Cognitive Algorithms
Lecture 6

Recap Lecture

Klaus-Robert Müller, Johannes Niediek,
Augustin Krause, Joanina Oltersdorff, Ken Schreiber

Technische Universität Berlin
Machine Learning Group

Organizational

Do not forget to register for your exam in Moses (or ISIS if Moses is not an option for you)!

Day	Time	Event	Room(s)
13th of February	12.15–15.00	Tutorial on test exam	MA 004
19th of February	14.00–16.30	Exam date 1	A151 (and EB301?)
8th of April	08.30–11.00	Exam date 2	A151 (and H1058?)

- The duration of the written exam itself is 90 minutes.
- Check ISIS for the schedule (room opening time, exam start time, etc.).
- If two rooms are used for the exam, instructions will be announced via ISIS.

Kernel ridge regression

Plan for this part

- Kernels in general
- Howto: Kernel ridge regression

Basic idea of kernel methods

The kernel trick (also called kernel substitution)

For any algorithm that can be formulated such that the input vectors $\boldsymbol{x}$, $\boldsymbol{y}$ enter only in terms of scalar products $\boldsymbol{x}^\top \boldsymbol{y}$:

We can replace each scalar product by a kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x})^\top \varphi(\boldsymbol{y})$.

Why should we do that?

- $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{d}\times 1}$ instead of $\boldsymbol{x}$ with typically $\tilde{d} \gg d$

  We can construct more complex (powerful) models.
- By using kernel $k(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{1\times 1}$

  We do not need to explicitly calculate high-dimensional $\varphi(\boldsymbol{x})$.

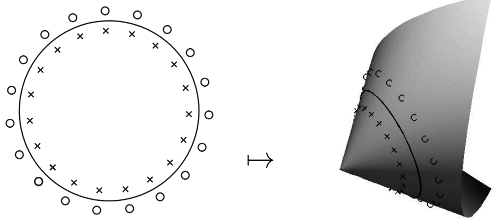
Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}$$

The corresponding kernel:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \varphi(\boldsymbol{x})^\top \cdot \varphi(\boldsymbol{y}) \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2) \cdot (y_1^2, \sqrt{2} y_1 y_2, y_2^2)^T \\ &= x_1^2 y_1^2 + 2 x_1 x_2 y_1 y_2 + x_2^2 y_2^2 \\ &= (x_1 y_1 + x_2 y_2)^2 = (\boldsymbol{x}^\top \boldsymbol{y})^2 \end{aligned}$$

Visualizing $\varphi(\boldsymbol{x})$



With $k(\cdot, \cdot)$ we implicitly work in $\mathbb{R}^3$, but only operate in $\mathbb{R}^2$

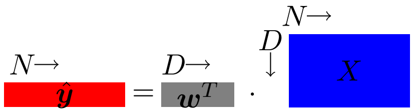
Some questions

- What are the relevant properties of kernel functions $k(\boldsymbol{x}, \boldsymbol{y})$ that we learned?
- What do we mean by *kernelizing* an algorithm?

Recap: linear regression

The linear regression model in matrix notation

$$\hat{\boldsymbol{y}} = \boldsymbol{w}^\top X.$$



Linear regression minimizes the least-squares loss function

$$\mathcal{E}_{\mathsf{LSQ}}(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 = \|\boldsymbol{y} - \boldsymbol{w}^\top X\|^2$$

($\boldsymbol{y}$ is a row vector, $\boldsymbol{w}$ is a column vector.)

Recap: ridge regression

Linear ridge regression finds $\boldsymbol{w}$ that minimizes the prediction error under constraints on the norm $\|\boldsymbol{w}\|$.

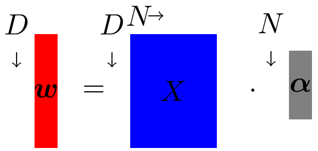
We can write the ridge regression error in several ways:

$$\begin{aligned}\mathcal{E}_{\mathsf{RR}}(\boldsymbol{w}) =& \sum_{i=1}^{n}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \lambda \sum_{i=1}^{d} w_i^2 \\ =& \|\boldsymbol{y} - \boldsymbol{w}^\top X\|^2 + \lambda \|\boldsymbol{w}\|^2 \\ =& \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}\end{aligned}$$

Kernel trick

We can use kernels if algorithm depends on training data X and test sample $\boldsymbol{x}$ only through scalar products.

From linear to kernel ridge regression



$$\mathcal{E}_{\text{RR}}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Main step: plug $\boldsymbol{w} := X\boldsymbol{\alpha}$ into the error function $\mathcal{E}_{\text{RR}}$:

$$\mathcal{E}_{\text{RR}}(\boldsymbol{\alpha}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{\alpha}^\top \underbrace{X^\top X}_{K}\boldsymbol{y}^\top + \boldsymbol{\alpha}^\top \underbrace{X^\top X}_{K}\underbrace{X^\top X}_{K}\boldsymbol{\alpha} + \lambda\boldsymbol{\alpha}^\top \underbrace{X^\top X}_{K}\boldsymbol{\alpha}$$

- We call this form **dual representation**
- Only scalar products appear, thus we can put in kernels:

$$\boldsymbol{x}_i^\top \boldsymbol{x}_j \to \varphi(\boldsymbol{x}_i)^\top \varphi(\boldsymbol{x}_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_{ij}$$

- We write $k(X, X)$ as K

How to compute α

$$\mathcal{E}_{\mathsf{RR}}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Computing the derivative with respect to $\boldsymbol{w}$ yields

$$\frac{\partial \mathcal{E}_{\mathsf{RR}}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -2X\boldsymbol{y}^\top + 2XX^\top \boldsymbol{w} + 2\lambda \boldsymbol{w}$$

Setting the gradient to 0 and rearranging terms the optimal $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = X\boldsymbol{\alpha} = X \underbrace{\frac{1}{\lambda}(\boldsymbol{y}^\top - X^\top \boldsymbol{w})}_{:=\boldsymbol{\alpha} \in \mathbb{R}^{n \times 1}} = \sum_i^n \alpha_i \boldsymbol{x}_i$$

Kernel ridge regression

$$
\begin{aligned}
\boldsymbol{\alpha} &= \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w}) \\
\lambda\boldsymbol{\alpha} &= \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha} \\
\boldsymbol{y}^\top &= (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha} \\
\boldsymbol{\alpha} &= (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)^{-1}\boldsymbol{y}^\top \\
\boldsymbol{\alpha} &= (K + \lambda I)^{-1}\boldsymbol{y}^\top
\end{aligned}
$$

This $\boldsymbol{\alpha}$ minimizes $\mathcal{E}_{\text{RR}}(\boldsymbol{\alpha})$.

Remember regular RR: train $\boldsymbol{w}$ which minimizes $\mathcal{E}_{\text{RR}}(\boldsymbol{w})$

$$\boldsymbol{w} = (\varphi(X)\varphi(X)^\top + \lambda I)^{-1}\varphi(X)y^\top$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{d}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^\top \boldsymbol{w})$

KRR trains $\boldsymbol{\alpha} \in \mathbb{R}^n$, RR trains $\boldsymbol{w} \in \mathbb{R}^{\tilde{d}}$

If $k(\boldsymbol{x}, \boldsymbol{y})$ comes from a feature map φ, the two models are equivalent (but not the runtime complexity)

Predictions for new data $\boldsymbol{x}_{new}$

$$\begin{aligned}
y_{new} &= \boldsymbol{w}^\top \varphi(\boldsymbol{x}_{new}) \\
&= (\varphi(X_{train})\boldsymbol{\alpha})^\top \varphi(\boldsymbol{x}_{new}) \\
&= \boldsymbol{\alpha}^\top \varphi(X_{train})^\top \varphi(\boldsymbol{x}_{new}) \\
&= \boldsymbol{\alpha}^\top k(X_{train}, \boldsymbol{x}_{new}) \\
&= \boldsymbol{y}_{train}(K + \lambda I)^{-1} k(X_{train}, \boldsymbol{x}_{new})
\end{aligned}$$

Optimal $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y}^\top$

$\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

$K(a, b) = K(b, a)^\top$

We call $K = k(X_{train}, X_{train})$ the *Gram matrix*

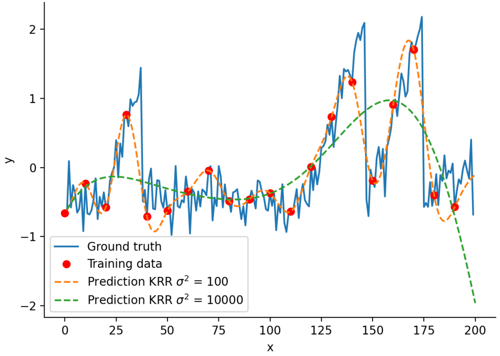
Kernel ridge regression example

KRR with Gaussian kernels

$$k(x,x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$$

Predictions:

$$y_{new} = y_{train}(K+\lambda I)^{-1}k(X_{train}, \boldsymbol{x}_{new})$$



Some questions

- Given a data matrix X and feature function φ, can you compute the Gram matrix K?
- Why can predicting y_{new} in kernel ridge regression be problematic for very large training data sets?
- Sketch the shape of the Gaussian kernel for two different values of σ.

Model evaluation

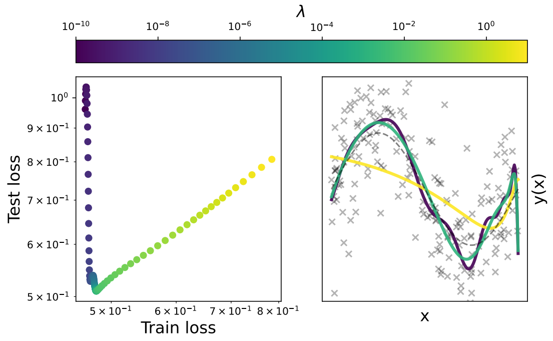
- Many algorithms have *hyper-parameters* that can be set by the user (their optimization is not part of the model training)
  How do we find good hyper-parameters?
- We want to know how good our models are
  How do we estimate our model's performance?

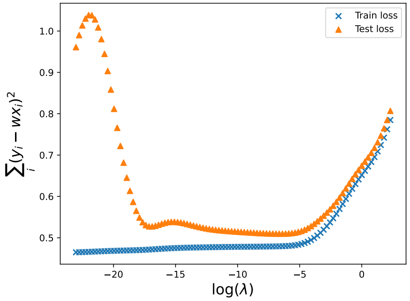
Model selection

How to find good hyper-parameters (e.g., λ and σ for KRR with radial basis functions)?

One option: grid search

$\rightarrow$ try out e.g. $\lambda \in \{0, 0.1, ..., 0.9, 1.0\}$ and choose the one with the lowest error on test set

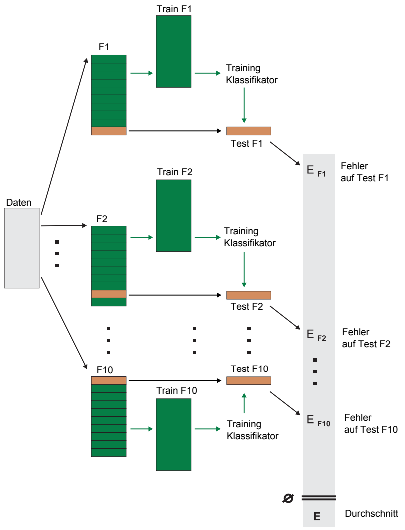




Cross-validation

Algorithm 1: Cross-Validation

```
Require: Data $(x_1, y_1) \dots, (x_N, y_N)$, Number of CV folds $F$
1: # Split data in $F$ disjunct folds
2: for folds $f = 1, \dots, F$ do
3:     # Train model on folds $\{1, \dots, F\} \setminus f$
4:     # Compute prediction error on fold $f$
5: end for
6: # Average prediction error
```



Cross-validation: Can be used differently

Model Evaluation
"How well does my model perform?"
Report **mean evaluation score**
(for example accuracy) across folds

Model Selection
"What hyperparameter should I use?"
Do grid search on every fold.
Take parameter with the highest mean test score across folds.

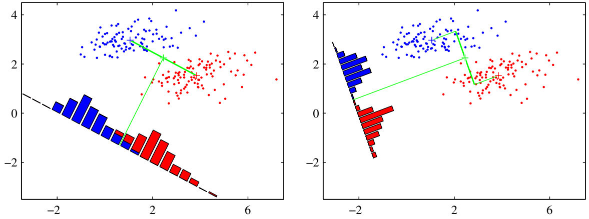
- We cannot select a model **and** evaluate it at the same time with simple CV.
  We would be too optimistic, because we use the same test set for optimizing and evaluating.
- After CV you still need to train your model on the whole data-set
- To estimate a model's performance under optimal hyper-parameters, use *nested CV*

Nested cross-validation

Algorithm 2: Cross-Validation for Model Selection and Evaluation

```
Require: Data $(x_1, y_1) \dots, (x_N, y_N)$, parameters $\sigma_1, \dots, \sigma_S$, Number of CV folds $F$
 1: Split data in $F$ disjunct folds
 2: for Outer folds $f_{\text{outer}} = 1, \dots, F$ do
 3:     Pick folds $\{1, \dots, F\} \setminus f_{\text{outer}}$ for Model Selection
 4:     Model Selection
 5:     for Fold $f_{\text{inner}} = 1, \dots, F-1$ do
 6:         for Parameter $s = 1, \dots, S$ do
 7:             Train model on folds $\{1, \dots, F\} \setminus \{f_{\text{outer}}, f_{\text{inner}}\}$ with parameter $\sigma_s$
 8:             Compute prediction on fold $f_{\text{inner}}$
 9:         end for
10:     end for
11:     Pick best parameter $\sigma_s$ for all $f_{\text{inner}}$
12:     Model Evaluation
13:     Train model on folds $\{1, \dots, F\} \setminus f_{\text{outer}}$ with parameter $\sigma_s$
14:     Performance$_{\text{outer}}$ $\leftarrow$ Test model on fold $f_{\text{outer}}$
15: end for
16: return Average of Performance$_{\text{outer}}$
```

Recap: Fisher's linear discriminant



Goal: Find a (normal vector of a linear decision boundary) $\boldsymbol{w} \in \mathbb{R}^d$ that

Maximizes mean class difference, and

Minimizes variance in each class

Maximize the **Fisher criterion**:

$$J(\boldsymbol{w}) = \frac{\text{between class variance}}{\text{within class variance}} = \frac{(\mu_o - \mu_\Delta)^2}{\sigma_o^2 + \sigma_\Delta^2}$$

where $\boldsymbol{x}_{1o}, \ldots, \boldsymbol{x}_{n_o o} \in \mathbb{R}^d$ and

Linear Discriminant Analysis

After some calculations, we see that the optimal weight vector $\boldsymbol{w}$ is given by

$$\boldsymbol{w} = \underset{\boldsymbol{w}'}{\operatorname{argmax}}\, J(\boldsymbol{w}') = \underset{\boldsymbol{w}'}{\operatorname{argmax}} \frac{\boldsymbol{w}'^\top S_B \boldsymbol{w}'}{\boldsymbol{w}'^\top S_W \boldsymbol{w}'}$$

To optimize the Fisher criterion, we set its derivative (with respect to $\boldsymbol{w}$) to 0

$$\begin{aligned} 0 &= \frac{\partial}{\partial \boldsymbol{w}} J(\boldsymbol{w})\Big|_{\boldsymbol{w}} = \frac{(\boldsymbol{w}^\top S_W \boldsymbol{w}) S_B \boldsymbol{w} - (\boldsymbol{w}^\top S_B \boldsymbol{w}) S_W \boldsymbol{w}}{(\boldsymbol{w}^\top S_W \boldsymbol{w})^2} \\ (\boldsymbol{w}^\top S_B \boldsymbol{w}) S_W \boldsymbol{w} &= (\boldsymbol{w}^\top S_W \boldsymbol{w}) S_B \boldsymbol{w} \\ S_W \boldsymbol{w} &= S_B \boldsymbol{w} \underbrace{\frac{\boldsymbol{w}^\top S_W \boldsymbol{w}}{\boldsymbol{w}^\top S_B \boldsymbol{w}}}_{scalar \equiv \lambda} \end{aligned}$$

Linear Discriminant Analysis

$$\boldsymbol{w} = \operatorname*{argmax}_{\boldsymbol{w}'} \frac{\boldsymbol{w}'^\top S_B \boldsymbol{w}'}{\boldsymbol{w}'^\top S_W \boldsymbol{w}'}$$

$$\rightarrow S_W \boldsymbol{w} = S_B \boldsymbol{w} \lambda$$

Now we plug $S_B = (\bar{\boldsymbol{x}}_o - \bar{\boldsymbol{x}}_\Delta)(\bar{\boldsymbol{x}}_o - \bar{\boldsymbol{x}}_\Delta)^\top$ in

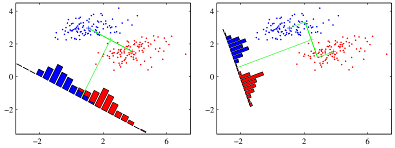
$$S_B \boldsymbol{w} = (\bar{\boldsymbol{x}}_o - \bar{\boldsymbol{x}}_\Delta) \underbrace{(\bar{\boldsymbol{x}}_o - \bar{\boldsymbol{x}}_\Delta)^\top \boldsymbol{w}}_{\text{scalar}}$$

finally, left multiplying with S_W^{-1} yields

$$\boldsymbol{w} \propto S_W^{-1}(\bar{\boldsymbol{x}}_o - \bar{\boldsymbol{x}}_\Delta).$$

($\propto$ denotes proportionality, e.g. $x \propto 2x$)

Interim summary



Goal

Find $\boldsymbol{w} \in \mathbb{R}^d$ that (when used for projection)

- maximizes mean class difference
- minimizes variance in each class

Formalization

Maximize the **Fisher criterion**

$$J(\boldsymbol{w}) = \frac{\text{between class variance}}{\text{within class variance}} = \frac{(\mu_o - \mu_\Delta)^2}{\sigma_o^2 + \sigma_\Delta^2}$$

Solution

After some calculations...

$$\boldsymbol{w} \propto S_W^{-1}(\bar{\boldsymbol{x}}_o - \bar{\boldsymbol{x}}_\Delta)$$

Probabilistic modeling

Fisher's LDA. . .

- . . . makes no assumptions about the data
- . . . does not directly yield a decision rule of the form $w^\top x - \beta > 0$, because β still has to be set

Let's take a different approach and let's assume the data are normally distributed.

- For class Δ assume

$$\boldsymbol{x} \sim \mathcal{N}(\bar{\boldsymbol{x}}_\Delta, S_\Delta), \text{that is } p(\boldsymbol{x}|\Delta) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{\det(S_\Delta)}} e^{-\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}}_\Delta)^\top S_\Delta^{-1}(\boldsymbol{x}-\bar{\boldsymbol{x}}_\Delta)}$$

- For class $\bigcirc$ assume

$$\boldsymbol{x} \sim \mathcal{N}(\bar{\boldsymbol{x}}_\bigcirc, S_\bigcirc), \text{that is } p(\boldsymbol{x}|\bigcirc) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{\det(S_\bigcirc)}} e^{-\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}}_\bigcirc)^\top S_\bigcirc^{-1}(\boldsymbol{x}-\bar{\boldsymbol{x}}_\bigcirc)}$$

Probabilistic modeling

To classify new $\boldsymbol{x}$, we would like to check if $p(\Delta|\boldsymbol{x}) > p(\bigcirc|\boldsymbol{x})$.
Use Bayes' theorem:

$$p(\Delta|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\Delta)p(\Delta)}{p(\boldsymbol{x})}.$$

Thus,

$$\begin{aligned} p(\Delta|\boldsymbol{x}) &> p(\bigcirc|\boldsymbol{x}) \\ \Leftrightarrow \frac{p(\boldsymbol{x}|\Delta)p(\Delta)}{p(\boldsymbol{x})} &> \frac{p(\boldsymbol{x}|\bigcirc)p(\bigcirc)}{p(\boldsymbol{x})} \\ \Leftrightarrow \frac{p(\Delta)}{p(\bigcirc)} \cdot \frac{p(\boldsymbol{x}|\Delta)}{p(\boldsymbol{x}|\bigcirc)} &> 1. \end{aligned}$$

Probabilistic modeling

So far, the decision rule is

$$\frac{p(\Delta)}{p(\bigcirc)} \cdot \frac{p(\boldsymbol{x}|\Delta)}{p(\boldsymbol{x}|\bigcirc)} > 1.$$

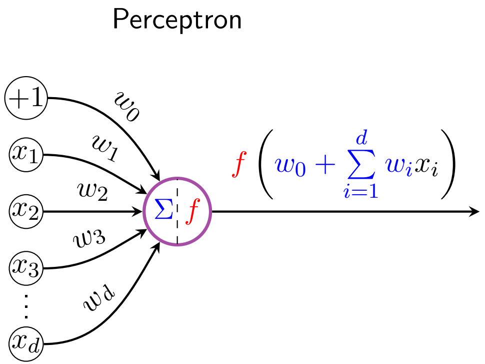
For additional simplification,

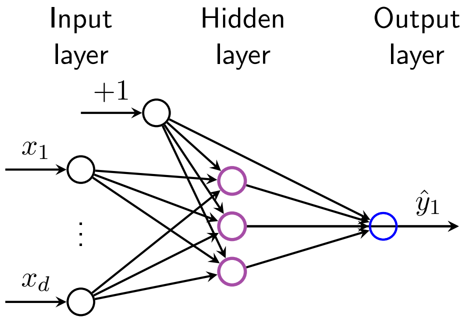
- estimate $\frac{p(\Delta)}{p(\bigcirc)}$ as $\frac{n_\Delta}{n_\bigcirc}$ (ratio of elements per class)
- assume $S_\Delta = S_\bigcirc =: S$.
- take the logarithm of the decision rule

Then, the decision rule becomes

$$\log\left(\frac{n_\Delta}{n_\bigcirc}\right) + \frac{1}{2}\left(-(\boldsymbol{x} - \bar{\boldsymbol{x}}_\Delta)^\top S^{-1}(\boldsymbol{x} - \bar{\boldsymbol{x}}_\Delta) + (\boldsymbol{x} - \bar{\boldsymbol{x}}_\bigcirc)^\top S^{-1}(\boldsymbol{x} - \bar{\boldsymbol{x}}_\bigcirc)\right) > 0$$

$$\Leftrightarrow \underbrace{(\bar{\boldsymbol{x}}_\Delta - \bar{\boldsymbol{x}}_\bigcirc)^\top S^{-1}}_{\boldsymbol{w}^\top} \boldsymbol{x} + \underbrace{\frac{1}{2}\left(\bar{\boldsymbol{x}}_\bigcirc S^{-1} \bar{\boldsymbol{x}}_\bigcirc - \bar{\boldsymbol{x}}_\Delta S^{-1} \boldsymbol{x}_\Delta\right) + \log\left(\frac{n_\Delta}{n_\bigcirc}\right)}_{-\beta} > 0$$



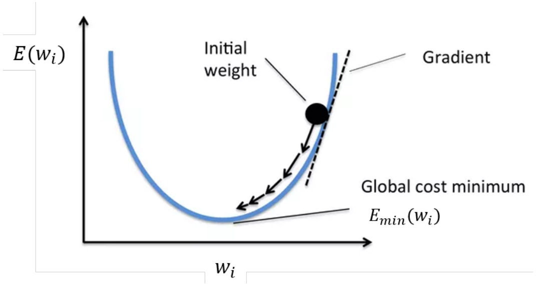


Adding more perceptrons in parallel yields
a multi-layer perceptron (MLP) with a single hidden layer

$$\hat{y}_1(x) = \sum_j w_j^o f(\boldsymbol{w}_j^{h\top} \boldsymbol{x} + w_{j,0}^h) + w_0^o$$

How to train a multi-layer perceptron

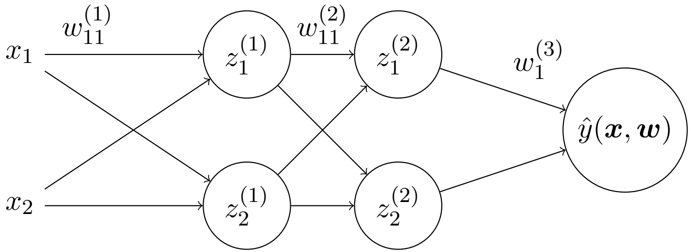
Gradient descent in parameter space



$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}},$$

where E is some error function and η is the learning rate.
Note: in practice, the error minimum is only a local minimum, not a global minimum.

Gradient descent in an MLP by example



$$z_1^{(1)} = \sigma(a_1^{(1)}) = \sigma(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2),$$

where σ is the logistic function ($\sigma(x) = \frac{1}{1+e^{-x}}$) and is used for all activations. The other $z_i^{(j)}$ are defined analogously, and $\hat{y}$ does not use any activation function. We use the quadratic error function.

Gradient descent in an MLP by example

Given a learning rate η, calculate a weight update for the weight $w_{11}^{(1)}$, step by step.

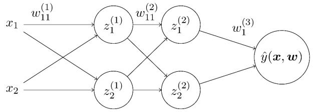
1. Write down the quadratic error as a function of $\boldsymbol{x}$, $\boldsymbol{w}$, and y.

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{w}, y) = \frac{1}{2}(\hat{y}(\boldsymbol{x}, \boldsymbol{w}) - y)^2$$

2. Calculate the partial derivative $\frac{\partial \mathcal{E}}{\partial w_{11}^{(1)}}$ (see slide 31).
3. Update the weight to decrease the error.

$$w_{11}^{(1)} \longleftarrow w_{11}^{(1)} - \eta \frac{\partial \mathcal{E}}{\partial w_{11}^{(1)}}.$$

Example: derivatives carried out



$$\frac{\partial}{\partial w_{11}^{(1)}}\mathcal{E}(\boldsymbol{x},\boldsymbol{w},y) = \frac{\partial}{\partial w_{11}^{(1)}}\frac{1}{2}(\hat{y}(\boldsymbol{x},\boldsymbol{w})-y)^2 = \frac{\partial\mathcal{E}}{\partial\hat{y}}\cdot\frac{\partial}{\partial w_{11}^{(1)}}\hat{y}$$

$$\frac{\partial\mathcal{E}}{\partial\hat{y}} = \hat{y}(\boldsymbol{x},\boldsymbol{w})-y$$

$$\frac{\partial}{\partial w_{11}^{(1)}}\hat{y} = \frac{\partial}{\partial w_{11}^{(1)}}\left(w_1^{(3)}z_1^{(2)}+w_2^{(3)}z_2^{(2)}\right) = w_1^{(3)}\frac{\partial}{\partial w_{11}^{(1)}}z_1^{(2)} + w_2^{(3)}\frac{\partial}{\partial w_{11}^{(1)}}z_2^{(2)}$$

$$\frac{\partial}{\partial w_{11}^{(1)}}z_1^{(2)} = \frac{\partial}{\partial w_{11}^{(1)}}\sigma(a_1^{(2)}) = \sigma'(a_1^{(2)})\frac{\partial}{\partial w_{11}^{(1)}}a_1^{(2)}$$

$$\frac{\partial}{\partial w_{11}^{(1)}}a_1^{(2)} = \frac{\partial}{\partial w_{11}^{(1)}}\left(w_{11}^{(2)}z_1^{(1)}+w_{21}^{(2)}z_2^{(1)}\right) = w_{11}^{(2)}\frac{\partial}{\partial w_{11}^{(1)}}z_1^{(1)} + w_{21}^{(2)}\cdot 0$$

$$\frac{\partial}{\partial w_{11}^{(1)}}z_1^{(1)} = \sigma'(a_1^{(1)})\frac{\partial}{\partial w_{11}^{(1)}}a_1^{(1)} = \sigma'(a_1^{(1)})\frac{\partial}{\partial w_{11}^{(1)}}\left(w_{11}^{1}x_1+w_{21}^{(1)}x_2\right) = x_1$$

Putting everything together

$$\frac{\partial}{\partial w_{11}^{(1)}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{w}, y) = (\hat{y}(\boldsymbol{x}, \boldsymbol{w}) - y) \cdot \left(w_1^{(3)} \sigma'(a_1^{(2)}) w_{11}^{(2)} \sigma'(a_1^{(1)}) x_1 + w_2^{(3)} \sigma'(a_2^{(2)}) w_{12}^{(2)} \sigma'(a_1^{(1)}) x_1 \right)$$

Note:

- We still have to plug-in σ' explicitly
- $\mathcal{E}$ depends on $w_{11}^{(1)}$ through $z_1^{(2)}$ and $z_2^{(2)}$, both depend on $z_1^{(1)}$
  Thus, $\frac{\partial}{\partial w_{11}^{(1)}} z_1^{(1)}$ appears twice
- If $\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = y$, no weight update performed

Non-negative matrix factorization (NMF)

- PCA is the main tool for dimensionality reduction and data preprocessing that we learned
- Please study the PCA algorithm and the k-means algorithm
- For some data PCA is not intuitive
- Example: Non-negative data
  - Principal directions will have negative entries
  - This can be hard to interpret
- Many data sets are strictly non-negative
  - Text data
  - Image data
  - Probabilistic data
- NMF is straightforward to implement
- Matrix factorization is relevant in recommender systems ("you might also like...")

Non-negative matrix factorization

Notation: $\mathbb{R}_+ := \{x \in \mathbb{R} \,|\, x \geq 0\}$.
Given non-negative data $X \in \mathbb{R}_+^{d \times n}$ we want to find $W \in \mathbb{R}_+^{d \times m},\ H \in \mathbb{R}_+^{m \times n}$ such that the distance between X and WH is minimal, where distance is measured as the Frobenius norm.
W and H are given by

$$\operatorname*{argmin}_{W,H} \|X - WH\|_{\mathsf{Fro}}^2,$$

which is by definition of $\|\cdot\|_{\mathsf{Fro}}$

$$= \operatorname*{argmin}_{W,H} \sum_{i=1}^{d} \sum_{j=1}^{n} \left(X_{ij} - (WH)_{ij}\right)^2 .$$

Non-negative matrix factorization

NMF: Important facts

- Write non-negative data as a product of non-negative matrices: $X = WH$ (all X_{ij}, W_{ij}, $H_{ij} \geq 0$).
- Find W and H by gradient descent, where the gradient is calculated on $\|X - WH\|_{\text{Fro}}$, with respect to all matrix elements W_{ij} and H_{ij} of W and H.
- The gradient update is *multiplicative*. This ensures that non-negative entries stay non-negative.