DISCLAMER: This summary has been written based on the Lecture 5263-5210-00 L Probabilistic Artificial Intelligence by Prof. A. Krause (Fall 2023). There is no guarantee for completeness and/or correctness regarding the content of this summary. Use it at your own discretion

Probabilistic Artificial Intelligence Summary

Autumn Semester 2023, ETH Zurich

Dr. Linda De Cave, Andrea Ghirlanda, Michael Etienne Van Huffel

Chapter 1

Preliminaries

1.1 Useful results

Theorem 1 (Jensen's Inequality). *Given a random variable $X : \Omega \to \mathbb{R}$ and a convex function $g : \mathbb{R} \to \mathbb{R}$, we have*

$$g(\mathbb{E}[X]) \leq \mathbb{E}[g(X)].$$

For a concave function $h, -h$ is convex, and hence

$$h(\mathbb{E}[X]) \geq \mathbb{E}[h(X)].$$

Definition 1. *A random variable $X : \Omega \to \mathbb{R}$ is called σ-*SUB-GAUSSIAN *for $\sigma > 0$ if for all $\lambda \in \mathbb{R}$,*

$$\mathbb{E}\left[e^{\lambda(X-\mathbb{E}X)}\right] \leq \exp\left(\frac{\sigma^2\lambda^2}{2}\right).$$

Intuitively, a random variable is sub-Gaussian if its tail probabilities decay at least as fast as those of a Gaussian.

Theorem 2 (Hoeffding's inequality). *Let $X : \Omega \to \mathbb{R}$ be a σ-subGaussian random variable. Then,*

$$\mathbb{P}\left(\left|\bar{X}_n - \mathbb{E}X\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right). \tag{1.1}$$

In words, the absolute error of the expectation estimate is bounded by an exponentially quickly decaying error probability δ. Solving for n, we obtain that for

$$n \geq \frac{2\sigma^2}{\epsilon^2}\log\frac{2}{\delta}$$

the probability that the absolute error is greater than ϵ is at most δ.

Theorem 3 (Strong law of large numbers, SLLN). *Given the random variable $X : \Omega \to \mathbb{R}$ with independent samples $x_i \sim X$ and finite variance. Let f be a "nice" function and $Y \doteq f(X)$. We have,*

$$\bar{Y}_n = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \overset{a.s.}{\to} \mathbb{E}[f(X)], \tag{1.2}$$

as $n \to \infty$. Here, $\bar{Y}_n$ denotes the sample mean of $Y_1, \ldots, Y_n$ and $\overset{a.s.}{\to}$ denotes the CONVERGENCE WITH PROBABILITY 1 *or* CONVERGENCE ALMOST SURELY*, namely Y_n converges to Y almost surely if it results*

$$\mathbb{P}\left(\left\{\omega \in \Omega : \lim_{n\to\infty} X_n(\omega) = X(\omega)\right\}\right) = 1.$$

Theorem 4 (Central limit theorem). *Given independent and identically distributed random variables $X_1, \ldots, X_n$ with finite expectation μ and variance σ^2. Then,*

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{n}(X_i - \mu) = \sqrt{n}\left(\bar{X}_n - \mu\right) \overset{\mathcal{D}}{\to} \mathcal{N}\left(0, \sigma^2\right) \quad \text{as } n \to \infty$$

Here, $\bar{X}_n$ denotes the sample mean of $X_1, \ldots, X_n$ and $\xrightarrow{\mathcal{D}}$ denotes the CONVERGENCE IN DISTRIBUTION*, namely X_n converges to X in distribution if for all points $x \in X(\Omega)$ at which P_X (the CDF of X) is continuous, it results*

$$\lim_{n\to\infty} P_{X_n}(x) = P_X(x).$$

Proposition 1 (Sum Rule)**.** *We have that*

$$p\left(x_{1:i-1}, x_{i+1:n}\right) = \sum_{x_i \in X_i(\Omega)} p\left(x_{1:i-1}, x_i, x_{i+1:n}\right), \quad p\left(x_{1:i-1}, x_{i+1:n}\right) = \int_{X_i(\Omega)} p\left(x_{1:i-1}, x_i, x_{i+1:n}\right) dx_i.$$

if X_i is discrete and continuous respectively.

Proposition 2 (Product Rule)**.** *Given random variables $X_{1:n}$,*

$$p\left(x_{1:n}\right) = p\left(x_1\right) \cdot \prod_{i=2}^{n} p\left(x_i \mid x_{1:i-1}\right).$$

Proposition 3 (Law of Total Probability)**.** *Combining sum rule and product rule, if $\mathbf{Y}$ is continuous, then*

$$p(\boldsymbol{x}) = \int_{\mathbf{Y}(\Omega)} p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} = \int_{\mathbf{Y}(\Omega)} p(\boldsymbol{x} \mid \boldsymbol{y}) \cdot p(\boldsymbol{y}) d\boldsymbol{y} \tag{1.3}$$

Analogously, one can condition on a discrete random variable by replacing the integral with a sum.

Theorem 5 (Law of total variance, LOTV)**.**

$$\operatorname{Var}[\mathbf{X}] = \mathbb{E}[\operatorname{Var}[\mathbf{X} \mid \mathbf{Y}]] + \operatorname{Var}[\mathbb{E}[\mathbf{X} \mid \mathbf{Y}]]$$

1.2 Common discrete distributions

Bernoulli. $\operatorname{Bern}(p)$ describes (biased) coin flips. Its domain is $\Omega = \{0, 1\}$ where 1 corresponds to heads and 0 corresponds to tails. $p \in [0, 1]$ is the probability of the coin landing heads, that is, $\mathbb{P}(X = 1) = p$ and $\mathbb{P}(X = 0) = 1 - p$.

Binomial. $\operatorname{Bin}(n, p)$ counts the number of heads in n independent Bernoulli trials, each with probability of heads p.

Categorical $\operatorname{Cat}\left(m, p_1, \ldots, p_m\right)$**.** It is a generalization of the Bernoulli distribution and represents throwing a (biased) m-sided die. Its domain is $\Omega = [m]$ and we have $\mathbb{P}(X = i) = p_i$. We require $p_i \geq 0$ and $\sum_{i \in [m]} p_i = 1$.

Multinomial. $\operatorname{Mult}\left(n, m, p_1, \ldots, p_m\right)$ It counts the number of outcomes of each side in n independent Categorical trials.

Uniform Unif(S) assigns identical probability mass to all values in the set S. That is, $\mathbb{P}(X = x) = \frac{1}{|S|} (\forall x \in S)$.

1.3 Gaussian distribution and Multivariate Gaussian distribution

An example of a continuous distribution is the NORMAL DISTRIBUTION, also called GAUSSIAN. We say, a random variable X is normally distributed, $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$, if its PDF is

$$\mathcal{N}\left(x; \mu, \sigma^2\right) \doteq \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{1.4}$$

We have $\mathbb{E}[X] = \mu$ and $\operatorname{Var}[X] = \sigma^2$. If $\mu = 0$ and $\sigma^2 = 1$, this distribution is called the STANDARD NORMAL DISTRIBUTION. The Gaussian CDF cannot be expressed in closed-form, and is given by

$$F(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \quad \text{where } \operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt$$

Note that the mean of a Gaussian distribution coincides with the maximizer of its PDF, also called mode of a distribution.

Much of the relevance of the normal distribution stems from the central limit theorem, which states that the sum of any i.i.d. (independent and identically distributed) samples tends to a normal distribution as the sample size goes to infinity, even if the samples themselves are not normally distributed (see Theorem 4).

A random vector $\mathbf{X}$ in $\mathbb{R}^n$ is normally distributed, $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, if its PDF is

$$\mathcal{N}(x; \mu, \Sigma) \doteq \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{\det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(x-\mu)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean and

$$\boldsymbol{\Sigma} \doteq \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1n} \\ \vdots & & & \vdots \\ \sigma_{n1} & \sigma_{n2} & \dots & \sigma_n^2 \end{pmatrix} \in \mathbb{R}^{n\times n}$$

is the covariance matrix.
$\mathbf{X}$ is also called a GAUSSIAN RANDOM VECTOR (GRV). $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is the MULTIVARIATE STANDARD NORMAL DISTRIBUTION.
In this definition, we assume that the covariance matrix $\boldsymbol{\Sigma}$ is invertible, i.e., does not have the eigenvalue 0.
Note that a Gaussian can be represented using only $\mathcal{O}\left(n^2\right)$ parameters. In the case of a diagonal covariance matrix we just need $\mathcal{O}(n)$ parameters.

1.4 Properties of Gaussians

Theorem 6 (Closedness of Gaussians under conditioning and marginalization)**.** *Consider the Gaussian random vector $\mathbf{X}$ and fix index sets $A \subseteq [n]$ and $B \subseteq [n]$. Then, we have that for any such marginal distribution,*

$$\mathbf{X}_A \sim \mathcal{N}\left(\boldsymbol{\mu}_A, \boldsymbol{\Sigma}_{AA}\right),$$

and that for any such conditional distribution,

$$\begin{aligned} \mathbf{X}_A \mid \mathbf{X}_B = \boldsymbol{x}_B &\sim \mathcal{N}\left(\boldsymbol{\mu}_{A|B}, \boldsymbol{\Sigma}_{A|B}\right) \quad \textit{where} \\ \boldsymbol{\mu}_{A|B} &\doteq \boldsymbol{\mu}_A + \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\left(\boldsymbol{x}_B - \boldsymbol{\mu}_B\right), \\ \boldsymbol{\Sigma}_{A|B} &\doteq \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA}. \end{aligned} \tag{1.5}$$

Observe that, when conditioning, the variance can only shrink. Moreover, how much the variance is reduced depends purely on where the observations are made (e.g., the choice of B) but not on what the observations are. Also observe that $\mu_{A|B}$ depends affinely on μ_B.

Theorem 7 (Closedness of Gaussians under affine transformations)**.** *Given an n-dimensional Gaussian $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$, then*

$$\boldsymbol{A}\boldsymbol{X} + \boldsymbol{b} \sim \mathcal{N}\left(\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top\right). \tag{1.6}$$

Theorem 8 (Sums of Gaussians are Gaussian)**.** *Given two independent Gaussian random vectors $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathbf{X}' \sim \mathcal{N}\left(\boldsymbol{\mu}', \boldsymbol{\Sigma}'\right)$ in $\mathbb{R}^n$,*

$$\mathbf{X} + \mathbf{X}' \sim \mathcal{N}\left(\boldsymbol{\mu} + \boldsymbol{\mu}', \boldsymbol{\Sigma} + \boldsymbol{\Sigma}'\right).$$

1.5 Conditional Linear Gaussians

It is often useful to consider a Gaussian random variable X in terms of another Gaussian random variable Y. Suppose $X \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right)$ and $Y \sim \mathcal{N}\left(\mu_Y, \sigma_Y^2\right)$ are jointly Gaussian. Then, using the closed-form expression of

Gaussian conditionals (1.5), their conditional distribution is given as

$$\begin{aligned} X \mid Y = y &\sim \mathcal{N}\left(\mu_{X|Y}, \sigma^2_{X|Y}\right) \quad \text{where} \\ \mu_{X|Y} &\doteq \mu_X + \sigma_{XY}\sigma_Y^{-2}\left(y - \mu_Y\right), \\ \sigma^2_{X|Y} &\doteq \sigma^2_X - \sigma_{XY}\sigma_Y^{-2}\sigma_{YX}. \end{aligned}$$

Here, σ_{XY} refers to the covariance of X and Y. Again, note that $\mu_{X|Y}$ depends affinely on μ_Y. Observe that this admits an equivalent characterization of X as an affine function of Y with added independent Gaussian noise. Formally, we define

$$\begin{aligned} X &\doteq aY + b + \epsilon \quad \text{where} \\ a &\doteq \sigma_{XY}\sigma_Y^{-2} \\ b &\doteq \mu_X - \sigma_{XY}\sigma_Y^{-2}\mu_Y \\ \epsilon &\sim \mathcal{N}\left(0, \sigma^2_{X|Y}\right) \end{aligned} \tag{1.7}$$

It directly follows from the closedness of Gaussians under linear transformations that the characterization of X via (1.7) is equivalent to $X \sim \mathcal{N}\left(\mu_X, \sigma^2_X\right)$, and hence, any Gaussian X can be modeled as a conditional linear Gaussian, i.e., a linear function of another Gaussian Y with additional independent Gaussian noise.

1.6 Bayesian learning and inference

BAYESIAN LEARNING is the process of updating a Bayesian prior $\mathbb{P}(A)$ to a Bayesian posterior $\mathbb{P}(A \mid B)$ upon observing B. Thereby, we typically replace A with the parameterization of a model θ, and B with labeled training data $\{(x_i, y_i)\}_{i=1}^n$.

Bayesian learning allows to quantify the uncertainty regarding the parameters one is trying to learn. It is helpful to differentiate between learning and inference. By LEARNING we refer to the aforementioned process of learning a model from data. In contrast, by INFERENCE we refer to the process of using our learned model to predict labels at new inputs.
At the core of Bayesian learning is Bayes' rule.

Theorem 9 (Bayes' rule)**.** *Given random vectors* $\mathbf{X}$ *in* $\mathbb{R}^n$ *and* $\mathbf{Y}$ *in* $\mathbb{R}^m$*, we have for any* $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$*,*

$$p(\boldsymbol{x} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y})}.$$

It is useful to consider the meaning of each term in Bayes' rule separately.

- $p(\boldsymbol{x} \mid \boldsymbol{y})$ is the POSTERIOR: is the updated belief about $\boldsymbol{x}$ after observing $\boldsymbol{y}$,
- $p(x)$ is the PRIOR: is the initial belief about x,
- $p(y \mid x)$ is the (CONDITIONAL) LIKELIHOOD: describes how likely the observations y are under a given value x,
- $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x})$ is the JOINT LIKELIHOOD OR GENERATIVE MODEL: combines prior and likelihood,
- $p(y)$ is the MARGINAL LIKELIHOOD (or EVIDENCE): describes how likely the observations y are across all values of $\boldsymbol{x}$.

The marginal likelihood can be computed using the law of total probability (1.3),

$$p(\boldsymbol{y}) = \int_{\mathbf{X}(\Omega)} p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x},$$

or the sum rule. Note, however, that as the marginal likelihood does not depend on x,

$$p(\boldsymbol{x} \mid \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x}),$$

so we do not need to compute it to optimize for x (e.g., find the gradient with respect to x). This is a helpful fact, which we will make use of quite often.

1.7 Using Bayes' rule for supervised learning

We want to learn a model from some training data.

We interpret $p(\boldsymbol{\theta})$ as the degree of our belief that the model parameterized by θ describes the data best. The likelihood describes how likely the training data is under a particular model. Often, we assume that our data was sampled independently, allowing us to write the likelihood as a product:

$$p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) = \prod_{i=1}^{n} p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) \quad \text{(LIKELIHOOD)}.$$

The posterior represents our belief about the best model after seeing the training data. Using Bayes' rule, we can write it as

$$p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) = \frac{1}{Z} p(\boldsymbol{\theta}) \prod_{i=1}^{n} p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) \quad \text{where} \quad Z \doteq \int p(\boldsymbol{\theta}) \prod_{i=1}^{n} p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) d\boldsymbol{\theta} \quad \text{(POSTERIOR)}$$

and again assuming that the training data was sampled independently. We often refer to Z as the normalizing constant.
Finally, we can use our learned model for inference (predictions) at a new input $x^\star$ by using the sum rule and the product rule as follows

$$p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}\right) = \int p\left(y^\star, \boldsymbol{\theta} \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}\right) d\boldsymbol{\theta} = \int p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) d\boldsymbol{\theta} \quad \text{(PREDICTIVE POSTERIOR)} \tag{1.8}$$

where we used the sum rule and the product rule.

Observation 1. *In general, the integrals in the normalizing constant Z and the predictive distribution cannot be expressed in closed-form.*
If the prior $p(\boldsymbol{x})$ and posterior $p(\boldsymbol{x} \mid \boldsymbol{y})$ are of the same family of distributions, the prior is called a CONJUGATE PRIOR *to the likelihood $p(\boldsymbol{y} \mid \boldsymbol{x})$. This is a very desirable property, as it allows us to apply the same learning mechanism recursively.*
Under some conditions the Gaussian is self-conjugate: that is, if we have a Gaussian prior and a Gaussian likelihood, then our posterior will also be a Gaussian.

1.8 Point estimations

In statistics, POINT ESTIMATION involves the use of sample data to calculate a single value, known as a point estimate since it identifies a point in some parameter space, which is to serve as a "best guess or "best estimate of an unknown population parameter. In what follows, we present two point estimates of the parameterization of a model θ.

1.8.1 Maximum Likelihood Estimation

Intuitively, the "best model" should have the property that the training data is as likely as possible under this model when compared to all other models. This is precisely the MAXIMUM LIKELIHOOD ESTIMATE or MLE:

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} \doteq \arg\max_{\boldsymbol{\theta}} p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right).$$

Taking the logarithm of the likelihood is a standard technique. The resulting $\log p\left(y_{1:n} \mid x_{1:n}, \boldsymbol{\theta}\right)$ is called log-LIKELIHOOD and its negative is known as the NEGATIVE LOG-LIKELIHOOD $\ell_{\mathrm{nll}}(\boldsymbol{\theta}; \mathcal{D})$. In particular

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}} \ell_{\mathrm{nll}}(\boldsymbol{\theta}; \mathcal{D}).$$

1.8.2 Maximum a Posteriori Estimation

Often, the MLE overfits to the training data. The danger of overfitting can be reduced by taking a "more Bayesian" approach and maximizing over the entire posterior distribution instead of only maximizing the likelihood. This results in the MAXIMUM A POSTERIORI ESTIMATE or MAP estimate:

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} \doteq \arg\max_{\boldsymbol{\theta}} p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) \underbrace{=}_{\text{by Bayes' rule}} \arg\max_{\boldsymbol{\theta}} p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) \cdot p(\boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) = \arg\min_{\boldsymbol{\theta}} \underbrace{-\log p(\boldsymbol{\theta})}_{\text{regularization}} + \underbrace{\ell_{\text{nll}}\left(\boldsymbol{\theta}; \mathcal{D}\right)}_{\text{quality of fit}}.$$

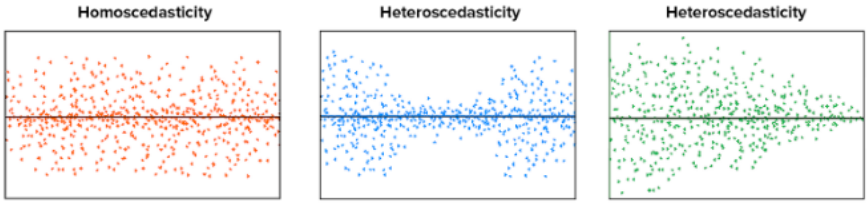
Here, the log-prior $\log p(\theta)$ acts as a regularizer. For example:

- if $p(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{\theta}; \mathbf{0}, \sigma_p^2 \boldsymbol{I}\right)$ then $-\log p(\boldsymbol{\theta}) = \frac{\sigma_p^2}{2}\|\boldsymbol{\theta}\|_2^2$
- if $p(\boldsymbol{\theta}) = \mathrm{Laplace}(\boldsymbol{\theta}; \mathbf{0}, h)$, where $\mathrm{Laplace}(\boldsymbol{\theta}; \mathbf{0}, h) \propto \exp\left(-\frac{|x-\mu|}{h}\right)$ then $-\log p(\boldsymbol{\theta}) = \frac{h}{2}\|\boldsymbol{\theta}\|_1^2$,
- for a uniform prior (i.e., a prior that is independent of θ), MAP estimation reduces to likelihood maximization.

Note that in case the posterior is Gaussian, the MAP estimate, i.e. the mode of the posterior distribution, corresponds to the mean.

1.9 Homoscedastic and Heteroscedastic Noise

HOMOSCEDASTIC noise is the noise that is assumed to be uniform across the domain. HETEROSCEDASTIC noise is the noise that varies depending on the input and which "region" of the domain the input is from.



1.10 Stochastic Gradient Descent

In this course, we primarily employ so-called first-order methods, which rely on (estimates of) the gradient of the objective function to determine a direction of local improvement. The main idea behind these methods is to repeatedly take a step in the opposite direction of the gradient scaled by a learning rate η_t, which may depend on the current iteration t.
We will often want to minimize a stochastic optimization objective

$$L(\boldsymbol{\theta}) \doteq \mathbb{E}_{\boldsymbol{x} \sim p}[\ell(\boldsymbol{\theta}; \boldsymbol{x})]$$

where ℓ and its gradient $\nabla \ell$ are known.
Based on our discussion in previous subsections, it is a natural first step to look for stationary points of L, that is, the roots of ∇L.

Theorem 10 (Robbins-Monro (RM) algorithm)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}^m$ be an unknown function of which we want to find a root and suppose that we have access to unbiased noisy observations $\mathbf{G}(\boldsymbol{\theta})$ of $\boldsymbol{g}(\boldsymbol{\theta})$. The scheme*

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t - \eta_t \boldsymbol{g}^{(t)}\left(\boldsymbol{\theta}_t\right)$$

where $\boldsymbol{g}^{(t)}(\boldsymbol{\theta}_t) \sim \mathbf{G}(\boldsymbol{\theta}_t)$ are independent and unbiased estimates of $\boldsymbol{g}(\boldsymbol{\theta}_t)$, is known as the Robbins-Monro algorithm. If the sequence of learning rates $\{\eta_t\}_{t=0}^{\infty}$ is chosen such that the ROBBINS-MONRO CONDITIONS

$$\eta_t \geq 0 \quad \forall t, \quad \sum_{t=0}^{\infty} \eta_t = \infty \quad \textit{and} \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty$$

and additional regularity assumptions are satisfied, then we have that

$$\boldsymbol{g}(\boldsymbol{\theta}_t) \overset{a.s.}{\to} \mathbf{0} \textit{ as } t \to \infty,$$

that is, the RM-algorithm converges to a root almost surely.

Using Robbins-Monro to find a root of ∇L is known as STOCHASTIC GRADIENT DESCENT. In particular, when ℓ is convex and the RM-conditions are satisfied, Robbins-Monro converges to a stationary point (and hence, a global minimum) of L almost surely.
A commonly used strategy to obtain unbiased gradient estimates is to take the sample mean of the gradient with respect to some set of samples B, also called a BATCH.

Stochastic gradient descent, SGD

initialize θ
while not converged **do**
  draw mini-batch $B \doteq \left\{x^{(1)}, \ldots, x^{(m)}\right\}, x^{(i)} \sim p$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}\right)$
end while

Chapter 2

Bayesian Linear Regression

Given a set of (x, y) pairs, linear regression aims to find a linear model that fits the data optimally. Given the linear model

$$y \approx \boldsymbol{w}^\top \boldsymbol{x} \doteq f(\boldsymbol{x}; \boldsymbol{w})$$

for $x \in \mathbb{R}^d$ and labeled training data $\{(x_i, y_i)\}_{i=1}^n$, we want to find optimal weights w. We define the DESIGN MATRIX

$$\boldsymbol{X} \doteq \begin{bmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{bmatrix} \in \mathbb{R}^{n \times d},$$

as the collection of inputs and the vector $\boldsymbol{y} \doteq [y_1 \cdots y_n]^\top \in \mathbb{R}^n$ as the collection of LABELS. For each noisy observation $(\boldsymbol{x}_i, y_i)$, we define the value of the approximation of our model, $f_i \doteq \boldsymbol{w}^\top \boldsymbol{x}_i$. Our model at the inputs $\boldsymbol{X}$ is described by the vector $f \doteq [f_1 \cdots f_n]^\top$ such that $f = Xw$.
There are many ways of estimating w from data, the most common being the LEAST SQUARES ESTIMATOR,

$$\hat{\boldsymbol{w}}_{\text{ls}} \doteq \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i\right)^2 = \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2,$$

minimizing the squared difference between the labels and predictions of the model. A slightly different estimator is used for RIDGE REGRESSION,

$$\hat{\boldsymbol{w}}_{\text{ridge}} \doteq \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{2.1}$$

where $\lambda > 0$. The squared ℓ_2 regularization term $\lambda \|\boldsymbol{w}\|_2^2$ penalizes large w and thus reduces the complexity of the resulting model. This reduces the potential of overfitting to the training data. We can optimize (2.1) using stochastic gradient descent, or we can find directly an analytical solution as:

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^T \mathbf{y} \tag{2.2}$$

2.1 Maximum Likelihood Estimation

Let us assume that the observations

$$y_i = \boldsymbol{w}^\top \boldsymbol{x}_i + \epsilon_i$$

are made under the influence of i.i.d. homoscedastic Gaussian noise $\epsilon_i \sim \mathcal{N}\left(0, \sigma_n^2\right)$. Recall from section 1.5 that this is equivalently characterized by the Gaussian likelihood,

$$y_i \mid \boldsymbol{x}_i, \boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma_n^2\right). \tag{2.3}$$

Computing the maximum likelihood estimate for the weights,

$$\hat{\boldsymbol{w}}_{\text{MLE}} = \arg\max_{\boldsymbol{w} \in \mathbb{R}^d} \log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}\right) = \arg\max_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^n \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}\right) = \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i\right)^2,$$

we observe that $\hat{w}_{\text{MLE}} = \hat{w}_{\text{ls}}$. Note that in the previous we used (2.3) and (1.4).

2.2 Maximum A Posteriori Estimation

Assume a Gaussian prior (independent of x)

$$\boldsymbol{w} \sim \mathcal{N}\left(\mathbf{0}, \sigma_p^2 \boldsymbol{I}\right)$$

that the labels are independent given the model and a Gaussian likelihood

$$y_i \mid \boldsymbol{x}_i, \boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma_n^2\right), \text{ namely} \quad y_i = w^T x_i + \epsilon_i \text{ where } \epsilon_i \sim \mathcal{N}\left(\mathbf{0}, \sigma_n^2\right).$$

Then, computing the MAP estimate for the weights, we get

$$\hat{\boldsymbol{w}}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{w}} \log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}\right) + \log p(\boldsymbol{w}) = \log p(\boldsymbol{w}) + \sum_{i=1}^{n} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}\right) \tag{2.4}$$

$$= -\arg\max_{\boldsymbol{w}} \left[\sigma_p^{-2} \|\boldsymbol{w}\|_2^2 + \sigma_n^{-2} \sum_{i=1}^{n} \left(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i\right)^2\right] = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \frac{\sigma_n^2}{\sigma_p^2} \|\boldsymbol{w}\|_2^2 \tag{2.5}$$

Observe that this is identical to ridge regression with $\lambda \doteq \sigma_n^2/\sigma_p^2$, and hence, $\hat{w}_{\text{MAP}} = \hat{w}_{\text{ridge.}}$. Also, this is simply the MLE loss with an additional ℓ_2 regularization term (originating from the prior) that encourages keeping weights small.
Also, recall that the MAP estimate corresponds to the mode of the posterior distribution, which in the case of a Gaussian is simply its mean μ. Ass to be expected, μ coincides with the analytical solution to ridge regression from eq. (2.2).

To make predictions at a test point $x^\star$, we define the (model-)predicted point $f^\star \doteq \hat{w}_{\text{MAP}}^\top x^\star$ and obtain the label prediction

$$y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n} \sim \mathcal{N}\left(f^\star, \sigma_n^2\right).$$

Note that using point estimates like the MAP estimate does not quantify uncertainty in the weights. The MAP estimate simply collapses all mass of the posterior around its mode. This is especially harmful when we are highly unsure about the best model (e.g., because we have observed insufficient data).

2.3 Bayesian Inference

Rather than selecting a single weight vector $\hat{\boldsymbol{w}}$ to make predictions, we can use the full posterior distribution. Doing so is also called BAYESIAN INFERENCE, and in the setting of linear regression known as BAYESIAN LINEAR REGRESSION.
We assume again a Gaussian prior (independent of x)

$$p(\mathbf{w}) = \mathcal{N}(0, \mathbf{I})$$

that the labels are independent given the model and a Gaussian likelihood

$$p\left(y \mid \mathbf{x}, \mathbf{w}, \sigma_n\right) = \mathcal{N}\left(y; \mathbf{w}^T \mathbf{x}, \sigma_n^2\right)$$

Then the posterior has closed form:

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{w}; \bar{\mu}, \bar{\Sigma}) \quad \text{where} \tag{2.6}$$

$$\bar{\mu} = \left(\mathbf{X}^T \mathbf{X} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{X}^T \mathbf{y}, \quad \bar{\Sigma} = \left(\sigma_n^{-2} \mathbf{X}^T \mathbf{X} + \mathbf{I}\right)^{-1}$$

How can we use this to make predictions?
For a test point $\boldsymbol{x}^\star$, we let $f^\star \doteq \boldsymbol{w}^\top \boldsymbol{x}^\star$. Using the closedness of Gaussians under linear transformations (1.6) and (2.6), we get

$$p(f^\star \mid x^\star, x_{1:n}, y_{1:n}) = \mathcal{N}\left(\bar{\boldsymbol{\mu}}^\top \boldsymbol{x}^\star, \boldsymbol{x}^{\star\top} \bar{\boldsymbol{\Sigma}} \boldsymbol{x}^\star\right).$$

The previous does not take into account the noise in the labels σ_n^2. So, by taking into account of it as well, for the label prediction $y^\star$, we obtain

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) = \mathcal{N}\left(\bar{\boldsymbol{\mu}}^\top \boldsymbol{x}^\star, \boldsymbol{x}^\star \bar{\boldsymbol{\Sigma}} \boldsymbol{x}^\star + \sigma_n^2\right). \tag{2.7}$$

Recall that the previous closed form is equivalent to (1.8), namely

$$p\left(y^{\star} \mid \boldsymbol{x}^{\star}, \boldsymbol{x}_{1:n}, y_{1:n}\right)=\int p\left(y^{\star} \mid \boldsymbol{x}^{\star}, \boldsymbol{w}\right) p\left(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) d \boldsymbol{w}$$

So we can conclude that, while Ridge Regression predicts using MAP estimate for weights, namely

$$\hat{\mathbf{w}}=\arg \max_{\mathbf{w}} p\left(\mathbf{w} \mid \mathbf{x}_{1:n}, y_{1:n}\right) \Longrightarrow p\left(y^{*} \mid \mathbf{x}^{*}, \hat{\mathbf{w}}\right)=\mathcal{N}\left(y^{*} ; \hat{\mathbf{w}}^{T} \mathbf{x}^{*}, \sigma_{n}^{2}\right),$$

Bayesian Linear Regression predicts by averaging all $\mathbf{w}$ accordingly to the posterior, namely

$$p\left(y^{*} \mid x^{*}, x_{1:n}, y_{1:n}\right)=\int p\left(y^{*} \mid \boldsymbol{x}^{*}, \boldsymbol{w}\right) p\left(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) d \boldsymbol{w}=N\left(y^{*} ; \widehat{\boldsymbol{w}}^{T} \boldsymbol{x}^{*}, \sigma_{n}^{2}+x^{* T} \bar{\Sigma} x^{*}\right).$$

In particular that Ridge Regression can be viewed as approximating the full posterior by placing all mass on its mode

$$p\left(y^{*} \mid \mathbf{x}^{*}, \mathbf{x}_{1:n}, y_{1:n}\right)=\int p\left(y^{*} \mid \mathbf{x}^{*}, \mathbf{w}\right) p\left(\mathbf{w} \mid \mathbf{x}_{1:n}, y_{1:n}\right) d \mathbf{w} \approx \int p\left(y^{*} \mid \mathbf{x}^{*}, \mathbf{w}\right) \delta_{\hat{\mathbf{w}}}(\mathbf{w}) d \mathbf{w}=p\left(y^{*} \mid \mathbf{x}^{*}, \hat{\mathbf{w}}\right)$$

$$\text{where} \quad \hat{\mathbf{w}}=\arg \max_{\mathbf{w}} p\left(\mathbf{w} \mid \mathbf{x}_{1:n}, y_{1:n}\right)$$

Observation 2. *In BLR, need to specify the (co-)variance of the prior $\boldsymbol{\sigma}_{\mathrm{p}}$ and the variance of the noise $\boldsymbol{\sigma}_{\mathrm{n}}$. These are hyperparameters of the model (governing the distribution of the parameters $\boldsymbol{w}$). How should we choose them? One option:*

- *choose $\hat{\lambda}=\frac{\hat{\sigma}_{n}^{2}}{\hat{\sigma}_{p}^{2}}$ via cross-validation*
- *then estimate $\hat{\sigma}_{n}^{2}=\frac{1}{n} \sum_{i=1}^{n}\left(y_{i}-\hat{w}^{T} \mathbf{x}_{i}\right)^{2}$ as the empirical variance of the residual, and solve for $\hat{\sigma}_{p}^{2}=\frac{\hat{\sigma}_{n}^{2}}{\hat{\lambda}}$*

2.4 Aleatoric and Epistemic Uncertainty

The predictive posterior distribution from eq. (2.7) highlights a decomposition of uncertainty wherein $x^{\star \top} \Sigma x^{\star}$ corresponds to the uncertainty about our model due to the lack of data, commonly referred to as the EPISTEMIC UNCERTAINTY, and σ_{n}^{2} corresponds to the uncertainty about the labels that cannot be explained by the inputs and any model from the model class, commonly referred to as the ALEATORIC UNCERTAINTY, irreducible noise, or simply label noise.

$$p(y^{*} \mid \mathbf{X}, \mathbf{y}, \mathbf{x}^{*})=\mathcal{N}(\bar{\mu}^{T} \mathbf{x}^{*}, \mathbf{x}^{* T} \bar{\Sigma} \mathbf{x}^{*}+\sigma_{n}^{2})$$

Uncertainty about f* (epistemic)

Noise / uncertainty about y* given f* (aleatoric)

this noise is present even with infinite data

A natural Bayesian approach is to represent epistemic uncertainty with a probability distribution over models. Intuitively, the variance of this distribution measures our uncertainty about the model and its mode corresponds to our current best (point) estimate.

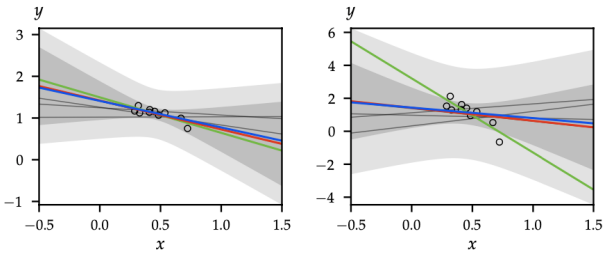


Figure 2.1: Comparison of linear regression (MLE), ridge regression (MAP estimate), and Bayesian linear regression when the data is generated according to $y \mid \boldsymbol{w}, \boldsymbol{x} \sim \mathcal{N}\left(\boldsymbol{w}^\top \boldsymbol{x}, \sigma_n^2\right)$. The true mean is shown in blue, the MLE in green, and the MAP estimate in red. The dark gray area denotes the epistemic uncertainty of Bayesian linear regression and the light gray area the additional homoscedastic noise. Plausible realizations are shown in black. On the left, $\sigma_n = 0.15$. On the right, $\sigma_n = 0.7$.

Epistemic and aleatoric uncertainty can be formally defined in terms of the law of total variance. We have

$$\operatorname{Var}\left[y^\star \mid x^\star\right] = \mathbb{E}_\theta\left[\operatorname{Var}_{y^\star}\left[y^\star \mid x^\star, \theta\right]\right] + \operatorname{Var}_\theta\left[\mathbb{E}_{y^\star}\left[y^\star \mid x^\star, \theta\right]\right]. \tag{2.8}$$

Here, the mean variability of predictions $y^\star$ averaged across all models $\boldsymbol{\theta}$ is the aleatoric uncertainty. In contrast, the variability of the mean prediction $y^\star$ under each model θ is the epistemic uncertainty.

2.5 Recursive Bayesian Updates

A more general feature of supervised Bayesian learning, which often leads to efficient algorithms, is its recursive structure. Suppose we are given a prior $p(\boldsymbol{\theta})$ and observations $y_{1:n}$. We define

$$p^{(t)}(\boldsymbol{\theta}) \doteq p\left(\boldsymbol{\theta} \mid y_{1:t}\right)$$

to be the posterior after the first t observations. We therefore have $p^{(0)}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$. Now, suppose that we have already computed $p^{(t)}(\boldsymbol{\theta})$ and observe y_{t+1}. We can then recursively update the posterior as follows,

$$p^{(t+1)}(\boldsymbol{\theta}) = p\left(\boldsymbol{\theta} \mid y_{1:t+1}\right) \propto p\left(\boldsymbol{\theta} \mid y_{1:t}\right) \cdot p\left(y_{t+1} \mid \boldsymbol{\theta}, y_{1:t}\right) = p^{(t)}(\boldsymbol{\theta}) \cdot p\left(y_{t+1} \mid \boldsymbol{\theta}\right)$$

Intuitively, the posterior distribution absorbs, or summarizes all seen data. Thus, as data arrives online, i.e., in real-time, we can obtain the new posterior and use it to replace our prior: ”today’s posterior is tomorrow’s prior”.

2.6 Summary BLR

- Bayesian Linear Regression makes same modeling assumptions as Ridge Regression, i.e. Gaussian prior on weights, Gaussian noise
- Bayesian Linear Regression uses full posterior distribution over the weights rather than the mode only as Ridge Regression
- Thus, it captures uncertainty in weights, and allows to separate epistemic from aleatoric uncertainty
- Due to independence of the noise, can do recursive updates on the weights

2.7 Non-linear Regression

We can use linear regression not only to learn linear functions. The trick is to apply a non-linear transformation $\phi : \mathbb{R}^d \to \mathbb{R}^e$ to the features x_i, where d is the dimension of the input space and e is the dimension of the designed FEATURE SPACE. We denote the design matrix comprised of transformed features by $\Phi \in \mathbb{R}^{n \times e}$. However, computational cost increases with dimensionality of the feature space!

2.7.1 Function-space View: the kernel trick

Let us look at Bayesian linear regression through a different lens. Previously, we have been interpreting it as a distribution over the weights $\boldsymbol{w}$ of a (linear) function $f = \boldsymbol{\Phi w}$. The key idea is that for a finite set of inputs (ensuring that the design matrix is well-defined), we can equivalently consider a distribution directly over the estimated function values f.
Nothing has changed but, instead of considering a prior over the weights $w \sim \mathcal{N}\left(\mathbf{0}, \sigma_p^2 \boldsymbol{I}\right)$ as we have done previously, we now impose a prior directly on the values of our model at the observations. Using that Gaussians are closed under linear maps (1.6), we obtain the equivalent prior

$$f \mid \boldsymbol{X} \sim= \mathcal{N}(\mathbf{0}, \underbrace{\sigma_p^2 \boldsymbol{\Phi}\boldsymbol{\Phi}^\top}_{\boldsymbol{K}})$$

where $K \in \mathbb{R}^{n \times n}$ is the so-called KERNEL MATRIX. Observe that the entries of the kernel matrix can be expressed as $\boldsymbol{K}(i,j) = \sigma_p^2 \cdot \boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^\top \boldsymbol{\phi}\left(\boldsymbol{x}_j\right)$. The kernel matrix $\boldsymbol{K}$ has entries only for the finite set of observed inputs. However, in principle, we could have observed any input, and this motivates the definition of the KERNEL FUNCTION

$$k\left(\boldsymbol{x}, \boldsymbol{x}'\right) \doteq \sigma_p^2 \cdot \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}\left(\boldsymbol{x}'\right)$$

for arbitrary inputs x and x'. A kernel matrix is simply a finite view of the kernel function,

$$\boldsymbol{K} = \begin{bmatrix} k\left(\boldsymbol{x}_1, \boldsymbol{x}_1\right) & \cdots & k\left(\boldsymbol{x}_1, \boldsymbol{x}_n\right) \\ \vdots & \ddots & \vdots \\ k\left(\boldsymbol{x}_n, \boldsymbol{x}_1\right) & \cdots & k\left(\boldsymbol{x}_n, \boldsymbol{x}_n\right) \end{bmatrix}$$

Thus, we have reformulated the learning algorithm such that the feature space is now implicit in the choice of kernel, and the kernel is defined by inner products of (nonlinearly transformed) inputs. In other words, the choice of kernel implicitly determines the class of functions that f is sampled from, and encodes our prior beliefs. This is known as the KERNEL TRICK, and can be summarized as follows:

- express problem s.t. it only depends on inner products,
- replace inner products by kernels.

Chapter 3

Gaussian Processes

In the last section of the previous chapter, the domain was given by the set of weights (weight-space view) or the set of (noisy) observations (function- space view). In either case, it was a finite domain.
We obtain GAUSSIAN PROCESSES by extending the function-space view of Bayesian linear regression which we introduced in section 2.7 to infinite domains.

We are still concerned with the problem of estimating the value of a function $f : \mathcal{X} \to \mathbb{R}$ at arbitrary points $x^\star \in \mathcal{X}$ given training data $\{x_i, y_i\}_{i=1}^n$, where the labels are assumed to be corrupted by homoscedastic Gaussian noise,

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}\left(0, \sigma_n^2\right).$$

As in the chapter on Bayesian linear regression, we denote by $\boldsymbol{X}$ the design matrix (collection of training inputs) and by $\boldsymbol{y}$ the vector of training labels.

Definition 2 (Gaussian process, GP)**.** *A* GAUSSIAN PROCESS *is an infinite set of random variables such that any finite number of them are jointly Gaussian.*
We use a set $\mathcal{X}$ to index the collection of random variables. A Gaussian process is characterized by a mean function $\mu : \mathcal{X} \to \mathbb{R}$ and a covariance function (or kernel function) $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for any $A \doteq \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$, we have

$$f_A \doteq [f_{\boldsymbol{x}_1} \cdots f_{\boldsymbol{x}_m}]^\top \sim \mathcal{N}(\boldsymbol{\mu}_A, \boldsymbol{K}_{AA}) \tag{3.1}$$

where

$$\boldsymbol{\mu}_A \doteq \begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \vdots \\ \mu(\boldsymbol{x}_m) \end{bmatrix}, \quad \boldsymbol{K}_{AA} \doteq \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_m, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{bmatrix}.$$

We write $f \sim \mathcal{GP}(\mu, k)$ and the random variable f_x represents the value of the function $f(x)$ at location $x \in \mathcal{X}$. Thus write $f(x) \doteq f_x$.

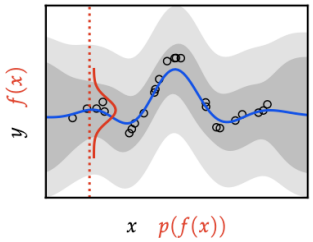


Figure 3.1: A Gaussian process can be interpreted as an infinite-dimensional Gaussian over functions. At any location x in the domain, this yields a distribution over values $f(x)$ shown in red. The blue line corresponds to the MAP estimate (i.e., mean function of the Gaussian process), the dark gray region corresponds to the epistemic uncertainty and the light gray region denotes the additional aleatoric uncertainty.

3.1 Learning and Inference

First, let us look at learning and inference in the context of Gaussian processes. Given a prior $f \sim \mathcal{GP}(\mu, k)$ and (noisy) observations $y_i = f(\boldsymbol{x}_i) + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}\left(0, \sigma_n^2\right)$ at locations $A \doteq \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, we can write the joint distribution of the observations $y_{1:n}$ and the noise-free prediction $f^\star$ at a test point $x^\star$ as

$$\begin{bmatrix} \boldsymbol{y} \\ f^\star \end{bmatrix} \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\mathbf{K}}), \quad \text{where}$$

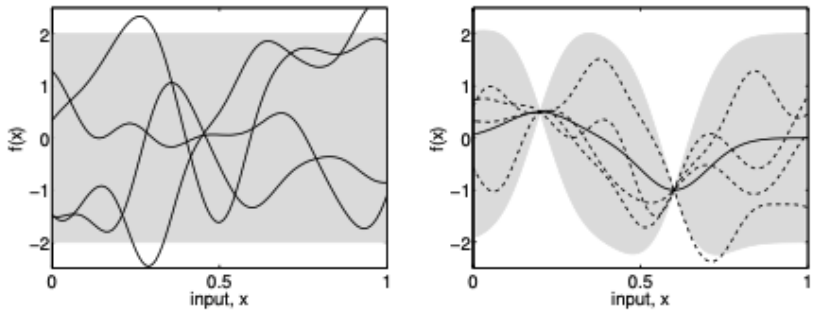
$$\tilde{\boldsymbol{\mu}} \doteq \begin{bmatrix} \boldsymbol{\mu}_A \\ \mu(\boldsymbol{x}^\star) \end{bmatrix}, \quad \tilde{\mathbf{K}} \doteq \begin{bmatrix} \boldsymbol{K}_{AA} + \sigma_n^2 \boldsymbol{I} & \boldsymbol{k}_{x^\star, A} \\ \boldsymbol{k}_{\boldsymbol{x}^\star, A}^\top & k(\boldsymbol{x}^\star, \boldsymbol{x}^\star) \end{bmatrix}, \quad \boldsymbol{k}_{\boldsymbol{x}, A} \doteq \begin{bmatrix} k(\boldsymbol{x}, \boldsymbol{x}_1) \\ \vdots \\ k(\boldsymbol{x}, \boldsymbol{x}_n) \end{bmatrix}.$$

Deriving the conditional distribution using (1.5), we obtain that the Gaussian process posterior is given by

$$\begin{aligned} f \mid \boldsymbol{x}_{1:n}, y_{1:n} &\sim \mathcal{GP}(\boldsymbol{\mu}', k'), \text{ where} \\ \mu'(\boldsymbol{x}) &\doteq \mu(\boldsymbol{x}) + \boldsymbol{k}_{\boldsymbol{x}, A}^\top \left(\boldsymbol{K}_{AA} + \sigma_n^2 \boldsymbol{I}\right)^{-1} (\boldsymbol{y}_A - \boldsymbol{\mu}_A), \\ k'(\boldsymbol{x}, \boldsymbol{x}') &\doteq k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}_{\boldsymbol{x}, A}^\top \left(\boldsymbol{K}_{AA} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_{\boldsymbol{x}', A}. \end{aligned} \tag{3.2}$$

Observe that analogously to Bayesian linear regression, the posterior covariance can only decrease when conditioning on additional data, and is independent of the observations y_A.
Using eq. (3.2) and our assumption of homoscedastic noise, the predictive posterior at the test point $x^\star$ is

$$y^\star \mid x^\star, x_{1:n}, y_{1:n} \sim \mathcal{N}\left(\mu^\star, k^\star + \sigma_n^2\right).$$



(a), prior (b), posterior

Figure 3.2: Panel (a) shows four samples drawn from the prior distribution. Panel (b) shows the situation after two datapoints have been observed. The mean prediction is shown as the solid line and four samples from the posterior are shown as dashed lines. In both plots the shaded region denotes twice the standard deviation at each input value x.

3.2 Sampling

Often, we are not interested in the full predictive posterior distribution, but merely want to obtain samples of our Gaussian process model. Recall the product rule (1.14),

$$p(f_1, \ldots, f_n) = \prod_{i=1}^{n} p(f_i \mid f_{1:i-1}).$$

That is the joint distribution factorizes neatly into a product where each factor only depends on the "outcomes" of preceding factors. We can therefore obtain samples one-by-one, each time conditioning on one more observation:

$$\begin{aligned} f_1 &\sim p\left(f_1\right) \\ f_2 &\sim p\left(f_2 \mid f_1\right) \\ f_3 &\sim p\left(f_3 \mid f_1, f_2\right) \\ &\vdots \end{aligned}$$

This general approach is known as FORWARD SAMPLING.

3.3 Kernel Functions

We have previously introduced kernels in section 2.7 in the context of a function-space view of Bayesian linear regression. There we have already seen that kernel functions define the shape of the functions that are realized from a Gaussian distribution over functions, namely a Gaussian process. In this section, we generalize the notion of kernels to encompass function classes beyond linear and polynomial functions.

Definition 3 (Kernel function)**.** *A* KERNEL FUNCTION k *is defined as*

$$k\left(x, x'\right) \doteq \operatorname{Cov}\left[f(x), f\left(x'\right)\right]$$

for locations x *and* x' *such that:*

- k *is symmetric, namely* $k\left(x, x'\right) = k\left(\boldsymbol{x}', \boldsymbol{x}\right)$ *for any* $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$
- $\boldsymbol{K}_{AA}$ *is positive semi-definite for any* $A \subseteq \mathcal{X}$*, namely all eigenvalues of the matrix* $\boldsymbol{K}_{AA}$ *are non-negative for any* $A \subseteq \mathcal{X}$*.*

We say that a kernel function is POSITIVE DEFINITE *if* $\boldsymbol{K}_{AA}$ *is positive definite for any* $A \subseteq \mathcal{X}$*.*

The two defining conditions ensure that for any $A \subseteq \mathcal{X}, K_{AA}$ is a valid covariance matrix. Indeed, it can be shown that a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying the above two conditions corresponds to a (not necessarily finite dimensional) feature representation in some feature space, that is, there exists a feature transformation $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^e$ where $e \in \mathbb{R} \cup \{\infty\}$ such that $k\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}\left(\boldsymbol{x}'\right)$. This is known as MERCER'S THEOREM. Intuitively, the kernel function evaluated at locations x and x' describes how $f(x)$ and $f\left(x'\right)$ are related. If x and x' are close, then $f(x)$ and $f\left(x'\right)$ are usually taken to be positively correlated, encoding a smooth function.

3.3.1 Common Kernels

Linear Kernel and Linear Kernel with features. The LINEAR KERNEL is defined as

$$k\left(x, x' ; \phi\right) \doteq \phi(x)^\top \phi\left(x'\right)$$

where ϕ is a nonlinear transformation as introduced in section 2.7 or the identity. A Gaussian process with a linear kernel with $\phi = \mathrm{id}.$ is equivalent to Bayesian linear regression.

Gaussian Kernel/RBF. The GAUSSIAN KERNEL, also known as RADIAL BASIS FUNCTION (RBF) KERNEL, is an analytic (infinitely differentiable) kernel defined as

$$k\left(x, x' ; \ell\right) \doteq \exp\left(-\frac{\left\|x - x'\right\|_2^2}{2\ell^2}\right)$$

where ℓ is its length scale. The larger the length scale ℓ, the smoother the resulting functions. Indeed as the length scale is increased, the exponent of the exponential increases, resulting in a higher dependency between locations.

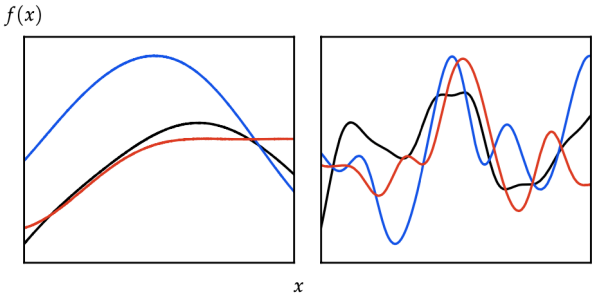


Figure 3.3: Functions sampled according to a Gaussian process with a Gaussian kernel and length scales $\ell = 5$ (left) and $\ell = 1$ (right).

Exponential Kernel/Laplace kernel. The LAPLACE KERNEL, also known as EXPONENTIAL KERNEL, is a continuous, but nowhere differentiable, kernel defined as

$$k\left(x, x'; \ell\right) \doteq \exp\left(-\frac{\left\|x - x'\right\|_1}{\ell}\right).$$

As can be seen in the below figure, samples from a GP with Laplace kernel are non-smooth as opposed to the samples from a GP with Gaussian kernel. Again, the larger the length scale ℓ, the smoother the resulting functions.

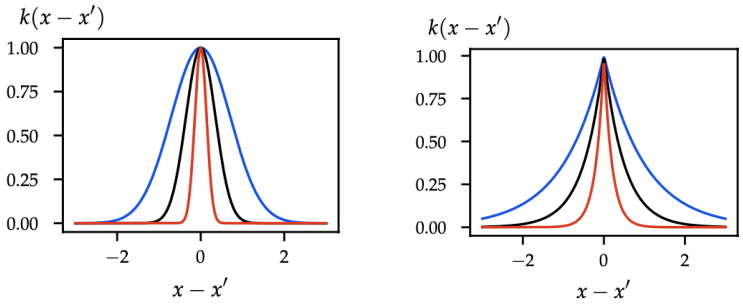


Figure 3.4: Left panel: Gaussian kernel with length scales $\ell = 1$ (blue), $\ell = 0.5$ (black), and $\ell = 0.2$ (red). Right panel: Laplace kernel with length scales $\ell = 1$ (blue), $\ell = 0.5$ (black), and $\ell = 0.2$ (red).

Matern Kernel. The Matern kernel trades the smoothness of the Gaussian and the Laplace kernels. It is defined as

$$k\left(x, x'; \nu, \ell\right) \doteq \frac{2^{1-v}}{\Gamma(v)}\left(\frac{\sqrt{2\nu}\left\|x - x'\right\|_2}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}\left\|x - x'\right\|_2}{\ell}\right)$$

where Γ is the Gamma function, K_ν the modified Bessel function of the second kind, and ℓ a length scale parameter. The resulting functions are $\lceil \nu \rceil - 1$ times mean square differentiable. For $\nu = 1/2$, the Matérn kernel is equivalent to the Laplace kernel and, for $\nu \to \infty$, the Matern kernel is equivalent to the Gaussian kernel.

3.3.2 Kernel Composition

Given kernels $k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, they can be composed to obtain a new kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ in the following ways:

- $k\left(x, x'\right) \doteq k_1\left(x, x'\right) + k_2\left(x, x'\right)$,
- $k\left(x, x'\right) \doteq k_1\left(x, x'\right) \cdot k_2\left(x, x'\right)$,

- $k(x, x') \doteq c \cdot k_1(x, x')$ for any $c > 0$,
- $k(x, x') \doteq f(k_1(x, x'))$ for any polynomial f with positive coefficients or f = exp.

For example, the additive structure of a function $f(\boldsymbol{x}) \doteq f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ can be easily encoded in GP models. Suppose that $f_1 \sim \mathcal{GP}(\mu_1, k_1)$ and $f_2 \sim \mathcal{GP}(\mu_2, k_2)$, then the distribution of the sum of those two functions $f = f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2)$ is another GP.

3.3.3 Stationarity and Isotropy

Definition 4 (Stationarity and isotropy)**.** *A kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called*

- STATIONARY*, or* SHIFT-INVARIANT*, if there exists a function $\tilde{k}$ such that $\tilde{k}(\boldsymbol{x} - \boldsymbol{x}') = k(x, \boldsymbol{x}')$, and*
- ISOTROPIC *if there exists a function $\tilde{k}$ such that $\tilde{k}(\|x - x'\|_2) = k(x, x')$.*

Note that stationarity is a necessary condition for isotropy. In other words, isotropy implies stationarity.

	stationary	isotropic
linear kernel	no	no
Gaussian kernel	yes	yes
$k(\boldsymbol{x}, \boldsymbol{x}') \doteq \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|_{\boldsymbol{M}}^2)$ where $\boldsymbol{M}$ is positive semi-definite	yes	no

Figure 3.5: Stationarity and isotropy of kernels. Here $\|x - x'\|_M^2 = (x - x')^T M (x - x')$.

3.4 Model selection

We have not yet discussed how to pick the hyperparameters θ (e.g., parameters of kernels). A common technique in supervised learning is to select hyperparameters θ, such that the resulting function estimate $\hat{f}_\theta$ leads to the most accurate predictions on hold-out validation data. After reviewing this approach, we contrast it with a Bayesian approach to model selection, which avoids using point estimates of $\hat{f}_{\boldsymbol{\theta}}$ and rather utilizes the full posterior.

3.4.1 Optimizing Validation Set Performance

A common approach to model selection is to split our data $\mathcal{D}$ into separate training set $\mathcal{D}^{\text{train}} \doteq \left\{\left(x_i^{\text{train}}, y_i^{\text{train}}\right)\right\}_{i=1}^n$ and validation sets $\mathcal{D}^{\text{val}} \doteq \left\{\left(\boldsymbol{x}_i^{\text{val}}, \boldsymbol{y}_i^{\text{val}}\right)\right\}_{i=1}^m$. We then optimize the model for a parameter candidate θ_j using the training set. This is usually done by picking a point estimate (like the MAP estimate),

$$\hat{f}_j \doteq \arg\max_f p\left(f \mid \boldsymbol{x}_{1:n}^{\text{train}}, y_{1:n}^{\text{train}}\right).$$

Then, we score $\boldsymbol{\theta}_j$ according to the performance of $\hat{f}_j$ on the validation set,

$$\hat{\boldsymbol{\theta}} \doteq \arg\max_{\boldsymbol{\theta}_j} p\left(y_{1:m}^{\text{val}} \mid \boldsymbol{x}_{1:m}^{\text{val}}, \hat{f}_j\right)$$

This ensures that $\hat{f}_j$ does not depend on $\mathcal{D}^{\text{val}}$. While this approach often is quite effective at preventing overfitting as compared to using the same data for training and picking $\hat{\boldsymbol{\theta}}$, it still collapses the uncertainty in f into a point estimate. Can we do better?

3.4.2 Maximizing the Marginal Likelihood

Instead of optimizing the effects of $\boldsymbol{\theta}$ for a specific point estimate $\hat{f}$ of the model f, maximizing the marginal likelihood optimizes the effects of $\boldsymbol{\theta}$ across all realizations of f. In this approach, we obtain our hyperparameter estimate via

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} \doteq \arg\max_{\boldsymbol{\theta}} p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) = \arg\max_{\boldsymbol{\theta}} \int p\left(y_{1:n}, f \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) df = \arg\max_{\boldsymbol{\theta}} \int p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, f, \boldsymbol{\theta}\right) p(f \mid \boldsymbol{\theta}) df. \tag{3.3}$$

where, in the second equality, we conditioned on f using the sum rule and, in the third one, we used the product rule.
In the last equation, $p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, f, \boldsymbol{\theta}\right)$ is the likelihood and $p(f \mid \boldsymbol{\theta})$ is the prior. Hence, since both factors appears in an integral, maximizing the marginal likelihood naturally encourages trading between a large likelihood and a large prior.
For an "underfit" model, the likelihood is mostly small as the data cannot be well described, while the prior is large as there are "fewer" functions to choose from. For an "overfit" model, the likelihood is large for "some" functions (which would be picked if we were only minimizing the training error and not doing cross validation) but small for "most" functions. The prior is small, as the probability mass has to be distributed among "more" functions. Thus, in both cases, one term in the product will be small.

	likelihood	prior
"underfit" model (too simple $\boldsymbol{\theta}$)	small for "almost all" f	large
"overfit" model (too complex $\boldsymbol{\theta}$)	large for "few" f small for "most" f	small
"just right"	moderate for "many" f	moderate

Figure 3.6: This table gives an intuitive explanation of the effects of parameter choices θ on the marginal likelihood.

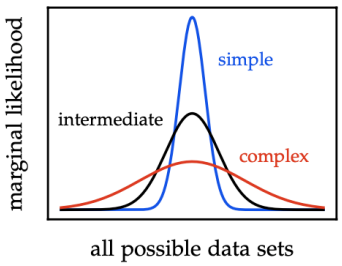


Figure 3.7: A schematic illustration of the marginal likelihood of a simple, in- termediate, and complex model across all possible data sets.

In the context of Gaussian process regression, recall that

$$y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{K}_{f,\boldsymbol{\theta}} + \sigma_n^2 \boldsymbol{I}\right)$$

where $\boldsymbol{K}_{f,\boldsymbol{\theta}}$ denotes the kernel matrix at the inputs $\boldsymbol{x}_{1:n}$ depending on the kernel function parameterized by $\boldsymbol{\theta}$. Also note that, for notational simplicity, the mean function is taken to be zero.
We write $\boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}} \doteq \boldsymbol{K}_{f,\boldsymbol{\theta}} + \sigma_n^2 \boldsymbol{I}$ and, continuing from eq. (3.3), we obtain

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \mathcal{N}\left(\boldsymbol{y}; \mathbf{0}, \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}\right) = \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \boldsymbol{y}^\top \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}^{-1} \boldsymbol{y} + \frac{1}{2} \log\det\left(\boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}\right) + \frac{n}{2} \log 2\pi$$

where in the second equality we took the negative logarithm. Let's consider the obtimization problem

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{2}\boldsymbol{y}^\top \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}^{-1}\boldsymbol{y} + \frac{1}{2}\log\det\left(\boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}\right).$$

The first term of the optimization objective describes the "goodness of fit" (i.e., the "alignment" of y with $K_{y,\boldsymbol{\theta}}$), while the second term characterizes the "volume" of the model class. Thus, this optimization naturally trades the aforementioned objectives.
Marginal likelihood maximization is an EMPIRICAL BAYES METHOD. Often it is simply referred to as empirical Bayes. It also has the nice property that the gradient of its objective can be expressed in closed-from

$$\frac{\partial}{\partial\theta_j}\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) = \frac{1}{2}\operatorname{tr}\left(\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - \boldsymbol{K}_{y,\boldsymbol{\theta}}^{-1}\right)\frac{\partial \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}}{\partial\theta_j}\right)$$

where $\boldsymbol{\alpha} \doteq \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}^{-1}\boldsymbol{y}$ and $\operatorname{tr}(\boldsymbol{M})$ is the trace of a matrix M. This optimization problem is, in general, non-convex, so gradient descent could end up in local optima. The next figure gives an example of two local optima of the marginal likelihood according to empirical Bayes.

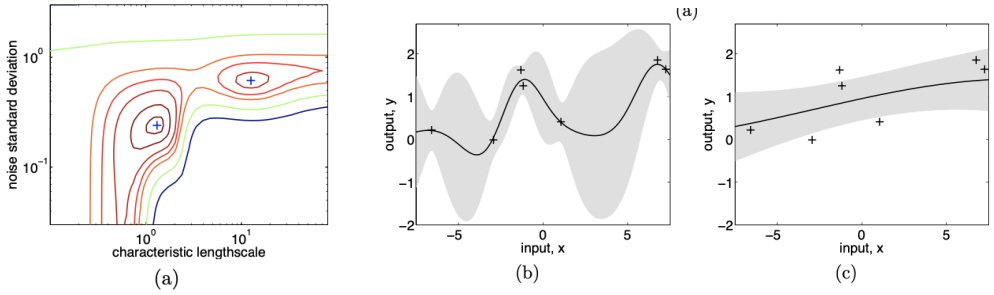


Figure 3.8: Panel (a) shows the marginal likelihood as a function of the hyperparameters ℓ (length-scale) and σ_n^2 (noise standard deviation), where $\sigma_f^2 = 1$ (signal standard deviation) for a data set of 7 observations (seen in panels (b) and (c)). There are two local optima, indicated with ' + ': the global optimum has low noise and a short length-scale; the local optimum has a high noise and a long length scale. In (b) and (c) the inferred underlying functions (and 95% confidence intervals) are shown for each of the two solutions. (b) corresponds to the left local optima, and indeed it corresponds to smaller noise (less observations are explained as noise). (c) corresponds to the right local optima, and indeed it corresponds to larger noise (more observations are explained as noise).

3.5 Approximations

To learn a Gaussian process, we need to invert matrices, hence the computational cost is $\mathcal{O}\left(n^3\right)$. Compare this to Bayesian linear regression which allows us to learn a regression model in $\mathcal{O}\left(nd^2\right)$ time (even online) where d is the feature dimension. Then one can either exploit parallelism using GPU computations (this yields substantial speedup, but doesn't address the cubic scaling in n) or look for ways of approximating a Gaussian process.

3.5.1 Local Methods

Recall that during forward sampling, we had to condition on a larger and larger number of previous samples. When sampling at a location x, a very simple approximation is to only condition on those samples $\boldsymbol{x}'$ that are "close" (where $|k\left(x, x'\right)| \geq \tau$ for some $\tau > 0$). Essentially, this method "cuts off the tails" of the kernel function k. However, τ has to be chosen carefully as if τ is chosen too large, samples become essentially independent. This is one example of a SPARSE APPROXIMATION of a Gaussian process but it us still expensive if "many" points close by.

3.5.2 Kernel function approximations

Another method is to approximate the kernel function directly. The idea is to construct a low-dimensional feature map $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^m$ that approximates the kernel,

$$k\left(x, x'\right) \approx \phi(x)^\top \boldsymbol{\phi}\left(\boldsymbol{x}'\right).$$

Then, we can apply Bayesian linear regression, resulting in a time complexity of $\mathcal{O}\left(nm^2 + m^3\right)$.
One example of this approach are RANDOM FOURIER features, which we will discuss in the following. Because a stationary kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ can be interpreted as a function in one variable, it has an associated Fourier transform which we denote by $p(\boldsymbol{\omega})$. That is,

$$k\left(\boldsymbol{x} - \boldsymbol{x}'\right) = \int_{\mathbb{R}^d} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top \left(\boldsymbol{x} - \boldsymbol{x}'\right)} d\boldsymbol{\omega}. \tag{3.4}$$

Theorem 11 (Bochner's theorem)**.** *A continuous stationary kernel on $\mathbb{R}^d$ is positive definite if and only if its Fourier transform $p(\boldsymbol{\omega})$ is non-negative.*

Bochner's theorem implies that when a continuous and stationary kernel is positive definite and scaled appropriately, its Fourier transform $p(\boldsymbol{\omega})$ is a proper probability distribution. In this case, $p(\boldsymbol{\omega})$ is called the SPECTRAL DENSITY of the kernel k. Viceversa, if $p(\boldsymbol{\omega}) \geq 0$, from (3.4) we automatically get a positive definite stationary kernel.

Example 1. *The Gaussian kernel*

$$k\left(x, x'\right) = \exp\left(-\frac{\left\|x - x'\right\|_2^2}{2}\right)$$

has the Fourier transform:

$$p(\omega) = (2\pi)^{-d/2} \exp\left(-\|\omega\|_2^2 / 2\right)$$

This is simply the standard Gaussian distribution in d dimensions.
In general, for stationary kernels $k\left(\mathrm{x} - \mathrm{x}'\right) = k(\Delta)$, the Fourier transform $p(\boldsymbol{\omega}) = \frac{1}{2\pi} \int_{\mathbb{R}^d} k(\Delta) e^{-j\omega^T \Delta} d\Delta$ is given as:

Kernel	$\boldsymbol{k(\Delta)}$	$\boldsymbol{p(\omega)}$
Gaussian	$e^{-\frac{\|\Delta\|_2^2}{2}}$	$(2\pi)^{\frac{d}{2}} e^{-\frac{\|\Delta\|_2^2}{2}}$
Exponential	$e^{-\|\Delta\|_1}$	$\prod_{j=1}^{d} \frac{1}{\pi(1+\omega_j^2)}$
Cauchy	$\prod_{j=1}^{d} \frac{2}{1+\Delta_j^2}$	$e^{-\|\omega\|_1}$

The key idea is now to interpret the kernel as an expectation. Using elementary trigonometric formulae, we get

$$\begin{aligned} k\left(\boldsymbol{x} - \boldsymbol{x}'\right) = \int_{\mathbb{R}^d} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top \left(\boldsymbol{x} - \boldsymbol{x}'\right)} d\boldsymbol{\omega} &= \mathbb{E}_{\boldsymbol{\omega} \sim p} \mathbb{E}_{b \sim \mathrm{Unif}([0, 2\pi])} \left[2 \cos\left(\boldsymbol{\omega}^\top \boldsymbol{x} + b\right) \cos\left(\boldsymbol{\omega}^\top \boldsymbol{x}' + b\right)\right] \\ &= \mathbb{E}_{\boldsymbol{\omega} \sim p, b \sim \mathrm{Unif}([0, 2\pi])} \left[z_{\boldsymbol{\omega}, b}(\boldsymbol{x}) \cdot z_{\boldsymbol{\omega}, b}\left(\boldsymbol{x}'\right)\right], \end{aligned}$$

where $z_{\omega, b}(x) \doteq \sqrt{2} \cos\left(\boldsymbol{\omega}^\top \boldsymbol{x} + b\right)$. Then, using Monte Carlo sampling to estimate the expectation and drawing independent samples $\omega^{(i)} \overset{\text{iid}}{\sim} p$ and $b^{(i)} \overset{\text{iid}}{\sim} \mathrm{Unif}([0, 2\pi])$ we get

$$k\left(\boldsymbol{x} - \boldsymbol{x}'\right) \approx \frac{1}{m} \sum_{i=1}^{m} z_{\boldsymbol{\omega}^{(i)}, b^{(i)}}(\boldsymbol{x}) \cdot z_{\boldsymbol{\omega}^{(i)}, b^{(i)}}\left(\boldsymbol{x}'\right) = z(x)^\top z\left(x'\right)$$

where

$$\boldsymbol{z}(\boldsymbol{x}) \doteq \frac{1}{\sqrt{m}} \left[z_{\boldsymbol{\omega}^{(1)}, b^{(1)}}(\boldsymbol{x}), \ldots, z_{\boldsymbol{\omega}^{(m)}, b^{(m)}}(\boldsymbol{x})\right]^\top$$

is the (RANDOMIZED) FEATURE MAP OF RANDOM FOURIER FEATURES.
Rahimi et al. (2007) show that Bayesian linear regression with the feature map z approximates Gaussian processes with a stationary kernel.

Theorem 12 (Uniform convergence of Fourier features)**.** *Suppose $\mathcal{M}$ is a compact subset of $\mathbb{R}^d$ with diameter* $\operatorname{diam}(\mathcal{M})$*. Then for a stationary kernel k, the random Fourier features z, and any $\epsilon > 0$ it holds that*

$$\mathbb{P}\left(\sup_{x,x' \in \mathcal{M}} \left|z(x)^\top z\left(x'\right) - k\left(x - x'\right)\right| \geq \epsilon\right) \leq 2^8 \left(\frac{\sigma_p \operatorname{diam}(\mathcal{M})}{\epsilon}\right)^2 \exp\left(-\frac{m\epsilon^2}{8(d+2)}\right)$$

where $\sigma_p^2 \doteq \mathbb{E}_{\boldsymbol{\omega} \sim p}\left[\boldsymbol{\omega}^\top \boldsymbol{\omega}\right]$ is the second moment of p, m is the dimension of $z(x)$, and d is the dimension of $\boldsymbol{x}$.

Note that the error probability decays exponentially fast in the dimen- sion of the Fourier feature space. Moreover, Fourier features approximate the kernel function uniformly well, but this could be wasteful: we only need accurate representation for training and test points, so we don't need uniform convergence.

3.5.3 Inducing point methods

Another natural approach is to only consider a random subset of the training data during learning. The naïve approach is to subsample uniformly at random. Not very surprisingly, we can do much better. One subsampling method is the so-called INDUCING POINTS METHOD. The idea is to summarize the data around so-called inducing points. The inducing points can be treated as hyperparameters. For now, let us consider an arbitrary set of inducing points,

$$U \doteq \{\bar{x}_1, \ldots, \bar{x}_k\}.$$

Then, the original Gaussian process can be recovered using marginalization,

$$p\left(f^\star, \boldsymbol{f}\right) = \int_{\mathbb{R}^k} p\left(f^\star, \boldsymbol{f}, \boldsymbol{u}\right) d\boldsymbol{u} = \int_{\mathbb{R}^k} p\left(f^\star, \boldsymbol{f} \mid \boldsymbol{u}\right) p(\boldsymbol{u}) d\boldsymbol{u},$$

where $f \doteq \left[f\left(x_1\right) \cdots f\left(x_n\right)\right]^\top$ and $f^\star \doteq f\left(x^\star\right)$ at some evaluation point $\boldsymbol{x}^\star \in \mathcal{X}$. We use $\boldsymbol{u} \doteq \left[f\left(\overline{\boldsymbol{x}}_1\right) \cdots f\left(\overline{\boldsymbol{x}}_k\right)\right]^\top \in \mathbb{R}^k$ to denote the predictions of the model at the inducing points U. Due to the marginalization property of Gaussian processes (3.1), we have that $\boldsymbol{u} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{K}_{UU}\right)$. The key idea is to approximate the joint prior, assuming that $f^\star$ and f are conditionally independent given $\boldsymbol{u}$,

$$p\left(f^\star, \boldsymbol{f}\right) \approx \int_{\mathbb{R}^k} p\left(f^\star \mid \boldsymbol{u}\right) p(\boldsymbol{f} \mid \boldsymbol{u}) p(\boldsymbol{u}) d\boldsymbol{u}.$$

Here, $p(\boldsymbol{f} \mid \boldsymbol{u})$ and $p\left(f^\star \mid \boldsymbol{u}\right)$ are commonly called the TRAINING CONDITIONAL and the TESTING CONDITIONAL, respectively. Still denoting the observations by $A = \{x_1, \ldots, x_n\}$ and defining $\star \doteq \{x^\star\}$, we know, using the closed-form expression for conditional Gaussians (1.5),

$$\begin{aligned} p(f \mid \boldsymbol{u}) &\sim \mathcal{N}\left(f; \boldsymbol{K}_{AU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \boldsymbol{K}_{AA} - \boldsymbol{Q}_{AA}\right), \\ p\left(f^\star \mid \boldsymbol{u}\right) &\sim \mathcal{N}\left(f^\star; \boldsymbol{K}_{\star U} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \boldsymbol{K}_{\star\star} - \boldsymbol{Q}_{\star\star}\right) \end{aligned} \tag{3.5}$$

where $\boldsymbol{Q}_{ab} \doteq \boldsymbol{K}_{aU} \boldsymbol{K}_{UU}^{-1} K_{Ub}$. Intuitively, $\boldsymbol{K}_{AA}$ represents the prior covariance and Q_{AA} represents the covariance "explained" by the inducing points.
Computing the full covariance matrix is expensive. The following two examples show two approximations to the covariance of the training conditional and testing conditional.

Example 2 (Subset of regressors (SoR))**.** *The* SUBSET OF REGRESSORS *(SoR) approximation is defined as*

$$\begin{aligned} p(f \mid \boldsymbol{u}) &\approx q_{\mathrm{SoR}}(\boldsymbol{f} \mid \boldsymbol{u}) \doteq \mathcal{N}\left(\boldsymbol{f}; \boldsymbol{K}_{AU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \mathbf{0}\right), \\ p\left(f^\star \mid \boldsymbol{u}\right) &\approx q_{\mathrm{SoR}}\left(f^\star \mid \boldsymbol{u}\right) \doteq \mathcal{N}\left(f^\star; \boldsymbol{K}_{\star U} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \mathbf{0}\right). \end{aligned}$$

Comparing to eq. (3.5)*, SoR simply forgets about all variance and covariance. The resulting model is a* DEGENERATE GAUSSIAN PROCESS *(the covariance function has a finite number of non-zero eigenvalues) with covariance function*

$$k_{\mathrm{SoR}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) \doteq \boldsymbol{k}_{\boldsymbol{x},U}^\top \boldsymbol{K}_{UU}^{-1} \boldsymbol{k}_{\boldsymbol{x}',U}.$$

Example 3 (Fully independent training conditional (FITC))**.** *The* FULLY INDEPENDENT TRAINING CONDITIONAL *(FITC) approximation is defined as*

$$\begin{aligned} q_{\mathrm{FITC}}(\boldsymbol{f} \mid \boldsymbol{u}) &\doteq \mathcal{N}\left(\boldsymbol{f}; \boldsymbol{K}_{AU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \operatorname{diag}\left\{\boldsymbol{K}_{AA} - \boldsymbol{Q}_{AA}\right\}\right), \\ q_{\mathrm{FITC}}\left(f^\star \mid \boldsymbol{u}\right) &\doteq \mathcal{N}\left(f^\star; \boldsymbol{K}_{\star U} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \operatorname{diag}\left\{\boldsymbol{K}_{\star\star} - \boldsymbol{Q}_{\star\star}\right\}\right). \end{aligned}$$

In contrast to SoR, FITC keeps track of the variances but forgets about the covariance (since it uses only the diag*).*

The computational cost for inducing point methods SoR and FITC is dominated by the cost of inverting K_{UU}. Thus, the time complexity is cubic in the number of inducing points, but only linear in the number of data points.

3.6 Summary

- Gaussian process = normal distribution over functions, kernelized Bayesian Linear Regression
- Finite marginals are multivariate Gaussians
- Closed form formulae for Bayesian posterior update exist
- Parameterized by covariance function k(x,x') = Cov(f(x),f(x'))
- Can optimize hyperparameters via maximizing the marginal likelihood
- Various fast approximations to exact Gaussian process inference exist

Chapter 4

Variational Inference

Let's recall how bayesian learining works in general:

$$p(\boldsymbol{\theta}) \quad \text{(PRIOR)}$$

$$p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \quad \text{(LIKELIHOOD)}$$

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) = \frac{1}{Z} p(\boldsymbol{\theta}) \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \quad \text{where} \quad Z \doteq \int p(\boldsymbol{\theta}) \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})\, d\boldsymbol{\theta} \quad \text{(POSTERIOR)}$$

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) = \int p(y^\star, \boldsymbol{\theta} \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n})\, d\boldsymbol{\theta} = \int p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})\, d\boldsymbol{\theta} \quad \text{(predictive posterior)}$$

For Bayesian linear regression and GP regression, these high-dimensional integrals are closed-form but, in general, namely if we work with other distributions, this is not the case. We'll assume we can evaluate the joint distribution, but not the normalizer Z.
In this and the following chapter, we will discuss two methods of approximate inference. We begin by discussing VARIATIONAL INFERENCE, which aims to find a good approximation of the posterior distribution from which it is easy to sample. In the next chapter instead, we discuss MARKOV CHAIN MONTE CARLO METHODS, which approximate the sampling from the posterior distribution directly.
The fundamental idea behind variational inference is to approximate the true posterior distribution using a simpler posterior that is as close as possible to the true posterior:

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) = \frac{1}{Z} p(\boldsymbol{\theta}, y_{1:n} \mid \boldsymbol{x}_{1:n}) \approx q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \doteq q_\lambda(\boldsymbol{\theta})$$

where λ represents the parameters of the VARIATIONAL POSTERIOR q_λ, also called VARIATIONAL PARAMETERS. In doing so, variational inference reduces learning and inference, where the fundamental difficulty lies in solving high-dimensional integrals, to an optimization problem.

4.1 Laplace Approximation

Before introducing a general framework of variational inference, we discuss a simpler method of approximate inference known as Laplace's method. This method was proposed as a method of approximating integrals and the idea is to use a Gaussian approximation, that is, a second-order Taylor approximation, of the posterior distribution around its mode, namely around the MAP estimate. Let

$$\psi(\boldsymbol{\theta}) \doteq \log p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})$$

denote the log-posterior. Then, using a second-order Taylor approximation around the mode $\hat{\boldsymbol{\theta}}$ of ψ, we obtain

$$\psi(\boldsymbol{\theta}) \approx \hat{\psi}(\boldsymbol{\theta}) \doteq \psi(\hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\nabla} \psi(\hat{\boldsymbol{\theta}}) + \frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}}) (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) = \psi(\hat{\boldsymbol{\theta}}) + \frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}}) (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}).$$

Comparing this expression to the log of the PDF of a Gaussian,

$$\log \mathcal{N}\left(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, \boldsymbol{\Lambda}^{-1}\right) = -\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\Lambda}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + \text{ const.}$$

we get

$$\hat{\psi}(\boldsymbol{\theta}) = \psi(\hat{\boldsymbol{\theta}}) + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) = \log \mathcal{N}\left(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, -\boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}})^{-1}\right) + \text{ const},$$

where we note that $\psi(\hat{\boldsymbol{\theta}})$ is a constant. We therefore define

$$\boldsymbol{\Lambda} \doteq -\boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}}) = -\left.\boldsymbol{H}_{\boldsymbol{\theta}} \log p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right)\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}. \tag{4.1}$$

Observe that $\boldsymbol{\Lambda}$ is symmetric and ψ is a concave function, so $\boldsymbol{H}_\psi(\boldsymbol{\theta})$ is negative semi-definite implying that $\boldsymbol{\Lambda}$ is positive semi-definite. Using that the inverse of a positive semi-definite matrix is also positive semi-definite, $\boldsymbol{\Lambda}^{-1}$ is a valid covariance matrix. The LAPLACE APPROXIMATION q of p is then given by

$$q(\boldsymbol{\theta}) \doteq \mathcal{N}\left(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, \boldsymbol{\Lambda}^{-1}\right) \propto \exp(\hat{\psi}(\boldsymbol{\theta})).$$

Intuitively, the Laplace approximation matches the shape of the true posterior around its mode but may not represent it accurately elsewhere - often leading to extremely overconfident predictions

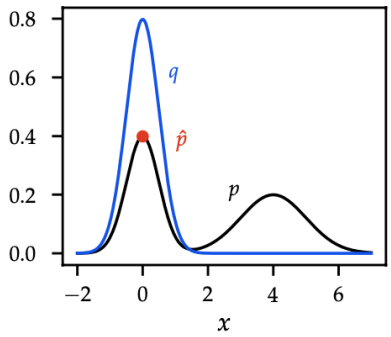


Figure 4.1: The Laplace approximation q greedily selects the mode of the true posterior distribution p and matches the curvature around the mode $\hat{p}$. As shown here, the Laplace approximation can be extremely overconfident when p is not approximately Gaussian.

4.1.1 Bayesian Logistic Regression

As an example, we will look at Laplace approximation in the context of Bayesian logistic regression. Logistic regression learns a classifier that decides for a given input whether it belongs to one of two classes. A SIGMOID FUNCTION, typically the logistic function,

$$\sigma(z) \doteq \frac{1}{1 + \exp(-z)} \in (0, 1), \quad z = \boldsymbol{w}^\top \boldsymbol{x}, \tag{4.2}$$

is used to obtain the class probabilities. BAYESIAN LOGISTIC REGRESSION corresponds to Bayesian linear regression with a Bernoulli likelihood,

$$y \mid \boldsymbol{x}, \boldsymbol{w} \sim \text{Bern}\left(\sigma\left(\boldsymbol{w}^\top \boldsymbol{x}\right)\right), \tag{4.3}$$

where $y \in \{-1, 1\}$ is the binary class label (the same approach extends to Gaussian processes where it is known as Gaussian process classification). Observe that given a data point (x, y), the probability of a correct classification is

$$p(y \mid \boldsymbol{x}, \boldsymbol{w}) = \begin{cases} \sigma\left(\boldsymbol{w}^\top \boldsymbol{x}\right) & \text{if } y = 1 \\ 1 - \sigma\left(\boldsymbol{w}^\top \boldsymbol{x}\right) & \text{if } y = -1 \end{cases} = \sigma\left(y \boldsymbol{w}^\top \boldsymbol{x}\right) \tag{4.4}$$

as the logistic function σ is symmetric around 0 . Also, we will use the same prior used in Bayesian linear regression, namely

$$p(\boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{w}; \boldsymbol{0}, \sigma_p^2 \boldsymbol{I}\right) \propto \exp\left(-\frac{1}{2\sigma_p^2}\|\boldsymbol{w}\|_2^2\right).$$

Let us first find the posterior mode, that is, the MAP estimate of the weights:

$$\hat{\boldsymbol{w}} = \arg\max_{w} p\left(w \mid x_{1:n}, y_{1:n}\right) = \arg\max_{w} p(\boldsymbol{w}) p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}\right) = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}) + \log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}\right) =$$
$$\arg\max_{\boldsymbol{w}} -\frac{1}{2\sigma_p^2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \log \sigma\left(y_i \boldsymbol{w}^\top \boldsymbol{x}_i\right) = \arg\min_{\boldsymbol{w}} \frac{1}{2\sigma_p^2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i\right)\right).$$

Note that for $\lambda = 1/2\sigma_p^2$, the above optimization is equivalent to standard (regularized) logistic regression where

$$\ell_{\log}\left(\boldsymbol{w}^\top \boldsymbol{x}; y\right) \doteq \log\left(1 + \exp\left(-y \boldsymbol{w}^\top \boldsymbol{x}\right)\right)$$

is called LOGISTIC LOSS. Also, we don't need to know normalizer Z to get the MAP estimate.
The gradient of the logistic loss is given by

$$\boldsymbol{\nabla}_{\boldsymbol{w}} \ell_{\log}\left(\boldsymbol{w}^\top \boldsymbol{x}; \boldsymbol{y}\right) = -y\boldsymbol{x} \cdot \sigma\left(-y \boldsymbol{w}^\top \boldsymbol{x}\right).$$

Recall that due to the symmetry of σ around $0, \sigma\left(-y w^\top \boldsymbol{x}\right)$ is the probability that x was not classified as y. Intuitively, if the model is "surprised" by the label, the gradient is large. We can therefore use SGD with the (regularized) gradient step and with batch size 1 ,

$$\boldsymbol{w} \leftarrow \boldsymbol{w}\left(1 - 2\lambda\eta_t\right) + \eta_t y x \sigma\left(-y \boldsymbol{w}^\top \boldsymbol{x}\right),$$

for the data point (x, y) picked uniformly at random from the training data.
We have already found the mode of the posterior distribution, $\hat{\boldsymbol{w}}$. Let us denote by

$$\pi_i \doteq \mathbb{P}\left(y_i = 1 \mid \boldsymbol{x}_i, \hat{\boldsymbol{w}}\right) = \sigma\left(\hat{\boldsymbol{w}}^\top \boldsymbol{x}_i\right)$$

the probability of x_i belonging to the positive class under the model given by the MAP estimate of the weights. For the precision matrix, using (4.1), the definition of the logistic loss and the Hessian of the logistic loss, we get

$$\boldsymbol{\Lambda} = -\boldsymbol{H}_{\boldsymbol{w}} \log p\left(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right)\big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} = -\boldsymbol{H}_{\boldsymbol{w}} \log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}\right)\big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} - \boldsymbol{H}_{\boldsymbol{w}} \log p(\boldsymbol{w})\big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}}$$
$$= \sum_{i=1}^{n} \boldsymbol{H}_{\boldsymbol{w}} \ell_{\log}\left(\boldsymbol{w}^\top \boldsymbol{x}_i; y_i\right)\Big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} + \sigma_p^{-2}\boldsymbol{I} = \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^\top \pi_i\left(1 - \pi_i\right) + \sigma_p^{-2}\boldsymbol{I} = \boldsymbol{X}^\top \operatorname{diag}_{i\in[n]}\left\{\pi_i\left(1 - \pi_i\right)\right\} \boldsymbol{X} + \sigma_p^{-2}\boldsymbol{I}.$$

Observe that $\pi_i\left(1 - \pi_i\right) \approx 0$ if $\pi_i \approx 1$ or $\pi_i \approx 0$. That is, if a training example is "well-explained" by $\hat{w}$, then its contribution to the precision matrix is small. In contrast, we have $\pi_i\left(1 - \pi_i\right) = 0.25$ for $\pi_i = 0.5$. Importantly, $\boldsymbol{\Lambda}$ does not depend on the normalization constant of the posterior distribution which is hard to compute. Overall, we have that $\mathcal{N}\left(\hat{\boldsymbol{w}}, \boldsymbol{\Lambda}^{-1}\right)$ is the Laplace approximation of $p\left(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right)$.

4.2 Inference with a Variational Posterior

How can we perform inference using our variational approximation? We simply approximate the (intractable) true posterior with our variational posterior,

$$p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}\right) = \int p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{w}\right) p\left(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) d\boldsymbol{w} \approx \int p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{w}\right) q_\lambda(\boldsymbol{w}) d\boldsymbol{w}.$$

In the case of Bayesian logistic regression with a Laplace approximation, the posterior is given by the Gaussian $q_\lambda(\boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{w}; \hat{\boldsymbol{w}}, \boldsymbol{\Lambda}^{-1}\right)$ and the likelihood is a Bernoulli distribution (see (4.3) and (4.4)). Using these properties, defining $f^\star = \boldsymbol{w}^\top \boldsymbol{x}^\star$ we get and performing a change of variables, we get

$$p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}\right) \approx \int \sigma\left(y^\star f^\star\right) \cdot \mathcal{N}\left(f^\star; \hat{\boldsymbol{w}}^\top \boldsymbol{x}^\star, \boldsymbol{x}^\star \boldsymbol{\Lambda}^{-1} \boldsymbol{x}^\star\right) df^\star$$

and thus we have replaced the high-dimensional integral over the model parameters w by the one-dimensional integral over the prediction of our variational posterior $f^\star$. While this integral is generally still intractable, it can be approximated efficiently using numerical quadrature methods such as the Gauss-Legendre quadrature.

4.3 Blueprint of Variational Inference

Laplace approximation approximates the true (intractable) posterior with a simpler one, by greedily matching mode and curvature around it. Can we find "less greedy" approaches? We can view variational Bayesian inference more generally as a family of approaches aiming to approximate the true posterior distribution by one that is closest (according to some criterion) among a "simpler" class of distributions. To this end, we need to fix a class of distributions and define suitable criteria, which we can then optimize numerically. The key benefit is that we can reduce the (generally intractable) problem of high-dimensional integration to the (often much more tractable) problem of optimization, that of finding the "closest" distribution, in a class of simpler ones, to the original one.

Definition 5. *Let $\mathcal{P}$ be the class of all probability distributions. A* VARIATIONAL FAMILY *$\mathcal{Q} \subseteq \mathcal{P}$ is a class of distributions such that each distribution $q \in \mathcal{Q}$ is characterized by unique variational parameters $\lambda \in \Lambda$.*

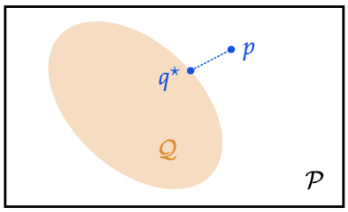


Figure 4.2: An illustration of variational inference in the space of distributions $\mathcal{P}$. The variational distribution $q^\star \in \mathcal{Q}$ is the optimal approximation of the true posterior p.

Example 4. *A straightforward example for a variational family is the family of* INDEPENDENT GAUSSIANS*,*

$$\mathcal{Q} \doteq \left\{ q(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{\theta}; \boldsymbol{\mu}, \operatorname{diag}_{i\in[d]}\left\{\sigma_i^2\right\}\right)\right\},$$

which is parameterized by $\lambda \doteq \left[\mu_{1:d}, \sigma^2_{1:d}\right]$. Importantly, this family of distributions is characterized by only $2d$ parameters.

A common notion of distance between two distributions q and p is the KULLBACK-LEIBLER DIVERGENCE $\mathrm{KL}(q\|p)$ which we will define in the next section. Using this notion of distance, we need to solve the following optimization problem:

$$q^\star \doteq \arg\min_{q\in\mathcal{Q}} \mathrm{KL}(q\|p) = \arg\min_{\lambda\in\Lambda} \mathrm{KL}\left(q_\lambda\|p\right)$$

In section 4.4, we introduce information theory and the Kullback-Leibler divergence. Then, in section 4.5, we discuss how the optimization problem of eq. (5.30) can be solved efficiently.

4.4 Information Theory

4.4.1 Surprise and Entropy

One of our main objectives throughout this course is to capture the "uncertainty" about events A in an appropriate probability space. One very natural measure of uncertainty is their probability, $\mathbb{P}(A)$. In this section, we will introduce an alternative measure of uncertainty, namely the so-called SUPRISE about the event A.

Definition 6. *The* SURPRISE *about an event with probability u is defined as*

$$\mathrm{S}[u] \doteq -\log u.$$

It is a function from $\mathbb{R}_{\geq 0}$ to $\mathbb{R}$, where we let $\mathrm{S}[0] \equiv \infty$. Moreover, since $p(x) \leq 1$, we have $\mathrm{S}[p(x)] \geq 0$.

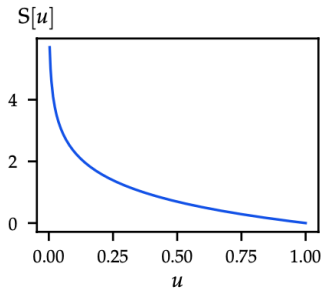


Figure 4.3: Surprise $S[u]$ associated with an event of probability u.

But why is it reasonable to measure surprise by $-\log u$?

Theorem 13 (Axiomatic characterization of surprise)**.** *The axioms*

1. $\mathrm{S}[1] = 0$*, namely "certain events are not surprising"*
2. $\mathrm{S}[u] > \mathrm{S}[v] \implies u < v$ *(anti-monotonicity), namely "we are more surprised by unlikely events"*
3. S *is continuous, namely "no jumps in surprise for infinitesimal changes of probability"*
4. $\mathrm{S}[uv] = \mathrm{S}[u] + \mathrm{S}[v]$ *for independent events*

characterize S up to a positive constant factor.

Importantly, surprise offers a different perspective on uncertainty as opposed to probability: the uncertainty about an event can either be interpreted in terms of its probability or in terms of its surprise, and the two "spaces of uncertainty" are related by a log-transform.

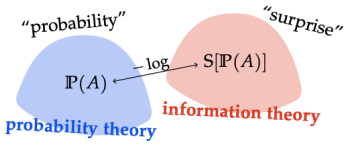


Figure 4.4: Illustration of the probability space and the corresponding "surprise space".

Definition 7. *The* ENTROPY *of a distribution p is the average surprise about samples from p. Thus entropy is a notion of uncertainty associated with the distribution p: if the entropy of p is large, we are more uncertain about $x \sim p$ than if the entropy of p were low. Formally,*

$$\mathrm{H}[p] \doteq \mathbb{E}_{x\sim p}[\,\mathrm{S}[p(x)]] = \mathbb{E}_{x\sim p}[-\log p(x)] = -\int p(\theta)\log(p(\theta))d\theta.$$

From the fourth axiom of surprise, we immediately obtain the following property of entropy.

Theorem 14. *For a multivariate distribution p that can be factorized into $p\left(x_{1:d}\right) = \prod_{i=1}^{d} p_i\left(x_i\right)$, we have that*

$$\mathrm{H}[p] = \sum_{i=1}^{d} \mathrm{H}\left[p_i\right]$$

Example 5. *The entropy of a univariate Gaussian, whose PDF is*

$$\mathcal{N}\left(x; \mu, \sigma^2\right) = \frac{1}{Z}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad \text{where } Z = \sqrt{2\pi\sigma^2}$$

is given by

$$\mathrm{H}\left[\mathcal{N}\left(\mu, \sigma^2\right)\right] = \log(\sigma\sqrt{2\pi e}).$$

In general, the entropy of a Gaussian is

$$\mathrm{H}[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})] = \frac{1}{2}\log\det(2\pi e\boldsymbol{\Sigma}) = \frac{1}{2}\log\left((2\pi e)^d \det(\boldsymbol{\Sigma})\right)$$

How can we use entropy to measure our average surprise when assuming the data follows some distribution q but in reality the data follows a different distribution p?

Definition 8. *The* CROSS-ENTROPY *of a distribution q relative to the distribution p is*

$$\mathrm{H}[p\|q] \doteq \mathbb{E}_{x\sim p}[\,\mathrm{S}[q(x)]] = \mathbb{E}_{x\sim p}[-\log q(x)] = -\int p(\theta)\log q(\theta)d\theta.$$

4.4.2 Kullback-Leibler Divergence

As mentioned, the Kullback-Leibler divergence is a (non-metric) measure of distance between distributions.

Definition 9. *Given two distributions p and q, the*KULLBACK-LEIBLER DIVERGENCE *of q with respect to p, given by*

$$\mathrm{KL}(p\|q) \doteq \mathrm{H}[p\|q] - \mathrm{H}[p] = \mathbb{E}_{\boldsymbol{\theta}\sim p}[\,\mathrm{S}[q(\boldsymbol{\theta})] - \mathrm{S}[p(\boldsymbol{\theta})]] = \mathbb{E}_{\boldsymbol{\theta}\sim p}\left[\log\frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right] = \int q(\theta)\log\frac{q(\theta)}{p(\theta)}d\theta$$

measures how different q is from a reference distribution p.

In words, $\mathrm{KL}(p\|q)$ measures the additional expected surprise when observing samples from p that is due to assuming the (wrong) distribution q and which not inherent in the distribution p already. Intuitively, the KL-divergence measures the information loss when approximating p with q.

Theorem 15. *The KL-divergence has the following properties:*

- $\mathrm{KL}(p\|q) \geq 0$ *for any distributions p and q,*
- $\mathrm{KL}(p\|q) = 0$ *if and only if $p = q$ almost surely, and*
- *there exist distributions p and q such that* $\mathrm{KL}(p\|q) \neq \mathrm{KL}(q\|p)$.

The KL-divergence can simply be understood as a shifted version of cross-entropy, which is zero if we consider the divergence between two identical distributions.

Example 6 (KL-divergence of Gaussians)**.** *Suppose we are given two Gaussian distributions $p \doteq \mathcal{N}\left(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p\right)$ and $q \doteq \mathcal{N}\left(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q\right)$ with dimension d. Then we have a closed form for the KL-divergence of p and q:*

$$\mathrm{KL}(p\|q) = \frac{1}{2}\left(\mathrm{tr}\left(\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p\right) + \left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\right)^\top \boldsymbol{\Sigma}_q^{-1}\left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\right) - d + \log\frac{\det\left(\boldsymbol{\Sigma}_q\right)}{\det\left(\boldsymbol{\Sigma}_p\right)}\right). \tag{4.5}$$

For independent Gaussians with unit variance $p \doteq \mathcal{N}\left(\boldsymbol{\mu}_p, \boldsymbol{I}\right)$ an $q \doteq \mathcal{N}\left(\boldsymbol{\mu}_q, \boldsymbol{I}\right)$, the expression simplifies to the squared Euclidear distance,

$$\mathrm{KL}(p\|q) = \frac{1}{2}\left\|\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right\|_2^2.$$

If we approximate independent Gaussians with variances σ_i^2, $p \doteq \mathcal{N}\left(\boldsymbol{\mu}, \mathrm{diag}\left\{\sigma_1^2, \ldots, \sigma_d^2\right\}\right)$ by a standard normal distribution $q \doteq \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, the expression simplifies to

$$\mathrm{KL}(p\|q) = \frac{1}{2}\sum_{i=1}^{d}\left(\sigma_i^2 + \mu_i^2 - 1 - \log\sigma_i^2\right). \tag{4.6}$$

$\mathrm{KL}(p\|q)$ is also called the FORWARD KL-DIVERGENCE. In contrast, $\mathrm{KL}(q\|p)$ is called the REVERSE KL-DIVERGENCE. The next figure shows the approximations of a general Gaussian obtained when $\mathcal{Q}$ is the family of diagonal (independent) Gaussians. Thereby,

$$q_1^\star \doteq \underset{q\in\mathcal{Q}}{\arg\min}\mathrm{KL}(p\|q) \quad q_2^\star \doteq \underset{q\in\mathcal{Q}}{\arg\min}\mathrm{KL}(q\|p)$$

$q_1^\star$ is the result when using the forward KL-divergence and $q_2^\star$ is the result when using reverse KL-divergence.

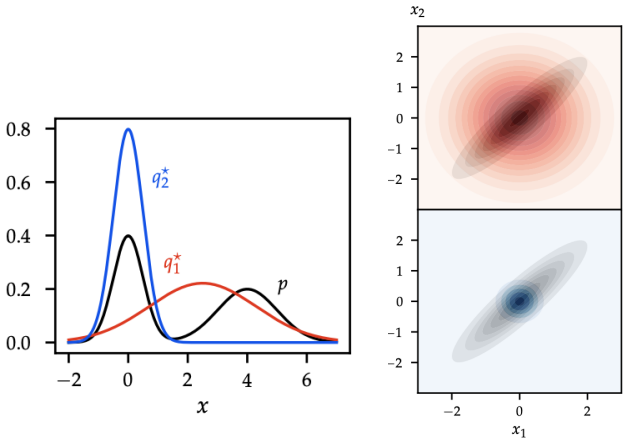


Figure 4.5: Comparison of the forward KL-divergence $q_1^\star$ and the reverse KL-divergence $q_2^\star$ when used to approximate the true posterior p. The first plot shows the PDFs in a one-dimensional feature space where p is a mixture of two univariate Gaussians. The second plot shows contour lines of the PDFs in a two-dimensional feature space where the non-diagonal Gaussian p is approximated by diagonal Gaussians $q_1^\star$ (forward KL) and $q_2^\star$ (reverse KL). It can be seen that $q_1^\star$ selects the variance and $q_2^\star$ selects the mode of p. The approximation $q_1^\star$ is more conservative than the (overconfident) approximation $q_2^\star$.

It can be seen that the reverse KL-divergence tends to greedily select the mode and underestimating the variance which, in this case, leads to an overconfident prediction. The forward KL-divergence, in contrast, is more conservative and yields what one could consider the "desired" approximation.
Recall that in the blueprint of variational inference (5.30) we used the reverse KL-divergence. This is for computational reasons. Indeed, to approximate the KL-divergence $\mathrm{KL}(p\|q)$ using Monte Carlo sampling, we would need to obtain samples from p yet p is the intractable posterior distribution which we were trying to approximate in the first place.

4.5 Evidence Lower Bound (ELBO)

To complete the blueprint of variational inference from section 4.3, it remains to show that the reverse KL-divergence can be minimized efficiently. By omitting the dependence on $\boldsymbol{x}_{1:n}$ and the subscript $1:n$ for y for notational simplicity, we have

$$q_\lambda^* = \arg\min_q KL(q\|p) = \arg\min_q \int q(\theta)\log\frac{q(\theta)}{\frac{1}{Z}p(\theta,y)}d\theta = \arg\max_q \int q(\theta)[\log p(\theta,y) - \log Z - \log q(\theta)]d\theta$$
$$= \arg\max_q \int q(\theta)\log p(\theta,y)d\theta + H(q) = \arg\max_q \mathbb{E}_{\theta\sim q(\theta)}[\log p(\theta,y)] + H(q) =$$
$$= \arg\max_q \mathbb{E}_{\theta\sim q(\theta)}[\log p(y\mid\theta)] - KL(q\|p) = \arg\max_q \ \mathrm{ELBO}(q,p,\mathcal{D})$$

where $\mathcal{D} = \{(x_i,y_i)\}_{i=1}^n$ and

$$\mathrm{ELBO}(q,p,\mathcal{D}) = \mathbb{E}_{\theta\sim q(\theta)}[\log p(y\mid\theta)] - KL(q\|p) = \mathbb{E}_{\theta\sim q(\theta)}[\log p(\theta,y)] + H(q) \quad \text{(also denoted with } L\left(q_\lambda,p;\mathcal{D}\right)\text{)}$$

is called EVIDENCE LOWER BOUND given the data $\mathcal{D}$.
Thus, maximizing the ELBO coincides with minimizing reverse-KL. Also note that, by maximizing the ELBO, we prefer distributions q that maximize the joint/conditional data likelihood, but are also uncertain/close to the prior.
Now, since $\log$ is a concave function, recalling Jensen's Inequality (see Theorem 1) and the law of Total Probability (1.3), we get

$$\log p(y) = \log\int p(y\mid\theta)p(\theta)d\theta = \log\int p(y\mid\theta)\frac{p(\theta)}{q(\theta)}q(\theta)d\theta = \log\mathbb{E}_{\theta\sim q}\left[p(y\mid\theta)\frac{p(\theta)}{q(\theta)}\right]$$
$$\geq \mathbb{E}_{\theta\sim q}\left[\log\left(p(y\mid\theta)\frac{p(\theta)}{q(\theta)}\right)\right] = \mathbb{E}_{\theta\sim q}[\log p(y\mid\theta)] - KL(q\|p(\cdot))$$

This explains the name of ELBO: the evidence lower bound is a uniform lower bound to the evidence $\log p\left(y_{1:n} \mid x_{1:n}\right)$.

Example 7 (ELBO for Bayesian logistic regression)**.** *Bayesian logistic regression uses the prior distribution $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Suppose we use the variational family $\mathcal{Q}$ of all diagonal Gaussians from example 4. We have already seen (see* (4.6)*) that, if we approximate $q \doteq \mathcal{N}\left(\boldsymbol{\mu}, \operatorname{diag}\left\{\sigma_1^2, \ldots, \sigma_d^2\right\}\right)$ by a standard normal distribution $p \doteq \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, we have*

$$\mathrm{KL}(q \| p(\cdot))=\frac{1}{2} \sum_{i=1}^{d}\left(\sigma_i^2+\mu_i^2-1-\log \sigma_i^2\right).$$

It remains to find the expected likelihood under models from our approximate posterior: using the independence of the data and the logistc loss, we get

$$\mathbb{E}_{\boldsymbol{w} \sim q_\lambda}\left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}\right)\right]=\mathbb{E}_{\boldsymbol{w} \sim q_\lambda}\left[\sum_{i=1}^{n} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}\right)\right]=\mathbb{E}_{\boldsymbol{w} \sim q_\lambda}\left[-\sum_{i=1}^{n} \ell_{\log}\left(\boldsymbol{w} ; \boldsymbol{x}_i, y_i\right)\right].$$

Thus the ELBO for Bayesian logistic regression is given by

$$\mathbb{E}_{w \sim q_\lambda}[\log p(y \mid w)]-K L(q \| p)=\mathbb{E}_{\boldsymbol{w} \sim q_\lambda}\left[-\sum_{i=1}^{n} \ell_{\log}\left(\boldsymbol{w} ; \boldsymbol{x}_i, y_i\right)\right]-\frac{1}{2} \sum_{i=1}^{d}\left(\sigma_i^2+\mu_i^2-1-\log \sigma_i^2\right).$$

4.5.1 Gradient of Evidence Lower Bound and Reparameterization Trick

The next problem is solving the optimization problem of maximizing the ELBO. Thus, we need to obtain gradient estimates of

$$\nabla_\lambda L\left(q_\lambda, p ; \mathcal{D}\right)=\nabla_\lambda \mathbb{E}_{\boldsymbol{\theta} \sim q_\lambda}\left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)\right]-\nabla_\lambda \mathrm{KL}\left(q_\lambda \| p(\cdot)\right)$$

Typically, the KL-divergence and its gradient can be computed exactly for commonly used variational families. For example, we have already seen a closed-form expression of the KL-divergence for Gaussians in eq. (4.5). Obtaining the gradient of the evidence lower bound is more difficult. This is because the expectation integrates over the measure q_λ, which depends on the variational parameters λ. Thus, we cannot move the gradient operator inside the expectation and the main technique used to rewrite the gradient in such a way that Monte Carlo sampling becomes possible is called REPARAMETERIZATION TRICK.

Theorem 16 (Reparameterization trick)**.** *Given a random variable $\epsilon \sim \phi$, which is independent of λ, and given a differentiable and invertible function $\boldsymbol{g}: \mathbb{R}^d \rightarrow \mathbb{R}^d$, we let $\boldsymbol{\theta} \doteq \boldsymbol{g}(\boldsymbol{\epsilon}, \boldsymbol{\lambda})$. Then it holds*

$$q_\lambda(\boldsymbol{\theta})=\phi(\boldsymbol{\epsilon}) \cdot\left|\nabla_\epsilon \boldsymbol{g}(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right|^{-1} \quad \text{and} \quad \mathbb{E}_{\boldsymbol{\theta} \sim q_\lambda}[\boldsymbol{f}(\boldsymbol{\theta})]=\mathbb{E}_{\boldsymbol{\epsilon} \sim \phi}[\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{\epsilon}, \boldsymbol{\lambda}))]$$

for a "nice" function $f: \mathbb{R}^d \rightarrow \mathbb{R}^e$.

Thus, after reparameterization, the expectation is w.r.t. to distribution ϕ that does not depend on λ and this allows to obtain stochastic gradients via

$$\nabla_\lambda \mathbb{E}_{\theta \sim q_\lambda}[f(\theta)]=\mathbb{E}_{\epsilon \sim \phi}\left[\nabla_\lambda f(g(\epsilon ; \lambda))\right].$$

If we can find a function g such that $\theta \sim q_\lambda$ for some sampling distribution ϕ independent of λ, we can use the reparameterization trick to simplify our gradient. We call a distribution q_λ which admits reparameterization REPARAMETERIZABLE.

Example 8 (Reparameterization trick for Gaussians)**.** *Suppose we use a Gaussian variational approximation,*

$$q_\lambda(\boldsymbol{\theta}) \doteq \mathcal{N}(\boldsymbol{\theta} ; \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where we assume Σ to be invertible. We have seen in eq. (1.6) *that a Gaussian random vector $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ following a standard normal distribution can be transformed to follow the Gaussian distribution q_λ by using the linear transformation,*

$$\boldsymbol{\theta}=\boldsymbol{g}(\boldsymbol{\epsilon} ; \lambda) \doteq \Sigma^{1 / 2} \epsilon+\mu.$$

In particular, we have

$$\boldsymbol{\epsilon}=\boldsymbol{g}^{-1}(\boldsymbol{\theta} ; \boldsymbol{\lambda})=\boldsymbol{\Sigma}^{-1 / 2}(\boldsymbol{\theta}-\boldsymbol{\mu}) \quad \text{and} \quad \phi(\boldsymbol{\epsilon})=q_\lambda(\boldsymbol{\theta}) \cdot\left|\det\left(\boldsymbol{\Sigma}^{1 / 2}\right)\right|.$$

In the following, we write $C \doteq \Sigma^{1/2}$. Let us now derive the gradient estimate for the evidence lower bound assuming the Gaussian variational approximation from example 8. This approach extends to any reparameterizable distribution. We have

$$\begin{aligned}\boldsymbol{\nabla}_{\boldsymbol{\lambda}} \mathbb{E}_{\boldsymbol{\theta} \sim q_\lambda} \left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)\right] &= \nabla_{\boldsymbol{C}, \boldsymbol{\mu}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})} \left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)\big|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\epsilon}+\boldsymbol{\mu}}\right] \\ &\approx n \cdot \frac{1}{m} \sum_{j=1}^{m} \boldsymbol{\nabla}_{\boldsymbol{C}, \boldsymbol{\mu}} \log p\left(y_{i(j)} \mid x_{i(j)}, \boldsymbol{\theta}\right)\Big|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\epsilon}^{(j)}+\boldsymbol{\mu}}\end{aligned} \tag{4.7}$$

where we used Monte Carlo sampling in the last approximation and $\boldsymbol{\epsilon}^{(j)} \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $i^{(j)} \overset{\text{iid}}{\sim} \operatorname{Unif}([n])$. This yields an unbiased gradient estimate, which we can use with stochastic gradient descent to maximize the evidence lower bound. We have successfully recast the difficult problems of learning and inference as an optimization problem!

The procedure of approximating the true posterior using a variational posterior by maximizing the evidence lower bound using stochastic optimization is also called black BOX STOCHASTIC VARIATIONAL INFERENCE. The only requirement is that we can obtain unbiased gradient estimates from the evidence lower bound (and the likelihood). If we use the variational family of diagonal Gaussians, we only require twice as many parameters as other inference techniques like MAP estimation. The performance can be improved by using natural gradients and variance reduction techniques for the gradient estimates such as control variates.

Chapter 5

Markov Chain Monte Carlo Methods

Variational inference approximates the entire posterior distribution. However, note that the key challenge in Bayesian inference is not learning the posterior distribution, but using the posterior distribution for predictions,

$$p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}\right) = \int p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) d\boldsymbol{\theta}.$$

This integral can be interpreted as an expectation over the posterior distribution,

$$= \mathbb{E}_{\boldsymbol{\theta} \sim p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})}\left[p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right)\right].$$

Observe that the likelihood $f(\boldsymbol{\theta}) \doteq p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right)$ is easy to evaluate. The difficulty lies in sampling from the posterior distribution.
Assuming we can obtain independent samples from the posterior distribution, the Law of Large Numbers suggests as that we can use Monte Carlo sampling to obtain an unbiased estimate of the expectation

$$\approx \frac{1}{m} \sum_{i=1}^{m} f\left(\boldsymbol{\theta}^{(i)}\right) \tag{5.1}$$

for independent $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} p\left(\cdot \mid x_{1:n}, y_{1:n}\right)$. Moreover, by Hoeffding's inequality, if f is bounded in $[0, C]$, then

$$\mathbb{P}\left(\left|\mathbb{E}_{\boldsymbol{\theta} \sim p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})}\left[p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right)\right] - \frac{1}{m} \sum_{i=1}^{m} f\left(\boldsymbol{\theta}^{(i)}\right)\right| \geq \epsilon\right) \leq 2 \exp\left(-\frac{m\epsilon^2}{2C^2}\right)$$

so, probability of error decreases exponentially in m.
Obtaining samples of the posterior distribution is therefore sufficient to perform approximate inference. Recall that the difficulty of computing the posterior p exactly, was in finding the normalizing constant Z,

$$p(x) = \frac{1}{Z} q(x).$$

The key idea of Markov chain Monte Carlo methods is to construct a Markov chain, which is efficient to simulate and has the stationary distribution p.

5.1 Markov Chains

Definition 10. *A* FINITE AND DISCRETE-TIME MARKOV CHAIN *over the state space $S \doteq \{0, \ldots, n-1\}$ is a stochastic process (a sequence of random variables) $(X_t)_{t \in \mathbb{N}_0}$ valued in S such that the* MARKOV PROPERTY

$$X_{t+1} \perp X_{0:t-1} \mid X_t$$

is satisfied. Intuitively, the Markov property states that future behavior is independent of past states given the present state.

We restrict our attention to time-homogeneous Markov chains (that is, the transition probabilities do not change over time) which can be characterized by a TRANSITION FUNCTION

$$p\left(x' \mid x\right) \doteq \mathbb{P}\left(X_{t+1} = x' \mid X_t = x\right).$$

As the state space is finite, we can describe the transition function by the TRANSITION MATRIX

$$\boldsymbol{P} \doteq \begin{bmatrix} p\left(x_1 \mid x_1\right) & \cdots & p\left(x_n \mid x_1\right) \\ \vdots & \ddots & \vdots \\ p\left(x_1 \mid x_n\right) & \cdots & p\left(x_n \mid x_n\right) \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

Note that each row of $\boldsymbol{P}$ must always sum to 1 . Such matrices are also called stochastic.

Definition 11. *A Markov chain is* ERGODIC *iff there exists a $t \in \mathbb{N}_0$ such that for any $x, x' \in S$ we have*

$$p^{(t)}\left(x' \mid x\right) > 0,$$

whereby $p^{(t)}\left(x' \mid x\right)$ is the probability to reach x' from x in exactly t steps. In other words, there exists a finite t such that every state can be reached from every state in exactly t steps. An equivalent condition is that the Markov chain is irreducible and aperiodic.

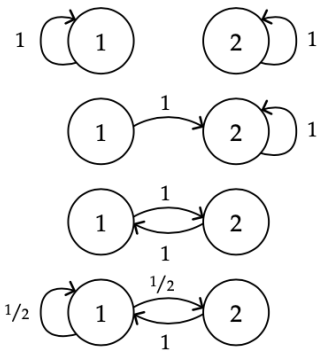


Figure 5.1: (1) is not ergodic as its transition diagram is not strongly connected; (2) is not ergodic for the same reason; (3) is irreducible but periodic and therefore not ergodic; (4) is ergodic with $t = 2$

Definition 12 (Stationary distribution)**.** *A distribution π is* STATIONARY *with respect to the transition function p iff*

$$\pi(x) = \sum_{x' \in S} p\left(x \mid x'\right) \pi\left(x'\right)$$

holds for all $x \in S$.

After entering a stationary distribution π, a Markov chain will always remain in the stationary distribution. In particular, suppose that X_t is distributed according to π, then for all $k \geq 0, X_{t+k} \sim \pi$.

Theorem 17 (Fundamental theorem of ergodic Markov chains)**.** *An ergodic Markov chain has a unique stationary distribution π (with full support) and*

$$\lim_{t \to \infty} P\left(X_t = x\right) = \pi(x)$$

for all x irrespectively of the initial distribution $P(X_1)$.

This naturally suggests constructing an ergodic Markov chain such that its stationary distribution coincides with the posterior distribution. For example, we can simulate a Markov Chain via forward sampling:

- sample $x_1 \sim P\left(X_1\right)$
- sample $x_2 \sim P\left(X_2 \mid X_1 = x_1\right)$

- ...
- sample $x_t \sim P\left(X_t \mid X_{t-1} = x_{t-1}\right)$
- ...

If we then sample "sufficiently long", X_t is drawn from a distribution that is "very close" to the posterior distribution. "Sufficiently long" and "very close" are commonly made precise by the notions of rapidly mixing Markov chains and total variation distance.

Definition 13 (Total variation distance and Mixing time)**.** *The* TOTAL VARIATION *distance between two probability distributions μ and v on $\mathcal{A}$ is defined by*

$$\|\mu - v\|_{\mathrm{TV}} \doteq 2 \sup_{A \subseteq \mathcal{A}} |\mu(A) - v(A)|.$$

It defines the distance between μ and v to be the maximum difference between the probabilities that μ and v assign to the same event.
For a Markov chain with stationary distribution π, its MIXING TIME *with respect to the total variation distance is*

$$\tau_{\mathrm{TV}}(\epsilon) \doteq \min\left\{t \mid \|q_t - \pi\|_{\mathrm{TV}} \leq \epsilon\right\}$$

where $X_t \sim q_t$. Thus, the mixing time measures the time required by a Markov chain for the distance to stationarity to be small. A Markov chain is typically said to be RAPIDLY MIXING *if*

$$\tau_{\mathrm{TV}}(\epsilon) \in \mathcal{O}(\mathrm{poly}(n, \log(1/\epsilon)))$$

How can we confirm that the stationary distribution of a Markov chain coincides with the posterior distribution? The detailed balance equation yields a very simple method.

Definition 14. *A Markov chain satisfies the* DETAILED BALANCE EQUATION *with respect to a distribution π iff*

$$\pi(x) p\left(x' \mid x\right) = \pi\left(x'\right) p\left(x \mid x'\right)$$

holds for any $x, x' \in S$. A Markov chain that satisfies the detailed balance equation with respect to π is called REVERSIBLE WITH RESPECT TO π.

Theorem 18. *Given a finite Markov chain, if the Markov chain is reversible with respect to π then π is a stationary distribution.*

That is, if we can show that the detailed balance equation holds for some distribution q, then we know that q is the stationary distribution of the Markov chain.
Next, reconsider our posterior distribution $p(x) = \frac{1}{Z} q(x)$. If we substitute the posterior for π in the detailed balance equation, we obtain

$$\frac{1}{Z} q(x) p\left(x' \mid x\right) = \frac{1}{Z} q\left(x'\right) p\left(x \mid x'\right),$$

or equivalently,

$$q(x) p\left(x' \mid x\right) = q\left(x'\right) p\left(x \mid x'\right).$$

In words, we do not need to know the true posterior p to check that the stationary distribution of our Markov chain coincides with p, it suffices to know q.

If we now suppose that we can construct a Markov chain whose stationary distribution coincides with the posterior distribution, and we will see later that this is possible, it is not apparent that this allows us to estimate expectations over the posterior distribution. Note that although constructing such a Markov chain allows us to obtain samples from the posterior distribution, they are not independent. In fact, due to the structure of a Markov chain, by design, they are strongly dependent. Thus, the law of large numbers and Hoeffding's inequality do not apply.
It turns out that there is a way to generalize the (strong) law of large numbers to Markov chains.

Theorem 19 (Ergodic theorem). *Given an ergodic Markov chain $(X_t)_{t\in\mathbb{N}_0}$ over a finite state space S with stationary distribution π and a function $f : S \to \mathbb{R}$*

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) \overset{a.s.}{\to} \sum_{x\in S} \pi(x) f(x) = \mathbb{E}_{x\sim\pi}[f(x)]$$

as $n \to \infty$ where $x_i \sim X_i \mid x_{i-1}$.

This result is the fundamental reason for why Markov chain Monte Carlo methods are possible. There are analogous results for continuous domains. Note, however, that the ergodic theorem only tells us that simulating a single Markov chain yields an unbiased estimator. It does not tell us anything about the rate of convergence and variance of such an estimator. The convergence rate depends on the mixing time of the Markov chain, which is difficult to establish in general. In practice, one observes that Markov chain Monte Carlo methods have a so-called "burn-in" time during which the distribution of the Markov chain does not yet approximate the posterior distribution well. Typically, the first t_0 samples are therefore discarded,

$$\mathbb{E}[f(X)] \approx \frac{1}{T - t_0} \sum_{t=t_0+1}^{T} f(X_t).$$

It is not clear in general how T and t_0 should be chosen such that the estimator is unbiased, rather they have to be tuned.

5.2 Elementary Sampling Methods

We will now examine methods for constructing and sampling from a Markov chain with the goal of approximating samples from the posterior distribution p. Note that in this setting the state space of the Markov chain is $\mathbb{R}^n$ and a single state at time t is described by the random vector $\mathbf{X} \doteq [X_1, \ldots, X_n]$.

5.2.1 Metropolis-Hastings Algorithm

Suppose we are given a proposal distribution $r(\boldsymbol{x}' \mid \boldsymbol{x})$ which, given we are in state x, proposes a new state $\boldsymbol{x}'$. Metropolis and Hastings showed that using the acceptance distribution $\mathrm{Bern}(\alpha(\boldsymbol{x}' \mid \boldsymbol{x}))$, where

$$\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) \doteq \min\left\{1, \frac{p(\boldsymbol{x}')\, r(\boldsymbol{x} \mid \boldsymbol{x}')}{p(\boldsymbol{x}) r(\boldsymbol{x}' \mid \boldsymbol{x})}\right\} = \min\left\{1, \frac{q(\boldsymbol{x}')\, r(\boldsymbol{x} \mid \boldsymbol{x}')}{q(\boldsymbol{x}) r(\boldsymbol{x}' \mid \boldsymbol{x})}\right\}$$

to decide whether to follow the proposal - namely, if $X_t = x$, with probability $\alpha(x' \mid x)$ we set $X_{t+1} = x'$ and with probability $1 - \alpha$ we set $X_{t+1} = x$ - yields a Markov chain with stationary distribution $p(\boldsymbol{x}) = \frac{1}{Z} q(\boldsymbol{x})$.

Metropolis-Hastings algorithm

initialize $x \in \mathbb{R}^n$
for $t = 1$ to T **do**
  sample $x' \sim r(x' \mid x)$
  sample $u \sim \mathrm{Unif}([0, 1])$
  if $u \leq \alpha(x' \mid x)$ **then** update $x \leftarrow x'$
  else update $x \leftarrow x$
end for

Theorem 20 (Metropolis-Hastings theorem). *Given an arbitrary proposal distribution r, the stationary distribution of the Markov chain simulated by the Metropolis-Hastings algorithm is $p(\boldsymbol{x}) = \frac{1}{Z} q(\boldsymbol{x})$.*

5.2.2 Gibbs Sampling

A popular example of a Metropolis-Hastings algorithm is Gibbs sampling.

Gibbs sampling

initialize $\boldsymbol{x} = [x_1, \ldots, x_n] \in \mathbb{R}^n$
for $t = 1$ to T **do**
  pick a variable i uniformly at random from $\{1, \ldots, n\}$
  set $x_{-i} \doteq [x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$
  update x_i by sampling according to the posterior distribution $p\left(x_i \mid \boldsymbol{x}_{-i}\right)$
end for

Theorem 21 (Gibbs sampling as Metropolis-Hastings)**.** *Gibbs sampling is a Metropolis-Hastings algorithm. For any fixed $i \in [n]$, it has proposal distribution*

$$r_i\left(\boldsymbol{x}' \mid \boldsymbol{x}\right) \doteq \begin{cases} p\left(x_i' \mid \boldsymbol{x}'_{-i}\right) & \boldsymbol{x}' \text{ differs from } \boldsymbol{x} \text{ only in entry } i \\ 0 & \text{otherwise} \end{cases}$$

and acceptance distribution $\alpha_i\left(x' \mid x\right) \doteq 1$.
As Gibbs sampling is a specific example of an MH-algorithm, the stationary distribution of the simulated Markov chain is $p(x)$.

Note that for the proposals of Gibbs sampling, we have

$$p\left(x_i \mid x_{-i}\right) = \frac{p\left(x_i, \boldsymbol{x}_{-i}\right)}{\sum_{x_i} p\left(x_i, \boldsymbol{x}_{-i}\right)} = \frac{q\left(x_i, \boldsymbol{x}_{-i}\right)}{\sum_{x_i} q\left(x_i, \boldsymbol{x}_{-i}\right)}.$$

Thus, re-sampling x_i only requires evaluating unnormalized joint distribution and renormalizing.

5.3 Sampling as Optimization

In this section, we discuss more advanced sampling methods. The main idea that we will study is the interpretation of sampling as an optimization problem. We will build towards an optimization view of sampling step-by-step, and first introduce what is commonly called the "energy" of a distribution.

Gibbs distributions are a special class of distributions that are widely used in machine learning, and which are characterized by an energy.

Definition 15. *Formally, a* GIBBS DISTRIBUTION*, also called a* BOLTZMANN DISTRIBUTION*, is a continuous distribution p whose* PDF *is of the form*

$$p(x) = \frac{1}{Z} \exp(-f(x)).$$

$f : \mathbb{R}^n \to \mathbb{R}$ is also called an ENERGY FUNCTION*. When the energy function f is convex, its Gibbs distribution is called* LOG-CONCAVE*.*

A useful property is that Gibbs distributions always have full support. It is often easier to reason about "energies" rather than probabilities as they are neither restricted to be non-negative nor do they have to integrate to 1. Also observe that the posterior distribution can always be interpreted as a Gibbs distribution as long as prior and likelihood have full support,

$$p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) = \frac{1}{Z} p(\boldsymbol{\theta}) p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) = \frac{1}{Z} \exp\left(-\left[-\log p(\boldsymbol{\theta}) - \log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)\right]\right).$$

Thus, defining the energy function

$$f(\boldsymbol{\theta}) \doteq -\log p(\boldsymbol{\theta}) - \log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right) = -\log p(\boldsymbol{\theta}) - \sum_{i=1}^{n} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right),$$

yields

$$p\left(\boldsymbol{\theta} \mid x_{1:n}, y_{1:n}\right) = \frac{1}{Z} \exp(-f(\boldsymbol{\theta})).$$

Using that the posterior is a Gibbs distribution, we can rewrite the acceptance distribution of Metropolis-Hastings,

$$\alpha\left(x' \mid x\right) = \min\left\{1, \frac{r\left(x \mid x'\right)}{r\left(x' \mid x\right)} \exp\left(f(x) - f\left(x'\right)\right)\right\}$$

Example 9 (Metropolis-Hastings with Gaussian proposals)**.** *Let us consider Metropolis-Hastings with the Gaussian proposal distribution,*

$$r\left(x' \mid x\right) \doteq \mathcal{N}\left(x'; x, \tau I\right)$$

Due to the symmetry of Gaussians, we have

$$\frac{r\left(x \mid x'\right)}{r\left(x' \mid \boldsymbol{x}\right)} = \frac{\mathcal{N}\left(x; x', \tau I\right)}{\mathcal{N}\left(x'; \boldsymbol{x}, \tau \boldsymbol{I}\right)} = 1.$$

Hence, the acceptance distribution is defined by

$$\alpha\left(x' \mid \boldsymbol{x}\right) = \min\left\{1, \exp\left(f(x) - f\left(\boldsymbol{x}'\right)\right)\right\}.$$

Intuitively, when a state with lower energy is proposed, that is $f\left(x'\right) \leq f(x)$, then the proposal will always be accepted. In contrast, if the energy of the proposed state is higher, the acceptance probability decreases exponentially in the difference of energies $f(\boldsymbol{x}) - f\left(\boldsymbol{x}'\right)$. Thus, Metropolis-Hastings minimizes the energy function, which corresponds to minimizing the negative loglikelihood and negative log-prior. The variance in the proposals τ helps in getting around local optima, but the search direction is uniformly random (i.e., "uninformed").

5.3.1 Langevin Dynamics

Until now, we have looked at Metropolis-Hastings algorithms with proposal distributions that do not explicitly take into account the curvature of the energy function around the current state. Langevin dynamics adapts the Gaussian proposals of the Metropolis-Hastings algorithm we have seen in example 6.27 to search the state space in an "informed" direction. The simple idea is to bias the sampling towards states with lower energy, thereby making it more likely that a proposal is accepted. A natural idea is to shift the proposal distribution perpendicularly to the gradient of the energy function. This yields the following proposal distribution,

$$r\left(\boldsymbol{x}' \mid \boldsymbol{x}\right) = \mathcal{N}\left(\boldsymbol{x}'; \boldsymbol{x} - \eta_t \boldsymbol{\nabla} f(\boldsymbol{x}), 2\eta_t \boldsymbol{I}\right)$$

The resulting variant of Metropolis-Hastings is known as the METROPOLIS ADJUSTED LANGEVIN ALGORITHM (MALA) or LANGEVIN MONTE CARLO (LMC). For log-concave distributions, the mixing time of the Markov chain underlying Langevin dynamics can be shown to be polynomial in the dimension n, so MALA efficiently converges to the stationary distribution.

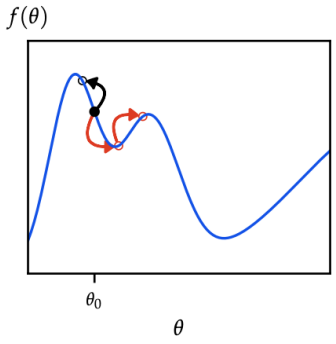


Figure 5.2: Metropolis-Hastings and Langevin dynamics minimize the energy function $f(\theta)$ shown in blue. Suppose we start at the black dot θ_0, then the black and red arrows denote possible subsequent samples. Metropolis-Hastings uses an "uninformed" search direction, whereas Langevin dynamics uses the gradient of $f(\theta)$ to make "more promising" proposals. The random proposals help get past local optima.

Note that computing the gradient of the energy function, which corresponds to computing exact gradients of the log-prior and log-likelihood, in every step can be expensive. The proposal step of MALA/LMC can be

made more efficient by approximating the gradient with an unbiased gradient estimate, leading to STOCHASTIC GRADIENT LANGEVIN DYNAMICS (SGLD), that can be interpreted as "SGD + Gaussian noise". Note that, under additional assumptions, SGLD is guaranteed to converge to the posterior distribution for decreasing learning rates $\eta_t = \Theta\left(t^{-1/3}\right)$.

Stochastic gradient Langevin dynamics, SGLD

initialize θ
for $t = 1$ to T **do**
  sample $i_1, \ldots, i_m \sim \mathrm{Unif}(\{1, \ldots, n\})$ independently
  sample $\epsilon \sim \mathcal{N}\left(\mathbf{0}, 2\eta_t I\right)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_t \left(\boldsymbol{\nabla} \log p(\boldsymbol{\theta}) + \frac{n}{m} \sum_{j=1}^{m} \boldsymbol{\nabla} \log p\left(y_{i_j} \mid \boldsymbol{x}_{i_j}, \boldsymbol{\theta}\right)\right) + \boldsymbol{\epsilon}$
end for

5.4 Summary

- Markov Chain Monte Carlo methods simulate a carefully designed Markov Chain to approximately sample from an intractable distribution
- Can be used for Bayesian learning
- For continuous distributions can make use of stochastic gradient information in the proposals
- Guaranteed, efficient convergence for log-concave densities (e.g., Bayesian logistic regression)
- In general, can guarantee convergence to the target distribution.however, for general distributions, convergence / mixing may be slow, so we have a tradeoff between accuracy and efficiency

Chapter 6

Bayesian Deep Learning

6.1 Artificial Neural Networks

So far, we've discussed effective approximate inference techniques for Bayesian linear regression and Bayesian logistic regression (linear classification)

$$p(y \mid \mathbf{x}, \theta) = \mathcal{N}\left(y; \theta^T \mathbf{x}, \sigma^2\right)$$
$$p(y \mid \mathbf{x}, \theta) = \mathrm{Ber}\left(y; \sigma\left(\theta^T \mathbf{x}\right)\right)$$

Here, likelihood functions have parameters linearly dependent on the inputs. In practice, can often get better performance by considering nonlinear dependencies. One widely used family of nonlinear functions are ARTIFICIAL DEEP NEURAL NETWORKS

$$f : \mathbb{R}^d \to \mathbb{R}^k, \quad \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}) \doteq \boldsymbol{\varphi}\left(\boldsymbol{W}_L \boldsymbol{\varphi}\left(\boldsymbol{W}_{L-1}\left(\cdots \boldsymbol{\varphi}\left(\boldsymbol{W}_1 \boldsymbol{x}\right)\right)\right)\right)$$

where $\boldsymbol{\theta} \doteq [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L]$ is a vector of weights (written as matrices $\boldsymbol{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$)[2] and $\varphi : \mathbb{R} \to \mathbb{R}$ is a component-wise nonlinear function. Thus, a deep neural network can be seen as nested ("deep") linear functions composed with nonlinearities.

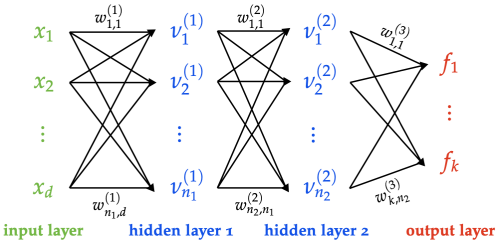


Figure 6.1: Computation graph of a neural network with two hidden layers. This type of neural network is also called a MULTILAYER PERCEPTRON.

The ACTIVATIONS of an individual (hidden) layer l of the neural network are described by

$$\boldsymbol{v}^{(l)} \doteq \boldsymbol{\varphi}\left(\boldsymbol{W}_l \boldsymbol{\nu}^{(l-1)}\right)$$

where $\boldsymbol{v}^{(0)} = \boldsymbol{x}$. The activation of the i-th node is $v_i^{(l)} = \boldsymbol{v}^{(l)}(i)$.

The non-linearity φ is called an ACTIVATION FUNCTION. The following two activation functions are particularly common.

Definition 16. *The* HYPERBOLIC TANGENT *(Tanh) is defined as*

$$\mathrm{Tanh}(z) \doteq \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \in (-1, 1).$$

Tanh is a scaled and shifted variant of the sigmoid function (4.2) *which we have previously seen in the context of logistic regression as* $\mathrm{Tanh}(z) = 2\sigma(2z) - 1$.
The RECTIFIED LINEAR UNIT *(ReLU) is defined as*

$$\mathrm{ReLU}(z) \doteq \max\{z, 0\} \in [0, \infty)$$

In particular, the ReLU activation function leads to "sparser" gradients as it selects the halfspace of inputs with positive sign. Moreover, the gradients of ReLU do not "vanish" as $z \to \pm\infty$ *which can lead to faster training.*

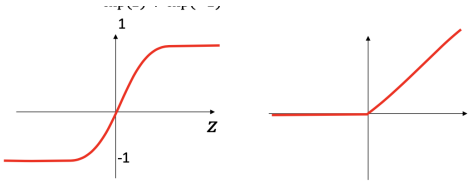


Figure 6.2: Tanh (left) and Relu (right).

Although we mainly focus on regression, neural networks can equally well be used for classification. If we want to classify inputs into c separate classes, we can simply construct a neural network with c outputs, $f = [f_1, \ldots, f_c]$, and normalize them into a probability distribution. Often, the SOFTMAX FUNCTION is used for normalization,

$$\sigma_i(f) \doteq \frac{\exp(f_i)}{\sum_{j=1}^{c} \exp(f_j)}$$

where $\sigma_i(\boldsymbol{f})$ corresponds to the probability mass of class i.

We will study neural networks under the lens of supervised learning, where we are provided some independently-sampled (noisy) data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ generated according to an unknown process $(\boldsymbol{x}, \boldsymbol{y}) \sim p$, which we wish to approximate. Upon initialization, the network does generally not approximate this process well, so a key element of deep learning is "learning" a parameterization $\boldsymbol{\theta}$ that is a good approximation. To this end, one typically considers a loss function $\ell(\boldsymbol{\theta}; \boldsymbol{y})$ which quantifies the approximation error of the network outputs $f(x; \theta)$. In the classical setting of regression, i.e., $y \in \mathbb{R}$ and $k = 1$, ℓ is often taken to be the (EMPIRICAL) MEAN SQUARED ERROR,

$$\ell_{\mathrm{mse}}(\boldsymbol{\theta}; \mathcal{D}) \doteq \frac{1}{n} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_i; \boldsymbol{\theta}\right) - y_i\right)^2.$$

As we have already seen, in the context of linear regression, minimizing mean squared error corresponds to maximum likelihood estimation under a Gaussian likelihood.
In the setting of classification where $y \in \{0, 1\}^c$ is a one-hot encoding of class membership, it is instead common to interpret the outputs of a neural network as probabilities. We denote by $q_\theta(\cdot \mid \boldsymbol{x})$ the resulting probability distribution over classes with PMF $[\sigma_1(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})), \ldots, \sigma_c(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}))]$, and aim to find $\boldsymbol{\theta}$ such that $q_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \approx p(\boldsymbol{y} \mid \boldsymbol{x})$. In this context, it is common to minimize the cross-entropy,

$$\mathrm{H}\left[p \| q_\theta\right] = \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim p}\left[-\log q_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})\right] \approx \underbrace{-\frac{1}{n} \sum_{i=1}^{n} \log q_{\boldsymbol{\theta}}\left(\boldsymbol{y}_i \mid \boldsymbol{x}_i\right)}_{\doteq \ell_{\mathrm{ce}}(\boldsymbol{\theta}; \mathcal{D})}$$

which can be understood as minimizing the surprise about the training data under the model. ℓ_{ce} is called the CROSS-ENTROPY LOSS. Disregarding the constant $1/n$, we can rewrite the cross-entropy loss as

$$\propto -\sum_{i=1}^{n} \log q_{\boldsymbol{\theta}}\left(y_i \mid x_i\right) = \ell_{\mathrm{nll}}(\boldsymbol{\theta}; \mathcal{D})$$

Recall that $\ell_{\mathrm{nll}}(\theta; \mathcal{D})$ is the negative log-likelihood of the training data, and thus, empirically minimizing cross-entropy can equivalently be interpreted as maximum likelihood estimation.

A crucial property of neural networks is that they are differentiable. That is, we can compute gradients $\nabla_{\boldsymbol{\theta}} \ell$ of f with respect to the parameterization of the model $\boldsymbol{\theta} = \boldsymbol{W}_{1:L}$ and some loss function $\ell(f; \boldsymbol{y})$. Being able to obtain these gradients efficiently allows for "learning" a particular function from data using first-order optimization methods. The algorithm for computing gradients of a neural network is called BACKPROPAGATION and is essentially a repeated application of the chain rule. Note that using the chain rule for every path through the network is computationally infeasible, as this quickly leads to a combinatorial explosion as the number of hidden layers is increased. The key insight of backpropagation is that we can use the FEED-FORWARD structure of our neural network to memorize computations of the gradient, yielding a linear time algorithm. Obtaining gradients by backpropagation is often called AUTOMATIC DIFFERENTIATION (auto-diff). Computing the exact gradient for each data point is still fairly expensive when the size of the neural network is large. Typically, stochastic gradient descent is used to obtain unbiased gradient estimates using batches of only m of the n data points, where $m \ll n$.

6.2 Bayesian Neural Networks

How can we adapt our Bayesian approach to neural networks? We use the same strategy which we already used for Bayesian logistic regression, we impose a Gaussian prior on the weights,

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0}, \sigma_p^2 \mathbf{I}\right).$$

Similarly, we can use a Gaussian likelihood to describe how well the data is described by the model f,

$$y \mid \boldsymbol{x}, \boldsymbol{\theta} \sim \mathcal{N}\left(f(\boldsymbol{x}; \boldsymbol{\theta}), \sigma_n^2\right). \tag{6.1}$$

Thus, instead of considering weights as point estimates which are learned exactly, BAYESIAN NEURAL NETWORKS learn a distribution over the weights of the network.

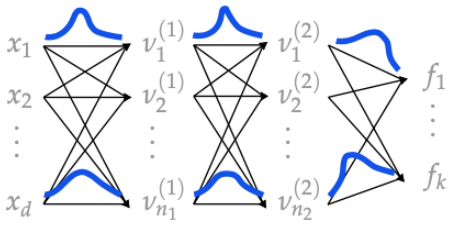


Figure 6.3: Bayesian neural networks model a distribution over the weights of a neural network.

6.2.1 Maximum A Posteriori Estimation

Before studying Bayesian inference, let us first consider MAP estimation in the context of neural networks. The MAP estimate of the weights is obtained by

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right). \tag{6.2}$$

Since we have a Gaussian likelihood (6.1), we can proceed as in (2.4) to get the MAP estimate

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma_p^2} \|\boldsymbol{\theta}\|_2^2 + \frac{1}{2\sigma_n^2} \sum_{i=1}^{n} \left(y_i - f\left(\boldsymbol{x}_i; \boldsymbol{\theta}\right)\right)^2$$

and, as in the context of Bayesian linear regression, we notice that using a Gaussian prior is equivalent to applying weight decay (namely L^2 regularization). Using gradient ascent, we obtain the following update rule,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}\left(1 - \lambda \eta_t\right) + \eta_t \sum_{i=1}^{n} \boldsymbol{\nabla} \log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right)$$

where $\lambda \doteq 1/\sigma_p^2$. The gradients of the likelihood can be obtained using automatic differentiation.

6.2.2 Heteroscedastic Noise

Equation (6.1) uses the scalar parameter σ_n^2 to model the aleatoric uncertainty (label noise), similarly to how we modeled the label noise in Bayesian linear regression and Gaussian processes. Such a noise model is called homoscedastic as the noise is assumed to be uniform across the domain. In many settings, however, the noise is inherently heteroscedastic. We already know that this means that the noise varies depending on the input and which "region" of the domain the input is from. There is a natural way of modeling heteroscedastic noise with Bayesian neural networks. We use a neural network with two outputs f_1 and f_2 and define

$$y \mid \boldsymbol{x}, \boldsymbol{\theta} \sim \mathcal{N}\left(\mu(\boldsymbol{x};\boldsymbol{\theta}), \sigma^2(\boldsymbol{x};\boldsymbol{\theta})\right) \quad \text{where } \mu(\boldsymbol{x};\boldsymbol{\theta}) \doteq f_1(\boldsymbol{x};\boldsymbol{\theta})\sigma^2(\boldsymbol{x};\boldsymbol{\theta}) \doteq \exp\left(f_2(\boldsymbol{x};\boldsymbol{\theta})\right).$$

Using this model, the likelihood term in (6.2) is

$$\begin{aligned}\log p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) &= \log \mathcal{N}\left(y_i; \mu\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right), \sigma^2\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)\right) = \log \frac{1}{\sqrt{2\pi\sigma^2\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)}} - \frac{\left(y_i - \mu\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)\right)^2}{2\sigma^2\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)} \\ &= \underbrace{\log \frac{1}{\sqrt{2\pi}}}_{\text{const}} - \frac{1}{2}\left[\log \sigma^2\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right) + \frac{\left(y_i - \mu\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)\right)^2}{\sigma^2\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)}\right].\end{aligned}$$

Hence, since we want to maximize the previous and we have a minus sign in front of the square brackets, we conclude that the model can either explain the label y_i by an accurate model $\mu\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)$ or by a large variance $\sigma^2\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)$, yet, it is penalized for choosing large variances. Intuitively, this allows to attenuate losses for some data points by attributing them to large variance (when no model reflecting all data points simultaneously can be found). This allows the model to "learn" its aleatoric uncertainty. However, recall that the MAP estimate still corresponds to a point estimate of the weights, so we forgo modeling the epistemic uncertainty.

6.3 Approximate Inference

Naturally, we want to understand the epistemic uncertainty of our model too. However, learning and inference in Bayesian neural networks are generally intractable (even when using a Gaussian prior and likelihood) when the noise is not assumed to be homoscedastic and known. Thus, we are led to the techniques of approximate inference.

6.3.1 Variational Inference

As we have already discussed, we can apply black box stochastic variational inference which, in the context of neural networks, is also known as BAYES BY BACKPROP. As variational family, we use the family of independent Gaussians which we have already encountered. Recall that the fundamental objective of variational inference is

$$\arg\min_{q\in\mathcal{Q}} \mathrm{KL}\left(q \| p\left(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n}\right)\right) = \arg\max_{q\in\mathcal{Q}} \mathbb{E}_{\boldsymbol{\theta}\sim q}\left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)\right] - \mathrm{KL}(q\|p(\cdot)).$$

The KL-divergence $\mathrm{KL}(q\|p(\cdot))$ can be expressed in closed-form for Gaussians (see (4.5)) and we can obtain unbiased gradient estimates of the expectation using the reparameterization trick (see (4.7)):

$$\mathbb{E}_{\boldsymbol{\theta}\sim q}\left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)\right] = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\log p\left(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}\right)|_{\boldsymbol{\theta}=\boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}+\boldsymbol{\mu}}\right].$$

As $\boldsymbol{\Sigma}$ is the diagonal matrix $\operatorname{diag}\left\{\sigma_1^2, \ldots, \sigma_d^2\right\}, \Sigma^{1/2} = \operatorname{diag}\left\{\sigma_1, \ldots, \sigma_d\right\}$. The gradients of the likelihood can be obtained using backpropagation.
We can now use the variational posterior q_λ to perform approximate inference,

$$\begin{aligned}p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}\right) &= \int p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}\right) d\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}\sim p(\cdot|\boldsymbol{x}_{1:n}, y_{1:n})}\left[p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right)\right] \qquad (6.3) \\ &\approx \mathbb{E}_{\theta\sim q_\lambda}\left[p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right)\right] \approx \frac{1}{m}\sum_{i=1}^{m} p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(i)}\right)\end{aligned}$$

for independent samples $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} q_\lambda$,

$$= \frac{1}{m} \sum_{i=1}^{m} \mathcal{N}\left(y^\star; \mu\left(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}\right), \sigma^2\left(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}\right)\right).$$

Intuitively, variational inference in Bayesian neural networks can be interpreted as averaging the predictions of multiple neural networks drawn according to the variational posterior q_λ and, for Gaussian likelihoods, approximate predictive distribution becomes a mixture of Gaussians.
Using the Monte Carlo samples $\boldsymbol{\theta}^{(i)}$, we can also estimate the mean of our predictions,

$$\mathbb{E}\left[y^\star \mid x^\star, x_{1:n}, y_{1:n}\right] \approx \frac{1}{m} \sum_{i=1}^{m} \mu\left(x^\star; \theta^{(i)}\right) \doteq \bar{\mu}\left(x^\star\right),$$

and the variance of our predictions,

$$\operatorname{Var}\left[y^\star \mid x^\star, x_{1:n}, y_{1:n}\right] = \mathbb{E}_\theta\left[\operatorname{Var}_{y^\star}\left[y^\star \mid x^\star, \theta\right]\right] + \operatorname{Var}_\theta\left[\mathbb{E}_{y^\star}\left[y^\star \mid x^\star, \theta\right]\right] = \mathbb{E}_\theta\left[\sigma^2\left(x^\star; \theta\right)\right] + \operatorname{Var}_\theta\left[\mu\left(x^\star; \theta\right)\right].$$

Recall from eq. (2.8) that the first term corresponds to the aleatoric uncertainty of the data and the second term corresponds to the epistemic uncertainty of the model. We can approximate them using the Monte Carlo samples $\boldsymbol{\theta}^{(i)}$,

$$\operatorname{Var}\left[\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}\right] \approx \frac{1}{m} \sum_{i=1}^{m} \sigma^2\left(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}\right) + \frac{1}{m-1} \sum_{i=1}^{m} \left(\mu\left(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}\right) - \bar{\mu}\left(\boldsymbol{x}^\star\right)\right)^2$$

using sample mean and sample variance.

6.3.2 Approximation Inference: Markov Chain Monte Carlo

As we have discussed in the previous chapter, an alternative method to approximating the full posterior distribution is to sample from it directly. By the ergodic theorem 19, we can use any of the discussed sampling strategies to obtain samples $\boldsymbol{\theta}^{(t)}$ such that (see (5.1))

$$p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}\right) \approx \frac{1}{T} \sum_{t=1}^{T} p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(t)}\right).$$

Here, we omit the offset t_0 which is commonly used to avoid the "burn-in" period for simplicity. Algorithms such as SGLD are often used as they rely only on stochastic gradients of the loss function which can be computed efficiently using automatic differentiation.
Typically, for large networks, we cannot afford to store all T samples of models. Thus, we need to summarize the iterates.
One approach is to keep track of m snapshots of weights $\left[\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(m)}\right]$ according to some schedule and use those for inference (e.g., by averaging the predictions of the corresponding neural networks). This approach of sampling a subset of some data is generally called SUBSAMPLING.
Another approach is to summarize (that is, approximate) the weights by learning a Gaussian approximation:

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \text{where} \quad \boldsymbol{\mu} \doteq \frac{1}{T} \sum_{i=1}^{T} \boldsymbol{\theta}^{(i)}, \; \boldsymbol{\Sigma} \doteq \frac{1}{T-1} \sum_{i=1}^{T} \left(\boldsymbol{\theta}^{(i)} - \boldsymbol{\mu}\right)\left(\boldsymbol{\theta}^{(i)} - \boldsymbol{\mu}\right)^\top \tag{6.4}$$

using sample means and sample (co)variances. This can be implemented efficiently using running averages of the first and second moments,

$$\boldsymbol{\mu} \leftarrow \frac{1}{T+1}(T\boldsymbol{\mu} + \boldsymbol{\theta}) \quad \text{and} \quad \boldsymbol{A} \leftarrow \frac{1}{T+1}\left(T\boldsymbol{A} + \boldsymbol{\theta}\boldsymbol{\theta}^\top\right)$$

upon observing the fresh sample $\boldsymbol{\theta}$. $\boldsymbol{\Sigma}$ can easily be calculated from these moments,

$$\boldsymbol{\Sigma} = \frac{T}{T-1}\left(\boldsymbol{A} - \boldsymbol{\mu}\boldsymbol{\mu}^\top\right)$$

To predict, we can sample weights θ from the learned Gaussian. It turns out that this approach works well even without injecting additional Gaussian noise during training, e.g., when using SGD rather than SGLD. Simply taking the mean of the iterates of SGD is called STOCHASTIC WEIGHT AVERAGING (SWA). Describing the iterates of SGD by using a Gaussian approximation, as in (6.4), is known as STOCHASTIC WEIGHT AVERAGING-GAUSSIAN (SWAG).

6.4 Specialized Inference Techniques for Bayesian deep learning

We will now discuss two approximate inference techniques that are tailored to neural networks.

6.4.1 Dropconnect/Dropout

Traditionally, DROPOUT REGULARIZATION randomly omits vertices of the computation graph during training. In contrast, DROPCONNECT REGULARIZATION randomly omits edges of the computation graph.

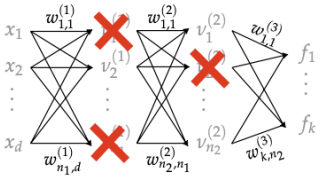


Figure 6.4: Illustration of dropout regularization. Some vertices of the computation graph are randomly omitted. In contrast, dropconnect regularization randomly omits edges of the computation graph.

The key idea that we will present here is to interpret this type of regularization as performing variational inference. In our exposition, we will focus on dropconnect, but the same approach also applies to dropout. Suppose that we omit an edge of the computation graph (i.e., set its weight to zero) with probability p. Then our variational posterior is given by

$$q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \doteq \prod_{j=1}^{d} q_j\left(\theta_j \mid \lambda_j\right)$$

where d is the number of weights of the neural network and

$$q_j\left(\theta_j \mid \lambda_j\right) \doteq p\delta_0\left(\theta_j\right) + (1-p)\delta_{\lambda_j}\left(\theta_j\right).$$

Here, δ_α is the DIRAC DELTA function with point mass at α. The variational parameters λ correspond to the "original" weights of the network. In words, the variational posterior expresses that the j-th weight has value 0 with probability p and value λ_j with probability $1-p$. This allows to interpret the result of ordinarily training a neural network with dropconnect/dropout as performing approximate Bayesian inference.

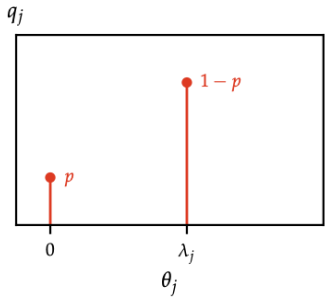


Figure 6.5: Interpretation of dropconnect regularization as variational inference. The only coordinates where the variational posterior q_j has positive probability are 0 and λ_j.

Crucially, for the interpretation of dropconnect regularization as variational inference to be valid, we also need to perform dropconnect regularization during inference,

$$p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}\right) \approx \mathbb{E}_{\boldsymbol{\theta}\sim q_\lambda}\left[p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}\right)\right] \approx \frac{1}{m}\sum_{i=1}^{m} p\left(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(i)}\right)$$

where $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} q_\lambda$ are independent samples. This coincides with our earlier discussion of variational inference for Bayesian neural networks in eq. (6.3). In words, we average the predictions of m neural networks for each of which we randomly "drop out" weights.

6.4.2 Probabilistic Ensembles

We have seen that variational inference in the context of Bayesian neural networks can be interpreted as averaging the predictions of m neural networks drawn according to the variational posterior. A natural adaptation of this idea is to immediately learn the weights of m neural networks. The idea is to randomly choose m training sets by sampling uniformly from the data with replacement. Then, using (6.2), we obtain m MAP estimates of the weights $\boldsymbol{\theta}^{(i)}$, yielding the approximation

$$p\left(y^{\star} \mid \boldsymbol{x}^{\star}, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}\right)=\mathbb{E}_{\boldsymbol{\theta} \sim p\left(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n}\right)}\left[p\left(y^{\star} \mid \boldsymbol{x}^{\star}, \boldsymbol{\theta}\right)\right] \approx \frac{1}{m} \sum_{i=1}^{m} p\left(y^{\star} \mid \boldsymbol{x}^{\star}, \boldsymbol{\theta}^{(i)}\right). \tag{6.5}$$

Here, the randomly chosen m training sets induce "diversity" of the models and, since it will be more probable to sample from denser regions of the training data and vice-versa, probabilistic ensembles give a natural way to take care of uncertainty.
Note that (6.5) is not equivalent to Monte Carlo sampling, although it looks very similar. The key difference is that this approach does not sample from the true posterior distribution p, but instead from the empirical posterior distribution $\hat{p}$ given the (re-sampled) MAP estimates. Intuitively, this can be understood as the difference between sampling from a distribution p directly (Monte Carlo sampling) versus sampling from an approximate (empirical) distribution $\hat{p}$ (corresponding to the training data), which itself is constructed from samples of the true distribution p. This approach is known as BOOTSTRAPPING.

6.5 Calibration

A key challenge of Bayesian deep learning (and also other Bayesian methods) is the calibration of models. We say that a model is well calibrated if its confidence coincides with its accuracy across many predictions. More formally, a model is PERFECTLY CALIBRATED if

$$\mathbb{P}(\hat{Y}=Y \mid \hat{P}=p)=p, \quad \forall p \in[0,1].$$

Consider a classification model that predicts that the label of a given input belongs to some class with probability 80%. If the model is well-calibrated, then the prediction is correct about 80% of the time. In other words, during calibration, we adjust the probability estimation of the model.

RELIABILITY DIAGRAMS take a frequentist perspective to estimate the calibration of a model. For simplicity, we assume a calibration problem with two classes, 1 and -1 (similarly to logistic regression). We group the predictions of a validation set into M interval bins of size $1/M$ according to the class probability predicted by the model, $\mathbb{P}\left(Y_i=1 \mid \boldsymbol{x}_i\right)$. We then compare within each bin, how often the model thought the inputs belonged to the class (confidence) with how often the inputs actually belonged to the class (frequency). Formally, we define B_m as the set of samples falling into bin m and let

$$\operatorname{freq}\left(B_m\right) \doteq \frac{1}{\left|B_m\right|} \sum_{i \in B_m}\left|\left\{Y_i=1\right\}\right|$$

be the proportion of samples in bin m that belong to class 1 and let

$$\operatorname{conf}\left(B_m\right) \doteq \frac{1}{\left|B_m\right|} \sum_{i \in B_m} \mathbb{P}\left(Y_i=1 \mid \boldsymbol{x}_i\right)$$

be the average confidence of samples belonging to class 1 within the bin m.
Thus, a model is well calibrated if $\operatorname{freq}\left(B_m\right) \approx \operatorname{conf}\left(B_m\right)$ for each bin $m \in[M]$. There are two common metrics of calibration that quantify how "close" a model is to being well calibrated.

- The EXPECTED CALIBRATION ERROR (ECE) is the average deviation of a model from perfect calibration,

$$\ell_{\mathrm{ECE}} \doteq \sum_{m=1}^{M} \frac{\left|B_m\right|}{n}\left|\operatorname{freq}\left(B_m\right)-\operatorname{conf}\left(B_m\right)\right|$$

  where n is the size of the validation set.

- The MAXIMUM CALIBRATION ERROR (MCE) is the maximum deviation of a model from perfect calibration among all bins,

$$\ell_{\mathrm{MCE}} \doteq \max_{m \in [M]} \left| \mathrm{freq}\left(B_m\right) - \mathrm{conf}\left(B_m\right) \right| .$$

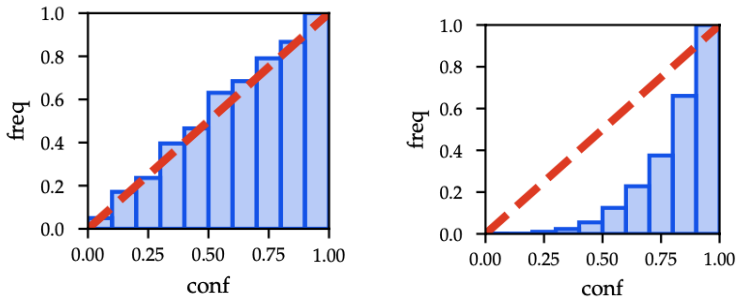


Figure 6.6: Examples of reliability diagrams with ten bins. A perfectly calibrated model approximates the diagonal dashed red line. The first reliability diagram shows a well-calibrated model. In contrast, the second reliability diagram shows an overconfident model. Indeed the bottom right half of the unit square is the region in which $\mathrm{conf} > \mathrm{freq}$.

Chapter 7

Active Learning

Probabilistic machine learning is very useful for estimating the uncertainty in our models, i.e. EPISTEMIC UNCERTAINTY (due to lack of data) and in the data, i.e. ALEATORIC UNCERTAINTY (due to measurement noise that cannot be reduced). Now we will discuss how one can use uncertainty to effectively collect more data. In other words, we want to figure out where in the domain we should sample to obtain the most useful information. Throughout most of this chapter, we focus on the most common way of quantifying "useful information", namely the expected reduction in entropy which is also called the MUTUAL INFORMATION. Mutual information helps us to quantify utility of an observation and to find a best set of observations to make.

7.1 Conditional Entropy

Recall that the entropy $\mathrm{H}[\mathbf{X}]$ of a random vector $\mathbf{X}$ can be interpreted as the average surprise when observing realizations $\boldsymbol{x} \sim \mathbf{X}$. Thus, entropy can be considered as a quantification of the uncertainty about a random vector.
A natural extension is to consider the entropy of $\mathbf{X}$ given the occurrence of an event corresponding to another random variable, e.g. $\mathbf{Y} = \boldsymbol{y}$ for a random vector $\mathbf{Y}$,

$$\mathrm{H}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}] \doteq \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x} \mid \boldsymbol{y})}[-\log p(\boldsymbol{x} \mid \boldsymbol{y})].$$

Instead of averaging over the surprise of samples from the distribution $p(\boldsymbol{x})$ (like the entropy $\mathrm{H}[\mathbf{X}]$), this quantity simply averages over the surprise of samples from the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{y})$.

Definition 17. *The* CONDITIONAL ENTROPY *of a random vector* $\mathbf{X}$ *given the random vector* $\mathbf{Y}$ *is defined as*

$$\begin{aligned}\mathrm{H}[\mathbf{X} \mid \mathbf{Y}] &\doteq \mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y})}[\mathrm{H}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}]] \\ &= \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim p(\boldsymbol{x}, \boldsymbol{y})}[-\log p(\boldsymbol{x} \mid \boldsymbol{y})].\end{aligned}$$

Intuitively, the conditional entropy of $\mathbf{X}$ given $\mathbf{Y}$ describes our average surprise about realizations of $\mathbf{X}$ given a particular realization of $\mathbf{Y}$, averaged over all such possible realizations of $\mathbf{Y}$. In other words, conditional entropy corresponds to the expected remaining uncertainty in $\mathbf{X}$ after we observe $\mathbf{Y}$. Note that, in general, $\mathrm{H}[\mathbf{X} \mid \mathbf{Y}] \neq \mathrm{H}[\mathbf{Y} \mid \mathbf{X}]$.
It is crucial to stress the difference between $\mathrm{H}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}]$ and the conditional entropy $\mathrm{H}[\mathbf{X} \mid \mathbf{Y}]$. The former simply corresponds to a Bayesian update of our uncertainty in $\mathbf{X}$ after we have observed the realization $\boldsymbol{y} \sim \mathbf{Y}$. In contrast, conditional entropy "predicts" how much uncertainty will remain about $\mathbf{X}$ (in expectation) after we "will observe" $\mathbf{Y}$.

Definition 18. *The* JOINT ENTROPY *of random vectors* $\mathbf{X}$ *and* $\mathbf{Y}$ *is the combined uncertainty about* $\mathbf{X}$ *and* $\mathbf{Y}$*:*

$$\mathrm{H}[\mathbf{X}, \mathbf{Y}] \doteq \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim p(\boldsymbol{x}, \boldsymbol{y})}[-\log p(\boldsymbol{x}, \boldsymbol{y})].$$

Observe that joint entropy is symmetric.

Proposition 4. *The following hold:*

CHAIN RULE FOR ENTROPY: $\begin{cases} \mathrm{H}[\mathbf{X}, \mathbf{Y}] = \mathrm{H}[\mathbf{Y}] + \mathrm{H}[\mathbf{X} \mid \mathbf{Y}] = \mathrm{H}[\mathbf{X}] + \mathrm{H}[\mathbf{Y} \mid \mathbf{X}] \\ \textit{(using the product rule, the definition of conditional entropy and symmetry of joint entropy)} \end{cases}$

BAYES'S RULE FOR ENTROPY: $\begin{cases} \mathrm{H}[\mathbf{X} \mid \mathbf{Y}] = \mathrm{H}[\mathbf{Y} \mid \mathbf{X}] + \mathrm{H}[\mathbf{X}] - \mathrm{H}[\mathbf{Y}] \\ \textit{(using the chain rule for entropy twice)} \end{cases}$

INFORMATION NEVER HURTS: $\mathrm{H}[\mathbf{X} \mid \mathbf{Y}] \leq \mathrm{H}[\mathbf{X}]$.

7.2 Mutual Information

We are interested in how much information we "gain" about the random vector X by choosing to observe a random vector Y. By our interpretation of conditional entropy from the previous section, this is described by the following quantity.

Definition 19. *The* MUTUAL INFORMATION *of* $\mathbf{X}$ *and* $\mathbf{Y}$*, also known as the* INFORMATION GAIN*, is defined as*

$$\mathrm{I}(\mathbf{X}; \mathbf{Y}) \doteq \mathrm{H}[\mathbf{X}] - \mathrm{H}[\mathbf{X} \mid \mathbf{Y}] = \mathrm{H}[\mathbf{X}] + \mathrm{H}[\mathbf{Y}] - \mathrm{H}[\mathbf{X}, \mathbf{Y}].$$

Thus we subtract the uncertainty left about $\mathbf{X}$ after observing $\mathbf{Y}$ from our initial uncertainty about $\mathbf{X}$. This measures the reduction in our uncertainty in $\mathbf{X}$ (as measured by entropy) upon observing $\mathbf{Y}$. Unlike conditional entropy, it follows from the definition that mutual information is symmetric, that is

$$\mathrm{I}(\mathbf{X}; \mathbf{Y}) = \mathrm{I}(\mathbf{Y}; \mathbf{X}).$$

Also, using Gibbs' inequality, it is possible to prove that $I(\mathbf{X}; \mathbf{Y}) \geq 0$ (with equality when $\mathbf{X}$ and $\mathbf{Y}$ are independent) and from this the information never hurts principle follows.

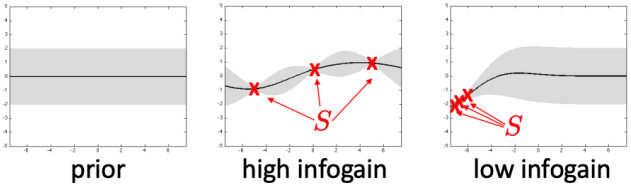


Figure 7.1: Information gain. The first graph shows the prior. The second graph shows a selection of samples with a large information gain (large reduction in uncertainty). The third graph shows a selection of samples with a small information gain (small reduction in uncertainty)

Example 10 (Mutual information of Gaussians)**.** *Given the Gaussian random vector* $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ *and the noisy observation* $\mathbf{Y} = \mathbf{X} + \boldsymbol{\epsilon}$ *where* $\boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \sigma_n^2 \boldsymbol{I}\right)$*, we want to find the information gain of* $\mathbf{X}$ *when observing* $\mathbf{Y}$*. Using our definitions from this chapter, we obtain*

$$\mathrm{I}(\mathbf{X}; \mathbf{Y}) = \mathrm{I}(\mathbf{Y}; \mathbf{X}) = \mathrm{H}[\mathbf{Y}] - \mathrm{H}[\mathbf{Y} \mid \mathbf{X}] = \mathrm{H}[\mathbf{Y}] - \mathrm{H}[\boldsymbol{\epsilon}] = \cdots = \frac{1}{2} \log \det \left(\boldsymbol{I} + \sigma_n^{-2} \boldsymbol{\Sigma}\right). \tag{7.1}$$

Intuitively, the larger the noise σ_n^2 *(less precise information about* $\mathbf{Y}$*), the smaller the information gain.*

Definition 20. *The* CONDITIONAL MUTUAL INFORMATION *of* $\mathbf{X}$ *and* $\mathbf{Y}$ *given* $\mathbf{Z}$ *is defined as*

$$\begin{aligned} \mathrm{I}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z}) &\doteq \mathrm{H}[\mathbf{X} \mid \mathbf{Z}] - \mathrm{H}[\mathbf{X} \mid \mathbf{Y}, \mathbf{Z}]. \\ &= \mathrm{H}[\mathbf{X}, \mathbf{Z}] + \mathrm{H}[\mathbf{Y}, \mathbf{Z}] - \mathrm{H}[\mathbf{Z}] - \mathrm{H}[\mathbf{X}, \mathbf{Y}, \mathbf{Z}] \\ &= \mathrm{I}(\mathbf{X}; \mathbf{Y}, \mathbf{Z}) - \mathrm{I}(\mathbf{X}; \mathbf{Z}). \end{aligned}$$

Thus, the conditional mutual information corresponds to the reduction of uncertainty in $\mathbf{X}$ when observing $\mathbf{Y}$, given we already observed $\mathbf{Z}$. It also follows that conditional mutual information is symmetric:

$$\mathrm{I}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z}) = \mathrm{I}(\mathbf{Y}; \mathbf{X} \mid \mathbf{Z})$$

7.2.1 Mutual Information as Utility Function

Given the definition of mutual information, it is natural to answer the question "where should I collect data?" by saying "wherever mutual information is maximized". More concretely, assume we are given a set $\mathcal{X}$ of possible observations of f, where y_x denotes a single such observation at $x \in \mathcal{X}$,

$$y_x \doteq f_x + \epsilon_x$$

where $f_x \doteq f(x)$ and ϵ_x is some zero-mean Gaussian noise. For a set of observations $S = \{x_1, \ldots, x_n\}$, we can write $y_S = f_S + \epsilon$ where

$$\boldsymbol{y}_S \doteq \begin{bmatrix} y_{x_1} \\ \vdots \\ y_{x_n} \end{bmatrix}, \quad f_S \doteq \begin{bmatrix} f_{x_1} \\ \vdots \\ f_{x_n} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \sigma_n^2 \boldsymbol{I}\right)$$

Note that both y_S and f_S are random vectors. Our goal is then to find a subset $S \subseteq \mathcal{X}$ of size n maximizing the information gain between our model f and $\boldsymbol{y}_S$. This yields the maximization objective,

$$I(S) \doteq \mathrm{I}\left(f_S; y_S\right) = \mathrm{H}\left[f_S\right] - \mathrm{H}\left[f_S \mid y_S\right]$$

Here, $\mathrm{H}\left[f_S\right]$ corresponds to the uncertainty about f_S before obtaining the observations y_S and $\mathrm{H}\left[f_S \mid y_S\right]$ corresponds to the uncertainty about f_S after obtaining the observations y_S.
Observe that picking a subset of points $S \subseteq \mathcal{X}$ from the domain $\mathcal{X}$ is a combinatorial problem. That is to say, we are optimizing a function over discrete sets. In general, such combinatorial optimization problems tend to be very difficult. It can be shown that maximizing mutual information is $\mathcal{NP}$-hard.

7.3 Submodularity of Mutual Information

We will look at optimizing mutual information in the following section. First, we want to introduce the notion of submodularity which is important in the analysis of discrete functions.

Definition 21. *Given a (discrete) function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$, the* MARGINAL GAIN *of $x \in \mathcal{X}$ given $A \subseteq \mathcal{X}$ is defined as*

$$\Delta_F(\boldsymbol{x} \mid A) \doteq F(A \cup \{\boldsymbol{x}\}) - F(A).$$

Intuitively, the marginal gain describes how much "adding" the additional x to A increases the value of F.

Observation 3. *When maximizing mutual information, the marginal gain corresponds to the difference between the uncertainty after observing y_A and the entropy of the noise $\mathrm{H}\left[\epsilon_x\right]$:*

$$\Delta_I(\boldsymbol{x} \mid A) = \mathrm{I}\left(f_x; y_x \mid \boldsymbol{y}_A\right) = \mathrm{H}\left[y_x \mid \boldsymbol{y}_A\right] - \mathrm{H}\left[\boldsymbol{\epsilon}_{\boldsymbol{x}}\right]$$

Altogether, the marginal gain represents the reduction in uncertainty by observing $\{x\}$.

Definition 22. *A (discrete) function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ is* SUBMODULAR *iff for any $x \in \mathcal{X}$ and any $A \subseteq B \subseteq \mathcal{X}$ it satisfies*

$$F(A \cup \{\boldsymbol{x}\}) - F(A) \geq F(B \cup \{\boldsymbol{x}\}) - F(B).$$

Equivalently, using our definition of marginal gain, we have that F is submodular iff for any $x \in \mathcal{X}$ and any $A \subseteq B \subseteq \mathcal{X}$,

$$\Delta_F(x \mid A) \geq \Delta_F(x \mid B).$$

A function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ is called MONOTONE *iff for any $A \subseteq B \subseteq \mathcal{X}$ it satisfies*

$$F(A) \leq F(B).$$

If F is also submodular, then F is called MONOTONE SUBMODULAR.

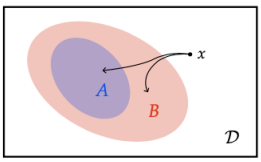


Figure 7.2: Monotone submodularity. The effect of "adding" x to the smaller set A is larger than the effect of adding x to the larger set B.

Theorem 22. *The objective, i.e. Mutual Information I, is monotone submodular.*

7.4 Maximizing Mutual Information

As we cannot efficiently pick a set $S \subseteq \mathcal{X}$ to maximize mutual information, a natural approach is to maximize mutual information greedily. That is, we pick the locations x_1 through x_n individually by greedily finding the location with the maximal mutual information. The following general result for monotone submodular function maximization shows that this greedy approach provides a good approximation.

Theorem 23 (Greedy submodular function maximization)**.** *If F is monotone submodular, then greedily maximizing F provides a constant-factor approximation where the factor is $(1 - 1/e)$:*

$$F(S_n) \geq \left(1 - \frac{1}{e}\right) \max_{\substack{S \subseteq \mathcal{X} \\ |S|=n}} F(S)$$

Observation 4. *Providing a* CONSTANT-FACTOR APPROXIMATION *means providing an efficient algorithms that can find an answer within some fixed multiplicative factor of the optimal answer.*

7.4.1 Uncertainty Sampling

When maximizing mutual information greedily, at time t, when we have already picked $S_t = \{x_1, \ldots, x_t\}$, we need to solve the following optimization problem,

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}} \Delta_I(\boldsymbol{x} \mid S_t) = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathrm{I}\left(f_{\boldsymbol{x}}; y_{\boldsymbol{x}} \mid \boldsymbol{y}_{S_t}\right).$$

Using our formula for the mutual information of conditional linear Gaussians (7.1), we can simplify to

$$= \arg\max_{x \in \mathcal{X}} \frac{1}{2} \log\left(1 + \frac{\sigma_t^2(x)}{\sigma_n^2}\right) \tag{7.2}$$

where $\sigma_t^2(x)$ is the (remaining) variance at x after observing S_t. Assuming the label noise is independent of x (i.e., homoscedastic), we get

$$= \arg\max_{x \in \mathcal{X}} \sigma_t^2(\boldsymbol{x}).$$

Therefore, if f is modeled by a Gaussian and we assume homoscedastic noise, greedily maximizing mutual information corresponds to simply picking the point x with the largest variance. so we take the point we are most uncertain about to get more information gain.

If the noise is heteroscedastic, most uncertain outcomes are not necessarily most informative. In this case, maximizing mutual information yields

$$x_{t+1} \doteq \arg\max_{x \in \mathcal{X}} \frac{1}{2} \log\left(1 + \frac{\sigma_t^2(x)}{\sigma_n^2(x)}\right) = \arg\max_{x \in \mathcal{X}} \frac{\sigma_t^2(x)}{\sigma_n^2(x)}$$

that is, we choose locations that trade large epistemic uncertainty $\sigma_t^2(x)$ with large aleatoric uncertainty $\sigma_n^2(x)$. Ideally, we find a location where the epistemic uncertainty $\sigma_t^2(x)$ is large, and the aleatoric uncertainty $\sigma_n^2(x)$ is low, which promises a significant reduction of uncertainty around this location.

7.4.2 Active learning for classification

While we focused on regression, one can apply active learning also for other settings, such as (probabilistic) classification. In this setting, for any input x, a model produces a categorical distribution over labels y_x. Here, uncertainty sampling corresponds to selecting samples that maximize the entropy of the predicted label y_x,

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{x \in \mathcal{X}} \mathrm{H}\left[y_x \mid \boldsymbol{x}_{1:t}, y_{1:t}\right].$$

The entropy of a categorical distribution is simply a finite sum of weighted surprise terms.
This approach generally leads to sampling points that are close to the decision boundary. Often, the uncertainty is mainly dominated by label noise rather than epistemic uncertainty, and hence, we do not learn much from our observations. This is a similar problem to the one we encountered with uncertainty sampling in the setting of heteroscedastic noise.
This naturally suggests distinguishing between the aleatoric and epistemic uncertainty of the model f (parameterized by $\boldsymbol{\theta}$). To this end, mutual information can be used similarly as we have done with uncertainty sampling for regression

$$\begin{aligned}
&\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathrm{I}\left(\boldsymbol{\theta}; y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}\right) = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathrm{I}\left(y_x; \boldsymbol{\theta} \mid \boldsymbol{x}_{1:t}, y_{1:t}\right) = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathrm{H}\left[y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}\right] - \mathrm{H}\left[y_{\boldsymbol{x}} \mid \boldsymbol{\theta}, \boldsymbol{x}_{1:t}, y_{1:t}\right] \\
&= \arg\max_{\boldsymbol{x} \subset \mathcal{X}} \mathrm{H}\left[y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}\right] - \mathbb{E}_{\boldsymbol{\theta} \mid x_{1:t}, y_{1:t}}\left[\mathrm{H}\left[y_x \mid \boldsymbol{\theta}, \boldsymbol{x}_{1:t}, y_{1:t}\right]\right] \\
&= \arg\max_{\boldsymbol{x} \in \mathcal{X}} \underbrace{\mathrm{H}\left[y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}\right]}_{\text{entropy of predictive posterior}} - \mathbb{E}_{\boldsymbol{\theta} \mid \boldsymbol{x}_{1:t}, y_{1:t}} \underbrace{\left[\mathrm{H}\left[y_{\boldsymbol{x}} \mid \boldsymbol{\theta}\right]\right]}_{\text{entropy of predictive posterior likelihood}}
\end{aligned}$$

The last equality follows by assuming $y_x \perp x_{1:t}, y_{1:t} \mid \boldsymbol{\theta}$ and its first term measures the entropy of the averaged prediction while its second term measures the average entropy of predictions. Thus, the first term looks for points where the average prediction is not confident. In contrast, the second term penalizes points where many of the sampled models are not confident about their prediction, and thus looks for points where the models are confident in their predictions. This identifies those points x where the models disagree about the label y_x (that is, each model is "confident" but the models predict different labels). For this reason, this approach is known as BAYESIAN ACTIVE LEARNING BY DISAGREEMENT (BALD). Also, observe that both terms in the last equality require approximate forms of the posterior distribution. The first term can be approximated by the predictive distribution of an approximated posterior which is obtained, for example, using variational inference. The nested entropy of the second term is typically easy to compute, as it corresponds to the entropy of the (discrete) likelihood distribution $p(y \mid \boldsymbol{x}, \boldsymbol{\theta})$ of the model $\boldsymbol{\theta}$. The outer expectation of the second term may be approximated using (approximate) samples from the posterior distribution obtained via variational inference, MCMC, or some other method.

7.5 Summary

- Active learning refers to a family of approaches that aim to collect data that maximally reduces uncertainty about the unknown model
- For Gaussian processes and homoscedastic noise, uncertainty sampling is equivalent to greedily maximizing mutual information
- In general, need to account for epistemic and aleatoric uncertainty (optimize their ratio / BALD)
- Due to submodularity, greedy algorithm selects near-optimal sets of observations

Chapter 8

Bayesian Optimization

Often, obtaining data is costly. In the previous chapter, this led us to investigate how we can optimally improve our understanding (i.e., reduce uncertainty) of the process we are trying to model. However, purely improving our understanding is often not good enough. In many cases, we want to use our improving understanding simultaneously to reach certain goals. One common instance of this problem is the setting of optimization.

Given some function $f^\star : \mathcal{X} \to \mathbb{R}$, suppose we want to find the

$$\arg\max_{x \in \mathcal{X}} f^\star(\boldsymbol{x}).$$

Now, contrary to classical optimization, we are interested in the setting where the function $f^\star$ is unknown to us (like a "black-box"). We are only able to obtain noisy observations of $f^\star$,

$$y_t = f^\star(x_t) + \epsilon_t.$$

Moreover, these noise-perturbed evaluations are costly to obtain. We will assume that similar alternatives yield similar results (that is, $f^\star$ is smooth), and this is commonly encoded by placing a Gaussian process prior on $f^\star$. This assumed correlation is fundamentally what will allow us to learn a model of $f^\star$ from relatively few samples.

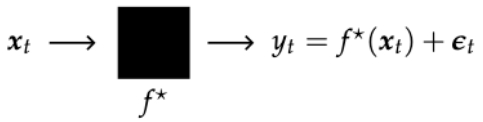


Figure 8.1: Illustration of Bayesian optimization. We pass an input x_t into the unknown function $f^\star$ to obtain noisy observations y_t.

8.1 Exploration-Exploitation Dilemma

In Bayesian optimization, we want to learn a model of $f^\star$ and use this model to optimize $f^\star$ simultaneously. These goals are somewhat contrary. Learning a model of $f^\star$ requires us to explore the input space while using the model to optimize $f^\star$ requires us to focus on the most promising well-explored areas. This trade-off is commonly known as the EXPLORATION-EXPLOITATION DILEMMA, whereby

exploration refers to choosing points that are informative with respect to the unknown function - for example, points that are far away from previously observed points (i.e., have high posterior variance),

exploitation refers to choosing promising points where we expect the function to have high values - for example, points that have a high posterior mean and a low posterior variance.

In other words, the exploration-exploitation dilemma refers to the challenge of learning enough to understand $f^\star$, but not learning too much to lose track of the objective - optimizing $f^\star$.

8.2 Online Learning and Bandits

Bayesian optimization is closely related to a form of online learning. In ONLINE LEARNING we are given a set of possible inputs $\mathcal{X}$ and an unknown function $f^\star : \mathcal{X} \to \mathbb{R}$. We are now asked to choose a sequence of inputs $x_1, \dots x_T$ online (online because data becomes available in a sequential order), and our goal is to maximize our cumulative reward $\sum_{t=1}^{T} f^\star(\boldsymbol{x}_t)$. Depending on what we observe about $f^\star$, there are different variants of online learning. Bayesian optimization is closest to the so-called (stochastic) BANDIT setting.

The MULTI-ARMED BANDITS (MAB) problem is a classical, canonical formalization of the exploration-exploitation dilemma. In the MAB problem, we are provided with k possible actions (arms) and want to maximize our reward online within the time horizon T. We do not know the reward distributions of the actions in advance, however, so we need to trade learning the reward distribution with following the most promising action. Bayesian optimization can be interpreted as a variant of the MAB problem where there can be a potentially infinite number of actions (arms), but their rewards are correlated (because of the smoothness of the Gaussian process prior). One of the key principles of the theory on multi-armed bandits and reinforcement learning is the principle of "optimism in the face of uncertainty", which suggests that it is a good guideline to explore where we can hope for the best outcomes.

The key performance metric in online learning is the regret.

Definition 23. *The* (CUMULATIVE) REGRET *for a time horizon T associated with choices $\{\boldsymbol{x}_t\}_{t=1}^{T}$ is defined as*

$$R_T \doteq \sum_{t=1}^{T} \underbrace{\left(\max_{\boldsymbol{x}} f^\star(\boldsymbol{x}) - f^\star\left(\boldsymbol{x}_t\right)\right)}_{\textit{instantaneous regret}} = T \max_{\boldsymbol{x}} f^\star(\boldsymbol{x}) - \sum_{t=1}^{T} f^\star\left(\boldsymbol{x}_t\right).$$

The regret can be interpreted as the additive loss with respect to the static optimum $\max_x f^\star(\boldsymbol{x})$. The goal is to find algorithms that achieve sublinear regret,

$$\lim_{T \to \infty} \frac{R_T}{T} = 0$$

Importantly, if we use an algorithm which explores forever, e.g., by going to a random point $\tilde{x}$ with a constant probability ϵ in each round, then the regret will grow linearly with time. This is because the instantaneous regret is at least $\epsilon \left(\max_x f^\star(x) - f^\star(\tilde{x})\right)$ and non-decreasing. Conversely, if we use an algorithm which never explores, then we might never find the static optimum, and hence, also incur constant instantaneous regret in each round, implying that regret grows linearly with time. Thus, achieving sublinear regret requires balancing exploration and exploitation.

Theorem 24. *Any algorithm achieves sublinear regret if and only if it converges to the static optimum, that is,*

$$\lim_{t \to \infty} f^\star\left(\boldsymbol{x}_t\right) = \max_{\boldsymbol{x}} f^\star(\boldsymbol{x}).$$

Typically, online learning (and Bayesian optimization) consider stationary environments, hence the comparison to the static optimum. Dynamic environments are studied in "online algorithms" and reinforcement learning. As we will later come to see in the context of reinforcement learning, operating in dynamic environments is deeply connected to a rich field of research called "control".

8.3 Acquisition Functions

It is common to use a so-called ACQUISITION FUNCTION to greedily pick the next point to sample based on the current model. Throughout our description of acquisition functions, we will focus on a setting where we model $f^\star$ using a Gaussian process which we denote by f. The methods generalize to other means of learning $f^\star$ such as Bayesian neural networks. The various acquisition functions F are used in the same way as is illustrated in the following algorithm.

Bayesian optimization (with GPs)

initialize $f \sim \mathcal{GP}(\mu_0, k_0)$
for $t = 1$ to T **do**
  choose $x_t = \arg\max_{x \in \mathcal{X}} F(\boldsymbol{x}; \mu_{t-1}, k_{t-1})$
  observe $y_t = f(x_t) + \epsilon_t$
  perform a Bayesian update to obtain μ_t and k_t
end for

One possible acquisition function is uncertainty sampling (7.2), which we discussed in the previous chapter. However, this acquisition function does not at all take into account the objective of maximizing $f^\star$ and focuses solely on exploration.

Suppose that our model f of $f^\star$ is well-calibrated, in the sense that the true function lies within its confidence bounds. Consider the best lower bound, that is, the maximum of the lower confidence bound. Now, if the true function is really contained in the confidence bounds, it must hold that the optimum is somewhere above this best lower bound. In particular, we can exclude all regions of the domain where the upper confidence bound (the optimistic estimate of the function value) is lower than the best lower bound. This is visualized in fig 8.2.

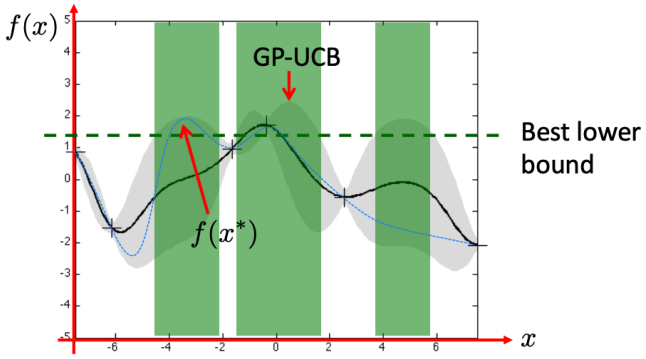


Figure 8.2: Optimism in Bayesian optimization. The unknown function is dotted blue, our model in black with shaded confidence bounds. The dotted green line denotes the maximum lower bound. We can therefore focus our exploration to the green regions where the upper confidence bound is higher than the maximum lower bound.

Therefore, we only really care how the function looks like in the regions where the upper confidence bound is larger than the best lower bound. The key idea behind the methods that we will explore is to focus exploration on these plausible maximizers.

8.3.1 Upper Confidence Bound

The principle of optimism in the face of uncertainty suggests picking the point where we can hope for the optimal outcome. In this setting, this corresponds to simply maximizing the UPPER CONFIDENCE BOUND (UCB),

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mu_t(\boldsymbol{x}) + \beta_{t+1}\sigma_t(\boldsymbol{x}),$$

where $\sigma_t(\boldsymbol{x}) \doteq \sqrt{k_t(\boldsymbol{x}, \boldsymbol{x})}$ is the standard deviation at $\boldsymbol{x}$ and β_t regulates how confident we are about our model f (i.e., the choice of confidence interval).
This acquisition function naturally trades exploitation by preferring a large posterior mean with exploration by preferring a large posterior variance. Note that if $\beta_t = 0$ then UCB is purely exploitative, whereas, if $\beta_t \to \infty$, UCB recovers uncertainty sampling (i.e., is purely explorative). UCB is an example of an optimism-based method, as it greedily picks the point where we can hope for the best outcome.

Observation 5. *GP-UCB requires solving the problem*

$$x_t = \arg\max_{x \in D} \mu_{t-1}(x) + \beta_t \sigma_{t-1}(x)$$

This is generally non-convex! In low-D, we can use Lipschitz-optimization, in high-D, can use gradient ascent (based on random initialization).

Theorem 25. *Assuming $f \sim GP$, if we choose β_t "correctly", then*

$$\frac{1}{T}\sum_{t=1}^{T}\left[f\left(x^*\right)-f\left(x_t\right)\right]=\mathcal{O}^*\left(\sqrt{\frac{\gamma_T}{T}}\right)$$

where

$$\gamma_T=\max_{|S|\leq T} I\left(f;y_S\right)$$

is the MAXIMUM INFORMATION GAIN *after T rounds γ_T.*

The maximum information gain after T rounds γ_T measures how much can be learned about $f^\star$ within T rounds and determines the regret, meaning that Regret depends on how quickly we can gain information. If information gain is sublinear in T, then we achieve sublinear regret and converge to the true optimum.

Theorem 26 (Information gain of common kernels)**.** *Due to submodularity, we have the following bounds on the information gain of common kernels:*

$$\begin{aligned}
&\textit{Linear kernel:} \quad \gamma_T=\mathcal{O}(d\log T)\\
&\textit{Gaussian kernel:} \quad \gamma_T=\mathcal{O}\left((\log T)^{d+1}\right)\\
&\textit{Matérn kernel for } v>\frac{1}{2} \quad \gamma_T=\mathcal{O}\left(T^{\frac{d}{2v+d}}(\log T)^{\frac{2v}{2v+d}}\right).
\end{aligned}$$

8.3.2 Thompson Sampling

We can also interpret the principle of optimism in the face of uncertainty in a slightly different way than we did with UCB: we can sample from the posterior distribution and then optimize it. The resulting algorithm is known as THOMPSON SAMPLING.
At time $t+1$, we sample a function $\tilde{f}_{t+1} \sim p\left(\cdot \mid \boldsymbol{x}_{1:t}, y_{1:t}\right)$ from our posterior distribution. Then, we simply maximize $\tilde{f}_{t+1}$,

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \tilde{f}_{t+1}(\boldsymbol{x}).$$

In many cases, the randomness in the realizations of $\tilde{f}_{t+1}$ is already sufficient to effectively trade exploration and exploitation. Similar regret bounds to those of UCB can also be established for Thompson sampling.

8.4 Bayesian Optimization VS Active Learning

Objective

Bayesian Optimization Primarily used for optimizing a target function that is expensive to evaluate, noisy, and lacks a closed-form expression. The goal is to find the global optimum with as few evaluations as possible.

Active Learning Generally used for selecting the most informative samples for model training. The primary focus is on improving model performance by reducing uncertainty or error in predictions.

Target Application

Bayesian Optimization Often applied in scenarios like hyperparameter tuning, experimental design, and optimizing complex simulations.

Active Learning Applied in scenarios where labeling or collecting data is expensive, and the goal is to minimize the number of labeled examples needed for model training. Commonly used in classification tasks.

Modeling Approach

Bayesian Optimization Typically involves modeling the target function with a probabilistic surrogate model, often using Gaussian processes. The model is updated as new evaluations are obtained to guide the search.

Active Learning Focuses on improving model accuracy or reducing uncertainty. Models may include support vector machines, decision trees, or neural networks. Different active learning strategies exist, such as uncertainty sampling, query-by-committee, and query-by-density.

Sample Selection

Bayesian Optimization Selects points to evaluate based on the current surrogate model's predictions and uncertainties, aiming to balance exploration and exploitation.

Active Learning Selects points for labeling based on various criteria, such as uncertainty about the model's predictions, disagreement among models, or potential for model improvement.

Optimization vs. Learning

Bayesian Optimization Primarily concerned with optimizing an objective function. The goal is to find the input that maximizes or minimizes the unknown function efficiently.

Active Learning Primarily concerned with learning a model from data. The goal is to reduce uncertainty or improve model performance by selecting the most informative samples for training.

Chapter 9

Markov Decision Processes

We will now turn to the topic of probabilistic planning. Planning deals with the problem of deciding which action an agent should play in a (stochastic) environment. A key formalism for probabilistic planning in known environments are so-called MARKOV DECISION PROCESSES. Starting from the next chapter, we will look at reinforcement learning, which extends probabilistic planning to unknown environments.
Consider the setting where we have a sequence of states $(X_t)_{t\in N_0}$ similarly to Markov chains. But now, the next state X_{t+1} of an agent does not only depend on the previous state X_t but also depends on the last action A_t of this agent.

Definition 24. *A* (FINITE) MARKOV DECISION PROCESS (MDP) *is specified by*

- *a (finite) set of* STATES $X \doteq \{1, \dots, n\}$,
- *a (finite) set of* ACTIONS $A \doteq \{1, \dots, m\}$,
- *the* TRANSITION PROBABILITIES, *also called the* DYNAMICS MODEL*:*

$$p\left(x' \mid x, a\right) \doteq \mathbb{P}\left(X_{t+1} = x' \mid X_t = x, A_t = a\right),$$

- *a* REWARD *function* $r : X \times A \to \mathbb{R}$ *which maps the current state* x *and an action* a *to some reward.*

Observation 6. *The reward function* r *induces the sequence of rewards* $(R_t)_{t\in\mathbb{N}_0}$*, where*

$$R_t \doteq r\left(X_t, A_t\right),$$

which is sometimes used in the literature instead of r*.*

We assume the dynamics model p and the reward function r to be known, that is we assume we operate in a KNOWN ENVIRONMENT. REINFORCEMENT LEARNING, that will be the subject of the last 3 chapter, deals with the problem of how to learn to act in unknown MDPs. Also, we begin by assuming that the environment is FULLY OBSERVABLE, namely we assume that our agent knows its current state.

The fundamental objective of MDPs is to learn how the agent should behave to optimize its reward. In other words, given its current state, the agent should decide optimally on the action to play. Such a decision map, whether optimal or not, is called a POLICY.

Definition 25. *A* POLICY *is a function that maps each state* $x \in X$ *to a probability distribution over the actions. That is, for any* $t > 0$*,*

$$\pi(a \mid x) \doteq \mathbb{P}\left(A_t = a \mid X_t = x\right)$$

In other words, a policy assigns to each action $a \in A$*, a probability of being played given the current state* $x \in X$*. We assume that policies are stationary, that is, do not change over time.*

Observation 7. *We will see later that in fully observable environments optimal policies are always deterministic. Thus, there is no need to consider stochastic policies in the context of MDP. For this chapter, you can think of a policy* π *simply as a deterministic mapping* $\pi : X \to A$ *from current state to played action. In the context of reinforcement learning, we will later see that randomized policies are important in trading exploration and exploitation.*

If the agent follows a fixed policy, then the evolution of the process is described fully by the Markov chain $(X_t^\pi)_{t\in\mathbb{N}_0}$ with transition probabilities given by

$$p^\pi(x' \mid x) \doteq \mathbb{P}\left(X_{t+1}^\pi = x' \mid X_t^\pi = x\right) = \sum_{a\in A} \pi(a \mid x) p(x' \mid x, a).$$

As mentioned, we want to maximize the reward. There are many models of calculating a score from the infinite sequence of rewards $(R_t)_{t\in\mathbb{N}_0}$. For the purpose of our discussion of Markov decision processes and reinforcement learning, we will focus on a very common reward called DISCOUNTED PAYOFF.

Definition 26. *The* DISCOUNTED PAYOFF*, also called* DISCOUNTED TOTAL REWARD*, from time t is defined as the random variable*

$$G_t \doteq \sum_{m=0}^{\infty} \gamma^m R_{t+m}$$

where $\gamma \in [0, 1)$ is the DISCOUNT FACTOR*.*

Observation 8. *Other well-known methods for combining rewards into a score are*

$$G_t \doteq R_t \text{ (instantaneous)}, \quad G_t \doteq \sum_{m=0}^{T-1} R_{t+m} \text{ (finite-horizon)}, \quad G_t \doteq \liminf_{T\to\infty} \frac{1}{T} \sum_{m=0}^{T} R_{t+m} \text{ (mean payoff)}.$$

The methods that we will discuss can also be analyzed using these or other alternative reward models.

We now want to understand the effect of the starting state and initial action on our optimization objective G_t. To analyze this, it is common to use the following two deterministic scalar-valued functions.

Definition 27. *The* STATE VALUE FUNCTION *measures the average discounted payoff from time t starting from state $x \in X$ and thereafter following π. It is defined as follows:*

$$v_t^\pi(x) \doteq \mathbb{E}_\pi\left[G_t \mid X_t = x\right]$$

The STATE-ACTION VALUE FUNCTION *(also called* Q-FUNCTION*) measures the average discounted payoff from time t starting from state $x \in X$, playing action $a \in A$ and thereafter following π. It is defined as follows:*

$$q_t^\pi(x, a) \doteq \mathbb{E}_\pi\left[G_t \mid X_t = x, A_t = a\right] = r(x, a) + \gamma \sum_{x'\in X} p(x' \mid x, a) \cdot v_{t+1}^\pi(x'), \tag{9.1}$$

The STATE-ACTION VALUE FUNCTION *combines the immediate return with the value of the next states.*

Observation 9. *Since following a fixed policy π induces a Markov chain $(X_t^\pi)_{t\in\mathbb{N}_0}$, we define*

$$\mathbb{E}_\pi[\cdot] \doteq \mathbb{E}_{(X_t^\pi)_{t\in\mathbb{N}_0}}[\cdot]$$

as an expectation over all possible sequences of states $(x_t)_{t\in\mathbb{N}_0}$ within this Markov chain.

Because we assumed stationary dynamics, rewards, and policies, the discounted payoff starting from a given state x will be independent of the start time t. Thus, we write $v^\pi(x) \doteq v_0^\pi(x)$ and $q^\pi(x, a) \doteq q_0^\pi(x, a)$ without loss of generality.

9.1 Bellman Expectation Equation

The BELLMAN EXPECTATION EQUATION is used to compute the value function and shows a recursive dependence of the value function on itself.

1. For deterministic policies the Bellman Expectation Equation is given by:

$$v^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'\in X} p(x' \mid x, \pi(x))\, \mathbb{E}_\pi\left[G_0 \mid X_0 = x'\right] = \tag{9.2}$$

$$= r(x, \pi(x)) + \gamma \sum_{x'\in X} p(x' \mid x, \pi(x)) \cdot v^\pi(x') = r(x, \pi(x)) + \gamma \mathbb{E}_{x'\mid x,\pi(x)}\left[v^\pi(x')\right].$$

   The intuition behind (9.2) is clear: the value of the current state corresponds to the reward from the next action plus the discounted sum of all future rewards obtained from the subsequent states. Note that the last equality in the previous follows by interpreting the sum as an expectation.

2. When the policy is stochastic, the Bellman Expectation Equation is as follows

$$v^\pi(x) = \sum_{a\in A} \pi(a \mid x) \left(r(x,a) + \gamma \sum_{x'\in X} p\left(x' \mid x, a\right) v^\pi\left(x'\right) \right) = \mathbb{E}_{a\sim\pi(x)}\left[q^\pi(x,a)\right]. \tag{9.3}$$

By recalling (9.1) and using (9.3), we obtain

$$q^\pi(x,a) = r(x,a) + \gamma \sum_{x'\in X} p\left(x' \mid x, a\right) \sum_{a'\in A} \pi\left(a' \mid x'\right) q^\pi\left(x', a'\right) = r(x,a) + \gamma \mathbb{E}_{x'|x,a} \mathbb{E}_{a'\sim\pi(x')}\left[q^\pi\left(x', a'\right)\right], \tag{9.4}$$

where the last equality follows by conditioning on the first action and interpreting the sum as an expectation.

Observation 10. *Note that it does not make sense to consider a similar recursive formula, as* (9.4)*, for the state-action value function in the setting of deterministic policies as the action played when in state $x \in X$ is uniquely determined as $\pi(x)$. In particular,*

$$v^\pi(x) = q^\pi(x, \pi(x)).$$

9.2 Fixed-point iteration to find an approximation of the value function

Bellman's expectation equation tells us that we can find the value function v^π of a fixed policy π using a system of linear equations. Indeed, using

$$\boldsymbol{v}^\pi \doteq \begin{bmatrix} v^\pi(1) \\ \vdots \\ v^\pi(n) \end{bmatrix}, \quad \boldsymbol{r}^\pi \doteq \begin{bmatrix} r(1,\pi(1)) \\ \vdots \\ r(n,\pi(n)) \end{bmatrix}, \quad \text{and} \quad \boldsymbol{P}^\pi \doteq \begin{bmatrix} p(1 \mid 1,\pi(1)) & \cdots & p(n \mid 1,\pi(1)) \\ \vdots & \ddots & \vdots \\ p(1 \mid n,\pi(n)) & \cdots & p(n \mid n,\pi(n)) \end{bmatrix}$$

the Bellman expectation equation (9.2) is equivalent to

$$\boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}^\pi \iff \quad \left(\boldsymbol{I} - \gamma \boldsymbol{P}^\pi\right) \boldsymbol{v}^\pi = \boldsymbol{r}^\pi \iff \quad \boldsymbol{v}^\pi = \left(\boldsymbol{I} - \gamma \boldsymbol{P}^\pi\right)^{-1} \boldsymbol{r}^\pi.$$

Solving this linear system of equations by performing matrix inversion, takes cubic time in the size of the state space and, to obtain an (approximate) solution of v^π, we can use that it is the unique fixed-point of the affine mapping $\boldsymbol{B}^\pi : \mathbb{R}^n \to \mathbb{R}^n$,

$$B^\pi \boldsymbol{v} \doteq \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}.$$

Theorem 27. *$\boldsymbol{v}^\pi$ is the unique fixed-point of $\boldsymbol{B}^\pi$.*

Fixed-point iteration

initialize $\boldsymbol{v}^\pi$ (e.g., as $\mathbf{0}$)
for $t = 1$ to T **do**
  $\boldsymbol{v}^\pi \leftarrow \boldsymbol{B}^\pi \boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}^\pi$
end for

It is possible to prove that that fixed-point iteration converges to $\boldsymbol{v}^\pi$ exponentially quickly.

9.3 Policy Optimization

Our goal is to find an optimal policy, namely

$$\pi^\star \doteq \arg\max_\pi \mathbb{E}_\pi\left[G_0\right].$$

If we define a partial ordering over policies by

$$\pi \geq \pi' \iff v^\pi(x) \geq v^{\pi'}(x) \quad \forall x \in X$$

then $\pi^\star$ is then simply a policy which is maximal according to this partial ordering. It follows that all optimal policies have identical value functions. Subsequently, we use $v^\star \doteq v^{\pi^\star}$ and $q^\star \doteq q^{\pi^\star}$ to denote the state value function and state-action value function arising from an optimal policy, respectively. As an optimal policy maximizes the value of each state, we have that

$$v^\star(x) = \max_\pi v^\pi(x), \quad q^\star(x,a) = \max_\pi q^\pi(x,a).$$

Simply optimizing over each policy is not a good idea as there are m^n deterministic policies in total.

9.3.1 Greedy Policies

Consider a policy that acts greedily according to the immediate return. It is fairly obvious that this policy will not perform well because the agent might never get to high-reward states. But what if someone could tell us not just the immediate return, but the long-term value of the states our agent can reach in a single step? If we knew the value of each state our agent can reach, then we can simply pick the action that maximizes the expected value. This thought experiment suggests the definition of a greedy policy with respect to a value function.

Definition 28. *The* GREEDY POLICY WITH RESPECT TO A STATE-ACTION VALUE FUNCTION q *is defined as*

$$\pi_q(x) \doteq \arg\max_{a \in A} q(x,a).$$

The GREEDY POLICY WITH RESPECT TO A STATE VALUE FUNCTION v *is defined as*

$$\pi_v(x) \doteq \arg\max_{a \in A} r(x,a) + \gamma \sum_{x' \in X} p\left(x' \mid x, a\right) \cdot v\left(x'\right).$$

Observation 11. *If q and v arise from the same policy, that is, q is defined in terms of v as per* (9.1)*, then*

$$\pi_v \equiv \pi_q.$$

This implies that we can use v and q interchangeably.

9.3.2 Bellman Optimality Equation

Observe that following the greedy policy π_v, will lead us to a new value function v^{π_v}. With respect to this value function, we can again obtain a greedy policy, of which we can then obtain a new value function. In this way, the correspondence between greedy policies and value functions induces a cyclic dependency.

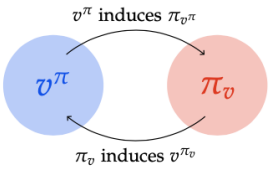


Figure 9.1: Cyclic dependency between value function and greedy policy.

Theorem 28 (Bellman's Theorem)**.** *A policy $\pi^\star$ is optimal iff it is greedy with respect to its own value function. In other words, $\pi^\star$ is optimal iff $\pi^\star(x)$ is a distribution over the set* $\arg\max_{a \in A} q^\star(x,a)$.
In particular, if for every state there is a unique action that maximizes the state-action value function, the policy $\pi^\star$ is deterministic and unique,

$$\pi^\star(x) = \arg\max_{a \in A} q^\star(x,a). \tag{9.5}$$

Bellman's Theorem confirms our intuition from the previous section that greedily following an optimal value function is itself optimal and shows that there always exists an optimal policy which is deterministic and stationary. Moreover Bellman's theorem tells us that $\pi^\star$ is a fixed-point of greedily picking the best action according to its state-action value function (see figure 9.1). The converse is also true.

Corollary 1. *The optimal value functions $v^\star$ and $q^\star$ are a fixed-point of the so-called* BELLMAN UPDATE*, i.e. of the mapping $\boldsymbol{B}^\star : \mathbb{R}^n \to \mathbb{R}^n$ given by*

$$\left(\boldsymbol{B}^\star \boldsymbol{v}\right)(x) \doteq \max_{a \in A} q(x,a)$$

where q is the state-action value function associated with the state value function v. Namely we have

$$v^\star(x) = \max_{a \in A} q^\star(x,a) = \max_{a \in A} r(x,a) + \gamma \mathbb{E}_{x'|x,a}\left[v^\star\left(x'\right)\right], \tag{9.6}$$

$$q^\star(x,a) = r(x,a) + \gamma \mathbb{E}_{x'|x,a}\left[\max_{a' \in A} q^\star\left(x',a'\right)\right]. \tag{9.7}$$

Equations (9.6) and (9.7) are also called the BELLMAN OPTIMALITY EQUATIONS. Intuitively, the Bellman optimality equations express that the value of a state under an optimal policy must equal the expected return for the best action from that state.

The two perspectives of Bellman's theorem naturally suggest two separate ways of finding the optimal policy. POLICY ITERATION uses the perspective from (9.5) of $\pi^\star$ as a fixed-point of the dependency between greedy policy and value function. In contrast, VALUE ITERATION uses the perspective from (9.6) of $v^\star$ as the fixed-point of the Bellman update.

9.3.3 Policy Iteration

Policy iteration

initialize π (arbitrarily)
repeat
  compute v^π
  compute π_{v^π}
until converged

Starting from an arbitrary initial policy, POLICY ITERATION uses the Bellman expectation equation to compute the value function of that policy and then chooses the greedy policy with respect to that value function as its next iterate.

Let π_t be the policy after t iterations, then π_t converges to the optimal policy. To prove this, one first proves that policy iteration improves policies monotonically, and then uses this fact to show that policy iteration converges.

Lemma 1 (Monotonic improvement of policy iteration)**.** *We have*

- $v^{\pi_{t+1}}(x) \geq v^{\pi_t}(x)$ *for all* $x \in X$
- $v^{\pi_{t+1}}(x) > v^{\pi_t}(x)$ *for at least one* $x \in X$*, unless* $v^{\pi_t} \equiv v^\star$*.*

Theorem 29 (Convergence of policy iteration)**.** *For finite MDP, policy iteration converges to an optimal policy.*

It can be shown that policy iteration converges to an exact solution in a polynomial number of iterations. Moreover, each iteration of policy iteration requires computing the value function, which we have seen to be of cubic complexity in the number of states.

9.3.4 Value Iteration

As already mentioned, another natural approach of finding the optimal policy is to interpret $v^\star$ as the fixed point of the Bellman update. The value iteration algorithm is shown in the following algorithm.
It is possible to prove the convergence of value iteration using the fixed point interpretation, that $\boldsymbol{B}^\star$ is a contraction and Banach's fixed-point theorem.

Theorem 30 (Convergence of value iteration)**.** *Value iteration converges to an optimal policy.*

Value iteration

initialize $v(x) \leftarrow \max_{a \in A} r(x, a)$ for each $x \in X$
for $t = 1$ to ∞ **do**
  $v(x) \leftarrow (\boldsymbol{B}^\star \boldsymbol{v})(x) = \max_{a \in A} q(x, a)$ for each $x \in X$
  if $\|v_t - v_{t-1}\|_\infty = \max_x |v_t(x) - v_{t-1}(x)| \leq \varepsilon$, **break**
end for
choose the greedy policy w.r.t. v.

Value iteration converges to an ϵ-optimal solution in a polynomial number of iterations. Unlike policy iteration, value iteration does not converge to an exact solution in general. Recalling the update rule of value iteration, its main benefit is that each iteration only requires a sum over all possible actions a in state x and a sum over all reachable states x' from x. In sparse Markov decision processes an iteration of value iteration can be performed in (virtually) constant time.

9.4 Partial Observability

So far we have focused on the fully observable setting, that is, at any time, our agent knows its current state. MDP can be extended to a partially observable setting where the agent can only access noisy observations Y_t of its state X_t.

Definition 29 (Partially observable Markov decision process, POMDP)**.** *A* PARTIALLY OBSERVABLE MARKOV DECISION PROCESS *is specified by*

- *a set of* STATES X*,*
- *a set of* ACTIONS A*,*
- TRANSITION PROBABILITIES $p(x' \mid x, a)$*,*
- *a reward function* $r : X \times A \to \mathbb{R}$*.*

Additionally, it is specified by

- *a set of* OBSERVATIONS Y
- OBSERVATION PROBABILITIES $o(y \mid x) \doteq \mathbb{P}(Y_t = y \mid X_t = x)$

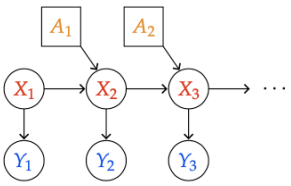


Figure 9.2: Directed graphical model of a partially observable Markov decision process with hidden states X_t, observables Y_t, and actions A_t.

Whereas MDPs are controlled Markov chains, POMDPs are controlled hidden Markov models.

Definition 30 (Hidden Markov model, HMM)**.** *A* HIDDEN MARKOV MODEL *is a Markovian process with unobservable states X_t and observations Y_t that depend on X_t in a known way. More precisely, a* HIDDEN MARKOV MODEL *is specified by*

- *a set of* STATES X*,*
- TRANSITION PROBABILITIES $p(x' \mid x) = \mathbb{P}(X_{t+1} = x' \mid X_t = x)$ *(also called* MOTION MODEL*),*

- *a* SENSOR MODEL $o(y \mid x) = \mathbb{P}(Y_t = y \mid X_t = x)$.

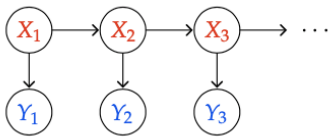


Figure 9.3: Directed graphical model of a hidden Markov model with hidden states X_t and observables Y_t.

POMDPs are a very powerful model, but very hard to solve in general. POMDPs can be reduced to a Markov decision process with an enlarged state space. The key insight is to consider an MDP whose states are the BELIEFS

$$b_t(x) \doteq \mathbb{P}(X_t = x \mid y_{1:t}, a_{1:t-1}),$$

about the current state in the POMDP. In other words, the states of the MDP are probability distributions over the states of the POMDP.

Definition 31 (Belief-state Markov decision process)**.** *Given a POMDP, the corresponding* BELIEF-STATE MARKOV DECISION PROCESS *is a MDP specified by*

- *the* BELIEF SPACE

$$\mathcal{B} \doteq \Delta^X = \Delta(\{1, \ldots, n\}) = \left\{ b : \{1, \ldots, n\} \to [0, 1], \sum_x b(x) = 1 \right\},$$

  depending on the hidden states X,

- *the set of* ACTIONS *A, same as original MDP*

- *a* TRANSITION MODEL *that consits of:*

  - *stochastic observations:* $\mathbb{P}(Y_{t+1} = y \mid b_t, a_t) = \sum_{x,x'} b_t(x) \mathbb{P}(x' \mid x, a_t)\, o(y \mid x')$,
  - *state update: given b_t,a_t,y_{t+1}, then*

$$b_{t+1}(x') = \frac{1}{Z} \sum_x b_t(x) \mathbb{P}(X_{t+1} = x' \mid X_t = x, a_t)\, o(y_{t+1} \mid x')$$

$$\text{where } Z \doteq \sum_{x \in X} o(y_{t+1} \mid x) \sum_{x' \in X} p(x \mid x', a_t)\, b_t(x').$$

- *a* REWARD FUNCTION: $r(b_t, a_t) = \sum_x b_t(x) r(x, a_t)$

In principle, we can apply arbitrary algorithms for planning in MDPs to POMDPs. Of course, the problem is that there are infinitely many beliefs, even for a finite state space X. The belief-state MDP has therefore an infinitely large belief space $\mathcal{B}$. Even when only planning over finite horizons, exponentially many beliefs can be reached. So the belief space blows-up very quickly. A key idea in approximate solutions to POMDPs is that most belief states are never reached. A common approach is to discretize the belief space by sampling or by applying a dimensionality reduction. Examples are POINT-BASED VALUE ITERATION (PBVI) and POINT-BASED POLICY ITERATION (PBPI).

Chapter 10

Tabular Reinforcement Learning

10.1 The Reinforcement Learning Problem

REINFORCEMENT LEARNING is concerned with probabilistic planning in unknown environments. This extends our study of known environments in the previous chapter. Those environments are still modeled by Markov decision processes, but in reinforcement learning, we do not know the dynamics p and rewards r in advance. Hence, reinforcement learning is at the intersection of the theories of probabilistic planning (i.e., Markov decision processes) and learning (e.g., multi-armed bandits).
We will continue to focus on the fully observed setting, where the agent knows its current state. As we have seen in the previous section, the partially observed setting corresponds to a fully observed setting with an enlarged state space.
In this chapter, we will begin by considering reinforcement learning with small state and action spaces. This setting is often called the TABULAR REINFORCEMENT LEARNING setting, as the value functions can be computed exhaustively for all states and stored in a table.
Clearly, the agent needs to trade exploring and learning about the environment with exploiting its knowledge to maximize rewards. Thus, the exploration-exploitation dilemma, which was at the core of Bayesian optimization, also plays a crucial role in reinforcement learning. In fact, Bayesian optimization can be viewed as reinforcement learning with a fixed state and a continuous action space: in each round, the agent plays an action, aiming to find the action that maximizes the reward. However, playing the same action multiple times yields the same reward, implying that we remain in a single state.
In the context of Bayesian optimization, we used "regret" as performance metric: in the jargon of planning, minimizing regret corresponds to maximizing the cumulative reward.
Another key challenge of reinforcement learning is that the observed data is dependent on the played actions. This is in contrast to the setting of supervised learning that we have been considering in earlier chapters, where the data is sampled independently.

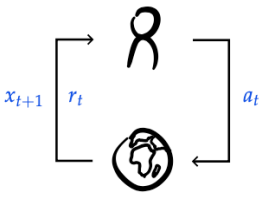


Figure 10.1: Figure 11.1: In reinforcement learning, an agent interacts with its environment in a sequence of rounds. After playing an action a_t, it observes rewards r_t and its new state x_{t+1}. The agent then uses this information to learn a model of the world.

10.1.1 Data in Reinforcement Learning

The data that the agent collects is modeled using so-called TRAJECTORIES.

Definition 32. *A* TRAJECTORY τ *is a (possibly infinite) sequence,*

$$\tau \doteq (\tau_0, \tau_1, \tau_2, \ldots)$$

of transitions,

$$\tau_i \doteq (x_i, a_i, r_i, x_{i+1}),$$

where $x_i \in X$ is the starting state, $a_i \in A$ is the played action, $r_i \in \mathbb{R}$ is the attained reward, and $x_{i+1} \in X$ is the ending state.

In the context of learning a dynamics and rewards model, x_i and a_i can be understood as inputs, and r_i and x_{i+1} can be understood as labels of a regression problem.

Crucially, the newly observed states x_{t+1} and the rewards r_t (across multiple transitions) are conditionally independent given the previous states x_t and actions a_t. This follows directly from the Markovian structure of the underlying Markov decision process. Formally, we have,

$$X_{t+1} \perp X_{t'+1} \mid X_t, X_{t'}, A_t, A_{t'}, \quad R_t \perp R_{t'} \mid X_t, X_{t'}, A_t, A_{t'}, \tag{10.1}$$

for any $t, t' \in \mathbb{N}_0$. In particular, if $x_t = x_{t'}$ and $a_t = a_{t'}$, then x_{t+1} and $x_{t'+1}$ are independent samples according to the transition model $p(X_{t+1} \mid x_t, a_t)$. Analogously, if $x_t = x_{t'}$ and $a_t = a_{t'}$, then r_t and $r_{t'}$ are independent samples of the reward model $r(x_t, a_t)$.

The collection of data is commonly classified into two settings.

Episodic setting Agent learns over multiple training EPISODES i, each resulting in a new trajectory τ_i , after which the environment resets to some initial state distribution. An example is playing a single game of chess.

Continuous setting The agent learns online, yielding a single trajectory. Especially, every action, every reward, and every state transition counts. An example is flying a drone.

10.1.2 Off-policy VS On-policy

Another important distinction in how data is collected, is the distinction between on-policy and off-policy methods. As the names suggest, ON-POLICY methods are used when the agent has control over its own actions, in other words, the agent can freely choose to follow any policy. Being able to follow a policy is helpful, for example because it allows the agent to experiment with trading exploration and exploitation.
In contrast, OFF-POLICY methods can be used even when the agent cannot freely choose its actions. Off-policy methods are therefore able to make use of purely observational data. This might be data that was collected by another agent, a fixed policy, or during a previous episode. Off-policy methods are therefore more sample-efficient than on-policy methods. This is crucial, especially in settings where con- ducting experiments (i.e., collecting new data) is expensive.

On-policy RL The agent has full control over which actions to pick, can choose how to trade exploration and exploitation and learns from the data it collects while interacting with the environment under its current policy. Example: policy gradient methods like REINFORCE.

Off-policy RL The agent does not (always) have control over actions, only gets observational data (e.g., data collected by applying a different policy) and learns from data collected by any policy, not necessarily the current policy. Example: Q-learning.

10.1.3 Model-based VS Model-free approaches

Approaches to reinforcement learning are largely categorized into two classes: MODEL-BASED APPROACHES and MODEL-FREE approaches.

Model-based approaches Model-based approaches aim to learn the underlying Markov decision process. More concretely, they estimate transition probabilities $P(x' \mid x, a)$ and the reward function $r(x, a)$, to then optimize policy based on estimated MDP.

Model-free approaches Model-free approaches Estimate the value function directly. Exaples are policy gradient methods or Actor-critic methods.

10.2 Model-based Approaches

10.2.1 Learning the Underlying Markov Decision Process

The underlying MDP is specified by its dynamics $p(x' \mid x, a)$ that correspond to the probability of entering state $x' \in X$ when playing action $a \in A$ from state $x \in X$, and its rewards $r(x, a)$ for playing action $a \in A$ in state $x \in X$. A natural first idea is to use maximum likelihood estimation to approximate these quantities.
We can think of each transition $x' \mid x, a$ as sampling from a categorical random variable of which we want to estimate the success probabilities for landing in each of the states. Therefore, the MLE of the dynamics model coincides with the sample mean,

$$\hat{p}(x' \mid x, a) = \frac{N(x' \mid x, a)}{N(a \mid x)}$$

where $N(x' \mid x, a)$ counts the number of transitions from state x to state x' when playing action a and $N(a \mid x)$ counts the number of transitions that start in state x and play action a (regardless of the next state). Similarly, for the rewards model, we obtain the following maximum likelihood estimate (i.e., sample mean),

$$\hat{r}(x, a) = \frac{1}{N(a \mid x)} \sum_{\substack{t=0 \\ x_t = x \\ a_t = a}}^{\infty} r_t.$$

Still, for the models of our environment to become accurate, our agent needs to visit each state-action pair (x, a) numerous times. Note that our estimators for dynamics and rewards are only well-defined when we visit the corresponding state-action pair at least once.

10.2.2 Balancing Exploration and Exploitation

The next natural question is how to use our current model of the environment to pick actions such that exploration and exploitation are traded effectively.
If we favour exploitation, given the estimated MDP given by $\hat{p}$ and $\hat{r}$, we can compute the optimal policy using either policy iteration or value iteration. For example, using value iteration, we can compute the optimal state-action value function $Q^\star$ within the estimated MDP, and then employ the greedy policy

$$\pi(x) = \arg\max_{a \in A} Q^\star(x, a).$$

Recall from eq. (9.5), that this corresponds to always picking the best action under the current model (that is, π is the optimal policy). But since the model is inaccurate, while potentially quickly generating some reward, we will likely get stuck in a suboptimal state.
If instead we favour exploration, we could always pick a random action and we will eventually(!) estimate the dynamics and rewards correctly, yet we will do extremely poorly in terms of maximizing rewards along the way. To trade exploration and exploitation, a natural idea is to balance these two extremes.

Arguably, the simplest idea is the following: at each time step, throw a biased coin. If this coin lands heads, we pick an action uniformly at random among all actions. If the coin lands tails, we pick the best action under our current model. This algorithm is called ϵ-GREEDY, where the probability of a coin landing heads at time t is ϵ_t.

ϵ-greedy algorithm

for $t = 1$ to ∞ **do**
  sample $u \in Unif([0, 1])$
  if $u \leq \epsilon_t$, pick action uniformly at random among all actions
  else pick best action under the current model
end for

The ϵ-greedy algorithm provides a general framework for addressing the exploration-exploitation dilemma. When the underlying MDP is learned using Monte Carlo estimation, the resulting algorithm is known as MONTE CARLO CONTROL. However, the same framework can also be used in the model-free setting where we pick the best action without estimating the full underlying MDP.

Theorem 31 (Convergence of Monte Carlo control)**.** *If the sequence ϵ_t satisfies Robbins Monro (RM) conditions, namely*

$$\epsilon_t \geq 0 \quad \forall t, \quad \sum_{t=0}^{\infty} \epsilon_t = \infty \quad \textit{and} \quad \sum_{t=0}^{\infty} \epsilon_t^2 < \infty$$

then will converge to optimal policy with probability 1

Amazingly, this simple algorithm already works quite well. Nevertheless, it can clearly be improved. The key problem of ϵ-greedy is that it explores the state space in an uninformed manner. In other words, it explores ignoring all past experience. It thus does not eliminate clearly suboptimal actions. This is a problem, especially as we typically have many state-action pairs and recalling that we have to explore each such pair many times to learn an accurate model.

10.2.3 The R-max algorithm

Recall from our discussion of multi-armed bandits that a key principle in effectively trading exploration and exploitation is "optimism in the face of uncertainty". Let us apply this principle to the reinforcement learning setting.
If $r(x, a)$ is unknown, we set $\hat{r}(x, a) = R_{\max}$, where $R_{\max}$ is the maximum reward our agent can attain during a single transition. Similarly, if $p(x' \mid x, a)$ is unknown, we set $\hat{p}(x^\star \mid x, a) = 1$, where $x^\star$ is a FAIRY-TALE STATE. The fairy-tale state corresponds to everything our agent could wish for, that is,

$$\begin{aligned} \hat{p}(x^\star \mid x^\star, a) &= 1 \qquad \forall a \in A, \\ \hat{r}(x^\star, a) &= R_{\max} \quad \forall a \in A. \end{aligned}$$

In using these optimistic estimates of p and r, we obtain an optimistic underlying Markov decision process that exhibits a bias towards exploration. In particular, the rewards attained in this MDP, are an upper bound of the true reward. The resulting algorithm is known as the $R_{\max}$ ALGORITHM.

$R_{\max}$ algorithm

input: starting state x_0 and discount factor γ
add the fairy-tale state $x^\star$ to the Markov decision process
set $\hat{r}(x, a) = R_{\max}$ for all $x \in X$ and $a \in A$
set $\hat{p}(x^\star \mid x, a) = 1$ for all $x \in X$ and $a \in A$
compute the optimal policy $\hat{\pi}$ for $\hat{r}$ and $\hat{p}$
for $t = 1$ to ∞ **do**
    execute policy $\hat{\pi}$ (for some number of steps)
    for each visited state-action pair (x, a), update $\hat{r}(x, a)$
    estimate transition probabilities $\hat{p}(x' \mid x, a)$
    after observing "enough" transitions and rewards recompute the optimal policy $\hat{\pi}$ according to current $\hat{p}$ and $\hat{r}$
end for

How many samples do we need to accurately estimate $P(x' \mid x, a)$ or $r(x, a)$? We can use HOEFFDING'S INEQUALITY.

Theorem 32. *If $Z_1, \ldots, Z_n$ are i.i.d. samples with mean μ and bounded in $[0, C]$, then*

$$\mathbb{P}\left(\left| \mu - \frac{1}{n} \sum_{i=1}^{n} Z_i \right| > \varepsilon \right) \leq 2 \exp\left(-2n\varepsilon^2 / C^2\right).$$

So, for example, if

$$\hat{r}(x, a) = \frac{1}{N(a \mid x)} \sum_{\substack{t=0 \\ x_t = x \\ a_t = a}}^{\infty} r_t.$$

and $r(x,a) \in [0, R_{\max}]$, then, by setting $C = R_{\max}$, to get

$$\mathbb{P}\left(|\hat{r}(x,a) - r(x,a)| > \varepsilon\right) \leq \delta$$

it is enough that

$$N(a \mid x) \geq \frac{R_{\max}^2}{\epsilon^2} \log \frac{1}{\delta}.$$

Lemma 2 (Exploration and exploitation of $R_{\max}$)**.** *Every T time steps, with high probability, $R_{\max}$ either obtains near-optimal reward or visits at least one unknown state-action pair.*

Theorem 33 (Convergence of $R_{\max}$)**.** *With probability at least $1-\delta$, $R_{\max}$ reaches an ϵ-optimal policy in a number of steps that is polynomial in $|X|, |A|, T, 1/\epsilon, 1/\delta$, and $R_{\max}$.*

10.2.4 Challenges of Model-based Approaches

We have seen that the $R_{\max}$ algorithm performs remarkably well in the tabular setting. However, there are important computational limitations to the model-based approaches that we discussed so far.

- First, observe that the (tabular) model-based approach requires us to store $\hat{p}\left(x' \mid x, a\right)$ and $\hat{r}(x,a)$ in a table. This table already has $\mathcal{O}\left(n^2 m\right)$ entries. Even though polynomial in the size of the state and action spaces, this quickly becomes unmanageable.
- Second, the model-based approach requires us to "solve" the learned Markov decision processes to obtain the optimal policy (using policy or value iteration). As we continue to learn over time, we need to find the optimal policy many times. $R_{\max}$ recomputes the policy after each state-action pair is observed sufficiently often, so $\mathcal{O}(\mathrm{nm})$ times.

10.3 Model-free Approaches

In the previous section, we have seen that learning and remembering the model as well as planning within the estimated model can potentially be quite expensive in the model-based approach. We therefore turn to model-free methods that estimate the value function directly. Thus, they require neither remembering the full model nor planning (i.e., policy optimization) in the underlying Markov decision process.
A significant benefit to model-based reinforcement learning is that it is inherently off-policy. That is, any trajectory regardless of the policy used to obtain it can be used to improve the model of the underlying Markov decision process. In the model-free setting, this not necessarily true. By default, estimating the value function according to the data from a trajectory, will yield an estimate of the value function corresponding to the policy that was used to sample the data.
We will start by discussing on-policy methods and later see how the value function can be estimated off-policy.

10.3.1 On-policy Value Estimation

Let us suppose, our agent follows a fixed policy π. Then, the corresponding value function v^π is given as

$$\begin{aligned} v^\pi(x) &= r(x, \pi(x)) + \gamma \sum_{x' \in X} p\left(x' \mid x, \pi(x)\right) \cdot v^\pi\left(x'\right) \\ &= \mathbb{E}_{R_0, X'}\left[R_0 + \gamma v^\pi\left(X'\right) \mid X_0 = x, A_0 = \pi(x)\right] \\ &= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \left[r^i + \gamma v^\pi(x'^{(i)}) \mid X_0 = x, A_0 = \pi(x)\right] \end{aligned} \tag{10.2}$$

where, the first equality follows from Bellman's expectation equation (9.2) and, in the last equality, $(r^i, x'^{(i)}) \sim \mathbb{P}\left(X', R \mid X_0 = x, A_0 = \pi(x)\right)$. Our first instinct might be to use a Monte Carlo estimate of this expectation, yielding

$$\approx r + \gamma v^\pi\left(x'\right) \tag{10.3}$$

where the agent observed the transition (x, a, r, x'). Note that to estimate this expectation we use a single(!) sample, unlike our previous applications of Monte Carlo sampling where we usually averaged over m samples. However, there is one significant problem in this approximation: our approximation of v^π does in turn depend on the (unknown) true value of v^π!
The key idea is to use a bootstrapping estimate of the value function instead. That is, in place of the true value function v^π, we will use a "running estimate" V^π. In other words, whenever observing a new transition, we use our previous best estimate of v^π to obtain a new estimate V^π. More generally, BOOTSTRAPPING refers to approximating a true quantity (e.g., v^π) by using an empirical quantity (e.g., V^π), which itself is constructed using samples from the true quantity that is to be approximated.
Due to its use in estimating the value function, bootstrapping is a core concept to model-free reinforcement learning. Crucially, using a bootstrapping estimate generally results in biased estimates of the value function. Moreover, due to relying on a single sample, the estimates from eq. (10.3) tend to have very large variance.
The variance of the estimate is typically reduced by mixing new estimates of the value function with previous estimates using a learning rate α_t. This yields the TEMPORAL-DIFFERENCE LEARNING ALGORITHM.

Temporal-difference (TD) learning

initialize V^π arbitrarily (e.g., as 0)
for $t = 1$ to ∞ **do**
  follow policy π to obtain the transition (x, a, r, x')
  update value estimate using bootstrapping: $V^\pi(x) \leftarrow (1 - \alpha_t)\, V^\pi(x) + \alpha_t \left(r + \gamma V^\pi\left(x'\right)\right)$
end for

Theorem 34 (Convergence of TD-learning)**.** *If $(\alpha_t)_{t\in\mathbb{N}_0}$ satisfies the RM-conditions, namely*

$$\alpha_t \geq 0 \quad \forall t, \quad \sum_{t=0}^{\infty} \alpha_t = \infty \quad \textit{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

and all state-action pairs are chosen infinitely often, then V^π converges to v^π with probability 1.

Importantly, since the transitions are obtained by following policy π, TD-learning is fundamentally on-policy. That is, for the estimates V^π to converge to the true value function v^π, the transitions that are used for the estimation must follow policy π.

10.3.2 Off-policy Value Estimation

If, instead of starting from the Bellman's expectation equaltion for state value function (9.2), we start from the Bellman's expectation equaltion for state-action value function (9.4), analogously to (10.2), we get

$$\begin{aligned} q^\pi(x,a) &= r(x,a) + \gamma \sum_{x'\in X} p\left(x' \mid x, a\right) \sum_{a'\in A} \pi\left(a' \mid x'\right) q^\pi\left(x', a'\right) \\ &= \mathbb{E}_{R_0, X'}\left[R_0 + \gamma \sum_{a'\in A} \pi\left(a' \mid X'\right) q^\pi\left(X', a'\right) \mid X_0 = x, A_0 = a\right] \\ &\approx r + \gamma \sum_{a'\in A} \pi\left(a' \mid x'\right) q^\pi\left(x', a'\right) \end{aligned}$$

where the agent observed the transition (x, a, r, x'). Note that action a need not be picked via π, so this method is off-policy! Using bootstrapping, this yields the update rule

$$Q^\pi(x,a) \leftarrow (1 - \alpha_t)\, Q^\pi(x,a) + \alpha_t \left(r + \gamma \sum_{a'\in A} \pi\left(a' \mid x'\right) Q^\pi\left(x', a'\right)\right).$$

Theorem 35. *If $(\alpha_t)_{t\in\mathbb{N}_0}$ satisfies the RM-conditions, namely*

$$\alpha_t \geq 0 \quad \forall t, \quad \sum_{t=0}^{\infty} \alpha_t = \infty \quad \textit{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

and all state-action pairs are chosen infinitely often, then Q^π converges to q^π with probability 1.

10.3.3 Q-learning: off-policy Control

It turns out that there is a way to estimate the value function of the optimal policy directly. Recall from Bellman's theorem (see (9.5)) that the optimal policy $\pi^\star$ can be characterized in terms of the optimal state-action value function $q^\star$,

$$\pi^\star(x) = \arg\max_{a \in A} q^\star(x, a).$$

$\pi^\star$ corresponds to greedily maximizing the value function. Analogously to what we did in the previous section, but using that the Q-function is a fixed-point of the Bellman update (see (9.7)) in place of Bellman's expectation equation (9.4), we obtain

$$\begin{aligned} q^\star(x,a) &= r(x,a) + \gamma \sum_{x' \in X} p\left(x' \mid x, a\right) \max_{a' \in A} q^\star\left(x', a'\right) \\ &= \mathbb{E}_{R_0, X'}\left[R_0 + \gamma \max_{a' \in A} q^\star\left(X', a'\right) \mid X_0 = x, A_0 = a\right] \\ &\approx r + \gamma \max_{a' \in A} q^\star\left(x', a'\right) \end{aligned}$$

where the agent observed the transition (x, a, r, x'). Using a bootstrapping estimate $Q^\star$ for $q^\star$, we obtain a structurally similar algorithm to TD-learning, only for estimating the optimal Q-function directly! This algorithm is known as Q-LEARNING.

Q-learning

initialize $Q^\star(x, a)$ arbitrarily (e.g., as 0)
for $t = 1$ to ∞ **do**
  observe the transition (x, a, r, x')
  update: $Q^\star(x,a) \leftarrow (1-\alpha_t)\, Q^\star(x,a) + \alpha_t \left(r + \gamma \max_{a' \in A} Q^\star\left(x', a'\right)\right)$
end for

Crucially, the Monte Carlo approximation of eq. (10.4) does not depend on the policy, thus, Q-learning is an off-policy method.

Theorem 36 (Convergence of Q-learning)**.** *If $(\alpha_t)_{t \in \mathbb{N}_0}$ satisfies the RM-conditions, namely*

$$\alpha_t \geq 0 \quad \forall t, \quad \sum_{t=0}^{\infty} \alpha_t = \infty \quad \textit{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

and all state-action pairs are chosen infinitely often, then $Q^\star$ converges to $q^\star$ with probability 1.
Moreover, with probability at least $1-\delta$, Q-learning converges to an ϵ-optimal policy in a number of steps that is polynomial in $\log|X|, \log|A|, 1/\epsilon$ and $\log 1/\delta$.

10.3.4 Optimistic Q-learning

The next natural question is how to effectively trade exploration and exploitation to both visit all state-action pairs many times, but also attain a high reward.
We will again use the principle of optimism in the face of uncertainty, which already led us to the $R_{\max}$ algorithm in the model-based setting. We will now additionally assume that the rewards are non-negative, that is, $0 \leq r(x,a) \leq R_{\max} (\forall x \in X, a \in A)$. It turns out that a similar algorithm to $R_{\max}$ also exists for (model-free) Q-learning: it is called OPTIMISTIC Q-LEARNING.
Here

$$V_{\max} \doteq \frac{R_{\max}}{1-\gamma} \geq \max_{x \in X, a \in A} q^\star(x,a)$$

is an upper bound on the discounted return and T_{init} is some initialization time. Intuitively, the initialization of $Q^\star$ corresponds to the best-case long-term reward, assuming that all individual rewards are upper bounded by $R_{\max}$. This is shown by the following lemma.

Optimistic Q-learning

initialize $Q^\star(x,a) = V_{\max} \prod_{t=1}^{T_{\text{init}}} (1-\alpha_t)^{-1}$
for $t = 1$ to ∞ **do**
  pick action $a_t = \arg\max_{a \in A} Q^\star(x,a)$ and observe the transition (x, a, r, x')
  update: $Q^\star(x,a) \leftarrow (1-\alpha_t)\, Q^\star(x,a) + \alpha_t \left(r + \gamma \max_{a' \in A} Q^\star(x', a')\right)$
end for

Lemma 3. *Denote by $Q_t^\star$, the approximation of $q^\star$ attained in the t-th iteration of optimistic Q-learning. Then, for any state-action pair (x,a) and iteration t such that $N(a \mid x) \leq T_{\text{init}}$, ($N(a \mid x)$ is the number of times action a is performed in state x),*

$$Q_t^\star(x,a) \geq V_{\max} \geq q^\star(x,a).$$

Now, if T_{init} is chosen large enough, it can be shown that optimistic Q-learning converges quickly to an optimal policy.

Theorem 37 (Convergence of optimistic Q-learning)**.** *With probability at least $1-\delta$, optimistic Q-learning obtains an ϵ-optimal policy after a number of steps that is polynomial in $|X|$, $|A|, 1/\epsilon, \log 1/\delta$, and $R_{\max}$ where the initialization time T_{init} is upper bounded by a polynomial in the same coefficients.*

Note that for Q-learning, we still need to store $Q^\star(x,a)$ for any state-action pair in memory. Thus, Q-learning requires $\mathcal{O}(\text{nm})$ memory. During each transition, we need to compute

$$\max_{a \in A} Q^\star(x', a')$$

once. If we run Q-learning for T iterations, this yields a time complexity of $\mathcal{O}(Tm)$. Crucially, for sparse Markov decision processes where, in most states, only few actions are permitted, each iteration of Q-learning can be performed in (virtually) constant time. This is a big improvement of the quadratic (in the number of states) performance of the model-based $R_{\max}$ algorithm.

10.3.5 Challenges of Model-free Approaches

We have seen that both the model-based $R_{\max}$ algorithm and the model-free Q-learning take time polynomial in the number of states $|X|$ and the number of actions $|A|$ to converge. While this is acceptable in small grid worlds, this is completely unacceptable for large state and action spaces.
Often, domains are continuous, for example when modeling beliefs about states in a partially observable environment. Also, in many structured domains (e.g., chess or multiagent planning), the size of the state and action space is exponential in the size of the input.
So the idea is to learn approximations of the value or action-value functions with a low-dimensional parametrization. In the final two chapters, we will therefore explore how model-free and model-based methods can be used (approximately) in such large domains.

Chapter 11

Model-free Reinforcement Learning

In the previous chapter, we have seen methods for tabular settings. Our goal now is to extend the model-free methods like TD-learning and Q-learning to large state-action spaces $\mathcal{X}$ and $\mathcal{A}$. We have seen that a crucial bottleneck of these methods is the parameterization of the value function. If we want to store the value function in a table, we need at least $\mathcal{O}(|\mathcal{X}|)$ space. If we learn the Q function, we even need $\mathcal{O}(|\mathcal{X}| \cdot |\mathcal{A}|)$ space. Also, for large state-action spaces, the time required to compute the value function for every state-action pair exactly will grow polynomially in the size of the state-action space. Hence, a natural idea is to learn approximations of these functions with a low-dimensional parameterization.

11.1 Tabular Reinforcement Learning as Optimization

To begin with, let us reinterpret the model-free methods from the previous section, TD-learning and Q-learning, as solving an optimization problem, where each iteration corresponds to a single gradient update. We will focus on TD-learning here, but the same interpretation applies to Q-learning. Recall the update rule of TD-learning

$$V^\pi(x) \leftarrow (1-\alpha_t)\, V^\pi(x) + \alpha_t \left(r + \gamma V^\pi\left(x'\right)\right).$$

This looks just like the update rule of an optimization algorithm! We can parameterize our estimates V^π with parameters θ that are updated according to the gradient of some loss function, assuming fixed bootstrapping estimates. In particular, in the tabular setting (i.e., over a finite domain), we can parameterize the value function exactly by learning a separate parameter for each state,

$$\boldsymbol{\theta} \doteq [\boldsymbol{\theta}(1), \ldots, \boldsymbol{\theta}(n)], \quad V^\pi(x; \boldsymbol{\theta}) \doteq \boldsymbol{\theta}(x)$$

To re-derive the above update rule as a gradient update, let us consider the following loss function,

$$\bar{\ell}(\boldsymbol{\theta}; x, r) \doteq \frac{1}{2}\left(v^\pi(x) - \boldsymbol{\theta}(x)\right)^2 = \frac{1}{2}\left(r + \gamma \mathbb{E}_{x'|x,\pi(x)}\left[v^\pi\left(x'\right)\right] - \boldsymbol{\theta}(x)\right)^2$$

Note that this loss corresponds to a standard squared loss of the difference between the parameter $\boldsymbol{\theta}(x)$ and the label $v^\pi(x)$ we want to learn. We can now find the gradient of this loss. Elementary calculations yield,

$$\nabla_{\boldsymbol{\theta}(x)} \bar{\ell}(\boldsymbol{\theta}; x, r) = \boldsymbol{\theta}(x) - \left(r + \gamma \mathbb{E}_{x'|x,\pi(x)}\left[v^\pi\left(x'\right)\right]\right).$$

Now, we cannot compute this derivative because we cannot compute the expectation. Firstly, the expectation is over the true value function which is unknown to us. Secondly, the expectation is over the transition model which we are trying to avoid in model-free methods.
To resolve the first issue, analogously to TD-learning, instead of learning the true value function v^π which is unknown, we learn the bootstrapping estimate V^π. To resolve the second issue, analogously to the introduction of TD-learning in the previous chapter, we will use a Monte Carlo estimate using a single sample.
Using the aforementioned shortcuts, let us define the loss ℓ after observing the single transition (x, a, r, x'),

$$\ell\left(\boldsymbol{\theta}; x, r, x'\right) \doteq \frac{1}{2}\left(r + \gamma V^\pi(x'; \theta_{\text{old}}) - V^\pi(x; \theta)\right)^2 = \frac{1}{2}\left(r + \gamma \theta_{\text{old}}\left(x'\right) - \theta(x)\right)^2.$$

The gradient of this loss with respect to $\theta(x)$ is

$$\delta_{\mathrm{TD}} \doteq \nabla_{\boldsymbol{\theta}(x)} \ell\left(\boldsymbol{\theta}; x, r, x'\right) = \boldsymbol{\theta}(x) - \left(r + \gamma \boldsymbol{\theta}^{\text{old}}\ \left(x'\right)\right).$$

This error term is also called temporal-difference (TD) error and we can use to perform stochastic gradient descent to get the following update rule

$$V^{\pi}(x;\boldsymbol{\theta}) = \boldsymbol{\theta}(x) \leftarrow \boldsymbol{\theta}(x) - \alpha_t \delta_{\mathrm{TD}} = (1-\alpha_t)\,\boldsymbol{\theta}(x) + \alpha_t \left(r + \gamma \boldsymbol{\theta}^{\text{old}}\ \left(x'\right)\right) = (1-\alpha_t)\, V^{\pi}(x;\boldsymbol{\theta}) + \alpha_t \left(r + \gamma V^{\pi}\left(x'; \boldsymbol{\theta}^{\text{old}}\right)\right).$$

Observe that this gradient update coincides with the update rule of TD-learning. Therefore, TD-learning is essentially performing stochastic gradient descent with respect to the loss ℓ. Crucially, TD-learning performs stochastic gradient descent with respect to the bootstrapping estimate of the value function V^{π} and not the true value function v^{π}! Stochastic gradient descent with a bootstrapping estimate is also called STOCHASTIC SEMI-GRADIENT DESCENT.

11.2 Value Function Approximation

To scale to large state spaces, it is natural to approximate the value function using a parameterized model, $V(\boldsymbol{x};\boldsymbol{\theta})$ or $Q(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})$. You may think of this as a regression problem where we map state(-action) pairs to a real number. A straightforward approach is to use a linear function approximation with the hand-designed feature map ϕ,

$$Q(x,a;\theta) \doteq \theta^{\top}\phi(x,a).$$

A common alternative is to use a deep neural network to learn these features instead. Doing so is also known as DEEP REINFORCEMENT LEARNING. Note that often non-Bayesian neural networks (i.e., point estimates of the weights) are used.

We will now apply the derivation from the previous section directly to Q-learning. For Q-learning, after observing the transition (x,a,r,x'), the loss function is given as

$$\ell\left(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}'\right) \doteq \frac{1}{2}\left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^{\star}\left(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}\right) - Q^{\star}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right)^2 \tag{11.1}$$

Here, we simply use Bellman's optimality equation (9.7) to estimate $q^{\star}(\boldsymbol{x},\boldsymbol{a})$, instead of the estimation of $v^{\pi}(\boldsymbol{x})$ using Bellman's expectation equation for TD-learning. The difference between the current approximation and the optimization target,

$$\delta_{\mathrm{B}} \doteq r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^{\star}\left(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}\right) - Q^{\star}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}),$$

is called the Bellman error. Analogously to TD-learning, we obtain the gradient update

$$\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha_t \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}'\right) \\
&= \boldsymbol{\theta} - \alpha_t \boldsymbol{\nabla}_{\boldsymbol{\theta}} \frac{1}{2}\left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^{\star}\left(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}\right) - Q^{\star}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right)^2 \\
&= \boldsymbol{\theta} + \alpha_t \delta_{\mathrm{B}} \nabla_{\boldsymbol{\theta}} Q^{\star}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}).
\end{aligned}$$

When using a neural network to learn $Q^{\star}$, we can use automatic differentiation to obtain unbiased gradient estimates. In the case of linear function approximation, we can compute the gradient exactly,

$$= \boldsymbol{\theta} + \alpha_t \delta_{\mathrm{B}} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \boldsymbol{\theta}^{\top} \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{\theta} + \alpha_t \delta_{\mathrm{B}} \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{a}).$$

So we have a straight forward generalization of tabular Q learning to function approximation as an online algorithm:

Vanilla Q-learning with function approximation

while not converged **do**
  in state x, pick action a
  observe x', reward r
  update: $\theta \leftarrow \theta - \alpha_t \delta \nabla_\theta Q^\star(x, a; \theta)$ where $\delta := Q^\star(x, a; \theta) - r - \gamma \max_{a'} Q^\star\left(x', a'; \theta^{\text{old}}\right)$
end while

11.2.1 DQN and DDQN

The previous vanilla stochastic semi-gradient descent is very slow. There are mainly two problems:

The optimization targets are not stable : The bootstrapping estimate changes after each iteration. As we are trying to learn an approximate value function that depends on the bootstrapping estimate, this means that the optimization target is "moving" between iterations. In practice, moving targets lead to stability issues.

Maximization Bias : The term $\max_{a' \in \mathcal{A}} Q^\star\left(x', a'; \theta^{\text{old}}\right)$ maximizes a noisy estimate of $q^\star$, which leads to a biased estimate of $\max q^\star$.

The NEURAL FITTED Q-ITERATION or DEEP Q-NETWORKS (DQN) aims to stabilize the optimization targets. DQN updates the neural network used for the approximate bootstrapping estimate infrequently to maintain a constant optimization target across multiple episodes. This is accomplished by cloning the neural network and maintain one changing neural network ("online network") for the most recent estimate of the Q-function which is parameterized by $\boldsymbol{\theta}$, and one fixed neural network ("target network") used as the target which is parameterized by θ^{old} and which is updated infrequently.
This can be implemented by maintaining a data set $\mathcal{D}$ of observed transitions, the so-called REPLAY BUFFER, and then every once in a while (e.g., once $|\mathcal{D}|$ is large enough) solving a regression problem, where the labels are determined by the target network. This yields a loss term where the target is fixed across all transitions in the replay buffer $\mathcal{D}$,

$$\ell_{\text{DQN}}(\boldsymbol{\theta}; \mathcal{D}) \doteq \frac{1}{2} \sum_{(x,a,r,\boldsymbol{x}') \in \mathcal{D}} \left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star\left(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}\right) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \right)^2$$

(compare to the Q-learning loss (11.1)). The loss can also be interpreted as performing regular Q-learning with the modification that the target network θ^{old} is not updated to $\boldsymbol{\theta}$ after every observed transition, but instead only after observing $|\mathcal{D}|$-many transitions. This technique is known as EXPERIENCE REPLAY.

DOUBLE DQN (DDQN) is an algorithm that addresses this maximization bias. Instead of picking the optimal action with respect to the old network, it picks the optimal action with respect to the new network,

$$\ell_{\text{DDQN}}(\boldsymbol{\theta}; \mathcal{D}) \doteq \frac{1}{2} \sum_{(\boldsymbol{x},\boldsymbol{a},r,\boldsymbol{x}') \in \mathcal{D}} \left(r + \gamma Q^\star\left(\boldsymbol{x}', \boldsymbol{a}^\star\left(\boldsymbol{x}'; \boldsymbol{\theta}\right); \boldsymbol{\theta}^{\text{old}}\right) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \right)^2$$
$$\text{where} \quad \boldsymbol{a}^\star\left(\boldsymbol{x}'; \boldsymbol{\theta}\right) \doteq \underset{\boldsymbol{a}' \in \mathcal{A}}{\arg\max} Q^\star\left(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}\right).$$

Note that, while differentiating, $\boldsymbol{a}^\star\left(\boldsymbol{x}'; \boldsymbol{\theta}\right)$ is treated as constant with respect to $\boldsymbol{\theta}$ and that $\boldsymbol{\theta}^{\text{old}}$ is updated to θ after observing $|\mathcal{D}|$-many transitions.

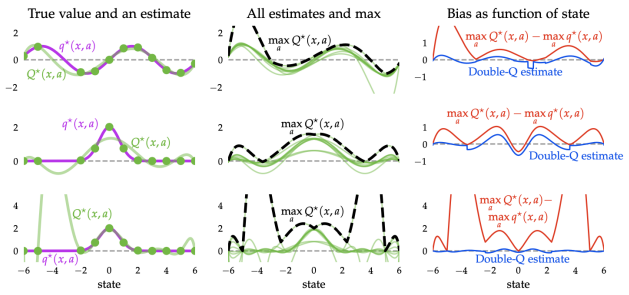


Figure 11.1: Illustration of overestimation during learning.

11.3 Policy Approximation

Q-learning defines a policy implicitly by

$$\pi^\star(x) \doteq \arg\max_{a \in \mathcal{A}} Q^\star(x, a).$$

Q-learning also maximizes over the set of all actions in its update step while learning the Q-function. This is intractable for large and, in particular, continuous action spaces. A natural idea to escape this limitation is to immediately learn an approximate parameterized policy,

$$\boldsymbol{\pi}^\star(x) \approx \boldsymbol{\pi}(x; \varphi) \doteq \boldsymbol{\pi}_\varphi(x). \tag{11.2}$$

Methods that find an approximate policy are also called POLICY SEARCH METHODS or POLICY GRADIENT METHODS. Note that the learning problem (11.2) can be interpreted as a classification problem when we have a finite set of actions, otherwise it can be interpreted as a regression problem.
Whereas with Q-learning, exploration can be encouraged by using an ϵ-greedy policy or optimistic Q-learning, policy gradient methods fundamentally rely on randomized policies for exploration.

Observation 12. *We refer to deterministic policies $\boldsymbol{\pi}$ in bold, as they can be interpreted as vector-valued functions from $\mathcal{X}$ to $\mathcal{A}$. We still refer to randomized policies by π, as for each state $x \in \mathcal{X}$ they are represented as a PDF over actions $\mathcal{A}$. In particular, we denote by $\pi_\varphi(\boldsymbol{a} \mid \boldsymbol{x})$ the probability of playing action $\boldsymbol{a}$ when in state $\boldsymbol{x}$ according to π_φ.*

11.3.1 Estimating Policy Values

We will begin by attributing a "value" to a policy. Recall the definition of the discounted payoff G_t from time t, which we are aiming to maximize,

$$G_t = \sum_{m=0}^{\infty} \gamma^m R_{t+m}.$$

We define $G_{t:T}$ to be the BOUNDED DISCOUNTED PAYOFF until time T,

$$G_{t:T} \doteq \sum_{m=0}^{T-1-t} \gamma^m R_{t+m}. \tag{11.3}$$

Based on these two random variables, we define the policy value function.

Definition 33. *The* POLICY VALUE FUNCTION*,*

$$j(\pi) \doteq \mathbb{E}_\pi [G_0] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \right],$$

measures the expected discounted payoff of policy π. We also define the bounded variant,

$$j_T(\pi) \doteq \mathbb{E}_\pi [G_{0:T}] = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t R_t \right].$$

For simplicity, we will abbreviate $j(\varphi) \doteq j(\pi_\varphi)$.

Naturally, we want to maximize $j(\varphi)$. That is, we want to solve

$$\varphi^\star \doteq \arg\max_{\varphi} j(\varphi)$$

which is a non-convex optimization problem. To deal with this, we will use a Monte Carlo estimate.
Recall that a fixed φ induces a unique Markov chain, which can be simulated. In the episodic setting, each episode, also called ROLLOUT, of length T yields an independently sampled trajectory,

$$\tau^{(i)} \doteq \left(\left(x_0^{(i)}, a_0^{(i)}, r_0^{(i)}, x_1^{(i)}\right), \left(x_1^{(i)}, a_1^{(i)}, r_1^{(i)}, x_2^{(i)}\right), \ldots\right)$$

Simulating m rollouts yields the samples,

$$\tau^{(1)}, \ldots, \tau^{(m)} \sim \Pi_{\boldsymbol{\varphi}}$$

where Π_φ is the distribution of all trajectories (i.e., rollouts) of the Markov chain induced by policy π_φ. We denote the (bounded) discounted payoff of the i-th rollout by

$$g_{0:T}^{(i)} \doteq \sum_{t=0}^{T-1} \gamma^t r_t^{(i)}$$

where $r_t^{(i)}$ is the reward at time t of the i-th rollout. Using a Monte Carlo approximation, we can then estimate $j_T(\boldsymbol{\varphi})$:

$$j(\boldsymbol{\varphi}) \approx j_T(\boldsymbol{\varphi}) \approx \frac{1}{m} \sum_{i=1}^{m} g_{0:T}^{(i)}.$$

11.3.2 Policy Gradient

Let us now formally define the distribution over trajectories Π_φ that we introduced in the previous section. We can specify the probability of a specific trajectory τ under a policy π_φ by

$$\Pi_{\boldsymbol{\varphi}}(\tau) = p(\boldsymbol{x}_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t)\, p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t).$$

For optimizing $j(\varphi)$ we need to obtain unbiased estimates of its gradient,

$$\nabla_\varphi j(\varphi) \approx \nabla_\varphi j_T(\varphi) = \nabla_\varphi \mathbb{E}_{\tau \sim \Pi_\varphi}[G_0]$$

Note that the expectation integrates over the measure Π_φ, which depends on the parameter φ. Thus, we cannot move the gradient operator inside the expectation. This should remind of the reparameterization trick that we used to solve a similar gradient in the context of variational inference. In this context, however, we cannot apply the reparameterization trick, because the distribution Π_φ is generally not reparameterizable. Fortunately, there is another way of estimating this gradient, the SCORE FUNCTION TRICK.

Theorem 38 (Score gradient estimator)**.** *Under some regularity assumptions, we have*

$$\nabla_\varphi \mathbb{E}_{\tau \sim \Pi_\varphi}[G_0] = \mathbb{E}_{\tau \sim \Pi_\varphi}[G_0 \nabla_\varphi \log \Pi_\varphi(\tau)]. \tag{11.4}$$

This estimator of the gradient is called the SCORE GRADIENT ESTIMATOR.

Intuitively, maximizing $j(\varphi)$ increases the probability of policies with high returns and decreases the probability of policies with low returns.
To use the score gradient estimator for estimating the gradient, we need to compute $\nabla_\varphi \log \Pi_\varphi(\tau)$.

$$\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) = \nabla_{\boldsymbol{\varphi}} \left(\log p(x_0) + \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) + \sum_{t=0}^{T-1} \log p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t) \right)$$

$$= \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log p(\boldsymbol{x}_0) + \sum_{t=0}^{T-1} \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) + \sum_{t=0}^{T-1} \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t) = \sum_{t=0}^{T-1} \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t)$$

The expectation of the score gradient estimator (11.4) can be approximated using Monte Carlo sampling,

$$\boldsymbol{\nabla}_{\boldsymbol{\varphi}} j_T(\boldsymbol{\varphi}) \approx \frac{1}{m} \sum_{i=1}^{m} g_{0:T}^{(i)} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}} \left(\boldsymbol{a}_t^{(i)} \mid \boldsymbol{x}_t^{(i)} \right).$$

These estimates are unbiased but typically their variance is very large. Using so-called BASELINES can reduce the variance dramatically.

Lemma 4 (Score gradients with baselines)**.** *We have,*

$$\mathbb{E}_{\tau \sim \Pi_\varphi} \left[G_0 \nabla_\varphi \log \Pi_\varphi(\tau) \right] = \mathbb{E}_{\tau \sim \Pi_\varphi} \left[(G_0 - b) \nabla_\varphi \log \Pi_\varphi(\tau) \right]$$

Here, $b \in \mathbb{R}$ is called a baseline.

A commonly used baseline is

$$b\left(\tau_{0:t-1}\right) \doteq \sum_{m=0}^{t-1} \gamma^m r_m.$$

This baseline subtracts the returns of all actions before time t. Intuitively, using this baseline, the score gradient only considers downstream returns. Recall from eq. (11.3) that we defined $G_{t:T}$ as the bounded discounted payoff from time t. It is also commonly called the (bounded) DOWNSTREAM RETURN (or REWARD TO GO) beginning at time t.
For a fixed trajectory τ that is bounded at time T, we have

$$G_0 - b\left(\tau_{0:t-1}\right) = \gamma^t G_{t:T},$$

yielding the gradient estimator,

$$\boldsymbol{\nabla}_{\boldsymbol{\varphi}} j(\boldsymbol{\varphi}) \approx \nabla_{\boldsymbol{\varphi}} j_T(\boldsymbol{\varphi}) = \mathbb{E}_{\tau \sim \Pi_\varphi} \left[G_0 \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right] = \mathbb{E}_{\tau \sim \Pi_\varphi} \left[\sum_{t=0}^{T-1} \gamma^t G_{t:T} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}} \left(\boldsymbol{a}_t \mid \boldsymbol{x}_t \right) \right]. \tag{11.5}$$

Performing stochastic gradient descent with the score gradient estimator and downstream returns is known as the REINFORCE algorithm (Williams, 1992).

REINFORCE algorithm

initialize policy weights $\boldsymbol{\varphi}$
while not converged **do**
  generate an episode (rollout) to obtain trajectory $\tau = (X_0, A_0, R_0, X_1, A_1, R_1, \ldots, X_T, X_T, R_T)$
  for $t = 0$ to $T - 1$ **do**
    set $g_{t:T}$ to the downstream return from time t
    update $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta \gamma^t g_{t:T} \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}} \left(\boldsymbol{a}_t \mid \boldsymbol{x}_t \right)$
  end for
end while

The variance of REINFORCE can be reduced further. A common technique is to subtract a term b_t from the downstream returns,

$$\boldsymbol{\nabla}_{\boldsymbol{\varphi}} j(\boldsymbol{\varphi}) = \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[\sum_{t=0}^{T} \gamma^t \left(G_{t:T} - b_t \right) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}} \left(\boldsymbol{a}_t \mid \boldsymbol{x}_t \right) \right].$$

For example, we can subtract the mean reward to go,

$$b_t \doteq \frac{1}{T} \sum_{t=0}^{T-1} G_{t:T}.$$

The main advantage of policy gradient methods such as REINFORCE is that they can be used in continuous action spaces. However, REINFORCE is not guaranteed to find an optimal policy. Even when operating in very small domains, REINFORCE can get stuck in local optima.
Next, we will combine value approximation techniques like Q-learning and policy gradient methods, leading to an often more practical family of methods called actor-critic methods.

11.4 On-policy Actor-Critics

ACTOR-CRITIC METHODS reduce the variance of policy gradient estimates by using ideas from value function approximation. They use function approximation both to approximate value functions and to approximate policies. The goal for these algorithms is to scale to reinforcement learning problems, where we both have large state spaces and large action spaces.
Actor-Critic methods consist of two components:

- a parameterized policy, $\pi(\boldsymbol{a} \mid \boldsymbol{x}; \boldsymbol{\varphi}) \doteq \pi_{\varphi}$, which is called ACTOR
- a value function approximation, $q^{\pi_\varphi}(x, a) \approx Q^{\pi_\varphi}(x, a; \theta)$, which is called CRITIC (in the following, we will abbreviate Q^{π_φ} by Q)

11.4.1 Advantage Function

A key concept of actor-critic methods is the advantage function.

Definition 34. *Given a policy π, the* ADVANTAGE FUNCTION

$$a^{\pi}(x, a) \doteq q^{\pi}(x, a) - v^{\pi}(x) = q^{\pi}(x, a) - \mathbb{E}_{a' \sim \pi(x)}\left[q^{\pi}\left(x, a'\right)\right], \tag{11.6}$$

measures the advantage of picking action $a \in \mathcal{A}$ when in state $x \in \mathcal{X}$ over simply following policy π.

It follows immediately from eq. (11.6) that for any policy π and state $x \in \mathcal{X}$, there exists an action $a \in \mathcal{A}$ such that $a^{\pi}(x, a)$ is non-negative,

$$\max_{a \in \mathcal{A}} a^{\pi}(x, a) \geq 0.$$

Moreover, it follows from Bellman's theorem (9.5) that

$$\pi \text{ is optimal } \iff \forall x \in \mathcal{X}, a \in \mathcal{A} : a^{\pi}(x, a) \leq 0.$$

In other words, quite intuitively, π is optimal if and only if there is no action that has an advantage in any state over the action that is played by π.
Finally, we can re-define the greedy policy π_q with respect to the state-action value function q as

$$\pi_q(x) \doteq \arg\max_{a \in \mathcal{A}} a(x, a)$$

since

$$\arg\max_{\boldsymbol{a} \in \mathcal{A}} a(\boldsymbol{x}, \boldsymbol{a}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} q(\boldsymbol{x}, \boldsymbol{a}) - v(\boldsymbol{x}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} q(\boldsymbol{x}, \boldsymbol{a}),$$

as $v(x)$ is independent of $\boldsymbol{a}$. Using this quantity rather than q often has numerical advantages.

11.4.2 Policy Gradient Theorem

Let's recall equation (11.5),

$$\nabla_{\boldsymbol{\varphi}} j_T(\boldsymbol{\varphi}) = \mathbb{E}_{\tau \sim \Pi_{\varphi}}\left[\sum_{t=0}^{T-1} \gamma^t G_{t:T} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}\left(\boldsymbol{a}_t \mid \boldsymbol{x}_t\right)\right].$$

and let us now reinterpret score gradients while taking into account the tails of $j(\varphi)$:

$$\boldsymbol{\nabla}_{\boldsymbol{\varphi}} j(\boldsymbol{\varphi}) = \lim_{T \to \infty} \boldsymbol{\nabla}_{\boldsymbol{\varphi}} j_T(\boldsymbol{\varphi}) = \sum_{t=0}^{\infty} \mathbb{E}_{\tau \sim \Pi_{\varphi}}\left[\gamma^t G_t \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}\left(\boldsymbol{a}_t \mid \boldsymbol{x}_t\right)\right] = \sum_{t=0}^{\infty} \mathbb{E}_{\tau_{t:\infty} \sim \Pi_{\varphi}}\left[\gamma^t G_t \nabla_{\varphi} \log \pi_{\varphi}\left(a_t \mid x_t\right)\right]. \tag{11.7}$$

where

$$\tau_{t:\infty} \doteq \left(\left(x_t, a_t, r_t, x_{t+1}\right), \left(x_{t+1}, a_{t+1}, r_{t+1}, x_{t+2}\right), \ldots\right),$$

and the last equality follows by the fact that the expectations only consider downstream returns, and so we can disregard all data from the trajectory prior to time t.
The DISCOUNTED STATE OCCUPANCY MEASURE is defined as

$$\rho_{\boldsymbol{\varphi}}^{\infty}(\boldsymbol{x}) \doteq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}_{\pi_\varphi}\left(\mathbf{X}_t = \boldsymbol{x}\right)$$

and, intuitively, it measures how often we visit state x when following policy π_φ. Then, from (11.7), we can arrive to the following result.

Theorem 39 (Policy gradient theorem in terms of ρ_{φ}^{∞})**.** *Policy gradients can be represented in terms of the Q-function as follows*

$$\nabla_\varphi j(\boldsymbol{\varphi}) \propto \mathbb{E}_{\boldsymbol{x}\sim\rho_{\varphi}^{\infty}}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\varphi}}(\cdot|x)}\left[q^{\pi_\varphi}(\boldsymbol{x},\boldsymbol{a})\nabla_{\boldsymbol{\varphi}}\log\pi_{\boldsymbol{\varphi}}(\boldsymbol{a}\mid\boldsymbol{x})\right]$$

This theorem tells us that we can represent policy gradients in terms of the Q-function q^{π_φ} and naturally suggests plugging in approximations Q^{π_φ} for the action-value function.

11.4.3 Online actor-critic

In deep reinforcement learning, neural networks are used to parameterize both actor and critic. Therefore, in principle, the actor-critic framework allows scaling to both large state spaces and large action spaces. We begin by discussing on-policy actor-critics.

One approach in the online setting (i.e., non-episodic setting), is to simply use TD-learning for learning the critic. To learn the actor, we use stochastic gradient descent with gradients obtained using single samples from

$$\nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \approx \sum_{t=0}^{\infty}\mathbb{E}_{(\boldsymbol{x}_t,\boldsymbol{a}_t)\sim\pi_{\boldsymbol{\varphi}}}\left[\gamma^t Q\left(\boldsymbol{x}_t,\boldsymbol{a}_t;\boldsymbol{\theta}\right)\nabla_{\boldsymbol{\varphi}}\log\pi_{\boldsymbol{\varphi}}\left(\boldsymbol{a}_t\mid\boldsymbol{x}_t\right)\right] \tag{11.8}$$

where Q is a bootstrapping estimate of q^{π_φ}. This algorithm is known as ONLINE ACTOR-CRITIC or Q ACTOR-CRITIC.

Online actor-critic

initialize parameters φ and θ
while not converged **do**
  use π_φ to obtain transition (x, a, r, x')
  actor update: $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta\gamma^t Q(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})\nabla_{\boldsymbol{\varphi}}\log\pi_{\boldsymbol{\varphi}}(\boldsymbol{a}\mid\boldsymbol{x})$
  critic update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\left(r + \gamma Q\left(\boldsymbol{x}', \pi_{\boldsymbol{\varphi}}\left(\boldsymbol{x}'\right);\boldsymbol{\theta}\right) - Q(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}} Q(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})$
end while

Due to the use of TD-learning for learning the critic, this algorithm is fundamentally on-policy.

11.4.4 Improved Actor-Critics

Reducing Variance To further reduce the variance of the gradient estimates, it turns out that a similar approach to the baselines we discussed in the previous section on policy gradient methods is useful. A common approach is to subtract the state value function from estimates of the Q-function,

$$\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta_t\gamma^t(Q(x,\boldsymbol{a};\boldsymbol{\theta}) - V(\boldsymbol{x};\boldsymbol{\theta}))\boldsymbol{\nabla}_{\boldsymbol{\varphi}}\log\pi_{\boldsymbol{\varphi}}(\boldsymbol{a}\mid\boldsymbol{x}) = \boldsymbol{\varphi} + \eta_t\gamma^t A(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})\boldsymbol{\nabla}_{\boldsymbol{\varphi}}\log\pi_{\boldsymbol{\varphi}}(\boldsymbol{a}\mid\boldsymbol{x})$$

where $A(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})$ is a bootstrapped estimate of the advantage function a^{π_φ}. This algorithm is known as ADVANTAGE ACTOR-CRITIC ($\mathrm{A_2C}$).This technique can be combined with Monte-Carlo Return estimation, blending between REINFORCE and actor critic methods, for example GENERALIZED ADVANTAGE ESTIMATION (GAE).

Improving sample efficiency A well-known variant that slightly improves the sample efficiency is TRUST-REGION POLICY OPTIMIZATION (TRPO). TRPO uses multiple iterations, where in each iteration a fixed critic is used to optimize the policy. During iteration k, we have,

$$\boldsymbol{\varphi}_{k+1} \leftarrow \arg\max_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \quad \text{subject to } \mathbb{E}_{\boldsymbol{x}\sim\rho_{\varphi_k}^{\infty}}\mathrm{KL}\left(\pi_{\boldsymbol{\varphi}_k}(\cdot\mid\boldsymbol{x})\|\pi_{\boldsymbol{\varphi}}(\cdot\mid\boldsymbol{x})\right) \leq \delta$$

where

$$J(\boldsymbol{\varphi}) \doteq \mathbb{E}_{\boldsymbol{x}\sim\rho^{\infty}_{\varphi_k}, \boldsymbol{a}\sim\pi_{\varphi_k}(\cdot\mid\boldsymbol{x})}\left[\frac{\pi_{\boldsymbol{\varphi}}(\boldsymbol{a}\mid\boldsymbol{x})}{\pi_{\boldsymbol{\varphi}_k}(\boldsymbol{a}\mid\boldsymbol{x})}A^{\pi_{\varphi_k}(\boldsymbol{x},\boldsymbol{a})}\right]$$

Also PROXIMAL POLICY OPTIMIZATION (PPO) is a heuristic variant of TRPO that often works well in practice.

11.5 Off-policy Actor-Critics

In many applications, sample efficiency is crucial. Either because requiring too many interactions is computationally prohibitive or because obtaining sufficiently many samples for learning a near-optimal policy is simply impossible. We therefore now want to look at a separate family of actor-critic methods, which are off-policy, and hence, allow for the reuse of past data. These algorithms use the reparameterization gradient estimates instead of score gradient estimators.

The on-policy methods that we discussed in the previous section can be understood as performing a variant of policy iteration, where we use an estimate of the state-action value function of the current policy and then try to improve that policy by acting greedily with respect to this estimate. They mostly vary in how improving the policy is traded with improving the estimate of its value. Fundamentally, these methods rely on policy evaluation. The techniques that we will introduce in this section are much more closely related to value iteration, essentially making use of Bellman's optimality principle to learn the optimal value function directly which characterizes the optimal policy.

To begin with, let us assume that the policy π is deterministic. We will later lift this restriction in the next subsection. Recall that our initial motivation to consider policy gradient methods and then actor-critic methods was the intractability of the DQN loss

$$\ell_{\mathrm{DQN}}(\boldsymbol{\theta};\mathcal{D}) = \frac{1}{2}\sum_{(x,a,r,x')\in\mathcal{D}}\left(r+\gamma\max_{a'\in\mathcal{A}}Q^{\star}\left(x',\boldsymbol{a}';\boldsymbol{\theta}^{\mathrm{old}}\right)-Q^{\star}(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})\right)^2$$

when the action space $\mathcal{A}$ is large. What if we simply replace the exact maximum over actions by a parameterized policy?

$$\ell_{\mathrm{DQN}}(\boldsymbol{\theta};\mathcal{D}) \approx \frac{1}{2}\sum_{(x,a,r,x')\in\mathcal{D}}\left(r+\gamma Q^{\star}\left(\boldsymbol{x}',\pi_{\varphi}\left(\boldsymbol{x}'\right);\boldsymbol{\theta}^{\mathrm{old}}\right)-Q^{\star}(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})\right)^2$$

We want to train our parameterized policy to learn the maximization over action, that is, to approximate the greedy policy

$$\pi_{\varphi}(x) \approx \pi^{\star}(x) = \arg\max_{a\in\mathcal{A}} Q^{\star}(x,a;\theta).$$

Note that, in the previous, we already apply the improvement of DDQN to use the most recent estimate of the Q-function for action selection.
The key idea is that if we use a "rich-enough" parameterization of policies, selecting the greedy policy with respect to $Q^{\star}$ is equivalent to

$$\varphi^{\star} = \arg\max_{\varphi}\mathbb{E}_{x\sim\mu}\left[Q^{\star}\left(x,\pi_{\varphi}(x);\theta\right)\right]$$

where $\mu(x) > 0$ is an exploration distribution over states with full support (it explores all states). Commonly, the exploration distribution μ is taken to be the distribution that samples states uniformly from a replay buffer. Note that we can easily obtain unbiased gradient estimates with respect to φ of

$$\mathbb{E}_{x\sim\mu}\left[Q^{\star}\left(x,\pi_{\varphi}(x);\theta\right)\right]$$

as

$$\mathbb{E}_{\boldsymbol{x}\sim\mu}\left[\boldsymbol{\nabla}_{\boldsymbol{\varphi}}Q^{\star}\left(\boldsymbol{x},\pi_{\boldsymbol{\varphi}}(\boldsymbol{x});\boldsymbol{\theta}\right)\right].$$

Analogously to on-policy actor-critics, see eq. (11.8), we use a bootstrapping estimate of $Q^{\star}$, that is, we neglect the dependence of the critic $Q^{\star}$ on the actor π_{φ}, and in particular, the policy parameters φ. We have seen that

bootstrapping works with Q-learning, so there is reason to hope that it will work in this context too. This then allows us to use the chain rule to compute the gradient,

$$\nabla_\varphi Q^\star\left(x, \pi_\varphi(x); \theta\right) = D_\varphi \pi_\varphi(x) \cdot \nabla_a Q^\star(x, a; \theta)|_{a=\pi_\varphi(x)} .$$

This corresponds to evaluating the bootstrapping estimate of the Q-function at $\pi_\varphi(x)$ and obtaining a gradient estimate of the policy estimate (e.g., through automatic differentiation). Note that as π_φ is vector-valued, $D_\varphi \pi_\varphi(x)$ is the Jacobian of π_φ evaluated at x.

Now that we have estimates of the gradient of our optimization target J_μ, it is natural to ask how we should select actions (based on π_φ) to trade exploration and exploitation. As we have seen, policy gradient techniques rely on the randomness in the policy to explore, but here we consider deterministic policies. As our method is off-policy, a simple idea in continuous action spaces is to add Gaussian noise to the action selected by π_φ - also known as GAUSSIAN NOISE DITHERING. This corresponds to an algorithm called DEEP DETERMINISTIC POLICY GRADIENTS (DDPG). This algorithm is essentially equivalent to Q-learning with function approximation (e.g., DQN), with the only exception that we replace the maximization over actions with the learned policy π_φ.

Deep deterministic policy gradients, DDPG

initialize φ, θ, a (possibly non-empty) replay buffer $\mathcal{D} = \varnothing$
set $\varphi^{\text{old}} = \varphi$ and $\theta^{\text{old}} = \theta$
for $t = 1$ to ∞ **do**
  observe state x, pick action $a = \pi_\varphi(x) + \epsilon$ for $\epsilon \sim \mathcal{N}(\mathbf{0}, \lambda I$
  execute a, observe r and x'
  add (x, a, r, x') to the replay buffer $\mathcal{D}$
  if collected "enough" data **then**
    `// policy improvement step`
    for some iterations **do**
      sample a mini-batch B of $\mathcal{D}$
      for each transition in B, compute the label $y = r + \gamma Q^\star\left(\boldsymbol{x}', \boldsymbol{\pi}\left(\boldsymbol{x}'; \varphi^{\text{old}}\right); \boldsymbol{\theta}^{\text{old}}\right)$
      `// critic update`
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \frac{1}{|B|} \sum_{(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}', y) \in B} \left(y - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right)^2$
      `// actor update`
      $\varphi \leftarrow \boldsymbol{\varphi} + \eta \nabla_\varphi \frac{1}{|B|} \sum_{(x, a, r, x', y) \in B} Q^\star(x, \pi(x; \varphi); \theta)$
      $\boldsymbol{\theta}^{\text{old}} \leftarrow (1 - \rho)\theta^{\text{old}} + \rho \boldsymbol{\theta}$
      $\varphi^{\text{old}} \leftarrow (1 - \rho)\varphi^{\text{old}} + \rho \boldsymbol{\varphi}$
    end for
  end if
end for

TWIN DELAYED DDPG (TD_3) is an extension of DDPG that uses two separate critic networks for predicting the maximum action and evaluating the policy. This addresses the maximization bias akin to Double-DQN. TD_3 also applies delayed updates to the actor network, which increases stability.

11.5.1 Off-Policy Actor Critics with Randomized Policies

We have seen that randomized policies naturally encourage exploration. With deterministic actor-critic methods like DDPG, we had to inject Gaussian noise to enforce sufficient exploration. A natural question is therefore whether we can also handle randomized policies in this framework of off-policy actor-critics.

In DDPG, had to inject random noise to ensure exploration Can we directly allow randomized policies?

Critic update For the critic update we have

$$\theta \leftarrow \theta - \eta \nabla_\theta \frac{1}{|B|} \sum_{(x, a, r, x', y) \in B} \left(Q\left(x, a; \theta\right) - y\right)^2$$

where $y = r + \gamma Q\left(x', \pi\left(x', \varphi^{\text{old}}\right), \theta^{\text{old}}\right)$ and we can obtain unbiased gradient estimates by sampling from

$$a \sim \pi\left(x', \varphi^{\text{old}}\right).$$

This provides us with a method of obtaining unbiased gradient estimates for the critic by allowing randomized policies.

Policy update We also need to reconsider the actor update. When using a randomized policy, the objective function changes from

$$\mathbb{E}_{x\sim\mu}\left[Q^\star\left(x, \pi_\varphi(x); \theta\right)\right]$$

to

$$\mathbb{E}_{\boldsymbol{x}\sim\mu}\mathbb{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{x};\boldsymbol{\varphi})}\left[Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right]$$

of which we can obtain gradients via

$$\mathbb{E}_{\boldsymbol{x}\sim\mu}\nabla_{\boldsymbol{\varphi}}\mathbb{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{x};\boldsymbol{\varphi})}\left[Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right].$$

The inner expectation is with respect to a measure that depends on the parameters φ, which we are trying to optimize. We therefore cannot move the gradient operator inside the expectation. This is a problem that we have already encountered several times. In the previous section on policy gradients, we used the score gradient estimator. Earlier, in the chapter on variational inference, we have already seen reparameterization gradients. Here, if our policy is reparameterizable, we can use the reparameterization trick. As we have seen, not only Gaussians are reparameterizable. In general, we called a distribution (in this context, a policy) REPARAMETERIZABLE iff $a \sim \pi(x; \varphi)$ is such that $a = g(\epsilon; x, \varphi)$, where $\epsilon \sim \phi$ is an independent random variable. Then, we have,

$$\nabla_\varphi\mathbb{E}_{\boldsymbol{a}\sim\pi(x;\varphi)}\left[Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right] = \mathbb{E}_{\epsilon\sim\phi}\left[\nabla_\varphi Q^\star(\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{\epsilon}; \boldsymbol{x}, \boldsymbol{\varphi}); \boldsymbol{\theta})\right] = \mathbb{E}_{\epsilon\sim\phi}\left[\nabla_a Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})|_{\boldsymbol{a}=g(\epsilon;x,\varphi)} \cdot \boldsymbol{D}_\varphi g(\boldsymbol{\epsilon}; \boldsymbol{x}, \boldsymbol{\varphi})\right].$$

In this way, we can obtain unbiased gradient estimates for reparameterizable policies. Note that this general technique does not only apply to continuous action spaces.

11.6 Entropy Regularization

A key issue with relying on randomized policies for exploration is that they might collapse to deterministic policies. That is, the algorithm might quickly reach a local optimum, where all mass is placed on a single action. A simple trick that encourages a little bit of extra exploration is to regularize the randomized policies away from putting all mass on a single action. In other words, we want to encourage the policies to exhibit some uncertainty. A natural measure of uncertainty is entropy, which we have already seen several times. This approach is known as ENTROPY REGULARIZATION or MAXIMUM ENTROPY REINFORCEMENT LEARNING (MERL). Canonically, entropy regularization is applied to finite horizon rewards, yielding the optimization problem of maximizing

$$j_\lambda(\boldsymbol{\varphi}) \doteq j(\boldsymbol{\varphi}) + \lambda \mathrm{H}\left[\Pi_{\boldsymbol{\varphi}}\right] = \sum_{t=1}^{T} \mathbb{E}_{(\boldsymbol{x}_t, \boldsymbol{a}_t)\sim\Pi_{\boldsymbol{\varphi}}}\left[r\left(\boldsymbol{x}_t, \boldsymbol{a}_t\right) + \lambda \mathrm{H}\left[\pi_{\boldsymbol{\varphi}}\left(\cdot \mid \boldsymbol{x}_t\right)\right]\right], \tag{11.9}$$

where we have a preference for entropy in the actor distribution to encourage exploration which is regulated by the temperature parameter λ. As $\lambda \to 0$, we recover the standard reinforcement learning objective (here for finite-horizon rewards):

$$j(\boldsymbol{\varphi}) = \sum_{t=1}^{T} \mathbb{E}_{(\boldsymbol{x}_t, \boldsymbol{a}_t)\sim\Pi_\varphi}\left[r\left(\boldsymbol{x}_t, \boldsymbol{a}_t\right)\right].$$

Here, for notational convenience, we begin the sum with $t = 1$ rather than $t = 0$. Without entering into details, we limit ourselves in noticing that, starting from the optimization problem of maximizing (11.9), we can suitably define regularized (action)-value functions, called SOFT VALUE FUNCTIONS and then we can use reparametrization gradients to obtain the SOFT ACTOR CRITIC (SAC) algorithm.

11.7 Summary Model-free RF algorithms for large state-action spaces X and A

11.7.1 On-policy

ON-POLICY algorithms are characterized as follows.

Technique: The agent executes its current policy to sample new trajectories and learn the optimal policy.

Exploration: The agent has full control over its actions when interacting with the environment.

Sampling: Cannot reuse data coming from other policies.

Examples of on-policy algorithms are the following.

REINFORCE. It is a policy approximation algorithm that performs stochastic gradient descent with the score gradient estimator and downstream returns to learn the policy. To reduce variance, uses baselines.

OnlineActorCritic. In this case neural networks are used to parameterize both the policy (actor) and the value function (critic). Uses TD-learning to learn the critic. The variance can be reduced by using the advantage function in the actor update, and in this cases the algorithm is called $\mathrm{A_2C}$.

TRPO It is an Online Actor Critic that improves sample efficiency by iteratively optimizing a surrogate objective within trust region

PPO It is a heuristic variant of TRPO that often works well in practice

11.7.2 Off-policy

OFF-POLICY algorithms are characterized as follows.

Technique: The agent uses observational data (coming from an arbitrary policy) to learn the optimal policy.

Exploration: The agent may not be able to (arbitrarily) interact with the environment

Sampling: Can use data coming from any arbitrary policy.

Examples of off-policy algorithms are the following.

DQN/DDQN: It is a value function approximation algorithm that can be considered as Q-learning with function approximation. In DQN the optimization target is stabilized by cloning the neural network and maintaing one changing neural network (online network) for the most recent estimate of the Q-function, and one fixed neural network (target network) used as the target which is updated infrequently. DDQN is a variant of DQN that address the maximization bias.

DDPG In this case neural networks are used to parameterize both the policy (actor) and the value function (critic). DDPG is essentially equivalent to Q-learning with function approximation (DQN), with the only exception that we replace the maximization over actions with the learned policy π_φ.

TD3 It is an extension of DDPG to avoid maximization bias.

SAC It is variant of DDPG/TD3 for entropy regularized MDPs.

Chapter 12

Model-based Reinforcement Learning

In chapter 10, we began by discussing model-based reinforcement learning which attempts to learn the underlying Markov decision process and then use it for planning. We have seen that in the tabular setting, computing and storing the entire model is computationally expensive. This led us to consider the family of model-free approaches, which learn the value function directly, and as such can be considered more economical in the amount of data that they store. In chapter 11, we saw that using function approximation, we were able to scale model-free methods to very large state and action spaces.
We will now explore similar ideas in the model-based framework. Namely, we will use function approximation to condense the representation of our model of the environment. More concretely, we will learn an approximate dynamics model $f \approx p$ and approximate rewards r.

Outline of Model-based reinforcement learning

start with an initial policy π and no (or some) initial data $\mathcal{D}$
for several episodes **do**
  roll out policy π to collect data
  learn a model of the dynamics f and rewards r from data
  plan a new policy π based on the estimated models
end for

We face three main challenges in model-based reinforcement learning.

- How do we plan given a model?
- How do we learn f and r?
- How do we trade exploration and exploitation?

12.1 Planning

There exists a large literature on planning in various settings. These settings can be mainly: characterized as

- discrete or continuous action spaces,
- fully- or partially observable state spaces,
- constrained or unconstrained,
- linear or non-linear dynamics.

In chapter 9, we have already seen algorithms such as policy iteration and value iteration, which can be used to solve planning exactly in tabular settings. In the following, we will now focus on the setting of continuous state and action spaces, fully observable state spaces, no constraints, and non-linear dynamics.

12.1.1 Deterministic Dynamics

To begin with, let us assume that our dynamics model is deterministic and known. That is, given a state-action pair, we know the subsequent state,

$$x_{t+1} = f(x_t, a_t)$$

We continue to focus on the setting of infinite-horizon discounted returns and then, our objective becomes

$$\max_{\boldsymbol{a}_{0:\infty}} \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{a}_t) \quad \text{such that } \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{a}_t).$$

The problem is that, because we are optimizing over an infinite time horizon, we cannot solve this optimization problem directly. This problem is studied ubiquitously in the area of "optimal control".

Model predictive control (MPC)

The key idea of a classical algorithm from optimal control called RECEDING HORIZON CONTROL (RHC) or MODEL PREDICTIVE CONTROL (MPC) is to iteratively plan over finite horizons. That is, in each round, we plan over a finite time horizon H, carry out the first action and then replan.

Model predictive control, MPC

for $t = 0$ to ∞ **do**
  observe x_t
  optimize performance over horizon H

$$\max_{\boldsymbol{a}_{t:t+H-1}} \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r(\boldsymbol{x}_\tau, \boldsymbol{a}_\tau) \quad \text{such that } \boldsymbol{x}_{\tau+1} = \boldsymbol{f}(\boldsymbol{x}_\tau, \boldsymbol{a}_\tau)$$

  carry out action a_t
end for

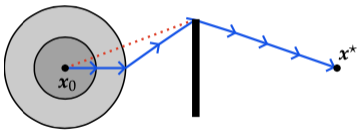


Figure 12.1: Illustration of MPC in a deterministic transition model. The agent starts in position x_0 and wants to reach $x^\star$ despite the black obstacle. We use the reward function $r(x) = -\|x - x^\star\|$. The gray concentric circles represent the length of a single step. We plan with a time horizon of $H = 2$. Initially, the agent does not "see" the black obstacle, and therefore moves straight towards the goal. As soon as the agent sees the obstacle, the optimal trajectory is replanned. The dotted red line corresponds to the optimal trajectory, the agent's steps are shown in blue.

The state $\boldsymbol{x}_\tau$ can be interpreted as a deterministic function $\boldsymbol{x}_\tau(\boldsymbol{a}_{t:\tau-1})$, which depends on all actions from time t to time $\tau - 1$ and the state $\boldsymbol{x}_t$. Thus at step $t$we need to maximize the following

$$J_H(\boldsymbol{a}_{t:t+H-1}) = \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r(\boldsymbol{x}_\tau(\boldsymbol{a}_{t:\tau-1}), \boldsymbol{a}_\tau) \tag{12.1}$$

This optimization problem is in general non-convex. If the actions are continuous and the dynamics and reward models are differentiable, we can nevertheless obtain analytic gradients of the previous. This can be done using the chain rule and "backpropagating" through time, analogously to backpropagation in neural networks.

Especially for large horizons H, this optimization problem becomes difficult to solve exactly due to local optima and vanishing/exploding gradients. Often, heuristic global optimization methods are used to optimize J_H. An example are RANDOM SHOOTING METHODS, which find the optimal choice among a set of random proposals.

Random shooting methods

generate m sets of random samples, $\boldsymbol{a}_{t:t+H-1}^{(i)}$

pick the sequence of actions $a_{t:t+H-1}^{(i^\star)}$ where $i^\star = \underset{i\in[m]}{\arg\max} J_H\left(\boldsymbol{a}_{t:t+H-1}^{(i)}\right)$

A common problem of finite-horizon methods is that in the setting of sparse rewards, there is often no signal that can be followed. A solution to this problem is to amend a long-term value estimate to the finite-horizon sum. The idea is to not only consider the rewards attained while following the actions $a_{t:t_H-1}$, but to also consider the value of the final state x_{t+H}, which estimates the discounted sum of future rewards:

$$J_H\left(\boldsymbol{a}_{t:t+H-1}\right) \doteq \underbrace{\sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r\left(\boldsymbol{x}_\tau\left(\boldsymbol{a}_{t:\tau-1}\right), \boldsymbol{a}_\tau\right)}_{\text{short-term}} + \underbrace{\gamma^H V\left(\boldsymbol{x}_{t+H}\right)}_{\text{long-term}}.$$

Intuitively, $\gamma^H V\left(x_{t+H}\right)$ is estimating the tail of the infinite sum. Observe that for $H = 1$, when we use the value function estimate associated with this MPC controller, maximizing J_H coincides with using the greedy policy π_V. That is,

$$\boldsymbol{a_t} = \underset{\boldsymbol{a}\in\mathcal{A}}{\arg\max} J_1(\boldsymbol{a}) = \pi_V\left(\boldsymbol{x}_t\right).$$

12.1.2 Stochastic Dynamics

How can we extend this approach to planning to a stochastic transition model? A natural extension of model predictive control is to do what is called TRAJECTORY SAMPLING. Like in MPC, we still optimize over a deterministic sequence of actions, but now we will average over all resulting trajectories.

Trajectory sampling

for $t = 0$ to ∞ **do**

observe x_t

optimize expected performance over a finite horizon H,

$$\max_{\boldsymbol{a}:t+H-1} \mathbb{E}_{\boldsymbol{x}_{t+1:t+H}}\left[\sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r_\tau + \gamma^H V\left(\boldsymbol{x}_{t+H}\right) \mid \boldsymbol{a}_{t:t+H-1}, f\right]$$

carry out action a_t

end for

Computing the expectation exactly is typically not possible as this involves solving a high-dimensional integral of non-linear functions. Instead, a common approach is to use Monte Carlo estimates of this expectation. This approach is known as MONTE CARLO TRAJECTORY SAMPLING. The key issue with using sampling based estimation is that the sampled trajectory (i.e., sampled sequence of states) we obtain, depends on the actions we pick. In other words, the measure we average over depends on the decision variables - the actions. This is a problem that we have seen several times already! It naturally suggests using the reparameterization trick. Previously, we have used the reparameterization trick to reparameterize variational distributions and to reparameterize policies. It turns out that we can use the exact same approach for reparameterizing the transition model. We say that a (stochastic) transition model f is reparameterizable iff $\boldsymbol{x}_{t+1} \sim f\left(\boldsymbol{x}_t, \boldsymbol{a}_t\right)$ is such that $x_{t+1} = \boldsymbol{g}\left(\boldsymbol{\epsilon}; x_t, a_t\right)$, where $\epsilon \sim \phi$ is a random variable that is independent of x_t and a_t. In this case, $\boldsymbol{x}_\tau$ is determined recursively by $\boldsymbol{a}_{t:\tau-1}$ and $\epsilon_{t:\tau-1}$,

$$x_\tau = x_\tau\left(\epsilon_{t:\tau-1}; a_{t:\tau-1}\right) \doteq g\left(\epsilon_{\tau-1}; g\left(\ldots; \left(\epsilon_{t+1}; g\left(\epsilon_t; x_t, a_t\right), a_{t+1}\right), \ldots\right), a_{\tau-1}\right). \tag{12.2}$$

This allows us to obtain unbiased estimates of J_H using Monte Carlo approximation,

$$J_H\left(\boldsymbol{a}_{t:t+H-1}\right) \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{\tau=t}^{t+H-1}\left(\gamma^{\tau-t} r\left(\boldsymbol{x}_\tau\left(\boldsymbol{\epsilon}_{t:\tau-1}^{(i)}; \boldsymbol{a}_{t:\tau-1}\right), \boldsymbol{a}_\tau\right) + \gamma^H V\left(\boldsymbol{x}_{t+H}\right)\right) \tag{12.3}$$

where $\epsilon_{t:t+H-1}^{(i)} \overset{\text{iid}}{\sim} \phi$ are independent samples. To optimize this approximation we can again compute analytic gradients or use shooting methods.

12.1.3 Parametric Policies

When using algorithms such as model predictive control for planning, planning needs to be done online before each time we take an action. This is called CLOSED-LOOP CONTROL and can be expensive. Especially when the time horizon is large, or we encounter similar states many times, it can be beneficial to "store" the planning decision in a (deterministic) policy,

$$a_t = \pi\left(x_t; \varphi\right) \doteq \pi_\varphi\left(x_t\right).$$

This policy can then be trained offline and evaluated cheaply online, which is known as OPEN-LOOP CONTROL. This is akin to a problem that we have discussed in detail in the previous chapter when extending Q-learning to large action spaces. There, this led us to discuss policy gradient and actor-critic methods. Recall that in Q-learning, we seek to follow the greedy policy,

$$\pi^\star(\boldsymbol{x}) = \arg\max_{\boldsymbol{a}\in\mathcal{A}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}),$$

and therefore had to solve an optimization problem over all actions. We accelerated this by learning an approximate policy that "mimicked" this optimization,

$$\boldsymbol{\varphi}^\star = \arg\max_{\boldsymbol{\varphi}} \underbrace{\mathbb{E}_{\boldsymbol{x}\sim\mu}\left[Q^\star\left(\boldsymbol{x}, \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}); \boldsymbol{\theta}\right)\right]}_{J_\mu(\boldsymbol{\varphi};\boldsymbol{\theta})}$$

where $\mu(x) > 0$ was some exploration distribution that has full support and thus leads to the exploration of all states. The key idea was that if we use a differentiable approximation Q and a differentiable parameterization of policies, which is "rich enough", then both optimizations are equivalent, and we can use the chain rule to obtain gradient estimates of the second expression. We then used this to derive the deep deterministic policy gradients (DDPG). It turns out that there is a very natural analogue to DDPG for model-based reinforcement learning. Instead of maximizing the Q-function directly, we use finite-horizon planning to estimate the immediate value of the policy within the next H time steps and simply use the Q-function to approximate the terminal value (i.e., the tails of the infinite sum). Then, our objective becomes,

$$J_{\mu,H}(\boldsymbol{\varphi}; \boldsymbol{\theta}) \doteq \mathbb{E}_{\boldsymbol{x}_0\sim\mu, \boldsymbol{x}_{1:H}|\pi_{\boldsymbol{\varphi}}, f}\left[\sum_{\tau=0}^{H-1}\gamma^\tau r_\tau + \gamma^H Q^\star\left(\boldsymbol{x}_H, \boldsymbol{\pi}_{\boldsymbol{\varphi}}\left(\boldsymbol{x}_H\right); \boldsymbol{\theta}\right)\right]$$

This approach naturally extends to randomized policies using reparameterization gradients. For $H = 0$, this coincides with the DDPG objective.

12.2 Learning

Thus far, we have considered known environments. That is, we assumed that the transition model f and the rewards r are known. In reinforcement learning, f and r are not known. Instead, we have to estimate them from data.

First, let us revisit one of our key observations when we first introduced the reinforcement learning problem. Namely, that the observed transitions x' and rewards r are conditionally independent given the state-action pairs (x, a), see (10.1). This is due to the Markovian structure of the underlying Markov decision process. This is the key observation that allows us to treat the estimation of the dynamics and rewards as a simple regression problem (or a density estimation problem when the quantities are stochastic rather than deterministic). More concretely,

we can estimate the dynamics and rewards off-policy using the standard supervised learning techniques we discussed in earlier chapters, from a replay buffer

$$\mathcal{D} = \left\{ (\underbrace{\boldsymbol{x}_t, \boldsymbol{a}_t}_{\text{"input"}}, \underbrace{r_t, \boldsymbol{x}_{t+1}}_{\text{"label"}}) \right\}_t .$$

Here, $\boldsymbol{x}_t$ and $\boldsymbol{a}_t$ are the "inputs", and r_t and x_{t+1} are the "labels" of the regression problem. Due to the conditional independence of the labels given the inputs, we have independent training data. The key difference to supervised learning is that the set of inputs depends on how we act. That is, the current inputs arise from previous policies, and the inputs which we will observe in the future will depend on the model (and policy) obtained from the current data: we have feedback loops! We will come back to this aspect of reinforcement learning in the next section on exploration. For now, recall we only assume that we have used an arbitrary policy to collect some data, which we then stored in a replay buffer, and which we now want to use to learn the "best-possible" model of our environment.

In the following, we will discuss how we can use the techniques from Bayesian learning, to learn the dynamics and reward models. Thereby, we will focus on learning the transition model f as learning the reward model r is completely analogous. For learning deterministic dynamics or rewards, we can use for example Gaussian processes or deep neural networks. We will now focus on the setting where the dynamics are stochastic, that is,

$$x_{t+1} \sim f\left(x_t, a_t; \boldsymbol{\psi}\right).$$

Example 11 (Conditional Gaussian dynamics)**.** *We could, for example, use a conditional Gaussian for the transition model,*

$$x_{t+1} \sim \mathcal{N}\left(\boldsymbol{\mu}\left(\boldsymbol{x}_t, a_t; \boldsymbol{\psi}\right), \boldsymbol{\Sigma}\left(\boldsymbol{x}_t, a_t; \boldsymbol{\psi}\right)\right). \tag{12.4}$$

As we have previously seen, we can rewrite the covariance matrix Σ as a product of a lower-triangular matrix $\mathcal{L}$ and its transpose using the Cholesky decomposition $\Sigma = \mathcal{L}\mathcal{L}^\top$ of $\boldsymbol{\Sigma}$. This allows us to represent the model by only $n(n+1)/2$ parameters. Moreover, we have learned that Gaussians are reparameterizable, which we have seen to be useful for planning.

A first approach might be to obtain a point estimate for f, either through maximum likelihood estimation, MLE, which we have seen to overfit easily, or through maximum a posteriori estimation, MAP. If, for example, f is represented as a deep neural network, we know that the MAP estimate of its weights is given by

$$\hat{\boldsymbol{\psi}} = \arg\min_{\boldsymbol{\psi}} -\log p(\boldsymbol{\psi}) - \sum_{t=1:T} \log N\left(x_{t+1} \mid \mu\left(x_t, a_t, \boldsymbol{\psi}\right), \Sigma\left(x_t, a_t, \boldsymbol{\psi}\right)\right)$$

and this can be optimized using stochastic gradient descent. However, using point estimates leads to a key pitfall of model based reinforcement learning. Using a point estimate of the model for planning - even if this point estimate is very accurate - often performs very poorly. The reason is that planning is very good at overfitting (i.e., exploiting) small errors in the transition model. Moreover, the errors in the model estimate compound over time when using a longer time horizon H. The key to remedy this pitfall lies in being robust to misestimated models. This naturally suggests quantifying the uncertainty in our model estimate and taking it into account during planning.

So the idea is to perform Bayesian learning of dynamics models, namely to learn a distribution over possible models f and r given prior beliefs, where f and r characterize the underlying Markov decision process. This goes to show another benefit of the model-based over the model-free approach to reinforcement learning: it is much easier to encode prior knowledge (as a Bayesian prior) about the transition and rewards model.
Also, we will differentiate between the epistemic uncertainty and the aleatoric uncertainty. Recall that EPISTEMIC UNCERTAINTY corresponds to our uncertainty about the model, $p(f \mid \mathcal{D})$, while ALEATORIC UNCERTAINTY corresponds to the uncertainty of the transitions in the underlying Markov decision process, which can be thought of as irreducible noise, $p\left(x_{t+1} \mid f, x_t, a_t\right)$.

Example 12 (Inference with conditional Gaussian dynamics)**.** *Let us revisit inference with our conditional Gaussian dynamics model from eq.* (12.4)*,*

$$x_{t+1} \sim \mathcal{N}\left(\mu\left(x_t, a_t; \psi\right), \Sigma\left(x_t, a_t; \psi\right)\right).$$

Recall that in the setting of Bayesian neural networks, most approximate inference techniques represented the approximate posterior using some form of a mixture of Gaussians

$$p\left(\boldsymbol{x}_{t+1} \mid \mathcal{D}, \boldsymbol{x}_t, \boldsymbol{a}_t\right) \approx \frac{1}{m} \sum_{i=1}^{m} \mathcal{N}\left(\boldsymbol{\mu}\left(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\psi}^{(i)}\right), \boldsymbol{\Sigma}\left(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\psi}^{(i)}\right)\right).$$

Hereby, the epistemic uncertainty is represented by the variance between mixture components (namely by the index of the mixture component i), and the aleatoric uncertainty by the average variance within the components (namely by the variance within component i).

Recall that the epistemic uncertainty corresponds to a distribution over Markov decision processes f, whereas the aleatoric uncertainty corresponds to the randomness in the transitions within one such MDP f. Once we selected a single MDP for planning, we should disregard the epistemic uncertainty and solely focus on the randomness of the transitions. Then, to take into account epistemic uncertainty, we should average our plan across the different realizations of f. This yields the following Monte Carlo estimate of our reward J_H (see also (12.3)),

$$\hat{J}_H\left(a_{t:t+H-1}\right) \simeq \frac{1}{m} \sum_{i=1:m} \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r\left(\boldsymbol{x}_\tau\left(\epsilon_{t:\tau-1}^{(i)}; \boldsymbol{a}_{t:\tau-1}, f\right), \boldsymbol{a}_\tau\right) + \gamma^H V\left(\boldsymbol{x}_{t+H}\right) \tag{12.5}$$

Here, $f(i) \overset{\text{iid}}{\sim} p(f \mid \mathcal{D})$ are independent samples of the transition model, and $\epsilon_{t:t+H-1}^{(i)} \overset{\text{iid}}{\sim} \phi$ parameterizes the dynamics analogously to eq. (12.2):

$$\boldsymbol{x}_\tau\left(\boldsymbol{\epsilon}_{t:\tau-1}; \boldsymbol{a}_{t:\tau-1}, f\right) \doteq f\left(\boldsymbol{\epsilon}_{\tau-1}; f\left(\ldots; \left(\boldsymbol{\epsilon}_{t+1}; f\left(\boldsymbol{\epsilon}_t; \boldsymbol{x}_t, \boldsymbol{a}_t\right), \boldsymbol{a}_{t+1}\right), \ldots\right), \boldsymbol{a}_{\tau-1}\right).$$

Observe that the epistemic and aleatoric uncertainty are treated differently. Within a particular MDP f (inner sum in (12.5)), we ensure that randomness (i.e., aleatoric uncertainty) is simulated consistently using our previous framework from our discussion of planning. The Monte Carlo samples of f take into account the epistemic uncertainty about the transition model. In our previous discussion of planning, we assumed the Markov decision process f to be fixed. Essentially, in in the outre sum in (12.5) we are now using Monte Carlo trajectory sampling as a subroutine and average over an "ensemble" of Markov decision processes.

The same approach that we have seen in section 12.1.3 can be used to compile these plans into a parametric policy that can be trained offline, in which case, we write $J_H(\pi)$ instead of $J_H\left(a_{t:t+H-1}\right)$. This leads us to a first greedily exploitative algorithm for model-based reinforcement learning, which is shown below. This algorithm is purely exploitative as it greedily maximizes the expected reward with respect to the transition model, taking into account epistemic uncertainty.

Greedy exploitation for model-based RL

start with (possibly empty) data $\mathcal{D} = \varnothing$ and a prior $p(f) = p(f \mid \mathcal{D})$
for several episodes **do**
  plan a new policy π to (approximately) maximize $\max_\pi \mathbb{E}_{f \sim p(\cdot \mid \mathcal{D})}\left[J_H(\pi; f)\right]$
  roll out policy π to collect more data
  update posterior $p(f \mid \mathcal{D})$
end for

In the context of Gaussian process models, this algorithm is called PROBABILISTIC INFERENCE FOR LEARNING CONTROL (PILCO), in the context of neural networks, this algorithm is called PROBABILISTIC ENSEMBLES WITH TRAJECTORY SAMPLING (PETS).

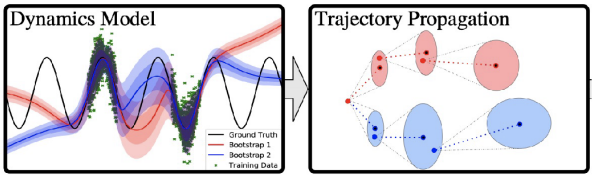


Figure 12.2: PETS algorithm. Uses an ensemble of neural networks each predicting conditional Gaussian transition distributions, trajectory sampling is used to evaluate performance and MPC is used for planning. Notably, PETS does not explicitly explore. Exploration only happens due to the uncertainty in the model, which already drives exploration to some extent.

12.3 Exploration

A key difference between RL and classical supervised learning is that the chosen actions affect the data we learn the models from, namely we are feced with the exploration-exploitation dilemma. One simple strategy to deal with this, that we already investigated, is the addition of some EXPLORATION NOISE. In other words, one follows a greedy exploitative strategy, but every once in a while, one chooses a random action (like in ϵ-greedy); or one adds additional noise to the selected actions (known as Gaussian noise "dithering"). We have already seen that in difficult exploration tasks, these strategies are not systematic enough. Two other approaches that we have considered before are THOMPSON SAMPLING and, more generally, the principle of OPTIMISM IN THE FACE OF UNCERTAINTY.

12.3.1 Thompson Sampling

Recall from section 8.3.2 the main idea behind Thompson sampling: namely, that the randomness in the realizations of f from the posterior distribution is already enough to drive exploration. That is, instead of picking the action that performs best on average across all realizations of f, Thompson sampling picks the action that performs best for a single realization of f. The epistemic uncertainty in the realizations of f leads to variance in the picked actions and provides an additional incentive for exploration. This yields the following algorithm.

Thompson sampling

start with (possibly empty) data $\mathcal{D} = \varnothing$ and a prior $p(f) = p(f \mid \mathcal{D})$
for several episodes **do**
  sample a model $f \sim p(\cdot \mid \mathcal{D})$
  plan a new policy π to (approximately) maximize $\max_\pi J_H(\pi; f)$
  roll out policy π to collect more data
  update posterior $p(f \mid \mathcal{D})$
end for

12.3.2 Optimistic Exploration

We have already seen in the context of Bayesian optimization and tab- ular reinforcement learning that optimism is a central pillar for exploration. But how can we explore optimistically in model-based reinforcement learning? Let us consider a set $\mathcal{M}(\mathcal{D})$ of plausible models given some data $\mathcal{D}$. Optimistic exploration would then optimize for the most advantageous model among all models that are plausible given the seen data.

Example 13. *In the context of conditional Gaussians, we can consider the set of all models such that the prediction of a single dimension i lies in some confidence interval,*

$$f_i(\boldsymbol{x}, \boldsymbol{a}) \in \mathcal{C}_i \doteq [\mu_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D}) - \beta_i \sigma_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D}), \mu_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D}) + \beta_i \sigma_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D})]\,,$$

where β_i tunes the width of the confidence interval. The set of plausible models is then given as

$$\mathcal{M}(\mathcal{D}) \doteq \{f \mid \forall x \in \mathcal{X}, a \in \mathcal{A}, i \in [d] : f_i(\boldsymbol{x}, \boldsymbol{a}) \in \mathcal{C}_i\}$$

When compared to greedy exploitation, instead of taking the optimal step on average with respect to all realizations of the transition model f, optimistic exploration, as shown in the algorithm below, takes the optimal step with respect to the most optimistic model among all transition models that are consistent with the data.

Optimistic exploration

start with (possibly empty) data $\mathcal{D} = \varnothing$ and a prior $p(f) = p(f \mid \mathcal{D})$
for several episodes **do**
  plan a new policy π to (approximately) maximize $\max_\pi \max_{f \in \mathcal{M}(\mathcal{D})} J_H(\pi; f)$
  roll out policy π to collect more data
  update posterior $p(f \mid \mathcal{D})$
end for

In general, the joint maximization over π and f is very difficult to solve computationally.