

*** 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

① Introduction

- What is Deep Learning?
- History of Deep Learning
- Biological Motivation
- Perceptron Learning Algorithm
- Radial Basis Function (RBFN) Networks

② Multi-Layer-Perceptrons (MLPs)

- Backpropagation

③ Further Network Architectures

- Convolutional Neural Networks

④ Wrap-Up

- Summary
- Self-Test Questions
- Lecture Outlook
- Recommended Literature and further Reading

Section: Introduction



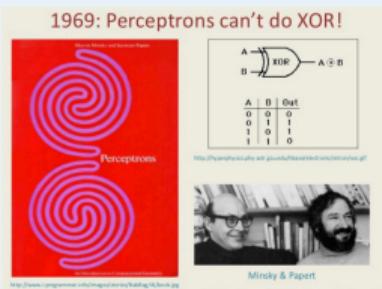
What is Deep Learning?

- ‘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

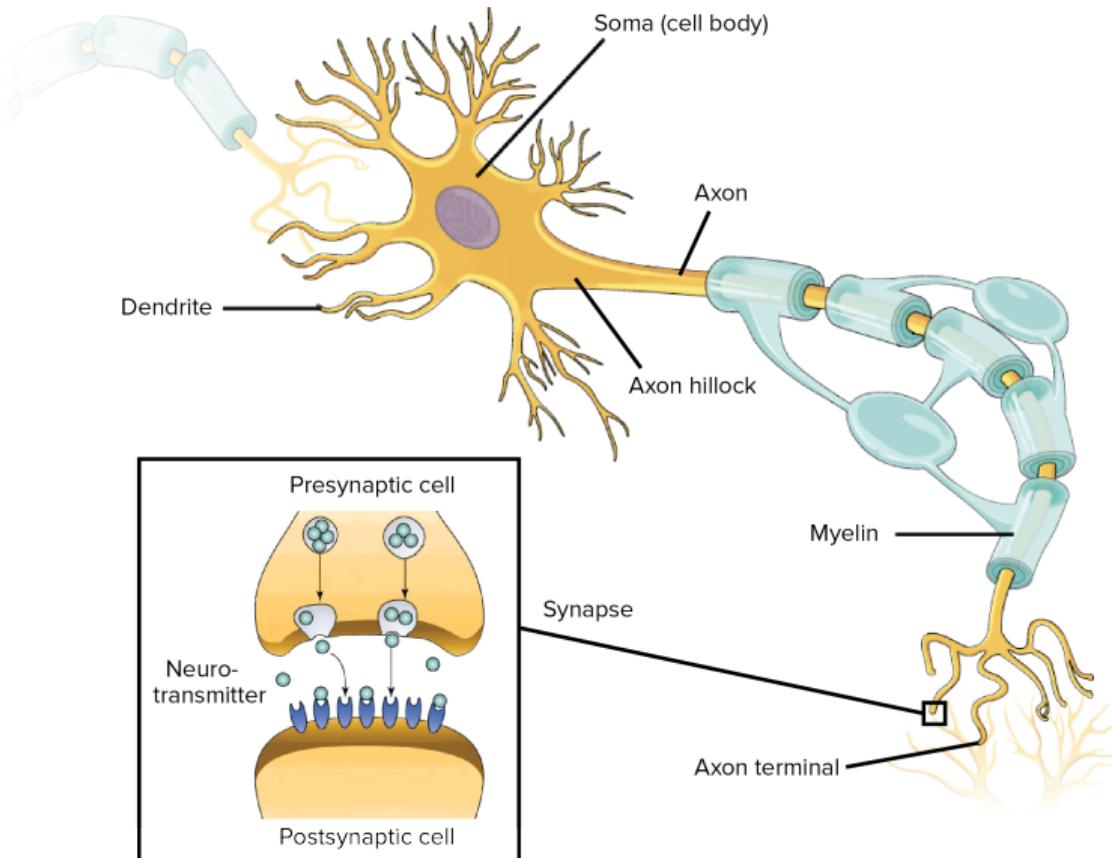
'Deep Learning' (since mid 2000)

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

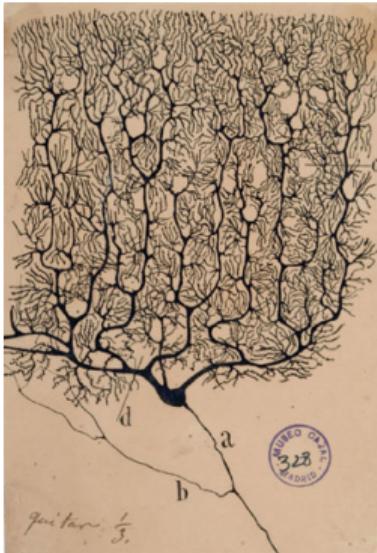


Biological Motivation

- 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



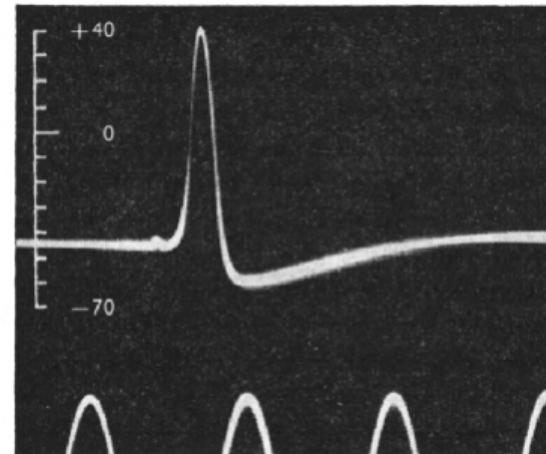
How can we know this?



- *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's¹ axon
- The right-hand-side image was the first **action potential** that was plotted



¹lat.: *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

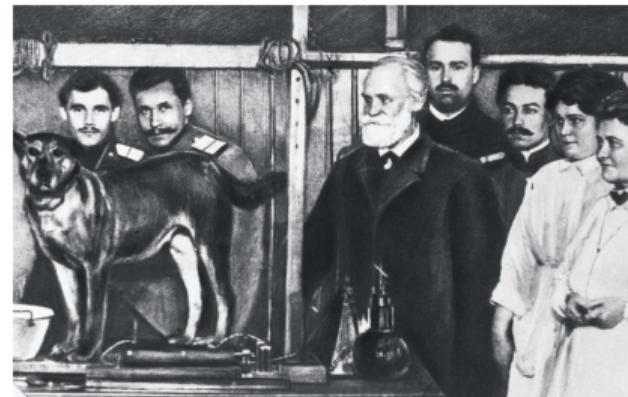
'When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.'

'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.)

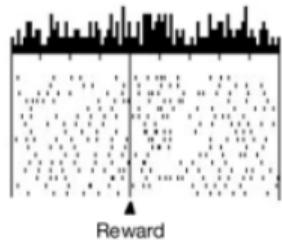


Blocking

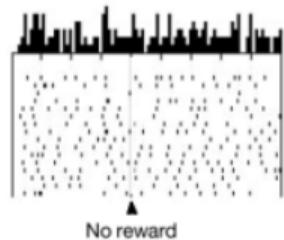
Group A	train N+	train LN+	test L-	⇒ no conditioning
Group B		train LN+	test L-	⇒ conditioning

- 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**

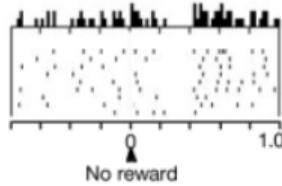
a
A+ Predicted reward
(no prediction error)



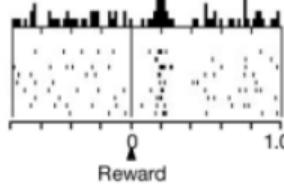
B- Predicted no reward
(no prediction error)



A- Unpredicted no reward
(negative prediction error)

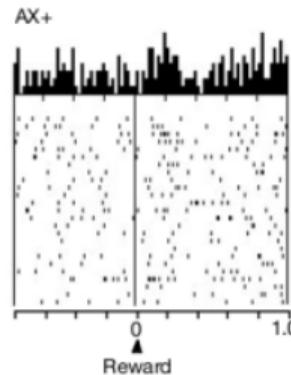


B+ Unpredicted reward
(positive prediction error)

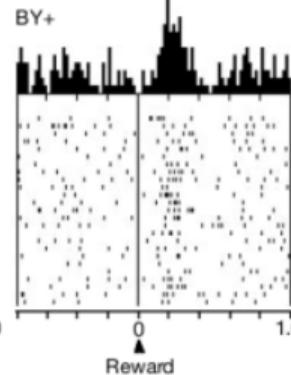


b

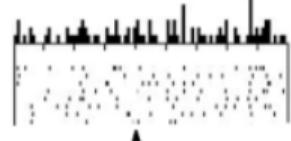
AX+



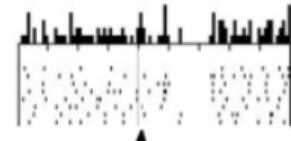
BY+



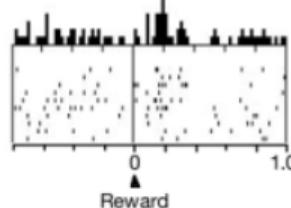
X- No reward predicted
(no prediction error)



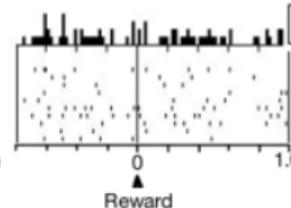
Y- Unpredicted no reward
(negative prediction error)



X+ Unpredicted reward
(positive prediction error)

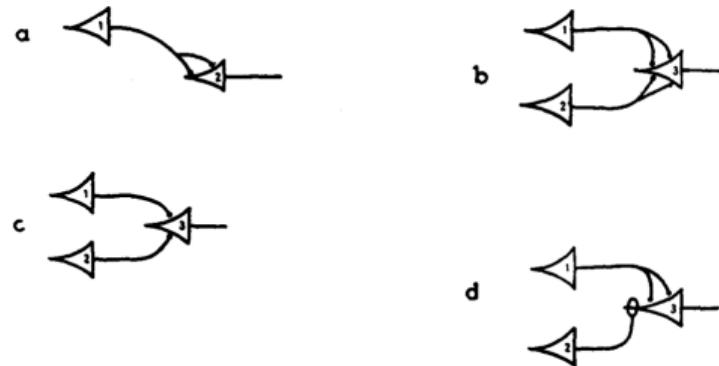


Y+ Predicted reward
(no prediction error)

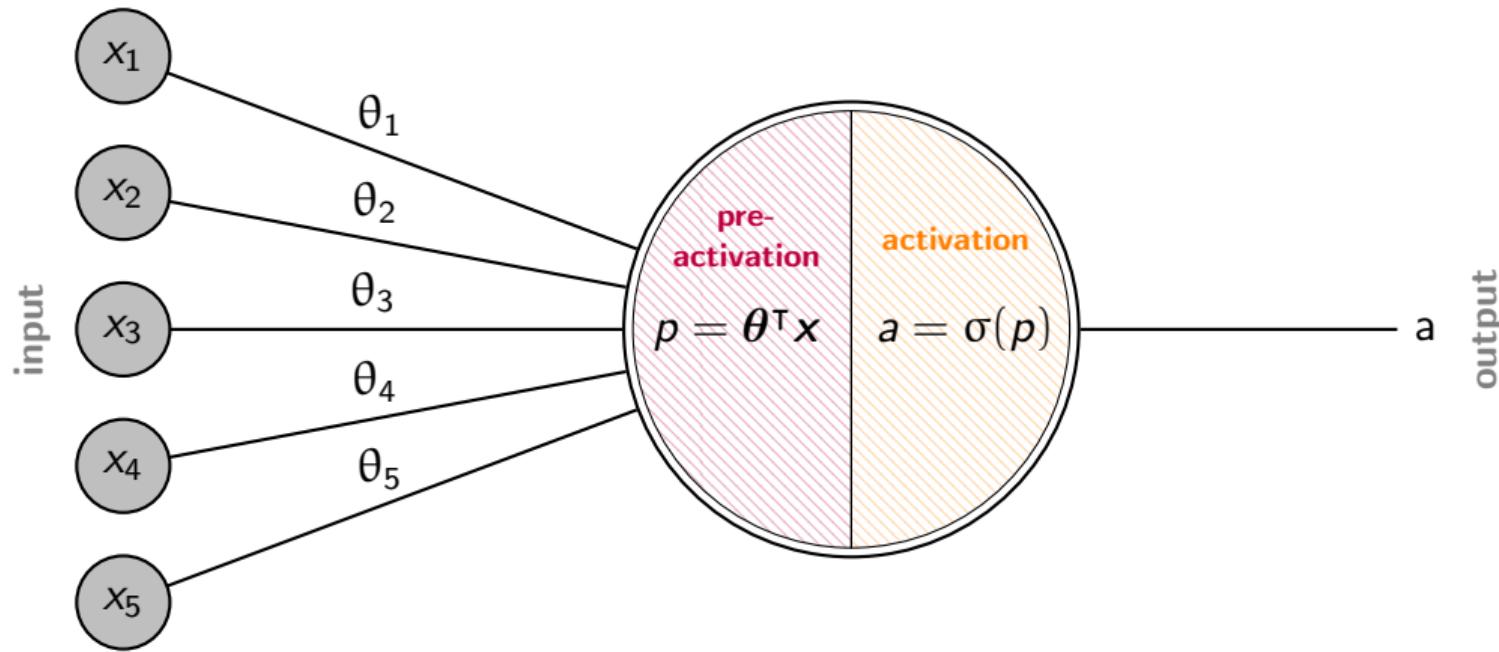


Artificial Neurons [McCulloch and Pitts 1943]

- 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 x

$$x = (x_1, x_2, \dots, x_m)^\top$$

- Each input signal is weighted by a factor² θ_j

$$\theta = (\theta_1, \theta_2, \dots, \theta_m)^\top$$

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

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

²weight of synaptic strength

Perceptron (Ctd.)

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

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

- Quick example: $x = (1, 0, 0.5)^T$; $\theta = (1, -0.5, -1)^T$; $\rho = 0$

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

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

³Not used, since not differentiable; alternatives later

Perceptron Learning

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

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

- How to learn the weights from a data set \mathcal{D} ?
- **Algorithm outline:**
 - ① Pick a training example $x \in \mathcal{D}$
 - 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 $\mathcal{J}(\theta)$:

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

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

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \nabla_{\theta} \mathcal{J}(\theta) \quad (3)$$

⇒ cf. slides 'Regression'

Algorithm 1: Perceptron Learning Algorithm

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

```
// initialization
1 set all weights  $\theta^{(0)}$  to small random numbers
2 for  $t \in \{0, 1, \dots, \infty\}$  do
3     pick a sample  $\langle \mathbf{x}, y \rangle \in \mathcal{D}$  randomly
        // predict the class label
4     compute the activation  $a = \sigma(\theta^\top \mathbf{x})$ 
        // stochastic gradient descent: update based on prediction error
5      $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \nabla_{\theta} J(\theta)$ 
6     if  $\|\theta^{(t+1)} - \theta^{(t)}\| \leq \varepsilon$  then
        // convergence
7         break
8 return  $\theta$ 
```

Perceptron Convergence Theorem

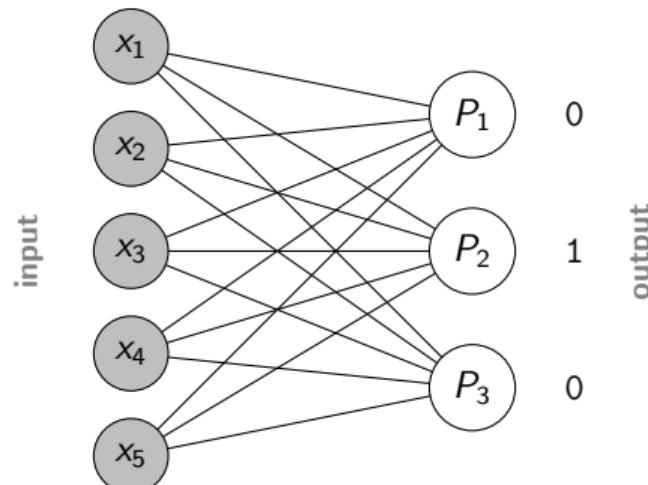
Perceptron Convergence Theorem

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 perceptrons⁴
- Use **one-hot encoding** for the classes and **soft-max** as activation function (later)
- Example for three classes:

C_1	1	0	0
C_2	0	1	0
C_3	0	0	1

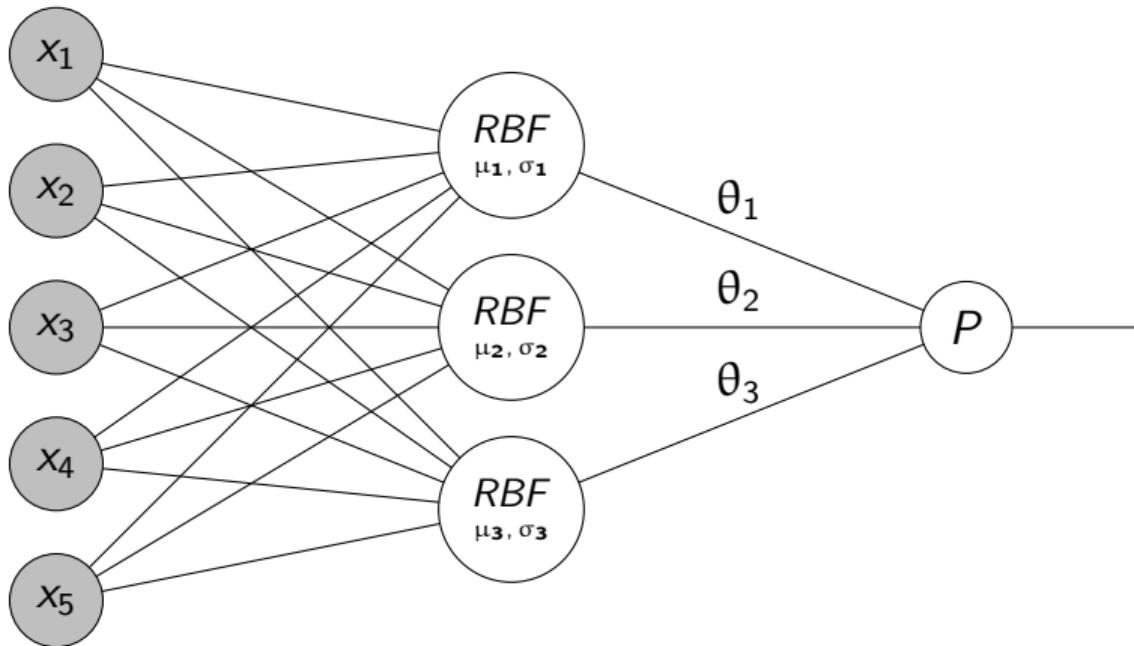


⁴This construct is still referred to as a perceptron.

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 ⇒ **Radial basis function (RBF) networks**
 - ② Add hidden layers ⇒ **Multi-layer perceptrons (MLP)**

Radial Basis Function (RBF) Networks

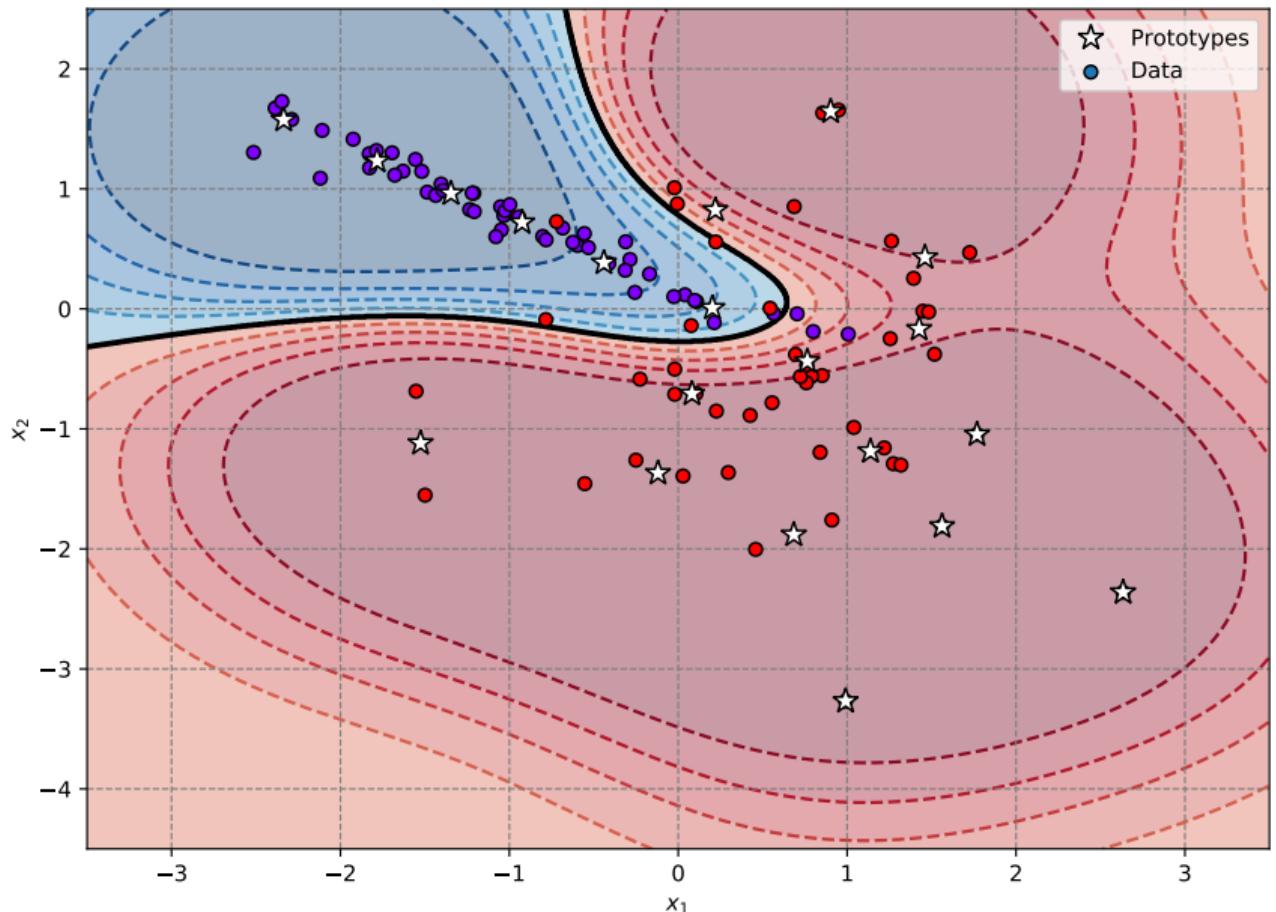


Inside an RBF Neuron

- Each RBF neuron computes the distance of the input x to an **internal prototype** vector:

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

- 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 μ (this avoids useless prototypes in areas without data points)
 - Choose σ neither too small nor too big
- There are no weights θ to tune for that neuron



Section:
Multi-Layer-Perceptrons (MLPs)



Backpropagation

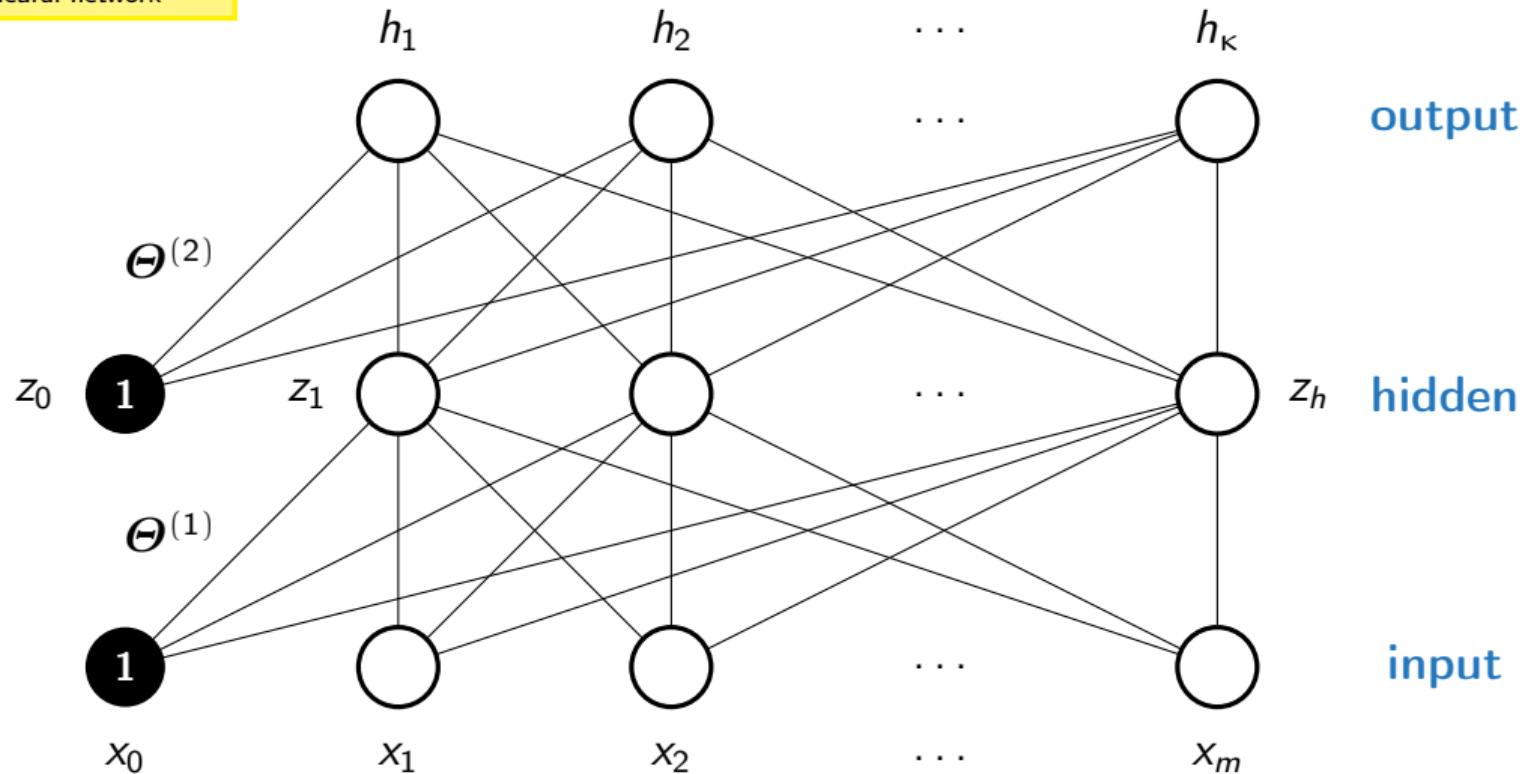
- In order to update the weights, we first have to perform a **forward pass**:

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

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

- $g(\cdot)$ \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: $\ell = (h_k(\mathbf{x}^{(i)}; \boldsymbol{\Theta}) - y_k^{(i)})^2$)

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

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

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

Backpropagation (Ctd.)

- Compute the weight gradient for the output layer:

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

Backpropagation (Ctd.)

- Compute the error gradient for the hidden layer:

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

Backpropagation (Ctd.)

- Compute the weight gradient for the hidden layer:

$$\begin{aligned}\frac{\partial \mathcal{J}^{(i)}(\Theta)}{\partial \Theta_{lj}^{(1)}} &= \frac{\partial \mathcal{J}^{(i)}(\Theta)}{\partial z_l} \cdot g'^{(1)}\left(\sum_{t=0}^m \Theta_{jt}^{(1)} x_j^{(i)}\right) \cdot x_j^{(i)} \\ &= \hat{\delta}_l^{(i)} \cdot g'^{(1)}\left(\sum_{t=0}^m \Theta_{jt}^{(1)} x_j^{(i)}\right) \cdot x_j^{(i)}\end{aligned}$$

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

Backpropagation Example



Section:
Further Network Architectures



Convolution Animation

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



[2] Deep Learning

Ian Goodfellow et al. MIT Press. 2016.

→ [Link](#), cf. chapters 6 *Deep Feedforward Networks*, especially chapter 6.5



[2] Pattern Recognition and Machine Learning

Christopher Bishop. Springer. 2006.

→ [Link](#), cf. chapter 5 *Neural Networks*, especially chapter 5.3

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?