*** Applied Machine Learning Fundamentals ***
Neural Networks / Deep Learning

M. Sc. Daniel Wehner

SAP SE

Winter term 2019/2020

Find all slides on GitHub

Lecture Overview

Unit I	Machine Learning Introduction
Unit II	Mathematical Foundations
Unit III	Bayesian Decision Theory
Unit IV	Probability Density Estimation
Unit V	Regression
Unit VI	Classification I
Unit VII	Evaluation
Unit VIII	**Classification II**
Unit IX	Clustering
Unit X	Dimensionality Reduction

Agenda for this Unit

Section:

Introduction

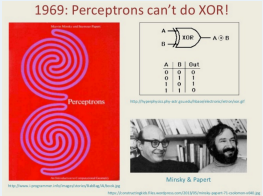
What is Deep Learning?

- 'Deep Learning' is a fancy new term for 'artificial neural networks'
- It is a **supervised** method and **model based**
- Artificial neural networks are inspired by the human brain
- Lots of different architectures exist:
  - **Multi-Layer perceptrons (MLPs)**
  - **Radial Basis Function Networks (RBFNs)**
  - **Convolutional neural networks (CNNs, ConvNets)**
  - **Recurrent neural networks (LSTMs, GRUs, etc.)**
  - **Residual networks (ResNets)**

History of Deep Learning

Early booming (1950s – early 1960s)

F. Rosenblatt suggests the **Perceptron** learning algorithm: `Click here!`



Setback I (mid 1960s – late 1970s)

M. Minsky and *S. Papert* (1969): Serious problems with perceptron algorithm. It cannot learn the **XOR problem**.

History of Deep Learning (Ctd.)

Renewed enthusiasm (1980s)

- New techniques available
- **Backpropagation** for deep nets

Setback II (1990s – mid 2000s)

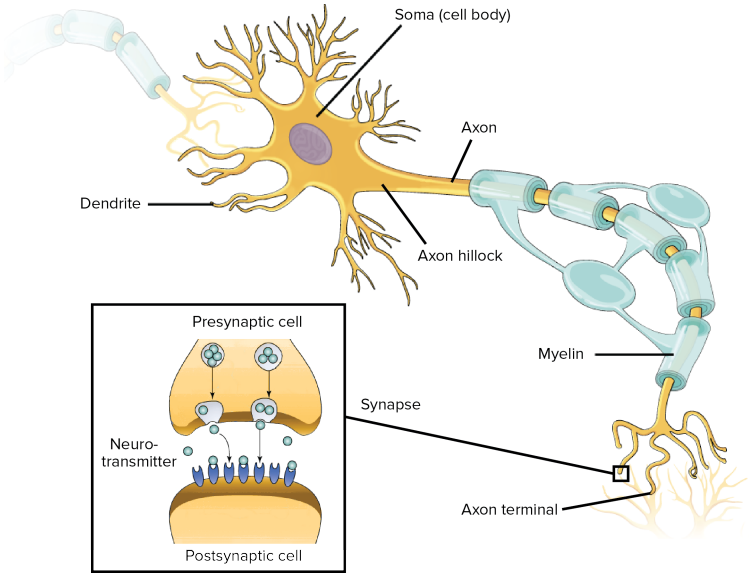
- Other techniques were considered superior (e. g. SVMs)
- CS journals rejected papers on neural networks

'Deep Learning' (since mid 2000)

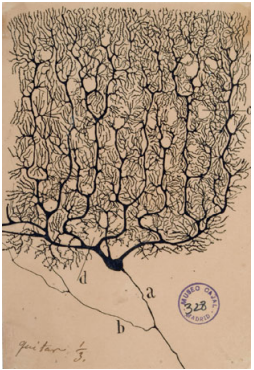
More data, faster computers, better optimization techniques...

Biological Motivation

- All neurons are connected and form a complex **network**
- **Transmitter chemicals** within the fluid of the brain influence the **electrical potential** inside the body of the neurons
- If the **membrane potential** reaches some threshold the neurons **fires** $\Rightarrow$ A pulse of fixed length is sent down the **axon**
- The axon connects the neuron with other neurons (via **synapses**)
- Probably there are 100 trillion (!!!) synapses in the human brain
- **Refractory period** after a neuron has fired


Soma (cell body)
Axon
Dendrite
Axon hillock
Myelin
Presynaptic cell
Synapse
Neuro-
transmitter
Axon terminal
Postsynaptic cell

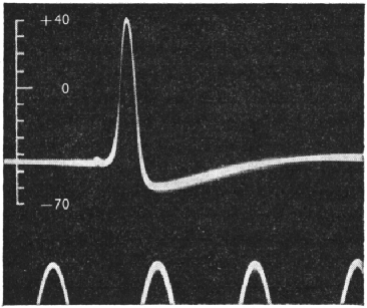
How can we know this?



- *Santiago Ramón y Cajal* made neurons visible by applying **Golgi's method**
- Golgi's method uses the Golgi stain to colorize the neurons
- Cajal establishes the **neuron doctrine** and won the Nobel price in 1906 for his work

How can we know this? (Ctd.)

- End of the 1940s *A. Hodgkin* and *A. Huxley* started investigating the electrical properties of neurons on the squid's[1] axon
- The right-hand-side image was the first **action potential** that was plotted



[1] lat.: *Loligo pealeii*

How do Humans / Animals learn?

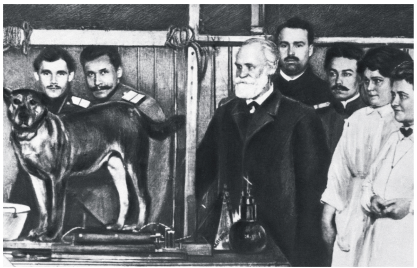
- **Idea:** Mechanism of learning is **association**
- **Hebbian learning:** If the firing of one neuron repeatedly assists in firing another neuron the synaptic connection will be strengthened

*'When an axon of cell A is near enough to excite a cell B and repeatedly or persistently **takes part in firing it**, some growth process or **metabolic change** takes place in one or both cells such that A's **efficiency**, as one of the cells firing B, is **increased**.'*

*'The general idea is an old one, that any two cells or systems of cells that are **repeatedly active at the same** time will tend to become **'associated'**, so that activity in one facilitates activity in the other.'* *Hebb*

Classical / Pavlovian Conditioning

- Dog salivates when given food
- Food is an **unconditioned stimulus (US)**
- Salivation in response to food is **unconditioned response (UR)**
- Food is paired with the sound of a bell
- Bell is **conditioned stimulus (CS)**
- Bell will eventually elicit salivation event without food
- Salivation is **conditioned response (CR)**

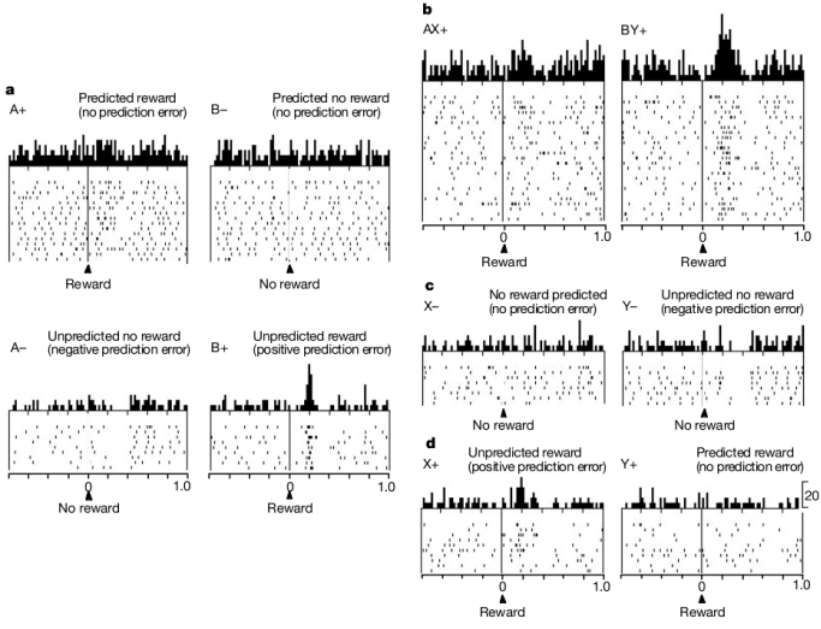


Classical / Pavlovian Conditioning (Ctd.)

Blocking

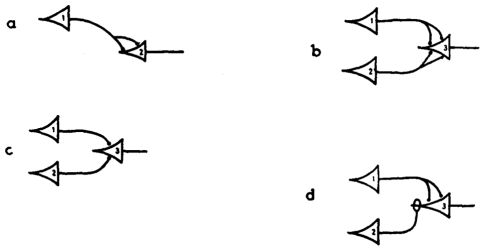
Group A	train N+	train LN+	test L-	$\Rightarrow$ no conditioning
Group B		train LN+	test L-	$\Rightarrow$ conditioning

- CS is a light (L), a noise (N), or a combination of both (LN)
- US is a mild shock that is paired with the CS in the training phase (+)
- In all conditions, after training fear response is tested when only L is presented without shock (-)
- Group B shows conditioning; Group A does not: **N blocks L**
- This is hard to explain with Hebbian learning
- **Idea:** Learning only happens if there is a **prediction error**

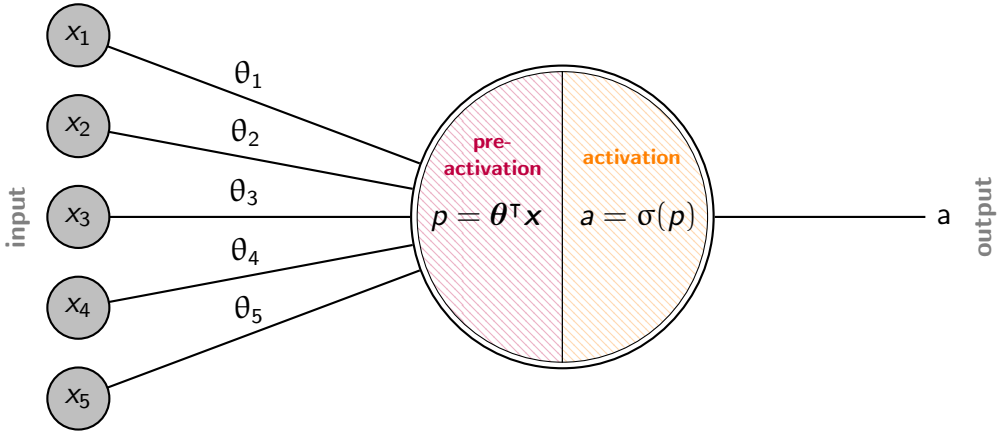

a
A+
Predicted reward
(no prediction error)
Reward
B–
Predicted no reward
(no prediction error)
No reward
A–
Unpredicted no reward
(negative prediction error)
0
1.0
No reward
B+
Unpredicted reward
(positive prediction error)
0
1.0
Reward
b
AX+
0
1.0
Reward
BY+
0
1.0
Reward
c
X–
No reward predicted
(no prediction error)
No reward
Y–
Unpredicted no reward
(negative prediction error)
No reward
d
X+
Unpredicted reward
(positive prediction error)
0
1.0
Reward
Y+
Predicted reward
(no prediction error)
20
0
1.0
Reward

Artificial Neurons [McCulloch and Pitts 1943]

- 1943 *W. S. McCulloch* and *W. H. Pitts* designed the first 'artificial neuron'
- These neurons can represent logical functions (e. g. b – `OR`, c – `AND`)



Perceptron [Rosenblatt 1957]



Perceptron (Ctd.)

- The neuron receives an input-vector $\boldsymbol{x}$

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_m)^\intercal$$

- Each input signal is weighted by a factor[2] θ_j

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_m)^\intercal$$

- We compute the **pre-activation** and the **activation**:

$$p = \boldsymbol{\theta}^\intercal \boldsymbol{x} + b = \sum_{j=1}^{m} \theta_j x_j + b \qquad a = \sigma(p) \qquad (1)$$

[2] weight of synaptic strength

Perceptron (Ctd.)

- The simplest activation function is to use a threshold ρ:[3]

$$\sigma(p) = \begin{cases} 0 & \text{for } p \leqslant \rho \\ 1 & \text{for } p > \rho \end{cases}$$

- Quick example: $\boldsymbol{x} = (1, 0, 0.5)^\intercal; \boldsymbol{\theta} = (1, -0.5, -1)^\intercal; \rho = 0$

$$p = \sum_{j=1}^{3} \theta_j x_j = 1 \cdot 1 + (-0.5) \cdot 0 + (-1) \cdot 0.5 = 0.5$$

$$a = \sigma_{\rho=0}(0.5) = 1$$

[3] Not used, since not differentiable; alternatives later

Perceptron Learning

- Learning means choosing the correct weights $\boldsymbol{\theta}^*$ from a set of possible hypotheses $\mathcal{H}$ (**hypothesis space**):

$$\mathcal{H} = \{\boldsymbol{\theta} | \boldsymbol{\theta} \in \mathbb{R}^m\}$$

- How to learn the weights from a data set $\mathcal{D}$?
- **Algorithm outline:**
  1. Pick a training example $\boldsymbol{x} \in \mathcal{D}$
  2. Calculate the activation a for that training example
  3. Update the weights θ based on the error

Perceptron Learning (Ctd.)

- Let the error be denoted by δ
- How can we compute the error? We need a loss function $\mathcal{J}(\boldsymbol{\theta})$:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{N} (h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2 \tag{2}$$

- Again, we use **gradient descent**: Compute gradient and go into the negative direction:

$$\boldsymbol{\theta}^{(t+1)} \longleftarrow \boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) \tag{3}$$

$\Rightarrow$ cf. slides 'Regression'

Algorithm 1: Perceptron Learning Algorithm

Input: Training data $\mathcal{D}$, convergence threshold ε

```
// initialization
```

set all weights $\boldsymbol{\theta}^{(0)}$ to small random numbers

for $t \in \{0, 1, \ldots, \infty\}$ **do**

  pick a sample $\langle \boldsymbol{x}, y \rangle \in \mathcal{D}$ randomly

  ```
  // predict the class label
  ```

  compute the activation $a = \sigma(\boldsymbol{\theta}^{\intercal} \boldsymbol{x})$

  ```
  // stochastic gradient descent:  update based on prediction error
  ```

  $\boldsymbol{\theta}^{(t+1)} \longleftarrow \boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$

  if $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\| \leqslant \varepsilon$ **then**

    ```
    // convergence
    ```

    break

return $\boldsymbol{\theta}$

Perceptron Convergence Theorem

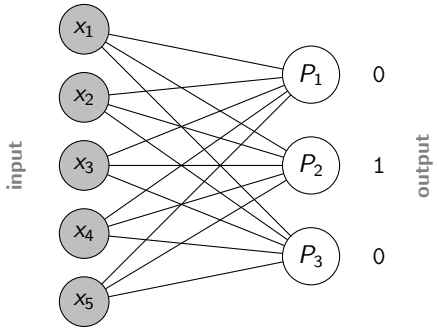
Perceptron Convergence Theorem

If the training data is **linearly separable**, then the perceptron learning algorithm is going to **converge after a finite amount of time** and classifies **all training data examples correctly**.

Generalization to multiple Classes

- A single neuron can only distinguish two classes
- If there are more than two classes: Simply use more perceptrons[4]
- Use **one-hot encoding** for the classes and **soft-max** as activation function (later)
- Example for three classes:

$\mathcal{C}_1$	1 0 0
$\mathcal{C}_2$	0 1 0
$\mathcal{C}_3$	0 0 1

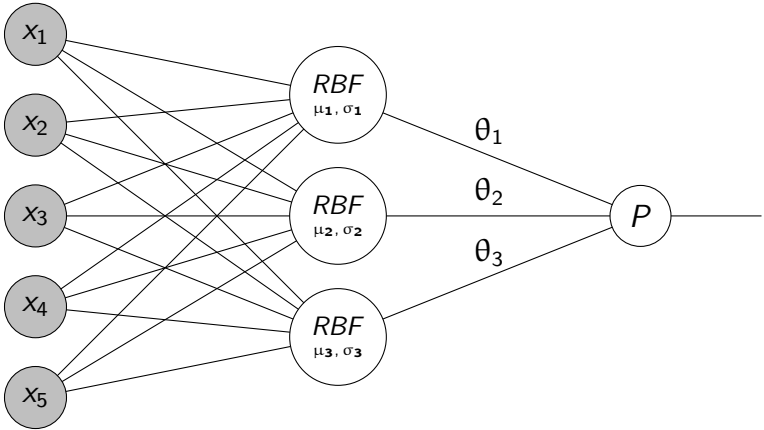


[4] This construct is still referred to as a perceptron.

What about non-linear Data Sets?

- If the data is not linearly separable then the perceptron cannot learn it
- Remember Marvin Minsky's/Seymour Papert's book *'Perceptrons'*
- What can we do?
  1. Add feature mapping $\Rightarrow$ **Radial basis function (RBF) networks**
  2. Add hidden layers $\Rightarrow$ **Multi-layer perceptrons (MLP)**

Radial Basis Function (RBF) Networks

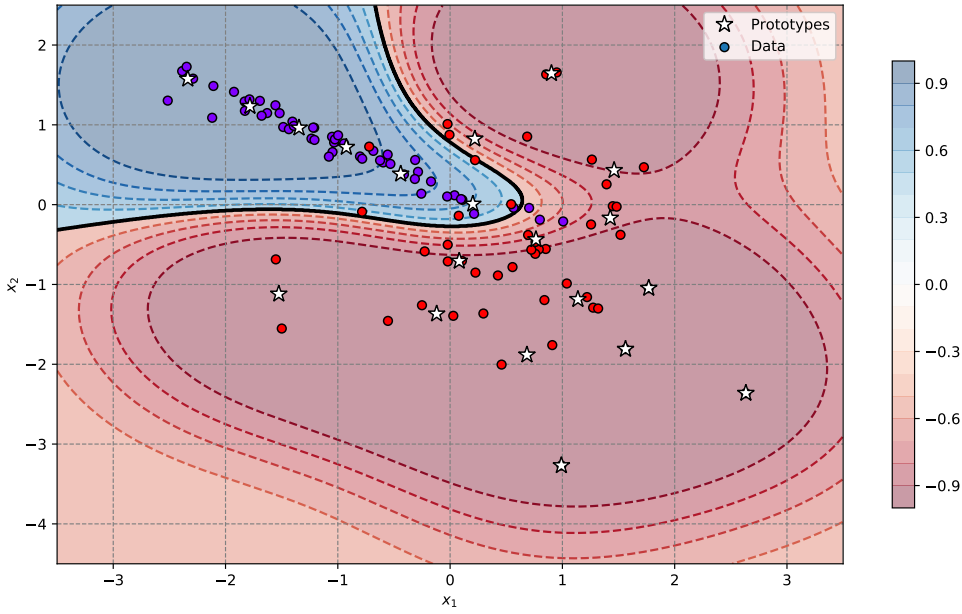


Inside an RBF Neuron

- Each RBF neuron computes the distance of the input $\boldsymbol{x}$ to an **internal prototype** vector:

$$\varphi_j(\boldsymbol{x}) = \exp\left\{-\frac{\|\boldsymbol{x} - \boldsymbol{z_j}\|^2}{2\sigma^2}\right\} \tag{4}$$

- These distances are then fed into the perceptron
- **How to get the prototypes?**
  - We can use clustering, e. g. **k-Means**, to get the locations $\boldsymbol{\mu}$ (this avoids useless prototypes in areas without data points)
  - Choose σ neither too small nor too big
- There are no weights $\boldsymbol{\theta}$ to tune for that neuron


Prototypes
Data
x_1
x_2
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9

Section:

Multi-Layer-Perceptrons (MLPs)

Backpropagation

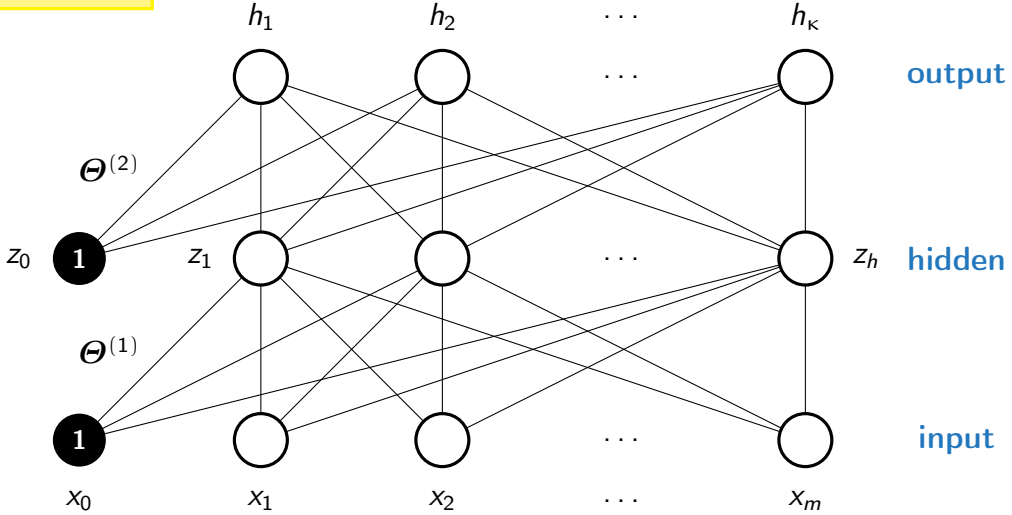
- In order to update the weights, we first have to perform a **forward pass**:

$$h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}) = g^{(2)}\left(\sum_{l=0}^{h} \Theta_{kl}^{(2)} g^{(1)}\left(\sum_{j=0}^{m} \Theta_{lj}^{(1)} x_j^{(i)}\right)\right)$$

$$z_l = g^{(1)}\left(\sum_{j=0}^{m} \Theta_{lj}^{(1)} x_j^{(i)}\right) \qquad \text{activation}$$

- $g(\cdot) \equiv$ activation function, e. g. sigmoid
- $\boldsymbol{\Theta}$ are the network parameters (to be learned)

This is a fully connected neural network



Backpropagation (Ctd.)

- Compute the network loss
- The loss function is given by: (assume square loss: $\ell = (h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}) - y_k^{(i)})^2$)

$$\mathcal{J}^{(i)}(\boldsymbol{\Theta}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{\kappa} \ell(h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}), y_k^{(i)})$$

- Compute the error gradient w. r. t. $h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta})$:

$$\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta})} = \ell'(h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}), y_k^{(i)}) \equiv \delta_k^{(i)}$$

Backpropagation (Ctd.)

- Compute the weight gradient for the output layer:

$$\begin{aligned}\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial \Theta_{kl}^{(2)}} &= \frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial h_k(\boldsymbol{x}^{(i)};\boldsymbol{\Theta})} \frac{\partial h_k(\boldsymbol{x}^{(i)};\boldsymbol{\Theta})}{\partial \Theta_{kl}^{(2)}} \\ &= \ell'(h_k(\boldsymbol{x}^{(i)};\boldsymbol{\Theta}), y_k^{(i)}) \cdot g'^{(2)} \left(\sum_{t=0}^{h} \Theta_{kt}^{(2)} z_t(\boldsymbol{x}^{(i)}) \right) \cdot z_l(\boldsymbol{x}^{(i)}) \\ &= \delta_k^{(i)} \cdot g'^{(2)} \left(\sum_{t=0}^{h} \Theta_{kt}^{(2)} z_t(\boldsymbol{x}^{(i)}) \right) \cdot z_l(\boldsymbol{x}^{(i)})\end{aligned}$$

Backpropagation (Ctd.)

- Compute the error gradient for the hidden layer:

$$\begin{aligned}\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_l} &= \sum_{k=1}^{K} \frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta})} \frac{\partial h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta})}{\partial z_l} \\ &= \sum_{k=1}^{K} \ell'(h_k(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}), y^{(i)}) \cdot g'^{(2)} \left(\sum_{t=0}^{h} \Theta_{kt}^{(2)} z_t(\boldsymbol{x}^{(i)}) \right) \cdot \Theta_{kl}^{(2)} \\ &= \sum_{k=1}^{K} \delta_k^{(i)} \cdot g'^{(2)} \left(\sum_{t=0}^{h} \Theta_{kt}^{(2)} z_t(\boldsymbol{x}^{(i)}) \right) \cdot \Theta_{kl}^{(2)} \equiv \widehat{\delta}_l^{(i)}\end{aligned}$$

Backpropagation (Ctd.)

- Compute the weight gradient for the hidden layer:

$$\frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial \Theta_{lj}^{(1)}} = \frac{\partial \mathcal{J}^{(i)}(\boldsymbol{\Theta})}{\partial z_l} \cdot g'^{(1)} \left(\sum_{t=0}^{m} \Theta_{jt}^{(1)} x_j^{(i)} \right) \cdot x_j^{(i)}$$

$$= \widehat{\delta}_l^{(i)} \cdot g'^{(1)} \left(\sum_{t=0}^{m} \Theta_{jt}^{(1)} x_j^{(i)} \right) \cdot x_j^{(i)}$$

- The weight derivatives are now used in the gradient descent update rule

Backpropagation Example

Section:

Wrap-Up

Summary

Self-Test Questions

1.

What's next...?

Unit I	Machine Learning Introduction
Unit II	Mathematical Foundations
Unit III	Bayesian Decision Theory
Unit IV	Probability Density Estimation
Unit V	Regression
Unit VI	Classification I
Unit VII	Evaluation
Unit VIII	**Classification II**
Unit IX	Clustering
Unit X	Dimensionality Reduction

Recommended Literature and further Reading

Thank you very much for the attention!

Topic: *** Applied Machine Learning Fundamentals *** Neural Networks / Deep Learning
Term: Winter term 2019/2020

Contact:
M. Sc. Daniel Wehner
SAP SE
daniel.wehner@sap.com

Do you have any questions?