Summary Machine Learning 1

Phillip Lippe

December 16, 2018

Contents

1 Probability Theory

1.1 Multivariate Gaussian

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}\cdot|\boldsymbol{\Sigma}|^{1/2}}\cdot\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

$$\frac{\partial}{\partial\boldsymbol{\mu}}\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}$$

1.2 Rules of probability

	Discrete	**Continuous**
Additivity	$p(X\in A) = \sum_{x\in A} p(x)$	$p(x\in(a,b)) = \int_a^b p(x)dx$
Positivity	$0\le p(x)\le 1$	$0\le p(x)\not\le 1$
Normalization	$\sum_x p(x) = 1$	$\int_\chi p(x)dx = 1$
Sum Rule	$p(x) = \sum_{y\in\mathcal{Y}} p(x,y)$	$p(x) = \int p(x,y)dy$
Product Rule	$p(x,y) = p(x\|y)p(y)$	$p(x,y) = p(x\|y)p(y)$

1.3 Bayes Rule

$$\underbrace{p(x|y)}_{\text{posterior}} = \frac{\overbrace{p(y|x)}^{\text{likelihood}}\overbrace{p(x)}^{\text{prior}}}{\underbrace{p(y)}_{\text{evidence}}} = \frac{p(y|x)p(x)}{\int p(y|x)p(x)dx} \quad \text{or} \quad \frac{p(y|x)p(x)}{\sum p(y|x)p(x)}$$

2 Linear Regression

2.1 Basic approaches

2.1.1 Maximum likelihood

- Given a dataset $D = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$ of N independent observations
- The likelihood of the dataset given the model parameters $\boldsymbol{w}$ is specified as $p(D|\boldsymbol{w})$
- *Maximum likelihood estimation*: the most likely "explanation" of D is $\boldsymbol{w}_{\text{ML}}$:

$$\boldsymbol{w}_{\text{ML}} = \arg\max_{\boldsymbol{w}} p(D|\boldsymbol{w})$$

- Using the i.i.d. assumption, we can state $p(D|\boldsymbol{w}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{w})$
- For preventing numerical overflow and mostly simplifying the derivation, we can take the logarithm $\log p(D|\boldsymbol{w})$
- Maximum where $\frac{\partial}{\partial\boldsymbol{w}}\log p(D|\boldsymbol{w}) = 0$
- We can check whether our estimation is biased by comparing the expected result by the distribution parameters:

$$\mathbb{E}\left[\sigma^2_{ML}\right] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \frac{1}{N}\sum_{n=1}^{N}x_n\right)^2\right] = \frac{N-1}{N}\sigma^2 \implies \text{biased estimator}$$

2.1.2 Maximum a posteriori

- Choose the most probable model parameters $\boldsymbol{w}$ given data D:

$$\boldsymbol{w}_{\text{MAP}} = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w}|D)$$

- By applying the Bayes rule (and log), we get:

$$\boldsymbol{w}_{\text{MAP}} = \arg\max_{\boldsymbol{w}} \log p(D|\boldsymbol{w}) + \log p(\boldsymbol{w}) - \log p(D)$$

- We can drop the evidence as it is independent of $\boldsymbol{w}$

2.1.3 Bayesian approach

- Frequentist approaches only consider point estimates without taking the uncertainty of the prediction into account
- Given a prior belief over $\boldsymbol{w}$, we are interested in the posterior distribution (not only maximum!)
- The predictive distribution for a new data point $\boldsymbol{x}'$ is therefore

$$p(t'|\boldsymbol{x}', D) = \int p(t'|\boldsymbol{x}', \boldsymbol{w}) \cdot p(\boldsymbol{w}|D) d\boldsymbol{w}$$

- Thus, we also consider our uncertainty when predicting
- However, we need to compute the evidence for that which is mostly quite hard (prefer less complex models)

2.2 Model selection for supervised learning

- Model selection comes with two main questions:
  1. How can we estimate the performance of a model on unknown data?
  2. How can we choose the optimal hyperparameters? ⇒ **model selection**
- Common approach for large datasets: split in train, val and test dataset

  Training dataset About 80% of the data should be used for training. On this, we try to minimize the error/loss $L\left(y(\boldsymbol{x}_i), t_i\right)$ for $(\boldsymbol{x}, t) \in D_{train}$ and find optimal parameters $\boldsymbol{w}^*$.

  Validation dataset About 10% of the data is used for estimating the test error $L\left(y(\boldsymbol{x}_{\text{val}}, \boldsymbol{w}^*), t_{\text{val}}\right)$ for various $\boldsymbol{w}^*$ from different hyperparameters. Hence, the hyperparameters are tuned on the validation dataset.

  Testing dataset The last 10% of the available data provides the final test of the chosen best weights and hyperparameters. This data is used to estimate the performance on unseen data, and should therefore not be used for any parameter choosing!
- However, for a small dataset, the validation and test set is very small and, hence, very noisy ⇒ use cross validation

2.2.1 Cross Validation

- Split data into K folds
- If $K = N$, it is also called leave-one-out cross validation as the validation is one single data point
- Train the model y on $K - 1$ folds, and test on the remaining fold k ⇒ model $\hat{y}^{-k}(x)$
- The estimation of the prediction error is the mean validation error over all folds. With the index function $\kappa : \{1, ..., N\} \mapsto \{1, ..., K\}$ (mapping data point to corresponding fold where it is used for validation), we get:

$$CV(\hat{y}) = \frac{1}{N} \sum_{i=1}^{N} L\left(\hat{y}^{-\kappa(i)}(\boldsymbol{x}_i), t_i\right)$$

- Task of model selection: Run cross validation for each possible parameter setting and choose the one with lowest cross validation error
- Task of test error estimation: after finding the best hyperparameters like α^*, retrain model on all K folds, and test this model on a held-out test set
- However, if test set is small, we again get a noisy estimation $\Rightarrow$ Nested cross validation
- Drawback of cross validation: it is computationally expensive and should therefore only be used for fast trainings/small datasets

2.2.2 Nested Cross Validation

- Cross validation for both model selection and model performance by reusing dataset for testing
- General algorithm:
  1. Split dataset into M cross validation folds
  2. For each of these folds $m = 1, ..., M$:
     (a) Let fold m be the test dataset
     (b) Apply cross validation on the remaining data by splitting it into K folds and find best hyperparameters $\alpha^*, \beta^*, ...$
     (c) Retrain the model with the best hyperparameters on all data besides the fold m
     (d) Test the model on unseen data fold m
  3. The final generalization error/loss on unseen data is the mean over all M folds
- For choosing the best hyperparameters $\alpha^*, \beta^*, ...$, we use single cross validation on the whole dataset again without a test dataset, but record the found generalization error as estimation for unknown data, also for the new model

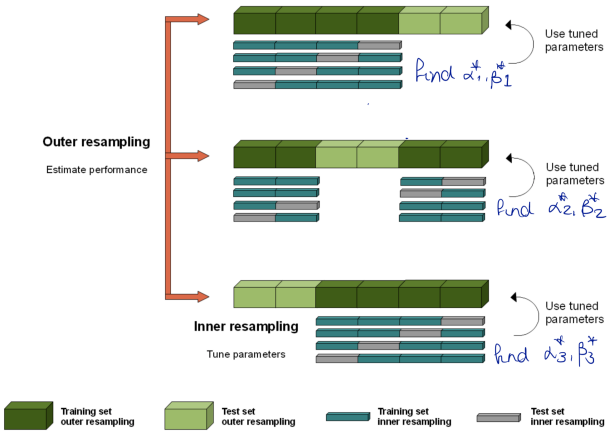


Figure 1: Illustration of nested cross validation. The outer loop splits dataset into test and trainval parts. Within the trainval parts, we apply cross validation to find optimal hyperparameters. Those are tested on the left-out fold from the outer loop, and the mean test error of all folds is the final generalization error. Note that every outer fold can lead to different optimal hyperparameters.

2.3 Bias variance decomposition

- Frequentist view on model complexity
- Common loss: the squared loss function, defined as $L(t, y(\boldsymbol{x})) = (t - y(\boldsymbol{x}))^2$
- An optimal model of $y(\boldsymbol{x})$ would minimize this loss which is given by

$$h(\boldsymbol{x}) = \mathbb{E}[t|\boldsymbol{x}] = \int t \cdot p(t|\boldsymbol{x}) \, \mathrm{d}t$$

  where the conditional distribution $p(t|\boldsymbol{x})$ is the actual, noisy data distribution (not known!)

- Thus, the expected squared loss can be written as

$$\mathbb{E}[L] = \int \underbrace{\{y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}]\}^2}_{\text{model loss}} p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} + \int \underbrace{\{\mathbb{E}[t|\boldsymbol{x}] - t\}^2}_{\text{intrinsic noise on data}} p(\boldsymbol{x}, t)\mathrm{d}\boldsymbol{x}\,\mathrm{d}t$$

  where the first term, the model loss, depends on how different the model $y(\boldsymbol{x})$ is from the actual data distribution, and the second term arises from the intrinsic noise and represents the minimum achievable expected loss

- In Bayesian approach, we would model $y(\boldsymbol{x}, \boldsymbol{w})$ where the uncertainty of $\boldsymbol{w}$ is expressed in the posterior distribution

- However, from a frequentist viewpoint, we use multiple datasets $\mathcal{D}$ on which we train our model and get a single estimation $\hat{\boldsymbol{w}}$ for each of them. The final model is the average over this ensemble of datasets.

- To apply this approach, we take the model loss for a single input $\boldsymbol{x}$, and add the expected model over all datasets:

$$\begin{aligned}\{y(\boldsymbol{x};\mathcal{D}) - h(\boldsymbol{x})\}^2 &= \{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2 \\ &= \{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2 \\ &\quad + 2\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}\end{aligned}$$

- The final model of the frequentist approach is the expected value of this loss over all datasets:

$$\mathbb{E}_{\mathcal{D}}\left[\{y(\boldsymbol{x};\mathcal{D}) - h(\boldsymbol{x})\}^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}^2\right]}_{\text{variance}} + \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2}_{(\text{bias})^2}$$

  where the first term is the **variance** of a model trained on a single dataset compared to the average, and the second term is the loss of the average/expected model over all datasets, or rather the **bias** of the model. The third term of the original equation is eliminated as only $y(\boldsymbol{x};\mathcal{D})$ is affected by the expectation operator $\mathbb{E}_{\mathcal{D}}$, and is the same as $\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]$.

- Coming back to the original expected squared loss, we can decompose it into three terms:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

  where

$$\begin{aligned}(\text{bias})^2 &= \int \{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2 p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \\ \text{variance} &= \int \mathbb{E}_{\mathcal{D}}\left[\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}^2\right] p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \\ \text{noise} &= \int \{h(\boldsymbol{x}) - t\}^2 p(\boldsymbol{x}, t)\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t\end{aligned}$$

- Now, the task is to find the best balance between bias and variance. An example for the data distribution $\mathbb{E}[t|x] = \sin(2\pi x)$ (note that noise is canceled by expectation), 24 Gaussian basis functions and regularized loss function is shown in Figure 2.

- Plotting the terms of the decomposed squared loss function over λ gives further insights of the model behavior (see Figure 3). For generating such a plot, the integrals are approximated by sums over all data points x as we have a limited number of samples.

- In conclusion, high values of λ reduce model complexity, and therefore increase bias loss and leads to underfitting. However, it provides a small variance.
  In contrast, small values of λ causes a low bias as the model is quite complex. Still, the variance is high indicating that the model overfits on the small datasets.

- The bias-variance decomposition is less practical as it is better to train on one large dataset instead of splitting it into several small ones. Furthermore, this reduces the risk of overfitting for a high model complexity on the data anyways.

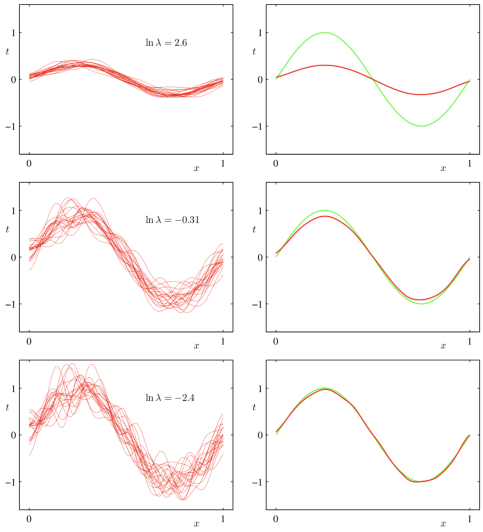


Figure 2: Illustration of dependence of bias and variance on model complexity controlled by λ. Less complex models (high λ) tend to have a high bias (be far off the correct distribution) but it is more robust regarding the actual dataset (therefore, a low variance). Decreasing λ results in a lower bias, but a high variance as models tend to overfit and are therefore sensitive to the dataset.

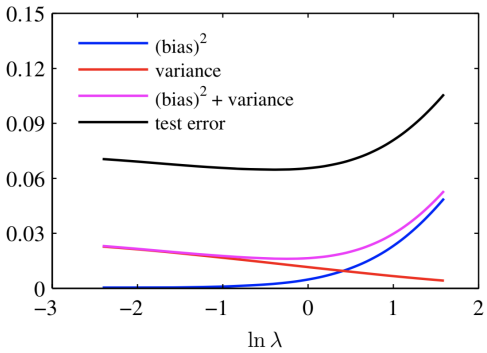


Figure 3: Plot of decomposed loss function for example of Figure 2. The goal is to minimize the test error. It is common that this is close to the minimum value of $(\text{bias})^2 + \text{variance}$. High variance as on the left indicates overfitting, high bias error on the left shows that the model is underfitting.

2.4 Bayesian Linear Regression

- Determining the suitable model complexity using the training data alone without overfitting
- Result is a distribution of $\boldsymbol{w}$ instead of single value as in maximum likelihood or posterior

2.4.1 Parameter distribution

- Prior over weights: $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_0, \boldsymbol{S}_0)$
- Likelihood: $p(t'|\boldsymbol{x}', \boldsymbol{w}, \beta) = \mathcal{N}\left(t'|\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{w}, \beta^{-1}\right)$
- Posterior distribution: $p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{X}) = \frac{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w},\beta)p(\boldsymbol{w})}{p(\boldsymbol{t}|\boldsymbol{X},\beta)} = \mathcal{N}(\boldsymbol{m}_N, \boldsymbol{S}_N)$, where

$$\boldsymbol{S}_N^{-1} = \boldsymbol{S}_0^{-1} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}$$
$$\boldsymbol{m}_N = \boldsymbol{S}_N\left(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\boldsymbol{\Phi}^T\boldsymbol{t}\right)$$

- Maximum a posteriori corresponds by $\boldsymbol{w}_{\text{MAP}} = \boldsymbol{m}_N$
- If no prior was given ($\boldsymbol{S}_0 = \alpha^{-1}\boldsymbol{I}$ with $\alpha \to 0$) the mean $\boldsymbol{m}_N$ reduces to $\boldsymbol{w}_{\text{ML}}$
- Mostly simpler Gaussian prior used: $p(\boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}\right)$

– Resulting parameters of posterior:

$$\boldsymbol{S}_N^{-1} = \alpha^{-1}\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

$$\boldsymbol{m}_N = \beta\boldsymbol{S}_N\boldsymbol{\Phi}^T\boldsymbol{t}$$

$$p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{X}) = \frac{1}{\sqrt{(2\pi)^M\,|\boldsymbol{S}_N|}}\exp\left[-\frac{1}{2}\left(\boldsymbol{w}-\boldsymbol{m}_N\right)^T\boldsymbol{S}_N^{-1}\left(\boldsymbol{w}-\boldsymbol{m}_N\right)\right]$$

– Corresponding log posterior:

$$\ln p\left(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{X}\right) = -\frac{\beta}{2}\sum_{n=1}^{N}\left\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right\}^2 - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + C$$

– Thus, maximizing this posterior is equal to having a regularization term with $\lambda = \frac{\alpha}{\beta}$

– Infinitely narrow prior by $\alpha \to \infty$ ($\alpha \to 0$ seen before ends up in maximum likelihood):

$$\lim_{\alpha\to\infty}\boldsymbol{S}_N = \lim_{\alpha\to\infty}\left(\alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1} = \lim_{\alpha\to\infty}\alpha^{-1}\boldsymbol{I} = \boldsymbol{0}$$

$$\lim_{\alpha\to\infty}\boldsymbol{m}_N = \lim_{\alpha\to\infty}\beta\left(\alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} = \lim_{\alpha\to\infty}\frac{\beta}{\alpha}\boldsymbol{\Phi}^T\boldsymbol{t} = \boldsymbol{0} = \boldsymbol{m}_0$$

– Infinite data $N \to \infty$:

$$\lim_{N\to\infty}\boldsymbol{S}_N = \lim_{N\to\infty}\left(\alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1} = \lim_{N\to\infty}\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1} = \boldsymbol{0}$$

$$\lim_{N\to\infty}\boldsymbol{m}_N = \lim_{N\to\infty}\beta\left(\alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} = \lim_{N\to\infty}\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} = \boldsymbol{w}_{\text{ML}}$$

$\Rightarrow$ At infinite data, all approaches agree: $\boldsymbol{m}_N = \boldsymbol{w}_{\text{ML}} = \boldsymbol{w}_{\text{MAP}}$

2.4.2 Sequential Bayesian Learning

- Data is sequences of input x and target t
- Posterior after $N-1$ data points constitutes the prior for the Nth data point!
- Posterior 1: $p(\boldsymbol{w}|x_1,t_1,\alpha,\beta) \propto p(t_1|x_1,\boldsymbol{w},\beta)p(\boldsymbol{w}|\alpha)$
- Posterior 2: $p\left(\boldsymbol{w}|(x_1,t_1),(x_2,t_2),\alpha,\beta\right) \propto p(t_2|x_2,\boldsymbol{w},\beta)p(\boldsymbol{w}|x_1,t_1,\alpha,\beta)$
- Posterior narrows down step by step until it gets very certain of the correct estimation

2.4.3 Predictive Distribution

- Predictive distribution is defined by ($\mathtt{t}$ targets in training set):

$$p\left(t|x,\mathtt{t},\boldsymbol{X},\alpha,\beta\right) = \int p\left(t|x,\boldsymbol{w},\beta\right)p\left(\boldsymbol{w}|\mathtt{t},\boldsymbol{X},\alpha,\beta\right)d\boldsymbol{w}$$

  where $p\left(t|x,\boldsymbol{w},\beta\right) = \mathcal{N}\left(t|y\left(\boldsymbol{x},\boldsymbol{w}\right),\beta^{-1}\right)$ is the conditional distribution of target variable, and $p\left(\boldsymbol{w}|\mathtt{t},\boldsymbol{X},\alpha,\beta\right) = \mathcal{N}\left(\boldsymbol{w}|\boldsymbol{m}_N,\boldsymbol{S}_N\right)$ the posterior weight distribution

- Predictive distribution is convolution of two Gaussians $\Rightarrow p\left(t|x,\mathtt{t},\boldsymbol{X},\alpha,\beta\right) = \mathcal{N}\left(t|\boldsymbol{m}_N^T\boldsymbol{\phi}(\boldsymbol{x}),\sigma_N^2(\boldsymbol{x})\right)$ where variance $\sigma_N^2(\boldsymbol{x}) = \frac{1}{\beta} + \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x})$ (first term data noise, second weight uncertainty, which goes to 0 for infinite data $N \to \infty$)
- Important points
  1. Uncertainty is smaller near training points
  2. Variance/uncertainty decreases with larger N
- The predictive distribution can be expressed by a **kernel formulation**:

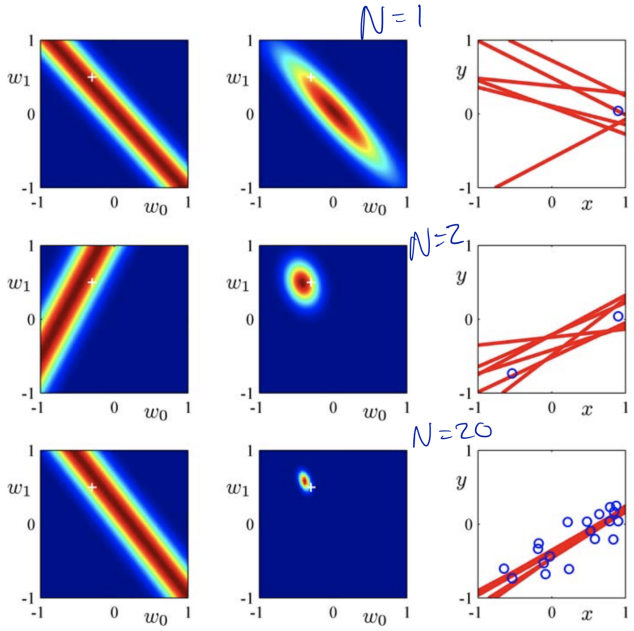


Figure 4: Example for Sequential Bayesian Learning on target $t = -0.3 + 0.5x + \epsilon$. First column: likelihood (not normalized for $\boldsymbol{w}$, but for t_n!), second column: posterior, third column: sampled weights

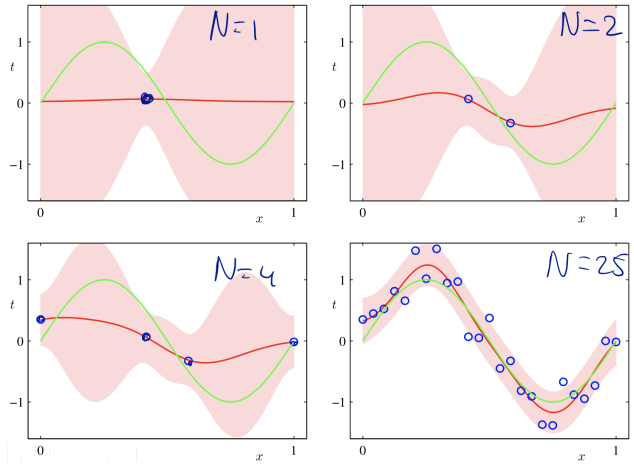


Figure 5: Example for predictive distributions. Green: ground truth data, blue: data points, red line: mean prediction, red area: 1-sigma area

– The predictive mean is

$$\begin{aligned} y\left(\boldsymbol{x}', \boldsymbol{m}_N\right) &= \boldsymbol{\phi}^T(\boldsymbol{x}')\boldsymbol{m}_N = \beta\boldsymbol{\phi}^T(\boldsymbol{x}')\boldsymbol{S}_N\boldsymbol{\Phi}^T\boldsymbol{t} \\ &= \beta\boldsymbol{\phi}^T(\boldsymbol{x}')\boldsymbol{S}_N\sum_{n=1}^{N}\boldsymbol{\Phi}_{:,n}^T t_n = \beta\sum_{n=1}^{N}\boldsymbol{\phi}^T(\boldsymbol{x}')\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_n)t_n \\ &= \sum_{n=1}^{N} k\left(\boldsymbol{x}', \boldsymbol{x}_n\right) t_n \qquad \text{where} \qquad k\left(\boldsymbol{x}', \boldsymbol{x}_n\right) = \beta\boldsymbol{\phi}^T(\boldsymbol{x}')\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_n) \end{aligned}$$

  $\Rightarrow$ Prediction is a linear combination of training set target values

– Kernel values depend on whole dataset by $\boldsymbol{S}_N$

– Closer data points to $\boldsymbol{x}'$ are given a higher weight than points further removed from $\boldsymbol{x}'$

– Thus, local evidence is weighted more strongly than distant evidence

– Kernel can also express covariance:

$$\begin{aligned}\text{cov}\left[t_1, t_2|\boldsymbol{x}_1, \boldsymbol{x}_2\right] &= \text{cov}_{\boldsymbol{w}}\left[y(\boldsymbol{x}_1, \boldsymbol{w}), y(\boldsymbol{x}_2, \boldsymbol{w})\right] = \text{cov}_{\boldsymbol{w}}\left[\boldsymbol{\phi}^T(\boldsymbol{x}_1)\boldsymbol{w}, \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2)\right] \\ &= \mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{\phi}^T(\boldsymbol{x}_1)\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2)\right] - \mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{\phi}^T(\boldsymbol{x}_1)\boldsymbol{w}\right]\mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2)\right] \\ &= \boldsymbol{\phi}^T(\boldsymbol{x}_1)\text{cov}\left[\boldsymbol{w}, \boldsymbol{w}\right]\boldsymbol{\phi}^T(\boldsymbol{x}_2) = \boldsymbol{\phi}^T(\boldsymbol{x}_1)\boldsymbol{S}_N\boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ &= \frac{1}{\beta}k\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right)\end{aligned}$$

– Based on that, we can see that predictive mean at nearby points will be highly correlated (high values of the kernel), and smaller for distant points

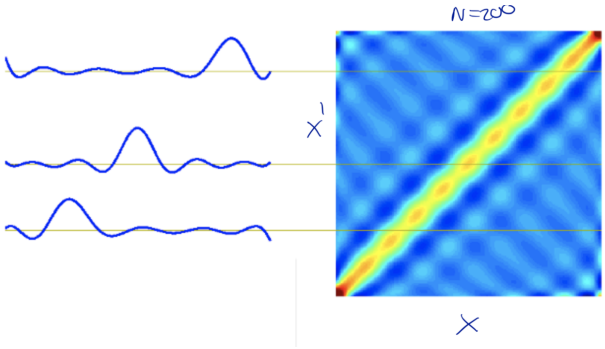


Figure 6: Right plot: matrix for (x', x) of kernel $k\left(x', x\right)$ for Gaussian basis function. Left plot: slices of this matrix for different values of x

2.5 Bayesian Model Comparison

- By marginalizing (integrating) over the model parameters instead of making point estimates of their values, models can be directly compared on the training data instead of separate validation data
- Compare L models $\{\mathcal{M}_i\}_{i=1}^L$
- Probabilities are used to represent uncertainty in the choice of model.
- We express our preference for different models by a prior distribution $p\left(\mathcal{M}_i\right)$, so that the posterior is:

$$p\left(\mathcal{M}_i|\mathcal{D}\right) \propto p\left(\mathcal{M}_i\right)p\left(\mathcal{D}|\mathcal{M}_i\right)$$

- Important term is the *model evidence* $p\left(\mathcal{D}|\mathcal{M}_i\right)$ which updates our preference based on the seen data $\mathcal{D}$. Marginalizes the parameters $\boldsymbol{w}$ of a model:

$$p\left(\mathcal{D}|\mathcal{M}_i\right) = \int p\left(\mathcal{D}|\boldsymbol{w}, \mathcal{M}_i\right)p\left(\boldsymbol{w}|\mathcal{M}_i\right)d\boldsymbol{w}$$

  – Can be viewed as the probability that $\mathcal{D}$ is generated by a random sample of $\boldsymbol{w}$ from the prior.
  – Is also the normalization constant for $p\left(\boldsymbol{w}|\mathcal{D}, \mathcal{M}_i\right)$

- Two models can be compared by dividing their posteriors:

$$\frac{p\left(\mathcal{M}_1|\mathcal{D}\right)}{p\left(\mathcal{M}_2|\mathcal{D}\right)} = \frac{p\left(\mathcal{M}_1\right)p\left(\mathcal{D}|\mathcal{M}_1\right)}{p\left(\mathcal{M}_2\right)p\left(\mathcal{D}|\mathcal{M}_2\right)} \quad \text{where} \quad \frac{p\left(\mathcal{D}|\mathcal{M}_1\right)}{p\left(\mathcal{D}|\mathcal{M}_2\right)} \text{ is called } \textit{Bayes factor}$$

- The predictive distribution is a weighted mean (based on the model probabilities) of our models:

$$p\left(t'|\boldsymbol{x}', \mathcal{D}\right) = \sum_{i=1}^{L} p\left(t'|\boldsymbol{x}', \mathcal{M}_i, \mathcal{D}\right)p\left(\mathcal{M}_i|\mathcal{D}\right)$$

- However, a simple approximation is using the single most probable model alone to make prediction $\Rightarrow$ also known as *model selection*

2.5.1 Approximated Model Evidence

- For a single parameter w, assume that posterior distribution $p(w|\mathcal{D}, \mathcal{M}_i)$ is sharply peaked around the most probably value w_{MAP} with width $\Delta w_{\text{posterior}}$
- Further, we assume that also the prior is a flat distribution with width Δw_{prior} so that $p(w|\mathcal{M}_i) = 1/\Delta w_{\text{prior}}$
- Integral of model evidence can be approximated by its maximum value times the width of the peak:

$$p(\mathcal{D}|\mathcal{M}_i) = \int p(\mathcal{D}|\boldsymbol{w}, \mathcal{M}_i)\, p(\boldsymbol{w}|\mathcal{M}_i)\, d\boldsymbol{w} \simeq p(\mathcal{D}|w_{\text{MAP}}, \mathcal{M}_i)\, \frac{\Delta w_{\text{posterior}}}{\Delta w_{\text{prior}}}$$

- Taking the log leads to:

$$\ln p(\mathcal{D}|\mathcal{M}_i) \simeq \underbrace{\ln p(\mathcal{D}|w_{\text{MAP}}, \mathcal{M}_i)}_{\text{model fit}} + \underbrace{\ln \frac{\Delta w_{\text{posterior}}}{\Delta w_{\text{prior}}}}_{\text{complexity penalty}}$$

- The first term is the likelihood of the data, and therefore describes how good the model fits to the given data (optimal: maximized)
- The second term penalizes model complexity as if $\Delta w_{\text{posterior}} < \Delta w_{\text{prior}}$ (distribution was finely tuned to the data), the term is negative and reduces the model evidence (optimal: minimized)
- Hence, model evidence favors models where we have a trade-off between model fit and complexity

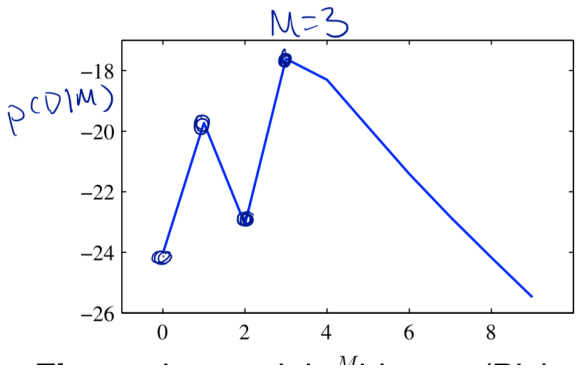


Figure 7: Plotting the curve of $\ln p(\mathcal{D}|\mathcal{M}_i)$ for different polynomials $M = 0, 1, \ldots$ for the task of fitting a sine. As the sine is an odd function, polynomials of odd order fit the best (give the most improvement for the model fit). However, increasing the model complexity increases the penalty.

- For a model with K parameters, we get a similar approximation:

$$\ln p(\mathcal{D}|\mathcal{M}_i) \simeq \ln p(\mathcal{D}|\boldsymbol{w}_{\text{MAP}}, \mathcal{M}_i) + K \ln \frac{\Delta w_{\text{posterior}}}{\Delta w_{\text{prior}}}$$

- Drawbacks of Bayesian approach:
  - Still need to make assumptions about possible models
  - If no model is suitable for the data, the algorithm gives bad estimations
  - Model evidence is sensitive regarding the prior
  - Thus, a small test set is commonly used for Bayesian comparison

2.5.2 Model Evidence for Linear Basis Models

- In fully Bayesian treatment, we must also consider all hyperparameters:

$$\begin{aligned} p(\boldsymbol{t}|\boldsymbol{X}, \mathcal{M}_i) &= \int\int\int p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta, \mathcal{M}_i)\, p(\boldsymbol{w}|\alpha)\, p(\alpha, \beta|\mathcal{M}_i)\, d\boldsymbol{w}\, d\alpha\, d\beta \\ &= \int\int \underbrace{p(\boldsymbol{t}|\boldsymbol{X}, \beta, \alpha, \mathcal{M}_i)}_{\text{peaked posterior/prior}}\, \underbrace{p(\alpha, \beta|\mathcal{M}_i)}_{\text{broad hyperprior}}\, d\alpha\, d\beta \end{aligned}$$

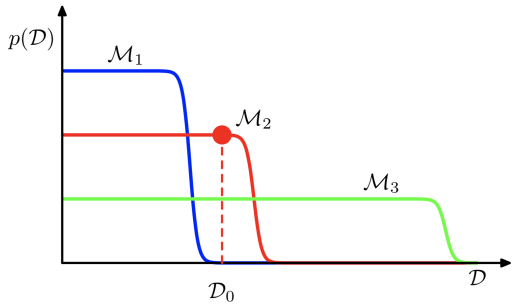


Figure 8: Illustration of three different models and there corresponding model evidences. Horizontal axis x: one dimensional representation of all possible datasets; Vertical axis y: probability that these models generate this specific dataset based on their prior distribution of parameters $\boldsymbol{w}$. $\mathcal{M}_1$ is the simplest model and is therefore only able to create a small set of different data $\mathcal{D}$. As the probability is normalized over all datasets $\mathcal{D}$, the probability is higher than for more complex models like $\mathcal{M}_2$ and $\mathcal{M}_3$. Given a certain dataset $\mathcal{D}_0$, we choose the model with the highest probability $\Rightarrow$ model which just is enough complex to generate this dataset

- Note that the hyperprior can again contain new hyperparameters, for which one might have to define a new prior (and so on)
- Approximation: take best hyperparameters α^* and β^*

$$p\left(\boldsymbol{t}|\boldsymbol{X}, \mathcal{M}_i\right) = \arg\max_{\alpha,\beta} p\left(\boldsymbol{t}|\boldsymbol{X}, \beta, \alpha, \mathcal{M}_i\right)$$

- Using this approximation, we come to following predictive distribution:

$$p\left(t'|\boldsymbol{x}', \boldsymbol{t}, \boldsymbol{X}, \mathcal{M}_i^*\right) \approx p\left(t'|\boldsymbol{x}', \boldsymbol{t}, \boldsymbol{X}, \beta^*, \alpha^*, \mathcal{M}_i\right)$$

2.6 Limitations of fixed basis functions

Advantages

+ Closed form solution for least-squares problem
+ Tractable Bayesian treatment
+ Nonlinear models mapping input variables to target variables through basis functions

Limitations

- Assumption: Basis functions $\phi_j(\boldsymbol{x})$ are fixed, not learned
- *Curse of dimensionality*: to cover growing dimensions D of input vectors, the number of basis functions needs to grow rapidly / exponentially

3 Linear classification

- Input $\boldsymbol{x} = (x_1, x_2, ..., x_D)^T$ with $\boldsymbol{x} \in \mathbb{R}^D$.
- Target $t \in \{C_1, C_2, ..., C_K\}$ with K classes (one-hot representation)
- Goal: divide input space $\mathbb{R}^D$ into K decision regions R_k with $k = 1, ..., K$
- Boundaries of decision regions are called *decision boundaries/surfaces*
  - Linear classification only considers *linear* decision boundaries $\Rightarrow D-1$ dimensional hyperplanes
  - A dataset is *linearly separable*, if its classes can be exactly separated by linear decision boundaries
- First, we derive the optimal solution for decision boundaries in general (3.1 Decision Theory), and then look at different models for deriving such solutions (3.2-3.4)

3.1 Decision Theory For Classification

- For every observed datapoint: label/ground truth $t_n = C_j$, prediction $t_n = C_k$
- Confusion matrix (row: GT class, columns: prediction region/class)

$$\begin{array}{c} \\ C_1 \\ C_2 \\ \vdots \\ C_K \end{array} \begin{array}{c} \begin{array}{cccc} R_1 & R_2 & \dots & R_K \end{array} \\ \begin{pmatrix} 6 & 1 & \dots & 0 \\ 4 & 2 & \dots & 3 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \dots & 5 \end{pmatrix} \end{array}$$

  - The elements on the diagonal represent the correctly classified examples
  - Try to minimize misclassified examples (off-diagonal elements), or the probability of a mistake: $p\,(\text{mistake}) = 1 - \sum_{k=1}^{K} p(\boldsymbol{x} \in R_k, C_k)$
- Assign x to class C_k if $\forall j \neq k : p\,(\boldsymbol{x}, t = C_k) > p\,(\boldsymbol{x}, t = C_j) \Rightarrow p\,(C_k|\boldsymbol{x}) > p\,(C_j|\boldsymbol{x})$
- Optimal decision boundary where $p\,(C_k|\boldsymbol{x}) = p\,(C_j|\boldsymbol{x})$
- Problem: *class imbalance* $\Rightarrow$ possible solution: weighted loss for balancing the importance of each class
- For imbalanced datasets, assign x to C_k if $\sum_{j=1}^{K} L_{jk} p\,(x, C_j)$ is minimal
  - L_{jk} is misclassification weight matrix where $L_{ii} = 0$
  - Example for dataset with 1% cancer patients:

$$L = \begin{array}{c} \begin{array}{cc} \text{pred. cancer} & \text{pred. healthy} \end{array} \\ \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix} \end{array} \begin{array}{l} \text{true cancer} \\ \text{true healthy} \end{array}$$

3.2 Probabilistic generative models

- Model the class conditional densities $p\,(x|C_k)$ **and** the prior class probabilities $p(C_k)$ to compute posterior probabilities $p\,(C_k|x)$ (as we know from Decision Theory that at $p\,(C_k|x) = p\,(C_j|x)$ are the optimal decision boundaries)
- For $K = 2$, the posterior is: $p\,(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_1)p(C_1)+p(\boldsymbol{x}|C_2)p(C_2)}$
- We can simplify the previous equation by using the sigmoid function:

$$p\,(C_1|\boldsymbol{x}) = \frac{1}{1 + \frac{p(\boldsymbol{x}|C_2)p(C_2)}{p(\boldsymbol{x}|C_1)p(C_1)}} = \frac{1}{1 + \exp\,(-a)} \quad \text{where} \quad a = \ln \frac{\sigma}{1 - \sigma} = \ln \frac{p\,(\boldsymbol{x}|C_2)\,p\,(C_2)}{p\,(\boldsymbol{x}|C_1)\,p\,(C_1)}$$

- For general K: $p\,(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{\sum_{j=1}^{K} p(\boldsymbol{x}|C_j)p(C_j)} = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$ with $a_k = \ln\left[p\,(\boldsymbol{x}|C_k)\,p\,(C_k)\right]$ (softmax)
- In the special case of $K = 2$: $a = a_1 - a_2$

3.2.1 Continuous inputs

- Assume that the class-conditional densities are Gaussian:

$$p\,(\boldsymbol{x}|C_k) = \mathcal{N}\,(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left\{\frac{1}{2}\,(\boldsymbol{x} - \boldsymbol{\mu}_k)^T\,\boldsymbol{\Sigma}^{-1}\,(\boldsymbol{x} - \boldsymbol{\mu}_k)\right\}$$

- We assume that all classes share the same covariance: $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$
  $\Rightarrow$ We are able to apply **linear discriminant analysis** (otherwise, decision boundaries would be quadratic)

- Determining posterior for $K = 2$:

$$a = \ln \frac{p(\boldsymbol{x}|C_2)\, p(C_2)}{p(\boldsymbol{x}|C_1)\, p(C_1)} = \ln \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) - \ln \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) + \ln \frac{p(C_1)}{p(C_2)}$$
$$= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 + \ln \frac{p(C_1)}{p(C_2)}$$

- Thus, the posterior can be expressed by

$$p(C_1|\boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T x + w_0\right) \quad \text{where} \quad \boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$
$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 + \ln \frac{p(C_1)}{p(C_2)}$$

- For general K, we get $p(C_k|\boldsymbol{x}) = \frac{\exp(a_k(\boldsymbol{x}))}{\sum_{j=1}^{K} \exp(a_j(\boldsymbol{x}))}$ with

$$a_k(\boldsymbol{x}) = \ln\left[p(\boldsymbol{x}|C_k) \cdot p(C_k)\right] = \boldsymbol{w}_k^T x + w_{k0} \quad \text{where} \quad \boldsymbol{w}_k = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k$$
$$w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \ln p(C_k)$$

- Decision boundaries are at $p(C_k|\boldsymbol{x}) = p(C_j|\boldsymbol{x}) \Rightarrow a_k = a_j \Rightarrow$ Linear decision boundaries!

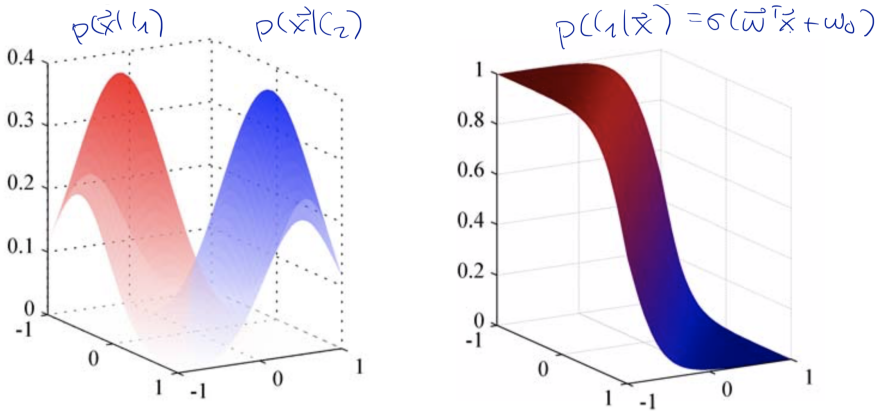


Figure 9: Left: Gaussian class-conditional densities, Right: corresponding posterior with sigmoid

3.2.2 Maximum likelihood solution for $K = 2$

- Binary targets $t_n \in \{0, 1\}$ (1 for C_1, 0 for C_0)
- We use maximum likelihood to find optimal solution for $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}$ and priors $p(C_k)$
- For $K = 2$, the priors are denoted by $p(C_1) = q$ and $p(C_2) = 1 - q$
- If $\boldsymbol{x}_n$ has target $t_n = 1$: $p(\boldsymbol{x}_n, C_1) = p(\boldsymbol{x}_n|C_1)\, p(C_1) = q\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$
- If $\boldsymbol{x}_n$ has target $t_n = 0$: $p(\boldsymbol{x}_n, C_2) = p(\boldsymbol{x}_n|C_2)\, p(C_2) = (1 - q)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$
- Combined likelihood: $p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \left[q\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right]^{t_n} \left[(1 - q)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\right]^{1 - t_n}$
- Log-likelihood:

$$\ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} t_n \ln q + t_n \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) + (1 - t_n) \ln(1 - q) + (1 - t_n) \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$$

- Estimate for q: $\frac{\partial}{\partial q} \ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \frac{t_n}{q} - \frac{1 - t_n}{1 - q} = \sum_{n=1}^{N} \frac{t_n - q}{q(1 - q)} \Rightarrow q_{\text{ML}} = \frac{1}{N} \sum_{n=1}^{N} t_n = \frac{N_1}{N}$
  - Thus, the estimate of $p(C_1)$ is the proportion of samples that are assigned to class 1

- Estimate for $\boldsymbol{\mu_1}$: $\boldsymbol{\mu}_{1,\text{ML}} = \frac{1}{N_1} \sum_{n=1}^{N} t_n \boldsymbol{x}_n$, $\boldsymbol{\mu}_{2,\text{ML}} = \frac{1}{N_2} \sum_{n=1}^{N} (1 - t_n) \boldsymbol{x}_n$

  - Thus, the estimate of $\boldsymbol{\mu}_k$ is the mean of the samples assigned to class k

- Estimate for $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma}_{\text{ML}} = \frac{N_1}{N} \underbrace{\left[\frac{1}{N_1} \sum_{n=1}^{N} t_n \left(\boldsymbol{x} - \boldsymbol{\mu}_{1,\text{ML}} \right) \left(\boldsymbol{x} - \boldsymbol{\mu}_{1,\text{ML}} \right)^T \right]}_{\text{sample covariance of class 1}} + \frac{N_2}{N} \underbrace{\left[\frac{1}{N_2} \sum_{n=1}^{N} (1 - t_n) \left(\boldsymbol{x} - \boldsymbol{\mu}_{2,\text{ML}} \right) \left(\boldsymbol{x} - \boldsymbol{\mu}_{2,\text{ML}} \right)^T \right]}_{\text{sample covariance of class 2}}$$

  - Thus, the estimate of $\boldsymbol{\Sigma}$ is a weighted average (based on number of samples for each class) of the class' sample covariance
  - Note that this assumes a similar covariance matrix for every class cluster. If this is not the case, the estimation gives bad results (see Figure 10)

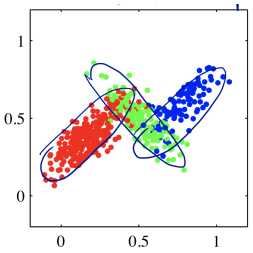


Figure 10: Example for three classes with different covariance matrices $\boldsymbol{\Sigma}_k^{-1}$. Linear discriminant analysis fails as it estimates a weighted sum of the sample covariance, and the distribution of green class significantly differs from the other two. The resulting estimate would tend to be a circle for each class instead of the drawn ellipses.

3.2.3 Discrete inputs

- In contrast to the previous subsections, we now assume that $\boldsymbol{x}_n \in \{0, 1\}^D$ and is therefore discrete
- As we know have no PDF anymore, we need $2^D - 1$ parameters per class to guarantee a perfect fit
- However, if we use the Naive Bayes assumption (feature values are treated as independent given C_k), we reduce the number of features to D per class (by using the Bernoulli distribution):

$$p\left(\boldsymbol{x}|C_k\right) = \prod_{i=1}^{D} p\left(x_i|C_k\right) = \prod_{i=1}^{D} \pi_{ki}^{x_i} \left(1 - \pi_{ki}\right)^{1 - x_i}$$

- We can apply this simplification to rewrite a_k:

$$a_k = \ln p\left(x|C_k\right) + \ln p\left(C_k\right) = \sum_{i=1}^{D} \left[x_i \ln \pi_{ki} + (1 - x_i) \ln(1 - \pi_{ki}) \right] + \ln p\left(C_k\right) = \boldsymbol{x}^T \boldsymbol{w} + w_0$$

  where $w_i = \ln \frac{\pi_{ki}}{1 - \pi_{ki}}$ and $w_0 = \ln p\left(C_k\right) + \sum_{i=1}^{D} \ln\left(1 - \pi_{ki}\right)$

3.3 Discriminant functions

- Direct mapping of input to target (similar to regression)
- We use $y\left(\boldsymbol{x}, \tilde{\boldsymbol{w}}\right) = f\left(\tilde{\boldsymbol{w}}^T \boldsymbol{\phi}\right)$, where f is the activation function and might be non-linear
- The decision boundary is defined at a point where $y\left(\boldsymbol{x}, \tilde{\boldsymbol{w}}\right) = \text{const}_1$. As y represents the application of f, we can rewrite it as $\tilde{\boldsymbol{w}}^T \boldsymbol{\phi} = \text{const}_2$
- We first review the application of the case of two classes, and then try to find a solution for multiple classes

3.3.1 Discriminant functions for two classes

- For a two class problem, we set the decision boundary to 0 as we are still able to shift it by w_0
- If $y(\boldsymbol{x}, \tilde{\boldsymbol{w}}) \geq 0$, the input $\boldsymbol{x}$ is assigned to class C_1, whereas if $y(\boldsymbol{x}, \tilde{\boldsymbol{w}}) < 0$, the class is $C_2 \Rightarrow w_0$ is considered as the activation threshold
- To determine how the weights $\tilde{\boldsymbol{w}}$ influence this classification, we assume two points $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ on the decision boundary $\Rightarrow y(\boldsymbol{x}_a) = y(\boldsymbol{x}_b) = 0 \Rightarrow \boldsymbol{w}^T(\boldsymbol{x}_a - \boldsymbol{x}_b) = 0$ (see Figure 11)
- Hence, $\boldsymbol{w}$ is orthogonal to every vector lying within the decision surface, so that $\boldsymbol{w}$ determines the orientation of the surface

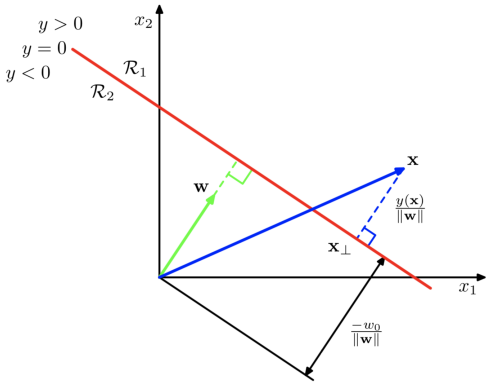


Figure 11: Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\boldsymbol{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\boldsymbol{x}$ from the decision surface is given by $y(\boldsymbol{x})/\boldsymbol{w}$.

- So, we can express every point by the summation of a point on the decision surface and the weights:

$$\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{||\boldsymbol{w}||}$$

- Applied in y, we get: $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = \boldsymbol{w}^T\boldsymbol{x}_\perp + w_0 + r\frac{\boldsymbol{w}^T\boldsymbol{w}}{||\boldsymbol{w}||} = r||\boldsymbol{w}|| \Rightarrow$
- So, the distance between a point $\boldsymbol{x}$ and the decision surface is $r = \frac{y(\boldsymbol{x})}{||\boldsymbol{w}||}$

3.3.2 Discriminant functions for multiple classes

- K-class discriminant: $\boldsymbol{y}_k(\boldsymbol{x}) = \boldsymbol{w}_k^T + w_{k0}$
- Assign $\boldsymbol{x}$ to C_k if $y_k(\boldsymbol{x}) > y_j(\boldsymbol{x})$ for all $j \neq k$
- Thus, the decision boundary between $\mathcal{R}_k$ and $\mathcal{R}_j$ is determined by: $y_k(\boldsymbol{x}) = y_j(\boldsymbol{x})$
- Note that decision regions of linear discriminant functions are convex (if two points are in $\mathcal{R}_k$, then all points between those are also in the same region $\mathcal{R}_k$)

3.3.3 Least squares discriminant

- Consider $\boldsymbol{t}_n$ as one-hot vector. We try to learn a function $y_k(\boldsymbol{x}, \tilde{\boldsymbol{w}}_k)$ for every class k that maps $\boldsymbol{x}$ to its corresponding value in the one-hot vector (basically regression task)
- For shorter notation, we write $\boldsymbol{y}(\boldsymbol{x}) = \tilde{\boldsymbol{W}}\tilde{\boldsymbol{x}}$ to combine all classes and weights into a single operation
- As before: assign $\boldsymbol{x}$ to class C_k if $k = \arg\max_j y_j(\boldsymbol{x})$
- The error function is the sum-of-squares:

$$E_D(\tilde{\boldsymbol{W}}) = \frac{1}{2}\text{Tr}\left[\left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{W}} - \boldsymbol{T}\right)^T\left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{W}} - \boldsymbol{T}\right)\right] = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{d=1}^{D}\left(\tilde{X}_{nd}\tilde{W}_{dk} - \tilde{T}_{nd}\right)$$

- Minimizing this error leads to: $\tilde{\boldsymbol{W}}_{\text{LS}} = \left(\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}}\right)^{-1} \tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{T}}$
- But: there are many problems with least squared errors
  - The decision boundaries are very sensitive to outliers (try to minimizes *mean* error to one-hot vector)
  - For $K > 2$, some decision regions become very small or are even completely ignored (also called masking)
  - components of $\boldsymbol{y}_{\text{LS}}$ are not probabilities and can be outside the interval $[0, 1]$

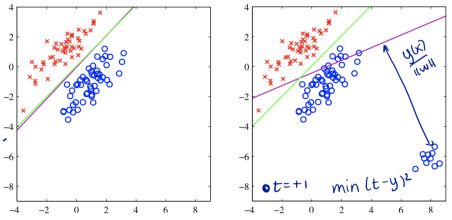


Figure: least squares is very sensitive to outliers (Bishop 4.4)

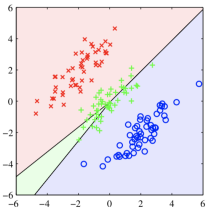


Figure: masking for least squares for K>2 (Bishop 4.5)

Figure 12: Illustration of the problems with least squared error discriminant

3.3.4 Perceptron

- For the perceptron, we use the step function as activation function:

$$y(\boldsymbol{x}) = f\left(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})\right) \qquad \text{where} \qquad f(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}$$

- Thus, assign $\boldsymbol{x}$ to class C_1 if $\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \geq 0$, otherwise C_2
- The goal is now to find a $\boldsymbol{w}$ such that $\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) t_n \geq 0$ ($t_n \in \{1, -1\}$)
- We can define the error of a perceptron based on the set of misclassified examples $\mathcal{M}$:
  $E_P(\boldsymbol{w}) = -\sum_{n \in \mathcal{M}} \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) t_n = \sum_{n \in \mathcal{M}} E_n(\boldsymbol{w})$
- Use Stochastic Gradient Descent (SGD) for each misclassified $\boldsymbol{x}_n$:

$$\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta \bigtriangledown^T E_n(\boldsymbol{w}) = \boldsymbol{w}^{\tau} + \eta \boldsymbol{\phi}(\boldsymbol{x}_n) t_n$$

- If $\boldsymbol{X}$ is linearly separable, SGD will converge
- However, there are some problems with the perceptron algorithm:
  - Perceptron only works for 2 classes
  - There might be many optimal solutions, so that the exact outcome depends on initialization of $\boldsymbol{w}$ and order of data that are used in SGD
  - If dataset is not linearly separable, the perceptron algorithm will not converge
  - Based on linear combination of fixed basis functions

3.3.5 Usage of Basis functions

- If the data in the input space is not linearly separable, we can use basis functions (that might be non-linear) to transform them into a new space, where they can be linearly separated!
- However, prior knowledge is required for this step as the general data distribution must be known and how to convert it into a linearly separable space. This step is especially hard/not possible for high dimensions

3.4 Probabilistic discriminative models

- Instead of specifying the class-conditional probabilities $p(\boldsymbol{x}|C_k)$ and applying maximum likelihood to find the best parameters, we can try to explicitly model the posterior class probability $p(C_k|\boldsymbol{x})$ and find its distribution $\Rightarrow$ posteriors are non-linear functions with a linear function of $\boldsymbol{\phi}$ as input: $p(C_k|\boldsymbol{\phi}, \boldsymbol{w}) = f(\boldsymbol{w}_k^T \boldsymbol{\phi})$
- The implicit method of finding the parameters of a generalized model is by fitting $p(\boldsymbol{x}|C_k)$ and $p(C_k)$ representing a generative model (generate synthetic data from $p(\boldsymbol{x})$)

3.4.1 Logistic regression for two classes

- Logistic regression uses the sigmoid function to model the posterior:

$$p(C_1|\boldsymbol{\phi}, \boldsymbol{w}) = y(\boldsymbol{\phi}) = \sigma\left(\boldsymbol{w}^T \boldsymbol{\phi}\right), p(C_2|\boldsymbol{\phi}, \boldsymbol{w}) = 1 - p(C_1|\boldsymbol{\phi}, \boldsymbol{w})$$

- For inference/classification, take the class with the higher probability (> 0.5) $\Rightarrow$ Decision boundaries: $\boldsymbol{w}\boldsymbol{\phi}(\boldsymbol{x}) = 0$
- If $\boldsymbol{w} \in \mathbb{R}^M$, we use M number of parameters (compared to $M(M+5)/2+1$ for modeling a Gaussian multivariate distribution)
- Use maximum likelihood to determine the parameters of the logistic regression model
- Conditional likelihood: $p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{n=1}^{N} p(t_n|\boldsymbol{x}_n, \boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}$
- Maximizing the likelihood is equal to minimizing the cross-entropy loss:

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}) = -\sum_{n=1}^{N} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)] = \sum_{n=1}^{N} E_n(\boldsymbol{w})$$

- The loss $E(\boldsymbol{w})$ is convex (has a single, **unique minimum**), but no closed-form solution exists ($y_n = \sigma(\boldsymbol{w}^T \phi_n)$ is nonlinear in $\boldsymbol{w}$) $\Rightarrow$ Use SGD/...
- For taking the gradient, we can make use of the property of the sigmoid function:

$$\frac{\partial E_n(\boldsymbol{w})}{\partial w_j} = \frac{\partial E_n(\boldsymbol{w})}{\partial y_n} \frac{\partial y_n}{\partial w_j} = \left[-\frac{t_n}{y_n} + \frac{1 - t_n}{1 - y_n}\right] \cdot \left[\sigma(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)) \left(1 - \sigma(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n))\right) \phi_j(\boldsymbol{x}_n)\right] = (y_n - t_n)\phi_j(\boldsymbol{x}_n)$$

- Update rule (SGD): $\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta \nabla^{\tau} E_n(\boldsymbol{w})^{\tau} = \boldsymbol{w}^{\tau} - \eta(y_n - t_n)\boldsymbol{\phi}(\boldsymbol{x}_n)$
- If η too large: no convergence. If η too small: very slow convergence
- Converged $\boldsymbol{w}^*$ minimizes the loss $E(\boldsymbol{w})$

3.4.2 Iterative reweighted least squares

- Also called the *Newton-Raphson iterative optimization scheme*
- We use a **quadratic approximation** instead of a linear at $E(\boldsymbol{w}^{\tau})$ as the difference between the difference of the loss function to a second order polynomial is quite small (find $E(\boldsymbol{w}^{\tau+1})$ that minimizes our quadratic approximation $\Rightarrow$ no learning rate)
- New update rule:

$$\boldsymbol{w}^{\tau} = \boldsymbol{w}^{\tau-1} - \boldsymbol{H}^{-1} \nabla E(\boldsymbol{w}^{\tau-1})$$

  where $\boldsymbol{H}$ is the Hessian matrix whose elements comprise the second derivatives of $E(\boldsymbol{w})$: $H_{ij} = \frac{\partial E(\boldsymbol{w})}{\partial w_i \partial w_j}$ ($\boldsymbol{H}$ is symmetric!)
- Derived from the previous section, the gradient for all N data-points is

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^T (\boldsymbol{y} - \boldsymbol{t})$$

- The elements of the Hessian derive this by a second parameter again:

$$H_{ij} = \frac{\partial E(\boldsymbol{w})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i} \sum_{n=1}^{N} (y_n - t_n)\phi_j(\boldsymbol{x}_n) = \sum_{n=1}^{N} \phi_j(\boldsymbol{x}_n) \frac{\partial y_n}{\partial w_i} = \sum_{n=1}^{N} y_n(1-y_n)\phi_i(\boldsymbol{x}_n)\phi_j(\boldsymbol{x}_n)$$

- The overall Hessian matrix is therefore

$$\boldsymbol{H} = \sum_{n=1}^{N} y_n(1-y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T = \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi}$$

  where $R_{nn} = y_n(1-y_n)$, and otherwise $R_{nm} = 0$ for $n \neq m$

- Applying this term in the update equation leads to:

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \left(\boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T (\boldsymbol{y} - \boldsymbol{t}) = \left(\boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{z} \quad \text{where} \quad \boldsymbol{z} = \boldsymbol{\Phi}\boldsymbol{w}^{(\tau-1)} - \boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{t})$$

- Note the similarity to the maximum likelihood solution $\boldsymbol{w}_{\text{ML}} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$

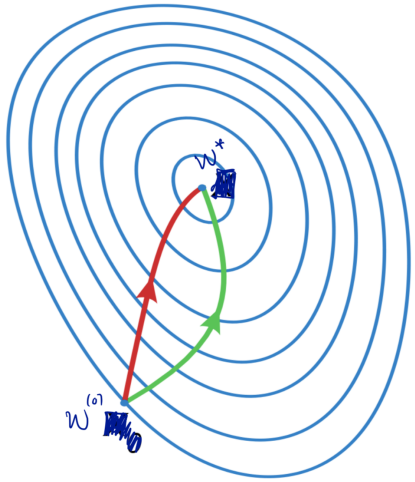


Figure 13: Illustration of SGD (green) and Newton Raphson (red). SGD always goes in the direction of the steepest gradient and is therefore slower than Newton-Raphson.

3.4.3 Logistic regression for multiple classes

- Our posterior distribution is now given by a softmax:

$$p(C_k|\boldsymbol{\phi}, \boldsymbol{w}_1, ..., \boldsymbol{w}_K) = y_k(\boldsymbol{\phi}) = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)} \quad \text{where} \quad a_k = \boldsymbol{w}_k^T \boldsymbol{\phi}(\boldsymbol{x})$$

- The derivation by a element a_j is $\frac{\partial y_k}{\partial a_j} = y_k(\mathbb{I}(k=j) - y_j)$
- We use again maximum likelihood to determine the optimal parameters
- Conditional likelihood $p(\boldsymbol{T}|\boldsymbol{X}, \boldsymbol{w}_1, ...\boldsymbol{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k|x_k, \boldsymbol{w}_1, ..., \boldsymbol{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_k(\boldsymbol{\phi}_n)^{t_{nk}}$
- Taking the negative logarithm gives the *cross-entropy* loss function for the multiclass classification problem

$$E(\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = -\ln p(\boldsymbol{T}|\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

- To minimize the function by SGD or Newton Raphson, we need to take the derivate:

$$\nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\boldsymbol{\phi}(\boldsymbol{x}_n)$$

- For Newton Raphson/Iterative reweighted least squares, we also need the Hessian matrix:

$$\frac{\partial}{\partial \boldsymbol{w}_k}\frac{\partial}{\partial \boldsymbol{w}_j}E\left(\boldsymbol{w}_1,...,\boldsymbol{w}_K\right)=\sum_{n=1}^{N}y_{nk}\left(\mathbb{I}\left(k=j\right)-y_{nj}\right)\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T$$

- Decision boundaries at $\left(\boldsymbol{w}_k^*\right)^T\boldsymbol{\phi}\left(\boldsymbol{x}'\right)=\left(\boldsymbol{w}_j^*\right)^T\boldsymbol{\phi}\left(\boldsymbol{x}'\right)$

4 Neural Networks

- Previously: fixed basis function $\boldsymbol{\phi}(\boldsymbol{x})=\left(\phi_0\left(\boldsymbol{x}\right),\phi_1\left(\boldsymbol{x}\right),...,\phi_M\left(\boldsymbol{x}\right)\right)^T$
- Neural networks: Create flexible non-linear features and learn them.
  - Basis function with extra parameters: $\phi_m\left(\boldsymbol{x},\boldsymbol{w}_m^{(1)}\right)=h\left(\left(\boldsymbol{w}_m^{(1)}\right)^T\boldsymbol{x}\right)=h\left(\sum_{d=0}^{D}w_{md}^{(1)}\right)$
  - Note that $\boldsymbol{x}_n=\left(1,x_{n0},...,x_{nD}\right)^T\Rightarrow\boldsymbol{x}_n\in\mathbb{R}^D$
  - h is the non-linear activation function
- We can define regression for a one-layer neural network:

$$\begin{aligned}y\left(\boldsymbol{x},\boldsymbol{W}^{(1)},\boldsymbol{w}^{(2)}\right)&=\sum_{m=0}^{M}w_m^{(2)}h\left(\sum_{d=0}^{D}w_{md}^{(1)}\right)\\&=\left(\boldsymbol{w}^{(2)}\right)^T h\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\quad\text{where}\quad\boldsymbol{W}^{(1)}=\begin{pmatrix}|&|&\dots&|\\\boldsymbol{w}_0^{(1)}&\boldsymbol{w}_1^{(1)}&\dots&\boldsymbol{w}_D^{(1)}\\|&|&\dots&|\end{pmatrix}\end{aligned}$$

- The same way, we can adjust a network for classification:

$$y\left(\boldsymbol{x},\boldsymbol{W}^{(1)},\boldsymbol{w}^{(2)}\right)=f\left(\sum_{m=0}^{M}w_m^{(2)}h\left(\sum_{d=0}^{D}w_{md}^{(1)}\right)\right)=f\left(\left(\boldsymbol{w}^{(2)}\right)^T h\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\right)$$

  where f is sigmoid for binary and softmax for multi-class classification (then $\boldsymbol{w}^{(2)}$ is $K\times M$ matrix)

4.1 Feed-forward Network Functions

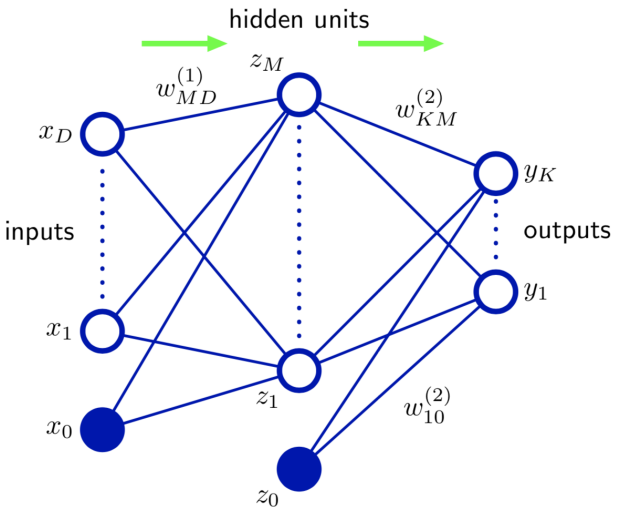


Figure 14: Illustration of a multilayer perceptron (2 layers).

- Input are $D+1$ units whereas $x_0=1$ is the bias
- First layer with $M\times D$ weight matrix $\boldsymbol{W}^{(1)}\Rightarrow M$ activations $a_m=\sum_{d=0}^{D}w_{md}^{(1)}x_d$ and bias $h^{(1)}(a_0)=1$

- We apply an activation function on these activations to get the hidden units: $z_m = h^{(1)}(a_m)$ where $z_0 = 1$
- Second layer with $K \times M$ weight matrix $\boldsymbol{W}^{(2)} \Rightarrow K$ output units $y_k = h^{(2)}\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right)$
- In conclusion, a output unit y_k is calculated as follows:

$$y_k\left(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}\right) = h^{(2)}\left(\sum_{m=0}^{M} w_{km}^{(2)} \cdot h^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)\right)$$

- Alternative notation: $y_k = h^{(2)} \circ \boldsymbol{a}^{(2)} \circ h^{(1)} \circ \boldsymbol{a}^{(1)}(\boldsymbol{x})$
- Additional forms:
  - *Skip connections*: Connection between for instance first and fourth layer
  - *Sparse connections*: For instance convolutions, can have weight sharing
- In general: $z_m = h\left(\sum_j w_{mj} z_j\right)$ where j are all incoming connections
- Note that no closed directed cycles are allowed

4.1.1 Universal approximator

- Let f by any continuous function on a compact area of $\mathbb{R}^D$ and h any fixed analytic function which is not polynomial (e.g. logistic function, tanh function, ...). Given any small number $\epsilon > 0$ of an acceptable error, we can find a number M and weights $w_m^{(2)}$ and $w_{md}^{(1)} \in \mathbb{R}$ such that:

$$\left|f(\boldsymbol{x}) - y\left(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{w}^{(2)}\right)\right| < \epsilon$$

  with y as two-layer NN
- For smaller ϵ we need more hidden units $\Rightarrow$ larger M
- We may also take deeper networks that are usually capable to approximate more complex functions with less units
- To approximate deep network with shallow one by error ϵ, the number of units M needed scales exponentially for decreasing ϵ

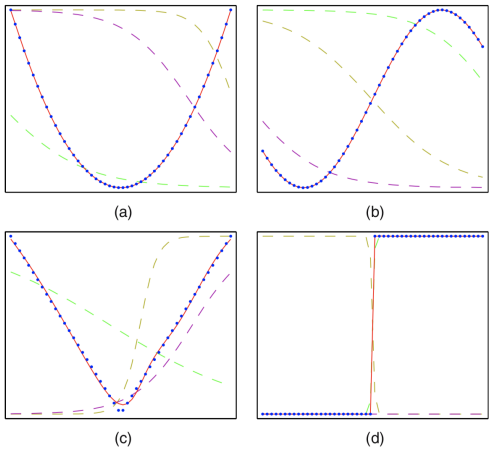


Figure 15: Example approximations by a 2-layer network with 3 hidden units of the function (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c) $f(x) = |x|$ and (d) $f(x) = \mathbb{I}(x > 0)$. The outputs of the three hidden units are shown as dashed lines.

4.2 Network Training

- Use probabilistic interpretation of the network outputs to choose number of outputs, output activation function and loss (e.g. $p(t|\boldsymbol{x}, \boldsymbol{w})$ for regression $\Rightarrow$ maximizing likelihood used as error function)

4.2.1 Network training for regression

- Data input $\boldsymbol{x}_n \in \mathbb{R}^D$ with continuous target $t_n \in \mathbb{R}$
- Single real-valued target $\rightarrow$ Single output unit with identity activation function $y\left(\boldsymbol{x}, \boldsymbol{w}\right) = a^{\text{out}}$
- Derive loss function by maximum likelihood:

$$E(\boldsymbol{w}) = -\ln p\left(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}\right) = \frac{\beta}{2}\sum_{n=1}^{N}\left\{y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n\right\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln 2\pi$$

$$\text{equivalent to minimizing } E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n\right\}^2$$

4.2.2 Network training for binary classification

- Targets are now binary values: $t_n \in \{0, 1\}$
- As $p\left(t = 1|\boldsymbol{x}\right) = 1 - p\left(t = 0|\boldsymbol{x}\right)$, we model only one output unit: $y\left(\boldsymbol{x}, \boldsymbol{w}\right) = p\left(t = 1|\boldsymbol{x}\right)$
- The output activation function is therefore a sigmoid: $y\left(\boldsymbol{x}, \boldsymbol{w}\right) = \sigma\left(a^{\text{out}}\right)$
- The maximum likelihood is here equivalent to minimizing BCE:

$$E\left(\boldsymbol{w}\right) = -\sum_{n=1}^{N} t_n \ln y\left(\boldsymbol{x}_n, \boldsymbol{w}\right) + \left(1 - t_n\right)\ln\left(1 - y\left(\boldsymbol{x}_n, \boldsymbol{w}\right)\right)$$

4.2.3 Network training for classification with K classes

- Targets are now one-hot vectors $\boldsymbol{t}_n = (0, ..., 1, ..., 0)^T$
- Now, we have to model all K class distributions by $y_k\left(\boldsymbol{x}, \boldsymbol{w}\right) = p\left(C_k|\boldsymbol{x}\right)$
- Activation function is softmax: $y_k\left(\boldsymbol{x}, \boldsymbol{w}\right) = \frac{\exp\left(a_k^{\text{out}}\right)}{\sum_{j=1}^{K}\exp\left(a_j^{\text{out}}\right)}$
- The maximum likelihood is here equivalent to:

$$E\left(\boldsymbol{w}\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_k\left(\boldsymbol{x}_n, \boldsymbol{w}\right)$$

4.2.4 Parameter optimization

- Optimal parameters minimize error function: $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w})$
- Problem: $E(\boldsymbol{w})$ is not convex so that many local minima (can) exist
- Different optimization strategies can be developed
- **Gradient Descent** uses full dataset for each update: $\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E\left(\boldsymbol{w}^{(\tau)}\right)$
- Always goes in the direction of steepest gradient
- Will easily get stuck in local minimum when $\nabla E\left(\boldsymbol{w}\right) = 0$
- **Stochastic Gradient Descent** uses single data point or minibatches for the update step: $\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla\sum_{i=1}^{M} E_i\left(\boldsymbol{w}^{(\tau)}\right)$
- Converges to area around local minimum
- More likely to escape local minimum as $\nabla E\left(\boldsymbol{w}^{(\tau)}\right) = 0$ does not imply $\nabla E_n\left(\boldsymbol{w}^{(\tau)}\right) = 0$ for all n
- Is more computational efficient at the beginning as all $E_n\left(\boldsymbol{w}\right)$ will point in a similar direction
- Choose learning rate carefully to get good results
  - If learning rate is too small: slow convergence
  - If learning rate is too high: oscillations around local minimum
  - Use learning rate scheduling with smaller learning rate over time

4.3 Error Backpropagation

- The error function is the sum of single point errors ($E(\boldsymbol{w}) = \sum_n E_n(\boldsymbol{w})$), so that we can calculate the gradients for each data point independently: $\frac{\partial E_n(\boldsymbol{w})}{\partial \boldsymbol{w}}$
- Therefore, we first apply *forward propagation*: calculate all $a_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)}$ and $z_j^{(l)} = h^{(l)}\left(a_j^{(l)}\right)$
- Then, apply *back propagation* by calculating all $\frac{\partial E_n}{\partial w_{ji}^{(l)}}$
- Backpropagation is based on the multi-dimensional chain rule:

$$\frac{\partial f\left(g_1\left(x\right), ..., g_D\left(x\right)\right)}{\partial x} = \sum_{d=1}^{D} \frac{\partial f\left(g_1\left(x\right), ..., g_D\left(x\right)\right)}{\partial g_d(x)} \frac{\partial g_d(x)}{\partial x}$$

- Thus, we can express the gradient regarding a single weight element by (only $a_j^{(l)}$ depends on $w_{ji}^{(l)}$):

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}$$

- The second part of the derivate is just $\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$, the first one we define as $\delta_j^{(l)} \equiv \frac{\partial E_n}{\partial a_j^{(l)}}$
- So, our derivate is $\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$
- $a_j^{(l)}$ effects the error only by its following units $a_k^{(l+1)} \Rightarrow \delta_j^{(l)} = \sum_k \frac{\partial E_n}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{a_j^{(l)}} = \sum_k \delta_k^{(l+1)} \frac{\partial a_k^{(l+1)}}{a_j^{(l)}}$
- As $a_j^{(l)}$ effects $a_k^{(l+1)}$ only by the weight $w_{kj}^{(l+1)}$, the derivate is $\frac{\partial a_k^{(l+1)}}{a_j^{(l)}} = w_{kj}^{(l+1)} h^{(l)'}\left(a_j^{(l)}\right)$
- Note that we need to be careful with skip connections
- Overall, backpropagation can be summarized in three steps:
  1. Compute δ_k for all output units
  2. Compute δ_j for all hidden units through backpropagation:

$$\delta_j^{(l)} = h^{(l)'}\left(a_j^{(l)}\right) \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)}$$

  3. Compute derivatives $\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$
  4. Apply iterative weight update: $w_{ji}^{(l)(\tau+1)} = w_{ji}^{(l)(\tau)} - \eta \delta_j^{(l)} z_i^{(l-1)}$

4.4 Issues with Neural Networks

- Initialization of weights: randomly start near zero such that activations fall into linear part of activation functions (e.g. for tanh and sigmoid) and gradients don't vanish
- Networks perform best when input has mean 0 and variance 1
- When you have a large number of parameters, we need regularization!
- Multiple local minima: Non-convex error function. Restart experiment with different seeds and choose model with lowest regularized error
- Use weight sharing to reflect symmetries in data if possible

5 Unsupervised learning

- We can express our data distribution by marginalizing latent variables (unobserved targets/values that make it easier to understand the data). This allows us to model the data with more tractable joint distributions with simpler components to understand:

$$\boldsymbol{z} \text{ continuous: } p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z})\, d\boldsymbol{z} = \int p(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})\, d\boldsymbol{z}$$
$$\boldsymbol{z} \text{ discrete: } p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})$$

- Discrete latent variables are typically used for clustering, whereas continuous are applied for dimensionality reduction

5.1 *K*-means Clustering

- Every single data point $\boldsymbol{x}$ is assigned to a cluster $\rightarrow$ a discrete latent variable $\boldsymbol{z}$
- Number of clusters/different values for $\boldsymbol{z}$ must be determined beforehand
- Cluster as comprising a group of data points whose inter-point distances are small compared with the distances to points outside the cluster
- Hence, we define $\boldsymbol{\mu}_k$ as a prototype (here also the mean) of the cluster k, and minimize the sum of squares of the distances of each data point to its closest vector $\boldsymbol{\mu}_k$:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

  where $\boldsymbol{z}_n$ is a one-hot vector with $z_{nk} = 1$ if k is closest cluster of $\boldsymbol{x}_n$

- Optimization algorithm (expectation-maximization (EM) algorithm):
  1. Means $\boldsymbol{\mu}_k \in \mathbb{R}^D$ are initialized randomly
  2. Repeat until convergence (μ_k and z_{nk} do not change for any n and k):
     (a) **Expectation step**: Find the assignment of the closest cluster for every data point:

$$\frac{\partial J}{\partial z_{nk}} = 0 \Rightarrow z_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j ||\boldsymbol{x}_n - \boldsymbol{\mu}_j||^2 \\ 0 & \text{otherwise} \end{cases}$$

     (b) **Maximization step**: Find the means of each cluster:

$$\frac{\partial J}{\partial \boldsymbol{\mu}_k} = 0 \Rightarrow \boldsymbol{\mu}_k = \frac{\sum_n z_{nk} \boldsymbol{x}_n}{\sum_n z_{nk}}$$

- The algorithm converges as each phase reduces the value of the objective function J, but they might converge to a local rather than global minimum (perform multiple random restarts and choose best minimum found)
- **Application**: image compression. Every pixel is a data point, and we search for K clusters representing different colors in the image. The image is compressed by only using the cluster means (colors) instead of specifying a color at every pixel. A problem of this method is that the position correlations are ignored.
- **Failures** of K-means:
  - K-means is only able to cluster spherical data due to the squared distance we try to minimize. Other shapes require different distance measures or data transformation by some basis functions.
  - K-means strongly prefers clusters of the same size/spread, and therefore tries to find cluster with the same spread in the dataset.
  - K-means is very sensitive to outliers. As it tries to minimize the squared distance, outliers may have a significant effect on the cluster means.
- **Improvements**:

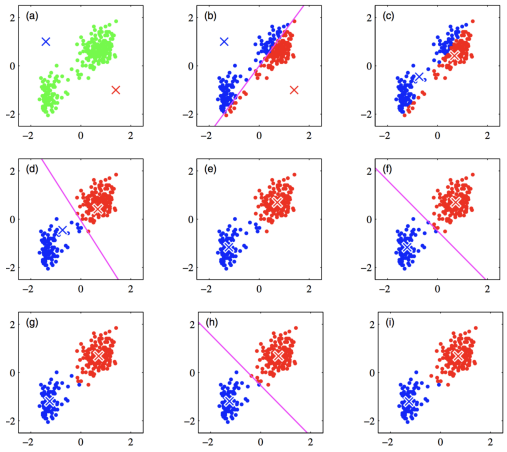


Figure 16: Illustration of the K-means algorithm. First, an expectation step is performed where the data points are assigned to a cluster (see *(b)*, *(d)*, *(f)*, and *(h)*), and then the maximization step optimizes the means of the clusters (see *(c)*, *(e)*, *(g)*, and *(i)*).

  - For a large dataset, we can use SGD to reduce computational effort. The update for a single data point would look like:

$$\boldsymbol{\mu}_k^{(\tau+1)} = \boldsymbol{\mu}_k^{(\tau)} - \eta \left(\frac{\partial J}{\partial \boldsymbol{\mu}_k^{(\tau)}} \right)^T = \boldsymbol{\mu}_k^{(\tau)} + 2\eta \left(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{(\tau)} \right)$$

  - Use other distance measure between points that is for example not so sensitive to outliers:

$$\tilde{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \mathcal{V} \left(\boldsymbol{x}_n, \boldsymbol{\mu}_k \right)$$

    where $\mathcal{V}$ measures the similarity of $\boldsymbol{x}_n$ and $\boldsymbol{\mu}_k$.

- **Pros and cons** of K-means
  - \+ Simple to implement
  - \+ Fast
  - – Local minima
  - – Only models spherical data
  - – Sensitive to feature scales and outliers
  - – Number of clusters K must be specified in advance with prior knowledge
  - – Cluster assignments are hard and not probabilistic

5.2 Mixture of Gaussians and EM algorithm

- Approximate the joint distribution $p\left(\boldsymbol{x}, \boldsymbol{z}\right) = p\left(\boldsymbol{x}|\boldsymbol{z}\right) \cdot p\left(\boldsymbol{z}\right)$ by a mixture of Gaussians ($\boldsymbol{z}$ chooses the mixture component, and points in the cluster are Gaussian distributed)
- We define the prior as $p\left(z_k = 1\right) = \pi_k$, where $\sum_k \pi_k = 1$ and $\pi_k \in [0, 1]$
- The single clusters are Gaussian: $p\left(\boldsymbol{x}|z_k = 1\right) = \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$
- Overall, the generative distribution is $p\left(\boldsymbol{x}\right) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$
- The posterior/conditional probability of $\boldsymbol{z}$ given $\boldsymbol{x}$ is also defined as the *responsibility* (that component k in the mixture model takes for 'explaining' the observation/data point $\boldsymbol{x}$):

$$p\left(z_k = 1|\boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x}|z_k = 1\right) \cdot p\left(z_k = 1\right)}{\sum_j p\left(\boldsymbol{x}|z_j = 1\right) \cdot p\left(z_j = 1\right)} = \frac{\pi_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)}{\sum_j \pi_j \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)} = \gamma\left(z_k\right)$$

- To optimize our parameters, we again maximize the log-likelihood:

$$\ln p\left(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$$

- However, maximizing the log-likelihood has no closed-form solution as stationary points depend on $\gamma(z_{nk})$ which again depends on $\boldsymbol{\pi}, \boldsymbol{\mu}$ and $\boldsymbol{\Sigma} \Rightarrow$ use expectation maximization algorithm by alternating the update of (*expected*) posterior $\gamma(z_{nk})$ and *maximizing* for parameters $\boldsymbol{\pi}, \boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$
- Maximizing with respect to $\boldsymbol{\mu}_k$:

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\mu}_k} \sum_{n=1}^{N} \ln p\left(\boldsymbol{x}_n | \{\pi_k\}_{k=1}^K, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{\Sigma}_k\}_{k=1}^K\right) \\
=&\sum_{n=1}^{N} \frac{1}{p\left(\boldsymbol{x}_n | \{\pi_k\}_{k=1}^K, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{\Sigma}_k\}_{k=1}^K\right)} \frac{\partial}{\partial \boldsymbol{\mu}_k} p\left(\boldsymbol{x}_n | \{\pi_k\}_{k=1}^K, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{\Sigma}_k\}_{k=1}^K\right) \\
=&\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} \\
=&\sum_{n=1}^{N} y(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} \\
\Rightarrow &\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}
\end{aligned}$$

- For maximizing π_k we need to use the Lagrange multiplier (as the sum must be 1):

$$\begin{aligned}
&\frac{\partial}{\partial \pi_k} \sum_{n=1}^{N} \ln p\left(\boldsymbol{x}_n | \{\pi_k\}_{k=1}^K, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{\Sigma}_k\}_{k=1}^K\right) + \lambda \left(\sum_{j=1}^{K} \pi_j - 1\right) \\
=&\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda \pi_k \\
=&\sum_{n=1}^{N} \gamma(z_{nk}) + \lambda \pi_k \\
\Rightarrow &\pi_k = -\frac{1}{\lambda} \sum_{n=1}^{N} \gamma(z_{nk})
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial}{\partial \lambda} \sum_{n=1}^{N} \ln p\left(\boldsymbol{x}_n | \{\pi_k\}_{k=1}^K, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\boldsymbol{\Sigma}_k\}_{k=1}^K\right) + \lambda \left(\sum_{j=1}^{K} \pi_j - 1\right) \\
=&\sum_{j=1}^{K} \pi_j - 1 = -\frac{1}{\lambda} \sum_{n=1}^{N} \underbrace{\sum_{j=1}^{K} \gamma(z_{nj})}_{=1} - 1 = 0 \\
\Rightarrow &\lambda = -N, \pi_k = \frac{N_k}{N} \quad \text{where} \quad N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \quad \text{(effective number of points in } k\text{)}
\end{aligned}$$

- Maximizing $\boldsymbol{\Sigma}_k$ is done by:

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T (\boldsymbol{x}_n - \boldsymbol{\mu}_k)$$

- Summarized EM algorithm steps for Gaussian mixture models:
  - **Expectation step**: update the posterior:

$$\gamma(z_k) = \frac{\pi_k \mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

- **Maximization step**: update the parameters:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}$$
$$\pi_k = \frac{N_k}{N}$$
$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T (\boldsymbol{x}_n - \boldsymbol{\mu}_k)$$

- Assigning points to clusters: either soft clusters (probability of belonging to k: $\gamma(z_k) = p(z_k = 1|\boldsymbol{x})$) or hard clusters (most likely cluster given by $k = \arg\min_j \gamma(z_j)$)
- **Pros and cons** of GMM:
  - \+ Allows soft-assignments in contrast to K-means
  - \+ More flexible as we can model different covariances per cluster
  - – Slower than K-means as every step requires more computation (can use K-means result as initialization)
  - – Same local convergence issues as K-means

5.3 Principal Component Analysis

- Find linear orthogonal projection to lower dimensional space to maximize variance ($\mathbb{R}^D \to \mathbb{R}^M$ where $M < D$)
- Try to find projection by capturing axes of maximal variation in the data, called *principal components*
- Covariance is given by $S = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T$ which is symmetric and positive semi-definite
- Project data into first latent dimension by $\boldsymbol{u}_1 \in \mathbb{R}^D$. As we only need its direction, we make sure that $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$
- The projection is given by $\boldsymbol{u}_1^T \boldsymbol{x}_n$.
- The variance of the projected data is $\boldsymbol{u}_1^T S \boldsymbol{u}_1$. We try to maximize this variance with respect to the constraint $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$ (Lagrangian multiplier):

$$\arg\max_{\boldsymbol{u}_1} \max_{\lambda_1} \boldsymbol{u}_1^T S \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1)$$

  Deriving by $\boldsymbol{u}_1$ gives us the equation $S\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$ which is the eigenvalue equation. Thus, $\boldsymbol{u}_1$ is an eigenvector and λ_1 and eigenvalue of S. As we try to maximize the equation, we choose λ_1 to be the *greatest* eigenvalue.

- $\boldsymbol{u}_1$ is called a *principal component*. The variance of the projected data is $\boldsymbol{u}_1^T S \boldsymbol{u}_1 = \lambda_1$
- We can repeat this procedure for M orthogonal vectors and get a projection $U_M = [\boldsymbol{u}_1, ..., \boldsymbol{u}_M] \in \mathbb{R}^{D \times M}$. Those are M eigenvectors of S, where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_M$. As S is positive semi-definite, we can ensure that $\lambda_i \geq 0$
- The variance of the projected data is $\sum_{j=1}^{M} \boldsymbol{u}_j^T S \boldsymbol{u}_j = \sum_{j=1}^{M} \lambda_j$
- **PCA**: Compute $\overline{\boldsymbol{x}}$ and the eigen-decomposition of S. The **projection** is $z = U_M^T (\boldsymbol{x} - \overline{\boldsymbol{x}})$
- The idea is that the information which is lost is only noise, so that we still keep the expressiveness of the data. However, eigen-decomposition might be expensive.
- Note that eigenvalues can be found by solving the equation $\det(S - \lambda \boldsymbol{I}) = 0$. We can represent S by its eigenvalue decomposition $S = U \Lambda U^T$. For the eigenvectors, we can state that $\boldsymbol{u}_j^T \boldsymbol{u}_i = 0$ if $i \neq j$, else 1.
- **Applications**
  - *Dimensionality reduction*: which M to choose? To preserve at least 90% of the variance we need to make sure that $\frac{\sum_{j=1}^{M} \lambda_j}{\sum_{j=1}^{D} \lambda_j} \geq 0.9$

- *Feature de-correlation*: PCA ensures that features have no correlation in projected space. The covariance matrix is diagonal: $S'_M = \Lambda_M$
- *Whitening*: center and de-correlate features by $\boldsymbol{z} = U_M^T(\boldsymbol{x} - \overline{\boldsymbol{x}})$ where M can be equals to D. If we want to rescale it (e.g. unit std. deviation), we apply a factor: $\boldsymbol{z} = \Lambda_M^{1/2} U_M^T(\boldsymbol{x} - \overline{\boldsymbol{x}})$
- *Compression*: transform input to lower dimensional space. Reconstruction can be performed by $\tilde{\boldsymbol{x}} = U_M \boldsymbol{z} + \overline{\boldsymbol{x}}$

- **Perspective of minimal reconstruction error**
  - An alternative view on PCA is minimizing the reconstruction error of the transformed data to the original space
  $$\min \frac{1}{N} \sum_{n=1}^{N} ||\boldsymbol{x}_n - \boldsymbol{z}_n||^2$$
  - We can express our data by $\boldsymbol{x}_n = \sum_{j=1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_j)\boldsymbol{u}_j$. The transformed data is thus $\boldsymbol{z}_n = \sum_{j=1}^{M} (\boldsymbol{x}_n^T \boldsymbol{u}_j)\boldsymbol{u}_j + \sum_{j=M+1}^{D} b_j \boldsymbol{u}_j$
  - (By doing some math) we can show that the objective function is actually $\sum_{j=M+1}^{D} \boldsymbol{u}_j^T S \boldsymbol{u}_j$. Thus, both approaches lead to the same result

5.3.1 Probabilistic PCA

- Generative probabilistic version of PCA where we learn by maximizing the likelihood (both latent and observed are Gaussian)
- Generative model works as $\boldsymbol{x} = W\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ represents the noise
- We define $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|0, \boldsymbol{I})$, $p(\epsilon) = \mathcal{N}(\epsilon|0, \sigma^2 \boldsymbol{I})$, and therefore $p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}, W\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$. By marginalizing out $\boldsymbol{z}$, we get $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, C)$ with $C = WW^T + \sigma^2 \boldsymbol{I}$
- New data points are generated by first sampling from low dimensional z space, and then sampling x based on z
- We can optimize this sample distribution based on the maximum likelihood of a given dataset (μ is as usual the mean, $\sigma^2 = \frac{1}{D-M} \sum_{j=M+1}^{D} \lambda_j$)
-

5.3.2 Non-linear variants of PCA

- Limitations of PCA: only linear transformation possible. So, we can also just capture variance along a linear axes through the x space
- We can get non-linear by using different forms
- **Kernel PCA**
  - We can define the covariance matrix by $C = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})^T$
  - Then, we can state that $z_i(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{u}_i = \sum_{n=1}^{N} a_{in} \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_{n=1}^{N} a_{in} k(\boldsymbol{x}, \boldsymbol{x}_n)$
  - By using a non-linear kernel, we are able to get non-linear projections
- **Autoencoders (NN)**
  - Non-linear dimensionality reduction with neural networks by having a low hidden dimension size, and trying to reproduce input
  - In variational auto-encoders, we introduce sampling from the latent space so that we can generate new data points

6 Kernel methods

- Standard parametric models have either fixed basis functions (like linear regression or linear classification models) or learnable basis functions like in neural networks. The training points are solely used to optimize the parameters $\boldsymbol{w}$, and all further predictions are based on these optimal parameters
- In contrast, kernel methods keep the training data points and also use them (or a subset) during prediction
- The predictions are based on a linear combination of the kernel function evaluated on the training data points:

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n k\left(\boldsymbol{x}, \boldsymbol{x}'\right)$$

- For linear models with fixed basis functions, the kernel is:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$$

- The kernel measures *similarity* between $\boldsymbol{x}$ and $\boldsymbol{x}'$ in features space defined by $\phi(x)$. Thus, it is symmetric:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$$

6.1 Kernelizing linear parametric models

- Many linear parametric model can be re-casted into a "dual representation" by using the **kernel trick**:
  - If we have an algorithm formulated in such a way that the input vector $\boldsymbol{x}$ enters only in the form of a scalar product, we can replace the scalar product with some other choice of kernel
- For instance, the linear regression model is determined by minimizing the regularized sum-of-squares error function given by:

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left\{\boldsymbol{w}^T \boldsymbol{\phi}\left(\boldsymbol{x}_n\right) - t_n\right\} + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}$$

- Solving the equation of the derivate being equals to 0, we obtain:

$$\boldsymbol{w} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I}_M\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} = \boldsymbol{\Phi}^T \left(\boldsymbol{\Phi} \boldsymbol{\Phi}^T + \lambda \boldsymbol{I}_M\right)^{-1} \boldsymbol{t}$$

- Here, we can replace the inner product $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ by the gram matrix $\boldsymbol{K}$ where $K_{ij} = \boldsymbol{\phi}(\boldsymbol{x}_i)^T \boldsymbol{\phi}(\boldsymbol{x}_j)$
- By defining the dual variable $\boldsymbol{\alpha} = \left(\boldsymbol{K} + \lambda \boldsymbol{I}_M\right)^{-1} \boldsymbol{t}$, we get the following equations:

$$\boldsymbol{w} = \boldsymbol{\Phi}^T \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$y\left(\boldsymbol{x}'\right) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}') = \sum_{n=1}^{N} \alpha_n \boldsymbol{\phi}\left(\boldsymbol{x}_n\right)^T \boldsymbol{\phi}\left(\boldsymbol{x}'\right) = \sum_{n=1}^{N} \alpha_n k\left(\boldsymbol{x}, \boldsymbol{x}'\right)$$

- Thus, we can express linear regression by a dual representation with kernel methods
- **Benefits** of kernel representation:
  - We have no explicit parameters/features anymore, only implicit by the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$
  - No need to handpick locations of basis functions
  - No increase in number of parameters when using kernel methods as those implicitly map inputs to a higher dimensional space
- **Disadvantages/problems**:
  - The computational cost to retrieve $\boldsymbol{\alpha}$ is $\mathcal{O}(N^3)$ as $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ compared to $\mathcal{O}(M^3)$ for calculating M on the standard way (the cost comes from the inverse)
  - During prediction, we need $\mathcal{O}(N \cdot M)$ to compute the output for a new point, but would only need $\mathcal{O}(N)$ with the primal parameters $\boldsymbol{w} \Rightarrow$ slow prediction for large datasets

6.1.1 Constructing valid kernels

- For a valid kernel, the gram matrix $\boldsymbol{K}$ must be positive semi-definite for all possible choices of $\{x_n\}_{n=1}^N$
- An equivalent constraint would be that $\boldsymbol{z}^T\boldsymbol{K}\boldsymbol{z} \geq 0$ for all $\boldsymbol{z} \in \mathbb{R}^N$ or the eigenvalues must all be positive (note that $\boldsymbol{K}$ can still contain negative elements)
- We can construct a kernel from an explicit set of basis functions when we use the expression $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})$
- Further, we can construct new kernels by using other valid kernels and extend them by for example multiplying with a constant (no need to know all variations)
- Given a valid kernel function, we can derive its corresponding feature vectors (which can be hard and possible infinite). Therefore, we need to express it in the form of $\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}')$ where $\boldsymbol{\phi}$ must be the same function applied on different points
- For example a polynomial kernel of $M = 2$ can be rewritten as:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{z}) &= \left(1 + \boldsymbol{x}^T\boldsymbol{z}\right)^2 = (1 + x_1z_1 + x_2z_2)^2 \\ &= 1 + 2x_1z_1 + 2x_1z_2 + (x_1z_1)^2 + (x_2z_2)^2 + 2x_1z_1x_2z_2 \\ &= \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1, x_2, \sqrt{2}x_1x_2\right] \cdot \left[1, \sqrt{2}z_1, \sqrt{2}z_2, z_1, z_2, \sqrt{2}z_1z_2\right]^T \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{z}) \end{aligned}$$

- Here we see that from a two-dimensional vector, we scaled it up to a 6-dimensional feature vector just from our kernel
- Some (popular) kernels:
  - Generalized polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \left(c + \boldsymbol{x}^T\boldsymbol{x}'\right)^M$ (feature vector only contains polynomial to order M)
  - Gaussian kernel with infinite feature dimensionality: $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2l^2}||\boldsymbol{x} - \boldsymbol{x}'||^2\right)$
  - Radial basis functions of the form $k(\boldsymbol{x}, \boldsymbol{x}') = k\left(||\boldsymbol{x} - \boldsymbol{x}'||^2\right)$

6.2 Support Vector Machines

- To overcome the slow prediction problem, support vector machines only uses a subset of the training points on which the kernel function needs to be evaluated (also called kernel methods with *sparse* solutions)
- It is a convex optimization problem so that only one single optimum exists
- No good probabilistic interpretation (see Gaussian Processes for that)

6.2.1 Maximum Margin Classifier

- Similar to discriminant functions in 3.3
- For a linearly separable dataset, the maximum margin is defined as the distance between the decision boundary and the closest training point $\Rightarrow$ most robust and stable for perturbations of the input
- The distance of a point to the decision boundary is (as previously) defined by:

$$r_n = \frac{|y(\boldsymbol{x}_n)|}{||w||} = \frac{t_n y(\boldsymbol{x}_n)}{||w||} \quad \text{if } \boldsymbol{x}_n \text{ correctly classified}$$

- The margin is defined as the minimum distance of decision boundary to any point:

$$\min_n \frac{t_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)}{||\boldsymbol{w}||}$$

- As we can easily increase the distance by increasing $\boldsymbol{w}$ by a factor κ and still get the same minimum ($\min_n \frac{t_n\left(\kappa\boldsymbol{w}^T\boldsymbol{x}_n + \kappa b\right)}{||\kappa\boldsymbol{w}||}$), we restrict the choice by setting $t_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) = 1$ for the closest point.
- Thus, for all other points, the following constraint must hold: $t_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$
- A maximum margin is found by maximizing $\frac{1}{||\boldsymbol{w}||}$ (as the upper part of the fraction is fixed to 1)

6.2.2 Optimizing Maximum Margin

- To maximize the margin, we try to minimize $\frac{1}{2}||\boldsymbol{w}||^2$ (has same optimum as $\frac{1}{||\boldsymbol{w}||}$ but is easier to optimize)
- By that, we need to fulfill the constraint $t_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$ for all data points
- To do that, we use Lagrange multiplier for inequalities
  - Given the problem to maximize $f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) \geq 0$, it is equivalent to optimize:

$$\max_{\boldsymbol{x}} \min_{\mu} L\left(\boldsymbol{x}, \mu\right) = \max_{\boldsymbol{x}} \min_{\mu} f(\boldsymbol{x}) + \mu g(\boldsymbol{x})$$

  - Note that if we want to minimize $f(\boldsymbol{x})$, it is equivalent to maximizing $-f(\boldsymbol{x})$:

$$\max_{\boldsymbol{x}} \min_{\mu} L\left(\boldsymbol{x}, \mu\right) = \max_{\boldsymbol{x}} \min_{\mu} -f(\boldsymbol{x}) + \mu g(\boldsymbol{x}) \Rightarrow \min_{\boldsymbol{x}} \max_{\mu} L\left(\boldsymbol{x}, \mu\right) = \min_{\boldsymbol{x}} \max_{\mu} f(\boldsymbol{x}) - \mu g(\boldsymbol{x})$$

  - We have the following (Karush-Kuhn-Tucker) conditions when optimizing this function:

$$\mu \geq 0, \qquad g(\boldsymbol{x}) \geq 0, \qquad \mu \cdot g(\boldsymbol{x}) = 0$$

  - There are two kinds of solutions:
    - If the stationary points lies in the region $g(\boldsymbol{x}) \geq 0$, we have $\nabla f(\boldsymbol{x}) = 0$ and $\mu = 0$
    - Otherwise, if stationary points lies on the boundary we have $\nabla f(\boldsymbol{x}) = -\mu \nabla g(\boldsymbol{x})$
  - We can solve the optimization problem by first getting a solution for $\tilde{L}(\mu) = \max_{\boldsymbol{x}} L(\boldsymbol{x}, \mu)$, and then optimizing it with respect to μ: $\max_{\mu} \tilde{L}(\mu)$
- When we apply this for our maximum margin classifier, we get the following optimization objective with N Lagrange multipliers a_n:

$$L\left(\boldsymbol{w}, b, \boldsymbol{a}\right) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{n=1}^{N} a_n \left\{t_n\left(\boldsymbol{w}^T\boldsymbol{x} + b\right) - 1\right\}$$

- First, minimize with respect to the primal variables $\boldsymbol{w}$ and b:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^T - \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n^T = 0 \rightarrow \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n^T$$

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N} a_n t_n = 0 \rightarrow \sum_{n=1}^{N} a_n t_n = 0$$

- Eliminating $\boldsymbol{w}$ and b gives the dual representation $\tilde{L}(\boldsymbol{a})$:

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \underbrace{\boldsymbol{x}_n^T \boldsymbol{x}_m}_{k(\boldsymbol{x}_n, \boldsymbol{x}_m)}$$

- For prediction, we use the previously derived result $\boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n^T$ to convert it into a kernel:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = \sum_{n=1}^{N} a_n t_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- For every data point, $a_n = 0$ or $t_n y(\boldsymbol{x}) = 1$. For all points that have $a_n > 0$ influence the prediction, so called support vectors. They lie on maximum margin hyperplanes
- The bias b can be determined by solving $t_n y(\boldsymbol{x}_n) = 1$ for a support vector x_n

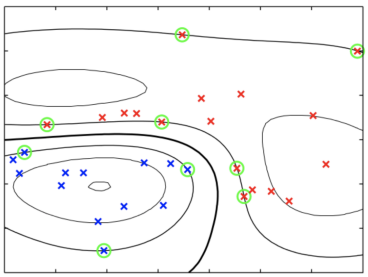


Figure 17: Visualization of non-linear support vectors

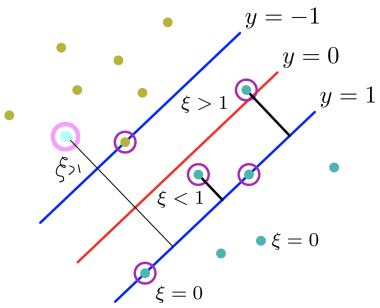


Figure 18: A soft margin classifier uses slack variables to penalize data points on the wrong side.

6.2.3 Soft Margin Classifier

- So far we assumed that dataset is (non-linear) separable. However, sometimes distributions overlap
- Thus, soft margin classifier allow data points to be on the ”wrong” side of the margin but causing a certain penalty
- We introduce **slack variables** $\xi_n \geq 0$ for $n = 1, ..., N$
- If a point is on the correct side of the margin, its slack variable is $\xi_n = 0$
- If it is one the wrong side of the margin, the slack variable is $\xi_n = |t_n - y(\boldsymbol{x}_n)|$
- Hence, we also have a “soft” constraint/margin $t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$
- The goal is now to maximize the margin while minimizing the penalty given by the slack variables:

$$\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=1}^{N}\xi_n$$

- Introducing the conditions $\xi_n \geq 0$ and $t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$ into the minimization problem, we get the following Lagrangian:

$$L\left(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu}\right) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}a_n\left\{t_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) - 1 + \xi_n\right\} - \sum_{n=1}^{N}\mu_n\xi_n$$

- The KKT conditions for the dual variables are:

$$\begin{aligned} &a_n \geq 0, \quad t_n y(\boldsymbol{x}_n) - 1 + \xi_n \geq 0, \quad a_n\left\{t_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) - 1 + \xi_n\right\} = 0 \\ &\mu_n \geq 0, \quad \xi_n \geq 0, \quad \mu_n\xi_n = 0 \end{aligned}$$

- Minimize w.r.t. primal variables $\boldsymbol{w}, b, \boldsymbol{\xi}$ and use these conditions to eliminate $\boldsymbol{w}, b, \boldsymbol{\xi}$ from the Lagrangian

to obtain the **dual representation**

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^T - \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n^T = 0 \implies \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n$$

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N} a_n t_n = 0 \implies \sum_{n=1}^{N} a_n t_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = C - a_n - \mu_n = 0 \implies a_n = C - \mu_n$$

$$\Rightarrow \tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{x}_n^T \boldsymbol{x}_m$$

- The remaining constraints are $0 \leq a_n \leq C$, and $\sum_{n=1}^{N} a_n t_n = 0$, and we try to *maximize* $\tilde{L}(\boldsymbol{a})$
- We can also express the dual representation with the kernel trick:

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

- When we want to predict the class for a new test data point $\boldsymbol{x}'$, we use:

$$y(\boldsymbol{x}') = \sum_{n=1}^{N} a_n t_n k(\boldsymbol{x}_n, \boldsymbol{x}') + b$$

- Points for different dual parameters:
  - Only points with $a_n > 0$ are support vectors and contribute to the prediction
  - If $C > a_n > 0$, then $t_n y(\boldsymbol{x}_n) = 1$ (points on the margin) as $\mu_n > 0$ and hence $\xi_n = 0$
  - If $a_n = C$, then $\mu_n = 0$ and $\xi_n \geq 0$. When $\xi_n \leq 1$, the points is still correctly classified but within the margin. Otherwise, the point is misclassified
- If $C \to \infty$, we recover the hard margin classifier again as we don't allow any outliers
- If $C \to 0$, the margin gets really large as we try to maximize the margin without caring about the misclassifications. Also, all points a_n will become support vectors

6.3 Gaussian Processes

6.3.1 Essentials of Gaussian distributions

- **Marginalization property**: if two random variables x_1 and x_2 are jointly Gaussian distributed, then marginalizing out one variables still leads to a Gaussian

$$p\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} | \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \implies p(x_1) = \mathcal{N}(\mu_1, \Sigma_1 1), \quad p(x_2) = \mathcal{N}(\mu_2, \Sigma_2 2)$$

- **Conditional property**: if two random variables x_1 and x_2 are jointly Gaussian distributed, then conditioning one variables on the other still leads to a Gaussian

$$p(x_1|x_2) = \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$$

- **Sum property**: Summing two independent Gaussian random variables lead to a new Gaussian variable:

$$x \sim \mathcal{N}(\mu_1, \Sigma_1) \text{ and } y \sim \mathcal{N}(\mu_2, \Sigma_2) \implies x + y = z \sim \mathcal{N}(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2)$$

- **Correlation property**: If $\boldsymbol{x}$ is an uncorrelated Gaussian random variable $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ then $\boldsymbol{y} = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{x}$ is correlated by $\boldsymbol{y} \sim \mathcal{N}(\mu, \boldsymbol{A}\boldsymbol{A}^T)$

6.3.2 Introduction to Gaussian Processes

- In Bayesian linear regression, we assume that the target is distributed as $t = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{w} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \beta^{-1})$. The posterior is also Gaussian distributed: $p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, \boldsymbol{S}_N)$.
- When we predict for new points, we use the mean $\mu_N = \sum_{n=1}^{N} \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N^{-1} \boldsymbol{\phi}(\boldsymbol{x}_n) t_n$
- Here we see that we can express the mean by the kernel $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{S}_N^{-1} \boldsymbol{\phi}(\boldsymbol{x}_m) \Rightarrow$ increase expressiveness of Linear Bayesian regression by using more complex kernels
- Definition of Gaussian Processes: A Gaussian process is a collection of random variables, any finite number of which is jointly Gaussian distributed
- Gaussian Processes represent distributions over random functions!

$$f(\circ) \sim \mathcal{N}(m(\circ), k(\circ, \circ))$$

- The function *evaluated* at a specific point $\boldsymbol{x}$ is a random variable, with $\mathbb{E}[f(\boldsymbol{x})] = m(\boldsymbol{x})$ and $\text{cov}(f(\boldsymbol{x}), f(\boldsymbol{x}')) = k(\boldsymbol{x}, \boldsymbol{x}')$ (covariance matrix is the gram matrix K)
- Thus, for a finite set of points $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, the random variables $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$ are:

$$p\left(\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}\right)$$

- Each entry is the sampled function evaluated at point $\boldsymbol{x}$. We can evaluate/sample functions by just using a fine-grained set of points
- The kernel has a significant influence on how the functions might look like. When we consider the kernel $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left(-\frac{1}{2\theta_1}||\boldsymbol{x}_n - \boldsymbol{x}_m||^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$, we see that:
  - θ_0 influences the amplitude of the samples of f
  - θ_1 scale the length of correlation
  - θ_2 introduces a random bias for sampled f (different bias for every sample)
  - θ_3 adds a linear component into the samples leading to a up-/down-ward trend

6.3.3 Regression with Gaussian Processes

- We have observed data which we model by $f(\boldsymbol{x}_i) = y(\boldsymbol{x}_i) + \epsilon$ ($\epsilon \sim \mathcal{N}(0, \beta^{-1})$)
- We can now model y as GP:

$$p\left(\begin{bmatrix} y(\boldsymbol{x}_1) \\ \vdots \\ y(\boldsymbol{x}_N) \end{bmatrix}\right) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}\right)$$

- Then, $f(\circ)$ is also a GP since $\boldsymbol{f} = \boldsymbol{y} + \epsilon$ ($\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{0}, K(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I})$)
- For new test data points, we can predict them by using:

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}^* \end{bmatrix}\right) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I} & K(\boldsymbol{X}, \boldsymbol{X}^*) \\ K(\boldsymbol{X}^*, \boldsymbol{X}) & K(\boldsymbol{X}^*, \boldsymbol{X}^*) + \beta^{-1}\boldsymbol{I} \end{bmatrix}\right)$$

- The more points we see the more certain our predictions gets
- Kernel parameters can be chosen based on MLE on training observations

7 Combining models

- Improve performance by combining different models
- For example, we can train L different models and take their average as prediction (called committee)
- Alternatively, we can also make the choice of which model we should use for an input $\boldsymbol{x}$ dependent on $\boldsymbol{x}$. This example includes Mixtures of experts
- **Bayesian model averaging vs. model combination methods**
  - In Bayesian model averaging, the entire dataset is generated by a single model. We are just unsure which one it is. The likelihood of the data is thus:

$$p(\boldsymbol{X}) = \sum_{h=1}^{H} p(\boldsymbol{X}|h)p(h)$$

  - In contrast, model combination methods consider that different data points can be generated by different components. So, every data point has its own latent variable $\boldsymbol{z}_n$. The likelihood is here given by:

$$p(\boldsymbol{X}) = \prod_{n=1}^{N} \sum_{\boldsymbol{z}_n} p(\boldsymbol{x}_n|\boldsymbol{z}_n)p(\boldsymbol{z}_n)$$

    Example methods include Gaussian mixture models and Mixture of experts.

7.1 Committees

- We can motivate the idea of committees by the bias-variance decomposition: when we average over models, we are able to reduce the variance of the model's predictions. Thus, by using complex models with low bias error, we can improve the performance by reducing the variance through averaging
- Averaging is therefore only effective if models are complex enough to overfit
- However, in practice, we have only one dataset on which we train $\Rightarrow$ introduce variability between the models within the committee by various methods

7.1.1 Bootstrap aggregation

- Suppose we have a dataset $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_N]^T$
- **Bootstrapping dataset**: we create B datasets by sampling N datapoints *with replacement* from the original dataset $\boldsymbol{X}$. So, in $\boldsymbol{X}_b$, some points will occur more than once and others might be absent
- For doing regression with this method, we train B models on their corresponding dataset, and use the average prediction for a new point:

$$y(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} y_b(\boldsymbol{x})$$

- This is called bootstrap aggregation or also *bagging*
- The average error made by one of the models is $E_{\text{AV}} = \frac{1}{B} \sum_{b=1}^{B} \mathbb{E}_{\boldsymbol{x}} \left[\epsilon_b(\boldsymbol{x})^2\right]$. In contrast, for the committee, we expect an error of:

$$E_{\text{COM}} = \mathbb{E}_{\boldsymbol{x}} \left[\left\{ \frac{1}{B} \sum_{b=1}^{B} \epsilon_b(\boldsymbol{x}) \right\}^2 \right]$$

- If all models would be independent (which they are not because of using very similar datasets), we would reduce the expected error by factor B. In practice, we can at least guarantee that $E_{\text{COM}} \leq E_{\text{AV}}$
- Still, bias error cannot be reduced by bagging!

7.1.2 Feature bagging

- Similar to bagging, but based on features: sample a subset of *features* of length $r < D$ for each learner.

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_D]^T \Rightarrow \tilde{\boldsymbol{x}} = [x_1, x_3, x_5, x_{D-1}]^T$$

- Also called *random subspace method*
- Works especially well if features are uncorrelated and/or if the number of features is much larger than the number of training points
- Decision trees with bagging and random subspaces lead to random forests

7.1.3 Boosting

- Use a set of simple individual models (also called weak classifiers) which even can be only slightly better than random
- In the following description, we concentrate on boosting for classification, but it can also be used for regression
- **AdaBoost**: adaptive boosting
- Base classifiers are trained in a sequence where every model uses a weighted form of the dataset
- The weight coefficients are associated to the performance of the previous models
- In the end, a prediction is based on the (weighted) majority voting scheme:

$$Y_M(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x})\right)$$

- AdaBoost algorithm:
  1. Initialize weights $w_n = 1/N$ for all $n = 1, ..., N$
  2. For all models $m = 1, ..., M$ sequentially:
     (a) Fit classifier $y_m(\boldsymbol{x})$ to minimize $J_m = \sum_{n=1}^{N} w_n^{(m)} \boldsymbol{I}[y_m(\boldsymbol{x}_n) \neq t_n]$
     (b) Compute weighted error rate $\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} \boldsymbol{I}[y_m(\boldsymbol{x}_n) \neq t_n]}{\sum_{n=1}^{N} w_n^{(m)}}$ and $\alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
     (c) Update weights $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m \boldsymbol{I}[y_m(\boldsymbol{x}_n) \neq t_n]\}$
  3. Make predictions $Y_M(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x})\right)$
- Note that the weight in the prediction (α_m) is based on the average error it has on the weighted training dataset (greater weights for more accurate models)
- When taking a huge number of basis models (large M), we can easily overfit
- Interpretation/Derivation of AdaBoost: minimizing exponential error function sequentially ($E_m = \sum_{n=1}^{N} \exp(-t_n f_m(\boldsymbol{x}_n))$)
- **Advantages**: simple boosting algorithm
- **Disadvantages**: very sensitive to outliers ($t_n y_m(\boldsymbol{x})$ very large and exponentially increasing weight), no probabilistic interpretation

7.2 Decision trees

- Split input space into rectangles which are aligned along the axes (parallel to axes)
- We use sequential binary decisions which can be summarized in a tree structure
- Used for classification and regression
- **Advantages**: interpretable, combining with boosting strongly increases performance
- **Disadvantages**: Not state-of-the-art, large trees easily overfit but small trees underfit (can be prevented by training large trees and sequentially removing nodes that reduce the error the least)
- Tree building process is recursively by minimizing the squared error (for regression). At each iteration, we add the feature boundary that reduces the error the most
- Stop criteria can be for example min. number of data points in region, depth/height,... or decrease of loss is lower than certain threshold
- *Pruning*: give a penalty to trees with large number of leafs to prevent unnecessary overfitting
- **Random forests**: By combining bootstrapping and feature bagging, we ensure that the models uses different features to build the trees. Thus, the models are less correlated and probably result in better accuracies.

A Appendix: Foundations

A.1 Important functions

A.1.1 Rectified Linear Unit

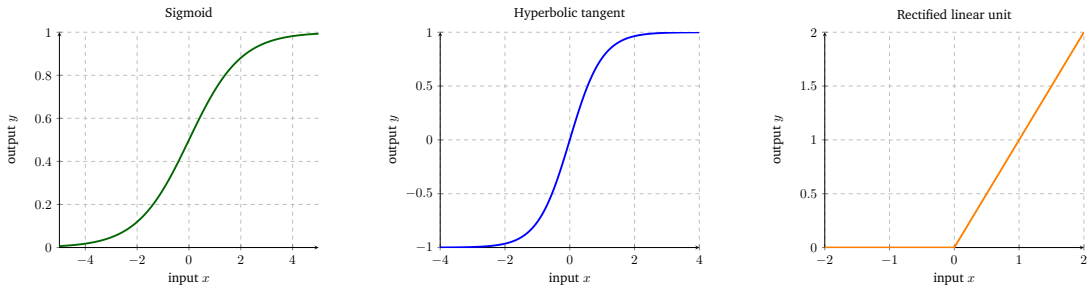
Properties of the ReLU function:

- $\text{ReLU}(x) = \max(x, 0)$
- $\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undef} & \text{if } x = 0 \end{cases}$ (last case usually set to 0)
- Variations:
  - Leaky ReLU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$
  - ELU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases}$
  - Self-normalizing ELU (carefully selected α and scaling, so that activations stay close to mean 0, variance 1)

A.1.2 Sigmoid

Properties of the sigmoid function:

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- $\sigma(-x) = 1 - \sigma(x)$
- $\sigma'(x) = \sigma(x)\,(1 - \sigma(x))$
- Output range: $[0, 1]$



(a) $\sigma(x)$ (b) $\tanh(x)$ (c) $\text{ReLU}(x)$

Figure 19: (a) The sigmoid function maps the inputs to a range of 0 to 1 while having high gradients near to $y = 0$ to bring the output more to either 0 or 1. (b) The hyperbolic tangent is similar to the sigmoid function but has a output range of -1 to 1. (c) A rectified linear unit (ReLU) is 0 for all input lower than 0. All other values are processed linearly so that they do not change.

A.1.3 Hyperbolic tan

Properties of the hyperbolic tan:

- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $\tanh(-x) = -\tanh(x)$
- $\tanh'(x) = 1 - \tanh(x)^2$
- Output range: $[-1, 1]$

A.1.4 Softmax

Properties:

- softmax$(x_k) = \frac{\exp(x_k)}{\sum_{i=1}^{N} \exp(x_i)}$
- If $x_k \gg x_j$, the softmax for all $j \neq k$ is approx. 0, whereas for k it is 1
- Maps vector from $\boldsymbol{y} \in \mathbb{R}^D$ to probability distribution $\boldsymbol{y}' \in [0,1]^D$ with $\sum_{i=1}^{D} y_i' = 1 \Rightarrow$ useful for multi-class classification
- Invariant to bias: $\frac{\exp(x_k+c)}{\sum_{i=1}^{N} \exp(x_i+c)} = \frac{\exp(x_k)}{\sum_{i=1}^{N} \exp(x_i)} =$ softmax(x_k)

A.2 Matrix operations

A.2.1 Properties of transposed and inverse matrices

Transpose

- $(AB)^T = B^T A^T$
- $\det\left(A^T\right) = \det\left(A\right)$

Inverse

- $(AB)^{-1} = B^{-1}A^{-1}$
- $\det\left(A^{-1}\right) = \det\left(A\right)^{-1}$

Combination

- $\left(A^{-1}\right)^T = \left(A^T\right)^{-1}$

A.2.2 Derivations

A.2.3 Hand-in 1: 1.3d

Derivation of multivariate Gaussian by matrix

A.3 Lagrange Multiplier

- Finding stationary points of a function with subject to one or more constraints
- **Equality constraint**
  - Maximize $f(\boldsymbol{x})$ with respect to constraint $g(\boldsymbol{x}) = 0$
  - At a constrained maximum, we know that $\nabla f(\boldsymbol{x}) = -\lambda \nabla g(\boldsymbol{x})$
  - The Lagrangian function is therefore

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$$

  - We solve it by maximizing regarding to $\boldsymbol{x}$ and λ:

$$\max_{\boldsymbol{x}} \max_{\lambda} L(\boldsymbol{x}, \lambda)$$

  - Note that the sign of the constraint is irrelevant. A minus sign leads to the same result as $g(x)$ must be zero at this point
  - We find solutions by setting the derivate of both primal and dual variables to 0:

$$\frac{\partial}{\partial \boldsymbol{x}} L(\boldsymbol{x}, \lambda) = 0, \quad \frac{\partial}{\partial \lambda} L(\boldsymbol{x}, \lambda) = 0$$

- **Inequality constraint**
  - Maximize $f(\boldsymbol{x})$ with respect to constraint $g(\boldsymbol{x}) \geq 0$ (introduce Lagrangian multiplier μ)
  - Two kinds of solutions:

    - If the optimum of $f(\boldsymbol{x})$ lies already in the region of $g(\boldsymbol{x}) \geq 0$, then we have an inactive constraint $\Rightarrow \mu = 0$
    - Otherwise, the optimum is on the boundary so that $g(\boldsymbol{x}) =$ and $\mu > 0$
  - Thus, our primal Lagrangian is defined as:

$$L(\boldsymbol{x}, \mu) = f(\boldsymbol{x}) + \mu g(\boldsymbol{x})$$

  - We now maximize regarding $\boldsymbol{x}$, but *minimize* for the Lagrangian multiplier as we prefer $f(x)$ being inside the constraint area:

$$\max_{\boldsymbol{x}} \min_{\mu} L(\boldsymbol{x}, \mu)$$

  - Note that the sign is here important. When we minimize $f(x)$, we can keep the max-min conditions for the Lagrangian but then have to switch the sign in front of the constraint!
  - Also, deriving by μ does not guarantee a valid solution anymore as we have the following KKT conditions for *every* Lagrangian multiplier:

$$\mu \geq 0 \qquad g(\boldsymbol{x}) \geq 0 \qquad \mu g(\boldsymbol{x}) = 0$$

  - We obtain the dual Lagrangian by optimizing with respect to only the primal variables $\boldsymbol{x}$, and replacing those in the primal Lagrangian:

$$\tilde{L}(\mu) = \max_{\boldsymbol{x}} L(\boldsymbol{x}, \mu)$$

  - Next, minimize with respect to the dual parameters μ by considering the constraint $\mu = 0$

- **Combined constraints**
  - If we have multiple constraints (can be pure (in-)equalities or mixed), we just add them all to our Lagrangian function
  - Solve with respect to all constraints

Summary Machine Learning 2

Phillip Lippe

October 22, 2019

Contents

1 Introduction to popular distributions and their properties

- This section (lecture 1 and 2) reviews different kinds of distributions, including the exponential family, Student-t distribution and common distributions for binary and discrete variables
- Furthermore, we shortly introduce Independent Component Analysis and Information theory
- In general, the first two lectures gave some fundamental knowledge we will use a couple of times for the rest of the course
- More mathematical tricks or examples of the exponential family can be found in the appendix

1.1 Exponential family distributions

(Bishop 2.4)

- A distribution is considered a member of the exponential family if it can be written as follows:

$$p(\boldsymbol{x}|\boldsymbol{\eta}) = h(\boldsymbol{x})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T \cdot \boldsymbol{u}(\boldsymbol{x})\right)$$

$\boldsymbol{\eta}$ natural parameters
$\boldsymbol{u}(\boldsymbol{x})$ sufficient statistics

- $\boldsymbol{u}(\boldsymbol{x})$ is called sufficient statistics because for the maximum likelihood estimate of $\boldsymbol{\eta}$, it is sufficient to record $\sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n)$ instead of the whole dataset $\{\boldsymbol{x}_n\}_{n=1}^{N}$ (see below for ML estimate)
- An important property of the exponential families is that the moments of distributions (i.e. mean and variance) can be determined by deriving $-\ln g(\boldsymbol{\eta})$ by $\boldsymbol{\eta}$:

$$\text{Normalization constant } \; z(\boldsymbol{\eta}) = \frac{1}{g(\boldsymbol{\eta})} = \int h(\boldsymbol{x})\exp\left(\boldsymbol{\eta}^T \cdot \boldsymbol{u}(\boldsymbol{x})\right) d\boldsymbol{x}$$

$$\frac{\partial}{\partial \boldsymbol{\eta}} - \ln g(\boldsymbol{\eta}) = \frac{1}{z(\boldsymbol{\eta})} \int h(\boldsymbol{x})\boldsymbol{u}(\boldsymbol{x})\exp\left(\boldsymbol{\eta}^T \cdot \boldsymbol{u}(\boldsymbol{x})\right) d\boldsymbol{x} = \mathbb{E}[\boldsymbol{u}(\boldsymbol{x})|\boldsymbol{\eta}]$$

  - Note that these moments are of the sufficient statistics $\boldsymbol{u}(\boldsymbol{x})$, and not $\boldsymbol{x}$
  - Additionally, the second moment around the mean can be determined by: $\nabla_{\boldsymbol{\eta}}^2 - \ln g(\boldsymbol{\eta})$
- From the first moment, we can show that the MLE solution of the natural parameters are:

$$-\nabla_{\boldsymbol{\eta}} \ln g(\boldsymbol{\eta}) = \mathbb{E}[\boldsymbol{u}(\boldsymbol{x})|\boldsymbol{\eta}] \implies -\nabla_{\boldsymbol{\eta}} \ln g(\boldsymbol{\eta}_{\text{ML}}) = \tfrac{1}{N}\sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x})$$

1.1.1 Conjugate priors

- A conjugate prior $p(\boldsymbol{\eta})$ is conjugate to the likelihood so that the posterior $p(\boldsymbol{\eta}|\boldsymbol{X})$ has the same form as the prior
- Each member of the exponential family has a conjugate prior
- To find the conjugate prior for a exponential distribution as likelihood, we only have to look at $\boldsymbol{\eta}$ of the likelihood and $\boldsymbol{u}(\boldsymbol{x})$ of the prior take on the same form. Then, we simply get:

$$\boldsymbol{u}(\boldsymbol{x})_{\text{posterior}} = \boldsymbol{\eta}_{\text{likelihood}} = \boldsymbol{u}(\boldsymbol{x})_{\text{prior}}$$

$$\boldsymbol{\eta}_{\text{posterior}} = \boldsymbol{u}(\boldsymbol{x})_{\text{likelihood}} + \boldsymbol{\eta}_{\text{prior}}$$

1.1.2 Bayesian Inference for Gaussian

- We can demonstrate the conjugate prior idea for Gaussians (one dimensional), where we have to distinguish three cases

1. Variance known, mean estimated

- Conjugate prior is a Gaussian $p(\mu) = \mathcal{N}(\mu|\mu_0, \sigma_0^2)$ such that our posterior has the distribution:

Variance known, mean estimated

$$p(\mu|\mathcal{D}) = \mathcal{N}(\mu|\mu_N, \sigma_N^2), \qquad \mu_N = \frac{\sigma^2\mu_0 + N\sigma_0^2\mu_{\text{ML}}}{N\sigma_0^2 + \sigma^2}, \qquad \frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}$$

2. Mean unknown, variance estimated

- Conjugate prior for the precision $\lambda = \frac{1}{\sigma^2}$ is a Gamma distribution $\text{Gamma}(\lambda|a_0, b_0)$ such that the posterior is:

Mean known, variance estimated

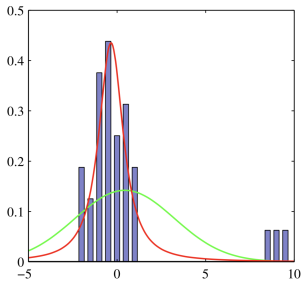
$$p(\lambda|\mathcal{D}) = \text{Gamma}(\lambda|a_N, b_N), \qquad a_N = a_0 + \frac{N}{2}, \qquad b_N = b_0 + \frac{1}{2}\sum_n (x_n - \mu)^2$$

3. Variance and mean estimated

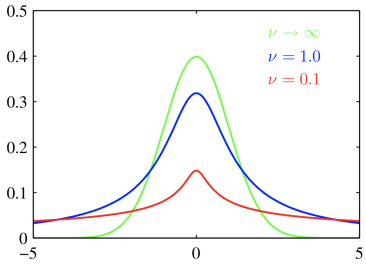
- If both are unknown, we have a "normal-Gamma" distribution as prior and posterior: $p(\mu, \lambda|a, b, \mu_0, \beta) = \mathcal{N}(\mu|\mu_0, (\beta\lambda^{-1}))\text{Gamma}(\lambda|a, b)$
- Finding the posterior is harder in this case because of the combined distribution. For details, see Bishop, but in the lecture it was not further discussed

1.2 Student's-t distribution

- The Student-t distribution is "heavy-tailed", meaning that the probability for data points decreases slower with the distance from the mean/center than for a Gaussian (polynomial $\text{St}(x) \propto |x|^{-\alpha}$ instead of exponential $\mathcal{N} \propto e^{-\frac{x^2}{\sigma^2}}$)
- This makes the distribution more robust against outliers as the MLE solution is less influenced by those and focuses more on the biggest data point mass (see Figure 1a)



(a) MLE estimate



(b) Effect of parameter ν

Figure 1: (a) Comparison of MLE solution of Student-t distribution (red) and Gaussian (green). (b) The parameter ν for fixed $\mu = 0$ and $\lambda = 1$. For $\nu \to \infty$,

- It emerges from a infinite mixture of Gaussians with a fixed mean and the precision (i.e. inverse variance) distributed as a Gamma distribution:

1. Draw precision $\tau \sim \text{Gamma}(a, b)$
2. Draw $x \sim \mathcal{N}(\mu, \tau^{-1})$

Then the resulting x will be distributed according to the Student-t distribution

$$p(x) \sim \text{St}(x \mid \mu, \lambda = a/b, \nu = 2a)$$

- By marginalizing out τ, we can derivate the PDF of the student distribution:

$$\text{Scalar} \quad \text{St}(x \mid \mu, \lambda = a/b, \nu = 2a) = \frac{b^a}{\Gamma(a)\sqrt{2\pi}} \left(b + \frac{(x-\mu)^2}{2}\right)^{-a-\frac{1}{2}} \Gamma\left(a + \frac{1}{2}\right)$$

$$\text{d-dimensional} \quad \text{St}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) = \frac{\Gamma\left(\frac{d}{2} + \frac{\nu}{2}\right)}{\Gamma\left(\frac{d}{2}\right)} \frac{1}{(\pi\nu)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \left(1 + \nu^{-1} (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right)^{-\frac{d}{2}-\frac{\nu}{2}}$$

- The parameter λ is often called precision, but does not exactly represent the inverse of the variance.
- ν is called the degrees of freedom (see Figure 1b). For $\nu \to \infty$, the student-t distribution becomes a Gaussian $\mathcal{N}(x|\mu, \lambda^{-1})$

1.3 Distributions for Binary and Discrete Variables

- In this section, we review common distributions for binary and discrete distributions. We can actually find one-to-one correlations in the binary and categorical space:

Binary	Discrete
Bernoulli	Categorical
Binomial	Multinomial
Beta	Dirichlet

1.3.1 Binary

Bernoulli distribution can be interpreted as a coin flip, and models a single binary outcome:

$$\text{Bern}(x|\mu) = \mu^x (1-\mu)^{1-x}, \quad x \in \{0, 1\}$$

- Expectation $\mathbb{E}[x|\mu] = \mu$
- Variance $\mathbb{V}\text{ar}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \mu(1-\mu)$
- Maximum likelihood estimate $\mu_{\text{ML}} = \frac{1}{N} \sum_{n=1}^{N} x_n$ (sensitive to overfitting for small dataset)
- Exponential family $p(x|\eta) = \sigma(-\eta) \exp(\eta \cdot x), \eta = \ln \frac{\mu}{1-\mu}$

Binomial distribution models N i.i.d. Bernoulli experiments, where we define m as $m = \sum_{i=1}^{N} x_i$, i.e. the number of times the outcome is 1:

$$\text{Bin}(m|N, \mu) = \frac{N!}{(N-m)!m!} \mu^m (1-\mu)^{N-m}$$

- Expectation $\mathbb{E}[m] = \sum_{i=1}^{N} \mathbb{E}[x_i] = N \cdot \mu$
- Variance $\mathbb{V}\text{ar}[x] = N \cdot \mu(1-\mu)$
- Maximum likelihood estimate $\mu_{\text{ML}} = \frac{m}{N}$
- Exponential family $p(m|\eta) = \frac{N!}{(N-m)!m!} \cdot \exp(N \log 1 - \mu) \cdot \exp(m \log \frac{\mu}{1-\mu}), \eta = \log \frac{\mu}{1-\mu}$
- Conjugate prior: Beta distribution. The posterior is: $\text{Beta}(\mu|a+m, b+N-m)$

Beta distribution is the conjugate prior for the binomial distribution

$$\text{Beta}(\mu|a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \mu^{a-1} (1-\mu)^{b-1}$$

- Expectation $\mathbb{E}[\mu] = \frac{a}{a+b}$
- Variance $\mathbb{V}\text{ar}[x] = \frac{ab}{(a+b)^2(a+b+1)}$
- Exponential family: see Appendix

1.3.2 Discrete

Categorical distribution considers a single sample, and assign each category a different probability. The input $\boldsymbol{x}$ is a one-hot vector.

$$\text{Cat}(\boldsymbol{x}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k} = \mu_{x_k}, \quad \sum_k \mu_k = 1$$

- Expectation $\mathbb{E}[\boldsymbol{x}] = \boldsymbol{\mu}$
- Covariance $\text{Cov}[\boldsymbol{x}] = \text{diag}(\boldsymbol{\mu}(1-\boldsymbol{\mu}))$
- Maximum likelihood estimate $\boldsymbol{\mu}_{\text{ML}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}$
- Exponential family $p(\boldsymbol{x}|\boldsymbol{\eta}) = \frac{1}{1+\sum_{k=1}^{K-1}\exp(\eta_k)} \cdot \exp(\boldsymbol{\eta}^T\boldsymbol{x})$, $\eta_k = \ln \frac{\mu_k}{1-\sum_{j=1}^{K-1}\mu_j}$

Multinomial distribution takes N i.i.d. categorical observations into account, where $m_k = \sum_{n=1}^{N} x_{nk}$.

$$\text{Mult}(m_1, ..., m_K|N, \boldsymbol{\mu}) = \frac{N!}{\prod_{k=1}^{K} m_k!} \prod_{k=1}^{K} \mu_k^{m_k}$$

- Expectation: $\mathbb{E}[\boldsymbol{x}] = N \cdot \mu$
- Covariance: $\text{Cov}[\boldsymbol{x}, \boldsymbol{x}] = N(\text{diag}(\boldsymbol{\mu}) - \boldsymbol{\mu}\boldsymbol{\mu}^T)$
- Maximum likelihood estimation: $\boldsymbol{\mu}_{\text{ML}} = \frac{\boldsymbol{m}}{N}$
- Exponential family: see Appendix

Dirichlet distribution is the conjugate prior for multinomial

$$\text{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1}$$

- Expectation $\mathbb{E}[\boldsymbol{x}] = \frac{1}{\sum_k \alpha_k}\boldsymbol{\alpha}$
- Covariance $\text{Cov}[\boldsymbol{x}] = -\frac{1}{\sum_k \alpha_k + 1}\boldsymbol{\alpha}\boldsymbol{\alpha}^T$
- Exponential family: see Appendix

1.4 Independent Component Analysis

- Independent Component Analysis (ICA) tries to reconstruct source signals from linearly mixed measurements. For example, for two sources $S(t) = \begin{bmatrix} S_1(t) \\ S_2(t) \end{bmatrix}$, we assume to have the measurements:

  $$X(t) = \begin{bmatrix} X_1(t) \\ X_2(t) \end{bmatrix} = \begin{bmatrix} \alpha_1 S_1(t) + \beta_1 S_2(t) \\ \alpha_2 S_2(t) + \beta_2 S_2(t) \end{bmatrix}$$

  The goal is now to find the parameter matrix

  $$\boldsymbol{A} = \begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \end{bmatrix}$$

  to reconstruct our signals $S(t)$ from the measurements $\boldsymbol{X}(t) = \boldsymbol{A}\boldsymbol{S}(t)$

- Note that we can only reconstruct $S(t)$ up to permutation and scaling/multiplicative factors as these give the same result

- As we assume the sources to be independent, we can write the joint probability distribution as:

  $$p(S_1, ..., S_I) = \prod_{i=1}^{I} p(S_i)$$

  One crucial element of ICA is that these prior distributions need to be designed by the user. This requires pre-knowledge of how the source signals can look like (e.g. Gaussian, bounded Uniform, etc.). The performance of the algorithm depend on this design choice, and can lead to ICA failing if the prior has a very different distribution than points in the sources.

- We will again use a maximum likelihood approach where we try to increase the probability of the observed data, which can be derived as:

$$\ln p(\boldsymbol{x}|\boldsymbol{A}) = \ln|\det \boldsymbol{A}| + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{I}\ln p_i\left(\sum_{j=1}^{I}\left(A^{-1}\right)_{ij}x_j^{(n)}\right)$$

  For simplicity, we replace $\boldsymbol{A}^{-1} = \boldsymbol{W}$, and aim to learn $\boldsymbol{W}$ which is slightly easier.

- We take now the derivative with respect to $\boldsymbol{W}$, and end up with the following expression:

$$\boldsymbol{W}^{t+1} = \boldsymbol{W}^t + \alpha \cdot \frac{1}{N}\sum_{n=1}^{N}\left(\nabla_{\boldsymbol{S}}\log p(\boldsymbol{S})\Big|_{S=S_n}\boldsymbol{S}_n^T + \boldsymbol{I}\right)\boldsymbol{W}$$

  where we estimate $\boldsymbol{S} = \boldsymbol{W}\boldsymbol{X}$. In addition, we see here that what we actually need from our prior is the derivative of its log. Hence, the prior is mostly designed to have a simple form of $\Phi_i = \frac{\partial \ln p_i(a_i)}{\partial a_i}$.

- We can slightly simplify the gradient calculation by splitting it into multiple parts. Summarizing the full algorithm, we get:

Independent Component Analysis

Choose prior and calculate log derivative $\Phi_i = \frac{\partial \ln p_i(a_i)}{\partial a_i}$;
Set learning rate η;
Initialize $\boldsymbol{W} = \boldsymbol{A}^{-1}$;
while $\nabla \boldsymbol{W}^{(t)} > \epsilon$ **do**
  Let $\hat{\boldsymbol{S}} = \boldsymbol{W}\boldsymbol{X}$ be the current estimate of $\boldsymbol{S}$;
  Let $\boldsymbol{Z}_i = \Phi_i(\hat{\boldsymbol{S}}_i)$;
  Let $\boldsymbol{X}' = \boldsymbol{W}^T\hat{\boldsymbol{S}}$;
  Calculate the gradients $\nabla \boldsymbol{W}^{(t)} = \boldsymbol{W}^{(t)} + \frac{1}{N}\left[\boldsymbol{Z}\boldsymbol{X}'^T\right]$;
  Apply gradient with learning rate $\boldsymbol{W}^{(t+1)} = \boldsymbol{W}^{(t)} + \eta\nabla\boldsymbol{W}^{(t)}$;
end
Reconstruct signals $\boldsymbol{S}_n = \boldsymbol{W}\boldsymbol{X}_n$;

- One issue with ICA is that the signals are not allowed to be Gaussian. If this would be the case, we can not reconstruct the signal up to rotation as Gaussians are rotation invariant. Hence, the signals will be messed up although we find an optimum

1.5 Information theory

- The information of an event A can be measured by:

$$\begin{aligned} h(A) &= -\log_2 p(A) \quad \text{(in bits)} \\ &= -\ln p(A) \quad \text{(in nats)} \end{aligned}$$

- An important measurement of a distribution in information theory is the Shannon entropy, which can be interpreted as the expected information of an event according to the distribution p:

$$H(X) = -\sum_{x\in D_x} p(x)\log_2 p(x)$$

  In case we have N independent events, the entropy is the sum of the single entropy of each of the N events.

- The entropy can also be defined for continuous space. It is then referred to as the differential entropy:

$$H(\boldsymbol{x}) = -\int p(\boldsymbol{x})\log_2 p(\boldsymbol{x})d\boldsymbol{x}$$

- We can also define conditional entropy, which is as follows:

$$H(\boldsymbol{y}|\boldsymbol{x}) = -\int p(\boldsymbol{x}) \left[\int p(\boldsymbol{y}|\boldsymbol{x}) \ln p(\boldsymbol{y}|\boldsymbol{x}) d\boldsymbol{y}\right] d\boldsymbol{x}$$

  with the property $H(\boldsymbol{x}, \boldsymbol{y}) = H(\boldsymbol{x}) + H(\boldsymbol{y}|\boldsymbol{x}) = H(\boldsymbol{y}) + H(\boldsymbol{x}|\boldsymbol{y})$

- Another well-known measurement is the Kullback-Leiber divergence (also referred to as relative entropy):

$$\mathrm{KL}(p(\boldsymbol{x})||q(\boldsymbol{x})) = -\int p(\boldsymbol{x}) \ln \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{x}$$

  Some properties of this divergence are:

  - Always positive: $\mathrm{KL}(p||q) \geq 0$
  - If $\mathrm{KL}(p||q) = 0$, then $p = q$ (if p,q are sufficient regular, i.e. strictly positive and integral defined)
  - The triangular inequality does not hold for KL, thus it is not a distance measure: $\mathrm{KL}(p||q) + \mathrm{KL}(q||r) \ngeq \mathrm{KL}(p||r)$

- Mutual information describes the amount of information that is shared among x and y:

$$I(\boldsymbol{x}; \boldsymbol{y}) = \mathrm{KL}(p(\boldsymbol{x}, \boldsymbol{y})||p(\boldsymbol{x}), p(\boldsymbol{y})) = H(\boldsymbol{x}) - H(\boldsymbol{x}|\boldsymbol{y}) = H(\boldsymbol{y}) - H(\boldsymbol{y}|\boldsymbol{x})$$

  In other words, how much information about y do I get by observing x. In a diagram, mutual information can be visualized as follows:

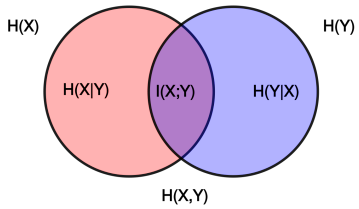
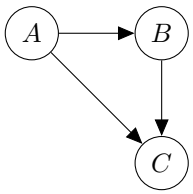


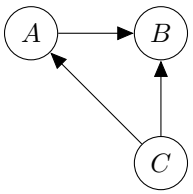
Figure 2: Visualizing the relationship between mutual information and entropy.

2 Probabilistic graphical models

- It is often beneficial to visualize a probabilistic model as a diagram, which we call *(probabilistic) graphical models*.
- They are good for:
  - causal reasoning/modeling
  - calculating inference and conditional distributions efficiently
  - Designing and communicating statistical model
  - Encoding (conditional) independence relations
- Note that there are often multiple ways to express the same probability distribution. For example, take a joint distribution $p(A, B, C)$, which we can either write as $p(A, B, C) = p(A)p(B|A)p(C|A, B)$ (see Figure 3a) or $p(A, B, C) = p(C)p(A|C)p(B|A, C)$ (see Figure 3b). Nevertheless, what we are interested in is the graphical representation with the least number of edges, as e.g. if A and B are independent (conditionally on C), we can drop the edge between those.



(a) $p(A, B, C) = p(A)p(B|A)p(C|A, B)$



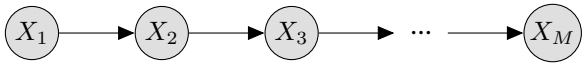
(b) $p(A, B, C) = p(C)p(A|C)p(B|A, C)$

Figure 3: Two different graphical models (here Bayesian Networks) for the same joint distribution $p(A, B, C)$.

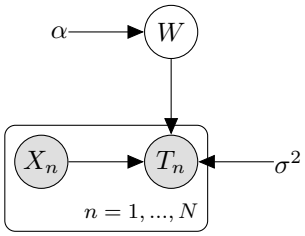
- We distinguish between directed acyclic graphs, which we call *Bayesian networks* (BN), and undirected graphs, which are *Markov Random Fields* (MRF)

2.1 Bayesian Networks

- There is a simple way for creating a Bayesian network for a given statistical model.
  1. Determine the ordering of the variables ("*topological ordering*")
  2. In this ordering, call the parents of the random variable X_i: pa_i, or $\text{pa}(X_i)$ which is a subset of variables with lower ordering: $\text{pa}_i \subseteq \{1, ..., i-1\}$. The joint probability distribution can be written as:
$$p(X_1, ..., X_M) = \prod_{i=1}^{M} p(X_i|X_{\text{pa}_i})$$
  3. In the graphical model, draw an edge from X_j to X_i if $j \in \text{pa}_i$
- Example: (first-order) Markov Chain
  - The joint probability distribution of a Markov Chain can be expressed by:
$$p(X_1, ..., X_M) = p(X_1) \cdot \prod_{i=2}^{M} p(X_i|X_{i-1})$$
  - The corresponding Bayesian Network looks as follows:



    where the filling expresses that X_i is an observed variable.
- Example: Regression
  - Suppose we have a simple regression problem where we want to learn parameters W to predict targets T from input X. We further assume that we know our sensory noise σ^2, and have a prior with hyperparameters α.



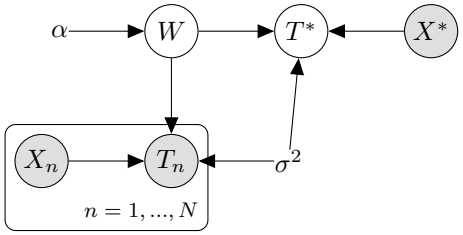
– We can express this in the graphical model above, which represents the probability distribution

$$p(W, \{T_n\}, \{X_n\} | \alpha, \sigma^2) p(W|\alpha) \prod_{n=1}^{N} \left[p(T_n|X_n, W, \sigma^2) p(X_n) \right]$$

Note that in the graphical model, α and σ^2 are assumed to be fixed and known, and the "plate" can be interpreted as copying the content N times (i.e. we have N X_i and T_i variables with the same edges).

Also, if desired, we could have used a constant for the data points X_i as well as these are often assumed to be fixed.

– If we also want to express the predictive distribution $p(T^*|X^*, W, \{T_n\}, \{X_n\}, \alpha, \sigma^2)$, we can extend our model as follows:



2.1.1 Conditional independence and D-separation

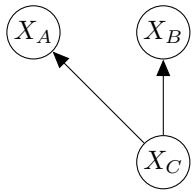
- A useful property of graphical models is that we can easily study the independence relations between random variables in our model.
- We call X and Y being independent iff $p(X, Y) = p(X)p(Y)$. The notation for this is $X \perp\!\!\!\perp Y$
- X is *conditionally* independent of Y given Z if $p(X, Y|Z) = p(X|Z)p(Y|Z)$. The notation for this is $X \perp\!\!\!\perp Y|Z$. Note that if X and Y are generally independent, we can also write $X \perp\!\!\!\perp Y|\emptyset$
- For proving/testing conditional independence, we can use **d-separation**. Supposed A, B, C are sets of variables. If A is d-separated from B given C, then $p(X_A, X_B|X_C) = p(X_A|X_C)p(X_B|X_C)$, which we can also write as $A \perp B|C \implies X_A \perp\!\!\!\perp X_B|X_C$
- Note that the other way round, $X_A \perp\!\!\!\perp X_B|X_C \not\Rightarrow A \perp B|C$ is not always valid (but mostly) as we will show in a later example. Hence, if A and B are not d-separated, it does not necessarily mean that X_A and X_B are not conditional independent.
- The algorithm can be summarized as follows:

D-separation

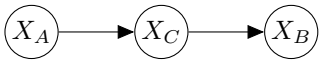
Given the sets of variables A, B, C:

1. Consider all paths (sequence of nodes, connected by edges, s.t. no node repeats) between any node in A and any node in B
2. Mark a path as blocked by C if
   (a) It contains a collider $... \to u \leftarrow ...$ such that u is not an ancestor of a node in C
   (b) It contains a non-collider $... \to u$, $... \to u \to ...$, $... \leftarrow u \to ...$ such that u is in C
3. If all paths are marked as blocked by C, then A is d-separated from B given C

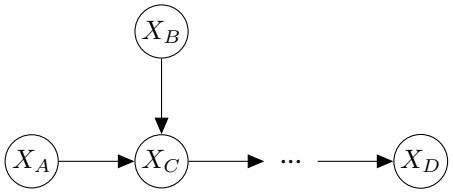
- Examples:
  - Consider the following graphical model:



    A is d-separated from B given C as the only way from B to A is through X_C, and it represents a non-collider: $X_A \perp\!\!\!\perp X_B | X_C$.
    Note that A is not d-separated from B given $\emptyset$ because X_C is then neither a non-collider nor a collider.
  - Consider the following graphical model:



    Similarly to the previous model, A is d-separated from B given C as the only way from B to A is through X_C, and it represents a non-collider: $X_A \perp\!\!\!\perp X_B | X_C$.
    However, here we can show a special case where conditional independence does not imply d-separation. Suppose that we model $p(C|A) = \delta_{C,A}$, hence being a deterministic mapping. Now, $C \perp\!\!\!\perp B|A$ holds because if we know A, we know C for certain. Nevertheless, the d-separation is not valid because there is a direct path from C to B!
  - Consider the following graphical model:



    A is d-separated from B given the empty set as X_C represents a collider which is not in the empty set: $X_A \perp\!\!\!\perp X_B | \emptyset$.
    A is *not* d-separated from B given C because X_C is then not a collider anymore: $A \not\perp\!\!\!\perp B|C$.
    A is *not* d-separated from B given D because X_C is an ancestor of a node in D, and hence, not a collider: $A \not\perp\!\!\!\perp B|D$.

2.1.2 Markov blanket

- A Markov blanket of a variable X_i is defined as the set of variables which are the parents, children or children's parents of X_i, except X_i itself:

$$\text{MB}(X_i) = \text{pa}_i \cup \text{ch}_i \cup \left(\text{pa}_{\text{ch}_i} \setminus i\right)$$

- The important property of the Markov blanket is that, for a random variable X_i in any BN, given its Markov blanket MB(X_i), it is conditionally independent of the rest of the graph:

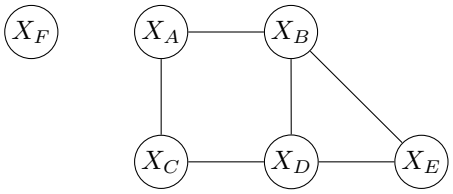
$$p\left(X_i | X_{\text{MB}(X_i)}, X_{\text{res}}\right) = p\left(X_i | X_{\text{MB}(X_i)}\right)$$

- Example: For the graphical model of the regression problem, the Markov blanket of T^* is $\text{MB}(T^*) = \{X^*, W\}$. This result is intuitive as once we have trained our model, we do not need to revisit our data or our prior over W. Note that σ^2 is a constant, and hence not in the Markov blanket.

2.2 Markov Random Fields

- A Markov Random Field is a undirected graphical models. Hence, our model consists now of two parts: the undirected graph G, and so called (maximum) cliques potentials $\{\psi_A\}$

- A clique in a undirected graph G is a fully connected subset of nodes. Hence, also single nodes are considered as a clique.
  - A clique is *maximal* if there is no clique that strictly contains it, i.e. we cannot add another node to the clique which is fully connected to all others.
  - Example: Consider the following graphical model:



    Then our maximum cliques are $\{X_A, X_C\}, \{X_A, X_B\}, \{X_C, X_D\}, \{X_B, X_D, X_E\}, \{X_F\}$

- We can now write our joint probability distribution in terms of the maximum cliques $\{\psi_A\}$:

$$p(x_1, ..., x_N) = \frac{1}{Z} \prod_A \psi_A(x_A)$$

  Note that we now need a normalization constant Z which we did not need for Bayesian networks. The reason for this is that clique potentials might not be normalized. The only requirement for them is to be positive for any x_A, and are thus often modeled by a energy function $\psi_A(x_A) = \exp(f(x_A))$ (hence the name *potential*)

- For the previous example, our probability distribution can be now written as:

$$p(x_A, ..., x_E) = \frac{1}{Z} \psi_{A,B}(x_A, x_B) \psi_{A,C}(x_A, x_C) \psi_{C,D}(x_C, x_D) \psi_{B,D,E}(x_B, x_D, x_E) \psi_F(x_F)$$

  where $Z = \sum_{x_A} \sum_{x_B} ... \sum_{x_F} \psi_{A,B}(x_A, x_B) \psi_{A,C}(x_A, x_C) ... \psi_F(x_F)$

- One disadvantage of undirected graphs, as we can see here, is that we need to calculate Z which grows exponentially with the number of variables.

- We can also define the properties of separation and Markov blanket for undirected graphs:

  Separation similarly to d-separation in BNs, two subsets of nodes A and B are separated given C if each path between a node in A and a node in B passes through (at least one) node C:

$$A \perp B | C \implies X_A \perp\!\!\!\perp X_B | X_C$$

  Markov blanket The Markov blanket for MRFs is defined as the neighbors of i, i.e. the nodes adjacent to i. In the previous example, the Markov blanket of X_B is: $\text{MB}(X_B) = \{X_A, X_D, X_E\}$

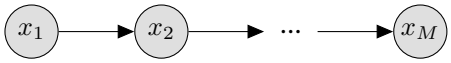
2.2.1 Converting Bayesian network to MRFs

- Sometimes it is the case that we want to represent a same statistical model which we have as a Bayesian network, also as a MRF. This is the case when we want to apply algorithms which are generally defined for undirected graphs (e.g. sum-product)

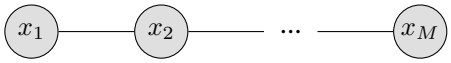
- The *Hammersley-Clifford* theorem states that any strictly positive, joint distribution $p(\boldsymbol{X}) \geq 0$ can be represented as a MRF. Hence, we can also do it with any Bayesian network

- Nevertheless, note that by converting a BN to a MRF, some properties/information might be lost, such as (conditional) independence relations.

- Examples:
  - Consider a first-order Markov chain:



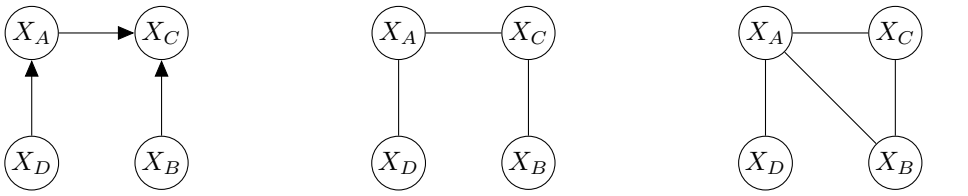
(a) BN



(b) MRF

As Bayesian network, we can represent it with the probability density function $p(x_1)\prod_{i=2}^{M} p(x_i|x_{i-1})$. In the case of the MRF, we have $\frac{1}{Z}\prod_{i=2}^{M}\psi_{i-1,i}(x_{i-1},x_i)$. Note that the prior $\psi_1(x_1)$ is integrated in $\psi_{1,2}(x_1,x_2)$ as the clique potentials are more flexible than the conditional probabilities in Bayesian networks.

– Consider the following Bayesian network:



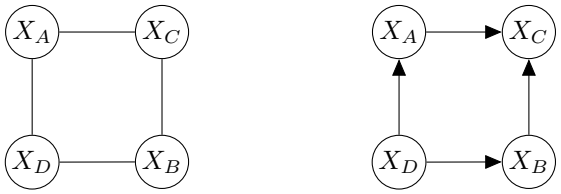
(a) BN (b) (Potential) MRF (c) MRF via mortalization

Figure 5: Comparing different conversions from BN to MRF

In this BN, $X_A \perp\!\!\!\perp X_B$, and (typically) $X_A \not\perp\!\!\!\perp X_B|X_C$. If we just replace the directed edges by undirected ones (see Figure 5b), we loose the independence $X_A \not\perp\!\!\!\perp X_B$. Furthermore, we would need to design the potentials in a way that is captures $p(X_C|X_A,X_B)$ correctly.

The easiest way is transforming BNs by **mortalization** (see Murphy, chapter 20.3). For each node, we "marry the parents", i.e. adding an edge between those if not already existing. By that, we ensure that we express all maximum clique potentials by the conditional probabilities of the Bayesian network. For example, see the MRF in Figure 5c which we got via mortalization. The clique potentials are now simply: $\psi_{A,D}(X_A,X_D) = p(X_D)p(X_A|X_D)$, $\psi_{A,B,C}(X_A,X_B,X_C) = p(X_C|X_A,X_B)p(X_B)$

– There are also MRFs which cannot be fully modeled by a Bayesian network. Consider for example the following graphical model:

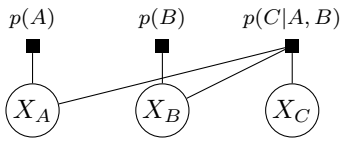


(a) MRF (b) (Potential) BN

The MRF models the following independence relations: $C \perp D|\{A,B\}$, $A \perp B|\{C,D\}$. If we would now want to model the same in a BN, we get into trouble as for the model in Figure 6b, as although $C \perp D|\{A,B\}$, we have $A \not\perp B|\{C,D\}$ because X_C is not a collider (is in set $\{C,D\}$) and also not non-collider.

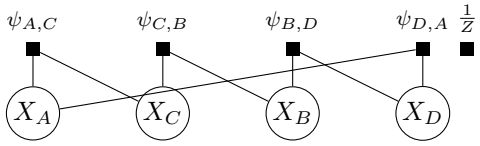
2.2.2 Factor graphs

- The third form of graphical models are Factor graphs. The idea is to represent the connections between variables by their factors in a bipartite graph. Hence, we have two sets of nodes: variable nodes, and factor nodes.
- Consider the statistical model $p(X_A,X_B,X_C) = p(X_A)p(X_B)p(X_C|X_A,X_B)$. The factor graph representation of this is:



- Similarly, for Markov Random Fields such as in Figure 6a, the factor graph representation is:

  where we could have merged $\frac{1}{Z}$ into any other factor if wanted.
- Note that in contrast ot MRFs, factor graph do not require to take the *maximum* cliques. Hence, for the same statistical model, there exist different factor graphs.



2.3 Learning in graphical models

- One of the applications of graphical models is to learn the conditional probabilities/potentials they model. We will first discuss the learning process for Bayesian networks, and afterwards do the same for MRFs

2.3.1 Learning in Bayesian networks

- Suppose that we replace all conditionals $p(x_i|\text{pa}_i)$ by a learnable function $\theta_i(x_i, \text{pa}_i)$ (equal to f_i with parameters θ_i) with the constraint of $\sum_{x_i} \theta_i(x_i, \text{pa}_i) = 1$.
- The likelihood of a dataset can then be written as:

$$p(\{\tilde{x}_{in}\}\,|\boldsymbol{\theta}) = \prod_{i=1}^{d}\prod_{n=1}^{N} \theta_i(\tilde{x}_{in}, \tilde{x}_{\text{pa}_i,n}) = \prod_{i}\prod_{n=1}^{N}\prod_{x_i}\prod_{x_{\text{pa}_i}} \theta_i(x_{in}, x_{\text{pa}_i,n})^{\delta(x_i=\tilde{x}_{in})\delta(x_{\text{pa}_i}=\tilde{x}_{\text{pa}_i,n})}$$

- Our objective to optimize is the log likelihood with the Lagrange multipliers:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{\lambda}) &= \sum_{i=1}^{d}\sum_{n=1}^{N}\sum_{x_i}\sum_{x_{\text{pa}_i}} \delta(x_i = \tilde{x}_{in}) \cdot \delta(x_{\text{pa}_i} = \tilde{x}_{\text{pa}_i,n}) \cdot \ln \theta_i(x_{in}, x_{\text{pa}_i,n}) - \sum_{i=1}^{d}\sum_{x_{\text{pa}_i}} \lambda_{i,x_{\text{pa}_i}} \left(\sum_{x_i} \theta_i(x_i, x_{\text{pa}_i}) - 1\right) \\ &= \sum_{i=1}^{d}\sum_{x_i}\sum_{x_{\text{pa}_i}} N(x_i, x_{\text{pa}_i}) \cdot \ln \theta_i(x_i, x_{\text{pa}_i}) - \sum_{i=1}^{d}\sum_{x_{\text{pa}_i}} \lambda_{i,x_{\text{pa}_i}} \left(\sum_{x_i} \theta_i(x_i, x_{\text{pa}_i}) - 1\right)\end{aligned}$$

  where $N(x_i, x_{\text{pa}_i})$ represent a counter of how often the combination of values for x_i and x_{pa_i} co-occur

- By taking the derivative, we get the following solution:

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{\lambda})}{\partial \theta_i(x_i, x_{\text{pa}_i})} &= \frac{N(x_i, x_{\text{pa}_i})}{\theta_i(x_i, x_{\text{pa}_i})} - \lambda_{i,x_{\text{pa}_i}} \stackrel{!}{=} 0 \\ \Leftrightarrow \theta_i(x_i, x_{\text{pa}_i}) &= \frac{N(x_i, x_{\text{pa}_i})}{\lambda_{i,x_{\text{pa}_i}}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{\lambda})}{\partial \lambda_{i,x_{\text{pa}_i}}} &= \sum_{x_i} \theta_i(x_i, x_{\text{pa}_i}) - 1 \stackrel{!}{=} 0 \\ \Leftrightarrow \sum_{x_i} \frac{N(x_i, x_{\text{pa}_i})}{\lambda_{i,x_{\text{pa}_i}}} &= 1 \\ \Leftrightarrow \lambda_{i,x_{\text{pa}_i}} &= \frac{1}{N(x_{\text{pa}_i})} \\ \implies \theta_i(x_i, x_{\text{pa}_i}) &= \frac{N(x_i, x_{\text{pa}_i})}{N(x_{\text{pa}_i})}\end{aligned}$$

  Hence, the optimal conditionals $\theta_i(x_i = a, x_{\text{pa}_i} = b)$ are simply the average number of times we have seen $x_i = a$ when $x_{\text{pa}_i} = b$.

- For each conditional θ_i, the optimum solely depends on x_i and its parents, and not over all variables as we had before (log-likelihood has the sum over all variables). This is why learning in Bayesian networks is fast and efficient

2.3.2 Learning in Markov Random Fields

- In the case of MRFs, we are interested in learning the potential ψ_A for all cliques. With a similar rewriting with the indicator function, we get the log-likelihood of the data as:

$$\mathcal{L}(\boldsymbol{\psi}, \boldsymbol{X}) = \sum_{n=1}^{N}\sum_{A}\sum_{x_A} \delta(x_A = \tilde{x}_A) \ln \psi_A(x_A) - N \ln Z = \sum_{A}\sum_{x_A} N(x_A) \ln \psi_A(x_A) - N \ln Z$$

- The derivative of the log-likelihood is:

$$\frac{\partial \mathcal{L}(\boldsymbol{\psi}, \boldsymbol{X})}{\partial \psi_A(x_A)} = \frac{N(x_A)}{\psi_A(x_A)} - \frac{N}{\psi_A(x_A)} \mathbb{E}_\psi[\delta(x_A = \cdot)] \stackrel{!}{=} 0$$

  where $\mathbb{E}_\psi[\delta(x_A = \cdot)]$ is the expected fraction of observations of x_A over all potentials $\boldsymbol{\psi}$. In other words, how much probability mass is assigned to states where $x_A = \tilde{x}_A$ (and other x anything), compared to all other states. This terms comes from our normalization constant Z which is of course influenced by our potentials.

- The optimal is therefore found when the fraction of observed $x_A = \tilde{x}_A$ is equal to the expected number of observations. To find a solution, we can use sampling to approximate the expectation:

$$\mathbb{E}_\psi[\delta(x_A = \cdot)] \approx \frac{N_\psi(x_A)}{N_\psi}$$

  where N_ψ is the sample size, and $N_\psi(x_A)$ the number of times we observed $x_A = \tilde{x}_A$ during sampling.

- Hence, the (approximated) optimum is:

$$\begin{aligned} &\frac{\partial \mathcal{L}(\boldsymbol{\psi}, \boldsymbol{X})}{\partial \psi_A(x_A)} = \frac{N(x_A)}{\psi_A(x_A)} - \frac{N}{\psi_A(x_A)} \frac{N_\psi(x_A)}{N_\psi} \stackrel{!}{=} 0 \\ \Leftrightarrow\ &\psi_A(x_A) = \frac{N(x_A)/N}{N_\psi(x_A)/N_\psi} \end{aligned}$$

  To interpret the result, if we sample less times x_A than in our dataset, we increase ψ_A. Otherwise, we reduce it. For stability, we can view this optimum as an update step, and repeat this procedure a couple of times until we converge

2.4 Inference in graphical models

- Another important aspect of graphical models is inference to be efficient. Here, our goal is to either marginalize out variables, or set some to observed, and calculate the posterior distribution of others
- Let's first consider again a first-order Markov chain. Its joint probability distribution can be expressed by:

$$p(x_1, ..., x_d) = \frac{1}{Z} \psi_{1,2}(x_1, x_2) \cdot \psi_{2,3}(x_2, x_3) \cdot ... \cdot \psi_{d-1,d}(x_{d-1}, x_d)$$

- Now suppose we want to calculate the marginal distribution $p(x_j)$. We can do this by marginalizing over all other variables:

$$\begin{aligned} p(x_j) &= \sum_{x_1} ... \sum_{x_{j-1}} \sum_{x_{j+1}} ... \sum_{x_d} p(x_1, ..., x_{j-1}, x_{j+1}, ..., x_d) \\ &= \sum_{x_1} ... \sum_{x_{j-1}} \sum_{x_{j+1}} ... \sum_{x_d} \frac{1}{Z} \psi_{1,2}(x_1, x_2) \cdot \psi_{2,3}(x_2, x_3) \cdot ... \cdot \psi_{d-1,d}(x_{d-1}, x_d) \end{aligned}$$

  Now, we can move the sums as the potentials only contain two variables, and are hence independent of all others:

$$\begin{aligned} p(x_j) &= \frac{1}{Z} \sum_{x_1} \sum_{x_2} ... \sum_{x_{j-1}} \sum_{x_{j+1}} ... \sum_{x_d} \psi_{1,2}(x_1, x_2) \cdot \psi_{2,3}(x_2, x_3) \cdot ... \cdot \psi_{d-1,d}(x_{d-1}, x_d) \\ &= \frac{1}{Z} \sum_{x_1} \sum_{x_2} ... \sum_{x_{j-1}} \psi_{1,2}(x_1, x_2) \cdot ... \psi_{j-1,j}(x_{j-1}, x_j) \cdot \\ &\qquad \sum_{x_{j+1}} \psi_{j,j+1}(x_j, x_{j+1}) \sum_{x_{j+2}} \psi_{j+1,j+2}(x_{j+1}, x_{j+2}) ... \sum_{x_d} \psi_{d-1,d}(x_{d-1}, x_d) \\ &= \frac{1}{Z} \underbrace{\sum_{x_{j-1}} \psi_{j-1,j}(x_{j-1}, x_j) \sum_{x_{j-2}} \psi_{j-2,j-1}(x_{j-2}, x_{j-1}) ... \sum_{x_1} \psi_{1,2}(x_1, x_2) \cdot}_{\mu_\alpha(x_j)} \\ &\qquad \underbrace{\sum_{x_{j+1}} \psi_{j,j+1}(x_j, x_{j+1}) \sum_{x_{j+2}} \psi_{j+1,j+2}(x_{j+1}, x_{j+2}) ... \sum_{x_d} \psi_{d-1,d}(x_{d-1}, x_d)}_{\mu_\beta(x_j)} \\ &= \frac{1}{Z} \mu_\alpha(x_j) \mu_\beta(x_j) \end{aligned}$$

- This shows us that we can split the marginal into two separate parts, μ_α and μ_β, which both are a recursive functions:

$$\mu_\alpha(x_j) = \sum_{x_{j-1}} \psi_{j-1,j}(x_{j-1}, x_j)\mu_\alpha(x_{j-1})$$

$$\mu_\alpha(x_j) = \sum_{x_{j+1}} \psi_{j,j+1}(x_j, x_{j+1})\mu_\beta(x_{j+1})$$

  We can view these recursive functions also as *messages* which are passed across the chain. $\mu_\alpha(x_j)$ and $\mu_\beta(x_j)$ would be then the incoming messages of x_j, and $\mu_\alpha(x_{j+1})$ and $\mu_\beta(x_{j-1})$ the outgoing messages.

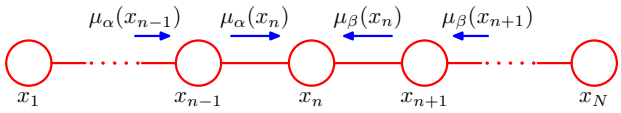


Figure 7: Message passing in graphical models (Bishop 8.38)

- Even the normalization term Z can be expressed by the messages: $Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$
- The benefit of this recursive implementation is that we don't have to take a sum over d variables ($\mathcal{O}(K^d)$), but only have to take sums over two variables at a time ($\mathcal{O}(K^2 d)$). Hence, the computational time scales linear with number of nodes instead exponential.

  Furthermore, we can share calculations between nodes, as the same messages can be re-used. Thus, calculating the marginals for all variables reduces to $\mathcal{O}(2 \cdot K^2 d) = \mathcal{O}(K^2 d)$

2.4.1 Sum-product algorithm

- The message passing idea is not limited to a Markov chain, but can be applied to any graph. For simplicity, we focus here on *trees*, i.e. graphs where all nodes are connected, but without any loops/cycles, independent of the direction of the edges.
- In our discussion, we will focus on factor graphs as they represent the most general form of graphical models. Furthermore, cycles in MRFs can often be resolved in a factor graph so that we have less problem getting a tree-structured graph
- Now we will send messages from variables to factors, and from factors to variables. As a result, we get the marginalization for all variables. This algorithm is called **sum-product** algorithm as we only take sums and products

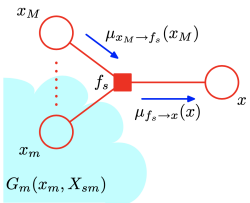


Figure 8: Message passing to and from a factor node (Bishop 8.47)

- The messages can be calculated as follows where we start at leaf nodes, and recursively go to the center of the tree:

Factor→Variable: $\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_{\alpha\setminus i}} f_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in\alpha\setminus i} \mu_{j\to\alpha}(x_j)$

If α leaf node: $\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_\alpha} f_\alpha(\boldsymbol{x}_\alpha)$

Variable→Factor: $\mu_{j\to\alpha}(x_j) = \prod_{\beta\in \mathrm{ne}(j)\setminus\alpha} \mu_{\beta\to j}(x_j)$

If j leaf node: $\mu_{j\to\alpha}(x_j) = 1$

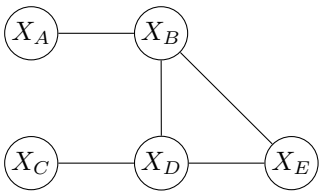
The marginalizations/beliefs are in the end:

$$\textbf{Variable belief: } p(x_i) = \frac{1}{Z} \prod_{\alpha \in \text{ne}(i)} \mu_{\alpha \to i}(x_i) \quad \text{where } Z = \sum_{x_i} \prod_{\alpha \in \text{ne}(i)} \mu_{\alpha \to i}(x_i)$$

$$\textbf{Factor belief: } p(\boldsymbol{x}_\alpha) = \frac{1}{Z} f_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in \text{ne}(\alpha)} \mu_{i \to \alpha}(x_i)$$

where a factor belief is the marginalization of all variables except those with an direct edge to the factor.

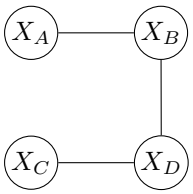
- The complexity of this algorithm scales with $\mathcal{O}(EK^M)$ where E are the number of edges, M the maximum number of variables that are connected to a factor, and K the maximum domain size
- Note that this algorithm is only exact on trees or forest (group of trees). For general graphs, we can first bring them into the shape of a tree by e.g. **variable elimination**
  - Given a MRF or factor graph, we will marginalize out these variable nodes which cause a loop in our graphical model. Let's for example consider this model:



  - We can do this by simply writing down the joint probability distribution and determine a order of variables $X_1, ..., X_M$ where the last variables, e.g. X_M, should be those which are eliminated (likely the easiest to marginalize). We then sort the sums according to the selected order.
    In our example, we want to eliminate X_E as it has the least connections from those in the loop. The sum order is therefore:

$$p(X_A, ..., X_E) = \frac{1}{Z} \sum_{X_B} \sum_{X_A} \psi_{A,B}(X_A, X_B) \sum_{X_D} \psi_{B,D}(X_B, X_D) \sum_{X_C} \psi_{C,D}(X_C, X_D) \sum_{X_E} \psi_{B,E}(X_B, X_E) \psi_{D,E}(X_D, X_E)$$

  - Finally, replace the marginalized terms by a new factor, and remove node from graph. In the example, we would replace $\tau(X_B, X_D) = \sum_{X_E} \psi_{B,E}(X_B, X_E) \psi_{D,E}(X_D, X_E)$. This can be merged into the potential $\psi_{B,D}$, hence not changing the graph structure. Our final graph is a tree again, and looks as follows:



- The sum-product algorithm so far only calculated marginals. To set some observed variables as observed, we can use the same algorithm, but simply add additional "hard evidence" factor nodes, which is nothing else than a hard prior that x_j has the value ξ_j:

$$f_{\xi_j}(x_j) = \delta(x_j = \xi_j)$$

Then if we apply the sum product algorithm on the extended graph again, we get the conditionals $p(x_i | x_j = \xi_j)$

2.4.2 Max-sum algorithm

- The sum-product algorithm calculates the full marginal distribution $p(x_j)$, but sometimes, we just want to know the most likely value of x_j, especially if we set some variables to observed states

- As it turns out, we can use a very similar algorithm for this, but simply replace sums by maximum operators, and products by sums.
- First, let's consider what the optimum is in the general case:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \prod_{\alpha} f_\alpha(\boldsymbol{x}_\alpha)$$

  where we can ignore the normalization constant. Furthermore, to simplify optimization, we can apply the log:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \sum_{\alpha} \ln f_\alpha(\boldsymbol{x}_\alpha)$$

- Our messages passed across the graph are then as follows:

Factor→Variable: $\nu_{\alpha\to i}(x_i) = \max_{\boldsymbol{x}_{\alpha\setminus i}} \log f_\alpha(\boldsymbol{x}_\alpha) + \sum_{j\in\alpha\setminus i} \nu_{j\to\alpha}(x_j)$

If α leaf node: $\nu_{\alpha\to i}(x_i) = \max_{\boldsymbol{x}_{\alpha\setminus i}} \log f_\alpha(\boldsymbol{x}_\alpha)$

Variable→Factor: $\nu_{j\to\alpha}(x_j) = \sum_{\beta\in \mathrm{ne}(j)\setminus\alpha} \nu_{\beta\to j}(x_j)$

If j leaf node: $\nu_{j\to\alpha}(x_j) = 0$

- The maximum beliefs/marginals are:

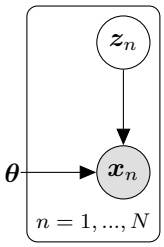
Max-marginals: $q_i(x_i) = \sum_{\alpha\in\mathrm{ne}(i)} \nu_{\alpha\to i}(x_i)$

- In the case that $q_i(x_i)$ has a unique maximum, we can simply take the argmax to get the optimum: $x_i^* = \arg\max_{x_i} q_i(x_i)$

  If this is not the case, we need to run the Viterbi algorithm (Bishop 8.4.5, we do not go in detail for this in the exam) to get the global optimum. The general idea is that some optima might depend on each other. For example, if we have something similar to a XOR, $(x_0 = 0, x_1 = 1)$ and $(x_0 = 1, x_1 = 0)$ are the optimums, but just looking at the independent marginals, $(x_0 = 0, x_1 = 0)$ and $(x_0 = 1, x_1 = 1)$ would be also optima. We can prevent this by slightly extending the messages passed around.

3 Variational Expectation Maximization

- The expectation maximization algorithm can be viewed from a different angle where we focus on the latent variables z_n
- For each observed data point $\boldsymbol{x}_n$, we create a latent variable $\boldsymbol{z}_n$ which *explains* the observation by our underlying model
  - For example, in case we have a mixture model, the latent variable z_n indicates from which component $\boldsymbol{x}_n$ was created
  - We create the model in such a way that $p(X, Z|\theta)$ can be easily calculated
- As a graphical model, we can represent it as follows:



- The objective we want to optimize is the log-likelihood

$$\ln p(\boldsymbol{X}|\boldsymbol{\theta}) = \ln \left\{ \sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) \right\}$$

- However, note that in the standard setting, we are not given $\boldsymbol{Z}$ so that we need to work with the posterior $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta})$ as knowledge over latent variables. Hence, we calculate the log-likelihood under expectation of our posterior:

$$\mathbb{E}_{\boldsymbol{Z} \sim p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta})} \left[\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) \right] = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$$

- As the optimization problem for the posterior and the parameters has often no analytical closed-form solution, we optimize both sequentially, leading to the general idea of the EM algorithm

  E-step Find the posterior distribution $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$ where $\boldsymbol{\theta}^{\text{old}}$ means that we fix the other parameters

  M-step Optimize the log-likelihood with respect to parameters $\boldsymbol{\theta}$ while keeping the posterior fixed

$$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \arg\max_{\boldsymbol{\theta}} \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$$

- In case we want to find the MAP instead of the MLE, we simply have to add the prior term $\ln p(\boldsymbol{\theta})$ to $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ in the M-step

3.1 Generalizing EM

- We can further generalize the EM algorithm to a form, which was originally used to prove its optimization objective.
- However, we can also look at it from a different perspective. It might be sometimes the case, that the posterior $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$ is hard to determine. Instead, we can introduce an approximation $q(\boldsymbol{Z})$ for which we can choose the form ourselves (e.g. Gaussian, or softmax distribution over discrete states, etc.)
- Using this approximation, we can calculate the log likelihood by:

$$\begin{aligned}
\ln p(\boldsymbol{X}|\boldsymbol{\theta}) &= \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} q(\boldsymbol{z}_n) \ln \frac{p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{\theta})}{p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})} \\
&= \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} q(\boldsymbol{z}_n) \ln \frac{p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{\theta})}{q(\boldsymbol{z}_n)} \frac{q(\boldsymbol{z}_n)}{p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})} \\
&= \sum_{n=1}^{N} \left[\mathbb{E}_{q_n} \left[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{\theta}) \right] + H(q_n) + \underbrace{\text{KL}\left(q_n(\boldsymbol{z}_n) || p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta}) \right)}_{\geq 0} \right]
\end{aligned}$$

- We can define an ELBO for our objective function:

$$\ln p(\boldsymbol{X}|\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) \geq \sum_{n=1}^{N} \left[\mathbb{E}_{q_n}\left[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{\theta})\right] + H(q_n)\right] = \mathcal{L}(\boldsymbol{\theta}, q)$$

- Hence, for any set of distributions $\{q_n(\boldsymbol{z}_n)\}_{n=1}^N$, we can define a lower bound optimization objective $\mathcal{L}(\boldsymbol{\theta}, q)$, which is equal to $\mathcal{L}(\boldsymbol{\theta})$ iff $q_n(\boldsymbol{z}_n) = p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})$
- During the E-step, we optimize $\mathcal{L}(\boldsymbol{\theta}, q) = \ln p(\boldsymbol{X}|\boldsymbol{\theta}) - \mathrm{KL}(q||p)$ regarding q. As $\ln p(\boldsymbol{X}|\boldsymbol{\theta})$ is independent of q, we minimize the KL-divergence, meaning that we push $q_n(\boldsymbol{z}_n)$ to be similar to $p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})$
- In the M-step, we increase $p(\boldsymbol{X}|\boldsymbol{\theta})$ by optimizing the parameters $\boldsymbol{\theta}$. Note that as q is fixed in this step, the KL-divergence will increase as well, due to q being now sub-optimal for the new parameters. Hence, we perform another E-step again a.s.o.
- Keep in mind that when optimizing $\mathcal{L}(\boldsymbol{\theta}, q)$, we need to add Lagrangian for the constraint that q_n is a valid PDF:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, q) = \mathcal{L}(\boldsymbol{\theta}, q) + \sum_{n=1}^{N} \lambda_n \left(\sum_{\boldsymbol{z}_n} q_n(\boldsymbol{z}_n) - 1\right)$$

  Depending on our model, we might need to add additional Lagrangian to $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q)$ (e.g. over $\boldsymbol{\pi}$ in mixture models)
- In summary, the variational EM algorithm can be summarized as follows:

Variational EM algorithm

1. Choose initial $\boldsymbol{\theta}^{(0)}$
2. Iterate until $\Delta\boldsymbol{\theta}^{(t)} < \epsilon$
   (a) **E-step**: Given fixed $\boldsymbol{\theta}^{(t)}$,
      – If the posterior can be determined, evaluate $q_n^{(t)}(\boldsymbol{z}_n) = p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta}^{(t)})$
      – Otherwise, use Variational EM by increasing $\tilde{\mathcal{L}}(\boldsymbol{\theta}^{(t)}, q)$ over q, e.g. gradient ascend
   (b) **M-step**: Given fixed $q^{(t)}$,
      – Solve, if possible, $\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \tilde{\mathcal{L}}(\boldsymbol{\theta}, q^{(t)})$
      – Otherwise, increase $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q^{(t)})$ over $\boldsymbol{\theta}$, e.g. gradient ascend

3.1.1 Example: Mixture of multivariate Bernoulli's

- As an example, we outline the EM algorithm to optimize a mixture of multivariate Bernoulli's here. Our model distribution looks like:

$$p(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \prod_{i=1}^{D} \mu_{ki}^{x_{ni}} (1 - \mu_{ki})^{1 - x_{ni}} \quad \text{where} \quad \sum_{k=1}^{K} \pi_k = 1$$

- Optimizing the parameters without the EM algorithm does not has a closed-form solution because of a sum in the log:

$$\log p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \log \sum_{z_n=1}^{K} \pi_{z_n} p(\boldsymbol{x}_n|\boldsymbol{\mu}_{\boldsymbol{z_n}})$$

- Our EM objective is instead:

$$\mathcal{L}(q, \boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} q_n(\boldsymbol{z}_n) \left\{ \left(\log \pi_{z_n} + \sum_{i=1}^{D} \left[x_{ni} \log \mu_{z_n,i} + (1 - x_{ni}) \log(1 - \mu_{z_n,i}) \right] \right) - \log q_n(\boldsymbol{z}_n) \right\}$$

$$\tilde{\mathcal{L}}(q, \boldsymbol{\mu}, \boldsymbol{\pi}, \lambda, \{\lambda_n\}) = \mathcal{L}(q, \boldsymbol{\mu}, \boldsymbol{\pi}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) + \sum_{n=1}^{N} \lambda_n \left(\sum_{\boldsymbol{z}_n} q_n(\boldsymbol{z}_n) - 1 \right)$$

- The update equation we get by deriving $\tilde{\mathcal{L}}(q, \boldsymbol{\mu}, \boldsymbol{\pi}, \lambda, \{\lambda_n\})$ with respect to the according parameters

  E-Step Optimize q

$$\frac{\partial \tilde{\mathcal{L}}}{\partial q_n(z_n)} = \log \pi_{z_n} + \left[\sum_{i=1}^{D} x_{ni} \log \mu_{z_n,i} + (1 - x_{ni}) \log(1 - \mu_{z_n,i})\right] - \log q_n(z_n) - 1 + \lambda_n = 0$$

$$\implies q_n(z_n) = \exp(\lambda_n - 1)\pi_{z_n} \prod_{i=1}^{D} \mu_{z_n,i}^{x_{ni}} (1 - \mu_{z_n,i})^{1-x_{ni}}$$

  By solving for the Lagrangian λ_n, we would see that $\exp(\lambda_n - 1) = 1/(\sum_{z_n} q(z_n))$, and hence, being a normalization factor.

  M-step Optimize parameters $\boldsymbol{\pi}$:

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \pi_k} = \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} q_n(\boldsymbol{z}_n) \frac{z_{nk}}{\pi_k} + \lambda \stackrel{!}{=} 0$$

$$\Leftrightarrow \pi_k = \frac{1}{\lambda} \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} z_{nk} q_n(\boldsymbol{z}_n)$$

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \lambda} = \sum_{k=1}^{K} \pi_k - 1 \stackrel{!}{=} 0$$

$$\Leftrightarrow 1 = \sum_{k=1}^{K} \frac{1}{\lambda} \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} z_{nk} q_n(\boldsymbol{z}_n)$$

$$\Leftrightarrow \lambda = \sum_{k=1}^{K} \sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} z_{nk} q_n(\boldsymbol{z}_n) = \sum_{n=1}^{N} 1 = N$$

$$\implies \pi_k = \frac{\sum_{n=1}^{N} \sum_{\boldsymbol{z}_n} z_{nk} q_n(\boldsymbol{z}_n)}{N}$$

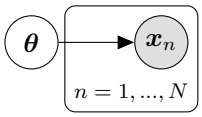
  Optimize parameter $\boldsymbol{\mu}$:

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \mu_{ki}} = \sum_{n=1}^{N} q_n(k) \left[\frac{x_{ni}}{\mu_{ki}} - \frac{1 - x_{ni}}{1 - \mu_{ki}}\right]$$

$$\implies \mu_{ki} = \frac{\sum_{n=1}^{N} q_n(k) x_{ni}}{\sum_{n=1}^{N} q_n(k)}$$

3.2 Variational Inference: Variational Bayes

- In standard EM, we treat $\boldsymbol{\theta}$ to be a parameter that we want to optimize with a single value. However, looking from a Bayesian perspective, we would also treat it as a (latent) random variable and calculate its posterior, leading to the following graphical model:



- Note that we could also include latent variables $\boldsymbol{z}_n$ as in the EM algorithm, but for generality, we could simply include them in $\boldsymbol{\theta}$ as the calculations are exactly the same

- We perform the same derivation as for variational EM, with the difference, that q is now over all parameters $\boldsymbol{\theta}$, giving us the following objective:

$$L = \log p(X) \geq \mathcal{L}(q)$$

$$\text{where } \mathcal{L}(q) = \int q(\boldsymbol{\theta}) \log\left[p(X|\boldsymbol{\theta}) p(\boldsymbol{\theta})\right] d\boldsymbol{\theta} - \int q(\boldsymbol{\theta}) \log q(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log\left(p(X|\boldsymbol{\theta}) p(\boldsymbol{\theta})\right)\right] + H(q)$$

  Note that we use integrals here as parameters are often continuous.

- The posterior can be found by optimizing q:

$$p(\boldsymbol{\theta}|\boldsymbol{X}) = \arg\max_{q(\boldsymbol{\theta})} \mathcal{L}(q)$$

- In practice, we restrict $q(\boldsymbol{\theta})$ to be in a function family Q for which we can calculate the maximum of $\mathcal{L}(q)$ analytically, or optimize by gradient ascend.
- One example for Q is to assume that all (or at least certain parts) of the parameters are independent of each other, hence:

$$q(\boldsymbol{\theta}) \in Q = \left\{ \prod_{i=1}^{D} q_i(\boldsymbol{\theta}_i) \right\}$$

  where $\boldsymbol{\theta}_i$ can be either a scalar or a vector.
- For this case, we can simplify the objective:

$$\tilde{L}(q) = \int \left(\prod_{i=1}^{D} q_i(\boldsymbol{\theta}_i) \right) \log\left[p(X|\boldsymbol{\theta})p(\boldsymbol{\theta})\right] d\boldsymbol{\theta} - \sum_{i=1}^{D} \int q_i(\boldsymbol{\theta}_i) \log q_i(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i + \sum_{i=1}^{D} \lambda_i \left(\int q_i(\theta_i) d\boldsymbol{\theta}_i - 1 \right)$$

$$\frac{\partial \tilde{L}}{\partial q_i(\boldsymbol{\theta}_i)} = \int \left(\prod_{j \neq i} q_j(\boldsymbol{\theta}_j) \right) \log\left[p(X|\boldsymbol{\theta})p(\boldsymbol{\theta})\right] d\boldsymbol{\theta}_{\setminus i} - \log q_i(\theta_i) - 1 + \lambda_i$$

$$\Leftrightarrow q_i(\boldsymbol{\theta}_i) = \exp\left(\lambda_i - 1\right) \exp\left\{ \int \left(\prod_{i \neq j} q_j(\boldsymbol{\theta}_j) \right) \log\left(p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})\right) \right\}$$

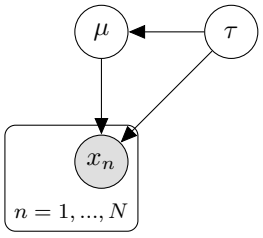
- Hence, our approximated posterior over $\boldsymbol{\theta}_i$ is the expectation of $\log p(X, \boldsymbol{\theta})$ over all other parameters:

$$p(\boldsymbol{\theta}|\boldsymbol{X}) = \textstyle\prod_{i=1}^{D} q_i(\boldsymbol{\theta}_i), \quad q_i(\boldsymbol{\theta}_i) = \frac{1}{Z} \exp\left(\mathbb{E}_{q_{\setminus i}}\left[\log p(\boldsymbol{X}, \boldsymbol{\theta})\right]\right)$$

  We iterate this update for each q until convergence.
- Note that Variational Bayes is highly related to Gibbs sampling as there, we do not calculate the full probability distribution q, but simply alternate between sampling from the different q_i's. By iterating until convergence/for a sufficient number of time, we also reach the true posteriors.

3.2.1 Example: Gaussian with mean and variance as latent variables

- We consider the following graphical model:



  where

$$p(\boldsymbol{X}|\mu, \tau) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \tau^{-1})$$
$$p(\tau) = \text{Gamma}(\tau|a_0, b_0)$$
$$p(\mu|\tau) = \mathcal{N}(\mu|\mu_0, (\lambda_0\tau)^{-1})$$

- Now, let's assume that we approximate the posteriors by $q(\tau, \mu) = q(\tau)q(\mu)$. Our objective (excluding the Lagrangian) is:

$$\mathcal{L}(q_\mu, q_\tau) = \int q_\mu(\mu) q_\tau(\tau) \left[\log p(\boldsymbol{X}|\mu, \tau) p(\mu|\tau) p(\tau)\right] + H(q_\mu) + H(q_\tau)$$

- When solving this, we will end up with:

$$q_\mu(\mu) = \mathcal{N}\left(\mu \middle| \frac{\lambda_0\mu_0 + N\overline{x}}{\lambda_0 + N}, [(\lambda_0 + N)\mathbb{E}_{q_\tau}[\tau]]^{-1}\right)$$
$$q_\tau(\tau) = \text{Gamma}\left(\tau \middle| a_0 + \frac{N}{2}, b_0 + \frac{1}{2}\mathbb{E}_{q_\mu}\left[\sum_n (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\right]\right)$$
$$\Rightarrow p(\mu, \tau|\boldsymbol{X}) \approx q_\mu(\mu)q_\tau(\tau)$$

3.2.2 Combining Variational EM and Variational Bayes

- The Variational EM algorithm and the variational Bayes are strongly related. In fact, we can generalize the framework to combine both of them
- Our goal is to optimize the parameters $\boldsymbol{\theta}$, and marginalize over latent variables $\boldsymbol{Z}$, leading to the objective:

$$\ln p(\boldsymbol{X}) = \mathbb{E}_{q(\boldsymbol{Z},\boldsymbol{\theta})}[\ln p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\theta})] + \underbrace{H(q_Z) + H(q_\theta)}_{\text{assume } q(\boldsymbol{Z},\boldsymbol{\theta}) = q_Z(\boldsymbol{Z})q_\theta(\boldsymbol{\theta})} + \underbrace{\text{KL}\left(q_Z q_\theta || p(\boldsymbol{Z}, \boldsymbol{\theta}|\boldsymbol{X})\right)}_{\geq 0}$$

- Again, we can optimize an ELBO instead:

$$\ln p(\boldsymbol{X}) \geq \mathbb{E}_{q_Z q_\theta}[\ln p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\theta})] = \mathcal{L}(q_Z, q_\theta)$$

- Assuming $\tilde{\mathcal{L}}(q_Z, q_\theta)$ being the objective function including the Lagrangian, we have the following steps:

  $\mathbb{E}_Z$**-step** : $\max_{q_Z} \tilde{\mathcal{L}}(q_Z, q_\theta)$

  $\mathbb{E}_\theta$**-step** : $\max_{q_\theta} \tilde{\mathcal{L}}(q_Z, q_\theta)$

- As this is a generalization, we can find both the EM algorithm and variational Bayes in it. The EM algorithm is found if we choose $q_\theta(\theta) = \delta(\theta' - \theta)$, where θ' is optimized $\Rightarrow$ equal to replacing q_θ with MLE/MAP solution
- For variational Bayes, we can simply ignore the Z part as we fully focus on q_θ. The iteration over the $\mathbb{E}_\theta$-step is equal to variational Bayes

3.3 Variational Auto-Encoder (VAE)

- One implementation with neural networks of this variational framework are VAEs, where we model the likelihood by:

$$p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) = \underbrace{p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\theta}_2)}_{\text{decoder}} \underbrace{p(\boldsymbol{Z}|\boldsymbol{\theta}_1)}_{\text{encoder/prior}}$$

- The aim is to find a lower-dimensional representation Z of the data X (*encode* X). For approximating the posterior, we use again $q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\lambda}) \approx p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$, where q is now a neural network
- As a prior distribution, we use e.g. $p(\boldsymbol{Z}|\boldsymbol{\theta}_1) \sim \mathcal{N}(0, 1)$ which encourages the latents to be independent. Furthermore, by using this prior, we can treat VAEs as a non-linear version of PCA (in case we choose student-t distribution, we would get non-linear ICA)
- Optimize the ELBO of the log-likelihood via SGD:

$$\mathcal{L}(\boldsymbol{\theta}) = \ln p(\boldsymbol{X}|\boldsymbol{\theta}) = \ln \int p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\theta}) p(\boldsymbol{Z}|\boldsymbol{\theta}) d\boldsymbol{Z}$$
$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbb{E}_{q(Z|X,\lambda)}[\ln p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\theta}_2) + \ln p(\boldsymbol{Z}|\boldsymbol{\theta}_1)] + H(q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\lambda}))$$

- To prevent the integral, we can sample instead. To make these samples differentiable, we need to use the reparameterization trick:

$$z^{(k)} \sim q(z|x, \lambda) \Rightarrow z = g_\lambda(x, \epsilon), \epsilon \sim p(\epsilon)$$

  As for a multivariate Gaussian with diagonal covariance, we have $g_\lambda(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu}_\lambda(x) + \boldsymbol{\sigma}_\lambda \odot \boldsymbol{\epsilon}, \epsilon \sim \mathcal{N}(0, 1)$

4 Sampling methods

- In the previous chapter, we have seen that we can perform inference by approximating the posterior distribution. However, as alternative, we can also consider Monte Carlo techniques, i.e. sampling
- In most practical cases, we often want to evaluate expectations over the posterior. This we can approximate by an average over samples:

$$\mathbb{E}_{p(\boldsymbol{x})}\left[f(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{n=1}^{N} f(\boldsymbol{x}_n), \quad \boldsymbol{x}_n \sim p(\boldsymbol{x})$$

This can be for example used for prediction:

$$p(y^*|x^*) = \int p(y^*|x^*,\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})d\boldsymbol{\theta} = \mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})}\left[p(y^*|x^*,\boldsymbol{\theta})\right] \approx \frac{1}{K}\sum_{k=1}^{K} p(y^*|x^*,\boldsymbol{\theta}^{(k)}), \quad \boldsymbol{\theta}^{(k)} \sim p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})$$

- Hence, we will look at different techniques for sampling from more complex distributions than standard Gaussians

4.1 Regular Sampling

- As an introduction, we look at how we can sample from simple, known distributions, and verify the correctness of the Monte Carlo approximation
- Assume we draw N samples from $p(z)$: $z_i \sim p(z)$, $i = 1, ..., N$
- We can calculate $\mathbb{E}\left[f\right] \approx \widehat{E\left[f\right]} = \langle f \rangle = \frac{1}{N}\sum_{n=1}^{N} f(z_i)$ for approximating the expectation. Note that we introduced here the different notations for the Monte Carlo approximation
- To verify that our approximation makes sense, we first check whether we have an *unbiased* estimator:

$$\mathbb{E}\left[\langle f \rangle\right] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} f(z_i)\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[f(z_i)\right] = \mathbb{E}\left[f(z)\right]$$

- Furthermore, we would like that with an infinite amount of samples, our variance goes to zero:

$$\mathbb{V}\text{ar}\left[\langle f \rangle\right] = \frac{1}{N^2}\sum_{i=1}^{N}\mathbb{V}\text{ar}\left[f(z_i)\right] = \frac{1}{N}\mathbb{V}\text{ar}\left[f\right]$$

Hence, with $N \to \infty$, we linearly reduce the variance compared to a single sample.

- As a last part, we will look into how we can sample from any *known* distribution, given a uniform sampler

Discrete random variables In case of a discrete random variable $z \in \{1, 2, ..., K\}$, and given the distribution $p(z)$, we first have to calculate the cumulative density function: $p(z \leq \zeta)$. Then, given $u_i \sim U(0, 1)$, the sample k of $p(z)$ is where $p(z \leq k-1) \leq u_i < p(z \leq k)$

Continuous random variables For continuous variables, we need to calculate the CDF by an integral:

$$F(\zeta) = \int_{-\infty}^{\zeta} p(z)dz = p(z \leq \zeta)$$

, and take its inverse F^{-1}. Then, given $u_i \sim U(0, 1)$, our sample is $z_i = F^{-1}(u_i)$ which is a change of variables.

4.2 Rejection sampling

- Assume we have a probability density $p(z)$ from which we want to sample. Choose another distribution $q(z)$, called *proposal* distribution, from which we can sample. Further constraints are that the unnormalized distributions $\tilde{p}(z) \propto p(z), \tilde{q}(z) \propto q(z)$ fulfill:

$$\int \tilde{q}(z)dz < \infty, \tilde{p}(z) \leq \tilde{q}(z) \forall z$$

- Now, we can generate samples from $p(z)$ by a simple algorithm:

Pseudocode for rejection sampling

for $n = 1, ..., N$ **do**
  while *No sample for* z_n *accepted* **do**
    Sample $\hat{z}$ from $q(z)$;
    Sample $u \sim U(0, 1)$;
    if $u < \frac{\tilde{p}(\hat{z})}{\tilde{q}(\hat{z})}$ **then**
      Accept sample $z_n = \hat{z}$;
    else
      Reject sample $\hat{z}$ and re-sample;
    end
  end
end
Return samples $\{z_n\}_{n=1}^{N}$;

- The principle of rejection sampling is visualized in Figure 9

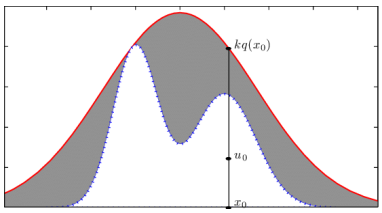


Figure 9: Visualizing rejection sampling. In the figure, $\tilde{q}(z)$ is denoted as $kq(z)$, and x_0 is the initial sample from q.

- To show that we actually generate samples from $p(z)$, we can write down the probability for a value z_i to be picked. First, the chance of z_i being generated in first place is $q(z_i)$. Next, the chance of z_i being accepted, is $\frac{\tilde{p}(z_i)}{\tilde{q}(z_i)}$. Together, we get the probability of z_i:

$$\hat{p}(z_i) = q(z_i)\frac{\tilde{p}(z_i)}{\tilde{q}(z_i)} \propto p(z_i)$$

  Hence, we actually generate samples from z_i although we initially sample from $q(z)$

- One requirement of rejection sampling to work well is that the area between $\tilde{p}(z)$ and $\tilde{q}(z)$ is small. The efficience of this sampler can be measured by the acceptance rate, which is $\mathbb{E}_{z_i \sim q}\left[\frac{\tilde{p}(z_i)}{\tilde{q}(z_i)}\right]$. If this value is low, it means that a lot of samples are rejected, hence the sampling process takes longer. This gets especially critical in higher dimensions as we need to make sure that *for all* z_i, $\tilde{q}(z_i)$ is greater than $\tilde{p}(z_i)$. Finding a simple distribution in high dimensions that fulfills this requirement is often not trivial

4.3 Importance sampling

- Another approach for estimating an expectation is not generating actual samples from p, but simply weight samples by their *importance*
- Again, we need two distributions: p over which we want to determine the expectation, and q that we actually sample from. Note that for this algorithm to work, none of these distributions need to be normalized. The only thing that is required is that we can sample from the normalized density of q.
- The pseudo-code for the importance sampling is fairly simple and straight forward:

Pseudocode for importance sampling

Sample $\{z_n\}_{n=1}^N$ from $q(z)$;
Calculate the weights $w_n = \frac{p(z_n)}{q(z_n)}$;
Determine expectation by $\mathbb{E}_p[f] \approx \frac{\sum_n w_n f(z_n)}{\sum_n w_n}$;

- The intuition of importance sampling can be shown when plugging in the two distributions:

$$\mathbb{E}_p[f] = \int p(z) f(z) dz = \frac{\int q(z) \frac{p(z)}{q(z)} f(z) dz}{\underbrace{\int q(z) \frac{p(z)}{q(z)} dz}_{=1}} = \frac{\mathbb{E}_q\left[w_z \cdot f(z)\right]}{\mathbb{E}_q\left[w_z\right]} \approx \mathbb{E}_q\left[\frac{\sum_i w_i f(z_i)}{\sum_i w_i}\right]$$

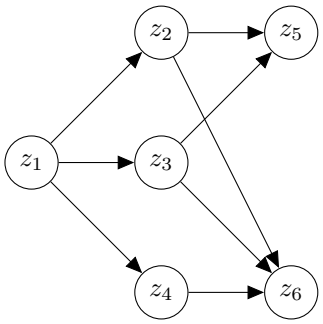
- Although importance sampling can use all samples, it has two major drawbacks:
  - The estimate of $\mathbb{E}[f]$ is <u>not</u> unbiased. Imagine we would sample only a single time ($N = 1$ in pseudo code). As the sum drops out, we end up with $f(z_i)$ where z_i is sampled from q, and not p! This bias decreases with the number of samples, but should always be kept in mind as an accurate estimate might require more samples, especially when it occurs with the second drawback.
  - The importance weighting estimate has a high variance if p differs strongly from q, especially when $p(z_i) \gg q(z_i)$ as the weight is very high for this data point. Imagine q is a Gaussian with a large variance, while p is a peaked Gaussian around 0. For most of the samples, $p(z) \ll q(z)$, and hence their weight is low. But for a small set of points, namely those close to 0, $p(z)$ is much greater than $q(z)$ resulting in a high weight. If we now sample e.g. 100 times, all points except those around 0 are neglected due to their low weight. And as those important points are rarely sampled from q, we need many samples to reduce the variance. This problem occurs even stronger in high-dimensional spaces.
- Furthermore, note that importance sampling can only be used for approximating an expectation, and not for generating independent samples from p

4.4 Ancestral Sampling

- Assume we have given a Bayesian network, and want to sample from the joint probability. We can write the joint probability as:

$$p(\boldsymbol{z}) = \prod_{i=1}^{d} p(z_i | z_{\text{pa}(i)})$$

  where we use a topological ordering $z_1, ..., z_d$ with $z_j < z_i$ if $j \in \text{pa}(i)$
- Now we can simply sample from the joint distribution by sampling from each of the conditionals, in the topological ordering. For example, assume we have the following Bayesian network:



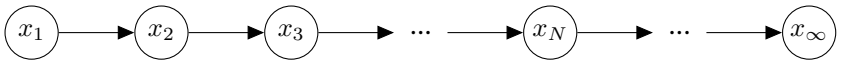
Then we can sample as follows:

$$\begin{aligned}
\tilde{z}_1 &\sim p(z_1) \\
\tilde{z}_2 &\sim p(z_2|z_1 = \tilde{z}_1) \\
\tilde{z}_3 &\sim p(z_3|z_1 = \tilde{z}_1) \\
\tilde{z}_4 &\sim p(z_4|z_1 = \tilde{z}_1) \\
\tilde{z}_5 &\sim p(z_5|z_2 = \tilde{z}_2, z_3 = \tilde{z}_3) \\
\tilde{z}_6 &\sim p(z_6|z_2 = \tilde{z}_2, z_3 = \tilde{z}_3, z_4 = \tilde{z}_4)
\end{aligned}$$

- Note that for sampling from each of the individual distributions, we can use the other techniques like rejection or importance sampling.

4.5 Markov-Chain Monte Carlo

- Given a target distribution $p(x)$ that we want to sample from, we setup a Markov chain such that $p(x_n) \to p(x)$ as $N \to \infty$, i.e. such that $p(x)$ is its equilibrium distribution



- Note that as MCMC runs ancestral sampling on a chain, two consecutive samples are no longer independent. Still, we can assume that the dependency between two fairly distant samples is neglectable
- We start sampling x_1 from an initial distribution $q(x_1)$ which can be chosen arbitrarily (but the closer q to p, the faster the convergence and hence, faster sampling). For the next samples, we use a transition kernel T (which we have to specify/find by ourselves) to get $x_2 \sim T(x_2|x_1)$
- The transition kernel is usually independent of time (most efficient), but can be extended to multiple steps (e.g. x_t is sampled by T_1, x_{t+1} from T_2, and x_{t+2} again from T_1)
- The marginal distribution can be determined by:

$$q(x_n) = \int q(x_{n-1})T(x_n|x_{n-1})dx_{n-1}$$

  and the joint probability for all $x_1, ..., x_N$ is:

$$q(x_1, ..., x_N) = q(x_1)\prod_{i=2}^{N} T(x_i|x_{i-1})$$

- The equilibrium is reached when $x_N \sim p(x)$ (i.e. $p(x)$ is an invariant of the chain):

$$p(x_{N+1}) = \int T(x_{N+1}|x_N)p(x_N)dx_N$$

  In this case, $p(x)$ is called to be an invariant of the chain. Note that a chain can have multiple invariants, as if T is the identity transformation, any distribution is an invariant of that chain
- A sufficient but not necessary condition of an invariant $p(x)$ is that it satisfies the property of detailed balance:

$$p(x_t)T(x_{t+1}|x_t) = p(x_{t+1})T(x_t|x_{t+1})$$

  This property can be interpreted as the Markov chain being reversible. Although detailed balance is not required, it is mostly easier to fulfill when designing a kernel
- Another property we are looking for is ergodicity. A sufficient condition for ergodicity is that any state x_N has a positive probability, for any N.
- If we have an ergodic Markov chain and $p^\star(x)$ being an invariant, then $p^\star(x)$ is a unique equilibrium, where for any start distribution $q(x_0)$, the distribution $q(x_N)$ with $N \to \infty$ converges to the required distribution $p^\star(x)$.

- The sketch of the sampling process is:

Sample initial state x_0 from $q(x_0)$;
for $t = 0, ..., N$ **do**
  Sample $x_{t+1} \sim T(x_{t+1}|x_t)$;
Output x_N as sample of $p^\star(x)$;

N has to be large enough in this case, and is often referred to as *burn-in* time.

In addition, note that the required N can be reduced by reducing the distance between q and p. Thus, for generating multiple samples, we can take the first sample x_N, and continue the chain for additional M steps, where usually $M \ll N$ (but M must be certain size to guarantee independence of samples, depends on T). Then, x_{N+M} is a new sample from p.

- The problem of MCMC is how we can find the transition kernel T for a distribution p. Once we have found two kernels, we can combine those to new kernels:

$$T_3 = \alpha T_1 + (1-\alpha)T_2, \quad (\alpha \in [0,1])$$
$$T_3 = T_2 \circ T_1 \quad \text{(composition, first apply } T_1\text{, then } T_2\text{)}$$

- One way to overcome this problem is the Metropolis-Hastings algorithm, which uses a form of rejection sampling to allow any transition kernel T

4.5.1 Metropolis-Hastings algorithm

- Choose a proposal transition kernel $Q(x_{t+1}|x_t)$ (e.g. a random walk). Then we can sample from $p^\star(x)$ as follows:

Pseudocode for Metropolis-Hastings algorithm

Sample initial state x_0 from $q(x_0)$;
for $t = 0, ..., N$ **do**
  Sample $\tilde{x}_{t+1} \sim Q(\tilde{x}_{t+1}|x_t)$;
  Compute acceptance probability $\alpha(\tilde{x}_{t+1}|x_t) = \min\left(1, \frac{p^\star(\tilde{x}_{t+1})Q(x_t|\tilde{x}_{t+1})}{p^\star(x_t)Q(\tilde{x}_{t+1}|x_t)}\right)$;
  Sample $u_t \sim U(0,1)$;
  if $u_t \leq \alpha(\tilde{x}_{t+1}|x_t)$ **then**
    accept sample $x_{t+1} = \tilde{x}_{t+1}$;
  else
    reject sample and stay at current state: $x_{t+1} = x_t$;
  end
end
Output x_N as sample of $p^\star(x)$;

We can view the combination of Q and acceptance probability α as our transition kernel $T = \alpha \circ Q$.

- Given this simple algorithm, we can design the transition kernel Q with minimal knowledge of $p^\star$. For example, a kernel which mostly works well is to combine larger and smaller random walk steps. This can be very helpful in high-dimensional space as it can explore in different scales, and hence, in contrast to rejection and importance sampling, still works well in high dimensions

- Nonetheless, keep in mind that the performance now depends on the transition kernel Q. If it is designed poorly, many samples are rejected and we again end up with an inefficient sampling process. For example, if we have a highly multi-modal distribution, we need to have large enough steps to be able to jump between modes. But as said before, this is much less knowledge we need of $p^\star$ compared to the other discussed sampling algorithms

- We can sketch the proof here for the detailed balance. If a new sample x_{t+1} is accepted, it has the

probability:

$$\begin{aligned} p^\star(x_t)T(x_{t+1}|x_t) &= p^\star(x_t)Q(x_{t+1}|x_t)\min\left(1, \frac{p^\star(x_{t+1})Q(x_t|x_{t+1})}{p^\star(x_t)Q(x_{t+1}|x_t)}\right) \\ &= \min\left(p^\star(x_t)Q(x_{t+1}|x_t), p^\star(x_{t+1})Q(x_t|x_{t+1})\right) \\ &= p^\star(x_{t+1})Q(x_t|x_{t+1})\min\left(1, \frac{p^\star(x_t)Q(x_{t+1}|x_t)}{p^\star(x_{t+1})Q(x_t|x_{t+1})}\right) \\ &= p^\star(x_{t+1})T(x_t|x_{t+1}) \end{aligned}$$

As we can interchange x_t and x_{t+1}, $p^\star$ is an invariant of the Markov chain.

In case $\tilde{x}_{t+1}$ was rejected, we stay at x_t which satisfies the detailed balance anyways.

4.5.2 Gibbs sampling

- Gibbs sampling is a special case of the Metropolis-Hastings algorithm, where the acceptance probability is always 1. The idea is that we cannot easily sample from a big joint distribution, but sampling a single variable given all others is feasible/much simpler.
- Hence, we sample a D-dimensional vector $(x_1, x_2, ..., x_D)$ by sampling from the conditional distributions of a single variable x_i where we keep all other variables fixed. In pseudo-code, we can define Gibbs sampling as:

Pseudocode for Gibbs sampling

Choose an initial state $\{x_i : i = 1, ..., M\}$;
for $t = 0, ..., N$ **do**
  Sample $x_1^{(t+1)} \sim p(x_1|x_2^{(t)}, x_3^{(t)}, ..., x_M^{(t)})$;
  Sample $x_2^{(t+1)} \sim p(x_2|x_1^{(t+1)}, x_3^{(t)}, ..., x_M^{(t)})$;
  ...
  Sample $x_M^{(t+1)} \sim p(x_M|x_2^{(t+1)}, x_3^{(t+1)}, ..., x_{M-1}^{(t+1)})$;
end
Output $\left\{x_1^{(N)}, ..., x_M^{(N)}\right\}$ as sample of $p(x_1, ..., x_M)$;

- Clearly, $p(\boldsymbol{x})$ is an invariant of the Markov chain because at each step, we sample from the correct conditional $p(x_i, \boldsymbol{x}_{\setminus i})$. But note that detailed balance can only be guaranteed if the order of x_i's is randomized every iteration.
- For ergodicity, we just need to make sure that the conditionals are not zero for any point. Otherwise, we have to prove ergodicity explicitly.
- The acceptance probability of a sample according to the Metropolis-Hastings algorithm is in case of Gibbs sampling:

$$\alpha(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)}) = \frac{p^\star(\boldsymbol{x}^{(t+1)})Q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t+1)})}{p^\star(\boldsymbol{x}^{(t)})Q(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)})} = \frac{p^\star(x_i^{(t+1)}|\boldsymbol{x}_{\setminus i}^{(t+1)})p^\star(\boldsymbol{x}_{\setminus i}^{(t+1)}) \cdot p^\star(x_i^{(t)}|\boldsymbol{x}_{\setminus i}^{(t+1)})}{p^\star(x_i^{(t)}|\boldsymbol{x}_{\setminus i}^{(t)})p^\star(\boldsymbol{x}_{\setminus i}^{(t)}) \cdot p^\star(x_i^{(t+1)}|\boldsymbol{x}_{\setminus i}^{(t)})} = 1$$

where $\boldsymbol{x}_{\setminus i}^{(t)} = \boldsymbol{x}_{\setminus i}^{(t+1)}$ as the other variables do not change.

5 Sequential Data

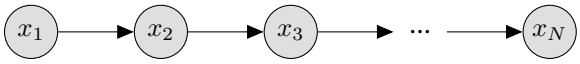
- Most models we discussed so far assumed that multiple data points $\boldsymbol{x}_n$ are independent of each other. However, in many use-cases, we have e.g. temporal data where consecutive data points dependent on each other
- The likelihood of such data can be written as:

$$p(x_1, ..., x_N) = \prod_{n=1}^{N} p(x_n | x_1, ..., x_{n-1})$$

- Here, we will focus on Markov models and its different variations

5.1 Markov models

- One of the simplest models for sequential data are Markov models, where we limit the conditionals to a fixed size. For example, a *first-order* Markov model has the likelihood $p(x_1) \prod_{n=2}^{N} p(x_n | x_{n-1})$:



  A second-order MM would add connections between x_1 and x_3, x_2 and x_4 etc.

- We call a Markov model homogeneous if all conditionals $p(x_n | x_{n-1})$ are the same, i.e. the transition probability is steady over time:

$$p(x_n = a | x_{n-1} = b) = p(x_{n+m} = a | x_{n+m-1} = b)$$

- Suppose each x_n has K possible states. Then the parameters for a homogeneous MM increases over the order by:

Order	Number of params	
	Prior	Conditionals
0	$K-1$	0
1	$K-1$	$K(K-1)$
2	K^2-1	$K^2(K-1)$
...	...	...
M	K^M-1	$K^M(K-1)$

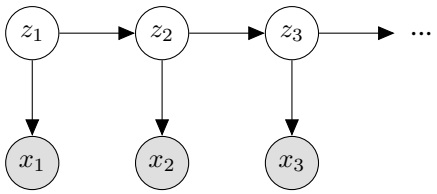
  For inhomogeneous MM, we would have $N-M$-times the conditional parameters where N is the length of the chain (needs fixed size if conditionals are not shared!).

  As the number of parameters increases exponentially with the order M, it is often impractical to use high-order MM.

- Our next discussions will focus on the first-order MM but can be applied to any order. This can be easily shown by reducing a M'th order MM to 1st order MM. Suppose we introduce new variables $y_n = (x_n, x_{n+1}, ..., xn + M - 1)$, then we can express our MM by:

$$p(y_n = (x_n, x_{n+1}, ..., x_{n+M-1}) | y_{n-1} = (\tilde{x}_{n-1}, \tilde{x}_n, ..., \tilde{x}_{n+M-2})) = \begin{cases} 0 & \text{if for any } i, x_i \neq \tilde{x}_i \\ p(x_{n+M-1} | x_n, ..., x_{n+M-2}) & \text{otherwise} \end{cases}$$

- Often, we want Markov models that are more expressive than a first-order, but at the same time, prevent having too many parameters. One way of enriching the class of MMs is by introducing latent variables as shown in this graphical model:



- The joint distribution for this model is given by:

$$p(x_1, ..., x_N, z_1, ..., z_N) = p(z_1) \cdot \left[\prod_{n=2}^{N} p(z_n|z_{n-1})\right] \cdot \left[\prod_{n=1}^{N} p(x_n|z_n)\right]$$

- We now look at two variants of this model:
  1. *Hidden Markov Models* assume that the latent space z is discrete
  2. *Linear Dynamical Systems* use a continuous latent space z such as linear Gaussian

5.2 Hidden Markov Models

- Commonly, when using discrete latent variables, we assume that we have a mixture model, and z_n as discrete multinomial variables specify the component from which x_n was generated
- For the homogeneous case, we get the joint distribution:

$$p(\boldsymbol{x}_1, ..., \boldsymbol{x}_N|\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}) = \sum_{\boldsymbol{z}_1} ... \sum_{\boldsymbol{z}_N} p(\boldsymbol{z}_1|\boldsymbol{\pi}) \left[\prod_{n=2}^{N} \underbrace{p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}, \boldsymbol{A})}_{\text{Transition probabilities}}\right] \cdot \left[\prod_{n=1}^{N} \underbrace{p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\phi})}_{\text{Emission probabilities}}\right]$$

  where $\boldsymbol{A}$ can be seen as a table with $A_{jk} \equiv p(z_{nk} = 1|z_{n-1,j} = 1)$ which gives the constraints $\sum_k A_{jk} = 1$, $0 \leq A_{jk} \leq 1$.

  The parameter $\boldsymbol{\pi}$ is again a prior over latent states for z_1, where $\sum_k \pi_k = 1$.

  The mapping between latent and observed variable is described as emission probabilities, which we can write as $p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\phi}) = \prod_{k=1}^{K} p(\boldsymbol{x}_n|\boldsymbol{\phi}_k)^{z_{nk}}$

- For optimizing the parameters, we again use the EM algorithm because otherwise we would need to calculate $p(\boldsymbol{X}|\boldsymbol{\theta}) = \sum_Z p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$. The sum has a complexity which increases exponentially with the number of hidden variables, and makes it inefficient for large N.

5.2.1 Maximum Likelihood for HMM

- Remember that our objective in the EM algorithm was:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{Z} \sim p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})} [\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})]$$

  which can be written in our case as:

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = {} & \mathbb{E}_{\boldsymbol{z}_1 \sim p(\boldsymbol{z}_1|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})} \left[\sum_{k=1}^{K} z_{1k} \ln \pi_k\right] + \\ & \sum_{n=2}^{N} \mathbb{E}_{(\boldsymbol{z}_n, \boldsymbol{z}_{n-1}) \sim p(\boldsymbol{z}_n, \boldsymbol{z}_{n-1}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})} \left[\sum_{j=1}^{K} \sum_{k=1}^{K} z_{nj} \cdot z_{n-1,k} \cdot \ln A_{jk}\right] + \\ & \sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}_n \sim p(\boldsymbol{z}_n|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})} \left[\sum_{k=1}^{K} z_{nk} \ln p(\boldsymbol{x}_n|\boldsymbol{\phi}_k)\right] \\ = {} & \sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \zeta(z_{n-1,j}, z_{nk}) \cdot \ln A_{jk} + \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \ln p(\boldsymbol{x}_n|\boldsymbol{\phi}_k) \end{aligned}$$

  where we use $\gamma(\boldsymbol{z}_n) = p(\boldsymbol{z}_n|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$ and $\zeta(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n) = p(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$.

- Now, let's take a closer look at the steps of the EM algorithm

  E-step We need to determine $p(z_1, ..., z_N|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$ which we split into $\gamma(\boldsymbol{z}_n)$ and $\zeta(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n)$. Hence, we need to calculate marginals, which can be done efficiently with the sum-product algorithm.

  First, we need to convert the Bayesian network into a factor graph. We do this by replacing the prior with $h(\boldsymbol{z}_1) = p(\boldsymbol{z}_1|\boldsymbol{\pi}^{\text{old}})p(\boldsymbol{x}_1|\boldsymbol{z}_1, \boldsymbol{\phi}^{\text{old}})$, and the transitions by $f_n(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n) = p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}, \boldsymbol{A}^{\text{old}})p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}^{\text{old}})$. The corresponding factor graph looks like in Figure 10.

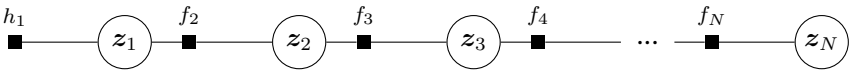


Figure 10: Drawing of the factor graph.

Using the factor graph, we want to determine the normalized beliefs $p(\boldsymbol{z}_n|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$. The sum product updates are:

$$\begin{aligned}
\mu_{z_{n-1}\to f_n}(\boldsymbol{z}_{n-1}) &= \mu_{f_{n-1}\to z_{n-1}}(\boldsymbol{z}_{n-1}) \\
\alpha_n(\boldsymbol{z}_n) = \mu_{f_n\to z_n}(\boldsymbol{z}_n) &= \sum_{\boldsymbol{z}_{n-1}} f_n(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n)\alpha_{n-1}(\boldsymbol{z}_{n-1}) \\
\mu_{z_n\to f_n}(\boldsymbol{z}_n) &= \mu_{f_{n+1}\to z_n}(\boldsymbol{z}_n) \\
\beta_n(\boldsymbol{z}_n) &= \sum_{z_{n+1}} f_{n+1}(\boldsymbol{z}_n, \boldsymbol{z}_{n+1})\beta_{n+1}(\boldsymbol{z}_{n+1})
\end{aligned}$$

Our beliefs can be calculated by:

$$\begin{aligned}
\text{Variable belief} \quad p(\boldsymbol{z}_n, \boldsymbol{X}|\boldsymbol{\theta}^{\text{old}}) &= \alpha_n(\boldsymbol{z}_n)\beta_n(\boldsymbol{z}_n) \\
\text{Factor belief} \quad p(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n, \boldsymbol{X}|\boldsymbol{\theta}^{\text{old}}) &= \mu_{f_{n-1}\to z_{n-1}}(z_{n-1})\mu_{f_{n+1}\to z_n}(\boldsymbol{z}_n)f_n(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n) \\
\text{Normalization constant} \quad p(\boldsymbol{X}|\boldsymbol{\theta}^{\text{old}}) &= \sum_{\boldsymbol{z}_n} \alpha_n(\boldsymbol{z}_n)\beta_n(\boldsymbol{z}_n)
\end{aligned}$$

Finally, we can calculate our sufficient statistics:

$$\begin{aligned}
\gamma(\boldsymbol{z}_n) &= \frac{p(\boldsymbol{z}_n, \boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})}{p(\boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})} = \frac{\alpha_n(\boldsymbol{z}_n)\beta_n(\boldsymbol{z}_n)}{p(\boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})} \\
\zeta(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n) &= \frac{p(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n, \boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})}{p(\boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})} = \frac{\alpha_{n-1}(\boldsymbol{z}_{n-1})\beta_n(\boldsymbol{z}_n)p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}, \boldsymbol{A}^{\text{old}})p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\phi}^{\text{old}})}{p(\boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})}
\end{aligned}$$

M-step In the maximization step, we optimize $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ regarding the parameters $\boldsymbol{\pi}$, $\boldsymbol{A}$ and $\boldsymbol{\phi}$. Note that we need to add Lagrangian for the constraints on $\boldsymbol{A}$ and $\boldsymbol{\pi}$:

$$\tilde{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) + \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right) + \sum_{j=1}^{K}\lambda_j\left(\sum_{k=1}^{K}A_{jk} - 1\right)$$

Performing the maximization, we get:

$$\pi_k^{\text{new}} = \frac{\gamma(z_{1k})}{\sum_{j=1}^{K}\gamma(z_{1j})}, \qquad A_{jk} = \frac{\sum_{n=2}^{N}\zeta(z_{n-1,j}, z_{nk})}{\sum_{l=1}^{K}\sum_{n=2}^{N}\zeta(z_{n-1,j}, z_{nl})}$$

Solving the same for the parameter $\boldsymbol{\phi}$ depends on the form of emission probability that was chosen. For example, if we have a Gaussian density $p(\boldsymbol{x}|\boldsymbol{\phi}_k)$, the optimized parameters are:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N}\gamma(z_{nk})\boldsymbol{x}_n}{\sum_{n=1}^{N}\gamma(z_{nk})}, \qquad \boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^{N}\gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N}\gamma(z_{nk})}$$

Similarly, if we would have discrete observations and model it with a multinomial distribution, i.e. $p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\mu}) = \prod_{i=1}^{D}\prod_{k=1}^{K}\mu_{ik}^{x_i z_k}$, we would get as solution:

$$\mu_{ik} = \frac{\sum_{n=1}^{N}\gamma(z_{nk})x_{ni}}{\sum_{n=1}^{N}\gamma(z_{nk})}$$

- To conclude, the EM algorithm for Hidden Markov Models can be summarized as follows:

EM for HMM

1. Choose initial values $\boldsymbol{\theta}^{\text{old}}$ with $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi})$
2. Iterate until $\Delta\boldsymbol{\theta}^{(t)} < \epsilon$
   (a) **E-step**: Calculate $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ by:
      i. Run forward α recursion to calculate $\alpha(z_1), ..., \alpha(z_N)$
      ii. Run backward β recursion to calculate $\beta(z_N), ..., \beta(z_1)$
      iii. Calculate sufficient statistics $\gamma(z_n)$, $\zeta(z_{n-1}, z_n)$ for $n = 1, .., N$, and normalization constant $p(\boldsymbol{X}|\boldsymbol{\theta}^{\text{old}})$
   (b) **M-step**: Calculate $\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \tilde{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$
      – $\pi_k^{\text{new}} = \gamma(z_{1k}) / \sum_{k=1}^{K} \gamma(z_{1j})$
      – $A_{jk} = \sum_{n=2}^{N} \zeta(z_{n-1,j}, z_{nk}) / \sum_{l=1}^{K} \sum_{n=2}^{N} \zeta(z_{n-1,j}, z_{nl})$
      – $\boldsymbol{\phi}^{\text{new}}$ depending on choice of emission probability

5.2.2 Viterbi Algorithm (Max-sum for HMMs)

- Sometimes we might be interested in the latent variables $\boldsymbol{Z}$ for a given fixed model as they can be interpreted, and thus we want to determine the most likely values
- For this, we can use an adaptation of the max-sum algorithm where we use dynamic programming for the backward path
- We use the same factor graph as in Figure 10. For simplification, we denote the message from f_n (or h_1 for $n = 1$) to $\boldsymbol{z}_n$ as $\omega(\boldsymbol{z}_n)$.
- The whole algorithm can be then summarized as:

Pseudocode of Viterbi algorithm

// Forward pass
Set initial message $\omega(\boldsymbol{z}_1) = \ln p(\boldsymbol{z}_1) + \ln p(\boldsymbol{x}_1|\boldsymbol{z}_1)$;
for $n = 1, ..., N-1$ **do**
  $\omega(\boldsymbol{z}_{n+1}) = \ln p(\boldsymbol{x}_{n+1}|\boldsymbol{z}_{n+1}) + \max_{\boldsymbol{z}_n} (\ln p(\boldsymbol{z}_{n+1}|\boldsymbol{z}_n) + \omega(\boldsymbol{z}_n))$;
  $\psi_n(\boldsymbol{z}_{n+1}) = \arg\max_{\boldsymbol{z}_n} (\ln p(\boldsymbol{z}_{n+1}|\boldsymbol{z}_n) + \omega(\boldsymbol{z}_n))$
end
// Backward pass
$\boldsymbol{z}_N^{\max} = \arg\max_{\boldsymbol{z}_N} \omega(\boldsymbol{z}_N)$;
for $n = N-1, ..., 1$ **do**
  $\boldsymbol{z}_n^{\max} = \psi_n(\boldsymbol{z}_{n+1}^{\max})$;
end
Return $\boldsymbol{z}_1^{\max}, \boldsymbol{z}_2^{\max}, ..., \boldsymbol{z}_N^{\max}$;

- Coming back to the discussion about the max-sum algorithm on trees, we have said that for multiple optima, we need to use the Viterbi algorithm. Assume that for any n, we have two solutions for $\boldsymbol{z}_n^{\max} = \psi_n(\boldsymbol{z}_{n+1}^{\max})$ (i.e. we have two values for z_n that maximize our objective). If we are just interested in one, arbitrary solution, we get pick any of them and continue. Otherwise, we select one of the optimums, continue the Viterbi algorithm, and afterwards go back to this step, and select the next optimum. In the end, we can combine all solutions to return a full set of all optima.

  We prevent the problem of returning undesirable combinations of independent optima by conditioning them on each other (i.e. $\boldsymbol{z}_n^{\max}$ depends on $\boldsymbol{z}_{n+1}^{\max}$).

5.3 Linear Dynamical Systems

- As mentioned before, Linear Dynamical Systems use a continuous latent space $\boldsymbol{z}_n \in \mathbb{R}^{d_z}$ instead of a discrete as in HMM. This also influences the models we take because, as we will see later, specific distribution properties help to simplify the calculations

- One popular model is a *Linear-Gaussian*: all conditional distributions in the Bayesian network are Gaussians with means that depend linearly on its parents. This leads to the probabilities:

$$\begin{aligned}\text{Transition probability } \; p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}) &= \mathcal{N}(\boldsymbol{z}_n|\boldsymbol{A}\boldsymbol{z}_{n-1}, \boldsymbol{\Gamma}) \\ \text{Emission probability } \; p(\boldsymbol{x}_n|\boldsymbol{z}_n) &= \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{C}\boldsymbol{z}_n, \boldsymbol{\Sigma}) \\ \text{Initial state } \; p(\boldsymbol{z}_1) &= \mathcal{N}(\boldsymbol{z}_1|\boldsymbol{\mu}_0, \boldsymbol{V}_0)\end{aligned}$$

  with parameters $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\Gamma}, \boldsymbol{C}, \boldsymbol{\Sigma}, \boldsymbol{\mu}_0, \boldsymbol{V}_0)$ where $\boldsymbol{\Sigma}$ is the observation noise, and $\boldsymbol{\Gamma}$ the transition uncertainty.

- We first consider inference in LDS which represents the E-step, and then complete the learning process with the M-step in the second subsection

5.3.1 Inference in Linear Dynamical Systems

- To find the marginal distributions for the latent variables, we use again message passing. The forward equations are denoted with the normalized marginal distributions $\widehat{\alpha}(\boldsymbol{z}_n)$:

$$p(\boldsymbol{z}_n|\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = \widehat{\alpha}(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n|\boldsymbol{\mu}_n, \boldsymbol{V}_n)$$

- Similarly to the HMM, our forward propagation takes the form:

$$c_n\widehat{\alpha}(\boldsymbol{z}_n) = p(\boldsymbol{x}_n|\boldsymbol{z}_n)\int \widehat{\alpha}(\boldsymbol{z}_{n-1})p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1})d\boldsymbol{z}_{n-1}$$

  where we now have an integral instead of the sum, and c_n is a constant making sure of the right scale as $\widehat{\alpha}(\boldsymbol{z}_n)$ is normalized.

- Plugging in the Gaussian distributions, we get:

$$\begin{aligned}c_n\mathcal{N}(\boldsymbol{z}_n|\boldsymbol{\mu}_n, \boldsymbol{V}_n) &= \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{C}\boldsymbol{z}_n, \boldsymbol{\Sigma})\int \mathcal{N}(\boldsymbol{z}_{n-1}|\boldsymbol{\mu}_{n-1}, \boldsymbol{V}_{n-1})\mathcal{N}(\boldsymbol{z}_n|\boldsymbol{A}\boldsymbol{z}_{n-1}, \boldsymbol{\Sigma})d\boldsymbol{z}_{n-1} \\ &= \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{C}\boldsymbol{z}_n, \boldsymbol{\Sigma})\mathcal{N}(\boldsymbol{z}_n|\boldsymbol{A}\boldsymbol{\mu}_{n-1}, \underbrace{\boldsymbol{\Gamma} + \boldsymbol{A}\boldsymbol{V}_{n-1}\boldsymbol{A}^T}_{\boldsymbol{P}_{n-1}})\end{aligned}$$

  Here we see the first point where the chosen distributions can make a difference. The integral can be calculated due to the Gaussian, and we can get $\boldsymbol{\mu}_n$ and $\boldsymbol{V}_n$ by applying more mathematical tricks with Gaussians and matrices (which we will not detail here, but can be found in the lecture notes). The end result is:

$$\boldsymbol{\mu}_N = \boldsymbol{A}\boldsymbol{\mu}_{n-1} + \boldsymbol{K}_n(\boldsymbol{x}_n - \boldsymbol{C}\boldsymbol{A}\boldsymbol{\mu}_{n-1}), \quad \boldsymbol{V}_n = (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{C})\boldsymbol{P}_{n-1}$$

  The important thing here is that without observation, $\boldsymbol{\mu}_n$ would be simply $\boldsymbol{\mu}_{n-1}$ moved/shifted by A, but the second term corrects it for the observation.

- For the EM algorithm, in the backward pass, we get our sufficient statistics:

$$\gamma(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n|\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{V}}_n), \quad \zeta(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n) = \mathcal{N}\left(\begin{bmatrix}\hat{\boldsymbol{\mu}}_{n-1} \\ \hat{\boldsymbol{\mu}}_n\end{bmatrix}, \begin{bmatrix}\hat{\boldsymbol{V}}_{n-1} & \boldsymbol{J}_{n-1}\hat{\boldsymbol{V}}_n \\ \hat{\boldsymbol{V}}_n\boldsymbol{J}_{n-1}^T & \hat{\boldsymbol{V}}_n\end{bmatrix}\right)$$

5.3.2 Learning in LDS using EM

- Above we have seen the results of the E-step in Linear Dynamical Systems. Now we take a closer look at the M-step, where we want to optimize for the parameters $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\Gamma}, \boldsymbol{C}, \boldsymbol{\Sigma}, \boldsymbol{\mu}_0, \boldsymbol{V}_0)$

- The complete data likelihood which we want to optimize in expectation to the posterior, is:

$$\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) = \ln p(\boldsymbol{z}_1|\boldsymbol{\mu}_0, \boldsymbol{V}_0) + \sum_{n=2}^{N}\ln p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}, \boldsymbol{A}, \boldsymbol{\Gamma}) + \sum_{n=1}^{N}\ln p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{C}, \boldsymbol{\Sigma})$$

- If we now e.g. want to optimize for $\boldsymbol{\mu}_0$ and $\boldsymbol{V}_0$, we need to derive $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ with respect to these variables. The solution for those is:

$$\begin{aligned}\boldsymbol{\mu}_0, \boldsymbol{V}_0 : Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) &= -\frac{1}{2}\ln|\boldsymbol{V}_0| - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}_1|\boldsymbol{\theta}^{\text{old}}}\left[(\boldsymbol{z}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{V}_0^{-1}(\boldsymbol{z}_1 - \boldsymbol{\mu}_0)\right] + \text{const} \\ \boldsymbol{\mu}_0^{\text{new}} &= \mathbb{E}[\boldsymbol{z}_1|\boldsymbol{\theta}^{\text{old}}] \\ \boldsymbol{V}_0^{\text{new}} &= \mathbb{E}[\boldsymbol{z}_1\boldsymbol{z}_1^T|\boldsymbol{\theta}^{\text{old}}] - \mathbb{E}[\boldsymbol{z}_1|\boldsymbol{\theta}^{\text{old}}]\mathbb{E}[\boldsymbol{z}_1|\boldsymbol{\theta}^{\text{old}}]^T\end{aligned}$$

6 Causality

- Causality is about testing whether one event (*effect*) is the result of the occurrence of another event (*cause*), i.e. a change in the cause will lead to a change in the effect. It differs from correlation by *explaining/finding* the relationship behind variables
- While we look at the data distribution for correlation, we are focusing on the generation mechanism of the data in causality. While in statistics we then like to predict the next observation (or its likelihood), causality is interested in what happens if we perform interventions (setting a variable to a certain value)
- The most important operator in causality is the **do-operator**: $p(A = a|\text{do}(B = b))$. It differs from the standard conditional probability by not assuming that we have observed $B = b$, but that we externally set the value of B. This means that we cannot infer anything from its parents as in standard conditionals (if we observe $B = b$, then this usually gives us information about its parents).

  Note that there are cases where $p(A = a|\text{do}(B = b)) = p(A = a|B = b)$. One obvious example for this is when B has no parents in its corresponding graphical model.
- We start with a discussion about the terminology in causality, and then take a closer look at Causal Bayesian networks and causal reasoning

6.1 Causality terminology

- A is said to cause B if changing A leads to a change in B
- Similar to graphical models, we can define Causal graphs that represent causal relationships. An edge from A to B in the graph means that A *causes* B even if all other variables are kept fixed. Figure 11 gives an overview.

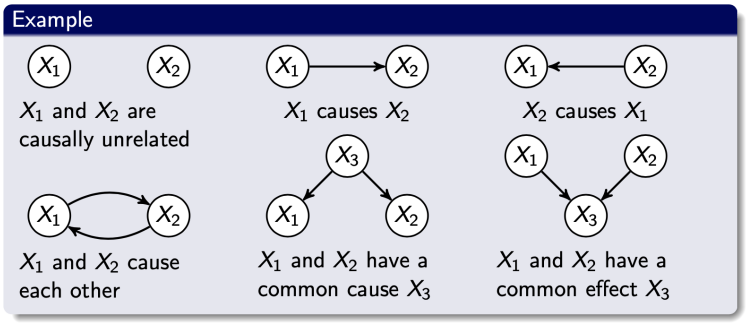


Figure 11: Examples of causal graphs (retrieved from lecture notes).

Note that these graphs can contain loops, which represents a feedback loop (a change in A leads to a change in B, and a change in B leads to a change in A). However, we will not take a closer look at those.

- We can interpreted node relationships in the graph as causal relations:
  - A is a *parent* of $B \implies A$ is a direct cause of B
  - A is a *child* of $B \implies A$ is a direct effect of B
  - A is a *ancestor* of B (e.g. $A \to C \to B$) $\implies A$ is a cause of B. Note that if we fix C, there is no effect of A on B. Hence, there is no direct edge.
  - A is a *descendant* of B (e.g. $B \to C \to A$) $\implies A$ is an effect of B.
- We use the notation $\mathcal{G}_{\overline{X}}$ to denote a sub-graph of $\mathcal{G}$ in which the incomming edges of X are removed. This is useful for discussing when X is set externally (hence, no influence of parents of X in that case).

  Similarly, $\mathcal{G}_{\underline{X}}$ is $\mathcal{G}$ without the outgoing edges of X.
- A **perfect intervention** $\text{do}(X = \xi)$ means that we force X to be the value ξ. Thereby, the graph $\mathcal{G}$ changes to $\mathcal{G}_{\overline{X}}$
  - To perform intervention, we require *modularity*, meaning that we can manipulate X without influencing any other variables in the graph $\boldsymbol{V} \setminus X$
  - This can be a challenge in real systems, but in our theoretical models, we can assume that we are able to do so

- A variable H is a **confounder** of X and Y (i.e. H confounds X and Y) if there is a directed path from H to X which does not include Y, and same from H to Y. Note that it is still allowed to have other paths between X and Y. Examples of confounders are shown in Figure 12.

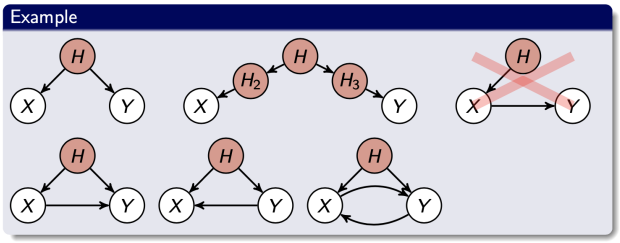


Figure 12: Examples of when a node H is a confounder of X and Y (retrieved from lecture notes).

- **Reichenbach's principle** sets correlation and causality into relation: if X and Y are correlated/depending on each other, then there is either a causal relation of the type $X \to Y$, $Y \to X$ or there exists a confounder H of X and Y.
  - Note that this principle can fail if we have a selection bias, meaning that the dataset of X and Y was obtained by only including samples that are conditional on some (possibly latent) event.

6.2 Causal Bayesian Networks

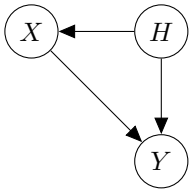
- An subspace of causal networks with many assumptions/limitations, but therefore easier to work with, are Causal Bayesian Networks. We make the following assumptions:
  - No confounding
  - A graph $\mathcal{G}$ does not contain any loops
  - We do not have any selection bias in the data, nor measurements error or time dependencies
- We call a Bayesian Network causal if:
  - Directed edges correspond with directed causal relations
  - After a perfect intervention $\text{do}(X_I = x_I)$, the probability density becomes:

$$p\left(\boldsymbol{X}_{\boldsymbol{V}\setminus I}|\text{do}(X_I = x_I)\right) = \prod_{i \in \boldsymbol{V}\setminus I} p\left(x_i|\boldsymbol{x}_{\text{pa}_i}\right)$$

6.3 Causal Reasoning

- The goal of causal reasoning is to estimate $p(y|\text{do}(X = x))$. If we can express it in terms of the observational distribution $p(x, y, ...)$ we say it is **identifiable** from the observational distribution.

  Note that it does not necessarily require all variables to be observable.
- Assume we have the following Bayesian Causal Network:



The standard conditional distribution is:

$$p(y|x) = \int p(y|h, x)p(h|x)dh$$

Now, assume we perform a perfect intervention on X, i.e. $\text{do}(X = x)$. What happens is that we neglect the effect of H on X, *but* we still need to consider the effect of H on Y. Hence, the conditional becomes:

$$p(y|\text{do}(X = x)) = \int p(h)p(y|h, x)dh$$

The important thing is that we have to prevent that changing X influences H by "back-reasoning" (i.e. H causes X, but observing X gives us information of H), which again influence Y. This is because we cannot change H by just forcing X to a value, as we *overwrite* the effect of H on X. Hence, we have to explicitly remove its dependency on X in the integral.

- We can derive a more general algorithm for deciding, for which variables we need to *adjust* our conditional probability for. This can be done in a very similar manner to d-separation, as we need to find all variables, that are implicitly changed by setting X to a certain value (i.e. variables that influence the decision of which value X can have), but then also influence Y. We do not want this influence because by forcing X to be a certain value, we cannot change variables that cause X. Hence, we are trying to find a set of variables S which break these kind of influences, and remove their dependency with X.
- In general, we can determine whether S is a sufficient set of variables we are adjusting by the following check:

Back-door criterion

A set of variables S satisfies the back-door criterion relative to a variable pair (X, Y), if:

1. $X, Y \notin S$
2. No node of S is a descendant (i.e. child of a child etc.) of X
3. S blocks all paths from Y to X where we have an incoming edge to X (other directions irrelevant for path itself). A path is blocked by S if:
   (a) It contains a collider $... \rightarrow u \leftarrow ...$ such that u is not an ancestor of a node in S
   (b) It contains a non-collider $... \rightarrow u$, $... \rightarrow u \rightarrow ...$, $... \leftarrow u \rightarrow ...$ such that u is in S

Then S is admissible for adjustment to find the causal effect of X on Y:

$$p(y|\text{do}(X = x)) = \int p(y|X = x, S = s)p(S = s)ds$$

If $S = \emptyset$: $p(y|\text{do}(X = x)) = p(y|X = x)$

To find the actual set S, we can simply perform the algorithm backwards. First, find all paths from Y to X with an incoming edge to X. Then, we start with $S = \emptyset$, and try to block all paths by adding variables to S. In case that all paths were blocked from the beginning on, or no paths exist, we can stop with $S = \emptyset$.

Note that simplest solution of S is the set of all the nodes with an edge to X. It might not be the smallest set, but should be always a valid one as we do not allow loops in BNs.

- Examples:
  - Consider the examples in Figure 13. Note that we usually try to find the smallest set of admissible variables as this simplifies the integral we have to take.

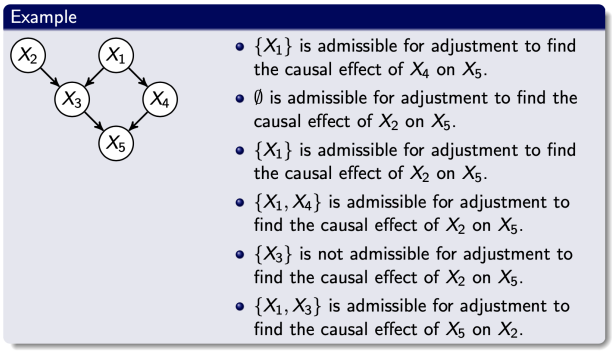
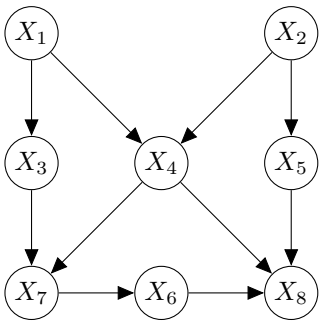


Figure 13: Example of admissible sets of variables for adjustment (retrieved from lecture notes).

– Consider the following, slightly more complicated Causal Bayesian Network:



We are trying to find the set S admissible for adjustment for the causal effect of X_7 on X_8 (the two nodes on the bottom, left and right). We have to consider all paths with incoming edges to X_7, so from X_3 and X_4. To block the path $X_8 \to X_4 \to X_7$, we add X_4 to S: $S = \{X_4\}$. However, by doing this, we unblocked another path: $X_8 \to X_5 \to X_2 \to X_4 \to X_1 \to X_3 \to X_7$. X_4 is not longer a collider anymore, as it is included in S. So, we can either add X_3, or X_5, or even X_1 or X_2 to block this path. For example, we can take $S = \{X_3, X_4\}$, which is then admissible for adjustment as all paths are blocked.

- Although we found a way to estimate what happens when we perform an intervention, the best way to find causal relations is to use randomized controlled trials. In a drug test, this would mean that we completely random assign a person to take the drug or not, ensuring that no underlying selection bias is in the process. By that, we should break all back-door paths (as there is nothing besides a coin flip that causes the event of ”taking the drug”)

A Appendix Math

Here we revisit some important mathematical tricks and equations to know.

A.1 Useful properties of a Gaussian

Given $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, suppose $\boldsymbol{x} = (\boldsymbol{x}_a, \boldsymbol{x}_b)$, define $\boldsymbol{\mu} = (\boldsymbol{\mu}_a, \boldsymbol{\mu}_b)$ and $\boldsymbol{\Sigma} = \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}$ ($\Sigma_{ab} = \Sigma_{ba}^T$, $\Sigma_{aa} = \Sigma_{aa}^T$)

Marginal distribution: $p(\boldsymbol{x}_a) = \mathcal{N}(\boldsymbol{x}_a \mid \boldsymbol{\mu}_a, \Sigma_{aa})$

Conditional distribution: $p(\boldsymbol{x}_a|\boldsymbol{x}_b) = \mathcal{N}(\boldsymbol{x}_a \mid \boldsymbol{\mu}_{a|b}, \Sigma_{a|b})$
where $\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}, \boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b)$

Multiplication of two Gaussians: $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{a}, \boldsymbol{A})\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{b}, \boldsymbol{B}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{c}, \boldsymbol{C})\ \overbrace{\mathcal{N}(\boldsymbol{a}|\boldsymbol{b}, \boldsymbol{A}+\boldsymbol{B})}^{\text{Normalization constant}}$
where $\boldsymbol{C} = \left(\boldsymbol{A}^{-1} + \boldsymbol{B}^{-1}\right)^{-1}$, $\boldsymbol{c} = \boldsymbol{C}\left(\boldsymbol{A}^{-1}\boldsymbol{a} + \boldsymbol{B}^{-1}\boldsymbol{b}\right)$

Conditional and marginals in graphical model

$$
\begin{aligned}
p(x) &= \mathcal{N}(x|\mu, \Lambda^{-1}) \\
p(y|x) &= \mathcal{N}(Ax + b, L^{-1}) \\
\\
\Rightarrow p(y) &= \mathcal{N}(y|A\mu + b, L^{-1} + A\Lambda^{-1}A^T) \\
\Rightarrow p(x|y) &= \mathcal{N}(x|\Sigma(A^T L(y-b) + \Lambda\mu), \Sigma), \quad \Sigma = (\Lambda + A^T LA)^{-1}
\end{aligned}
$$

A.2 Distributions from the exponential family

It is useful to know the exponential form of a few most popular distributions. Remember that in general, a distribution of the exponential family can be written in the form:

$$p(\boldsymbol{x}|\boldsymbol{\eta}) = h(\boldsymbol{x})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T \cdot \boldsymbol{u}(\boldsymbol{x})\right)$$

Some tricks to keep in mind:

- $a^b = \exp(b \cdot \log a)$ - helpful to find sufficient statistics and natural parameters
- If we have the constraint $\sum_k \pi_k = 1$, replace π_K with $\pi_K = 1 - \sum_{k \neq K} \pi_k \Rightarrow$ one less parameter

A.2.1 Gaussian

Univariate:

$$p(x|\mu, \sigma^2) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$\boldsymbol{\eta} = \begin{bmatrix} \frac{\mu}{\sigma^2} & -\frac{1}{2\sigma^2} \end{bmatrix}^T$
$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix} x & x^2 \end{bmatrix}^T$
$h(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}}$
$g(\boldsymbol{\eta}) = (-2\eta_2)^{-\frac{1}{2}}\exp\left(\frac{\eta_1^2}{4 \cdot \eta_2}\right)$

Multivariate:

$$p(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = (2\pi)^{-D/2}|\Sigma|^{-1} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

$$\boldsymbol{\eta} = \begin{bmatrix}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} & -\frac{1}{2}\boldsymbol{\Sigma}^{-1}\end{bmatrix}^T$$
$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix}\boldsymbol{x} & \boldsymbol{x}\boldsymbol{x}^T\end{bmatrix}^T$$
$$h(\boldsymbol{x}) = (2\pi)^{-D/2}$$
$$g(\boldsymbol{\eta}) = |-2\boldsymbol{\eta}_2|^{-\frac{1}{2}} \cdot \exp\left(\frac{1}{4}\boldsymbol{\eta}_1^T\boldsymbol{\eta}_2^{-1}\boldsymbol{\eta}_1\right)$$

A.2.2 Beta

$$p(x|\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \quad \text{where} \quad B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

$$\boldsymbol{\eta} = \begin{bmatrix}\alpha & \beta\end{bmatrix}^T$$
$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix}\log x & \frac{1}{x}\end{bmatrix}^T$$
$$h(\boldsymbol{x}) = 1$$
$$g(\boldsymbol{\eta}) = \frac{1}{B(\eta_1,\eta_2)}$$

A.2.3 Multinomial

$$p(\boldsymbol{x}|\boldsymbol{\pi}) = \frac{M!}{\prod_{i=1}^K x_i!}\prod_{i=1}^K \pi_i^{x_i}$$

$$\boldsymbol{\eta} = \begin{bmatrix}\ln\frac{\pi_1}{1-\sum_{i=1}^{K-1}\pi_i} & \ln\frac{\pi_2}{1-\sum_{i=1}^{K-1}\pi_i} & \dots & \ln\frac{\pi_{K-1}}{1-\sum_{i=1}^{K-1}\pi_i}\end{bmatrix}^T$$
$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix}x_1 & x_2 & \dots & x_{K-1}\end{bmatrix}^T$$
$$h(\boldsymbol{x}) = \frac{M!}{\prod_{i=1}^K x_i!}$$
$$g(\boldsymbol{\eta}) = \exp\left(-M\ln\left(1+\sum_{i=1}^{K-1}\exp(\eta_i)\right)\right)$$

A.2.4 Dirichlet

$$p(\boldsymbol{x}|\alpha_1,..,\alpha_K) = \frac{1}{B(\alpha_1,...,\alpha_K)}\prod_{i=1}^K x_i^{\alpha_i-1} \quad \text{where} \quad B(\alpha_1,...,\alpha_K) = \frac{\prod_{i=1}^K\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K\alpha_i)}$$

$$\boldsymbol{\eta} = \begin{bmatrix}\alpha_1 & \dots & \alpha_K\end{bmatrix}^T$$
$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix}\log x_1 & \dots & \log x_K\end{bmatrix}^T$$
$$h(\boldsymbol{x}) = \frac{1}{\prod_{i=1}^K x_i}$$
$$g(\boldsymbol{\eta}) = \frac{1}{B(\eta_1,...,\eta_K)}$$

A.2.5 Poisson

$$p(x|\lambda) = \frac{\lambda^x\exp(-x)}{x!}$$

$$\boldsymbol{\eta} = \begin{bmatrix}\ln\lambda\end{bmatrix}^T$$
$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix}x\end{bmatrix}^T$$
$$h(\boldsymbol{x}) = \frac{1}{x!}$$
$$g(\boldsymbol{\eta}) = \exp(-\exp(\eta))$$

A.2.6 Gamma

$$p(x|a,b) = \frac{1}{\Gamma(x)} b^a x^{a-1} \exp(-bx)$$

$$\boldsymbol{\eta} = \begin{bmatrix} (a-1) & -b \end{bmatrix}^T$$
$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix} \ln x & x \end{bmatrix}^T$$
$$h(\boldsymbol{x}) = 1$$
$$g(\boldsymbol{\eta}) = \frac{(-\eta_2)^{\eta_1+1}}{\Gamma(\eta_1+1)}$$

A.2.7 Conjugate priors

- Dirichlet $\rightarrow$ Multinomial
- Dirichlet $\rightarrow$ Categorical
- Beta $\rightarrow$ Bernoulli
- Gamma $\rightarrow$ Poisson
- Gaussian $\rightarrow$ Gaussian
- Gamma (precision) $\rightarrow$ Gaussian (known mean)