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

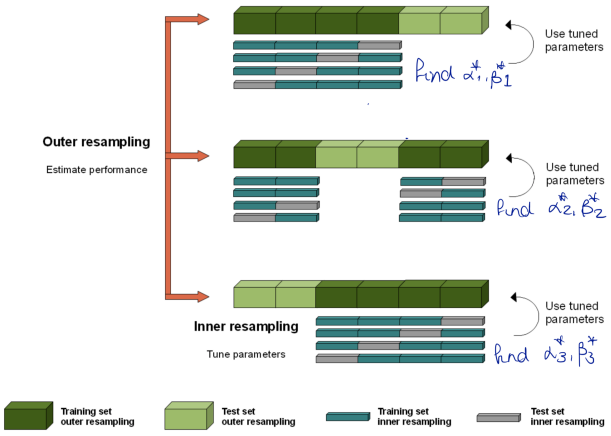


Figure 1: Illustration of nested cross validation. The outer loop splits dataset into test and trainval parts. Within the trainval parts, we apply cross validation to find optimal hyperparameters. Those are tested on the left-out fold from the outer loop, and the mean test error of all folds is the final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

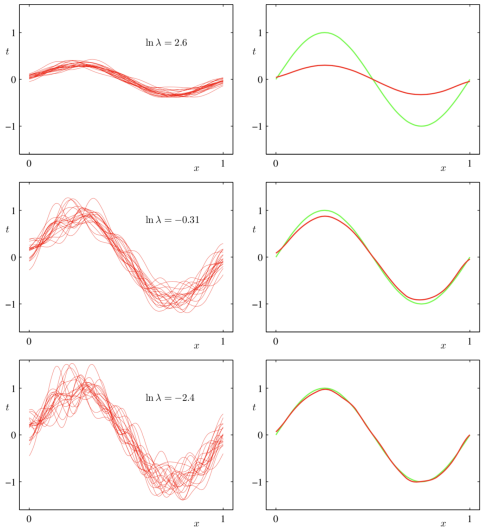


Figure 2: Illustration of dependence of bias and variance on model complexity controlled by λ. Less complex models (high λ) tend to have a high bias (be far off the correct distribution) but it is more robust regarding the actual dataset (therefore, a low variance). Decreasing λ results in a lower bias,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

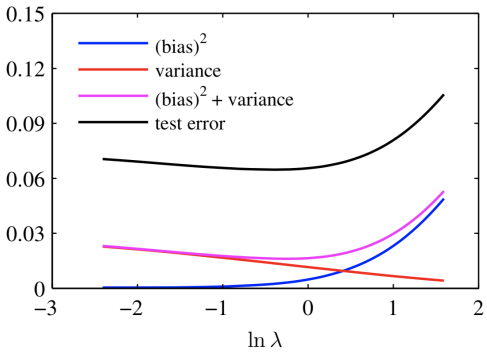


Figure 3: Plot of decomposed loss function for example of Figure 2. The goal is to minimize the test error. It is common that this is close to the minimum value of $(\text{bias})^2 + \text{variance}$. High variance as on the left indicates overfitting, high bias error on the left shows that the model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

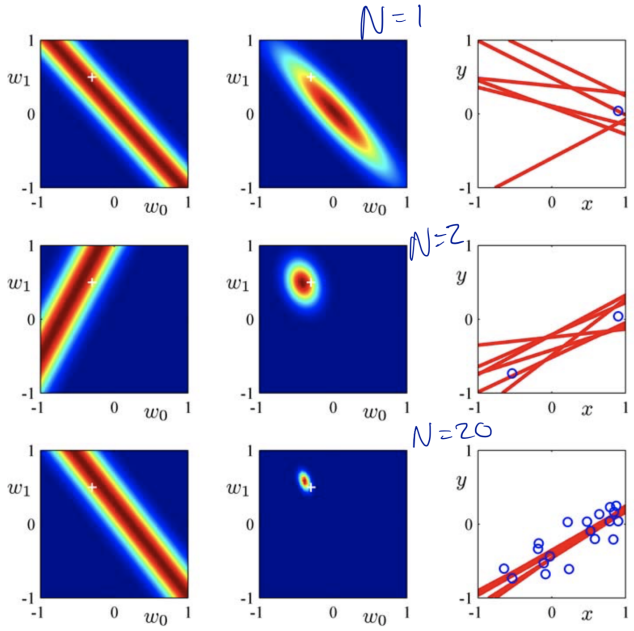


Figure 4: Example for Sequential Bayesian Learning on target $t = -0.3 + 0.5x + \epsilon$. First column: likelihood (not normalized for $\boldsymbol{w}$, but for t_n!), second column: posterior, third column: sampled weights

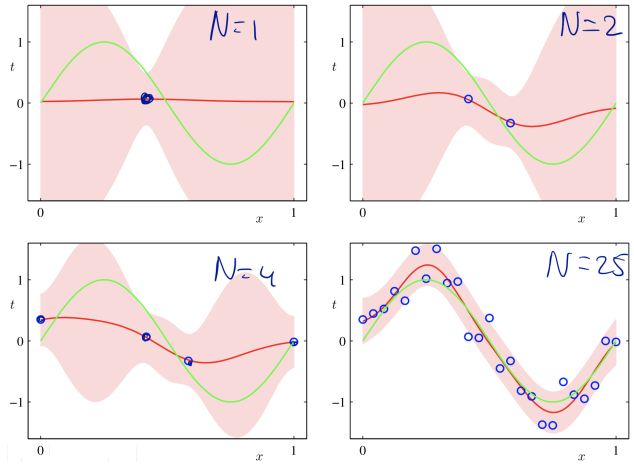


Figure 5: Example for predictive distributions. Green: ground truth data, blue: data points, red line: mean prediction, red area: 1-sigma area

– 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

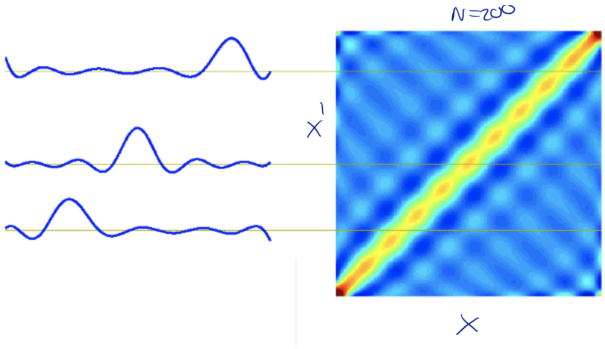


Figure 6: Right plot: matrix for (x', x) of kernel $k\left(x', x\right)$ for Gaussian basis function. Left plot: slices of this matrix for different values of x

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

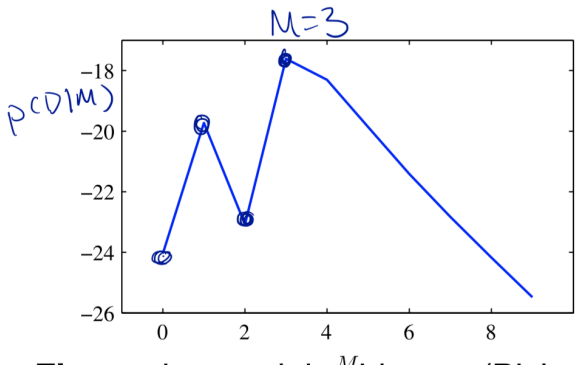


Figure 7: Plotting the curve of $\ln p(\mathcal{D}|\mathcal{M}_i)$ for different polynomials $M = 0, 1, \ldots$ for the task of fitting a sine. As the sine is an odd function, polynomials of odd order fit the best (give the most improvement for the model fit). However, increasing the model complexity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

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

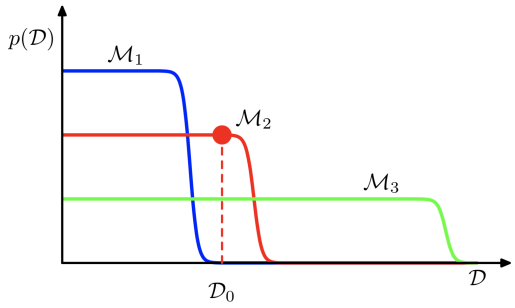


Figure 8: Illustration of three different models and there corresponding model evidences. Horizontal axis x: one dimensional representation of all possible datasets; Vertical axis y: probability that these models generate this specific dataset based on their prior distribution of parameters $\boldsymbol{w}$. $\mathcal{M}_1$ 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!

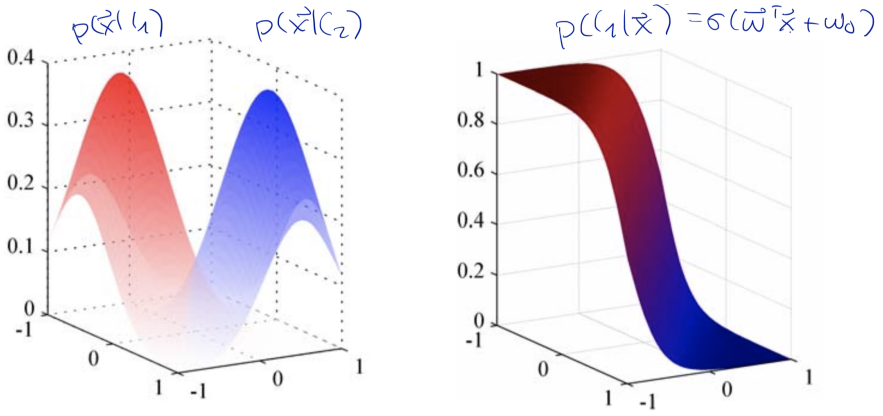


Figure 9: Left: Gaussian class-conditional densities, Right: corresponding posterior with sigmoid

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)

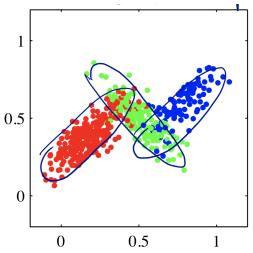


Figure 10: Example for three classes with different covariance matrices $\boldsymbol{\Sigma}_k^{-1}$. Linear discriminant analysis fails as it estimates a weighted sum of the sample covariance, and the distribution of green class significantly differs from the other two. The resulting estimate would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

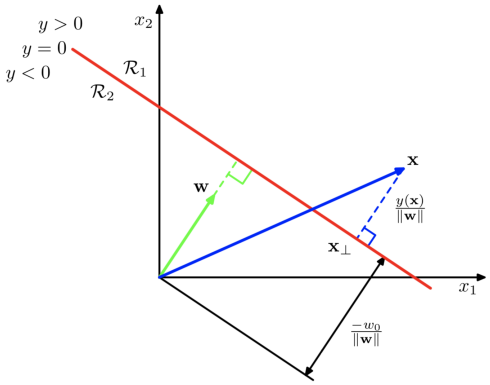


Figure 11: Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\boldsymbol{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\boldsymbol{x}$ 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 $[0, 1]$

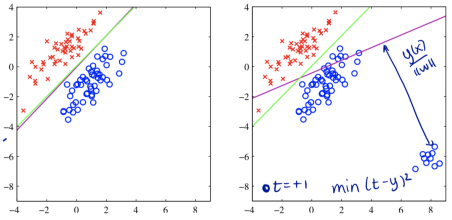


Figure: least squares is very sensitive to outliers (Bishop 4.4)

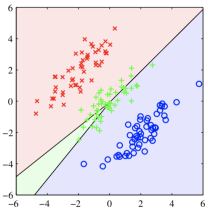


Figure: masking for least squares for K>2 (Bishop 4.5)

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 $\boldsymbol{w}_{\text{ML}} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$

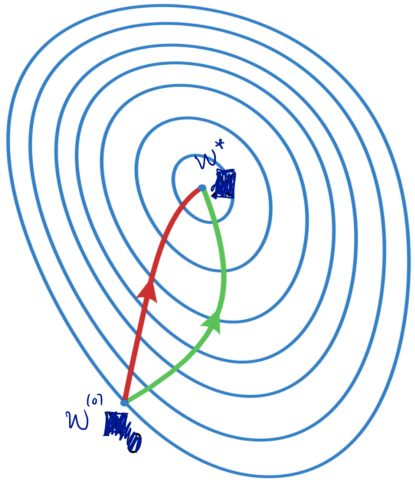


Figure 13: Illustration of SGD (green) and Newton Raphson (red). SGD always goes in the direction of the steepest gradient and is therefore slower than Newton-Raphson.

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

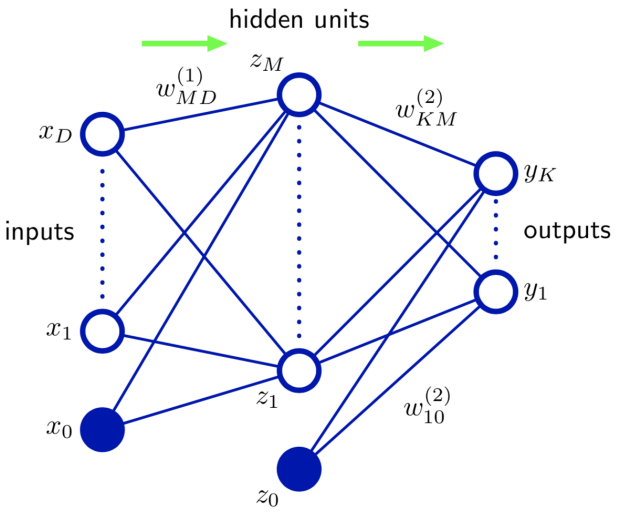


Figure 14: Illustration of a multilayer perceptron (2 layers).

- 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

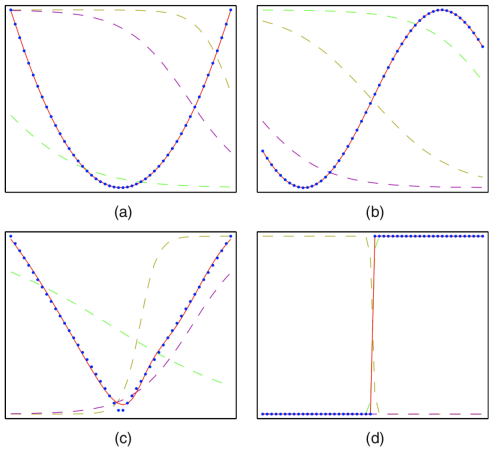


Figure 15: Example approximations by a 2-layer network with 3 hidden units of the function (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c) $f(x) = |x|$ and (d) $f(x) = \mathbb{I}(x > 0)$. The outputs of the three hidden units are shown as dashed lines.

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**:

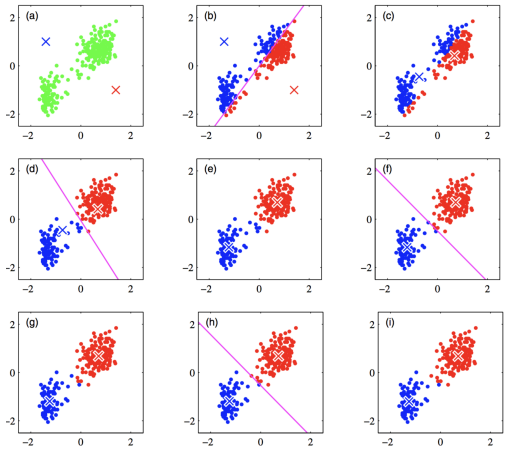


Figure 16: Illustration of the K-means algorithm. First, an expectation step is performed where the data points are assigned to a cluster (see *(b)*, *(d)*, *(f)*, and *(h)*), and then the maximization step optimizes the means of the clusters (see *(c)*, *(e)*, *(g)*, and *(i)*).

  - 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

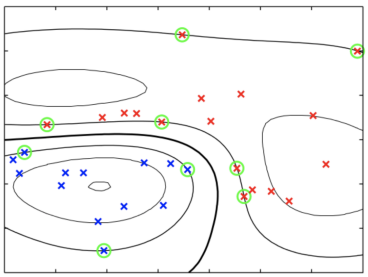


Figure 17: Visualization of non-linear support vectors

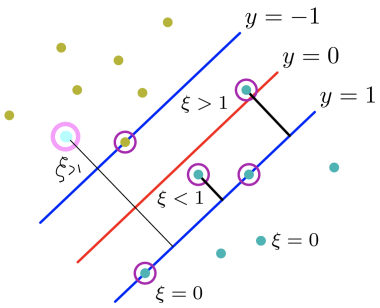


Figure 18: A soft margin classifier uses slack variables to penalize data points on the wrong side.

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

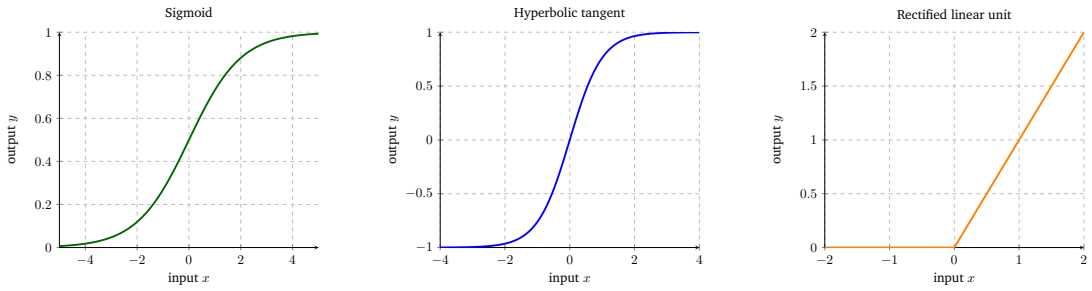
Properties of the ReLU function:

- $\text{ReLU}(x) = \max(x, 0)$
- $\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undef} & \text{if } x = 0 \end{cases}$ (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