the web page that the user is currently looking at, and
19 the action a_t represents the identity of the ad which the system chooses to show. Since the relevance
20 of the ad depends on the page, the reward function has the form $R(s_t, a_t)$, and hence the problem
21 is contextual. The goal is to maximize the expected reward, which is equivalent to the expected
22 number of times people click on ads; this is known as the **click through rate** or **CTR**. (See e.g.,
23 [Gra+10; Li+10; McM+13; Ago+14; Du+21; YZ22] for more information about this application.)

24 Another application of contextual bandits arises in **clinical trials** [VBW15]. In this case, the
25 state s_t are features of the current patient we are treating, and the action a_t is the treatment the
26 doctor chooses to give them (e.g., a new drug or a **placebo**). Our goal is to maximize expected
27 reward, i.e., the expected number of people who get cured. (An alternative goal is to determine
28 which treatment is best as quickly as possible, rather than maximizing expected reward; this variant
29 is known as **best-arm identification** [ABM10].)

31 36.4.3 Exploration-exploitation tradeoff

32 The fundamental difficulty in solving bandit problems is known as the **exploration-exploitation**
33 **tradeoff**. This refers to the fact that the agent needs to try multiple state/action combinations (this
34 is known as exploration) in order to collect enough data so it can reliably learn the reward function
35 $R(s, a)$; it can then exploit its knowledge by picking the predicted best action for each state. If the
36 agent starts exploiting an incorrect model too early, it will collect suboptimal data, and will get
37 stuck in a negative **feedback loop**, as illustrated in Figure 36.4. This is different from supervised
38 learning, where the data is drawn iid from a fixed distribution.

39 We discuss some solutions to the exploration-exploitation problem below.

42 36.4.4 The optimal solution

43 In this section, we discuss the optimal solution to the exploration-exploitation tradeoff. Let
44 us denote the posterior over the parameters of the reward function by $\mathbf{b}_t = p(\boldsymbol{\theta} | \mathbf{h}_t)$, where
45 $\mathbf{h}_t = \{s_{1:t-1}, a_{1:t-1}, r_{1:t-1}\}$ is the history of observations; this is known as the **belief state** or
46

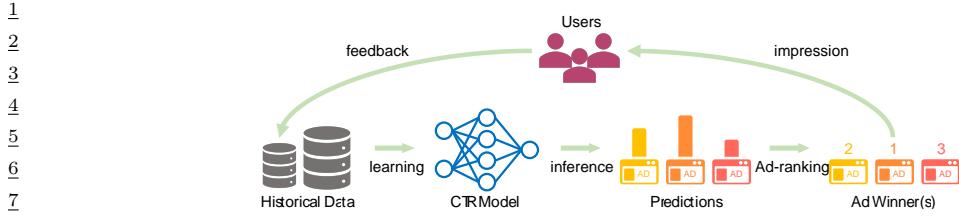


Figure 36.4:) Illustration of the feedback problem in online advertising and recommendation systems. The click through rate (CTR) model is used to decide what ads to show, which affects what data is collected, which affects how the model learns. From Figure 1–2 of [Du+21]. Used with kind permission of Chao Du.

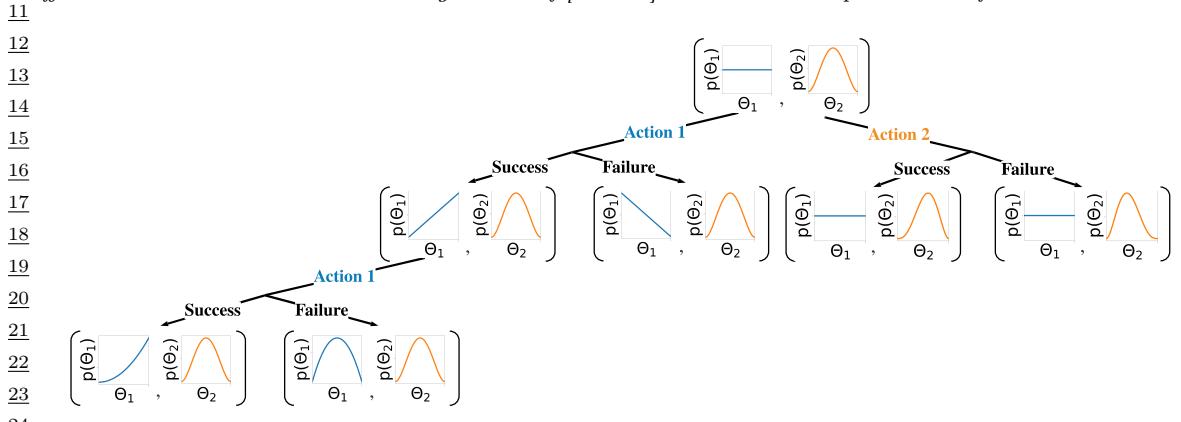


Figure 36.5: Illustration of sequential belief updating for a two-armed beta-Bernoulli bandit. The prior for the reward for action 1 is the (blue) uniform distribution Beta(1,1); the prior for the reward for action 2 is the (orange) unimodal distribution Beta(2,2). We update the parameters of the belief state based on the chosen action, and based on whether the observed reward is success (1) or failure (0). (Compare to Figure 3.5, where we display the predictive distribution for a single armed beta-Bernoulli model.)

information state. It is a finite sufficient statistic for the history \mathbf{h}_t . The belief state can be updated deterministically using Bayes rule:

$$\mathbf{b}_t = \text{BayesRule}(\mathbf{b}_{t-1}, a_t, r_t) \quad (36.31)$$

For example, consider a context-free **Bernoulli bandit**, where $p_R(r|a) = \text{Ber}(r|\mu_a)$, and $\mu_a = p_R(r=1|a) = R(a)$ is the expected reward for taking action a . Suppose we use a factored beta prior

$$p_0(\boldsymbol{\theta}) = \prod_a \text{Beta}(\mu_a | \alpha_0^a, \beta_0^a) \quad (36.32)$$

where $\boldsymbol{\theta} = (\mu_1, \dots, \mu_K)$. We can compute the posterior in closed form, as we discuss in Section 3.2.1. In particular, we find

$$p(\boldsymbol{\theta}|\mathcal{D}_t) = \prod_a \text{Beta}(\mu_a | \underbrace{\alpha_t^a + N_t^0(a)}_{\alpha_t^a}, \underbrace{\beta_t^a + N_t^1(a)}_{\beta_t^a}) \quad (36.33)$$

1 where
 2

$$N_t^r(a) = \sum_{s=1}^{t-1} \mathbb{I}(a_s = a, r_s = r) \quad (36.34)$$

3 This is illustrated in Figure 36.5 for a two-armed Bernoulli bandit.
 4

5 We can use a similar method for a **Gaussian bandit**, where $p_R(r|a) = \mathcal{N}(r|\mu_a, \sigma_a^2)$, using results
 6 from Section 3.2.3. In the case of contextual bandits, the problem becomes more complicated. If we
 7 assume a **linear regression bandit**, $p_R(r|s, a; \theta) = \mathcal{N}(r|\phi(s, a)^\top \theta, \sigma^2)$, we can use Bayesian
 8 linear regression to compute $p(\theta|\mathcal{D}_t)$ in closed form, as we discuss in Section 15.2. If we assume
 9 a **logistic regression bandit**, $p_R(r|s, a; \theta) = \text{Ber}(r|\sigma(\phi(s, a)^\top \theta))$, we can use Bayesian logistic
 10 regression to compute $p(\theta|\mathcal{D}_t)$, as we discuss in Section 15.3.4. If we have a **neural bandit** of
 11 the form $p_R(r|s, a; \theta) = \text{GLM}(r|f(s, a; \theta))$ for some nonlinear function f , then posterior inference
 12 becomes more challenging, as we discuss in Chapter 17. However, standard techniques, such as the
 13 extended Kalman filter (Section 17.6.1) can be applied. (For a way to scale this approach to large
 14 DNNs, see the “**subspace neural bandit**” approach of [DMKM22].)
 15

16 Regardless of the algorithmic details, we can represent the belief state update as follows:
 17

$$p(\mathbf{b}_t|\mathbf{b}_{t-1}, a_t, r_t) = \mathbb{I}(\mathbf{b}_t = \text{BayesRule}(\mathbf{b}_{t-1}, a_t, r_t)) \quad (36.35)$$

18 The observed reward at each step is then predicted to be
 19

$$p(r_t|\mathbf{b}_t) = \int p_R(r_t|s_t, a_t; \theta)p(\theta|\mathbf{b}_t)d\theta \quad (36.36)$$

20 We see that this is a special form of a (controlled) Markov decision process (Section 36.5) known as a
 21 **belief-state MDP**.
 22

23 In the special case of context-free bandits with a finite number of arms, the optimal policy of this
 24 belief state MDP can be computed using dynamic programming (c.f., Section 36.6); the result can
 25 be represented as a table of action probabilities, $\pi_t(a_1, \dots, a_K)$, for each step; this is known as the
 26 **Gittins index** [Git89]. However, computing the optimal policy for general contextual bandits is
 27 intractable [PT87], so we have to resort to approximations, as we discuss below.
 28

36.4.5 Upper confidence bounds (UCB)

29 The optimal solution to explore-exploit is intractable. However, an intuitively sensible approach is
 30 based on the principle known as “**optimism in the face of uncertainty**”. The principle selects
 31 actions greedily, but based on optimistic estimates of their rewards. The most important class of
 32 strategies with this principle are collectively called **upper confidence bound** or **UCB** methods.
 33

34 To use a UCB strategy, the agent maintains an optimistic reward function estimate \tilde{R}_t , so that
 35 $\tilde{R}_t(s_t, a) \geq R(s_t, a)$ for all a with high probability, and then chooses the greedy action accordingly:
 36

$$a_t = \underset{a}{\operatorname{argmax}} \tilde{R}_t(s_t, a) \quad (36.37)$$

37 UCB can be viewed a form of **exploration bonus**, where the optimistic estimate encourages
 38 exploration. Typically, the amount of optimism, $\tilde{R}_t - R$, decreases over time so that the agent
 39

¹ gradually reduces exploration. With properly constructed optimistic reward estimates, the UCB
² strategy has been shown to achieve near-optimal regret in many variants of bandits [LS19]. (We
³ discuss regret in Section 36.4.7.)

⁴ The optimistic function \tilde{R} can be obtained in different ways, sometimes in closed forms, as we
⁵ discuss below.

⁶

⁷ 36.4.5.1 Frequentist approach

⁸ One approach is to use a **concentration inequality** [BLM16] to derive a high-probability upper
⁹ bound of the estimation error: $|\hat{R}_t(s, a) - R_t(s, a)| \leq \delta_t(s, a)$, where \hat{R}_t is a usual estimate of R
¹⁰ (often the MLE), and δ_t is a properly selected function. An optimistic reward is then obtained by
¹¹ setting $\tilde{R}_t(s, a) = \hat{R}_t(s, a) + \delta_t(s, a)$.

¹² As an example, consider again the context-free Bernoulli bandit, $R(a) \sim \text{Ber}(\mu(a))$. The MLE
¹³ $\hat{R}_t(a) = \hat{\mu}_t(a)$ is given by the empirical average of observed rewards whenever action a was taken:

$$\hat{\mu}_t(a) = \frac{N_t^1(a)}{N_t(a)} = \frac{N_t^1(a)}{N_t^0(a) + N_t^1(a)} \quad (36.38)$$

¹⁴ where $N_t^r(a)$ is the number of times (up to step $t - 1$) that action a has been tried and the observed
¹⁵ reward was r , and $N_t(a)$ is the total number of times action a has been tried:

$$N_t(a) = \sum_{s=1}^{t-1} \mathbb{I}(a_s = a) \quad (36.39)$$

¹⁶ Then the **Chernoff-Hoeffding inequality** [BLM16] leads to $\delta_t(a) = c/\sqrt{N_t(a)}$ for some proper
¹⁷ constant c , so

$$\tilde{R}_t(a) = \hat{\mu}_t(a) + \frac{c}{\sqrt{N_t(a)}} \quad (36.40)$$

³¹ 36.4.5.2 Bayesian approach

³² We may also derive \tilde{R} from Bayesian inference. If we use a beta prior, we can compute the posterior
³³ in closed form, as shown in Equation (36.33). The posterior mean is $\hat{\mu}_t(a) = \mathbb{E}[\mu(a)|\mathbf{h}_t] = \frac{\alpha_t^a}{\alpha_t^a + \beta_t^a}$.
³⁴ From Equation (3.28), the posterior standard deviation is approximately

$$\hat{\sigma}_t(a) = \sqrt{\mathbb{V}[\mu(a)|\mathbf{h}_t]} \approx \sqrt{\frac{\hat{\mu}_t(a)(1 - \hat{\mu}_t(a))}{N_t(a)}} \quad (36.41)$$

³⁵ We can use similar techniques for a Gaussian bandit, where $p_R(R|a, \boldsymbol{\theta}) = \mathcal{N}(R|\mu_a, \sigma_a^2)$, μ_a is the
³⁶ expected reward, and σ_a^2 the variance. If we use a conjugate prior, we can compute $p(\mu_a, \sigma_a|\mathcal{D}_t)$
³⁷ in closed form (see Section 3.2.3). Using an uninformative version of the conjugate prior, we find
³⁸ $\mathbb{E}[\mu_a|\mathbf{h}_t] = \hat{\mu}_t(a)$, which is just the empirical mean of rewards for action a . The uncertainty in
³⁹ this estimate is the standard error of the mean, given by Equation (3.103), i.e., $\sqrt{\mathbb{V}[\mu_a|\mathbf{h}_t]} =$
⁴⁰ $\hat{\sigma}_t(a)/\sqrt{N_t(a)}$, where $\hat{\sigma}_t(a)$ is the empirical standard deviation of the rewards for action a .

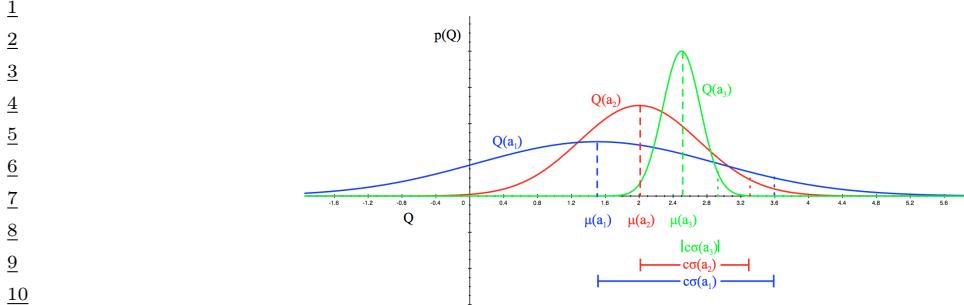


Figure 36.6: Illustration of the reward distribution $Q(a)$ for 3 different actions, and the corresponding lower and upper confidence bounds. From [Sil18]. Used with kind permission of David Silver.

This approach can also be extended to contextual bandits, modulo the difficulty of computing the belief state.

Once we have computed the mean and posterior standard deviation, we define the optimistic reward estimate as

$$\tilde{R}_t(a) = \hat{\mu}_t(a) + c\hat{\sigma}_t(a) \quad (36.42)$$

for some constant c that controls how greedy the policy is. We see that this is similar to the frequentist method based on concentration inequalities, but is more general.

36.4.5.3 Example

Figure 36.6 illustrates the UCB principle for a Gaussian bandit. We assume there are 3 actions, and we represent $p(R(a)|\mathcal{D}_t)$ using a Gaussian. We show the posterior means $Q(a) = \mu(a)$ with a vertical dotted line, and the scaled posterior standard deviations $c\sigma(a)$ as a horizontal solid line.

36.4.6 Thompson sampling

A common alternative to UCB is to use **Thompson sampling** [Tho33], also called **probability matching** [Sco10]. In this approach, we define the policy at step t to be $\pi_t(a|s_t, \mathbf{h}_t) = p_a$, where p_a is the probability that a is the optimal action. This can be computed using

$$p_a = \Pr(a = a_*|s_t, \mathbf{h}_t) = \int \mathbb{I}\left(a = \operatorname{argmax}_{a'} R(s_t, a'; \boldsymbol{\theta})\right) p(\boldsymbol{\theta}|\mathbf{h}_t) d\boldsymbol{\theta} \quad (36.43)$$

If the posterior is uncertain, the agent will sample many different actions, automatically resulting in exploration. As the uncertainty decreases, it will start to exploit its knowledge.

To see how we can implement this method, note that we can compute the expression in Equation (36.43) by using a single Monte Carlo sample $\tilde{\boldsymbol{\theta}}_t \sim p(\boldsymbol{\theta}|\mathbf{h}_t)$. We then plug in this parameter into our reward model, and greedily pick the best action:

$$a_t = \operatorname{argmax}_{a'} R(s_t, a'; \tilde{\boldsymbol{\theta}}_t) \quad (36.44)$$

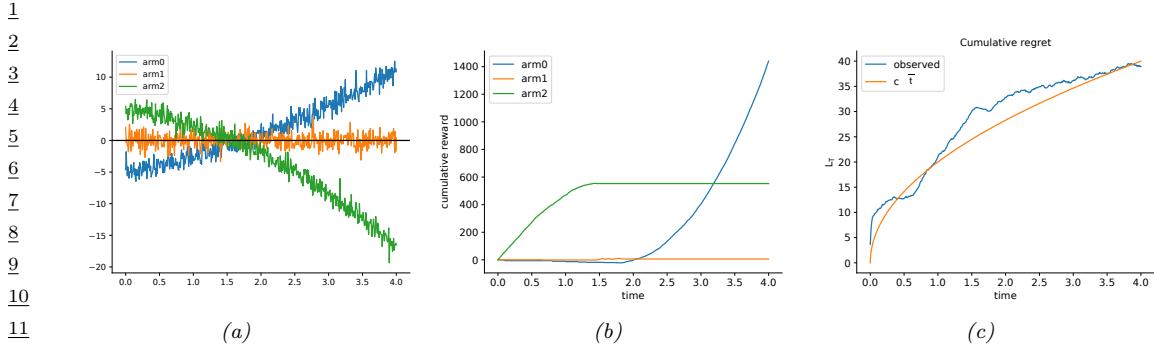


Figure 36.7: Illustration of Thompson sampling applied to a linear-Gaussian contextual bandit. The context has the form $s_t = (1, t, t^2)$. (a) True reward for each arm vs time. (b) Cumulative reward per arm vs time. (c) Cumulative regret vs time. Generated by `thompson_sampling_linear_gaussian.py`.

This sample-then-exploit approach will choose actions with exactly the desired probability, since

$$p_a = \int \mathbb{I} \left(a = \operatorname{argmax}_{a'} R(s_t, a'; \tilde{\theta}_t) \right) p(\tilde{\theta}_t | \mathbf{h}_t) = \Pr_{\tilde{\theta}_t \sim p(\theta | \mathbf{h}_t)} (a = \operatorname{argmax}_{a'} R(s_t, a'; \tilde{\theta}_t)) \quad (36.45)$$

Despite its simplicity, this approach can be shown to achieve optimal (logarithmic) regret (see e.g., [Rus+18] for a survey). In addition, it is very easy to implement, and hence is widely used in practice [Gra+10; Sco10; CL11].

In Figure 36.7, we give a simple example of Thompson sampling applied to a linear regression bandit. The context has the form $s_t = (1, t, t^2)$. The true reward function has the form $R(s_t, a) = \mathbf{w}_a^\top s_t$. The weights per arm are chosen as follows: $\mathbf{w}_0 = (-5, 2, 0.5)$, $\mathbf{w}_1 = (0, 0, 0)$, $\mathbf{w}_2 = (5, -1.5, -1)$. Thus we see that arm 0 is initially worse (large negative bias) but gets better over time (positive slope), arm 1 is useless, and arm 2 is initially better (large positive bias) but gets worse over time. The observation noise is the same for all arms, $\sigma^2 = 1$. See Figure 36.7(a) for a plot of the reward function.

We use a conjugate Gaussian-Gamma prior and perform exact Bayesian updating. Thompson sampling quickly discovers that arm 1 is useless. Initially it pulls arm 2 more, but it adapts to the non-stationary nature of the problem and switches over to arm 0, as shown in Figure 36.7(b).

34

35 36.4.7 Regret

36 We have discussed several methods for solving the exploration-exploitation tradeoff. It is useful 37 to quantify the degree of suboptimality of these methods. A common approach is to compute the 38 **regret**, which is defined as the difference between the expected reward under the agent's policy and 39 the oracle policy π_* , which knows the true reward function. (Note that the oracle policy will in 40 general be better than the Bayes optimal policy, which we discussed in Section 36.4.4.)

41 Specifically, let π_t be the agent's policy at time t . Then the **per-step regret** at t is defined as

$$42 \quad l_t \triangleq \mathbb{E}_{p(s_t)} [R(s_t, \pi_*(s_t))] - \mathbb{E}_{\pi_t(a_t | s_t) p(s_t)} [R(s_t, a_t)] \quad (36.46)$$

43 If we only care about the final performance of the best discovered arm, as in most optimization 44 problems, it is enough to look at the **simple regret** at the last step, namely l_T . Optimizing simple 45

regret results in a problem known as **pure exploration** [BMS11], since there is no need to exploit the information during the learning process. However, it is more common to focus on the the **cumulative regret**, also called the **total regret** or just the **regret**, which is defined as

$$L_T \triangleq \mathbb{E} \left[\sum_{t=1}^T l_t \right] \quad (36.47)$$

Here the expectation is with respect to randomness in determining π_t , which depends on earlier states, actions and rewards, as well as other potential sources of randomness.

Under the typical assumption that rewards are bounded, L_T is at most linear in T . If the agent's policy converges to the optimal policy as T increases, then the regret is sublinear: $L_T = o(T)$. In general, the slower L_T grows, the more efficient the agent is in trading off exploration and exploitation.

To understand its growth rate, it is helpful to consider again a simple context-free bandit, where $R_* = \operatorname{argmax}_a R(a)$ is the optimal reward. The total regret in the first T steps can be written as

$$L_T = \mathbb{E} \left[\sum_{t=1}^T R_* - R(a_t) \right] = \sum_{a \in \mathcal{A}} \mathbb{E}[N_{T+1}(a)] (R_* - R(a)) = \sum_{a \in \mathcal{A}} \mathbb{E}[N_{T+1}(a)] \Delta_a \quad (36.48)$$

where $N_{T+1}(a)$ is the total number of times the agent picks action a up to step T , and $\Delta_a = R_* - R(a)$ is the reward **gap**. If the agent under-explores and converges to choosing a suboptimal action (say, \hat{a}), then a linear regret is suffered with a per-step regret of $\Delta_{\hat{a}}$. On the other hand, if the agent over-explores, then $N_t(a)$ will be too large for suboptimal actions, and the agent also suffers a linear regret.

Fortunately, it is possible to achieve sublinear regrets, using some of the methods discussed above, such as UCB and Thompson sampling. For example, one can show that Thompson sampling has $O(\sqrt{KT \log T})$ regret [RR14]. This is shown empirically in Figure 36.7(c).

In fact, both UCB and Thompson sampling are optimal, in the sense that their regrets are essentially not improvable; that is, they match regret lower bounds. To establish such a lower bound, note that the agent needs to collect enough data to distinguish different reward distributions, before identifying the optimal action. Typically, the deviation of the reward estimate from the true reward decays at the rate of $1/\sqrt{N}$, where N is the sample size (see e.g., Equation (3.103)). Therefore, if two reward distributions are similar, distinguishing them becomes harder and requires more samples. (For example, consider the case of a bandit with Gaussian rewards with slightly different means and large variance, as shown in Figure 36.6.)

The following fundamental result is proved by [LR85] for the asymptotic regret (under certain mild assumptions not given here):

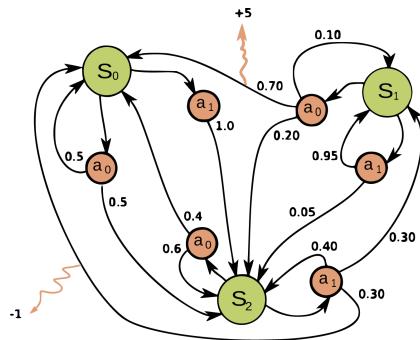
$$\liminf_{T \rightarrow \infty} L_T \geq \log T \sum_{a: \Delta_a > 0} \frac{\Delta_a}{D_{\text{KL}}(p_R(a) \| p_R(a_*))} \quad (36.49)$$

Thus, we see that the best we can achieve is logarithmic growth in the total regret. Similar lower bounds have also been obtained for various bandits variants.

36.5 Markov decision problems

In this section, we generalize the discussion of contextual bandits by allowing the state of nature to change depending on the actions chosen by the agent. The resulting model is called a **Markov**

1
2
3
4
5
6
7
8
9
10
11
12



¹³Figure 36.8: Illustration of an MDP as a finite state machine (FSM). The MDP has three discrete states ¹⁴(green circles), two discrete actions (orange circles), and two non-zero rewards (orange arrows). The numbers ¹⁵on the black edges represent state transition probabilities, e.g., $p(s' = s_0 | a = a_0, s = s_0) = 0.7$; most ¹⁶state transitions are impossible (probability 0), so the graph is sparse. The numbers on the yellow wiggly ¹⁷edges represent expected rewards, e.g., $R(s = s_1, a = a_0, s' = s_0) = +5$; state transitions with zero reward ¹⁸are not annotated. From https://en.wikipedia.org/wiki/Markov_decision_process. Used with kind ¹⁹permission of Wikipedia author waldoalvarez.

20
21

²²decision process or MDP, as we explain in detail below. This model forms the foundation of ²³reinforcement learning, which we discuss in Chapter 37.

24

²⁵36.5.1 Basics

26

²⁷A **Markov decision process** [Put94] can be used to model the interaction of an **agent** and an ²⁸**environment**. It is often described by a tuple $\langle \mathcal{S}, \mathcal{A}, p_T, p_R, p_0 \rangle$, where \mathcal{S} is a set of environment ²⁹states, \mathcal{A} a set of actions the agent can take, p_T a **transition model**, p_R a **reward model**, and p_0 ³⁰the initial state distribution. The interaction starts at time $t = 0$, where the initial state $s_0 \sim p_0$. ³¹Then, at time $t \geq 0$, the agent observes the environment state $s_t \in \mathcal{S}$, and follows a **policy** π to ³²take an action $a_t \in \mathcal{A}$. In response, the environment emits a real-valued reward signal $r_t \in \mathcal{R}$ and ³³enters a new state $s_{t+1} \in \mathcal{S}$. The policy is in general stochastic, with $\pi(a|s)$ being the probability of ³⁴choosing action a in state s . We use $\pi(s)$ to denote the conditional probability over \mathcal{A} if the policy ³⁵is stochastic, or the action it chooses if it is deterministic. The process at every step is called a ³⁶**transition**; at time t , it consists of the tuple (s_t, a_t, r_t, s_{t+1}) , where $a_t \sim \pi(s_t)$, $s_{t+1} \sim p_T(s_t, a_t)$, ³⁷and $r_t \sim p_R(s_t, a_t, s_{t+1})$. Hence, under policy π , the probability of generating a trajectory τ of ³⁸length T can be written explicitly as

$$\begin{aligned} & p(\tau) = p_0(s_0) \prod_{t=0}^{T-1} \pi(a_t | s_t) p_T(s_{t+1} | s_t, a_t) p_R(r_t | s_t, a_t, s_{t+1}) \end{aligned} \quad (36.50)$$

42

⁴³It is useful to define the **reward function** from the reward model p_R , as the average immediate ⁴⁴reward of taking action a in state s , with the next state marginalized:

$$R(s, a) \triangleq \mathbb{E}_{p_T(s'|s, a)} [\mathbb{E}_{p(r|s, a, s')} [r]] \quad (36.51)$$

47

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

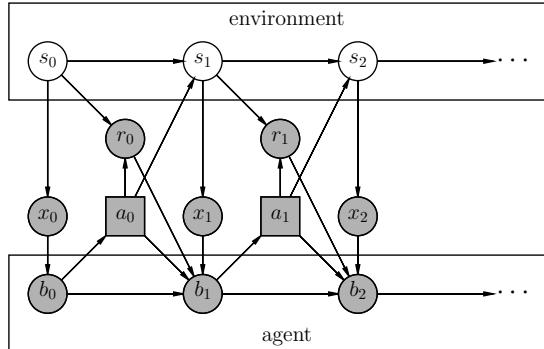


Figure 36.9: Illustration of a partially observable Markov decision process (POMDP) with hidden environment state s_t which generates the observation x_t , controlled by an agent with internal belief state b_t which generates the action a_t . The reward r_t depends on s_t and a_t . Nodes in this graph represent random variables (circles) and decision variables (squares).

Eliminating the dependence on next states does not lead to loss of generality in the following discussions, as our subject of interest is the total (additive) expected reward along the trajectory. For this reason, we often use the tuple $\langle \mathcal{S}, \mathcal{A}, p_T, R, p_0 \rangle$ to describe an MDP.

In general, the state and action sets of an MDP can be discrete or continuous. When both sets are finite, we can represent these functions as lookup tables; this is known as a **tabular representation**. In this case, we can represent the MDP as a **finite state machine**, which is a graph where nodes correspond to states, and edges correspond to actions and the resulting rewards and next states. Figure 36.8 gives a simple example of an MDP with 3 states and 2 actions.

The field of **control theory**, which is very closely related to RL, uses slightly different terminology. In particular, the environment is called the **plant**, and the agent is called the **controller**. States are denoted by $\mathbf{x}_t \in \mathcal{X} \subseteq \mathbb{R}^D$, actions are denoted by $\mathbf{u}_t \in \mathcal{U} \subseteq \mathbb{R}^K$, and rewards are denoted by costs $c_t \in \mathbb{R}$. Apart from this notational difference, the fields of RL and control theory are very similar (see e.g., [Son98; Rec19]), although control theory tends to focus on provably optimal methods (by making strong modeling assumptions), whereas RL tends to tackle harder problems with heuristic methods, for which optimality guarantees are often hard to obtain.

36.5.2 Partially observed MDPs

An important generalization of the MDP framework relaxes the assumption that the agent sees the hidden world state s_t directly; instead we assume it only sees a potentially noisy observation generated from the hidden state, $x_t \sim p(\cdot|s_t, a_t)$. The resulting model is called a **partially observable Markov decision process** or **POMDP** (pronounced “pom-dee-pee”). Now the agent’s policy is a mapping from all the available data to actions, $a_t \sim \pi(\mathcal{D}_{1:t-1}, x_t)$, $\mathcal{D}_t = (x_t, a_t, r_t)$. See Figure 36.9 for an illustration. MDPs are a special case where $x_t = s_t$.

In general, POMDPs are much harder to solve than MDPs. A common approximation is to use the last several observed inputs, say $\mathbf{x}_{t-h:t}$ for history of size h , as a proxy for the hidden state, and

¹
² then to treat this as a fully observed MDP.

³

⁴ 36.5.3 Episodes and returns

⁵ The Markov decision process describes how a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ is stochastically
⁶ generated. If the agent can potentially interact with the environment forever, we call it a **continuing**
⁷ **task**. Alternatively, the agent is in an **episodic task**, if its interaction terminates once the system
⁸ enters a **terminal state** or **absorbing state**; s is absorbing if the next state from s is always s
⁹ with 0 reward. After entering a terminal state, we may start a new **episode** from a new initial state
¹⁰ $s_0 \sim p_0$. The episode length is in general random. For example, the amount of time a robot takes to
¹¹ reach its goal may be quite variable, depending on the decisions it makes, and the randomness in the
¹² environment. Note that we can convert an episodic MDP to a continuing MDP by redefining the
¹³ transition model in absorbing states to be the initial-state distribution p_0 . Finally, if the trajectory
¹⁴ length T in an episodic task is fixed and known, it is called a **finite horizon problem**.

¹⁵ Let τ be a trajectory of length T , where T may be ∞ if the task is continuing. We define the
¹⁶ **return** for the state at time t to be the sum of expected rewards obtained going forward, where each
¹⁷ reward is multiplied by a **discount factor** $\gamma \in [0, 1]$:

$$\begin{aligned} G_t &\triangleq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1} \end{aligned} \quad (36.52)$$

$$\begin{aligned} &= \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} = \sum_{j=t}^{T-1} \gamma^{j-t} r_j \end{aligned} \quad (36.53)$$

²⁴ G_t is sometimes called the **reward-to-go**. For episodic tasks that terminate at time T , we define
²⁵ $G_t = 0$ for $t \geq T$. Clearly, the return satisfies the following recursive relationship:

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1} \quad (36.54)$$

²⁶

²⁷ The discount factor γ plays two roles. First, it ensures the return is finite even if $T = \infty$ (i.e.,
²⁸ infinite horizon), provided we use $\gamma < 1$ and the rewards r_t are bounded. Second, it puts more weight
²⁹ on short-term rewards, which generally has the effect of encouraging the agent to achieve its goals
³⁰ more quickly (see Section 36.5.5.1 for an example). However, if γ is too small, the agent will become
³¹ too greedy. In the extreme case where $\gamma = 0$, the agent is completely **myopic**, and only tries to
³² maximize its immediate reward. In general, the discount factor reflects the assumption that there
³³ is a probability of $1 - \gamma$ that the interaction will end at the next step. For finite horizon problems,
³⁴ where T is known, we can set $\gamma = 1$, since we know the life time of the agent a priori.³

³⁵

³⁶ 36.5.4 Value functions

³⁷ Let π be a given policy. We define the **state-value function**, or **value function** for short, as
³⁸ follows (with $\mathbb{E}_\pi[\cdot]$ indicating that actions are selected by π):

$$V_\pi(s) \triangleq \mathbb{E}_\pi[G_0 | s_0 = s] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right] \quad (36.55)$$

⁴⁶ 3. We may also use $\gamma = 1$ for continuing tasks, targeting the (undiscounted) average reward criterion [Put94].

⁴⁷

This is the expected return obtained if we start in state s and follow π to choose actions in a continuing task (i.e., $T = \infty$).

Similarly, we define the **action-value function**, also known as the **Q -function**, as follows:

$$Q_\pi(s, a) \triangleq \mathbb{E}_\pi [G_0 | s_0 = s, a_0 = a] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a \right] \quad (36.56)$$

This quantity represents the expected return obtained if we start by taking action a in state s , and then follow π to choose actions thereafter.

Finally, we define the **advantage function** as follows:

$$A_\pi(s, a) \triangleq Q_\pi(s, a) - V_\pi(s) \quad (36.57)$$

This tells us the benefit of picking action a in state s then switching to policy π , relative to the baseline return of always following π . Note that $A_\pi(s, a)$ can be both positive and negative, and $\mathbb{E}_{\pi(a|s)} [A_\pi(s, a)] = 0$ due to a useful equality: $V_\pi(s) = \mathbb{E}_{\pi(a|s)} [Q_\pi(s, a)]$.

36.5.5 Optimal value functions and policies

Suppose π_* is a policy such that $V_{\pi_*} \geq V_\pi$ for all $s \in \mathcal{S}$ and all policy π , then it is an **optimal policy**. There can be multiple optimal policies for the same MDP, but by definition their value functions must be the same, and are denoted by V_* and Q_* , respectively. We call V_* the **optimal state-value function**, and Q_* the **optimal action-value function**. Furthermore, any finite MDP must have at least one deterministic optimal policy [Put94].

A fundamental result about the optimal value function is **Bellman's optimality equations**:

$$V_*(s) = \max_a R(s, a) + \gamma \mathbb{E}_{p_T(s'|s,a)} [V_*(s')] \quad (36.58)$$

$$Q_*(s, a) = R(s, a) + \gamma \max_{a'} \mathbb{E}_{p_T(s'|s,a)} [Q_*(s', a')] \quad (36.59)$$

Conversely, the optimal value functions are the only solutions that satisfy the equations. In other words, although the value function is defined as the expectation of a sum of infinitely many rewards, it can be characterized by a recursive equation that involves only one-step transition and reward models of the MDP. Such a recursion play a central role in many RL algorithms we will see later in this chapter. Given a value function (V or Q), the discrepancy between the right- and left-hand sides of Equations (36.58) and (36.59) are called **Bellman error** or **Bellman residual**.

Furthermore, given the optimal value function, we can derive an optimal policy using

$$\pi_*(s) = \operatorname{argmax}_a Q_*(s, a) \quad (36.60)$$

$$= \operatorname{argmax}_a [R(s, a) + \gamma \mathbb{E}_{p_T(s'|s,a)} [V_*(s')]] \quad (36.61)$$

Following such an optimal policy ensures the agent achieves maximum expected return starting from any state. The problem of solving for V_* , Q_* or π_* is called **policy optimization**. In contrast, solving for V_π or Q_π for a given policy π is called **policy evaluation**, which constitutes an important subclass of RL problems as will be discussed in later sections. For policy evaluation, we have similar Bellman equations, which simply replace $\max_a \{\cdot\}$ in Equations (36.58) and (36.59) with $\mathbb{E}_{\pi(a|s)} [\cdot]$.

¹ In Equations (36.60) and (36.61), as in the Bellman optimality equations, we must take a maximum
² over all actions in \mathcal{A} , and the maximizing action is called the **greedy action** with respect to the
³ value functions, Q_* or V_* . Finding greedy actions is computationally easy if \mathcal{A} is a small finite set.
⁴ For high dimensional continuous spaces, we can treat a as a sequence of actions, and optimize one
⁵ dimension at a time [Met+17], or use gradient-free optimizers such as cross-entropy method (see
⁶ supplementary), as used in the **QT-Opt** method [Kal+18a]. Recently, **CAQL** (continuous action
⁷ Q -learning, [Ryu+20]) proposed to use mixed integer programming to solve the argmax problem,
⁸ leveraging the ReLU structure of the Q -network. We can also amortize the cost of this optimization
⁹ by training a policy $a_* = \pi_*(s)$ after learning the optimal Q -function.

¹⁰

¹¹ 36.5.5.1 Example

¹² In this section, we show a simple example, to make concepts like value functions more concrete.
¹³ Consider the 1d **grid world** shown in Figure 36.10(a). There are 5 possible states, among them S_{T1}
¹⁴ and S_{T2} are absorbing states, since the interaction ends once the agent enters them. There are 2
¹⁵ actions, \uparrow and \downarrow . The reward function is zero everywhere except at the goal state, S_{T2} , which gives a
¹⁶ reward of 1 upon entering. Thus the optimal action in every state is to move down.

¹⁷ Figure 36.10(b) shows the Q_* function for $\gamma = 0$. Note that we only show the function for
¹⁸ non-absorbing states, as the optimal Q -values are 0 in absorbing states by definition. We see that
¹⁹ $Q_*(s_3, \downarrow) = 1.0$, since the agent will get a reward of 1.0 on the next step if it moves down from s_3 ;
²⁰ however, $Q_*(s, a) = 0$ for all other state-action pairs, since they do not provide nonzero immediate
²¹ reward. This optimal Q -function reflects the fact that using $\gamma = 0$ is completely myopic, and ignores
²² the future.

²³ Figure 36.10(c) shows Q_* when $\gamma = 1$. In this case, we care about all future rewards equally. Thus
²⁴ $Q_*(s, a) = 1$ for all state-action pairs, since the agent can always reach the goal eventually. This is
²⁵ infinitely far-sighted. However, it does not give the agent any short-term guidance on how to behave.
²⁶ For example, in s_2 , it is not clear if it should go up or down, since both actions will eventually
²⁷ reach the goal with identical Q_* -values.

²⁸ Figure 36.10(d) shows Q_* when $\gamma = 0.9$. This reflects a preference for near-term rewards, while
²⁹ also taking future reward into account. This encourages the agent to seek the shortest path to the
³⁰ goal, which is usually what we desire. A proper choice of γ is up to the agent designer, just like the
³¹ design of the reward function, and has to reflect the desired behavior of the agent.

³²

³³ 36.6 Planning in an MDP

³⁴

³⁵ In this section, we discuss how to compute an optimal policy when the MDP model is known. This
³⁶ problem is called **planning**, in contrast to the learning problem where the models are unknown,
³⁷ which is tackled using reinforcement learning Chapter 37. The planning algorithms we discuss are
³⁸ based on **dynamic programming** (DP) and **linear programming** (LP).

³⁹ For simplicity, in this section, we assume discrete state and action sets with $\gamma < 1$. However, exact
⁴⁰ calculation of optimal policies often depends polynomially on the sizes of \mathcal{S} and \mathcal{A} , and is intractable,
⁴¹ for example, when the state space is a Cartesian product of several finite sets. This challenge is known
⁴² as the **curse of dimensionality**. Therefore, approximations are typically needed, such as using
⁴³ parametric or nonparametric representations of the value function or policy, both for computational
⁴⁴ tractability and for extending the methods to handle MDPs with general state and action sets.

⁴⁵

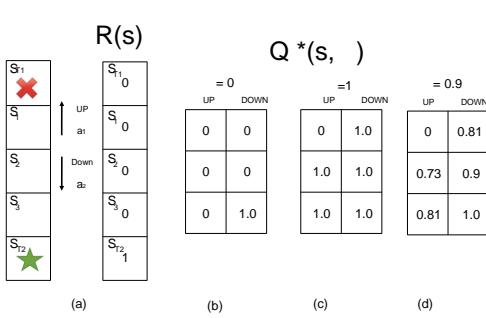


Figure 36.10: Left: illustration of a simple MDP corresponding to a 1d grid world of 3 non-absorbing states and 2 actions. Right: optimal Q-functions for different values of γ . Adapted from Figures 3.1, 3.2, 3.4 of [GK19].

In this case, we have **approximate dynamic programming** (ADP) and **approximate linear programming** (ALP) algorithms (see e.g., [Ber19]).

36.6.1 Value iteration

A popular and effective DP method for solving an MDP is **value iteration** (VI). Starting from an initial value function estimate V_0 , the algorithm iteratively updates the estimate by

$$V_{k+1}(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} p(s'|s, a) V_k(s') \right] \quad (36.62)$$

Note that the update rule, sometimes called a **Bellman backup**, is exactly the right-hand side of the Bellman optimality equation Equation (36.58), with the unknown V_* replaced by the current estimate V_k . A fundamental property of Equation (36.62) is that the update is a **contraction**: it can be verified that

$$\max_s |V_{k+1}(s) - V_*(s)| \leq \gamma \max_s |V_k(s) - V_*(s)| \quad (36.63)$$

In other words, every iteration will reduce the maximum value function error by a constant factor. It follows immediately that V_k will converge to V_* , after which an optimal policy can be extracted using Equation (36.61). In practice, we can often terminate VI when V_k is close enough to V_* , since the resulting greedy policy wrt V_k will be near optimal. Value iteration can be adapted to learn the optimal action-value function Q_* .

In value iteration, we compute $V_*(s)$ and $\pi_*(s)$ for all possible states s , averaging over all possible next states s' at each iteration, as illustrated in Figure 36.11(right). However, for some problems, we may only be interested in the value (and policy) for certain special starting states. This is the case, for example, in **shortest path problems** on graphs, where we are trying to find the shortest route from the current state to a goal state. This can be modeled as an episodic MDP by defining a

¹ transition matrix $p_T(s'|s, a)$ where taking edge a from node s leads to the neighboring node s' with probability 1. The reward function is defined as $R(s, a) = -1$ for all states s except the goal states, which are modeled as absorbing states.

⁵ In problems such as this, we can use a method known as **real-time dynamic programming** (RTDP) [BBS95], to efficiently compute an **optimal partial policy**, which only specifies what to do for the reachable states. RTDP maintains a value function estimate V . At each step, it performs a Bellman backup for the current state s by $V(s) \leftarrow \max_a \mathbb{E}_{p_T(s'|s, a)} [R(s, a) + \gamma V(s')]$. It can picks an action a (often with some exploration), reaches a next state s' , and repeats the process. This can be seen as a form of the more general **asynchronous value iteration**, that focuses its computational effort on parts of the state space that are more likely to be reachable from the current state, rather than synchronously updating all states at each iteration.

¹³

¹⁴ 36.6.2 Policy iteration

¹⁵ Another effective DP method for computing π_* is **policy iteration**. It is an iterative algorithm that ¹⁶ searches in the space of deterministic policies until converging to an optimal policy. Each iteration ¹⁷ consists of two steps, **policy evaluation** and **policy improvement**.

¹⁸ The policy evaluation step, as mentioned earlier, computes the value function for the current ¹⁹ policy. Let π represent the current policy, $\mathbf{v}(s) = V_\pi(s)$ represent the value function encoded as ²⁰ a vector indexed by states, $\mathbf{r}(s) = \sum_a \pi(a|s) R(s, a)$ represent the reward vector, and $\mathbf{T}(s'|s) = \sum_a \pi(a|s) p(s'|s, a)$ represent the state transition matrix. Bellman's equation for policy evaluation ²¹ ²² ²³ can be written in the matrix-vector form as

$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{T}\mathbf{v} \quad (36.64)$$

²⁴ This is a linear system of equations in $|\mathcal{S}|$ unknowns, We can solve it using matrix inversion: ²⁵ $\mathbf{v} = (\mathbf{I} - \gamma \mathbf{T})^{-1} \mathbf{r}$. Alternatively, we can use value iteration by computing $\mathbf{v}_{t+1} = \mathbf{r} + \gamma \mathbf{T}\mathbf{v}_t$ until near ²⁶ convergence, or some form of asynchronous variant that is computationally more efficient.

²⁷ Once we have evaluated V_π for the current policy π , we can use it to derive a better policy π' , thus ²⁸ the name policy improvement. To do this, we simply compute a deterministic policy π' that acts ²⁹ greedily with respect to V_π in every state; that is, $\pi'(s) = \operatorname{argmax}_a \{R(s, a) + \gamma \mathbb{E}[V_\pi(s')]\}$. We can ³⁰ ³¹ ³² guarantee that $V_{\pi'} \geq V_\pi$. To see this, define \mathbf{r}' , \mathbf{T}' and \mathbf{v}' as before, but for the new policy π' . The ³³ definition of π' implies $\mathbf{r}' + \gamma \mathbf{T}'\mathbf{v} \geq \mathbf{r} + \gamma \mathbf{T}\mathbf{v} = \mathbf{v}$, where the equality is due to Bellman's equation. ³⁴ Repeating the same equality, we have

$$\mathbf{v} \leq \mathbf{r}' + \gamma \mathbf{T}'\mathbf{v} \leq \mathbf{r}' + \gamma \mathbf{T}'(\mathbf{r}' + \gamma \mathbf{T}'\mathbf{v}) \leq \mathbf{r}' + \gamma \mathbf{T}'(\mathbf{r}' + \gamma \mathbf{T}'(\mathbf{r}' + \gamma \mathbf{T}'\mathbf{v})) \leq \dots \quad (36.65)$$

$$= (\mathbf{I} + \gamma \mathbf{T}' + \gamma^2 \mathbf{T}'^2 + \dots) \mathbf{r} = (\mathbf{I} - \gamma \mathbf{T}')^{-1} \mathbf{r} = \mathbf{v}' \quad (36.66)$$

³⁵ Starting from an initial policy π_0 , policy iteration alternates between policy evaluation (E) and ³⁶ improvement (I) steps, as illustrated below:

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_* \quad (36.67)$$

³⁷ The algorithm stops at iteration k , if the policy π_k is greedy with respect to its own value function ³⁸ V_{π_k} . In this case, the policy is optimal. Since there are at most $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies, and ³⁹ ⁴⁰ every iteration strictly improves the policy, the algorithm must converge after finite iterations.

⁴¹

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

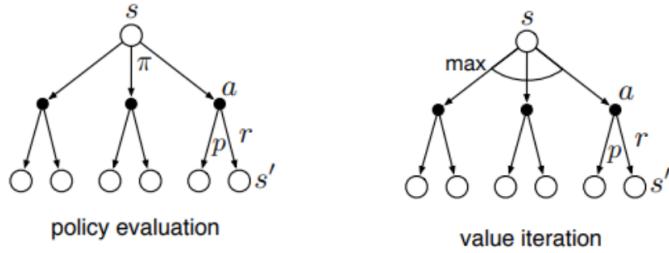


Figure 36.11: Policy iteration vs value iteration represented as backup diagrams. Empty circles represent states, solid (filled) circles represent actions. Adapted from Figure 8.6 of [SB18].

In PI, we alternate between policy evaluation (which involves multiple iterations, until convergence of V_π), and policy improvement. In VI, we alternate between one iteration of policy evaluation followed by one iteration of policy improvement (the “max” operator in the update rule). In **generalized policy improvement**, we are free to intermix any number of these steps in any order. The process will converge once the policy is greedy wrt its own value function.

Note that policy evaluation computes V_π whereas value iteration computes V_* . This difference is illustrated in Figure 36.11, using a **backup diagram**. Here the root node represents any state s , nodes at the next level represent state-action combinations (solid circles), and nodes at the leaves representing the set of possible resulting next state s' for each possible action. In the former case, we average over all actions according to the policy, whereas in the latter, we take the maximum over all actions.

36.6.3 Linear programming

While dynamic programming is effective and popular, linear programming (LP) provides an alternative that finds important uses, such as in off-policy RL (Section 37.5). The primal form of LP is given by

$$\min_V \sum_s p_0(s)V(s) \quad \text{s.t.} \quad V(s) \geq R(s, a) + \gamma \sum_{s'} p_T(s'|s, a)V(s), \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \quad (36.68)$$

where $p_0(s) > 0$ for all $s \in \mathcal{S}$, and can be interpreted as the initial state distribution. It can be verified that any V satisfying the constraint in Equation (36.68) is optimistic [Put94], that is, $V \geq V_*$. When the objective is minimized, the solution V will be “pushed” to the smallest possible, which is V_* . Once V_* is found, any action a that makes the constraint tight in state s is optimal in that state.

The dual LP form is sometimes more intuitive:

$$\max_{d \geq 0} \sum_{s, a} d(s, a)R(s, a) \quad \text{s.t.} \quad \sum_a d(s, a) = (1 - \gamma)p_0(s) + \gamma \sum_{\bar{s}, \bar{a}} p_T(s|\bar{s}, \bar{a})d(\bar{s}, \bar{a}) \quad \forall s \in \mathcal{S} \quad (36.69)$$

Any nonnegative d satisfying the constraint above is the **normalized occupancy distribution** of

1
2 some corresponding policy $\pi_d(a|s) \triangleq d(s, a) / \sum_{a'} d(s, a')$: ⁴

3
4 $d(s, a) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t p(s_t = s, a_t = a | s_0 \sim p_0, a_t \sim \pi_d(s_t))$ (36.70)
5
6

7 The constant $(1 - \gamma)$ normalizes d to be a valid distribution, so that it sums to unity. With this
8 interpretation of d , the objective in Equation (36.69) is just the average per-step reward under the
9 normalized occupancy distribution. Once an optimal solution d_* is found, an optimal policy can be
10 immediately obtained by $\pi_*(a|s) = d_*(s, a) / \sum_{a'} d_*(s, a')$.

11 A challenge in solving the primal or dual LPs for MDPs is the large number of constraints and
12 variables. Approximations are needed, where the variables are parameterized (either linearly or
13 nonlinearly), and the the constraints are sampled or approximated (see e.g., [dV04; LBS17; CLW18]).

1415161718192021222324252627282930313233343536373839404142434445

46 4. If $\sum_{a'} d(s, a') = 0$ for some state s , then $\pi_d(s)$ may be defined arbitrarily, since s is not visited under the policy.
47

37 Reinforcement learning

This chapter was co-authored with Lihong Li.

37.1 Introduction

Reinforcement learning or **RL** is a paradigm of learning where an agent sequentially interacts with an initially unknown environment. The interaction typically results in a **trajectory**, or multiple trajectories. Let $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, \dots, s_T)$ be a trajectory of length T , consisting of a sequence of states s_t , actions a_t , and rewards r_t .¹ The goal of the agent is to optimize her action-selection policy, so that the discounted cumulative reward, $G_0 \triangleq \sum_{t=0}^{T-1} \gamma^t r_t$, is maximized for some given **discount factor** $\gamma \in [0, 1]$.

In general, G_0 is a random variable. We will focus on maximizing its expectation, inspired by the maximum expected utility principle (Section 3.8.1), but note other possibilities such as **conditional value at risk**² that can be more appropriate in risk-sensitive applications.

We will focus on the Markov decision process, where the generative model for the trajectory τ can be factored into single-step models. When these model parameters are known, solving for an optimal policy is called **planning** (see Section 36.6); otherwise, RL algorithms may be used to obtain an optimal policy from trajectories, a process called **learning**.

In **model-free RL**, we try to learn the policy without explicitly representing and learning the models, but directly from the trajectories. In **model-based RL**, we first learn a model from the trajectories, and then use a planning algorithm on the learned model to solve for the policy. This chapter will introduce some of the key concepts and techniques, and will mostly follow the notation from [SB18]. More details can be found in textbooks such as [Sze10; SB18; Ber19; Aga+21a; Mey22], and reviews such as [WO12; Aru+17; FL+18; Li18].

37.1.1 Overview of methods

In this section, we give a brief overview of how to compute optimal policies when the MDP model is not known. Instead, the agent interacts with the environment and learns from the observed

1. Note that the time starts at 0 here, while it starts at 1 when we discuss bandits (Section 36.4). Our choices of notation is to be consistent with conventions in respective literature.

2. The conditional value at risk, or CVaR, is the expected reward conditioned on being in the worst 5% (say) of samples. See [Cho+15] for an example application in RL.

Method	Functions learned	On/Off	Section
SARSA	$Q(s, a)$	On	Section 37.2.4
Q -learning	$Q(s, a)$	Off	Section 37.2.5
REINFORCE	$\pi(a s)$	On	Section 37.3.2
A2C	$\pi(a s), V(s)$	On	Section 37.3.3.1
TRPO / PPO	$\pi(a s), A(s, a)$	On	Section 37.3.4
DDPG	$a = \pi(s), Q(s, a)$	Off	Section 37.3.5
Soft actor-critic	$\pi(a s), Q(s, a)$	Off	Section 37.6.1
Model-based RL	$p(s' s, a)$	Off	Section 37.4

11 *Table 37.1: Summary of some popular methods for RL. On/off refers to on-policy vs off-policy methods.*

12

13
14 trajectories. This is the core focus of RL. We will go into more details into later sections, but first
15 provide this roadmap.

16 We may categorize RL methods by the quantity the agent represents and learns: value function,
17 policy, and model; or by how actions are selected: on-policy (actions must be selected by the agent's
18 current policy), and off-policy. Table 37.1 lists a few representative examples. More details are given
19 in the subsequent sections. We will also discuss at greater depth two important topics of off-policy
20 learning and inference-based control in Sections 37.5 and 37.6.

21

22 37.1.2 Value based methods

23 In a value based method, we often try to learn the optimal Q -function from experience, and then
24 derive a policy from it using Equation (36.60). Typically, a function approximator (e.g., a neural
25 network), Q_w , is used to represent the Q -function, which is trained iteratively. Given a transition
26 (s, a, r, s') , we define the **temporal difference** (also called the **TD error**) as

27
$$r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)$$

28
29 Clearly, the expected TD error is the Bellman error evaluated at (s, a) . Therefore, if $Q_w = Q_*$, the
30 TD error is 0 on average by Bellman's optimality equation. Otherwise, the error provides a signal for
31 the agent to change w to make $Q_w(s, a)$ closer to $R(s, a) + \gamma \max_{a'} Q_w(s', a')$. The update on Q_w
32 is based on a target that is computed using Q_w . This kind of update is known as **bootstrapping**
33 in RL, and should not be confused with the statistical bootstrap (Section 13.2.3.1). Value based
34 methods such as **Q-learning** and **SARSA** are discussed in Section 37.2.

35

36 37.1.3 Policy search methods

37

38 In **policy search**, we try to directly maximize $J(\pi_\theta)$ wrt the policy parameter θ . If $J(\pi_\theta)$ is
39 differentiable wrt θ , we can use stochastic gradient ascent to optimize θ , which is known as **policy**
40 **gradient**, as described in Section 37.3.1. The basic idea is to perform **Monte Carlo rollouts**, in
41 which we sample trajectories by interacting with the environment, and then use the score function
42 estimator (Section 6.6.3) to estimate $\nabla_\theta J(\pi_\theta)$. Here, $J(\pi_\theta)$ is defined as an expectation whose
43 distribution depends on θ , so it is invalid to swap ∇ and \mathbb{E} in computing the gradient, and the score
44 function estimator can be used instead. An example of policy gradient is **REINFORCE**.

45

Policy gradient methods have the advantage that they provably converge to a local optimum for many common policy classes, whereas Q -learning may diverge when approximation is used (Section 37.5.3). In addition, policy gradient methods can easily be applied to continuous action spaces, since they do not need to compute $\text{argmax}_a Q(s, a)$. Unfortunately, the score function estimator for $\nabla_{\theta} J(\pi_{\theta})$ can have a very high variance, so the resulting method can converge slowly.

One way to reduce the variance is to learn an approximate value function, $V_w(s)$, and to use it as a baseline in the score function estimator. We can learn $V_w(s)$ using one of the value function methods similar to Q -learning. Alternatively, we can learn an advantage function, $A_w(s, a)$, and use it to estimate the gradient. These policy gradient variants are called **actor critic** methods, where the actor refers to the policy π_{θ} and the critic refers to V_w or A_w . See Section 37.3.3 for details.

37.1.4 Model-based RL

Value-based methods, such as Q -learning, and policy search methods, such as policy gradient, can be very **sample inefficient**, which means they may need to interact with the environment many times before finding a good policy. If an agent has prior knowledge of the MDP model, it can be more sample efficient to first learn the model, and then compute an optimal (or near-optimal) policy of the model without having to interact with the environment any more.

This approach is called **model-based RL**. The first step is to learn the MDP model including the $p_T(s'|s, a)$ and $R(s, a)$ functions, e.g., using DNNs. Given a collection of (s, a, r, s') tuples, such a model can be learned using standard supervised learning methods. The second step can be done by running an RL algorithm on synthetic experiences generated from the model, or by running a planning algorithm on the model directly (Section 36.6). In practice, we often interleave the model learning and planning phases, so we can use the partially learned policy to decide what data to collect. We discuss model-based RL in more detail in Section 37.4.

37.1.5 Exploration-exploitation tradeoff

A fundamental problem in RL with unknown transition and reward models is to decide between choosing actions that the agent knows will yield high reward, or choosing actions whose reward is uncertain, but which may yield information that helps the agent get to parts of state-action space with even higher reward. This is called the **exploration-exploitation tradeoff**, which has been discussed in the simpler contextual bandit setting in Section 36.4. The literature on efficient exploration is huge. In this section, we briefly describe several representative techniques.

37.1.5.1 ϵ -greedy

A common heuristic is to use an **ϵ -greedy** policy π_{ϵ} , parameterized by $\epsilon \in [0, 1]$. In this case, we pick the greedy action wrt the current model, $a_t = \text{argmax}_a \hat{R}_t(s_t, a)$ with probability $1 - \epsilon$, and a random action with probability ϵ . This rule ensures the agent's continual exploration of all state-action combinations. Unfortunately, this heuristic can be shown to be suboptimal, since it explores every action with at least a constant probability $\epsilon/|\mathcal{A}|$.

	$\hat{R}(s, a_1)$	$\hat{R}(s, a_2)$	$\pi_\epsilon(a s_1)$	$\pi_\epsilon(a s_2)$	$\pi_\tau(a s_1)$	$\pi_\tau(a s_2)$
1	1.00	9.00	0.05	0.95	0.00	1.00
2	4.00	6.00	0.05	0.95	0.12	0.88
3	4.90	5.10	0.05	0.95	0.45	0.55
4	5.05	4.95	0.95	0.05	0.53	0.48
5	7.00	3.00	0.95	0.05	0.98	0.02
6	8.00	2.00	0.95	0.05	1.00	0.00
7						
8						

9
10 Table 37.2: Comparison of ϵ -greedy policy (with $\epsilon = 0.1$) and Boltzmann policy (with $\tau = 1$) for a simple
11 MDP with 6 states and 2 actions. Adapted from Table 4.1 of [GK19].

12

13 37.1.5.2 Boltzmann exploration

14 A source of inefficiency in the ϵ -greedy rule is that exploration occurs uniformly over all actions.
15 The **Boltzmann policy** can be more efficient, by assigning higher probabilities to explore more
16 promising actions:

$$17 \quad \pi_\tau(a|s) = \frac{\exp(\hat{R}_t(s_t, a)/\tau)}{\sum_{a'} \exp(\hat{R}_t(s_t, a')/\tau)} \quad (37.1)$$

18 where $\tau > 0$ is a temperature parameter that controls how entropic the distribution is. As τ gets
19 close to 0, π_τ becomes close to a greedy policy. On the other hand, higher values of τ will make
20 $\pi(a|s)$ more uniform, and encourage more exploration. Its action selection probabilities can be much
21 “smoother” with respect to changes in the reward estimates than ϵ -greedy, as illustrated in Table 37.2.

22 37.1.5.3 Upper confidence bounds and Thompson sampling

23 The upper confidence bound (UCB) (Section 36.4.5) and Thompson sampling (Section 36.4.6)
24 approaches may also be extended to MDPs. In contrast to the contextual bandit case, where the
25 only uncertainty is in the reward function, here we must also take into account uncertainty in the
26 transition probabilities.

27 As in the bandit case, the UCB approach requires to estimate an upper confidence bound for all
28 actions' Q -values in the current state, and then take the action with the highest UCB score. One way
29 to obtain UCBs of the Q -values is to use **count-based exploration**, where we learn the optimal
30 Q -function with an **exploration bonus** added to the reward in a transition (s, a, r, s') :

$$31 \quad \tilde{r} = r + \alpha / \sqrt{N_{s,a}} \quad (37.2)$$

32 where $N_{s,a}$ is the number of times action a has been taken in state s , and $\alpha \geq 0$ is a weighting term
33 that controls the degree of exploration. This is the approach taken by the **MBIE-EB** method [SL08]
34 for finite-state MDPs, and in the generalization to continuous-state MDPs through the use of
35 hashing [Bel+16]. Other approaches also explicitly maintain uncertainty in state transition proba-
36 bilities, and use that information to obtain UCBs. Examples are **MBIE** [SL08], **UCRL2** [JOA10],
37 **UCBVI** [AOM17], among many others.

38 Thompson sampling can be similarly adapted, by maintaining the posterior distribution of the
39 reward and transition model parameters. In finite-state MDPs, for example, the transition model is a
40

categorical distribution conditioned on the state. We may use the conjugate prior of Dirichlet distributions (Section 3.2) for the transition model, so that the posterior distribution can be conveniently computed and sampled from. More details on this approach are found in [Rus+18].

Both UCB and Thompson sampling methods have been shown to yield efficient exploration with provably strong regret bounds (Section 36.4.7) [JOA10], or related PAC bounds [SLL09; DLB17], often under necessary assumptions such as finiteness of the MDPs. In practice, these methods may be combined with function approximation like neural networks and implemented approximately.

37.1.5.4 Optimal solution using Bayes-adaptive MDPs

The Bayes optimal solution to the exploration-exploitation tradeoff can be computed by formulating the problem as a special kind of POMDP known as a **Bayes-adaptive MDP** or **BAMDP** [Duf02]. This extends the Gittins index approach in Section 36.4.4 to the MDP setting.

In particular, a BAMDP has a **belief state** space, \mathcal{B} , representing uncertainty about the reward model $p_R(r|s, a, s')$ and transition model $p_T(s'|s, a)$. The transition model on this augmented MDP can be written as follows:

$$T^+(s_{t+1}, b_{t+1}|s_t, b_t, a_t, r_t) = T^+(s_{t+1}|s_t, a_t, b_t)T^+(b_{t+1}|s_t, a_t, r_t, s_{t+1}) \quad (37.3)$$

$$= \mathbb{E}_{b_t}[T(s_{t+1}|s_t, a_t)] \times \mathbb{I}(b_{t+1} = p(R, T|\mathbf{h}_{t+1})) \quad (37.4)$$

where $\mathbb{E}_{b_t}[T(s_{t+1}|s_t, a_t)]$ is the posterior predictive distribution over next states, and $p(R, T|\mathbf{h}_{t+1})$ is the new belief state given $\mathbf{h}_{t+1} = (s_{1:t+1}, a_{1:t+1}, r_{1:t+1})$, which can be computed using Bayes rule. Similarly, the reward function for the augmented MDP is given by

$$R^+(r|s_t, b_t, a_t, s_{t+1}, b_{t+1}) = \mathbb{E}_{b_{t+1}}[R(s_t, a_t, s_{t+1})] \quad (37.5)$$

For small problems, we can solve the resulting augmented MDP optimally. However, in general this is computationally intractable. [Gha+15] surveys many methods to solve it more efficiently. For example, [KN09] develop an algorithm that behaves similarly to Bayes optimal policies, except in a provably small number of steps; [GSD13] propose an approximate method based on Monte Carlo rollouts. More recently, [Zin+20] propose an approximate method based on meta-learning (Section 20.6), in which they train a (model-free) policy for multiple related tasks. Each task is represented by a task embedding vector m , which is inferred from \mathbf{h}_t using a VAE (Section 22.2). The posterior $p(m|\mathbf{h}_t)$ is used as a proxy for the belief state b_t , and the policy is trained to perform well given s_t and b_t . At test time, the policy is applied to the incrementally computed belief state; this allows the method to infer what kind of task this is, and then to use a pre-trained policy to quickly solve it.

37.2 Value-based RL

In this section, we assume the agent has access to samples from p_T and p_R by interacting with the environment. We will show how to use these samples to learn optimal Q -functions from which we can derive optimal policies.

¹
² **37.2.1 Monte Carlo RL**

³ Recall that $Q_\pi(s, a) = \mathbb{E}[G_t | s_t = s, a_t = a]$ for any t . A simple way to estimate this is to take action
⁴ a , and then sample the rest of the trajectory according to π , and then compute the average sum of
⁵ discounted rewards. The trajectory ends when we reach a terminal state, if the task is episodic, or
⁶ when the discount factor γ^t becomes negligibly small, whichever occurs first. This is the **Monte
⁷ Carlo estimation** of the value function.

⁸ We can use this technique together with policy iteration (Section 36.6.2) to learn an optimal policy.
⁹ Specifically, at iteration k , we compute a new, improved policy using $\pi_{k+1}(s) = \operatorname{argmax}_a Q_k(s, a)$,
¹⁰ where Q_k is approximated using MC estimation. This update can be applied to all the states visited
¹¹ on the sampled trajectory. This overall technique is called **Monte Carlo control**.

¹² To ensure this method converges to the optimal policy, we need to collect data for every (state,
¹³ action) pair, at least in the tabular case, since there is no generalization across different values of
¹⁴ $Q(s, a)$. One way to achieve this is to use an ϵ -greedy policy. Since this is an on-policy algorithm,
¹⁵ the resulting method will converge to the optimal ϵ -soft policy, as opposed to the optimal policy. It
¹⁶ is possible to use importance sampling to estimate the value function for the optimal policy, even if
¹⁷ actions are chosen according to the ϵ -greedy policy. However, it is simpler to just gradually reduce ϵ .
¹⁸

¹⁹
²⁰ **37.2.2 Temporal difference (TD) learning**

²¹ The Monte Carlo (MC) method in Section 37.2.1 results in an estimator for $Q_\pi(s, a)$ with very high
²² variance, since it has to unroll many trajectories, whose returns are a sum of many random rewards
²³ generated by stochastic state transitions. In addition, it is limited to episodic tasks (or finite horizon
²⁴ truncation of continuing tasks), since it must unroll to the end of the episode before each update
²⁵ step, to ensure it reliably estimates the long term return.

²⁶ In this section, we discuss a more efficient technique called **temporal difference** or **TD** learning
²⁷ [Sut88]. The basic idea is to incrementally reduce the Bellman error for sampled states or state-actions,
²⁸ based on transitions instead of a long trajectory. More precisely, suppose we are to learn the value
²⁹ function V_π for a fixed policy π . Given a state transition (s, a, r, s') where $a \sim \pi(s)$, we change the
³⁰ estimate $V(s)$ so that it moves toward the bootstrapping target (Section 37.1.2)

³¹
³²
$$V(s_t) \leftarrow V(s_t) + \eta [r_t + \gamma V(s_{t+1}) - V(s_t)] \quad (37.6)$$

³³ where η is the learning rate. The term multiplied by η above is known as the **TD error**. A more
³⁴ general form of TD update for parametric value function representations is
³⁵

³⁶
³⁷
$$\mathbf{w} \leftarrow \mathbf{w} + \eta [r_t + \gamma V_{\mathbf{w}}(s_{t+1}) - V_{\mathbf{w}}(s_t)] \nabla_{\mathbf{w}} V_{\mathbf{w}}(s_t) \quad (37.7)$$

³⁸ of which Equation (37.6) is a special case. The TD update rule for learning Q_π is similar.

³⁹ It can be shown that TD learning in the tabular case, Equation (37.6), converges to the correct
⁴⁰ value function, under proper conditions [Ber19]. However, it may diverge when approximation is
⁴¹ used (Equation (37.7)), an issue we will discuss further in Section 37.5.3.

⁴² The potential divergence of TD is also consistent with the fact that Equation (37.7) is not SGD
⁴³ (Section 6.3) on any objective function, despite a very similar form. Instead, it is an example of
⁴⁴ **bootstrapping**, in which the estimate, $V_{\mathbf{w}}(s_t)$, is updated to approach a target, $r_t + \gamma V_{\mathbf{w}}(s_{t+1})$,
⁴⁵ which is defined by the value function estimate itself. This idea is shared by DP methods like value
⁴⁶ iteration, although they rely on the complete MDP model to compute an exact Bellman backup. In
⁴⁷

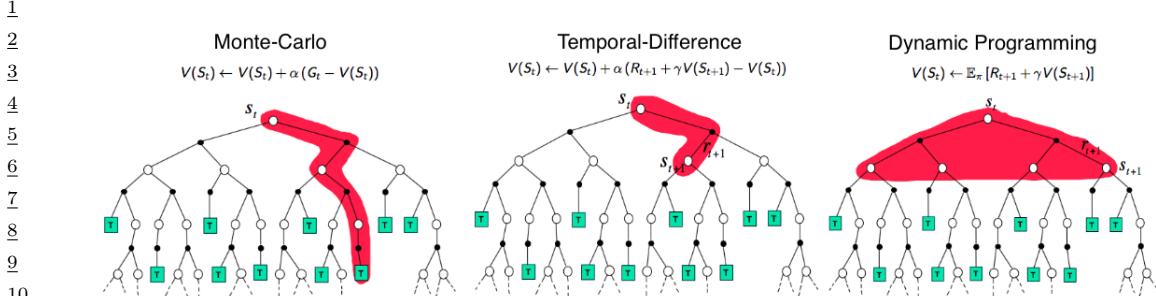


Figure 37.1: Backup diagrams of $V(s_t)$ for Monte Carlo, temporal difference, and dynamic programming updates of the state-value function. Used with kind permission of Andy Barto.

contrast, TD learning can be viewed as using sampled transitions to approximate such backups. An example of non-bootstrapping approach is the Monte Carlo estimation in the previous section. It samples a complete trajectory, rather than individual transitions, to perform an update, and is often much less efficient. Figure 37.1 illustrates the difference between MC, TD, and DP.

37.2.3 TD learning with eligibility traces

A key difference between TD and MC is the way they estimate returns. Given a trajectory $\tau = (s_0, a_0, r_0, s_1, \dots, s_T)$, TD estimates the return from state s_t by one-step lookahead, $G_{t:t+1} = r_t + \gamma V(s_{t+1})$, where the return from time $t+1$ is replaced by its value function estimate. In contrast, MC waits until the end of the episode or until T is large enough, then uses the estimate $G_{t:T} = r_t + \gamma r_{t+1} + \dots + \gamma^{T-t-1} r_{T-1}$. It is possible to interpolate between these by performing an n -step rollout, and then using the value function to approximate the return for the rest of the trajectory, similar to heuristic search (Section 37.4.1.1). That is, we can use the n -step estimate

$$G_{t:t+n} = r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}) \quad (37.8)$$

The corresponding n -step version of the TD update becomes

$$V(s_t) \leftarrow V(s_t) + \eta [G_{t:t+n} - V(s_t)] \quad (37.9)$$

Rather than picking a specific lookahead value, n , we can take a weighted average of all possible values, with a single parameter $\lambda \in [0, 1]$, by using

$$G_t^\lambda \triangleq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \quad (37.10)$$

This is called the **λ -return**. The coefficient of $1 - \lambda = (1 + \lambda + \lambda^2 + \dots)^{-1}$ in the front ensures this is a convex combination of n -step returns. See Figure 37.2 for an illustration.

An important benefit of using the geometric weighting in Equation (37.10) is that the corresponding TD learning update can be efficiently implemented, through the use of **eligibility traces**, even though G_t^λ is a sum of infinitely many terms. The method is called **TD(λ)**, and can be combined with many algorithms to be studied in the rest of the chapter. See [SB18] for a detailed discussion.

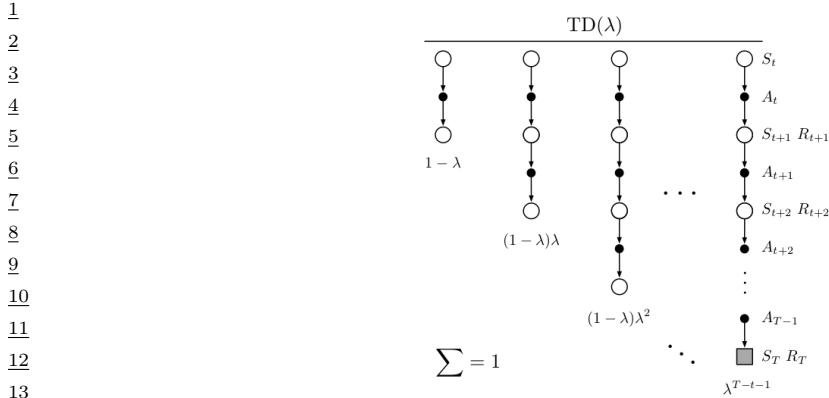


Figure 37.2: The backup diagram for TD(λ). Standard TD learning corresponds to $\lambda = 0$, and standard MC learning corresponds to $\lambda = 1$. From Figure 12.1 of [SB18]. Used with kind permission of Richard Sutton.

37.2.4 SARSA: on-policy TD control

TD learning is for policy evaluation, as it estimates the value function for a fixed policy. In order to find an optimal policy, we may use the algorithm as a building block inside generalized policy iteration (Section 36.6.2). In this case, it is more convenient to work with the action-value function, Q , and a policy π that is greedy with respect to Q . The agent follows π in every step to choose actions, and upon a transition (s, a, r, s') the TD update rule is

$$Q(s, a) \leftarrow Q(s, a) + \eta [r + \gamma Q(s', a') - Q(s, a)] \quad (37.11)$$

where $a' \sim \pi(s')$ is the action the agent will take in state s' . After Q is updated (for policy evaluation), π also changes accordingly as it is greedy with respect to Q (for policy improvement). This algorithm, first proposed by [RN94], was further studied and renamed to **SARSA** by [Sut96]; the name comes from its update rule that involves an augmented transition (s, a, r, s', a') .

In order for SARSA to converge to Q_* , every state-action pair must be visited infinitely often, at least in the tabular case, since the algorithm only updates $Q(s, a)$ for (s, a) that it visits. One way to ensure this condition is to use a “greedy in the limit with infinite exploration” (**GLIE**) policy. An example is the ϵ -greedy policy, with ϵ vanishing to 0 gradually. It can be shown that SARSA with a GLIE policy will converge to Q_* and π_* [Sin+00].

39

37.2.5 Q-learning: off-policy TD control

41

SARSA is an **on-policy** algorithm, which means it learns the Q -function for the policy it is currently using, which is typically not the optimal policy (except in the limit for a GLIE policy). However, with a simple modification, we can convert this to an **off-policy** algorithm that learns Q_* , even if a suboptimal policy is used to choose actions.

The idea is to replace the sampled next action $a' \sim \pi(s')$ in Equation (37.11) with a greedy action

in s' : $a' = \operatorname{argmax}_b Q(s', b)$. This results in the following update when a transition (s, a, r, s') happens

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[r + \gamma \max_b Q(s', b) - Q(s, a) \right] \quad (37.12)$$

This is the update rule of **Q-learning** for the tabular case [WD92]. The extension to work with function approximation can be done in a way similar to Equation (37.7). Since it is off-policy, the method can use (s, a, r, s') triples coming from any data source, such as older versions of the policy, or log data from an existing (non-RL) system. If every state-action pair is visited infinitely often, the algorithm provably converges to Q_* in the tabular case, with properly decayed learning rates [Ber19]. Algorithm 35 gives a vanilla implementation of Q-learning with ϵ -greedy exploration.

Algorithm 35: Q-learning with ϵ -greedy exploration

```

1 Initialize value function parameters  $w$ 
2 repeat
3   Sample starting state  $s$  of new episode
4   repeat
5     Sample action  $a = \begin{cases} \operatorname{argmax}_b Q_w(s, b), & \text{with probability } 1 - \epsilon \\ \text{random action,} & \text{with probability } \epsilon \end{cases}$ 
6     Observe state  $s'$ , reward  $r$ 
7     Compute the TD error:  $\delta = r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)$ 
8      $w \leftarrow w + \eta \delta \nabla_w Q_w(s, a)$ 
9      $s \leftarrow s'$ 
10    until state  $s$  is terminal;
11 until converged;

```

37.2.5.1 Example

Figure 37.3 gives an example of Q-learning applied to the simple 1d grid world from Figure 36.10, using $\gamma = 0.9$. We show the Q -function at the start and end of each episode, after performing actions chosen by an ϵ -greedy policy. We initialize $Q(s, a) = 0$ for all entries, and use a step size of $\eta = 1$, so the update becomes $Q_*(s, a) = r + \gamma Q_*(s', a_*)$, where $a_* = \downarrow$ for all states.

37.2.5.2 Double Q-learning

Standard Q-learning suffers from a problem known as the **optimizer's curse** [SW06], or the **maximization bias**. The problem refers to the simple statistical inequality, $\mathbb{E}[\max_a X_a] \geq \max_a \mathbb{E}[X_a]$, for a set of random variables $\{X_a\}$. Thus, if we pick actions greedily according to their random scores $\{X_a\}$, we might pick a wrong action just because random noise makes it appealing.

Figure 37.4 gives a simple example of how this can happen in an MDP. The start state is A. The right action gives a reward 0 and terminates the episode. The left action also gives a reward of 0, but then enters state B, from which there are many possible actions, with rewards drawn from $\mathcal{N}(-0.1, 1.0)$. Thus the expected return for any trajectory starting with the left action is

	Q-function episode start	Episode	Time Step	Action	(s, r, s')	$rQ^*(s', \cdot)$	Q-function episode end
Q_1	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 0 \\ \hline \end{array}$	1	1	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0 = 0$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 1 \\ \hline \end{array}$
	1	2	\uparrow	$(S, U, 0, S)$	$0 + 0.9 \times 0 = 0$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 1 \\ \hline \end{array}$	
	1	3	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0 = 0$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 1 & 0 \\ \hline \end{array}$	
	1	4	\downarrow	$(S, U, 0, S)$	$0 + 0.9 \times 0 = 0$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 1 \\ \hline S & 1 & 0 \\ \hline \end{array}$	
	1	5	\downarrow	$(S, D, 1, S)$		UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 1 \\ \hline S & 0 & 1 \\ \hline \end{array}$	
Q_2	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 1 \\ \hline \end{array}$	2	1	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0 = 0$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 0 \\ \hline \end{array}$
	2	2	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 1 = 0.9$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0.9 \\ \hline 0 & 0 & 0 \\ \hline S & 1 & 0 \\ \hline \end{array}$	
	2	3	\downarrow	$(S, D, 0, S)$		UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 1 \\ \hline \end{array}$	
Q_3	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 0.9 \\ \hline \end{array}$	3	1	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0.9 = 0.81$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0 \\ \hline 0 & 0 & 0.81 \\ \hline 0 & 0 & 0 \\ \hline S & 0 & 0.9 \\ \hline \end{array}$
	3	2	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 1 = 0.9$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0 & 0.9 \\ \hline 0 & 0 & 0 \\ \hline S & 1 & 0.81 \\ \hline \end{array}$	
	3	3	\uparrow	$(S, D, 0, S)$	$0 + 0.9 \times 0.9 = 0.81$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0 & 0.9 \\ \hline 0 & 0 & 0.81 \\ \hline S & 0 & 0.9 \\ \hline \end{array}$	
	3	4	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 1 = 0.9$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 1 & 0.81 \\ \hline \end{array}$	
	3	5	\downarrow	$(S, D, 0, S)$		UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 1 & 1 \\ \hline \end{array}$	
Q_4	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$	4	1	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0.9 = 0.81$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0.81 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$
	4	2	\uparrow	$(S, U, 0, S)$	$0 + 0.9 \times 0.81 = 0.73$	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0.81 & 0 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$	
	4	3	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0.9 = 0.81$	UP DOWN $\begin{array}{ c c } \hline S & 0.73 & 0.9 \\ \hline 0 & 0.81 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$	
	4	4	\uparrow	$(S, U, 0, S)$	$0 + 0.9 \times 0.81 = 0.73$	UP DOWN $\begin{array}{ c c } \hline S & 0.73 & 0.9 \\ \hline 0.81 & 0 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$	
	4	5	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 0.9 = 0.81$	UP DOWN $\begin{array}{ c c } \hline S & 0.81 & 0.9 \\ \hline 0.81 & 0 & 0.9 \\ \hline 0 & 0 & 1 \\ \hline S & 1 & 0 \\ \hline \end{array}$	
	4	6	\downarrow	$(S, D, 0, S)$	$0 + 0.9 \times 1 = 0.9$	UP DOWN $\begin{array}{ c c } \hline S & 0.81 & 0.9 \\ \hline 0.81 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline S & 1 & 1 \\ \hline \end{array}$	
	4	7	\downarrow	$(S, D, 0, S)$		UP DOWN $\begin{array}{ c c } \hline S & 0.81 & 0.9 \\ \hline 0.81 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline S & 1 & 1 \\ \hline \end{array}$	
Q_5	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0 & 0.73 & 0.9 \\ \hline 0 & 0.81 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$	5	1	\uparrow	$(S_1, U, 0, S)$	0	UP DOWN $\begin{array}{ c c } \hline S & 0 & 0.81 \\ \hline 0.73 & 0 & 0.9 \\ \hline 0.81 & 0 & 1 \\ \hline S & 0.81 & 1 \\ \hline \end{array}$

Figure 37.3: Illustration of Q learning for the 1d grid world in Figure 36.10 using ϵ -greedy exploration. At the end of episode 1, we make a transition from S_3 to S_{T2} and get a reward of $r = 1$, so we estimate $Q(S_3, \downarrow) = 1$. In episode 2, we make a transition from S_2 to S_3 , so S_2 gets incremented by $\gamma Q(S_3, \downarrow) = 0.9$. Adapted from Figure 3.3 of [GK19].

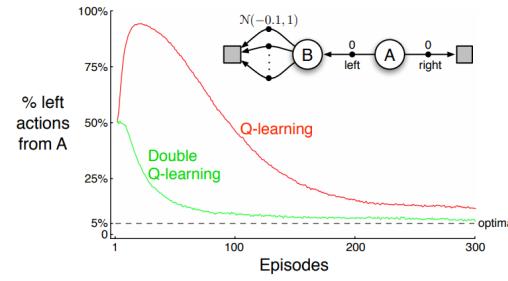


Figure 37.4: Comparison of Q -learning and double Q -learning on a simple episodic MDP using ϵ -greedy action selection with $\epsilon = 0.1$. The initial state is A , and squares denote absorbing states. The data are averaged over 10,000 runs. From Figure 6.5 of [SB18]. Used with kind permission of Rich Sutton.

46
47

-0.1 , making it suboptimal. Nevertheless, the RL algorithm may pick the left action due to the maximization bias making B appear to have a positive value.

One solution to avoid the maximization bias is to use two separate Q -functions, Q_1 and Q_2 , one for selecting the greedy action, and the other for estimating the corresponding Q -value. In particular, upon seeing a transition (s, a, r, s') , we perform the following update

$$Q_1(s, a) \leftarrow Q_1(s, a) + \eta \left[r + \gamma Q_2(s', \operatorname{argmax}_{a'} Q_1(s', a')) - Q_1(s, a) \right] \quad (37.13)$$

and may repeat the same update but with the roles of Q_1 and Q_2 swapped. This technique is called **double Q-learning** [Has10]. Figure 37.4 shows the benefits of the algorithm over standard Q-learning in a toy problem.

37.2.6 Deep Q-network (DQN)

When function approximation is used, Q-learning may be hard to use in practice due to instability problems. Here, we will describe two important heuristics, popularized by the **deep Q-network** or **DQN** work [Mni+15], which was able to train agents to outperform humans at playing Atari games, using CNN-structured Q -networks.

The first technique, originally proposed in [Lin92], is to leverage an **experience replace** buffer, which stores the most recent (s, a, r, s') transition tuples. In contrast to standard Q-learning which updates the Q -function when a new transition occurs, the DQN agent also performs additional updates using transitions sampled from the buffer. This modification has two advantages. First, it improves data efficiency as every transition can be used multiple times. Second, it improves stability in training, by reducing the correlation of the data samples that the network is trained on.

The second idea to improve stability is to regress the Q -network to a “frozen” **target network** computed at an earlier iteration, rather than trying to chase a constantly moving target. Specifically, we maintain an extra, frozen copy of the Q -network, $Q_{\mathbf{w}^-}$, of the same structure as $Q_{\mathbf{w}}$. This new Q -network is to compute bootstrapping targets for training $Q_{\mathbf{w}}$, in which the loss function is

$$\mathcal{L}^{\text{DQN}}(\mathbf{w}) = \mathbb{E}_{(s, a, r, s') \sim U(\mathcal{D})} \left[(r + \gamma \max_{a'} Q_{\mathbf{w}^-}(s', a') - Q_{\mathbf{w}}(s, a))^2 \right] \quad (37.14)$$

where $U(\mathcal{D})$ is a uniform distribution over the replay buffer \mathcal{D} . We then periodically set $\mathbf{w}^- \leftarrow \mathbf{w}$, usually after a few episodes. This approach is an instance of **fitted value iteration** [SB18].

Various improvements to DQN have been proposed. One is **double DQN** [HGS16], which uses the double learning technique (Section 37.2.5.2) to remove the maximization bias. The second is to replace the uniform distribution in Equation (37.14) with one that favors more important transition tuples, resulting in the use of **prioritized experience replay** [Sch+16a]. For example, we can sample transitions from \mathcal{D} with probability $p(s, a, r, s') \propto (|\delta| + \varepsilon)^\eta$, where δ is the corresponding TD error (under the current Q -function), $\varepsilon > 0$ a hyperparameter to ensure every experience is chosen with nonzero probability, and $\eta \geq 0$ controls the “inverse temperature” of the distribution (so $\eta = 0$ corresponds to uniform sampling). The third is to learn a value function $V_{\mathbf{w}}$ and an advantage function $A_{\mathbf{w}}$, with shared parameter \mathbf{w} , instead of learning $Q_{\mathbf{w}}$. The resulting **dueling DQN** [Wan+16] is shown to be more sample efficient, especially when there are many actions with similar Q -values.

¹ The **rainbow** method [Hes+18] combines all three improvements, as well as others, including
² multi-step returns (Section 37.2.3), distributional RL [BDM17] (which predicts the distribution
³ of returns, not just the expected return), and noisy nets [For+18b] (which adds random noise to
⁴ the network weights to encourage exploration). It produces state-of-the-art results on the Atari
⁵ benchmark.

⁶

⁷ 37.3 Policy-based RL

⁸

⁹ In the previous section, we considered methods that estimate the action-value function, $Q(s, a)$, from
¹⁰ which we derive a policy, which may be greedy or softmax. However, these methods have three main
¹¹ disadvantages: (1) they can be difficult to apply to continuous action spaces; (2) they may diverge if
¹² function approximation is used; and (3) the training of Q , often based on TD-style updates, is not
¹³ directly related to the expected return garnered by the learned policy.

¹⁴ In this section, we discuss **policy search** methods, which directly optimize the parameters of the
¹⁵ policy so as to maximize its expected return. However, we will see that these methods often benefit
¹⁶ from estimating a value or advantage function to reduce the variance in the policy search process.

¹⁷

¹⁸ 37.3.1 The policy gradient theorem

¹⁹

²⁰ We start by defining the objective function for policy learning, and then derive its gradient. We
²¹ consider the episodic case. A similar result can be derived for the continuing case with the average
²² reward criterion [SB18, Sec 13.6].

²³ We define the objective to be the expected return of a policy, which we aim to maximize:

$$\begin{aligned} \text{Equation (37.15)} \\ J(\pi) &\triangleq \mathbb{E}_{p_0, \pi}[G_0] = \mathbb{E}_{p_0(s_0)}[V_\pi(s_0)] = \mathbb{E}_{p_0(s_0)\pi(a_0|s_0)}[Q_\pi(s_0, a_0)] \end{aligned}$$

²⁴ We consider policies π_θ parameterized by θ , and compute the gradient of Equation (37.15) wrt θ :

$$\begin{aligned} \text{Equation (37.16)} \\ \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{p_0(s_0)} \left[\nabla_\theta \left(\sum_{a_0} \pi_\theta(a_0|s_0) Q_\pi(s_0, a_0) \right) \right] \end{aligned}$$

$$\begin{aligned} \text{Equation (37.17)} \\ &= \mathbb{E}_{p_0(s_0)} \left[\sum_{a_0} \nabla \pi_\theta(a_0|s_0) Q_\pi(s_0, a_0) \right] + \mathbb{E}_{p_0(s_0)\pi(a_0|s_0)}[\nabla_\theta Q_\pi(s_0, a_0)] \end{aligned}$$

²⁵ Now we calculate the term $\nabla_\theta Q_\pi(s_0, a_0)$:

$$\text{Equation (37.18)} \\ \nabla_\theta Q_\pi(s_0, a_0) = \nabla_\theta [R(s_0, a_0) + \gamma \mathbb{E}_{p_T(s_1|s_0, a_0)}[V_\pi(s_1)]] = \gamma \nabla_\theta \mathbb{E}_{p_T(s_1|s_0, a_0)}[V_\pi(s_1)]$$

²⁶ The right-hand side above is in a form similar to $\nabla_\theta J(\pi_\theta)$. Repeating the same steps as before gives

$$\begin{aligned} \text{Equation (37.19)} \\ \nabla_\theta J(\pi_\theta) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{p_t(s)} \left[\sum_a \nabla_\theta \pi_\theta(a|s) Q_\pi(s, a) \right] \end{aligned}$$

$$\begin{aligned} \text{Equation (37.20)} \\ &= \frac{1}{1-\gamma} \mathbb{E}_{p_\pi^\infty(s)} \left[\sum_a \nabla_\theta \pi_\theta(a|s) Q_\pi(s, a) \right] \end{aligned}$$

$$\begin{aligned} \text{Equation (37.21)} \\ &= \frac{1}{1-\gamma} \mathbb{E}_{p_\pi^\infty(s)\pi(a|s)} [\nabla_\theta \log \pi_\theta(a|s) Q_\pi(s, a)] \end{aligned}$$

⁴⁷

where $p_t(s)$ is the probability of visiting s in time t if we start with $s_0 \sim p_0$ and follow π_θ , and $p_\pi^\infty(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t p_t(s)$ is the normalized discounted state visitation distribution. Equation (37.21) is known as the **policy gradient theorem** [Sut+99].

In practice, estimating the policy gradient using Equation (37.21) can have a high variance. A baseline $b(s)$ can be used for variance reduction (Section 6.6.3.1):

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1 - \gamma} \mathbb{E}_{p_\pi^\infty(s)\pi(a|s)} [\nabla_\theta \log \pi_\theta(a|s)(Q_\pi(s, a) - b(s))] \quad (37.22)$$

A common choice for the baseline is $b(s) = V_\pi(s)$. We will discuss how to estimate it below.

37.3.2 REINFORCE

One way to apply the policy gradient theorem to optimize a policy is to use stochastic gradient ascent. Suppose $\tau = (s_0, a_0, r_0, s_1, \dots, s_T)$ is a trajectory with $s_0 \sim p_0$ and π_θ . Then,

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1 - \gamma} \mathbb{E}_{p_\pi^\infty(s)\pi(a|s)} [\nabla_\theta \log \pi_\theta(a|s) Q_\pi(s, a)] \quad (37.23)$$

$$\approx \sum_{t=0}^{T-1} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \quad (37.24)$$

where the return G_t is defined in Equation (36.52), and the factor γ^t is due to the definition of p_π^∞ where the state at time t is discounted.

We can use a baseline in the gradient estimate to get the following update rule:

$$\theta \leftarrow \theta + \eta \sum_{t=0}^{T-1} \gamma^t (G_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \quad (37.25)$$

This is called the **REINFORCE** algorithm [Wil92].³ The update equation can be interpreted as follows: we compute the sum of discounted future rewards induced by a trajectory, compared to a baseline, and if this is positive, we increase θ so as to make this trajectory more likely, otherwise we decrease θ . Thus, we reinforce good behaviors, and reduce the chances of generating bad ones.

We can use a constant (state-independent) baseline, or we can use a state-dependent baseline, $b(s_t)$ to further lower the variance. A natural choice is to use an estimated value function, $V_w(s)$, which can be learned, e.g., with MC. Algorithm 36 gives the pseudo code where stochastic gradient updates are used with separate learning rates.

37.3.3 Actor-critic methods

An **actor-critic** method [BSA83] uses the policy gradient method, but where the expected return is estimated using temporal difference learning of a value function instead of MC rollouts. The term “actor” refers to the policy, and the term “critic” refers to the value function. The use of bootstrapping

³ The term “REINFORCE” is an acronym for “REward Increment = nonnegative Factor x Offset Reinforcement x Characteristic Eligibility”. The phrase “Characteristic eligibility” refers to the $\nabla \log \pi_\theta(a_t|s_t)$ term; the phrase “offset reinforcement” refers to the $G_t - b(s_t)$ term; and the phrase “nonnegative factor” refers to the learning rate η of SGD.

1
2 **Algorithm 36:** REINFORCE with value function baseline
3 1 Initialize policy parameters θ , baseline parameters w
4 2 **repeat**
5 3 Sample an episode $\tau = (s_0, a_0, r_0, s_1, \dots, s_T)$ using π_θ
6 4 Compute G_t for all $t \in \{0, 1, \dots, T-1\}$ using Equation (36.52)
7 5 **for** $t = 0, 1, \dots, T-1$ **do**
8 6 $\delta = G_t - V_w(s_t)$ // scalar error
9 7 $w \leftarrow w + \eta_w \delta \nabla_w V_w(s_t)$
10 8 $\theta \leftarrow \theta + \eta_\theta \gamma^t \delta \nabla_\theta \log \pi_\theta(a_t | s_t)$
11 9 **until** converged;
13
14

15 in TD updates allows more efficient learning of the value function compared to MC. In addition, it
16 allows us to develop a fully online, incremental algorithm, that does not need to wait until the end of
17 the trajectory before updating the parameters (as in Algorithm 36).

18 Concretely, consider the use of the one-step TD(0) method to estimate the return in the episodic
19 case, i.e., we replace G_t with $G_{t:t+1} = r_t + \gamma V_w(s_{t+1})$. If we use $V_w(s_t)$ as a baseline, the REINFORCE
20 update in Equation (37.25) becomes

21
22 $\theta \leftarrow \theta + \eta \sum_{t=0}^{T-1} \gamma^t (G_{t:t+1} - V_w(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$ (37.26)

24
25 $= \theta + \eta \sum_{t=0}^{T-1} \gamma^t (r_t + \gamma V_w(s_{t+1}) - V_w(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$ (37.27)

28 **37.3.3.1 A2C and A3C**

30 Note that $r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$ is a single sample approximation to the advantage function
31 $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. This method is therefore called **advantage actor critic** or **A2C**
32 (Algorithm 37). If we run the actors in parallel and asynchronously update their shared parameters,
33 the method is called **asynchronous advantage actor critic** or **A3C** [Mni+16].

34

35 **37.3.3.2 Eligibility traces**

36 In A2C, we use a single step rollout, and then use the value function in order to approximate the
37 expected return for the trajectory. More generally, we can use the n -step estimate

39 $G_{t:t+n} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_w(s_{t+n})$ (37.28)

41 and obtain an n -step advantage estimate as follows:

42 $A_\pi^{(n)}(s_t, a_t) = G_{t:t+n} - V_w(s_t)$ (37.29)

45 The n steps of actual rewards are an unbiased sample, but have high variance. By contrast,
46 $V_w(s_{t+n+1})$ has lower variance, but is biased. By changing n , we can control the bias-variance
47

Algorithm 37: Advantage actor critic (A2C) algorithm

```

1 Initialize actor parameters  $\theta$ , critic parameters  $w$ 
2 repeat
3   Sample starting state  $s_0$  of a new episode
4   for  $t = 0, 1, 2, \dots$  do
5     Sample action  $a_t \sim \pi_\theta(\cdot|s_t)$ 
6     Observe next state  $s_{t+1}$  and reward  $r_t$ 
7      $\delta = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$ 
8      $w \leftarrow w + \eta_w \delta \nabla_w V_w(s_t)$ 
9      $\theta \leftarrow \theta + \eta_\theta \gamma^t \delta \nabla_\theta \log \pi_\theta(a_t|s_t)$ 
10  until converged;

```

tradeoff. Instead of using a single value of n , we can take an weighted average, with weight proportional to λ^n for $A_\pi^{(n)}(s_t, a_t)$, as in $\text{TD}(\lambda)$. The average can be shown to be equivalent to

$$A_\pi^{(\lambda)}(s_t, a_t) \triangleq \sum_{\ell=0}^{\infty} (\gamma \lambda)^\ell \delta_{t+\ell} \quad (37.30)$$

where $\delta_t = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$ is the TD error at time t . Here, $\lambda \in [0, 1]$ is a parameter that controls the bias-variance tradeoff: larger values decrease the bias but increase the variance, as in $\text{TD}(\lambda)$. We can implement Equation (37.30) efficiently using eligibility traces, as shown in Algorithm 38, as an example of **generalized advantage estimation (GAE)** [Sch+16b]. See [SB18, Ch.12] for further details.

Algorithm 38: Actor critic with eligibility traces

```

1 Initialize actor parameters  $\theta$ , critic parameters  $w$ 
2 repeat
3   Initialize eligibility trace vectors:  $z_\theta \leftarrow \mathbf{0}$ ,  $z_w \leftarrow \mathbf{0}$ 
4   Sample starting state  $s_0$  of a new episode
5   for  $t = 0, 1, 2, \dots$  do
6     Sample action  $a_t \sim \pi_\theta(\cdot|s_t)$ 
7     Observe state  $s_{t+1}$  and reward  $r_t$ 
8     Compute the TD error:  $\delta = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$ 
9      $z_w \leftarrow \gamma \lambda_w z_w + \nabla_w V_w(s)$ 
10     $z_\theta \leftarrow \gamma \lambda_\theta z_\theta + \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)$ 
11     $w \leftarrow w + \eta_w \delta z_w$ 
12     $\theta \leftarrow \theta + \eta_\theta \delta z_\theta$ 
13  until converged;

```

¹
² **37.3.4 Bound optimization methods**

³ In policy gradient methods, the objective $J(\theta)$ does not necessarily increase monotonically, but
⁴ rather can collapse especially if the learning rate is not small enough. We now describe methods that
⁵ guarantee monotonic improvement, similar to bound optimization algorithms (Section 6.7).

⁶ We start with a useful fact that relate the policy values of two arbitrary policies [KL02]:
⁷

⁸
$$J(\pi') - J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi'}^\infty(s)} [\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)]] \quad (37.31)$$

⁹

¹⁰ where π can be interpreted as the current policy during policy optimization, and π' a candidate new
¹¹ policy (such as the greedy policy wrt Q_π). As in the policy improvement theorem (Section 36.6.2),
¹² if $\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)] \geq 0$ for all s , then $J(\pi') \geq J(\pi)$. However, we cannot ensure this condition to
¹³ hold when function approximation is used, as such a uniformly improving policy π' may not be
¹⁴ representable by our parametric family, $\{\pi_\theta\}_{\theta \in \Theta}$. Therefore, nonnegativity of Equation (37.31) is
¹⁵ not easy to ensure, when we do not have a direct way to sample states from $p_{\pi'}^\infty$.

¹⁶ One way to ensure monotonic improvement of J is to improve the policy conservatively. Define
¹⁷ $\pi_\theta = \theta\pi' + (1-\theta)\pi$ for $\theta \in [0, 1]$. It follows from the policy gradient theorem (Equation (37.21), with
¹⁸ $\theta = [\theta]$) that $J(\pi_\theta) - J(\pi) = \theta L(\pi') + O(\theta^2)$, where

¹⁹
$$L(\pi') \triangleq \frac{1}{1-\gamma} \mathbb{E}_{p_\pi^\infty(s)} [\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)]] = \frac{1}{1-\gamma} \mathbb{E}_{p_\pi^\infty(s)\pi(a|s)} \left[\frac{\pi'(a|s)}{\pi(a|s)} A_\pi(s, a) \right] \quad (37.32)$$

²⁰

²¹ In the above, we have switched the state distribution from $p_{\pi'}^\infty$ in Equation (37.31) to p_π^∞ , while at
²² the same time introducing a higher order residual term of $O(\theta^2)$. The linear term, $\theta L(\pi')$, can be
²³ estimated and optimized based on episodes sampled by π . The higher order term can be bounded in
²⁴ various ways, resulting in different lower bounds of $J(\pi_\theta) - J(\pi)$. We can then optimize θ to make
²⁵ sure this lower bound is positive, which would imply $J(\pi_\theta) - J(\pi) > 0$. In **conservative policy**
²⁶ **iteration** [KL02], the following (slightly simplified) lower bound is used

²⁷
$$J^{\text{CPI}}(\pi_\theta) \triangleq J(\pi) + \theta L(\pi') - \frac{2\varepsilon\gamma}{(1-\gamma)^2} \theta^2 \quad (37.33)$$

²⁸

²⁹ where $\varepsilon = \max_s |\mathbb{E}_{\pi'(a|s)} [A_\pi(s, a)]|$.

³⁰ This idea can be generalized to policies beyond those in the form of π_θ , where the condition
³¹ of a small enough θ is replaced by a small enough divergence between π' and π . In **safe policy**
³² **iteration** [Pir+13], the divergence is the maximum total variation, while in **trust region policy**
³³ **optimization (TRPO)** [Sch+15b], the divergence is the maximum KL-divergence. In the latter
³⁴ case, π' may be found by optimizing the following lower bound

³⁵
$$J^{\text{TRPO}}(\pi') \triangleq J(\pi) + L(\pi') - \frac{\varepsilon\gamma}{(1-\gamma)^2} \max_s D_{\text{KL}}(\pi(s) \| \pi'(s)) \quad (37.34)$$

³⁶

³⁷ where $\varepsilon = \max_{s,a} |A_\pi(s, a)|$.

³⁸ In practice, the above update rule can be overly conservative, and approximations are used.
³⁹ [Sch+15b] propose a version that implements two ideas: one is to replace the point-wise maximum
⁴⁰ KL-divergence by some average KL-divergence (usually averaged over p_π^∞); the second is to maximize
⁴¹ the first two terms in Equation (37.34), with π' lying in a KL-ball centered at π . That is, we solve

⁴²
$$\underset{\pi'}{\operatorname{argmax}} L(\pi') \text{ s.t. } \mathbb{E}_{p_\pi^\infty(s)} [D_{\text{KL}}(\pi(s) \| \pi'(s))] \leq \delta \quad (37.35)$$

⁴³

⁴⁴

for some threshold $\delta > 0$.

In Section 6.4.2.1, we show that the trust region method, using a KL penalty at each step, is equivalent to natural gradient descent (see e.g., [Kak02; PS08b]). This is important, because a step of size η in parameter space does not always correspond to a step of size η in the policy space:

$$d_{\theta}(\theta_1, \theta_2) = d_{\theta}(\theta_2, \theta_3) \not\Rightarrow d_{\pi}(\pi_{\theta_1}, \pi_{\theta_2}) = d_{\pi}(\pi_{\theta_2}, \pi_{\theta_3}) \quad (37.36)$$

where $d_{\theta}(\theta_1, \theta_2) = \|\theta_1 - \theta_2\|$ is the Euclidean distance, and $d_{\pi}(\pi_1, \pi_2) = D_{\text{KL}}(\pi_1 \| \pi_2)$ the KL distance. In other words, the effect on the policy of any given change to the parameters depends on where we are in parameter space. This is taken into account by the natural gradient method, resulting in faster and more robust optimization. The natural policy gradient can be approximated using the KFAC method (Section 6.4.4), as done in [Wu+17].

Other than TRPO, another approach inspired by Equation (37.34) is to use the KL-divergence as a penalty term, replacing the factor $2\varepsilon\gamma/(1-\gamma)^2$ by a tuning parameter. However, it often works better, and is simpler, by using the following clipped objective, which results in the **proximal policy optimization** or **PPO** method [Sch+17]:

$$J^{\text{PPO}}(\pi') \triangleq \frac{1}{1-\gamma} \mathbb{E}_{p_{\pi}^{\infty}(s)\pi(a|s)} \left[\kappa_{\epsilon} \left(\frac{\pi'(a|s)}{\pi(a|s)} \right) A_{\pi}(s, a) \right] \quad (37.37)$$

where $\kappa_{\epsilon}(x) \triangleq \text{clip}(x, 1-\epsilon, 1+\epsilon)$ ensures $|\kappa(x) - 1| \leq \epsilon$. This method can be modified to ensure monotonic improvement as discussed in [WHT19], making it a true bound optimization method.

37.3.5 Deterministic policy gradient methods

In this section, we consider the case of a deterministic policy, that predicts a unique action for each state, so $a_t = \mu_{\theta}(s_t)$, rather than $a_t \sim \pi_{\theta}(s_t)$. We assume the states and actions are continuous, and define the objective as

$$J(\mu_{\theta}) \triangleq \frac{1}{1-\gamma} \mathbb{E}_{p_{\mu}^{\infty}(s)} [R(s, \mu_{\theta}(s))] \quad (37.38)$$

The **deterministic policy gradient theorem** [Sil+14] provides a way to compute the gradient:

$$\nabla_{\theta} J(\mu_{\theta}) = \frac{1}{1-\gamma} \mathbb{E}_{p_{\mu}^{\infty}(s)} [\nabla_{\theta} Q_{\mu}(s, \mu_{\theta}(s))] \quad (37.39)$$

$$= \frac{1}{1-\gamma} \mathbb{E}_{p_{\mu}^{\infty}(s)} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q_{\mu}(s, a)|_{a=\mu(s)}] \quad (37.40)$$

where $\nabla_{\theta} \mu_{\theta}(s)$ is the $M \times N$ Jacobian matrix, and M and N are the dimensions of \mathcal{A} and θ , respectively. For stochastic policies of the form $\pi_{\theta}(a|s) = \mu_{\theta}(s) + \text{noise}$, the standard policy gradient theorem reduces to the above form as the noise level goes to zero.

Note that the gradient estimate in Equation (37.40) integrates over the states but not over the actions, which helps reduce the variance in gradient estimation from sampled trajectories. However, since the deterministic policy does not do any exploration, we need to use an off-policy method, that collects data from a stochastic behavior policy β , whose stationary state distribution is p_{β}^{∞} . The original objective, $J(\mu_{\theta})$, is approximated by the following:

$$J_b(\mu_{\theta}) \triangleq \mathbb{E}_{p_{\beta}^{\infty}(s)} [V_{\mu}(s)] = \mathbb{E}_{p_{\beta}^{\infty}(s)} [Q_{\mu}(s, \mu_{\theta}(s))] \quad (37.41)$$

¹ with the off-policy deterministic policy gradient approximated by (see also Section 37.5.1.2)
²

$$\nabla_{\theta} J_b(\mu_{\theta}) \approx \mathbb{E}_{p_{\beta}^{\infty}(s)} [\nabla_{\theta} [Q_{\mu}(s, \mu_{\theta}(s))]] = \mathbb{E}_{p_{\beta}^{\infty}(s)} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q_{\mu}(s, a)|_{a=\mu(s)} ds] \quad (37.42)$$

³ where we have dropped a term that depends on $\nabla_{\theta} Q_{\mu}(s, a)$ and is hard to estimate [Sil+14].
⁴ To apply Equation (37.42), we may learn $Q_w \approx Q_{\mu}$ with TD, giving rise to the following updates:
⁵

$$\delta = r_t + \gamma Q_w(s_{t+1}, \mu_{\theta}(s_{t+1})) - Q_w(s_t, a_t) \quad (37.43)$$

$$w_{t+1} \leftarrow w_t + \eta_w \delta \nabla_w Q_w(s_t, a_t) \quad (37.44)$$

$$\theta_{t+1} \leftarrow \theta_t + \eta_{\theta} \nabla_{\theta} \mu_{\theta}(s_t) \nabla_a Q_w(s_t, a)|_{a=\mu(s_t)} \quad (37.45)$$

⁶ This avoids importance sampling in the actor update because of the deterministic policy gradient,
⁷ and avoids importance sampling in the critic update because of the use of Q-learning.
⁸

⁹ If Q_w is linear in w , and uses features of the form $\phi(s, a) = a^T \nabla_{\theta} \mu_{\theta}(s)$, where a is the vector
¹⁰ representation of a , then we say the function approximator for the critic is **compatible** with the
¹¹ actor; in this case, one can show that the above approximation does not bias the overall gradient.
¹² The method has been extended in various ways. The **DDPG** algorithm of [Lil+16], which stands
¹³ for “deep deterministic policy gradient”, uses the DQN method (Section 37.2.6) to update Q that
¹⁴ is represented by deep neural networks. The **TD3** algorithm [FHM18], which stands for “twin
¹⁵ delayed DDPG”, extends DDPG by using double DQN (Section 37.2.5.2) and other heuristics to
¹⁶ further improve performance. Finally, the **D4PG** algorithm [BM+18], which stands for “distributed
¹⁷ distributional DDPG”, extends DDPG to handle distributed training, and to handle **distributional**
¹⁸ **RL** (i.e., working with distributions of rewards instead of expected rewards [BDM17]).
¹⁹

²⁰ 37.3.6 Gradient-free methods

²¹ The policy gradient estimator computes a “zeroth order” gradient, which essentially evaluates the
²² function with randomly sampled trajectories. Sometimes it can be more efficient to use a derivative-
²³ free optimizer (Section 6.12), that does not even attempt to estimate the gradient. For example,
²⁴ [MGR18] obtain good results by training linear policies with random search, and [Sal+17b] show
²⁵ how to use evolutionary strategies to optimize the policy of a robotic controller.
²⁶

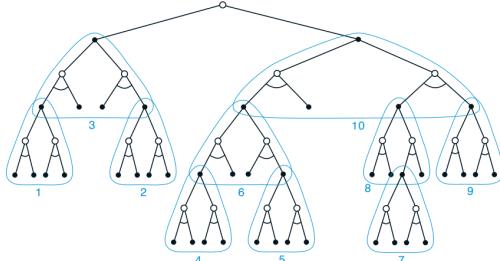
²⁷ 37.4 Model-based RL

²⁸ Model-free approaches to RL typically need a lot of interactions with the environment to achieve
²⁹ good performance. For example, state of the art methods for the Atari benchmark, such as rainbow
³⁰ (Section 37.2.6), use millions of frames, equivalent to many days of playing at the standard frame rate.
³¹ By contrast, humans can achieve the same performance in minutes [Tsi+17]. Similarly, OpenAI’s
³² robot hand controller [And+20] learns to manipulate a cube using 100 years of simulated data.
³³

³⁴ One promising approach to greater sample efficiency is **model-based RL (MBRL)**. In this
³⁵ approach, we first learn the transition model and reward function, $p_T(s'|s, a)$ and $R(s, a)$, then
³⁶ use them to compute a near-optimal policy. This approach can significantly reduce the amount of
³⁷ real-world data that the agent needs to collect, since it can “try things out” in its imagination (i.e.,
³⁸ the models), rather than having to try them out empirically.
³⁹

⁴⁰ There are several ways we can use a model, and many different kinds of model we can create. Some
⁴¹ of the algorithms mentioned earlier, such as MBIE and UCLR2 for provably efficient exploration
⁴²

1
2
3
4
5
6
7
8
9
10



11
12
13

Figure 37.5: Illustration of heuristic search. In this figure, the subtrees are ordered according to a depth-first search procedure. From Figure 8.9 of [SB18]. Used with kind permission of Richard Sutton.

14
15

(Section 37.1.5.3), are examples of model-based methods. MBRL also provides a natural connection between RL and planning (Section 36.6) [Sut90]. We discuss some examples in the sections below, and refer to [MBJ20; PKP21; MH20] for more detailed reviews.

19
20

37.4.1 Model predictive control (MPC)

21
22
23
24
25
26
27
28
29
30
31

So far in this chapter, we have focused on trying to learn an optimal policy $\pi_*(s)$, which can then be used at run time to quickly pick the best action for any given state s . However, we can also avoid performing all this work in advance, and wait until we know what state we are in, call it s_t , and then use a model to predict future states and rewards that might follow for each possible sequence of future actions we might pursue. We then take the action that looks most promising, and repeat the process at the next step. More precisely, we compute

$$\mathbf{a}_{t:t+H-1}^* = \underset{\mathbf{a}_{t:t+H-1}}{\operatorname{argmax}} \mathbb{E} \left[\sum_{h=0}^{H-1} R(s_{t+h}, a_{t+h}) + \hat{V}(s_{t+H}) \right] \quad (37.46)$$

32
33
34
35
36
37
38

where the expectation is over state sequences that might result from executing $\mathbf{a}_{t:t+H-1}$ from state s_t . Here, H is called the **planning horizon**, and $\hat{V}(s_{t+H})$ is an estimate of the reward-to-go at the end of this H -step look-ahead process. This is known as **receding horizon control** or **model predictive control (MPC)** [MM90; CA13]. We discuss some special cases of this below.

37.4.1.1 Heuristic search

39
40
41
42
43
44
45
46
47

If the state and action spaces are finite, we can solve Equation (37.46) exactly, although the time complexity will typically be exponential in H . However, in many situations, we can prune off unpromising trajectories, thus making the approach feasible in large scale problems.

In particular, consider a discrete, deterministic MDP where reward maximization corresponds to finding a shortest path to a goal state. We can expand the successors of the current state according to all possible actions, trying to find the goal state. Since the search tree grows exponentially with depth, we can use a **heuristic function** to prioritize which nodes to expand; this is called **best-first search**, as illustrated in Figure 37.5.

¹ If the heuristic function is an optimistic lower bound on the true distance to the goal, it is called
² **admissible**; If we aim to maximize total rewards, admissibility means the heuristic function is an
³ upper bound of the true value function. Admissibility ensures we will never incorrectly prune off
⁴ parts of the search space. In this case, the resulting algorithm is known as **A^* search**, and is optimal.
⁵ For more details on classical AI **heuristic search** methods, see [Pea84; RN19].
⁶

⁷

⁸ 37.4.1.2 Monte-Carlo tree search (MCTS)

⁹ **Monte-Carlo tree search** or **MCTS** is similar to heuristic search, but learns a value function for
¹⁰ each encountered state, rather than relying on a manually designed heuristic. It is inspired by UCB
¹¹ for bandits (Section 36.4.5), but applies to general sequential decision making problems including
¹² MDPs [KS06] where the UCB is used for efficient exploration of the state space.

¹³ The MCTS method forms the basis of the famous **AlphaGo** and **AlphaZero** programs [Sil+16;
¹⁴ Sil+18], which can play expert-level Go, chess and shogi (Japanese chess). The action-value functions
¹⁵ for the intermediate nodes in the search tree are represented by deep neural networks, and updated
¹⁶ by RL methods that we discuss in Section 37.2. MCTS can also be applied to many other kinds of
¹⁷ sequential decision problems, such as experiment design for sequentially creating molecules [SPW18].
¹⁸ For more details on MCTS, see e.g., [Mun14].
¹⁹

²⁰

²¹ 37.4.1.3 Trajectory optimization for continuous actions

²² For continuous actions, we cannot enumerate all possible branches in the search tree. Instead,
²³ Equation (37.46) can be viewed as a nonlinear program, where $a_{t:t+H-1}$ are the real-valued variables
²⁴ to be optimized. If the system dynamics are linear and the reward function corresponds to negative
²⁵ quadratic cost, the optimal action sequence can be solved mathematically, as in the **linear-quadratic-**
²⁶ **Gaussian (LQG)** controller (see e.g., [AM89]). However, this problem is hard in general and often
²⁷ solved by numerical methods such as **shooting** and **collocation** [Die+07; Rao10; Kal+11]. Many
²⁸ of them work in an iterative fashion, starting with an initial action sequence followed by a step to
²⁹ improve it. This process repeats until convergence of the cost.

³⁰ An example is **differential dynamic programming (DDP)** [JM70; TL05]. In each iteration,
³¹ DDP starts with a reference trajectory, and linearizes the system dynamics around states on the
³² trajectory to form a locally quadratic approximation of the reward function. This system can be
³³ solved using LQG, whose optimal solution results in a new trajectory. The algorithm then moves to
³⁴ the next iteration, with the new trajectory as the reference trajectory.

³⁵ Other alternatives are possible, including black-box (gradient-free) optimization methods like the
³⁶ cross entropy method. (See the supplementary material for details.)
³⁷

³⁸ 37.4.2 Combining model-based and model-free

³⁹

⁴⁰ In Section 37.4.1, we discussed MPC, which uses the model to decide which action to take at each
⁴¹ step. However, this can be slow, and can suffer from problems when the model is inaccurate. An
⁴² alternative is to use the learned model to help reduce the sample complexity of policy learning.

⁴³ There are many ways to do this. One approach is to generate rollouts from the model, and then
⁴⁴ train a policy or Q -function on the “hallucinated” data. This is the basis of the famous **dyna** method
⁴⁵ [Sut90]. In [Jan+19], they propose a similar method, but generate short rollouts from previously
⁴⁶ visited real states; this ensures the model only has to extrapolate locally.

⁴⁷

¹ In [Web+17], they train a model to predict future states and rewards, but then use the hidden
² states of this model as additional context for a policy-based learning method. This can help overcome
³ partial observability. They call their method **imagination-augmented agents**. A related method
⁴ appears in [Jad+17], who propose to train a model to jointly predict future rewards and other
⁵ auxiliary signals, such as future states. This can help in situations when rewards are sparse or absent.
⁶

⁸ 37.4.3 MBRL using Gaussian processes

¹⁰ This section gives some examples of dynamics models that have been learned for low-dimensional
¹¹ continuous control problems. Such problems frequently arise in robotics. Since the dynamics are
¹² often nonlinear, it is useful to use a flexible and sample-efficient model family, such as Gaussian
¹³ processes (Section 18.1). We will use notation like s and a for states and actions to emphasize they
¹⁴ are vectors.

¹⁶ 37.4.3.1 PILCO

¹⁸ We first describe the **PILCO** method [DR11; DFR15], which stands for “probabilistic inference for
¹⁹ learning control”. It is extremely data efficient for continuous control problems, enabling learning
²⁰ from scratch on real physical robots in a matter of minutes.

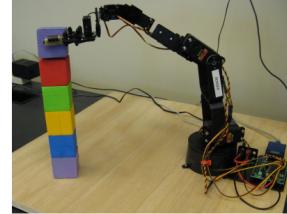
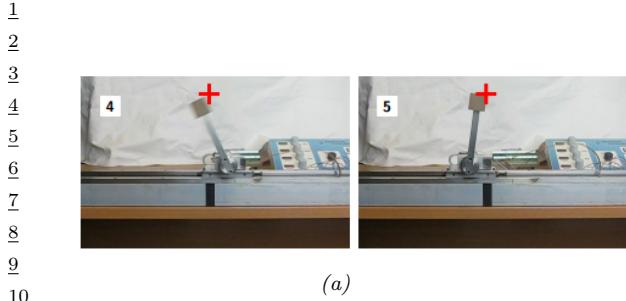
²¹ PILCO assumes the world model has the form $s_{t+1} = f(s_t, a_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \Sigma)$, and f
²² is an unknown, continuous function.⁴ The basic idea is to learn a Gaussian process (Section 18.1))
²³ approximation of f based on some initial random trajectories, and then to use this model to generate
²⁴ “fantasy” rollout trajectories of length T , that can be used to evaluate the expected cost of the current
²⁵ policy, $J(\pi_\theta) = \sum_{t=1}^T \mathbb{E}_{a_t \sim \pi} [c(s_t)]$, where $s_0 \sim p_0$. This function and its gradients wrt θ can be
²⁶ computed deterministically, if a Gaussian assumption about the state distribution at each step is
²⁷ made, because the Gaussian belief state can be propagated deterministically through the GP model.
²⁸ Therefore, we can use deterministic batch optimization methods, such as Levenberg-Marquardt, to
²⁹ optimize the policy parameters θ , instead of applying SGD to sampled trajectories.

³⁰ Due to its data efficiency, it is possible to apply PILCO to real robots. Figure 37.6a shows the
³¹ results of applying it to solve a **cart-pole swing-up** task, where the goal is to make the inverted
³² pendulum swing up by applying a horizontal force to move the cart back and forth. The state of the
³³ system $s \in \mathbb{R}^4$ consists of the position x of the cart (with $x = 0$ being the center of the track), the
³⁴ velocity \dot{x} , the angle θ of the pendulum (measured from hanging downward), and the angular velocity
³⁵ $\dot{\theta}$. The control signal $a \in \mathbb{R}$ is the force applied to the cart. The target state is $s_* = (0, \star, \pi, \star)$,
³⁶ corresponding to the cart being in the middle and the pendulum being vertical, with velocities
³⁷ unspecified. The authors used an RBF controller with 50 basis functions, amounting to a total of
³⁸ 305 policy parameters. The controller was successfully trained using just 7 real world trials.⁵

³⁹ Figure 37.6b shows the results of applying PILCO to solve a **block stacking** task using a low-
⁴⁰ quality robot arm with 6 degrees of freedom. A separate controller was trained for each block. The
⁴¹ state space $s \in \mathbb{R}^3$ is the 3d location of the center of the block in the arm’s gripper (derived from
⁴² an RGBD sensor), and the control $a \in \mathbb{R}^4$ corresponds to the pulse widths of four servo motors. A
⁴³ linear policy was successfully trained using as few as 10 real world trials.

⁴⁵ 4. An alternative, which often works better, is to use f to model the residual, so that $s_{t+1} = s_t + f(s_t, a_t) + \epsilon_t$.

⁴⁶ 5. 2 random initial trials, each 5 seconds, and then 5 policy-generated trials, each 2.5 seconds, totalling 17.5 seconds.



11 *Figure 37.6: (a) A cart-pole system being controlled by a policy learned by PILCO using just 17.5 seconds*
 12 *of real-world interaction. The goal state is marked by the red cross. The initial state is where the cart is*
 13 *stationary on the right edge of the workspace, and the pendulum is horizontal. For a video of the system*
 14 *learning, see <https://bit.ly/35fpLmR>. (b) A low-quality robot arm being controlled by a block-stacking*
 15 *policy learned by PILCO using just 230 seconds of real-world interaction. From Figures 11, 12 from [DFR15].*
 16 *Used with kind permission of Marc Deisenroth.*

17
18
19

20 37.4.3.2 GP-MPC

21 [KD18a] have proposed an extension to PILCO that they call **GP-MPC**, since it combines a GP
 22 dynamics model with model predictive control (Section 37.4.1). In particular, they use an open-loop
 23 control policy to propose a sequence of actions, $\mathbf{a}_{t:t+H-1}$, as opposed to sampling them from a policy.
 24 They compute a Gaussian approximation to the future state trajectory, $p(\mathbf{s}_{t+1:t+H}|\mathbf{a}_{t:t+H-1}, \mathbf{s}_t)$, by
 25 moment matching, and use this to deterministically compute the expected reward and its gradient
 26 wrt $\mathbf{a}_{t:t+H-1}$ (as opposed to the policy parameters θ). Using this, they can solve Equation (37.46)
 27 to find $\mathbf{a}_{t:t+H-1}^*$; finally, they execute the first step of this plan, a_t^* , and repeat the whole process.
 28 The advantage of GP-MPC over policy-based PILCO is that it can handle constraints more easily,
 29 and it can be more data efficient, since it continually updates the GP model after every step (instead
 30 of at the end of an trajectory).

31
32

33 37.4.4 MBRL using DNNs

34

35 Gaussian processes do not scale well to large sample sizes and high dimensional data. Deep neural
 36 networks (DNNs) work much better in this regime. However, they do not naturally model uncertainty,
 37 which can cause MPC methods to fail. We discuss various methods for representing uncertainty with
 38 DNNs in Section 17.1. Here, we mention a few approaches that have been used for MBRL.

39 The **deep PILCO** method uses DNNs together with Monte Carlo dropout (Section 17.4.5) to
 40 represent uncertainty [GMAR16]. [Chu+18] proposed **probabilistic ensembles with trajectory**
 41 **sampling** or **PETS**, which represents uncertainty using an ensemble of DNNs (Section 17.4.8).
 42 Many other approaches are possible, depending on the details of the problem being tackled.

43 Since these are all stochastic methods (as opposed to the GP methods above), they can suffer from
 44 a high variance in the predicted returns, which can make it difficult for the MPC controller to pick
 45 the best action. We can reduce variance with the **common random number** trick [KSN99], where
 46 all rollouts share the same random seed, so differences in $J(\pi_\theta)$ can be attributed to changes in θ

47

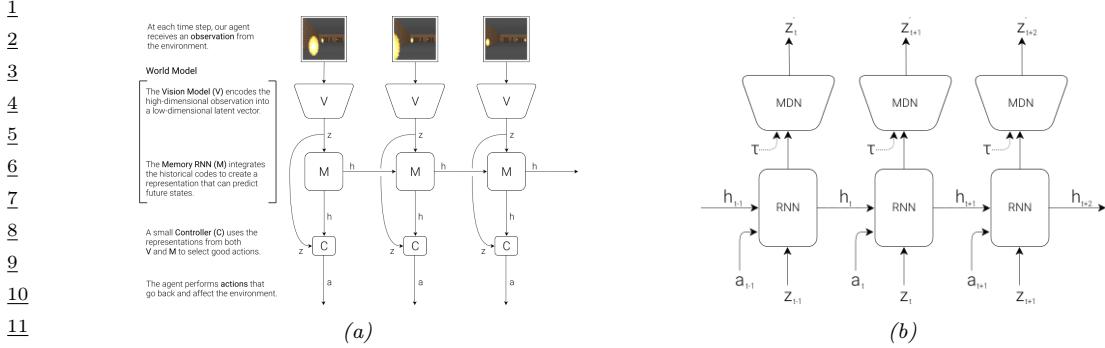


Figure 37.7: (a) Illustration of an agent interacting with the VizDoom environment. (The yellow blobs represent fireballs being thrown towards the agent by various enemies.) The agent has a world model, composed of a vision system V and a memory RNN M , and has a controller C . (b) Detailed representation of the memory model. Here h_t is the deterministic hidden state of the RNN at time t , which is used to predict the next latent of the VAE, z_{t+1} , using a mixture density network (MDN). Here τ is a temperature parameter used to increase the variance of the predictions, to prevent the controller from exploiting model inaccuracies. From Figures 4, 6 of [HS18]. Used with kind permission of David Ha.

but not other factors. This technique was used in **PEGASUS** [NJ00]⁶ and in [HMD18].

37.4.5 MBRL using latent-variable models

In this section, we describe some methods that learn latent variable models, rather than trying to predict dynamics directly in the observed space, which is hard to do when the states are images.

37.4.5.1 World models

The “world models” paper [HS18] showed how to learn a generative model of two simple video games (CarRacing and a VizDoom-like environment), such that the model can be used to train a policy entirely in simulation. The basic idea is shown in Figure 37.7. First, we collect some random experience, and use this to fit a VAE model (Section 22.2) to reduce the dimensionality of the images, $\mathbf{x}_t \in \mathbb{R}^{64 \times 64 \times 3}$, to a latent $\mathbf{z}_t \in \mathbb{R}^{64}$. Next, we train an RNN to predict $p(\mathbf{z}_{t+1} | \mathbf{z}_t, \mathbf{a}_t, \mathbf{h}_t)$, where \mathbf{h}_t is the deterministic RNN state, and \mathbf{a}_t is the continuous action vector (3-dimensional in both cases). The emission model for the RNN is a mixture density network, in order to model multi-modal futures. Finally, we train the controller using \mathbf{z}_t and \mathbf{h}_t as inputs; here \mathbf{z}_t is a compact representation of the current frame, and \mathbf{h}_t is a compact representation of the predicted distribution over \mathbf{z}_{t+1} .

The authors of [HS18] trained the controller using a derivative free optimizer called **CMA-ES** (covariance matrix adaptation evolutionary strategy, see supplementary). It can work better than policy gradient methods, as discussed in Section 37.3.6. However, it does not scale to high dimensions. To tackle this, the authors use a linear controller, which has only 867 parameters.⁷ By contrast,

⁶ PEGASUS stands for “Policy Evaluation-of-Goodness And Search Using Scenarios”, where the term “scenario” refers to one of the shared random samples.

⁷ The input is a 32-dimensional \mathbf{z}_t plus a 256-dimensional \mathbf{h}_t , and there are 3 outputs. So the number of parameters

¹ VAE has 4.3M parameters and MDN-RNN 422k. Fortunately, these two models can be trained in an
² unsupervised way from random rollouts, so sample efficiency is less critical than when training the
³ policy.
⁴

⁵ So far, we have described how to use the representation learned by the generative model as
⁶ informative features for the controller, but the controller is still learned by interacting with the
⁷ real world. Surprisingly, we can also train the controller entirely in “dream mode”, in which the
⁸ generated images from the VAE decoder at time t are fed as input to the VAE encoder at time $t + 1$,
⁹ and the MDN-RNN is trained to predict the next reward r_{t+1} as well as \mathbf{z}_{t+1} . Unfortunately, this
¹⁰ method does not always work, since the model (which is trained in an unsupervised way) may fail to
¹¹ capture task-relevant features (due to underfitting) and may memorize task-irrelevant features (due
¹² to overfitting). The controller can learn to exploit weaknesses in the model (similar to an adversarial
¹³ attack) and achieve high simulated reward, but such a controller may not work well when transferred
¹⁴ to the real world.

¹⁵ One approach to combat this is to artificially increase the variance of the MDN model (by using
¹⁶ a temperature parameter τ), in order to make the generated samples more stochastic. This forces
¹⁷ the controller to be robust to large variations; the controller will then treat the real world as just
¹⁸ another kind of noise. This is similar to the technique of domain randomization, which is sometimes
¹⁹ used for sim-to-real applications; see e.g., [MAZA18].

²⁰

²¹ 37.4.5.2 PlaNet and Dreamer

²²

²³ In [HS18], they first learn the world model on random rollouts, and then train a controller. On harder
²⁴ problems, it is necessary to iterate these two steps, so the model can be trained on data collected by
²⁵ the controller, in an iterative fashion.

²⁶ In this section, we describe one method of this kind, known as **PlaNet** [Haf+19]. PlaNet
²⁷ uses a POMDP model, where \mathbf{z}_t are the latent states, \mathbf{s}_t are the observations, \mathbf{a}_t are the ac-
²⁸ tions, and r_t are the rewards. It fits a recurrent state space model (Section 31.5.2) of the form
²⁹ $p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{a}_{t-1}) p(\mathbf{s}_t | \mathbf{z}_t) p(r_t | \mathbf{z}_t)$ using variational inference, where the posterior is approximated by
³⁰ $q(\mathbf{z}_t | \mathbf{s}_{1:t}, \mathbf{a}_{1:t-1})$. After fitting the model to some random trajectories, the system uses the inference
³¹ model to compute the current belief state, and then uses the cross entropy method to find an
³² action sequence for the next H steps to maximize expected reward, by optimizing in latent space.
³³ The system then executes \mathbf{a}_t^* , updates the model, and repeats the whole process. To encourage
³⁴ the dynamics model to capture long term trajectories, they use the “latent overshooting” training
³⁵ method described in Section 31.5.3. The PlaNet method outperforms model-free methods, such as
³⁶ A3C (Section 37.3.3.1) and D4PG (Section 37.3.5), on various image-based continuous control tasks,
³⁷ illustrated in Figure 37.8.

³⁸ Although PlaNet is sample efficient, it is not computationally efficient. For example, they use
³⁹ CEM with 1000 samples and 10 iterations to optimize trajectories with a horizon of length 12, which
⁴⁰ requires 120,000 evaluations of the transition dynamics to choose a single action. [AY19] improve
⁴¹ this by replacing CEM with differentiable CEM (see supplementary), and then optimize in a latent
⁴² space of action sequences. This is much faster, but the results are not quite as good. However, since
⁴³ the whole policy is now differentiable, it can be fine-tuned using PPO (Section 37.3.4), which closes
⁴⁴ the performance gap at negligible cost.

⁴⁵

⁴⁶ is $(32 + 256) \times 3 + 3 = 867$, to account for the weights and biases.

⁴⁷

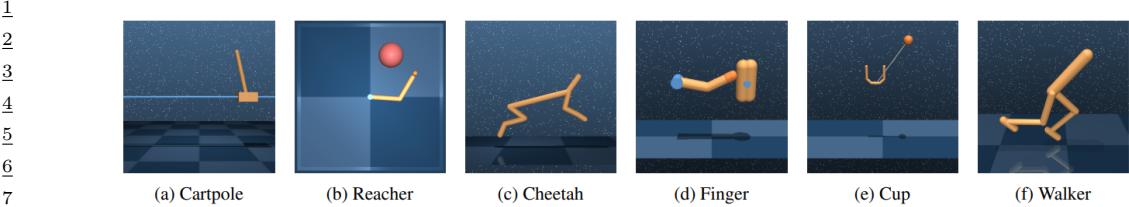


Figure 37.8: Illustration of some image-based control problems used in the PlaNet paper. Inputs are $64 \times 64 \times 3$. (a) The cartpole swingup task has a fixed camera so the cart can move out of sight, making this a partially observable problem. (b) The reacher task has a sparse reward. (c) The cheetah running task includes both contacts and a larger number of joints. (d) The finger spinning task includes contacts between the finger and the object. (e) The cup task has a sparse reward that is only given once the ball is caught. (f) The walker task requires balance and predicting difficult interactions with the ground when the robot is lying down. From Figure 1 of [Haf+19]. Used with kind permission of Danijar Hafner.

A recent extension of the PlaNet paper, known as **Dreamer**, was proposed in [Haf+20]. In this paper, the online MPC planner is replaced by a policy network, $\pi(a_t|z_t)$, which is learned using gradient-based actor-critic in latent space. The inference and generative models are trained by maximizing the ELBO, as in PlaNet. The policy is trained by SGD to maximize expected total reward as predicted by the value function, and the value function is trained by SGD to minimize MSE between predicted future reward and the TD- λ estimate (Section 37.2.2). They show that Dreamer gives better results than PlaNet, presumably because they learn a policy to optimize the long term reward (as estimated by the value function), rather than relying on MPC based on short-term rollouts.

37.4.6 Robustness to model errors

The main challenge with MBRL is that errors in the model can result in poor performance of the resulting policy, due to the distribution shift problem (Section 20.2). That is, the model is trained to predict states and rewards that it has seen using some behavior policy (e.g., the current policy), and then is used to compute an optimal policy under the learned model. When the latter policy is followed, the agent will experience a different distribution of states, under which the learned model may not be a good approximation of the real environment.

We require the model to generalize in a robust way to new states and actions. (This is related to the off-policy learning problem that we discuss in Section 37.5.) Failing that, the model should at least be able to quantify its uncertainty (Section 20.4). These topics are the focus of much recent research (see e.g., [Luo+19; Kur+19; Jan+19; Isl+19; Man+19; WB20; Eys+21]).

37.5 Off-policy learning

We have seen examples of off-policy methods such as Q-learning. They do not require that training data be generated by the policy it tries to evaluate or improve. Therefore, they tend to have greater data efficiency than their on-policy counterparts, by taking advantage of data generated by other policies. They are also easier to be applied in practice, especially in domains where costs and risks of

¹ following a new policy must be considered. This section covers this important topic.

² A key challenge in off-policy learning is that the data distribution is typically different from the
³ desired one, and this mismatch must be dealt with. For example, the probability of visiting a state s
⁴ at time t in a trajectory depends not only on the MDP's transition model, but also on the policy
⁵ that is being followed. If we are to estimate $J(\pi)$, as defined in Equation (37.15), but the trajectories
⁶ are generated by a different policy π' , simply averaging rewards in the data gives us $J(\pi')$, not $J(\pi)$.
⁷ We have to somehow correct for the gap, or "bias". Another challenge is that off-policy data can also
⁸ make an algorithm unstable and divergent, which we will discuss in Section 37.5.3.

⁹ Removing distribution mismatches is not unique in off-policy learning, and is also needed in
¹⁰ supervised learning to handle covariate shift (Section 20.2.3), and in causal effect estimation (Chap-
¹¹ ter 38), among others. Off-policy learning is also closely related to **offline reinforcement learning**
¹²(also called **batch reinforcement learning**): the former emphasizes the distributional mismatch
¹³ between data and the agent's policy, and the latter emphasizes that the data is static and no further
¹⁴ online interaction with the environment is allowed [LP03; EGW05; Lev+20]. Clearly, in the offline
¹⁵ scenario with fixed data, off-policy learning is typically a critical technical component. Recently,
¹⁶ several datasets have been prepared to facilitate empirical comparisons of offline RL methods (see
¹⁷e.g., [Gul+20; Fu+20]).

¹⁸ Finally, while this section focuses on MDPs, most methods can be simplified and adapted to the
¹⁹ special case of contextual bandits (Section 36.4). In fact, off-policy methods have been successfully
²⁰ used in numerous industrial bandit applications (see e.g., [Li+10; Bot+13; SJ15; HLR16]).

²¹

²² 37.5.1 Basic techniques

²³ We start with four basic techniques, and will consider more sophisticated ones in subsequent sections.
²⁴ The off-policy data is assumed to be a collection of trajectories: $\mathcal{D} = \{\tau^{(i)}\}_{1 \leq i \leq n}$, where each
²⁵ trajectory is a sequence as before: $\tau^{(i)} = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, \dots)$. Here, the reward and next states
²⁶ are sampled according to the reward and transition models; the actions are chosen by a **behavior**
²⁷ **policy**, denoted π_b , which is different from the **target policy**, π_e , that the agent is evaluating or
²⁸ improving. When π_b is unknown, we are in a **behavior-agnostic off-policy** setting.

²⁹

³⁰ 37.5.1.1 Direct method

³¹

³² A natural approach to off-policy learning starts with estimating the unknown reward and transition
³³ models of the MDP from off-policy data. This can be done using regression and density estimation
³⁴ methods on the reward and transition models, respectively, to obtain \hat{R} and \hat{P} ; see Section 37.4 for
³⁵ further discussions. These estimated models then give us an inexpensive way to (approximately)
³⁶ simulate the original MDP, and we can apply on-policy methods on the simulated data. This method
³⁷ directly models the outcome of taking an action in a state, thus the name **direct method**, and is
³⁸ sometimes known as **regression estimator** and **plug-in estimator**.

³⁹ While the direct method is natural and sometimes effective, it has a few limitations. First, a small
⁴⁰ estimation error in the simulator has a compounding effect in long-horizon problems (or equivalently,
⁴¹ when the discount factor γ is close to 1). Therefore, an agent that is optimized against an MDP
⁴² simulator may overfit the estimation errors. Unfortunately, learning the MDP model, especially the
⁴³ transition model, is generally difficult, making the method limited in domains where \hat{R} and \hat{P} can be
⁴⁴ learned to high fidelity. See Section 37.4.6 for a related discussion.

⁴⁵

1 **37.5.1.2 Importance sampling**

2 The second approach relies on importance sampling (IS) (Section 11.5) to correct for distributional
3 mismatches in the off-policy data. To demonstrate the idea, consider the problem of estimating the
4 target policy value $J(\pi_e)$ with a fixed horizon T . Correspondingly, the trajectories in \mathcal{D} are also of
5 length T . Then, the IS off-policy estimator, first adopted by [PSS00], is given by
6

7

$$\hat{J}_{\text{IS}}(\pi_e) \triangleq \frac{1}{n} \sum_{i=1}^n \frac{p(\boldsymbol{\tau}^{(i)}|\pi_e)}{p(\boldsymbol{\tau}^{(i)}|\pi_b)} \sum_{t=0}^{T-1} \gamma^t r_t^{(i)} \quad (37.47)$$

8 It can be verified that $\mathbb{E}_{\pi_b} [\hat{J}_{\text{IS}}(\pi_e)] = J(\pi_e)$, that is, $\hat{J}_{\text{IS}}(\pi_e)$ is **unbiased**, provided that $p(\boldsymbol{\tau}|\pi_b) > 0$
9 whenever $p(\boldsymbol{\tau}|\pi_e) > 0$. The **importance ratio**, $\frac{p(\boldsymbol{\tau}^{(i)}|\pi_e)}{p(\boldsymbol{\tau}^{(i)}|\pi_b)}$, is used to compensate for the fact that the
10 data is sampled from π_b and not π_e . Furthermore, this ratio does *not* depend on the MDP models,
11 because for any trajectory $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, \dots, s_T)$, we have from Equation (36.50) that
12

13

$$\frac{p(\boldsymbol{\tau}|\pi_e)}{p(\boldsymbol{\tau}|\pi_b)} = \frac{p(s_0) \prod_{t=0}^{T-1} \pi_e(a_t|s_t) p_T(s_{t+1}|s_t, a_t) p_R(r_t|s_t, a_t, s_{t+1})}{p(s_0) \prod_{t=0}^{T-1} \pi_b(a_t|s_t) p_T(s_{t+1}|s_t, a_t) p_R(r_t|s_t, a_t, s_{t+1})} = \prod_{t=0}^{T-1} \frac{\pi_e(a_t|s_t)}{\pi_b(a_t|s_t)} \quad (37.48)$$

14 This simplification makes it easy to apply IS, as long as the target and behavior policies are known. If
15 the behavior policy is unknown, we can estimate it from \mathcal{D} (using e.g., logistic regression or DNNs), and
16 replace π_b by its estimate $\hat{\pi}_b$ in Equation (37.48). For convenience, define the **per-step importance**
17 **ratio** at time t by $\rho_t(\boldsymbol{\tau}) \triangleq \pi_e(a_t|s_t)/\pi_b(a_t|s_t)$, and similarly, $\hat{\rho}_t(\boldsymbol{\tau}) \triangleq \pi_e(a_t|s_t)/\hat{\pi}_b(a_t|s_t)$.
18

19 Although IS can in principle eliminate distributional mismatches, in practice its usability is often
20 limited by its potentially high variance. Indeed, the importance ratio in Equation (37.47) can be
21 arbitrarily large if $p(\boldsymbol{\tau}^{(i)}|\pi_e) \gg p(\boldsymbol{\tau}^{(i)}|\pi_b)$. There are many improvements to the basic IS estimator.
22 One improvement is based on the observation that the reward r_t is independent of the trajectory
23 beyond time t . This leads to a **per-decision importance sampling** variant that often yields lower
24 variance (see Section 11.6.1 for a statistical motivation, and [LBB20] for a further discussion):
25

26

$$\hat{J}_{\text{PDIS}}(\pi_e) \triangleq \frac{1}{n} \sum_{i=1}^n \sum_{t=0}^{T-1} \prod_{t' \leq t} \rho_{t'}(\boldsymbol{\tau}^{(i)}) \gamma^t r_t^{(i)} \quad (37.49)$$

27 There are many other variants such as self-normalized IS and truncated IS, both of which aim to
28 reduce variance possibly at the cost of a small bias; precise expressions of these alternatives are found,
29 e.g., in [Liu+18b]. In the next subsection, we will discuss another systematic way to improve IS.
30

31 IS may also be applied to improve a policy against the policy value given in Equation (37.15).
32 However, directly applying the calculation of Equation (37.48) runs into a fundamental issue with IS,
33 which we will discuss in Section 37.5.2. For now, we may consider the following approximation of
34 policy value, averaging over the state distribution of the behavior policy:
35

36

$$J_b(\pi_\theta) \triangleq \mathbb{E}_{p_\beta^\infty(s)} [V_\pi(s)] = \mathbb{E}_{p_\beta^\infty(s)} \left[\sum_a \pi_\theta(a|s) Q_\pi(s, a) \right] \quad (37.50)$$

¹ Differentiating this and ignoring the term $\nabla_{\boldsymbol{\theta}} Q_{\pi}(s, a)$, as suggested by [DWS12], gives a way to
² (approximately) estimate the **off-policy policy-gradient** using a one-step IS correction ratio:
³

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J_b(\pi_{\boldsymbol{\theta}}) &\approx \mathbb{E}_{p_{\beta}^{\infty}(s)} \left[\sum_a \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q_{\pi}(s, a) \right] \\ &= \mathbb{E}_{p_{\beta}^{\infty}(s) \beta(a|s)} \left[\frac{\pi_{\boldsymbol{\theta}}(a|s)}{\beta(a|s)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q_{\pi}(s, a) \right] \end{aligned}$$

⁴
⁵ Finally, we note that in the tabular MDP case, there exists a policy π_* that is optimal in all states
⁶ (Section 36.5.5). This policy maximizes J and J_b simultaneously, so Equation (37.50) can be a good
⁷ proxy for Equation (37.15) as long as all states are “covered” by the behavior policy π_b . The situation
⁸ is similar when the set of value functions or policies under consideration is sufficiently expressive: an
⁹ example is a Q-learning like algorithm called Retrace [Mun+16; ASN20]. Unfortunately, in general
¹⁰ when we work with parametric families of value functions or policies, such a uniform optimality is
¹¹ lost, and the distribution of states has a direct impact on the solution found by the algorithm. We
¹² will revisit this problem in Section 37.5.2.
¹³

¹⁴

¹⁵ 37.5.1.3 Doubly robust

¹⁶ It is possible to combine the direct and importance sampling methods discussed previously. To
¹⁷ develop intuition, consider the problem of estimating $J(\pi_e)$ in a contextual bandit (Section 36.4),
¹⁸ that is, when $T = 1$ in \mathcal{D} . The **doubly robust** (DR) estimator is given by
¹⁹

$$\hat{J}_{\text{DR}}(\pi_e) \triangleq \frac{1}{n} \sum_{i=1}^n \left(\frac{\pi_e(a_0^{(i)}|s_0^{(i)})}{\hat{\pi}_b(a_0^{(i)}|s_0^{(i)})} \left(r_0^{(i)} - \hat{Q}(s_0^{(i)}, a_0^{(i)}) \right) + \hat{V}(s_0^{(i)}) \right) \quad (37.51)$$

²⁰ where \hat{Q} is an estimate of Q_{π_e} , which can be obtained using methods discussed in Section 37.2,
²¹ and $\hat{V}(s) = \mathbb{E}_{\pi_e(a|s)} [\hat{Q}(s, a)]$. If $\hat{\pi}_b = \pi_b$, the term \hat{Q} is canceled by \hat{V} on average, and we get the
²² IS estimate that is unbiased; if $\hat{Q} = Q_{\pi_e}$, the term \hat{Q} is canceled by the reward on average, and we get the
²³ estimator as in the direct method that is also unbiased. In other words, the estimator
²⁴ Equation (37.51) is unbiased, as long as one of the estimates, $\hat{\pi}_b$ and \hat{Q} , is right. This observation
²⁵ justifies the name doubly robust, which has its origin in causal inference (see e.g., [BR05]).

²⁶ The above DR estimator may be extended to MDPs recursively, starting from the last step. Given
²⁷ a length- T trajectory τ , define $\hat{J}_{\text{DR}}[T] \triangleq 0$, and for $t < T$,

$$\hat{J}_{\text{DR}}[t] \triangleq \hat{V}(s_t) + \hat{\rho}_t(\tau) \left(r_t + \gamma \hat{J}_{\text{DR}}[t+1] - \hat{Q}(s_t, a_t) \right) \quad (37.52)$$

²⁸

²⁹ where $\hat{Q}(s_t, a_t)$ is the estimated cumulative reward for the remaining $T - t$ steps. The DR estimator
³⁰ of $J(\pi_e)$, denoted $\hat{J}_{\text{DR}}(\pi_e)$, is the average of $\hat{J}_{\text{DR}}[0]$ over all n trajectories in \mathcal{D} [JL16]. It can be
³¹ verified (as an exercise) that the recursive definition is equivalent to
³²

$$\hat{J}_{\text{DR}}[0] = \hat{V}(s_0) + \sum_{t'=0}^{T-1} \left(\prod_{t'=0}^t \hat{\rho}_{t'}(\tau) \right) \gamma^t \left(r_t + \gamma \hat{V}(s_{t+1}) - \hat{Q}(s_t, a_t) \right) \quad (37.53)$$

³³

This form can be easily generalized to the infinite-horizon setting by letting $T \rightarrow \infty$ [TB16]. Other than double robustness, the estimator is also shown to result in minimum variance under certain conditions [JL16]. Finally, the DR estimator can be incorporated into policy gradient for policy optimization, to reduce gradient estimation variance [HJ20].

37.5.1.4 Behavior regularized method

The three methods discussed previously do not impose any constraint on the target policy π_e . Typically, the more different π_e is from π_b , the less accurate our off-policy estimation can be. Therefore, when we optimize a policy in offline RL, a natural strategy is to favor target policies that are “close” to the behavior policy. Similar ideas are discussed in the context of conservative policy gradient (Section 37.3.4).

One approach is to impose a hard constraint on the proximity between the two policies. For example, we may modify the loss function of DQN (Equation (37.14)) as follows

$$\mathcal{L}_1^{\text{DQN}}(\mathbf{w}) \triangleq \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(r + \gamma \max_{\pi: D(\pi, \pi_b) \leq \varepsilon} \mathbb{E}_{\pi(a'|s')} [Q_{\mathbf{w}^-}(s', a')] - Q_{\mathbf{w}}(s, a))^2 \right] \quad (37.54)$$

In the above, we replace the $\max_{a'}$ operation by an expectation over a policy that stays close enough to the behavior policy, measured by some distance function D . For various instantiations and further details, see e.g., [FMP19; Kum+19a].

We may also impose a soft constraint on the proximity, by penalizing target policies that are too different. The DQN loss function can be adapted accordingly:

$$\mathcal{L}_2^{\text{DQN}}(\mathbf{w}) \triangleq \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(r + \gamma \max_{\pi} \mathbb{E}_{\pi(a'|s')} [Q_{\mathbf{w}^-}(s', a')] - \alpha \gamma D(\pi(s'), \pi_b(s')) - Q_{\mathbf{w}}(s, a))^2 \right] \quad (37.55)$$

This idea has been used in contextual bandits [SJ15] and empirically studied in MDPs by [WTN19].

There are many choices for the function D , such as the KL-divergence, for both hard and soft constraints. More detailed discussions and examples can be found in [Lev+20].

Finally, behavior regularization and previous methods like IS can be combined, where the former ensures lower variance and greater generalization of the latter (e.g., [SJ15]). Furthermore, most proposed behavior regularized methods consider one-step difference in D , comparing $\pi(s)$ and $\pi_b(s)$ conditioned on s . In many cases, it is desired to consider the difference between the long-term distributions, p_{β}^{∞} and p^{∞} , which we will discuss next.

37.5.2 The curse of horizon

The IS and DR approaches presented in the previous section all rely on an importance ratio to correct distributional mismatches. The ratio depends on the entire trajectory, and its variance grows exponentially in the trajectory length T . Correspondingly, the off-policy estimate of either the policy value or policy gradient can suffer an exponentially large variance (and thus very low accuracy), a challenge called the **curse of horizon** [Liu+18b]. Policies found by approximate algorithms like Q-learning and off-policy actor-critic often have hard-to-control error due to distribution mismatches.

This section discusses an approach to tackling this challenge, by considering corrections in the state-action distribution, rather than in the trajectory distribution. This change is critical: [Liu+18b]

¹ describes an example, where the state-action distributions under the behavior and target policies
² are identical, but the importance ratio of a trajectory grows exponentially large. It is now more
³ convenient to assume the off-policy data consists of a set of transitions: $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{1 \leq i \leq m}$,
⁴ where $(s_i, a_i) \sim p_{\mathcal{D}}$ (some fixed but unknown sampling distribution, such as p_{β}^{∞}), and r_i and
⁵ s'_i are sampled from the MDP's reward and transition models. Given a policy π , we aim to
⁶ estimate the correction ratio $\zeta_*(s, a) = p_{\pi}^{\infty}(s, a)/p_{\mathcal{D}}(s, a)$, as it allows us to rewrite the policy value
⁷ (Equation (37.15)) as
⁸

$$\frac{9}{10} J(\pi) = \frac{1}{1 - \gamma} \mathbb{E}_{p_{\pi}^{\infty}(s, a)} [R(s, a)] = \frac{1}{1 - \gamma} \mathbb{E}_{p_{\beta}^{\infty}(s, a)} [\zeta_*(s, a) R(s, a)] \quad (37.56)$$

$$\underline{11}$$

¹² For simplicity, we assume the initial state distribution p_0 is known, or can be easily sampled from.
¹³ This assumption is often easy to satisfy in practice.

¹⁴ The starting point is the following linear program formulation for any given π :

$$\frac{15}{16} \max_{d \geq 0} -D_f(d \| p_{\mathcal{D}}) \quad \text{s.t.} \quad d(s, a) = (1 - \gamma)\mu_0(s)\pi(a|s) + \gamma \sum_{\bar{s}, \bar{a}} p(s|\bar{s}, \bar{a})d(\bar{s}, \bar{a})\pi(a|s) \quad \forall(s, a)$$

$$\underline{17}$$

$$\underline{18} \quad \quad \quad (37.57)$$

$$\underline{19}$$

²⁰ where D_f is the f -divergence (Section 2.9.1). The constraint is a variant of Equation (36.69), giving
²¹ similar flow conditions in the space of $\mathcal{S} \times \mathcal{A}$ under policy π . Under mild conditions, p_{π}^{∞} is only
²² solution that satisfies the flow constraints, so the objective does not affect the solution, but will
²³ facilitate the derivation below. We can now obtain the Lagrangian, with multipliers $\{\nu(s, a)\}$, and
²⁴ use the change-of-variables $\zeta(s, a) = d(s, a)/p_{\mathcal{D}}(s, a)$ to obtain the following optimization problem:

$$\frac{25}{26} \max_{\zeta \geq 0} \min_{\nu} \mathcal{L}(\zeta, \nu) = \mathbb{E}_{p_{\mathcal{D}}(s, a)} [-f(\zeta(s, a))] + (1 - \gamma)\mathbb{E}_{p_0(s)\pi(a|s)} [\nu(s, a)] \quad (37.58)$$

$$\underline{27} \quad \quad \quad + \mathbb{E}_{\pi(a'|s')p(s'|s, a)p_{\mathcal{D}}(s, a)} [\zeta(s, a) (\gamma\nu(s', a') - \nu(s, a))] \quad \underline{28}$$

²⁹ It can be shown that the saddle point to Equation (37.58) must coincide with the desired correction
³⁰ ratio ζ_* . In practice, we may parameterize ζ and ν , and apply two-timescales stochastic gradient
³¹ descent/ascent on the off-policy data \mathcal{D} to solve for an approximate saddle-point. This is the
³² **DualDICE** method [Nac+19a], which is extended to **GenDICE** [Zha+20c].

³³ Compared to the IS or DR approaches, Equation (37.58) does not compute the importance ratio of
³⁴ a trajectory, thus generally has a lower variance. Furthermore, it is behavior-agnostic, without having
³⁵ to estimate the behavior policy, or even to assume data consists of a collection of trajectories. Finally,
³⁶ this approach can be extended to be doubly robust (e.g., [UHJ20]), and to optimize a policy [Nac+19b]
³⁷ against the true policy value $J(\pi)$ (as opposed to approximations like Equation (37.50)). For more
³⁸ examples along this line of approach, see [ND20] and the references therein.

$$\underline{39}$$

⁴⁰ 37.5.3 The deadly triad

⁴¹ Other than introducing bias, off-policy data may also make a value-based RL method unstable and
⁴² even divergent. Consider the simple MDP depicted in Figure 37.9a, due to [Bai95]. It has 7 states
⁴³ and 2 actions. Taking the dashed action takes the environment to the 6 upper states uniformly at
⁴⁴ random, while the solid action takes it to the bottom state. The reward is 0 in all transitions, and
⁴⁵ $\gamma = 0.99$. The value function V_w uses a linear parameterization indicated by the expressions shown
⁴⁶

$$\underline{47}$$

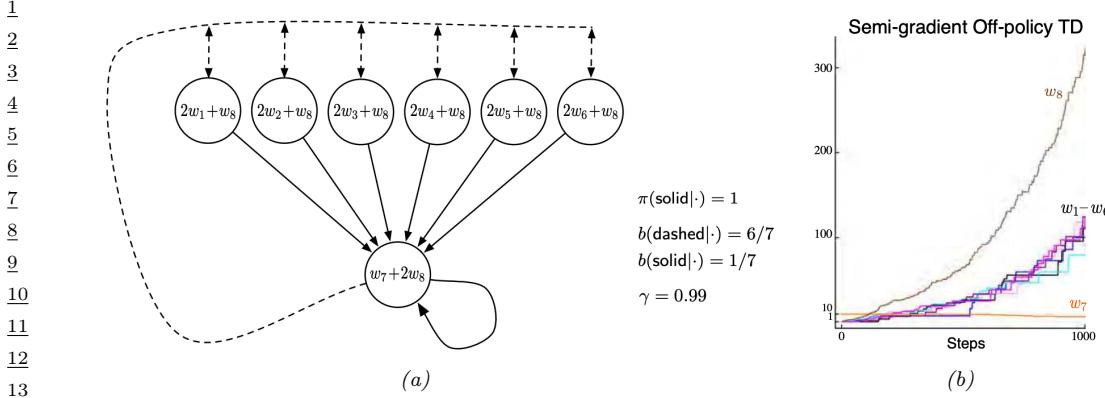


Figure 37.9: (a) ... (b) ... From Figures 11.1 & 11.2 of [SB18]. Used with kind permission of Richard Sutton.

inside the states, with $\mathbf{w} \in \mathbb{R}^8$. The target policies π always chooses the solid action in every state. Clearly, the true value function, $V_\pi(s) = 0$, can be exactly represented by setting $\mathbf{w} = \mathbf{0}$.

Suppose we use a behavior policy b to generate a trajectory, which chooses the dashed and solid actions with probabilities $6/7$ and $1/7$, respectively, in every state. If we apply TD(0) on this trajectory, the parameters diverge to ∞ (Figure 37.9b), even though the problem appears simple! In contrast, with on-policy data (that is, when b is the same as π), TD(0) with linear approximation can be guaranteed to converge to a good value function approximate [TR97].

The divergence behavior is demonstrated in many value-based bootstrapping methods, including TD, Q-learning, and related approximate dynamic programming algorithms, where the value function is represented either linearly (like the example above) or nonlinearly [Gor95; Ber19]. The root cause of these divergence phenomena is that the contraction property in the tabular case (Equation (36.63)) may no longer hold when V is approximated by $V_{\mathbf{w}}$. An RL algorithm can become unstable when it has these three components: off-policy learning, bootstrapping (for faster learning, compared to MC), function approximation (for generalization in large scale MDPs). This combination is known as **the deadly triad** [SB18]. It highlights another important challenge introduced by off-policy learning, and is a subject of ongoing research (e.g., [van+18; Kum+19a]).

A general way to ensure convergence in off-policy learning is to construct an objective function function, the minimization of which leads to a good value function approximation; see [SB18, Ch. 11] for more background. A natural candidate is the discrepancy between the left and right hand sides of the Bellman optimality equation Equation (36.58), whose unique solution is V_* . However, the “max” operator is not friendly to optimization. Instead, we may introduce an entropy term to smooth the greedy policy, resulting in a differential square loss in **path consistency learning (PCL)** [Nac+17]:

$$\min_{V, \pi} \mathcal{L}^{\text{PCL}}(V, \pi) \triangleq \mathbb{E} \left[\frac{1}{2} (r + \gamma V(s') - \lambda \log \pi(a|s) - V(s))^2 \right] \quad (37.59)$$

where the expectation is over (s, a, r, s') tuples drawn from some off-policy distribution (e.g., uniform over \mathcal{D}). Minimizing this loss, however, does not result in the optimal value function and policy in general, due to an issue known as “double sampling” [SB18, Sec. 11.5].

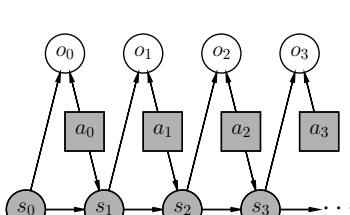


Figure 37.10: A graphical model for optimal control. States and actions are observed, while optimality variables are not. Adapted from Figure 1b of [Lev18].

This problem can be mitigated by introducing a dual function in the optimization [Dai+18]

$$\min_{V, \pi} \max_{\nu} \mathcal{L}^{\text{SBEED}}(V, \pi; \nu) \triangleq \mathbb{E} \left[\nu(s, a) (r + \gamma V(s') - \lambda \log \pi(a|s) - V(s))^2 - \nu(s, a)^2 / 2 \right] \quad (37.60)$$

where ν belongs to some function class (e.g., a DNN [Dai+18] or RKHS [FLL19]). It can be shown that optimizing Equation (37.60) forces ν to model the Bellman error. So this approach is called **smoothed Bellman error embedding**, or **SBEED**. In both PCL and SBEED, the objective can be optimized by gradient-based methods on parameterized value functions and policies.

37.6 Control as inference

In this section, we will discuss another approach to policy optimization, by reducing it to probabilistic inference. This approach allows one to incorporate domain knowledge in modeling, and apply powerful tools from approximate inference (see e.g., Chapter 7), in a consistent and flexible framework, with applications to, for example, robotics [PS07; Tou09; Neu11; Haa+18c] epidemiological decision making [Woo+20], and driver route preference modeling [Zie+08].

37.6.1 Maximum entropy reinforcement learning

We now describe a graphical model that exemplifies such a reduction, which results in RL algorithms that are closely related to some discussed previously. The model allows a trade-off between reward and entropy maximization, and recovers the standard RL setting when the entropy part vanishes in the trade-off. Our discussion mostly follows the approach of [Lev18].

Figure 37.10 gives a probabilistic model, which not only captures state transitions as before, but also introduces a new variable, o_t . This variable is binary, indicating whether the action at time t is optimal or not, and has the following probability distribution:

$$p(o_t = 1 | s_t, a_t) = \exp(\lambda^{-1} R(s_t, a_t)) \quad (37.61)$$

for some temperature parameter $\lambda > 0$ whose role will be clear soon. In the above, we have assumed without much loss of generality that $R(s, a) < 0$, so that Equation (37.61) gives a valid probability. Furthermore, we can assume a non-informative, uniform action prior, $p(a_t | s_t)$, to simplify

the exposition, for we can always push $p(a_t|s_t)$ into Equation (37.61). Under these assumptions, the likelihood of observing a length- T trajectory τ , when optimality achieved in every step, is:

$$\begin{aligned} p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1}) &\propto p(\tau, \mathbf{o}_{0:T-1} = \mathbf{1}) \propto p(s_0) \prod_{t=0}^{T-1} p(o_t = 1|s_t, a_t) p_T(s_{t+1}|s_t, a_t) \\ &= p(s_0) \prod_{t=0}^{T-1} p_T(s_{t+1}|s_t, a_t) \exp\left(\frac{1}{\lambda} \sum_{t=0}^{T-1} R(s_t, a_t)\right) \end{aligned} \quad (37.62)$$

The intuition of Equation (37.62) is clearest when the state transitions are deterministic. In this case, $p_T(s_{t+1}|s_t, a_t)$ is either 1 or 0, depending on whether the transition is dynamically feasible or not. Hence, $p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1})$ is either proportional to $\exp(\lambda^{-1} \sum_{t=0}^{T-1} R(s_t, a_t))$ if τ is feasible, or 0 otherwise. Maximizing reward is equivalent to inferring a trajectory with maximum $p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1})$.

The optimal policy in this probabilistic model is given by

$$\begin{aligned} p(a_t|s_t, \mathbf{o}_{t:T-1} = \mathbf{1}) &= \frac{p(s_t, a_t|\mathbf{o}_{t:T-1} = \mathbf{1})}{p(s_t|\mathbf{o}_{t:T-1} = \mathbf{1})} = \frac{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t)p(a_t|s_t)p(s_t)}{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t)p(s_t)} \\ &\propto \frac{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t)}{p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t)} \end{aligned} \quad (37.63)$$

The two probabilities in Equation (37.63) can be computed as follows, starting with $p(o_{T-1} = 1|s_{T-1}, a_{T-1}) = \exp(\lambda^{-1} R(s_{T-1}, a_{T-1}))$,

$$p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t) = \int_S p(\mathbf{o}_{t+1:T-1} = \mathbf{1}|s_{t+1}) p_T(s_{t+1}|s_t, a_t) \exp(\lambda^{-1} R(s_t, a_t)) ds_{t+1} \quad (37.64)$$

$$p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t) = \int_A p(\mathbf{o}_{t:T-1} = \mathbf{1}|s_t, a_t) p(a_t|s_t) da_t \quad (37.65)$$

The calculation above is expensive. In practice, we can approximate the optimal policy using a parametric form, $\pi_\theta(a_t|s_t)$. The resulted probability of trajectory τ now becomes

$$p_\theta(\tau) = p(s_1) \prod_{t=0}^{T-1} p_T(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t) \quad (37.66)$$

If we optimize θ so that $D_{\text{KL}}(p_\theta(\tau) \| p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1}))$ is minimized, which can be simplified to

$$D_{\text{KL}}(p_\theta(\tau) \| p(\tau|\mathbf{o}_{0:T-1} = \mathbf{1})) = -\mathbb{E}_{p_\theta} \left[\sum_{t=0}^{T-1} \lambda^{-1} R(s_t, a_t) + \mathbb{H}(\pi_\theta(s_t)) \right] + \text{const} \quad (37.67)$$

where the constant term only depends on the uniform action prior $p(a_t|s_t)$, but not θ . In other words, the objective is to maximize total reward, with an entropy regularization favoring more uniform policies. Thus this approach is called **maximum entropy RL**, or **MERL**. If π_θ can represent all stochastic policies, a softmax version of the Bellman equation can be obtained for Equation (37.67):

$$Q_*(s_t, a_t) = \lambda^{-1} R(s_t, a_t) + \mathbb{E}_{p_T(s_{t+1}|s_t, a_t)} \left[\log \int_A \exp(Q_*(s_{t+1}, a_{t+1})) da \right] \quad (37.68)$$

¹ with the convention that $Q_*(s_T, a) = 0$ for all a , and the optimal policy has a softmax form:
² $\pi_*(a_t|s_t) \propto \exp(Q_*(s_t, a_t))$. Note that the Q_* above is different from the usual optimal Q -function
³ (Equation (36.59)), due to the introduction of the entropy term. However, as $\lambda \rightarrow 0$, their difference
⁴ vanishes, and the softmax policy becomes greedy, recovering the standard RL setting.
⁵

⁶ The **soft actor-critic (SAC)** algorithm [Haa+18b; Haa+18c] is an off-policy actor-critic method
⁷ whose objective function is equivalent to Equation (37.67) (by taking T to ∞):

$$\frac{8}{9} J^{\text{SAC}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{p_{\pi}^{\infty}(s) \pi(a|s)} [R(s, a) + \lambda \mathbb{H}(\pi_{\boldsymbol{\theta}}(s))] \quad (37.69)$$

¹⁰ Note that the entropy term has also the added benefit of encouraging exploration.

¹¹ To compute the optimal policy, similar to other actor-critic algorithms, we will work with the “soft”
¹² state- and action-function approximations, parameterized by \mathbf{w} and \mathbf{u} , respectively:

$$\frac{13}{14} Q_{\mathbf{w}}(s, a) = R(s, a) + \gamma \mathbb{E}_{p_T(s'|s, a)} [V_{\mathbf{u}}(s', a') - \lambda \log \pi_{\boldsymbol{\theta}}(a'|s')] \quad (37.70)$$

$$\frac{15}{16} V_{\mathbf{u}}(s, a) = \lambda \log \sum_a \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a)) \quad (37.71)$$

¹⁷ This induces an improved policy (with entropy regularization): $\pi_{\mathbf{w}}(a|s) = \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a))/Z_{\mathbf{w}}(s)$,
¹⁸ where $Z_{\mathbf{w}}(s) = \sum_a \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a))$ is the normalization constant. We then perform a soft policy
¹⁹ improvement step to update $\boldsymbol{\theta}$ by minimizing $\mathbb{E}[D_{\text{KL}}(\pi_{\boldsymbol{\theta}}(s)\|\pi_{\mathbf{w}}(s))]$ where the expectation may be
²⁰ approximated by sampling s from a replay buffer D .

²¹ In [Haa+18c; Haa+18b], they show that the SAC method outperforms the off-policy DDPG
²² algorithm (Section 37.3.5) and the on-policy PPO algorithm (Section 37.3.4) by a wide margin on
²³ various continuous control tasks. For more details, see [Haa+18c].

²⁴ There is a variant of soft actor-critic, which only requires to model the action-value function. It is
²⁵ based on the observation that both the policy and soft value function can be induced by the soft
²⁶ action-value function as follows:

$$\frac{28}{29} V_{\mathbf{w}}(s) = \lambda \log \sum_a \exp(\lambda^{-1} Q_{\mathbf{w}}(s, a)) \quad (37.72)$$

$$\frac{30}{31} \pi_{\mathbf{w}}(a|s) = \exp(\lambda^{-1}(Q_{\mathbf{w}}(s, a) - V_{\mathbf{w}}(s))) \quad (37.73)$$

³¹ We then only need to learn \mathbf{w} , using approaches similar to DQN (Section 37.2.6). The resulting
³² algorithm, **soft Q-learning** [SAC17], is convenient if the number of actions is small (when \mathcal{A} is
³³ discrete), or if the integral in obtaining $V_{\mathbf{w}}$ from $Q_{\mathbf{w}}$ is easy to compute (when \mathcal{A} is continuous).

³⁵ It is interesting to see that algorithms derived in the maximum entropy RL framework bears a
³⁶ resemblance to PCL and SBEED in Section 37.5.3, both of which were to minimize an objective
³⁷ function resulting from the entropy-smoothed Bellman equation.

³⁸

³⁹ 37.6.2 Active inference

⁴⁰ Control as inference is also closely related to **active inference**; this is based on the **free energy**
⁴¹ **principle** which is popular in neuroscience (see e.g., [Fri09; Buc+17; Fri+22]). The FEP is equivalent
⁴² to using variational inference (see Section 10.1) to perform state estimation (perception) and parameter
⁴³ estimation (learning). In particular, consider a latent variable model with hidden states \mathbf{s} , observations
⁴⁴ \mathbf{o} and parameters $\boldsymbol{\theta}$. Following Section 10.1.1, we define the variational free energy to be
⁴⁵

$$\frac{46}{47} \mathcal{F}(\mathbf{o}) = D_{\text{KL}}(q(\mathbf{s}, \boldsymbol{\theta}|\mathbf{o})\|p(\mathbf{s}, \mathbf{o}, \boldsymbol{\theta})) \quad (37.74)$$

State estimation corresponds to solving $\min_{q(s|o)} \mathcal{F}(o)$, and parameter estimation corresponds to solving $\min_{q(\theta|o)} \mathcal{F}(o)$, just as in variational Bayes EM (Section 10.2.5). (Minimizing the VFE for certain hierarchical Gaussian models also forms the foundation of predictive coding, which we discuss in Section 8.4.3.)

To extend this to decision making problems we define the **expected free energy** as $\bar{\mathcal{F}}(a) = \mathbb{E}_{q(o|a)} [\mathcal{F}(o)]$, where $q(o|a)$ is the posterior predictive distribution over observations given actions sequence a . We then define the policy to be $\pi(a) = \mathcal{S}(\bar{\mathcal{F}}(a))$, where \mathcal{S} is the softmax function. To guide the agent towards preferred outcomes, we define the prior over states as $p(s) \propto e^{R(s)}$, where R is the reward function. Alternatively, we can define the prior over observations as $p(o) \propto e^{R(o)}$. Either way, the generative model is defined in terms of what the agent wants to achieve, rather than being an “objective” model of reality. The advantage of this approach is that it automatically induces *goal-directed* information-seeking behavior, rather than the maxent approach which models uncertainty in a goal-independent way. Despite this difference, the technique of active inference is very similar to control as inference, as explained in [Mil+20].

Note that originally active inference was defined in the tabular case (see e.g., [DC+20]), but it has recently been extended to the “deep” setting in [Uel18; Mil19b]. It can also be implemented in a message-passing framework [LV19], c.f. Section 10.2.7.

37.6.3 Other approaches

VIREL is an alternative model to maximum entropy RL [Fel+19]. Similar to soft actor-critic, it uses an approximate action-value function, Q_w , a stochastic policy, π_θ , and a binary optimality random variable o_t at time t . A different probability model for o_t is used

$$p(o_t = 1|s_t, a_t) = \exp\left(\frac{Q_w(s_t, a_t) - \max_a Q_w(s_t, a)}{\lambda_w}\right) \quad (37.75)$$

The temperature parameter λ_w is also part of the parameterization, and can be updated from data.

An EM method can be used to maximize the objective

$$\mathcal{L}(w, \theta) = \mathbb{E}_{p(s)} \left[\mathbb{E}_{\pi(a|s)} \left[\frac{Q_w(s, a)}{\lambda_w} \right] + \mathbb{H}(\pi_\theta(s)) \right] \quad (37.76)$$

for some distribution p that can be conveniently sampled from (e.g., in a replay buffer). The algorithm may be interpreted as an instance of actor-critic. In the E-step, the critic parameter w is fixed, and the actor parameter θ is updated using gradient accent with stepsize η_θ (for policy improvement):

$$\theta \leftarrow \theta + \eta_\theta \nabla_\theta \mathcal{L}(w, \theta) \quad (37.77)$$

In the M-step, the actor parameter is fixed, and the critic parameter is updated (for policy evaluation):

$$w \leftarrow w + \eta_w \nabla_w \mathcal{L}(w, \theta) \quad (37.78)$$

Finally, there are other possibilities of reducing optimal control to probabilistic inference, in addition to MERL and VIREL. For example, we may aim to maximize the expectation of the trajectory return G , by optimizing the policy parameter θ :

$$J(\pi_\theta) = \int G(\tau) p(\tau|\theta) d\tau \quad (37.79)$$

¹ It can be interpreted as a pseudo-likelihood function, when the $G(\tau)$ is treated as probability density,
² and solved (approximately) by a range of algorithms (see e.g., [PS07; Neu11; Abd+18]). Interestingly,
³ some of these methods have a similar objective as MERL (Equation (37.67)), although the distribution
⁴ involving θ appears in the second argument of KL . As discussed in Section 2.9.1, this forward
⁵ KL-divergence is mode-covering, which in the context of RL is argued to be less preferred than the
⁶ mode-seeking, reverse KL-divergence used by MERL. For more details and references, see [Lev18].
⁷

⁸

⁹ 37.6.4 Imitation learning

¹⁰

¹¹ In previous sections, an RL agent is to learn an optimal sequential decision making policy so that the
¹² total reward is maximized. **Imitation learning** (IL), also known as **apprenticeship learning** and
¹³ **learning from demonstration** (LfD), is a different setting, in which the agent does not observe
¹⁴ rewards, but has access to a collection \mathcal{D}_{exp} of trajectories generated by an expert policy π_{exp} ; that
¹⁵ is, $\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$ and $a_t \sim \pi_{\text{exp}}(s_t)$ for $\tau \in \mathcal{D}_{\text{exp}}$. The goal is to learn a good policy by
¹⁶ imitating the expert, in the absence of reward signals. IL finds many applications in scenarios where
¹⁷ we have demonstrations of experts (often humans) but designing a good reward function is not easy,
¹⁸ such as car driving and conversational systems. See [Osa+18] for a survey up to 2018.

¹⁹²⁰

²¹ 37.6.4.1 Imitation learning by behavior cloning

²² A natural method is **behavior cloning**, which reduces IL to supervised learning; see [Pom89] for
²³ an early application to autonomous driving. It interprets a policy as a classifier that maps states
²⁴ (inputs) to actions (labels), and finds a policy by minimizing the imitation error, such as
²⁵

$$\min_{\pi} \mathbb{E}_{p_{\pi_{\text{exp}}}^{\infty}(s)} [D_{\text{KL}}(\pi_{\text{exp}}(s) \| \pi(s))] \quad (37.80)$$

²⁶

²⁷ where the expectation wrt $p_{\pi_{\text{exp}}}^{\infty}$ may be approximated by averaging over states in \mathcal{D}_{exp} . A challenge
²⁸ with this method is that the loss does not consider the sequential nature of IL: future state distribution
²⁹ is not fixed but instead depends on earlier actions. Therefore, if we learn a policy $\hat{\pi}$ that has a low
³⁰ imitation error under distribution $p_{\pi_{\text{exp}}}^{\infty}$, as defined in Equation (37.80), it may still incur a large
³¹ error under distribution $p_{\hat{\pi}}^{\infty}$ (when the policy $\hat{\pi}$ is actually run). Further expert demonstrations or
³² algorithmic augmentations are often needed to handle the distribution mismatch (see e.g., [DLM09;
³³ RGB11]).
³⁴

³⁵

³⁶ 37.6.4.2 Imitation learning by inverse reinforcement learning

³⁷

³⁸ An effective approach to IL is **inverse reinforcement learning** (IRL) or **inverse optimal control**
³⁹ (IOC). Here, we first infer a reward function that “explains” the observed expert trajectories, and
⁴⁰ then compute a (near-)optimal policy against this learned reward using any standard RL algorithms
⁴¹ studied in earlier sections. The key step of reward learning (from expert trajectories) is the opposite
⁴² of standard RL, thus called inverse RL [NR00a].
⁴³

⁴⁴ It is clear that there are infinitely many reward functions for which the expert policy is optimal,
⁴⁵ for example by several optimality-preserving transformations [NHR99]. To address this challenge,
⁴⁶ we can follow the maximum entropy principle (Section 2.5.7), and use an energy-based probability
⁴⁷

1 model to capture how expert trajectories are generated [Zie+08]:
2

3

$$\begin{aligned} \text{4} \quad p(\tau) &\propto \exp\left(\sum_{t=0}^{T-1} R_\theta(s_t, a_t)\right) \\ \text{5} \end{aligned} \tag{37.81}$$

6

7 where R_θ is an unknown reward function with parameter θ . Abusing notation slightly, we denote
8 by $R_\theta(\tau) = \sum_{t=0}^{T-1} R_\theta(s_t, a_t)$ the cumulative reward along the trajectory τ . This model assigns
9 exponentially small probabilities to trajectories with lower cumulative rewards. The partition function,
10 $Z_\theta \triangleq \int_\tau \exp(R_\theta(\tau))$, is in general intractable to compute, and must be approximated. Here, we can
11 take a sample-based approach. Let \mathcal{D}_{exp} and \mathcal{D} be the sets of trajectories generated by an expert, and
12 by some known distribution q , respectively. We may infer θ by maximizing the likelihood, $p(\mathcal{D}_{\text{exp}}|\theta)$,
13 or equivalently, minimizing the negative log-likelihood loss

14

$$\begin{aligned} \text{15} \quad \mathcal{L}(\theta) &= -\frac{1}{|\mathcal{D}_{\text{exp}}|} \sum_{\tau \in \mathcal{D}_{\text{exp}}} R_\theta(\tau) + \log \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\exp(R_\theta(\tau))}{q(\tau)} \\ \text{16} \end{aligned} \tag{37.82}$$

17

18 The term inside the log of the loss is an importance sampling estimate of Z that is unbiased as long
19 as $q(\tau) > 0$ for all τ . However, in order to reduce the variance, we can choose q adaptively as θ is
20 being updated. The optimal sampling distribution (Section 11.5), $q_*(\tau) \propto \exp(R_\theta(\tau))$, is hard to
21 obtain. Instead, we may find a policy $\hat{\pi}$ which induces a distribution that is close to q_* , for instance,
22 using methods of maximum entropy RL discussed in Section 37.6.1. Interestingly, the process above
23 produces the inferred reward R_θ as well as an approximate optimal policy $\hat{\pi}$. This approach is used
24 by **guided cost learning** [FLA16], and found effective in robotics applications.

26 37.6.4.3 Imitation learning by divergence minimization

27 We now discuss a different, but related, approach to IL. Recall that the reward function depends
28 only on the state and action in an MDP. It implies that if we can find a policy π , so that $p_\pi^\infty(s, a)$
29 and $p_{\pi_{\text{exp}}}^\infty(s, a)$ are close, then π receives similar long-term reward as π_{exp} , and is a good imitation of
30 π_{exp} in this regard. A number of IL algorithms find π by minimizing the divergence between p_π^∞ and
31 $p_{\pi_{\text{exp}}}^\infty$. We will largely follow the exposition of [GZG19]; see [Ke+19b] for a similar derivation.

32 Let f be a convex function, and D_f the f -divergence (Section 2.9.1). From the above intuition, we
33 want to minimize $D_f(p_{\pi_{\text{exp}}}^\infty \| p_\pi^\infty)$. Then, using a variational approximation of D_f [NWJ10a], we can
34 solve the following optimization problem for π :

35

$$\min_{\pi} \max_{\mathbf{w}} \mathbb{E}_{p_{\pi_{\text{exp}}}^\infty(s, a)} [T_{\mathbf{w}}(s, a)] - \mathbb{E}_{p_\pi^\infty(s, a)} [f^*(T_{\mathbf{w}}(s, a))] \tag{37.83}$$

36

37 where $T_{\mathbf{w}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a function parameterized by \mathbf{w} . The first expectation can be estimated
38 using \mathcal{D}_{exp} , as in behavior cloning, and the second can be estimated using trajectories generated by
39 policy π . Furthermore, to implement this algorithm, we often use a parametric policy representation
40 π_θ , and then perform stochastic gradient updates to find a saddle-point to Equation (37.83).

41 With different choices of the convex function f , we can obtain many existing IL algorithms,
42 such as **generative adversarial imitation learning (GAIL)** [HE16b] and **adversarial inverse**
43 **RL (AIRL)** [FLL18], as well as new algorithms like **f-Divergence Max-Ent IRL (f-MAX)** and
44 **forward adversarial inverse RL (FAIRL)** [GZG19; Ke+19b].

1 Finally, the algorithms above typically require running the learned policy π to approximate the
2 second expectation in Equation (37.83). In risk- or cost-sensitive scenarios, collecting more data is not
3 always possible. Instead, we are in the off-policy IL setting, working with trajectories collected by some
4 policy other than π . Hence, we need to correct the mismatch between p_π^∞ and the off-policy trajectory
5 distribution, for which techniques from Section 37.5 can be used. An example is **ValueDICE** [KNT20],
6 which uses a similar distribution correction method of DualDICE (Section 37.5.2).
7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

38 Causality

This chapter was written by Victor Veitch and Alex D'Amour.

38.1 Introduction

38.1.1 Why is causality different than other forms of ML?

The bulk of machine learning considers relationships between observed variables with the goal of summarizing these relationships in a manner that allows predictions on similar data. However, for many problems, our main interest is to predict how system would change if it were observed under different conditions. For instance, in healthcare, we are interested in whether a patient will recover if given a certain treatment (as opposed to whether treatment and recovery are associated in the observed data). **Causal inference** addresses how to formalize such problems, determine whether they can be solved, and, if so, how to solve them. This chapter covers the fundamentals of this subject. Code examples for the discussed methods are available at <https://github.com/vveitch/causality-tutorials>.

To make the gap between observed data modeling and causal inference concrete, consider the relationships depicted in Figure 38.1 and Figure 38.2. Figure 38.1 shows the relationship between deaths by drowning and ice cream production in the United States in 1931 (the pattern holds across most years). Figure 38.2 shows the relationship between smoking and lung cancer across various countries. In each case, there is a strong positive association. Faced with this association, we might ask: could we reduce drowning deaths by banning ice cream? Could we reduce lung cancer by banning cigarettes? We intuitively understand that these interventional questions have different answers, despite the fact that the observed associations are similar. Determining the causal effect of some intervention in the world requires some such causal hypothesis about the world.

For concreteness, consider three possible explanations for the association between ice cream and drowning. Perhaps eating ice cream does cause people to drown—due to stomach cramps or similar. Or, perhaps, drownings increase demand for ice cream—the survivors eat huge quantities of ice cream to handle their grief. Or, the association may be due (at least in part) to a common cause: warm weather makes people more likely to eat ice cream and more likely to go swimming (and, hence, to drown). Under all three scenarios, we can observe exactly the same data, but the implications for an ice cream ban are very different. Hence, answering questions about what will happen under an intervention requires us to incorporate some causal knowledge of the world—e.g., which of these scenarios is plausible?

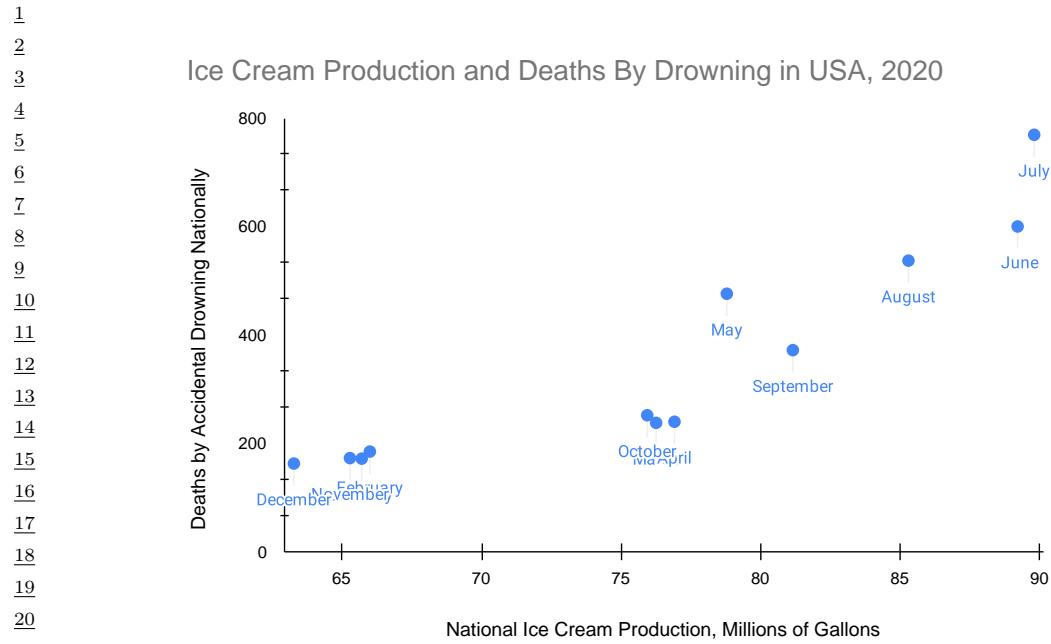


Figure 38.1: Ice cream production is strongly associated with deaths by drowning. Ice cream production data from the US Department of Agriculture National Agricultural Statistics Service. Drowning data from the National Center for Health Statistics at the United States Centers for Disease Control.

Our goal in this chapter to introduce the essentials of estimating causal effects. The high-level approach has three steps.

- **Causal Estimands:** The first step is to formally define the quantities we want to estimate. These are summaries of how the world would change under intervention, rather than summaries of the world as it has already been observed. E.g., we want to formalize “The expected number of drownings in the United States if we ban ice cream”.
- **Identification:** The next step is to identify the causal estimands with quantities that can, in principle, be estimated from observational data. This step involves codifying our causal knowledge of the world and translating this into a statement such as, “The causal effect is equal to the expected number of drownings after adjusting for month”. This step tells us what causal questions we could answer with perfect knowledge of the observed data distribution.
- **Estimation:** Finally, we must estimate the observable quantity using a finite data sample. The form of causal estimands favors certain efficient estimation procedures that allow us to exploit non-parametric (e.g., machine learning) predictive models.

In this chapter, we'll mainly focus on the estimation of the causal effect of an intervention averaged over all members of a population, known as the **Average Treatment Effect** or **ATE**. This is the

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

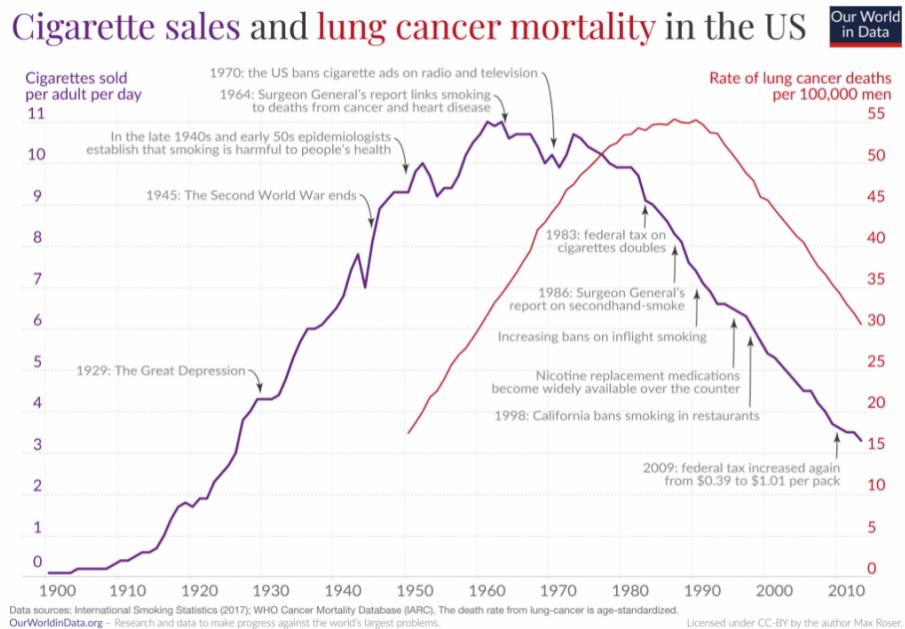


Figure 38.2: Smoking is strongly associated with lung cancer. Figure by Max Roser, ourworldindata.org/smoking-big-problem-in-brief

most common problem in applied causal inference work. It is in some sense the simplest problem, and will allow us to concretely explain the use and importance of the fundamental causal concepts. These causal concepts include structural causal models, causal graphical models, the do-calculus, and efficient estimation using influence function techniques. This problem is also useful for understanding the role that standard predictive modeling and machine learning play in estimating causal quantities.

38.2 Causal Formalism

In causal inference, the goal is to use data to learn about how the outcome in the world would change under intervention. In order to make such inferences, we must also make use of our causal knowledge of the world. This requires a formalism that lets us make the notion of intervention precise and lets us encode our causal knowledge as assumptions.

38.2.1 Structural Causal Models

Consider a setting in which we observe four variables from a population of people: A_i , an indicator of whether or not person i smoked at a particular age, Y_i , an indicator of whether or not person i developed lung cancer at a later age, H_i , a "health consciousness" index that measures a person's health-consciousness (perhaps constructed from a set of survey responses about attitudes toward

¹ health), and G_i , an indicator for whether the person has a genetic predisposition towards cancer.
² Suppose we observe a dataset of these variables drawn independently and identically from a population,
³ $(A_i, Y_i, H_i) \stackrel{\text{iid}}{\sim} P^{\text{obs}}$, where “obs” stands for “observed”.

⁴ In standard practice, we model data like these using probabilistic models. Notably, there are many
⁵ different ways to specify a probabilistic model for the same observed distribution. For example, we
⁶ could write a probabilistic model for P^{obs} as

$$\underline{8} \quad A \sim P^{\text{obs}}(A) \tag{38.1}$$

$$\underline{9} \quad H|A \sim P^{\text{obs}}(H|A) \tag{38.2}$$

$$\underline{10} \quad Y|A, H \sim P^{\text{obs}}(Y|H, A) \tag{38.3}$$

$$\underline{11} \quad G|A, H, Y \sim P^{\text{obs}}(G|A, H, Y) \tag{38.4}$$

¹²This is a valid factorization, and sampling variables in this order would yield valid samples from the
¹³joint distribution P^{obs} . However, this factorization does not map well to a mechanistic understanding
¹⁴of how these variables are causally related in the world. In particular, it is perhaps more plausible
¹⁵that health-consciousness H causally precedes smoking status A , since a person’s health-consciousness
¹⁶would influence their decision to smoke.

¹⁷These intuitions about causal ordering are intimately tied to the notion of intervention. Here, we
¹⁸will focus on a notion of intervention that can be represented in terms of “structural” models that
¹⁹describe mechanistic relationships between variables. The fundamental objects that we will reason
²⁰about are **structural causal models**, or SCM’s. SCM’s resemble probabilistic models, but they
²¹encode additional assumptions. Specifically, SCM’s serve two purposes: they describe a probabilistic
²²model *and* they provide semantics for transforming the data-generating process through intervention.
²³Formally, SCM’s describe a mechanistic data generating process with an ordered sequence of
²⁴equations that resemble assignment operations in a program. Each variable in a system is determined
²⁵by combining other modeled variables (the causes) with exogenous “noise” according to some
²⁶(unknown) deterministic function. For instance, a plausible SCM for P^{obs} might be

$$\underline{27} \quad G \leftarrow f_G(\xi_0) \tag{38.5}$$

$$\underline{28} \quad H \leftarrow f_H(\xi_1) \tag{38.6}$$

$$\underline{29} \quad A \leftarrow f_A(H, \xi_2) \tag{38.7}$$

$$\underline{30} \quad Y \leftarrow f_Y(G, H, A, \xi_3) \tag{38.8}$$

³¹where the (unknown) functions f are fixed, and the variables ξ are unmeasured causes, modeled
³²as independent random “noise” variables. Conceptually, the functions f_G, f_H, f_A, f_Y describe deter-
³³ministic physical relationships in the real world, while the variables ξ are hidden causes that are
³⁴sufficient to distinguish each unit i in the population. Because we assume that each observed unit i
³⁵is drawn at random from the population, we model ξ as random noise.

³⁶SCM’s imply probabilistic models, but not the other way around. For example, our example SCM
³⁷implies probabilistic model for the observed data based on the factorization $P^{\text{obs}}(G, H, A, Y) =$
³⁸ $P^{\text{obs}}(G)P^{\text{obs}}(H)P^{\text{obs}}(A | H)P^{\text{obs}}(Y | A, H)$. Thus, we could sample from the SCM in the same way
³⁹we would from a probabilistic model: draw a set of noise variables ξ and evaluate each assignment
⁴⁰operation in the SCM in order.

⁴¹Beyond the probabilistic model, an SCM encodes additional assumptions about the effects of
⁴²interventions. In an SCM, interventions are represented by replacing assignment statements. For
⁴³

⁴⁴

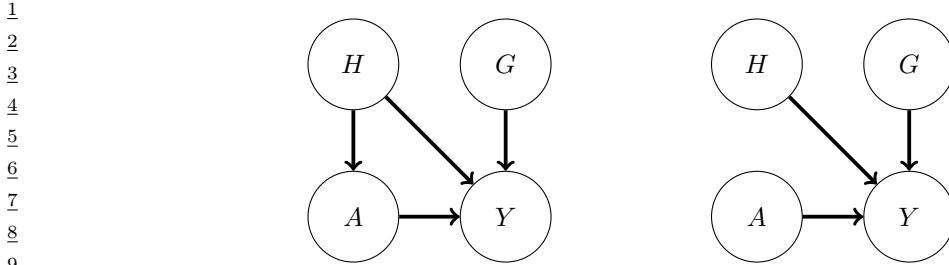


Figure 38.3: (Left) Causal graph illustrating relationships between smoking A , cancer Y , health consciousness H , and genetic cancer pre-disposition G . (Right) “Mutilated” causal graph illustrating relationships under an intervention on smoking A .

example, if we were interested in the distribution of Y in the hypothetical scenario that smoking were eliminated, we could set the second line of the SCM to be $A \leftarrow 0$. Because the f functions in the SCM are assumed to be invariant mechanistic relationships, the SCM encodes the assumption that this edited SCM generates data that we would see if we really applied this intervention in the world. Thus, the ordering of statements in an SCM are load-bearing: they imply substantive assumptions about how the world changes in response to interventions. This is in contrast to more standard probabilistic models where variables can be rearranged by applications of Bayes Rule without changing the substantive implications of the model.

We note that structural causal model may not incorporate all possible notions of causality. For example, laws based on conserved quantities or equilibria—e.g., the ideal gas law—do not trivially map to SCMs, though these are fundamental in disciplines such as physics and economics. Nonetheless, we will confine our discussion to SCMs.

38.2.2 Causal DAGs

SCM’s encode many details about the assumed generative process of a system, but often it is useful to reason about causal problems at a higher level of abstraction. In particular, it is often useful to separate the causal structure of a problem from the particular functional form of those causal relationships. **Causal graphs** provide this level of abstraction. A causal graph specifies which variables causally affect other variables, but leaves the parametric form of the structural equations f unspecified. Given an SCM, the corresponding causal graph can be drawn as follows: for each line of the SCM, draw arrows from the variables on the right hand side to variables on the left hand side. The causal DAG for our smoking-cancer example is shown in Figure 38.3. In this way, causal DAGs are related to SCMs in the same way that probabilistic graphical models (PGMs) are related to probabilistic models.

In fact, in the same way that SCMs imply a probabilistic model, causal DAGs imply a PGM. Functionally, causal graphs behave as probabilistic graphical models (Chapter 4). They imply conditional independence relationships between the variables in the observed data in same way. They obey the Markov property: If $X \leftarrow Y \rightarrow Z$ then $X \perp\!\!\!\perp Z|Y$; recall d-separation (Section 4.2.3.1). Additionally, if $X \rightarrow Y \leftarrow Z$ then, usually, $X \not\perp\!\!\!\perp Z|Y$ (even if X and Z are marginally independent).

In this case, Y is called a **collider** for X and Z .

¹ Conceptually, the difference between causal DAGs and PGMs is that probabilistic graphical models
² encode our assumptions about statistical relationships, whereas causal graphs encode our (stronger)
³ assumptions about causal relationships. Such causal relationships can be used to derive how statistical
⁴ relationships would change under intervention.
⁵

⁶ Causal graphs also allow us to reason about the causal and non-causal origins of statistical
⁷ dependencies in observed data without specifying a full SCM. In a causal graph, two variables—say,
⁸ A and D —can be statistically associated in different ways. First, there can be a directed path from
⁹ (ancestor) A to (descendant) D . In this case, A is a causal ancestor of D and interventions on A will
¹⁰ propagate through to change D ; $P(D|\text{do}(A = a)) \neq P(D|\text{do}(A = a'))$. For example, smoking is a
¹¹ causal ancestor of cancer in our example. Alternatively, A and D could share a common cause—there
¹² is some variable C such that there is a directed path from C to A and from C to D . If A and D
¹³ are associated only through such a path then interventions on A will not change the distribution of
¹⁴ D . However, it is still the case that $P(D|A = a) \neq P(D|A = a')$ —observing different values of A
¹⁵ changes our guess for the value of D . The reason is that A carries information about C , which carries
¹⁶ information about D . For example, suppose we lived in a world where there was no effect of smoking
¹⁷ on developing cancer (e.g., everybody vapes), there would nevertheless be an association between
¹⁸ smoking and cancer because of the path $A \leftarrow H \rightarrow Y$. The existence of such “backdoor paths” is one
¹⁹ core reason that statistical and causal association are not the same. Of course, more complicated
²⁰ variants of these associations are possible—e.g., C is itself only associated with A through a backdoor
²¹ path—but this already captures the key distinction between causal and non-causal paths.

²² Recall that our aim in introducing SCMs and causal graphs is to enable us to formalize our causal
²³ knowledge of the world and to make precise what interventional quantities we’d like to estimate.
²⁴ Writing down a causal graph gives a simple formal way to encode our knowledge of the causal
²⁵ structure of a problem. Usefully, this causal structure is sufficient to directly reason about the
²⁶ implications of interventions without fully specifying the underlying SCM. The key observation is that
²⁷ if a variable A is intervened on then, after intervention, none of the other variables are causes of A .
²⁸ That is, when we replace a line of an SCM with a statement directly assigning a variable a particular
²⁹ value, we cut off all dependencies that variable had on its causal parents. Accordingly, in the causal
³⁰ graph, the intervened on variable has no parents. This leads us to the **graph surgery** notion of
³¹ intervention: an intervention that sets A to a is the operation that deletes all incoming edges to A in
³² the graph, and then conditions on $A = a$ in the resulting probability distribution (which is defined
³³ by the conditional independence structure of the post-surgery graph). We’ll use Pearl’s do notation
³⁴ to denote this operation. $P(\mathbf{X}|\text{do}(A = a))$ is the distribution of \mathbf{X} given $A = a$ under the mutilated
³⁵ graph that results from deleting all edges going into A . Similarly, $\mathbb{E}[\mathbf{X}|\text{do}(A = a)] \triangleq \mathbb{E}_{P(\mathbf{X}|\text{do}(A=a))}[\mathbf{X}]$.
³⁶ Thus, we can formalize statements such as “The average effect of receiving drug A ” as

$$\text{ATE} = E[Y|\text{do}(A = 1)] - E[Y|\text{do}(A = 0)], \quad (38.9)$$

³⁸ where ATE stands for Average Treatment Effect.

³⁹ For concreteness, consider our running example. We contrast the distribution that results by
⁴⁰ conditioning on A with the distribution that results from intervening on A :

$$\text{P}(Y, H, G|A = a) = \text{P}(Y|H, G, A = a)\text{P}(G)\text{P}(H|A = a) \quad (38.10)$$

$$\text{P}(Y, H, G|\text{do}(A = a)) = \text{P}(Y|H, G, A = a)\text{P}(G)\text{P}(H) \quad (38.11)$$

⁴⁵ The key difference between these two distributions is that the standard conditional distribution
⁴⁶ describes a population where health consciousness H has the distribution that we observe among
⁴⁷

1 individuals with smoking status $A = a$, while the interventional distribution described a population
2 where health consciousness H follows the marginal distribution among all individuals. For example,
3 we would expect $P(H | A = \text{smoker})$ to put more mass on lower values of H than the marginal
4 health consciousness distribution than the marginal distribution $P(H)$, which would also include
5 non-smokers. The intervention distribution thus incorporates a hypothesis of how smoking would
6 affect the subpopulation individuals who tend to be too health conscious to smoke in the observed
7 data.
8

9 38.2.3 Identification

10 A central challenge in causal inference is that many different SCM's can produce identical distributions
11 of observed data. This means that, on the basis of observed data alone, we cannot uniquely identify
12 the SCM that generated it. This is true no matter how large of a data sample is available to us.

13 For example, consider the setting where there is a treatment A that may or may not have an
14 effect on outcome Y , and where both the treatment and outcome are known to be affected by
15 some *unobserved* common binary cause U . Now, we might be interested in the causal estimand
16 $E[Y|\text{do}(A = 1)]$. In general, we can't learn this quantity from the observed data. The problem is
17 that, we can't tell apart the case where the treatment has a strong effect from the case where the
18 treatment has no effect, but $U = 1$ both causes people to tend to be treated and causes increases the
19 probability of a positive outcome. The same observation shows we can't learn the (more complicated)
20 interventional distribution $P(Y|\text{do}(A = 1))$ (if we could learn this, then we'd get the average effect
21 automatically).

22 Thus, an important part of causal inference is to augment the observed data with knowledge about
23 the underlying causal structure of the process under consideration. Often, these assumptions can
24 narrow the space of SCM's sufficiently so that there is only one value of the causal estimand that is
25 compatible with the observed data. We say that the causal estimand is **identified** or **identifiable**
26 under a given set of assumptions if those assumptions are sufficient to provide a unique answer.
27 There are many different sets of sufficient conditions that yield identifiable causal effects; we call
28 each set of sufficient conditions an **identification strategy**.

29 Given a set of assumptions about the underlying SCM, the most common way to show that a
30 causal estimand is identified is by construction. Specifically, if the causal estimand can be written
31 entirely in terms of observable probability distributions, then it is identified. We call such a function of
32 observed distributions a **statistical estimand**. Once such a statistical estimand has been recovered,
33 we can then construct and analyze an estimator for that quantity using standard statistical tools.
34 As an example of a statistical estimand, in the SCM above, it can be shown the ATE as defined in
35 Equation (38.9), is equal to the following statistical estimand

$$\text{ATE} \stackrel{(*)}{=} \tau^{\text{ATE}} \triangleq \mathbb{E}[\mathbb{E}[Y|H, A = 1] - \mathbb{E}[Y|H, A = 0]], \quad (38.12)$$

36 where the equality $(*)$ only holds because of some specific properties of the SCM. Note that the RHS
37 above only involves conditional expectations between observed variables (there are no do operators),
38 so τ^{ATE} is only a function of observable probability distributions.

39 There are many kinds of assumptions we might make about the SCM governing the process under
40 consideration. For example, the following are assertions we might make about the system in our
41 running example:

¹ 1. The probability of developing cancer is additive on the logit scale in A , G , and H (i.e., logistic
² regression is a well-specified model).

⁴ 2. For each individual, smoking can never decrease the probability of developing cancer.

⁵ 3. Whether someone smokes is influenced by their health consciousness H , but not by their genetic
⁶ predisposition to cancer G .

⁸ These assumptions range from strong parametric assumptions fully specifying the form of the SCM
⁹ equations, to non-parametric assumptions that only specify what the inputs to each equation are,
¹⁰ leaving the form fully unspecified. Typically, assumptions that fully specify the parametric form are
¹¹ very strong, and would require far more detailed knowledge of the system under consideration than
¹² we actually have. The goal in identification arguments is to find a set of assumptions that are weak
¹³ enough that they might be plausibly true for the system under consideration, but which are also
¹⁴ strong enough to allow for identification of the causal effect.

¹⁵ If we are not willing to make any assumptions about the functional form of the SCM, then our
¹⁶ assumptions are just about which variables affect (and do not affect) the other variables. In this sense,
¹⁷ such which-affects-which assumptions are minimal. These assumptions are exactly the assumptions
¹⁸ captured by writing down a (possibly incomplete) causal DAG, showing which variables are parents
¹⁹ of each other variable. The graph may be incomplete because we may not know whether each possible
²⁰ edge is present in the physical system. For example, we might be unsure whether the gene G actually
²¹ has a causal effect on health consciousness H . It is natural to ask to what extent we can identify
²² causal effects only on the basis of partially specified causal DAGs. It turns out much progress can be
²³ made based on such non-parametric assumptions; we discuss this in detail in Section 38.8.

²⁴ We will also discuss certain assumptions that cannot be encoded in a causal graph, but that are
²⁵ still weaker than assuming that full functional forms are known. For example, we might assume that
²⁶ the outcome is affected additively by the treatment and any confounders, with no interaction terms
²⁷ between them. These weaker assumptions can enable causal identification even when assuming the
²⁸ causal graph alone does not.

²⁹ It is worth emphasizing that every causal identification strategy relies on assumptions that have
³⁰ some content that cannot be validated in the observed data. This follows directly from the ill-posedness
³¹ of causal problems: if the assumptions used to identify causal quantities could be validated, that
³² would imply that the causal estimand was identifiable from the observed data alone. However, since
³³ we know that there are many values of the causal estimand that are compatible with observed data,
³⁴ it follows that the assumptions in our identification strategy must have unobservable implications.
³⁵

³⁶ 38.2.4 Counterfactuals and the Causal Hierarchy

³⁷ Structural causal models let us formalize and study a hierarchy of different kinds of query about the
³⁸ system under consideration. The most familiar is observational queries: questions that are purely
³⁹ about statistical associations (e.g., “Are smoking and lung cancer associated in the population this
⁴⁰ sample was drawn from?”). Next is interventional queries: questions about causal relationships at
⁴¹ the population level (e.g., “How much does smoking increase the probability of cancer in a given
⁴² population?”). The rest of this chapter is focused on the definition, identification, and estimation of
⁴³ interventional queries. Finally, there are counterfactual queries: questions about causal relationships
⁴⁴ at the level of specific individuals, had something been different (e.g., “Would Alice have developed
⁴⁵ cancer had she not smoked?”). This causal hierarchy was popularized by [Pea09a, Ch. 1].
⁴⁶

⁴⁷

Interventional queries concern the prospective effect of an intervention on an outcome; for example, if we intervene and prevent a randomly sampled individual from smoking, what is the probability they develop lung cancer? Ultimately, the probability statement here is about our uncertainty about the “noise” variables ξ in the SCM. These are the unmeasured factors specific to the randomly selected individual. The distribution is determined by the population from which that individual is sampled. Thus, interventional queries are statements about populations. Interventional queries can be written in terms of conditional distributions using **do-notation**, e.g.

$$P(Y|\text{do}(A = 0)) \quad (38.13)$$

where conditioning on the “do” clause means that the line defining A in the underlying SCM was changed to an intervention setting $A \leftarrow 0$. In our example, this represents the distribution of lung cancer outcomes for an individual selected at random and prevented from smoking.

Counterfactual queries concern how an observed outcome might have been different had an intervention been applied in the past. Counterfactual queries are often framed in terms of attributing a given outcome to a particular cause. For example, would Alice have developed cancer had she not smoked? Did most smokers with lung cancer develop cancer because they smoked? Counterfactual queries are so called because they require a comparison of counterfactual outcomes within individuals. In the formalism of SCM’s, counterfactual outcomes for an individual i are generated by running the same values of ξ_i through differently intervened SCM’s. Counterfactual outcomes are often written in terms of *potential outcomes* notation. In our running smoking example, this would look like:

$$Y_i(a) \triangleq f_Y(G_i, H_i, a, \xi_{3,i}). \quad (38.14)$$

That is, $Y_i(a)$ is the outcome we would have seen had A been set to a while all of $G_i, H_i, \xi_{3,i}$ were kept fixed.

It is important to understand what distinguishes interventional and fundamentally counterfactual queries. Just because a query can be written in terms of potential outcomes does not make it a counterfactual query. For example, the average treatment effect, which is the canonical interventional query, is easy to write in potential outcomes notation:

$$\text{ATE} = \mathbb{E}[Y_i(1) - Y_i(0)]. \quad (38.15)$$

Instead, the key dividing line between counterfactual and interventional queries is whether the query requires knowing the joint distribution of potential outcomes within individuals, or whether marginal distributions of potential outcomes across individuals will suffice. An important signature of a counterfactual query is conditioning on the value of one potential outcome. For example, “the lung cancer rate among smokers who developed cancer, had they not smoked” is a counterfactual query, and can be written as:

$$\mathbb{E}[Y_i(0) | Y_i(1) = 1, A_i = 1] \quad (38.16)$$

Answering this query requires knowing how individual-level cancer outcomes are related (through $\xi_{3,i}$) across the worlds where the each individual i did and did not smoke. Notably, this query cannot be rewritten using do-notation, because it requires a distinction between $Y(O)$ and $Y(1)$ while the ATE can: $\mathbb{E}[Y | \text{do}(A = 1)] - \mathbb{E}[Y | \text{do}(A = 0)]$.

¹ Counterfactual queries require categorically more assumptions for identification than interventional
² ones. For identifying interventional queries, knowing the DAG structure of an SCM is often sufficient,
³ while for counterfactual queries, some assumptions about the functional forms in the SCM are
⁴ necessary. This is because only one potential outcome is ever observed for each individual, so the
⁵ dependence between potential outcomes within individuals is not observable. For example, the data
⁶ in our running example provide no information on how individual-level smoking and non-smoking
⁷ cancer risk are related. Thus, answering a question like “Did smokers who developed cancer have lower
⁸ non-smoking cancer risk than smokers who did not develop cancer?”, requires additional assumptions
⁹ about how characteristics encoded in ξ_i are translated to cancer outcomes. To answer this question
¹⁰ without such assumptions, we would need to observe smokers who developed cancer in the alternate
¹¹ world where they did not smoke. Because they compare how individuals would have turned out under
¹² different generating processes, counterfactual queries are often referred to as “cross-world” quantities.
¹³ On the other hand, interventional queries only require understanding the marginal distributions of
¹⁴ potential outcomes $Y_i(0)$ and $Y_i(1)$ across individuals; thus, no cross-world information is necessary
¹⁵ at the individual level.

¹⁶ We conclude this section by noting that counterfactual outcomes and potential outcomes notation
¹⁷ are often conceptually useful, even if they are not used to explicitly answer counterfactual queries.
¹⁸ Many causal queries are more intuitive to formalize in terms of potential outcomes. E.g., “Would I
¹⁹ have smoked if I was more health conscious?” may be more intuitive than “Would a randomly sampled
²⁰ individual from the same population have smoked had they been subject to an intervention that made
²¹ them more health conscious?”. In fact, some schools of causal inference use potential outcomes, rather
²² than DAGs, as their primary conceptual building block [See IR15]. Causal graphs and potential
²³ outcomes both provide ways to formalize interventional queries and causal assumptions. Ultimately,
²⁴ these are mathematically equivalent. Nevertheless, practically, they have different strengths. The
²⁵ main advantage of potential outcomes is that counterfactual statements often map more directly to
²⁶ our mechanistic understanding of the world. This can make it easier to articulate causal desiderata
²⁷ and causal assumptions we may wish to use. On the other hand, the potential outcomes notation
²⁸ does not automatically distinguish between interventional and counterfactual queries. Additionally,
²⁹ causal graphs often give an intuitive and easy way of articulating assumptions about structural
³⁰ causal models involving many variables—potential outcomes get quickly unwieldy. In short: both
³¹ formalizations have distinct advantages, and those advantages are simply about how easy it is to
³² translate our causal understanding of the world into crisp mathematical assumptions.
³³

³⁴

³⁵

³⁶ 38.3 Randomized Control Trials

³⁷

³⁸ We now turn to the business of estimating causal effects from data. We begin with **randomized**
³⁹ **control trials**, which are experiments designed to make the causal concerns as simple as possible.
⁴⁰ The simplest situation for causal estimation is when there are no common causes of A and Y . The
⁴¹ world is rarely so obliging as to make this the case. However, sometimes we can design an experiment
⁴² to enforce the no-common-causes structure. In randomized control trials we assign each participant
⁴³ to either the treatment or control group at random. Because random assignment does not depend on
⁴⁴ any property of the units in the study, there are no causes of treatment assignment, and hence also
⁴⁵ no common causes of Y and A .

⁴⁶ In this case, it’s straightforward to see that $P(Y|do(A = a)) = P(Y|a)$. This is essentially by
⁴⁷

1 definition of the graph surgery: since A has no parents, the mutilated graph is the same as the original
2 graph. Indeed, the graph surgery definition is chosen to make this true: any sensible formalization of
3 causality should have this identification result.
4

5 It is common to use RCTs to study the average treatment effect,

$$\text{ATE} = E[Y|\text{do}(A = 1)] - \mathbb{E}[Y|\text{do}(A = 0)]. \quad (38.17)$$

6 This is the expected difference between being assigned treatment and assigned no treatment for a
7 randomly chosen member of the population. It's easy to see that in an RCT this causal quantity is
8 identified as a parameter τ^{RCT} of the observational distribution:
9

$$\tau^{\text{RCT}} = \mathbb{E}[Y|A = 1] - \mathbb{E}[Y|A = 0].$$

10 Then, a natural estimator is:

$$\hat{\tau}^{\text{RCT}} \triangleq \frac{1}{n_A} \sum_{i:A_i=1} Y_i - \frac{1}{n - n_A} \sum_{i:A_i=0} Y_i, \quad (38.18)$$

11 where n_A is the number of units who received treatment. That is, we estimate the average treatment
12 effect as the difference between the average outcome of the treated group and the average outcome of
13 the untreated (control) group. ¹
14

15 Randomized control trials are the gold standard for estimating causal effects. This is because we
16 know *by design* that there are no confounders that can produce alternative causal explanations of the
17 data. In particular, the assumption of the triangle DAG—there are no unobserved confounders—is
18 enforced by design. However, there are limitations. Most obviously, randomized control trials are
19 sometimes infeasible to conduct. This could be due to expense, regulatory restrictions, or more
20 fundamental difficulties (e.g., in developmental economics, the response of interest is sometimes
21 collected decades after treatment). Additionally, it may be difficult to ensure that the participants in
22 an RCT are representative of the population where the treatment will be deployed. For instance,
23 participants in drug trials may skew younger and poorer than the population of patients who will
24 ultimately take the drug.
25

26 38.4 Confounder Adjustment

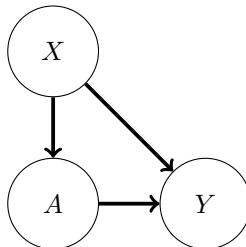
27 We now turn to the problem of estimating causal effects using observational (i.e., not experimental)
28 data. The most common application of causal inference is estimating the average treatment effect
29 (ATE) of an intervention. The ATE is also commonly called the **average causal effect**, or ACE.
30 Here, we focus on the important special case where the treatment A is binary, and we observe the
31 outcome Y as well as a set of common causes X that influence both A and Y .
32

33 38.4.1 Causal Estimand, Statistical Estimand, and Identification

34 Consider a problem where we observe treatment A , outcome Y , and covariates X , which are drawn
35 i.i.d. from some unknown distribution P . We wish to learn the average treatment effect: the expected
36

37 1. There is a literature on efficient estimation of causal effects in RCT's going back to Fisher [Fis25] that employ more
38 sophisticated estimators. See also Lin [Lin13a] and Bloniarz et al. [Blo+16] for more modern treatments.
39

1
2
3
4
5
6
7
8
9
10



11 *Figure 38.4: A causal DAG illustrating a situation where treatment A and outcome Y are both influenced by*
12 *observed confounders X.*

13
14

15 difference between being assigned treatment and assigned no treatment for a randomly chosen member
16 of the population. Following the discussion in the introduction, there are three steps to learning this
17 quantity: mathematically formalize the causal estimand, give conditions for the causal estimand to
18 be identified as a statistical estimand, and, finally, estimate this statistical estimand from data. We
19 now turn to the first two steps.

20 The average treatment effect is defined to be the difference between the average outcome if we
21 *intervened* and set A to be 0, versus the average outcome if we intervened and set A to be 1. Using
22 the do notation, we can write this formally as

23
24

$$\text{ATE} = \mathbb{E}[Y|\text{do}(A = 1)] - \mathbb{E}[Y|\text{do}(A = 0)]. \quad (38.19)$$

25

26 The next step is to articulate sufficient conditions for the ATE to be identified as a statistical
27 estimand (a parameter of distribution P). The key issue is the possible presence of **confounders**.
28 Confounders are “common cause” variables that affect both the treatment and outcome. When there
29 are confounding variables in observed data, the sub-population of people who are observed to have
30 received one level of the treatment A will differ from the rest of the population in ways that are
31 relevant to their observed Y . For example, there is a strong positive association between horseback
32 riding in childhood (treatment) and healthiness as an adult (outcome) [RB16]. However, both of these
33 quantities are influenced by wealth X . The population of people who rode horses as children ($A = 1$)
34 is wealthier than the population of people who did not. Accordingly, horseback-riding population
35 will have better health outcomes even if there is no actual causal benefit of horseback riding for adult
36 health.

37 We’ll express the assumptions required for causal identification in the form of a causal DAG.
38 Namely, we consider the simple triangle DAG in Figure 38.4, where the treatment and outcome
39 are influenced by *observed* confounders X . It turns out that the assumption encoded by this DAG
40 suffices for identification. To understand why this is so, recall that the target causal effect is defined
41 according to the distribution we would see if the edge from X to A was removed (that’s the meaning
42 of do). The key insight is that because the intervention only modifies the relationship between X
43 and A , the structural equation that generates outcomes Y given X and A , illustrated in Figure 38.4
44 as the $A \rightarrow Y \leftarrow X$, is the same even after the $X \rightarrow Y$ edge is removed. For example, we might
45 believe that the physiological processes by which smoking status A and confounders X produce
46 lung cancer Y remain the same, regardless of how the decision to smoke or not smoke was made.
47

Secondly, because the intervention does not change the composition of the population, we would also expect the distribution of background characteristics X to be the same between the observational and intervened processes.

With these insights about invariances between observed and interventional data, we can derive a statistical estimand for the ATE as follows.

Theorem 2 (Adjustment with No Unobserved Confounders). *We observe $A, Y, X \sim P$. Suppose that*

1. (*Confounders observed*) *The data obeys the causal structure in Figure 38.4. In particular, X contains all common causes of A and Y and no variable in X is caused by A or Y .*

2. (*Overlap*) $0 < P(A = 1|X = x) < 1$ for all values of x . That is, there are no individuals for whom treatment is always or never assigned.

Then, the average treatment effect is identified as $\text{ATE} = \tau$, where

$$\tau = \mathbb{E}[\mathbb{E}[Y|A = 1, X]] - \mathbb{E}[\mathbb{E}[Y|A = 0, X]]. \quad (38.20)$$

Proof. First, we expand the ATE using the tower property of expectation, conditioning on X . Then, we apply the invariances discussed above:

$$\text{ATE} = \mathbb{E}[Y|\text{do}(A = 1)] - \mathbb{E}[Y|\text{do}(A = 0)] \quad (38.21)$$

$$= \mathbb{E}[\mathbb{E}[Y|\text{do}(A = 1), X]] - \mathbb{E}[\mathbb{E}[Y|\text{do}(A = 0), X]] \quad (38.22)$$

$$= \mathbb{E}[\mathbb{E}[Y|A = 1, X]] - \mathbb{E}[\mathbb{E}[Y|A = 0, X]] \quad (38.23)$$

The final equality is the key to passing from a causal to observational quantity. This follows because, from the causal graph, the conditional distribution of Y given A, X is the same in both the original graph, and in the mutilated graph created by removing the edge from X to A . This mutilated graph defines $P(Y|\text{do}(A = 1), X)$, so the equality holds.

The condition that $0 < P(A = 1|X = x) < 1$ is required for the first equality (the tower property) to be well defined. \square

Note that Equation (38.20) is a function of only conditional expectations and distributions that appear in the observed data distribution (in particular, it contains no “do” operators). Thus, if we can fully characterize the observed data distribution P , we can map that distribution to a unique ATE.

It is useful to note how τ differs from the naive estimand $\mathbb{E}[Y|A = 1] - \mathbb{E}[Y|A = 0]$ that just reports the treatment-outcome association without adjusting for confounding. The comparison is especially clear when we write out the outer expectation in τ explicitly as an integral over X :

$$\tau = \int \mathbb{E}[Y | A = 1, X]P(X)dX - \int \mathbb{E}[Y | A = 0, X]P(X)dX \quad (38.24)$$

We can write the naive estimand in a similar form by applying the tower property of expectation:

$$\mathbb{E}[Y | A = 1] - \mathbb{E}[Y | A = 0] = \int \mathbb{E}[Y | A = 1, X]P(X | A = 1)dX - \int \mathbb{E}[Y | A = 0, X]P(X | A = 0)dX$$

1 (38.25)

2

3

4 The key difference is the probability distribuiton over X that is being integrated over. The observational difference in means integrates over the over distinct conditional distributions of confounders 5 X , depending on the value of A . On the other hand, in the ATE estimand τ , we integrate over the 6 same distribution $P(X)$ for both levels of the treatment.

7

8

9 **Overlap** In addition to the assumption on the causal structure, identification requires that there is 10 sufficient random variation in how treatments are assigned.

11

12 **Definition 1.** A distribution P on A, X satisfies **overlap** if $0 < P(A = 1|x) < 1$ for all x . It 13 satisfies **strict overlap** if $\epsilon < P(A = 1|x) < 1 - \epsilon$ for all x and some $\epsilon > 0$.

14

15 Overlap is the requirement that any unit could have either recieived the treatment or not.

16 To see the necessity of overlap, consider estimating the effectiveness of a drug in a study where 17 patient sex is a confounder, but the drug was only ever prescribed to male patients. Then, conditional 18 on a patient being female, we would know that patient was assigned to control. Without further 19 assumptions, it's impossible to know the effect of the drug on a population with female patients, 20 because there would be no data to inform the expected outcome for treated female patients, that is, 21 $E[Y | A = 1, X = \text{female}]$. In this case, the statistical estimand (38.20) would not be identifiable. In 22 the same vein, strict overlap ensures that the conditional distributions at each stratum of X can be 23 estimated in finite samples.

24 Overlap can be particularly limiting in settings where we are adjusting for a large number of 25 covariates (in an effort to satisfy no unobserved confounding). Then, certain combinations of traits 26 may be very highly predictive of treatment assignment, even if individual traits are not. E.g., male 27 patients over age 70 with BMI greater than 25 are very rarely assigned the drug. If such groups 28 represent a significant fraction of the target population, or have significantly different treatment 29 effects, then this issue can be problematic. In this case, the strict overlap assumption puts very strong 30 restrictions on observational studies: for an observational study to satisfy overlap, most dimensions 31 of the confounders X would need to closely mimic the balance we would expect in an RCT [D'A+21].

32

33 38.4.2 ATE Estimation with Observed Confounders

35 We now return to estimating the ATE using observed—i.e., not experimental—data. We've shown 36 that in the case where we observe all common causes of the treatment and outcome, the ATE is 37 causally identified with a statistical estimand τ . We now consider several strategies for estimating this 38 quantity using a finite data sample. Broadly, these techniques are known as backdoor adjustment.² 39 Recall that the defining characteristic of a confounding variable is that it affects both treatment 40 and outcome. Thus, an adjustment strategy may aim to account for the influence of confounders on 41 the observed outcome, the influence of confounders on treatment, or both. We discuss each of these 42 strategies in turn.

43

44 2. As we discuss in Section 38.8, this backdoor adjustment references the estimand returned by the do-calculus to 45 eliminate confounding from a backdoor path. This also generalizes the approaches discussed here to some cases where 46 we do not observe all common causes.

47

38.4.2.1 Outcome Model Adjustment

We begin with an approach to covariate adjustment that relies on modeling the conditional expectation of the outcome Y given treatment A and confounders X . This strategy is often referred to as g-computation or outcome adjustment.³ To begin, we define

Definition 2. *The conditional expected outcome is the function Q given by*

$$Q(a, x) = \mathbb{E}[Y|A = a, X = x]. \quad (38.26)$$

Substituting this definition into the definition of our estimand τ , Equation (38.20), we have $\tau = \mathbb{E}[Q(1, x) - Q(0, x)]$. This suggests a procedure for estimating τ : fit a model \hat{Q} for Q and then report

$$\hat{\tau}^Q \triangleq \frac{1}{n} \sum_i \hat{Q}(1, x_i) - \hat{Q}(0, x_i). \quad (38.27)$$

To fit \hat{Q} , recall that $E[Y|a, x] = \operatorname{argmin}_Q \mathbb{E}[(Y - Q(A, X))^2]$. That is, the minimizer (among all functions) of the squared loss risk is the conditional expected outcome.⁴ So, to approximate Q , we simply use mean squared error to fit a predictor that predicts Y from A and X .

The estimation procedure takes several steps. We first fit a model \hat{Q} to predict Y . Then, for each unit i , we predict that unit's outcome had they received treatment $\hat{Q}(1, x_i)$ and we predict their outcome had they not received treatment $\hat{Q}(0, x_i)$.⁵ If the unit actually did receive treatment ($a_i = 1$) then $\hat{Q}(0, x_i)$ is our guess about what would have happened in the counterfactual case that they did not. The estimated expected gain from treatment for this individual is $\hat{Q}(1, x_i) - \hat{Q}(0, x_i)$ —the difference in expected outcome between being treated and not treated. Finally, we estimate the outer expectation with respect to $P(X)$ —the true population distribution of the confounders—using the empirical distribution $\hat{P}(X) = 1/n \sum_i \delta_{x_i}$. In effect, this means we substitute the expectation (over an unknown distribution) by an average over the observed data.

Linear regression It's worth saying something more about the special case where Q is modeled as a linear function of both the treatment and all the covariates. That is, the case where we assume the identification conditions of Theorem 2 and we additionally assume that the true, causal law (the SCM) governing Y yields: $Q(A, X) = \mathbb{E}[Y|A, X] = \mathbb{E}[f_Y(A, X, \xi)|A, X] = \beta_0 + \beta_A A + \beta_X X$. Plugging in, we see that $Q(1, X) - Q(0, X) = \beta_A$ (and so also $\tau = \beta_A$). Then, the estimator for the average treatment effect reduces to the estimator for the regression coefficient β_A . This “fit linear regression and report the regression coefficient” remains a common way of estimating the association between two variables in practice. The expected-outcome-adjustment procedure here may be viewed as a generalization of this procedure that removes the linear parametric assumption.

38.4.2.2 Propensity Score Adjustment

Outcome model adjustment relies on modeling the relationship between the confounders and the outcome. A popular alternative is to model the relationship between the confounders and the

3. The “g” stands for generalized, for now-inscrutable historical reasons [Rob86].

4. To be precise, this definition applies when X and Y are square-integrable, and the minimization taken over measurable functions.

5. this interpretation is justified by the same conditions as Theorem 2

¹ treatment. This strategy adjusts for confounding by directly addressing sampling bias in the treated
² and control groups. This bias arises from the relationship between the confounders and the treatment.
³ Intuitively, the effect of confounding may be viewed as due to the difference between $P(X|A = 1)$
⁴ and $P(X|A = 0)$ —e.g., the population of people who rode horses as children is wealthier than the
⁵ population of people who did not. When we observe all confounding variables X , this degree of over-
⁶ or under-representation can be adjusted away by reweighting samples such that the confounders X
⁷ have the same distribution in the treated and control groups. When the confounders are balanced
⁸ between the two groups, then any differences between them must be attributable to the treatment.
⁹ A key quantity for balancing treatment and control groups is the **propensity score**, which
¹⁰ summarises the relationship between confounders and treatment.

¹¹
¹² **Definition 3.** *The propensity score is the function g given by $g(x) = P(A = 1|X = x)$.*

¹³ To make use of the propensity score in adjustment, we first rewrite the estimand τ in a suggestive
¹⁴ form:
¹⁵

$$\tau = \mathbb{E}\left[\frac{YA}{g(X)} - \frac{Y(1-A)}{1-g(X)}\right]. \quad (38.28)$$

¹⁶ This identity can be verified by noting that $\mathbb{E}[YA|X] = \mathbb{E}[Y|A = 1, X]P(A = 1|X) + 0$, rearranging
¹⁷ for $\mathbb{E}[Y|A = 1, X]$, doing the same for $\mathbb{E}[Y|A = 0, X]$, and substituting in to Equation (38.20). Note
¹⁸ that the identity is just a mathematical fact about the statistical estimand—it does not rely on any
¹⁹ causal assumptions, and holds whether or not τ can be interpreted as a causal effect.

²⁰ This expression suggests the **inverse probability of treatment weighted estimator**, or IPTW
²¹ estimator:

$$\hat{\tau}^{\text{IPTW}} \triangleq \frac{1}{n} \sum_i \frac{Y_i A_i}{\hat{g}(X_i)} - \frac{Y_i (1 - A_i)}{1 - \hat{g}(X_i)}. \quad (38.29)$$

²² Here, \hat{g} is a estimate of the propensity score function. Recall from Section 14.2.1 that if a model is well-
²³ specified and the loss function is a proper scoring rule then risk minimizer $g^* = \operatorname{argmin}_g \mathbb{E}[L(A, g(X))]$
²⁴ will be $g^*(X) = P(A = 1|X)$. That is, we can estimate the propensity score by fitting a model that
²⁵ predicts A from X . Cross-entropy and squared loss are both proper scoring rules, so we may use
²⁶ standard pipelines.

²⁷ In summary, the procedure is to estimate the propensity score function (with machine learning),
²⁸ and then to plug the estimated propensity scores $\hat{g}(x_i)$ into Equation (38.29). The IPTW estimator
²⁹ computes a difference of weighted averages between the treated and untreated group. The effect is to
³⁰ upweight the outcomes of units who were unlikely to be treated but who nevertheless actually, by
³¹ chance, received treatment (and similarly for untreated). Intuitively, such units are typical for the
³² untreated population. So, their outcomes under treatment are informative about what would have
³³ happened had a typical untreated unit received treatment.

³⁴ A word of warning is in order. Although the IPTW is asymptotically valid and popular in practice,
³⁵ it can be very unstable in finite samples. If estimated propensity scores are extreme for some values
³⁶ of x (that is, very close to 0 or 1), then the corresponding IPTW weights can be very large, resulting
³⁷ in a high-variance estimator. In some cases, this instability can be mitigated by instead using the
³⁸ Hajek version of the estimator.

$$\hat{\tau}^{\text{h-IPTW}} \triangleq \sum_i Y_i A_i \frac{1/\hat{g}(X_i)}{\sum_i A_i/\hat{g}(X_i)} - \sum_i Y_i (1 - A_i) \frac{1/(1-\hat{g}(X_i))}{\sum_i (1-A_i)/(1-\hat{g}(X_i))}. \quad (38.30)$$

However, extreme weights can persist even after self-normalization, either because there are truly strata of X where treatment assignment is highly imbalanced, or because the propensity score estimation method has overfit. In such cases, it is common to apply heuristics such as weight clipping.

See Khan and Ugander [KU21] for a longer discussion of inverse-propensity type estimators, including some practical improvements.

38.4.2.3 Double Machine Learning

We have seen how to estimate the average treatment effect using either the relationship between confounders and outcome, or the relationship between confounders and treatment. In each case, we follow a two step estimation procedure. First, we fit models for the expected outcome or the propensity score. Second, we plug these fitted models into a downstream estimator of the effect.

Unsurprisingly, the quality of the estimate of τ depends on the quality of the estimates \hat{Q} or \hat{g} . This is problematic because Q and g may be complex functions that require large numbers of samples to estimate. Even though we're only interested in the 1-dimensional parameter τ , the naive estimators described thus far can have very slow rates of convergence. This leads to unreliable inference or very large confidence intervals.

Remarkably, there are strategies for combining Q and g in estimators that, in principle, do better than using either Q or g alone. The **Augmented Inverse Probability of Treatment Weighted Estimator (AIPTW)** is one such estimator. It is defined as

$$\hat{\tau}^{\text{AIPTW}} \triangleq \frac{1}{n} \sum_i \hat{Q}(1, X_i) - \hat{Q}(0, X_i) + A_i \frac{Y_i - \hat{Q}(1, X_i)}{\hat{g}(x_i)} - (1 - A_i) \frac{Y_i - \hat{Q}(0, X_i)}{1 - \hat{g}(X_i)}. \quad (38.31)$$

That is, $\hat{\tau}^{\text{AIPTW}}$ is the outcome adjustment estimator plus a stabilization term that depends on the propensity score. This estimator is a particular case of a broader class of estimators that are referred to as **semi-parametrically efficient** or **double machine-learning** estimators [Che+17e; Che+17d]. We'll use the later terminology here.

We now turn to understanding the sense in which double machine learning estimators are robust to misestimation of the **nuisance functions** Q and g . To this end, we define the **influence curve** of τ to be the function ϕ defined by⁶

$$\phi(X_i, A_i, Y_i; Q, g, \tau) \triangleq Q(1, X_i) - Q(0, X_i) + A_i \frac{Y_i - Q(1, X_i)}{g(x_i)} - (1 - A_i) \frac{Y_i - Q(0, X_i)}{1 - g(X_i)} - \tau. \quad (38.32)$$

By design, $\hat{\tau}^{\text{AIPTW}} - \tau = \frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \tau)$. We begin by considering what would happen if we simply knew Q and g , and didn't have to estimate them. In this case, the estimator would be $\hat{\tau}^{\text{ideal}} = \frac{1}{n} \sum_i \phi(\mathbf{X}_i; Q, g, \tau)$ and, by the central limit theorem, we would have:

$$\sqrt{n}(\hat{\tau}^{\text{ideal}} - \tau) \xrightarrow{d} \text{Normal}(0, \mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]). \quad (38.33)$$

⁶ Influence curves are the foundation of what follows, and the key to generalizing the analysis beyond the ATE. Unfortunately, going into the general mathematics would require a major digression, so we omit it. However, see references at the end of the chapter for some pointers to the relevant literature.

¹ This result characterizes the estimation uncertainty in the best possible case. If we knew Q and g ,
² we could rely on this result for, e.g., finding confidence intervals for our estimate.
³

⁴ The question is: what happens when Q and g need to be estimated? For general estimators and
⁵ nuisance function models, we don't expect the \sqrt{n} -rate of Equation (38.33) to hold. For instance,
⁶ $\sqrt{n}(\hat{\tau}^Q - \tau)$ only converges if $\sqrt{n}\mathbb{E}[(\hat{Q} - Q)^2]^{\frac{1}{2}} \rightarrow 0$. That is, for the naive estimator we only get the
⁷ \sqrt{n} rate for estimating τ if we can also estimate Q at the \sqrt{n} rate—a much harder task! This is the
⁸ issue that the double machine learning estimator helps with.

⁹ To understand how, we decompose the error in estimating τ as follows:

$$\begin{aligned} \text{10} \quad & \sqrt{n}(\hat{\tau}^{\text{AIPTW}} - \tau) \\ \text{11} \quad & = \end{aligned} \tag{38.34}$$

$$\begin{aligned} \text{12} \quad & = \frac{1}{\sqrt{n}} \sum_i \phi(\mathbf{X}_i; Q, g, \tau) \\ \text{13} \quad & + \frac{1}{\sqrt{n}} \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}_i; Q, g, \tau) - \mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}; Q, g, \tau)] \end{aligned} \tag{38.35}$$

$$\begin{aligned} \text{14} \quad & + \sqrt{n}\mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}; Q, g, \tau)] \\ \text{15} \quad & + \sqrt{n}\mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}, \tau) - \phi(\mathbf{X}; Q, g, \tau)] \end{aligned} \tag{38.36}$$

¹⁶ We recognize the first term, Equation (38.35), as $\sqrt{n}(\hat{\tau}^{\text{ideal}} - \tau)$, the estimation error in the optimal
¹⁷ case where we know Q and g . Ideally, we'd like the error of $\hat{\tau}^{\text{AIPTW}}$ to be asymptotically equal to
¹⁸ this ideal case—which will happen if the other two terms go to 0.

¹⁹ The second term, Equation (38.36), is a penalty we pay for using the same data to estimate Q, g
²⁰ and to compute τ . For many model classes, it can be shown that such “empirical process” terms go
²¹ to 0. This can also be guaranteed in general by using different data for fitting the nuisance functions
²² and for computing the estimator (see the next section).

²³ The third term, Equation (38.37), captures the penalty we pay for misestimating the nuisance
²⁴ functions. This is where the particular form of the AIPTW is key. With a little algebra, we can show
²⁵ that

$$\begin{aligned} \text{26} \quad & \mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}) - \phi(\mathbf{X}; Q, g)] = \mathbb{E}\left[\frac{1}{g(X)}(\hat{g}(X) - g(X))(\hat{Q}(1, X) - Q(1, X))\right] \\ \text{27} \quad & + \frac{1}{1-g(X)}(\hat{g}(X) - g(X))(\hat{Q}(0, X) - Q(0, X)). \end{aligned} \tag{38.37}$$

²⁸ The important point is that estimation errors of Q and g are multiplied together. Using the Cauchy-
²⁹ Schwarz inequality, we find that $\sqrt{n}\mathbb{E}[\phi(\mathbf{X}; \hat{Q}, \hat{g}) - \phi(\mathbf{X}; Q, g)] \rightarrow 0$ as long as $\sqrt{n} \max_a \mathbb{E}[(\hat{Q}(a, X) -$
³⁰ $Q(a, X))^2]^{\frac{1}{2}} \mathbb{E}[(\hat{g}(X) - g(X))^2]^{\frac{1}{2}} \rightarrow 0$. That is, the misestimation penalty will vanish so long as the
³¹ product of the misestimation errors is $o(\sqrt{n})$. For example, this means that τ can be estimated at
³² the (optimal) \sqrt{n} rate even when the estimation error of each of Q and g only decreases as $o(n^{-1/4})$.
³³ The upshot here is that the double machine learning estimator has the special property that the
³⁴ weak condition $\sqrt{n}\mathbb{E}(\hat{Q}(T, X) - Q(T, X))^2 \mathbb{E}(\hat{g}(X) - g(X))^2 \rightarrow 0$ suffices to imply that

$$\begin{aligned} \text{35} \quad & \sqrt{n}(\hat{\tau}^{\text{AIPTW}} - \tau) \xrightarrow{d} \text{Normal}(0, \mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]) \\ \text{36} \quad & \end{aligned} \tag{38.40}$$

³⁷(though strictly speaking this requires some additional technical conditions we haven't discussed).
³⁸This is *not* true for the earlier estimators we discussed, which require a much faster rate of convergence
³⁹for the nuisance function estimation.

⁴⁰

The AIPTW estimator has two further nice properties that are worth mentioning. First, it is **non-parametrically efficient**. This means that this estimator has the smallest possible variance of any estimator that does not make parametric assumptions; namely, $\mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]$. This means, for example, that this estimator yields the smallest confidence intervals of any approach that does not rely on parametric assumptions. Second, it is **double robust**: the estimator is consistent (converges to the true τ as $n \rightarrow \infty$) as long as at least one of either \hat{Q} or \hat{g} is consistent.

38.4.2.4 Cross Fitting

The term Equation (38.36) in the error decomposition above is the penalty we pay for reusing the same data to both fit Q, g and to compute the estimator. For many choices of model for Q, g , this term goes to 0 quickly as n gets large and we achieve the (best case) \sqrt{n} error rate. However, this property doesn't always hold.

As an alternative, we can always randomly split the available data and use one part for model fitting, and the other to compute the estimator. Effectively, this means the nuisance function estimation and estimator computation are done using independent samples. It can then be shown that the reuse penalty will vanish. However, this comes at the price of reducing the amount of data available for each of nuisance function estimation and estimator computation.

This strategy can be improved upon by a **cross fitting** approach. We divide the data into K folds. For each fold j we use the other $K - 1$ folds to fit the nuisance function models $\hat{Q}^{-j}, \hat{g}^{-j}$. Then, for each datapoint i in fold j , we take $\hat{Q}(a_i, x_i) = \hat{Q}^{-j}(a_i, x_i)$ and $\hat{g}(x_i) = \hat{g}^{-j}(x_i)$. That is, the estimated conditional outcomes and propensity score for each datapoint are predictions from a model that was not trained on that datapoint. Then, we estimate τ by plugging $\{\hat{Q}(a_i, x_i), \hat{g}(x_i)\}_i$ into Equation (38.31). It can be shown that this cross fitting procedure has the same asymptotic guarantee—the central limit theorem at the \sqrt{n} rate—as described above.

38.4.3 Uncertainty Quantification

In addition to the point estimate $\hat{\tau}$ of the average treatment effect, we'd also like to report a measure of the uncertainty in our estimate. For example, in the form of a confidence interval. The asymptotic normality of $\sqrt{n}\hat{\tau}$ (Equation (38.40)) provides a means for this quantification. Namely, we could base confidence intervals and similar on the limiting variance $\mathbb{E}[\phi(\mathbf{X}_i; Q, g, \tau)^2]$. Of course, we don't actually know any of Q, g , or τ . However, it turns out that it suffices to estimate the asymptotic variance with $\frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2$ [Che+17e]. That is, we can estimate the uncertainty by simply plugging in our fitted nuisance models and our point estimate of τ into

$$\hat{\mathbb{V}}[\hat{\tau}] = 1/n \sum_i \phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2. \quad (38.41)$$

This estimated variance can then be used to compute confidence intervals in the usual manner. E.g., we'd report a 95% confidence interval for τ as $\hat{\tau} \pm 1.96\sqrt{\hat{\mathbb{V}}[\hat{\tau}]/n}$.

Alternatively, we could quantify the uncertainty by bootstrapping. Note, however, that this would require refitting the nuisance functions with each bootstrap model. Depending on the model and data, this can be prohibitively computationally expensive.

¹
² **38.4.4 Matching**

³ One particularly popular approach to adjustment-based causal estimation is **matching**. Intuitively,
⁴ the idea is to match each treated to unit to an untreated unit that has the same (or at least similar)
⁵ values of the confounding variables and then compare the observed outcomes of the treated unit and
⁶ its matched control. If we match on the full set of common causes, then the difference in outcomes is,
⁷ intuitively, a noisy estimate of the effect the treatment had on that treated unit. We'll now build
⁸ this up a bit more carefully. In the process we'll see that matching can be understood as, essentially,
⁹ a particular kind of outcome model adjustment.

¹⁰ For simplicity, consider the case where X is a discrete random variable. Define \mathcal{A}_x to be the set of
¹¹ treated units with covariate value x , and \mathcal{C}_x to be the set of untreated units with covariate value x .
¹² In this case, the matching estimator is:

¹³

$$\hat{\tau}^{\text{matching}} = \sum_x \hat{P}(x) \left(\frac{1}{|\mathcal{A}_x|} \sum_{i \in \mathcal{A}_x} Y_i - \frac{1}{|\mathcal{C}_x|} \sum_{j \in \mathcal{C}_x} Y_j \right), \quad (38.42)$$

¹⁴

¹⁵ where $\hat{P}(x)$ is an estimator of $P(X = x)$ —e.g., the fraction of units with $X = x$. Now, we can rewrite
¹⁶ $Y_i = Q(A_i, X_i) + \xi_i$ where ξ_i is a unit-specific noise term defined by the equation. In particular, we
¹⁷ have that $\mathbb{E}[\xi_i | A_i, X_i] = 0$. Subbing this in, we have:

¹⁸

$$\hat{\tau}^{\text{matching}} = \sum_x \hat{P}(x) (Q(1, x) - Q(0, x)) + \sum_x \frac{1}{|\mathcal{A}_x|} \sum_{i \in \mathcal{A}_x} \xi_i - \frac{1}{|\mathcal{C}_x|} \sum_{j \in \mathcal{C}_x} \xi_j. \quad (38.43)$$

¹⁹

²⁰ We can recognize the first term as an estimator of usual target parameter τ (it will be equal to τ if
²¹ $\hat{P}(x) = P(x)$). The second term is a difference of averages of random variables with expectation 0,
²² and so each term will converge to 0 as long as $|\mathcal{A}_x|$ and $|\mathcal{C}_x|$ each go to infinity as we see more and
²³ more data. Thus, we see that the matching estimator is a particular way of estimating the parameter
²⁴ τ . The procedure can be extended to continuous covariates by introducing some notion of values of
²⁵ X being close, and then matching close treatment and control variables.

²⁶ There are two points we should emphasize here. First, notice that the argument here has nothing
²⁷ to do with causal identification. Matching is a particular technique for estimating the observational
²⁸ parameter τ . Whether or not τ can be interpreted as an average treatment effect is determined by
²⁹ the conditions of Theorem 2—the particular estimation strategy doesn't say anything about this.
³⁰ Second, notice that in essence matching amounts to a particular choice of model for \hat{Q} . Namely,
³¹ $\hat{Q}(1, x) = \frac{1}{|\mathcal{A}_x|} \sum_{i \in \mathcal{A}_x} Y_i$ and similarly for $\hat{Q}(0, x)$. That is, we estimate the conditional expected
³² outcome as a sample mean over units with the same covariate value. Whether this is a good idea
³³ depends on how good of a model for Q this is. In situations where better models are possible (e.g.,
³⁴ a machine-learning model fits the data well), we might expect to get a more accurate estimate by
³⁵ using the conditional expected outcome predictor directly.

³⁶ There is another important case we mention in passing. In general, when using adjustment based
³⁷ identification, it suffices to adjust for any function $\phi(X)$ of X such that $A \perp\!\!\!\perp X | \phi(X)$. To see that
³⁸ adjusting for only $\phi(X)$ suffices, first notice that $g(X) = P(A = 1 | X) = P(A = 1 | \phi(X))$ only depends
³⁹ on $\phi(X)$, and then recall that can write the target parameter as $\tau = \mathbb{E}[\frac{YA}{g(X)} - \frac{Y(1-A)}{1-g(X)}]$, whence
⁴⁰ τ only depends on X through $g(X)$. That is: replacing X by a reduced version $\phi(X)$ such that
⁴¹ $g(X) = P(A = 1 | \phi(X))$ can't make any difference to τ . Indeed, the most popular choice of $\phi(X)$ is
⁴²

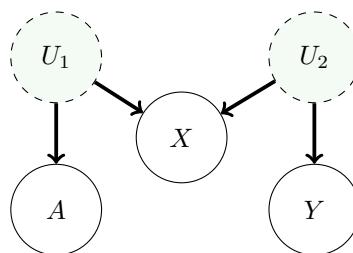


Figure 38.5: The M-bias causal graph. Here, A and Y are not confounded. However, conditioning on the covariate X opens a backdoor path, passing through U_1 and U_2 (because X is a collider). Thus, adjusting for X creates bias. This is true even though X need not be a pre-treatment variable.

the propensity score itself, $\phi(X) = g(X)$. This leads to **propensity score matching**, a two step procedure where we first fit a model for the propensity score, and then run matching based on the estimated propensity score values for each unit. Again, this is just a particular estimation procedure for the observational parameter τ , and says nothing about whether it's valid to interpret τ as a causal effect.

38.4.5 Practical Considerations and Procedures

Lots of issues can arise in practice.

38.4.5.1 What to adjust for

Choosing which variables to adjust for is a key detail in estimating causal effects using covariate adjustment. The criterion is clear when one has a full causal graph relating A , Y , and all covariates X to each other. Namely, adjust for all variables that are actually causal parents of A and Y . In fact, with access to the full graph, this criteria can be generalized somewhat—see Section 38.8.

In practice, we often don't actually know the full causal graph relating all of our variables. As a result, it is common to apply simple heuristics to determine which variables to adjust for. Unfortunately, these heuristics have serious limitations. However, exploring these are instructive.

A key condition in Theorem 2 is that the covariates X that we adjust for must include all the common causes. In the absence of a full causal graph, it is tempting to condition on as many observed variables as possible to try to ensure this condition holds. However, this can be problematic. For instance, suppose that M is a mediator of the effect of A on Y —i.e., M lies on one of the directed paths between A and Y . Then, conditioning on M will block this path, removing some of the causal effect. Note that this does not always result in an attenuated, or smaller-magnitude, effect estimate. The effect through a given mediator may run in the opposite direction of other causal pathways from the treatment; thus conditioning on a mediator can inflate or even flip the sign of a treatment effect. Alternatively, if C is a collider between A and Y —a variable that is caused by both—then conditioning on C will induce an extra statistical dependency between A and Y .

Both pitfalls of the “condition on everything” heuristic discussed above both involve conditioning

¹ on variables that are downstream of the treatment A . A natural response is to this is to limit
² conditioning to all pre-treatment variables, or those that are causally upstream of the treatment.
³ Importantly, if there is a valid adjustment set in the observed covariates X , then there will also be a
⁴ valid adjustment set among the pre-treatment covariates. This is because any open backdoor path
⁵ between A and Y must include a parent of A , and the set of pre-treatment covariates includes these
⁶ parents. However, it is still possible that conditioning on the full set of pre-treatment variables can
⁷ induce new backdoor paths between A and Y through colliders. In particular, if there is a covariate
⁸ D that is separately confounded with the treatment A and the outcome Y then D is a collider, and
⁹ conditioning on D opens a new backdoor path. This phenomenon is known as m-bias because of the
¹⁰ shape of the graph [Pea09c], see Figure 38.5.

¹¹ A practical refinement of the pre-treatment variable heuristic is given in VanderWeele TJ [VT11].
¹² Their heuristic suggests conditioning on all pre-treatment variables that are causes of the treatment,
¹³ outcome, or both. The essential qualifier in this heuristic is that the variable is causally upstream of
¹⁴ treatment and/or outcome. This eliminates the possibility of conditioning on covariates that are
¹⁵ only confounded with treatment and outcome, avoiding m-bias. Notably, this heuristic requires more
¹⁶ causal knowledge than the above heuristics, but does not require detailed knowledge of how different
¹⁷ covariates are causally related to each other.

¹⁸ The VanderWeele TJ [VT11] criterion is a useful rule of thumb, but other practical considerations
¹⁹ often arise. For example, if one has more knowledge about the causal structure among covariates, it
²⁰ is possible to optimize adjustment sets to minimize the variance of the resulting estimator [RS20].
²¹ One important example of reducing variance by pruning adjustment sets is the exclusion of variables
²² that are known to only be a parent of the treatment, and not of the outcome (so called instruments,
²³ as discussed in Section 38.5).

²⁴ Finally, adjustment set selection criteria operate under the assumption that there actually exists a
²⁵ valid adjustment set among observed covariates. When there is no set of observed covariates in X
²⁶ that block all backdoor paths, then any adjusted estimate will be biased. Importantly, in this case,
²⁷ the bias does not necessarily decrease as one conditions on more variables. For example, conditioning
²⁸ on an instrumental variable often results in an estimate that has higher bias, in addition to the
²⁹ higher variance discussed above. This phenomenon is known as bias amplification or z-bias; see
³⁰ Section 38.7.2. A general rule of thumb is that variables that explain away much more variation in
³¹ the treatment than in the outcome can potentially amplify bias, and should be treated with caution.

³²

³³ 38.4.5.2 Overlap

³⁴

³⁵ Recall that in addition to no-unobserved-confounders, identification of the average treatment effect
³⁶ requires overlap: the condition that $0 < P(A = 1|x) < 1$ for the population distribution P . With
³⁷ infinite data, any amount of overlap will suffice for estimating the causal effect. In realistic settings,
³⁸ even near failures can be problematic. Equation (38.40) gives an expression for the (asymptotic)
³⁹ variance of our estimate: $\mathbb{E}[\phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2]/n$. Notice that $\phi(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau})^2$ involves terms that are
⁴⁰ proportional to $1/g(x)$ and $1/(1 - g(x))$. Accordingly, the variance of our estimator will balloon
⁴¹ if there are units where $g(x) \approx 0$ or $g(x) \approx 1$ (unless such units are rare enough that they don't
⁴² contribute much to the expectation).

⁴³ In practice, a simple way to deal with potential overlap violation is to fit a model \hat{g} for the
⁴⁴ treatment assignment probability—which we need to do anyways—and check that the values $\hat{g}(x)$
⁴⁵ are not too extreme. In the case that some values are too extreme, the simplest resolution is to cheat.

⁴⁶

We can simply exclude all the data with extreme values of $\hat{g}(x)$. This is equivalent to considering the average treatment effect over only the subpopulation where overlap is satisfied. This changes the interpretation of the estimand. The restricted subpopulation ATE may or may not provide a satisfactory answer to the real-world problem at hand, and this needs to be justified based on knowledge of the real-world problem.

38.4.5.3 Choice of Estimand and Average Treatment Effect on the Treated

Usually, our goal in estimating a causal effect is qualitative. We want to know what the sign of the effect is, and whether it's large or small. The utility of the ATE is that it provides a concrete query we can use to get a handle on the qualitative question. However, it is not sacrosanct; sometimes we're better off choosing an alternative causal estimand that still answers the qualitative question but which is easier to estimate statistically. The **average treatment effect on the treated** or ATT,

$$\text{ATT} \triangleq \mathbb{E}_{X|A=1}[\mathbb{E}[Y|X, \text{do}(A = 1)] - E[Y|X, \text{do}(A = 0)]], \quad (38.44)$$

is one such an estimand that is frequently useful.

The ATT is useful when many members of the population are very unlikely to receive treatment, but the treated units had a reasonably high probability of receiving the control. This can happen if, e.g., we sample control units from the general population, but the treatment units all self-selected into treatment from a smaller subpopulation. In this case, it's not possible to (non-parametrically) determine the treatment effect for the control units where no similar unit took treatment. The ATT solves this obstacle by simply omitting such units from the average.

If we have the causal structure Figure 38.4, and the overlap condition $P(A = 1|X = x) < 1$ for all $X = x$ then the ATT is causally identified as

$$\tau^{\text{ATT}} = \mathbb{E}_{X|A=1}[\mathbb{E}[Y|A = 1, X] - E[Y|A = 0, X]]. \quad (38.45)$$

Note that the required overlap condition here is weaker than for identifying the ATE. (The proof is the same as Theorem 2.)

The estimation strategies for the ATE translate readily to estimation strategies for the ATT. Namely, estimate the nuisance functions the same way and then simply replace averages over all data points by averages over the treated datapoints only. In principle, it's possible to do a little better than this by making use of the untreated datapoints as well. A corresponding double machine learning estimator is

$$\hat{\tau}^{\text{ATT-AIPTW}} \triangleq \frac{1}{n} \sum_i \frac{A_i}{P(A = 1)} (Y - \hat{Q}(0, X_i)) - \frac{(1 - A_i)g(X)}{P(A = 1)(1 - g(X))} (Y - \hat{Q}(0, X_i)). \quad (38.46)$$

. The variance of this estimator can be estimated by

$$\phi^{\text{ATT}}(\mathbf{X}_i; Q, g, \tau) \triangleq \frac{1}{n} \sum_i \frac{A_i}{P(A = 1)} (Y - \hat{Q}(0, X_i)) - \frac{(1 - A_i)g(X)}{P(A = 1)(1 - g(X))} (Y - \hat{Q}(0, X_i)) - \frac{A\tau}{P(A = 1)} \quad (38.47)$$

$$\hat{\mathbb{V}}[\hat{\tau}^{\text{ATT-AIPTW}}] \triangleq \frac{1}{n} \sum_i \phi^{\text{ATT}}(\mathbf{X}_i; \hat{Q}, \hat{g}, \hat{\tau}^{\text{ATT-AIPTW}}). \quad (38.48)$$

¹ Notice that the estimator for the ATT doesn't require estimating $Q(1, X)$. This can be a considerable
² advantage when the treated units are rare.
³

⁴ See Chernozhukov et al. [Che+17e] for details.

⁵

⁶ 38.4.6 Summary and Practical Advice

⁷ We have seen a number of estimators that follow the general procedure:
⁸

- ⁹ 1. fit statistical or machine-learning models $\hat{Q}(a, x)$ as a predictor for Y , and/or $\hat{g}(x)$ as a predictor
¹⁰ for A
- ¹¹ 2. compute the predictions $\hat{Q}(0, x_i), \hat{Q}(1, x_i), \hat{g}(x_i)$ for each data point, and
¹²
- ¹³ 3. combine these predictions into an estimate of the average treatment effect.
¹⁴

¹⁵ Importantly, no single estimation approach is a silver bullet. For example, the double machine-
¹⁶ learning estimator has appealing theoretical properties, such as asymptotic efficiency guarantees and
¹⁷ a recipe for estimating uncertainty without needing to bootstrap the model fitting. However, in
¹⁸ terms of the quality of point estimates, the double ML estimators can sometimes underperform their
¹⁹ more naive counterparts [KS07]. In fact, there are cases where each of outcome regression, propensity
²⁰ weighting, or doubly robust methods will outperform the others.

²¹ One difficulty in choosing an estimator in practice is that there are fewer guardrails in causal
²² inference than there are in standard predictive modeling. In predictive modeling, we construct a
²³ train-test split and validate our prediction models using the true labels or outcomes in the held-out
²⁴ dataset. However, for causal problems, the causal estimands are functionals of a different data-
²⁵ generating process from the one that we actually observed. As a result, it is impossible to empirically
²⁶ validate many aspects of causal estimation using standard techniques.

²⁷ The effectiveness of a given approach is often determined by how much we trust the specification of
²⁸ our propensity score or outcome regression models $\hat{g}(x)$ and $\hat{Q}(a, x)$, and how well the treatment and
²⁹ control groups overlap in the dataset. Using flexible models for the nuisance functions g and Q can
³⁰ alleviate some of the concerns about model misspecification, but our freedom to use such models is
³¹ often constrained by dataset size. When we have the luxury of large data, we can use flexible models;
³² on the other hand, when the dataset is relatively small, we may need to use a smaller parametric
³³ family or stringent regularization to obtain stable estimates of Q and g . Similarly, if overlap is poor
³⁴ in some regions of the covariate space, then flexible models for Q may be highly variable, and inverse
³⁵ propensity score weights may be large. In these cases, IPTW or AIPTW estimates may fluctuate
³⁶ wildly as a function of large weights. Meanwhile, outcome regression estimates will be sensitive to
³⁷ the specification of the Q model and its regularization, and can incur bias that is difficult to measure
³⁸ if the specification or regularization does not match the true outcome process.

³⁹ There are a number of practical steps that we can take to sanity-check causal estimates. The
⁴⁰ simplest check is to compute many different ATE estimators (e.g., outcome regression, IPTW, doubly
⁴¹ robust) using several comparably complex estimators of Q and g . We can then check whether they
⁴² agree, at least qualitatively. If they do agree then this can provide some peace of mind (although it
⁴³ is not a guarantee of accuracy). If they disagree, caution is warranted, particularly in choosing the
⁴⁴ specification of the Q and g models.

⁴⁵ It is also important to check for failures of overlap. Often, issues such as disagreement between
⁴⁶ alternative estimators can be traced back to poor overlap. A common way to do this, particularly
⁴⁷

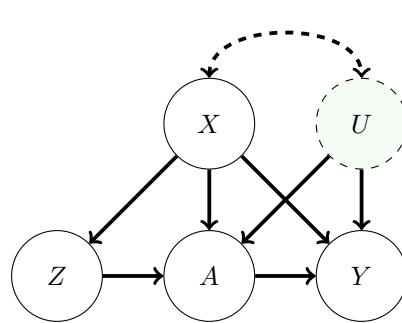


Figure 38.6: Causal graph illustrating the Instrumental Variable setup. The treatment A and outcome Y are both influenced by unobserved confounder U . Nevertheless, identification is sometimes possible due to the presence of the instrument Z . We also allow for observed covariates X that we may need to adjust for. The dashed arrow between U and X indicates a statistical dependency where we remain agnostic to the particular causal relationship.

with high-dimensional data, is to examine the estimated (ideally cross-fitted) propensity scores $\hat{g}(x_i)$. This is a useful diagnostic, even if the intention is to use an outcome regression approach that only incorporates and estimated outcome regression function $\hat{Q}(a, x_i)$. If overlap issues are relevant, it may be better to instead estimate either the average treatment effect on the treated, or the “trimmed” estimand given by discarding units with extreme propensities.

Uncertainty quantification is also an essential part of most causal analyses. This frequently take the form of an estimate of the estimator’s variance, or a confidence interval. This may be important for downstream decision-making, and can also be a useful diagnostic. We can calculate variance either by bootstrapping the entire procedure (including refitting the models in each bootstrap replicate), or computing analytical variance estimates from the AIPTW estimator. Generally, large variance estimates may indicate issues with the analysis. For example, poor overlap will often (although not always) manifest as extremely large variances under either of these methods. Small variance estimates should be treated with caution, unless other checks, such as overlap checks, or stability across different Q and g models, also pass.

The previous advice only addresses the statistical problem of estimating τ from a data sample. It does not speak to whether or not τ can reasonably be interpreted as an average treatment effect. Considerable care should be devoted to whether or not the assumption that there are no unobserved confounders is reasonable. There are several methods for assessing the sensitivity of the ATE estimate to violations of this assumption. See Section 38.7. Bias due to unobserved confounding can be substantial in practice—often overwhelming bias due to estimation error—so it is wise to conduct such an analysis.

38.5 Instrumental Variable Strategies

Adjustment-based methods rely on observing all confounders affecting the treatment and outcome. In some situations, it is possible to identify interesting causal effects even when there are unobserved confounders. We now consider strategies based on **instrumental variables**. The instrumental

¹ variable graph is shown in Figure 38.6. The key ingredient is the instrumental variable Z , a variable
² that has a causal effect on Y only through its causal effect on A . Informally, the identification
³ strategy is to determine to the causal effect of Z on Y , the causal effect of Z on A , and then combine
⁴ these into an estimate of the causal effect of A on Y .

⁵ For this identification strategy to work the instrument must satisfy three conditions. There are
⁶ observed variables (confounders) X such that:

- ⁷ **1. Instrument Relevance** $Z \not\perp\!\!\!\perp A | X$: the instrument must actually affect the treatment assignment.
- ⁸ **2. Instrument Unconfoundedness** Any backdoor path between Z and Y is blocked by X , even
⁹ conditional on A .
- ¹⁰ **3. Exclusion Restriction** All directed paths from Z to Y pass through A . That is, the instrument
¹¹ affects the outcome *only* through its effect on A .

¹²(It may help conceptually to first think through the case where X is the empty set—i.e., where
¹³the only confounder is the unobserved U). These assumptions are necessary for using instrumental
¹⁴variables for causal identification, but they are not quite sufficient. In practice, they must be
¹⁵supplemented by an additional assumption that depends more closely on the details of the problem
¹⁶at hand. Historically, this additional assumption was usually that both the instrument-treatment and
¹⁷treatment-outcome relationship are linear. We'll examine some less restrictive alternatives below.

¹⁸ Before moving on to how to use instrumental variables for identification, let's consider how we
¹⁹might encounter instruments in practice. The key is that it's often possible to find, and measure,
²⁰variables that affect treatment and that are assigned (as if) at random. For example, suppose we are
²¹interested in measuring the effect of taking a drug A on some health outcome Y . The challenge is that
²²whether a study participant actually takes the drug can be confounded with Y —e.g., sicker people
²³may be more likely to take their medication, but have worse outcomes. However, the assignment
²⁴of treatments to patients can be randomized and this random assignment can be viewed as an
²⁵instrument. This **random assignment with non-compliance** scenario is common in practice.
²⁶The random assignment—the instrument—satisfies relevance (so long as assigning the drug affects
²⁷the probability of the patient taking the drug). It also satisfies unconfoundedness (because the
²⁸instrument is randomized). And, it plausibly satisfies exclusion restriction: telling (or not telling)
²⁹a patient to take a drug has no effect on their health outcome except through influencing whether
³⁰they actually take the drug. As a second example, the **judge fixed effects** research design
³¹uses the identity of the judge assigned to each criminal case to infer the effect of incarceration on
³²some life outcome of interest (e.g., total lifetime earnings). Relevance will be satisfied so long as
³³different judges have different propensities to hand out severe sentences. The assignment of trial
³⁴judges to cases is randomized, so unconfoundedness will also be satisfied. And, exclusion restriction
³⁵is also plausible: the particular identity of the judge assigned to your case has no bearing on your
³⁶years-later life outcomes, except through the particular sentence that you're subjected to.

³⁷ It's important to note that these assumptions require some care, particularly exclusion restriction.
³⁸Relevance can be checked directly from the data, by fitting a model to predict the treatment from the
³⁹instrument (or vice versa). Unconfoundedness is often satisfied by design: the instrument is randomly
⁴⁰assigned. Even when literal random assignment doesn't hold, we often restrict to instruments
⁴¹where unconfoundedness is "obviously" satisfied—e.g., using number of rainy days in a month as
⁴²an instrument for sun exposure. Exclusion restriction is trickier. For example, it might fail in the
⁴³drug assignment case if patients who are not told to take a drug respond by seeking out alternative
⁴⁴therapies.

treatment. Or, it might fail in the judge fixed effects case if judges hand out additional, unrecorded, punishments in addition to incarceration. Assessing the plausibility of exclusion restriction requires careful consideration based on domain expertise.

We now return to the question of how to make use of an instrument once we have it in hand. As previously mentioned, getting causal identification using instrumental variables requires supplementing the IV assumptions with some additional assumption about the causal process.

38.5.1 Additive Unobserved Confounding

We first consider **additive unobserved confounding**. That is, we assume that the structural causal model for the outcome has the form:⁷

$$Y \leftarrow f(A, X) + f_U(U). \quad (38.49)$$

In words, we assume that there are no interaction effects between the treatment and the unobserved confounder—everyone responds to treatment in the same way. With this additional assumption, we see that $\mathbb{E}[Y|X, \text{do}(A = a)] - \mathbb{E}[Y|X, \text{do}(A = a')] = f(a, X) - f(a', X)$. In this setting, our goal is to learn this contrast.

Theorem 3 (Additive Confounding Identification). *If the instrumental variables assumptions hold and also additive unobserved confounding holds, then there is a function $\tilde{f}(a, x)$ where*

$$\mathbb{E}[Y|x, \text{do}(A = a)] - \mathbb{E}[Y|x, \text{do}(A = a')] = \tilde{f}(a, x) - \tilde{f}(a', x), \quad (38.50)$$

for all x, a, a' and such that \tilde{f} satisfies

$$\mathbb{E}[Y|z, x] = \int \tilde{f}(a, x)p(a|z, x)da. \quad (38.51)$$

Here, $p(a|z, x)$ is the conditional probability density of treatment.

In particular, if there is a unique function g that satisfies

$$\mathbb{E}[Y|z, x] = \int g(a, x)p(a|z, x)da, \quad (38.52)$$

then $g = \tilde{f}$ and this relation identifies the target causal effect.

Before giving the proof, let's understand the point of this identification result. The key insight is that both the left hand side of Equation (38.52) and $p(a|z, x)$ (appearing in the integrand) are identified by the data, since they involve only observational relationships between observed variables. So, \tilde{f} is identified implicitly as one of the functions that makes Equation (38.52) true. If there is a unique such function, then this fully identifies the causal effect.

Proof. With the additive unobserved confounding assumption, the instrument unconfoundedness implies that $U \perp\!\!\!\perp Z|X$. Then, we have that:

$$\mathbb{E}[Y|Z, X] = \mathbb{E}[f(A, X)|Z, X] + \mathbb{E}[f_U(U)|Z, X] \quad (38.53)$$

$$= \mathbb{E}[f(A, X)|Z, X] + \mathbb{E}[f_U(U)|X] \quad (38.54)$$

$$= \mathbb{E}[\tilde{f}(A, X)|Z, X], \quad (38.55)$$

⁷ We roll the unit-specific variables ξ into U to avoid notational overload.

¹ where $\tilde{f} = f(A, X) + \mathbb{E}[f_U(U)|X]$. Now, identifying just \tilde{f} would suffice for us, because we could then
² identify contrasts between treatments: $f(a, x) - f(a', x) = \tilde{f}(a, x) - \tilde{f}(a', x)$. (The term $\mathbb{E}[f_U(U)|x]$
³ cancels out). Accordingly, we rewrite Equation (38.55) as:
⁴

⁵
$$\mathbb{E}[Y|z, x] = \int \tilde{f}(a, x)p(a|z, x)da. \quad (38.56)$$

⁶

□

⁷ It's worth dwelling briefly on how the IV assumptions come into play here. The exclusion restriction
⁸ is implied by the additive unobserved confounding assumption, which we use explicitly. We also use
⁹ the unconfoundedness assumption to conclude $U \perp\!\!\!\perp Z|X$. However, we do not use relevance. The
¹⁰ role of relevance here is in ensuring that few functions solve the relation Equation (38.52). Informally,
¹¹ the solution g is constrained by the requirement that it hold for all values of Z . However, different
¹² values of Z only add non-trivial constraints if $p(a|z, x)$ differ depending on the value of z —this is
¹³ exactly the relevance condition.
¹⁴

¹⁵ **Estimation** The basic estimation strategy is to fit models for $\mathbb{E}[Y|z, x]$ and $p(a|z, x)$ from the data,
¹⁶ and then solve the implicit equation Equation (38.52) to find g consistent with the fitted models.
¹⁷ The procedures for doing this can vary considerably depending on the particulars of the data (e.g., if
¹⁸ Z is discrete or continuous) and the choice of modeling strategy. We omit a detailed discussion, but
¹⁹ [see e.g. NP03; Dar+11; Har+17; SSG19; BKS19; Mua+20; Dik+20] for various concrete approaches.
²⁰

²¹ It's also worth mentioning an additional nuance to the general procedure. Even if relevance holds,
²² there will often be more than one function that satisfies Equation (38.52). So, we have only identified
²³ \tilde{f} as a member of this set of functions. In practice, this ambiguity is defeated by making some
²⁴ additional structural assumption about \tilde{f} . For example, we model \tilde{f} with a neural network, and then
²⁵ choose the network satisfying Equation (38.52) that has minimum l_2 -norm on the parameters (i.e.,
²⁶ we pick the l_2 -regularized solution).
²⁷

²⁸ 38.5.2 Instrument Monotonicity and Local Average Treatment Effect

²⁹ We now consider an alternative assumption to additive unobserved confounding that is applicable
³⁰ when both the instrument and treatment are binary. It will be convenient to conceptualize the
³¹ instrument as assignment-to-treatment. Then, the population divides into four subpopulations:
³²

- ³³ 1. Compliers, who take the treatment if assigned to it, and who don't take the treatment otherwise.
- ³⁴ 2. Always takers, who take the treatment no matter their assignment
- ³⁵ 3. Never takers, who refuse the treatment no matter their assignment
- ³⁶ 4. Defiers, who refuse the treatment if assigned to it, and who take the treatment if not assigned.

³⁷ Our goal in this setting will be to identify the average treatment effect among the compliers. The
³⁸ **local average treatment effect** (or **complier average treatment effect**) is defined to be⁸

³⁹
$$\text{LATE} = \mathbb{E}[Y|\text{do}(A = 1), \text{complier}] - \mathbb{E}[Y|\text{do}(A = 0), \text{complier}]. \quad (38.57)$$

⁴⁰

⁴¹ 8. We follow the econometrics literature in using "LATE" because "CATE" is already commonly used for conditional
⁴² average treatment effect.
⁴³

The LATE requires an additional assumption for identification. Namely, **instrument monotonicity**: being assigned (not assigned) the treatment only increases (decreases) the probability that each unit will take the treatment. Equivalently, $P(\text{defier}) = 0$.

We can then write down the identification result.

Theorem 4. *Given the instrumental variable assumptions and instrument monotonicity, the local average treatment is identified as a parameter τ^{LATE} of the observational distributional; that is, $\text{LATE} = \tau^{\text{LATE}}$. Namely,*

$$\tau^{\text{LATE}} = \frac{\mathbb{E}[\mathbb{E}[Y|X, Z = 1] - \mathbb{E}[Y|X, Z = 0]]}{\mathbb{E}[P(A = 1|X, Z = 1) - P(A = 1|X, Z = 0)]}. \quad (38.58)$$

Proof. We now show that, given the IV assumptions and monotonicity, $\text{LATE} = \tau^{\text{LATE}}$. First, notice that

$$\tau^{\text{LATE}} = \frac{\mathbb{E}[Y|\text{do}(Z = 1)] - \mathbb{E}[Y|\text{do}(Z = 0)]}{P(A = 1|\text{do}(Z = 1)) - P(A = 1|\text{do}(Z = 0))}. \quad (38.59)$$

This follows from backdoor adjustment, Theorem 2, applied to the numerator and denominator separately. Our strategy will be to decompose $\mathbb{E}[Y|\text{do}(Z = z)]$ into the contributions from the compliers, the units that ignore the instrument (the always/never takers), and the defiers. To that end, note that $P(\text{complier}|\text{do}(Z = z)) = P(\text{complier})$ and similarly for always/never takers and defiers—interventions on the instrument don’t change the composition of the population. Then,

$$\mathbb{E}[Y|\text{do}(Z = 1)] - \mathbb{E}[Y|\text{do}(Z = 0)] \quad (38.60)$$

$$= (\mathbb{E}[Y|\text{complier}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{complier}, \text{do}(Z = 0)])P(\text{complier}) \quad (38.61)$$

$$+ (\mathbb{E}[Y|\text{always/never}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{always/never}, \text{do}(Z = 0)])P(\text{always/never}) \quad (38.62)$$

$$+ (\mathbb{E}[Y|\text{defier}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{defier}, \text{do}(Z = 0)])P(\text{defier}). \quad (38.63)$$

The key is the effect on the complier subpopulation, Equation (38.61). First, by definition of the complier population, we have that:

$$\mathbb{E}[Y|\text{complier}, \text{do}(Z = z)] = \mathbb{E}[Y|\text{complier}, \text{do}(A = z)]. \quad (38.64)$$

That is, the causal effect of the treatment is the same as the causal effect of the instrument in this subpopulation—this is the core reason why access to an instrument allows identification of the local average treatment effect. This means that

$$\text{LATE} = \mathbb{E}[Y|\text{complier}, \text{do}(Z = 1)] - \mathbb{E}[Y|\text{complier}, \text{do}(Z = 0)]. \quad (38.65)$$

Further, we have that $P(\text{complier}) = P(A = 1|\text{do}(Z = 1)) - P(A = 1|\text{do}(Z = 0))$. The reason is simply that, by definition of the subpopulations,

$$P(A = 1|\text{do}(Z = 1)) = P(\text{complier}) + P(\text{always taker}) \quad (38.66)$$

$$P(A = 1|\text{do}(Z = 0)) = P(\text{always taker}). \quad (38.67)$$

¹ Now, plugging the expression for $P(\text{complier})$ and Equation (38.65) into Equation (38.61) we have
² that:

$$\begin{aligned} \text{ (38.68)} \\ (\mathbb{E}[Y|\text{complier}, \text{do}(Z=1)] - \mathbb{E}[Y|\text{complier}, \text{do}(Z=0)])P(\text{complier}) \end{aligned}$$

$$\begin{aligned} \text{ (38.69)} \\ = \text{LATE} \times (P(A=1|\text{do}(Z=1)) - P(A=1|\text{do}(Z=0))) \end{aligned}$$

⁷ This gives us an expression for the local average treatment effect in terms of the effect of the instrument
⁸ on the compliers and the probability that a unit takes the treatment when assigned/not-assigned.

¹⁰ The next step is to show that the remaining instrument effect decomposition terms, Equations
¹¹ (38.62) and (38.63), are both 0. Equation (38.62) is the causal effect of the instrument on the
¹² always/never takers. It's equal to 0 because, by definition of this subpopulation, the instrument
¹³ has no causal effect in the subpopulation—they ignore the instrument! Mathematically, this is just
¹⁴ $\mathbb{E}[Y|\text{always/never}, \text{do}(Z=1)] = \mathbb{E}[Y|\text{always/never}, \text{do}(Z=0)]$. Finally, Equation (38.63) is 0 by the
¹⁵ instrument monotonicity assumption: we assumed that $P(\text{defier}) = 0$.

¹⁶ In totality, we now have that Equations (38.61) to (38.63) reduces to:

$$\begin{aligned} \text{ (38.70)} \\ \mathbb{E}[Y|\text{do}(Z=1)] - \mathbb{E}[Y|\text{do}(Z=0)] \end{aligned}$$

$$\begin{aligned} \text{ (38.71)} \\ = \text{LATE} \times (P(A=1|\text{do}(Z=1)) - P(A=1|\text{do}(Z=0))) + 0 + 0 \end{aligned}$$

²⁰ Rearranging for LATE and plugging in to Equation (38.59) gives claimed identification result. \square
²¹

²² 38.5.2.1 Estimation

²⁴ For estimating the local average treatment effect under the monotone instrument assumption, there
²⁵ is a double-machine learning approach that works with generic supervised learning approaches. Here,
²⁶ we want an estimator $\hat{\tau}^{\text{LATE}}$ for the parameter

$$\begin{aligned} \text{ (38.72)} \\ \tau^{\text{LATE}} = \frac{\mathbb{E}[\mathbb{E}[Y|X, Z=1] - \mathbb{E}[Y|X, Z=0]]}{\mathbb{E}[P(A=1|X, Z=1) - P(A=1|X, Z=0)]}. \end{aligned}$$

³⁰ To define the estimator, it's convenient to introduce some additional notation. First, we define the
³¹ nuisance functions:

$$\begin{aligned} \text{ (38.73)} \\ \mu(z, x) = \mathbb{E}[Y|z, x] \end{aligned}$$

$$\begin{aligned} \text{ (38.74)} \\ m(z, x) = P(A=1|x, z) \end{aligned}$$

$$\begin{aligned} \text{ (38.75)} \\ p(x) = P(Z=1|x). \end{aligned}$$

³⁷ We also define the score ϕ by:

$$\begin{aligned} \text{ (38.76)} \\ \phi_{Z \rightarrow Y}(\mathbf{X}; \mu, p) \triangleq \mu(1, X) - \mu(0, X) + \frac{Z(Y - \mu(1, X))}{p(X)} - \frac{(1-Z)(Y - \mu(0, X))}{1-p(X)} \end{aligned}$$

$$\begin{aligned} \text{ (38.77)} \\ \phi_{Z \rightarrow A}(\mathbf{X}; m, p) \triangleq m(1, X) - m(0, X) + \frac{Z(A - m(1, X))}{p(X)} - \frac{(1-Z)(A - m(0, X))}{1-p(X)} \end{aligned}$$

$$\begin{aligned} \text{ (38.78)} \\ \phi(\mathbf{X}; \mu, m, p, \tau) \triangleq \phi_{Z \rightarrow Y}(\mathbf{X}; \mu, p) - \phi_{Z \rightarrow A}(\mathbf{X}; m, p) \times \tau \end{aligned}$$

⁴⁴ Then, the estimator is defined by a two stage procedure:

⁴⁵

- 1 1. Fit models $\hat{\mu}, \hat{m}, \hat{p}$ for each of μ, m, p (using supervised machine learning).
- 2 2. Define $\hat{\tau}^{\text{LATE}}$ as the solution to $\frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{\mu}, \hat{m}, \hat{p}, \hat{\tau}^{\text{LATE}}) = 0$. That is,

$$\hat{\tau}^{\text{LATE}} = \frac{\frac{1}{n} \sum_i \phi_{Z \rightarrow Y}(\mathbf{X}_i; \hat{\mu}, \hat{p})}{\frac{1}{n} \sum_i \phi_{Z \rightarrow A}(\mathbf{X}_i; \hat{m}, \hat{p})} \quad (38.79)$$

8 It may help intuitions to notice that the double machine learning estimator of the LATE is effectively
9 the double machine learning estimator of the average treatment effect of Z on Y divided by the
10 double machine learning estimator of the average treatment effect of Z on A .

11 Similarly to Section 38.4, the nuisance functions can be estimated by:

- 12 1. fit a model $\hat{\mu}$ that predicts Y from Z, X by minimizing mean square error
- 13 2. fit a model \hat{m} that predicts A from Z, X by minimizing mean cross-entropy
- 14 3. fit a model \hat{p} that predicts Z from X by minimizing mean cross-entropy.

17 As in Section 38.4, reusing the same data for model fitting and computing the estimator can
18 potentially cause problems. This can be avoided with use a cross-fitting procedure as described in
19 Section 38.4.2.4. In this case, we split the data into K folds and, for each fold k , use all the but
20 the k th fold to compute estimates $\hat{\mu}_{-k}, \hat{m}_{-k}, \hat{p}_{-k}$ of the nuisance parameters. Then we compute
21 the nuisance estimates for each datapoint i in fold k by predicting the required quantity using the
22 nuisance model fit on the other folds. That is, if unit i is in fold k , we compute $\hat{\mu}(z_i, x_i) \triangleq \hat{\mu}_{-k}(z_i, x_i)$
23 and so forth.

24 The key result is that if we use the cross-fit version of the estimator and the estimators for the
25 nuisance functions converge to their true values in the sense that

- 27 1. $\mathbb{E}(\hat{\mu}(Z, X) - \mu(Z, X))^2 \rightarrow 0$, $\mathbb{E}(\hat{m}(Z, X) - m(Z, X))^2 \rightarrow 0$, and $\mathbb{E}(\hat{p}(X) - p(X))^2 \rightarrow 0$
 - 28 2. $\sqrt{\mathbb{E}[(\hat{p}(X) - p(X))^2]} \times (\sqrt{\mathbb{E}[(\hat{\mu}(Z, X) - \mu(Z, X))^2]} + \sqrt{\mathbb{E}[(\hat{m}(Z, X) - m(Z, X))^2]}) = o(\sqrt{n})$
- 30 then (with some omitted technical conditions) we have asymptotic normality at the \sqrt{n} -rate:

$$\sqrt{n}(\hat{\tau}^{\text{LATE}-\text{cf}} - \tau^{\text{LATE}}) \xrightarrow{d} \text{Normal}(0, \frac{\mathbb{E}[\phi(\mathbf{X}; \mu, m, p, \tau^{\text{LATE}})^2]}{\mathbb{E}[m(1, X) - m(0, X)]^2}). \quad (38.80)$$

34 As with double machine learning for the confounder adjustment strategy, the key point here is that
35 we can achieve the (optimal) \sqrt{n} rate for estimating the LATE under a relatively weak condition on
36 how well we estimate the nuisance functions—what matters is the *product* of the error in p and the
37 errors in μ, m . So, for example, a very good model for how the instrument is assigned (p) can make
38 up for errors in the estimation of the treatment-assignment (m) and outcome (μ) models.

39 The double machine learning estimator also gives a recipe for quantifying uncertainty. To that
40 end, define

$$\hat{\tau}_{Z \rightarrow A} \triangleq \frac{1}{n} \sum_i \phi_{Z \rightarrow A}(\mathbf{X}_i; \hat{m}, \hat{p}) \quad (38.81)$$

$$\hat{\mathbb{V}}[\hat{\tau}^{\text{LATE}}] \triangleq \frac{1}{\hat{\tau}_{Z \rightarrow A}^2} \frac{1}{n} \sum_i \phi(\mathbf{X}_i; \hat{\mu}, \hat{m}, \hat{p}, \hat{\tau}^{\text{LATE}})^2. \quad (38.82)$$

¹ Then, subject to suitable technical conditions, $\hat{V}[\hat{\tau}^{\text{LATE}-\text{cf}}]$ can be used as an estimate of the variance
² of the estimator. More precisely,

$$\sqrt{n}(\hat{\tau}^{\text{LATE}} - \tau^{\text{LATE}}) \xrightarrow{d} \text{Normal}(0, \hat{V}[\hat{\tau}^{\text{LATE}}]). \quad (38.83)$$

⁶ Then, confidence intervals or p -values can be computed using this variance in the usual way. The main
⁷ extra condition required for the variance estimator to be valid is that the nuisance parameters must
⁸ all converge at rate $O(n^{-1/4})$ (so an excellent estimator for one can't fully compensate for terrible
⁹ estimators of the others). In fact, even this condition is unnecessary in certain special cases—e.g.,
¹⁰ when p is known exactly, which occurs when the instrument is randomly assigned. See Chernozhukov
¹¹ et al. [Che+17e] for technical details.

¹³ ¹⁴ 38.5.3 Two Stage Least Squares

¹⁵ Commonly, the IV assumptions are supplemented with the following linear model assumptions:

$$A_i \leftarrow \alpha_0 + \alpha Z_i + \delta_A X_i + \gamma_A X_i + \xi_i^A \quad (38.84)$$

$$Y_i \leftarrow \beta_0 + \beta A_i + \delta_Y X_i + \gamma_Y X_i + \xi_i^Y \quad (38.85)$$

²⁰ That is, we assume that the real-world process for treatment assignment and the outcome are both
²¹ linear. In this case, plugging Equation (38.84) into Equation (38.85) yields

$$Y_i \leftarrow \tilde{\beta}_0 + \beta \alpha Z_i + \tilde{\delta} X_i + \tilde{\gamma} X_i + \tilde{\xi}_i. \quad (38.86)$$

²⁴ The point is that β , the average treatment effect of A on Y , is equal to the coefficient $\beta \alpha$ of the
²⁵ instrument in the outcome-instrument model divided by the coefficient α of the instrument in the
²⁶ treatment-instrument model. So, to estimate the treatment effect, we simply fit both linear models
²⁷ and divide the estimated coefficients. This procedure is called **two stage least squares**.

²⁹ The simplicity of this procedure is seductive. However, the required linearity assumptions are hard
³⁰ to satisfy in practice and frequently lead to severe issues. A particularly pernicious version of this
³¹ is that linear-model misspecification together with weak relevance can yield standard errors for the
³² estimate that are far too small. In practice, this can lead us to find large, significant estimates from
³³ two stage least squares when the truth is actually a weak or null effect. See [Rei16; You19; ASS19;
³⁴ Lal+21] for critical evaluations of two stage least squares in practice.

³⁵ ³⁶ 38.6 Difference in Differences

³⁸ Unsurprisingly, time plays an important role in causality. Causes precede effects, and we should be
³⁹ able to incorporate this knowledge into causal identification. We now turn to a particular strategy
⁴⁰ for causal identification that relies on observing each unit at multiple time points. Data of this kind
⁴¹ is sometimes called **panel data**. We'll consider the simplest case. There are two time periods. In
⁴² the first period, none of the units are treated, and we observe an outcome Y_{0i} for each unit. Then,
⁴³ a subset of the units are treated, denoted by $A_i = 1$. In the second time period, we again observe
⁴⁴ the outcomes Y_{1i} for each unit, where now the outcomes of the treated units are affected by the
⁴⁵ treatment. Our goal is to determine the average effect receiving the treatment had on the treated
⁴⁶ units. That is, we want to know the average difference between the outcomes we actually observed
⁴⁷

for the treated units, and the outcomes we would have observed on those same units if they had not been treated. The general strategy we look at is called **difference in differences**.

As a concrete motivating example, consider trying to determine the effect raising minimum wage on employment. The concern here is that, in an efficient labor market, increasing the price of workers will reduce the demand for them, thereby driving down employment. As such, it seems increasing minimum wage may hurt the people the policy is nominally intended to help. The question is: how strong is this effect in practice? Card and Krueger [CK94a] studied this effect using difference in differences. The Philadelphia metropolitan area includes regions in both Pennsylvania and New Jersey (different US states). On April 1st 1992, New Jersey raised its minimum wage from \$4.25 to \$5.05. In Pennsylvania, the wage remained constant at \$4.25. The strategy is to collect employment data from fast food restaurants (which pay many employees minimum wage) in each state before and after the change in minimum wage. In this case, for restaurant i , we have Y_{0i} , the number of full time employees in February 1992, and Y_{1i} , the number of full time employees in November 1992. The treatment is simply $A_i = 1$ if the restaurant was located in New Jersey, and $A_i = 0$ if located in Pennsylvania. Our goal is to estimate the average effect of the minimum wage hike on employment in the restaurants affected by it (i.e., the ones in New Jersey).

The assumption in classical difference-in-differences is the following structural equation:

$$Y_{ti} \leftarrow W_i + S_t + \tau A_i 1[t = 1] + \xi_{ti}, \quad (38.87)$$

with $\mathbb{E}[\xi_{ti}|W_i, S_t, A_i] = 0$. Here, W_i is a unit specific effect that is constant across time (e.g., the location of the restaurant or competence of the management) and S_t is a time-specific effect that applies to all units (e.g., the state of the US economy at each time). Both of these quantities are treated as unobserved, and not explicitly accounted for. The parameter τ captures the target causal effect. The (strong) assumption here is that unit, time, and treatment effects are all additive. This assumption is called **parallel trends**, because it is equivalent to assuming that, in the absence of treatment, the trend over time would be the same in both groups. It's easy to see that under this assumption, we have:

$$\tau = \mathbb{E}[Y_{1i} - Y_{0i}|A = 1] - \mathbb{E}[Y_{1i} - Y_{0i}|A = 0]. \quad (38.88)$$

That is, the estimand first computes the difference across time for both the treated and untreated group, and then computes the difference between these differences across the groups. The obvious estimator is then

$$\hat{\tau} = \frac{1}{n_A} \sum_{i:A_i=1} Y_{1i} - Y_{0i} - \frac{1}{n - n_A} \sum_{i:A_i=0} Y_{1i} - Y_{0i}, \quad (38.89)$$

where n_A is the number of treated units.

The root identification problem addressed by difference-in-differences is that $\mathbb{E}[W_i|A_i = 1] \neq \mathbb{E}[W_i|A_i = 0]$. That is, restaurants in New Jersey may be systematically different from restaurants in Pennsylvania in unobserved ways that affect employment.⁹ This is why we can't simply compare average outcomes for the treated and untreated. The identification assumption is that this unit-specific effect is the only source of statistical association with treatment; in particular we assume the

⁹ This is similar to the issue that arises from unobserved confounding, except W_i need not be a cause of the treatment assignment.

¹ time-specific effect has no such issue: $\mathbb{E}[S_{1i} - S_{0i}|A_i = 1] = \mathbb{E}[S_{1i} - S_{0i}|A_i = 0]$. Unfortunately, this
² assumption can be too strong. For instance, administrative data shows employment in Pennsylvania
³ falling relative to employment in New Jersey between 1993 and 1996 [AP08, §5.2]. Although this
⁴ doesn't directly contradict the parallel trends assumption used for identification, which needs to
⁵ hold only in 1992, it does make it seem less credible.

⁶ To weaken the assumption, we'll look at a version that requires parallel trends to hold only after
⁷ adjusting for covariates. To motivate this, we note that there were several different types of fast
⁸ food restaurant included in the employment data. These vary, e.g., in the type of food they serve,
⁹ and in cost per meal. Now, it seems reasonable the trend in employment may depend on the type
¹⁰ of restaurant. For example, more expensive chains (such as Kentucky Fried Chicken) might be
¹¹ more affected by recessions than cheaper chains (such as McDonald's). If expensive chains are more
¹² common in New Jersey than in Pennsylvania, this effect can create a violation of parallel trends—if
¹³ there's recession affecting both states, we'd expect employment to go down more in New Jersey than
¹⁴ in Pennsylvania. However, we may find it credible that McDonald's restaurants in New Jersey have
¹⁵ the same trend as McDonald's in Pennsylvania, and similarly for Kentucky Fried Chicken.

¹⁶ The next step is to give a definition of the target causal effect that doesn't depend on a parametric
¹⁷ model, and a non-parametric statement of the identification assumption to go with it. In words, the
¹⁸ causal estimand will be the average treatment effect on the units that received the treatment. To
¹⁹ make sense of this mathematically, we'll introduce a new piece of notation:

$$\begin{aligned} \text{P}^{A=1}(Y|\text{do}(A=a)) &\triangleq \int P(Y|A=a, \text{parents of } Y)dP(\text{parents of } Y|A=1) \end{aligned} \quad (38.90)$$

$$\mathbb{E}^{A=1}[Y|\text{do}(A=a)] \triangleq \mathbb{E}_{\text{P}^{A=1}(Y|\text{do}(A=a))}[Y]. \quad (38.91)$$

²⁰ In words: recall that the ordinary do operator works by replacing $P(\text{parents}|A=a)$ by the marginal
²¹ distribution $P(\text{parents})$, thereby breaking the backdoor associations. Now, we're replacing the
²² distribution $P(\text{parents}|A=a)$ by $P(\text{parents}|A=1)$, irrespective of the actual treatment value. This
²³ still breaks all backdoor associations, but is a better match for our target of estimating the treatment
²⁴ effect only among the treated units.

²⁵ To formalize a causal estimand using the do calculus, we need to assume some partial causal
²⁶ structure. We'll use the graph in Figure 38.7. With this in hand, our causal estimand is the average
²⁷ treatment effect on the units that received the treatment, namely:

$$\text{ATT}^{\text{DiD}} = \mathbb{E}^{A=1}[Y_1 - Y_0|\text{do}(A=1)] - \mathbb{E}^{A=1}[Y_1 - Y_0|\text{do}(A=0)] \quad (38.92)$$

²⁸ In the minimum wage example, this is the average effect of the minimum wage hike on employment
²⁹ in the restaurants affected by it (i.e., the ones in New Jersey).

³⁰ Finally, we formalize the identification assumption that, conditional on X , the trends in the treated
³¹ and untreated groups are the same. The **conditional parallel trends** assumption is:

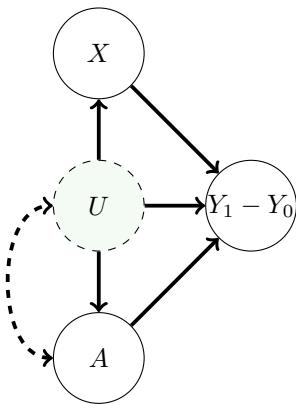
$$\mathbb{E}^{A=1}[Y_1 - Y_0|X, \text{do}(A=0)] = \mathbb{E}[Y_1 - Y_0|X, A=0]. \quad (38.93)$$

³² In words, this says that for treated units with covariates X , the trend we would have seen had we not
³³ assigned treatment is the same as the trend we actually saw for the untreated units with covariates
³⁴ X . That is, if New Jersey had not raised its minimum wage, then McDonald's in New Jersey would
³⁵ have the same expected change in employment as McDonald's in Pennsylvania.

³⁶ With this in hand, we can give the main identification result:

³⁷

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



16 *Figure 38.7: Causal graph assumed for the difference-in-differences setting. Here, the outcome of interest
17 is the difference between the pre- and post-treatment period, $Y_1 - Y_0$. This difference is influenced by the
18 treatment, unobserved factors U , and observed covariates X . The dashed arrow between U and A indicates a
19 statistical dependency between the variables, but where we remain agnostic to the precise causal mechanism.
20 For example, in the minimum wage example, U might be the average income in restaurant's neighbourhood,
21 which is dependent on the state, and hence also the treatment.*

22

23 **Theorem 5** (Difference in Differences Identification). *We observe $A, Y_0, Y_1, X \sim P$. Suppose that*

24
25
26

1. *(Causal Structure) The data follows the causal graph in Figure 38.7.*

27
28

2. *(Conditional Parallel Trends) $\mathbb{E}^{A=1}[Y_1 - Y_0|X, \text{do}(A = 0)] = \mathbb{E}[Y_1 - Y_0|X, A = 0]$.*

29
30

3. *(Overlap) $P(A = 1) > 0$ and $P(A = 1|X = x) < 1$ for all values of x in the sample space. That is,
there are no covariate values that only exist in the treated group.*

31
32

Then, the average treatment effect on the treated is identified as $\text{ATT}^{\text{DiD}} = \tau^{\text{DiD}}$, where

33
34

$$\tau^{\text{DiD}} = \mathbb{E}[\mathbb{E}[Y_1 - Y_0|A = 1, X] - \mathbb{E}[Y_1 - Y_0|A = 0, X]|A = 1]. \quad (38.94)$$

35
36

Proof. First, by unrolling definitions, we have that

37
38

$$\mathbb{E}^{A=1}[Y_1 - Y_0|\text{do}(A = 1), X] = \mathbb{E}[Y_1 - Y_0|A = 1, X]. \quad (38.95)$$

39
40
41

The interpretation is the near-tautology that the average effect among the treated under treatment is equal to the actually observed average effect among the treated. Next,

42
43

$$\mathbb{E}^{A=1}[Y_1 - Y_0|\text{do}(A = 0), X] = \mathbb{E}[Y_1 - Y_0|A = 0, X]. \quad (38.96)$$

44
45
46
47

is just the conditional parallel trends assumption. The result follows immediately.

(The overlap assumption is required to make sure all the conditional expectations are well defined). \square

¹
² **38.6.1 Estimation**

³ With the identification result in hand, the next task is to estimate the observational estimand
⁴ Equation (38.94). To that end, we define $\tilde{Y} \triangleq Y_1 - Y_0$. Then, we've assumed that $\tilde{Y}, X, A \stackrel{\text{iid}}{\sim} P$ for
⁵ some unknown distribution P , and our target estimand is $\mathbb{E}[\mathbb{E}[\tilde{Y}|A=1, X] - \mathbb{E}[\tilde{Y}|A=0, X]|A=1]$.
⁶ We can immediately recognize this as the observational estimand that occurs in estimating the
⁷ average treatment effect through adjustment, described in Section 38.4.5.3. That is, even though
⁸ the causal situation and the identification argument are different between the adjustment setting
⁹ and the difference in differences setting, the statistical estimation task we end up with is the same.
¹⁰ Accordingly, we can use all of the estimation tools we developed for adjustment. That is, all of the
¹¹ techniques there—expected outcome modeling, propensity score methods, double machine learning,
¹² and so forth—were purely about the *statistical* task, which is the same between the two scenarios.
¹³

¹⁴ So, we're left with the same general recipe for estimation we saw in Section 38.4.6. Namely,

- ¹⁵ 1. fit statistical or machine-learning models $\hat{Q}(a, x)$ as a predictor for $\tilde{Y} = Y_1 - Y_0$, and/or $\hat{g}(x)$ as a
¹⁶ predictor for A
- ¹⁷ 2. compute the predictions $\hat{Q}(0, x_i), \hat{Q}(1, x_i), \hat{g}(x_i)$ for each data point, and

¹⁸ 3. combine these predictions into an estimate of the average treatment effect on the treated.

¹⁹
²⁰ The estimator in the third step can be the expected outcome model estimator, the propensity weighted
²¹ estimator, the double machine learning estimator, or any other strategy that's valid in the adjustment
²² setting.
²³

²⁴
²⁵ **38.7 Credibility Checks**

²⁶
²⁷ Once we've chosen an identification strategy, fit our models, and produced an estimate, we're faced
²⁸ with a basic question: should we believe it? Whether the reported estimate succeeds in capturing
²⁹ the true causal effect depends on whether the assumptions required for causal identification hold, the
³⁰ quality of the machine learning models, and the variability in the estimate due to only having access
³¹ to a finite data sample. The latter two problems are already familiar from machine learning and
³² statistical practice. We should, e.g., assess our models by checking performance on held out data,
³³ examining feature importance, and so forth. Similarly, we should report measures of the uncertainty
³⁴ due to finite sample (e.g., in the form of confidence intervals). Because these procedures are already
³⁵ familiar practice, we will not dwell on them further. However, model evaluation and uncertainty
³⁶ quantification are key parts of any credible causal analysis.

³⁷ Assessing the validity of identification assumptions is trickier. First, there are assumptions that
³⁸ can in fact be checked from data. For example, overlap should be checked in analysis using backdoor
³⁹ adjustment or difference in differences, and relevance should be checked in the instrumental variable
⁴⁰ setting. Again, checking these conditions is absolutely necessary for a credible causal analysis. But,
⁴¹ again, this involves only familiar data analysis, so we will not discuss it further. Next, there are the
⁴² causal assumptions that cannot be verified from data; e.g., no unobserved confounding in backdoor
⁴³ adjustment, the exclusion restriction in IV, and conditional parallel trends in DiD. Ultimately, the
⁴⁴ validity of these assumptions must be assessed using substantive causal knowledge of the particular
⁴⁵ problem under consideration. However, it is possible to conduct some supplementary analyses that
⁴⁶ make the required judgement easier. We now discuss two such two such techniques.

⁴⁷

1 **38.7.1 Placebo Checks**

2
3 In many situations we may be able to find a variable that can be interpreted as a “treatment” that is
4 known to have no effect on the outcome, but which we expect to be confounded with the outcome in
5 a very similar fashion to the true treatment of interest. For example, if we’re trying to estimate the
6 efficacy of a COVID vaccine in preventing symptomatic COVID, we might take our placebo treatment
7 to be vaccination against HPV. We do not expect that there’s any causal effect here. However, it
8 seems plausible that latent factors that cause an individual to seek (or avoid) HPV vaccination and
9 COVID vaccination are similar; e.g., health conscientiousness, fear of needles, and so forth. Then, if
10 our identification strategy is valid for the COVID vaccine, we’d also expect it to be valid for
11 HPV vaccination. Accordingly, our estimation procedure we use for estimating the COVID effect
12 should, when applied to HPV, yield $\hat{\tau} \approx 0$. Or, more precisely, the confidence interval should contain
13 0. If this does not happen, then we may suspect that there are still some confounding factors lurking
14 that are not adequately handled by the identification procedure.

15 A similar procedure works when there is a variable that can be interpreted as an outcome which
16 is known to not be affected by the treatment, but that shares confounders with the outcome we’re
17 actually interested in. For example, in the COVID vaccination case, we might take the null outcome
18 to be symptomatic COVID within 7 days of vaccination [Dag+21]. Our knowledge of both the
19 biological mechanism of vaccination and the amount of time it takes to develop symptoms after
20 COVID infection (at least 2 days) lead us to conclude that it’s unlikely that the treatment has a
21 causal effect on the outcome. However, the properties of the treated people that affect how likely they
22 are to develop symptomatic COVID are largely the same in the 7 day and, e.g., 6 month window.
23 That includes factors such as risk aversion, baseline health, and so forth. Again, we can apply our
24 identification strategy to estimate the causal effect of the treatment on the null outcome. If the
25 confidence interval does not include 0, then we should doubt the credibility of the analysis.

27
28 **38.7.2 Sensitivity Analysis to Unobserved Confounding**

29
30 We now specialize to the case of estimating the average causal effect of a binary treatment by
31 adjusting for confounding variables, as described in Section 38.4. In this case, causal identification
32 is based on the assumption of ‘no unobserved confounding’; i.e., the assumption that the observed
33 covariates include all common causes of the treatment assignment and outcome. This assumption is
34 fundamentally untestable from observed data, but its violation can induce bias in the estimation of
35 the treatment effect—the unobserved confounding may completely or in part explain the observed
36 association. Our aim in this part is to develop a sensitivity analysis tool to aid in reasoning about
37 potential bias induced by unobserved confounding.

38 Intuitively, if we estimate a large positive effect then we might expect the real effect is also
39 positive, even in the presence of mild unobserved confounding. For example, consider the association
40 between smoking and lung cancer. One could argue that this association arises from a hormone
41 that both predisposes carriers to both an increased desire to smoke and to a greater risk of lung
42 cancer. However, the association between smoking and lung cancer is large—is it plausible that
43 some unknown hormonal association could have a strong enough influence to explain the association?
44 Cornfield et al. [Cor+59] showed that, for a particular observational dataset, such an unmeasured
45 hormone would need to increase the probability of smoking by at least a factor of nine. This is
46 an unreasonable effect size for a hormone, so they conclude it’s unlikely the causal effect can be
47

¹
2 explained away.

³ We would like a general procedure to allow domain experts to make judgments about whether
⁴ plausible confounding is “mild” relative to the “large” effect. In particular, the domain expert must
⁵ translate judgments about the strength of the unobserved confounding into judgments about the
⁶ bias induced in the estimate of the effect. Accordingly, we must formalize what is meant by strength
⁷ of unobserved confounding, and to show how to translate judgments about confounding strength into
⁸ judgments about bias.

⁹ A prototypical example, due to Imbens [Imb03] (building on [RR83]), illustrates the broad approach.
¹⁰ As above, the observed data consists of a treatment A , an outcome Y , and covariates X that may
¹¹ causally affect the treatment and outcome. Imbens [Imb03] then posits an additional unobserved
¹² binary confounder U for each patient, and supposes that the observed data and unobserved confounder
¹³ were generated according to the following assumption, known as **Imbens’ Sensitivity Model**:

$$\begin{aligned} \text{14} \quad U_i &\stackrel{\text{iid}}{\sim} \text{Bern}(1/2) \\ \text{15} \quad A_i | X_i, U_i &\stackrel{\text{ind}}{\sim} \text{Bern}(\text{sig}(\gamma X_i + \alpha U_i)) \end{aligned} \tag{38.97}$$

$$\begin{aligned} \text{16} \quad Y_i | X_i, A_i, U_i &\stackrel{\text{ind}}{\sim} \mathcal{N}(\tau A_i + \beta X_i + \delta U_i, \sigma^2). \\ \text{17} \quad \text{18} \quad \text{19} \end{aligned} \tag{38.98}$$

where sig is the sigmoid function.

²¹ If we had observed U_i , we could estimate $(\hat{\tau}, \hat{\gamma}, \hat{\beta}, \hat{\alpha}, \hat{\delta}, \hat{\sigma}^2)$ from the data and report $\hat{\tau}$ as the
²² estimate of the average treatment effect. Since U_i is not observed, it is not possible to identify
²³ the parameters from the data. Instead, we make (subjective) judgments about plausible values of
²⁴ α —how strongly U_i affects the treatment assignment—and δ —how strongly U_i affects the outcome.
²⁵ Contingent on plausible $\alpha = \alpha^*$ and $\delta = \delta^*$, the other parameters can be estimated. This yields an
²⁶ estimate of the treatment effect $\hat{\tau}(\alpha^*, \delta^*)$ under the presumed values of the sensitivity parameters.

²⁷ The approach just outlined has a major drawback: it relies on a parametric model for the full data
²⁸ generating process. The assumed model is equivalent to assuming that, had U been observed, it
²⁹ would have been appropriate to use logistic regression to model treatment assignment, and linear
³⁰ regression to model the outcome. This assumption also implies a simple, parametric model for the
³¹ relationships governing the observed data. This restriction is out of step with modern practice, where
³² we use flexible machine-learning methods to model these relationships. For example, the assumption
³³ forbids the use of neural networks or random forests, though such methods are often state-of-the-art
³⁴ for causal effect estimation.

³⁵
³⁶ **Austen plots** We now turn to developing an alternative an adaptation of Imbens’ approach that
³⁷ fully decouples sensitivity analysis and modeling of the observed data. Namely, the **Austen plots**
³⁸ of [VZ20]. An example Austen plot is shown in Figure 38.8. The high-level idea is to posit a
³⁹ generative model that uses a simple, interpretable parametric form for the influence of the unobserved
⁴⁰ confounder, but that *puts no constraints on the model for the observed data*. We then use the
⁴¹ parametric part of the model to formalize “confounding strength” and to compute the induced bias
⁴² as a function of the confounding.

⁴³ Austen plots further adapt two strategies pioneered by Imbens [Imb03]. First, we find a parametrization
⁴⁴ of the model so that the sensitivity parameters, measuring strength of confounding, are on
⁴⁵ a standardized, unitless scale. This allows us to compare the strength of hypothetical unobserved
⁴⁶ confounding to the strength of observed covariates, measured from data. Second, we plot the curve
⁴⁷

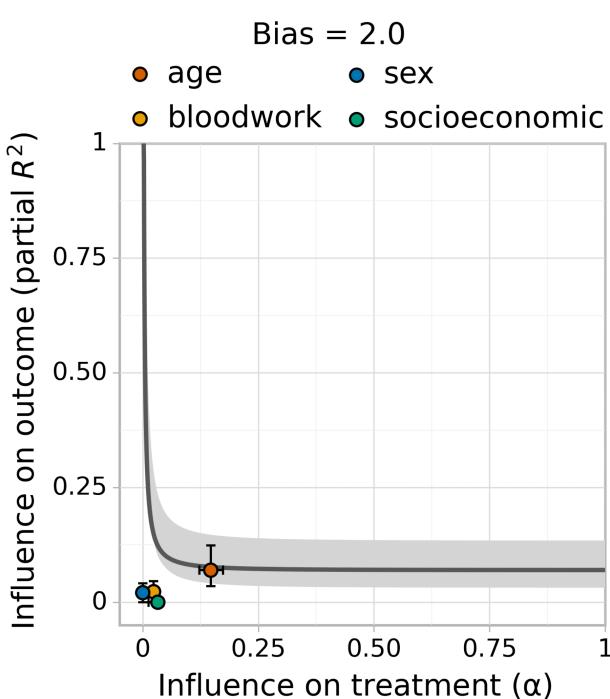


Figure 38.8: Austen plot showing how strong an unobserved confounder would need to be to induce a bias of 2 in an observational study of the effect of combination blood pressure medications on diastolic blood pressure [Dor+16]. We chose this bias to equal the nominal average treatment effect estimated from the data. We model the outcome with Bayesian Additive Regression Trees and the treatment assignment with logistic regression. The curve shows all values treatment and outcome influence that would induce a bias of 2. The colored dots show the influence strength of (groups of) observed covariates, given all other covariates. For example, an unobserved confounder with as much influence as the patient's age might induce a bias of about 2.

of all values of the sensitivity parameter that would yield given level of bias. This moves the analyst judgment from “what are plausible values of the sensitivity parameters?” to “are sensitivity parameters this extreme plausible?”

Figure 38.8, an Austen plot for an observational study of the effect of combination medications on diastolic blood pressure, illustrates the idea. A bias of 2 would suffice to undermine the qualitative conclusion that the blood-pressure treatment is effective. Examining the plot, an unobserved confounder as strong as age could induce this amount of confounding, but no other (group of) observed confounders has so much influence. Accordingly, if a domain expert thinks an unobserved confounder as strong as age is unlikely then they may conclude that the treatment is likely effective. Or, if such a confounder is plausible, they may conclude that the study fails to establish efficacy.

¹
² **Setup** The data are generated independently and identically $(Y_i, A_i, X_i, U_i) \stackrel{\text{iid}}{\sim} P$, where U_i is not
³ observed and P is some unknown probability distribution. The approach in Section 38.4 assumes that
⁴ the observed covariates X contain all common causes of Y and A . If this ‘no unobserved confounding’
⁵ assumption holds, then the ATE is equal to parameter, τ , of the observed data distribution, where
⁶
⁷
$$\tau = \mathbb{E}[\mathbb{E}[Y|X, A = 1] - \mathbb{E}[Y|X, A = 0]]. \quad (38.100)$$

⁸
⁹ This observational parameter is then estimated from a finite data sample. Recall from Section 38.4 that
¹⁰ this involves estimating the conditional expected outcome $Q(A, X) = \mathbb{E}[Y|A, X]$ and the propensity
¹¹ score $g(X) = P(A = 1|X)$, then plugging these into an estimator $\hat{\tau}$.

¹² We are now concerned with the case of possible unobserved confounding. That is, where U causally
¹³ affects Y and A . If there is unobserved confounding then the parameter τ is not equal to the
¹⁴ ATE, so $\hat{\tau}$ is a biased estimate. Inference about the ATE then divides into two tasks. First, the
¹⁵ statistical task: estimating τ as accurately as possible from the observed data. And, second, the
¹⁶ causal (domain-specific) problem of assessing $\text{bias} = \text{ATE} - \tau$. We emphasize that our focus here is
¹⁷ bias due to causal misidentification, not the statistical bias of the estimator. Our aim is to reason
¹⁸ about the bias induced by unobserved confounding—the second task—in a way that imposes no
¹⁹ constraints on the modeling choices for \hat{Q} , \hat{g} and $\hat{\tau}$ used in the statistical analysis.

²⁰
²¹ **Sensitivity Model** Our sensitivity analysis should impose no constraints on how the *observed*
²² data is modeled. However, sensitivity analysis demands some assumption on the relationship between
²³ the observed data and the *unobserved* confounder. It is convenient to formalize such assumptions
²⁴ by specifying a probabilistic model for how the data is generated. The strength of confounding is
²⁵ then formalized in terms of the parameters of the model (the sensitivity parameters). Then, the
²⁶ bias induced by the confounding can be derived from the assumed model. Our task is to posit a
²⁷ generative model that both yields a useful and easily interpretable sensitivity analysis, and that
²⁸ avoids imposing any assumptions about the observed data.

²⁹ To begin, consider the functional form of the sensitivity model used by Imbens [Imb03].

³⁰
³¹ $\text{logit}P(A = 1|x, u) = h(x) + \alpha u \quad (38.101)$

³² $\mathbb{E}[Y|a, x, u] = l(a, x) + \delta u, \quad (38.102)$

³³
³⁴ for some functions h and l . That is, the propensity score is logit-linear in the unobserved confounder,
³⁵ and the conditional expected outcome is linear.

³⁶
³⁷ By rearranging Equation (38.101) to solve for u and plugging in to Equation (38.102), we see
³⁸ that it’s equivalent to assume $\mathbb{E}[Y|t, x, u] = \tilde{l}(t, x) + \tilde{\delta} \text{logit}P(A = 1|x, u)$. That is, the unobserved
³⁹ confounder u only influences the outcome through the propensity score. Accordingly, by positing a
⁴⁰ distribution on $P(A = 1|x, u)$ directly, we can circumvent the need to explicitly articulate U (and h).

⁴¹ **Definition 38.7.1.** Let $\tilde{g}(x, u) = P(A = 1|x, u)$ denote the propensity score given observed covariates
⁴² x and the unobserved confounder u .

⁴³ The insight is that we can posit a sensitivity model by defining a distribution on \tilde{g} directly. We
⁴⁴ choose:

⁴⁵
⁴⁶ $\tilde{g}(X, U)|X \sim \text{Beta}(g(X)(1/\alpha - 1), (1 - g(X))(1/\alpha - 1)).$

That is, the full propensity score $\tilde{g}(X, U)$ for each unit is assumed to be sampled from a Beta distribution centered at the observed propensity score $g(X)$. The sensitivity parameter α plays the same role as in Imbens' model: it controls the influence of the unobserved confounder U on treatment assignment. When α is close to 0 then $\tilde{g}(X, U)|X$ is tightly concentrated around $g(X)$, and the unobserved confounder has little influence. That is, U minimally affects our belief about who is likely to receive treatment. Conversely, when α is close to 1 then \tilde{g} concentrates near 0 and 1; i.e., knowing U would let us accurately predict treatment assignment. Indeed, it can be shown that α is the change in our belief about how likely a unit was to have gotten the treatment, given that they were actually observed to be treated (or not):

$$\alpha = \mathbb{E}[\tilde{g}(X, U)|A = 1] - \mathbb{E}[\tilde{g}(X, U)|A = 0]. \quad (38.103)$$

With the \tilde{g} model in hand, we define the **Austen Sensitivity Model** as follows:

$$\tilde{g}(X, U)|X \sim \text{Beta}(g(X)(1/\alpha - 1), (1 - g(X))(1/\alpha - 1)) \quad (38.104)$$

$$A|X, U \sim \text{Bern}(\tilde{g}(X, U)) \quad (38.105)$$

$$\mathbb{E}[Y|A, X, U] = Q(A, X) + \delta(\text{logit}\tilde{g}(X, U) - \mathbb{E}[\text{logit}\tilde{g}(X, U)|A, X]). \quad (38.106)$$

This model has been constructed to satisfy the requirement that the propensity score and conditional expected outcome are the g and Q actually present in the observed data:

$$\mathbb{P}(A = 1|X) = \mathbb{E}[\mathbb{E}[T|X, U]|X] = \mathbb{E}[\tilde{g}(X, U)|X] = g(X)$$

$$\mathbb{E}[Y|A, X] = \mathbb{E}[\mathbb{E}[Y|A, X, U]|A, X] = Q(A, X).$$

The sensitivity parameters are α , controlling the dependence between the unobserved confounder the treatment assignment, and δ , controlling the relationship with the outcome.

Bias We now turn to calculating the bias induced by unobserved confounding. By assumption, X and U together suffice to render the average treatment effect identifiable as:

$$\text{ATE} = \mathbb{E}[\mathbb{E}[Y|A = 1, X, U] - \mathbb{E}[Y|A = 0, X, U]].$$

Plugging in our sensitivity model yields,

$$\text{ATE} = \mathbb{E}[Q(1, X) - Q(0, X)] + \delta(\mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 1] - \mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 0]).$$

The first term is the observed-data estimate τ , so

$$\text{bias} = \delta(\mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 1] - \mathbb{E}[\text{logit}\tilde{g}(X, U)|X, A = 0]).$$

Then, by invoking Beta-Bernoulli conjugacy and standard Beta identities,¹⁰ we arrive at,

Theorem 6. *Under the Austen sensitivity model, Equation (38.106), an unobserved confounder with influence α and δ induces bias in the estimated treatment effect equal to*

$$\text{bias} = \frac{\delta}{1/\alpha - 1} \mathbb{E}\left[\frac{1}{g(X)} + \frac{1}{1 - g(X)}\right].$$

¹⁰ We also use the recurrence relation $\psi(x+1) - \psi(x) = 1/x$, where ψ is the digamma function.

¹ That is, the amount of bias is determined by the sensitivity parameters and by the *realized*
² propensity score. Notice that more extreme propensity scores lead to more extreme bias in response
³ to unobserved confounding. This means, in particular, that conditioning on a covariate that affects
⁴ the treatment but that does not directly affect the outcome (an instrument) will increase any bias
⁵ due to unobserved confounding. This general phenomena is known as **z-bias**.
⁶

⁷
⁸ **Sensitivity Parameters** The Austen model provides a formalization of confounding strength
⁹ in terms of the parameters α and δ and tells us how much bias is induced by a given strength of
¹⁰ confounding. This lets us translate judgments about confounding strength to judgments about bias.
¹¹ However, it is not immediately obvious how to translate qualitative judgements such as “I think any
¹² unobserved confounder would be much less important than Age” to judgements about the possible
¹³ values of the sensitivity parameters.

¹⁴ First, because the scale of δ is not fixed, it may be difficult to compare the influence of potential
¹⁵ unobserved confounders to the influence of reference variables. To resolve this, we reexpress the
¹⁶ outcome-confounder strength in terms of the (non-parametric) partial coefficient of determination:

$$\begin{aligned} \text{17} \\ \text{18} \quad R_{Y,\text{par}}^2(\alpha, \delta) &= 1 - \frac{\mathbb{E}(Y - \mathbb{E}[Y|A, X, U])^2}{\mathbb{E}(Y - Q(A, X))^2}. \\ \text{19} \end{aligned}$$

²⁰ The key to computing the reparameterization is the following result

²¹ **Theorem 7.** Under the Austen sensitivity model, Equation (38.106), the outcome influence is
²²

$$\begin{aligned} \text{23} \\ \text{24} \quad R_{Y,\text{par}}^2(\alpha, \delta) &= \delta^2 \sum_{a=0}^1 \frac{\mathbb{E}[\psi_1(g(X)^a(1-g(X))^{1-a}(1/\alpha-1) + 1[A=a])]}{\mathbb{E}[(Y - Q(A, X))^2]}, \\ \text{25} \end{aligned}$$

²⁶ where ψ_1 is the trigamma function.

²⁷ See Veitch and Zaveri [VZ20] for the proof.

²⁸ By design, α —the strength of confounding influence on on treatment assignment—is already on
²⁹ a fixed, unitless scale. However, because the measure is tied to the model it may be difficult to
³⁰ interpret, and it is not obvious how to compute reference confounding strength values from the
³¹ observed data. The next result clarifies these issues.

³²
³³ **Theorem 8.** Under the Austen sensitivity model, Equation (38.106),

$$\begin{aligned} \text{34} \\ \text{35} \quad \alpha &= 1 - \frac{\mathbb{E}[\tilde{g}(X, U)(1 - \tilde{g}(X, U))]}{\mathbb{E}[g(X)(1 - g(X))]} \\ \text{36} \end{aligned}$$

³⁷ See Veitch and Zaveri [VZ20] for the proof. That is, the sensitivity parameter α is measures how
³⁸ much more extreme the propensity scores become when we condition on U . That is, α is a measure
³⁹ of the extra predictive power U adds for A , above and beyond the predictive power in X . It may
⁴⁰ also be insightful to notice that

$$\begin{aligned} \text{41} \\ \text{42} \quad \alpha &= R_{A,\text{par}}^2 = 1 - \frac{\mathbb{E}[(A - \tilde{g}(X, U))^2]}{\mathbb{E}[(A - g(X))^2]}. \\ \text{43} \end{aligned} \tag{38.107}$$

⁴⁴ That is, α is just the (non-parametric) partial coefficient of determination of U on A —the same
⁴⁵ measure used for the outcome influence. (To see this, just expand the expectations conditional on
⁴⁶ $A = 1$ and $A = 0$).

⁴⁷

Estimating bias In combination, Theorems 6 and 7 yield an expression for the bias in terms of α and $R_{Y,\text{par}}^2$. In practice, we can estimate the bias induced by confounding by fitting models for \hat{Q} and \hat{g} and replacing the expectations by means over the data.

38.7.2.1 Calibration using observed data

The analyst must make judgments about the influence a hypothetical unobserved confounder might have on treatment assignment and outcome. To calibrate such judgments, we'd like to have a reference point for how much the observed covariates influence the treatment assignment and outcome. In the sensitivity model, the degree of influence is measured by partial R_Y^2 and α . We want to measure the degree of influence of an observed covariate Z given the other observed covariates $X \setminus Z$.

For the outcome, this can be measured as:

$$R_{Y \cdot Z|T,X \setminus Z}^2 \triangleq 1 - \frac{\mathbb{E}(Y - Q(A, X))^2}{\mathbb{E}(Y - \mathbb{E}[Y|A, X \setminus Z])^2}.$$

In practice, we can estimate the quantity by fitting a new regression model \hat{Q}_Z that predicts Y from A and $X \setminus Z$. Then we compute

$$R_{Y \cdot Z|T,X \setminus Z}^2 = 1 - \frac{\frac{1}{n} \sum_i (y_i - \hat{Q}(t_i, x_i))^2}{\frac{1}{n} \sum_i (y_i - \hat{Q}_Z(t_i, x_i \setminus z_i))^2}.$$

Using Theorem 8, we can measure influence of observed covariate Z on treatment assignment given $X \setminus Z$ in an analogous fashion to the outcome. We define $g_{X \setminus Z}(X \setminus Z) = P(A = 1|X \setminus Z)$, then fit a model for $g_{X \setminus Z}$ by predicting A from $X \setminus Z$, and estimate

$$\hat{\alpha}_{Z|X \setminus Z} = 1 - \frac{\frac{1}{n} \sum_i \hat{g}(x_i)(1 - \hat{g}(x_i))}{\frac{1}{n} \sum_i \hat{g}_{X \setminus Z}(x_i \setminus z_i)(1 - \hat{g}_{X \setminus Z}(x_i \setminus z_i))}.$$

Grouping covariates The estimated values $\hat{\alpha}_{X \setminus Z}$ and $\hat{R}_{Y \cdot X \setminus Z}^2$ measure the influence of Z conditioned on all the other confounders. In some cases, this can be misleading. For example, if some piece of information is important but there are multiple covariates providing redundant measurements, then the estimated influence of each covariate will be small. To avoid this, group together related or strongly dependent covariates and compute the influence of the entire group in aggregate. For example, grouping income, location, and race as ‘socioeconomic variables’.

38.7.2.2 Practical Use

We now have sufficient results to produce Austen plots such as Figure 38.8. At a high level, the procedure is:

1. Produce an estimate $\hat{\tau}$ using any modeling tools. As a component of this, estimate the propensity score \hat{g} and conditional outcome model \hat{Q}
2. Pick a level of bias that would suffice to change the qualitative interpretation of the estimate (e.g., the lower bound of a 95% confidence interval).

¹ 3. Plot the values of α and $R_{Y,\text{par}}^2$ that would suffice to induce that much bias. This is the black
² curve on the plot. To calculate these values, use Theorems 6 and 7 together with the estimated \hat{g}
³ and \hat{Q} .
⁴

⁵ 4. Finally, compute reference influence level for (groups of) observed covariates. In particular, this
⁶ requires fitting reduced models for the conditional expected outcome and propensity that do not
⁷ use the reference covariate as a feature.
⁸

⁹ In practice, an analyst only needs to do the model fitting parts themselves. The bias calculations,
¹⁰ reference value calculations, and plotting can be done automatically with standard libraries.¹¹

¹¹ Austen plots are predicated on Equation (38.106). This assumption replaces the purely parametric
¹² Equation (38.99) with a version that eliminates any parametric requirements on the observed data.
¹³ However, we emphasize that Equation (38.106) does, implicitly, impose some parametric assumption
¹⁴ on the structural causal relationship between U and A, Y . Ultimately, any conclusion drawn from
¹⁵ the sensitivity analysis depends on this assumption, which is not justified on any substantive grounds.
¹⁶ Accordingly, such sensitivity analyses can only be used to informally guide domain experts. They
¹⁷ do not circumvent the need to thoroughly adjust for confounding. This reliance on a structural
¹⁸ assumption is a generic property of sensitivity analysis.¹² Indeed, there are now many sensitivity
¹⁹ analysis models that allow the use of any machine learning model in the data analysis [e.g., RRS00;
²⁰ FDF19; She+11; HS13; BK19; Ros10; Yad+18; ZSB19; Sch+21a]. However, none of these are yet in
²¹ routine use in practice. We have presented Austen plots here not because they make an especially
²² virtuous modeling assumption, but because they are (relatively) easy to understand and interpret.

²³ Austen plots are most useful in situations where the conclusion from the plot would be ‘obvious’
²⁴ to a domain expert. For instance, in Figure 38.8, we can be confident that an unobserved confounder
²⁵ similar to socioeconomic status would not induce enough bias to change the qualitative conclusion.
²⁶ By contrast, Austen plots should not be used to draw conclusions such as, “I think a latent confounder
²⁷ could only be 90% as strong as ‘age’, so there is evidence of a small non-zero effect”. Such nuanced
²⁸ conclusions might depend on issues such as the particular sensitivity model we use, or finite-sample
²⁹ variation of our bias and influence estimates, or on incautious interpretation of the calibration dots.
³⁰ These issues are subtle, and it would be difficult resolve them to a sufficient degree that a sensitivity
³¹ analysis would make an analysis credible.
³²

³³ **Calibration using observed data** The interpretation of the observed-data calibration requires
³⁴ some care. The sensitivity analysis requires the analyst to make judgements about the strength
³⁵ of influence of the unobserved confounder U , *conditional on the observed covariates X* . However,
³⁶ we report the strength of influence of observed covariate(s) Z , *conditional on the other observed*
³⁷ *covariates $X \setminus Z$* . The difference in conditioning sets can have subtle effects.
³⁸

³⁹ Cinelli and Hazlett [CH20] give an example where Z and U are identical variables in the true
⁴⁰ model, but where influence of U given A, X is larger than the influence of Z given $A, X \setminus Z$. (The
⁴¹ influence of Z given $X \setminus Z, U$ would be the same as the influence of U given X). Accordingly, an
⁴² analyst is *not* justified in a judgement such as, “I know that U and Z are very similar. I see Z has
⁴³ substantial influence, but the dot is below the line. Thus, U will not undo the study conclusions.” In

⁴⁴ 11. See github.com/vveitch/causality-tutorials/blob/main/Sensitivity_Analysis.ipynb

⁴⁵ 12. In extreme cases, there can be so little unexplained variation in A or Y that only a very weak confounder could be
⁴⁶ compatible with the data. In this case, essentially assumption free sensitivity analysis is possible [Man90].

essence, if the domain expert suspects a strong interaction between U and Z then naively eyeballing the dot-vs-line position may be misleading. A particular subtle case is when U and Z are independent variables that both strongly influence A and Y . The joint influence on A creates an interaction effect between them when A is conditioned on (the treatment is a collider). This affects the interpretation of $R^2_{Y,U|X,A}$. Indeed, we should generally be skeptical of sensitivity analysis interpretation when it is expected that a strong confounder has been omitted. In such cases, our conclusions may depend substantively on the particular form of our sensitivity model, or other unjustifiable assumptions.

Although the interaction problem is conceptually important, its practical significance is unclear. We often expect the opposite effect: if U and Z are dependent (e.g., race and wealth) then omitting U should increase the apparent importance of Z —leading to a conservative judgement (a dot artificially towards the top right part of the plot).

38.8 The Do Calculus

We have seen several strategies for identifying causal effects as parameters of observational distributions. Confounder adjustment (Section 38.4) relied only on the assumed causal graph (and overlap), which specified that we observe all common causes of A and Y . On the other hand, instrumental variable methods and difference-in-differences each relied on both an assumed causal graph and partial functional form assumptions about the underlying structural causal model. Because functional form assumptions can be quite difficult to justify on substantive grounds, it's natural to ask when causal identification is possible from the causal graph alone. That is, when can we be agnostic to the particular functional form of the structural causal models?

There is a general “**calculus of intervention**”, known as the **do-calculus**, that gives a general recipe for determining when the causal assumptions expressed in a causal graph can be used to identify causal effects [Pea09c]. The do-calculus is a set of three rewrite rules that allows us to replace statements where we condition on variables being set by intervention, e.g. $P(Y|\text{do}(A = a))$, with statements involving only observational quantities, e.g. $\mathbb{E}_X[P(Y|A = a, X)]$. When causal identification is possible, we can repeatedly apply the three rules to boil down our target causal parameter into an expression involving only the observational distribution.

38.8.1 The three rules

To express the rules, let X , Y , Z , and W be arbitrary disjoint sets of variables in a causal DAG G .

Rule 1 The first rule allows us to insert or delete observations z :

$$p(y|\text{do}(x), z, w) = p(y|\text{do}(x), w) \text{ if } (Y \perp Z|X, W)_{G_{\overline{X}}} \quad (38.108)$$

where $G_{\overline{X}}$ denotes cutting edges going into X , and $(Y \perp Z|X, W)_{G_{\overline{X}}}$ denotes conditional independence in the mutilated graph. The rule follows from d-separation in the mutilated graph. This rule just says that conditioning on irrelevant variables leaves the distribution invariant (as we would expect).

Rule 2 The second rule allows us to replace $\text{do}(z)$ with conditioning on (seeing) z . The simplest case where we can do this is: if Z is a root of the causal graph (i.e., it has no causal parents) then $p(y|\text{do}(z)) = p(y|z)$. The reason is that the do operator is equivalent to conditioning in the mutilated

¹ causal graph where all the edges into Z are removed, but, because Z is a root, the mutilated graph
² is just the original causal graph. The general form of this rule is:
³

$$\frac{4}{5} \quad p(y|\text{do}(x), \text{do}(z), w) = p(y|\text{do}(x), z, w) \text{ if } (Y \perp Z|X, W)_{G_{\overline{X}Z}} \quad (38.109)$$

⁶ where $G_{\overline{X}Z}$ cuts edges going into X and out of Z . Intuitively, we can replace $\text{do}(z)$ by z as long as
⁷ there are no backdoor (non-directed) paths between z and y . If there are in fact no such paths, then
⁸ cutting all the edges going out of Z will mean there are no paths connecting Z and Y , so that $Y \perp Z$.
⁹ The rule just generalizes this line of reasoning to allow for extra observed and intervened variables.
¹⁰

¹¹ **Rule 3** The third rule allows us to insert or delete actions $\text{do}(z)$:

$$\frac{13}{14} \quad p(y|\text{do}(x), \text{do}(z), w) = p(y|\text{do}(x), w) \text{ if } (Y \perp Z|X, W)_{G_{\overline{X}Z^*}} \quad (38.110)$$

¹⁵ where $G_{\overline{X}Z^*}$ cuts edges going into X and Z^* , and where Z^* is the set of Z -nodes that are not
¹⁶ ancestors of any W -node in $G_{\overline{X}}$. Intuitively, this condition corresponds to intervening on X , and
¹⁷ checking whether the distribution of Y is invariant to *any* intervention that we could apply on Z .
¹⁸

¹⁹ 38.8.2 Revisiting Backdoor Adjustment

²¹ We begin with a more general form of the adjustment formula we used in Section 38.4.

²² First, suppose we observe all of A 's parents, call them X . For notational simplicity, we'll assume
²³ for the moment that X is discrete. Then,
²⁴

$$\frac{25}{26} \quad p(Y = y|\text{do}(A = a)) = \sum_x p(Y = y|x, \text{do}(A = a))p(x|\text{do}(A = a)) \quad (38.111)$$

$$\frac{27}{28} \quad = \sum_x p(Y = y|x, A = a)p(x). \quad (38.112)$$

³⁰ The first line is just a standard probability relation (marginalizing over z). We are using causal
³¹ assumptions in two ways in the second line. First, $p(x|\text{do}(A = a)) = p(x)$: the treatment has
³² no causal effect on Z , so interventions on A don't change the distribution of Z . This is rule 3,
³³ Equation (38.110). Second, $p(Y = y|z, \text{do}(A = a)) = p(Y = y|z, A = a)$. This equality holds because
³⁴ conditioning on the parents blocks all non-directed paths from A to Y , reducing the causal effect to
³⁵ be the same as the observational effect. The equality is an application of rule 2, Equation (38.109).
³⁶

³⁷ Now, what if we don't observe all the parents of A ? The key issue is **backdoor paths**: paths
³⁸ between A and Y that contain an arrow into A . These paths are the general form of the problem
³⁹ that occurs when A and Y share a common cause. Suppose that we can find a set of variables S
⁴⁰ such that (1) no node in S is a descendant of A ; and (2) S blocks every backdoor path between A
⁴¹ and Y . Such a set is said to satisfy the **backdoor criterion**. In this case, we can use S instead of
⁴² the parents of X in the adjustment formula, Equation (38.112). That is,

$$\frac{43}{44} \quad p(Y = y|\text{do}(A = a)) = \mathbb{E}_S[p(Y = y|S, A = a)]. \quad (38.113)$$

⁴⁵ The proof follows the invocation of rules 3 and 2, in the same way as for the case where S is just the
⁴⁶ parents of A . Notice that requiring S to not contain any descendants of A means that we don't risk
⁴⁷

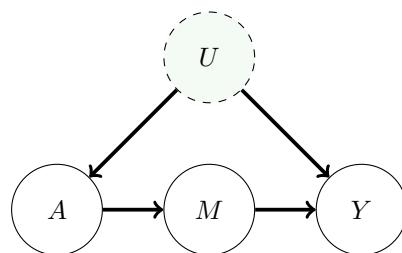


Figure 38.9: Causal graph illustrating the frontdoor criterion setup. The effect of the treatment A on outcome Y is entirely mediated by mediator M . This allows us to infer the causal effect even if the treatment and outcome are confounded by U .

conditioning on any variables that mediate the effect, nor any variables that might be colliders—either would undermine the estimate.

The backdoor adjustment formula generalizes the adjust-for-parents approach and adjust-for-all-common-causes approach of Section 38.4. That’s because both the parents of A and the common causes satisfy the backdoor criterion.

In practice, the full distribution $p(Y = y|\text{do}(A = a))$ is rarely used as the causal target. Instead, we try to estimate a low-dimensional parameter of this distribution, such as the average treatment effect. The adjustment formula immediately translates in the obvious way. If we define

$$\tau = \mathbb{E}_S[\mathbb{E}[Y|A = 1, S] - \mathbb{E}[Y|A = 0, S]],$$

then we have that $\text{ATE} = \tau$ whenever S satisfies the backdoor criteria. The parameter τ can then be estimated from finite data using the methods described in Section 38.4, using S in place of the common causes X .

38.8.3 Frontdoor Adjustment

Backdoor adjustment is applicable if there’s at least one observed variable on every backdoor path between A and Y . As we have seen, identification is sometimes still possible even when this condition doesn’t hold. Frontdoor adjustment is another strategy of this kind. Figure 38.9 shows the causal structure that allows this kind of adjustment strategy. Suppose we’re interested in the effect of smoking A on developing cancer Y , but we’re concerned about some latent genetic confounder U .

Suppose that all of the directed paths from A to Y pass through some set of variables M . Such variables are called **mediators**. For example, the effect of smoking on lung cancer might be entirely mediated by the amount of tar in the lungs and measured tissue damage. It turns out that if all such mediators are observed, and the mediators do not have an unobserved common cause with A or Y , then causal identification is possible. To understand why this is true, first notice that we can identify the causal effect of A on M and the causal effect of M on A , both by backdoor adjustment. Further, the mechanism of action of A on Y is: A changes M which in turn changes Y . Then, we

1
2 can combine these as:

3
4 $p(Y|\text{do}(A = a)) = \sum_m p(Y|\text{do}(M = m))p(M = m|\text{do}(A = a))$ (38.114)

5
6 $= \sum_m \sum_{a'} p(Y|a', m)p(a')p(m|a)$ (38.115)

7

8
9 The second line is just backdoor adjustment applied to identify each of the do expressions (note that
10 A blocks the $M-Y$ backdoor path through U).

11 Equation (38.115) is called the **front-door formula** [Pea09b, §3.3.2]. To state the result in more
12 general terms, let us introduce a definition. We say a set of variables M satisfies the **front-door**
13 **criterion** relative to an ordered pair of variables (A, Y) if (1) M intercepts all directed paths from
14 A to Y ; (2) there is no unblocked backdoor path from A to M ; and (3) all backdoor paths from M
15 to Y are blocked by A . If M satisfies this criterion, and if $p(A, M) > 0$ for all values of A and M ,
16 then the causal effect of A on Y is identifiable and is given by Equation (38.115).

17 Let us interpret this theorem in terms of our smoking example. Condition 1 means that smoking
18 A should have no effect on cancer Y except via tar and tissue damage M . Conditions 2 and 3 mean
19 that the genotype U cannot have any effect on M except via smoking A . Finally, the requirement
20 that $p(A, M) > 0$ for all values implies that high levels of tar in the lungs must arise not only due to
21 smoking, but also other factors (e.g., pollutants). In other words, we require $p(A = 0, M = 1) > 0$ so
22 we can assess the impact of the mediator in the untreated setting.

23 We can now use the do-calculus to derive the frontdoor criterion; following [PM18b, p236]. Assuming
24 the causal graph G shown in Figure 38.9:

25

26 $p(y|\text{do}(a)) = \sum_m p(y|\text{do}(a), m)p(m|\text{do}(a))$ (probability axioms)

27 $= \sum_m p(y|\text{do}(a), \text{do}(m))p(m|\text{do}(a))$ (rule 2 using $G_{\bar{S}T}$)

28 $= \sum_m p(y|\text{do}(a), \text{do}(m))p(m|a)$ (rule 2 using G_S)

29 $= \sum_m p(y|\text{do}(m))p(m|a)$ (rule 3 using $G_{\bar{S}T^*}$)

30 $= \sum_{a'} \sum_m p(y|\text{do}(m), a')p(a'|\text{do}(m))p(m|a)$ (probability axioms)

31 $= \sum_{a'} \sum_m p(y|m, a')p(a'|\text{do}(m))p(m|a)$ (rule 2 using G_U)

32 $= \sum_{a'} \sum_m p(y|m, a')p(a')p(m|a)$ (rule 3 using $G_{\bar{T}^*}$)

33

34 **Estimation** To estimate the causal distribution from data using the frontdoor criterion we need to
35 estimate each of $p(y|m, a)$, $p(a)$, and $p(m|a)$. In practice, we can fit models $\hat{p}(y|m, a)$ by predicting
36 Y from M and A , and $\hat{p}(m|a)$ by predicting M from A . Then, using the empirical distribution to
37

38

39 $\hat{p}(a)$ To estimate the causal distribution from data using the frontdoor criterion we need to
40 estimate each of $p(y|m, a)$, $p(a)$, and $p(m|a)$. In practice, we can fit models $\hat{p}(y|m, a)$ by predicting
41 Y from M and A , and $\hat{p}(m|a)$ by predicting M from A . Then, using the empirical distribution to
42

43

1 estimate $p(a)$, the final estimate is:

$$\frac{1}{|A|} \sum_{a'} \sum_m \hat{p}(y|m, a') \hat{p}(m|a), \quad (38.116)$$

6 where $|A|$ is the number of treatments.

8 We usually have more modest targets than the full distribution $p(y|\text{do}(a))$. For instance, we may
9 be content with just estimating the average treatment effect. It's straightforward to derive a formula
10 for this using the frontdoor adjustment. Similarly to backdoor adjustment, more advanced estimators
11 of the ATE through frontdoor effect are possible in principle. For example, we might combine fitted
12 models for $\mathbb{E}[Y|m, a]$ and $P(M|a)$. See Fulcher et al. [Ful+20] for an approach to robust estimation
13 via front door adjustment, as well as a generalization of the front door approach to more general
14 settings.

16 38.9 Further Reading

18 There is an enormous and growing literature on the intersection of causality and machine learning.

19 First, there are many textbooks on theoretical and practical elements of causal inference. These
20 include Pearl [Pea09c], focused on causal graphs, Angrist and Pischke [AP08], focused on econometrics,
21 Hernán and Robins [HR20b], with roots in epidemiology, Imbens and Rubin [IR15], with origin in
22 statistics, and Morgan and Winship [MW15], for a social sciences perspective. The introduction to
23 causality in Shalizi [Sha22, §7] is also recommended, particularly the treatment of matching.

24 Double machine-learning has featured prominently in this chapter. This is a particular instantiation
25 of non-parametric estimation. This topic has substantial theoretical and practical importance in
26 modern causal inference. The double machine learning work includes estimators for many commonly
27 encountered scenarios [Che+17e; Che+17d]. Good references for a lucid explanation of how and
28 why non-parametric estimation works include [Ken16; Ken17; FK21]. Usually, the key guarantees of
29 non-parametric estimator are asymptotic. Generally, there are many estimators that share optimal
30 asymptotic guarantees (e.g. the AIPTW estimator given in Equation (38.31)). Although these are
31 asymptotically equivalent, in finite samples their behavior can be very different. There are estimators
32 that preserve asymptotic guarantees but aim to improve performance in practical finite sample
33 regimes [e.g., vR11].

34 There is also considerable interest in the estimation of heterogeneous treatment effects. The
35 question here is: what effect would this treatment have when applied to a unit with such-and-such
36 specific characteristics? E.g., what is the effect of this drug on women over the age of 50? The causal
37 identification arguments used here are more-or-less the same as for the estimation of average case
38 effects. However, the estimation problems can be substantially more involved. Some reading includes
39 [Kün+19; NW20; Ken20; Yad+21].

40 There are several commonly applicable causal identification and estimation strategies beyond the
41 ones we've covered in this chapter. **Regression discontinuity designs** rely on the presence of
42 some sharp, arbitrary non-linearity in treatment assignment. For example, eligibility for some aid
43 programs is determined by whether an individual has income below or above a fixed amount. The
44 effect of the treatment can be studied by comparing units just below and just above this threshold.
45 **Synthetic controls** are a class of methods that try to study the effect of a treatment on a given
46 unit by constructing a synthetic version of that unit that acts as a control. For example, to study the
47

1 effect of legislation banning smoking indoors in California, we can construct a synthetic California
2 as a weighted average of other states, with weights chosen to balance demographic characteristics.
3 Then, we can compare the observed outcome of California with the outcome of the synthetic control,
4 constructed as the weighted average of the outcomes of the donor states. See Angrist and Pischke
5 [AP08] for a textbook treatment of both strategies. Closely related are methods that use time series
6 modeling to create synthetic outcomes. For example, to study the effect of an advertising campaign
7 beginning at time T on product sales Y_t , we might build a time series model for Y_t using data in the
8 $t < T$ period, and then use this model to predict the values of $(\hat{Y}_t)_{t>T}$ we would have seen had the
9 campaign not been run. We can estimate the causal effect by comparing the factual, realized Y_t to
10 the predicted, counterfactual, \hat{Y}_t . See Brodersen et al. [Bro+15] for an instantiation of this idea.

11 In this chapter, our focus has been on using machine learning tools to estimate causal effects.
12 There is also a growing interest in using the ideas of causality to improve machine learning tools.
13 This is mainly aimed at building predictors that are robust against when deployed in new domains
14 [SS18c; SCS19; Arj+20; Mei18b; PBM16a; RC+18; Zha+13a; Sch+12b; Vei+21] or that do not rely
15 on particular ‘spurious’ correlations in the training data [RPH21; Wu+21; Gar+19; Mit+20; WZ19;
16 KCC20; KHL20; TAH20; Vei+21].

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

Index

- Q-function, 1119
 χ -divergence VI, 439
 χ -divergence upper bound, 440
 f -MAX, 1161
 (xed interval) smoothing, 316
 #P-hard, 385
 Global explanation, 1074
 Local explanation, 1073
 intrinsic, 1080
 "arms", 1102
 "do" operator, 177
 1/f, 14
 2d lattice model, 146
 3-SAT, 385
 80-20 rule, 15
- Sylvester determinant lemma, 677
- A* search, 1039, 1144
 A/B test, 1102
 A2C, 1138
 A3C, 1138
 ABC, 543, 886
 abduction, 181
 ABOBA, 508
 absorbing state, 52, 1118
 abstain, 745
 accept, 464
 acquisition function, 263, 265
 acquisition shift, 736
 action, 1107
 action nodes, 1099
 action-value function, 1119
 Actionability, 1078
 actions, 119, 991
 activation function, 598
 active inference, 1158
 active learning, 262, 267, 676
 actor critic, 1127
 actor-critic, 1137
 acyclic directed mixed graph, 165
 AD, 219
 Adam, 261
 adaptive importance sampling, 824
 adaptive MCMC, 468
 adaptive policies, 1107
 adaptive prediction sets, 561
 adaptive proposal distributions, 531
 adaptive rejection Metropolis sampling, 477
 adaptive rejection sampling, 454
 adaptive resampling, 522
 adaptive tempering, 539
 add-one smoothing, 50, 72
 additive Gaussian noise, 991
 additive intervention, 178
 additive scalar bijection, 842
 additive unobserved confounding, 1189
 ADF, 356
 adjacency matrix, 612
 adjacency list, 612
 adjoint sensitivity method, 854
 ADMG, 165
- admissible, 1144
 admixture mixture, 952
 admixture model, 24, 950
 advantage actor critic, 1138
 advantage function, 1119
 adversarial stickers, 763
 adversarial attack, 763
 adversarial autoencoder, 800
 adversarial bandit, 1107
 adversarial image, 763
 adversarial inverse RL, 1161
 adversarial risk, 767
 adversarial training, 767
 ADVI, 307, 422
 AEP, 198
 a ne autoregressive ows, 846
 a ne ow, 841
 a ne scalar bijection, 842
 a ne transformation, 841
 a nity propagation, 379
 agent, 120, 1107, 1116
 aggregate, 614
 aggregate posterior, 209
 aggregated posterior, 215, 794
 AI, 208
 AIPTW, 1179
 AIRL, 1161
 Akaike information criterion, 116
 aleatoric uncertainty, 66, 556
 ALI, 906
 alpha divergence, 55
 alpha posterior, 623
 AlphaGo, 1144
 AlphaZero, 1144
 alternative hypothesis, 111
 amortization gap, 426
 amortized ELBO, 426
 amortized inference, 381, 425, 531, 785, 786, 826
 analysis by synthesis, 919
 ancestor, 520
 ancestral graph, 155
 ancestral sampling, 134
 annealed importance sampling, 456, 538, 779, 857
 annotation shift, 737
 annulus, 198
 anomaly detection, 725, 742, 945
 anti-causal prediction, 735
 anti-Hebbian term, 861
 antiferromagnetic, 147
 antithetic sampling, 460
 aperiodic, 53
 apprenticeship learning, 1160
 Approximate Bayesian Computation, 543, 886
 approximate dynamic programming, 1121
 approximate inference, 304
 approximate linear programming, 1121
 AR, 771
 architectural methods, 761
 ARD, 579, 657
 ARD kernel, 657
- arithmetic circuits, 170
 arithmetic coding, 208
 ARMA, 719
 arrow of time, 729
 artifical process noise, 527
 ASOS, 999
 ASR, 327
 associative Markov model, 149
 associative Markov network, 147
 assumed density Itering, 356, 360
 asynchronous advantage actor critic, 1138
 asymmetric numeral systems, 208
 asymptotic consistency, 1040
 asymptotic equipartition property, 198
 asynchronous updates, 376
 asynchronous value iteration, 1122
 ATE, 178, 1164
 atoms, 1032
 ATT, 1185
 attention layer, 603
 attention score, 603
 attention weight, 603
 attribute, 171
 audio-visual speech recognition, 988
 augmented DAG, 178
 Augmented Inverse Probability of Treatment Weighted Estimator, 1179
 augmented reality, 1001
 Austen plots, 1200
 Austen Sensitivity Model, 1203
 auto-regressive HMM, 964
 auto-regressive model, 829
 Auto-Sklearn, 268
 Auto-Weka, 268
 autoconj, 414
 autocorrelation function, 500
 autocorrelation time, 501
 autocovariance function, 501
 autodi, 219
 automatic differentiation, 219
 automatic differentiation variational inference, 307, 422
 Automatic Relevance Determination, 657
 automatic relevance determination, 945
 automatic relevancy determination, 579, 622
 automatic speech recognition, 327, 774, 984
 autoregressive, 771
 autoregressive bijection, 846
 autoregressive ows, 846
 autoregressive models, 914
 auxiliary latent variables, 434
 auxiliary variable deep generative model, 434
 auxiliary variables, 480
 average causal effect, 1173
 Average Treatment Effect, 1164
 average treatment effect, 178, 1173
 average treatment effect on the treated, 1185

- 1
 2 axis aligned, 16
 3 BA lower bound, 206
 4 back translation, 741
 5 backcasting, 730
 6 backdoor criterion, 1208
 7 backdoor paths, 1208
 8 back smoothing, 108
 9 backpropagation, 228
 10 backup diagram, 1123
 11 backward Euler method, 993
 12 backwards kernel, 537
 13 backwards transfer, 760
 14 bagging, 630
 15 balance the dataset, 738
 16 Bambi, 593
 17 BAMDP, 1129
 18 bandit problem, 1107
 19 bandwidth, 656 , 775
 20 BAOAB, 508
 21 base distribution, 837
 22 base measure, 29 , 1030
 23 base-rate, 1055
 24 baseline, 459
 25 baseline function, 240
 26 basic random variables, 171
 27 basis functions, 958
 28 batch ensemble, 631
 29 batch normalization, 601
 30 batch optimization, 239
 31 batch reinforcement learning, 1150
 32 batched Bayesian optimization, 268
 33 Baum-Welch, 971 , 971
 34 Baum-Welch algorithm, 140
 35 Bayes ball algorithm, 129
 36 Bayes by backprop, 625
 37 Bayes estimator, 120
 38 Bayes factor, 111
 39 Bayes Iter, 318
 40 Bayes nets, 123
 41 Bayes' rule, 64
 42 Bayes' rule for Gaussians, 18
 43 Bayes-adaptive MDP, 1129
 44 BayesBiNN, 262
 45 Bayesian approach, 67
 46 Bayesian dark knowledge, 632
 47 Bayesian decision theory, 119 , 1099
 48 Bayesian deep learning, 619
 49 Bayesian factor regression, 937
 50 Bayesian gradient descent, 432
 51 Bayesian hypothesis testing, 111
 52 Bayesian inference, 64
 53 Bayesian information criterion, 115
 54 Bayesian lasso, 577
 55 Bayesian learning rule, 255
 56 Bayesian model selection, 110
 57 Bayesian multi net, 144
 58 Bayesian networks, 123
 59 Bayesian neural network, 619
 60 Bayesian nonparametric models, 512
 61 Bayesian nonparametrics, 1029
 62 Bayesian Occam's razor, 110
 63 Bayesian online changepoint detection, 982
 64 Bayesian optimization, 262 , 655
 65 Bayesian p-value, 117
 66 Bayesian statistics, 63
 67 Bayesian structural time series, 719
 68 BayesOpt, 262
 69 BBB, 625
 70 BBMM, 685
 71 BBVI, 414
 72 beam search, 1039
 73 Bean Machine, 174
 74 behavior cloning, 1160
 75 behavior policy, 1150
 76 behavior-agnostic α -policy, 1150
 77 belief networks, 123
 78 belief propagation, 366
 79 belief state, 315 , 319 , 556 , 1109 , 1129
 80 belief states, 366
 81 belief-state MDP, 1111
 82 Bellman backup, 1121
 83 Bellman error, 1119
 84 Bellman residual, 1119
 85 Bellman's optimality equations, 1119
 86 Berkson's paradox, 126 , 131
 87 Bernoulli bandit, 1110
 88 Bernoulli distribution, 5
 89 Bernoulli mixture model, 924
 90 BERT, 609 , 806
 91 Bessel function, 24 , 657
 92 best arm identification, 262 , 1105
 93 best-arm identification, 1109
 94 best-first search, 1143
 95 beta distribution, 12 , 70
 96 beta function, 12
 97 Beta process, 1046
 98 beta-binomial, 74
 99 beta-VAE, 796
 100 Bethe free energy, 377
 101 Bhattacharya distance, 525
 102 bi-directed graph, 166
 103 BIC, 115 , 693
 104 BIC loss, 115
 105 BIC score, 115
 106 big data, 64
 107 BiGAN, 906
 108 bigram model, 48 , 200
 109 bigram statistics, 50
 110 bijection, 839
 111 bilinear form, 605
 112 bilinear method, 993 , 1015
 113 binary entropy function, 196
 114 binary logistic regression, 581
 115 binary neural network, 262
 116 binary partition transformer, 616
 117 binomial coefficient, 5
 118 binomial distribution, 5
 119 binomial regression, 566
 120 BIO, 713
 121 bit error, 372
 122 bits, 188
 123 bits back coding, 211
 124 bits per dimension, 778
 125 BIVA, 813
 126 bivariate Gaussian, 16
 127 black box attack, 764
 128 black box shift estimation, 741
 129 black box variational inference, 414
 130 blackbox EP, 444
 131 blackbox matrix-matrix multiplication, 685
 132 Blackwell-MacQueen, 1034
 133 blind inverse problem, 929
 134 blind source separation, 953
 135 block length, 211
 136 block stacking, 1145
 137 blocked Gibbs sampling, 477
 138 BLOG, 174
 139 BN, 601
 140 BNN, 619
 141 Bochner's theorem, 663
 142 BOPCD, 982
 143 Boltzmann machine, 149
 144 Boltzmann policy, 1128
 145 bond variables, 483
 146 Bonnet's theorem, 235 , 242
 147 bootstrap Iter, 519
 148 bootstrapping, 1126 , 1130
 149 borrow statistical strength, 100
 150 bottom-up inference model, 813
 151 bouncy particle sampler, 464
 152 bound optimization, 245 , 248
 153 Box-Muller, 451
 154 BP, 366
 155 BPT, 616
 156 brain, 336
 157 branching factor, 200
 158 Bregman divergence, 195 , 237 , 238 , 428
 159 Brier score, 552
 160 BRMS, 593
 161 Brownian motion, 490 , 491 , 658 , 1052
 162 Brownian noise, 505 , 508
 163 BSS, 953
 164 BSTs, 719
 165 bucket elimination, 381
 166 building blocks, 597
 167 burn-in phase, 493
 168 burn-in time, 463
 169 burstiness, 1055
 170 calculus of intervention, 1207
 171 calculus of variations, 38
 172 calibrated, 552
 173 calibration set, 560
 174 canonical correlation analysis, 939
 175 canonical form, 17 , 30 , 33
 176 canonical link function, 568
 177 canonical parameters, 17 , 29 , 33 , 153
 178 CAQL, 1120
 179 cart-pole swing-up, 1145
 180 catastrophic forgetting, 759
 181 categorical, 6
 182 categorical PCA, 940
 183 CatPCA, 940
 184 Cauchy, 10
 185 Cauchy sequence, 670
 186 causal convolution, 831
 187 causal DAGs, 735
 188 causal discovery, 1026
 189 Causal graphs, 1167
 190 causal hierarchy, 180
 191 causal impact, 181 , 729
 192 Causal inference, 1163
 193 causal Markov assumption, 175
 194 causal models, 175
 195 causal prediction, 735
 196 causally sufficient, 175
 197 causes of effects, 180
 198 CAVI, 399
 199 cavity distribution, 443
 200 CCA, 939
 201 cdf, 8
 202 CEB, 216
 203 CelebA, 777 , 792
 204 centering matrix, 86
 205 central composite design, 690
 206 central limit theorem, 449
 207 central moment, 11
 208 certify, 767
 209 ceteris paribus, 181
 210 chain components, 164
 211 chain compositions, 224
 212 chain graph, 156 , 164 , 165
 213 chain rule, 223
 214 chance nodes, 1099
 215 change of variables, 44
 216 changepoint detection, 742 , 981 , 982
 217 channel coding, 183 , 211
 218 channel coding theorem, 379
 219 Chapman-Kolmogorov, 47
 220 Chapman-Kolmogorov equation, 318
 221 characteristic length scale, 657
 222 Chernoff-Hoeffding inequality, 1112
 223 chi-squared distance, 56
 224 Chi-squared distribution, 13
 225 children, 123
 226 Chinese restaurant process, 1034
 227 CHIVI, 439
 228 choice theory, 588
 229 Cholesky decomposition, 451 , 504
 230 Chomsky normal form, 715
 231 chordal, 162 , 388
 232 CI, 123
 233 circuits, 226
 234 circular law, 852
 235 circular normal, 24
 236 citation matching, 174
 237 clamped phase, 157 , 861
 238 class incremental learning, 758
 239 classical statistics, 63
 240 Claude Shannon, 208
 241 click-through rate, 1106 , 1109
 242 clinical trials, 1109
 243 CLIP, 835
 244 clique, 144
 245 cliques, 383
 246 closed world assumption, 173 , 1006
 247 closing the loop, 1002
 248 closure, 154
 249 cluster variational method, 377
 250 clustering, 922
 251 clutter problem, 309
 252 CMA-ES, 1147
 253 CNN, 607
 254 co-information, 203
 255 co-parents, 133
 256 coagulation, 1043
 257 coalescence, 520
 258 cocktail party problem, 953
 259 code words, 208
 260 codebook, 820
 261 codebook loss, 820
 262 codewords, 183

- 1
 2 coffee, lemon, milk, and tea, 278
 3 cold start problem, 65
 4 collapsed, 140
 5 collapsed Gibbs sampler, 478 , 1037
 6 collider, 129 , 1167
 7 collocation, 1144
 8 coloring, 472
 9 commitment loss, 821
 10 common random number, 1146
 11 commutative semi-ring, 392
 12 commutative semiring, 393
 13 compact support, 685
 14 Compactness, Sparsity, 1078
 15 compatible, 1142
 16 complementary log-log, 568
 17 complete, 670
 18 complete data, 157
 19 completely random measures, 1050
 20 Completeness, 1078 , 1080
 21 completing the square, 88
 22 complexity penalty, 114
 23 compiler average treatment effect, 1190
 24 components, 719
 25 composite likelihood, 159
 26 compositional kernel, 700
 27 compositional pattern-producing net-work, 765
 28 Compression Lemma, 190
 29 computation graph, 597
 30 computation tree, 374
 31 concave, 37
 32 concentration inequality, 1112
 33 concentration of measure, 768
 34 concentration parameter, 1030
 35 concept drift, 756
 36 concept shift, 737
 37 concrete distribution, 243
 38 condensation, 519
 39 conditional entropy bottleneck, 216
 40 conditional expected outcome, 1177
 41 conditional GAN, 905
 42 conditional generative model, 771
 43 Conditional generative models, 905
 44 conditional independence, 123
 45 conditional KL divergence, 187
 46 conditional maxent model, 711
 47 conditional parallel trends, 1196
 48 conditional probability distribution, 124
 49 conditional probability table, 125
 50 conditional random eid, 711
 51 conditional random elds, 144 , 146
 52 Conditional shift, 737
 53 conditional SMC, 546
 54 conditional value at risk, 1125
 55 conditionally conjugate, 87
 56 conditioner, 845 , 846
 57 conditioning case, 125
 58 conditioning matrix, 229
 59 conductance, 494
 60 confidence score, 743
 61 conformal prediction, 559 , 742 , 744
 62 conformal score, 559
 63 conformalized quantile regression, 562
 64 confounder, 167
 65 confounders, 1174
 66 conical combination, 843
 67 conjugate, 69 , 303
 68 conjugate gradients, 685
 69 conjugate prior, 19 , 29 , 69 – 71
 70 conjugate-computation variational inference, 426
 71 conjunction of features, 858
 72 consensus sequence, 970
 73 conservative policy iteration, 1140
 74 consistent, 1030
 75 constraint satisfaction problems, 392 , 393
 76 content constrained, 766
 77 context free grammar, 714
 78 context variables, 751
 79 Context., 1065
 80 contextual bandit, 1108
 81 continual learning, 644 , 747 , 756
 82 continuation method, 470
 83 continuing task, 1118
 84 continuous task-agnostic learning, 759
 85 continuous time, 992
 86 continuous-ranked probability score, 728
 87 continuous-time ows, 853
 88 contraction, 1121
 89 contrastive divergence, 158 , 860
 90 Contrastiveness, 1080
 91 control, 1102
 92 control variate, 459 , 506
 93 control variates, 240 , 415
 94 controller, 1117
 95 controls, 991
 96 converge, 494
 97 conversions, 1106
 98 convex combination, 72
 99 ConvNeXt, 607
 100 convolution, 599
 101 convolutional layer, 600
 102 convolutional Markov model, 831
 103 convolutional neural network, 607 , 830
 104 convolutional neural networks, 716
 105 cooling schedule, 471
 106 cooperative cut, 284
 107 coordinate ascent variational inference, 399
 108 core sets, 541
 109 coresnet, 676
 110 correlation coefficient, 17
 111 correspondence, 1005
 112 cosine distance, 24
 113 cosine kernel, 659
 114 count-based exploration, 1128
 115 Counterfactual queries, 1171
 116 counterfactual question, 179
 117 counterfactual reasoning, 728
 118 coupled HMM, 988
 119 coupling ows, 845
 120 coupling layer, 845
 121 covariance function, 653
 122 covariance graph, 166 , 1027
 123 covariance matrix, 16
 124 covariate shift, 736
 125 coverage, 560
 126 Cox process, 674
 127 CPD, 124
 128 CPPN, 765
 129 CPT, 125
 130 CQR, 562
 131 credible interval, 66 , 84
 132 CRF, 144 , 711
 133 CRFs, 146
 134 critic, 888
 135 critical temperature, 147 , 484
 136 cross correlation, 599
 137 cross entropy, 193 , 199
 138 cross entropy method, 1144
 139 cross fitting, 1181
 140 cross validation, 112
 141 cross-entropy method, 543
 142 CRP, 1034
 143 CRPS, 728
 144 CTR, 1109
 145 cubature, 447
 146 cubature Kalman Iter, 355
 147 CUBO, 440
 148 cumulants, 34
 149 cumulative distribution function, 8
 150 cumulative regret, 1115
 151 cumulative reward, 1108
 152 curse of dimensionality, 775 , 1120
 153 curse of horizon, 1153
 154 curved exponential family, 30
 155 CVI, 258 , 426
 156 cycle consistency, 913
 157 cyclical annealing, 810
 158 d-separated, 129
 159 D4PG, 1142
 160 DAGs, 123
 161 DALL-E, 834
 162 damped updates, 402
 163 damping, 374 , 445
 164 dark knowledge, 632
 165 DARN, 129
 166 data assimilation, 352
 167 data association, 1005
 168 data augmentation, 590 , 741
 169 data cleaning, 742
 170 Data compression, 208
 171 data compression, 183 , 775
 172 data generating process, 735
 173 data processing inequality, 191 , 201
 174 data stream classification, 759
 175 data tempering, 515
 176 data-driven MCMC, 468
 177 dataset shift, 733
 178 daydream estimator, 441
 179 daydream phase, 826
 180 DBN, 151 , 990
 181 DCGAN, 907
 182 DDP, 1144
 183 DDPG, 1142
 184 DDPM, 877
 185 dead leaves, 928
 186 decision diagram, 1099
 187 decision nodes, 1099
 188 decision tree, 1101
 189 declarative approach, 175
 190 decoder, 785 , 792 , 921
 191 decomposable, 159 , 162 , 389
 192 decompose, 137
 193 decoupled EKF, 646
 194 deep autoregressive network, 129
 195 deep belief network, 151
 196 deep Boltzmann machine, 151
 197 deep Boltzmann network, 151
 198 deep CCA, 939
 199 deep deterministic policy gradient, 1142
 200 deep ensembles, 629
 201 Deep Factors, 728
 202 deep fakes, 773 , 911
 203 deep Gaussian process, 705
 204 deep generative model, 128
 205 deep generative models, 771
 206 deep image prior, 622
 207 Deep kernel learning, 697
 208 deep latent Gaussian model, 785
 209 deep latent variable model, 785
 210 deep learning, 597
 211 deep Markov model, 1011
 212 deep neural network, 597
 213 deep PILCO, 1146
 214 deep Q-network, 1135
 215 deep state space models, 1011
 216 deep submodular function, 284
 217 deep unfolding, 381
 218 DeepAR, 728
 219 DeepGLO, 728
 220 DeepPSM, 728
 221 default prior, 94
 222 deformable parts model, 717
 223 degenerate kernel, 662
 224 degree of normality, 10
 225 degrees of freedom, 10 , 82 , 115
 226 deleted interpolation, 108
 227 delta function, 66
 228 delta method, 243
 229 delta VAE, 810
 230 demand forecasting, 1007
 231 denoising di usion probabilistic models, 868 , 877
 232 Denoising Score Matching, 865
 233 density estimation, 775
 234 density model, 744
 235 Derivative free optimization, 297
 236 derivative function, 221
 237 derivative operator, 221
 238 detailed balance, 465
 239 detailed balance equations, 54
 240 determinantal point process, 328
 241 Determinantal point processes, 1058
 242 determinantal projection point processes, 1060
 243 deterministic annealing, 974
 244 deterministic inducing conditional, 679
 245 deterministic policy gradient theorem, 1141
 246 deterministic training conditional, 679
 247 deviance, 114
 248 DFO, 297
 249 DGM, 123
 250 DGP, 705
 251 diagonal covariance matrix, 17
 252 diameter, 370
 253 DIC, 679
 254 dictionary learning, 958
 255 di eomorphism, 839
 256 di erence in di erences, 1195

- 1
 2 differential dynamic programming, 1144
 3 differential entropy, 196
 4 di use prior, 94
 5 di usion matrix, 491
 6 di usion models, 771
 7 di usion process, 877
 8 di usion term, 491
 9 digamma function, 409
 10 dilated convolution, 831
 11 diminishing returns, 280
 12 direct cause, 179
 13 direct method, 1150
 14 directed acyclic graphs, 123
 15 directed Gaussian graphical model, 127
 16 directed graphical models, 123
 17 Dirichlet, 27
 18 Dirichlet distribution, 77
 19 Dirichlet process, 480, 1021, 1023, 1030
 20 Dirichlet process mixture models, 411
 21 discount factor, 1108, 1118, 1125
 22 discount parameter, 1041
 23 discrete task-agnostic learning, 759
 24 discrete with probability one, 1032
 25 discriminative model, 549
 26 discriminative reranking, 327
 27 discriminator, 888
 28 disease mapping, 674
 29 disease transmission, 1019
 30 disentangled, 798, 957
 31 dispersion parameter, 38
 32 distillation, 632
 33 distortion, 208
 34 distributed representation, 150, 986
 35 distribution free, 559
 36 distribution shift, 733, 767, 1149
 37 distributional particles, 533
 38 distributional RL, 1136, 1142
 39 distributionally robust optimization, 742
 40 distributive law, 392
 41 divergence metric, 54
 42 DLGM, 785
 43 DLM, 719
 44 DLVM, 785
 45 DNN, 597
 46 DNN factor analysis, 941
 47 do-calculus, 1207
 48 do-notation, 1171
 49 domain adaptation, 740
 50 domain adversarial learning, 740
 51 domain drift, 756
 52 domain generalization, 649, 751, 752
 53 domain randomization, 740, 1148
 54 domain shift, 736
 55 domains, 751
 56 donor, 730
 57 Donsker Varadhan lower bound, 207
 58 Donsker-Varadhan, 190
 59 dot product attention, 603
 60 double descent risk curve, 637, 639
 61 double DQN, 1135
 62 double loop algorithms, 375
 63 double machine-learning, 1179
 64 double Q-learning, 1135
 65 double robust, 1181
 66 double sided exponential, 10
 67 doubly intractable, 157
 68 doubly reparameterized gradient estimator, 438
 69 doubly robust, 1152
 70 doubly stochastic, 417
 71 downstream, 777
 72 DQN, 1135
 73 Dreamer, 1149
 74 drift, 508
 75 dropout, 601
 76 DTC, 679
 77 DualDICE, 1154
 78 dueling DQN, 1135
 79 dyna, 1144
 80 dynamic Bayesian network, 990
 81 dynamic linear model, 335, 719, 997
 82 dynamic programming, 304, 365, 388,
 83 dynamic programming, 386
 84 dynamic topic model, 776
 85 dynamical variational autoencoders, 1011
 86 exchangeable, 100
 87 Exclusion Restriction, 1188
 88 execution traces, 175
 89 exogenous, 175
 90 exp-sine-squared kernel, 659
 91 expanded parameterization, 925
 92 expectation backpropagation, 647
 93 expectation maximization, 248
 94 expectation propagation, 443
 95 expected calibration error, 553
 96 expected complete data log likelihood, 249
 97 expected free energy, 1159
 98 expected improvement, 266
 99 expected LPPD, 113
 100 expected patch log likelihood, 927
 101 expected sufficient statistics, 140, 249
 102 experience replace, 1135
 103 experience replay, 761
 104 explainability, 180
 105 explaining away, 87, 126, 131, 135, 204
 106 explicit duration HMM, 979
 107 explicit layers, 606
 108 explicit probabilistic models, 885
 109 exploration bonus, 1111, 1128
 110 exploration-exploitation tradeo, 1102,
 1109, 1127
 111 exponential cooling schedule, 471
 112 exponential dispersion family, 38
 113 Exponential distribution, 13
 114 exponential family, 28, 29, 39, 92
 115 Exponential family EKF, 345
 116 exponential family factor analysis, 939
 117 exponential family harmonium, 150
 118 exponential family PCA, 939
 119 exponential family state space model, 1006
 120 Exponential linear unit, 599
 121 exponentiated quadratic, 656
 122 extended Kalman Iter, 342, 1000
 123 extended particle Iter, 530
 124 external eld, 148
 125 external validity, 733, 735
 126 extrapolation, 687
 127 extrinsic variables, 406
 128 f-divergence, 55
 129 f-Divergence Max-Ent IRL, 1161
 130 Facebook, 1022
 131 factor, 144
 132 factor analysis, 142
 133 factor graph, 167, 370, 379
 134 factor loading matrix, 142, 929
 135 factor of variation, 798
 136 factor rotations problem, 932
 137 factorial HMM, 434, 986
 138 factorization property, 133
 139 FAIRL, 1161
 140 fairness, 180
 141 Faithfulness, Fidelity, 1078
 142 family marginal, 140
 143 fan-in, 226
 144 fan-out, 223, 226
 145 fantasy data, 862
 146 fast geometric ensembles, 627
 147 fast gradient sign, 764
 148 fast ICA, 956
 149 fast weights, 631
 150 FastSLAM, 536
 151 feature-based, 282
 152 feedback loop, 1109
 153 feedforward neural network, 607
 154 ferromagnetic, 147
 155 few-shot learning, 749, 833
 156 Feynman-Kac, 513
 157 FFG, 169
 158 FFJORD, 854
 159 FFNN, 607
 160 FGS, 764
 161 FIC, 680
 162 FID, 780
 163 II-in, 388
 164 II-in edges, 383
 165 FILM, 606
 166 Iter, 599
 167 Iter response normalization, 601
 168 Itering, 315
 169 FIM, 39
 47

- 1
 nite horizon, 1108
 nite horizon problem, 1118
 nite state machine, 1117
 nite sum objective, 239
 nite-state Markov chain, 47
 rst order stationary, 501
 rst-order delta method, 243
 Fisher divergence, 864
 Fisher information, 95
 Fisher information matrix, 36 , 39 , 95
 FITC, 680
 ited value iteration, 1135
 xed e ects, 593
 Fixed lag smoothing, 316
 at minima, 491 , 633
 uctuation, 508
 folded over, 9
 folds, 112
 FOO-VB, 431
 fooling images, 765
 force, 508
 forest plot, 102
 fork, 129
 Forney factor graph, 169
 Forney factor graphs, 167
 forward adversarial inverse RL, 1161
 forward transfer, 760
 forward-mode automatic di erentiation, 224
 forwards algorithm, 319 , 385 , 971
 forwards Itering backwards sampling, 328
 forwards Itering, backwards smoothing, 321
 forwards kernel, 537
 forwards mapping, 919
 forwards process, 877
 forwards-backwards, 386 , 713
 forwards-backwards algorithm, 321 , 323
 founder variables, 932
 Fourier basis, 959
 Fourier transform, 663
 Fréchet Inception Distance, 780
 fragmentation, 1043
 Fréchet inception distance, 60
 free bits, 810
 free energy, 398
 free energy principle, 1158
 freeze-thaw algorithm, 269
 frequentist sampling distribution, 573
 frequentist statistics, 63
 friction, 508
 front-door criterion, 1210
 front-door formula, 1210
 frustrated, 484
 frustrated system, 147
 full conditional, 133 , 472
 full conditionals, 308
 full conformal prediction, 563
 full covariance matrix, 17
 fully connected CRF, 716
 fully connected layer, 598
 fully independent conditional, 680
 fully independent training conditional, 680
 function space, 667
 functional, 219
 functional causal model, 175
 functional kernel learning, 699
 fundamental problem of causal inference, 181
 funnel shape, 104
 funnel transformer, 806
 fuzzy clustering, 1021
 g-prior, 574
 GAE, 1139
 GAIL, 1161
 GAM, 692
 Gamma, 980
 gamma distribution, 12
 GAN, 771
 GANs, 885
 GaP, 949
 gap, 1115
 GAT, 616
 gated recurrent unit, 609
 gather, 614
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
- Gauss-Hermite integration, 355
 Gaussian bandit, 1111 , 1112
 Gaussian Bayes net, 127
 Gaussian copula, 728
 Gaussian distribution, 8
 Gaussian Iter, 353
 Gaussian Itering, 357
 Gaussian graphical model, 152
 Gaussian kernel, 656
 Gaussian mixture model, 922
 Gaussian MRF, 152
 Gaussian process, 264 , 653 , 1029 , 1030
 Gaussian processes, 428 , 726 , 1145
 Gaussian scale mixture, 252 , 578 , 924
 Gaussian soap bubble, 198
 Gaussian SSMs, 992
 Gaussian sum Iter, 357
 Gaussian VI, 418
 Gaussianizing the likelihood, 429
 GELU, 599
 GEM, 254
 Gen, 175
 GenDICE, 1154
 Generality, 1081
 generalization, 778
 generalized additive model, 692 , 725
 generalized advantage estimation, 1139
 generalized Bayesian inference, 257 , 551
 generalized belief propagation, 377
 generalized bilinear transform, 993 , 1015
 generalized CCA, 939
 generalized distributive law, 392
 generalized EM, 254
 generalized Gauss-Newton, 624
 generalized linear mixed model, 593
 generalized linear model, 565
 generalized low rank models, 940
 generalized online learning, 762
 generalized policy improvement, 1123
 generalized pseudo Bayes Iter, 359
 generalized variational continual learning, 432
 generate and test, 468
 generative adversarial imitation learning, 1161
 Generative Adversarial Networks, 885
 generative adversarial networks, 771
 generative model, 771
 generative weights, 954
 geometric distribution, 7 , 979
 geometric path, 538
 GGM, 152
 GGN, 624
 Gibbs distribution, 146 , 857
 Gibbs point processes, 1057
 Gibbs posterior, 551
 Gibbs sampling, 147 , 309 , 472
 Gittins index, 1111
 Glauber dynamics, 472
 GLIDE, 836
 GLIE, 1132
 GLM, 565
 GLM predictive distribution, 632
 GLMM, 593
 global balance equations, 52
 global latent variables, 302
 global localization, 526
 global Markov property, 129 , 153
 global variables, 136
 globally normalized, 712
 Glorot initialization, 620
 GMM, 922
 GNNs, 612
 goodness-of- t, 54
 GP, 264
 GP-LVM, 941
 GP-MPC, 1146
 GP-SSM, 1010
 GP-UCB, 266
 GPT, 833
 GPT-2, 833
 GPT-3, 833
 gradient descent, 229
 gradient sign reversal, 740
 gradient-based meta-learning, 755
 Gram matrix, 655
 grammar VAE, 809
 grammars, 714
 graph attention network, 616
 Graph Nets, 612
 graph neural networks, 612
 graph surgery, 177 , 1168
 graphical lasso, 153 , 1025
 greatest common divisor, 53
 greedy action, 1120
 grid approximation, 304
 grid world, 1120
 ground network, 172
 ground set, 1058
 ground states, 147
 ground terms, 171
 group lasso, 578
 group normalization, 601
 GRU, 609
 GSM, 924
 guided cost learning, 1161
 guided particle Iter, 528
 Gumbel distribution, 243
 Gumbel-Max trick, 243
 Gumbel-Softmax, 822
 Gumbel-Softmax distribution, 243
 half Cauchy, 10
 half-Cauchy distribution, 579
 half-edge, 169
 half-normal distribution, 9
 Halton sequence, 461
 Hamilton's equations, 485
 Hamiltonian, 484
 Hamiltonian mechanics, 484
 Hamiltonian Monte Carlo, 309 , 484 , 507 , 587
 Hamiltonian Variational Inference, 437
 Hammersley-Cli ord theorem, 145
 Hamming distance, 68
 Hamming loss, 68
 hard clustering, 923
 hard EM, 254
 hard tanh, 244
 hardcore repulsive process, 1058
 harmonic mean estimator, 111
 harmonium, 150
 Hastings correction, 464
 Hawkes processes, 1055
 hazard function, 981
 heat bath, 472
 heavy tailed, 15
 heavy tails, 10 , 15
 Hebb's rule, 861
 Hebbian learning, 861
 Hebbian term, 861
 Hebbian update rule, 338
 Hellinger distance, 56
 Helmholtz machine, 786
 Hessian free optimization, 234
 heuristic function, 1143
 heuristic search, 1144
 hidden Gibbs random eld, 159
 hidden Markov model, 312 , 829 , 961
 hidden semi-Markov model, 979
 hidden state, 829
 hidden variable, 961
 hidden variables, 134 , 248
 hierarchical Bayesian model, 100 , 592 , 649
 hierarchical Bayesian models, 502
 hierarchical Dirichlet process, 1042
 hierarchical generalized linear model, 592
 hierarchical HMM, 984
 hierarchical kernel learning, 692
 hierarchical VAE, 813
 hierarchical variational model, 434
 Hilbert space, 670
 hindsight, 316
 Hinton diagram, 49
 Hinton diagrams, 945
 HiPPO, 1016
 histogram binning, 554
 histogram Iter, 526
 HMC, 309 , 484
 HMM, 961
 HMM Iter, 329
 homogeneous, 46
 homogeneous Poisson process, 1053
 horizon, 317
 horseshoe distribution, 925

- 1
 2 horseshoe prior, 578
 2 HSMM, 979
 3 Human coding, 208
 3 Hungarian algorithm, 975 , 1005
 4 Hutchinson trace estimator, 852
 4 Hutter prize, 208
 5 hybrid MC, 484
 5 hybrid system, 357
 6 hyper-parameters, 71
 6 hypergeometric distribution, 8
 7 hypergraphs, 615
 7 hypernetwork, 605
 8 hyperparameters, 100
 8 hypothesis test, 1105
 9
 10 I-map, 134
 10 I-projection, 442
 11 IAF, 849
 11 IBIS, 541
 12 ICA, 954
 12 ID, 742
 13 identifiable, 1169
 13 identification strategy, 1169
 14 identified, 1169
 14 identity uncertainty, 173
 15 iid, 70
 15 image captioning, 774
 16 image deblurring, 927
 16 image denoising, 927
 17 image imputation, 949
 17 image inpainting, 927
 18 image super-resolution, 927
 18 image to image translation, 911
 19 imagination-augmented agents, 1145
 19 Imbens' Sensitivity Model, 1200
 20 imitation learning, 1160
 20 IMM, 360
 21 implicit generative models, 779 , 885
 21 implicit layers, 606
 22 implicit models, 543 , 771
 22 implicit probabilistic model, 885
 23 implicit probabilistic models, 885
 23 implicit probability distributions, 436
 24 implicit probability model, 799
 24 importance ratio, 1151
 25 importance sampling, 308 , 454
 25 importance weighted autoencoder, 437
 26 importance weighted autoencoders, 456
 26 importance weights, 455
 27 imputation, 775
 27 in-context learning, 833
 28 in-distribution, 742
 29 in-domain, 733
 29 in-painting, 775
 29 Inception, 60
 30 Inception Score, 780
 30 income inequality, 15
 31 incomplete data, 159
 31 incremental EM, 255
 32 incremental importance weights, 516
 33 independence sampler, 466
 33 independent and identically distributed, 70
 34 independent components analysis, 954
 35 indirect cause, 179
 35 induced width, 384
 36 inducing points, 678
 36 inference, 64 , 134 , 301
 37 inference compilation, 826
 37 inference network, 425 , 597 , 785
 38 in node hidden relational model, 1023
 38 in node horizon, 1108
 39 in node mixture model, 1036
 39 in node relational model, 1021 , 1023
 40 in nitely divisible distribution, 1052
 40 in nitely wide neural networks, 621
 41 in uence curve, 1179
 41 in uence diagram, 178 , 1099
 42 in uence model, 989
 42 infomax, 958
 43 InfoNCE, 207
 43 information arc, 1100
 44 information bottleneck principle, 213
 44 information criterion, 114
 45 information diagram, 201 , 204
 45 information diagrams, 201
 46 information Iter, 333
 47
 1 information form, 17 , 33 , 153
 1 information gain, 185 , 267
 1 information processing, 183
 1 information projection, 357 , 442
 1 information state, 1110
 1 informative vector machine, 676
 1 InfoVAE, 799 , 811
 1 inhomogeneous Poisson process, 1053
 1 inner product, 669
 1 innovation, 330
 1 input nodes, 226
 1 inside outside, 984
 1 inside-outside algorithm, 714
 1 instance normalization, 601
 1 instantaneous ELBO, 426
 1 instrument monotonicity, 1191
 1 Instrument Relevance, 1188
 1 instrumental variables, 1187
 1 integer-valued random measure, 1051
 1 integral probability metric, 56
 1 integrating out, 66
 1 inter-causal reasoning, 131
 1 interaction information, 203
 1 interactive multiple models, 360
 1 Interactivity, 1079
 1 interpolated Kneser-Ney, 110
 1 interpolator, 664
 1 intervention, 181
 1 interventions, 177
 1 Interventional queries, 1171
 1 intrinsic uncertainty, 66
 1 intrinsic variables, 406
 1 invariant, 95 , 465
 1 invariant causal prediction, 752
 1 invariant distribution, 51
 1 invariant risk minimization, 753
 1 inventory data, 1009
 1 inverse autoregressive, 849
 1 inverse autoregressive fw, 435
 1 inverse chi-squared distribution, 82
 1 inverse Gamma, 81 , 571
 1 inverse Gamma distribution, 13
 1 inverse mass matrix, 488
 1 inverse optimal control, 1160
 1 inverse probability of treatment weighted estimator, 1178
 1 inverse probability theory, 63
 1 inverse probability transform, 450
 1 inverse reinforcement learning, 1160
 1 inverse temperature, 538
 1 inverse Wishart, 26 , 85 , 87
 1 IPF, 158
 1 IPM, 56
 1 iResNet, 851
 1 IRLS, 247
 1 IRM, 753 , 1023
 1 irreducible, 52
 1 Ising model, 146 , 148
 1 Ising models, 473
 1 isotonic regression, 554
 1 isotropic covariance matrix, 17
 1 iterated batch importance sampling, 541
 1 iterated EKF, 343
 1 iterative amortized inference, 426
 1 iterative proportional tting, 158
 1 IWAE, 437
 1 IWAE bound, 438
 1 jackknife+, 563
 1 Jacobi, 376
 1 Jacobian, 44
 1 Jacobian determinant, 838
 1 Jacobian vector product, 867
 1 Jacobian-vector product (JVP), 221
 1 JamBayes, 1026
 1 Je rey's conditionalization rule, 194
 1 Je rey's prior, 83 , 95
 1 Jensen's inequality, 186 , 248 , 438
 1 JMLR, 358
 1 JPDA, 1005
 1 JTA, 386
 1 jtree, 387
 1 judge sed e ects, 1188
 1 jump Markov linear system, 358
 1 junction graph, 388
 1 junction tree, 386 , 387
 1 junction tree algorithm, 386
 1 junction trees, 1003
 1 K-means clustering, 924
 1 Kalahari, 575
 1 Kalman Iter, 20 , 329 , 335 , 338 , 995
 1 Kalman gain matrix, 330 , 336
 1 Kalman smoother, 20 , 430 , 995
 1 Kalman smoothing, 338
 1 KDE, 775
 1 kernel, 464 , 599
 1 kernel basis function expansion, 581
 1 kernel density estimation, 25 , 775
 1 kernel function, 655 , 670
 1 Kernel Inception Distance, 781
 1 kernel inception distance, 60
 1 kernel mean embedding, 58
 1 kernel PCA, 941
 1 kernel ridge regression, 669 , 671
 1 kernel trick, 58 , 662
 1 keys, 602
 1 KFAC, 233 , 625
 1 kick, 508
 1 kinetic energy, 485
 1 KISS, 692
 1 KISS-GP, 686
 1 KL annealing, 810
 1 KL divergence, 185 , 249
 1 knots, 844
 1 knowledge gradient, 267
 1 knowledge graph, 615
 1 known knowns, 743
 1 known unknowns, 743
 1 Kolmogorov-Smirnov test, 45
 1 kriging, 654
 1 Kronecker Factored Curvature, 625
 1 Krylov subspace methods, 684
 1 Kullback Leibler divergence, 55
 1 Kullback-Leibler divergence, 185 , 249
 1 kurtosis, 11 , 957
 1 L-ensembles, 1059
 1 L₀ norm, 576
 1 L₀ regularization, 576
 1 L₂, 670
 1 L₂ loss, 122
 1 Lévy processes, 1051
 1 Lévy subordinators, 1051
 1 Lévy-Itô decomposition, 1053
 1 Lévy-Kintchine, 1052
 1 label bias, 712
 1 label shift, 737
 1 label smoothing, 555
 1 ladder network, 814
 1 lag, 316 , 500
 1 Lagrange multipliers, 38
 1 Lagrangian, 38
 1 lambda-return, 1131
 1 Lanczos algorithm, 936
 1 Langevin di usion, 490 , 504
 1 Langevin MCMC, 860
 1 Langevin Monte Carlo, 489
 1 language models, 48
 1 LapGAN, 908
 1 Laplace approximation, 305 , 584
 1 Laplace distribution, 10
 1 Laplace Gaussian Iter, 529
 1 Laplace propagation, 444
 1 Laplace's rule of succession, 73
 1 Laplace-EM, 1006 , 1007
 1 lasso, 576 , 577
 1 latent Dirichlet allocation, 480 , 953
 1 latent factors, 919
 1 latent overshooting, 1014
 1 latent space interpolation, 776 , 807
 1 latent variable, 921
 1 latent variable collapse, 817
 1 latent variable model, 919 , 921
 1 Lauritzen-Spiegelhalter, 391
 1 layer, 598
 1 layer normalization, 601
 1 layers, 597
 1 lazy training, 704
 1 LBP, 373
 1 LDA, 480 , 953
 1 LDPC code, 378
 1 LDS, 992
 1 Leaky ReLU, 599
 1 Leapfrog integrator, 486

- 1 learned loss function, 907
 2 learning, 1125
 3 learning from demonstration, 1160
 4 least mean squares, 336
 5 least squares classification, 637
 6 leave-one-out cross validation, 112
 7 LeCun initialization, 620
 8 left-to-right, 324
 9 left-to-right transition matrix, 47
 10 legal reasoning, 180
 11 leptokurtic, 11
 12 level sets, 16
 13 LG-SSM, 992
 14 life-long learning, 756
 15 likelihood function, 64
 16 likelihood ratio, 111 , 745
 17 likelihood tempering, 515
 18 likelihood-free inference, 543 , 886
 19 lilly pads, 963
 20 limiting distribution, 53
 21 linear assignment problem, 975
 22 linear dynamical system, 314 , 992
 23 linear Gaussian CPD, 127
 24 linear Gaussian system, 18
 25 linear layer, 598
 26 linear programming, 1120
 27 Linear regression, 570
 28 linear regression bandit, 1111
 29 Linear State Space Layer, 1015
 30 linear-Gaussian CPD, 165
 31 linear-Gaussian state space model, 314 , 992
 32 linear-Gaussian state space models, 329
 33 linear-quadratic-Gaussian, 1144
 34 linearization point, 221
 35 link function, 565 , 568
 36 Lipschitz constant, 57
 37 LKJ distribution, 94
 38 LM, 48
 39 LMC, 489
 40 LMS, 336
 41 local and global latent variables, 303
 42 local average treatment effect, 1190
 43 local factor, 443
 44 local latent variables, 302
 45 local level model, 720
 46 local linear model, 998
 47 local linear trend, 720
 48 local Markov property, 133
 49 local variables, 136
 50 local+global, 728
 51 localist representation, 150
 52 locally normalized, 163 , 712
 53 locally optimal proposal distribution, 528
 54 location-scale family, 98
 55 log derivative trick, 240 , 414
 56 log loss, 550 , 552
 57 log partition function, 29
 58 log-derivative trick, 158
 59 log-linear model, 152
 60 log-odds score, 971
 61 log-pointwise predictive-density, 113
 62 log-returns, 1003
 63 logical reasoning problems, 393
 64 logistic, 582
 65 logistic distribution, 589 , 955
 66 Logistic regression, 581
 67 logistic regression bandit, 1111
 68 logits, 582
 69 long range dependencies, 1015
 70 long short term memory, 609
 71 long tail, 65
 72 long tails, 15
 73 LOO-CV, 112
 74 loopy belief propagation, 373 , 379
 75 Lorentz, 10
 76 loss function, 120
 77 lossless compression, 208
 78 lossy compression, 208
 79 low discrepancy sequences, 461
 80 low variance resampling, 521
 81 low-density parity check code, 378
 82 low-resource languages, 969
 83 LPPD, 113
 84 LQG, 1144
 85 LSSL, 1015
 86 LSTM, 609
 87 LVM, 921
 88 M step, 248
 89 M-projection, 442
 90 m-separation, 165
 91 M2, 805
 92 M4 forecasting competition, 727
 93 machine translation, 774
 94 MADE, 830
 95 MAF, 848
 96 Mahalanobis distance, 16
 97 majorize-minimize, 245
 98 MALA, 489
 99 MAML, 755
 100 manifestation shift, 737
 101 MAP estimate, 65 , 122 , 317 , 325 , 550
 102 MAPIE, 559
 103 MAR, 804
 104 marginal calibration error, 553
 105 marginal likelihood, 64 , 75 , 76 , 105 , 110 , 189
 106 marginalizing out, 66
 107 Mariner 10, 333
 108 marked, 1056
 109 Markov, 134
 110 Markov assumption, 46 , 311 , 829
 111 Markov blanket, 133 , 154 , 472
 112 Markov chain, 46
 113 Markov chain Monte Carlo, 308 , 463
 114 Markov decision process, 1116
 115 Markov kernel, 46
 116 Markov logic networks, 171
 117 Markov mesh, 144
 118 Markov model, 46 , 829
 119 Markov model of order n, 48
 120 Markov network, 144
 121 Markov process, 1029
 122 Markov random eld, 144
 123 masked attention, 603
 124 masked autoregressive ow, 848
 125 masked convolution, 831
 126 matching, 1182
 127 Matern kernel, 657
 128 matrix determinant lemma, 852
 129 matrix inversion lemma, 330
 130 matrix variate Gaussian, 24
 131 matrix vector multiplication, 684
 132 max marginals, 69 , 371
 133 max-product belief propagation, 371
 134 maxent, 152
 135 maxent prior, 95
 136 maximal clique, 144
 137 maximal weight bipartite matching, 1005
 138 maximization bias, 1133
 139 maximizer of posterior marginals, 69
 140 maximizer of the max marginal, 371
 141 maximizer of the posterior marginal, 371
 142 maximum a posteriori, 122
 143 maximum entropy, 94 , 152
 144 maximum entropy Markov models, 712
 145 maximum entropy model, 38
 146 maximum entropy RL, 1157
 147 maximum expected utility principle, 120
 148 maximum likelihood estimation, 550
 149 maximum mean discrepancy, 57 , 57 , 799 , 893
 150 MBEI, 1128
 151 MBEI-EB, 1128
 152 MBRL, 1142
 153 MCAR, 803
 154 MCEM, 254
 155 MCMC, 308 , 463
 156 MCTS, 1144
 157 MDL, 115
 158 MDP, 1116
 159 mean eld, 307 , 399
 160 mean function, 565 , 653
 161 mean value imputation, 775
 162 measure, 670
 163 measurement step, 330
 164 mediated by, 179
 165 mediators, 1209
 166 MEMMs, 712
 167 MEMO, 747
 168 memory methods, 761
 169 memorylessness, 1054
 170 Mental Model, 1079
 171 Mercer kernel, 653 , 655
 172 Mercer's theorem, 662
 173 merit function, 265
 174 MERL, 1157
 175 message passing, 366 , 614
 176 message passing algorithms, 365
 177 message passing neural network, 612
 178 message passing schedule, 366
 179 messages, 324 , 365
 180 meta-data, 751
 181 meta-learning, 753
 182 Method., 1066
 183 Metropolis Adjusted Langevin Algorithm, 489
 184 Metropolis Hastings, 463 , 469
 185 Metropolis Hastings algorithm, 308
 186 Metropolis within Gibbs, 477
 187 MH, 463
 188 midi format, 833
 189 min-II heuristic, 390
 190 min-max, 897
 191 min-max optimization problem, 742
 192 min-weight heuristic, 390
 193 minibatch, 239
 194 minimal, 29
 195 minimal I-map, 134
 196 minimal representation, 30
 197 minimal sufficient statistic, 202 , 213 , 213
 198 minimally informative prior, 94
 199 minimum description length, 115
 200 minimum mean squared error, 122
 201 minimum weight matching, 975
 202 minorize-maximize, 245
 203 mirror descent, 236 , 238 , 428
 204 missing at random, 804
 205 missing completely at random, 803
 206 missing data, 248 , 250 , 803 , 1023
 207 missing data mechanism, 804
 208 mixed effects, 649
 209 mixed effects model, 593
 210 mixed graph, 165
 211 mixed membership model, 950 , 952
 212 mixed membership stochastic block model, 1021
 213 mixing matrix, 954
 214 mixing time, 463 , 493 , 494
 215 mixing weights, 76
 216 mixture model, 921
 217 mixture of Bernoullis, 924
 218 mixture of beta distributions, 76
 219 mixture of experts, 630 , 858 , 1021
 220 mixture of factor analysers, 943
 221 mixture of Gaussians, 922
 222 mixture of Kalman filters, 533
 223 mixture of truncated basis functions, 393
 224 mixture proposal, 468
 225 ML-PIP, 753
 226 MLE, 550
 227 MLP, 607
 228 MM, 245
 229 MMD, 57 , 57 , 799 , 893
 230 MMD VAE, 799
 231 MMI, 203
 232 MMM, 371
 233 MMSE, 122
 234 Mobius inversion formula, 205
 235 mode, 122
 236 mode collapse, 899
 237 mode connectivity, 634
 238 mode hopping, 900
 239 mode-covering, 189
 240 mode-seeking, 189
 241 model checking, 116
 242 model misspecification, 624
 243 model predictive control, 1143
 244 model-agnostic meta-learning, 755
 245 model-based RL, 1125 , 1127 , 1142
 246 model-free RL, 1125
 247 Modied Euler's method, 486
 248 Modularity, 1078
 249 MoG, 922
 250 molecular graph structure, 809
 251 moment matching, 37 , 107 , 157 , 354 , 360 , 442 , 894
 252 moment parameters, 17 , 30
 253 moment projection, 357 , 442

- 1
 2 monference, 786
 2 monks, 1022
 3 Monte Carlo, 69
 3 Monte Carlo approximation, 308
 4 Monte Carlo control, 1130
 4 Monte Carlo dropout, 602 , 626
 5 Monte Carlo EM, 254
 5 Monte Carlo estimation, 1130
 6 Monte Carlo integration, 447
 6 Monte Carlo intergration, 450
 7 Monte Carlo localization, 526 , 535
 7 Monte Carlo methods, 447
 8 Monte Carlo rollouts, 1126
 8 Monte-Carlo tree search, 1144
 9 moralization, 155 , 162
 9 Mormons, 119
 10 most probable explanation, 373
 motion capture, 942
 11 MPC, 1143
 mPCA, 950
 12 MPE, 373
 MPM, 69 , 371
 13 MQ-CNN, 728
 MRF, 144
 14 multi-armel bandit, 1107
 multi-entity Bayesian networks, 174
 15 multi-head attention, 604
 multi-headed DNN, 759
 16 multi-information, 203
 multi-layer perceptron, 607
 17 multi-level model, 100
 multi-moons, 649
 18 multi-sample ELBO, 438
 multi-scale, 817
 19 multi-stage likelihood, 1009
 multi-target tracking, 173 , 1004
 20 multi-task learning, 649 , 750
 multiclass logistic regression, 581
 21 multigraphs, 615
 multimodal VAE, 800
 22 multinomial distribution, 6
 Multinomial logistic regression, 582
 23 multinomial logistic regression, 581
 multinomial PCA, 950
 24 multinomial probit, 592
 multinomial resampling, 520
 25 multinoulli, 6
 multiple hypothesis tracking, 173 , 359
 26 multiple imputation, 775
 multiple kernel learning, 691
 27 multiple mutual information, 203
 multiple plays, 1108
 28 multiple restarts, 973
 multiple sequence alignment, 971
 29 multiplexer, 172
 multiplicative interactions, 605
 30 multiplicative layers, 605
 multiplicative noise, 925
 31 MultiSWAG, 629
 multivariate Gaussian, 16
 32 multivariate mutual information, 203
 multivariate normal, 16
 33 multivariate probit, 592
 multivariate Student distribution, 23
 34 music transformer, 833
 multilevel model, 592
 35 mutual information, 200
 MVAE, 800
 36 MVG, 24
 MVM, 684
 37 MVN, 16
 myopic, 1118 , 1120
 38
 39 N-BEATS, 728
 N-best list, 327
 40 n-gram model, 48
 NADE, 830
 41 NAGVAC, 626
 NAGVAC-1, 420
 42 naive Bayes classifier, 143
 named entity extraction, 714
 43 named variable, 219
 nats, 188
 44 natural evolutionary strategies, 234
 natural exponential family, 30
 45 natural gradient, 42 , 231 , 956
 natural gradient descent, 39 , 230 , 238 ,
 46 257 , 1141
 47
 Natural Gradient Gaussian variational
 approximation, 420
 natural gradient VI, 428
 natural language processing, 713
 natural parameters, 17 , 29 , 33
 Neal's funnel, 503 , 595
 nearest neighbor data association, 1005
 NEF, 30
 negative binomial, 979
 negative binomial distribution, 7
 negative ELBO, 788
 negative log likelihood, 550 , 778
 negative phase, 157
 negative transfer, 650
 negentropy, 957
 nested dissection order, 390
 nested plates, 143
 nested SMC, 530
 neural architecture search, 623
 neural auto-regressive density estimator, 830
 neural bandit, 1111
 neural CRF, 713
 neural CRF parser, 715
 neural enhanced BP, 381
 neural net kernel, 701
 neural network Gaussian process, 621
 Neural ODE, 854
 neural processes, 756
 neural spike trains, 529
 neural tangent kernel, 703
 neural-linear, 626
 neuron, 598
 NeuTra, 493
 NeuTra HMC, 488
 neutral process, 1050
 Newton's laws of motion, 992
 Newton's method, 229
 NGD, 230
 NGVI, 428
 NICE, 854
 NIW, 87
 NIX, 82
 NLDS, 1000
 NLP, 713
 NMAR, 804
 NMF, 949
 NNGP, 700
 No-U-Turn Sampler, 488
 node potentials, 153
 Noise Conditional Score Network, 870
 Noise Contrastive Estimation, 871
 noisy channel, 211
 noisy channel model, 967
 noisy nets, 1136
 non-centered parameterization, 104 ,
 503 , 596
 non-descendants, 133
 non-factorial prior, 581
 non-linear squared ow, 842
 non-Markovian models, 514
 non-negative matrix factorization, 949
 non-null recurrent, 54
 non-parametric Bayesian models, 920
 non-parametric BP, 373
 non-parametric models, 549
 non-parametrically e cient, 1181
 non-terminals, 714
 nondecreasing, 1052
 noninformative, 94
 nonlinear dynamical system, 1000
 nonparametric copula, 728
 nonparametric models, 653
 nonstationary kernel, 701
 normal distribution, 8
 normal inverse chi-squared, 82
 normal inverse Gamma, 82 , 571
 Normal-inverse-Wishart, 87
 normalization layers, 601
 normalized completely random measures,
 1051
 normalized occupancy distribution, 1123
 normalized random measures (NRMs),
 1051
 normalized stable process, 1051
 normalized weights, 456 , 516
 normalizes, 837
 Normalizing ows, 434
 normalizing ows, 771 , 837
 not missing at random, 804
 noun phrase chunking, 713
 Nouveau VAE, 818
 novelty detection, 742
 NP-hard, 385
 NTK, 703
 nuisance functions, 1179
 nuisance variables, 134 , 373
 null hypothesis, 111
 numerical integration, 447
 NUTS, 488
 NWJ lower bound, 207
 Nyström approximation, 678
 object detection, 716
 objective, 94
 observation model, 962
 observation noise, 995
 observation overshooting, 1013
 Occam factor, 115
 Occam's razor, 945
 occasionally dishonest casino, 313
 occlusion, 717
 occupancy grid, 525
 o-policy, 1132
 o-policy policy-gradient, 1152
 one reinforcement learning, 1150
 OGN, 260
 Olivetti faces dataset, 698
 on-policy, 1132
 one-armed bandit, 1107
 one-shot decision problems, 121
 one-shot learning, 749
 one-step ahead prediction distribution, 319
 one-step-ahead predictive distribution, 356
 online adaptation, 756
 online advertising system, 1109
 online Bayesian inference, 334 , 997
 Online elastic weight consolidation, 433
 online EM, 255
 Online Gauss-Newton, 260
 online learning, 644 , 761
 Online structured Laplace, 433
 online variational inference, 430
 ontological uncertainty, 1021
 ontology, 1023
 OOD, 742
 open class, 714
 open set recognition, 747
 open universe probability models, 174
 open world, 1006
 open world classification, 747
 open world recognition, 737
 OpenGAN, 745
 opportunity cost, 1102
 optimal action-value function, 1119
 optimal partial policy, 1122
 optimal policy, 120 , 1119
 optimal resampling, 534
 optimal state-value function, 1119
 optimal transport, 269
 optimism in the face of uncertainty, 1111
 optimization problems, 219
 optimizer's curse, 1133
 Optimus, 806
 oracle, 262
 ordered Markov property, 123 , 133
 ordinal regression, 591
 ordinary differential equation, 992
 Ornstein-Uhlenbeck process, 658
 orthogonal Monte Carlo, 462
 orthogonal random features, 664
 OUPM, 174
 out-of-distribution detection, 742
 out-of-distribution generalization, 733
 out-of-domain, 733
 outlier detection, 742 , 775
 outlier exposure, 743
 over-complete representation, 30
 over-parameterized models, 637
 overcomplete representation, 958
 overcounting, 366
 overdispersed, 495
 over tting, 73 , 550
 overlap, 1176

1	
2	Pólya urn, 1034
3	PAC-Bayes, 551 , 640
4	padding, 600
5	PageRank, 51
6	paired data, 904
7	pairwise Markov property, 154
8	pairwise potentials, 1057
9	palindromic, 508
10	panel data, 1010 , 1194
11	parallel pre x scan, 324 , 339 , 686 , 726
12	parallel tempering, 495 , 512
13	parallel trends, 1195
14	parallel wavenet, 850
15	parameter learning, 136
16	parameter tying, 46 , 100 , 170
17	parameterized ReLU, 839
18	parametric models, 549
19	parametric prior, 1030
20	parents, 123
21	Pareto distribution, 14
22	pareto index, 15
23	Pareto smoothed importance sampling, 114
24	parity check bits, 212
25	part of speech tagging, 712
26	partial least squares, 938
27	partially directed acyclic graph, 164
28	partially observable Markov decision process, 1117
29	partially observed, 120
30	partially observed data, 250
31	partially observed Markov process, 961
32	partially pooled model, 592
33	particle BP, 373
34	particle Itering, 309 , 513 , 1000 , 1039
35	particle Gibbs sampling, 546
36	particle Gibbs with ancestor sampling, 546
37	particle impoverishment, 517
38	particle independent MH, 545
39	particle marginal Metropolis Hastings, 545
40	particle MCMC, 1010
41	partition function, 29 , 145 , 365 , 857
42	partition of the integers, 1034
43	parts, 949
44	patchGAN, 912
45	path consistency learning, 1155
46	path degeneracy, 520
47	path diagram, 177
48	path sampling, 438
49	pathwise derivative, 241
50	patience, 416
51	PBP, 626 , 647
52	PCFG, 714
53	PCL, 1155
54	PDAG, 164
55	peaks function, 470
56	peeling algorithm, 381
57	PEGASUS, 1147
58	per-decision importance sampling, 1151
59	per-sample ELBO, 426
60	per-step importance ratio, 1151
61	per-step regret, 1114
62	perceptual aliasing, 526 , 919
63	perceptual distance metrics, 780
64	perfect elimination ordering, 389
65	perfect information, 1105
66	perfect intervention, 177
67	perfect map, 161
68	period, 53
69	periodic kernel, 659 , 672
70	permuted MNIST, 759
71	perplexity, 199 , 779
72	persistent contrastive divergence, 863
73	persistent variational inference, 157
74	personalized recommendations, 65
75	perturbation, 221
76	PETS, 1146
77	PGD, 764
78	PGMs, 123
79	phase space, 484
80	phase transition, 148
81	phi-exponential family, 34
82	phone, 985
83	Picard-Lindelöf theorem, 853
84	pictorial structure, 717
85	PILCO, 1145
86	Pilot Studies, 1089
87	
	pinball loss, 561
	pipe, 129
	pipeline, 713
	Pitman-Koopman-Darmois theorem, 37 , 202
	pix2pix, 912
	pixelCNN, 831
	pixelCNN++, 832
	pixelRNN, 832
	PixelSNAIL, 822
	placebo, 1109
	planar ow, 852
	PlanNet, 1148
	planning, 1120 , 1125
	planning horizon, 1143
	plant, 1117
	plates, 142
	Platt scaling, 554
	platykurtic, 11
	PLS, 938
	plug-in approximation, 73
	plug-in estimator, 1150
	plugin approximation, 66
	plutocratic, 15
	PMMH, 545
	PoE, 858
	point estimate, 63 , 65
	point process, 1051
	Poisson, 6
	Poisson process, 1051
	Poisson regression, 567
	policy, 1107 , 1116
	policy evaluation, 1119 , 1122
	policy gradient, 1126
	policy gradient theorem, 1137
	policy improvement, 1122
	policy iteration, 1122
	policy optimization, 1119
	policy search, 1126 , 1136
	Polya's urn, 8
	Polyak-Ruppert averaging, 627
	polymatroid function, 281
	polynomial kernel, 659
	polynomial regression, 575
	polytree, 366
	POMDP, 1117
	pooled, 100
	pooling, 613
	pooling layer, 600
	population shift, 736
	POS, 713
	position-speci c scoring matrix, 971
	positional encoding, 612
	positive de nite, 24
	positive de nite kernel, 655
	positive phase, 157
	possible worlds, 172
	posterior collapse, 412 , 809
	posterior distribution, 64
	posterior expected loss, 120
	posterior inference, 64 , 313
	posterior marginal, 134
	posterior mean, 122
	posterior predictive check, 117
	posterior predictive distribution, 66 , 73
	posterior-predictive p-value, 117
	potential, 21
	potential energy, 485
	potential function, 144
	potential outcome, 180
	Potts model, 149
	Potts models, 473
	power EP, 445
	power law, 14
	power posterior, 623
	PPCA, 933
	PPL, 175
	PPO, 1141
	pre-train and ne-tune, 749
	precision, 8 , 80
	precision matrix, 17 , 33
	preconditioned SGLD, 505
	predict-update cycle, 320
	prediction, 181
	prediction step, 318
	predictive coding, 336 , 1159
	predictive distribution, 317
	predictive entropy search, 267
	predictive model, 549
	predictive sparse decomposition, 958
	predictive state representation, 978
	predictive uncertainty, 66
	preferences, 121
	prequential analysis, 112
	prequential inference, 761
	prescribed probabilistic models, 885
	prevalence shift, 737
	Price's theorem, 235 , 242
	primitive nodes, 226
	primitive operations, 224
	principle of insu cient reason, 95
	prior distribution, 63
	prior predictive distribution, 575
	prior shift, 737
	prioritized experience replay, 1135
	Probabilistic backpropagation, 647
	probabilistic backpropagation, 446 , 626
	probabilistic circuit, 170
	probabilistic ensembles with trajectory sampling, 1146
	Probabilistic graphical models, 123
	probabilistic graphical models, 771
	probabilistic principal components analysis, 933
	probabilistic programming language, 175
	probabilistic programming systems, 544
	probability integral transform, 45
	probability matching, 1113
	probability of improvement, 265
	probability simplex, 27
	probit approximation, 585
	probit function, 588
	probit link function, 568
	probit regression, 588
	procedural approach, 175
	process noise, 644 , 994 , 997
	product density function, 1059
	product of experts, 150 , 802 , 858
	product partition model, 982
	production rules, 714
	pro le HMM, 971
	projected gradient descent, 764
	projecting, 357
	projection, 164
	projection pursuit, 957
	prompts, 749
	propensity score, 1178
	propensity score matching, 1183
	proper scoring rule, 552 , 889
	Properties, 1066
	Prophet, 725
	proposal distribution, 308 , 452 , 454 , 464 , 471
	propose, 464
	protein sequence alignment, 970
	protein-protein interaction networks, 1019
	prototypical networks, 755
	proximal policy optimization, 1141
	proximal update, 238
	pseudo counts, 66 , 71
	pseudo inputs, 678
	pseudo likelihood, 158 , 159
	pseudo marginal, 1010
	pseudo random number generator, 450
	pseudo-marginal methods, 544
	pseudo-observations, 429
	PSIS, 114
	pure exploration, 1115
	pushforward, 839
	pushing sums inside products, 381
	Pyro, 175
	Q-learning, 1126 , 1133
	QALY, 121
	QKF, 355
	QT-Opt, 1120
	quadratic approximation, 305
	quadratic kernel, 662
	quadratic loss, 122
	quadrate, 447
	quadrate EP, 362
	quadrate Kalman Iter, 355
	Qualitative Studies, 1088
	quality-adjusted life years, 121
	quantile loss, 561

- 1
 2 quantile regression, 561 , 728
 3 quantization, 196
 Quasi Monte Carlo, 461
 4 queries, 262
 query, 602
 query nodes, 134
 5
 R-hat, 498
 radar, 1005
 radial basis function kernel, 656
 radon, 593
 rainbow, 1136 , 1142
 random accelerations model, 994
 random assignment with non-compliance, 1188
 random effects, 593
 random node sets, 1006
 random Fourier features, 637 , 664
 random measure, 1046
 random walk kernel, 661
 random walk Metropolis, 308 , 467
 random walk on the integers, 53
 random walk proposal, 471
 randomized control trials, 1172
 Randomized QMC, 461
 15 Rao-Blackwellisation, 458
 Rao-Blackwellised particle Itering, 533
 rare event, 542
 raster scan, 831
 rate, 209
 rate distortion curve, 210
 rational quadratic, 659 , 727
 rats, 101
 RBM, 150
 RBPF, 533
 Real NVP, 854
 real-time dynamic programming, 1122
 receding horizon control, 1143
 recognition network, 425 , 785
 recognition weights, 954
 recommender system, 172
 reconstruction error, 209 , 745
 record linkage, 173
 Rectified linear unit, 599
 recurrent, 53
 recurrent layer, 605
 recurrent neural network, 607 , 829
 recurrent neural networks, 605
 recurrent SSM, 1012
 recursive, 175
 recursive least squares, 334 , 570 , 644 , 997
 recursive nested particle iter, 544
 redundancy, 204 , 212
 reference prior, 99
 refractoriness, 1055
 regime switching Markov model, 964
 Regression discontinuity designs, 1211
 regression estimator, 1150
 regression model, 549
 regressive tax, 411
 regret, 762 , 1105 , 1114 , 1115
 regular, 36 , 53
 regularization methods, 761
 rehearsals, 761
 REINFORCE, 414 , 1126 , 1137
 Reinforcement learning, 1125
 reject action, 122
 rejection sampling, 452
 relational data, 1023
 relational probability models, 171
 relational uncertainty, 172
 relative entropy, 185
 relative risk, 674
 relevance network, 1027
 relevance vector machine, 581
 reliability diagram, 553
 Renewal processes, 1054
 reparameterization gradient, 241
 reparameterization trick, 417 , 625 , 789
 reparameterized VI, 417
 repeated trials, 63
 representation, 213
 Representation learning, 777
 representation learning, 687
 representer theorem, 671
 Reproducing Kernel Hilbert Space, 670
 reproducing property, 670
 47
 resample, 518
 residual, 330
 residual belief propagation, 376
 residual block, 851
 residual connections, 600 , 851
 residual ows, 851
 ResNet, 607
 resource allocation, 1109
 response surface model, 262
 responsibility, 923
 restless bandit, 1109
 restricted Boltzmann machine, 150
 return, 1118
 reverse process, 877
 reverse-model automatic differentiation, 225
 reversible jump MCMC, 509 , 945
 reward, 1102 , 1107
 reward function, 1116
 reward model, 1116
 reward-to-go, 1118
 reweighted wake sleep, 825 , 826
 RFF, 664
 Riccati equations, 331
 rich get richer, 8 , 411 , 1034
 ridge regression, 571 , 573 , 669
 ridgeless regression, 704
 Riemann Manifold HMC, 489
 Riemannian manifold, 231
 risk, 120
 RJMCMC, 509
 RKHS, 670
 RL, 1125
 RLS, 334 , 997
 RMSprop, 261 , 261
 RNNADE, 830
 RNN, 607
 RNN-HSMM, 980
 robust optimization, 767
 robust priors, 92
 robustness analysis, 92
 roll call, 950
 row stochastic matrix, 125 , 961
 RPMs, 171
 RStanARM, 593
 RTS smoother, 338
 Rubin Causal Model, 181
 run length, 981
 running intersection property, 387
 Russian roulette estimator, 852
 RVI, 417
 S4, 1015
 SAC, 1158
 safe policy iteration, 1140
 SAGA-LD, 507
 SAGAN, 907
 sample diversity, 778
 sample ine cient, 1127
 sample quality, 778
 sample size, 70
 sample standard deviation, 84
 sampling distribution, 63 , 573
 sampling with replacement, 7
 sampling without replacement, 8
 sandwich estimate, 439
 SARS-CoV2, 352
 SARSA, 1126 , 1132
 satisfying assignment, 385
 SBED, 1156
 scale-invariant prior, 99
 scaled inverse chi-squared, 14
 scaling-binning calibrator, 554
 SCFGs, 984
 SCM, 175
 SCMC, 541
 scope, 133
 score, 864
 score function, 39 , 41 , 860
 score function estimator, 240 , 414 , 915
 score matching, 864 , 882
 score-based generative models, 868
 seasonality, 721
 second order stationary, 501
 second-order delta method, 243
 segment, 313
 segmental HMM, 980
 selection bias, 132 , 738
 selective prediction, 745
 self attention, 832
 Self Attention GAN, 907
 self-attention, 604
 self-normalized importance sampling, 455
 semantic network, 1023
 semantic segmentation, 716
 semi-amortized VI, 426
 semi-Markov model, 979
 semi-parametric model, 667
 semi-parametrically e cient, 1179
 semi-supervised learning, 805
 semilogical linear trend, 720
 semivariogram, 501
 sensible PCA, 933
 sensitivity analysis, 92
 sensor fusion, 19
 separator, 388
 sequence memoizer, 1043
 sequential Bayesian inference, 309 , 644
 sequential Bayesian updating, 319 , 334 , 997
 sequential decision problem, 1107
 sequential decision problems, 122
 sequential importance sampling, 516
 sequential importance sampling with re-sampling, 517
 sequential model-based optimization, 263
 sequential Monte Carlo, 309 , 513
 sFA, 950
 SFE, 240
 SG-HMC, 507
 SGD, 230
 SGLD, 490 , 505
 SGLD-Adam, 505
 SGLD-CV, 506
 SGRD, 505
 shaded nodes, 142
 Shafer-Shenoy, 391
 sharp minima, 633
 shift equivariance, 600
 shift invariance, 600
 shift-invariant kernel, 662
 shooting, 1144
 shortcut features, 734
 shortest path problems, 1121
 shrinkage, 80 , 102
 sigma point BP, 373
 sigma points, 347
 sigma-VAE, 796
 sigmoid, 582
 sigmoid belief net, 128
 silent state, 969
 sim2real gap, 740
 simple regret, 1114
 simplex factor analysis, 950
 Simplicity, 1081
 Simulability, 1079
 Simulated annealing, 469
 simulated annealing, 262
 simulation-based inference, 543 , 886
 simultaneous localization and mapping, 1001
 single site updating, 477
 single world intervention graph, 181
 singular statistical model, 116
 SIS, 516
 SISR, 517
 site parameters, 428
 site potential, 443
 sketch-rnn, 808
 SKI, 686 , 687
 SKIP, 686
 skip connections, 600 , 811 , 813
 skip-chain CRF, 714
 skip-VAE, 811
 SLAM, 535 , 1001
 SLDS, 358
 SLDS, 533
 sleep phase, 825
 slice sampling, 482
 sliced Fisher divergence, 866
 Sliced Score Matching, 866
 sliding window detector, 716
 slippage, 1002
 slot machines, 1107

- 1
 2 slow weights, 631
 2 SMBO, 263
 3 SMC, 309 , 513
 3 SMC sampler, 513
 4 SMC samplers, 537
 5 SMC 2 , 544
 5 SMC-ABC, 543
 6 SMILES, 809
 6 smoothed Bellman error embedding, 1156
 7 snapshot ensembles, 627
 8 SNGP, 626
 8 Sobol sequence, 461
 9 social networks, 1019
 9 soft actor-critic, 1158
 10 soft clustering, 923 , 1021
 10 soft constraint, 858
 11 soft Q-learning, 1158
 11 soft-thresholding, 411
 12 softmax, 31
 12 softmax function, 582
 12 Softplus, 599
 13 SOLA, 433
 14 SOR, 679
 14 Soundness, 1080
 15 source coding, 183 , 208
 15 source coding theorem, 208
 16 source distributions, 748
 16 source domain, 912
 16 space filling, 461
 17 SPADA, 1005
 18 sparse, 27 , 576
 18 sparse Bayesian learning, 579
 19 sparse coding, 958
 19 sparse factor analysis, 933
 20 sparse GP, 678
 20 sparse GP regression, 683
 20 sparse variational GP, 681
 21 sparsity, 626
 22 sparsity promoting priors, 621
 22 spatial statistics community, 501
 23 spectral density, 524 , 663 , 695 , 995
 23 spectral estimation, 978
 24 spectral estimation method, 1000
 24 spectral mixture kernel, 695
 25 spectral mixture kernels, 663
 25 spelling correction, 967
 26 sphere the data, 956
 26 spherical covariance matrix, 17
 27 spherical cubature integration, 355
 27 spherical K-means algorithm, 24
 28 spherical topic model, 24
 28 spherling, 954
 29 spike and slab, 925
 29 spike-and-slab, 576
 29 spin, 146
 30 splines, 844
 31 split conformal prediction, 563
 31 split MNIST, 759
 32 split-Rhat, 498
 32 splitting, 508
 33 spring mass system, 992
 33 spurious correlation, 734
 34 SQF-RNN, 728
 35 square root Iter, 333
 35 square root information Iter, 333
 35 square-integrable functions, 670
 36 squared exponential, 656
 36 squared exponential (SE) kernel, 656
 37 SS, 576
 38 SSID, 999
 38 SSM, 312 , 719 , 991
 39 Stability, 1078
 39 stacking, 630
 40 standard Brownian motion, 1053
 40 standard error, 66 , 72 , 449
 41 standard error of the mean, 84
 41 standard Normal, 8
 42 state of nature, 120
 42 state space model, 312 , 991
 43 state transition diagram, 47
 43 state-space models, 961
 44 state-value function, 1118
 44 stateful, 605
 45 static calibration error, 554
 45 stationary, 46
 46 stationary distribution, 51
 46 stationary kernels, 656
 47 statistical estimand, 1169
 47 Statistics, 63
 47 steepest descent, 229
 47 stepping out, 482
 47 stepwise EM, 255
 47 stick-breaking construction, 1032
 47 stick-breaking process, 1033
 47 sticking the landing, 418
 47 sticky, 467
 47 stochastic approximation, 254 , 262
 47 stochastic approximation EM, 254
 47 stochastic automaton, 47
 47 stochastic averaged gradient acceleration, 507
 47 stochastic bandit, 1107
 47 stochastic block model, 1020
 47 stochastic computation graph, 244
 47 stochastic context free grammars, 984
 47 stochastic EP, 446
 47 stochastic gradient descent, 230
 47 stochastic gradient Langevin dynamics, 505
 47 Stochastic Gradient Riemannian Langevin Dynamics, 505
 47 stochastic Lanczos quadrature, 685
 47 stochastic local search, 297 , 469
 47 stochastic matrix, 47
 47 stochastic meta descent, 718
 47 stochastic process, 1029
 47 stochastic relaxation, 262
 47 stochastic RNN, 1011
 47 stochastic variance reduced gradient, 506
 47 stochastic variational inference, 307 , 417 , 684
 47 stochastic video generation, 1015
 47 stochastic volatility model, 1003
 47 stochastic weight averaging, 627
 47 stochastically ordered, 1033
 47 stop gradient, 820
 47 straight-through estimator, 244 , 819
 47 stratified resampling, 521
 47 streaks, 313
 47 streaming variational Bayes, 430
 47 strict overlap, 1176
 47 strictly monotonic scalar function, 843
 47 string kernel, 661
 47 structural causal models, 175 , 177 , 1166
 47 structural equation model, 165 , 177
 47 structural time series, 719 , 998
 47 structural zeros, 153
 47 structured conjugate models, 393
 47 structured kernel interpolation, 686
 47 structured mean eld, 434
 47 structured prediction, 711
 47 Structured Prediction Energy Networks, 718
 47 Structured State Space Sequence model, 1015
 47 STS, 719
 47 Student distribution, 9
 47 Student network, 125
 47 student network, 126 , 162 , 381
 47 Student t distribution, 9
 47 style transfer, 913
 47 sub-Gaussian, 11
 47 subjective probability, 123
 47 Submodular, 280
 47 subphones, 985
 47 Subscale Pixel Network, 832
 47 subset of regressors, 679
 47 subspace identication, 999
 47 subspace neural bandit, 1111
 47 su cient, 213
 47 su cient statistic, 46 , 202
 47 su cient statistics, 29 , 29 , 70
 47 sum-product belief propagation, 371
 47 sum-product networks, 170
 47 super-Gaussian, 11
 47 supervised PCA, 937
 47 supply chain, 1009
 47 surrogate function, 245 , 262
 47 survival of the fittest, 519
 47 subset-of-data, 676
 47 SUTVA, 181
 47 SVB, 430
 47 SVG, 1015
 47 SVP, 681
 47 SVI, 417
 47 SVRG-LD, 506
 47 SWA, 627
 47 SWAG, 628
 47 Swendsen Wang, 483
 47 SWIG, 181
 47 Swish, 599
 47 swiss roll, 868
 47 switching linear dynamical system, 358 , 533
 47 Sylvester owl, 852
 47 symamd, 390
 47 symmetric, 464
 47 symplectic, 486
 47 synchronous updates, 376
 47 synergy, 204
 47 syntactic sugar, 142
 47 synthetic control, 730
 47 Synthetic controls, 1211
 47 systems biology, 1025
 47 systems identication, 999
 47 systolic array, 369
 47 T5, 806
 47 tabular representation, 1117
 47 tacotron, 831
 47 TAN, 143
 47 target aware Bayesian inference, 456
 47 target distribution, 451 , 454 , 748
 47 target domain, 912
 47 target function, 454
 47 target network, 1135
 47 target policy, 1150
 47 targeted attack, 764
 47 task, 759
 47 task incremental learning, 758
 47 task interference, 650
 47 task-aware continual learning, 759
 47 Taylor series, 305
 47 Taylor series expansion, 342
 47 TD, 1130
 47 TD error, 1126 , 1130
 47 TD(λ), 1131
 47 TD3, 1142
 47 telescoping sum, 881
 47 temperature, 491
 47 temperature scaling, 554
 47 tempered posterior, 623
 47 tempering, 433
 47 template, 172
 47 templates, 928
 47 temporal di erence, 1126 , 1130
 47 tensor decomposition, 978
 47 tensor train decomposition, 687
 47 terminal state, 1118
 47 terminals, 714
 47 test and roll, 1102
 47 test statistics, 117
 47 test-time training, 747
 47 text to speech, 831
 47 text-to-speech, 914
 47 the deadly triad, 1155
 47 thermodynamic integration, 438
 47 thermodynamic variational objective, 438
 47 thin junction tree Iter, 1003
 47 thin shell, 198
 47 thinning theorem, 1053
 47 Thompson sampling, 266 , 1113
 47 threat model, 766
 47 tilted distribution, 443
 47 time reversal, 538
 47 time reversible, 54
 47 time series forecasting, 719 , 998
 47 time update step, 329
 47 time-invariant, 46
 47 time-series forecasting, 165
 47 Toeplitz, 686
 47 top-down inference model, 813
 47 topic, 1043
 47 topological order, 134
 47 topological ordering, 123
 47 total correlation, 203 , 798
 47 total derivative, 242
 47 total regret, 1115
 47 total variation distance, 60
 47 trace plot, 495
 47 trace rank plot, 496
 47 traceback, 326 , 372
 47 track, 359

- 1
- 2 tracking, 314
2 tracking by detection, 525
3 tractable substructure, 434
4 trajectory, 1125
4 trankplot, 496
5 trans-dimensional MCMC, 509
5 transductive active learning, 558
6 transductive learning, 742
6 transfer learning, 738, 748
6 transformer, 609, 609
7 Transformer-XL, 616
8 transformers, 806, 1015
9 transient, 53
9 transition, 1116
9 transition function, 46
10 transition kernel, 46
10 transition matrix, 47, 47
11 transition model, 961, 1116
11 translation invariant, 959
12 translation-invariant prior, 98
12 Translucence, 1079
13 Transparency, 1076
13 transportable, 735
14 treatment, 178
14 treatment, 1102
15 treatment effect, 730
16 tree decomposition, 387
16 tree-augmented naive Bayes classifier, 143
17 treewidth, 384, 388
17 trellis diagram, 325
18 triangulated, 388
18 triangulation, 388
19 triggering kernel, 1055
19 trigram model, 48
20 TRPO, 1140
20 TrueSkill, 446
21 truncated Gaussian, 590
21 trust region policy optimization, 1140
22 TT-GP, 687
22 turbocodes, 378
23 Turing, 175
23 turning the Bayesian crank, 301
24 TVO, 438
24 twin network, 181
25 two part code, 211
26 two sample tests, 782
26 two stage least squares, 1194
26 two-iter smoothing, 323, 339
27 two-moons, 649
27 two-sample test, 54
28 two-sample testing, 742
28 two-slice marginal, 323
29 type II maximum likelihood, 105
29 type signature, 171
30 typical set, 197, 198
31 UCB, 266, 1111
31 UCBVI, 1128
32 UCRL2, 1128
33 UGM, 144
33 UKF, 347
34 ULA, 490
34 ULD, 507
35 UME, 59
35 Unadjusted Langevin Algorithm, 490
36 unary terms, 148, 153
36 unbiased, 1151
37 uncertainty metric, 743
37 uncertainty quantification, 559
38 unconstrained monotonic neural net-
works, 843
39 underdamped Langevin dynamics, 507
40 underlying predictive model, 982
40 undirected graphical model, 144
41 unfaithful, 154
42 unidentified, 66
43
44
45
46
47
- Uni ed Medical Language System, 1023
uniform dequantization, 779
unigram model, 48
unigram statistics, 50
uninformative, 71, 94
uninformative prior, 572
units, 180
unnormalized mean embedding, 59
unnormalized target distribution, 455
unnormalized weights, 456, 516
unpaired data, 912
unroll, 172
unrolled, 142
unscented Kalman Iter, 347, 1000
unscented Kalman Itering, 373
unscented particle iter, 530
unscented transform, 347
unsupervised domain translation, 912
untargeted attack, 764
update, 614
update step, 318
UPM, 982
upper con dence bound, 266, 1111
user rating pro le model, 950
User studies, 1082
utility function, 120
utility nodes, 1099
v-structure, 129, 130
VAE, 152, 771, 785
VAE-RNN, 806
VAFC, 420
vague prior, 575
validation set, 112
value function, 1118
value iteration, 1121
value nodes, 1099
value of perfect information, 1102
ValueDICE, 1162
values, 602
VAR, 165
variable binding problem, 836
variable duration HMM, 979
variable elimination, 381, 1102
Variational Approximation with Factor Covariance, 420
variational autoencoder, 785
variational autoencoders, 771, 941
variational Bayes, 306, 402
variational Bayes EM, 407
variational continual learning, 430, 648, 761
variational EM, 249, 254, 407
variational free energy, 336, 398, 681, 1158
variational gap, 787
variational GP, 434
variational IB, 214
variational inference, 29, 253, 306, 397, 551, 779
variational message passing, 258, 413
variational method, 397
Variational Online Gauss-Newton, 260, 261
variational online Gauss-Newton, 625
Variational Online Generalized Gauss-Newton, 261
variational optimization, 234, 262
variational over-pruning, 626
variational overpruning, 432, 809
variational parameters, 307, 397
variational pruning, 817
variational pruning effect, 412
variational RNN, 1014
variational SMC, 532
varimax, 933
variogram, 501
VB, 306
VCL, 430, 648
VD-VAE, 814
vector auto-regressive, 165
vector quantization, 208
vector-Jacobian product (VJP), 221
VERSA, 755
very deep VAE, 814
VFE, 681
VI, 306
VIB, 214
VIM, 824
VIREL, 1159
virtual samples, 90
visible nodes, 134
vision as inverse graphics, 919
visual clutter, 525
visual SLAM, 1001
visual tracking, 525
Visualization, 1076
Viterbi algorithm, 317, 325, 971
Viterbi training, 974
VMP, 413
VOGNN, 261
VOGN, 260, 261, 625
von Mises, 24
von Mises-Fisher, 24
VQ-GAN, 824
VQ-VAE, 818
VRNN, 1014
wake phase, 825
wake sleep, 786
wake-phase q update, 826
wake-sleep algorithm, 824
warmup, 488
Wasserstein-1 distance, 57
Watanabe-Akaike information criterion, 116
wavenet, 831
weak marginalization, 360
weak prior, 575
weakly informative, 93
wealth, 15
website testing, 1106
weight degeneracy, 517
weight of evidence, 189
weight perturbation, 261
weight space, 666
weighted conformal prediction, 739
weighted least squares, 253
Weiner process, 1052
Weinstein–Aronszajn identity, 852
well-log dataset, 983
white noise kernel, 672
white noise process, 501, 994
whitebox attack, 764
whitened coordinate system, 231
whitening, 954
widely applicable information criterion, 116
width, 387
Wiener noise, 505, 508
wildcatter, 1099
Wishart, 24
witness function, 57
word error, 372
word2vec, 777
Xavier initialization, 620
XMC-GAN, 836
z-bias, 1204
zero-avoiding, 189
zero-forcing, 189, 441
zero-in ated Poisson, 567, 1009
zero-one loss, 122
zero-shot learning, 750
zero-sum losses, 897
ZIP, 567, 1009
Zipf’s law, 15

Bibliography

- [AA18] D. Amir and O. Amir. "Highlights: Summarizing agent behavior to people". In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. 2018, pp. 1168–1176.
- [AB08] C. Archambeau and F. Bach. "Sparse probabilistic projections". In: NIPS. 2008.
- [AB17] M. Arjovsky and L. Bottou. "Towards principled methods for training generative adversarial networks". In: (2017).
- [AB21] A. N. Angelopoulos and S. Bates. "A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification". In: (July 2021). arXiv: 2107.07511 [cs.LG].
- [Aba] A. Abadie. "Using Synthetic Controls: Feasibility, Data Requirements, and Methodological Aspects". In: J. of Economic Literature () .
- [Abd+18] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. A. Riedmiller. "Maximum a Posteriori Policy Optimisation". In: ICLR. 2018.
- [ABM10] J.-Y. Audibert, S. Bubeck, and R. Munos. "Best Arm Identification in Multi-Armed Bandits". In: COLT. 2010, pp. 41–53.
- [ABV21] S. Akbayrak, I. Bocharov, and B. de Vries. "Extended Variational Message Passing for Automated Approximate Bayesian Inference". en. In: Entropy 23.7 (June 2021).
- [AC17] S. Aminikhahahi and D. J. Cook. "A Survey of Methods for Time Series Change Point Detection". en. In: Inf. Syst. 51.2 (May 2017), pp. 339–367.
- [AC93] J. Albert and S. Chib. "Bayesian analysis of binary and polychotomous response data". In: JASA 88.422 (1993), pp. 669–679.
- [ACB17] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein generative adversarial networks". In: ICML. 2017, pp. 214–223.
- [ACL16] L. Aitchison, N. Corradi, and P. E. Latham. "Zipf's Law Arises Naturally When There Are Underlying Unobserved Variables". en. In: PLoS Comput. Biol. 12.12 (Dec. 2016), e1005110.
- [ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. "Complexity of noding embeddings in a k-tree". In: SIAM J. on Algebraic and Discrete Methods 8 (1987), pp. 277–284.
- [Ada00] L. Adamic. Zipf, Power-laws, and Pareto - a ranking tutorial. Tech. rep. 2000.
- [Ada+20] V. Adam, S. Eleftheriadis, N. Durrande, A. Artemev, and J. Hensman. "Doubly Sparse Variational Gaussian Processes". In: AISTATS. 2020.
- [Ade+20a] J. Adelbayo, J. Gilmer, M. Muell, I. Goodfellow, M. Hardt, and B. Kim. "Sanity Checks for Saliency Maps". arXiv: 1810.03292 [cs.CV].
- [Ade+20b] J. Adelbayo, M. Muell, I. Licardi, and B. Kim. "Debugging tests for model explanations". In: arXiv preprint arXiv:2011.05429 (2020).
- [ADH10] C. Andrieu, A. Doucet, and R. Holenstein. "Particle Markov chain Monte Carlo methods". en. In: J. of Royal Stat. Soc. Series B 72.3 (June 2010), pp. 269–342.
- [Adl] "Understanding Double Descent Requires a Fine-Grained Bias-Variance Decomposition". In: ICML. 2020.
- [Adl+18] P. Adler, C. Falk, S. A. Friedler, T. Nix, G. Rybeck, C. Scheidegger, B. Smith, and S. Venkatasubramanian. "Auditing black-box models for indirect in uence". In: Knowledge and Information Systems 54.1 (2018), pp. 95–122.
- [Ado] Taking It to the MAX: Adobe Photoshop Gets New NVIDIA AI-Powered Neural Filters. <https://blogs.nvidia.com/blog/2020/10/20/adobe-max-ai/>. Accessed: 2021-08-12.
- [AE+20] D. Agudelo-España, S. Gomez-Gonzalez, S. Bauer, B. Schölkopf, and J. Peters. "Bayesian Online Prediction of Change Points". In: UAI. Vol. 124. Proceedings of Machine Learning Research. PMLR, 2020, pp. 320–329.
- [AFD01] C. Andrieu, N. de Freitas, and A. Doucet. "Robust Full Bayesian Learning for Radial Basis Networks". In: Neural Computation 13.10 (2001), pp. 2359–2407.
- [AFG19] M. Akten, R. Flebrink, and M. Grierson. "Learning to See: You Are What You See". In: ACM SIGGRAPH 2019 Art Gallery. SIGGRAPH '19. Los Angeles, California: Association for Computing Machinery, 2019.
- [AG11] A. Allahverdyan and A. Galstyan. "Comparative Analysis of Viterbi Training and Maximum Likelihood Estimation for HMMs". In: NIPS. 2011, pp. 1674–1682.
- [AG13] S. Agrawal and N. Goyal. "Further Optimal Regret Bounds for Thompson Sampling". In: AISTATS. 2013.
- [Agv+14] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. "LASER: a scalable response prediction platform for online advertising". In: WSDM. 2014.
- [Agv+21a] A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun. Reinforcement Learning: Theory and Algorithms. 2021.
- [Agv+21b] R. Agarwal, L. Melnick, N. Frost, X. Zhang, B. Lengerich, R. Caruana, and G. Hinton. Neural Additive Models: Interpretable Machine Learning with Neural Nets. 2021. arXiv: 2004.13912 [cs.LG].
- [AH09] I. Arasaratnam and S. Haykin. "Cubature Kalman Filters". In: IEEE Trans. Automat. Contr. 54.6 (June 2009), pp. 1254–1269.
- [AHE07] I. Arasaratnam, S. Haykin, and R. J. Elliott. "Discrete-Time Nonlinear Filtering Algorithms Using Gauss-Hermite Quadrature". In: Proc. IEEE 95.5 (May 2007), pp. 953–977.
- [AHG20] L. Ambrogioni, M. Hinne, and M. van Gerven. "Automatic structured variational inference". In: (Feb. 2020). arXiv: 2002.00643 [stat.ML].
- [AHK01] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Space". In: Database Theory — ICDT 2001. Springer Berlin Heidelberg, 2001, pp. 420–434.
- [AHK12] A. Anandkumar, D. Hsu, and S. M. Kakade. "A Method of Moments for Mixture Models and Hidden Markov Models". In: COLT. Vol. 23. Proceedings of Machine Learning Research. Edinburgh, Scotland: PMLR, 2012, pp. 33.1–33.34.
- [AHK65] K. Abend, T. J. Harley, and L. N. Kanal. "Classification of Binary Random Patterns". In: IEEE Transactions on Information Theory 11(4) (1965), pp. 538–544.
- [Ahm+17] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. "Unsupervised real-time anomaly detection for streaming data". In: Neurocomputing 262 (Nov. 2017), pp. 134–147.
- [AHp+05] P. K. Agarwal, S. Har-Peled, et al. "Geometric approximation via coresets". In: Combinatorial and computational geometry 52.1–30 (2005), p. 3.
- [AHS85] D. Ackley, G. Hinton, and T. Sejnowski. "A Learning Algorithm for Boltzmann Machines". In: Cognitive Science 9 (1985), pp. 147–169.
- [AHt+07] Y. Altun, T. Hofmann, and I. Tsacharidis. "Support Vector Machine Learning for Interdependent and Structured Output Spaces". In: Predicting Structured Data. Ed. by G. Bakir, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan. MIT Press, 2007.
- [Ahu+21] K. Ahuja, J. Wang, A. Dhurandhar, K. Shanmugam, and K. R. Varshney. "Empirical or Invariant Risk Minimization? A Sample Complexity Perspective". In: ICLR. 2021.
- [AI19] AI Artists. Creative Tools to Generate AI Art. 2019.
- [Air+08] E. Airoldi, D. Blei, S. Fienberg, and E. Xing. "Mixed-membership stochastic blockmodels". In: JMLR 9 (2008), pp. 1981–2014.
- [Ait20] L. Aitchison. "A unified theory of adaptive stochastic gradient descent as Bayesian Iterating". In: (July 2018). arXiv: 1807.07540 [stat.ML].
- [Ait20] L. Aitchison. "Why bigger is not always better: on nite and in nite neural networks". In: ICML. 2020.
- [Ait21] L. Aitchison. "A statistical theory of cold posteriors in deep neural networks". In: ICLR. 2021.
- [Aka74] H. Akaike. "A new look at the statistical model identification". In: IEEE Trans. on Automatic Control 19.6 (1974).
- [AKO18] S.-I. Amari, R. Karakida, and M. Oizumi. "Fisher Information and Natural Gradient Learning of Random Deep Networks". In: (Aug. 2018). arXiv: 1808.07172 [cs.LG].
- [Ale+16] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. "Deep Variational Information Bottleneck". In: ICLR. 2016.
- [Ale+18] A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy. "Fixing a broken ELBO". In: ICML. 2018.

- 1
- 2 [Alg21] P. Alquier. "User-friendly introduction to PAC-Bayes
bounds". In: (Oct. 2021). arXiv: [2110.11216 \[stat.ML\]](#).
- 3 [Als+19] J. Alsing, T. Charnock, S. Feeney, and B. Wandelt.
"Fast likelihood-free cosmology with neural density estimators
and active learning". In: Monthly Notices of the Royal Astro-
nomical Society 488.3 (July 2019), pp. 4440–4458.
- 4 [ALS20] B. Axelrod, Y. P. Liu, and A. Sidford. "Near-optimal
approximate discrete and continuous submodular function
minimization". In: Proceedings of the Fourteenth Annual
ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2020,
pp. 837–853.
- 5 [AM00] S. M. Aji and R. J. McEliece. "The Generalized Dis-
tributive Law". In: IEEE Trans. Info. Theory 46.2 (2000),
pp. 325–343.
- 6 [AM05] E. Amir and S. McIlraith. "Partition-Based Logical Rea-
soning for First-Order and Propositional Theories". In: Arti-
ficial Intelligence 162.1 (2005), pp. 49–88.
- 7 [AM07] R. P. Adams and D. J. C. MacKay. "Bayesian On-
line Changepoint Detection". In: (Oct. 2007). arXiv:
[0710.3742 \[stat.ML\]](#).
- 8 [AM+16] M. Auger-Méthé, C. Field, C. M. Albertsen, A. E.
Derocher, M. A. Lewis, I. D. Jonsen, and J. Mills Flemming.
"State-space models' dirty little secrets: even simple linear
Gaussian models can have estimation problems". In: Sci-
Rep. 6 (May 2016), p. 26677.
- 9 [AM74] D. Andrews and C. Mallows. "Scale mixtures of Normal
distributions". In: J. of Royal Stat. Soc. Series B 36 (1974),
pp. 99–102.
- 10 [AM89] B. D. Anderson and J. B. Moore. Optimal Control: Lin-
ear Quadratic Methods. Prentice-Hall International, Inc., 1989.
- 11 [Ama09] S.-I. Amari. " α -Divergence Is Unique, Belonging to
Both f -Divergence and Bregman Divergence Classes". In: IEEE
Trans. Inf. Theory 55.11 (Nov. 2009), pp. 4925–4931.
- 12 [Ama98] S. Amari. "Natural Gradient Works Efficiently in Learn-
ing". In: Neural Comput. 10.2 (Feb. 1998), pp. 251–276.
- 13 [Ame+19] S. Amersh, D. Weld, M. Vorvoreanu, A. Fournier, B.
Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen,
et al. "Guidelines for human-AI interaction". In: Proceedings of
the 2019 chi conference on human factors in computing sys-
tems. 2019, pp. 1–13.
- 14 [Ami01] E. Amir. "Efficient Approximation for Triangulation of
Minimum Treewidth". In: UAI, 2001.
- 15 [AMJ18a] D. Alvarez-Melis and T. S. Jaakkola. "On the ro-
bustness of interpretability methods". In: arXiv preprint
arXiv:1806.08049 (2018).
- 16 [AMJ18b] D. Alvarez-Melis and T. S. Jaakkola. "Towards robust
interpretability with self-explaining neural networks". In: arXiv
preprint arXiv:1806.07533 (2018).
- 17 [Amo+16] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano,
J. Schulman, and D. Mané. "Concrete Problems in AI Safety".
In: arXiv:1606.06565 (2016). arXiv: [1606.06565](#).
- 18 [Amo+18] B. Amos, L. Dinh, S. Cabri, T. Rothörl, S. G. Col-
menarejo, A. Muldal, T. Erez, Y. Tassa, N. de Freitas, and M.
Denil. "Learning Awareness Models". In: ICLR, 2018.
- 19 [Amo22] B. Amos. "Tutorial on amortized optimization for
learning to optimize over continuous domains". In: (Feb. 2022).
arXiv: [2202.00665 \[cs.LG\]](#).
- 20 [AMO88] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. "Net-
work flows". In: (1988).
- 21 [Ana+14] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M.
Telgarsky. "Tensor Decompositions for Learning Latent Variable
Models". In: JMLR 15 (2014), pp. 2773–2832.
- 22 [And01] J. L. Anderson. "An Ensemble Adjustment Kalman Fil-
ter for Data Assimilation". In: Mon. Weather Rev. 129.12 (Dec.
2001), pp. 2884–2903.
- 23 [Andr03] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan.
"An introduction to MCMC for machine learning". In: Machine
Learning 50 (2003), pp. 5–43.
- 24 [And+20] O. M. Andrychowicz et al. "Learning dexterous in-
hand manipulation". In: Int. J. Rob. Res. 39.1 (Jan. 2020),
pp. 3–20.
- 25 [Ang+18] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer,
and C. Rudin. "Learning Certifiably Optimal Rule Lists for Cat-
egorical Data". In: ICLR, 2018. arXiv: [1704.01701 \[stat.ML\]](#).
- 26 [Ang+20] C. Angermueller, D. Dohan, D. Belanger, R. Deshpande,
K. Murphy, and L. Colwell. "Model-based reinforcement
learning for biological sequence design". In: ICLR, 2020.
- 27 [Ang+21] A. N. Angelopoulos, S. Bates, E. J. Candès, M. I.
Jordan, and L. Lei. "Learn then Test: Calibrating Predictive
Algorithms to Achieve Risk Control". In: (Oct. 2021). arXiv:
[2110.01052 \[cs.LG\]](#).
- 28 [Ani+18] R. Anirudh, J. J. Thiagarajan, B. Kaikhura, and T.
Bremer. "An Unsupervised Approach to Solving Inverse Prob-
lems using Generative Adversarial Networks". In: (May 2018).
arXiv: [1805.07281 \[cs.CV\]](#).
- 29 [Ano19] Anonymous. "Neural Tangents: Fast and Easy In nite
Neural Networks in Python". In: (Sept. 2019).
- 30 [AO03] J.-H. Ahn and J.-H. Oh. "A Constrained EM Algorithm
for Principal Component Analysis". In: Neural Computation 15
(2003), pp. 57–65.
- 31 [AOm17] M. G. Azar, I. Osband, and R. Munos. "Minimax
Regret Bounds for Reinforcement Learning". In: ICML, 2017,
pp. 263–272.
- 32 [AP08] J. D. Angrist and J.-S. Pischke. Mostly harmless eco-
metrics: An empiricist's companion. Princeton university
press, 2008.
- 33 [AP09] J. Angrist and J.-S. Pischke. Mostly Harmless Econo-
metrics, 2009.
- 34 [AP19] M. Abadi and G. D. Plotkin. "A simple differentiable
programming language". In: Proceedings of the ACM on Pro-
gramming Languages 4.POPL (2019), pp. 1–28.
- 35 [AR09] C. Andrieu and G. O. Roberts. "The pseudo-marginal
approach for efficient Monte Carlo computations". In: An-
nals of Statistics 37.2 (Apr. 2009), pp. 697–725.
- 36 [Ara+09] A. Aravkin, B. Bell, J. Burke, and G. Pillonetto. An
L1-Laplace Robust Kalman Smoother. Tech. rep. U. Wash-
ington, 2009.
- 37 [Ara10] A. Aravkin. Student's t Kalman Smoother. Tech. rep.
U. Washington, 2010.
- 38 [Ara+17] A. Aravkin, J. V. Burke, L. Ljung, A. Lozano, and
G. Pillonetto. "Generalized Kalman smoothing: Modeling and
algorithms". In: Automatica 86 (Dec. 2017), pp. 63–86.
- 39 [Arb+18] M. Arbel, D. Sutherland, M. Bikowski, and A. Gret-
ton. "On gradient regularizers for MMD GANs". In: Advances
in neural information processing systems. 2018, pp. 6700–6710.
- 40 [Arj+19] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-
Paz. "Invariant Risk Minimization". In: (July 2019). arXiv:
[1809.02893 \[stat.ML\]](#).
- 41 [Arj+20] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-
Paz. Invariant Risk Minimization. 2020. arXiv:
[1907.02893 \[stat.ML\]](#).
- 42 [Arn+10] C. W. Arnold, S. M. El-Saden, A. A. Bui, and R.
Taira. "Clinical case-based retrieval using latent topic analysis".
In: AMIA annual symposium proceedings. Vol. 2010. American
Medical Informatics Association, 2010, p. 26.
- 43 [Aro+19] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov,
and R. Wang. "On Exact Computation with an Ininitely Wide
Neural Net". In: (Apr. 2019). arXiv: [1904.11955 \[cs.LG\]](#).
- 44 [Aro+21] R. Arora et al. Theory of deep learning. 2021.
- 45 [ARS13] N. S. Arora, S. Russell, and E. Sudderth. "NET-
VISA: Network Processing Vertically Integrated Seismic
AnalysisNET-VISA: Network Processing Vertically Integrated
Seismic Analysis". In: Bull. Seismol. Soc. Am. 103.2A (Apr.
2013), pp. 709–729.
- 46 [Aru+02] M. Arulampalam, S. Maskell, N. Gordon, and
T. Clapp. "A Tutorial on Particle Filters for Nonlinear/Non-Gaussian Bayesian Tracking". In: IEEE Trans.
on Signal Processing 50.2 (2002), pp. 174–189.
- 47 [Aru+17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and
A. A. Bharath. "A Brief Survey of Deep Reinforcement Learn-
ing". In: IEEE Signal Processing Magazine, Special Issue on
Deep Learning for Image Understanding (2017).
- 48 [Ach+17] A. Achille and S. Soatto. "On the Emergence of Invari-
ance and Disentanglement in Deep Representations". In: (June
2017). arXiv: [1706.01350 \[cs.LG\]](#).
- 49 [Ach+18] A. Achille and S. Soatto. "On the Emergence of Invari-
ance and Disentangling in Deep Representations". In: JMLR 18
(2018), pp. 1–34.
- 50 [AS66] S. M. Ali and S. D. Silvey. "A General Class of Coe-
fficients of Divergence of One Distribution from Another". In:
J. Roy. Stat. Soc. Series B Stat. Methodol. 28.1 (1966), pp. 131–
142.
- 51 [Asa00] C. Asavathirathan. "The Influence Model: A Tractable
Representation for the Dynamics of Networked Markov Chains".
PhD thesis. MIT, Dept. EECS, 2000.
- 52 [ASD20] A. Agrawal, D. Sheldon, and J. Domke. "Advances in
Black-Box VI: Normalizing Flows, Importance Weighting, and
Optimization". In: NIPS, June 2020.
- 53 [ASM17] A. Azuma, M. Shimbo, and Y. Matsumoto. "An Al-
gebraic Formalization of Forward and Forward-backward Algo-
rithms". In: (Feb. 2017). arXiv: [1702.06941 \[cs.LG\]](#).
- 54 [ASN20] R. Agarwal, D. Schuurmans, and M. Norouzi. "An Op-
timistic Perspective on Online Reinforcement Learning". In:
ICML, 2020.
- 55 [ASS19] I. Andrews, J. H. Stock, and L. Sun. "Weak Instruments
in Instrumental Variables Regression: Theory and Practice". In:
Annual Review of Economics 11.1 (2019).

BIBLIOGRAPHY

- 1
- 2 [ASS20] B. Adlam, J. Snoek, and S. L. Smith. "Cold Posteriors
and Aleatoric Uncertainty". In: (July 2020). arXiv:
[stat.ML].
- 3 [AT08] C. Andrieu and J. Thoms. "A tutorial on adaptive
MCMC". In: Statistical Computing 18 (2008), pp. 343–373.
- 4 [AT20] E. Agustsson and L. Theis. "Universally Quantized Neu-
ral Compression", 2020.
- 5 [Att00] H. Attias. "A Variational Bayesian Framework for
Graphical Models". In: NIPS-12. 2000.
- 6 [Aue12] J. E. Auerbach. "Automated evolution of interesting im-
ages". In: Artificial Life 13. 2012.
- 7 [AWR17] J. Altschuler, J. Weed, and P. Rigollet. "Near-
linear time approximation algorithms for optimal transport
via Sinkhorn iteration". In: arXiv preprint arXiv:1705.09634
(2017).
- 8 [AXK17] B. Amos, L. Xu, and J. Z. Kolter. "Input convex neural
networks". In: International Conference on Machine Learning.
PMLR. 2017, pp. 146–155.
- 9 [AY19] B. Amos and D. Yarats. "The Differentiable Cross-
Entropy Method". In: (Sept. 2019). arXiv: 1909.12830 [cs.LG].
- 10 [AZ17] S. Arora and Y. Zhang. "Do gans actually
learn the distribution? an empirical study". In: arXiv preprint
arXiv:1706.08224 (2017).
- 11 [Aze+20] E. M. Azevedo, A. Deng, J. L. Montiel Olea, J. Rao,
and E. G. Weyl. "A/B Testing with Fat Tails". In: J. Polit.
Econ. (July 2020), pp. 000–000.
- 12 [BA03] D. Barber and F. Agakov. "The IM Algorithm: A Vari-
ational Approach to Information Maximization". In: NIPS.
NIPS'03. Cambridge, MA, USA: MIT Press, 2003, pp. 201–208.
- 13 [BA05] W. Bechtel and A. Abrahamsen. "Explanation: A mechan-
istic alternative". In: Studies in History and Philosophy of Sci-
ence Part C: Studies in History and Philosophy of Biological
and Biomedical Sciences 36.2 (2005), pp. 421–441.
- 14 [Bac09] F. Bach. "High-Dimensional Non-Linear Variable Selec-
tion through Hierarchical Kernel Learning". In: (Sept. 2009).
arXiv: 0909.0844 [cs.LG].
- 15 [Bac+13] F. Bach et al. "Learning with Submodular Functions:
A Convex Optimization Perspective". In: Foundations and
Trends® in Machine Learning 6.2-3 (2013), pp. 145–373.
- 16 [Bac+15] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R.
Müller, and W. Samek. "On pixel-wise explanations for non-
linear classifier decisions by layer-wise relevance propagation".
In: PLoS one 10.7 (2015), e0130140.
- 17 [Bac+18] E. Bach, J. Dusart, L. Hellerstein, and D. Kletenik.
"Submodular goal value of Boolean functions". In: Discrete Ap-
plied Mathematics 238 (2018), pp. 1–13.
- 18 [Bad+18] M. A. Baddeley, J. R. Zech, L. Oakden-Rayner, B. S.
Glicksberg, M. Liu, W. Gale, M. V. McConnell, B. Percha,
T. M. Snyder, and J. T. Dudley. "Deep Learning Predicts Hip
Fracture using Confounding Patient and Healthcare Variables".
In: CoRR abs/1811.03695 (2018). arXiv: 1811.03695.
- 19 [Bah+20] Y. Bahri, J. Kadmon, J. Pennington, S. Schoenholz, J.
Sohl-Dickstein, and S. Ganguli. "Statistical Mechanics of Deep
Learning". In: Ann. Rev. Condens. Matter Phys. (Mar. 2020).
- 20 [Bai+15] R. Bairi, R. Iyer, G. Ramakrishnan, and J. Bilmes.
"Summarization of multi-document topic hierarchies using sub-
modular mixtures". In: Proceedings of the 53rd Annual Meeting
of the Association for Computational Linguistics and the 7th
International Joint Conference on Natural Language Process-
ing (Volume 1: Long Papers). 2015, pp. 553–563.
- 21 [Bai95] L. C. Baird. "Residual Algorithms: Reinforcement
Learning with Function Approximation". In: ICML. 1995,
pp. 30–37.
- 22 [Bak+17] J. Baker, P. Fearnhead, E. B. Fox, and C. Nemeth.
"Control variates for Stochastic Gradient MCMC". In: (June
2017). arXiv: 1706.05439 [stat.CO].
- 23 [Bal+18] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K.
Tuyis, and T. Graepel. "The mechanics of n-player differentiable
games". In: International Conference on Machine Learn-
ing. PMLR. 2018, pp. 354–363.
- 24 [Ban+05] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra. "Clus-
tering on the unit hypersphere using von Mises-Fisher distribu-
tions". In: JMLR. 2005, pp. 1345–1382.
- 25 [Ban06] A. Banerjee. "On bayesian bounds". In: ICML. 2006,
pp. 81–88.
- 26 [Ban+18] A. Bansal, S. Ma, D. Ramanan, and Y. Sheikh.
"Recycle-gan: Unsupervised video retargeting". In: Proceedings
of the European conference on computer vision (ECCV). 2018,
pp. 119–135.
- 27 [Bar+16] P. Barqué, T. Bagautdinov, F. Fleuret, and P. Fua.
"Principled parallel mean-field inference for discrete random
elds". In: CVPR. 2016, pp. 5848–5857.
- 28 [Bar17] D. Barber. Evolutionary Optimization as a Variational
Method. 2017.
- 29 [Bar19] R. F. Barber, E. J. Candès, A. Ramdas, and R. J. Tib-
shirani. "Predictive inference with the jackknife". In: (May
2019). arXiv: 1905.02928 [stat.ME].
- 30 [Bas+01] S. Basu, T. Choudhury, B. Clarkson, and A. Pentland.
Learning Human Interactions with the Influence Model. Tech.
rep. 539. MIT Media Lab, 2001.
- 31 [Bat+18] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-
Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Ra-
poso, A. Santoro, R. Faulkner, et al. "Relational inductive bi-
ases, deep learning, and graph networks". In: arXiv preprint
arXiv:1806.01261 (2018).
- 32 [Bau+17] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Tor-
ralba. "Network Dissection: Quantifying Interpretability of
Deep Visual Representations". In: Computer Vision and Pat-
tern Recognition. 2017.
- 33 [Bau+18] D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenen-
baum, W. T. Freeman, and A. Torralba. "Gan dissection: Visu-
alizing and understanding generative adversarial networks". In:
arXiv preprint arXiv:1811.10597 (2018).
- 34 [Bau+20] D. Bau, J.-Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou,
and A. Torralba. "Understanding the role of individual units
in a deep neural network". In: Proceedings of the National
Academy of Sciences (2020).
- 35 [Bau+70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A
maximization technique occurring in the statistical analysis of
probabilistic functions in Markov chains". In: The Annals of
Mathematical Statistics 41 (1970), pp. 164–171.
- 36 [Bay+15] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and
J. M. Siskind. "Automatic differentiation in machine learning:
a survey". In: (Feb. 2015). arXiv: 1502.05767 [cs.SC].
- 37 [BB15a] A. Bendale and T. Boult. "Towards Open World Recog-
nition". In: CVPR. 2015.
- 38 [BB15b] J. Bornschein and Y. Bengio. "Reweighted Wake-Sleep".
In: ICLR. 2015.
- 39 [BB17] J. Bilmes and W. Bai. "Deep Submodular Functions". In:
Arxiv abs/1701.08939 (2017).
- 40 [BB18] W. Bai and J. Bilmes. "Greedy is Still Good: Maximiz-
ing Monotone Submodular+Supermodular (BP) Functions". In:
International Conference on Machine Learning (ICML). http://
/proceedings.mlr.press/v80/bail18a.html. Stockholm, Sweden, 2018.
- 41 [BBH12] C. Blundell, J. Beck, and K. A. Heller. "Modelling re-
ciprocating relationships with Hawkes processes". In: Advances
in Neural Information Processing Systems. 2012, pp. 2600–
2608.
- 42 [BBM07] A. Banerjee, S. Basu, and S. Merugu. "Multi-way Clus-
tering on Relation Graphs". In: Proc. SIAM Int'l. Conf. on
Data Mining (SDM). 2007.
- 43 [BBM10] N. Bhatnagar, C. Boutilier, and E. Mossel. The Com-
putational Complexity of Estimating Convergence Time. Tech.
rep. arxiv, 2010.
- 44 [BBM09] J. O. Berger, J. M. Bernardo, and D. Sun. "The For-
mal Definition of Reference Priors". In: Ann. Stat. 37.2 (2009),
pp. 905–938.
- 45 [BBS95] A. G. Barto, S. J. Bradtke, and S. P. Singh. "Learn-
ing to act using real-time dynamic programming". In: AIJ 72.1
(1995), pp. 81–138.
- 46 [BBV11a] R. Benassi, J. Bect, and E. Vazquez. "Bayesian opti-
mization using sequential Monte Carlo". In: (Nov. 2011). arXiv:
1111.4802 [math.OC].
- 47 [BBV11b] R. Benassi, J. Bect, and E. Vazquez. "Robust Gaus-
sian Process-Based Global Optimization Using a Fully Bayesian
Expected Improvement Criterion". In: Int'l. Conf. on Learn-
ing and Intelligent Optimization (LION). Jan. 2011, pp. 176–
190.
- 48 [BC07] D. Barber and S. Chiappa. "Unified inference for varia-
tional Bayesian linear Gaussian state space models". In: NIPS.
2007.
- 49 [BC08] M. Bouloumi and K. L. Clarkson. "Optimal core-sets for
balls". In: Computational Geometry 40.1 (2008), pp. 14–22.
- 50 [BC14] J. Ba and R. Caruana. "Do Deep Nets Really Need to
be Deep?". In: Advances in Neural Information Processing Sys-
tems 27. 2014.
- 51 [BC17] D. Beck and T. Cohn. "Learning Kernels over Strings
using Gaussian Processes". In: Proceedings of the Eighth In-
ternational Joint Conference on Natural Language Processing
(Volume 2: Short Papers). Vol. 2. 2017, pp. 67–73.
- 52 [BC89] D. P. Bertsekas and D. A. Castanon. "The auction al-
gorithm for the transportation problem". In: Annals of Opera-
tions Research 20.1 (1989), pp. 67–96.
- 53 [BC94] P. Baldi and Y. Chauvin. "Smooth online learning algo-
rithms for Hidden Markov Models". In: Neural Computation 6
(1994), pp. 305–316.
- 54 [BCF10] E. Brochu, V. M. Cora, and N. de Freitas. "A Tutorial
on Bayesian Optimization of Expensive Cost Functions, with

- 1 Application to Active User Modeling and Hierarchical Reinforcement Learning". In: (Dec. 2010). arXiv: [1012.2599 \[cs.LG\]](#).
- 2 [BCH20] M. Briers, M. Charalambides, and C. Holmes. "Risk scoring calculation for the current NHSx contact tracing app". In: (May 2020). arXiv: [2005.11057 \[cs.CY\]](#).
- 3 [BCJ20] A. Buchholz, N. Chopin, and P. E. Jacob. "Adaptive Tuning Of Hamiltonian Monte Carlo Within Sequential Monte Carlo". In: Bayesian Anal. (2020).
- 4 [BCN18] L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: SIAM Rev. 60.2 (2018), pp. 223–311.
- 5 [BCNM06] C. Burila, R. Caruana, and A. Niculescu-Mizil. "Model compression". In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. 2006, pp. 535–541.
- 6 [BCVD18] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. "The Bouncy Particle Sampler: A Nonreversible Rejection-Free Markov Chain Monte Carlo Method". In: JASA 113.522 (Apr. 2018), pp. 855–867.
- 7 [BD11] A. Bhattacharya and D. B. Dunson. "Simplex factor model for multivariate unordered categorical data". In: JASA 106.500 (2011).
- 8 [BD87] G. Box and N. Draper. Empirical Model-Building and Response Surfaces. Wiley, 1987.
- 9 [BD92] D. Bayer and P. Diaconis. "Trailing the dovetail shuffle to its lair". In: The Annals of Applied Probability 2.2 (1992), pp. 294–313.
- 10 [BDM09] R. Burkard, M. Dell'Amico, and S. Martello. Assignment Problems. SIAM, 2009.
- 11 [BDM10] M. Briers, A. Doucet, and S. Maskell. "Smoothing algorithms for state-space models". In: Annals of the Institute of Statistical Mathematics 62.1 (2010), pp. 61–89.
- 12 [BDM17] M. G. Bellmère, W. Dabney, and R. Munos. "A Distributional Perspective on Reinforcement Learning". In: ICML. 2017.
- 13 [BDM18] N. Brosse, A. Durmus, and E. Moulines. "The promises and pitfalls of Stochastic Gradient Langevin Dynamics". In: NIPS. Nov. 2018.
- 14 [BDS18] A. Brock, J. Donahue, and K. Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: (Sept. 2018). arXiv: [1809.11096 \[cs.LG\]](#).
- 15 [Bea03] M. Beal. "Variational Algorithms for Approximate Bayesian Inference". Phd thesis. Gatsby Unit, 2003.
- 16 [Bea19] M. A. Beaumont. "Approximate Bayesian Computation". In: Annual Review of Statistics and Its Application 6.1 (2019), pp. 379–403.
- 17 [Béd08] M. Bédard. "Optimal acceptance rates for Metropolis algorithms: Moving beyond 0.234". In: Stochastic Process. Appl. 118.12 (Dec. 2008), pp. 2198–2222.
- 18 [Beh+19] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen. "Invertible Residual Networks". In: ICML. 2019.
- 19 [Bel03] A. J. Bell. "The co-information lattice". In: ICA conference. 2003.
- 20 [Bel+16] M. G. Bellmère, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. "Unifying Count-Based Exploration and Intrinsic Motivation". In: NIPS. 2016.
- 21 [Bel+18] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm. "Mutual Information Neural Estimation". In: ICML Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholm, Sweden: PMLR, 2018, pp. 531–540.
- 22 [Bel+19a] D. Belanger, S. Vora, Z. Mariet, R. Deshpande, D. Dolan, C. Angermueller, K. Murphy, O. Chapelle, and L. Colwell. "Biological Sequence Design using Batched Bayesian Optimization". In: NIPS workshop on ML for the sciences. 2019.
- 23 [Bel+19b] M. Belkin, D. Hsu, S. Ma, and S. Mandal. "Reconciling modern machine-learning practice and the classical bias-variance trade-off". en. In: PNAS 116.32 (Aug. 2019), pp. 15849–15854.
- 24 [Ben13] Y. Bengio. "Estimating or Propagating Gradients Through Stochastic Neurons". In: (May 2013). arXiv: [\[cs.LG\]](#).
- 25 [Ben+19] G. W. Benton, W. J. Maddox, J. P. Salley, J. Albinati, and A. G. Wilson. "Function-space Distributions over Kernels". In: Advances in Neural Information Processing Systems. 2019.
- 26 [Ben+20] K. Benidis et al. "Neural forecasting: Introduction and literature overview". In: (Apr. 2020). arXiv: [\[cs.LG\]](#).
- 27 [Bén+21] C. Bénard, G. Blau, S. Veiga, and E. Scornet. "Interpretable random forests via rule extraction". In: International Conference on Artificial Intelligence and Statistics. PMLR. 2021, pp. 937–945.
- 28 [Ber+16] M. Betancourt. "Hamiltonian Monte Carlo for Hierarchical Models". In: (Dec. 2013). arXiv: [1312.0906 \[stat.ME\]](#).
- 29 [Ber99] A. Berchtold. "The double chain Markov model". In: Comm. Stat. Theor. Methods 28 (1999), pp. 2569–2589.
- 30 [Bes75] J. Besag. "Statistical analysis of non-lattice data". In: The Statistician 24 (1975), pp. 179–196.
- 31 [Bet13] M. Betancourt. "A General Metric for Riemannian Manifold Hamiltonian Monte Carlo". In: Geometric Science of Information. Springer Berlin Heidelberg, 2013, pp. 327–334.
- 32 [Bet17] M. Betancourt. "A Conceptual Introduction to Hamiltonian Monte Carlo". In: (Oct. 2017). arXiv: [1701.02434 \[stat.ME\]](#).
- 33 [BFH75] Y. Bishop, S. Fienberg, and P. Holland. Discrete Multivariate Analysis: Theory and Practice. MIT Press, 1975.
- 34 [BG06] M. Beal and Z. Ghahramani. "Variational Bayesian Learning of Directed Graphical Models with Hidden Variables". In: Bayesian Analysis 1.4 (2006).
- 35 [BG13] M. J. Betancourt and M. Girolami. "Hamiltonian Monte Carlo for Hierarchical Models". In: (Dec. 2013). arXiv: [1312.0906 \[stat.ME\]](#).
- 36 [BG96] A. Becker and D. Geiger. "A su ciently fast algorithm for nding close to optimal junction trees". In: UAI. 1996.
- 37 [BGHM17] J. Boyd-Graber, Y. Hu, and D. Mimno. "Applications of Topic Models". In: Foundations and Trends® in Information Retrieval 11.2–3 (2017), pp. 143–296.
- 38 [BGM17] J. Ba, R. Grosse, and J. Martens. "Distributed Second-Order Optimization using Kronecker-Factored Approximations". In: ICLR. openreview.net, 2017.
- 39 [BGS16] Y. Burda, R. Grosse, and R. Salakhutdinov. "Importance Weighted Autoencoders". In: ICLR. 2016.
- 40 [BTG93] C. Berrou, A. Glavieux, and P. Thitimajashima. "Near Shannon limit error-correcting coding and decoding: Turbo codes". In: Proc. IEEE Int'l. Comm. Conf. (1993).
- 41 [BH11] M.-F. Balcan and N. J. Harvey. "Learning submodular functions". In: Proceedings of the forty-third annual ACM symposium on Theory of computing. 2011, pp. 793–802.
- 42 [BH12] R. G. Brown and P. Y. C. Hwang. Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises. en. 4th ed. Wiley, Feb. 2012.
- 43 [BH20] M. T. Bahadori and D. Heckerman. "Debiasing Concept-based Explanations with Causal Analysis". In: International Conference on Learning Representations. 2020.
- 44 [BH92] D. Barry and J. A. Hartigan. "Product partition models for change point problems". In: Annals of statistics 20 (1992), pp. 260–279.
- 45 [Bha+19] A. Bhadra, J. Datta, N. G. Polson, and B. T. Willard. "Lasso Meets Horseshoe: a survey". In: Bayesian Anal. 34.3 (2019), pp. 405–427.
- 46 [Bha+21] K. Bhatia, N. Kuang, Y. Ma, and Y. Wang. Statistical and computational tradeoffs in variational Bayes: a case study of inferential model selection. Tech. rep. 2021.

BIBLIOGRAPHY

- [BHC19] M. Binkowski, D. Hjelm, and A. Courville. "Batch weight for domain adaptation with mass shift". In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 1844–1853.
- [BHP02] M. Brödlou, S. Har-Peled, and P. Indyk. "Approximate clustering via core-sets". In: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing. 2002, pp. 250–257.
- [BHW16] P. G. Bissiri, C. Holmes, and S. Walker. "A General Framework for Updating Belief Distributions". In: JRSSB 78.5 (2016), 1103–1130.
- [Bic09] D. Bickson. "Gaussian Belief Propagation: Theory and Application". PhD thesis. Hebrew University of Jerusalem, 2009.
- [Bie06] G. J. Bierman. Factorization Methods for Discrete Sequential Estimation (Dover Books on Mathematics). en. Illustrated edition. Dover Publications, May 2006.
- [Big+11] B. Biggio, G. Fumera, I. Pillai, and F. Roli. "A survey and experimental evaluation of image spam filtering techniques". In: Pattern recognition letters 32.10 (2011), pp. 1436–1446.
- [Bil01] J. A. Bilmes. Graphical Models and Automatic Speech Recognition. Tech. rep. UWEETR-2001-0005. Univ. Washington, Dept. of Elec. Eng., 2001.
- [Bil22] J. Bilmes. "Submodularity In Machine Learning and Artificial Intelligence". In: (2022). arXiv: 2202.00132 [cs.LG].
- [Bi+18] M. Bickowski, D. J. Sutherland, M. Arbel, and A. Gretton. "Demystifying MMD GANs". In: ICLR. 2018.
- [Bin+18] M. Bickowski, D. J. Sutherland, M. Arbel, and A. Gretton. "Demystifying MMD GANs". In: International Conference on Learning Representations. 2018.
- [Bin+19] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletos, R. Singh, P. Szepit, P. Horsfall, and N. D. Goodman. "Pyro: Deep Universal Probabilistic Programming". In: JMLR 20.28 (2019), pp. 1–6.
- [Bi+19] M. Bickowski, J. Donahue, S. Dieleman, A. Clark, E. Elsen, N. Casagrande, L. C. Cobo, and K. Simonyan. "High Fidelity Speech Synthesis with Adversarial Networks". In: International Conference on Learning Representations. 2019.
- [Bin+97] J. Binder, D. Koller, S. J. Russell, and K. Kanazawa. "Adaptive Probabilistic Networks with Hidden Variables". In: Machine Learning 29 (1997), pp. 213–244.
- [Bis06] C. Bishop. Pattern recognition and machine learning. Springer, 2006.
- [Bis99] C. Bishop. "Bayesian PCA". In: NIPS. 1999.
- [Bit16] S. Bitzer. The UKF exposed: How it works, when it works and when it's better to sample. 2016.
- [Bit+21] J. Bitterwolf, A. Meinke, M. Augustin, and M. Hein. "Revisiting out-of-distribution detection: A simple baseline is surprisingly effective". In: ICML Workshop on Uncertainty in Deep Learning (UDL). 2021.
- [BJ05] F. Bach and M. Jordan. A probabilistic interpretation of canonical correlation analysis. Tech. rep. 688. U. C. Berkeley, 2005.
- [BJ+06] D. M. Blei, M. I. Jordan, et al. "Variational inference for Dirichlet process mixtures". In: Bayesian analysis 1.1 (2006), pp. 121–143.
- [BJ06] W. Buntine and A. Jakulin. "Discrete Component Analysis". In: Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop. 2006.
- [BK01] Y. Boykov and V. Kolmogorov. "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Computer Vision". In: Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition. 2001.
- [BK10] R. Bardenet and B. Kegl. "Surrogating the surrogate: accelerating Gaussian-process-based global optimization with a mixture cross-entropy algorithm". In: ICML. 2010.
- [BK15] D. Belanger and S. Kakade. "A Linear Dynamical System Model for Text". en. In: ICML. June 2015, pp. 833–842.
- [BK19] M. Bonvini and E. H. Kennedy. "Sensitivity Analysis via the Proportion of Unmeasured Confounding". In: arXiv e-prints, arXiv:1912.02793 (Dec. 2019), arXiv:1912.02793. arXiv: 1912.02793 [stat.ME].
- [BK817] O. Bastani, C. Kim, and H. Bastani. "Interpreting blackbox models via model extraction". In: arXiv preprint arXiv:1705.08504 (2017).
- [BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer Normalization". In: (2016). arXiv: 1607.06450 [stat.ML].
- [BKJ13] T. Broderick, B. Kulis, and M. Jordan. "MAD-Bayes: MAP-based asymptotic derivations from Bayes". In: International Conference on Machine Learning. PMLR. 2013, pp. 226–234.
- [BKM16] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. "Variational Inference: A Review for Statisticians". In: JASA (2016).
- [BKS19] A. Bennett, N. Kallus, and T. Schnabel. "Deep Generalized Method of Moments for Instrumental Variable Analysis". In: Advances in Neural Information Processing Systems. 2019, pp. 3564–3574.
- [BL06] D. Blei and J. Laerty. "Dynamic topic models". In: ICML. 2006, pp. 113–120.
- [BLBV12] N. Boulaenger-Lewandowski, Y. Bengio, and P. Vincent. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: ICML. June 2012.
- [BLC18] A. J. Bone, H. Ling, and Y. Cao. "Adversarial Contrastive Estimation". In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2018, pp. 1021–1032.
- [Ble12] D. M. Blei. "Probabilistic topic models". In: Commun. ACM 55.4 (2012), pp. 77–84.
- [Ble17] D. Blei. Variational inference: foundations and innovations (Lecture). Simons Institute Lecture. 2017.
- [BLM16] S. Boucheron, G. Lugosi, and P. Massart. Concentration Inequalities: A Nonasymptotic Theory of Independence. Oxford University Press, 2016.
- [Blo+16] A. Bloniarz, H. Liu, C.-H. Zhang, J. S. Sekhon, and B. Yu. "Lasso adjustments of treatment effect estimates in randomized experiments". In: Proceedings of the National Academy of Sciences 113.27 (2016), pp. 7383–7390.
- [BLS11] G. Blanchard, G. Lee, and C. Scott. "Generalizing from several related classification tasks to a new unlabeled sample". In: NIPS. 2011.
- [BLS17] J. Ballé, V. Laparra, and E. P. Simoncelli. "End-to-end Optimized Image Compression". In: ICLR. 2017.
- [Blu+15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. "Weight Uncertainty in Neural Networks". In: ICML. May 2015.
- [BM+18] G. Barth-Maron, M. W. Ho man, D. Budden, W. Dabney, D. Horgan, A. B. Dhruva, A. Muldal, N. Heess, and T. Lillicrap. "Distributed Distributional Deterministic Policy Gradients". In: ICLR. 2018.
- [BM19] Y. Blau and T. Michaeli. "Rethinking Lossy Compression: The Rate-Distortion-Perception Tradeoff". In: ICML. 2019.
- [BM+73] D. Blackwell, J. B. MacQueen, et al. "Ferguson distributions via Polya urn schemes". In: The annals of statistics (1973), pp. 353–356.
- [BMK20] Z. Borsos, M. Mutny, and A. Krause. "Coresets via Bilevel Optimization for Continual Learning and Streaming". In: Advances in Neural Information Processing Systems 33 (2020).
- [BMP19] A. Brunel, D. Mazza, and M. Pagani. "Backpropagation in the Simply Typed Lambda-Calculus with Linear Negation". In: Proc. ACM Program Lang. 4.POPL (Dec. 2019).
- [BMR97a] J. Binder, K. Murphy, and S. Russell. "Space-e cient inference in dynamic probabilistic networks". In: IJCAI. 1997.
- [BMR97b] S. Bistarelli, U. Montanari, and F. Rossi. "Semiring-Based Constraint Satisfaction and Optimization". In: JACM 44.2 (1997), pp. 201–236.
- [BMS11] S. Bubeck, R. Munos, and G. Stoltz. "Pure Exploration in Finitely-armed and Continuous-armed Bandits". In: Theoretical Computer Science 412.19 (2011), pp. 1832–1852.
- [BNJ03a] D. Blei, A. Ng, and M. Jordan. "Latent Dirichlet allocation". In: JMLR 3 (2003), pp. 993–1022.
- [BNJ03b] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation". In: JMLR 3 (2003), pp. 993–1022.
- [Boh92] D. Boening. "Multinomial logistic regression algorithm". In: Annals of the Inst. of Statistical Math. 44 (1992), pp. 197–200.
- [Bol89] K. Bollen. Structural Equation Models with Latent Variables. John Wiley & Sons, 1989.
- [Bon64] G. Bonnet. "Transformations des signaux aléatoires à travers les systèmes non linéaires sans mémoire". In: Annales des Télécommunications 19 (1964).
- [Bor16] A. Borodin. "Determinantal point processes". In: The Oxford Handbook of Random Matrix Theory.
- [Bor16] S. M. Boraodachev. "Recursive least squares method of regression coefficients estimation as a special case of Kalman Iter". In: Int'l. Conf. of numerical analysis and applied mathematics. Vol. 1738. American Institute of Physics, 2016, p. 110013.
- [Bös+17] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. "Probabilistic Demand Forecasting at Scale". In: Proceedings VLDB Endowment 10.12 (Aug. 2017), pp. 1694–1705.
- [Bot+13] L. Bottou, J. Peters, J. Quiñonero-Candela, D. X. Charles, D. M. Chickering, E. Portugaly, D. Ray, P. Simard, and E. Snelson. "Counterfactual Reasoning and Learning Sys-

- tems: The Example of Computational Advertising". In: JMLR 14 (2013), pp. 3207–3260.
- [Bow+16a] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. "Generating Sentences from a Continuous Space". In: CONLL. 2016.
- [Bow+16b] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. "Generating Sentences from a Continuous Space". In: CONLL. 2016.
- [BP13] K. A. Bollen and J. Pearl. "Eight Myths About Causality and Structural Equation Models". In: Handbook of Causal Analysis for Social Research. Ed. by S. L. Morgan. Dordrecht: Springer Netherlands, 2013, pp. 301–328.
- [BP16] E. Bareinboim and J. Pearl. "Causal inference and the data-fusion problem". en. In: Proc. Natl. Acad. Sci. U. S. A. 113(27) (July 2016), pp. 7345–7352.
- [BP21] T. Bricken and C. Pehlevan. "Attention Approximates Sparse Distributed Memory". In: NIPS. May 2021.
- [BP92] J. Blair and B. Peyton. An introduction to chordal graphs and clique trees. Tech. rep. Oak Ridge National Lab, 1992.
- [BPK16] S. Bulo, L. Porzi, and P. Koutscheder. "Distillation dropout". In: ICML. 2016.
- [BPK18] M. Benatan and E. O. Pyzer-Knapp. "Practical Considerations for Probabilistic Backpropagation". In: NIPS Workshop on Bayesian Deep Learning. 2018.
- [BPS16] A. G. Baydin, B. A. Pearlmutter, and J. M. Siskind. "Di Sharp: An AD library for .NET languages". In: arXiv preprint arXiv:1611.03423 (2016).
- [BR05] H. Bang and J. M. Robins. "Doubly Robust Estimation in Missing Data and Causal Inference Models". In: Biometrics 61.4 (2005), pp. 962–973.
- [BR18] B. Biggio and F. Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: Pattern Recognition 84 (2018), pp. 317–331.
- [BR98] S. Brooks and G. Roberts. "Assessing convergence of Markov Chain Monte Carlo algorithms". In: Statistics and Computing 8 (1998), pp. 319–335.
- [Bra+18] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. JAX: composable transformations of Python+NumPy programs. Version 0.1.55. 2018.
- [Bra96] M. Brand. Coupled hidden Markov models for modeling interacting processes. Tech. rep. 405. MIT Lab for Perceptual Computing, 1996.
- [Bre01] L. Breiman. "Statistical modeling: The two cultures (with comments and a rejoinder by the author)". In: Statistical science 16.3 (2001), pp. 199–231.
- [Bre+17] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Routledge, 2017.
- [Bre+20a] J. Breher, G. Louppe, J. Pavé, and K. Cranmer. "Mining gold from implicit models to improve likelihood-free inference". en. In: Proc. Natl. Acad. Sci. U. S. A. 117.10 (Mar. 2020), pp. 5242–5249.
- [Bre+20b] R. Brekelmans, V. Masrani, F. Wood, G. Ver Steeg, and A. Galstyan. "All in the Exponential Family: Bregman Duality in Thermodynamic Variational Inference". In: ICML. 2020.
- [Bre67] L. M. Breiman. "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming". In: USSR Computational Mathematics and Mathematical Physics 7.3 (Jan. 1967), pp. 200–217.
- [Bre91] Y. Brenier. "Polar factorization and monotone rearrangement of vector-valued functions". In: Communications on pure and applied mathematics 44.4 (1991), pp. 375–417.
- [Bre92] J. Breese. "Construction of belief and decision networks". In: Computational Intelligence 8 (1992), 624–647.
- [Bre96] L. Breiman. "Stacked regressions". In: Mach. Learn. 24.1 (July 1996), pp. 49–64.
- [BRG20] R. Bai, V. Rockova, and E. I. George. "Spike-and-slab meets LASSO: A review of the spike-and-slab LASSO". In: (Oct. 2020), arXiv: 2010.06451 [stat.ME].
- [Bri50] G. W. Brier. "Verification of forecasts expressed in terms of probability". In: Monthly Weather Review 78.1 (Jan. 1950), pp. 1–3.
- [Bro09] G. Brown. "A new perspective on information theoretic feature selection". In: AISTATS. 2009.
- [Bro+13] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan. "Streaming Variational Bayes". In: NIPS. 2013.
- [Bro+15] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. "Inferring causal impact using Bayesian structural time-series models". In: Ann. Appl. Stat. 9.1 (2015), pp. 247–274.
- [Bro18] T. Broderick. Tutorial: Variational Bayes and Beyond. 2018.
- [Bro19] J. Brownlee. Generative Adversarial Networks with Python. Accessed: 2019-8-27. Machine Learning Mastery, Sept. 2019.
- [Bro+20a] D. Brown, R. Coleman, R. Srinivasan, and S. Nieku. "Safe imitation learning via fast bayesian reward inference from preferences". In: International Conference on Machine Learning. PMLR, 2020, pp. 1165–1177.
- [Bro+20b] T. B. Brown et al. "Language Models are Few-Shot Learners". In: (May 2020), arXiv: 2005.14165 [cs.CL].
- [BRS17] E. Balkanski, A. Rubinstein, and Y. Singer. "The Limitations of Optimization from Samples". In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. STOC 2017. Montreal, Canada: Association for Computing Machinery, 2017, 1016–1027.
- [BRS18] N. Bou-Rabee and J. M. Sanz-Serna. "Geometric integrators and the Hamiltonian Monte Carlo method". In: Acta Numer. (2018).
- [Bru+18] M. Brundage et al. "The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation". In: (Feb. 2018), arXiv: 1802.07228 [cs.AI].
- [BS17] E. Balkanski and Y. Singer. "Minimizing a Submodular Function from Samples". In: NIPS. 2017, pp. 814–822.
- [BS18] S. Barratt and R. Sharma. "A note on the inception score". In: arXiv preprint arXiv:1801.01973 (2018).
- [BS20] E. Balkanski and Y. Singer. "A lower bound for parallel submodular minimization". In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. 2020, pp. 130–139.
- [BS21] S. Bubeck and M. Sellke. "A Universal Law of Robustness via Isoperimetry". In: (May 2021), arXiv: 2105.12806 [cs.LG].
- [BS95] A. J. Bell and T. J. Sejnowski. "An information maximization approach to blind separation and blind deconvolution". In: Neural Computation 7.6 (1995), pp. 1129–1159.
- [BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. "Neurolike adaptive elements that can solve difficult learning control problems". In: SMC 13.5 (Sept. 1983), pp. 634–646.
- [BSF88] Y. Bar-Shalom and T. Fortmann. Tracking and data association. Academic Press, 1988.
- [BSL93] Y. Bar-Shalom and X. Li. Estimation and Tracking: Principles, Techniques and Software. Artech House, 1993.
- [BSWT11] Y. Bar-Shalom, P. K. Willett, and X. Tian. Tracking and Data Fusion: A Handbook of Algorithms. en. Yaakov Bar-Shalom, Apr. 2011.
- [BT00] C. Bishop and M. Tipping. "Variational relevance vector machines". In: UAI. 2000.
- [BT03] A. Beck and M. Teboulle. "Mirror descent and nonlinear projected subgradient methods for convex optimization". In: Operations Research Letters 31.3 (2003), pp. 167–175.
- [BT73] G. Box and G. Tiao. Bayesian inference in statistical analysis. Addison-Wesley, 1973.
- [BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. Robust optimization. Vol. 28. Princeton University Press, 2009.
- [Buc+12] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. "A tight (1/2) linear-time approximation to unconstrained submodular maximization". In: FOCS (2012).
- [Buc+17] C. L. Buckley, C. S. Kim, S. McGregor, and A. K. Selbst. "The free energy principle for action and perception: A mathematical review". In: J. Math. Psychol. 81 (Dec. 2017), pp. 55–79.
- [Bul11] A. D. Bull. "Convergence rates of efficient global optimization algorithms". In: JMLR 12 (2011), 2879–2904.
- [Bul+20] S. Bulusu, B. Kalikhura, B. Li, P. K. Varshney, and D. Song. "Anomalous Example Detection in Deep Learning: A Survey". In: IEEE Access 8 (2020), pp. 132330–132347.
- [BV04] S. Boyd and L. Vandenberghe. Convex optimization. Cambridge, 2004.
- [BVHP18] S. Beery, G. Van Horn, and P. Perona. "Recognition in terra incognita". In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, pp. 456–473.
- [BVW02] H. Bui, S. Venkatesh, and G. West. "Policy Recognition in the Abstract Hidden Markov Model". In: JAIR 17 (2002), pp. 451–499.
- [BVZ01] Y. Boykov, O. Veksler, and R. Zabih. "Fast Approximate Energy Minimization via Graph Cuts". In: IEEE PAMI 23.11 (2001).

BIBLIOGRAPHY

- 1 [BVZ99] Y. Boykov, O. Veksler, and R. Zabih. "Fast Approximate Energy Minimization via Graph Cuts". In: ICCV (1). 1999, pp. 377–384.
- 2 [BW20] G. J. J. van den Burg and C. K. I. Williams. "An Evaluation of Change Point Detection Algorithms". In: (Mar. 2020). arXiv: 2003.06222 [stat.ML].
- 3 [BW21] G. J. van den Burg and C. K. Williams. "On Memorization in Probabilistic Deep Generative Models". In: NIPS. 2021.
- 4 [BWM18] A. Buchholz, F. Wenzel, and S. Mandt. "Quasi-Monte Carlo Variational Inference". In: ICML. 2018.
- 5 [BWR16] M. Bauer, M. van der Wilk, and C. E. Rasmussen. "Understanding Probabilistic Sparse Gaussian Process Approximations". In: NIPS. 2016, pp. 1533–1541.
- 6 [BYH20] O. Bohdai, Y. Yang, and T. Hospedales. "Flexible Dataset Distillation Learn Labels Instead of Images". In: arXiv preprint arXiv:2006.08572 (2020).
- 7 [BYM17] D. Belanger, B. Yang, and A. McCallum. "End-to-End Learning for Structured Prediction Energy Networks". In: ICML. Ed. by D. Precup and Y. W. Teh, Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 429–439.
- 8 [Byr+16] R. Byrd, S. Hansen, J. Nocedal, and Y. Singer. "A Stochastic Quasi-Newton Method for Large-Scale Optimization". In: SIAM J. Optim. 26.2 (Jan. 2016), pp. 1008–1031.
- 9 [BZ20] A. Barbu and S.-C. Zhu. Monte Carlo Methods. en. Springer, 2020.
- 10 [CA13] E. F. Camacho and C. B. Alba. Model predictive control. Springer, 2013.
- 11 [Cac+18] M. Caccia, L. Caccia, W. Fedus, H. Larochelle, J. Pineau, and L. Charlin. "Language GANs Falling Short". In: CoRR abs/1811.02549 (2018). arXiv: 1811.02549.
- 12 [CAI20] V. Coscato, M. H. de Almeida Inácio, and R. Izbicki. "The NN-Stacking: Feature weighted linear stacking through neural networks". In: Neurocomputing (2020).
- 13 [Cal+07] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. "Maximizing a submodular set function subject to a matroid constraint". In: Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO). 2007, pp. 182–196.
- 14 [Cal20] O. Calin. Deep Learning Architectures: A Mathematical Approach. en. 1st ed. Springer, Feb. 2020.
- 15 [Can04] J. Canny. "GaP: a factor model for discrete data". In: SIGIR. 2004, pp. 122–129.
- 16 [Cao+15] Y. Cao, M. A. Brubaker, D. J. Fleet, and A. Hertzmann. "Efficient Optimization for Sparse Gaussian Process Regression". en. In: IEEE PAMI 37.12 (Dec. 2015), pp. 2415–2427.
- 17 [Car03] P. Carbonetto. "Unsupervised Statistical Models for General Object Recognition". MA thesis. University of British Columbia, 2003.
- 18 [Car+15] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad. "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission". In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. 2015, pp. 1721–1730.
- 19 [Car+17] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. "Stan: A Probabilistic Programming Language". In: Journal of Statistical Software, Articles 76.1 (2017), pp. 1–32.
- 20 [Car+19] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. "On evaluating adversarial robustness". In: arXiv preprint arXiv:1902.06705 (2019).
- 21 [Car97] R. Caruana. "Multitask Learning". In: Machine Learning 28.1 (1997), pp. 41–75.
- 22 [Cat08] A. Caticha. Lectures on Probability, Entropy, and Statistical Physics. 2008. arXiv: 0808.0012 [physics.data-an].
- 23 [Cat+11] A. Caticha, A. Mohammad-Djafari, J.-F. Bercher, and P. Bessière. "Entropic Inference". In: AIP Conference Proceedings 1305.1 (2011), pp. 20–29. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.3573619
- 24 [Cau+20] M. Cauchois, S. Gupta, A. Ali, and J. C. Duchi. "Robust Validation: Counterfactual Predictions Even When Distributions Shift". In: (Aug. 2020). arXiv: 2008.04267 [stat.ML].
- 25 [CB20] Y. Chen and P. Bühlmann. "Domain adaptation under structural causal models". In: JMLR (Oct. 2020).
- 26 [CBL20] K. Cranmer, J. Brehmer, and G. Louppe. "The frontier of simulation-based inference". In: Proceedings of the National Academy of Sciences 117.48 (2020), pp. 30055–30062.
- 27 [CBR20] Z. Chen, Y. Bei, and C. Rudin. "Concept whitening for interpretable image recognition". In: Nature Machine Intelligence 2.12 (2020), pp. 772–782.
- 28 [CC84] M. Conforti and G. Cornuejols. "Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem". In: Discrete Applied Mathematics 7.3 (1984), pp. 251–274.
- 29 [CC96] M. Cowles and B. Carlin. "Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review". In: JASA 91 (1996), pp. 883–904.
- 30 [CDC15] C. Chen, N. Ding, and L. Carin. "On the Convergence of Stochastic Gradient MCMC Algorithms with High-Order Integrators". In: NIPS. 2015.
- 31 [CDS02] M. Collins, S. Dasgupta, and R. E. Schapire. "A Generalization of Principal Components Analysis to the Exponential Family". In: NIPS. 2002.
- 32 [CDS19] A. Clark, J. Donahue, and K. Simonyan. "Adversarial video generation on complex datasets". In: arXiv preprint arXiv:1907.06571 (2019).
- 33 [Cér+12] F. Cérou, P. Del Moral, T. Furon, and A. Guyader. "Sequential Monte Carlo for rare event estimation". In: Stat. Comput. 22.3 (May 2012), pp. 795–808.
- 34 [CFG14a] T. Chen, E. B. Fox, and C. Guestrin. "Stochastic Gradient Hamiltonian Monte Carlo". In: ICML. 2014.
- 35 [CFG14b] T. Chen, E. B. Fox, and C. Guestrin. "Stochastic Gradient Hamiltonian Monte Carlo". In: ICLR. 2014.
- 36 [CG00] S. S. Chen and R. A. Gopinath. "Gaussianization". In: NIPS. 2000, pp. 423–429.
- 37 [CG18] C. Ceylan and M. U. Gutmann. "Conditional Noise-Contrastive Estimation of Unnormalised Models". In: International Conference on Machine Learning. 2018, pp. 726–734.
- 38 [CG96] S. Chen and J. Goodman. "An empirical study of smoothing techniques for language modeling". In: Proc. 34th ACL. 1996, pp. 310–318.
- 39 [CG98] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Tech. rep. TR-10-98. Dépt. Comp. Sci., Harvard, 1998.
- 40 [CGR06] S. R. Cook, A. Gelman, and D. B. Rubin. "Validation of Software for Bayesian Models Using Posterior Quantiles". In: J. Comput. Graph. Stat. 15.3 (Sept. 2006), pp. 675–692.
- 41 [CH20] C. Cinelli and C. Hazlett. "Making sense of sensitivity: extending omitted variable bias". In: Journal of the Royal Statistical Society: Series B (Statistical Methodology) 82.1 (2020), pp. 39–67. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12348.
- 42 [Cha12] K. M. A. Chai. "Variational Multinomial Logit Gaussian Process". In: JMLR 13.Jul (2012), pp. 1745–1808.
- 43 [Cha14] N. Chapados. "Effective Bayesian Modeling of Groups of Related Count Time Series". In: ICML. 2014.
- 44 [Cha17] D. Chakrabarty, Y. T. Lee, A. Sidford, and S. C.-w. Wong. "Subquadratic submodular function minimization". In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. 2017, pp. 1220–1231.
- 45 [Cha+18] N. S. Chatterji, N. Flammarion, Y.-A. Ma, P. L. Bartlett, and M. I. Jordan. "On the theory of variance reduction for stochastic gradient Monte Carlo". In: ICML. Feb. 2018.
- 46 [Cha+19a] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros. "Everybody dances now". In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 5933–5942.
- 47 [Cha+19b] O. Chang, Y. Yao, D. Williams-King, and H. Lipson. "Ensemble model patching: A parameter-client variational Bayesian neural network". In: arXiv preprint arXiv:1905.09453 (2019).
- 48 [Cha+19c] T. Chavdarova, G. Gidel, F. Fleuret, and S. Lacoste-Julien. "Reducing noise in GAN training with variance reduced extragradient". In: Advances in Neural Information Processing Systems. 2019, pp. 393–403.
- 49 [Cha+20] P. E. Chang, W. J. Wilkinson, M. E. Khan, and A. Solin. "Fast Variational Learning in State-Space Gaussian Process Models". In: 2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP). Sept. 2020, pp. 1–6.
- 50 [Cha21] S. H. Chan. Introduction to Probability for Data Science. Michigan Publishing, 2021.
- 51 [Che+05] G. Chechik, A. Globerson, N. Tishby, and Y. Weiss. "Information Bottleneck for Gaussian Variables". In: JMLR 6.Jan (2005), pp. 165–188.
- 52 [Che14] C. Chekuri. "Treewidth, Applications, and some Recent Developments". In: NIPS Tutorial. 2014.
- 53 [Che+15] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: ICLR. 2015.
- 54 [Che+17] T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Song, and Y. Bengio. "Maximum-likelihood augmented discrete generative adversarial networks". In: arXiv preprint arXiv:1702.07983 (2017).
- 55 [Che17] C. Cheiba. Language Modeling in the Era of Abundant Data. AI with the best. 2017.

- 1
- 2 [Che+17a] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy,
and A. L. Yuille. "DeepLab: Semantic Image Segmentation with
Deep Convolutional Nets, Atrous Convolution, and Fully Con-
nected CRFs". In: IEEE PAMI (2017).
- 3 [Che+17b] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P.
Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. "Varia-
tional Lossy Autoencoder". In: ICLR. 2017.
- 4 [Che+17c] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel.
"PixelSNAIL: An Improved Autoregressive Generative Model".
In: (Dec. 2017). arXiv: 1712.09763 [cs.LG].
- 5 [Che+17d] V. Chernozhukov, D. Chetverikov, M. Demirer, E.
Dudoit, C. Hansen, and W. Newey. "Double/Debiased/Neyman
Machine Learning of Treatment Effects". In: American Eco-
nomic Review 107.5 (2017), pp. 261–65.
- 6 [Che+17e] V. Chernozhukov, D. Chetverikov, M. Demirer, E.
Dudoit, C. Hansen, W. Newey, and J. Robins. "Double/Debiased
Machine Learning for Treatment and Structural parameters".
In: The Econometrics Journal (2017).
- 7 [Che+18a] C. Chen, W. Wang, Y. Zhang, Q. Su, and L. Carin.
"A convergence analysis for a class of practical variance-
reduction stochastic gradient MCMC". In: Sci. China Inf. Sci.
62 (Dec. 2018), p. 12101.
- 8 [Che+18b] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C.
Rudin. "This looks like deep learning for interpretable im-
age recognition". In: arXiv preprint arXiv:1806.10574 (2018).
- 9 [Che+18c] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D.
Duvenaud. "Neural Ordinary Differential Equations". In: NIPS.
2018.
- 10 [Che+18d] X. Cheng, N. S. Chatterji, P. L. Bartlett, and M. I.
Jordan. "Underdamped Langevin MCMC: A non-asymptotic
analysis". In: COLT. 2018.
- 11 [Che+19] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H.
Jacobsen. "Residual Flows for Invertible Generative Modeling".
In: NIPS. 2019.
- 12 [Che+20a] M. Chen, Y. Wang, T. Liu, Z. Yang, X. Li,
Wang, and T. Zhao. "On computation and generalization of
generative adversarial imitation learning". In: arXiv preprint
arXiv:2001.02792 (2020).
- 13 [Che+20b] N. Chen, Y. Zhang, H. Zen, R. J. Weiss, M. Norouzi,
and W. Chan. "WaveGrad: Estimating Gradients for Waveform
Generation". In: arXiv preprint arXiv:2009.00713 (2020).
- 14 [Che95] Y. Cheng. "Mean shift, mode seeking, and clustering".
In: IEEE PAMI 17.8 (1995).
- 15 [Chi14] S. Chiappa. "Explicit-Duration Markov Switching Mod-
els". In: Foundations and Trends in Machine Learning 7.6
(2014), pp. 803–886.
- 16 [Chi21] R. Child. "Very Deep VAEs Generalize Autoregressive
Models and Can Outperform Them on Images". In: ICLR. 2021.
- 17 [Cho02] N. Chopin. "A Sequential Particle Filter Method for
Static Models". In: Biometrika 89.3 (2002), pp. 539–551.
- 18 [Cho11] M. J. Choi. "Trees and Beyond: Exploiting and Im-
proving Tree-structured Graphical Models". PhD thesis. MIT, 2011.
- 19 [Cho15] Y. Chow, A. Tamar, S. Mannor, and M. Pavone. "Risk-
Sensitive and Robust Decision-Making: A CVaR Optimization
Approach". In: NIPS. 2015, pp. 1522–1530.
- 20 [Cho21] F. Chollet. Deep learning with Python (second edition).
Manning, 2021.
- 21 [Cho57] N. Chomsky. Syntactic Structures. Mouton, 1957.
- 22 [Chr+21] R. Christensen, N. Pster, M. E. Jakobsen, N.
Gnecco, and J. Peters. "A causal framework for distribution
generalization". In: IEEE PAMI (2021).
- 23 [Chu+15] J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville,
and Y. Bengio. "A Recurrent Latent Variable Model for Se-
quential Data". In: NIPS. 2015.
- 24 [Chu+18] K. Chua, R. Calandra, R. McAllister, and S. Levine.
"Deep Reinforcement Learning in a Handful of Trials using
Probabilistic Dynamics Models". In: NIPS. 2018.
- 25 [Chw+15] K. Chwialkowski, A. Ramdas, D. Sejdinovic, and A.
Gretton. "Fast Two-Sample Testing with Analytic Representa-
tions of Probability Measures". In: NIPS. 2015.
- 26 [CJ80] D. R. Cox and V. Isham. Point processes. Vol. 12. CRC
Press, 1980.
- 27 [CJ21] A. D. Cobb and B. Jalaiyan. "Scaling Hamiltonian Monte
Carlo inference for Bayesian neural networks with symmetric
splitting". In: UAI. Vol. 161. Proceedings of Machine Learning
Research. PMLR, 2021, pp. 675–685.
- 28 [CK05] M. Collins and T. Koo. "Discriminative Reranking for
Natural Language Parsing". In: Proc. ACL. 2005.
- 29 [CK07] J. F. Commandeur and S. J. Koopman. An Introduc-
tion to State Space Time Series Analysis (Practical Economet-
rics). en. 1st ed. Oxford University Press, Aug. 2007.
- 30 [CK21] D. Chakrabarty and S. Khanna. "Better and simpler er-
ror analysis of the Sinkhorn-Knopp algorithm for matrix scal-
ing". In: Mathematical Programming 188.1 (2021), pp. 395–
407.
- 31 [CK94a] D. Card and A. B. Krueger. "Minimum Wages and Em-
ployment: A Case Study of the Fast-Food Industry in New Jer-
sey and Pennsylvania". In: American Economic Review 84.4
(1994), pp. 772–793.
- 32 [CK94b] C. Carter and R. Kohn. "On Gibbs sampling for state
space models". In: Biometrika 81.3 (1994), pp. 541–553.
- 33 [CKK17] L. Chen, A. Krause, and A. Karbasi. "Interactive Sub-
modular Bandit". In: NIPS. 2017, pp. 141–152.
- 34 [CL00] R. Chen and S. Liu. "Mixture Kalman Filters". In: J.
Royal Stat. Soc. B (2000).
- 35 [CL04] L. Carvalho and C. Lawrence. "Centroid estimation in
discrete high-dimensional spaces with applications in biology".
In: PNAS 101.4 (2004).
- 36 [CL11] O. Chapelle and L. Li. "An empirical evaluation of
Thompson sampling". In: NIPS. 2011.
- 37 [CL18] Z. Chen and B. Liu. Lifelong Machine Learning. Syn-
thesis Lectures on Artificial Intelligence and Machine Learning.
Morgan Claypool, 2018.
- 38 [CL96] B. P. Carlin and T. A. Louis. Bayes and Empirical
Bayes Methods for Data Analysis. Chapman and Hall, 1996.
- 39 [Cla20] P. Clavier. "Sum-Product Network in the context of
missing data". en: MA thesis. KTH, 2020.
- 40 [Cla21] A. Clayton. Bernoulli's Fallacy: Statistical Illogic and
the Crisis of Modern Science. en: Columbia University Press,
Aug. 2021.
- 41 [CLD18] C. Cremer, X. Li, and D. Duvenaud. "Inference Subop-
timality in Variational Autoencoders". In: ICML. 2018.
- 42 [Clo19] J. R. Clough, I. Otsuzi, E. Puyol-Antón, B. Ruijsink,
A. P. King, and J. A. Schnabel. "Global and local interpretability
for cardiac MRI classification". In: International Conference
on Medical Image Computing and Computer-Assisted In-
tervention. Springer, 2019, pp. 656–664.
- 43 [Clo20] Cloudera. Causality for ML. 2020.
- 44 [CLV19] M. Cox, T. van de Laar, and B. de Vries. "A factor
graph approach to automated design of Bayesian signal process-
ing algorithms". In: Int. J. Approx. Reason. 104 (Jan. 2019),
pp. 185–204.
- 45 [CLW18] Y. Chen, L. Li, and M. Wang. "Scalable Bilinear π-
Learning Using State and Action Features". In: ICML. 2018,
pp. 833–842.
- 46 [CM09] O. Cappé and E. Moulines. "Online EM Algorithm for
Latent Data Models". In: J. of Royal Stat. Soc. Series B 71.3
(2009), pp. 593–613.
- 47 [CM18] D. Crisan and J. Míguez. "Nested particle filters for on-
line parameter estimation in discrete-time state-space Markov
models". en: In: Bernoulli 24.4A (Nov. 2018), pp. 3039–3086.
- 48 [CMD17] C. Cremer, Q. Morris, and D. Duvenaud. "Reinterpret-
ing Importance-Weighted Autoencoders". In: ICLR Workshop.
- 49 [CM008] J. Cornebise, E. Moulines, and J. Olsson. "Adaptive
methods for sequential importance sampling with application
to state space models". In: Statistics and Computing (Mar.
2008).
- 50 [CMR05] O. Cappe, E. Moulines, and T. Ryden. Inference in
Hidden Markov Models. Springer, 2005.
- 51 [CN07] H. Choset and K. Nagatani. "Topological simultaneous
localization and mapping (SLAM): toward exact localization
without explicit localization". In: IEEE Trans. Robotics and
Automation 17.2 (2001).
- 52 [CNW20] M. Collier, A. Nazabal, and C. K. I. Williams. "VAEs
in the Presence of Missing Data". In: ICML Workshop on the
Art of Learning with Missing Values. June 2020.
- 53 [CO06] N. Chater and M. Oaksford. "Mental mechanisms". In:
Information sampling and adaptive cognition (2006), pp. 210–
236.
- 54 [COB18] L. Chizat, E. Oyallon, and F. Bach. "On Lazy Train-
ing in Differentiable Programming". In: (Dec. 2018). arXiv:
1812.07956 [math.OC].
- 55 [COP12] G. Cormode, M. Garofalakis, P. J. Haas, and C.
Jiannat. "Synopses for massive data: Samples, histograms,
wavelets, sketches". In: Foundations and Trends in Databases
4.1–3 (2012), pp. 1–294.
- 56 [Cor17] G. Cormode. "Data sketching". In: Communications of
the ACM 60.9 (2017), pp. 48–55.
- 57 [Cor+59] J. Cornfield, W. Haenszel, E. C. Hammond, A. M.
Lilienfeld, M. B. Shimkin, and E. L. Wynder. "Smoking and
Lung Cancer: Recent Evidence and a Discussion of Some Ques-
tions". In: JNCI: Journal of the National Cancer Institute 22.1
(Jan. 1959), pp. 173–203. eprint: <https://academic.oup.com/jnci/article-pdf/22/1/173/2704718/22-1-173.pdf>.

BIBLIOGRAPHY

- 1 [Cor+87] A Corana, M Marchesi, C Martini, and S Ridella. "Minimizing Multimodal Functions of Continuous Variables with the Simulated Annealing Algorithm". In: ACM Trans. Math. Softw. 13.3 (Sept. 1987), pp. 262–280.
- 2 [Cou16] Council of European Union. General Data Protection Regulation. 2016.
- 3 [Cov99] T. M. Cover. Elements of information theory. John Wiley & Sons, 1999.
- 4 [Cow+99] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. Probabilistic Networks and Expert Systems. Springer, 1999.
- 5 [CP20a] R. Chen and I. C. Paschalidis. Distributionally Robust Learning. NOW Foundations and Trends in Optimization, 2020.
- 6 [CP20b] N. Chopin and O. Papaspiliopoulos. An Introduction to Sequential Monte Carlo (Springer Series in Statistics). en. 1st ed. Springer, Oct. 2020.
- 7 [CPD17] P. Constantinou and A. Philip Dawid. "Extended conditional independence and applications in causal inference". en. In: Ann. Stat. 45.6 (Dec. 2017), pp. 2618–2653.
- 8 [CPS10] C. Carvalho, N. Polson, and J. Scott. "The horseshoe estimator for sparse signals". In: Biometrika 97.2 (2010), p. 465.
- 9 [CRK19] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. "Certified adversarial robustness via randomized smoothing". In: arXiv preprint arXiv:1902.02918 (2019).
- 10 [Cro+11] D. F. Crouse, P. Willett, K. Pattipati, and L. Svensson. "A look at Gaussian mixture reduction algorithms". In: 14th International Conference on Information Fusion. July 2011, pp. 1–8.
- 11 [CS04] I. Csiszár and P. C. Shields. "Information theory and statistics: A tutorial". In: (2004).
- 12 [CS09] Y. Cho and L. K. Saul. "Kernel Methods for Deep Learning". In: NIPS. 2009, pp. 342–350.
- 13 [CS18] P. Chaudhari and S. Soatto. "Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks". In: ICLR. 2018.
- 14 [Csi67] I. Csiszár. "Information-Type Measures of Divergence of Probability Distributions and Indirect Observations". In: Studia Scientiarum Mathematicarum Hungarica 2 (1967), pp. 299–318.
- 15 [CSN21] J. Coullon, L. South, and C. Nemeth. "Stochastic Gradient MCMC with Multi-Armed Bandit Tuning". In: (May 2021). arXiv: 2105.13059 [stat.CO].
- 16 [CT06] T. M. Cover and J. A. Thomas. Elements of Information Theory. 2nd edition. John Wiley, 2006.
- 17 [CT+19] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. "Gen: a general-purpose probabilistic programming system with programmable inference". In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI 2019. Phoenix, AZ, USA: Association for Computing Machinery, June 2019, pp. 221–236.
- 18 [CT91] T. M. Cover and J. A. Thomas. Elements of Information Theory. John Wiley, 1991.
- 19 [CTM17] M. F. Cusumano-Towner and V. K. Mansinghka. "AIDE: An algorithm for measuring the accuracy of probabilistic inference algorithms". In: NIPS. 2017.
- 20 [CTN17] Y. Chai, M. Tanvee, and M. T. Nayyer. "Towards abstractive multi-document summarization using submodular function-based framework, sentence compression and merging". In: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers). 2017, pp. 418–424.
- 21 [CTS78] C. Cannings, E. A. Thompson, and M. H. Skolnick. "Probability functions in complex pedigrees". In: Advances in Applied Probability 10 (1978), pp. 26–61.
- 22 [CUH16] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: ICLR. 2016.
- 23 [Cun83] W. H. Cunningham. "Decomposition of submodular functions". In: Combinatorica 3.1 (1983), pp. 53–68.
- 24 [Cut13] M. Cuturi. "Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances". In: NIPS. 2013.
- 25 [CW07] C. M. Carvalho and M. West. "Dynamic Matrix-Variate Graphical Models". In: Bayesian Analysis 2.1 (2007), pp. 69–98.
- 26 [CW16] T. Cohen and M. Welling. "Group Equivariant Convolutional Networks". en. In: ICML. June 2016, pp. 2990–2999.
- 27 [CWG20] D. C. Castro, I. Walker, and B. Glocker. "Causality matters in medical imaging". en. In: Nat. Commun. 11.1 (July 2020), p. 3673.
- 28 [CY20] G. Cormode and K. Yi. Small Summaries for Big Data. Cambridge University Press, 2020.
- 29 [Cza+20] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison. "DeepFactors: Real-Time Probabilistic Dense Monocular SLAM". In: ICRA. 2020.
- 30 [CZG20] B. Charpentier, D. Zügner, and S. Günnemann. "Posterior network: Uncertainty estimation without *ood* samples via density-based pseudo-counts". In: arXiv preprint arXiv:2006.09239 (2020).
- 31 [D'A+21] A. D'Amour, P. Ding, A. Feller, L. Lei, and J. Sekhon. "Overlap in observational studies with high-dimensional covariates". In: Journal of Econometrics 221.2 (2021), pp. 644–654.
- 32 [Dag+21] N. Dagan, N. Barde, E. Kepten, O. Miron, S. Perchik, M. A. Katz, M. A. Hernán, M. Lipsitch, B. Reis, and R. D. Balicer. "BNT162b2 mRNA Covid-19 Vaccine in a Nationwide Mass Vaccination Setting". In: New England Journal of Medicine 384.15 (2021), pp. 1412–1423. eprint: <https://doi.org/10.1056/NEJMoa2101765>.
- 33 [Dai+17] H. Dai, B. Dai, Y.-M. Zhang, S. Li, and L. Song. "Recurrent Hidden Semi-Markov Model". In: ICLR. 2017.
- 34 [Dai+18] B. Dai, A. Shaw, L. Li, L. Xiao, N. He, Z. Liu, Chen, and L. Song. "SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation". In: ICML. 2018, pp. 1133–1142.
- 35 [Dai+19a] B. Dai, H. Dai, A. Gretton, L. Song, D. Schuurmans, and N. He. "Kernel exponential family estimation via doubly dual embedding". In: AISTATS. PMLR. 2019, pp. 2321–2330.
- 36 [Dai+19b] B. Dai, Z. Liu, H. Dai, N. He, A. Gretton, L. Song, and D. Schuurmans. "Exponential family estimation via adversarial dynamics embedding". In: Advances in Neural Information Processing Systems. 2019, pp. 10979–10990.
- 37 [Dai+19c] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov. "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context". In: Proc. ACL. 2019, pp. 2978–2988.
- 38 [Dai+20a] C. Dai, J. Heng, P. E. Jacob, and N. Whiteley. "An invitation to sequential Monte Carlo samplers". In: (July 2020). arXiv: 2007.11936 [stat.CO].
- 39 [Dai+20b] Z. Dai, G. Lai, Y. Yang, and Q. V. Le. "Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing". In: NIPS. June 2020.
- 40 [Dar+04] N. Dalvi, P. Domingos, S.anghai, and D. Verma. "Adversarial classification". In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. 2004, pp. 99–108.
- 41 [Dar03] A. Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: J. ACM 50.3 (May 2003), pp. 280–305.
- 42 [Dar09] A. Darwiche. Modeling and Reasoning with Bayesian Networks. Cambridge, 2009.
- 43 [Dar+11] S. Darolles, Y. Fan, J.-P. Florens, and E. Renault. "Nonparametric instrumental regression". In: Econometrica 79.5 (2011), pp. 1541–1565.
- 44 [Dar80] R. A. Darton. "Rotation in Factor Analysis". In: Journal of the Royal Statistical Society. Series D (The Statistician) 29.3 (1980), pp. 167–194.
- 45 [Dau07] H. Daume. "Fast search for Dirichlet process mixture models". In: AISTATS. 2007.
- 46 [Dav+04] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. Ng. "A Column Approximate Minimum Degree Ordering Algorithm". In: ACM Trans. Math. Softw. 30.3 (Sept. 2004), pp. 353–376.
- 47 [Dav+18] T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak. "Hyperspherical Variational Auto-Encoders". In: UAI. 2018.
- 48 [Daw00] A. P. Dawid. "Causal Inference Without Counterfactuals". In: JASA 95.450 (2000), pp. 407–424.
- 49 [Daw02] A. P. Dawid. "Influence diagrams for causal modelling and inference". In: Int'l. Stat. Review 70 (2002). Corrections p437, pp. 161–169.
- 50 [Daw15] A. P. Dawid. "Statistical Causality from a Decision-Theoretic Perspective". In: Annu. Rev. Stat. Appl. 2.1 (Apr. 2015), pp. 273–303.
- 51 [Daw82] A. P. Dawid. "The Well-Calibrated Bayesian". In: JASA 77.379 (1982), pp. 605–610.
- 52 [Daw92] A. P. Dawid. "Applications of a general propagation algorithm for probabilistic expert systems". In: Statistics and Computing 2 (1992), pp. 25–36.
- 53 [Dax+21] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. "Laplace Redux—Effortless Bayesian Deep Learning". In: NIPS. 2021.
- 54 [Day+95] P. Dayan, G. Hinton, R. Neal, and R. Zemel. "The Helmholtz machine". In: Neural Networks 9.8 (1995).
- 55 [DB+13] P. De Blasi, S. Favaro, A. Lijoi, R. H. Mena, I. Prünster, and M. Ruggiero. "Are Gibbs-type priors the most natural generalization of the Dirichlet process?". In: IEEE PAMI 37.2 (2013), pp. 212–229.

- 1
- 2 [DBB20] S. Daulton, M. Balandat, and E. Bakshy. "Di erentiable Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Optimization". In: NIPS. 2020.
- 3 [DBP19] C. Durkan, A. Bekasov, and I. M. G. Papamakarios. "Neural Spline Flows". In: NIPS. 2019.
- 4 [DBW20] I. A. Delbridge, D. S. Bindel, and A. G. Wilson. "Randomly Projected Additive Gaussian Processes for Regression". In: International Conference on Machine Learning. 2020.
- 5 [DC+20] L. Da Costa, N. Sajid, T. Parr, K. Friston, and R. Smith. "The relationship between dynamic programming and active inference: the discrete, nite-horizon case". In: (Sept. 2020). arXiv: 2009.08111 [cs.AI].
- 6 [DC20] H.-D. Dau and N. Chopin. "Waste-free Sequential Monte Carlo". In: (Nov. 2020). arXiv: 2011.02328 [stat.CO].
- 7 [DCF+15] E. Denton, S. Chintala, R. Fergus, et al. "Deep generative image models using a Laplacian pyramid of adversarial networks". In: NIPS. 2015.
- 8 [DE00] R. Dahlhaus and M. Eichler. "Causality and graphical models for time series". In: Highly structured stochastic systems. Ed. by P. Green, N. Hjort, and S. Richardson. Oxford University Press, 2000.
- 9 [DE04] J. Dow and J. Endersby. "Multinomial probit and multinomial logit: a comparison of choice models for voting research". In: Electoral Studies 23.1 (2004), pp. 107–122.
- 10 [Dec03] R. Dechter. Constraint Processing. Morgan Kaufmann, 2003.
- 11 [Dec19] R. Dechter. "Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms (2nd edn)". In: Synthesis Lectures on Articial Intelligence and Machine Learning 7.3 (2019), pp. 1–191.
- 12 [Dec96] R. Dechter. "Bucket elimination: a unifying framework for probabilistic inference". In: UAI. 1996.
- 13 [DeG70] M. DeGroot. Optimal Statistical Decisions. McGraw-Hill, 1970.
- 14 [DEL20] H. M. Dolatabadi, S. Erfani, and C. Leckie. "Invertible Generative Modeling using Linear Rational Splines". In: AIS-TATS. 2020, pp. 4236–4246.
- 15 [Del+21] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardi, G. Slabaugh, and T. Tuytelaars. "A continual learning survey: Defying forgetting in classi cation tasks". en. In: IEEE PAMI (Feb. 2021).
- 16 [Den+02] D. Denison, C. Holmes, B. Mallick, and A. Smith. Bayesian methods for nonlinear classi cation and regression. Wiley, 2002.
- 17 [Den+20] Y. Deng, A. Bakhtin, M. Ott, A. Szlam, and M. Ranzato. "Residual Energy-Based Models for Text Generation". In: International Conference on Learning Representations. 2020.
- 18 [Dev+19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: NAACL. 2019.
- 19 [Dev+21] L. Devlin, P. Horridge, P. L. Green, and S. Maskell. "The No-U-Turn Sampler as a Proposal Distribution in a Sequential Monte Carlo Sampler with a Near-Optimal L-Kernel". In: (Aug. 2021). arXiv: 2108.02498 [stat.CO].
- 20 [Dev85] P. A. Devijver. "Baum's forward-backward algorithm revisited". In: Pattern Recognition Letters 3.6 (1985), pp. 369–373.
- 21 [Dex] Dex: Research language for array processing in the Haskell/ML family. <https://github.com/google-research/dex-lang>. 2019.
- 22 [DF18] E. Denton and R. Fergus. "Stochastic Video Generation with a Learned Prior". In: ICML. 2018.
- 23 [DF19] X. Ding and D. J. Freedman. "Learning Deep Generative Models with Annealed Importance Sampling". In: (June 2019). arXiv: 1906.04904 [stat.ML].
- 24 [DF21] S. Dozinski, U. Feige, and M. Feldman. "Are Gross Substitutes for Submodular Valuations?". In: arXiv preprint arXiv:2102.13343 (2021).
- 25 [DFR15] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. "Gaussian Processes for Data-E cient Learning in Robotics and Control". en. In: IEEE PAMI 37.2 (Feb. 2015), pp. 408–423.
- 26 [DFS16] A. Daniely, R. Frostig, and Y. Singer. "Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity". In: NIPS. 2016, pp. 2253–2261.
- 27 [DG17] P. Dabkowski and Y. Gal. "Real time image saliency for black box classiers". In: NeurIPS (2017).
- 28 [DG84] P. J. Diggle and R. J. Gratton. "Monte Carlo methods of inference for implicit statistical models". In: Journal of the Royal Statistical Society, Series B (Methodological) (1984), pp. 193–227.
- 29 [DGA00] A. Doucet, S. Godsill, and C. Andrieu. "On sequential Monte Carlo Sampling Methods for Bayesian Filtering". In: Statistics and Computing 10.3 (2000), pp. 197–208.
- 30 [DGFB20] Y. Dubois, J. Gordon, and A. Y. Foong. Neural Process Family. <http://yanndubs.github.io/Neural-Process-Family/>. 2020.
- 31 [DKG01] A. Doucet, N. Gordon, and V. Krishnamurthy. "Particle Filters for State Estimation of Jump Markov Linear Systems". In: IEEE Trans. on Signal Processing 49.3 (2001), pp. 613–624.
- 32 [DHK14] A. Deshpande, L. Hellerstein, and D. Kletenik. "Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover". In: Proceedings of the twenty- th annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2014, pp. 1453–1466.
- 33 [Dia88] P. Diaconis. "Sufficiency as statistical symmetry". In: Proceedings of the AMS Centennial Symposium. 1988, pp. 15–26.
- 34 [Die+07] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber. "Fast Direct Multiple Shooting Algorithms for Optimal Robot Control". In: Lecture Notes in Control and Inform. Sci. 340 (July 2007).
- 35 [Die10] L. Dietz. Directed Factor Graph Notation for Generative Models. Tech. rep. MPI, 2010.
- 36 [Die+17] A. B. Dieng, D. Tran, R. Ranganath, J. Paisley, and D. Blei. "Variational Inference via chi Upper Bound Minimization". In: NIPS. Curran Associates, Inc., 2017, pp. 2732–2741.
- 37 [Die+19a] A. B. Dieng, Y. Kim, A. M. Rush, and D. M. Blei. "Avoiding Latent Variable Collapse With Generative Skip Models". In: AISTATS. 2019.
- 38 [Die+19b] A. B. Dieng, F. J. Ruiz, D. M. Blei, and M. K. Titsias. "Prescribed generative adversarial networks". In: arXiv preprint arXiv:1910.04302 (2019).
- 39 [Dik+20] N. Dikkala, G. Lewis, L. Mackey, and V. Syrgkanis. "Minimax Estimation of Conditional Moment Models". In: Advances in Neural Information Processing Systems. 2020.
- 40 [Din+17] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. "Sharp Minima Can Generalize For Deep Nets". In: (2017). arXiv: 04933 [cs.LG].
- 41 [Din+21] S. U. Din, J. Shao, J. Kumar, C. B. Mawuli, S. M. H. Mahmud, W. Zhang, and Q. Yang. "Data stream classi cation with novel class detection: a review, comparison and challenges". In: Knowl. Inf. Syst. 63.9 (Sept. 2021), pp. 2231–2276.
- 42 [DJ11] A. Doucet and A. M. Johansen. "A Tutorial on Particle Filtering and Smoothing: Fifteen years later". In: Handbook of Nonlinear Filtering. Ed. by D Crisan and B Rozovsk. 2011.
- 43 [DJ15] S. Dray and J. Josse. "Principal component analysis with missing values: a comparative survey of methods". In: Plant Ecol. 216.5 (May 2015), pp. 657–667.
- 44 [DKJ18] J. Dijolonga, S. Jegelka, and A. Krause. "Provable Variational Inference for Constrained Log-Submodular Models". In: NeurIPS. 2018.
- 45 [DJL21] A. J. DeGrave, J. Janizek, and S.-I. Lee. "AI for radiographic COVID-19 detection selects shortcuts over signal". en. In: Nature Machine Intelligence 3.7 (May 2021), pp. 610–619.
- 46 [DK12] J. Durbin and S. J. Koopman. Time Series Analysis by State Space Methods: Second Edition. en. Revised ed. edition. Oxford University Press, July 2012.
- 47 [DK14] J. Dijolonga and A. Krause. "From map to marginals: Variational inference in bayesian submodular models". In: Advances in Neural Information Processing Systems. 2014, pp. 244–252.
- 48 [DK15a] J. Dijolonga and A. Krause. "Scalable variational inference in log-supermodular models". In: International Conference on Machine Learning. PMLR, 2015, pp. 1804–1813.
- 49 [DK15b] G. Durrett and D. Klein. "Neural CRF Parsing". In: Proc. ACL. 2015.
- 50 [DKB15] L. Dinh, D. Krueger, and Y. Bengio. "NICE: Non-linear Independent Components Estimation". In: ICLR. 2015.
- 51 [DKD16] J. Donahue, P. Krakenbuhl, and T. Darrell. "Adversarial feature learning". In: arXiv preprint arXiv:1605.09782 (2016).
- 52 [DL09] P. Domingos and D. Lowd. Markov Logic: An Interface Layer for AI. Morgan & Claypool, 2009.
- 53 [DL10] J. V. Dillon and G. Lebanon. "Stochastic Composite Likelihood". In: J. Mach. Learn. Res. 11 (2010), pp. 2597–2633.
- 54 [DL13] A. Damianou and N. Lawrence. "Deep Gaussian Processes". en. In: AISTATS. Apr. 2013, pp. 207–215.
- 55 [DL93] P. Dagum and M. Luby. "Approximating probabilistic inference in Bayesian belief networks is NP-hard". In: Articial Intelligence 60 (1993), pp. 141–153.
- 56 [DLB17] C. Dann, T. Lattimore, and E. Brunskill. "Unifying PAC and Regret: Uniform PAC Bounds for Episodic Reinforcement Learning". In: NIPS. 2017, pp. 5717–5727.
- 57 [DLM09] H. Daumé III, J. Langford, and D. Marcu. "Search-based Structured Prediction". In: MLJ 75.3 (2009), pp. 297–325.

1703.

BIBLIOGRAPHY

- 1
- [DLM99] B. Delyon, M. Lavielle, and E. Moulines. "Convergence
of a stochastic approximation version of the EM algorithm". In:
Annals of Statistics 27.1 (1999), pp. 94–128.
- 2
- [DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum
likelihood from incomplete data via the EM algorithm".
In: J. of the Royal Statistical Society, Series B 34 (1977),
pp. 1–38.
- 3
- [DLT21] M. De Lange and T. Tuytelaars. "Continual Prototype
Evolution: Learning Online from Non-Stationary Data
Streams". In: ICCV. 2021.
- 4
- [DM01] D. van Dyk and X.-L. Meng. "The Art of Data Augmentation". In: J. Computational and Graphical Statistics 10.1
(2001), pp. 1–50.
- 5
- [DM19a] Y. Du and I. Mordatch. "Implicit Generation and Generalization in Energy-Based Models". In: (Mar. 2019). arXiv:
1903.08689 [cs.LG].
- 6
- [DM19b] Y. Du and I. Mordatch. "Implicit Generation and Modeling with Energy Based Models". In: NIPS. 2019, pp. 3608–
3618.
- 7
- [DMDJ12] P. Del Moral, A. Doucet, and A. Jasra. "An adaptive
sequential Monte Carlo method for approximate Bayesian com-
putation". In: Stat. Comput. 22.5 (Sept. 2012), pp. 1009–1020.
- 8
- [DMK22] G. Duran-Martin, A. Kara, and K. Murphy. "E-
cient Online Bayesian Inference for Neural Bandits". In: AIS-
TATS. 2022.
- 9
- [DMM17] A. P. Dawid, M. Musio, and R. Murtas. "The proba-
bility of causation". In: Law, Probability and Risk 16.4 (Dec.
2017), pp. 163–179.
- 10
- [DMP18] C. Donahue, J. McAuley, and M. Puckette. "Adversarial
Audio Synthesis". In: International Conference on Learning
Representations. 2018.
- 11
- [DMV15] S. Dash, D. Malioutov, and K. R. Varshney. "Learn-
ing interpretable classification rules using sequential rowsum-
pling". In: 2015 IEEE International Conference on Acoustics,
Speech and Signal Processing (ICASSP). IEEE. 2015, pp. 3337–
3341.
- 12
- [DN21] P. Dhariwal and A. Nichol. "Diusion Models Beat
GANs on Image Synthesis". In: (May 2021). arXiv:
2105.05233 [cs.LG].
- 13
- [Dnp] .
- 14
- [DNR11] D. Duvenaud, H. Nickisch, and C. E. Rasmussen. "Additive
Gaussian Processes". In: NIPS. 2011.
- 15
- [Dom+06] P. Domingos, S. Kok, H. Poon, M. Richardson, and P.
Singla. "Unifying Logical and Statistical AI". In: IJCAI. 2006.
- 16
- [Dom+19] A.-K. Dombrowski, M. Alber, C. J. Anders, M. Ack-
ermann, K.-R. Müller, and P. Kessel. "Explanations can be
manipulated and geometry is to blame". In: arXiv preprint
arXiv:1906.07983 (2019).
- 17
- [Don+17a] K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and
A. G. Wilson. "Scalable Log Determinants for Gaussian
Process Kernel Learning". In: NIPS. 2017, pp. 6327–6337.
- 18
- [Don+17b] Y. Dong, H. Su, J. Zhu, and F. Bao. "Towards inter-
pretable deep neural networks by leveraging adversarial exam-
ples". In: arXiv preprint arXiv:1708.05493 (2017).
- 19
- [Dor+16] V. Dorie, M. Harada, N. B. Carnegie, and J. Hill. "A
flexible, interpretable framework for assessing sensitivity to un-
measured confounding". In: Statistics in Medicine 35.20 (2016),
pp. 3453–3470. print: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.6973>.
- 20
- [Doy+07] K. Doya, S. Ishii, A. Pouget, and R. P. N. Rao, eds.
Bayesian Brain: Probabilistic Approaches to Neural Coding.
MIT Press, 2007.
- 21
- [DR11] M. P. Deisenroth and C. E. Rasmussen. "PILCO: A
Model-Based and Data-E cient Approach to Policy Search".
In: ICML. 2011.
- 22
- [DR17] G. K. Dziugaite and D. M. Roy. "Computing nonvacuous
generalization bounds for deep (stochastic) neural networks
with many more parameters than training data". In: UAI. 2017.
- 23
- [DRG15] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani.
"Training generative neural networks via Maximum Mean Dis-
crepancy optimization". In: ICML. 2015.
- 24
- [Dru08] J. Dragomir. Bayesian linear regression. Tech. rep.
U. Rochester, 2008.
- 25
- [DS15] J. Dahlin and T. B. Schön. "Getting Started with Parti-
cle Metropolis-Hastings for Inference in Nonlinear Dynamical
Models". In: J. Stat. Softw. (Nov. 2015).
- 26
- [DS18] J. Domke and D. R. Sheldon. "Importance Weighting
and Variational Inference". In: NIPS. Curran Associates, Inc.,
2018, pp. 4474–4483.
- 27
- [DS19] J. Donahue and K. Simonyan. "Large scale adversarial
representation learning". In: arXiv preprint arXiv:1907.02544
(2019).
- 28
- [DSDB17] L. Dinh, J. Sohl-Dickstein, and S. Bengio. "Density
estimation using Real NVP". In: ICLR. 2017.
- 29
- [DSK16] V. Dumoulin, J. Shlens, and M. Kudlur. "A Learned
Representation For Artistic Style". In: (2016).
- 30
- [DSZ16] A. Datta, S. Sen, and Y. Zick. "Algorithmic trans-
parency via quantitative input uence: Theory and experi-
ments with learning systems". In: 2016 IEEE symposium on
security and privacy (SP). IEEE. 2016, pp. 598–617.
- 31
- [DTK16] J. Diplonga, S. Tschiatschek, and A. Krause. "Vari-
ational inference in mixed probabilistic submodular models".
In: Advances in Neural Information Processing Systems. 2016,
pp. 1759–1767.
- 32
- [Du+16] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-
Rodriguez, and J. Song. "Recurrent market-temporal point pro-
cesses: Embedding multi-hop context". In: Proceedings of the
22nd ACM SIGKDD International Conference on Knowl-
edge Discovery and Data Mining. 2016, pp. 1555–1564.
- 33
- [Du+18] J. Du, S. Ma, Y.-C. Wu, S. Kar, and J. M. F. Moura.
"Convergence Analysis of Distributed Inference with Vector-
Valued Gaussian Belief Propagation". In: JMLR 18.172 (2018),
pp. 1–38.
- 34
- [Du+19] S. S. Du, K. Hou, B. Póczos, R. Salakhutdinov, R.
Wang, and K. Xu. "Graph Neural Tangent Kernel: Fusing
Graph Neural Networks with Graph Kernels". In: (May 2019).
arXiv: 1905.13192 [cs.LG].
- 35
- [Du+20] Y. Du, S. Li, J. Tenenbaum, and I. Mordatch. "Im-
proved Contrastive Divergence Training of Energy-Based Mod-
els". In: arXiv preprint arXiv:2012.01316 (2020).
- 36
- [Du+21] C. Du, Z. Gao, S. Yuan, L. Gao, Z. Li, Y. Zeng, X. Zhu,
J. Xu, K. Gai, and K.-C. Lee. "Exploration in Online Advertis-
ing Systems with Deep Uncertainty-Aware Learning". In: KDD.
KDD '21, Virtual Event, Singapore: Association for Computing
Machinery, Aug. 2021, pp. 2792–2801.
- 37
- [Du+87] S. Duane, A. Kennedy, B. Pendleton, and D. Roweth.
"Hybrid Monte Carlo". In: Physics Letters B 195.2 (1987),
pp. 216–222.
- 38
- [Dub+16] A. Dubey, S. J. Reddi, B. Póczos, A. J. Smola, E. P.
Xing, and S. A. Williamson. "Variance Reduction in Stochas-
tic Gradient Langevin Dynamics". en: In: NIPS. Vol. 29. Dec.
2016, pp. 1154–1162.
- 39
- [Dud13] J. Duda. "Asymmetric numeral systems: entropy cod-
ing combining speed of Hu man coding with compression rate
of arithmetic coding". In: (Nov. 2013). arXiv: 1311.2540 [cs.IT].
- 40
- [Duf02] M. Du. "Optimal Learning: Computational procedures
for Bayes-adaptive Markov decision processes". PhD thesis. U.
Mass. Dept. Comp. Sci., 2002.
- 41
- [Dum+16] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro,
A. Lamb, M. Arjovsky, and A. Courville. "Adversarially learned
inference". In: arXiv preprint arXiv:1606.00704 (2016).
- 42
- [Dum19] V. Dumoulin, N. Houlsby, U. Evci, X. Zhai, R.
Goroshin, S. Gelly, and H. Larochelle. "A Unified Few-Shot
Classification Benchmark to Compare Transfer and Meta Learn-
ing Approaches". In: NIPS Datasets and Benchmarks Track.
June 2019.
- 43
- [Dur+19] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios.
"Cubic-Spline Flows". In: ICML Workshop on Invertible
Neural Networks and Normalizing Flows. 2019.
- 44
- [Dur+98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison.
Biological Sequence Analysis: Probabilistic Models of Proteins and
Nucleic Acids. Cambridge University Press, 1998.
- 45
- [Dus+20] M. Dusenberry, G. Jerfel, Y. Wen, Y. Ma, J. Snoek,
K. Heller, B. Lakshminarayanan, and D. Tran. "E cient and
scalable bayesian neural nets with rank-1 factors". In: Interna-
tional conference on machine learning. PMLR. 2020, pp. 2782–
2792.
- 46
- [Dut+21] V. Dutordoir, J. Hensman, M. van der Wilk, C. H. Ek,
Z. Ghahramani, and N. Durrande. "Deep Neural Networks as
Point Estimates for Deep Gaussian Processes". In: (May 2021).
arXiv: 2105.04504 [stat.ML].
- 47
- [Duv+13] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum,
and G. Zoubin. "Structure Discovery in Nonparametric Regres-
sion through Compositional Kernel Search". en: In: ICML. Feb.
2013, pp. 1166–1174.
- 48
- [Duv14] D. Duvenaud. "Automatic Model Construction with
Gaussian Processes". PhD thesis. Computational and Biologi-
cal Learning Laboratory, University of Cambridge, 2014.
- 49
- [dv04] D. P. de Faria and B. Van Roy. "On Constraint Sam-
pling in the Linear Programming Approach to Approximate Dy-
namic Programming". In: Mathematics of Operations Research
29.3 (2004), pp. 462–478.
- 50
- [Dv+17] H. De Vries, F. Strub, J. Mary, H. Larochelle, O.
Pitkänen, and A. C. Courville. "Modulating early visual pro-
cessing by language". In: Advances in Neural Information Pro-
cessing Systems. 2017, pp. 6594–6604.
- 51
- [Dv75] M. D. Donsker and S. S. Varadhan. "Asymptotic eval-
uation of certain Markov process expectations for large time, I".
In: Communications on Pure and Applied Mathematics 28.1
(1975), pp. 1–47.
- 52
- [Dv99] A. P. Dawid and V. Vovk. "Prequential probability: Prin-
ciples and properties". In: Bernoulli 5 (1999), pp. 125–162.

- 1
- 2 [DVK17] F. Doshi-Velez and B. Kim. Towards A Rigorous Science
3 of Interpretable Machine Learning. 2017. eprint:
4 1702.08608
- 5 [DW19] B. Dai and D. Wipf. "Diagnosing and Enhancing VAE
6 Models". In: ICLR. 2019.
- 7 [DWS12] T. Degris, M. White, and R. S. Sutton. "O-Policy
8 Actor-Critic". In: ICML. 2012.
- 9 [DWW19] B. Dai, Z. Wang, and D. Wipf. "The Usual Suspects?
10 Reassessing Blame for VAE Posterior Collapse". In: (Dec. 2019).
11 arXiv: 1912.10702 [cs.LG].
- 12 [DWW20] B. Dai, Z. Wang, and D. Wipf. "The Usual Suspects?
13 Reassessing Blame for VAE Posterior Collapse". In: ICML. Ed.
14 by H. D. Iii and A. Singh. Vol. 119. Proceedings of Machine
15 Learning Research. PMLR, 2020, pp. 2313–2322.
- 16 [DY79] P. Diaconis and D. Ylvisaker. "Conjugate priors for ex-
17 ponential families". In: vol. 7. 1979, pp. 269–281.
- 18 [ECM18] P. Etoori, M. Chinakota, and R. Mamidi. "Auto-
19 matic Spelling Correction for Resource-Scarce Languages us-
20 ing Deep Learning". In: Proceedings of ACL 2018, Student Re-
21 search Workshop. Melbourne, Australia: Association for Com-
22 putational Linguistics, 2018, pp. 146–152.
- 23 [ED05] D. Earl and M. Deem. "Parallel tempering: Theory,
24 applications, and new perspectives". In: Phys. Chem. Chem. Phys. 7 (2005), p. 3910.
- 25 [Edm69] H. P. Edmundson. "New methods in automatic extract-
26 ing". In: Journal of the ACM (JACM) 16.2 (1969), pp. 264–
27 285.
- 28 [Edm70] J. Edmonds. "Matroids, submodular functions, and cer-
29 tain polyhedra". In: Combinatorial Structures and Their Ap-
30 plications (1970), pp. 69–87.
- 31 [EFL04] E. Erosheva, S. Fienberg, and J. La hery. "Mixed-
32 membership models of scienti c publications". In: PNAS 101
33 (2004), pp. 5220–2227.
- 34 [Efr11] B. Efron. "Tweedie's Formula and Selection Bias". en. In:
35 J. Am. Stat. Assoc. 106.496 (2011), pp. 1602–1614.
- 36 [Efr86] B. Efron. "Why Isn't Everyone a Bayesian?" In: The
37 American Statistician 40.1 (1986).
- 38 [EGW05] D. Ernst, P. Geurts, and L. Wehenkel. "Tree-Based
39 Batch Mode Reinforcement Learning". In: JMLR 6 (2005),
40 pp. 503–556.
- 41 [Eis16] J. Eisner. "Inside-Outside and Forward-Backward Algo-
42 rithms Are Just Backprop (Tutorial Paper)". In: EMNLP Work-
43 shop on Structured Prediction for NLP. 2016.
- 44 [Eke+13] M. Ekeberg, C. Lökvist, Y. Lan, M. Weigt, and E.
45 Aurell. "Improved contact prediction in proteins: using pseudo-
46 likelihoods to infer Potts models". en. In: Phys. Rev. E Stat.
47 Nonlin. Soft Matter Phys. 87.1 (Jan. 2013), p. 012707.
- 48 [Ele+17] S. Elefteriadi, T. F. W. Nicholson, M. P. Deisen-
49 roth, and J. Hensman. "Identification of Gaussian Process State
50 Space Models". In: NIPS. 2017.
- 51 [EMH19] T. Elsken, J. H. Metzen, and F. Hutter. "Neural Ar-
52 chitecture Search: A Survey". In: JMLR 20 (2019), pp. 1–21.
- 53 [EMK06] G. Elidan, I. McGraw, and D. Koller. "Residual be-
54 lief propagation: Informed scheduling for asynchronous mes-
55 sage passing". In: UAI. 2006.
- 56 [Eng+18] J. Engel, K. K. Agarwal, S. Chen, I. Gulrajani, C.
57 Donahue, and A. Roberts. "GANSynth: Adversarial Neural Au-
58 dio Synthesis". In: International Conference on Learning Rep-
59 resentations. 2018.
- 60 [Erh+09] D. Erhan, Y. Bengio, A. Courville, and P. Vincent.
61 "Visualizing higher-layer features of a deep network". In: Uni-
62 versity of Montreal 1341.3 (2009), p. 1.
- 63 [ERO21] P. Esser, R. Rombach, and B. Ommer. "Taming Trans-
64 formers for High-Resolution Image Synthesis". In: CVPR. 2021.
- 65 [Eve09] G. Evensen. Data Assimilation: The Ensemble Kalman
66 Filter. en. 2nd ed. 2009 edition. Springer, Aug. 2009.
- 67 [Ew95] M. D. Escobar and M. West. "Bayesian density estima-
68 tion and inference using mixtures". In: JASA 90.430 (1995),
69 pp. 577–588.
- 70 [Ewe72] W. J. Ewens. "The sampling theory of selectively neu-
71 tral alleles". In: Theoretical population biology 3.1 (1972),
72 pp. 87–112.
- 73 [Ewe90] W. J. Ewens. "Population genetics theory - the past and
74 the future". In: Mathematical and Statistical Developments of
75 Evolutionary Theory. Ed. by S. Lessard. Reidel, 1990, pp. 177–
76 227.
- 77 [EY09] M. Elad and I. Yavneh. "A plurality of sparse represen-
78 tations is better than the sparest one alone". In: IEEE Trans. on Info.
79 Theory 55.10 (2009), pp. 4701–4714.
- 80 [Eyk+18] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rah-
81 mati, C. Xiao, A. Prakash, T. Kohno, and D. Song. "Robust
82 Physical-World Attacks on Deep Learning Models". In: CVPR.
83 2018.
- 84 [Fad+20] S. G. Fadel, S. Mair, R. da S. Torres, and U. Brefeld.
85 "Principled Interpolation in Normalizing Flows". In: (Oct.
86 2020). arXiv: 2010.12059 [stat.ML].
- 87 [FAL17] C. Finn, P. Abbeel, and S. Levine. "Model-Agnostic
88 Meta-Learning for Fast Adaptation of Deep Networks". In:
89 ICML. 2017.
- 90 [Fan+17] H. Fang, N. Tian, Y. Wang, M. Zhou, and M. A. Haile.
91 "Nonlinear Bayesian Estimation: From Kalman Filtering to a
92 Broader Horizon". In: (Dec. 2017). arXiv: 1712.01406 [cs.SY].
- 93 [Fan+19] C. Fang, Y. Gu, W. Zhang, and T. Zhang. "Convex
94 Formulation of Overparameterized Deep Neural Networks". In:
95 (Nov. 2019). arXiv: 1911.07626 [cs.LG].
- 96 [Fau+18] L. Fauré, F. Vasile, C. Calauzén, and O. Fercoq.
97 "Neural Generative Models for Global Optimization with Gra-
98 dients". In: (May 2018). arXiv: 1805.08594 [cs.NE].
- 99 [FB19] E. M. Feit and R. Berman. "Test & Roll:
100 Maximizing A/B Tests". In: Marketing Science 38.6 (Nov.
101 2019), pp. 1038–1058.
- 102 [FBW21] M. Finzi, G. Benton, and A. G. Wilson. "Residual
103 Pathway Priors for Soft Equivariance Constraints". In: NIPS.
104 2021.
- 105 [FC03] P. Fearnhead and P. Clifford. "On-line inference for hid-
106 den Markov models via particle filters". en. In: J. of Royal Stat.
107 Soc. Series B 63.4 (Nov. 2003), pp. 887–899.
- 108 [FCR14] R. Frigola, Y. Chen, and C. E. Rasmussen. "Variational
109 Gaussian Process State-Space Models". In: NIPS. 2014.
- 110 [FD07a] B. Frey and D. Dueck. "Clustering by Passing Mes-
111 sages Between Data Points". In: Science 315 (2007), 972–976.
- 112 [FD07b] B. J. Frey and D. Dueck. "Clustering by passing
113 messages between data points". In: science 315.5814 (2007),
114 pp. 972–976.
- 115 [FDF19] A. M. Franks, A. D'Amour, and A. Feller. "Flexible
116 Sensitivity Analysis for Observational Studies Without Observa-
117 tional Implications". In: Journal of the American Statistical As-
118 sociation 0.0 (2019), pp. 1–33. eprint: <https://doi.org/10.1080/01621459.2019.1604369>.
- 119 [FDZ19] A. Fasano, D. Durante, and G. Zanella. "Scalable and
120 Accurate Variational Bayes for High-Dimensional Binary Re-
121 gression Models". In: (Nov. 2019). arXiv: 1911.06743 [stat.ME].
- 122 [FE73] M. Fischler and R. Elschlager. "The representation and
123 matching of pictorial structures". In: IEEE Trans. on Com-
124 puter 22.1 (1973).
- 125 [Fed+18] W. Fedus, M. Rosca, B. Lakshminarayanan, A. M.
126 Dai, S. Mohamed, and I. Goodfellow. "Many Paths to Equilib-
127 rium: GANs Do Not Need to Decrease a Divergence At Every
128 Step". In: International Conference on Learning Representa-
129 tions. 2018.
- 130 [Fei98] U. Feige. "A threshold of $\ln n$ for approximating set
131 cover". In: Journal of the ACM (JACM) (1998).
- 132 [Fel+10] P. Felzenszwalb, R. Girshick, D. McAllester, and D.
133 Ramanan. "Object Detection with Discriminatively Trained
134 Part-Based Models". In: IEEE PAMI 32.9 (2010).
- 135 [Fel+19] M. Fellows, A. Mahajan, T. G. J. Rudner, and S.
136 Whiteson. "VIREL: A Variational Inference Framework for Re-
137 enforcement Learning". In: NeurIPS. 2019, pp. 7120–7134.
- 138 [Fen+21] S. Y. Feng, V. Gangal, J. Wei, S. Chandar,
139 Vosoughi, T. Mitamura, and E. Hovy. "A Survey of Data Aug-
140 mentation Approaches for NLP". In: (May 2021). arXiv:
141 2007.05078 [cs.CL].
- 142 [Fer73] T. S. Ferguson. "A Bayesian analysis of some nonpara-
143 metric problems". In: The annals of statistics (1973), pp. 209–
144 230.
- 145 [Feu+15] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg,
146 M. Blum, and F. Hutter. "Ecient and Robust Automated Ma-
147 chine Learning". In: NIPS. 2015, pp. 2962–2970.
- 148 [FG15] N. Fournier and A. Guillin. "On the rate of convergence
149 in Wasserstein distance of the empirical measure". In: Prob-
150 ability Theory and Related Fields 162.3 (2015), pp. 707–738.
- 151 [FG18] S. Farquhar and Y. Gal. "Towards Robust Evaluations of
152 Continual Learning". In: (May 2018). arXiv: 1805.09733 [stat.ML].
- 153 [FGG97] N. Friedman, D. Geiger, and M. Goldszmidt. "Bayesian
154 network classiers". In: MLJ 29 (1997), pp. 131–163.
- 155 [FH17] N. Frost and G. Hinton. Distilling a Neural Network
156 into a Soft Decision Tree. 2017. arXiv: 1711.09784 [cs.LG].
- 157 [FH20] E. Fong and C. Holmes. "On the marginal likelihood and
158 cross-validation". In: Biometrika 107.2 (May 2020).
- 159 [FH21] E. Fong and C. C. Holmes. "Conformal Bayesian Com-
160 putation". In: NIPS. May 2021.

BIBLIOGRAPHY

- <https://arxiv.org/abs/1912.02757>
- [FH75] K Fukunaga and L Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Trans. Inf. Theory* 21.1 (Jan. 1975), pp. 32–40.
- [FH97] B. J. Frey and G. Hinton. "Efficient stochastic source coding and an application to a Bayesian network source model". In: *Computer Journal* (1997).
- [FHDV20] J. Futoma, M. C. Hughes, and F. Doshi-Velez. "Popcorn: Partially observed prediction constrained reinforcement learning". In: *ICSTATS* (2020).
- [FKH03] P. Felzenszwalb, D. Huttenlocher, and J. Kleinberg. "Fast Algorithms for Large State Space HMMs with Applications to Web Usage Analysis". In: *NIPS*, 2003.
- [FHL19] S. Fort, H. Hu, and B. Lakshminarayanan. "Deep Ensembles: A Loss Landscape Perspective". In: (Dec. 2019). arXiv: 1912.02757 [stat.ML].
- [FHM18] S. Fujimoto, H. van Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *ICLR*, 2018.
- [FHT08] J. Friedman, T. Hastie, and R. Tibshirani. "Sparse inverse covariance estimation: the graphical lasso". In: *Biostatistics* 9.3 (2008), pp. 432–441.
- [FJ10] A. Fischer and C. Igel. "Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines". In: *International conference on artificial neural networks*. Springer, 2010, pp. 208–217.
- [Fie70] S. Fiernberg. "An Iterative Procedure for Estimation in Contingency Tables". In: *Annals of Mathematical Statistics* 41.3 (1970), pp. 907–917.
- [Fin+16] C. Finn, P. Christiano, P. Abbeel, and S. Levine. "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models". In: arXiv preprint arXiv:1611.03852 (2016).
- [Fis20] I. Fischer. "The Conditional Entropy Bottleneck". In: *Entropy* 22.9 (2020).
- [Fis25] R. Fisher. *Statistical Methods for Research Workers*. Biological monographs and manuals. Oliver and Boyd, 1925.
- [FJ02] M. A. T. Figueiredo and A. K. Jain. "Unsupervised Learning of Finite Mixture Models". In: *IEEE PAMI* 24.3 (2002), pp. 381–396.
- [FJL18] R. Frostig, M. J. Johnson, and C. Leary. "Compiling machine learning programs via high-level tracing". In: *Machine Learning and Systems (MLSys)* (2018).
- [FK13a] V. Feldman and P. Kothari. "Learning Coverage Functions". In: *CoRR* abs/1304.2079 (2013). arXiv: 1304.2079.
- [FK13b] M. Frei and H. R. Künsch. "Bridging the ensemble Kalman and particle filters". In: *Biometrika* 100.4 (Dec. 2013), pp. 781–800.
- [FK14] V. Feldman and P. Kothari. "Learning Coverage Functions and Private Release of Marginals". In: *Proceedings of The 27th Conference on Learning Theory*, Ed. by F. Balcan, V. Feldman, and C. Szepesvári, Vol. 35. *Proceedings of Machine Learning Research*. Barcelona, Spain: PMLR, 2014, pp. 679–702.
- [FK21] A. Fisher and E. H. Kennedy. "Visually Communicating and Teaching Intuition for Inverse Functions". In: *The American Statistician* 75.2 (2021), pp. 162–172. eprint: <https://doi.org/10.1080/00031305.2020.1717620>.
- [FKH17] S. Falkner, A. Klein, and F. Hutter. "Combining Hyperband and Bayesian Optimization". In: *NIPS 2017 Bayesian Optimization Workshop*. Dec. 2017.
- [FKV13] V. Feldman, P. Kothari, and J. Vondrák. "Representation, Approximation and Learning of Submodular Functions Using Low-rank Decision Trees". In: *Proceedings of the 26th Annual Conference on Learning Theory*, Ed. by S. Shalev-Shwartz and I. Steinwart, Vol. 30. *Proceedings of Machine Learning Research*. Princeton, NJ, USA: PMLR, 2013, pp. 711–740.
- [FKV14] V. Feldman, P. Kothari, and J. Vondrák. "Nearly tight bounds on ℓ_1 approximation of self-bounding functions". In: *CoRR*, abs/1404.4702 1 (2014).
- [FKV17] V. Feldman, P. Kothari, and J. Vondrák. "Tight Bounds on ℓ_1 Approximation and Learning of Self-Bounding Functions". In: *International Conference on Algorithmic Learning Theory*. PMLR, 2017, pp. 540–559.
- [FKV20] V. Feldman, P. Kothari, and J. Vondrák. "Tight bounds on ℓ_1 approximation and learning of self-bounding functions". In: *Theoretical Computer Science* 808 (2020), pp. 86–98.
- [FL07] P. Fearnhead and Z. Liu. "Online Inference for Multiple Changepoint Problems". In: *J. of Royal Stat. Soc. Series B* 69 (2007), pp. 589–605.
- [FL11] P. Fearnhead and Z. Liu. "Efficient Bayesian analysis of multiple changepoint models with dependence across segments". In: *Statistics and Computing* 21.2 (2011), pp. 217–229.
- [FL18] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3 (2018).
- [FL+16] C. Finn, S. Levine, and P. Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *ICML*, 2016, pp. 49–58.
- [Fla+16] S. Flaxman, D. Sejdinovic, J. P. Cunningham, and S. Filippi. "Bayesian Learning of Kernel Embeddings". In: *UAI*, 2016.
- [FLL18] J. Fu, K. Luo, and S. Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *ICLR*, 2018.
- [FLL19] Y. Feng, L. Li, and Q. Liu. "A Kernel Loss for Solving the Bellman Equation". In: *NeurIPS*, 2019, pp. 15430–15441.
- [FMM18] M. Figurnov, S. Mohamed, and A. Mnih. "Implicit Reparameterization Gradients". In: *NIPS*, 2018.
- [FMP19] S. Fujimoto, D. Meger, and D. Precup. "On-Policy Deep Reinforcement Learning without Exploration". In: *ICML*, 2019, pp. 2052–2062.
- [FN00] N. de Freitas, M. Niranjan, and A. Gee. "Hierarchical Bayesian models for regularization in sequential learning". In: *Neural Computation* 12.4 (2000), pp. 955–993.
- [FNW78] M. Fisher, G. Nemhauser, and L. Wolsey. "An analysis of approximations for maximizing submodular set functions—I". In: *Polyhedral combinatorics* (1978), pp. 73–87.
- [FO20] F. Farnia and A. Ozdaglar. "Do GANs always have Nash equilibria?" In: *Proceedings of the 37th International Conference on Machine Learning*, Ed. by H. D. III and A. Singh, Vol. 119. *Proceedings of Machine Learning Research*. PMLR, 2020, pp. 3029–3039.
- [Fon+21] D. Fontanelli, F. Cermelli, M. Mancini, and B. Caputo. "On the Challenges of Open World Recognition under Shifting Visual Domains". In: *ICRA*. July 2021.
- [For01] G. D. Forney. "Codes on graphs: normal realizations". In: *IEEE Trans. Inf. Theory* 47.2 (Feb. 2001), pp. 520–548.
- [For+18a] V. Fortuin, G. Dresdner, H. Strathmann, and G. Rätsch. "Scalable Gaussian Processes on Discrete Domains". In: (Oct. 2018). arXiv: 1810.10388 [stat.ML].
- [For+18b] M. Fortunato et al. "Noisy Networks for Exploration". In: *ICLR*, 2018.
- [For+19] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk. "Adversarial Examples Are a Natural Consequence of Test Error in Noise". In: (Jan. 2019). arXiv: 1901.10513 [cs.LG].
- [For21] V. Fortuin. "Priors in Bayesian Deep Learning: A Review". In: (May 2021). arXiv: 2105.06868 [stat.ML].
- [For+95] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. "The BATmobile: Towards a Bayesian Automated Taxi". In: *IJCAI*, 1995.
- [For+14] N. Foti, J. Xu, D. Laird, and E. Fox. "Stochastic variational inference for hidden Markov models". In: *NIPS*, 2014, pp. 3599–3607.
- [FOP08] N. Friel and A. N. Pettitt. "Marginal Likelihood Estimation via Power Posteriors". In: *J. of Royal Stat. Soc. Series B* 70.3 (2008), pp. 589–607.
- [Fro69] D. C. Fraser and J. E. Potter. "The optimum linear smoother as a combination of two optimum linear filters". In: *IEEE Trans. on Automatical Control* (1969), pp. 387–390.
- [FPD09] P. Frazier, W. Powell, and S. Dayanik. "The knowledge-gradient policy for correlated normal beliefs". In: *INFORMS J. on Computing* 21.4 (2009), pp. 599–613.
- [Fra08] A. Fraser. *Hidden Markov Models and Dynamical Systems*. SIAM Press, 2008.
- [Fra+16] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. "Sequential Neural Models with Stochastic Layers". In: *NIPS*, 2016.
- [Fra08] P. I. Frazier. "Bayesian Optimization". In: *Recent Advances in Optimization and Modeling of Contemporary Problems*. INFORMS Tutorials in Operations Research. INFORMS, Oct. 2018, pp. 255–278.
- [Fre14] A. A. Freitas. "Comprehensible classification models: a position paper". In: *ACM SIGKDD explorations newsletter* 15.1 (2014), pp. 1–10.
- [Fre98] B. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.
- [Fre99] R. M. French. "Catastrophic forgetting in connectionist networks". In: *Trends in Cognitive Science* (1999).
- [Fri03] K. Friston. "Learning and inference in the brain". en. In: *Neural Netw.* 16.9 (Nov. 2003), pp. 1325–1352.
- [Fri09] K. Friston. "The free-energy principle: a rough guide to the brain?". en. In: *Trends Cogn. Sci.* 13.7 (July 2009), pp. 293–301.

- 1
- 2 [Fri+22] K. Friston, L. Da Costa, N. Sajid, C. Heins, R. B. Grosse, S. Ancha, and D. M. Roy. "Measuring
3 Ueltzhofer, G. A. Pavlotti, and T. Parr. "The free energy principle made simpler but not too simple". In: (Jan. 2022). arXiv:
4 2201.06387 [cond-mat.stat-mech]. [Gar16]
- 5 [Fro+21] R. Frostig, M. Johnson, D. Maclaurin, A. Paszke, and D. M. Roy. "Measuring the reliability of MCMC inference with bidirectional Monte
6 Radul. "Decomposing reverse-mode automatic differentiation". In: LAFI workshop at POPL. 2021. Carlo". In: NIPS. June 2016.
- 7 [FS07] S. Frühwirth-Schnatter. Finite Mixture and Markov
8 Switching Models. Springer, 2007.
- 9 [FSF10] S. Frühwirth-Schnatter and R. Frühwirth. "Data Augmentation and MCMC for Binary and Multinomial Logit Models". In: Statistical Modelling and Regression Structures. Ed.
10 by T. Klein and G. Tutz. Springer, 2010, pp. 111–132.
- 11 [FST98] S. Fine, Y. Singer, and N. Tishby. "The Hierarchical Hidden Markov Model: Analysis and Applications". In: Machine Learning 32 (1998), p. 41.
- 12 [FT05] M. Fasching and C. Tomasi. "Mean shift is a bound optimization". In: IEEE Trans. Pattern Anal. Mach. Intell. 27.3 (Mar. 2005), pp. 471–474.
- 13 [FT19] A. Finke and A. H. Thiery. "On the relationship between variational inference and adaptive importance sampling". In: July 2019. arXiv: 1907.10477 [stat.ML].
- 14 [FT74] J. H. Friedman and J. W. Tukey. "A Projection Pursuit Algorithm for Exploratory Data Analysis". In: IEEE Trans. Comput. C-23.9 (Sept. 1974), pp. 881–890.
- 15 [Fu15] M. Fu, ed. Handbook of Simulation Optimization. 1st ed. Springer-Verlag New York, 2015.
- 16 [Fu+17] Z. Fu, X. Tan, N. Peng, D. Zhao, and R. Yan. and "Style transfer in text: Exploration and evaluation". In: arXiv preprint arXiv:1711.06861 (2017).
- 17 [Fu+19] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin. "Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing". In: NAACL. Mar. 2019.
- 18 [Fu+20] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. arXiv:2004.07219, 2020.
- 19 [Fuji05] S. Fujishige. Submodular functions and optimization. Vol. 58. Elsevier Science, 2005.
- 20 [Ful+20] I. R. Fulcher, I. Shpitser, S. Marealle, and E. J. Tchetgen Tchetgen. "Robust inference on population indirect causal effects: the generalized front door criterion". In: Journal of the Royal Statistical Society: Series B (Statistical Methodology) 82.2 (2020), pp. 199–214. eprint: <https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12345>.
- 21 [FV15] V. Feldman and J. Vondrák. "Tight Bounds on Low-Degree Spectral Concentration of Submodular and XOS Functions". In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science. 2015, pp. 923–942.
- 22 [FV16] V. Feldman and J. Vondrák. "Optimal bounds on approximation of submodular and XOS functions by junta". In: SIAM Journal on Computing 45.3 (2016), pp. 1129–1170.
- 23 [FV17] R. C. Feng and A. Vedaldi. "Interpretable explanations of black boxes by meaningful perturbation". In: Proceedings of the IEEE international conference on computer vision. 2017, pp. 3429–3437.
- 24 [FW12] N. Friel and J. Wyse. "Estimating the evidence – a review". In: Stat. Neerl. 66.3 (Aug. 2012), pp. 288–308.
- 25 [FW21] R. Friedman and Y. Weiss. "Posterior Sampling for Image Restoration using Explicit Patch Priors". In: (Apr. 2021). arXiv: 2104.09895 [cs.CV].
- 26 [FWW21] M. Finzi, M. Welling, and A. G. Wilson. "A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups". In: ICML. 2021.
- 27 [Gai+19] A. Gajewski, J. Clune, K. O. Stanley, and J. Lehman. "Evolvability: ES: Scalable and Direct Optimization of Evolvability". In: Proc. of the Conf. on Genetic and Evolutionary Computation. 2019.
- 28 [Gan+16a] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, and others. "Domain-adversarial training of neural networks". In: JMLR (2016).
- 29 [Gan+16b] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. "Domain-adversarial training of neural networks". In: The journal of machine learning research 17.1 (2016), pp. 2096–2030.
- 30 [Gao+18] R. Gao, J. Xie, S.-C. Zhu, and Y. N. Wu. "Learning Grid-like Units with Vector Representation of Self-Position and Matrix Representation of Self-Motion". In: arXiv preprint arXiv:1810.05597 (2018).
- 31 [Gao+20] R. Gao, E. Nijkamp, D. P. Kingma, Z. Xu, A. M. Dai, and Y. N. Wu. "Flow contrastive estimation of energy-based models". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 7518–7528.
- 32 [Gär03] T. Gärtner. "A Survey of Kernels for Structured Data". In: SIGKDD Explor. Newslett. 5.1 (July 2003), pp. 49–58.
- 33 [Gar16] J. R. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. "GPY Torch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration". In: NIPS. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2016, pp. 7576–7586.
- 34 [Gar18a] J. R. Gardner, G. Pleiss, R. Wu, K. Q. Weinberger, and A. G. Wilson. "Product Kernel Interpolation for Scalable Gaussian Processes". In: AISTATS. 2018.
- 35 [Gar18c] T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, and A. G. Wilson. "Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs". In: NIPS. 2018.
- 36 [Gar18d] M. Garnelo, D. Rosenthal, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. M. A. Esfandi. "Conditional Neural Processes". In: ICML. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1704–1713.
- 37 [Gar18e] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. M. Ali Esfandi, and Y. W. Teh. "Neural Processes". In: ICML workshop on Theoretical Foundations and Applications of Deep Generative Models. 2018.
- 38 [Gar19] S. Garg, V. Perot, N. Lintiaoco, A. Taly, E. H. Chi, and A. Beutel. "Counterfactual Fairness in Text Classification through Robustness". In: Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society. AIES '19. Association for Computing Machinery, 2019, 219–226.
- 39 [Gar22] R. Garnett. Bayesian Optimization. in preparation. Cambridge University Press, 2022.
- 40 [Gas+19] J. Gasthaus, K. Benidis, Y. Wang, S. S. Rangapuram, D. Salinas, V. Flunkert, and T. Januschowski. "Probabilistic Forecasting with Spline Quantile Function RNNs". In: ICML. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1901–1910.
- 41 [GASG18] D. G. A. Smith and J. Gray. "opt-einsum - A Python package for optimizing contraction order for einsum-like expressions". In: JOSS 3.26 (June 2018), p. 753.
- 42 [GAZ19] A. Ghorbani, A. Abid, and J. Zou. "Interpretation of neural networks is fragile". In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 01. 2019, pp. 3681–3688.
- 43 [GB00] Z. Ghahramani and M. Beal. "Variational inference for Bayesian mixtures of factor analysers". In: NIPS-12. 2000.
- 44 [GB09] A. Guillory and J. Bilmes. "Label Selection on Graphs". In: NIPS. Vancouver, Canada, 2009.
- 45 [GB10] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: AISTATS. 2010, pp. 249–256.
- 46 [GB11] A. Guillory and J. Bilmes. "Active Semi-Supervised Learning using Submodular Functions". In: UAI. Barcelona, Spain. AUAI, 2011.
- 47 [GB18] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Parraguez, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules". In: American Chemical Society Central Science 4.2 (Feb. 2018), pp. 268–276.
- 48 [GBB11] X. Glorot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks". In: AISTATS. 2011.
- 49 [GBC16] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. <http://www.deeplearningbook.org>. MIT Press, 2016.
- 50 [GBT95] W. Gilks, N. Best, and K. Tan. "Adaptive rejection Metropolis sampling". In: Applied Statistics 44 (1995), pp. 455–472.
- 51 [GC90] R. P. Goldman and E. Charniak. "Dynamic Construction of Belief Networks". In: UAI. 1990.
- 52 [GCW19] M. Girolami and B. Calderhead. "Riemann manifold Langevin and Hamiltonian Monte Carlo methods". In: J. of Royal Stat. Soc. Series B 73.2 (Mar. 2011), pp. 123–214.
- 53 [GC99] R. P. Goldman and E. Charniak. "Dynamic Construction of Belief Networks". In: UAI. 1999.
- 54 [GCW19] M. Girolami, N. Chopin, and N. Whiteley. "Negative association, ordering and convergence of resampling methods". In: Ann. Stat. 47.4 (2019), pp. 2236–2260.
- 55 [GD20] T. Geiger and J. Domke. "A Rule for Gradient Estimator Selection, with an Application to Variational Inference". In: AISTATS. Ed. by S. Chappell and R. Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1803–1812.
- 56 [GDFY16] S. Ghosh, F. M. Delle Fave, and J. Yedidia. "Assume Density Filtering Methods for Learning Bayesian Neural Networks". In: AAAI. 2016.
- 57 [Geb+21] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. III, and K. Crawford. "Datasheets for datasets". In: Communications of the ACM 64.12 (2021), pp. 86–92.
- 58 [Gei+20a] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann. "Shortcut

BIBLIOGRAPHY

- 1 Learning in Deep Neural Networks". In: arXiv preprint
2 arXiv:2004.07780 (2020).
- 3 [Gei+20b] R. Geirhos, J. Jacobsen, C. Michaelis, R. S. Zemel,
4 W. Brendel, M. Bethge, and F. A. Wichmann. "Shortcut
5 Learning in Deep Neural Networks". In: CoRR abs/2004.07780
6 (2020). arXiv: 2004.07780.
- 7 [Gel+04] A. Gelman, J. Carlin, H. Stern, and D.
8 Bayesian data analysis. 2nd edition. Chapman and Hall, 2004.
- 9 [Gel06] A. Gelman. "Prior distributions for variance parameters
10 in hierarchical models (comment on article by Browne and
11 Draper)". en. In: Bayesian Anal. 1.3 (Sept. 2006), pp. 515–534.
- 12 [Gel+08] A. Gelman, A. Jakulin, M. G. Pittau, and Y.-S. Su. "A
13 weakly informative default prior distribution for logistic and
14 other regression models". en. In: The Annals of Applied Statistics
15 2.4 (Dec. 2008), pp. 1360–1383.
- 16 [Gel+14a] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson,
17 A. Vehtari, and D. B. Rubin. Bayesian Data Analysis, Third
18 Edition. Third edition. Chapman and Hall/CRC, 2014.
- 19 [Gel+14b] A. Gelman, A. Vehtari, P. Jylänki, C. Robert, N.
20 Chopin, and J. P. Cunningham. "Expectation propagation as
21 a way of life". In: (Dec. 2014). arXiv: 1412.4869 [stat.CO].
- 22 [Gel+20] A. Gelman, A. Vehtari, D. Simpson, C. C. Margossian,
23 B. Carpenter, Y. Yao, L. Kennedy, J. Gabry, P.-C. Bürkner,
24 and M. Modrák. "Bayesian Workflow". In: (Nov. 2020). arXiv:
25 2011.01808 [stat.ME].
- 26 [Gel90] M. Gelbrich. "On a formula for the L_2 Wasserstein metric
27 between measures on Euclidean and Hilbert spaces". In:
28 Mathematische Nachrichten 147.1 (1990), pp. 185–203.
- 29 [Gen+19] A. Genevay, L. Chizat, F. Bach, M. Cuturi, and G.
30 Peyré. "Sample complexity of sinkhorn divergences". In:
31 The 22nd International Conference on Artificial Intelligence and
32 Statistics. PMLR, 2019, pp. 1574–1583.
- 33 [Geo+18] T. George, C. Laurent, X. Bouthillier, N. Ballas, and
34 P. Vincent. "Fast Approximate Natural Gradient Descent in a
35 Kronecker Factored Eigenbasis". In: NIPS. Curran Associates,
36 Inc., 2018, pp. 9550–9560.
- 37 [Geo88] H.-O. Georgii. Gibbs Measures and Phase Transitions.
38 en. Walter De Gruyter Inc, Oct. 1988.
- 39 [Ger+15] M. Germain, K. Gregor, I. Murray, and H. Larochelle.
40 "MADE: Masked Autoencoder for Distribution Estimation". In:
41 ICML. Ed. by F. Bach and D. Blei, Vol. 37. Proceedings of
42 Machine Learning Research. Lille, France: PMLR, 2015, pp. 881–
43 889.
- 44 [Gér19] A. Géron. Hands-On Machine Learning with Scikit-
45 Learn and TensorFlow: Concepts, Tools, and Techniques
46 for Building Intelligent Systems (2nd edition). en. O'Reilly Media,
47 Incorporated, 2019.
- 48 [Gey92] C. Geyer. "Practical Markov chain Monte Carlo". In:
49 Statistical Science 7 (1992), pp. 473–483.
- 50 [GF00] E. George and D. Foster. "Calibration and empirical
51 Bayes variable selection". In: Biometrika 87.4 (2000), pp. 731–
52 747.
- 53 [GF09] I. E. Givoni and B. J. Frey. "A Binary Alternative Model
54 for Acity Propagation". In: Neural Computation 21.6 (2009),
55 pp. 1589–1600.
- 56 [GF17] B. Goodman and S. Flaxman. "European Union regulations
57 on algorithmic decision-making and a "right to explanation". In:
58 AI magazine 38.3 (2017), pp. 50–57.
- 59 [GFW020] T. Galy-Fajou, F. Wenzel, and M. Opper. "Automated
60 Augmented Conjugate Inference for Non-conjugate Gaussian
61 Process Models". In: AISTATS. 2020.
- 62 [GG11] T. L. Grifths and Z. Ghahramani. "The Indian Buffet
63 Process: An Introduction and Review." In: JMLR 12.4 (2011).
- 64 [GG14] S. J. Gershman and N. D. Goodman. "Amortized Inference
65 in Probabilistic Reasoning". In: 36th Annual Conference
66 of the Cognitive Science Society. 2014.
- 67 [GG16] Y. Gal and Z. Ghahramani. "Dropout as a Bayesian Ap-
68 proximation: Representing Model Uncertainty in Deep Learn-
69 ing". In: ICML. 2016.
- 70 [GG84] S. German and D. German. "Stochastic Relaxation, Gibbs
71 Distributions, and the Bayesian Restoration of Images". In:
72 IEEE PAMI 6.8 (1984).
- 73 [GGA15] R. B. Grosse, Z. Ghahramani, and R. P. Adams.
74 "Sandwiching the marginal likelihood using bidirectional Monte
75 Carlo". In: (Nov. 2015). arXiv: 1511.02543 [stat.ML].
- 76 [GGG15] M. Gygli, H. Grabner, and L. Gool. "Video summariza-
77 tion by learning submodular mixtures of objectives". In: 2015
78 IEEE Conference on Computer Vision and Pattern Recog-
79 nition (CVPR) (2015), pp. 3090–3098.
- 80 [GGO19] F. Gurkan, B. Gunsel, and C. Ozer. "Robust object
81 tracking via integration of particle filtering with deep detec-
82 tion". In: Digit. Signal Process. 87 (Apr. 2019), pp. 112–124.
- 83 [GGR21a] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling
84 Long Sequences with Structured State Spaces". In: (Oct. 2021).
85 arXiv: 2111.00396 [cs.LG].
- 86 [GGR21b] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling
87 Long Sequences with Structured State Spaces". In: (Oct. 2021).
88 arXiv: 2111.00396 [cs.LG].
- 89 [GGT15] S. Gu, Z. Ghahramani, and R. E. Turner. "Neural
90 Adaptive Sequential Monte Carlo". In: NIPS. 2015.
- 91 [GH07] A. Gelman and J. Hill. Data analysis using regression
92 and multilevel/hierarchical models. Cambridge, 2007.
- 93 [GH10] M. Gutmann and A. Hyvärinen. "Noise-contrastive es-
94 timation: A new estimation principle for unnormalized statistical
95 models". In: Proceedings of the Thirteenth International Con-
96 ference on Artificial Intelligence and Statistics. 2010, pp. 297–
97 304.
- 98 [GH12a] F. Gustafsson and G. Hendeby. "Some Relations Be-
99 tween Extended and Unscented Kalman Filters". In: IEEE
100 Trans. Signal Process. 60.2 (Feb. 2012), pp. 545–555.
- 101 [GH12b] M. Gutmann and J.-i. Hayama. "Bregman divergence
102 as general framework to estimate unnormalized statistical mod-
103 els". In: arXiv:1202.3727 (2012).
- 104 [GH96a] Z. Ghahramani and G. Hinton. Parameter estimation
105 for linear dynamical systems. Tech. rep. CRG-TR-96-2. Dept.
106 Comp. Sci., Univ. Toronto, 1996.
- 107 [GH96b] Z. Ghahramani and G. Hinton. The EM Algorithm for
108 Mixtures of Factor Analyzers. Tech. rep. Dept. of Comp. Sci.,
109 Univ. Toronto, 1996.
- 110 [Gha+15] M. Ghavamzadeh, S. Mannor, J. Pineau, and A.
111 Tamar. "Bayesian Reinforcement Learning: A Survey". In:
112 Foundations and Trends in ML (2015).
- 113 [Gha+21] A. Ghandeharioun, B. Kim, C.-L. Li, B. Jou, B. Eo,
114 and R. W. Picard. DISSECT: Disentangled Simultaneous Ex-
115 planations via Concept Traversals. 2021. arXiv:
116 2105.15164 [cs.LG].
- 117 [GHC20] C. Geng, S.-J. Huang, and S. Chen. "Recent Advances
118 in Open Set Recognition: A Survey". In: IEEE PAMI (2020).
- 119 [GHC21] S. Gould, R. Hartley, and D. J. Campbell. "Deep
120 Declarative Networks". en. In: IEEE PAMI PP (Feb. 2021).
- 121 [GHK17] Y. Gal, J. Hron, and A. Kendall. "Concrete Dropout".
122 In: (May 2017). arXiv: 1705.07832 [stat.ML].
- 123 [Gho+19] A. Ghorbani, J. Wexler, J. Zou, and B. Kim. Towards
124 Automatic Concept-based Explanations. 2019. arXiv:
125 1902.03129 [stat.ML].
- 126 [GHV20] A. Gelman, J. Hill, and A. Vehtari. Regression and
127 Other Stories. en. 1st ed. Cambridge University Press, July
128 2020.
- 129 [Gil+17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals,
130 and G. E. Dahl. "Neural message passing for quantum chem-
131 istry". In: ICML. 2017, pp. 1263–1272.
- 132 [Gil+18a] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen,
133 and G. E. Dahl. "Motivating the rules of the game for adver-
134 sarial example research". In: arXiv preprint arXiv:1807.06732
135 (2018).
- 136 [Gil+18b] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz,
137 M. Raghu, M. Wattenberg, and I. Goodfellow. "Adversarial
138 spheres". In: arXiv preprint arXiv:1801.02774 (2018).
- 139 [Gil88] J. R. Gilbert. "Some nested dissection order is nearly
140 optimal". In: Inf. Process. Lett. 26.6 (Jan. 1988), pp. 325–328.
- 141 [Gir+15] R. Girshick, F. Iandola, T. Darrell, and J. Malik. "De-
142 formable Part Models are Convolutional Neural Networks". In:
143 CVPR. 2015.
- 144 [Gir+20] L. Girin, S. Leglaive, X. Bie, J. Diard, T. Hueber,
145 and X. Alameda-Pineda. "Dynamical Variational Autoencoders:
146 A Comprehensive Review". In: (Aug. 2020). arXiv:
147 2008.12595 [cs.LG].
- 148 [Gir89] J. Gittins. Multi-armed Bandit Allocation Indices. Wi-
149 ley, 1989.
- 150 [GJ97] Z. Ghahramani and M. Jordan. "Factorial Hidden
151 Markov Models". In: Machine Learning 29 (1997), pp. 245–273.
- 152 [GK19] L. Graesser and W. L. Keng. Foundations of Deep Re-
153 infrelement-Learning: Theory and Practice in Python. en. 1
154 edition. Addison-Wesley Professional, Dec. 2019.
- 155 [GKS05] C. Guestrin, A. Krause, and A. P. Singh. "Near-
156 optimal sensor placements in gaussian processes". In: Proced-
157 ings of the 22nd international conference on Machine learning.
158 2005, pp. 265–272.
- 159 [GL10] K. Gregor and Y. LeCun. "Learning fast approximations
160 of sparse coding". In: ICML. 2010, pp. 399–406.
- 161 [Gla03] P. Glasserman. Monte Carlo Methods in Financial En-
162 gineering. 1st ed. Stochastic Modelling and Applied Probability.
163 Springer-Verlag New York, 2003.
- 164 [Gle02] S. Glennan. "Rethinking mechanistic explanation". In:
165 Philosophy of science 69.S3 (2002), S342–S353.

- 1
- 2 [GLM15] J. Ghosh, Y. Li, and R. Mitra. "On the Use of Cauchy
3 Prior Distributions for Bayesian Logistic Regression". In: July
4 2015. arXiv: 1507.07170 [stat.ME].
- 5 [GLP21] I. Gulrajani and D. Lopez-Paz. "In Search of Lost Do-
6 main Generalization". In: ICLR. 2021.
- 7 [GLS81] M. Grötschel, L. Lovász, and A. Schrijver. "The ellip-
8 soid method and its consequences in combinatorial optimiza-
9 tion". In: Combinatorica 1.2 (1981), pp. 169–197.
- 10 [GM12] G. Gordon and J. McNulty. Matroids: a geometric in-
11 troduction. Cambridge University Press, 2012.
- 12 [GM15] J. Gorham and L. Mackey. "Measuring sample quality
13 with Stein's method". In: Advances in Neural Information Pro-
14 cessing Systems. 2015, pp. 226–234.
- 15 [GM16] R. Grossé and J. Martens. "A Kronecker-factored ap-
16 proximate Fisher matrix for convolution layers". In: ICML.
17 2016.
- 18 [GM98] A. Gelman and X.-L. Meng. "Simulating normalizing
19 constants: from importance sampling to bridge sampling to
20 path sampling". In: Statistical Science 13 (1998), pp. 163–185.
- 21 [GMAR16] Y. Gal, R. T. McAllister, and C. E. Rasmussen. "Im-
22 proving PILCO with Bayesian Neural Network Dynamics Mod-
23 els". In: ICML workshop on Data-e cient machine learning.
2016.
- 24 [GMH20] M. Gorinova, D. Moore, and M. Ho man. "Automatic
25 Reparameterisation of Probabilistic Programs". In: ICML.
26 Vol. 119. Proceedings of Machine Learning Research. PMLR,
27 2020, pp. 3648–3657.
- 28 [GNM19] D. Greenberg, M. Nonnenmacher, and J. Macke. "Au-
29 tomatic Posterior Transformation for Likelihood-Free Infer-
30 ence". In: ICML. 2019.
- 31 [GO17] P. Grünwald and T. van Ommen. "Inconsistency of
32 Bayesian Inference for Misspecified Linear Models, and a Pro-
33 posal for Repairing It". en. In: Bayesian Analysis 12.4 (Dec.
34 2017), pp. 1069–1103.
- 35 [Goe+09] M. X. Goemans, N. J. Harvey, S. Iwata, and V. Mir-
36 rokni. "Approximating submodular functions everywhere". In:
37 Proceedings of the twentieth annual ACM-SIAM symposium
38 on Discrete algorithms. SIAM, 2009, pp. 535–544.
- 39 [Gol+17] N. Gold, M. G. Frasch, C. Herry, B. S. Richardson,
40 and X. Wang. "A Doubly Stochastic Change Point Detection
41 Algorithm for Noisy Biological Signals". en. In: Front. Physiol.
42 (Oct. 2017), p. 106088.
- 43 [Gom+17] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grossé.
44 "The Reversible Residual Network: Backpropagation Without
45 Storing Activations". In: NIPS. 2017.
- 46 [Gon+11] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin.
47 "Parallel gibbs sampling: From colored elds to thin junction
48 trees". In: AISTATS. 2011, pp. 324–332.
- 49 [Gon+14] B. Gong, W.-L. Chao, K. Grauman, and F. Sha. "Di-
50 verse sequential subset selection for supervised video summa-
51 rization". In: Advances in neural information processing sys-
52 tems 27 (2014), pp. 2069–2077.
- 53 [Gon+20] P. J. Goncalves et al. "Training deep neural density
54 estimators to identify mechanistic models of neural dynamics".
55 In: Elife 9 (2020).
- 56 [Goo+14a] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu,
57 D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Gen-
58 erative adversarial nets". In: NIPS. 2014, pp. 2672–2680.
- 59 [Goo+14b] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu,
60 D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Gen-
61 erative Adversarial Networks". In: NIPS. 2014.
- 62 [Goo+16] I. Goodfellow. "NIPS 2016 Tutorial: Generative Ad-
63 versarial Networks". In: NIPS Tutorial. 2016.
- 64 [Goo85] I. Good. "Weight of evidence: A brief survey". In:
65 Bayesian statistics 2 (1985), pp. 249–270.
- 66 [Gor+14] A. D. Gordon, T. A. Henzinger, A. V. Nori, and
67 S. K. Rajamani. "Probabilistic programming". In: Intl. Conf.
68 on Software Engineering. 2014.
- 69 [Gor+19] J. Gordon, J. Bronskill, M. Bauer, S. Nowozin, and
70 R. E. Turner. "Meta-Learning Probabilistic Inference For Pre-
71 diction". In: ICLR. 2019.
- 72 [Gor93] N. Gordon. "Novel Approach to Nonlinear/Non-
73 Gaussian Bayesian State Estimation". In: IEE Proceedings (F)
74 140.2 (1993), pp. 107–113.
- 75 [Gor95] G. J. Gordon. "Stable Function Approximation in Dy-
76 namic Programming". In: ICML. 1995, pp. 261–268.
- 77 [Gou+96] C. Gourieroux, M. Gourieroux, A. Monfort, and D. A.
78 Monfort. Simulation-based econometric methods. Oxford uni-
79 versity press, 1996.
- 80 [Goy+19] Y. Goyal, Z. Wu, J. Ernst, D. Batra, D. Parikh, and
81 S. Lee. Counterfactual Visual Explanations. 2019. arXiv:
82 07451 [cs.LG].
- 83 [GPS89] D. Greig, B. Porteous, and A. Seheult. "Exact maxi-
84 mum a posteriori estimation for binary images". In: J. of Royal
85 Stat. Soc. Series B 51.2 (1989), pp. 271–279.
- 86 [Gro6a] M. Girolami and S. Rogers. "Variational Bayesian Multi-
87 nomial Probit Regression with Gaussian Process Priors". In:
88 Neural Comput. 18.8 (Aug. 2006), pp. 1790–1817.
- 89 [Gro6b] M. Girolami and S. Rogers. "Variational Bayesian
90 multinomial probit regression with Gaussian process priors". In:
91 Neural Computation 18.8 (2006), pp. 1790 –1817.
- 92 [Gro7a] T. Gneiting and A. E. Raftery. "Strictly Proper Scoring
93 Rules, Prediction, and Estimation". In: JASA 102.477
94 (2007), pp. 359–378.
- 95 [Gro7b] T. Gneiting and A. E. Raftery. "Strictly proper scoring
96 rules, prediction, and estimation". In: Journal of the American
97 statistical Association 102.477 (2007), pp. 359–378.
- 98 [Gra10] T. Graepel, J. Quinonero-Candela, T. Borchert, and
99 R. Herbrich. "Web-Scale Bayesian Click-Through Rate Pre-
100 diction for Sponsored Search Advertising in Microsoft's Bing
101 Search Engine". In: ICML. 2010.
- 102 [Gra11] A. Graves. "Practical Variational Inference for Neural
103 Networks". In: NIPS. 2011.
- 104 [Gra18] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I.
105 Sutskever, and D. Duvenaud. "FFJORD: Free-form Continuous
106 Dynamics for Scalable Reversible Generative Models". In: (Oct.
107 2018). arXiv: 1810.01367 [cs.LG].
- 108 [Gra20a] W. Grathwohl, J. Kelly, M. Hashemi, M. Norouzi, K.
109 Swersky, and D. Duvenaud. "No MCMC for me: Amortized sam-
110 pling for fast and stable training of energy-based models". In:
111 arXiv preprint arXiv:2010.04230 (2020).
- 112 [Gra20b] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duve-
113 nau, M. Norouzi, and K. Swersky. "Your classifier is secretly
114 an energy based model and you should treat it like one". In:
115 ICLR. 2020.
- 116 [Gra20c] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duve-
117 nau, and R. Zemel. "Cutting out the Middle-Man: Training
118 and Evaluating Energy-Based Models without Sampling". In:
119 arXiv preprint arXiv:2002.05616 (2020).
- 120 [Gre03] P. Green. "Tutorial on trans-dimensional MCMC". In:
121 Highly Structured Stochastic Systems. Ed. by P. Green, N.
122 Hjort, and S. Richardson. OUP, 2003.
- 123 [Gre12] A. Gretton, K. M. Borgwardt, M. J. Rasch,
124 Schölkopf, and A. Smola. "A Kernel Two-Sample Test". In:
125 JMLR 13 Mar (2012), pp. 723–773.
- 126 [Gre14] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D.
127 Wierstra. "Deep AutoRegressive Networks". In: ICML. 2014.
- 128 [Gre20] F. Greenlee. Transformer VAE. 2020.
- 129 [Gre98] P. Green. "Reversible Jump Markov Chain Monte
130 Carlo computation and Bayesian model determination". In:
131 Biometrika 85 (1998), pp. 711–732.
- 132 [Gri20] T. L. Griffiths. "Understanding Human Intelligence
133 through Human Limitations". en. In: Trends Cogn. Sci. 24.11
134 (Nov. 2020), pp. 873–883.
- 135 [GRS96] W. Gilks, S. Richardson, and D. Spiegelhalter. Markov
136 Chain Monte Carlo in Practice. Chapman and Hall, 1996.
- 137 [GS08] Y. Guo and D. Schuurmans. "Efficient global optimization
138 for exponential family PCA and low-rank matrix factorization".
139 In: 2008 46th Annual Allerton Conference on Communication,
140 Control, and Computing. Sept. 2008, pp. 1100–1107.
- 141 [GS15] R. B. Grossé and R. Salakhutdinov. "Scaling Up Natural
142 Gradient by Sparsely Factoring the Inverse Fisher Matrix".
143 In: ICML. 2015.
- 144 [GS16] A. Gelfand and A. Smith. "Sampling-based approaches
145 to calculating marginal densities". In: JASA 85 (1990), pp. 385–
146 409.
- 147 [GS92] G. Grimmett and D. Stirzaker. Probability and Random
148 Processes. Oxford, 1992.
- 149 [GSA14] M. A. Gelbart, J. Snoek, and R. P. Adams. "Bayesian
150 Optimization with Unknown Constraints". In: UAI. 2014.
- 151 [GSD13] A. Guez, D. Silver, and P. Dayan. "Scalable and E-
152 cient Bayes-Adaptive Reinforcement Learning Based on Monte-
153 Carlo Tree Search". In: JAIR 48 (2013), pp. 841–883.
- 154 [GSZ19] A. Gretton, D. Sutherland, and W. Jitkrittum. NIPS
155 tutorial on interpretable comparison of distributions and mod-
156 els. 2019.
- 157 [GSS15] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explain-
158 ing and Harnessing Adversarial Examples". In: ICLR. 2015.
- 159 [GSZ21] P. Grünwald, T. Steinke, and L. Zakkynthinou. "PAC-
160 Bayes, MAC-Bayes and Conditional Mutual Information: Fast
161 rate bounds that handle general VC classes". In: COLT. June
162 2021.
- 163 [Gu+20] A. Gu, T. Dao, S. Ermon, A. Rudra, and C.
164 "HIPPO: Recurrent Memory with Optimal Polynomial Projec-
165 tions". In: NIPS 33 (2020).
- 166 [Gue19] B. Guedj. "A primer on PAC-Bayesian learning". In:
167 arXiv preprint arXiv:1901.05353 (2019).

B.

Re.

BIBLIOGRAPHY

- 1 [Gul+17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. "Improved training of wasserstein gans". In: NIPS. 2017, pp. 5767–5777.
- 2 [Gul+20] C. Gulcehre et al. RL Unplugged: Benchmarks for Offline Reinforcement Learning. arXiv:2006.13888. 2020.
- 3 [Gum54] E. J. Gumbel. Statistical theory of extreme values and some practical applications: A series of lectures (United States National Bureau of Standards. Applied mathematics series). en. 1st edition. U.S. Govt. Print. Offce, 1954.
- 4 [Guo09] Y. Guo. "Supervised exponential family principal component analysis via convex optimization". In: NIPS. 2009.
- 5 [Guo+17] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. "On Calibration of Modern Neural Networks". In: ICML. 2017.
- 6 [Gur+18] S. Gururangan, S. Swamyamdipta, O. Levy, R. Schwartz, S. R. Bowman, and N. A. Smith. "Annotation Artifacts in Natural Language Inference Data". In: CoRR abs/1803.02324 (2018). arXiv: 1803.02324.
- 7 [Gus01] M. Gustafsson. "A probabilistic derivation of the partial least-squares algorithm". In: Journal of Chemical Information and Modeling 41 (2001), pp. 288–294.
- 8 [Gut+14] M. U. Gutmann, R. Datta, S. Kaski, and J. Corander. "Statistical inference of intractable generative models via classification". In: arXiv preprint arXiv:1407.4981 (2014).
- 9 [GW08] A. Griewank and A. Walther. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Second. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.
- 10 [GW92] W. Gilks and P. Wild. "Adaptive rejection sampling for Gibbs sampling". In: Applied Statistics 41 (1992), pp. 337–348.
- 11 [GXG18] H. Ge, K. Xu, and Z. Ghahramani. "Turing: a language for exible probabilistic inference". In: AISTATS. 2018, pp. 1682–1690.
- 12 [GZG19] S. K. S. Ghasemipour, R. S. Zemel, and S. Gu. "A Divergence Minimization Perspective on Imitation Learning Methods". In: CORL. 2019, pp. 1259–1277.
- 13 [HA21] R. J. Hyndman and G. Athanasopoulos. Forecasting: Principles and Practice. en. 3rd ed. Otexts, May 2021.
- 14 [Haa+17] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. "Reinforcement learning with deep energy-based policies". In: Proceedings of the 34th International Conference on Machine Learning. 70. 2017, pp. 1352–1361.
- 15 [Haa+18a] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft Actor-Critic: α -Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: International Conference on Machine Learning. 2018, pp. 1861–1870.
- 16 [Haa+18b] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft Actor-Critic: α -Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: ICML. 2018.
- 17 [Haa+18c] T. Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: (Dec. 2018). arXiv: 1812.05905 [cs.LG].
- 18 [Had+20] R. Hadsell, D. Rao, A. A. Rusu, and R. Pascanu. "Embracing Change: Continual Learning in Deep Neural Networks". en. In: Trends Cogn. Sci. 24.12 (Dec. 2020), pp. 1028–1040.
- 19 [Haf18] D. Hafner. Building Variational Auto-Encoders in TensorFlow. Blog post. 2018.
- 20 [Haf+19] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. "Learning Latent Dynamics for Planning from Pixels". In: ICML. 2019.
- 21 [Haf+20] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. "Dream to Control: Learning Behaviors by Latent Imagination". In: ICLR. 2020.
- 22 [Hag+17] M. Hagen, M. Potthast, M. Goehsen, A. Rathgeber, and B. Stein. "A Large-Scale Query Spelling Correction Corpus". In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '17. Shinjuku, Tokyo, Japan: ACM, 2017, pp. 1261–1264.
- 23 [Haj88] B. Hajek. "Cooling Schedules for Optimal Annealing". In: Math. Oper. Res. 13.2 (1988), pp. 311–329.
- 24 [Hal76] R. Halin. "S-functions for graphs". en. In: J. Geom. 8.1–2 (Mar. 1976), pp. 171–186.
- 25 [Ham90] J. Hamilton. "Analysis of time series subject to changes in regime". In: J. Econometrics 45 (1990), pp. 39–70.
- 26 [Han+20] K. Han et al. "A Survey on Vision Transformer". In: (Dec. 2020). arXiv: 2012.12556 [cs.CV].
- 27 [Han80] T. S. Han. "Multiple mutual informations and multiple interactions in frequency data". In: Information and Control 46.1 (1980), pp. 26–45.
- 28 [Har+17] J. Hartford, G. Lewis, K. Leyton-Brown, and M. Taddy. "Deep IV: A exible approach for counterfactual prediction". In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR.org, 2017, pp. 1414–1423.
- 29 [Har+18] K. Hartnett. "To Build Truly Intelligent Machines, Teach Them Cause and Effect". In: Quanta Magazine (2018).
- 30 [Har90] A. C. Harvey. Forecasting, Structural Time Series Models, and the Kalman Filter. Cambridge University Press, 1990.
- 31 [Has10] H. van Hasselt. "Double Q-learning". In: NIPS. Ed. by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta. Curran Associates, Inc., 2010, pp. 2613–2621.
- 32 [Has70] W. Hastings. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: Biometrika 57.1 (1970), pp. 97–109.
- 33 [Has97] J. Haslett. "On the sample variogram and the sample autocovariance for non-stationary time series". In: J Royal Statistical Soc D 46.4 (Dec. 1997), pp. 475–484.
- 34 [Hau+10] J. R. Hauser, O. Toubia, T. Evgeniou, R. Befurt, and D. Dzubura. "Disjunctions of conjunctions, cognitive simplicity, and consideration sets". In: Journal of Marketing Research 47.3 (2010), pp. 485–496.
- 35 [Haw71] A. G. Hawkes. "Point spectra of some mutually exciting point processes". In: Journal of the Royal Statistical Society: Series B (Methodological) 33.3 (1971), pp. 438–443.
- 36 [Hoogendoorn+19] E. Hoogendoorn, R. van den Berg, and M. Welling. "Emerging Convolutions for Generative Normalizing Flows". In: ICML. 2019.
- 37 [Hoc93] G. Hinton and D. V. Camp. "Keeping Neural Networks Simple by Minimizing the Description Length of the Weights". In: in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory. ACM Press, 1993, pp. 5–13.
- 38 [HCG20] S. Huang, Y. Cao, and R. Grosse. "Evaluating Lossy Compression Rates of Deep Generative Models". In: ICML. 2020.
- 39 [Hed+19] D. Hendrycks and T. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: ICLR. 2019.
- 40 [HDL17] D. Ha, A. M. Dai, and Q. V. Le. "HyperNetworks". In: ICLR. 2017.
- 41 [He+16a] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: CVPR. 2016.
- 42 [HE16a] J. Ho and S. Ermon. "Generative adversarial imitation learning". In: Proceedings of the 30th International Conference on Neural Information Processing Systems. 2016, pp. 4572–4580.
- 43 [He+16b] K. He, X. Zhang, S. Ren, and J. Sun. "Identity Mappings in Deep Residual Networks". In: ECCV. 2016.
- 44 [HE16b] J. Ho and S. Ermon. "Generative Adversarial Imitation Learning". In: NIPS. 2016, pp. 4565–4573.
- 45 [HE18] D. Ha and D. Eck. "A Neural Representation of Sketch Drawings". In: ICLR. 2018.
- 46 [He+19] J. He, D. Spokony, G. Neubig, and K. Kirpatrick. "Lagging Inference Networks and Posterior Collapse in Variational Autoencoders". In: ICLR. Jan. 2019.
- 47 [Hei13] M. Heinz. Tree-Decomposition Graph Minor Theory and Algorithmic Implications. PhD thesis. U. Wien, 2013.
- 48 [Hel17] J. Heltske. "KFAS: Exponential Family State Space Models in R". In: J. Stat. Softw. (2017).
- 49 [Hen+15] J. Hensman, A. Matthews, M. Filippone, and Z. Ghahramani. "MCMC for Variationally Sparse Gaussian Processes". In: NIPS. 2015, pp. 1648–1656.
- 50 [Hen+16] L. A. Hendrycks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T. Darrell. "Generating visual explanations". In: European conference on computer vision. Springer, 2016, pp. 3–19.
- 51 [Hen+18] G. E. Henter, J. Lorenzo-Trueba, X. Wang, and J. Yamagishi. "Deep Encoder-Decoder Models for Unsupervised Learning of Controllable Speech Synthesis". In: (July 2018). arXiv: 1807.11470 [eess.AS].
- 52 [Hen+19a] O. J. Hena, A. Razavi, C. Doersch, S. M. Ali Eslam, and A. van den Oord. "Data-E cient Image Recognition with Contrastive Predictive Coding". In: arXiv [cs.CV] (May 2019).
- 53 [Hen+19b] D. Hendrycks, S. Basart, M. Mazeika, A. Zou, J. Kwon, M. Mostajabi, J. Steinhardt, and D. Song. "Scaling Out-of-Distribution Detection for Real-World Settings". In: (Nov. 2019). arXiv: 1911.11132 [cs.CV].
- 54 [Hen+20] D. Hendrycks*, N. Mu*, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan. "AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty". In: ICLR. 2020.
- 55 [Hen+21] C. Henning, M. R. Cervera, F. D'Angelo, J. von Oswald, R. Traber, B. Ehret, S. Kobayashi, B. F. Grewe, and J. Sacramento. "Posterior Meta-Replay for Continual Learning". In: NIPS. Mar. 2021.
- 56 [Hes00] T. Heskes. "On "Natural" Learning and Pruning in Multilayered Perceptrons". In: Neural Comput. 12.4 (Apr. 2000), pp. 881–901.

- 1
- 2 [Hes+18] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G.
Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D.
Silver. "Rainbow: Combining Improvements in Deep Reinforce-
ment Learning". In: AAAI. 2018.
- 3 [Heu+17a] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler,
and S. Hochreiter. "GANs Trained by a Two Time-Scale Up-
date Rule Converge to a Local Nash Equilibrium". In: NIPS.
2017.
- 4 [Heu+17b] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler,
and S. Hochreiter. "Gans trained by a two time-scale update
rule converge to a local nash equilibrium". In: Advances in neural
information processing systems. 2017, pp. 6626–6637.
- 5 [HF09] R. Hess and A. Fern. "Discriminatively Trained Particle
Filters for Complex Multi-Object Tracking". In: CVPR. 2009.
- 6 [HFL13] J. Hensman, N. Fusi, and N. D. Lawrence. "Gaussian
Processes for Big Data". In: UAI. 2013.
- 7 [HFM17] D. W. Hogg and D. Foreman-Mackey. "Data analysis
recipes: Using Markov Chain Monte Carlo". In: (Oct. 2017).
arXiv: 1710.06068 [astro-ph.IM].
- 8 [HG12] D. I. Hastie and P. J. Green. "Model Choice using Re-
versible Jump Markov Chain Monte Carlo". In: Statistica Neer-
landica 66 (2012), pp. 309–338.
- 9 [HG14] M. D. Ho man and A. Gelman. "The No-U-Turn Sam-
pler: Adaptively Setting Path Lengths in Hamiltonian Monte
Carlo". In: JMLR 15 (2014), pp. 1593–1623.
- 10 [HG16] D. Hendrycks and K. Gimpel. "Gaussian Error Linear
Units (GELUs)". In: arXiv [cs.LG] (June 2016).
- 11 [HGMG18] J. Hron, A. G. de G. Matthews, and Z. Ghahramani.
"Variational Bayesian dropout: pitfalls and xes". In: ICML.
2018.
- 12 [HGS16] H. van Hasselt, A. Guez, and D. Silver. "Deep Re-
inforcement Learning with Double Q-Learning". In: AAAI.
AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.
- 13 [HH06] C. Holmes and L. Held. "Bayesian auxiliary variable
models for binary and multinomial regression". In: Bayesian
Analysis 1.1 (2006), pp. 145–168.
- 14 [HHC12] J. Hu, P. Hu, and H. S. Chang. "A Stochastic Approx-
imation Framework for a Class of Randomized Optimization
Algorithms". In: IEEE Trans. Automatic Control 57.1 (2012).
- 15 [HHH09] A. Hyvärinen, J. Hurri, and P. Hoyer. Natural Image
Statistics: a probabilistic approach to early computational vision.
Springer, 2009.
- 16 [HHK19] M. Häufmann, F. A. Hamprecht, and M. Kandemir.
"Sampling-Free Variational Inference of Bayesian Neural Net-
works by Variance Backpropagation". In: UAI. 2019.
- 17 [HHLB11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Se-
quential Model-Based Optimization for General Algorithm Con-
figuration". In: Int. Conf. on Learning and Intelligent Opti-
mization (LION). 2011, pp. 507–523.
- 18 [HHLMF18] M. Havasi, J. M. Hernández-Lobato, and J. J.
Murillo-Fuentes. "Inference in Deep Gaussian Processes using
Stochastic Gradient Hamiltonian Monte Carlo". In: (June 2018).
arXiv: 1806.05490 [stat.ML].
- 19 [Hig+17] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glor-
rot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-VAE:
Learning Basic Visual Concepts with a Constrained Variational
Framework". In: ICLR. 2017.
- 20 [Hin02] G. E. Hinton. "Training products of experts by minimiz-
ing contrastive divergence". In: Neural Computation 14.8
(Aug. 2002), pp. 1771–1800.
- 21 [Hin10] G. Hinton. A Practical Guide to Training Restricted
Boltzmann Machines. Tech. rep. U. Toronto, 2010.
- 22 [Hin14] G. Hinton. Lecture 6e on neural networks (RMSProp:
Divide the gradient by a running average of its recent magni-
tude). 2014.
- 23 [Hin+95] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal.
"The "wake-sleep" algorithm for unsupervised neural networks".
In: Science 268.5214 (May 1995), pp. 1158–1161.
- 24 [HYI19] K. Hayashi, M. Imaizumi, and Y. Yoshida. "On Ran-
dom Subsampling of Gaussian Process Regression: A Graphon-
Based Analysis". In: (Jan. 2019). arXiv: 1901.09541 [stat.ML].
- 25 [HJ20] J. Huang and N. Jiang. "From Importance Sampling to
Doubly Robust Policy Gradient". In: ICML. 2020.
- 26 [HJA20] J. Ho, A. Jain, and P. Abbeel. "Denoising Di fusion
Probabilistic Models". In: NIPS. 2020.
- 27 [HJT18] M. D. Ho man, M. J. Johnson, and D. Tran. "Autoconj:
Recognizing and Exploiting Conjugacy Without a Domain-
Specific Language". In: NIPS. 2018.
- 28 [HKZ12] D. Hsu, S. Kakade, and T. Zhang. "A spectral algo-
rithm for learning hidden Markov models". In: J. of Computer
and System Sciences 78.5 (2012), pp. 1460–1480.
- 29 [HL04] D. R. Hunter and K. Lange. "A Tutorial on MM Algo-
rithms". In: The American Statistician 58 (2004), pp. 30–37.
- 30 [HL16a] J. Hernandez-Lobato, Y. Li, M. Rowland, T. Bui, D.
Hernandez-Lobato, and R. Turner. "Black-Box Alpha Diver-
gence Minimization". In: ICML. June 2016, pp. 1511–1520.
- 31 [HL16b] M. Hernandez-Lobato, M. A. Gelbart, R. P. Adams,
M. W. Ho man, and Z. Ghahramani. "A General Framework for
Constrained Bayesian Optimization using Information-based
Search". In: JMLR (2016).
- 32 [HL20] X. Hu and J. Lei. "A Distribution-Free Test of Covari-
ate Shift Using Conformal Prediction". In: (Oct. 2020). arXiv:
2010.07147 [stat.ME].
- 33 [HLA15a] J. M. Hernández-Lobato and R. P. Adams. "Prob-
abilistic Backpropagation for Scalable Learning of Bayesian Neu-
ral Networks". In: ICML. Feb. 2015.
- 34 [HLA15b] J. M. Hernández-Lobato and R. P. Adams. "Prob-
abilistic Backpropagation for Scalable Learning of Bayesian Neu-
ral Networks". In: ICML. Feb. 2015.
- 35 [HLA19] J. Ho, E. Lohn, and P. Abbeel. "Compression with
Flows via Local Bits-Back Coding". In: NeurIPS. 2019.
- 36 [HLC19] Y.-P. Hsieh, C. Liu, and V. Cevher. "Finding mixed
nash equilibria of generative adversarial networks". In: Interna-
tional Conference on Machine Learning. 2019, pp. 2810–2819.
- 37 [HLHG14] J. Hernandez-Lobato, M. W. Ho man, and Z.
Ghahramani. "Predictive entropy search for efficient global op-
timization of black-box functions". In: NIPS. 2014.
- 38 [HLHD13] D. Hernández-Lobato, J. M. Hernández-Lobato,
and P. Dupont. "Generalized Spike-and-Slab Priors for
Bayesian Group Feature Selection Using Expectation Propaga-
tion". In: JMLR 14 (2013), pp. 1891–1945.
- 39 [HLR16] K. Hofmann, L. Li, and F. Radlinski. "Online Evalua-
tion for Information Retrieval". In: Foundations and Trends in
Information Retrieval 10.1 (2016), pp. 1–117.
- 40 [HLW14] J. R. Hershey, J. Le Roux, and F. Weninger. "Deep
Unfolding: Model-Based Inspiration of Novel Deep Architec-
tures". In: (2014). arXiv: 1409.2574 [cs.LG].
- 41 [HLS03] R. Herbrich, N. D. Lawrence, and M. Seeger. "Fast
Sparse Gaussian Process Methods: The Informative Vector Ma-
chine". In: NIPS. MIT Press, 2003, pp. 625–632.
- 42 [HM20] M. Hosseini and A. Maida. "Hierarchical Predictive Cod-
ing Models in a Deep-Learning Framework". In: (May 2020).
arXiv: 2005.03230 [cs.CV].
- 43 [HM81] R. Howard and J. Matheson. "Influence diagrams". In:
Readings on the Principles and Applications of Decision Analy-
sis, volume II. Ed. by R. Howard and J. Matheson. Strategic
Decisions Group, 1981.
- 44 [HMD18] J. C. Higuera, D. Meger, and G. Dudek. "Synthesizing
Neural Network Controllers with Probabilistic Model-Based Re-
inforcement Learning". In: IROS. Oct. 2018, pp. 2538–2544.
- 45 [HMD19] D. Hendrycks, M. Mazeika, and T. Dietterich. "Deep
Anomaly Detection with Outlier Exposure". In: ICLR. 2019.
- 46 [HMG07] R. Herbrich, T. Minka, and T. Graepel. "TrueSkill: A
Bayesian skill rating system". In: NIPS. 2007.
- 47 [HMK04] D. Heckerman, C. Meek, and D. Koller. Probabilistic
Models for Relational Data. Tech. rep. MSR-TR-2004-30. Mi-
crosoft Research, 2004.
- 48 [HNK18] J. He, G. Neubig, and T. Berg-Kirkpatrick. "Unsu-
pervised Learning of Syntactic Structure with Invertible Neural
Projections". In: EMNLP. 2018.
- 49 [HN09] A. Halevy, P. Norvig, and F. Pereira. "The unreason-
able effectiveness of data". In: IEEE Intelligent Systems 24.2
(2009), pp. 8–12.
- 50 [HO00] A. Hyvärinen and E. Oja. "Independent component analy-
sis: algorithms and applications". In: Neural Networks 13
(2000), pp. 411–430.
- 51 [HO48] C. G. Hempel and P. Oppenheim. "Studies in the Logic
of Explanation". In: Philosophy of science 15.2 (1948), pp. 135–
175.
- 52 [Hob69] A. Hobson. "A new theorem of information theory". In:
Journal of Statistical Physics 1.3 (1969), pp. 383–391.
- 53 [Hoe12] J. M. ver Hoef. "Who invented the delta method?" In:
The American Statistician 66.2 (2012), pp. 124–127.
- 54 [Hoe+21] T. Hoe er, D. Alistarh, T. Ben-Nun, N. Dryden, and
A. Peste. "Sparsity in Deep Learning: Pruning and growth for
efficient inference and training in neural networks". In: (Jan.
2021). arXiv: 2102.00554 [cs.LG].
- 55 [Hof09] P. D. Ho. A First Course in Bayesian Statistical Meth-
ods. Springer, 2009.
- 56 [Hof13] M. D. Ho man, D. M. Blei, C. Wang, and J. Pais-
ley. "Stochastic Variational Inference". In: JMLR 14 (2013),
pp. 1303–1347.
- 57 [Hof17] M. D. Ho man. "Learning Deep Latent Gaussian Models
with Markov Chain Monte Carlo". In: ICML. 2017, pp. 1510–
1519.
- 58 [Hof+18] J. Ho man, E. Tseng, T. Park, J.-Y. Zhu, P. Isola, K.
Saenko, A. Efros, and T. Darrell. "Cycada: Cycle-consistent ad-

BIBLIOGRAPHY

- 1 versarial domain adaptation". In: International conference on
2 machine learning. PMLR, 2018, pp. 1989–1998.
- 3 [Hof+19] M. Ho man, P. Sountsov, J. V. Dillon, J. Langmore,
4 D. Tran, and S. Vasudevan, "NeuTra-lizing Bad Geometry in
 Hamiltonian Monte Carlo Using Neural Transport". In: (Mar.
5 2019). arXiv: 1903.03704 [stat.CO].
- 6 [Hol21] P. Ho., "Bayes-optimal prediction with frequentist cov-
 erage control". In: (May 2021). arXiv: 2105.14045 [math.ST].
- 7 [Hoh+20] F. Hohman, M. Conlen, J. Heer, and D. H.
8 Chau, "Communicating with interactive articles". In: Distill 5.9
 (2020), e28.
- 9 [Hol86] P. W. Holland, "Statistics and Causal Inference". In:
10 JASA 81.396 (1986), pp. 945–960.
- 11 [Hon+10] A. Honkela, T. Raiko, M. Kuusela, M. Tornio, and
12 J. Karhunen, "Approximate Riemannian Conjugate Gradient
 Learning for Fixed-Form Variational Bayes". In: JMLR 11.Nov
 (2010), pp. 3235–3268.
- 13 [Hor+05] E. Horwitz, J. Apacible, R. Sarin, and L. Liao, "Prediction,
14 Expectation, and Surprise: Methods, Designs, and Study of a Deployed Tra c Forecasting Service". In: UAI, 2005.
- 15 [Hor61] P. Horst, "Generalized canonical correlations and their
16 applications to experimental data". en. In: J. Clin. Psychol. 17
 (Oct. 1961), pp. 331–347.
- 17 [Hos+20] T. Hospedales, A. Antoniou, P. Micaelli, and A.
18 Storkey, "Meta-Learning in Neural Networks: A Survey". In:
 (Apr. 2020). arXiv: 2004.05439 [cs.LG].
- 19 [HOT06] G. Hinton, S. Osindero, and Y. Teh, "A fast learning
20 algorithm for deep belief nets". In: Neural Computation 18 (2006), pp. 1527–1554.
- 21 [Hot36] H. Hotelling, "Relations Between Two Sets of Variates".
22 In: Biometrika 28.3/4 (1936), pp. 321–377.
- 23 [HOW11] P. Hall, J. T. Ormerod, and M. P. Wand, "Theory of
24 Gaussian Variational Approximation for a Generalised Linear
 Mixed Model". In: Statistica Sinica 21 (2011), pp. 269–389.
- 25 [HP10] J. Hacker and P. Pierson, Winner-Take-All Politics:
26 How Washington Made the Rich Richer — and Turned Its Back on the Middle Class. Simon & Schuster, 2010.
- 27 [HR20a] M. Hernan and J. Robins, Causal Inference: What If.
28 CRC Press, 2020.
- 29 [HR20b] M. Hernan and J. Robins, Causal Inference: What If.
30 Boca Raton: Chapman & Hall/CRC, 2020.
- 31 [HS06] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks". In: Science 313.5786
 (2006), pp. 504–507.
- 32 [HS09] M. Heaton and J. Scott, Bayesian computation and the
33 linear model. Tech. rep. Duke, 2009.
- 34 [HS12] P. Hennig and C. Schuler, "Entropy search
35 information-e cient global optimization". In: JMLR 13 (2012),
 pp. 1809–1837.
- 36 [HS13] J. Y. Hsu and D. S. Small, "Calibrating Sensitivity Anal-
37 yses to Observed Covariates in Observational Studies". In: Bio-
38 metrics 69.4 (2013), pp. 803–811. <https://onlinelibrary.wiley.com/doi/pdf/10.1111/biom.12101>.
- 39 [HS18] D. Ha and J. Schmidhuber, "World Models". In: NIPS.
40 2018.
- 41 [HS88] D. S. Hochbaum and D. B. Shmoys, "A polynomial ap-
42 proximation scheme for scheduling on uniform processors: Us-
 ing the dual approximation approach". In: SICOMP, 1988.
- 43 [HS97] S. Hochreiter and J. Schmidhuber, "Flat minima". en. In:
 Neural Comput. 9.1 (1997), pp. 1–42.
- 44 [HSG06] J. D. Hol, T. B. Schon, and F. Gustafsson, "On Resam-
45 pling Algorithms for Particle Filters". In: 2006 IEEE Nonlinear
 Statistical Signal Processing Workshop. Sept. 2006, pp. 79–82.
- 46 [HSCF21] S. Hassan, S. Sárká, and Á. F. García-Fernández,
47 "Temporal Parallelization of Inference in Hidden Markov Models". In: IEEE Trans. Signal Processing 69 (Feb. 2021),
 pp. 4875–4887.
- 48 [Hsu+18] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira, "Re-
49 evaluating Continual Learning Scenarios: A Categorization and
 Case for Strong Baselines". In: NIPS Continual Learning Work-
 shop. Oct. 2018.
- 50 [HT01] G. E. Hinton and Y. Teh, "Discovering multiple con-
51 straints that are frequently approximately satis ed". In: UAI.
52 2001.
- 53 [HT09] H. Hoe ing and R. Tibshirani, "Estimation of Sparse Bi-
54 nary Pairwise Markov Networks using Pseudo-likelihood". In:
 JMLR 10 (2009).
- 55 [HT15] J. H. Huggins and J. B. Tenenbaum, "Risk and regret of
56 hierarchical Bayesian learners". In: ICML. May 2015.
- 57 [HT17] T. J. Hastie and R. J. Tibshirani, Generalized additive
 models. Routledge, 2017.
- 58 [HTF09] T. Hastie, R. Tibshirani, and J. Friedman, The Ele-
59 ments of Statistical Learning, 2nd edition. Springer, 2009.
- 60 [HTW15] T. Hastie, R. Tibshirani, and M. Wainwright, Statis-
61 tical Learning with Sparsity: The Lasso and Generalizations.
 CRC Press, 2015.
- 62 [Hu+00] M. Hu, C. Ingram, M. Sirski, C. Pal, S. Swamy, and C.
63 Patten, A Hierarchical HMM Implementation for Vertebrate
 Gene Splice Site Prediction. Tech. rep. Dept. Computer Sci-
 ence, Univ. Waterloo, 2000.
- 64 [Hu+12] J. Hu, Y. Wang, E. Zhou, M. C. Fu, and S. I. Mar-
65 cus, "A Survey of Some Model-Based Methods for Global Opti-
 mization", en. In: Optimization, Control, and Applications of
 Stochastic Systems. Systems & Control: Foundations & Appli-
 cations. Birkhäuser, Boston, 2012, pp. 157–179.
- 66 [Hu+17] W. Hu, C. Li, L. Li, and J.-G. Liu, "On the di u-
67 sion approximation of nonconvex stochastic gradient descent".
 In: (May 2017). arXiv: 1705.07562 [stat.ML].
- 68 [Hu+18] W. Hu, G. Niu, I. Sato, and M. Sugiyama, "Does Dis-
69 tributionally Robust Supervised Learning Give Robust Classi-
 fiers". In: ICML 2018.
- 70 [Hua+17a] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. Hopcroft, and
71 K. Weinberger, "Snapshot ensembles: train 1, get M for free".
 In: ICLR, 2017.
- 72 [Hua+17b] Y. Huang, Y. Zhang, N. Li, Z. Wu, and J. A. Chambers,
73 "A Novel Robust Student's t-Based Kalman Filter". In: IEEE
 Trans. Aerosp. Electron. Syst. 53.3 (June 2017), pp. 1545–1554.
- 74 [Hua+18a] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N.
75 Shazeer, C. Simon, C. Hawthorne, A. M. Dai, M. D. Ho man, M.
 Dinculescu, and D. Eck, "Music Transformer". In: (Sept. 2018).
 arXiv: 1809.04281 [cs.LG].
- 76 [Hua+18b] C.-W. Huang, D. Krueger, A. Lacoste, and A.
77 Courville, "Neural Autoregressive Flows". In: ICML, 2018.
- 78 [Hua+19a] W. R. Huang, Z. Erman, M. Goldblum, L. Fowl,
79 J. K. Terry, F. Huang, and T. Goldstein, "Understand-
 ing generalization through visualizations". In: arXiv preprint
 arXiv:1906.03291 (2019).
- 80 [Hua+19b] Y. Huang, Y. Zhang, Y. Zhao, and J. A. Chambers,
81 "A Novel Robust Gaussian-Student's t Mixture Distribution
 Based Kalman Filter". In: IEEE Trans. Signal Process. 67.13
 (July 2019), pp. 3606–3620.
- 82 [Hug+19] J. H. Huggins, T. Campbell, M. Kasprzak, and T.
83 Broderick, "Scalable Gaussian Process Inference with Finite-
84 data Mean and Variance Guarantees". In: AISTATS. 2019.
- 85 [Hug+20] J. H. Huggins, T. Campbell, M. Kasprzak, and T.
86 Broderick, "Validated Variational Inference via Practical Posterior
 Error Bounds". In: AISTATS. Ed. by S. Chiappa and R. Ca-
 landra, Vol. 108. Proceedings of Machine Learning Research.
 PMLR, 2020, pp. 1792–1802.
- 87 [Hus17a] F. Huszár, Is Maximum Likelihood Useful for Repre-
88 sentation Learning? 2017.
- 89 [Hus17b] F. Huszár, "Variational inference using implicit distri-
90 butions". In: arXiv preprint arXiv:1702.08235 (2017).
- 91 [Hut89] M. F. Hutchinson, "A stochastic estimator of the trace
92 of the in uence matrix for Laplacian smoothing splines". In:
 Communications in Statistics-Simulation and Computation
 18.3 (1989), pp. 1059–1076.
- 93 [Hvi21] J. Helske and M. Vi hola, "bsbm: Bayesian Inference of
94 Non-linear and Non-Gaussian State Space Models in R". In:
 (Jan. 2021). arXiv: 2101.08492 [stat.CO].
- 95 [HVD14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the
96 Knowledge in a Neural Network". In: NIPS DL workshop. 2014.
- 97 [HW13] A. Huang and M. P. Wand, "Simple Marginally Nonin-
98 formative Prior Distributions for Covariance Matrices". en. In:
 Bayesian Analysis 8.2 (June 2013), pp. 439–452.
- 99 [HWI17] H. He, B. Xin, and D. Wipf, "From Bayesian Sparsity
100 to Gated Recurrent Nets". In: NIPS. 2017.
- 101 [HY01] M. Hansen and B. Yu, "Model selection and the prin-
102 ciple of minimum description length". In: JASA (2001).
- 103 [Hyv05] A. Hyvärinen, "Estimation of non-normalized statis-
104 tical models by score matching". In: JMLR 6.Apr (2005),
 pp. 695–709.
- 105 [Hyv07a] A. Hyvärinen, "Connections between score matching,
106 contrastive divergence, and pseudolikelihood for continuous-
 valued variables". In: IEEE Transactions on neural networks
 18.5 (2007), pp. 1529–1531.
- 107 [Hyv07b] A. Hyvärinen, "Some extensions of score matching". In:
 Computational statistics & data analysis 51.5 (2007), pp. 2499–
 2512.
- 108 [IB12] R. Iyer and J. Bilmes, "Algorithms for Approximate Min-
109 imization of the Errors Between Submodular Functions,
 with Application". In: Uncertainty in Artifical Intelligence
 (UAI), Catalina Island, USA: AUAI, 2012.
- 110 [IB13] R. Iyer and J. Bilmes, "Submodular Optimization with
 Submodular Cover and Submodular Knapsack Constraints". In:

- 1 Neural Information Processing Society (NeurIPS, formerly
2 NIPS), Lake Tahoe, CA, 2013.
- 3 [IB15] R. K. Iyer and J. A. Bilmes. "Polyhedral aspects of
4 Submodularity, Convexity and Concavity". In: Arxiv, CoRR
abs/1506.07329 (2015).
- 5 [IB98] M. Isard and A. Blake. "CONDENSATION - conditional
density propagation for visual tracking". In: Intl. J. of Com-
puter Vision 29.1 (1998), pp. 5–18.
- 6 [IBK06] E. L. Ionides, C. Bretó, and A. A. King. "Inference
for nonlinear dynamical systems". en. In: PNAS 103.49 (Dec.
2006), pp. 18438–18443.
- 7 [IFF00] S. Iwata, L. Fleischer, and S. Fujishige. "A combina-
tional strongly polynomial algorithm for minimizing submodular
functions". In: Journal of the ACM (2000).
- 8 [IFW05] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. "Loopy
Belief Propagation: Convergence and Effects of Message Er-
rors". In: JMLR 6 (Dec. 2005), pp. 905–936.
- 9 [IJ01] H. Ishwaran and L. F. James. "Gibbs sampling methods
for stick-breaking priors". In: Journal of the American Statis-
tical Association 96.453 (2001), pp. 161–173.
- 10 [IJB13] R. Iyer, S. Jegelka, and J. Bilmes. "Curvature and Opti-
mal Algorithms for Learning and Minimizing Submodular Func-
tions". In: NIPS, Lake Tahoe, CA, 2013.
- 11 [IKB21] A. Immer, M. Korzepa, and M. Bauer. "Improving pre-
dictions of Bayesian neural nets via local linearization". In:
AISTATS, Ed. by A. Banerjee and K. Fukumizu, Vol. 130. Pro-
ceedings of Machine Learning Research, PMLR, 2021, pp. 703–
711.
- 12 [IM17] J. Ingraham and D. Marks. "Bayesian Sparsity for In-
tractable Undirected Models". In: ICML 2017.
- 13 [Imb03] G. Imbens. "Sensitivity to Exogeneity Assumptions in
Program Evaluation". In: The American Economic Review
19 (2003).
- 14 [Imb19] G. W. Imbens. "Potential Outcome and Directed
Acyclic Graph Approaches to Causality: Relevance for Empirical
Practice in Economics". In: (July 2019). arXiv:
[stat.ME].
- 15 [IN09] S. Iwata and K. Nagano. "Submodular function minimiza-
tion under covering constraints". In: Proceedings of the 50th
Annual IEEE Symposium on Foundations of Computer Sci-
ence (FOCS), 2009, pp. 671–680.
- 16 [INK18] P. Izmailov, A. Novikov, and D. Kropotov. "Scalable
Gaussian Processes with Billions of Inducing Inputs via Tensor
Train Decomposition". In: ICML 2018.
- 17 [Inn20] M. Innes. "Sense & Sensitivities: The Path to
General-Purpose Algorithmic Differentiation". In: Proceedings
of Machine Learning and Systems, Ed. by I. Dhillon, D. Pa-
palopoulos, and V. Sze, Vol. 2, 2020, pp. 58–69.
- 18 [IO09] S. Iwata and J. B. Orlin. "A simple combinatorial algo-
rithm for submodular function minimization". In: Proceedings
of the twentieth annual ACM-SIAM symposium on Discrete
algorithms. SIAM, 2009, pp. 1230–1237.
- 19 [IR00] D. R. Insua and F. Ruggeri. Robust Bayesian Analysis.
Springer, 2000.
- 20 [IR10] A. Ilin and T. Raiko. "Practical Approaches to Princi-
pal Component Analysis in the Presence of Missing Values".
In: JMLR 11 (2010), pp. 1957–2000.
- 21 [IR15] G. Imbens and D. Rubin. Causal Inference in Statistics,
Social and Biomedical Sciences: An Introduction. Cambridge
University Press, 2015.
- 22 [IS15] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerat-
ing Deep Network Training by Reducing Internal Covariate
Shift". In: ICML 2015, pp. 448–456.
- 23 [Iso03] M. Isard. "PAMPAS: Real-Valued Graphical Models for
Computer Vision". In: CVPR, Vol. 1, 2003, p. 613.
- 24 [Isl+19] R. Islam, R. Seraji, S. Y. Arnob, and D. Precup. "Doubly
Robust Q-Policy Actor-Critic Algorithm for Reinforce-
ment Learning". In: NeurIPS Workshop on Safety and Robust-
ness in Decision Making, 2019.
- 25 [Iso+17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-
to-Image Translation with Conditional Adversarial Networks".
In: CVPR, 2017.
- 26 [IX00] K. Ito and K. Xiong. "Gaussian Iters for nonlinear I-
terating problems". In: IEEE Trans. Automat. Contr. 45.5 (May
2000), pp. 910–927.
- 27 [Iye+21] R. Iyer, N. Khargonkar, J. Bilmes, and H. Asnani.
"Generalized Submodular Information Measures: Theoretical
Properties, Examples, Optimization Algorithms, and Applica-
tions". In: IEEE Transactions on Information Theory 2021.
- 28 [Izm+18] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov,
and A. G. Wilson. "Averaging Weights Leads to Wider Optima
and Better Generalization". In: UAI, 2018.
- 29 [Izm+19] P. Izmailov, W. J. Maddox, P. Kirichenko, T. Garipov,
D. Vetrov, and A. G. Wilson. "Subspace Inference for Bayesian
Deep Learning". In: UAI, 2019.
- 30 [Izm+21a] P. Izmailov, P. Nicholson, S. Lot, and A. G. Wilson.
"Danger of Bayesian Model Averaging under Covariate Shift".
In: NIPS, 2021.
- 31 [Izm+21b] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G.
Wilson. "What Are Bayesian Neural Network Posteriors Really
Like?". In: ICML, 2021.
- 32 [Jaa01] T. Jaakkola. "Tutorial on variational approximation
methods". In: Advanced mean field methods. Ed. by M. Opper
and D. Saad. MIT Press, 2001.
- 33 [Jac+21] M. Jacobs, M. F. Pradier, T. H. McCoy, R. H. Perlis,
F. Doshi-Velez, and K. Z. Gajos. "How machine-learning rec-
ommendations influence clinician treatment selections: the ex-
ample of antidepressant selection". In: Translational psychiatry
11.1 (2021), pp. 1–9.
- 34 [Jad+17] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul,
J. Z. Leibo, D. Silver, and K. Kavukcuoglu. "Reinforcement
Learning with Unsupervised Auxiliary Tasks". In: ICLR, 2017.
- 35 [Jak21] K. Jakkala. "Deep Gaussian Processes: A Survey". In:
(June 2021). arXiv: 2106.12135 [cs.LG].
- 36 [Jan+17] P. A. Jang, A. Loeb, M. Davidow, and A. G. Wilson.
"Scalable Levy Process Priors for Spectral Kernel Learning". In:
Advances in Neural Information Processing Systems, 2017.
- 37 [Jan18] E. Jang. Normalizing Flows Tutorial, 2018.
- 38 [Jan+19] M. Janner, J. Fu, M. Zhang, and S. Levine. "When
to Trust Your Model: Model-Based Policy Optimization". In:
NIPS, 2019.
- 39 [Jas] Jason Antic and Jeremy Howard and Uri Manor. Decap-
pitation, DeOldification, and Super Resolution (Blog post).
- 40 [Jay03] E. T. Jaynes. Probability theory: the logic of science.
Cambridge University press, 2003.
- 41 [Jay+20] S. M. Jayakumar, W. M. Czarnecki, J. Menick, J.
Schwarz, J. Rae, S. Osindero, Y. W. Teh, T. Harley, and R. Pas-
calou. "Multiplicative Interactions and Where to Find Them".
In: ICLR, 2020.
- 42 [JB03] A. Jakulin and I. Bratko. "Analyzing Attribute Depen-
dencies". In: Proc. 7th European Conf. on Principles and Practice
of Knowledge Discovery in Databases, 2003.
- 43 [JB16a] S. Jegelka and J. Bilmes. "Graph cuts with interacting
edge weights: examples, approximations, and algorithms". In:
Mathematical Programming 2016, pp. 1–42.
- 44 [JB16b] R. Jonschkowski and O. Brock. "End-to-end learnable
histogram Iters". In: NIPS Workshop on Deep Learning for
Action and Interaction, 2016.
- 45 [JB65] C. Jacobi and C. W. Borchardt. "De investigando or-
dine systematico aequationum di erentialem vulgarium cuius-
cunque." In: Journal für die reine und angewandte Mathematik
1865.64 (1865), pp. 297–320.
- 46 [Jef04] R. Jeffrey. Subjective Probability: The Real Thing. Cam-
bridge, 2004.
- 47 [Jen+17] R. Jenatton, C. Archambeau, J. González, and M.
Seeger. "Bayesian Optimization with Tree-structured Dependencies".
en. In: ICML, July 2017, pp. 1655–1664.
- 48 [JG20] A. Jacovi and Y. Goldberg. "Towards faithfully inter-
pretable NLP systems: How should we de ne and evaluate faith-
fulness?". In: arXiv preprint arXiv:2004.03685 (2020).
- 49 [JG21] A. Jacovi and Y. Goldberg. "Aligning faithful interpre-
tations with their social attribution". In: Transactions of the
Association for Computational Linguistics 9 (2021), pp. 294–
310.
- 50 [JGH18] A. Jacob, F. Gabriel, and C. Hongler. "Neural Tangent
Kernel: Convergence and Generalization in Neural Networks".
In: NIPS, 2018.
- 51 [JGP17] E. Jang, S. Gu, and B. Poole. "Categorical Reparameter-
ization with Gumbel-Softmax". In: ICLR, 2017.
- 52 [Jia+19] R. Jia, A. Raghunathan, K. Göksel, and P. Liang.
"Certified Robustness to Adversarial Word Substitutions". In:
EMNLP, 2019.
- 53 [Jia21] H. Jiang. "Minimizing convex functions with integral
minimizers". In: Proceedings of the 2021 ACM-SIAM Sympo-
sium on Discrete Algorithms (SODA). SIAM, 2021, pp. 976–
985.
- 54 [Jih+12] Jihong Min, J. Kim, Seunghak Shin, and I. S. Kweon.
"Efficient Data-Driven MCMC sampling for vision-based 6D
SLAM". In: ICRA, May 2012, pp. 3025–3032.
- 55 [Jit+16] W. Jitkrittum, Z. Szabó, K. P. Chwialkowski, and
A. Gretton. "Interpretable Distribution Features with Maxi-
mum Testing Power". In: NIPS. Curran Associates, Inc., 2016,
pp. 181–189.
- 56 [JJ00] T. S. Jaakkola and M. I. Jordan. "Bayesian parameter
estimation via variational methods". In: Statistics and Com-
puting 10 (2000), pp. 25–37.
- 57 [JJ94] F. V. Jensen and F. Jensen. "Optimal Junction Trees".
In: UAI, 1994.
- 58 [JKJ12] K. Jiang, B. Kulis, and M. Jordan. "Small-variance
asymptotics for exponential family Dirichlet process mixture

1907.07271

BIBLIOGRAPHY

- 1 models". In: Advances in Neural Information Processing Systems 25 (2012), pp. 3158–3166.
- 2 [JKK95] C. S. Jensen, A. Kong, and U. Kjaerul, "Blocking-Gibbs Sampling in Very Large Probabilistic Expert Systems". In: Int'l. J. Human-Computer Studies (1995), pp. 647–666.
- 3 [JL15a] V. Jalali and D. Leake, "CBR meets big data: A case study of large-scale adaptation rule generation". In: International Conference on Case-Based Reasoning. Springer, 2015, pp. 181–196.
- 4 [JL15b] V. Jalali and D. B. Leake, "Enhancing case-based regression with automatically-generated ensembles of adaptations". In: Journal of Intelligent Information Systems 46 (2015), pp. 237–258.
- 5 [JL16] N. Jiang and L. Li, "Doubly Robust Q-policy Evaluation for Reinforcement Learning". In: ICML, 2016, pp. 652–661.
- 6 [JM00] D. Jurafsky and J. H. Martin, Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice-Hall, 2000.
- 7 [JM08] D. Jurafsky and J. H. Martin, Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd edition. Prentice-Hall, 2008.
- 8 [JM18] A. Jolicoeur-Martineau, "The relativistic discriminator: a key element missing from standard GAN". In: arXiv preprint arXiv:1807.0734 (2018).
- 9 [JM70] D. H. Jacobson and D. Q. Mayne, Differential Dynamic Programming. Elsevier Press, 1970.
- 10 [JMW06] J. K. Johnson, D. M. Malioutov, and A. S. Willsky, "Walk-sum interpretation and analysis of Gaussian belief propagation". In: NIPS, 2006, pp. 579–586.
- 11 [JN20] C. Jin, P. Netrapalli, and M. I. Jordan, "What is local optimality in nonconvex-nonconcave minimax optimization?". In: Proceedings of the 34th International Conference on Machine Learning-Volume 73. 2020.
- 12 [JO18] M. Jankowiak and F. Obermeyer, "Pathwise Derivatives Beyond the Reparameterization Trick". In: ICML, 2018.
- 13 [JOA10] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal Regret Bounds for Reinforcement Learning". In: JMLR 11 (2010), pp. 1563–1600.
- 14 [JOA17] P. E. Jacob, J. O'Leary, and Y. F. Atchadé, "Unbiased markov chain monte carlo with couplings". In: arXiv preprint arXiv:1708.03625 (2017).
- 15 [Joh12] M. J. Johnson, "A Simple Explanation of A Spectral Algorithm for Learning Hidden Markov Models". In: (Apr. 2012). arXiv: 1204.2477 [stat.ME].
- 16 [Jon01] D. R. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces". In: J. Global Optimiz. 21.4 (Dec. 2001), pp. 345–383.
- 17 [Jor07] M. I. Jordan, An Introduction to Probabilistic Graphical Models. In preparation, 2007.
- 18 [Jor11] M. I. Jordan, "The era of Big Data". In: ISBA Bulletin. Vol. 18. 2011, pp. 1–3.
- 19 [Jor+98] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models". In: Learning in Graphical Models. Ed. by M. Jordan. MIT Press, 1998.
- 20 [Jos20] C. Joshi, Transformers are Graph Neural Networks. Tech. rep. 2020.
- 21 [Jos+22] L. V. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun, "Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users". In: (July 2022).
- 22 [JP95] R. Jirousek and S. Preulic, "On the effective implementation of the iterative proportional fitting procedure". In: Computational Statistics & Data Analysis 19 (1995), pp. 177–189.
- 23 [JS19] C. Ji and H. Shen, "Stochastic Variational Inference via Upper Bound". In: (Dec. 2019). arXiv: 1912.00650 [cs.LG].
- 24 [JS93] M. Jerrum and A. Sinclair, "Polynomial-time approximation algorithms for the Ising model". In: SIAM J. on Computing 22 (1993), pp. 1087–1116.
- 25 [JS96] M. Jerrum and A. Sinclair, "The Markov chain Monte Carlo method: an approach to approximate counting and integration". In: Approximation Algorithms for NP-hard problems. Ed. by D. S. Hochbaum. PWS Publishing, 1996.
- 26 [JSY19] P. Jaini, K. A. Selby, and Y. Yu, "Sum-of-Squares Polynomial Flow". In: ICML, 2019, pp. 3009–3018.
- 27 [JU97] S. Julier and J. Uhlmann, "A New Extension of the Kalman Filter to Nonlinear Systems". In: Proc. of AeroSense: The 11th Intl. Symp. on Aerospace/Defence Sensing, Simulation and Controls, 1997.
- 28 [JW14] M. Johnson and A. Willsky, "Stochastic Variational Inference for Bayesian Time Series Models". In: ICML, 2014, pp. 1854–1862.
- 29 [JW19] S. Jain and B. C. Wallace, "Attention is not explanation". In: arXiv preprint arXiv:1902.10186 (2019).
- 30 [Kaa12] Kaare Brandt Petersen and Michael Syskind Pedersen, The Matrix Cookbook. 2012.
- 31 [KAH19] F. H. Kingma, P. Abbeel, and J. Ho, "Bit-Swap: Recursive Bits-Back Coding for Lossless Compression with Hierarchical Latent Variables". In: ICML, 2019.
- 32 [Kai58] H. Kaiser, "The varimax criterion for analytic rotation in factor analysis". In: Psychometrika 23.3 (1958).
- 33 [Kak02] S. M. Kakade, "A Natural Policy Gradient". In: NIPS, 2002, pp. 1531–1538.
- 34 [Kal06] O. Kallenberg, Foundations of modern probability. Springer Science & Business Media, 2006.
- 35 [Kal+11] M. Kalakrishnan, S. Chitta, E. A. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic Trajectory Optimization for Motion Planning". In: ICRA, 2011, pp. 4569–4574.
- 36 [Kal+18a] D. Kalashnikov et al, "OT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: CORL, 2018.
- 37 [Kal+18b] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis". In: International Conference on Machine Learning. PMLR, 2018, pp. 2410–2419.
- 38 [Kam16] E. Kamar, "Directions in Hybrid Intelligence: Complementing AI Systems with Human Intelligence". In: IJCAI, 2016, pp. 4070–4073.
- 39 [Kan+15] N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin, "On Particle Methods for Parameter Estimation in State-Space Models". In: Stat. Sci. 30.3 (Aug. 2015), pp. 328–351.
- 40 [Kan+20] T. Kaneko, H. Kameoka, K. Tanaka, and N. Hojo, "CycleGAN-VC3: Examining and Improving CycleGAN-VCs for Mel-spectrogram Conversion". In: InterSpeech conference proceedings (2020).
- 41 [Kan42] L. Kantorovich, "On the transfer of masses (in Russian)". In: Doklady Akademii Nauk 37.2 (1942), pp. 227–229.
- 42 [Kar+18] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: ICLR, 2018.
- 43 [Kar+20a] A.-H. Karimi, G. Barthe, B. Balle, and I. Valera, "Model-Agnostic Counterfactual Explanations for Consequential Decisions". In: arXiv preprint 1905.11190 [cs.LG].
- 44 [Kar+20b] A.-H. Karimi, G. Barthe, B. Schölkopf, and I. Valera, "A survey of algorithmic recourse: definitions, formulations, solutions, and prospects". In: arXiv preprint arXiv:2010.04050 (2020).
- 45 [Kar+20c] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 8110–8119.
- 46 [Kar+21] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila, "Alias-Free Generative Adversarial Networks". In: arXiv preprint arXiv:2106.12423 (2021).
- 47 [Kat05] T. Katayama, Subspace Methods for Systems Identification. Springer Verlag, 2005.
- 48 [Kat+06] H. G. Katzgraber, S. Trebst, D. A. Huse, and M. Troyer, "Feedback-optimized parallel tempering Monte Carlo". In: Journal of Statistical Mechanics: Theory and Experiment 2006.03 (2006), P03018.
- 49 [Kat+17] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks". In: International Conference on Computer Aided Verification. Springer, 2017, pp. 97–117.
- 50 [Kat+19] N. Kato, H. Osone, K. Oomori, C. W. Ooi, and Y. Ochiai, "GANs-Based Clothes Design: Pattern Maker Is All You Need to Design Clothing". In: Proceedings of the 10th Augmented Human International Conference 2019, New York, NY, USA: Association for Computing Machinery, 2019.
- 51 [Kau+19] V. Kaushal, R. Iyer, S. Kothawade, R. Mahadev, K. Doctor, and G. Ramakrishnan, "Learning from less data: A unified data subset selection and active learning framework for computer vision". In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2019, pp. 1289–1299.
- 52 [Kau+21] D. Kaushik, A. Setlur, E. Hovy, and Z. C. Lipton, "Explaining The E cacy of Counterfactually Augmented Data". In: ICLR, 2021.
- 53 [KB00] H. J. Kushner and A. S. Budhiraja, "A nonlinear Iterating algorithm based on an approximation of the conditional distribution". In: IEEE Trans. Automat. Contr. 45.3 (Mar. 2000), pp. 580–585.
- 54 [KB14a] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization". In: arXiv preprint arXiv:1412.6980 (2014).

- 1
- 2 [KB14b] K. Kirchhoff and J. Bilmes. "Submodularity for Data Selection in Machine Translation". In: Empirical Methods in Natural Language Processing (EMNLP). 2014.
- 3 [KB15] D. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: ICLR. 2015.
- 4 [KB16] T. Kim and Y. Bengio. "Deep directed generative models with energy-based probability estimation". In: arXiv preprint arXiv:1606.03439 (2016).
- 5 [KBH19] F. Kunstrner, L. Balles, and P. Hennig. "Limitations of the Empirical Fisher Approximation". In: (May (2019). arXiv: 1905.12550 [cs.LG].
- 6 [KCC20] V. Kumar, A. Choudhary, and E. Cho. "Data Augmentation using Pre-trained Transformer Models". In: Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems. Suzhou, China: Association for Computational Linguistics, Dec. 2020, pp. 18–26.
- 7 [KD18a] S. Kanithi and M. P. Deisenroth. "Data-Ecient Reinforcement Learning with Probabilistic Model Predictive Control". In: AISTATS. 2018.
- 8 [KD18b] D. P. Kingma and P. Dhariwal. "Glow: Generative Flow with Invertible 1×1 Convolutions". In: NIPS. 2018.
- 9 [Ke+19a] L. Ke, M. Barnes, W. Sun, G. Lee, S. Choudhury, and S. Srinivas. "Imitation Learning as f -Divergence Minimization". In: arXiv preprint arXiv:1905.12888 (2019).
- 10 [Ke+19b] L. Ke, S. Choudhury, M. Barnes, W. Sun, G. Lee, and S. Srinivas. "Imitation Learning as f -Divergence Minimization". arXiv:1905.12888. 2019.
- 11 [Kei06] F. C. Keil. "Explanation and understanding". In: Ann. Rev. Psychol. 57 (2006), pp. 227–254.
- 12 [Kel+20] J. Kelly, J. Bettencourt, M. J. Johnson, and D. Duvenaud. "Learning Differential Equations that are Easy to Solve". In: Neural Information Processing Systems. 2020.
- 13 [Kem+06] C. Kemp, J. Tenenbaum, T. Y. T. Grifths, and N. Ueda. "Learning systems of concepts with an nite relational model". In: AAAI. 2006.
- 14 [Kem+10] C. Kemp, J. Tenenbaum, S. Niyogi, and T. Grifths. "A probabilistic model of theory formation". In: Cognition 114 (2010), pp. 165–196.
- 15 [Ken16] E. H. Kennedy. "Semiparametric theory and empirical processes in causal inference". In: Statistical causal inferences and their applications in public health research. ICSA Book Ser. Stat. Springer, [Cham], 2016, pp. 141–167.
- 16 [Ken17] E. H. Kennedy. Semiparametric theory. 2017. arXiv: 1709.06418 [stat.ME].
- 17 [Ken20] E. H. Kennedy. Optimal doubly robust estimation of heterogeneous causal effects. 2020. arXiv: 2004.14497 [math.ST].
- 18 [Kes+17] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: ICLR. 2017.
- 19 [KF09a] D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.
- 20 [KF09b] D. Krishnan and R. Fergus. "Fast Image Deconvolution using Hyper-Laplacian Priors". In: NIPS. 2009, pp. 1033–1041.
- 21 [KFL01] F. Kschischang, B. Frey, and H.-A. Loeliger. "Factor Graphs and the Sum-Product Algorithm". In: IEEE Trans Info. Theory (2001).
- 22 [KG05] A. Krause and C. Guestrin. "Near-optimal Nonmyopic Value of Information in Graphical Models". In: Proc. of the 21st Annual Conf. on Uncertainty in Artifical Intelligence (UAI 2005). AUAI Press, 2005, pp. 324–331.
- 23 [KG17] A. Kendall and Y. Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?". In: NIPS. Curran Associates, Inc., 2017, pp. 5574–5584.
- 24 [KGW11] M. Kalli, J. E. Grifn, and S. G. Walker. "Slice sampling mixture models". In: Statistics and computing 21.1 (2011), pp. 93–105.
- 25 [Kha+10] M. E. Khan, B. Marlin, G. Bouchard, and K. P. Murphy. "Variational bounds for mixed-data factor analysis". In: NIPS. 2010.
- 26 [Kha12] M. E. Khan. Variational Bounds for Learning and Inference with Discrete Data in Latent Gaussian Models. PhD thesis, UBC, 2012.
- 27 [Kha+18] M. E. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. "Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam". In: ICLR. 2018.
- 28 [Kha20] M. E. Khan. Deep learning with Bayesian principles. NeurIPS tutorial, 2020.
- 29 [Kha+21] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. "Transformers in Vision: A Survey". In: ACM Computing Surveys December (Jan. 2021).
- 30 [KHH20] A. Kristiadi, M. Hein, and P. Hennig. "Being Bayesian, Even Just a Bit, Fixes Overconvergence in ReLU Networks". In: ICLR. 2020.
- 31 [KHL20] D. Kaushik, E. Hovy, and Z. C. Lipton. Learning the Difference that Makes a Difference with Counterfactually-Augmented Data. 2020. arXiv: 1909.12434 [cs.CL].
- 32 [Kil+20] K. Killamsetty, D. Sivasubramanian, G. Ramakrishnan, and R. Iyer. "GLISTER: Generalization based Data Subset Selection for Ecient and Robust Learning". In: arXiv preprint arXiv:2012.10630 (2020).
- 33 [Kim+18a] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viégas, and R. Sayres. "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)". In: ICML. 2018, pp. 2668–2677.
- 34 [Kim+18c] Y. Kim, S. Wiseman, A. C. Miller, D. Sontag, and A. M. Rush. "Semi-Amortized Variational Autoencoders". In: ICML. 2018.
- 35 [Kim+19] S. Kim, S.-G. Lee, J. Song, J. Kim, and S. Yoon. "FlowWaveNet : A Generative Flow for Raw Audio". In: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 3370–3378.
- 36 [Kin+14] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. "Semi-Supervised Learning with Deep Generative Models". In: NIPS. 2014.
- 37 [Kin+16] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. "Improved Variational Inference with Inverse Autoregressive Flow". In: NIPS. 2016.
- 38 [Kin+19] P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dahne, D. Erhan, and B. Kim. "The (un)reliability of saliency methods". In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Springer, 2019, pp. 267–280.
- 39 [Kir+17] J. Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". en. In: PNAS 114.13 (2017), pp. 3521–3526.
- 40 [Kir+21] A. Kirsch, J. M. J. van Amerfoort, P. H. S. Torr, and Y. Gal. "On pitfalls in OoD detection: Predictive entropy considered harmful". In: ICML Workshop on Uncertainty in Deep Learning. 2021.
- 41 [Kit04] G. Kitagawa. "The two-Iter formula for smoothing and an implementation of the Gaussian-sum smoother". In: Annals of the Institute of Statistical Mathematics 46.4 (2004), pp. 605–623.
- 42 [Kiw+20] P. Kirichenko, P. Izmailov, and A. G. Wilson. "Why Normalizing Flows Fall to Detect Out-of-Distribution Data". In: (June 2020). arXiv: 2006.08549 [stat.ML].
- 43 [Kja+12a] J. Z. Kolter and T. S. Jaakkola. "Approximate Inference in Additive Factorial HMMs with Application to Energy Disaggregation". In: AISTATS. 2012.
- 44 [KJ12b] B. Kulis and M. I. Jordan. "Revisiting K-Means: New Algorithms via Bayesian Nonparametrics". In: Proceedings of the 29th International Conference on International Conference on Machine Learning (ICML'12). Edinburgh, Scotland: Omnipress, 2012, 1131–1138.
- 45 [Kja90] U. Kjaerul. Triangulation of graphs – algorithms giving small total state space. Tech. rep. R-90-09. Dept. of Math. and Comp. Sci., Aalborg Univ., Denmark, 1990.
- 46 [Kja92] U. Kjaerul. "Optimal decomposition of probabilistic networks by simulated annealing". In: Statistics and Computing. Vol. 2. 1992, pp. 7–17.
- 47 [KJD18] J. Knoblauch, J. Jewson, and T. Damoulas. "Doubly Robust Bayesian Inference for Non-Stationary Streaming Data with β -Divergences". In: NIPS. June 2018.
- 48 [KJD19] J. Knoblauch, J. Jewson, and T. Damoulas. "Generalized Variational Inference: Three arguments for deriving new Postiors". In: (Apr. 2019). arXiv: 1904.02063 [stat.ML].
- 49 [KJD21] J. Knoblauch, J. Jewson, and T. Damoulas. "An Optimization-centric View on Bayes' Rule: Reviewing and Generalizing Variational Inference". In: JMLR (2021).
- 50 [KJM19] N. M. Kriege, F. D. Johansson, and C. Morris. "A Survey on Graph Kernels". In: (Mar. 2019). arXiv: 1903.11835 [cs.LG].
- 51 [KJV83] S. Kirkpatrick, C. G. Jr., and M. Vecchi. "Optimization by simulated annealing". In: Science 220 (1983), pp. 671–680.
- 52 [KK11] P. Krähenbühl and V. Koltun. "Ecient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: NIPS. 2011.
- 53 [KKH20] I. Khemakhem, D. P. Kingma, and A. Hyvärinen. "Variational Autoencoders and Nonlinear ICA: A Unifying Framework". In: AISTATS. 2020.
- 54 [KKL20] N. Kitae, L. Kaiser, and A. Levskaya. "Reformer: The Ecient Transformer". In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. OpenReview.net, 2020.

BIBLIOGRAPHY

- [KKR95] K. Kanazawa, D. Koller, and S. Russell. "Stochastic Simulation Algorithms for Dynamic Probabilistic Networks". In: UAI. 1995.
- [KKS20] F. Kunstner, R. Kumar, and M. Schmidt. "Homeomorphic-Invariance of EM: Non-Asymptotic Convergence in KL Divergence for Exponential Families via Mirror Descent". In: (Nov. 2020). arXiv: 2011.01170 [cs.LG].
- [KKT03] D. Kempe, J. Kleinberg, and É. Tardos. "Maximizing the spread of influence through a social network". In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. 2003, pp. 137–146.
- [KL02] S. Kakade and J. Langford. "Approximately Optimal Approximate Reinforcement Learning". In: ICML. ICML '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 267–274.
- [KL09] H. Kawakatsu and A. Largey. "EM algorithms for ordered probit models with endogenous regressors". In: The Econometrics Journal 12.1 (2009), pp. 164–186.
- [KL10] D. P. Kingma and Y. LeCun. "Regularized estimation of image statistics by score matching". In: Advances in neural information processing systems. 2010, pp. 1126–1134.
- [KL17a] M. E. Khan and W. Lin. "Conjugate-Computation Variational Inference : Converting Variational Inference in Non-Conjugate Models to Inferences in Conjugate Models". In: AIS-TATS. 2017.
- [KL17b] P. W. Koh and P. Liang. "Understanding black-box predictions via in uence functions". In: International Conference on Machine Learning. PMLR. 2017, pp. 1885–1894.
- [KL21a] W. M. Kouw and M. Loog. "A review of domain adaptation without target labels". en. In: IEEE PAMI (Oct. 2021).
- [KL21b] W. M. Kouw and M. Loog. "A review of domain adaptation without target labels". en. In: IEEE Trans. Pattern Anal. Mach. Intell. (2021).
- [KLA19] T. Karras, S. Laine, and T. Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: CVPR. 2019.
- [Kle17] G. A. Klein. Sources of power: How people make decisions. MIT press, 2017.
- [KLM19] A. Kumar, P. Liang, and T. Ma. "Veri ed Uncertainty Calibration". In: NIPS. 2019.
- [KM00] J. Kwon and K. Murphy. Modeling Freeway Tra c with Coupled HMMs. Tech. rep. Univ. California, Berkeley, 2000.
- [KM08] U. Kjaerul and A. Madsen. Bayesian Networks and In uence Diagrams: A Guide to Construction and Analysis. Springer, 2008.
- [KMR16] J. Kleinberg, S. Mullainathan, and M. Raghavan. "Inherent trade-o s in the fair determination of risk scores". In: arXiv preprint arXiv:1609.05807 (2016).
- [KMY04] D. Kersten, P. Mamassian, and A. Yuille. "Object perception as Bayesian inference". en. In: Annu. Rev. Psychol. 55 (2004), pp. 271–304.
- [KN09] J. Z. Kolter and A. Y. Ng. "Near-Bayesian Exploration in Polynomial Time". In: ICML. 2009.
- [KN95] R. Kneser and H. Ney. "Improved backoing-o for n-gram language modeling". In: ICASSP. Vol. 1. 1995, pp. 181–184.
- [KNT20] I. Kostrikov, O. Nachum, and J. Tompson. "Imitation Learning via O-Policy Distribution Matching". In: ICLR. 2020.
- [Koe05] R. Koenker. Quantile Regression. en. Cambridge University Press, May 2005.
- [Koh+20] P. W. Koh, T. Nguyen, Y. S. Tang, S. Mussmann, E. Pierson, B. Kim, and P. Liang. "Concept bottleneck models". In: International Conference on Machine Learning. PMLR. 2020, pp. 5338–5348.
- [Koo03] G. Koop. Bayesian econometrics. Wiley, 2003.
- [Kor+15] A. Korattikara, V. Rathod, K. Murphy, and M. Welling. "Bayesian Dark Knowledge". In: NIPS. 2015.
- [Kor+20] A. Korotin, V. Egiazarian, A. Asadulaev, A. Sa n, and E. Burnaev. "Wasserstein-2 Generative Networks". In: International Conference on Learning Representations. 2020.
- [Kot+17] L. Kottho , C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA". In: JMLR 18.25 (2017), pp. 1–5.
- [Kot+22] S. Kothawade, V. Kaushal, G. Ramakrishnan, J. Bilmes, and R. Iyer. "PRISM: A Rich Class of Parameterized Submodular Information Measures for Guided Subset Selection". In: Proceedings of the AAAI Conference on Arti cial Intelligence. 2022.
- [Koy+10] S. Koyama, L. C. Pérez-Bolde, C. R. Shalizi, and R. E. Wasserman. "Approximate Methods for State-Space Models". en. In: JASA 105.489 (Mar. 2010), pp. 170–180.
- [KP20] A. Kumar and B. Poole. "On Implicit Regularization in β-VAEs". In: ICML. 2020.
- [KPB19] I. Kobyzev, S. Prince, and M. A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: (Aug. 2019). arXiv: 1908.09257 [stat.ML].
- [KPHL17] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. "Grammar Variational Autoencoder". In: ICML. 2017.
- [KPT21] J. S. Kim, G. Plumb, and A. Talwalkar. "Sensitivity Simulations for Saliency Methods". In: arXiv preprint arXiv:2105.06506 (2021).
- [KR21a] M. Khan and H. Rue. "The Bayesian Learning Rule". In: (2021).
- [KR21b] S. Kong and D. Ramanan. "OpenGAN: Open-Set Recognition via Open Data Generation". In: ICCV. Apr. 2021.
- [Kra08] A. Krause, H. Brendan McMahan, C. Guestrin, and A. Gupta. "Robust Submodular Observation Selection". In: JMLR 9 (2008), pp. 2761–2801.
- [Kri+05] B. Krishnapuram, L. Carin, M. Figueiredo, and A. Hartemink. "Learning sparse Bayesian classiers: multi-class formulation, fast algorithms, and generalization bounds". In: IEEE Transaction on Pattern Analysis and Machine Intelligence (2005).
- [KRL08] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. "Fast Inference in Sparse Coding Algorithms with Applications to Object Recognition". In: NIPS workshop on optimization in machine learning. 2008.
- [KRS14] B. Kim, C. Rudin, and J. A. Shah. "The bayesian case model: A generative approach for case-based reasoning and prototype classi cation". In: Advances in neural information processing systems. 2014, pp. 1952–1960.
- [Kru13] J. K. Kruschke. "Bayesian estimation supersedes the t test". In: J. Experimental Psychology: General 142.2 (2013), pp. 573–603.
- [KS06] L. Kocsis and C. Szepesvári. "Bandit Based Monte-Carlo Planning". In: ECML. 2006, pp. 282–293.
- [KS07] J. D. Y. Kang and J. L. Schafer. "Demystifying double robustness: a comparison of alternative strategies for estimating a population mean from incomplete data". In: Statist. Sci. 22.4 (2007), pp. 523–539.
- [KS15] H. Kaya and A. A. Salah. "Adaptive Mixtures of Factor Analyzers". In: (July 2015). arXiv: 1507.02801 [stat.ML].
- [KSB21] B. Kompa, J. Snoek, and A. Beam. "Empirical Frequency Coverage of Deep Learning Uncertainty Quantification Procedures". In: Entropy 23.12 (2021).
- [KSC98] S. Kim, N. Shephard, and S. Chib. "Stochastic Volatility: Likelihood Inference and Comparison with ARCH Models". In: Review of Economic Studies 65.3 (1998), pp. 361–393.
- [KSH12] A. Krizhevsky, I. Sutskever, and G. Hinton. "Imagenet classi cation with deep convolutional neural networks". In: NIPS. 2012.
- [KSL21] S. Kim, Q. Song, and F. Liang. "Stochastic gradient Langevin dynamics with adaptive drifts". In: J. Stat. Comput. Simul. (July 2021), pp. 1–19.
- [KSN99] N. L. Kleinman, J. C. Spall, and D. Q. Naiman. "Simulation-Based Optimization with Stochastic Approximation Using Common Random Numbers". In: Manage. Sci. 45.11 (1999), pp. 1570–1579.
- [KSS17] R. G. Krishnan, U. Shalit, and D. Sontag. "Structured Inference Networks for Nonlinear State Space Models". In: AAAI. 2017.
- [KT11a] A. Kulesza and B. Taskar. "k-DPPs: Fixed-size determinantal point processes". In: ICML. 2011.
- [KT11b] A. Kulesza and B. Taskar. "Learning Determinantal Point Processes". In: UAI. 2011.
- [KT+12] A. Kulesza, B. Taskar, et al. "Determinantal Point Processes for Machine Learning". In: Foundations and Trends in Machine Learning 5.2–3 (2012), pp. 123–286.
- [KTB11] D. P. Kroese, T. Taimre, and Z. I. Botev. Handbook of Monte Carlo Methods. en. 1 edition. Wiley, Mar. 2011.
- [KTX20] R. Kohavi, D. Tang, and Y. Xu. Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing. en. 1st ed. Cambridge University Press, Apr. 2020.
- [KU21] S. Khan and J. Ugander. Adaptive normalization for IPW estimation. 2021, arXiv: 2106.07695 [stat.ME].
- [Kua+09] P. Kuan, G. Pan, J. A. Thomson, R. Stewart, and S. Keles. "A hierarchical semi-Markov model for detecting enrichment with application to ChIP-Seq experiments". Tech. rep. U. Wisconsin, 2009.
- [Kub04] M. Kubale. Graph colorings. Vol. 352. American Mathematical Society, 2004.
- [Kuc+16] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei. "Automatic Di erentiation Variational Inference". In: JMLR (2016).

- 1
- 2 [Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: Naval Research Logistics Quarterly 2 (1955), pp. 83–97.
- 3 [Kul+13] T. Kulesza, S. Stumpf, M. Burnett, S. Yang, I. Kwan, and W.-K. Wong. "Too much, too little, or just right? Why explanations impact end users' mental models". In: 2013 IEEE Symposium on visual languages and human centric computing. IEEE, 2013, pp. 3–10.
- 4 [Kull+19] M. Kull, M. Perello-Nieto, M. Kängsepp, T. S. Filho, H. Song, and P. Flach. "Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration". In: NeurIPS. 2019, pp. 11761–11771.
- 5 [Kum+19a] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. "Stabilizing O-Policy Q-Learning via Bootstrapping Error Reduction". In: NeurIPS. 2019, pp. 11761–11771.
- 6 [Kum+19b] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. P. Kingma. "VideoFlow: A few-based generative model for video". In: ICML Workshop on Invertible Neural Networks and Normalizing Flows (2019).
- 7 [Kum+19c] R. Kumar, S. Ozair, A. Goyal, A. Courville, and Y. Bengio. "Maximum entropy generators for energy-based models". In: arXiv preprint arXiv:1901.08508 (2019).
- 8 [Kün+19] S. R. Künzel, J. S. Sekhon, P. J. Bickel, and B. Yu. "Metalearners for estimating heterogeneous treatment effects using machine learning". In: Proceedings of the National Academy of Sciences 116:10 (2019), pp. 4156–4165. eprint: <https://www.pnas.org/content/116/10/4156.full.pdf>.
- 9 [Kur+19] T. Kuratach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. "Model-Ensemble Trust-Region Policy Optimization". In: ICLR. 2019.
- 10 [Kur+20] R. Kurle, B. Cseke, A. Klushyn, P. van der Smagt, and S. Günnemann. "Continual Learning with Bayesian Neural Networks for Non-Stationary Data". In: ICLR. 2020.
- 11 [Kus+18] M. J. Kusner, J. R. Loftus, C. Russell, and R. Silva. Counterfactual Fairness. 2018. arXiv: [1703.06856 \[stat.ML\]](https://arxiv.org/abs/1703.06856).
- 12 [Kus64] H. J. Kushner. "A New Method of Locating the Maximum Point of an Arbitrary Multipiece Curve in the Presence of Noise". In: J. Basic Eng 86.1 (Mar. 1964), pp. 97–106.
- 13 [KVK10] A. Klami, S. Virtanen, and S. Kaski. "Bayesian exponential family projections for coupled data sources". In: UAI. 2010.
- 14 [KW14] D. P. Kingma and M. Welling. "Auto-encoding variational Bayes". In: ICLR. 2014.
- 15 [KW18] A. S. I. Kim and M. P. Wand. "On expectation propagation for generalised, linear and mixed models". In: Aust. N. Z. J. Stat. 60.1 (Mar. 2018), pp. 75–102.
- 16 [KW19a] D. P. Kingma and M. Welling. "An Introduction to Variational Autoencoders". In: Foundations and Trends in Machine Learning 12.4 (2019), pp. 307–392.
- 17 [KW19b] M. J. Kochenderfer and T. A. Wheeler. Algorithms for Optimization. en: The MIT Press, Mar. 2019.
- 18 [KW70] G. S. Kimeldorf and G. Wahba. "A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines". en: In: Ann. Math. Stat. 41.2 (Apr. 1970), pp. 495–502.
- 19 [KW96] R. E. Kass and L. Wasserman. "The Selection of Prior Distributions by Formal Rules". In: JASA 91.435 (1996), pp. 1343–1370.
- 20 [KW06] K. Kurihara, M. Welling, and N. Vlassis. "Accelerated variational DP mixture models". In: NIPS. 2006.
- 21 [KWW22] M. J. Kochenderfer, T. A. Wheeler, and K. Wray. Algorithms for Decision Making. The MIT Press, 2022.
- 22 [KY94] J. J. Kosowsky and A. L. Yuille. "The invisible hand algorithm: Solving the assignment problem with statistical physics". In: Neural networks 7.3 (1994), pp. 477–490.
- 23 [Kyn+19] T. Kynkääniemi, T. Karras, S. Laine, J. Lehtinen, and T. Aila. "Improved Precision and Recall Metric for Assessing Generative Models". In: NeurIPS. 2019.
- 24 [KZ02] V. Kolmogorov and R. Zabih. "What energy functions can be minimized via graph cuts?". In: Computer Vision—ECCV 2002 (2002), pp. 185–208.
- 25 [LA87] P. J. M. Laarhoven and E. H. L. Aarts, eds. Simulated Annealing: Theory and Applications. Norwell, MA, USA: Kluwer Academic Publishers, 1987.
- 26 [Lab18] R. Labbe. Kalman and Bayesian Filters in Python. 2018.
- 27 [Lag+19] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. J. Gershman, and F. Doshi-Velez. "Human evaluation of models built for interpretability". In: Proceedings of the AAAI Conference on Human Computation and Crowdsourcing. Vol. 7. 1. 2019, pp. 59–67.
- 28 [Lak+17] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. "Building Machines That Learn and Think Like People". en: In: Behav. Brain Sci. (2017), pp. 1–101.
- 29 [Lai+21] A. Lal, M. W. Lockhart, Y. Xu, and Z. Zu. "How Much Should We Trust Instrumental Variable Estimates in Political Science? Practical Advice based on Over 60 Replicated Studies". In: (2021).
- 30 [Lam92] D. Lambert. "Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing". In: Technometrics 34.1 (1992), pp. 1–14.
- 31 [Lan+12] H. Langseth, T. D. Nielsen, R. Rum', and A. Salmeron. "Mixtures of truncated basis functions". In: Int. J. Approx. Reason. 53.2 (Feb. 2012), pp. 212–227.
- 32 [Lao+20] J. Lao, C. Suter, I. Langmore, C. Chimisov, A. Saxena, P. Sountsov, D. Moore, R. A. Saurous, M. D. Ho man, and J. V. Dillon. "tfpmcmc: Modern Markov Chain Monte Carlo Tools Built for Modern Hardware". In: PROBPROG. 2020.
- 33 [Lar+16] A. B. L. Larsen, S. K. Sonderby, H. Larochelle, and O. Winther. "Autoencoding beyond pixels using a learned similarity metric". In: International conference on machine learning. PMLR, 2016, pp. 1558–1566.
- 34 [Las08] K. B. Laskey. "MEBN: A language for first-order Bayesian knowledge bases". In: Artif. Intell. 172.2 (Feb. 2008), pp. 140–178.
- 35 [Lau92] S. L. Lauritzen. "Propagation of probabilities, means and variances in mixed graphical association models". In: JASA 87.420 (1992), pp. 1098–1108.
- 36 [Lau95] S. L. Lauritzen. "The EM algorithm for graphical association models with missing data". In: Computational Statistics and Data Analysis 19 (1995), pp. 191–201.
- 37 [Lau96] S. Lauritzen. Graphical Models. OUP, 1996.
- 38 [Law05] N. D. Lawrence. "Probabilistic non-linear principal component analysis with Gaussian process latent variable models". In: JMLR 6 (2005), pp. 1783–1816.
- 39 [Law19] N. Lawrence. Deep Gaussian Processes (Machine learning summer school tutorial). 2019.
- 40 [Lin+09] H. Lin and J. A. Bilmes. "How to Select a Good Training-Data Subset for Transcription: Submodular Active Selection for Sequences". In: Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH). Brighton, UK, 2009.
- 41 [LB10a] H. Lin and J. A. Bilmes. "Multi-document Summarization via Budgeted Maximization of Submodular Functions". In: North American chapter of the Association for Computational Linguistics/Human Language Technology Conference (NAACL/HLT-2010). Los Angeles, CA, 2010.
- 42 [LB10b] H. Lin and J. A. Bilmes. "An Application of the Submodular Principal Partition to Training Data Subset Selection". In: Neural Information Processing Society (NeurIPS, formerly NIPS) Workshop, NeurIPS (formerly NIPS) Workshop on Discrete Optimization in Machine Learning: Submodularity, Sparsity & Polyhedra (DISCML). Vancouver, Canada, 2010.
- 43 [LB11] H. Lin and J. A. Bilmes. "A Class of Submodular Functions for Document Summarization". In: The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL/HLT-2011). (long paper). Portland, OR, 2011.
- 44 [LB12] H. Lin and J. Bilmes. "Learning Mixtures of Submodular Shells with Application to Document Summarization". In: Uncertainty in Artificial Intelligence (UAI). Catalina Island, USA: AUAI, 2012.
- 45 [LB19] A. Levry and V. Belle. "Learning Tractable Probabilistic Models in Open Worlds". In: (Jan. 2019). arXiv: [1901.05847 \[cs.LG\]](https://arxiv.org/abs/1901.05847).
- 46 [LB20] Y. Liu, P.-L. Bacon, and E. Brunskill. "Understanding the Curse of Horizon in O-Policy Evaluation via Conditional Importance Sampling". In: ICML. 2020.
- 47 [LBL16] H. Lakkaraju, S. H. Bach, and J. Leskovec. "Interpretable decision sets: A joint framework for description and prediction". In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016, pp. 1675–1684.
- 48 [LBS17] C. Lakshminarayanan, S. Bhattacharjee, and C. Szepesvári. "A Linearly Relaxed Approximate Linear Program for Markov Decision Processes". In: IEEE Transactions on Automatic Control 63.4 (2017), pp. 1185–1191.
- 49 [LBW17] T. A. Le, A. G. Baydin, and F. Wood. "Inference Compilation and Universal Probabilistic Programming". In: AISATS. 2017.
- 50 [LC02] J. Langford and R. Caruana. "(Not) bounding the true error". In: NIPS. 2002.
- 51 [LCG12] Y. Lou, R. Caruana, and J. Gehrke. "Intelligible models for classification and regression". In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. 2012, pp. 150–158.
- 52 [LCR21] T. Lesort, M. Caccia, and I. Rish. "Understanding Continual Learning Settings with Data Distribution Drift Analysis". In: (Apr. 2021). arXiv: [2104.01678 \[cs.LG\]](https://arxiv.org/abs/2104.01678).

1901.05847

BIBLIOGRAPHY

- [LDZ11] Y Li, H Duan, and C. X. Zhai. "Cloudspeller: Spelling correction for search queries by using a unified hidden markov model with web-scale resources". In: SIGIR. 2011.
- [LDZ12] Y. Li, H. Duan, and C. Zhai. "A Generalized Hidden Markov Model with Discriminative Training for Query Spelling Correction". In: SIGIR. 2012, pp. 611–620.
- [Le+18] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood. "Auto-Encoding Sequential Monte Carlo". In: ICLR. 2018.
- [L'E18] P. L'Ecuyer. "Randomized Quasi-Monte Carlo: An Introduction for Practitioners". In: Monte Carlo and Quasi-Monte Carlo Methods. Springer International Publishing, 2018, pp. 29–52.
- [Le+19] T. A. Le, A. R. Kosirok, N Siddharth, Y. W. Teh, and F. Wood. "Revisiting Reweighted Wake-Sleep for Models with Stochastic Control Flow". In: UAI. 2019.
- [LeC98] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. "Ecient BackProp". en. In: Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1998, pp. 9–50.
- [Lee06] J. de Leeuw. "Principal Component Analysis of Binary Data by Iterated Singular Value Decomposition". In: Comput. Stat. Data Anal. 50.1 (Jan. 2006), pp. 21–39.
- [Lee+10] J. Lee, V. Mirrokni, V. Nagarajan, and M. Sviridenko. "Maximizing Nonmonotone Submodular Functions under Matroid or Knapsack Constraints". In: SIAM Journal on Discrete Mathematics 23.4 (2010), pp. 2053–2078.
- [Lee+18] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. "Deep Neural Networks as Gaussian Processes". In: ICLR. 2018.
- [Lei+18] J. Lei, M. G'Sell, A. Rinaldo, R. Tibshirani, and L. Wasserman. "Distribution-Free Predictive Inference For Regression". In: JASA (2018).
- [Lei+20] F. Fleißner, V. Dutordoir, S. T. John, and N. Durande. "A Tutorial on Sparse Gaussian Processes and Variational Inference". In: (Dec. 2020). arXiv: 2012.13962 [cs.LG].
- [Lem09] C. Lemieux. Monte Carlo and Quasi-Monte Carlo Sampling. Springer, New York, NY, 2009.
- [Léo14] C. Léonard. "A survey of the Schrödinger problem and some of its connections with optimal transport". In: Discrete & Continuous Dynamical Systems 34.4 (2014), p. 1533.
- [Let+15a] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. "Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model". In: The Annals of Applied Statistics 9.3 (2015).
- [Let+15b] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model". In: The Annals of Applied Statistics 9.3 (2015), pp. 1350–1371.
- [Lev18] S. Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: (May 2018). arXiv: 1805.00909 [cs.LG].
- [Lev+20] S. Levine, A. Kumar, G. Tucker, and J. Fu. "One ine Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. arXiv:2005.01643. 2020.
- [LF+21] L. Le Folgoc, V. Baltatzis, S. Desai, A. Devaraj, S. Ellis, O. E. Martinez Mamanera, A. Nair, H. Qiu, J. Schnabel, and B. Glocker. "Is MC Dropout Bayesian?". In: (Oct. 2021). arXiv: 2110.04286 [cs.LG].
- [LFG21] F. Lin, X. Fang, and Z. Gao. "Distributionally Robust Optimization: A review on theory and applications". In: Numer. Algebra Control Optim. 12.1 (Nov. 2021), pp. 159–212.
- [LGMT11] F. Le Gland, V. Monbet, and V.-D. Tran. "Large Sample Asymptotics for the Ensemble Kalman Filter". In: Oxford Handbook of Nonlinear Filtering. Ed. by D Crisan And. 2011.
- [LHLT15] Y. Li, J. M. Hernandez-Lobato, and R. E. Turner. "Stochastic Expectation Propagation". In: NIPS. 2015.
- [LHR20] A. Lucic, H. Hanned, and M. de Rijke. "Why does my model fail? contrastive visual explanations for retail forecasting". In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency. 2020, pp. 90–98.
- [Li+10] L. Li, W. Chu, J. Langford, and R. E. Schapire. "A contextual-bandit approach to personalized news article recommendation". In: WWW. 2010.
- [Li+16] C. Li, C. Chen, D. Carlson, and L. Carin. "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks". In: AAAI. 2016.
- [Li+17a] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Poczos. "Mmd gan: Towards deeper understanding of moment matching network". In: Advances in Neural Information Processing Systems. 2017, pp. 2203–2213.
- [Li+17b] L. Li, K. Jamieson, G. De Salvo, A. Rostamizadeh, and A. Talwalkar. "Hyperband: bandit-based configuration evaluation for hyperparameter optimization". In: ICLR. 2017.
- [Li+17c] O. Li, H. Liu, C. Chen, and C. Rudin. Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions. 2017. arXiv: 1710.04806 [cs.AI].
- [Li+17d] T.-C. Li, J.-Y. Su, W. Liu, and J. M. Corchado. "Approximate Gaussian conjugacy: parametric recursive Itering under nonlinearity, multimodality, uncertainty, and constraint, and beyond". In: Frontiers of Information Technology & Electronic Engineering 18.12 (Dec. 2017), pp. 1913–1939.
- [Li+18] X. Li, C. Li, J. Chi, J. Ouyang, and W. Wang. "Black-box Expectation Propagation for Bayesian Models". In: ICDM. Proceedings. Society for Industrial and Applied Mathematics, May 2018, pp. 603–611.
- [Li18] Y. Li. "Deep Reinforcement Learning". In: (Oct. 2018). arXiv: 1810.06339 [cs.LG].
- [Li+19] J. Li, S. Qu, X. Li, J. Szurley, J. Z. Kolter, and F. Metze. "Adversarial Music: Real world Audio Adversary against Wake-word Detection System". In: NIPS. Curran Associates, Inc., 2019, pp. 11908–11918.
- [Li+20a] C. Li, X. Gao, Y. Li, B. Peng, X. Li, Y. Zhang, and J. Gao. "Optimus: Organizing Sentences via Pre-trained Modeling of a Latent Space". In: EMNLP. Apr. 2020.
- [Li+20b] R. Li, S. Pei, B. Chen, Y. Song, T. Zhang, W. Yang, and J. Shaman. "Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (SARS-CoV2)". en. In: Science (Mar. 2020).
- [Lia+07] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. "Learning and Inferring Transportation Routines". In: Artificial Intelligence 171.5 (2007), pp. 311–331.
- [Lia+08] F. Liang, R. Paulo, G. Molina, M. Clyde, and J. Berger. "Mixtures of g-priors for Bayesian Variable Selection". In: JASA 103.481 (2008), pp. 410–423.
- [Lia+19] V. Liao, R. Ballamy, M. Muller, and H. Candello. "Human-AI Collaboration: Towards Socially-Guided Machine Learning". In: CHI Workshop on Human-Centered Machine Learning Perspectives. 2019.
- [Lin+16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning". In: ICLR. 2016.
- [Lin13a] W. Lin. "Agnostic notes on regression adjustments to experimental data: Reexamining Freedman's critique". In: The Annals of Applied Statistics 7.1 (2013), pp. 295–318.
- [Lin13b] D. A. Linzer. "Dynamic Bayesian Forecasting of Presidential Elections in the States". In: JASA 108.501 (Mar. 2013), pp. 124–134.
- [Lin+17] K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun. "Adversarial ranking for language generation". In: Advances in Neural Information Processing Systems. 2017, pp. 3155–3165.
- [Lin+20] H. Lin, H. Chen, T. Zhang, C. Laroche, and K. Choromanski. "Demystifying Orthogonal Monte Carlo and Beyond". In: (May 2020). arXiv: 2005.13590 [cs.LG].
- [Lin+21] T. Lin, Y. Wang, X. Liu, and X. Qiu. "A Survey of Transformers". In: (June 2021). arXiv: 2106.04554 [cs.LG].
- [Lin88a] B. Lindsay. "Composite Likelihood Methods". In: Contemporary Mathematics. 80.1 (1988), pp. 221–239.
- [Lin88b] R. Linsker. "Self-organization in a perceptual network". In: Computer. 21.3 (Mar. 1988), pp. 105–117.
- [Lin92] L.-J. Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning, and Teaching". In: Mach. Learn. 8.3-4 (May 1992), pp. 293–321.
- [Lip18] Z. C. Lipton. "The Myths of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery". In: Queue 16.3 (2018), pp. 31–57.
- [Liu01] J. Liu. Monte Carlo Strategies in Scientific Computation. Springer, 2001.
- [Liu+15] Z. Liu, P. Luo, X. Wang, and X. Tang. "Deep Learning Face Attributes in the Wild". In: ICCV. 2015.
- [Liu+18a] L. Liu, X. Liu, C.-J. Hsieh, and D. Tao. "Stochastic Second-order Methods for Non-convex Optimization with Inexact Hessian and Gradient". In: (Sept. 2018). arXiv: 1809.09853 [math.OC].
- [Liu+18b] Q. Liu, L. Li, Z. Tang, and D. Zhou. "Breaking the Curse of Horizon: In nite-horizon O-policy Estimation". In: NeurIPS. Curran Associates Inc., 2018, pp. 5361–5371.
- [Liu+19a] H. Liu, Y.-S. Ong, Z. Yu, J. Cai, and X. Shen. "Scalable Gaussian Process Classification with Additive Noise for Various Likelihoods". In: (Sept. 2019). arXiv: 1909.06541 [stat.ML].
- [Liu+19b] R. Liu, J. Regier, N. Tripathaneni, M. I. Jordan, and J. McAuley. "Rao-Blackwellized Stochastic Gradients for Discrete Distributions". In: ICML. 2019.
- [Liu+20a] F. Liu, W. Xu, J. Lu, G. Zhang, A. Gretton, and D. J. Sutherland. "Learning Deep Kernels for Non-Parametric Two-Sample Tests". In: ICML. Feb. 2020.
- [Liu+20b] F. Liu, W. Xu, J. Lu, G. Zhang, A. Gretton, and D. J. Sutherland. "Learning deep kernels for non-parametric

- 1 two-sample tests". In: International Conference on Machine
 2 Learning. PMLR. 2020, pp. 6316–6326.
- 3 [Liu+20c] H. Liu, Y.-S. Ong, X. Shen, and J. Cai. "When Gaussian Process Meets Big Data: A Review of Scalable GPs". In:
 4 IEEE Transactions on Neural Networks and Learning Systems 31.1 (2020).
- 5 [Liu+20d] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-
 6 Weiss, and B. Lakshminarayanan. "Simple and Principled Un-
 7 certainty Estimation with Deterministic Deep Learning via Dis-
 8 tance Awareness". In: NIPS. 2020.
- 9 [Liu+21] W. Liu, X. Wang, J. D. Owens, and Y. Li. "Energy-
 10 based Out-of-distribution Detection". In: NIPS. 2021.
- 11 [Liu+22] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Dar-
 12 rell, and S. Xie. "A ConvNet for the 2020s". In: (Jan. 2022).
 arXiv: 2201.03545 [cs.CV].
- 13 [LJ08] P. Liang and M. I. Jordan. "An Asymptotic Analysis of
 14 Generative, Discriminative, and Pseudolikelihood Estimators". In:
 15 International Conference on Machine Learning (ICML). 2008.
- 16 [LJS14] F. Lindsten, M. I. Jordan, and T. B. Schön. "Parti-
 17 ciple Gibbs with Ancestor Sampling". In: JMLR 15.63 (2014),
 pp. 2145–2184.
- 18 [Lju87] L. Ljung. System Identification: Theory for the User.
 Prentice Hall, 1987.
- 19 [LK07] P. Liang and D. Klein. Structured Bayesian Nonpara-
 20 metric Models with Variational Inference. ACL Tutorial. 2007.
- 21 [LK09] P. Liang and D. Klein. "Online EM for Unsupervised
 22 Models". In: NAACL. 2009.
- 23 [LJK09] D. Lewandowski, D. Kurowicka, and H. Joe. "Gen-
 24 erating random correlation matrices based on vines and ex-
 tended onion method". In: J. Multivar. Anal. 100.9 (Oct. 2009),
 pp. 1989–2001.
- 25 [LL17] S. M. Lundberg and S.-I. Lee. "A unified approach to in-
 26 terpreting model predictions". In: NIPS. 2017, pp. 4765–4774.
- 27 [LLC20] M. Locher, K. B. Laskey, and P. C. G. Costa. "Design
 28 patterns for modeling first-order expressive Bayesian networks". In:
 Knowl. Eng. Rev. 35 (2020).
- 29 [LLJ16] Q. Liu, J. Lee, and M. Jordan. "A kernelized Stein dis-
 30 crepancy for goodness-of-fit tests". In: International conference
 on machine learning. 2016, pp. 276–284.
- 31 [LLN06] B. Lehmann, D. Lehmann, and N. Nisan. "Combinato-
 32 rial auctions with decreasing marginal utilities". In: Games and
 Economic Behavior 55.2 (2006), pp. 270–296.
- 33 [Llo+14] J. R. Lloyd, D. Duvenaud, R. Grosse, J. B. Tenenbaum,
 34 and Z. Ghahramani. "Automatic Construction and Natural-
 Language Description of Nonparametric Regression Models". In:
 AAAI. 2014.
- 35 [LLT89] K. Lange, R. Little, and J. Taylor. "Robust Statistical
 36 Modeling Using the T Distribution". In: JASA 84.408 (1989),
 pp. 881–896.
- 37 [LM11] H. Larochelle and I. Murray. "The neural autoregressive
 38 distribution estimator". In: AISTATS. Vol. 15. 2011, pp. 29–37.
- 39 [LM20] M. L. Leavitt and A. Morcos. "Towards falsifiable in-
 40 terpretability research". In: arXiv preprint arXiv:2010.12016
 (2020).
- 41 [LMP01] J. Laréty, A. McCallum, and F. Pereira. "Conditional
 42 Random Fields: Probabilistic Models for Segmenting and Label-
 43 ing Sequence Data". In: ICML. 2001.
- 44 [LMR15] F. Lavancier, J. Møller, and E. Rubak. "Determinant-
 45 al point process models and statistical inference". In: Journal of
 46 the Royal Statistical Society: Series B (Statistical Methodology)
 77.4 (2015), pp. 853–877.
- 47 [LMS16] B. Leimkuhler, C. Matthews, and G. Stoltz. "The
 48 computation of averages from equilibrium and nonequilibrium
 49 Langevin molecular dynamics". In: IMA J. Numer. Anal. 36.1
 (Jan. 2016), pp. 13–79.
- 50 [LN01] S. Lauritzen and D. Nilsson. "Representing and solving
 51 decision problems with limited information". In: Management
 Science 47 (2001), pp. 1238–1251.
- 52 [LN19] H. Lin and V. Ng. "Abstractive summarization: A survey
 53 of the state of the art". In: Proceedings of the AAAI Conference
 54 on Artificial Intelligence. Vol. 33. 01. 2019, pp. 9815–9822.
- 55 [LN96] J. Lee and J. A. Nelder. "Hierarchical Generalized Lin-
 56 ear Models". In: J. of Royal Stat. Soc. Series B 58.4 (1996),
 pp. 619–678.
- 57 [Loc+18] F. Locatello, S. Bauer, M. Lucic, G. Rätsch, S. Gelly,
 58 B. Schölkopf, and O. Bachem. "Challenging Common Assump-
 59 tions in the Unsupervised Learning of Disentangled Repre-
 60 sentations". In: (Nov. 2018). arXiv: 1811.12359 [cs.LG].
- 61 [Lod+02] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini,
 62 and C. Watkins. "Text classification using string kernels". In:
 63 J. Mach. Learn. Res. (Mar. 2002).
- 64 [Loe04] H. Loeliger. "An introduction to factor graphs". In:
 65 IEEE Signal Process. Magazine 21.1 (Jan. 2004), pp. 28–41.
- 66 [Loe+07] H. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, and F. R.
 67 Kschischang. "The Factor Graph Approach to Model-Based Sig-
 68 nal Processing". In: Proc. IEEE 95.6 (June 2007), pp. 1295–
 1322.
- 69 [Lon+18] M. Long, Z. Cao, J. Wang, and M. I. Jordan. "Condi-
 70 tional adversarial domain adaptation". In: Neural Information
 71 Processing Systems (2018).
- 72 [Lov+20] T. Lovett, M. Briers, M. Charalambides, R. Jersakova,
 73 J. Lomax, and C. Holmes. "Inferring proximity from Blue-
 74 tooth Low Energy RSSI with Unscented Kalman Smoothers". In:
 July 2020. arXiv: 2007.05057 [eess.SP].
- 75 [Lov83] L. Lovász. "Submodular functions and convexity". In:
 Mathematical programming the state of the art. Springer, 1983,
 pp. 235–257.
- 76 [LP01] U. Lerner and R. Parr. "Inference in Hybrid Networks:
 Theoretical Limits and Practical Algorithms". In: UAI. 2001.
- 77 [LP03] M. G. Lagoudakis and R. Parr. "Least-Squares Policy
 Iteration". In: JMLR 4 (2003), pp. 1107–1149.
- 78 [LP06] N. Lartillot and H. Philippe. "Computing Bayes factors
 using thermodynamic integration". In: In: Systematic Biology
 55.2 (Apr. 2006), pp. 195–207.
- 79 [LPB17] B. Lakshminarayanan, A. Pritzel, and C. Blundell.
 "Simple and Scalable Predictive Uncertainty Estimation using
 Deep Ensembles". In: NIPS. 2017.
- 80 [LPO17] D. Lopez-Paz and M. Oquab. "Revisiting classi-
 81 er two-sample tests". In: International Conference on Learning
 Representations. 2017.
- 82 [LPR17] D. Lopez-Paz and M. Ranzato. "Gradient Episodic
 Memory for Continual Learning". In: NIPS. June 2017.
- 83 [LR18] T. Liang and A. Rakhlil. "Just Interpolate: Kernel
 "Ridgeless" Regression Can Generalize". In: (Aug. 2018). arXiv:
 1808.00387 [math.ST].
- 84 [LR85] T. L. Lai and H. Robbins. "Asymptotically efficient
 adaptive allocation rules". In: Adv. Appl. Math. (Mar.
 1985).
- 85 [LR87] R. J. Little and D. B. Rubin. Statistical Analysis with
 Missing Data. New York: Wiley and Son, 1987.
- 86 [LR95] C. Liu and D. Rubin. "ML Estimation of the T distri-
 87 bution using EM and its extensions, ECM and ECME". In: Sta-
 tistica Sinica 5 (1995), pp. 19–39.
- 88 [LRC19] Y. Li, B. I. P. Rubinstein, and T. Cohn. "Truth In-
 89 ference at Scale: A Bayesian Model for Adjudicating Highly
 Redundant Crowd Annotations". In: WWW. Feb. 2019.
- 90 [LS01] D. Lee and S. Seung. "Algorithms for non-negative ma-
 trix factorization". In: NIPS. 2001.
- 91 [LS19] T. Lattimore and C. Szepesvári. Bandit Algorithms.
 Cambridge, 2019.
- 92 [LS79] P. W. Lewis and G. S. Shedler. "Simulation of nonho-
 93 mogeneous Poisson processes by thinning". In: Naval research
 logistics quarterly 26.3 (1979), pp. 403–413.
- 94 [LS88] S. L. Lauritzen and D. J. Spiegelhalter. "Local computa-
 95 tions with probabilities on graphical structures and their appli-
 96 cations to expert systems". In: J. of Royal Stat. Soc. Series B
 B.50 (1988), pp. 127–224.
- 97 [LS98] V. Lepar and P. P. Shenoy. "A Comparison of Lauritzen-
 98 Spiegelhalter-Hugin and Shenoy-Shafer Architectures for Com-
 puting Marginals of Probability Distributions". In: UAI. Ed. by
 G. Cooper and S. Moral. Morgan Kaufmann, 1998, pp. 328–337.
- 99 [LS99] D. D. Lee and H. S. Seung. "Learning the parts of objects
 by non-negative matrix factorization". In: Nature 401.6755
 (1999), pp. 788–791.
- 100 [LS16] N. Loo, S. Swaroop, and R. E. Turner. "Generalized
 101 Variational Continual Learning". In: ICLR. 2021.
- 102 [LST90] J. K. Lenstra, D. B. Shmoys, and É. Tardos. "Approx-
 103 imation algorithms for scheduling unrelated parallel machines". In:
 Mathematical programming. 1990.
- 104 [LSV09] J. Lee, M. Srividenko, and J. Vondrák. "Submodular
 105 maximization over multiple matroids via generalized exchange
 properties". In: Approximation, Randomization, and Com-
 binatorial Optimization. Algorithms and Techniques (2009),
 pp. 244–257.
- 106 [LSW15] Y. T. Lee, A. Sidford, and S. C.-w. Wong. "A faster cut-
 107 ting plane method and its implications for combinatorial and
 108 convex optimization". In: 2015 IEEE 56th Annual Symposium
 on Foundations of Computer Science. IEEE. 2015, pp. 1049–
 1065.
- 109 [LSZ15] Y. Li, K. Swersky, and R. Zemel. "Generative Moment
 Matching Networks". In: ICML. 2015.
- 110 [LT16] M.-Y. Liu and O. Tuzel. "Coupled Generative Adversar-
 111 ial Networks". In: NIPS. 2016, pp. 469–477.
- 112 [LTW15] Q. Li, C. Tai, and E. Weinan. "Stochastic modi ed
 113 equations and adaptive stochastic gradient algorithms". In:
 ICML. Nov. 2015.

BIBLIOGRAPHY

- [Lu+20] J. Lu, P. Gong, J. Ye, and C. Zhang. "Learning from Very Few Samples: A Survey". In: (Sept. 2020). arXiv: 2009.02653 [cs.LG].
- [Lu+21] X. Lu, I. Osband, B. Van Roy, and Z. Wen. "Evaluating Probabilistic Inference in Deep Learning: Beyond Marginal Predictions". In: (July 2021). arXiv: 2107.09224 [cs.LG].
- [Lu+22] X. Lu, I. Osband, B. Van Roy, and Z. Wen. "From Predictions to Decisions: The Importance of Joint Predictive Distributions". In: (Feb. 2022). arXiv: 2107.09224 [cs.LG].
- [Luc+18] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos. "Using Deep Neural Networks for Inverse Problems in Imaging: Beyond Analytical Methods". In: IEEE Signal Process. Mag. 35.1 (Jan. 2018), pp. 20–36.
- [Luc+19] J. Lucas, C. Tucker, R. Grosse, and M. Norouzi. "Don't blame the ELBO! A linear VAE perspective on posterior collapse". In: NIPS. 2019.
- [Luh58] H. P. Luhn. "The automatic creation of literature abstracts". In: IBM Journal of research and development 2.2 (1958), pp. 159–165.
- [Lun+20] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. "From local explanations to global understanding with explainable AI for trees". In: Nature machine intelligence 2.1 (2020), pp. 56–67.
- [Luo+19] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma. "Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees". In: ICLR. 2019.
- [Lut16] J. Lutinen. "BayesPy: variational Bayesian inference in Python". In: JMLR (2016).
- [LUW17] C. Louizos, K. Ullrich, and M. Welling. "Bayesian Compression for Deep Learning". In: NIPS. 2017.
- [LV06] F. Liese and I. Vajda. "On divergences and informations in statistics and information theory". In: IEEE Transactions on Information Theory 52.10 (2006), pp. 4394–4412.
- [LV19] T. W. van de Laar and B. de Vries. "Simulating Active Inference Processes by Message Passing". In: Frontiers in Robotics and AI 6 (2019), p. 20.
- [LW04] H. Lopes and M. West. "Bayesian model assessment in factor analysis". In: Statistica Sinica 14 (2004), pp. 41–67.
- [LW16] C. Louizos and M. Welling. "Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors". In: ICML. 2016.
- [LW17] C. Louizos and M. Welling. "Multiplicative Normalizing Flows for Variational Bayesian Neural Networks". In: Proceedings of the 34th International Conference on Machine Learning. 2017, pp. 2218–2227.
- [LWS18] Z. C. Lipton, Y.-X. Wang, and A. Smola. "Detecting and Correcting for Label Shift with Black Box Predictors". In: ICML. 2018.
- [LY17] J. H. Lim and J. C. Ye. "Geometric gan". In: arXiv preprint arXiv:1705.02894 (2017).
- [Lyu11] S. Lyu. "Unifying non-maximum likelihood learning objectives with minimum KL contraction". In: Advances in Neural Information Processing Systems. 2011, pp. 64–72.
- [Lyu12] S. Lyu. "Interpretation and generalization of score matching". In: arXiv preprint arXiv:1205.2629 (2012).
- [LZ20] B. Lim and S. Zohren. "Time Series Forecasting With Deep Learning: A Survey". In: (Apr. 2020). arXiv: 2004.13408 [stat.ML].
- [MA10] I. Murray and R. P. Adams. "Slice sampling covariance hyperparameters of latent Gaussian models". In: NIPS. 2010, pp. 1732–1740.
- [Maa+16] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. "Auxiliary Deep Generative Models". In: ICML. 2016.
- [Maa+19] L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. "BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling". In: NIPS. 2019.
- [Mac03] D. MacKay. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, 2003.
- [Mac+11] J. H. Macke, L. Büsing, J. P. Cunningham, B. M. Y. Ece, K. V. Shenoy, and M. Sahani. "Empirical models of spiking in neural populations". In: NIPS. 2011.
- [Mac+15] D. MacLaurin, D. Duvenaud, M. Johnson, and R. P. Adams. Autograd: Reverse-mode differentiation of native Python. 2015.
- [Mac+19] D. MacLaurin, A. Radul, M. J. Johnson, and D. Vytiniotis. "Dex: array programming with typed indices". In: NeurIPS workshop: Program Transformations for Machine Learning (2019).
- [Mac75] O. Macchi. "The coincidence approach to stochastic point processes". In: Advances in Applied Probability 7.1 (1975), pp. 83–122.
- [Mac92a] D. MacKay. "The evidence framework applied to classification networks". In: Neural Computation 4.5 (1992), pp. 720–736.
- [Mac92b] D. J. C. MacKay. "A Practical Bayesian Framework for Backpropagation Networks". In: Neural Comput. 4.3 (May 1992), pp. 448–472.
- [Mac95] D. MacKay. "Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks". In: Network: Computation in Neural Systems 6.3 (1995), pp. 469–505.
- [Mac98] D. MacKay. "Introduction to Gaussian Processes". In: Neural Networks and Machine Learning. Ed. by C. Bishop. 1998.
- [Mac99a] S. N. MacEachern. "Dependent nonparametric processes". In: ASA proceedings of the section on Bayesian statistical science. Vol. 1. Alexandria, Virginia. Virginia: American Statistical Association; 1999. 1999, pp. 50–53.
- [Mac99b] D. MacKay. "Comparison of approximate methods for handling hyperparameters". In: Neural Computation 11.5 (1999), pp. 1035–1068.
- [Mad+17] C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh. "Filtering Variational Objectives". In: NIPS. 2017.
- [Mad+18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: ICLR. 2018.
- [Mad+19] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. "A Simple Baseline for Bayesian Uncertainty in Deep Learning". In: NIPS. Curran Associates, Inc., 2019, pp. 13153–13164.
- [MAD20] W. R. Morningstar, A. A. Alemi, and J. V. Dillon. "PAC^m-Bayes: Narrowing the Empirical Risk Gap in the Misspecified Bayesian Regime". In: (Oct. 2020). arXiv: 2010.09629 [cs.LG].
- [Mah07] R. P. S. Mahler. Statistical Multisource-Multitarget Information Fusion. Norwood, MA, USA: Artech House, Inc., 2007.
- [Mah08] H. Mahmoud. Polya Urn Models. en. 1 edition. Chapman and Hall/CRC, June 2008.
- [Mah13] R. Mahler. "Statistics 102 for Multisource-Multitarget Detection and Tracking". In: IEEE J. Sel. Top. Signal Process. 7.3 (June 2013), pp. 376–389.
- [Mai+10] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. "Online learning for matrix factorization and sparse coding". In: JMLR 11 (2010), pp. 19–60.
- [Mai13] J. Mairal. "Stochastic Majorization-minimization Algorithms for Large-scale Optimization". In: NIPS. 2013, pp. 2283–2291.
- [Mai15] J. Mairal. "Incremental Majorization-Minimization Optimization with Application to Large-Scale Machine Learning". In: SIAM J. Optim. 25.2 (Jan. 2015), pp. 829–855.
- [Mak+15a] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. "Adversarial Autoencoders". In: arXiv preprint arXiv: 1511.05644 [cs.LG].
- [Mak+15b] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. "Adversarial autoencoders". In: arXiv preprint arXiv: 1511.05644 [cs.LG].
- [Mak+20] A. Makhzani, A. Taghvaei, S. Oh, and J. Lee. "Optimal transport mapping via input convex neural networks". In: International Conference on Machine Learning. PMLR. 2020, pp. 6672–6681.
- [Mal+17] D. M. Malioutov, K. R. Varshney, A. Emad, and S. Dash. "Learning interpretable classification rules with boolean compressed sensing". In: Transparent Data Mining for Big and Small Data. Springer, 2017, pp. 95–121.
- [Man] B. Mann. How many times should you shuffle a deck of cards? Tech. rep. Dartmouth.
- [Man07] V. Mansinghka, D. Roy, R. Rifkin, and J. Tenenbaum. "AClass: An online algorithm for generative classification". In: AISTATS. 2007.
- [Man19] D. J. Mankowitz, N. Levine, R. Jeong, Y. Shi, J. Kay, A. Abdolmaleki, J. T. Springenberg, T. Mann, T. Hester, and M. Riedmiller. "Robust Reinforcement Learning for Continuous Control with Model Misspecification". In: (June 2019). arXiv: 1906.07516 [cs.LG].
- [Man90] C. F. Manski. "Nonparametric Bounds on Treatment Effects". In: The American Economic Review 80.2 (1990), pp. 319–323.
- [Mao+17] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and S. P. Smolley. "Least Squares Generative Adversarial Networks". In: ICCV. 2017.
- [Mar03] B. Marlin. "Modeling User Rating Profiles for Collaborative Filtering". In: NIPS. 2003.
- [Mar+06] A. Margolin, I. Nememan, K. Basso, C. Wiggins, G. Stolovitzky, and R. F. Abd A. Califano. "ARACNE: An Al-

- 1
- 2 gorithm for the Reconstruction of Gene Regulatory Networks
in a Mammalian Cellular Context". In: *BMC Bioinformatics* 7
(2006).
- 3 [Mar08] B. Marlin. "Missing Data Problems in Machine Learning". PhD thesis. U. Toronto, 2008.
- 4 [Mar+10] B. M. Marlin, K. Swersky, B. Chen, and N. de Freitas. "Inductive Principles for Restricted Boltzmann Machine
Learning". In: *AISTATS*. 2010.
- 5 [Mar10a] J. Martens. "Deep learning via Hessian-free optimization". In: *ICML*. 2010.
- 6 [Mar10b] J. Martens. "Learning the Linear Dynamical System
with ASOS". In: *ICML*. ICML'10. Haifa, Israel: Omnipress,
2010, pp. 743–750.
- 7 [Mar16] J. Martens. "Second-order optimization for neural networks". PhD thesis. Toronto, 2016.
- 8 [Mar18] O. Martin. Bayesian analysis with Python. Packt,
2018.
- 9 [Mar20] J. Martens. "New insights and perspectives on the natural gradient method". In: *JMLR* (2020).
- 10 [Mar21] J. Marino. "Predictive Coding, Variational Autoencoders, and Biological Connections". en. In: *Neural Comput.* 34.1 (Dec. 2021), pp. 1–44.
- 11 [Mas+20] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer. "Class-incremental learning:
survey and performance evaluation on image classification". In:
(Oct. 2020). arXiv: 2010.15277 [cs.LG].
- 12 [Mat14] C. Mattfeld. "Implementing spectral methods for hidden Markov models with real-valued emissions". MA thesis. ETH Zurich, Apr. 2014.
- 13 [Mat+16] A. Matthews, J. Hensman, R. Turner, and Z. Ghahramani. "On Sparse Variational Methods and the Kullback-Leibler Divergence between Stochastic Processes". en. In: *AISTATS*. May 2016, pp. 231–239.
- 14 [Mav16] Mavrogiatou, L And Vyshevskiy. "Sequential Importance Sampling for Online Bayesian Changepoint Detection". In: 22nd International Conference on Computational Statistics. 2016.
- 15 [Mav17] D. Molchanov, A. Ashukha, and D. Vetrov. "Variational Dropout Sparsifies Deep Neural Networks". In: *ICML*. 2017.
- 16 [May79] P. Maybeck. Stochastic models, estimation, and control. Academic Press, 1979.
- 17 [MAZA18] X. B. P. and Marcin Andrychowicz, W. Zaremba, and P. Abbeel. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *ICRA*. 2018, pp. 1–8.
- 18 [MB16] Y. Miao and P. Blunsom. "Language as a Latent Variable: Discrete Generative Models for Sentence Compression". In: *EMNLP*. 2016.
- 19 [MB18] A. Mensch and M. Blondel. "Differentiable Dynamic Programming for Structured Prediction and Attention". In: *ICML*. 2018.
- 20 [MB21] M. Y. Michelin and Q. Becker. "On Linear Interpolation in the Latent Space of Deep Generative Models". In: *ICLR Workshop on Geometrical and Topological Representation Learning*. May 2021.
- 21 [MB88] T. Mitchell and J. Beauchamp. "Bayesian Variable Selection in Linear Regression". In: *JASA* 83 (1988), pp. 1023–1036.
- 22 [MBJ06] J. McAuley, D. Blei, and M. Jordan. "Nonparametric empirical Bayes for the Dirichlet process mixture model". In: *Statistics and Computing* 16.1 (2006), pp. 5–14.
- 23 [MBJ20] T. M. Moerland, J. Broekens, and C. M. Jonker. "Model-based Reinforcement Learning: A Survey". In: (June 2020). arXiv: 2006.16712 [cs.LG].
- 24 [MBK20] X. Meng, R. Bachmann, and M. E. Khan. "Training Binary Neural Networks using the Bayesian Learning Rule". In: *ICML*. Feb. 2020.
- 25 [MBL20] B. Mirzaoleiman, J. Bilmes, and J. Leskovec. "Core-sets for data-efficient training of machine learning models". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6950–6960.
- 26 [MBW20] W. J. Maddox, G. Benton, and A. G. Wilson. "Rethinking Parameter Counting in Deep Models: Effective Dimensionality Revisited". In: arXiv preprint arXiv:2003.02139 (2020).
- 27 [MC19] P. Moreno Comellas. "Vision as inverse graphics for detailed scene understanding". en. PhD thesis. July 2019.
- 28 [McA99] D. A. McAllester. "PAC-Bayesian model averaging". In: Proceedings of the twelfth annual conference on computational learning theory. 1999.
- 29 [McC03] A. McCray. "An upper level ontology for the biomedical domain". In: *Comparative and Functional Genomics* 4 (2003), pp. 80–84.
- 30 [McG20] R. McElreath. Statistical Rethinking: A Bayesian Course with Examples in R and Stan (2nd edition). en. Chapman and Hall/CRC, 2020.
- 31 [McG54] W. McGill. "Multivariate information transmission". In: *Psychometrika* 19 (1954), pp. 97–116.
- 32 [McM+13] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. "Ad click prediction: a view from the trenches". In: *KDD*. 2013, pp. 1222–1230.
- 33 [Md+19] C. de Masson d'Autume, M. Rosca, J. Rae, and S. Mohamed. "Training language GANs from Scratch". In: (May 2019). arXiv: 1905.09922 [cs.CL].
- 34 [MD97] X. L. Meng and D. van Dyk. "The EM algorithm — an old folk song sung to a fast new tune (with Discussion)". In: *Royal Stat. Soc. B* 59 (1997), pp. 511–567.
- 35 [MDA15] D. Maclaurin, D. Duvenaud, and R. P. Adams. "Gradient-based Hyperparameter Optimization through Reversible Learning". In: *ICML*. 2015.
- 36 [MDM19] S. Mahloujifar, D. I. Diochnos, and M. Mahmoodi. "The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4536–4543.
- 37 [MDR94] S. Muggleton and L. De Raedt. "Inductive logic programming: Theory and methods". In: *The Journal of Logic Programming* 19 (1994), pp. 629–679.
- 38 [ME17] H. Mei and J. M. Eisner. "The neural hawkes process: A neurally self-modulating multivariate point process". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6754–6764.
- 39 [Med+21] M. A. Medina, J. L. M. Olea, C. Rush, and A. Velez. "On the Robustness to Misspecification of α -Posterior and Their Variational Approximations". In: (Apr. 2021). arXiv: 08324 [stat.ML].
- 40 [Mee+18] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. "An introduction to probabilistic programming. Foundations and Trends in Machine Learning".
- 41 [Mei18a] N. Meinshausen. "Causality from a distributional business point of view". In: 2018 IEEE Data Science Workshop (DSW). June 2018, pp. 6–10.
- 42 [Mei18b] N. Meinshausen. "CAUSALITY FROM A DISTRIBUTIONAL ROBUSTNESS POINT OF VIEW". In: 2018 IEEE Data Science Workshop (DSW). 2018, pp. 6–10.
- 43 [Mer] Definition of interpret. 2022. url: <https://www.merriam-webster.com/dictionary/interpret>.
- 44 [Mer+00] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan. "The Unscented Particle Filter". In: *NIPS*-13. 2000.
- 45 [Met16] C. Metz. In Two Moves, AlphaGo and Lee Sedol Redefine the Future. 2016. url: <https://www.wired.com/2016/03/two-moves-alphago-lee-sedol-redefined-future/> (visited on 01/07/2022).
- 46 [Met+16] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. "Unrolled Generative Adversarial Networks". In: (2016).
- 47 [Met+17] L. Metz, J. Ibarz, N. Jaitly, and J. Davidson. "Discrete Sequential Prediction of Continuous Actions for Deep RL". In: (May 2017). arXiv: 1705.05035 [cs.LG].
- 48 [Met+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. "Equation of state calculations by fast computing machines". In: *J. of Chemical Physics* 21 (1953), pp. 1087–1092.
- 49 [Mey+18] F. Meyer, T. Kropf, J. Williams, R. Lau, F. Hlawatsch, P. Braca, and M. Win. "Message passing algorithms for scalable multitarget tracking". In: *Proc. IEEE* 106.2 (2018).
- 50 [Mey+21] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodru. "Hutch++: Optimal Stochastic Trace Estimation". In: *SIAM Symposium on Simplicity in Algorithms (SOSA21)*. 2021.
- 51 [Mey22] S. Meyn. Control Systems and Reinforcement Learning. Cambridge, 2022.
- 52 [MG15] J. Martens and R. Grosse. "Optimizing Neural Networks with Kronecker-factored Approximate Curvature". In: *ICML*. 2015.
- 53 [GMG06] I. Murray, Z. Ghahramani, and D. J. C. MacKay. "MCMC for doubly-intractable distributions". In: *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*. AUAI Press, 2006, pp. 359–366.
- 54 [MGN18a] L. Mescheder, A. Geiger, and S. Nowozin. "Which Training Methods for GANs do actually Converge?". In: *ICML*. 2018.
- 55 [MGN18b] L. Mescheder, A. Geiger, and S. Nowozin. "Which training methods for GANs do actually converge?". In: *International conference on machine learning*. PMLR. 2018, pp. 3481–3490.
- 56 [MGR18] H. Mania, A. Guy, and B. Recht. "Simple random search of static linear policies is competitive for reinforcement learning". In: *NIPS*. Ed. by S. Bengio, H. Wallach, H. Larochelle, 2104.

BIBLIOGRAPHY

- 1
- K Grauman, N Cesa-Bianchi, and R Garnett. Curran Associates, Inc., 2018, pp. 1800–1809.
- 2
- [MH12] R. Mazumder and T. Hastie. The Graphical Lasso: New Insights and Alternatives. Tech. rep. Stanford Dept. Statistics, 2012.
- 3
- [MH14] J. W. Miller and M. T. Harrison. “Inconsistency of Pitman-Yor process mixtures for the number of components”. In: JMLR 15.1 (2014), pp. 3333–3370.
- 4
- [MH20] I. Mordatch and J. Harwick. ICML tutorial on model-based methods in reinforcement learning. <https://sites.google.com/corp/view/mbr-tutorial>. 2020.
- 5
- [MHB17] S. Mandt, M. D. Hoffman, and D. M. Blei. “Stochastic Gradient Descent As Approximate Bayesian Inference”. In: JMLR 18.1 (Jan. 2017), pp. 4873–4907.
- 6
- [MHN14] F. Meyer, O. Hlinka, and F. Hlawatsch. “Sigma point belief propagation”. In: IEEE Signal Processing Letters 21.2 (2014), pp. 145–149.
- 7
- [MHN13] A. L. Maas, A. Y. Hannun, and A. Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: ICML. Vol. 28. 2013.
- 8
- [MHS03] J. R. Movellan, J. Hershey, and J. Susskind. “Large-scale convolutional HMMs for real-time video tracking”. 2003.
- 9
- [Mid+19] L. Middleton, G. Deligiannidis, A. Doucet, and P. E. Jacob. “Unbiased Smoothing using Particle Independent Metropolis-Hastings”. In: AISTATS. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2378–2387.
- 10
- [Mil+13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed representations of words and phrases and their compositionality”. In: NIPS. 2013, pp. 3111–3119.
- 11
- [Mil+05] B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov. “BLOG: Probabilistic Models with Unknown Objects”. In: IJCAI. 2005.
- 12
- [Mil19a] T. Miller. “Explanation in artificial intelligence: Insights from the social sciences”. In: Artificial intelligence 267 (2019), pp. 1–38.
- 13
- [Mil19b] B. Millidge. “Deep Active Inference as Variational Policy Gradients”. In: J. Math. Psychol. (July 2019).
- 14
- [Mil+20] B. Millidge, A. Tschantz, A. K. Seth, and C. L. Buckley. “On the Relationship Between Active Inference and Control as Inference”. In: International Workshop on Active Inference. June 2020.
- 15
- [Mil21] B. Millidge. “Applications of the Free Energy Principle to Machine Learning and Neuroscience”. PhD thesis. U. Edinburgh, June 2021.
- 16
- [Mil+21] B. Millidge, A. Tschantz, A. Seth, and C. Buckley. “Neural Kalman Filtering”. In: (Feb. 2021). arXiv: [2102.10021](https://arxiv.org/abs/2102.10021) [cs.NE].
- 17
- [Mil+21a] J. P. Miller, R. Taori, A. Raghunathan, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon, and L. Schmidt. “Accuracy on the Line: On the Strong Correlation Between Out-of-Distribution and In-Distribution Generalization”. In: ICML. Ed. by M. Mella and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 7721–7735.
- 18
- [Min00a] T. Minka. Bayesian linear regression. Tech. rep. MIT, 2000.
- 19
- [Min00b] T. Minka. Bayesian model averaging is not model combination. Tech. rep. MIT Media Lab, 2000.
- 20
- [Min00c] T. Minka. Estimating a Dirichlet distribution. Tech. rep. MIT, 2000.
- 21
- [Min01a] T. Minka. “A family of algorithms for approximate Bayesian inference”. PhD thesis. MIT, 2001.
- 22
- [Min01b] T. Minka. “Expectation Propagation for approximate Bayesian inference”. In: UAI. 2001.
- 23
- [Min04] T. Minka. Power EP. Tech. rep. MSR-TR-2004-149. 2004.
- 24
- [Min05] T. Minka. Divergence measures and message passing. Tech. rep. MSR Cambridge, 2005.
- 25
- [Min+18] T. Minka, J. Winn, J. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. Infer.NET 0.3. Microsoft Research Cambridge, 2018.
- 26
- [Min78] M. Minoux. “Accelerated greedy algorithms for maximizing submodular set functions”. In: Optimization Techniques. Ed. by J. Stoer. Vol. 7. Lecture Notes in Control and Information Sciences. 10.1007/BFb0006528. Springer Berlin / Heidelberg, 1978, pp. 234–243.
- 27
- [Mis+18] A. Mishkin, F. Kunstner, D. Nielsen, M. Schmidt, and M. E. Khan. “SLANG: Fast Structured Covariance Approximations for Bayesian Deep Learning with Natural Gradient”. In: NIPS. Curran Associates, Inc., 2018, pp. 6245–6255.
- 28
- [Mit+19] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru.
- 29
- “Model cards for model reporting”. In: Proceedings of the conference on fairness, accountability, and transparency. 2019, pp. 220–229.
- 30
- [Mit+20] J. Mitrovic, B. McWilliams, J. Walker, L. Buesing, and C. Blundell. Representation Learning via Invariant Causal Mechanisms. 2020. arXiv: [2010.07922](https://arxiv.org/abs/2010.07922) [cs.LG].
- 31
- [Miy+18a] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks”. In: ICLR. 2018.
- 32
- [Miy+18b] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks”. In: arXiv. 2018.
- 33
- [Miy+18c] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. “Spectral Normalization for Generative Adversarial Networks”. In: International Conference on Learning Representations. 2018.
- 34
- [Mj97] M. Meila and M. Jordan. Triangulation by continuous embedding. Tech. rep. 1605. MIT AI Lab, 1997.
- 35
- [MK05] J. Mooij and H. Kappen. “Sufficient conditions for convergence of loopy belief propagation”. In: UAI. 2005.
- 36
- [MK18] T. Miyato and M. Koyama. “cGANs with Projection Discriminator”. In: International Conference on Learning Representations. 2018.
- 37
- [MK19] J. Menick and N. Kalchbrenner. “Generating high-quality images with subscale pixel networks and multidimensional upscaling”. In: ICLR. 2019.
- 38
- [MK97] G. J. McLachlan and T. Krishnan. The EM Algorithm and Extensions. Wiley, 1997.
- 39
- [MKG21] W. J. Ma, K. Kording, and D. Goldreich. Bayesian models of perception and action. MIT Press, 2021.
- 40
- [MKH19] R. Müller, S. Kornblith, and G. E. Hinton. “When does label smoothing help?”. In: NIPS. 2019, pp. 4694–4703.
- 41
- [MLK11] O. Martin, R. Kumar, and J. Lao. Bayesian Modeling and Computation in Python. CRC Press, 2011.
- 42
- [MKS21] K. Murphy, A. Kumar, and S. Serghiu. “Risk score learning for COVID-19 contact tracing apps”. In: Machine Learning for Healthcare. Apr. 2021.
- 43
- [ML02] T. Minka and J. Laerty. “Expectation-propagation for the Generative Aspect Model”. In: UAI. Morgan Kaufmann Publishers Inc., 2002, pp. 352–359.
- 44
- [ML16] S. Mohamed and B. Lakshminarayanan. “Learning in Implicit Generative Models”. In: (2016). arXiv: [1610.03483](https://arxiv.org/abs/1610.03483) [stat.ML].
- 45
- [MLN19] P. Michel, O. Levy, and G. Neubig. “Are Sixteen Heads Really Better than One?”. In: NIPS. 2019.
- 46
- [MLW19] V. Masrani, T. A. Le, and F. Wood. “The Thermodynamic Variational Objective”. In: NIPS. Curran Associates, Inc., 2019, pp. 11521–11530.
- 47
- [MM01] T. K. Marks and J. R. Movellan. Division networks, products of experts, and factor analysis. Tech. rep. University of California San Diego, 2001.
- 48
- [MM93] D. Q. Mayne and H. Michalska. “Receding horizon control of nonlinear systems”. In: IEEE Trans. Automat. Contr. 35 (July 1990), pp. 814–824.
- 49
- [MMC98] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng. “Turbo decoding as an instance of Pearl’s ‘belief propagation’ algorithm”. In: IEEE J. on Selected Areas in Comm. 16.2 (1998), pp. 140–152.
- 50
- [MMR13] L. Malago, M. Matteucci, and G. Pistoni. “Natural gradient, fitness modelling and model selection: A unifying perspective”. In: IEEE Congress on Evolutionary Computation. 2013.
- 51
- [MMP87] J. Marroquin, S. Mitter, and T. Poggio. “Probabilistic solution of ill-posed problems in computational vision”. In: JASA 82.297 (1987), pp. 76–89.
- 52
- [MMT17] C. J. Maddison, A. Mnih, and Y. W. Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”. In: ICLR. 2017.
- 53
- [MN89] P. McCullagh and J. Nelder. Generalized linear models. 2nd edition. Chapman and Hall, 1989.
- 54
- [MNG17a] L. Mescheder, S. Nowozin, and A. Geiger. “Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks”. In: International Conference on Machine Learning. PMLR, 2017, pp. 2391–2400.
- 55
- [MNG17b] L. Mescheder, S. Nowozin, and A. Geiger. “The numerics of gans”. In: Advances in Neural Information Processing Systems. 2017, pp. 1825–1835.
- 56
- [Mni+15] V. Mnih et al. “Human-level control through deep reinforcement learning”. In: Nature 518.7540 (2015), pp. 529–533.
- 57
- [Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. “Asynchronous Methods for Deep Reinforcement Learning”. In: ICML. 2016.
- 58
- [MO14] M. Mirza and S. Osindero. “Conditional generative adversarial nets”. In: arXiv preprint arXiv:1411.1784 (2014).

- 1
- 2 [Mob16] H. Mobahi. "Closed Form for Some Gaussian Convolutions". In: (Feb. 2016). arXiv: [1602.05610 \[math.CA\]](https://arxiv.org/abs/1602.05610).
- 3 [Moc+96] J. Mockus, W. Eddy, A. Mockus, L. Mockus, and G. Reklaitis. Bayesian Heuristic Approach to Discrete and Global Optimization: Algorithms, Visualization, Software, and Applications. Kluwer, 1996.
- 4 [Moh+19a] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. "Monte Carlo Gradient Estimation in Machine Learning". In: (June 2019). arXiv: [1906.10652 \[stat.ML\]](https://arxiv.org/abs/1906.10652).
- 5 [Moh+19b] H. Mohammadi, P. Challenor, M. Goodfellow, and D. Williamson. "Emulating computer models with step-discontinuous outputs using Gaussian processes". In: (Mar. 2019). arXiv: [1903.02071 \[stat.ME\]](https://arxiv.org/abs/1903.02071).
- 6 [Mon81] G. Monge. "Mémoire sur la théorie des déblais et des remblais". In: Histoire de l'Académie Royale des Sciences (1781), pp. 666–704.
- 7 [Mor+21] W. Morningstar, C. Ham, A. Gallagher, B. Lakshminarayanan, A. Alemi, and J. Dillon. "Density of States Estimation for Out of Distribution Detection". In: AISTATS. Ed. by A. Banerjee and K. Fukumizu, Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 3232–3240.
- 8 [Mor63] T. Morimoto. "Markov Processes and the H-Theorem". In: J. Phys. Soc. Jpn. 18.3 (Mar. 1963), pp. 328–331.
- 9 [MOT15] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going Deeper into Neural Networks. <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>. Accessed: NA-NA. 2015.
- 10 [Mov08] J. R. Movellan. "A minimum velocity approach to learning". In: unpublished draft, Jan. (2008).
- 11 [MP01] K. Murphy and M. Paskin. "Linear time inference in hierarchical HMMs". In: NIPS. 2001.
- 12 [MP21] D. Mazza and M. Pagani. "Automatic Differentiation in PCF". In: Proc. ACM Program. Lang. 5.POPL (Jan. 2021).
- 13 [MP95] D. MacKay and L. Petro. "A hierarchical dirichlet language model". In: Natural Language Engineering 1.3 (1995), pp. 289–307.
- 14 [MPS18] O. Mangoubi, N. S. Pillai, and A. Smith. "Does Hamiltonian Monte Carlo mix faster than a random walk on multimodal densities?". In: (Aug. 2018). arXiv: [1808.03230 \[math.PR\]](https://arxiv.org/abs/1808.03230).
- 15 [MR09] A. Melkumyan and F. Ramos. "A Sparse Covariance Function for Exact Gaussian Process Inference in Large Datasets". In: UCAI. 2009, pp. 1936–1942.
- 16 [MR10] B. Milch and S. Russell. "Extending Bayesian Networks to the Open-Universe Case". In: Heuristics, Probability and Causality: A Tribute to Judea Pearl. Ed. by R. Dechter, H. Geiger, and J. Y. Halpern. College Publications, 2010.
- 17 [MRW19] B. Mittelstadt, C. Russell, and S. Wachter. "Explaining explanations in AI". In: Proceedings of the conference on fairness, accountability, and transparency. 2019, pp. 279–288.
- 18 [MS96] V. Matveev and R. Shrock. "Complex-temperature singularities in Potts models on the square lattice". In: In: Phys. Rev. E. Stat. Phys. Plasmas Fluids Relat. Interdiscip. Topics 54.6 (Dec. 1996), pp. 6174–6185.
- 19 [MS99] C. Manning and H. Schütze. Foundations of statistical natural language processing. MIT Press, 1999.
- 20 [MSA18] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. "The M4 Competition: Results, findings, conclusion and way forward". In: Int. J. Forecast. 34.4 (Oct. 2018), pp. 802–808.
- 21 [MSB21] B. Millidge, A. Seth, and C. L. Buckley. "Predictive Coding: a Theoretical and Experimental Review". In: (July 2021). arXiv: [2107.12979 \[cs.AI\]](https://arxiv.org/abs/2107.12979).
- 22 [MT12] A. Mnih and Y. W. Teh. "A fast and simple algorithm for training neural probabilistic language models". In: ICML. 2012, pp. 419–426.
- 23 [MTB20] B. Millidge, A. Tschantz, and C. L. Buckley. "Predictive Coding Approximates Backprop along Arbitrary Computation Graphs". In: arXiv preprint arXiv:2006.04182 (2020).
- 24 [MTM14] C. J. Maddison, D. Tarlow, and T. Minka. "A* Sampling". In: NIPS. 2014.
- 25 [Muq+17] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf. "Kernel Mean Embedding of Distributions: A Review and Beyond". In: Foundations and Trends 10.1-2 (2017), pp. 1–141.
- 26 [Mua+20] K. Muandet, A. Mehrjou, S. K. Lee, and A. Raj. Dual Instrumental Variable Regression. 2020.
- 27 [Muk+18] S. Mukherjee, D. Shankar, A. Tathawadekar, P. Kompalli, S. Sarawagi, and K. Chaudhury. "ARMDN: Associative and Recurrent Mixture Density Networks for eRetail Demand Forecasting". In: (Mar. 2018). arXiv: [1803.03800 \[cs.LG\]](https://arxiv.org/abs/1803.03800).
- 28 [Mül+19a] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák. "Neural Importance Sampling". In: SIGGRAPH. 2019.
- 29 [Mül+19b] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák. "Neural importance sampling". In: ACM Transactions on Graphics 38.5 (2019), p. 145.
- 30 [Mun14] R. Munos. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning". In: Foundations and Trends in Machine Learning 7.1 (2014), pp. 1–129.
- 31 [Mun+16] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. "Safe and Efficient Q-Policy Reinforcement Learning". In: NIPS. 2016, pp. 1046–1054.
- 32 [Mun57] J. Munkres. "Algorithms for the assignment and transportation problems". In: Journal of the society for industrial and applied mathematics 5.1 (1957), pp. 32–38.
- 33 [Mur00] K. Murphy. "Bayesian Map Learning in Dynamic Environments". In: NIPS. Vol. 12. 2000.
- 34 [Mur02] K. Murphy. Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis. Dept. Computer Science, UC Berkeley, 2002.
- 35 [Mur+19] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. "Deonitions, methods, and applications in interpretable machine learning". In: Proceedings of the National Academy of Sciences 116.44 (2019), pp. 22071–22080.
- 36 [Mur22] K. P. Murphy. Probabilistic Machine Learning: An introduction. MIT Press, 2022.
- 37 [MW07] J. Møller and R. P. Waagepetersen. "Modern statistics for spatial point processes". In: Scandinavian Journal of Statistics 34.4 (2007), pp. 643–684.
- 38 [MW15] S. Morgan and C. Winship. Counterfactuals and Causal Inference. 2nd. Cambridge University Press, 2015.
- 39 [MW18] C. Matthews and J. Weare. "Langevin Markov Chain Monte Carlo with stochastic gradients". In: (May 2018). arXiv: [1805.08863 \[stat.ME\]](https://arxiv.org/abs/1805.08863).
- 40 [MYJ99] K. Murphy, Y. Weiss, and M. Jordan. "Loopy Belief Propagation for Approximate Inference: an Empirical Study". In: UAI. 1999.
- 41 [MYW18] J. Marino, Y. Yue, and S. Mandt. "Iterative Amortized Inference". In: ICML. 2018.
- 42 [Nac+17] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. "Bridging the Gap Between Value and Policy Based Reinforcement Learning". In: NIPS. 2017, pp. 2772–2782.
- 43 [Nac+18a] O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans. Algae: Policy Gradient from Arbitrary Experience. CoRR abs/1912.02074. 2019.
- 44 [Nac+19c] M. S. Nacson, J. D. Lee, S. Gunasekar, P. H. P. Avarese, N. Srivastava, and D. Soudry. "Convergence of Gradient Descent on Separable Data". In: AISTATS. 2019.
- 45 [Nad+19] S. Naderi, K. He, R. Aghajani, S. Sclaro, and P. Felzenszwalb. "Generalized Majorization-Minimization". In: ICML. 2019.
- 46 [Nae+18] C. A. Naeseth, S. W. Linderman, R. Ranganath, and D. M. Blei. "Variational Sequential Monte Carlo". In: AISTATS. 2018.
- 47 [Nak+19] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. "Deep Double Descent: Where Bigger Models and More Data Hurt". In: (Dec. 2019). arXiv: [1906.09161 \[cs.LG\]](https://arxiv.org/abs/1906.09161).
- 48 [Nal18] E. T. Nalisnick. "On Priors for Bayesian Neural Networks". PhD thesis. UC Irvine, 2018.
- 49 [Nal+19a] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. "Do Deep Generative Models Know What They Don't Know?". In: ICLR. 2019.
- 50 [Nal+19b] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. "Hybrid Models with Deep and Invertible Features". In: ICML. 2019, pp. 4723–4732.
- 51 [Nau04] J. Naudts. "Estimators, escort probabilities and ϕ -exponential families in statistical physics". In: J. of Inequalities in Pure and Applied Mathematics 5.4 (2004).
- 52 [NB05] M. Narasimhan and J. Bilmes. "A Submodular-Supermodular Procedure with Applications to Discriminative Structure Learning". In: Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI-2004). Edinburgh, Scotland: Morgan Kaufmann Publishers, 2005.
- 53 [NB06] M. Narasimhan and J. Bilmes. Learning Graphical Models over partial k -trees. Tech. rep. UWETR-2006-0001. University of Washington, Department of Electrical Engineering, 2006. <https://uwavevar.ece.uw.edu/techsite/papers/refer/UWETR-2006-0001.html>.
- 54 [NBS18] B. Neyshabur, S. Bhojanapalli, and N. Srebro. "A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks". In: ICLR. 2018.
- 55 [NCH15] M. Naeini, G. Cooper, and M. Hauskrecht. "Obtaining well calibrated probabilities using Bayesian binning". In: AAAI. 2015.

BIBLIOGRAPHY

- 1
- 2 [NCL20] T. Nguyen, Z. Chen, and J. Lee. "Dataset Meta-
Learning from Kernel Ridge-Regression". In: International
Conference on Learning Representations. 2020.
- 3 [NCT16a] S. Nowozin, B. Cseke, and R. Tomioka. "f-GAN:
Training Generative Neural Samplers using Variational Diver-
gence Minimization". In: NIPS. Ed. by D. D. Lee, M Sugiyama,
U. V. Luxburg, I Guyon, and R Garnett. Curran Associates,
Inc., 2016, pp. 271–279.
- 4 [NCT16b] S. Nowozin, B. Cseke, and R. Tomioka. "f-gan: Train-
ing generative neural samplers using variational divergence
minimization". In: NIPS. 2016, pp. 271–279.
- 5 [NCT16c] S. Nowozin, B. Cseke, and R. Tomioka. "f-gan: Train-
ing generative neural samplers using variational divergence
minimization". In: Advances in neural information processing
systems. 2016, pp. 271–279.
- 6 [NDL20] O. Nachum and B. Dai. "Reinforcement Learning via
Fenchel-Rockafellar Duality". In: (Jan. 2020). arXiv:
[cs.LG].
- 7 [NDL20] A. Nishimura, D. Dunson, and J. Lu. "Discontinuous
Hamiltonian Monte Carlo for discrete parameters and discon-
tinuous likelihoods". In: Biometrika (2020).
- 8 [Nea00] R. Neal. "Markov Chain Sampling Methods for Dirichlet
Process Mixture Models". In: JCGS 9.2 (2000), pp. 249–265.
- 9 [Nea01] R. M. Neal. "Annealed Importance Sampling". In:
Statistics and Computing 11 (2001), pp. 125–139.
- 10 [Nea03] R. Neal. "Slice sampling". In: Annals of Statistics 31.3
(2003), pp. 7–576.
- 11 [Nea+08] R. Neal et al. "Computing likelihood functions for
high-energy physics experiments when distributions are de ned
by simulators with nuisance parameters". In: (2008).
- 12 [Nea10] R. Neal. "MCMC using Hamiltonian Dynamics". In:
Handbook of Markov Chain Monte Carlo. Ed. by S. Brooks,
A. Gelman, G. Jones, and X.-L. Meng. Chapman & Hall, 2010.
- 13 [Nea11] R. M. Neal et al. "MCMC using Hamiltonian dynam-
ics". In: Handbook of markov chain monte carlo 2.11 (2011),
p. 2.
- 14 [Nea12] R. C. Neath. "On Convergence Properties of the Monte
Carlo EM Algorithm". In: arXiv [math.ST] (June 2012).
- 15 [Nea20] B. Neal. Introduction to Causal Inferencefrom a Ma-
chine Learning Perspective. 2020.
- 16 [Nea92] R. Neal. "Connectionist learning of belief networks". In:
Artifical Intelligence 56 (1992), pp. 71–113.
- 17 [Nea93] R. M. Neal. Probabilistic Inference using Markov Chain
Monte Carlo Methods. Tech. rep. CRG-TR-93-1, 144pp. Dept.
of Computer Science, University of Toronto, 1993.
- 18 [Nea95] R. M. Neal. "Bayesian Learning for Neural Networks".
PhD thesis. University of Toronto, 1995.
- 19 [Nea96] R. Neal. Bayesian learning for neural networks.
Springer, 1996.
- 20 [NeF+02] A. Ne an, L. Liang, X. Pi, X. Liu, and K. Murphy.
"Dynamic Bayesian Networks for Audio-Visual Speech Recog-
nition". In: J. Applied Signal Processing (2002).
- 21 [Neg+21] J. Negrea, J. Yang, H. Feng, D. M. Roy, and J. H. Hug-
gins. "Statistical inference with stochastic gradient algorithms".
In: arXiv preprint arXiv:1812.00279 (2018).
- 22 [Nei+18] D. Neil, J. Brody, A. Lacoste, A. Sim, P. Creed, and
A. Saari. "Interpretable graph convolutional neural networks
for inference on noisy knowledge graphs". In: arXiv preprint
arXiv:1812.00279 (2018).
- 23 [Nei21] Nelson Elhage and Neel Nanda and Catherine Olsson
and Tom Henighan and Nicholas Joseph and Ben Mann and
Amanda Askell and Yuntao Bai and Anna Chen and Tom Con-
nelly and Nova DasSarma and David Drury and Deep Ganguli
and Zachary Golens and James Hays and Jennifer Heiman and
Jackson Kamen and Liane Lovins and Kamal Ndousse and
Dario Amodei and Tom Brown and Jack Clark and Jared
Kaplan and Sam McCandlish and Chris Olah. A Mathemati-
cal Framework for Transformer Circuits. Tech. rep. Anthropic,
2021.
- 24 [Neu11] G. Neumann. "Variational Inference for Policy Search
in Changing Situations". In: ICML. 2011, pp. 817–824.
- 25 [Ney+17] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N.
Srebro. "Exploring generalization in deep learning". In: NIPS.
2017.
- 26 [NGO01] D. Nilsson and J. Goldberger. "Sequentially nding
the N-Best Lists in Hidden Markov Models". In: IJCAI. 2001,
pp. 1280–1285.
- 27 [Ngi+11] J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng. "Learn-
ing deep energy models". In: Proceedings of the 28th inter-
national conference on machine learning (ICML-11). 2011,
pp. 1105–1112.
- 28 [Ngu+16] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and
J. Clune. Synthesizing the preferred inputs for neurons in neu-
ral networks via deep generator networks. 2016. arXiv:
09304 [cs.NE].
- 29 [Ngu+18a] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner.
"Variational Continual Learning". In: ICLR. 2018.
- 30 [Ngu+18b] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner.
"Variational Continual Learning". In: ICLR. 2018.
- 31 [Ngu+19] T. T. Nguyen, Q. V. H. Nguyen, D. T. Nguyen, D. T.
Nguyen, Thien Huynh-The, S. Nahavandi, T. T. Nguyen, Q.-V.
Pham, and C. M. Nguyen. "Deep Learning for Deepfakes Cre-
ation and Detection: A Survey". In: (Sept. 2019). arXiv:
1909.11573 [cs.CV].
- 32 [Ngu+21] T. Nguyen, R. Novak, L. Xiao, and J. Lee. "Dataset
Distillation with Nitely Wide Convolutional Networks". In:
Advances in Neural Information Processing Systems. Ed. by A.
Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. 2021.
- 33 [NH98a] R. M. Neal and G. E. Hinton. "A new view of
EM algorithm that justis incremental and other variants". In:
Learning in Graphical Models. Ed. by M. Jordan. MIT Press,
1998.
- 34 [NH98b] R. M. Neal and G. E. Hinton. "A View of the EM Algo-
rithm that Justis Incremental, Sparse, and other Variants". In:
Learning in Graphical Models. Ed. by M. I. Jordan. Dor-
recht: Springer Netherlands, 1998, pp. 355–368.
- 35 [NHL19] E. Nalisnick, J. M. Hernández-Lobato, and P. Smyth.
"Dropout as a Structured Shrinkage Prior". In: ICML. 2019.
- 36 [NHR99] A. Ng, D. Harada, and S. Russell. "Policy invariance
under reward transformations: Theory and application to re-
ward shaping". In: ICML. 1999.
- 37 [NI92] H. Nagamochi and T. Ibaraki. "Computing edge-
connectivity of multigraphs and capacitated graphs". In: SIAM
J. Discrete Math. 5 (1992), pp. 54–66.
- 38 [Nic+21] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P.
Mishkin, B. McGrew, I. Sutskever, and M. Chen. "GLIDE: To-
wards Photo-realistic Image Generation and Editing with Text-
Guided Disuion Models". In: (Dec. 2021). arXiv:
[cs.CV].
- 39 [Nij+19] E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu. "Learn-
ing non-convergent non-persistent short-run MCMC toward
energy-based model". In: Advances in Neural Information Pro-
cessing Systems. 2019, pp. 5232–5242.
- 40 [Nil98] D. Nilsson. "A e cient algorithm for nding the M
most probable con gurations in a probabilistic expert system".
In: Statistics and Computing 8 (1998), pp. 159–173.
- 41 [Nix+19] J. Nixon, M. Dusenberry, L. Zhang, G. Jerfel, and
D. Tran. "Measuring Calibration in Deep Learning". In: (Apr.
2019). arXiv: 1904.01685 [cs.LG].
- 42 [NJ00] A. Y. Ng and M. Jordan. "PEGASUS: A policy search
method for large MDPs and POMDPs". In: UAI. 2000.
- 43 [NJB05] M. Narasimhan, N. Jojic, and J. A. Bilmes. "Q
clustering". In: Advances in Neural Information Processing
Systems 18 (2005), pp. 979–986.
- 44 [NKL17] V. Nagarajan and J. Z. Kolter. "Gradient descent GAN
optimization is locally stable". In: Advances in neural informa-
tion processing systems. 2017, pp. 5585–5595.
- 45 [NKL10] K. Negano, Y. Kawahara, and S. Iwata. "Minimum av-
erage cost clustering". In: Advances in Neural Information Pro-
cessing Systems 23 (2010), pp. 1759–1767.
- 46 [NKG03] K. Nummiaro, E. Koller-Meier, and L. V. Gool. "An
Adaptive Color-Based Particle Filter". In: Image and Vision
Computing 21.2 (2003), pp. 99–110.
- 47 [NLS15] C. A. Naesseth, F. Lindsten, and T. B. Schön. "Nested
Sequential Monte Carlo Methods". In: ICML. Feb. 2015.
- 48 [NLS19] C. A. Naesseth, F. Lindsten, and T. B. Schön. "Ele-
ments of Sequential Monte Carlo". In: Foundations and Trends
in Machine Learning (2019).
- 49 [Nen+12] A. Nenkova and K. McKeown. "A survey of text sum-
marization techniques". In: Mining text data. Springer, 2012,
pp. 43–76.
- 50 [NMC05] A. Niculescu-Mizil and R. Caruana. "Predicting Good
Probabilities with Supervised Learning". In: ICML. 2005.
- 51 [NNP19] W. Nie, N. Narodytska, and A. Patel. "RelGAN: Re-
lational Generative Adversarial Networks for Text Genera-
tions". In: International Conference on Learning Representa-
tions. 2019.
- 52 [Noé+19] F. Noé, S. Olsson, J. Köhler, and H. Wu. "Boltzmann
generators: Sampling equilibrium states of many-body systems
with deep learning". In: Science 365 (2019).
- 53 [Nou+02] M. N. Nounou, B. R. Bakshi, P. K. Goel, and X.
Shen. "Process modeling by Bayesian latent variable regres-
sion". In: Am. Inst. Chemical Engineers Journal 48.8 (Aug.
2002), pp. 1775–1793.
- 54 [Nov+19] R. Novak, L. Xiao, J. Lee, Y. Bahri, G. Yang, J. Hron,
D. A. Abola a, J. Pennington, and J. Sohl-Dickstein. "Bayesian
Deep Convolutional Networks with Many Channels are Gaus-
sian Processes". In: ICLR. 2019.
- 55 [Now+14] S. Nowozin, P. V. Gehler, J. Jancsary, and C. H. Lam-
pert, eds. Advanced Structured Prediction. en. The MIT Press,
Dec. 2014.
- 56 [2001 . 01866]
- 57 [2112 . 10741]

- 1
- 2 [NP03] W. K. Newey and J. L. Powell. "Instrumental variable
estimation of nonparametric models". In: *Econometrica* 71.5
(2003), pp. 1565–1578.
- 3 [NP12] N. Nunn and D. Puga. "Ruggedness: The blessing of bad
geography in Africa". In: *Rev. Econ. Stat.* 94.1 (2012).
- 4 [NR00a] A. Ng and S. Russell. "Algorithms for inverse reinforcement
learning". In: *ICML*. 2000.
- 5 [NR00b] A. Y. Ng and S. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proc. 17th International Conf. on Machine Learning*. Citeseer, 2000.
- 6 [NR08] H. Nickisch and C. E. Rasmussen. "Approximations for Binary Gaussian Process Classification". In: *JMLR* 9.Oct (2008), pp. 2035–2078.
- 7 [NR94] M. Newton and A. Raftery. "Approximate Bayesian Inference with the Weighted Likelihood Bootstrap". In: *J. of Royal Stat. Soc. Series B* 56.1 (1994), pp. 3–48.
- 8 [NS01] K. Nowicki and T. A. B. Snijders. "Estimation and Prediction for Stochastic Blockstructures". In: *Journal of the American Statistical Association* 96.455 (2001), pp. 1077–1087.
- 9 [NS17] E. Nalisnick and P. Smyth. "Variational Reference Priors". In: *ICLR* Workshop, 2017.
- 10 [NS18] E. Nalisnick and P. Smyth. "Learning Priors for Invariance". In: *AISTATS*. 2018.
- 11 [NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.
- 12 [NW20] X. Nie and S. Wagner. "Quasi-oracle estimation of heterogeneous treatment effects". In: *Biometrika* 108.2 (Sept. 2020), pp. 299–319. *print*: <https://academic.oup.com/biomet/article-pdf/108/2/299/37938939/asaa076.pdf>.
- 13 [NWF78] G. Nemhauser, L. Wolsey, and M. Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical Programming* 14.1 (1978), pp. 265–294.
- 14 [NWJ09] X. Nguyen, M. J. Wainwright, and M. I. Jordan. "On Surrogate Loss Functions and f-Divergences". In: *Ann. Stat.* 37.2 (2009), pp. 876–904.
- 15 [NWJ+09] X. Nguyen, M. J. Wainwright, M. I. Jordan, et al. "On surrogate loss functions and f-divergences". In: *The Annals of Statistics* 37.2 (2009), pp. 876–904.
- 16 [NWJ10a] X. Nguyen, M. J. Wainwright, and M. I. Jordan. "Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization". In: *IEEE Trans. Inf. Theory* 56.11 (Nov. 2010), pp. 5847–5861.
- 17 [NWJ10b] X. Nguyen, M. J. Wainwright, and M. I. Jordan. "Estimating divergence functionals and the likelihood ratio by convex risk minimization". In: *IEEE Transactions on Information Theory* 56.11 (2010), pp. 5847–5861.
- 18 [NY83] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983.
- 19 [NYC15] A. Nguyen, J. Yosinski, and J. Clune. "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images". In: *CVPR*. 2015.
- 20 [OA09] M. Opper and C. Archambeau. "The variational Gaussian approximation revisited". en. In: *Neural Comput.* 21.3 (Mar. 2009), pp. 786–792.
- 21 [OA21] S. W. Ober and L. Aitchison. "Global inducing point variational posteriors for Bayesian neural networks and deep Gaussian processes". In: *ICML*. 2021.
- 22 [Obe+19] F. Obermeyer, E. Bingham, M. Jankowiak, J. Chiu, N. Pradhan, A. Rush, and N. Goodman. "Tensor Variable Elimination for Partial Factor Graphs". In: *ICML*. 2019.
- 23 [OCM21] L. A. Ortega, R. Cabafas, and A. R. Masegosa. "Diversity and Generalization in Neural Network Ensembles". In: *(Oct. 2021)*. arXiv: [2110.13786 \[cs.LG\]](https://arxiv.org/abs/2110.13786).
- 24 [ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. "From HMMs to segment models: a unified view of stochastic modeling for speech recognition". In: *IEEE Trans. on Speech and Audio Processing* 4.5 (1996), pp. 360–378.
- 25 [OF96] B. A. Olshausen and D. J. Field. "Emergence of simple cell receptive field properties by learning a sparse code for natural images". In: *Nature* 381 (1996), pp. 607–609.
- 26 [O'H78] A. O'Hagan. "Curve Fitting and Optimal Design for Prediction". In: *J. of Royal Stat. Soc. Series B* 40 (1978), pp. 1–42.
- 27 [OKK16] A. Van den Oord, N. Kalchbrenner, and Kavukcuoglu. "Pixel Recurrent Neural Networks". In: *ICML*. 2016.
- 28 [Oll18] Y. Olivier. "Online natural gradient as a Kalman Iter". en. In: *Electron. J. Stat.* 12.2 (2018), pp. 2930–2961.
- 29 [OLV18] A. van den Oord, Y. Li, and O. Vinyals. "Representation Learning with Contrastive Predictive Coding". In: *(July 2018)*. arXiv: [1807.03748 \[cs.LG\]](https://arxiv.org/abs/1807.03748).
- 30 [OM96] P. V. Overschee and B. D. Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer Academic Publishers, 1996.
- 31 [OMS17] C. Olah, A. Mordvintsev, and L. Schubert. "Feature Visualization". In: *Distill* 2.11 (Nov. 2017).
- 32 [ONS18] V. M.-H. Ong, D. J. Nott, and M. S. Smith. "Gaussian Variational Approximation With a Factor Covariance Structure". In: *J. Comput. Graph. Stat.* 27.3 (July 2018), pp. 465–478.
- 33 [oor+16] A. Van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: *(Dec. 2016)*. arXiv: [1609.03499 \[cs.SD\]](https://arxiv.org/abs/1609.03499).
- 34 [Oor+16] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. "Conditional Image Generation with PixelCNN Decoders". In: *(2016)*. arXiv: [1605.05328 \[cs.CV\]](https://arxiv.org/abs/1605.05328).
- 35 [Oor+18] A. van den Oord et al. "Parallel WaveNet: Fast High-Fidelity Speech Synthesis". In: *ICML*. Ed. by J. Dy and A. Krause. Vol. 80. *Proceedings of Machine Learning Research*. Stockholm Sweden: PMLR, 2018, pp. 3918–3926.
- 36 [Oor+19] A. van den Oord, B. Poole, O. Vinyals, and A. Razavi. "Fixing Posterior Collapse with delta-VAEs". In: *ICLR*. 2019.
- 37 [OOS17] A. Odena, C. Olah, and J. Shlens. "Conditional image synthesis with auxiliary classifiers". In: *International conference on machine learning*. 2017, pp. 2642–2651.
- 38 [Opt88] Optimal information processing and Bayes' theorem. In: *The American Statistician* 42.4 (1988), pp. 278–280.
- 39 [ORo20] A. Owen and D. Rudolf. "A strong law of large numbers for scrambled net integration". In: *(Feb. 2020)*. arXiv: [2002.07859](https://arxiv.org/abs/2002.07859)
- 40 [Ore+20] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. "N-BEATS: Neural basis expansion for interpretable time series forecasting". In: *ICLR*. 2020.
- 41 [Ort+19] P. A. Ortega et al. "Meta-learning of Sequential Strategies". In: *(May 2019)*. arXiv: [1905.03030 \[cs.LG\]](https://arxiv.org/abs/1905.03030).
- 42 [Osa+18] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. "An Algorithmic Perspective on Imitation Learning". In: *Foundations and Trends in Robotics* 7.1–2 (2018), pp. 1–179.
- 43 [Osa+19a] K. Osawa, S. Swaroop, A. Jain, R. Eschenhagen, R. E. Turner, R. Yokota, and M. E. Khan. "Practical Deep Learning with Bayesian Principles". In: *NIPS*. 2019.
- 44 [Osa+19b] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota, and S. Matsuka. "Large-Scale Distributed Second-Order Optimization Using Kronecker-Factored Approximate Curvature for Deep Convolutional Neural Networks". In: *CVPR*. 2019.
- 45 [Osb16] I. Osband. "Risk versus Uncertainty in Deep Learning: Bayes, Bootstrap and the Dangers of Dropout". In: *NIPS workshop on Bayesian deep learning*. 2016.
- 46 [Osb+21] I. Osband, Z. Wen, S. M. Asghari, V. Dwarkartha, B. Hao, M. Ibrahim, D. Lawson, X. Lu, B. O'Donoghue, and B. Van Roy. "Evaluating Predictive Distributions: Does Bayesian Deep Learning Work?". In: *(Oct. 2021)*. arXiv: [2110.04629 \[cs.LG\]](https://arxiv.org/abs/2110.04629).
- 47 [Ose11] I. V. Oseledets. "Tensor-Train Decomposition". In: *SIAM J. Sci. Comput.* 33.5 (Jan. 2011), pp. 2295–2317.
- 48 [OT05] A. B. Owen and S. D. Tribble. "A quasi-Monte Carlo Metropolis algorithm". en. In: *PNAS* 102.25 (June 2005), pp. 8844–8849.
- 49 [OTT19] M. Okada, S. Takenaka, and T. Taniguchi. "Multi-person Pose Tracking using Sequential Monte Carlo with Probabilistic Neural Pose Predictor". In: *(Sept. 2019)*. arXiv: [1907.07031 \[cs.CV\]](https://arxiv.org/abs/1907.07031).
- 50 [Ova+19] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. V. Dillon, B. Lakshminarayanan, and J. Snoek. "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift". In: *NIPS*. 2019.
- 51 [OvK17] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. "Neural Discrete Representation Learning". In: *NIPS*. 2017.
- 52 [Owe13] A. B. Owen. Monte Carlo theory, methods and examples. 2013.
- 53 [Owe17] A. B. Owen. "A randomized Halton algorithm in R". In: *arXiv [stat.CO]* (June 2017).
- 54 [Oxl11] J. Oxley. *Matroid Theory*. Second Edition. Oxford University Press, 2011.
- 55 [Pac+14] J. Pacheco, S. Zu, M. Black, and E. Sudderth. "Preserving Modes and Messages via Diverse Particle Selection". en. In: *ICML*. Jan. 2014, pp. 1152–1160.
- 56 [Pai] Explainable AI in Practice Falls Short of Transparency Goals. <https://partnershiponai.org/xai-in-practice/>. Accessed: 2021-11-23.
- 57 [Pan+10] L. Paninski, Y. Ahmadian, D. G. Ferreira, S. Koyama, K. Rahnama Rad, M. Vidne, J. Vogelstein, and W. Wu. "A new

BIBLIOGRAPHY

- 1 look at state-space models for neural data". en. In: J. Comput.
 2 Neurosci. 29:1-2 (Aug. 2010), pp. 107–126.
- 3 [Pan+21] G. Pang, C. Shen, L. Cao, and A. Van Den Hengel.
 4 "Deep Learning for Anomaly Detection: A Review". In: ACM
 5 Comput. Surv. 54:2 (Mar. 2021), pp. 1–38.
- 6 [Pan+22] K. Pandey, A. Mukherjee, P. Rai, and A. Kumar. "DiffuseVAE: Efficient, Controllable and High-Fidelity Generation
 7 from Low-Dimensional Latents". In: (Jan. 2022). arXiv:
 8 2201.00308 [cs.LG].
- 9 [Pap+17] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z
 10 Berkay Celik, and A. Swami. "Practical Black-Box Attacks
 11 against Deep Learning Systems using Adversarial Examples".
 12 In: ACM Asia Conference on Computer and Communications
 13 Security. 2017.
- 14 [Pap+19] G. Papamakarios, E. Nalisnick, D. J. Rezende, S.
 15 Mohamed, and B. Lakshminarayanan. "Normalizing Flows for
 16 Probabilistic Modeling and Inference". In: (Dec. 2019). arXiv:
 17 1912.02762 [stat.ML].
- 18 [Par+19] S. Parameswaran, C. Deledalle, L. Denis, and T. Q.
 19 Nguyen. "Accelerating GMM-Based Patch Priors for Image
 20 Restoration: Three Ingredients for a 100x Speed-Up". In: IEEE
 21 Trans. Image Process. 28:2 (Feb. 2019), pp. 687–698.
- 22 [Par81] G. Parisi. "Correlation functions and computer simulations". In: Nuclear Physics B 180:3 (1981), pp. 378–384.
- 23 [Pas+02] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. "Identify Uncertainty and Citation Matching". In: NIPS.
 24 2002.
- 25 [Pas03] M. Paskin. "Thin Junction Tree Filters for Simultaneous
 26 Localization and Mapping". In: IJCAI. 2003.
- 27 [Pas+21a] A. Paszke, D. Johnson, D. Duvenaud, D. Vytisno-
 28 tis, A. Radul, M. Johnson, J. Ragan-Kelley, and D. Maclaurin.
 29 "Getting to the Point: Index Sets and Parallelism-Preserving
 30 Autodiff for Pointful Array Programming". In: Proc. ACM Pro-
 31 gram. Lang. 5.ICFP (Aug. 2021).
- 32 [Pas+21b] A. Paszke, M. J. Johnson, R. Frostig, and
 33 Maclaurin. "Parallelism-Preserving Automatic Differentiation
 34 for Second-Order Array Languages". In: Proceedings of the 9th
 35 ACM SIGPLAN International Workshop on Functional High-
 36 Performance and Numerical Computing, FHPNC 2021, Virtual
 37 Republic of Korea: Association for Computing Machinery,
 38 2021, 13–23.
- 39 [PB14] R. Pascanu and Y. Bengio. "Revisiting Natural Gradient
 40 for Deep Networks". In: ICLR. 2014.
- 41 [PBM16a] J. Peters, P. Bühlmann, and N. Meinshausen. "Causal
 42 inference by using invariant prediction: identifiability and con-
 43 fidence intervals". In: Journal of the Royal Statistical Society,
 44 Series B (Statistical Methodology) 78:5 (2016), pp. 947–1012.
- 45 [PBM16b] J. Peters, P. Bühlmann, and N. Meinshausen. "Causal
 46 inference using invariant prediction: identifiability and con-
 47 fidence intervals". In: J. of Royal Stat. Soc. Series B 78:5 (2016),
 48 pp. 947–1012.
- 49 [PC08] T. Park and G. Casella. "The Bayesian Lasso". In: JASA
 50 103:482 (2008), pp. 681–686.
- 51 [PC09] J. Paisley and L. Carin. "Nonparametric Factor Analysis
 52 with Beta Process Priors". In: ICML. 2009.
- 53 [PC12] N. Pinto and D. D. Cox. "High-throughput-derived
 54 biologically-inspired features for unconstrained face recogni-
 55 tion". In: Image Vis. Comput. 30:3 (Mar. 2012), pp. 159–168.
- 56 [PC21] G. Pleiss and J. P. Cunningham. "The Limitations of
 57 Large Width in Neural Networks: A Deep Gaussian Process
 58 Perspective". In: NIPS. June 2021.
- 59 [PD03] J. D. Park and A. Darwiche. "A Differential Semantics
 60 for Jointree Algorithms". In: NIPS. MIT Press, 2003, pp. 801–
 61 808.
- 62 [PD11] H. Poon and P. Domingos. "Sum-Product
 63 works: A New Deep Architecture". In: UAI. Java code
 64 at <http://alchemy.cs.washington.edu/spn/>. Short intro at
 65 <http://lessoned.blogspot.com/2011/10/intro-to-sum-product->
 66 networks.html. 2011.
- 67 [Pdc20] F.-P. Paty, A. d'Aspremont, and M. Cuturi. "Regula-
 68 rity as Regularization: Smooth and Strongly Convex Brenier
 69 Potentials in Optimal Transport". In: Proceedings of the Twenty
 70 Third International Conference on Artificial Intelligence and
 71 Statistics. Ed. by S. Chiappa and R. Calandra. Vol. 108. Pro-
 72 ceedings of Machine Learning Research, PMLR, 2020, pp. 1222–
 73 1232.
- 74 [PDL+12] M. Parry, A. P. Dawid, S. Lauritzen, et al. "Proper
 75 local scoring rules". In: The Annals of Statistics 40:1 (2012),
 76 pp. 561–592.
- 77 [PE16] V. Papyan and M. Elad. "Multi-Scale Patch-Based Image
 78 Restoration". en. In: IEEE Trans. Image Process. 25:1 (Jan.
 79 2016), pp. 249–261.
- 80 [Pea09a] J. Pearl. Causality. 2nd. Cambridge University Press,
 81 2009.
- 82 [Pea09b] J. Pearl. Causality: Models, Reasoning and Inference
 83 (Second Edition). Cambridge Univ. Press, 2009.
- 84 [Pea09c] J. Pearl. "Causal inference in statistics: An overview".
 85 In: Stat. Surv. 3:0 (2009), pp. 96–146.
- 86 [Pea12] J. Pearl. "The Causal Foundations of Structural Equa-
 87 tion Modeling". In: Handbook of structural equation modeling.
 88 Ed. by R. H. Hoyle. Vol. 68. 2012.
- 89 [Pea19] J. Pearl. "The Seven Tools of Causal Inference, with Re-
 90 actions on Machine Learning". In: Comm. of the ACM 62:3
 91 (Mar. 2019), pp. 54–60.
- 92 [Pea36] K. Pearson. "Method of moments and method of maxi-
 93 mum likelihood". In: Biometrika 28:1/2 (1936), pp. 34–59.
- 94 [Pea84] J. Pearl. Heuristics: Intelligent Search Strategies for
 95 Computer Problem Solving. Boston, MA, USA: Addison-Wesley
 96 Longman Publishing Co., Inc., 1984.
- 97 [Pea88] J. Pearl. Probabilistic Reasoning in Intelligent Systems:
 98 Networks of Plausible Inference. Morgan Kaufmann, 1988.
- 99 [Pea94] B. A. Pearlmutter. "Fast Exact Multiplication by the
 100 Hessian". In: Neural Comput. 6:1 (Jan. 1994), pp. 147–160.
- 101 [Peh+20] R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina,
 102 M. Trapp, G. Van den Broeck, K. Kersting, and Z. Ghahramani.
 103 "Einsun Networks: Fast and Scalable Learning of Tractable
 104 Probabilistic Circuits". In: (Apr. 2020). arXiv:
 105 2004.06231 [cs.LG].
- 106 [Pen13] J. Pena. "Reading dependencies from covariance
 107 graphs". In: Int. J. of Approximate Reasoning 54:1 (2013).
- 108 [Per+18] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A.
 109 Courville. "FiLM: Visual Reasoning with a General Condition-
 110 ing Layer". In: AAAI. 2018.
- 111 [Pes+21] H. Pesonen et al. "ABC of the Future". In: (Dec. 2021).
 112 arXiv: 2112.12841 [stat.AP].
- 113 [Pey20] G. Peyré. "Course notes on Optimization for Machine
 114 Learning". 2020.
- 115 [PF03] G. V. Puskorius and L. A. Feldkamp. "Parameter-
 116 based Kalman Iter training: Theory and implementation". In:
 117 Kalman Filtering and Neural Networks. New York, USA: John
 118 Wiley & Sons, Inc., 2003, pp. 23–67.
- 119 [PF91] G. V. Puskorius and L. A. Feldkamp. "Decoupled ex-
 120 tended Kalman Iter training of feedforward layered networks".
 121 In: International Joint Conference on Neural Networks. Vol. 1.
 122 July 1991, 771–777 vol.1.
- 123 [PFW21] R. Prado, M. Ferreira, and M. West. Time Series:
 124 Modelling, Computation and Inference (2nd ed.). CRC Press,
 125 2021.
- 126 [PG98] M. Popescu and P. D. Gader. "Image content retrieval
 127 from image databases using feature integration by Choquet
 128 integral". In: Storage and Retrieval for Image and Video
 129 Databases. VII, Ed. by C. M. Yeung, B.-L. Yeo, and C. A.
 130 Bouman. Vol. 3656. International Society for Optics and Pho-
 131 tonics SPIE, 1998, pp. 552–560.
- 132 [PGJ16] J. Pearl, M. Glymour, and N. Jewell. Causal inference
 133 in statistics: a primer. Wiley, 2016.
- 134 [PHDVi19] M. F. Pradier, M. C. Hughes, and F. Doshi-Velez.
 135 "Challenges in Computing and Optimizing Upper Bounds of
 136 Marginal Likelihood based on Chi-Square Divergences". In:
 137 AABI Symposium. 2019.
- 138 [Phu+18] M. Phuong, M. Welling, N. Kushman, R. Tomioka,
 139 and S. Nowozin. "The Mutual Autoencoder: Controlling Infor-
 140 mation in Latent Code Representations". In: Arxiv (2018).
- 141 [Pir+13] M. Pirotta, M. Restelli, A. Pecorino, and D. Calan-
 142 driello. "Safe Policy Iteration". In: ICML. 3 (2013), pp. 307–
 143 317.
- 144 [PJ17] J. Peters, D. Janzing, and B. Schölkopf. Elements
 145 of Causal Inference: Foundations and Learning Algorithms
 146 (Adaptive Computation and Machine Learning series). The
 147 MIT Press, Nov. 2017.
- 148 [PKP21] A. Plaat, W. Kosters, and M. Preuss. "High-Accuracy
 149 Model-Based Reinforcement Learning, a Survey". In: (July
 150 2021). arXiv: 2107.08241 [cs.LG].
- 151 [PL03] M. A. Paskin, G. D. Lawrence. Junction Tree
 152 Algorithms for Solving Sparse Linear Systems. Tech. rep.
 153 UCB/CSD-03-1271. UC Berkeley, 2003.
- 154 [Pla00] J. Platt. "Probabilities for SV machines". In: Advances
 155 in Large Margin Classifiers. Ed. by A. Smola, P. Bartlett, B.
 156 Schoelkopf, and D. Schuurmans. MIT Press, 2000.
- 157 [Ple+18] G. Pleiss, J. R. Gardner, K. Q. Weinberger, and A. G.
 158 Wilson. "Constant-Time Predictive Distributions for Gaussian
 159 Processes". In: International Conference on Machine Learning.
- 160 [Plu+20] G. Plumb, M. Al-Shedivat, Á. A. Cabrera, A. Perer,
 161 E. Xing, and A. Talwalkar. "Regularizing black-box models for
 162 improved interpretability". In: Advances in Neural Information
 163 Processing Systems 33 (2020).
- 164 [PM18a] N. Papernot and P. McDaniel. Deep k-Nearest Neigh-
 165 bors: Towards Compact, Interpretable and Robust Deep
 166 Learning. 2018. arXiv: 1803.04765 [cs.LG].

- 1
- 2 [PM18b] J. Pearl and D. Mackenzie. *The book of why: the new science of cause and effect*. 2018.
- 3 [PMT18] G. Plumb, D. Molitor, and A. Talwalkar. "Supervised Local Modeling for Interpretability". In: CoRR abs/1807.02910 (2018). arXiv: 1807.02910.
- 4 [Pol19] A. A. Pol, V. Berger, G. Cerninara, C. Germain, and M. Pierini. "Anomaly Detection With Conditional Variational Autoencoders". In: IEEE International Conference on Machine Learning and Applications. 2019.
- 5 [Pom89] D. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: NIPS. 1989, pp. 305–313.
- 6 [Poo+19] B. Poole, S. Ozair, A. van den Oord, A. A. Alemi, and G. Tucker. "On variational lower bounds of mutual information". In: ICML. 2019.
- 7 [Pou04] M. Pourahmadi. *Simultaneous Modelling of Covariance Matrices: GLM, Bayesian and Nonparametric Perspectives*. Tech rep. Northern Illinois University. 2004.
- 8 [Pou+20] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, and X.-Z. Wang. "A Review of Generalized Zero-Shot Learning Methods". In: (Nov. 2020). arXiv: 2011.08641 [cs.CV].
- 9 [Poy+20] R. Poyiadzi, K. Sokol, R. Santos-Rodriguez, T. De Bie, and P. Flach. "FACE: Feasible and actionable counterfactual explanations". In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society. 2020, pp. 344–350.
- 10 [PPC09] G. Petris, S. Petrone, and P. Campagnoli. *Dynamic linear models with R*. Springer, 2009.
- 11 [PPG91] C. S. Pomerleau, O. F. Pomerleau, and A. W. Garcia. "Biobehavioral research on nicotine use in women". In: British Journal of Addiction 86.5 (1991), pp. 527–531.
- 12 [PPM17] G. Papamakarios, T. Pavlakou, and I. Murray. "Masked Autoregressive Flow for Density Estimation". In: NIPS. 2017.
- 13 [PPS18] T. Pierrot, N. Perrin, and O. Sigaud. "First-order and second-order variants of the gradient descent in a unified framework". In: (Oct. 2018). arXiv: 1810.08102 [cs.LG].
- 14 [PR03] O. Papaspiliopoulos and G. O. Roberts. "Non-Centered Parameterisations for Hierarchical Models and Data Augmentation". In: Bayesian Statistics 7 (2003), pp. 307–326.
- 15 [Pra+18] S. Prabhumoye, Y. Tsvetkov, R. Salakhutdinov, and A. W. Black. "Style Transfer Through Back-Translation". In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2018, pp. 866–876.
- 16 [Pre05] S. J. Press. *Applied multivariate analysis, using Bayesian and frequentist methods of inference*. Second edition. Dover, 2005.
- 17 [Pre+17] O. Press, A. Bar, B. Bogin, J. Berant, Wolf, "Language generation with recurrent generative adversarial networks without pre-training". In: arXiv preprint arXiv:1706.01399 (2017).
- 18 [Pre+88] W. Press, B. Vetterling, S. Teukolsky, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Second. Cambridge University Press, 1988.
- 19 [Prz58] R. Price. "A useful theorem for nonlinear devices having Gaussian inputs". In: IRE Trans. Info. Theory 4.2 (1958), pp. 69–72.
- 20 [PS07] J. Peters and S. Schaal. "Reinforcement Learning by Reward-Weighted Regression for Operational Space Control". In: ICML 2007, pp. 745–750.
- 21 [PS08a] B. A. Pearlmutter and J. M. Siskind. "Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Back-propagator". In: ACM Trans. Program. Lang. Syst. 30.2 (Mar. 2008).
- 22 [PS08b] J. Peters and S. Schaal. "Reinforcement Learning of Motor Skills with Policy Gradients". In: Neural Networks 21.4 (2008), pp. 682–697.
- 23 [PS12] N. G. Polson and J. G. Scott. "On the Half-Cauchy Prior for a Global Scale Parameter". en. In: Bayesian Anal. 7.4 (Dec. 2012), pp. 887–902.
- 24 [PS17] N. G. Polson and V. Sokolov. "Deep Learning: Bayesian Perspective". en. In: Bayesian Anal. 12.4 (Dec. 2017), pp. 1275–1304.
- 25 [PSCD20] I. París, R. Sánchez-Caude, and F. J. Díez. "Sum-product networks: A survey". In: (Apr. 2020). arXiv: 2004.01167 [cs.LG].
- 26 [PSD00] J. K. Pritchard, M. Stephens, and P. Donnelly. "Inference of population structure using multilocus genotype data". In: Genetics 155.2 (June 2000), pp. 945–959.
- 27 [PSM19] G. Papamakarios, D. Sterratt, and I. Murray. "Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows". In: AISTATS. 2019.
- 28 [PSS00] D. Precup, R. S. Sutton, and S. P. Singh. "Eligibility Traces for Q-Policy Policy Evaluation". In: ICML. ICML '00.
- 29 [PTD20] A. Prabhu, P. H. S. Torr, and P. K. Dokania. "GDumb: A simple approach that questions our progress in continual learning". In: ECCV. Lecture notes in computer science. Cham: Springer International Publishing, 2020, pp. 524–540.
- 30 [Put94] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.
- 31 [PVC19] R. Prenger, R. Valle, and B. Catanzaro. "WaveGLOW: A flow-based generative network for speech synthesis". In: Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2019, pp. 3617–3621.
- 32 [PVP18] A. Pesaranghader, H. Viktor, and E. Paquet. "McDiarmid Drift Detection Methods for Evolving Data Streams". In: ICANN. 2018.
- 33 [PW05] S. Parise and M. Welling. "Learning in Markov Random Fields: An Empirical Study". In: Joint Statistical Meeting. 2005.
- 34 [PW16] B. Paige and F. Wood. "Inference Networks for Sequential Monte Carlo in Graphical Models". In: ICML. Feb. 2016.
- 35 [PY97] J. Pitman and M. Yor. "The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator". In: The Annals of Probability (1997), pp. 855–900.
- 36 [QC+06] J. Quiñonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. "Evaluating Predictive Uncertainty Challenge". In: Machine Learning Challenges: Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 1–27.
- 37 [QC+08] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, eds. *Dataset Shift in Machine Learning (Neural Information Processing series)*. en. Illustrated edition. The MIT Press, Dec. 2008.
- 38 [QCR05] J. Quiñonero-Candela and C. Rasmussen. "A unifying view of sparse approximate Gaussian process regression". In: JMLR 6.3 (2005), pp. 1939–1959.
- 39 [Qin+20] C. Qin, Y. Wu, J. T. Springenberg, A. Brock, J. Donahue, T. P. Lillicrap, and P. Kohli. "Training Generative Adversarial Networks by Solving Ordinary Differential Equations". In: arXiv preprint arXiv:2010.15040 (2020).
- 40 [Qu+21] H. Qu, H. Rahmani, L. Xu, B. Williams, and J. Liu. "Recent Advances of Continual Learning in Computer Vision: An Overview". In: (Sept. 2021). arXiv: 2109.11369 [cs.CV].
- 41 [Qua+07] A. Quattoni, S. Wang, L.-P. Morency, M. Collins, and T. Darrell. "Hidden conditional random fields". In: IEEE PAMI 29.10 (2007), pp. 1848–1852.
- 42 [Que98] M. Queyranne. "Minimizing symmetric submodular functions". In: Math. Programming 82 (1998), pp. 3–12.
- 43 [QZW19] Y. Qiu, L. Zhang, and X. Wang. "Unbiased Contrastive Divergence Algorithm for Training Energy-Based Latent Variable Models". In: International Conference on Learning Representations. 2019.
- 44 [RA13] O. Rippel and R. P. Adams. "High-dimensional probability estimation with deep density models". In: ArXiv Preprint arXiv:1302.5125 (2013).
- 45 [Rab89] L. R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: Proc. of the IEEE 77.2 (1989), pp. 257–286.
- 46 [Rad+18] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving Language Understanding by Generative Pre-Training. Tech. rep. OpenAI, 2018.
- 47 [Rad+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners. Tech. rep. OpenAI, 2019.
- 48 [Raf+19] C. Raef, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: (Oct. 2019). arXiv: 1910.10683 [cs.LG].
- 49 [RAG04] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House Radar Library, 2004.
- 50 [Rai+18] T. Rainforth, A. R. Kosiorek, T. A. Le, C. J. Maddison, M. Igl, F. Wood, and Y. W. Teh. "Tighter Variational Bounds Are Not Necessarily Better". In: ICML. 2018.
- 51 [Rai+20] T. Rainforth, A. Golinski, F. Wood, and S. Zaidi. "Target-Aware Bayesian Inference: How to Beat Optimal Conventional Estimators". In: JMLR 21.88 (2020), pp. 1–54.

BIBLIOGRAPHY

- [Rai68] H. Rai a. Decision Analysis. Addison Wesley, 1968.
- [Rak+08] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. "SimpleMKL". In: JMLR 9 (2008), pp. 2491–2521.
- [Ran+21] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. "Zero-Shot Text-to-Image Generation". In: (Feb. 2021). arXiv: 2102.12092 [cs.CV].
- [Ran16] R. Ranganath. "Hierarchical Variational Models". In: ICML. 2016.
- [Ran+18] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. "Deep State Space Models for Time Series Forecasting". In: NIPS. Curran Associates, Inc., 2018, pp. 7796–7805.
- [Rao10] A. V. Rao. "A Survey of Numerical Methods for Optimal Control". In: Adv. Astronaut. Sci. 135.1 (Jan. 2010).
- [Rao99] R. P. Rao. "An optimal estimation approach to visual perception and learning". en. In: Vision Res. 39.11 (June 1999), pp. 1963–1989.
- [Ras00] C. Rasmussen. "The Infinite Gaussian Mixture Model". In: NIPS. 2000.
- [Rat+09] M. Ratray, O. Stegle, K. Sharp, and J. Winn. "Inference algorithms and learning theory for Bayesian sparse factor analysis". In: Proc. Intl. Workshop on Statistical-Mechanical Informatics. 2009.
- [Rav+18] S. Ravuri, S. Mohamed, M. Rosca, and O. Vinyals. "Learning Implicit Generative Models with the Method of Learned Moments". In: International Conference on Machine Learning. 2018, pp. 4314–4323.
- [RB16] G. P. Rigby BR. "The Efficacy of Equine-Assisted Activities and Therapies on Improving Physical Function". In: J Altern Complement Med. (2016).
- [RB99] R. P. Rao and D. H. Ballard. "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". en. In: Nat Neurosci. 2.1 (Jan. 1999), pp. 79–87.
- [RBB18a] H. Ritter, A. Botev, and D. Barber. "A Scalable Laplace Approximation for Neural Networks". In: ICLR. 2018.
- [RBB18b] H. Ritter, A. Botev, and D. Barber. "Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting". In: NIPS. Curran Associates, Inc., 2018, pp. 3738–3748.
- [RBS16] L. J. Ratti, S. A. Burden, and S. S. Sastry. "On the characterization of local Nash equilibria in continuous games". In: IEEE transactions on automatic control 61.8 (2016), pp. 2301–2307.
- [RC04] C. Robert and G. Casella. Monte Carlo Statistical Methods. 2nd edition. Springer, 2004.
- [RC+18] M. Rojas-Carulla, B. Schölkopf, R. Turner, and J. Peters. "Invariant Models for Causal Transfer Learning". In: Journal of Machine Learning Research 19.36 (2018), pp. 1–34.
- [RD06] M. Richardson and P. Domingos. "Markov logic networks". In: Machine Learning 62 (2006), pp. 107–136.
- [RDL21] O. Rybkin, K. Daniilidis, and S. Levine. "Simple and Effective VAE Training with Calibrated Decoders". In: ICML. 2021.
- [RDV18] A. S. Ross and F. Doshi-Velez. "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients". In: Thirty-second AAAI conference on artificial intelligence. 2018.
- [Rec19] B. Recht. "A Tour of Reinforcement Learning: The View from Continuous Control". In: Annual Review of Control, Robotics, and Autonomous Systems 2 (2019), pp. 253–279.
- [Ree+17] S. Reed, A. van den Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. de Freitas. "Parallel Multiscale Autoregressive Density Estimation". In: (2017). arXiv: 1703.03664 [cs.CV].
- [Rei+10] J. Reisinger, A. Waters, B. Silverthorn, and R. Mooney. "Spherical topic models". In: ICML. 2010.
- [Rei13] S. Reich. "A Nonparametric Ensemble Transform Method for Bayesian Inference". In: SIAM J. Sci. Comput. 35.4 (Jan. 2013), A2013–A2024.
- [Rei16] P. C. Reiss. "Just How Sensitive are Instrumental Variable Estimates?". In: Foundations and Trends in Accounting 10.2–4 (2016).
- [Ren+19] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. A. DePristo, J. V. Dillon, and B. Lakshminarayanan. "Likelihood Ratios for Out-of-Distribution Detection". In: NIPS. 2019.
- [Rén61] A. Rényi. "On Measures of Entropy and Information". en. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics. The Regents of the University of California, 1961.
- [RF96] J. J. K. Ruan and W. J. Fitzgerald. Numerical Bayesian Methods Applied to Signal Processing. en. 1996th ed. Springer, Feb. 1996.
- [RG17] M. Roth and F. Gustafsson. "Computation and visualization of posterior densities in scalar nonlinear and non-Gaussian Bayesian Itering and smoothing problems". In: ICASSP. Mar. 2017, pp. 4686–4690.
- [RGB11] S. Ross, G. J. Gordon, and D. Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: AISTATS. 2011, pp. 627–635.
- [RGB14] R. Ranganath, S. Gerrish, and D. M. Blei. "Black Box Variational Inference". In: AISTATS. 2014.
- [RGL19] S. Rabanser, S. Günnemann, and Z. C. Lipton. "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift". In: NIPS. 2019.
- [RH05] H. Rue and L. Held. Gaussian Markov Random Fields: Theory and Applications. Vol. 104. Monographs on Statistics and Applied Probability. London: Chapman & Hall, 2005.
- [RHDV17] A. S. Ross, M. C. Hughes, and F. Doshi-Velez. "Right for the right reasons: Training differentiable models by constraining their explanations". In: IJCAI (2017).
- [RHG16a] D. Ritchie, P. Horsfall, and N. D. Goodman. "Deep Amortized Inference for Probabilistic Programs". In: (Oct. 2016). arXiv: 1610.05735 [cs.AI].
- [RHG16b] M. Roth, G. Hendraby, and F. Gustafsson. "Nonlinear Kalman Filters Explained: A Tutorial on Moment Computations and Sigma Point Methods". In: J. Advanced Information Fusion (2016).
- [RHW86a] D. Rumelhart, G. Hinton, and R. Williams. "Learning internal representations by error propagation". In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Ed. by D. Rumelhart, J. McClelland, and the PDP Research Group. MIT Press, 1986.
- [RHW86b] D. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: Nature 323 (1986), pp. 533–536.
- [Ric03] T. Richardson. "Markov properties for acyclic directed mixed graphs". In: Scandinavian J. of Statistics 30 (2003), pp. 145–157.
- [Ric95] J. Rice. Mathematical statistics and data analysis. 2nd edition. Duxbury, 1995.
- [Rip05] B. D. Ripley. Spatial statistics. Vol. 575. John Wiley & Sons, 2005.
- [Rip77] B. D. Ripley. "Modelling spatial patterns". In: Journal of the Royal Statistical Society: Series B (Methodological) 39.2 (1977), pp. 172–192.
- [Ris+08] I. Rish, G. Grabarnik, G. Cecchi, F. Pereira, and G. Gordon. "Closed-form supervised dimensionality reduction with generalized linear models". In: ICML. 2008.
- [Riv87] R. L. Rivest. "Learning decision lists". In: Machine learning 2.3 (1987), pp. 229–246.
- [RK04] R. Rifkin and A. Klautau. "In defense of One-Vs-All classification". In: JMLR 5 (2004), pp. 101–141.
- [RL17] S. Ravi and H. Larochelle. "Optimization as a Model for Few-Shot Learning". In: ICLR. 2017.
- [RM15a] G. Raskutti and S. Mukherjee. "The information geometry of mirror descent". In: IEEE Trans. Info. Theory 61.3 (2015), pp. 1451–1457.
- [RM15b] D. J. Rezende and S. Mohamed. "Variational Inference with Normalizing Flows". In: ICML. 2015.
- [RM18] N. L. Roux, P.-A. Manzagol, and Y. Bengio. "Top-moumoute Online Natural Gradient Algorithm". In: NIPS. 2008, pp. 849–856.
- [RMC09] H. Rue, S. Martino, and N. Chopin. "Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations". In: J. of Royal Stat. Soc. Series B 71 (2009), pp. 319–392.
- [RMC15] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: arXiv (2015).
- [RMC16] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: ICLR. 2016.
- [RMW14a] D. Rezende, S. Mohamed, and D. Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: ICML. 2014.
- [RMW14b] D. J. Rezende, S. Mohamed, and D. Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: ICML. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research. Beijing, China: PMLR, 2014, pp. 1278–1286.
- [RN02] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. 2nd edition. Prentice Hall, 2002.
- [RN10] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. 3rd edition. Prentice Hall, 2010.

- 1
- 2 [RN19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 4th edition. Prentice Hall, 2019.
- 3 [RN94] G. A. Rummery and M. Niranjan. On-Line Q-Learning Using Connectionist Systems. Tech. rep. Cambridge Univ. Engineering Dept., 1994.
- 4 [RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.
- 5 [Rob07] C. P. Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, en. 2nd edition. Springer Verlag, New York, 2007.
- 6 [Rob+13] S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson, and S. Agrain. "Gaussian processes for time-series modelling". In: *Philos. Trans. A Math. Phys. Eng. Sci.* 371.1984 (Feb. 2013), p. 201050.
- 7 [Rob+18] C. P. Robert, V. Elvira, N. Tawn, and C. Wu. "Accelerating MCMC Algorithms". In: (Apr. 2018). arXiv: 1804.02719 [stat.CO].
- 8 [Rob86] J. Robins. "A new approach to causal inference in mortality studies with a sustained exposure period—application to control of the healthy worker survivor effect". In: *Mathematical Modelling* 7.9 (1986), pp. 1393–1512.
- 9 [Rob95a] C. Robert. "Simulation of truncated normal distributions". In: *Statistics and computing* 5 (1995), pp. 121–125.
- 10 [Rob95b] A. Robins. "Catastrophic Forgetting, Rehearsals and Pseudorehearsal". In: *Conn. Sci.* 7.2 (June 1995), pp. 123–146.
- 11 [Rod14] J. Rodu. "Spectral estimation of hidden Markov models". PhD thesis. U. Penn, 2014.
- 12 [RÖG13] M. Roth, E. Özkan, and F. Gustafsson. "A Student's t Iter for heavy tailed process and measurement noise". In: *ICASSP*. May 2013, pp. 5770–5774.
- 13 [Ros10] P. Rosenbaum. *Design of Observational Studies*. 2010.
- 14 [Ros+21] M. Rosca, Y. Wu, B. Dherin, and D. G. Barrett. "Discretization Drift in Two-Player Games". In: (2021).
- 15 [Rot+17] M. Roth, G. Hendeby, C. Fritzsche, and F. Gustafsson. "The Ensemble Kalman Iter: a signal processing perspective". In: *EUSIPCO J. Adv. Signal Processing* 2017.1 (Aug. 2017), p. 56.
- 16 [Rot96] D. Roth. "On the hardness of approximate reasoning". In: *Artificial Intelligence* 82.1-2 (1996), pp. 273–302.
- 17 [ROV19] A. Razavi, A. van den Oord, and O. Vinyals. "Generating diverse high resolution images with VA-VAE-2". In: *NIPS*. 2019.
- 18 [Row97] S. Roweis. "EM algorithms for PCA and SPCA". In: *NIPS*. 1997.
- 19 [RPC19] Y. Romano, E. Patterson, and E. J. Candès. "Conformalized Quantile Regression". In: *NIPS*. May 2019.
- 20 [RPH21] A. Robey, G. J. Pappas, and H. Hassani. Model-Based Domain Generalization. 2021. arXiv: 2102.11436 [stat.ML].
- 21 [RR01a] A. Rao and K. Rose. "Deterministically Annealed Design of Hidden Markov Model Speech Recognizers". In: *IEEE Trans. on Speech and Audio Proc.* 9.2 (2001), pp. 111–126.
- 22 [RR01b] G. Roberts and J. Rosenthal. "Optimal scaling for various Metropolis-Hastings algorithms". In: *Statistical Science* 16 (2001), pp. 351–367.
- 23 [RR08] A. Rahimi and B. Recht. "Random Features for Large-Scale Kernel Machines". In: *NIPS*. Curran Associates, Inc., 2008, pp. 1177–1184.
- 24 [RR09] A. Rahimi and B. Recht. "Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning". In: *NIPS*. Curran Associates, Inc., 2009, pp. 1313–1320.
- 25 [RR11] T. S. Richardson and J. M. Robins. "Single World Intervention Graphs: A Primer". In: Second UAI workshop on causal structure learning. 2011.
- 26 [RR13] T. S. Richardson and J. M. Robins. "Single World Intervention Graphs (SWIGs): A Unification of Counterfactual and Graphical Approaches to Causality". 2013.
- 27 [RR14] D. Russo and B. V. Roy. "Learning to Optimize via Posterior Sampling". In: *Math. Oper. Res.* 39.4 (2014), pp. 1221–1243.
- 28 [RR83] P. R. Rosenbaum and D. B. Rubin. "Assessing Sensitivity to an Unobserved Binary Covariate in an Observational Study with Binary Outcome". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 45.2 (1983), pp. 212–218.
- 29 [RRR21] E. Rosenfeld, P. Ravikumar, and A. Risteski. "The Risks of Invariant Risk Minimization". In: *ICML*. 2021.
- 30 [RRS00] J. M. Robins, A. Rotnitzky, and D. O. Scharfstein. "Sensitivity analysis for selection bias and unmeasured confounding in missing data and causal inference models". In: *Statistical models in epidemiology, the environment, and clinical trials*. Springer, 2000, pp. 1–94.
- 31 [RS07] M. Raphan and E. P. Simoncelli. "Learning to be Bayesian without supervision". In: *Advances in neural information processing systems*. 2007, pp. 1145–1152.
- 32 [RS11] M. Raphan and E. P. Simoncelli. "Least squares estimation without priors or supervision". In: *Neural computation* 23.2 (2011), pp. 374–420.
- 33 [RS20] A. Rotnitzky and E. Smucler. "Efficient Adjustment Sets for Population Average Causal Treatment Effect Estimation in Graphical Models". In: *J. Mach. Learn. Res.* 21 (2020), pp. 188–1.
- 34 [RS84] N. Robertson and P. D. Seymour. "Graph minors. III. Planar tree-width". In: *J. Combin. Theory Ser. B* 36.1 (Feb. 1984), pp. 49–64.
- 35 [RS97a] G. O. Roberts and S. K. Sahu. "Updating Schemes, Correlation Structure, Blocking and Parameterization for the Gibbs Sampler". In: *J. of Royal Stat. Soc. Series B* 59.2 (1997), pp. 291–317.
- 36 [RS97b] G. O. Roberts and S. K. Sahu. "Updating schemes, correlation structure, blocking and parameterization for the Gibbs sampler". In: *J. of Royal Stat. Soc. Series B* 59.2 (1997), pp. 291–317.
- 37 [RSC20] Y. Romano, M. Slesia, and E. J. Candès. "Classification with Valid and Adaptive Coverage". In: *NIPS*. June 2020.
- 38 [RSG16a] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.
- 39 [RSG16b] M. T. Ribeiro, S. Singh, and C. Guestrin. "Model-agnostic interpretability of machine learning". In: *arXiv preprint arXiv:1606.05386* (2016).
- 40 [RSG17] S. Rabanser, O. Shchur, and S. Günnemann. "Introduction to Tensor Decompositions and their Applications in Machine Learning". In: (Nov. 2017). arXiv: 1711.10781 [stat.ML].
- 41 [RT16] S. Reid and R. Tibshirani. "Sparse regression and marginal testing using cluster prototypes". In: *Bioinformatics* 32.2 (2016), pp. 364–376.
- 42 [RT96] G. O. Roberts and R. L. Tweedie. "Exponential convergence of Langevin distributions and their discrete approximations". en. In: *Bernoulli* 2.4 (Dec. 1996), pp. 341–363.
- 43 [RTS65] H. E. Rauch, F. Tung, and C. T. Striebel. "Maximum likelihood estimates of linear dynamic systems". In: *AAIA Journal* 3.8 (1965), pp. 1445–1450.
- 44 [Rub+20] Y. Rubanova, D. Dohan, K. Swersky, and K. Murphy. "Amortized Bayesian Optimization over Discrete Spaces". In: *UAI*. 2020.
- 45 [Rub74] D. B. Rubin. "Estimating causal effects of treatments in randomized and nonrandomized studies". In: *J. Educ. Psychol.* 66.5 (1974), pp. 688–701.
- 46 [Rub84] D. B. Rubin. "Bayesian Justifiable and Relevant Frequency Calculations for the Applied Statistician". In: *Ann. Stat.* 12.4 (1984), pp. 1151–1172.
- 47 [Rud19] C. Rudin. *Stop Explaining Black Box Machine Learning Models for High-Stakes Decisions and Use Interpretable Models Instead*. 2019. arXiv: 1811.10154 [stat.ML].
- 48 [Ruf+21] L. Ru, J. R. Kau mann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller. "A Unifying Review of Deep and Shallow Anomaly Detection". In: *Proc. IEEE* 109.5 (2021), pp. 756–795.
- 49 [Rus15] S. Russell. "Unifying Logic and Probability". In: *Commun. ACM* 58.7 (June 2015), pp. 88–97.
- 50 [Rus+16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. "Progressive Neural Networks". In: (2016). arXiv: 1606.04671 [cs.LG].
- 51 [Rus+18] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen. "A Tutorial on Thompson Sampling". In: *Foundations and Trends in Machine Learning* 11.1 (2018), pp. 1–96.
- 52 [Rus22] S. Rush. *Illustrated S4*. Tech. rep. 2022.
- 53 [Rus+95] S. Russell, J. Binder, D. Koller, and K. Kanazawa. "Local learning in probabilistic networks with hidden variables". In: *IJCAI*. 1995.
- 54 [RV19] S. Ravuri and O. Vinyals. "Classification accuracy score for conditional generative models". In: *Advances in Neural Information Processing Systems*. 2019, pp. 12268–12279.
- 55 [RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- 56 [RW11] M. D. Reid and R. C. Williamson. "Information, Divergence and Risk for Binary Experiments". In: *Journal of Machine Learning Research* 12.3 (2011).
- 57 [RW15] D. Rosenbaum and Y. Weiss. "The Return of the Gating Network: Combining Generative Models and Discriminative Training in Natural Image Priors". In: *NIPS*. 2015, pp. 2665–2673.

BIBLIOGRAPHY

- 1 [RW18] E. Richardson and Y. Weiss. "On GANs and GMMS". In: NIPS.
- 2 [RWD17] G. Roeder, Y. Wu, and D. Duvenaud. "Sticking the Landing: An Asymptotically Zero-Variance Gradient Estimator for Variational Inference". In: NIPS. 2017.
- 3 [RY21] D. Roberts and S. Yaida. The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Networks. 2021.
- 4 [Ryc+19] B. Rychalska, D. Basaj, A. Gosiewska, and P. Biecek. "Models in the Wild: On Corruption Robustness of Neural NLP Systems". In: International Conference on Neural Information Processing (ICONIP). Springer International Publishing, 2019, pp. 235–247.
- 5 [Ryu+20] M. Ryu, Y. Chow, R. Anderson, C. Tjandraatmadja, and C. Bouttier. "CAQL: Continuous Action Q-Learning". In: ICLR. 2020.
- 6 [RZL17] P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for Activation Functions". In: (Oct. 2017). arXiv: 1710.05941 [cs.NE].
- 7 [SA19] F. Schäfer and A. Anandkumar. "Competitive gradient descent". In: NIPS. 2019, pp. 7625–7635.
- 8 [Sac+05] K. Sachs, O. Perez, D. Pe'er, D. Lau enburger, and G. Nolan. "Causal Protein-Signaling Networks Derived from Multitiparameter Single-Cell Data". In: Science 308 (2005).
- 9 [SAC17] J. Schulman, P. Abbeel, and X. Chen. Equivalence Between Policy Gradients and Soft Q-Learning. arXiv:1704.06440. 2017.
- 10 [Sai+20] M. Saito, S. Saito, M. Koyama, and S. Kobayashi. "Train Sparsely, Generate Densely: Memory-Efficient Unsupervised Training of High-Resolution Temporal GAN". In: International Journal of Computer Vision 128 (2020), pp. 2586–2606.
- 11 [Saj+18] M. S. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, and S. Gelly. "Assessing generative models via precision and recall". In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2018, pp. 5234–5243.
- 12 [Sal16] T. Salimans. "A Structured Variational Auto-encoder for Learning Deep Hierarchies of Sparse Features". In: (2016). arXiv: 1602.08734 [stat.ML].
- 13 [Sal+16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved Techniques for Training GANs". In: (Mar. 2016). arXiv: 1606.03498 [cs.LG].
- 14 [Sal+17a] M. Salehi, A. Karbasi, D. Scheinost, and R. T. Comello. "A Submodular Approach to Create Individualized Parcellations of the Human Brain". In: Medical Image Computing and Computer Assisted Intervention - MICCAI 2017. Ed. by M. Descoteaux, L. Maier-Hein, A. Franz P. Jannin, D. L. Collins, and S. Duchesne. Cham: Springer International Publishing, 2017, pp. 478–485.
- 15 [Sal+17b] T. Salimans, J. Ho, X. Chen, and I. Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: (Mar. 2017). arXiv: 1703.03864 [stat.ML].
- 16 [Sal+17c] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications". In: ICLR. 2017.
- 17 [Sal+19a] D. Salinas, M. Bohlik-Schneider, L. Callot, R. Medico, and J. Gasthaus. "High-Dimensional Multivariate Forecasting with Low-Rank Gaussian Copula Processes". In: NIPS. 2019.
- 18 [Sal+19b] D. Salinas, V. Flunkert, J. Gasthaus, and Januschowski. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: International Journal of Forecasting (2019).
- 19 [Sal+20] H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry. "Do Adversarially Robust ImageNet Models Transfer Better?". In: arXiv preprint arXiv:2007.08489 (2020).
- 20 [Sal+21] M. Salehi, H. Mirzaei, D. Hendrycks, Y. Li, M. H. Rohban, and M. Sabokrou. "A Unified Survey on Anomaly, Novelty, Open-Set, and Out-of-Distribution Detection: Solutions and Future Challenges". In: (Oct. 2021). arXiv: [cs.CV].
- 21 [Sam68] F. Sampson. "A Novitiate in a Period of Change: An Experimental and Case Study of Social Relationships". PhD thesis. Cornell, 1968.
- 22 [Sam74] P. A. Samuelson. "Complementarity: An essay on the 40th anniversary of the Hicks-Allen revolution in demand theory". In: Journal of Economic literature 12.4 (1974), pp. 1255–1289.
- 23 [San+17] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. "A simple neural network module for relational reasoning". In: Advances in neural information processing systems. 2017, pp. 4967–4976.
- 24 [Sar13] S. Sarkka. Bayesian Filtering and Smoothing. Cambridge University Press, 2013.
- 25 [Sar18] H. Sarin. "Playing a game of GANstruction". In: The Gradient (2018).
- 26 [Sat15] C. Satzinger. "On Tree-Decompositions and their Algorithmic Implications for Bounded-Treewidth Graphs". PhD thesis. U. Wien, 2015.
- 27 [Say+19] R. Sayres, S. Xu, T. Saensuksopa, M. Le, and D. R. Webster. "Assistance from a deep learning system improves diabetic retinopathy assessment in optometrists". In: Investigative Ophthalmology & Visual Science 60.9 (2019), pp. 1433–1433.
- 28 [SB01] A. J. Smola and P. L. Bartlett. "Sparse Greedy Gaussian Process Regression". In: NIPS. Ed. by T. K. Leen, T. G. Dietterich, and V. Tresp. MIT Press, 2001, pp. 619–625.
- 29 [SB12] J. Staines and D. Barber. "Variational Optimization". In: (Dec. 2012). arXiv: 1212.4507 [stat.ML].
- 30 [SB13] J. Staines and D. Barber. "Optimization by Variational Bounding". In: European Symposium on ANNs. elen.ul.ac.be, 2013.
- 31 [SB18] R. Sutton and A. Barto. Reinforcement learning: an introduction (2nd edn). MIT Press, 2018.
- 32 [SBG07] S. Siddiqi, B. Boots, and G. Gordon. "A constraint generation approach to learning stable linear dynamical systems". In: NIPS. 2007.
- 33 [SBP17] Y. Sun, P. Babu, and D. P. Palomar. "Majorization-Minimization Algorithms in Signal Processing, Communications, and Machine Learning". In: IEEE Trans. Signal Process. 65.3 (Feb. 2017), pp. 794–816.
- 34 [SC13] C. Schäfer and N. Chopin. "Sequential Monte Carlo on large binary sampling spaces". In: Stat. Comput. 23.2 (Mar. 2013), pp. 163–184.
- 35 [SC86] R. Smith and P. Cheeseman. "On the Representation and Estimation of Spatial Uncertainty". In: Int'l. J. Robotics Research 5.4 (1986), pp. 56–68.
- 36 [SC90] R. Schwarz and Y. Chow. "The n-best algorithm: an efficient and exact procedure for finding the n most likely hypotheses". In: ICASSP. 1990.
- 37 [Sci21] S. Scardapane. Lecture 8: Beyond single-task supervised learning. 2021.
- 38 [SCC17] S. Sun, C. Chen, and L. Carin. "Learning Structured Weight Uncertainty in Bayesian Neural Networks". In: AISTATS. 2017, pp. 1283–1292.
- 39 [Sch00] A. Schrijver. "A combinatorial algorithm minimizing submodular functions in strongly polynomial time". In: Journal of Combinatorial Theory, Series B 80.2 (2000), pp. 346–355.
- 40 [Sch02] N. N. Schraudolph. "Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent". In: Neural Computation 14 (2002).
- 41 [Sch04] A. Schrijver. Combinatorial Optimization. Springer, 2004.
- 42 [Sch+12a] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. "On Causal and Anticausal Learning". In: ICML. 2012.
- 43 [Sch+12b] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. "On causal and anticausal learning". In: Proceedings of the 29th International Conference on International Conference on Machine Learning. 2012, pp. 459–466.
- 44 [Sch+15a] J. Schulman, N. Heess, T. Weber, and P. Abbeel. "Gradient Estimation Using Stochastic Computation Graphs". In: NIPS. 2015.
- 45 [Sch+15b] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". In: ICML. 2015.
- 46 [Sch+16a] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized Experience Replay". In: ICLR. 2016.
- 47 [Sch+16b] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: ICLR. 2016.
- 48 [Sch+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". In: (July 2017). arXiv: 1707.06347 [cs.LG].
- 49 [Sch+18] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. "Progress & Compression: A scalable framework for continual learning". In: ICML. May 2018.
- 50 [Sch19] B. Schölkopf. "Causality for Machine Learning". In: (Nov. 2019). arXiv: 1911.10500 [cs.LG].
- 51 [Sch+21a] D. O. Scharfstein, R. Nabi, E. H. Kennedy, M.-Y. Huang, M. Bonvini, and M. Smid. Semiparametric Sensitivity Analysis: Unmeasured Confounding In Observational Studies. 2021. arXiv: 2110.08300 [stat.ME].
- 52 [Sch+21b] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio. "Toward Causal Representation Learning". In: Proc. IEEE 109.5 (May 2021), pp. 612–634.
- 53 [Sch78] G. Schwarz. "Estimating the dimension of a model". In: Annals of Statistics 6.2 (1978), pp. 461–464.

- 1
- 2 [Sco02] S Scott. "Bayesian methods for hidden Markov models: Recursive computing in the 21st century". In: JASA (2002).
- 3 [Sco09] S. Scott. "Data augmentation, frequentist estimation, and the Bayesian analysis of multinomial logit models". In: Statistical Papers (2009).
- 4 [Sco10] S. Scott. "A modern Bayesian look at the multi-armed bandit". In: Applied Stochastic Models in Business and Industry 26 (2010), pp. 639–658.
- 5 [SCS19] A. Subbaswamy, B. Chen, and S. Saria. A Universal Hierarchy of Shift-Stable Distributions and the Tradeoff Between Stability and Performance. 2019. arXiv: 1905.11374 [stat.ML].
- 6 [SD12] J. Sohl-Dickstein. "The Natural Gradient by Analogy to Signal Whitening, and Recipes and Tricks for its Use". In: (May 2012). arXiv: 1205.1828 [cs.LG].
- 7 [SD+15a] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: ICML. 2015, pp. 2256–2265.
- 8 [SD+15b] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: ICML. Mar. 2015.
- 9 [SD17] H. Salimbeni and M. Deisenroth. "Doubly Stochastic Variational Inference for Deep Gaussian Processes". In: NIPS. 2017.
- 10 [SDBD11] J. Sohl-Dickstein, P. Battaglino, and M. R. DeWeese. "Minimum Probability of Error Learning". In: Proceedings of the 28th International Conference on International Conference on Machine Learning. 2011, pp. 905–912.
- 11 [SE19] Y. Song and S. Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". In: NIPS. 2019, pp. 11895–11907.
- 12 [SE20a] J. Song and S. Ermon. "Multi-label Contrastive Predictive Coding". In: NIPS. 2020.
- 13 [SE20b] Y. Song and S. Ermon. "Improved Techniques for Training Score-Based Generative Models". In: NIPS. 2020.
- 14 [Sel+17] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: Proceedings of the IEEE international conference on computer vision. 2017, pp. 618–626.
- 15 [Sel+19] A. D. Sebst, D. Boyd, S. A. Friedler, S. Venkatasubramanian, and J. Verma. "Fairness-Aware Abstraction in Sociotechnical Systems". In: Proceedings of the Conference on Fairness, Accountability, and Transparency. FAT*19, Atlanta, GA, USA: Association for Computing Machinery, 2019, 59–68.
- 16 [Ser15] O. Serang. "Fast Computation of Semirings Isomorphic to (x, \max) on R^+ ". In: (Nov. 2015). arXiv: 1511.05690 [cs.DS].
- 17 [Ser+20] J. Serrà, D. Álvarez, V. Gómez, O. Slizovská, J. F. Núñez, and J. Luque. "Input complexity and out-of-distribution detection with likelihood-based generative models". In: ICLR. 2020.
- 18 [Set09] B. Settles. Active learning literature survey. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 2009.
- 19 [Sei94] J. Sethuraman. "A constructive definition of Dirichlet priors". In: Statistica Sinica (1994), pp. 639–650.
- 20 [SF08] Z. Svítkova and L. Fleischer. "Submodular approximation: Sampling-based algorithms and lower bounds". In: FOCS. 2008.
- 21 [SF20] K. Sokol and P. Flach. "Explainability fact sheets: a framework for systematic assessment of explainable approaches". In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency. 2020, pp. 56–67.
- 22 [SFB18] S. A. Sisson, Y. Fan, and M. A. Beaumont. "Overview of ABC". In: Handbook of approximate Bayesian computation. Chapman and Hall/CRC, 2018, pp. 3–54.
- 23 [SG05] E. Snelson and Z. Ghahramani. "Compact Approximations to Bayesian Predictive Distributions". In: ICML. 2005.
- 24 [SG06a] E. Snelson and Z. Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: NIPS. 2006.
- 25 [SG06b] E. Snelson and Z. Ghahramani. "Sparse Gaussian Processes using Pseudo-inputs". In: Advances in Neural Information Processing Systems. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press, 2006.
- 26 [SG09] R. Silva and Z. Ghahramani. "The Hidden Life of Latent Variables: Bayesian Learning with Mixed Graph Models". In: JMLR 10 (2009), pp. 1187–1238.
- 27 [SGF21] S. Särkkä and Á. F. García-Fernández. "Temporal Parallelization of Bayesian Filters and Smoothers". In: IEEE Trans. Automat. Contr. 66.1 (2021).
- 28 [SGS16] A. Sharghi, B. Gong, and M. Shah. "Query-focused extractive video summarization". In: European Conference on Computer Vision. Springer. 2016, pp. 3–19.
- 29 [SH07] R. Salakhutdinov and G. Hinton. "Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes". In: NIPS. 2007.
- 30 [SH09] R. Salakhutdinov and G. Hinton. "Deep Boltzmann Machines". In: AISTATS. Vol. 5. 2009, pp. 448–455.
- 31 [SH10] R. Salakhutdinov and G. Hinton. "Replicated Softmax: an Undirected Topic Model". In: NIPS. 2010.
- 32 [SH22] T. Salimans and J. Ho. "Progressive Distillation for Fast Sampling of Discrete Models". In: ICLR. Feb. 2022.
- 33 [Sha+16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: Proc. IEEE 104.1 (Jan. 2016), pp. 148–175.
- 34 [Sha16] L. S. Shapley. 17. A value for n-person games. Princeton University Press, 2016.
- 35 [Sha+20] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. "Accessory to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 1528–1540.
- 36 [Sha+19] A. Shaikhah, A. Fitzgibbon, D. Vytiniotis, and S. Peyton Jones. "E cient erentiable programming in a functional array-processing language". In: Proceedings of the ACM on Programming Languages 3.ICFP (2019), pp. 1–30.
- 37 [Sha+20] H. Shah, K. Tamuly, A. Raghunathan, P. Jain, and P. Netrapalli. "The Pitfalls of Simplicity Bias in Neural Networks". In: NIPS. June 2020.
- 38 [Sha22] C. Shalizi. Advanced Data Analysis from an Elementary Point of View. Cambridge University Press, 2022.
- 39 [Sha48] C. Shannon. "A mathematical theory of communication". In: Bell Systems Tech. Journal 27 (1948), pp. 379–423.
- 40 [Sha98] R. Shachter. "Bayes-Ball: The Rational Pastime (for determining Irrelevance and Requisite Information in Belief Networks and Influence Diagrams)". In: UAI. 1998.
- 41 [She+11] C. Shen, X. Li, L. Li, and M. C. Were. "Sensitivity analysis for causal inference using inverse probability weighting". In: Biometrical Journal 53.5 (2011), pp. 822–837. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/bimj.201100042>.
- 42 [She+17] T. Shen, T. Lei, R. Barzilay, and T. Jaakkola. "Style transfer from non-parallel text by cross-alignment". In: Advances in neural information processing systems 30 (2017), pp. 6830–6841.
- 43 [She+20] T. Shen, J. Mueller, R. Barzilay, and T. Jaakkola. "Educating Text Autoencoders: Latent Representation Guidance via Denoising". In: ICML. 2020.
- 44 [She+21] Z. Shen, J. Liu, Y. He, X. Zhang, R. Xu, H. Yu, and P. Cui. "Towards Out-Of-Distribution Generalization: A Survey". In: (Aug. 2021). arXiv: 2108.13624 [cs.LG].
- 45 [SHF15] R. Steorts, R. Hall, and S. Fienberg. "A Bayesian Approach to Graphical Record Linkage and De-duplication". In: JASA (2015).
- 46 [Shi00a] H. Shimodaira. "Improving predictive inference under covariate shift by weighting the log-likelihood function". In: J. Stat. Plan. Inference 90.2 (Oct. 2000), pp. 227–244.
- 47 [Shi00b] B. Shipley. Cause and Correlation in Biology: A User's Guide to Path Analysis, Structural Equations and Causal Inference. Cambridge, 2000.
- 48 [Shi+21] C. Shi, D. Sridhar, V. Misra, and D. M. Blei. "On the Assumptions of Synthetic Control Methods". In: (Dec. 2021). arXiv: 2112.05671 [stat.ME].
- 49 [SHU97] P. Smyth, D. Heckerman, and M. I. Jordan. "Probabilistic Independence Networks for Hidden Markov Probability Models". In: Neural Computation 9.2 (1997), pp. 227–269.
- 50 [SHM14] D. Soudry, I. Hubara, and R. Meir. "Expectation back-propagation: Parameter-free training of multilayer neural networks with continuous discrete weights". In: NIPS. 2014.
- 51 [Shr+16] A. Shrikumar, P. Greenside, A. Shcherbinin, and A. Kundaje. "Not just a black box: Learning important features through propagating activation differences". In: arXiv preprint arXiv:1605.01713 (2016).
- 52 [SHS] D. Stutz, M. Hein, and B. Schiele. "Con dence-calibrated adversarial training: Generalizing to unseen attacks". In: (J.).
- 53 [SHS01] B. Schölkopf, R. Herbrich, and A. J. Smola. "A Generalized Representer Theorem". In: COLT, COLT '01/EuroCOLT '01. London, UK, UK: Springer-Verlag, 2001, pp. 416–426.
- 54 [Shu+19] K. Shu, L. Cui, S. Wang, D. Lee, and H. Liu. "defend: Explainable fake news detection". In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019, pp. 395–405.
- 55 [SII00] M. Sato and S. Ishii. "On-line EM algorithm for the normalized Gaussian network". In: Neural Computation 12 (2000), pp. 407–432.
- 56 [Sil+14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. "Deterministic Policy Gradient Algorithms".

BIBLIOGRAPHY

- 1 In: ICML. ICML'14. Beijing, China: JMLR.org, 2014, pp. I–387–I–395.
- 2 [Sil+16] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". en. In: Nature 529.7587 (2016), pp. 484–489.
- 3 [Sil18] D. Silver. Lecture 9L Exploration and Exploitation. 2018.
- 4 [Sil+18] D. Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". en. In: Science 362.6419 (Dec. 2018), pp. 1140–1144.
- 5 [Sil85] B. W. Silverman. "Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting". In: J. R. Stat. Soc. Series B Stat. Methodol. 47.1 (1985), pp. 1–52.
- 6 [Sim06] D. Simon. Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley, 2006.
- 7 [Sin+00] S. Singh, T. Jaakkola, M. L. Littman, and Szepesvári. "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms". In: MLJ 38.3 (Mar. 2000), pp. 287–308.
- 8 [Sin67] R. Sinkhorn. "Diagonal Equivalence to Matrices with Prescribed Row and Column Sums". In: The American Mathematical Monthly 74.4 (1967), pp. 402–405.
- 9 [SJR08] S. Shirdhonkar and D. W. Jacobs. "Approximate earth mover's distance in linear time". In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2008, pp. 1–8.
- 10 [SJ15] A. Swaminathan and T. Joachims. "Batch Learning from Logged Bandit Feedback through Counterfactual Risk Minimization". In: JMLR 16.1 (2015), pp. 1731–1755.
- 11 [SJ95] L. Saul and M. Jordan. "Exploiting tractable substructures in intractable networks". In: NIPS. Vol. 8. 1995.
- 12 [SJ96] L. Saul, T. Jaakkola, and M. Jordan. "Mean Field Theory for Sigmoid Belief Networks". In: JAIR 4 (1996), pp. 61–76.
- 13 [SJR04] S. Singh, M. James, and M. Ruday. "Predictive state representations: A new theory for modeling dynamical systems". In: UAI. 2004.
- 14 [SK19] C. Shorten and T. M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". en. In: Journal of Big Data 6.1 (July 2019), pp. 1–48.
- 15 [SK20] S. Singh and S. Krishnan. "Filter Response Normalization Layer: Eliminating Batch Dependence in the Training of Deep Neural Networks". In: CVPR. 2020.
- 16 [SK21] Y. Song and D. P. Kingma. "How to Train Your Energy-Based Models". In: (Jan. 2021). arXiv: 2101.03288 [cs.LG].
- 17 [SK89] R. Shachter and C. R. Kenley. "Gaussian Influence Diagrams". In: Management Science 35.5 (1989), pp. 527–550.
- 18 [Ski89] J. Skilling. "The eigenvalues of mega-dimensional matrices". In: Maximum Entropy and Bayesian Methods. Springer, 1989, pp. 455–466.
- 19 [SKM07] M. Sugiyama, M. Krauledat, and K.-R. Müller. "Covariate Shift Adaptation by Importance Weighted Cross Validation". In: J. Mach. Learn. Res. 8.35 (2007), pp. 985–1005.
- 20 [SKTF18] H. Shao, A. Kumar, and P. Thomas Fletcher. "The Riemannian Geometry of Deep Generative Models". In: CVPR. 2018, pp. 315–323.
- 21 [SKW15] T. Salimans, D. Kingma, and M. Welling. "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap". In: ICML. 2015, pp. 1218–1226.
- 22 [SKZ19] J. Shi, M. E. Khan, and J. Zhu. "Scalable Training of Inference Networks for Gaussian-Process Models". In: ICML. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5758–5768.
- 23 [SL08] A. L. Strehl and M. L. Littman. "An Analysis of Model-based Interval Estimation for Markov Decision Processes". In: J. of Comp. and Sys. Sci. 74.8 (2008), pp. 1309–1331.
- 24 [SL18] S. L. Smith and Q. V. Le. "A Bayesian Perspective on Generalization and Stochastic Gradient Descent". In: ICLR. 2018.
- 25 [SL+21] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko. "A Gentle Introduction to Graph Neural Networks". In: Distill. (2021). <https://distill.pub/2021/gnn-intro>.
- 26 [SL90] D. J. Spiegelhalter and S. L. Lauritzen. "Sequential updating of conditional probabilities on directed graphical structures". In: Networks 20 (1990).
- 27 [Sla+20] D. Slack, S. Hilgard, E. Jia, S. Singh, Lakkaraju. "Fooling lime and shap: Adversarial attacks on post hoc explanation methods". In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society. 2020, pp. 180–186.
- 28 [SLG17] A. Sharghi, J. S. Laurel, and B. Gong. "Query-focused video summarization: Dataset, evaluation, and a memory network based approach". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017, pp. 4788–4797.
- 29 [Sli19] A. Slivkins. "Introduction to Multi-Armed Bandits". In: Foundations and Trends in Machine Learning (2019).
- 30 [SLL09] A. L. Strehl, L. Li, and M. L. Littman. "Reinforcement Learning in Finite MDPs: PAC Analysis". In: JMLR 10 (2009), pp. 2413–2444.
- 31 [SLW19] M. Sadinle, J. Lei, and L. Wasserman. "Least Ambiguous Set-Valued Classifiers With Bounded Error Levels". In: JASA 114.525 (Jan. 2019), pp. 223–234.
- 32 [SM07] C. Sutton and A. McCallum. "Improved Dynamic Schedules for Belief Propagation". In: UAI. 2007.
- 33 [SM12] Y. Saika and K. Morimoto. "Generalized MAP estimation via parameter scheduling and maximizer of the posterior marginal estimate for image reconstruction using multiple halftone images". In: 12th International Conference on Control, Automation and Systems. 2012, pp. 1285–1289.
- 34 [SMB10] H. Schulz, A. Müller, and S. Behnke. "Investigating convergence of restricted boltzmann machine learning". In: NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning. Vol. 1. 2. 2010, pp. 6–1.
- 35 [SMH07] R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. "Restricted Boltzmann machines for collaborative filtering". In: ICML. Vol. 24. 2007, pp. 791–798.
- 36 [Smi+00] G. Smith, J. F. G. de Freitas, T. Robinson, and M. Niranjani. "Speech Modelling Using Subspace and EM Techniques". In: NIPS. MIT Press, 2000, pp. 796–802.
- 37 [Smi+00] V. Smith, J. Yu, T. Smulders, A. Hartemink, and E. Jarvis. "Computational Inference of Neural Information Flow Networks". In: PLOS Computational Biology 2 (2006), pp. 1436–1439.
- 38 [Smi+17] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. "SmoothGrad: removing noise by adding noise". 2017. arXiv: 1706.03825 [cs.LG].
- 39 [Smol86] P. Smolensky. "Information processing in dynamical systems: foundations of harmony theory". In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1. Ed. by D. Rumelhart and J. McClelland. McGraw-Hill, 1986.
- 40 [SMS17] M. Saito, E. Matsumoto, and S. Saito. "Temporal Generative Adversarial Nets with Singular Value Clipping". In: ICCV. 2017.
- 41 [SMT18] M. R. U. Saputra, A. Markham, and N. Trigoni. "Visual SLAM and Structure from Motion in Dynamic Environments: A Survey". In: ACM Computing Surveys 51.2 (Feb. 2018), pp. 1–36.
- 42 [Smy20] S. Smyl. "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting". In: Int. J. Forecast. 36.1 (Jan. 2020), pp. 75–85.
- 43 [S+16] C. K. Sønderby, T. Raiko, L. Maaløe, S. R. K. Sønderby, and O. Winther. "Ladder Variational Autoencoders". In: NIPS. Curran Associates, Inc., 2016, pp. 3738–3746.
- 44 [SNM16] M. Suzuki, K. Nakayama, and Y. Matsuo. "Joint Multimodal Learning with Deep Generative Models". In: (2016). arXiv: 1611.01891 [stat.ML].
- 45 [S+16] C. Sonderby, T. Raiko, L. Maaløe, S. Sønderby, and O. Winther. "How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks". In: ICM. 2016.
- 46 [Son+19] Y. Song, S. Garg, J. Shi, and S. Ermon. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation". In: Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence. UAI 2019, Tel Aviv, Israel, July 22–25, 2019, 2019, p. 204.
- 47 [Son+21] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: ICLR. 2021.
- 48 [Son98] E. D. Sontag. Mathematical Control Theory: Deterministic Finite Dimensional Systems. 2nd. Vol. 6. Texts in Applied Mathematics. Springer, 1998.
- 49 [SPD92] S. Shah, F. Palmieri, and M. Datum. "Optimal Iterating algorithms for fast learning in feedforward neural networks". In: Neural Netw. 5.5 (Sept. 1992), pp. 779–787.
- 50 [Spi71] M. Spivak. Calculus On Manifolds: A Modern Approach To Classical Theorems Of Advanced Calculus. Westview Press; 5th edition, 1971.
- 51 [Spr+14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. "Striving for simplicity: The all convolutional net". In: arXiv preprint arXiv:1412.6806 (2014).
- 52 [Spr+16] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. "Bayesian Optimization with Robust Bayesian Neural Networks". In: NIPS. 2016, pp. 4141–4149.
- 53 [Spr17] M. W. Spratling. "A review of predictive coding algorithms". en. In: Brain Cogn. 112 (Mar. 2017), pp. 92–97.
- 54 [SPW18] M. H. S. Segler, M. Preuss, and M. P. Waller. "Planning chemical syntheses with deep neural networks and symbolic AI". en. In: Nature 555.7698 (Mar. 2018), pp. 604–610.

and H.

- 1 [SPZ09] P. Schniter, L. C. Potter, and J. Ziniel. "Fast Bayesian
2 Matching Pursuit: Model Uncertainty and Parameter Estimation
3 for Sparse Linear Models". In: IEEE Trans. on Signal Processing
3 (2009).
- 4 [SRG03] R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani.
5 "Optimization with EM and Expectation-Conjugate-Gradient".
5 In: ICML. 2003.
- 6 [Sri+09] B. K. Sriperumbudur, K. Fukumizu, A. Gretton, B.
7 Schölkopf, and G. R. G. Lanckriet. "On integral probability
8 metrics, φ -divergences and binary classification". In: (Jan.
9 2009). arXiv: 0901.2698 [cs.IT].
- 10 [Sri+10] N. Srinivas, A. Krause, S. Kakade, and M. Seeger.
11 "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: ICML. 2010, pp. 1015–1022.
- 12 [Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever,
13 and R. Salakhutdinov. "Dropout: A Simple Way to Prevent
14 Neural Networks from Overfitting". In: JMLR (2014).
- 15 [Sri+17] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann,
16 and C. Sutton. "Veegan: Reducing mode collapse in gans using
17 implicit variational learning". In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, pp. 3310–3320.
- 18 [SRS10] P. Schnitzspahn, S. Roth, and B. Schiele. "Automatic discovery of meaningful object parts with latent CRFs". In: CVPR. 2010.
- 19 [SS17a] A. Srivastava and C. Sutton. "Autoencoding Variational
20 Inference For Topic Models". In: ICLR. 2017.
- 21 [SS17b] A. Svensson and T. B. Schön. "A flexible state-space
22 model for learning nonlinear dynamical systems". In: Automatica
80 (June 2017), pp. 189–199.
- 23 [SS18a] O. Sener and S. Savarese. "Active Learning for Convolutional Neural Networks: A Core-Set Approach". In: International Conference on Learning Representations. 2018.
- 24 [SS18b] A. Subbaswamy and S. Saria. "Counterfactual Normalization: Proactively Addressing Dataset Shift and Improving Reliability Using Causal Mechanisms". In: UAI. Aug. 2018.
- 25 [SS18c] A. Subbaswamy and S. Saria. "Counterfactual Normalization: Proactively Addressing Dataset Shift and Improving Reliability Using Causal Mechanisms". In: Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI). 2018. 2018.
- 26 [SS19] S. Sarkka and A. Solin. Applied stochastic differential equations. en: Cambridge University Press, 2019.
- 27 [SS20] K. E. Smith and A. O. Smith. "Conditional GAN for time-series generation". In: arXiv preprint arXiv:2006.16477 (2020).
- 28 [SS21] I. Sucholutsky and M. Schonlau. "Soft-Label Dataset Distillation and Text Dataset Distillation". In: 2021 International Joint Conference on Neural Networks (IJCNN). 2021, pp. 1–8.
- 29 [SS90] G. R. Shafer and P. P. Shenoy. "Probability propagation". In: Annals of Mathematics and Artificial Intelligence 2 (1990), pp. 327–352.
- 30 [SSA14] K. Swersky, J. Snoek, and R. P. Adams. "Freeze-Thaw Bayesian Optimization". In: (June 2014). arXiv: 1406.3896 [stat.ML].
- 31 [SSA18] K. Shmelkov, C. Schmid, and K. Alahari. "How good is my GAN?". In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, pp. 213–229.
- 32 [SSB17] S. Semeniuta, A. Severyn, and E. Barth. "A Hybrid Convolutional Variational Autoencoder for Text Generation". In: (2017). arXiv: 1702.02390 [cs.CL].
- 33 [SSE18] Y. Song, J. Song, and S. Ermon. "Accelerating Natural Gradient with Higher-Order Invariance". In: ICML. 2018.
- 34 [SSF16] M. W. Seeger, D. Salinas, and V. Flunkert. "Bayesian Intermittent Demand Forecasting for Large Inventories". In: NIPS. 2016, pp. 4646–4654.
- 35 [SSG18a] S. Semeniuta, A. Severyn, and S. Gelly. "On Accurate Evaluation of GANs for Language Generation". In: arXiv preprint arXiv:1806.04938 (2018).
- 36 [SSG18b] S. Semeniuta, A. Severyn, and S. Gelly. "On Accurate Evaluation of GANs for Language Generation". In: (June 2018). arXiv: 1806.04936 [cs.CL].
- 37 [SSG19] R. Singh, M. Sahani, and A. Gretton. "Kernel Instrumental Variable Regression". In: Advances in Neural Information Processing Systems. 2019, pp. 4593–4605.
- 38 [SSH13] S. Sarkka, A. Solin, and J. Hartikainen. "Spatio-Temporal Learning via Infinite-Dimensional Bayesian Filtering and Smoothing: A look at Gaussian process regression through Kalman Itering". In: IEEE Signal Processing Magazine (2013).
- 39 [SSJ12] R. Sipos, P. Shivaswamy, and T. Joachims. "Large-margin learning of submodular summarization models". In: Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics. 2012, pp. 224–233.
- 40 [SSK12] M. Sugiyama, T. Suzuki, and T. Kanamori. Density Ratio Estimation in Machine Learning. en: Cambridge University Press, Feb. 2012.
- 41 [SSM18] S. Santurkar, L. Schmidt, and A. Madry. "Classification-based study of covariate shift in gan distributions". In: International Conference on Machine Learning. PMLR. 2018, pp. 4480–4489.
- 42 [SSZ17] J. Snell, K. Swersky, and R. Zemel. "Prototypical networks for few-shot learning". In: NIPS. 2017, pp. 4077–4087.
- 43 [Sta] Scientific Explanation. <https://plato.stanford.edu/entries/scientific-explanation/#ConcOpenIssuFutuDir>. Accessed: 2021-11-23.
- 44 [Sta07] K. O. Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: Genet. Program. Evolvable Mach. 8.2 (Oct. 2007), pp. 131–162.
- 45 [Sta17] N. Stallard, F. Miller, S. Day, S. W. Hee, J. Madan, S. Zohar, and M. Posch. "Determination of the optimal sample size for a clinical trial accounting for the population size". en: Biom. J. 59.4 (July 2017), pp. 609–625.
- 46 [Ste81] C. M. Stein. "Estimation of the mean of a multivariate normal distribution". In: The annals of Statistics (1981), pp. 1135–1151.
- 47 [Sto09] A. J. Storkey. "When Training and Test Sets are Different: Characterising Learning Transfer". In: Dataset Shift in Machine Learning. 2009.
- 48 [Sto17] J. Stoeck. "A review on statistical inference methods for discrete Markov random fields". In: (Apr. 2017). arXiv: 1704.03331 [stat.ME].
- 49 [STR10] Y. Saatchi, R. Turner, and C. E. Rasmussen. "Gaussian Process Change Point Models". In: ICML. unknown, Aug. 2010, pp. 927–934.
- 50 [Str15] S. H. Strogatz. Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering, Second Edition. CRC Press, 2015.
- 51 [Str17] H. Strobelt, S. Gehrmann, H. Peter, and A. M. Rush. "Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks". In: IEEE transactions on visualization and computer graphics. 24.1 (2017), pp. 667–676.
- 52 [Str19] M. Streeter. "Bayes Optimal Early Stopping Policies for Black-Box Optimization". In: (Feb. 2019). arXiv: 1902.08285 [cs.LG].
- 53 [STY17] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic Attribution for Deep Networks. 2017. arXiv: 1703.01365 [cs.LG].
- 54 [Suc+20] F. P. Such, A. Rawal, J. Lehman, K. Stanley, and J. Clune. "Generating training networks: Accelerating neural architecture search by learning to generate synthetic training data". In: International Conference on Machine Learning. PMLR. 2020, pp. 9206–9216.
- 55 [Sud+10] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. "Nonparametric Belief Propagation". In: CVPR. 2003.
- 56 [Sud06] E. Sudderth. Graphical Models for Visual Object Recognition and Tracking. PhD thesis. MIT, 2006.
- 57 [Sud+10] E. Sudderth, A. Ihler, M. Isard, W. Freeman, and A. Willsky. "Nonparametric Belief Propagation". In: Comm. of the ACM 53.10 (2010).
- 58 [Sug+13] M. Sugiyama, T. Kanamori, T. Suzuki, M. C. du Plessis, S. Liu, and I. Takeuchi. "Density-difference estimation". en: In: Neural Comput. 25.10 (Oct. 2013), pp. 2734–2775.
- 59 [Sun+09] L. Sun, S. Ji, S. Yu, and J. Ye. "On the Equivalence Between Canonical Correlation Analysis and Orthonormalized Partial Least Squares". In: IJCAI. 2009.
- 60 [Sun+18] S. Sun, G. Zhang, C. Wang, W. Zeng, J. Li, and R. Grosse. "Differentiable Compositional Kernel Learning for Gaussian Processes". In: ICML. 2018.
- 61 [Sun+19a] S. Sun, G. Zhang, J. Shi, and R. Grosse. "Functional variational bayesian neural networks". In: arXiv preprint arXiv:1903.05779 (2019).
- 62 [Sun+19b] S. Sun, Z. Cao, H. Zhu, and J. Zhao. "A Survey of Optimization Methods from a Machine Learning Perspective". In: (June 2019). arXiv: 1906.06821 [cs.LG].
- 63 [Sun+19c] M. Sundararajan, J. Xu, A. Taly, R. Sayres, and A. Najmi. "Exploring Principled Visualizations for Deep Network Attributions". In: IUI Workshops. Vol. 4. 2019.
- 64 [Sun+20] Y. Sun, X. Wang, Z. Liu, J. Miller, A. Efros, and M. Hardt. "Test-Time Training with Self-Supervision for Generalization under Distribution Shifts". In: ICML. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 9229–9248.
- 65 [Sut+17] D. J. Sutherland, H.-Y. Tung, H. Strathmann, S. De, A. Ramdas, A. Smola, and A. Gretton. "Generative Models and Model Criticism via Optimized Maximum Mean Discrepancy". In: ICLR. 2017.
- 66 [Sut88] R. Sutton. "Learning to predict by the methods of temporal differences". In: Machine Learning 3.1 (1988), pp. 9–44.
- 67 [Sut90] R. S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: ICML. Ed. by B. Porte and R. Mooney. San Francisco (CA): Morgan Kaufmann, 1990, pp. 216–224.

BIBLIOGRAPHY

- [[1](#)] [Sut96] R. S. Sutton. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding". In: NIPS. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press, 1996, pp. 1038–1044.
- [[2](#)] [Sut+99] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: NIPS. 1999.
- [[3](#)] [[SV08](#)] G. Shafer and V. Vovk. "A Tutorial on Conformal Prediction". In: JMLR 9.Mar (2008), pp. 371–421.
- [[4](#)] [[SV98](#)] M. Studeny and J. Vejnarova. "The multi-information function as a tool for measuring stochastic dependence". In: Learning in graphical models. Ed. by M. Jordan. MIT Press, 1998, pp. 261–297.
- [[5](#)] [[SVE04](#)] A. Smola, S. V. N. Vishwanathan, and E. Eskin. "Laplace Propagation". In: NIPS. MIT Press, 2004, pp. 441–448.
- [[6](#)] [[Svi04](#)] M. Sviridenko. "A note on maximizing a submodular set function subject to a knapsack constraint". In: Operations Research Letters 32.1 (2004), pp. 41–43.
- [[7](#)] [[SVK19](#)] J. Su, D. V. Vargas, and S. Kouichi. "One pixel attack for fooling deep neural networks". In: IEEE Trans. Evol. Comput. 23.5 (2019).
- [[8](#)] [[SVZ13](#)] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep inside convolutional networks: Visualising image classification model and saliency maps". In: arXiv preprint arXiv:1312.6034 (2013).
- [[9](#)] [[SW06](#)] J. E. Smith and R. L. Winkler. "The Optimizer's Curse: Skepticism and Postdecision Surprise in Decision Analysis". In: Manage. Sci. 52.3 (Mar. 2006), pp. 311–322.
- [[10](#)] [[SW13](#)] G. J. Stillman and J. Wisdom. Functional Differential Geometry. MIT Press, 2013.
- [[11](#)] [[SW20](#)] V. G. Satorras and M. Welling. "Neural Enhanced Belief Propagation on Factor Graphs". In: (Mar. 2020). arXiv: 2003.01998 [[cs.LG](#)].
- [[12](#)] [[SW87](#)] R. Swendsen and J.-S. Wang. "Nonuniversal critical dynamics in Monte Carlo simulations". In: Physical Review Letters 58 (1987), pp. 88–88.
- [[13](#)] [[SW89](#)] S. Singh and L. Wu. "Training Multilayer Perceptrons with the Extended Kalman Algorithm". In: NIPS. Vol. 1. 1989.
- [[14](#)] [[Swe+10](#)] K. Swersky, B. Chen, B. Marlin, and N. de Freitas. "A Tutorial on Stochastic Approximation Algorithms for Training Restricted Boltzmann Machines and Deep Belief Nets". In: Information Theory and Applications (ITA) Workshop. 2010.
- [[15](#)] [[Swe+13](#)] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne. "Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces". In: NIPS BayesOpt workshop. 2013.
- [[16](#)] [[SWL03](#)] M. Seeger, C. K. I. Williams, and N. D. Lawrence. "Fast Forward Selection to Speed Up Sparse Gaussian Process Regression". In: AISTATS. 2003.
- [[17](#)] [[SWW08](#)] E. Sudderth, M. Wainwright, and A. Willsky. "Loop series and Bethe variational bounds in attractive graphical models". In: NIPS (2008).
- [[18](#)] [[SYD19](#)] R. Sen, H.-F. Yu, and I. Dhillon. "Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting". In: NIPS. 2019.
- [[19](#)] [[SZ22](#)] R. Schwartz-Ziv. "Information Flow in Deep Neural Networks". PhD thesis. Feb. 2022.
- [[20](#)] [[Sze10](#)] C. Szepesvári. Algorithms for Reinforcement Learning. Morgan Claypool, 2010.
- [[21](#)] [[Sze+14](#)] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. "Intriguing properties of neural networks". In: ICLR. 2014.
- [[22](#)] [[Sze+15](#)] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: CVPR. 2015.
- [[23](#)] [[TAH20](#)] D. Teney, E. Abbasnejad, and A. van den Hengel. "Learning What Makes a Difference from Counterfactual Examples and Gradient Supervision". In: CoRR abs/2004.09034 (2020). arXiv: 2004.09034.
- [[24](#)] [[Tan+16](#)] X. Tan, S. A. Naqvi, K. A. Heller, and V. A. Rao. "Content-based Modeling of Reciprocal Relationships using Hawkes and Gaussian Processes". In: Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence. 2016.
- [[25](#)] [[TB16](#)] P. S. Thomas and E. Brunskill. "Data-Ecient Policy Policy Evaluation for Reinforcement Learning". In: ICML. 2016, pp. 2139–2148.
- [[26](#)] [[TB99](#)] M. Tipping and C. Bishop. "Probabilistic principal component analysis". In: J. of Royal Stat. Soc. Series B 21.3 (1999), pp. 611–622.
- [[27](#)] [[TBA19](#)] N. Tremblay, S. Barthélémy, and P.-O. Amblard. "Determinantal Point Processes for Coresets". In: J. Mach. Learn. Res. 20 (2019), pp. 168–1.
- [[28](#)] [[TBB19](#)] J. Townsend, T. Bird, and D. Barber. "Practical Lossless Compression with Latent Variables using Bits Back Coding". In: ICLR. 2019.
- [[29](#)] [[TBF06](#)] S. Thrun, W. Burgard, and D. Fox. Probabilistic Robotics. MIT Press, 2006.
- [[30](#)] [[TBS13](#)] R. Turner, S. Bottone, and C. Stachniss. "Online variational approximations to non-exponential family change point models: With application to radar tracking". In: NIPS. 2013.
- [[31](#)] [[TDR10](#)] R. Turner, M. Deisenroth, and C. Rasmussen. "State-Space Inference and Learning with Gaussian Processes". In: AISTATS. Vol. 9. Proceedings of Machine Learning Research. Chi Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 868–875.
- [[32](#)] [[Teh06](#)] Y. W. Teh. "A hierarchical Bayesian language model based on Pitman-Yor processes". In: Proc. of the Assoc. for Computational Linguistics. 2006, pp. 985–992.
- [[33](#)] [[Teh+06](#)] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. "Hierarchical Dirichlet Processes". In: JASA 101.476 (2006), pp. 1566–1581.
- [[34](#)] [[Teh+20](#)] N. Tehrani, N. S. Arora, Y. L. Li, K. D. Shah, D. Nouris, M. Tingley, N. Torabi, S. Masouleh, E. Lippert, and E. Meijer. "Bean Machine: A Declarative Probabilistic Programming Language For E cient Programmatic Inference". In: Proceedings of the 10th International Conference on Probabilistic Graphical Models. Ed. by M. Jaeger and T. D. Nielsen. Vol. 138. Proceedings of Machine Learning Research. PMLR, 2020, pp. 485–496.
- [[35](#)] [[Ten+20](#)] I. Tenney, J. Wexler, J. Bastings, T. Bolukbasi, A. Cohen, S. Gehrmann, E. Jiang, M. Pushkarna, C. Radebraugh, E. Reif, et al. "The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models". In: arXiv preprint arXiv:2008.05122 (2020).
- [[36](#)] [[TF03](#)] M. Tipping and A. Faul. "Fast marginal likelihood maximisation for sparse Bayesian models". In: AI/Stats. 2003.
- [[37](#)] [[TG09](#)] Y. W. Teh and D. Gorur. "Indian buffet processes with power-law behavior". In: NIPS. 2009, pp. 1838–1846.
- [[38](#)] [[TG18](#)] L. Tu and K. Gimpel. "Learning Approximate Inference Networks for Structured Prediction". In: ICLR. 2018.
- [[39](#)] [[Tho+19](#)] V. Thomas, F. Pedregosa, B. van Merriënboer, P.-A. Mangazol, Y. Bengio, and N. Le Roux. "Information matrices and generalization". In: (June 2019). arXiv: 1906.07774 [[cs.LG](#)].
- [[40](#)] [[Tho33](#)] W. R. Thompson. "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: Biometrika 25.3/4 (1933), pp. 285–294.
- [[41](#)] [[Thr04](#)] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. "FastSLAM: An E cient solution to the simultaneous localization and mapping problem with unknown data association". In: JMLR 2004 (2004).
- [[42](#)] [[Thr98](#)] S. Thrun. "Lifelong learning algorithms". In: Learning to learn. Ed. by S. Thrun and L. Pratt. Kluwer, 1998, pp. 181–209.
- [[43](#)] [[Thu+21](#)] S. Thulasidasan, S. Thapa, S. Dhaubhadel, G. Chennupati, T. Bhattacharya, and J. Bilmes. "An E ective Baseline for Robustness to Distributional Shift". In: (May 2021). arXiv: 2105.07107 [[cs.LG](#)].
- [[44](#)] [[Tib+19](#)] R. J. Tibshirani, R. F. Barber, E. J. Candès, and A. Ramdas. "Conformal Prediction Under Covariate Shift". In: NIPS. Apr. 2019.
- [[45](#)] [[Tib96](#)] R. Tibshirani. "Regression shrinkage and selection via the lasso". In: J. Royal. Statist. Soc. B 58.1 (1996), pp. 267–288.
- [[46](#)] [[Tie08](#)] T. Tieleman. "Training restricted Boltzmann machines using approximations to the likelihood gradient". In: Proceedings of the 25th international conference on Machine learning. ACM New York, NY, USA, 2008, pp. 1064–1071.
- [[47](#)] [[Tip01](#)] M. Tipping. "Sparse Bayesian learning and the relevance vector machine". In: JMLR 1 (2001), pp. 211–244.
- [[48](#)] [[Tip98](#)] M. Tipping. "Probabilistic visualization of high-dimensional binary data". In: NIPS. 1998.
- [[49](#)] [[Tit09](#)] M. K. Titsias. "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: AISTATS. 2009.
- [[50](#)] [[TK86](#)] L. Tierney and J. Kadane. "Accurate approximations for posterior moments and marginal densities". In: JASA 81.393 (1986).
- [[51](#)] [[TKW08](#)] Y. W. Teh, K. Kurihara, and M. Welling. "Collapsed variational inference for HDP". In: Advances in neural information processing systems. 2008, pp. 1481–1488.
- [[52](#)] [[TL05](#)] E. Todorov and W. Li. "A Generalized Iterative LOG Method for Locally-optimal Feedback Control of Constrained Nonlinear Stochastic Systems". In: ACC. 2005, pp. 303–308.
- [[53](#)] [[TL18a](#)] S. J. Taylor and B. Letham. "Forecasting at scale". In: The American Statistician 72.1 (Jan. 2018), pp. 37–45.
- [[54](#)] [[TL18b](#)] G. Tucker and D. Lawson. "Doubly Reparameterized Gradient Estimators for Monte Carlo Objectives". In: 1st Symposium on Advances in Approximate Bayesian Inference. 2018.

- 1
- 2 [TLG14] M. Titsias and M. Lázaro-Gredilla. "Doubly Stochastic Variational Bayes for non-Conjugate Inference". In: ICML. 2014, pp. 1971–1979.
- 3 [TMD12] A. Tahouk, K. Murphy, and A. Doucet. "Ecient Bayesian Inference for Multivariate Probit Models with Sparse Inverse Correlation Matrices". In: J. Comp. Graph. Statist. 21.3 (2012), pp. 739–757.
- 4 [TMK04] G. Theodorou, K. Murphy, and L. Kaelbling. "Representing hierarchical POMDPs as DBNs for multi-scale robot localization". In: ICRA. 2004.
- 5 [TN13] L. S. L. Tan and D. J. Nott. "Variational Inference for Generalized Linear Mixed Models Using Partially Noncentered Parametrizations". In: Stat. Sci. (2013).
- 6 [TN18] L. S. L. Tan and D. J. Nott. "Gaussian variational approximation with sparse precision matrices". In: Stat. Comput. 28.2 (Mar. 2018), pp. 259–275.
- 7 [TND21] M.-N. Tran, T.-N. Nguyen, and V.-H. Dao. "A practical tutorial on Variational Bayes". In: (Mar. 2021). arXiv: 2103.01327 [stat.CO].
- 8 [TOB16] L. Theis, A. van den Oord, and M. Bethge. "A note on the evaluation of generative models". In: ICLR. 2016.
- 9 [Tob+17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain Randomization for Transferring Neural Networks from Simulation to the Real World". In: IROS. 2017.
- 10 [Tom+20] R. Tomsett, D. Harborne, S. Chakraborty, P. Gurnam, and A. Preece. "Sanity checks for saliency metrics". In: Proceedings of the AAAI conference on artificial intelligence. Vol. 34. 04, 2020, pp. 6021–6029.
- 11 [Tom22] J. M. Tomczak. Deep Generative Modeling. en. 1st ed. Springer, Feb. 2022.
- 12 [Tou09] M. Toussaint. "Robot Trajectory Optimization using Approximate Inference". In: ICML. 2009, pp. 1049–1056.
- 13 [Tou+19] J. Toubeau, J. Bottreau, F. Vallée, and Z. De Grève. "Deep Learning-Based Multivariate Probabilistic Forecasting for Short-Term Scheduling in Power Markets". In: IEEE Trans. Power Syst. 34.2 (Mar. 2019), pp. 1203–1215.
- 14 [TOV18] C. Truong, L. Oudre, and N. Vayatis. "Selective review of one change point detection methods". In: (Jan. 2018). arXiv: 1801.00718 [cs.CE].
- 15 [TP97] S. Thrun and L. Pratt, eds. Learning to learn. Kluwer, 1997.
- 16 [TPB99] N. Tishby, F. Pereira, and W. Biale. "The Information Bottleneck method". In: The 37th annual Allerton Conf. on Communication, Control, and Computing. 1999, pp. 368–377.
- 17 [TR19] M. K. Titsias and F. Ruiz. "Unbiased Implicit Variational Inference". In: AISTATS. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 167–176.
- 18 [TR97] J. Tsitsiklis and B. V. Roy. "An analysis of temporal-difference learning with function approximation". In: IEEE Trans. on Automatic Control 42.5 (1997), pp. 674–690.
- 19 [Tra+19] D. Tran, K. Vafa, K. K. Agrawal, L. Dinh, and B. Poole. "Discrete Flows: Invertible Generative Models of Discrete Data". In: Advances in Neural Information Processing Systems. 2019.
- 20 [Tra+20a] L. Tran, B. S. Veeling, K. Roth, J. Swiatkowski, J. V. Dillon, J. Snoek, S. Mandt, T. Salimans, S. Nowozin, and R. Jegou. "Hydra: Preserving Ensemble Diversity for Model Distillation". In: (Jan. 2020). arXiv: 2001.04694 [cs.LG].
- 21 [Tra+20b] M.-N. Tran, N. Nguyen, D. Nott, and R. "Bayesian Deep Net GLM and GLMM". In: J. Comput. Graph. Stat. 29.1 (Jan. 2020), pp. 97–113.
- 22 [TRB16] D. Tran, R. Ranganath, and D. M. Blei. "The Variational Gaussian Process". In: ICLR. 2016.
- 23 [Tri21] K. Triantafyllopoulos. Bayesian Inference of State Space Models: Kalman Filtering and Beyond. en. 1st ed. Springer, Nov. 2021.
- 24 [Tsa+18] Y.-H. Tsai, W.-C. Hung, S. Schuler, K. Sohn, M.-H. Yang, and M. Chandraker. "Learning to adapt structured output space for semantic segmentation". In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018, pp. 7472–7481.
- 25 [Tsa+19] Y.-H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov. "Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel". In: EMNLP. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4344–4353.
- 26 [Tsa88] C. Tsallis. "Possible generalization of Boltzmann-Gibbs statistics". In: J. of Statistical Physics 52 (1988), pp. 479–487.
- 27 [Tsc+14] S. Tschiatschek, R. Iyer, H. Wei, and J. Bilmes. "Learning Mixtures of Submodular Functions for Image Collection Summarization". In: NIPS. Montreal, Canada, 2014.
- 28 [Tsi+17] P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman. "Human Learning in Atari". en. In: AAAI Spring Symposium Series. Mar. 2017.
- 29 [Tub+13] E. G. Tabak and C. V. Turner. "A family of nonparametric density estimation algorithms". In: Communications on Pure and Applied Mathematics 66.2 (2013), pp. 145–164.
- 30 [TT17] B. Trippe and R. Turner. "Overpruning in Variational Bayesian Neural Networks". In: NIPS Workshop on Advances in Approximate Bayesian Inference. 2017.
- 31 [Tuc+19] G. Tucker, D. Lawson, B. Dai, and R. Ranganath. "Revisiting auxiliary latent variables in generative models". In: (2019).
- 32 [TUI17] T. Taketomi, H. Uchiyama, and S. Ikeda. "Visual SLAM algorithms: a survey from 2010 to 2016". en. In: IPSJ Transactions on Computer Vision and Applications 9.1 (June 2017), p. 16.
- 33 [Tul+18] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. "Mocogan: Decomposing motion and content for video generation". In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018, pp. 1526–1535.
- 34 [Tur+08] R. Turner, P. Berkes, M. Sahani, and D. Mackay. Counterexamples to variational free energy compactness folk theorems. Tech. rep. U. Cambridge, 2008.
- 35 [TVE10] E. G. Tabak and E. Vanden-Eijnden. "Density estimation by dual ascent of the log-likelihood". In: Communications in Mathematical Sciences 8.1 (2010), pp. 217–233.
- 36 [TW16] J. M. Tomczak and M. Welling. "Improving variational auto-encoders using Householder flow". In: NeurIPS Workshop on Bayesian Deep Learning (2016).
- 37 [TX00] J. B. Tenenbaum and F. Xu. "Word learning as Bayesian inference". In: Proc. 22nd Annual Conf. of the Cognitive Science Society. 2000.
- 38 [TZ02] Z. Tu and S. Zhu. "Image Segmentation by Data-Driven Markov Chain Monte Carlo". In: IEEE PAMI 24.5 (2002), pp. 657–673.
- 39 [Tze+17] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. "Adversarial discriminative domain adaptation". In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, pp. 7167–7176.
- 40 [UCS17] S. Ubaru, J. Chen, and Y. Saad. "Fast Estimation of $\text{tr}(f(A))$ via Stochastic Lanczos Quadrature". In: SIAM J. Matrix Anal. Appl. 38.4 (Jan. 2017), pp. 1075–1099.
- 41 [Ude+16] M. Udell, C. Horn, R. Zadeh, and S. Boyd. "Generalized Low Rank Models". In: Foundations and Trends in Machine Learning 9.1 (2016), pp. 1–118.
- 42 [Uel18] K. Ueltzhöer. "Deep Active Inference". In: Biol. Cybern. (2018).
- 43 [UFF06] R. Urtasun, D. Fleet, and P. Fua. "3D people tracking with Gaussian process dynamical models". In: CVPR. 2006.
- 44 [UHJ20] M. Uehara, J. Huang, and N. Jiang. "Minimax Weight and Q-Function Learning for O-Policy Evaluation". In: ICLR. 2020.
- 45 [UML13] B. Uriu, I. Murray, and H. Larochelle. "RNADE: The real-valued neural autoregressive density-estimator". In: NIPS. 2013.
- 46 [UML14] B. Uriu, I. Murray, and H. Larochelle. "A Deep and Tractable Density Estimator". In: ICML. 2014.
- 47 [UN98] N. Ueda and R. Nakano. "Deterministic annealing EM algorithm". In: Neural Networks 11 (1998), pp. 271–282.
- 48 [UR16] B. Ustun and C. Rudin. "Supersparse linear integer models for optimized medical scoring systems". In: Machine Learning 102.3 (2016), pp. 349–391.
- 49 [Ura+16] B. Uriu, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. "Neural Autoregressive Distribution Estimation". In: JMLR (May 2016).
- 50 [UTR14] B. Ustun, S. Tracá, and C. Rudin. Supersparse Linear Integer Models for Interpretable Classification. 2014. arXiv: 1306.6877 [stat.ML].
- 51 [Urt+17] V. Urtio, J. M. Monteiro, J. Kandola, J. Shawe-Taylor, D. Fernández-Reyes, and J. Rousu. "A Tutorial on Canonical Correlation Methods". In: ACM Computing Surveys (2017).
- 52 [UVL16] D. Ulyanov, A. Vedaldi, and V. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: (2016). arXiv: 1607.08022 [cs.CV].
- 53 [UVL18] D. Ulyanov, A. Vedaldi, and V. Lempitsky. "Deep Image Prior". In: CVPR. 2018.
- 54 [VA15] L. Vandenberghe and M. S. Andersen. "Chordal Graphs and Semidefinite Optimization". In: Foundations and Trends in Optimization 1.4 (2015), pp. 241–433.
- 55 [Vaa00] A. W. Van der Vaart. Asymptotic statistics. Vol. 3. Cambridge university press, 2000.
- 56 [Val00] H. Valpola. "Bayesian Ensemble Learning for Nonlinear Factor Analysis". PhD thesis. Helsinki University of Technology, 2000.

BIBLIOGRAPHY

- 1 [Van10] J. Vanhatalo. "Speeding up the inference in Gaussian process models". PhD thesis, Helsinki Univ. Technology, 2010.
- 2 [van+18] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep Reinforcement Learning and the Deadly Triad. arXiv:1812.02648. 2018.
- 3 [Vas+17a] A. B. Vasudevan, M. Gygli, A. Volokitin, and L. Van Gool. "Query-adaptive video summarization via quality-aware relevance estimation". In: Proceedings of the 25th ACM international conference on Multimedia. 2017, pp. 582–590.
- 4 [Vas+17b] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is all you need". In: NIPS. 2017, pp. 5998–6008.
- 5 [Vas+17c] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is All You Need". In: NIPS. 2017.
- 6 [Vaz+22] S. Vaze, K. Han, A. Vedaldi, and A. Zisserman. "Open-Set Recognition: A Good Closed-Set Classifier is All You Need". In: ICLR. 2022.
- 7 [VBW15] S. S. Villar, J. Bowden, and J. Wason. "Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges". en: In: Stat. Sci. 30.2 (2015), pp. 199–215.
- 8 [Ved+18] R. Vedantam, I. Fischer, J. Huang, and K. Murphy. "Generative Models of Visually Grounded Imagination". In: ICLR. 2018.
- 9 [Veh+19] A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner. "Rank-normalization, folding, and localization: An improved R for assessing convergence of MCMC". In: (Mar. 2019). arXiv: 1903.08008 [stat.CO].
- 10 [Vel+21] V. Veitch, A. D'Amour, S. Yadlowsky, and J. Eisenstein. "Counterfactual Invariance to Spurious Correlations: Why and How to Pass Stress Tests". In: Advances in Neural Information Processing Systems. 2021.
- 11 [Vel+17] P. Veli kovi , G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. "Graph attention networks". In: arXiv preprint arXiv:1710.10903 (2017).
- 12 [Vel+18] P. Veli kovi , G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. "Graph attention networks". In: ICLR. 2018.
- 13 [Ver18] R. Vershynin. High-Dimensional Probability: An Introduction with Applications in Data Science. en. 1 edition. Cambridge University Press, Sept. 2018.
- 14 [Ver+19] A. Vergari, A. Molina, R. Peherz, Z. Ghahramani, K. Kersting, and I. Valera. "Automatic Bayesian Density Analysis". In: AAAI. 2019.
- 15 [VF+09] B. C. Van Fraassen et al. The scientific image. Oxford University Press, 2009.
- 16 [VG+09] L. Van Gool, M. D. Breitenstein, F. Reichlin, B. Leibe, and E. Koller-Meier. "Robust Tracking-by-Detection using a Detector Confidence Particle Filter". In: ICCV. 2009.
- 17 [VGG17] A. Vehtari, A. Gelman, and J. Gabry. "Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC". In: Stat. Comput. 27.5 (Sept. 2017), pp. 1413–1432.
- 18 [VGS05] V. Vovk, A. Gammerman, and G. Shafer. Algorithmic Learning in a Random World. en. 2005th ed. Springer, Mar. 2005.
- 19 [Vid99] P. Vidoni. "Exponential Family State Space Models Based on a Conjugate Latent Process". In: J. R. Stat. Soc. Series B Stat. Methodol. 61.1 (1999), pp. 213–221.
- 20 [Vil08] C. Villani. Optimal Transport: Old and New. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008.
- 21 [Vil+19] R. Villegas, A. Pathak, H. Kannan, D. Erhan, Q. V. Le, and H. Lee. "High Fidelity Video Prediction with Large Stochastic Recurrent Neural Networks". In: NIPS. 2019.
- 22 [Vin+10] M. Vinys, J. Cerdà, J. Rodríguez-Aguilar, and A. Farinelli. "Worst-case bounds on the quality of max-product fixed-points". In: NIPS. 2010.
- 23 [Vin11] P. Vincent. "A connection between score matching and denoising autoencoders". In: Neural computation 23.7 (2011), pp. 1661–1674.
- 24 [Vir10] S. Virtanen. "Bayesian exponential family projections". MA thesis, Aalto University, 2010.
- 25 [Vis+06] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods". In: ICML. Pittsburgh, Pennsylvania: ACM Press, 2006.
- 26 [Vis+10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. "Graph Kernels". In: JMLR 11 (2010), pp. 1201–1242.
- 27 [Vit67] A. Viterbi. "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm". In: IEEE Trans. on Information Theory 13.2 (1967), pp. 260–269.
- 28 [Vik20] A. Vahdat and J. Kautz. "NVAE: A Deep Hierarchical Variational Autoencoder". In: NIPS. 2020.
- 29 [VLT21] G. M. van de Ven, Z. Li, and A. S. Tolias. "Class-Incremental Learning with Generative Classifiers". In: CVPR workshop on Continual Learning in Computer Vision (CLVi-sion). Apr. 2021.
- 30 [Vo+15] B.-N. Vo, M. Mallick, Y. Bar-Shalom, S. Coraluppi, R. Osborne, R. Mahler, B. t Vo, and J. Webster. Multitarget tracking. John Wiley and Sons, 2015.
- 31 [Von13] P. Vontobel. "The Bethe permanent of a non-negative matrix". In: IEEE Trans. Info. Theory 59.3 (2013).
- 32 [Vov13] V. Vovk. "Kernel Ridge Regression". In: Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik. Ed. by B. Schölkopf, Z. Luo, and V. Vovk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 105–116.
- 33 [PVV10] J. Vanhatalo, V. Pietiläinen, and A. Vehtari. "Approximate inference for disease mapping with sparse Gaussian processes". In: Statistics in Medicine 29.15 (2010), pp. 1580–1607.
- 34 [vR11] M. van der Laan and S. Rose. Targeted Learning: Causal Inference for Observational and Experimental Data. Jan. 2011.
- 35 [VRF11] C. Varin, N. Reid, and D. Firth. "An overview of composite likelihood methods". In: Stat. Sin. 21.1 (2011), pp. 5–42.
- 36 [VT11] S. I. VanderWeele TJ. "A new criterion for confounder selection". In: Biometrics (2011).
- 37 [VT18] G. M. van de Ven and A. S. Tolias. "Three scenarios for continual learning". In: NeurIPS Continual Learning workshop. 2018.
- 38 [Vyt+19] D. Vytniotis, D. Belov, R. Wei, G. Plotkin, and M. Abadi. "The di erentiable curry". In: NeurIPS 2019 Workshop Program Transformations. 2019.
- 39 [VZ20] V. Veitch and A. Zaveri. "Sense and Sensitivity Analysis: Simple Post-Hoc Analysis of Bias Due to Unobserved Confounding". In: Advances in Neural Information Processing Systems. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 10999–11009.
- 40 [WA18] A. Wilson and R. Adams. "Gaussian process kernels for pattern discovery and extrapolation". In: International conference on machine learning. 2013, pp. 1067–1075.
- 41 [Wan+16] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. "Dueling Network Architectures for Deep Reinforcement Learning". In: ICML. 2016.
- 42 [Wan17] M. P. Wand. "Fast Approximate Inference for Arbitrarily Large Semiparametric Regression Models via Message Passing". In: J. Am. Stat. Assoc. 112.517 (Jan. 2017), pp. 137–168.
- 43 [Wan+17a] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klamp, and P. MacNeille. "A bayesian framework for learning rule sets for interpretable classification". In: The Journal of Machine Learning Research 18.1 (2017), pp. 2357–2393.
- 44 [Wan+17b] X. Wang, T. Li, S. Sun, and J. M. Corchado. "A Survey of Recent Advances in Particle Filters and Remaining Challenges for Multitarget Tracking". en: In: Sensors 17.12 (Nov. 2017).
- 45 [Wan+17c] Y. Wang et al. "Tacotron: Towards End-to-End Speech Synthesis". In: Interspeech. 2017.
- 46 [Wan+18] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro. "Video-to-video synthesis". In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2018, pp. 1152–1164.
- 47 [Wan+19a] K. Wang, G. Pleiss, J. Gardner, S. Tyree, K. Q. Weinberger, and A. G. Wilson. "Exact Gaussian Processes on a million Data Points". In: NIPS. 2019, pp. 14622–14632.
- 48 [Wan+19b] S. Wang, W. Bai, C. Lavanja, and J. Bilmes. "Fixing Mini-batch Sequences with Hierarchical Robust Partitioning". In: Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3352–3361.
- 49 [Wan+19c] Y. Wang, A. Smola, D. C. Maddix, J. Gasthaus, D. Foster, and T. Jaśnuschowski. "Deep Factors for Forecasting". In: ICML. 2019.
- 50 [Wan+20a] H. Wang, X. Wu, Z. Huang, and E. P. Xing. "High-frequency component helps explain the generalization of convolutional neural networks". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 8684–8694.
- 51 [Wan+20b] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros. Dataset Distillation. 2020. arXiv: 1811.10959 [cs.LG].
- 52 [Wan+20c] Z. Wang, S. Cheng, L. Yueru, J. Zhu, and B. Zhang. "A Wasserstein Minimum Velocity Approach to Learning Unnormalized Models". In: Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Ed. by S. Chiappa and R. Calandri. Vol. 108. Proceedings of Machine Learning Research. Online: PMLR, 2020, pp. 3728–3738.
- 53 [Wan+21] J. Wang, C. Lan, C. Liu, Y. Ouyang, W. Zeng, and T. Qin. "Generalizing to Unseen Domains: A Survey on Domain Generalization". In: IJCAI. Mar. 2021.

- 1
- 2 [Was06] L. Wasserman. All of Nonparametric Statistics. Springer, 2006.
- 3 [Wat10] S. Watanabe. "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory". In: JMLR 11 (Dec. 2010), pp. 3571–3594.
- 4 [Wat13] S. Watanabe. "A Widely Applicable Bayesian Information Criterion". In: JMLR 14 (2013), pp. 867–897.
- 5 [Wat60] S. Watanabe, "Information theoretical analysis of multivariate correlation". In: IBM J. of Research and Development 4 (1960), pp. 66–82.
- 6 [WB05] J. Winn and C. Bishop. "Variational Message Passing". In: JMLR 6 (2005), pp. 661–694.
- 7 [WB12] C. Wang and D. M. Blei. "Truncation-free online variational inference for Bayesian nonparametric models". In: Advances in neural information processing systems. 2012, pp. 413–421.
- 8 [WB20] T. Wang and J. Ba. "Exploring Model-based Planning with Policy Networks". In: ICLR. 2020.
- 9 [WBC21] Y. Wang, D. M. Blei, and J. P. Cunningham. "Posterior Collapse and Latent Variable Non-identifiability". In: NIPS. 2021.
- 10 [WCS08] M. Welling, C. Chemudugunta, and N. Suttor. "Deterministic Latent Variable Models and their Pitfalls". In: ICDM. 2008.
- 11 [WD92] C. Watkins and P. Dayan. "Q-learning". In: Machine Learning 8.3 (1992), pp. 279–292.
- 12 [WDN15] A. G. Wilson, C. Dann, and H. Nickisch. "Thoughts on Massively Scalable Gaussian Processes". In: arxiv preprint arXiv:1511.01870 (2015). <https://arxiv.org/abs/1511.01870>.
- 13 [Web17] T. Weber et al. "Imagination-Augmented Agents for Deep Reinforcement Learning". In: NIPS. 2017.
- 14 [Wei00] Y. Weiss. "Correctness of local probability propagation in graphical models with loops". In: Neural Computation 12 (2000), pp. 1–41.
- 15 [Wei+13] K. Wei, Y. Liu, K. Kircho , and J. Bilmes. "Using Document Summarization Techniques for Speech Data Subset Selection". In: HLT-NAACL. 2013, pp. 721–726.
- 16 [Wei+14] K. Wei, Y. Liu, K. Kircho , and J. Bilmes. "Unsupervised Submodular Subset Selection for Speech Data". In: Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing. Florence, Italy, 2014.
- 17 [Wei+15a] K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes. "How to Intelligently Distribute Training Data to Multiple Compute Nodes: Distributed Machine Learning via Submodular Partitioning". In: Neural Information Processing Society (NeurIPS, formerly NIPS) Workshop, LearningSys Workshop, <http://learningsys.org>. Montreal, Canada, 2015.
- 18 [Wei+15b] K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes. "Mixed Robust/Average Submodular Partitioning: Fast Algorithms, Guarantees, and Applications". In: Neural Information Processing Society (NeurIPS, formerly NIPS). Montreal, Canada, 2015.
- 19 [Wei11] M. Welling. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: ICML. 2011.
- 20 [Wen+17] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. "A Multi-Horizon Quantile Recurrent Forecaster". In: NIPS Time Series Workshop. 2017.
- 21 [Wen+19a] L. Wenliang, D. Sutherland, H. Strathmann, and A. Gretton. "Learning deep kernels for exponential family densities". In: International Conference on Machine Learning. 2019, pp. 6737–6746.
- 22 [Wen+19b] F. Wenzel, T. Galy-Fajou, C. Donner, M. Kloft, and M. Opper. "Efficient Gaussian Process Classification Using Poly-Gamma Data Augmentation". In: AAAI. 2019.
- 23 [Wen+20a] C. Wendler, A. Amrollahi, B. Seifert, A. Krause, and M. Püschel. "Learning set functions that are sparse in non-orthogonal Fourier bases". In: arXiv preprint arXiv:2010.00439 (2020).
- 24 [Wen+20b] F. Wenzel, K. Roth, B. S. Veeling, J. wi tkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin. "How Good is the Bayes Posterior in Deep Neural Networks Really?". In: ICML. 2020.
- 25 [Wen+20c] F. Wenzel, J. Snoek, D. Tran, and R. Jenatton. "Hyperparameter Ensembles for Robustness and Uncertainty Quantification". In: NIPS. 2020.
- 26 [Wen21] L. Weng. "What are diffusion models?". In: lilianweng.github.io/lil-log (2021).
- 27 [Wes03] M. West. "Bayesian Factor Regression Models in the Large p, Small n Paradigm". In: Bayesian Statistics 7 (2003).
- 28 [Wes87] M. West. "On scale mixtures of normal distributions". In: Biometrika 74 (1987), pp. 646–648.
- 29 [WF01a] Y. Weiss and W. T. Freeman. "Correctness of belief propagation in Gaussian graphical models of arbitrary topology". In: Neural Computation 13.10 (2001), pp. 2173–2200.
- 30 [WF01b] Y. Weiss and W. T. Freeman. "On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs". In: IEEE Trans. Information Theory, Special Issue on Codes on Graphs and Iterative Algorithms 47.2 (2001), pp. 723–735.
- 31 [WF14] Z. Wang and N. de Freitas. "Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters". In: (June 2014). arXiv: [1406.7758 \[stat.ML\]](https://arxiv.org/abs/1406.7758).
- 32 [WF16] Z. Wang and N. de Freitas. "Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters". In: BayesOpt Workshop. 2016.
- 33 [WG18] M. Wu and N. Goodman. "Multimodal Generative Models for Scalable Weakly-Supervised Learning". In: NIPS. Feb. 2018.
- 34 [WGY21] G. Weiss, Y. Goldberg, and E. Yahav. "Thinking Like Transformers". In: ICML. June 2021.
- 35 [WH02] M. Welling and G. E. Hinton. "A new learning algorithm for mean eld Boltzmann machines". In: International Conference on Artificial Neural Networks. Springer. 2002, pp. 351–357.
- 36 [WH18] Y. Wu and K. He. "Group Normalization". In: ECCV. 2018.
- 37 [WH97] M. West and J. Harrison. Bayesian forecasting and dynamic models. Springer, 1997.
- 38 [WH08] J. T. Wilson, F. Hutter, and M. P. Deisenroth. "Maximizing acquisition functions for Bayesian optimization". In: NIPS. 2018.
- 39 [WHF06] J. Wang, A. Hertzmann, and D. J. Fleet. "Gaussian Process Dynamical Models". In: NIPS. MIT Press, 2006, pp. 1441–1448.
- 40 [Whi16] T. White. "Sampling Generative Networks". In: arXiv (2016).
- 41 [Whi88] P. Whittle. "Restless bandits: activity allocation in a changing world". In: J. Appl. Probab. 25.A (1988), pp. 287–298.
- 42 [WHT19] Y. Wang, H. He, and X. Tan. "Truly Proximal Policy Optimization". In: UAI. 2019.
- 43 [WI20] A. G. Wilson and P. Izmailov. "Bayesian Deep Learning and a Probabilistic Perspective of Generalization". In: NIPS. Feb. 2020.
- 44 [WI15] K. Wei, R. Iyer, and J. Bilmes. "Submodularity in Data Subset Selection and Active Learning". In: Proceedings of the 32nd international conference on Machine learning. Lille, France, 2015.
- 45 [Wik21] Wikipedia contributors. Cl sNotes — Wikipedia, The Free Encyclopedia. [Online; accessed 29-December-2021]. 2021.
- 46 [Wil+14] A. G. Wilson, E. Gilboa, A. Nehorai, and J. P. Cunningham. "Fast kernel learning for multidimensional pattern extrapolation". In: Advances in Neural Information Processing Systems. 2014, pp. 3626–3634.
- 47 [Wil11] A. G. Wilson. "Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes". Phd thesis. University of Cambridge, 2014.
- 48 [Wil+16a] A. G. Wilson, Z. Hu, R. R. Salakhutdinov, and E. P. Xing. "Stochastic Variational Deep Kernel Learning". In: NIPS. Curran Associates, Inc., 2016, pp. 2586–2594.
- 49 [Wil+16b] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. "Deep Kernel Learning". en. In: AISTATS. May 2016, pp. 370–378.
- 50 [Wil+17] A. G. Wills, J. Hendriks, C. Renton, and B. Ninness. "A Bayesian Filtering Algorithm for Gaussian Mixture Models". In: (May 2017). arXiv: [1705.05495 \[stat.ML\]](https://arxiv.org/abs/1705.05495).
- 51 [Wil20] A. G. Wilson. "The Case for Bayesian Deep Learning". In: (Jan. 2020). arXiv: [2001.10995 \[cs.LG\]](https://arxiv.org/abs/2001.10995).
- 52 [Wil+20a] J. T. Wilson, V. Borovitskiy, A. Terenin, M. Mostovsky, and M. P. Deisenroth. "Efficient Sampling Functions from Gaussian Process Posteriors". In: ICML. Feb. 2020.
- 53 [Wil+20b] A. B. Wiltschko, B. Sanchez-Lengeling, B. Lee, E. Reif, J. Wei, K. J. McCloskey, L. Colwell, W. Qian, and Y. Wang. "Evaluating Attribution for Graph Neural Networks". In:
- 54 [Wil69] A. G. Wilson. "The use of entropy maximising models, in the theory of trip distribution, mode split and route split". In: Journal of transport economics and policy (1969), pp. 108–126.
- 55 [Wil92] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: MLJ 8.3–4 (1992), pp. 229–256.
- 56 [Wil98] C. Williams. "Computation with infinite networks". In: Neural Computation 10.5 (1998), pp. 1203–1216.
- 57 [Win] J. Winn. VIBES.

BIBLIOGRAPHY

- 1 [Wit] DEEFAKES: PREPARE NOW (PERSPECTIVES
FROM BRAZIL). <https://lab.witness.org/brazil-deeefakes-prepare-now/>. Accessed: 2021-08-18.
- 2 [Wiy+19] R. R. Wiyatno, A. Xu, O. Dia, and A. de Berker. "Adversarial Examples in Modern Machine Learning: A Review". In: (Nov. 2019). arXiv: 1911.05268 [cs.LG].
- 3 [WJ08] M. J. Wainwright and M. I. Jordan. "Graphical models, exponential families, and variational inference". In: Foundations and Trends in Machine Learning 1-2 (2008), pp. 1-305.
- 4 [WJW03] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. "Tree-based reparameterization framework for analysis of sum-product and related algorithms". In: IEEE Trans. on Information Theory 49.5 (2003), pp. 1120-1146.
- 5 [WK18] E. Wong and Z. Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: International Conference on Machine Learning. PMLR. 2018, pp. 5286-5295.
- 6 [WK96] G. Widmer and M. Kubat. "Learning in the presence of concept drift and hidden contexts". In: Mach. Learn. 23.1 (Apr. 1996), pp. 69-101.
- 7 [WKS21] V. Wild, M. Kanagawa, and D. Sejdinovic. "Connections and Equivalences between the Nyström Method and Sparse Variational Gaussian Processes". In: (June 2021). arXiv: 2106.01121 [stat.ML].
- 8 [WL14] J. L. Williams and R. A. Lau. "Approximate evaluation of marginal association probabilities with belief propagation". In: IEEE Trans. Aerosp. Electron. Syst. 50.4 (2014).
- 9 [WL19] A. Wehenkel and G. Louppe. "Unconstrained Monotonic Neural Networks". In: NIPS. 2019.
- 10 [WLL16] W. Wang, H. Lee, and K. Livescu. "Deep Variational Canonical Correlation Analysis". In: arXiv (Nov. 2016).
- 11 [WLZ19] D. Widmann, F. Lindsten, and D. Zachariah. "Calibration tests in multi-class classification: A unifying framework". In: NIPS. Curran Associates, Inc., 2019, pp. 12236-12246.
- 12 [WM01] E. A. Wan and R. V. der Merwe. "The Unscented Kalman Filter". In: Kalman Filtering and Neural Networks. Ed. by S. Haykin. Wiley, 2001.
- 13 [WM12] K. Wakabayashi and T. Miura. "Forward-Backward Activation Algorithm for Hierarchical Hidden Markov Models". In: NIPS. 2012.
- 14 [WMR17] S. Wachter, B. Mittelstadt, and C. Russell. "Counterfactual explanations without opening the black box: Automated decisions and the GDPR". In: Harv. JL & Tech. 31 (2017), p. 841.
- 15 [WMR18] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. 2018. arXiv: 1711.00399 [cs.AI].
- 16 [WN07] D. Wipf and S. Nagarajan. "A new view of automatic relevancy determination". In: NIPS. 2007.
- 17 [WN10] D. Wipf and S. Nagarajan. "Iterative Reweighted ℓ_1 and ℓ_2 Methods for Finding Sparse Solutions". In: J. of Selected Topics in Signal Processing (Special Issue on Compressive Sensing) 4.2 (2010).
- 18 [WN15] A. G. Wilson and H. Nickisch. "Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)". In: ICML'15. Lille, France: JMLR.org, 2015, pp. 1775-1784.
- 19 [WN18] C. K. I. Williams and C. Nash. "Autoencoders and Probabilistic Inference with Missing Data: An Exact Solution for The Factor Analysis Case". In: (Jan. 2018). arXiv: [cs.LG].
- 20 [Wol12] M. Wiering and M. van Otterlo, eds. Reinforcement learning: State-of-the-art. Springer, 2012.
- 21 [Wo+21] M. Woczyk, M. Zajc, R. Pascanu, . Kuciński, and P. Mioc. "Continual World: A Robotic Benchmark For Continual Reinforcement Learning". In: NIPS. May 2021.
- 22 [Wol76] P. Wolfe. "Finding the nearest point in a polytope". In: Mathematical Programming 11 (1976), pp. 128-149.
- 23 [Wol92] D. Wolpert. "Stacked Generalization". In: Neural Networks 5.2 (1992), pp. 241-259.
- 24 [Woo+09] F. Wood, C. Archambeau, J. Gasthaus, L. James, and Y. W. Teh. "A Stochastic Memoizer for Sequence Data". In: ICML. 2009.
- 25 [Woo+11] F. Wood, J. Gasthaus, C. Archambeau, L. James, and Y. W. Teh. "The sequence memoizer". In: Comm. of the ACM 54.2 (2011), pp. 91-98.
- 26 [Woo+19] B. Woodworth, S. Gunasekar, P. Savarese, E. Moroshko, I. Galan, J. Lee, D. Soudry, and N. Srebro. "Kernel and Rich Regimes in Overparametrized Models". In: (June 2019). arXiv: 1906.05827 [cs.LG].
- 27 [Woo+20] F. Wood, A. Warrington, S. Naderiparizi, C. Weilbach, V. Mariani, W. Harvey, A. Scibior, B. Beronov, and A. Nasseri. Planning as Inference in Epidemiological Models. arXiv:2003.13221. 2020.
- 28 [WP19] S. Wiegreffe and Y. Pinter. "Attention is not not explanation". In: arXiv preprint arXiv:1908.04626 (2019).
- 29 [WR15] F. Wang and C. Rudin. "Falling Rule Lists". In: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics. Ed. by G. Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, 2015, pp. 1013-1022.
- 30 [WRN10] D. Wipf, B. Rao, and S. Nagarajan. "Latent Variable Bayesian Models for Promoting Sparsity". In: IEEE Transactions on Information Theory (2010).
- 31 [WRZH04] M. Welling, M. Rosen-Zvi, and G. Hinton. "Exponential family harmoniums with an application to information retrieval". In: NIPS-14. 2004.
- 32 [WS01] C. K. I. Williams and M. Seeger. "Using the Nyström Method to Speed Up Kernel Machines". In: NIPS. Ed. by T. K. Leen, T. G. Dietterich, and V. Tresp. MIT Press, 2001, pp. 682-688.
- 33 [WS05] M. Welling and C. Sutton. "Learning in Markov Random Fields with Contrastive Free Energies". In: Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS). 2005.
- 34 [WS93] D. Wolpert and C. Strauss. "What Bayes has to say about the evidence procedure". In: Proc. Workshop on Maximum Entropy and Bayesian methods. 1993.
- 35 [WSG21] C. Wang, S. Sun, and R. Grosse. "Beyond Marginal Uncertainty: How Accurately can Bayesian Regression Models Estimate Posterior Predictive Correlations?". In: AISTATS. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 2476-2484.
- 36 [WSN00] B. Williams, T. Santner, and W. Notz. "Sequential design of computer experiments to minimize integrated response functions". In: Statistica Sinica 10 (2000), pp. 1133-1152.
- 37 [WT01] M. Welling and Y.-W. Teh. "Belief Optimization for Binary Networks: a Stable Alternative to Loopy Belief Propagation". In: UAI. 2001.
- 38 [WT19] R. Wen and K. Torkkola. "Deep Generative Quantile-Copula Models for Probabilistic Forecasting". In: ICML. 2019.
- 39 [WT90] G. Wei and M. Tanner. "A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms". In: JASA 85.411 (1990), pp. 699-704.
- 40 [WTB20] Y. Wen, D. Tran, and J. Ba. "BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning". In: ICLR. 2020.
- 41 [WTN19] Y. Wu, G. Tucker, and O. Nachum. Behavior Regularized Online Reinforcement Learning. arXiv:1911.11361. 2019.
- 42 [WU+06] Y. Wu, D. Hu, M. Wu, and X. Hu. "A Numerical Integration Perspective on Gaussian Filters". In: IEEE Trans. Signal Process. 54.8 (Aug. 2006), pp. 2910-2921.
- 43 [WU+17] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. "Scalable region method for deep reinforcement learning using Kronecker-factored approximation". In: NIPS. 2017.
- 44 [WU+19a] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernández-Lobato, and A. L. Gaunt. "Fixing Variational Bayes: Deterministic Variational Inference for Bayesian Neural Networks". In: ICLR. 2019.
- 45 [WU+19b] M. Wu, S. Parhoo, M. Hughes, R. Kindle, L. Celi, M. Zazzi, V. Roth, and F. Doshi-Velez. "Regional tree regularization for interpretability in black box models". In: AAAI (2019).
- 46 [WU+21] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld. "Polyjuice: Generating Counterfactuals for Explaining, Evaluating, and Improving Models". In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2021.
- 47 [Wüt+16] M. Wüthrich, S. Trimpe, C. García Cifuentes, D. Kappler, and S. Schaal. "A new perspective and extension of the Gaussian Filter". en. In: The International Journal of Robotics Research 35.14 (Dec. 2016), pp. 1731-1749.
- 48 [WW12] Y. Wu and D. P. Wipf. "Dual-Space Analysis of the Sparse Linear Model". In: NIPS. 2012.
- 49 [WY02] D. Wilkinson and S. Yeung. "Conditional simulation from highly structured Gaussian systems with application to blocking-MCMC for the Bayesian analysis of very large linear models". In: Statistics and Computing 12 (2002), pp. 287-300.
- 50 [WYG14] J. Wan, C.-N. Yu, and R. Greiner. "Robust Learning under Uncertain Test Distributions: Relating Covariate Shift to Model Misspecification". In: ICML. Vol. 32. Proceedings of Machine Learning Research. Beijing, China: PMLR, 2014, pp. 631-639.
- 51 [WZ19] J. Wei and K. Zou. "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks". In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Conference on Oral Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 6382-6388.

- 1
- 2 [WZR20] S. Wu, H. R. Zhang, and C. Ré. "Understanding and
Improving Information Transfer in Multi-Task Learning". In:
International Conference on Learning Representations. 2020.
- 3 [XC19] Z. Xia and A. Chakrabarti. "Training Image Estimators
without Image Ground-Truth". In: NIPS. 2019.
- 4 [XD18] J. Xu and G. Durrett. "Spherical Latent Spaces for Sta-
ble Variational Autoencoders". In: EMNLP. 2018.
- 5 [Xia+21] K. Xiao, L. Engstrom, A. Ilyas, and A. Madry. "Noise
or Signal: The Role of Image Backgrounds in Object Recogni-
tion". In: ICLR. 2021.
- 6 [Xie+16] J. Xie, Y. Lu, S.-C. Zhu, and Y. N. Wu. "A Theory of
Generative ConvNet". In: ICML. 2016.
- 7 [Xie+18] J. Xie, Y. Lu, R. Gao, and Y. N. Wu. "Cooperative
Learning of Energy-Based Model and Latent Variable Model
via MCMC Teaching." In: AAAI. Vol. 1. 6. 2018, p. 7.
- 8 [XJ96] L. Xu and M. I. Jordan. "On Convergence Properties of
the EM Algorithm for Gaussian Mixtures". In: Neural Compu-
tation 8 (1996), pp. 129–151.
- 9 [Xu+06] Z. Xu, V. Tresp, K. Yu, and H.-P. Kriegel. "In nite
hidden relational models". In: UAI. 2006.
- 10 [Xu+07] Z. Xu, V. Tresp, S. Yu, K. Yu, and H.-P. Kriegel. "Fast
Inference in In nite Hidden Relational Models". In: Workshop
on Mining and Learning with Graphs. 2007.
- 11 [Xu+15] J. Xu, L. Mukherjee, Y. Li, J. Warner, J. M. Rehg,
and V. Singh. "Gaze-enabled egocentric video summarization
via constrained submodular maximization". In: Proceedings of
the IEEE conference on computer vision and pattern recogni-
tion. 2015, pp. 2235–2244.
- 12 [Xu+19] M. Xu, M. Quiroz, R. Kohn, and S. A. Sisson. "Vari-
ance reduction properties of the reparameterization trick". In:
AISTATS. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. Pro-
ceedings of Machine Learning Research. PMLR, 2019, pp. 2711–
2720.
- 13 [Yad+18] S. Yadlowsky, H. Namkoong, S. Basu, J. Duchi, and
L. Tian. "Bounds on the conditional and average treatment ef-
fect with unobserved confounding factors". In: arXiv e-prints,
arXiv:1808.09521 (Aug. 2018), arXiv:1808.09521. arXiv:
1808 . 09521 [stat.ME].
- 14 [Yad+21] S. Yadlowsky, S. Fleming, N. Shah, E. Brunskill, and
S. Wager. Evaluating Treatment Prioritization Rules via Rank-
Weighted Average Treatment Effects. 2021. arXiv:
2111 . 07966 [stat.ME].
- 15 [Yan+17] S. Yang, L. Xie, X. Chen, X. Lou, X. Zhu, D. Huang, and
H. Li. "Statistical parametric speech synthesis using generative
adversarial networks under a multi-task learning framework".
In: IEEE Automatic Speech Recognition and Understanding
Workshop (ASRU). Dec. 2017, pp. 685–691.
- 16 [Yan19] G. Yang. "Scaling Limits of Wide Neural Networks with
Weight Sharing: Gaussian Process Behavior, Gradient Independence,
and Neural Tangent Kernel Derivation". In: (Feb. 2019),
arXiv: 1902.04760 [cs.NE].
- 17 [Yan+21] J. Yang, K. Zhou, Y. Li, and Z. Liu. "Generalized
OOD Detection: A Survey". In: (2021).
- 18 [Yan81] M. Yannakakis. "Computing the minimum ℓ_1 -in is NP-
complete". In: SIAM J. Alg. Discrete Methods 2 (1981), pp. 77–
79.
- 19 [Yao+18a] Y. Yao, A. Vehtari, D. Simpson, and A. Gelman.
"Using Stacking to Average Bayesian Predictive Distributions
(with Discussion)". en. In: Bayesian Analysis 13.3 (Sept. 2018),
pp. 917–1007.
- 20 [Yao+18b] Y. Yao, A. Vehtari, D. Simpson, and A. Gelman.
"Yes, but Did It Work?: Evaluating Variational Inference". In:
ICML. Vol. 80. Proceedings of Machine Learning Research.
PMLR, 2018, pp. 5581–5590.
- 21 [YBM20] Y. Yang, R. Bamler, and S. Mandt. Improving Infer-
ence for Neural Image Compression. 2020. arXiv:
1906 . 04240 [eess.IV].
- 22 [YBS11] P. Yadollahpour, D. Batra, and G. Shakhnarovich. "Di-
verse M-best Solutions in MRFs". In: NIPS workshop on Dis-
crete Optimization in Machine Learning. 2011.
- 23 [YBW15] F. Yang, S. Balakrishnan, and M. J. Wainwright. "Sta-
tistical and Computational Guarantees for the Baum-Welch Al-
gorithm". In: (2015), arXiv: 1512.08269 [stat.ML].
- 24 [Ye+19] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang. "BP-
Transformer: Modelling Long-Range Context via Binary Parti-
tioning". In: (Nov. 2019), arXiv: 1911.04070 [cs.CL].
- 25 [Yed11] J. S. Yedidia. "Message-Passing Algorithms for Infer-
ence and Optimization". In: J. Stat. Phys. 145.4 (Nov. 2011),
pp. 860–890.
- 26 [Yeh+18] C.-K. Yeh, J. S. Kim, I. E. H. Yen, and P. Ravikumar.
"Representer Point Selection for Explaining Deep Neural
Networks". In: 2018. arXiv: 1811.09720 [cs.LG].
- 27 [Yeh+19a] C.-K. Yeh, C.-Y. Hsieh, A. Sugiyama, D. I. Inouye, and
P. K. Ravikumar. "On the (in) dentity and sensitivity of expla-
nations". In: Advances in Neural Information Processing Sys-
tems 32 (2019), pp. 10967–10978.
- 28 [Yeh+19b] C.-K. Yeh, B. Kim, S. O. Arik, C.-L. Li, T. Ps-
ter, and P. Ravikumar. "On completeness-aware concept-based
explanations in deep neural networks". In: arXiv preprint
arXiv:1910.07969 (2019).
- 29 [Yeu+17] S. Yeung, A. Kannan, Y. Dauphin, and L. Fei-
Fei. "Tackling Over-pruning in Variational Autoencoders". In:
ICML Workshop on "Principled Approaches to Deep Learning".
June 2017.
- 30 [Yeu91a] R. W. Yeung. "A new outlook on Shannon's informa-
tion measures". In: IEEE Trans. Inf. Theory 37.3 (May 1991),
pp. 466–474.
- 31 [Yeu91b] R. W. Yeung. "A new outlook on Shannon's informa-
tion measures". In: IEEE Trans. on Information Theory 37
(1991), pp. 466–474.
- 32 [YFW00] J. Yedidia, W. T. Freeman, and Y. Weiss. "General-
ized Belief Propagation". In: NIPS. 2000.
- 33 [Yin+19a] D. Yin, R. G. Lopes, J. Shiels, E. D. Cubuk, and J.
Gilmer. "A Fourier Perspective on Model Robustness in Com-
puter Vision". In: NIPS. 2019.
- 34 [Yin+19b] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin.
"Understanding Straight-Through Estimator in Training Acti-
vation Quantized Neural Nets". In: ICLR. 2019.
- 35 [Yin+19c] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J.
Leskovec. "Grnexplainer: Generating explanations for graph
neural networks". In: Advances in neural information process-
ing systems 32 (2019), p. 9240.
- 36 [Yin+20] D. Yin, M. Farajtabar, A. Li, N. Levine, and A. Mott.
"Optimization and Generalization of Regularization-Based Con-
tinual Learning: a Loss Approximation Viewpoint". In: (June
2020), arXiv: 2006.10974 [cs.LG].
- 37 [YK06] A. Yuille and D. Kersten. "Vision as Bayesian inference:
analysis by synthesis?" en. In: Trends Cogn. Sci. 10.7 (July
2006), pp. 301–308.
- 38 [YK19] M. Yang and B. Kim. Benchmarking Attribution Meth-
ods with Relative Feature Importance. 2019. arXiv:
1907 . 09701 [cs.LG].
- 39 [YMT22] Y. Yang, S. Mandt, and L. Theis. "An Introduction to
Neural Data Compression". In: (Feb. 2022). arXiv:
2202 . 06533 [cs.LG].
- 40 [Yoo+18] K. Yoon, R. Liao, Y. Xiong, L. Zhang, E. Fetaya, R.
Urtasun, R. Zemel, and X. Pitkow. "Inference in Probabilistic
Graphical Models by Graph Neural Networks". In: ICLR Work-
shop. 2018.
- 41 [You19] A. Young. "Consistency without inference: Instrumen-
tal variables in practical application". In: (2019).
- 42 [Youn89] L. Younes. "Parameter estimation for imperfectly ob-
served Gibbs elds". In: Probab. Theory and Related Fields
82 (1989), pp. 625–645.
- 43 [Youn99] L. Younes. "On the convergence of Markovian stochas-
tic algorithms with rapidly decreasing ergodicity rates". In:
Stochastics: An International Journal of Probability and
Stochastic Processes 63.3-4 (1999), pp. 177–228.
- 44 [Yu+06] S. Yu, K. Yu, V. Tresp, K. H.-P., and M. Wu. "Su-
pervised probabilistic principal component analysis". In: KDD.
2006.
- 45 [Yu+10] S.-Z. Yu. "Hidden Semi-Markov Models". In: Artif
Intelligence J. 174.2 (2010).
- 46 [Yu+16] F. X. Yu, A. T. Suresh, K. M. Choromanski, D. N.
Holtmann-Rice, and S. Kumar. "Orthogonal Random Features".
In: NIPS. Curran Associates, Inc., 2016, pp. 1975–1983.
- 47 [Yu+17] L. Yu, W. Zhang, J. Wang, and Y. Yu. "Seqgan: Se-
quence generative adversarial nets with policy gradient". In:
Thirtyrst AAAI conference on artifcial intelligence. 2017.
- 48 [Yu+18] Y. Yu, et al. "Dynamic Control Flow in Large-Scale
Machine Learning". In: Proceedings of the Thirteenth EuroSys
Conference, EuroSys '18, Porto, Portugal: Association for Com-
puting Machinery, 2018.
- 49 [Yu+20] L. Yu, Y. Song, J. Song, and S. Ermon. "Training
Deep Energy-Based Models with f-Divergence Minimization".
In: arXiv preprint arXiv:2003.03463 (2020).
- 50 [Yu+21] J. Yu, X. Li, J. Y. Koh, H. Zhang, R. Pang, J. Qin,
A. Ku, Y. Xu, J. Baldridge, and Y. Wu. "Vector-quantized Im-
age Modeling with Improved VQGAN". In: (Oct. 2021), arXiv:
2110.04627 [cs.CV].
- 51 [Yu+19] X. Yuan, P. He, Q. Zhu, and X. Li. "Adversarial Ex-
amples: Attacks and Defenses for Deep Learning". en. In: IEEE
Trans. Neural Networks and Learning Systems 30.9 (Sept.
2019), pp. 2805–2824.
- 52 [Yui01] A. Yuille. "CCCP algorithms to minimize the Bethe and
Kikuchi free energies: convergent alternatives to belief propa-
gation". In: Neural Computation 14 (2001), pp. 1691–1722.
- 53 [YW04] C. Yanover and Y. Weiss. "Finding the M Most Prob-
able Conurations in Arbitrary Graphical Models". In: NIPS.
2004.

BIBLIOGRAPHY

- [WX17] J.-g. Yao, X. Wan, and J. Xiao. "Recent advances in document summarization". In: Knowledge and Information Systems 53.2 (2017), pp. 297–336.
- [YZ19] G. Yaroslavtsev and S. Zhou. "Approximate F -Sketching of Valuation Functions". In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019). Ed. by D. Achlioptas and L. A. Végh. Vol. 145. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 69:1–69:21.
- [YZ22] Y. Yang and P. Zhai. "Click-through rate prediction in online advertising: A literature review". In: Inf. Process. Manag. 59.2 (Mar. 2022), p. 102853.
- [ZA12] J. Zou and R. Adams. "Priors for Diversity in Generative Latent Variable Models". In: NIPS. 2012.
- [Zaf+22] M. Zaran, A. Dieuleveut, O. Féron, Y. Goude, and J. Josse. "Adaptive Conformal Predictions for Time Series". In: (Feb. 2022). arXiv: 2202.07282 [stat.ML].
- [Zai+20] S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and Y. W. Teh. "Neural Ensemble Search for Performant and Calibrated Predictions". In: (June 2020). arXiv: 2006.08573 [cs.LG].
- [Zan21] N. Zanichelli. IAML Distill Blog: Transformers in Vision. 2021.
- [ZB18] T. Zhou and J. Bilmes. "Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity". In: International Conference on Learning Representations. 2018.
- [ZB21] B. Zhao and H. Bilen. "Dataset Condensation with Differentiable Siamese Augmentation". In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18–24 July 2021, Virtual Event. Ed. by M. Meila and T. Z. 0001, Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 12674–12685.
- [ZCC07] G. Zhang, T. Chen, and X. Chen. "Performance Recovery in Digital Implementation of Analogue Systems". In: SIAM J. Control Optim. 45.6 (Jan. 2007), pp. 2207–2233.
- [ZDK15] J. Zhang, J. Djolonga, and A. Krause. "Higher-order inference for multi-class log-supermodular models". In: Proceedings of the IEEE International Conference on Computer Vision. 2015, pp. 1859–1867.
- [ZE01a] B. Zadrozny and C. Elkan. "Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers". In: KDD. 2001.
- [ZE01b] B. Zadrozny and C. Elkan. "Transforming classifier scores into accurate multiclass probability estimates". In: KDD. 2001.
- [Zec+18] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann. "Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study". In: PLoS medicine 15.11 (2018), e1002683.
- [Zel76] A. Zellner. "Bayesian and non-Bayesian analysis of the regression model with multivariate Student-t error terms". In: JASA 71.354 (1976), pp. 400–405.
- [Zel86] A. Zellner. "On assessing prior distributions and Bayesian regression analysis with g-prior distributions". In: Bayesian inference and decision techniques, Studies of Bayesian and Econometrics and Statistics volume 6. North Holland, 1986.
- [Zen+18] C. Zeno, I. Golan, E. Hoer, and D. Soudry. "Task-Agnostic Continual Learning Using Online Variational Bayes". In: (Mar. 2018). arXiv: 1803.10123 [stat.ML].
- [Zen+21] C. Zeno, I. Golan, E. Hoer, and D. Soudry. "Task-Agnostic Continual Learning Using Online Variational Bayes With Fixed-Point Updates". en. In: Neural Comput. 33.11 (Oct. 2021), pp. 3139–3177.
- [Zer+19] J. Zerilli, A. Knott, J. MacLaurin, and C. Gavaghan. "Transparency in algorithmic and human decision-making: Is there a double standard?". In: Philosophy & Technology 32.4 (2019) (2019), pp. 661–683.
- [ZF14] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks". In: European conference on computer vision. Springer, 2014, pp. 818–833.
- [ZFV20] G. Zeni, M. Fontana, and S. Vantini. "Conformal Prediction: A Uni Review of Theory and New Challenges". In: (May 2020). arXiv: 2005.07972 [cs.LG].
- [ZG21] D. Zou and Q. Gu. "On the Convergence of Hamiltonian Monte Carlo with Stochastic Gradients". In: ICML. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 13012–13022.
- [ZGH10] X. Zhang, T. Graepel, and R. Herbrich. "Bayesian Online Learning for Multi-label and Multi-variate Performance Measures". In: AISTATS. 2010.
- [ZGR21] L. Zhang, M. Goldstein, and R. Rangapuram. "Understanding Failures in Out-of-Distribution Detection with Deep Generative Models". In: ICML. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 12427–12436.
- [ZH05] O. Zoeter and T. Heskes. "Gaussian Quadrature Based Expectation Propagation". In: AISTATS. 2005.
- [Zha+13a] K. Zhang, B. Schölkopf, K. Muandet, and Z. Wang. "Domain Adaptation under Target and Conditional Shift". In: Proceedings of the 30th International Conference on Machine Learning. 2013, pp. 819–827.
- [Zha+13b] K. Zhang, B. Schölkopf, K. Muandet, and Z. Wang. "Domain Adaptation under Target and Conditional Shift". In: ICML. Vol. 28. 2013.
- [Zha+16] S. Zhai, Y. Cheng, R. Feris, and Z. Zhang. "Generative adversarial networks as variational training of energy based models". In: arXiv preprint arXiv:1611.01799 (2016).
- [Zha+17] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. "Understanding deep learning requires rethinking generalization". In: ICLR. 2017.
- [Zha+18a] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse. "Noisy Natural Gradient as Variational Inference". In: ICML. 2018.
- [Zha+18b] Y. Zhang, A. M. Saxe, M. S. Advani, and A. A. Lee. "Energy-entropy competition and the effectiveness of stochastic gradient descent in machine learning". In: (Mar. 2018). arXiv: 1803.01927 [cs.LG].
- [Zha+19a] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt. "Advances in Variational Inference". In: IEEE PAMI (2019), pp. 2008–2026.
- [Zha+19b] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. "Self-attention generative adversarial networks". In: International conference on machine learning. PMLR, 2019, pp. 7354–7363.
- [Zha+19c] L. Zhao, K. Korovina, W. Si, and M. Cheung. "Approximate inference with Graph Neural Networks". 2019.
- [Zha+20a] A. Zhang, Z. Lipton, M. Li, and A. Smola. Dive into deep learning. 2020.
- [Zha+20b] H. Zhang, A. Li, J. Guo, and Y. Guo. "Hybrid models for open set recognition". In: European Conference on Computer Vision. Springer, 2020, pp. 102–117.
- [Zha+20d] R. Zhang, B. Dai, L. Li, and D. Schuurmans. "GenDICE: Generalized DICE Estimation of Stationary Values". In: ICLR. 2020.
- [Zha+20e] X. Zhang, Y. Li, Z. Zhang, and Z.-L. Zhang. "f-GAIL: Learning f-Divergence for Generative Adversarial Imitation Learning". In: Neural Information Processing Systems (2020).
- [Zha+21] H. Zhang, J. Y. Koh, J. Baldridge, H. Lee, and Y. Yang. "Cross-Model Contrastive Learning for Text-to-Image Generation". In: CVPR. 2021.
- [Zhe+15] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Viñet, Z. Su, D. Du, C. Huang, and P. Torr. "Conditional Random Fields as Recurrent Neural Networks". In: ICCV. 2015.
- [Zho+19a] S. Zhou, M. Gordon, R. Krishna, A. Narcomey, L. Fei-Fei, and M. Bernstein. "HYPE: A Benchmark for Human Eye Perceptual Evaluation of Generative Models". In: NIPS. Curran Associates, Inc., 2019, pp. 3444–3456.
- [Zho+19b] S. Zhou, M. L. Gordon, R. Krishna, A. Narcomey, L. Fei-Fei, and M. S. Bernstein. "HYPE: a benchmark for human eye perceptual evaluation of generative models". In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. 2019, pp. 3449–3461.
- [Zho20] G. Zhou. "Mixed Hamiltonian Monte Carlo for Mixed Discrete and Continuous Variable". In: (2020). arXiv: 1909.04852
- [Zho+20] Y. Zhou, H. Yang, Y. W. Teh, and T. Rainforth. "Divide, Conquer, and Combine: a New Inference Strategy for Probabilistic Programs with Stochastic Support". In: ICML. Ed. by H. D. Ilii and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 11534–11545.
- [ZHT06] H. Zou, T. Hastie, and R. Tibshirani. "Sparse principal component analysis". In: JCGS 15.2 (2006), pp. 262–286.
- [Zhu+17] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: ICCV. 2017.
- [Zhu+18] X. Zhu, A. Singla, S. Zilles, and A. Aferry. "An overview of machine teaching". In: arXiv preprint arXiv:1801.05927 (2018).
- [Zhu+21] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. "A Comprehensive Survey on Transfer Learning". In: Proc. IEEE 109.1 (2021).
- [Zie+08] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. "Maximum Entropy Inverse Reinforcement Learning". In: AAAI. 2008, pp. 1433–1438.
- [Zin+20] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. "VarIBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: ICLR. 2020.

- 1
- 2 [ZLF21] M. Zhang, S. Levine, and C. Finn. "MEMO: Test Time
Robustness via Adaptation and Augmentation". In: (Oct. 2021).
arXiv: [2110.09505](https://arxiv.org/abs/2110.09505) [cs.LG].
- 3 [ZLG20] R. Zivan, O. Lev, and R. Galik. "Beyond Trees: Analysis
and Convergence of Belief Propagation in Graphs with Multiple
Cycles". In: AAAI. 2020.
- 4 [ZMB21] B. Zhao, K. R. Mopuri, and H. Bilen. "Dataset Condensation
with Gradient Matching". In: International Conference on Learning Representations. 2021.
- 5 [ZMG19] G. Zhang, J. Martens, and R. B. Grosse. "Fast Convergence of Natural Gradient Descent for Over-Parameterized Neural Networks". In: NIPS. 2019, pp. 8082–8093.
- 6 [ZML16] J. J. Zhao, M. Mathieu, and Y. LeCun. "Energy-based
Generative Adversarial Network". In: (2016).
- 7 [ZN20] Y. Zhang and E. Nalisnick. "On the inconsistency of
Bayesian inference for misspecified neural networks". In: 3rd Symposium on Advances in Approximate Bayesian Inference.
2020.
- 8 [Zob09] O. Zobay. "Mean field inference for the Dirichlet process mixture model". In: Electronic J. of Statistics 3 (2009),
pp. 507–545.
- 9 [Zoe07] O. Zoeter. "Bayesian generalized linear models in a terabyte world". In: Proc. 5th International Symposium on Image
and Signal Processing and Analysis. 2007.
- 10 [Zon+18] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu,
D. Cho, and H. Chen. "Deep Autoencoding Gaussian Mixture
Model for Unsupervised Anomaly Detection". In: ICLR. 2018.
- 11 [ZP00] G. Zweig and M. Padmanabhan. "Exact alpha-beta com-
putation in logarithmic space with application to map word
graph construction". In: ICMLP. 2000.
- 12 [ZP96] N. Zhang and D. Poole. "Exploiting causal independence
in Bayesian network inference". In: JAIR (1996), pp. 301–328.
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- [ZR19a] Z. Ziegler and A. Rush. "Latent Normalizing Flows for Discrete Sequences". In: Proceedings of the 36th International Conference on Machine Learning. 2019, pp. 7673–7682.
- [ZR19b] Z. M. Ziegler and A. M. Rush. "Latent Normalizing Flows for Discrete Sequences". In: ICML. 2019.
- [ZS11] Y. Zhang and C. Sutton. "Quasi-newton methods for Markov chain Monte Carlo". In: NIPS. 2011.
- [ZSB19] Q. Zhao, D. S. Small, and B. B. Bhattacharya. "Sensitivity analysis for inverse probability weighting estimators via the percentile bootstrap". In: Journal of the Royal Statistical Society: Series B (Statistical Methodology) 81.4 (2019), pp. 735–761. eprint: <https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12327>.
- [ZSE17] S. Zhao, J. Song, and S. Ermon. "Towards a Deeper Understanding of Variational Autoencoding Models". In: (Feb. 2017). arXiv: [1702.08658](https://arxiv.org/abs/1702.08658) [cs.LG].
- [ZSE19] S. Zhao, J. Song, and S. Ermon. "InfoVAE: Information Maximizing Variational Autoencoders". In: AAAI. 2019.
- [ZVP21] J. A. Zavatone-Veth and C. Pehlevan. "Exact marginal prior distributions of finite Bayesian neural networks". In: NIPS. May 2021.
- [ZW11] D. Zoran and Y. Weiss. "From learning models of natural image patches to whole image restoration". In: ICCV. 2011.
- [ZW12] D. Zoran and Y. Weiss. "Natural Images, Gaussian Mixtures and Dead Leaves". In: NIPS. 2012, pp. 1736–1744.
- [ZY21] Y. Zhang and Q. Yang. "A Survey on Multi-Task Learning". In: IEEE Trans. Knowl. Data Eng. (2021).
- [ZY97] Z. Zhang and R. W. Yeung. "A non-shannon-type conditional inequality of information quantities". In: IEEE Transactions on Information Theory 43.6 (1997), pp. 1982–1986.
- [ZY98] Z. Zhang and R. W. Yeung. "On characterization of entropy function via information inequalities". In: IEEE Transactions on Information Theory 44.4 (1998), pp. 1440–1452.