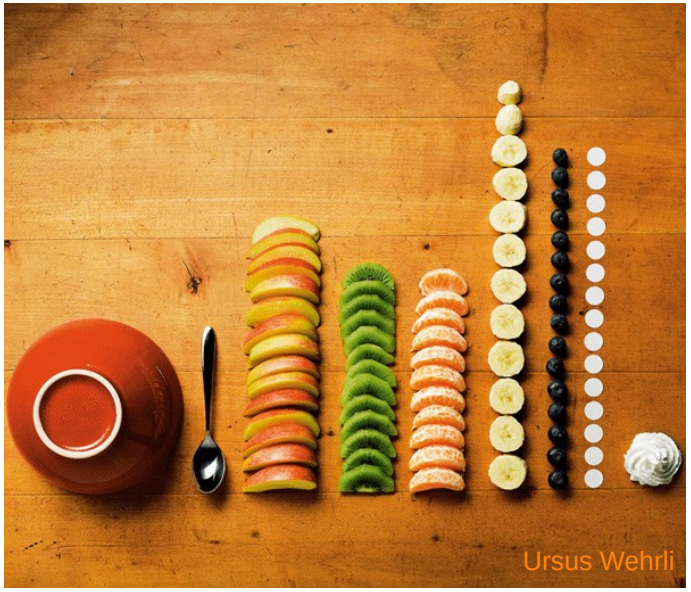


Grundlagen der Künstlichen Intelligenz

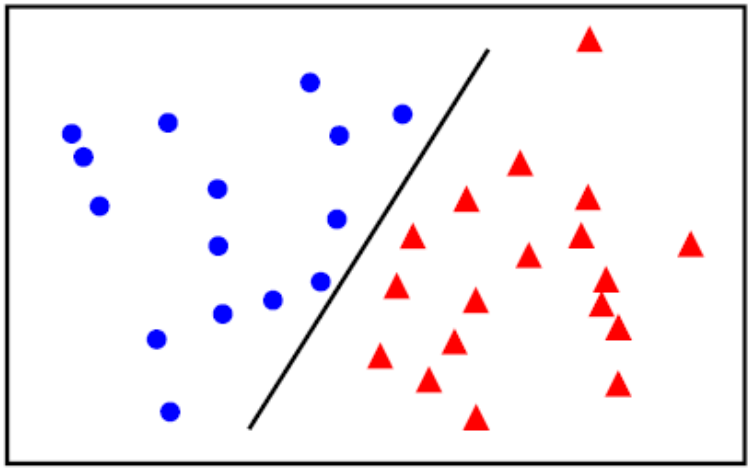
Wintersemester 25/26

Vorlesung 3

Maschinelles Lernen I:
Lernende Algorithmen,
Klassifikation & Logistische Regression

Prof. Dr. Pascal Friederich
T.T.-Prof. Dr. Peer Nowack

Teil 3

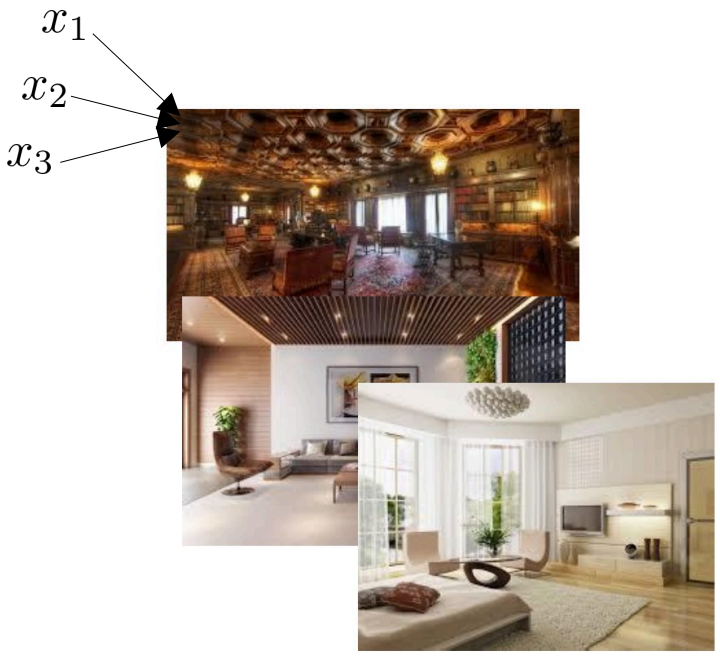


1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

Beispiel: Klassifizierung von Bildern



Innen
c

Außen
c

(Bilddaten: Vorlesung 9)

Beispiel: Spam Erkennung

	x_1	x_2	x_3		c
	#"$"	#"Mr."	#"sale"	...	Spam?
Email 1	2	1	1		Yes
Email 2	0	1	0		No
Email 3	1	1	1		Yes
...					
Email n	0	0	0		No
New email	0	0	1		??

Klassifikation: Formale Definition

Gegeben: Datensatz $\mathcal{D} = \{(\boldsymbol{x}_i, c_i)\}_{i=1\ldots N}$

mit Input Samples $\boldsymbol{x}_i \in \mathbb{R}^n$ und Klassen $c \in \{1 \ldots K\}$

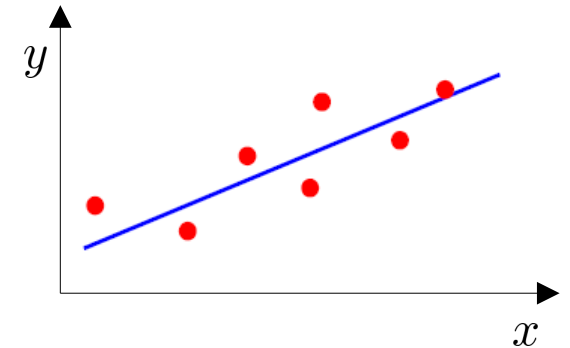
Ziel: Lernen einer Funktion $c = f(\boldsymbol{x})$

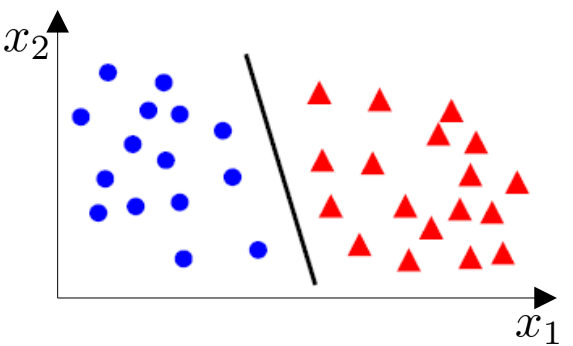
Nomenklatur: $K = 2$: Binäre Klassifikation

$K > 2$: Multiklassen Klassifikation

Vergleich: Regression: Kontinuierlicher Output

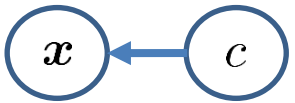
Klassifikation: Diskreter Output

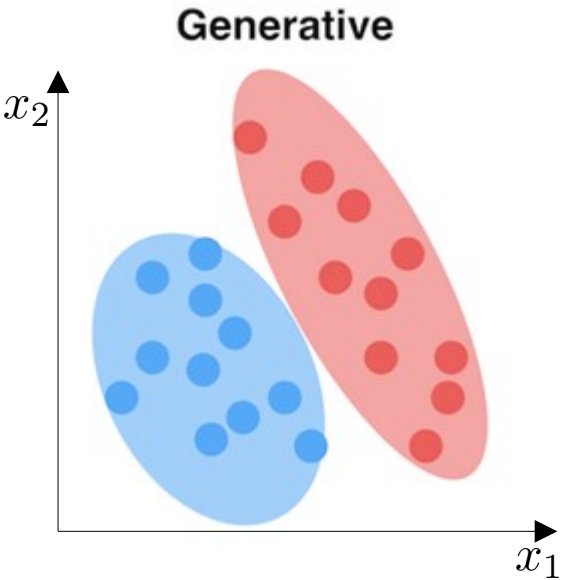




Generative vs. diskriminative Modelle

Generative Modelle

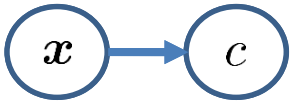




- Annahme: $p(c)$
- Lernen eines Modells $p(\boldsymbol{x} \mid c)$ aus den Daten
- Damit: Erzeugung von neuen Datenpunkten mit gegebenen Klassen
- Vorhersage der Klassen: $p(c \mid \boldsymbol{x}) = \dfrac{p(\boldsymbol{x} \mid c)p(c)}{p(\boldsymbol{x})}$
- Das Lernen dieser Wahrscheinlichkeitsverteilungen ist sehr schwer!

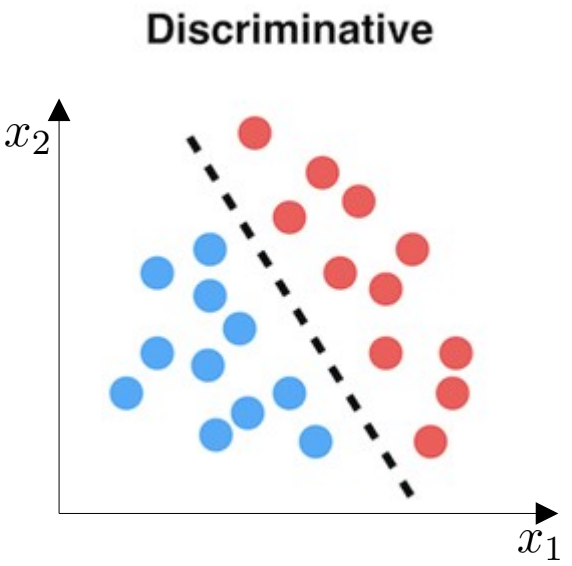
Generative vs. diskriminative Modelle

Diskriminative Modelle

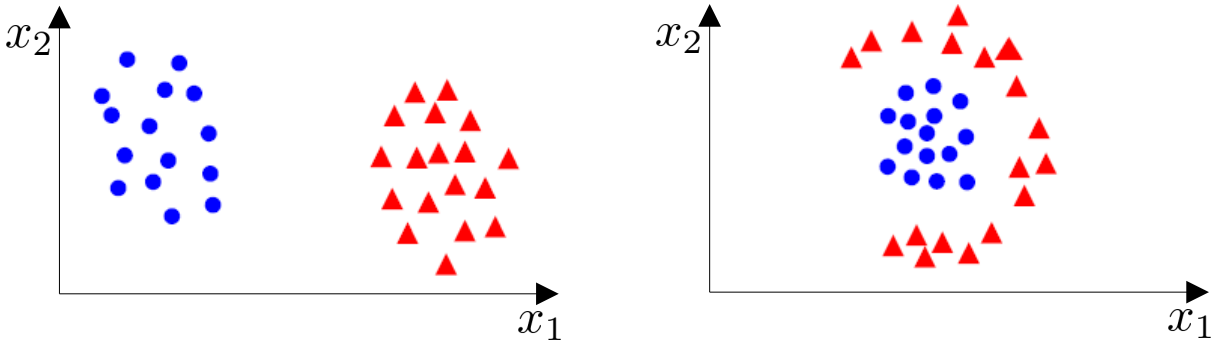


- Lernen einer Funktion $f(\boldsymbol{x})$ oder einer $p(c \mid \boldsymbol{x})$ Wahrscheinlichkeitsverteilung
- Die Funktion kann verschiedene $\boldsymbol{x}$ unterscheiden
- Modellierung muss die Punkte an der Grenze berücksichtigen
- Normalerweise sehr viel einfacher als Generation

  → Wir fokussieren uns heute auf diskriminative Modelle



Binäre Klassifikation



Diskriminative Modelle

- Daten: $(\boldsymbol{x}_i, c_i)\,, \mathrm{i} = 1 \ldots \mathrm{m}$ mit $\boldsymbol{x}_i \in \mathbb{R}^n$ und $c_i \in \{0, 1\}$
- Lerne einen Klassifikator:

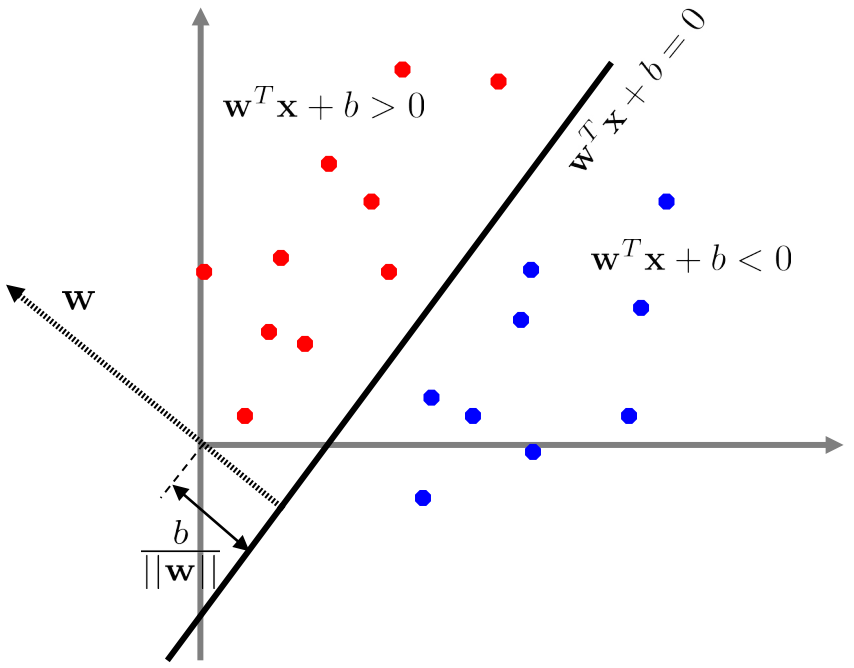
$$f\left(\boldsymbol{x}_i\right) = \begin{cases} > 0, & \text{if } c_i = 1 \\ < 0, & \text{if } c_i = 0 \end{cases}$$

Lineare Klassifikation

Linearer Klassifikator

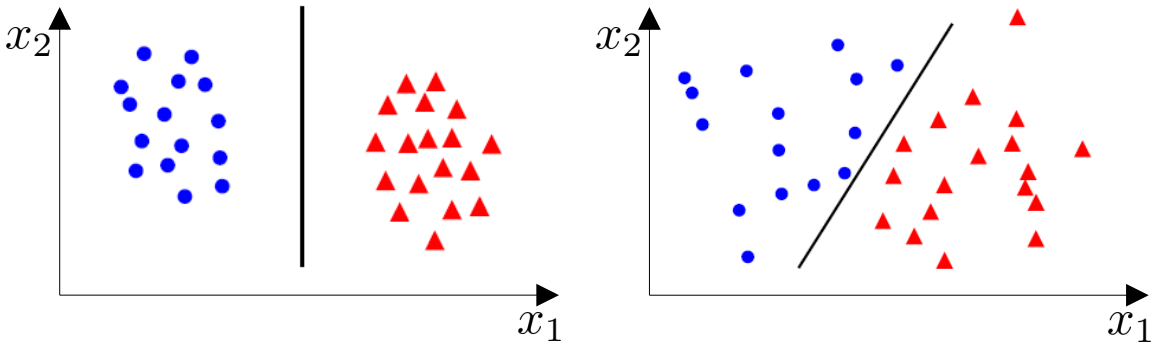
$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

- **In 2D: Linie**
  - $\mathbf{w}$ ist orthogonal zur Linie
  - b ist der y-Achsen-Abschnitt („bias“)
- **In 3D: Ebene**
- **In nD: Hyperebene**

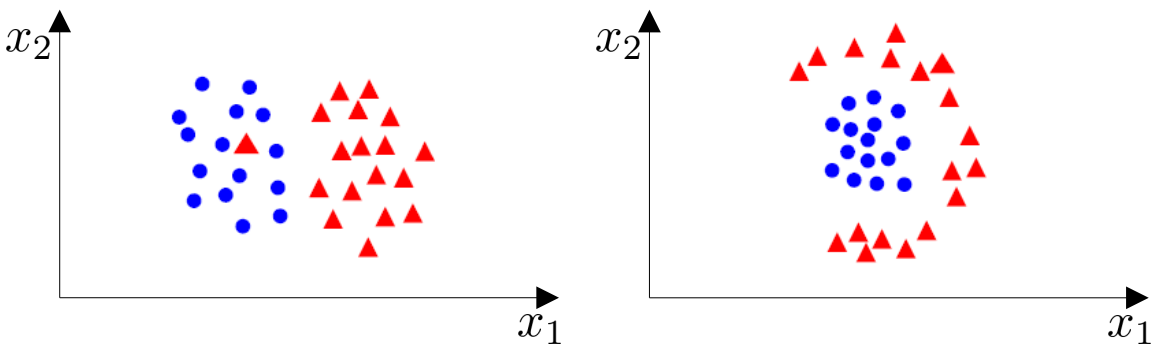


Lineare Separararabilität

Linear separierbar



Nicht linear separierbar



Lineare Klassifizierung: Erster Versuch

Vorhersage: $y = \text{step}(f(\boldsymbol{x})) = \text{step}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$

- Falls $f(\boldsymbol{x}) < 0$: Klasse 0
- Falls $f(\boldsymbol{x}) \geq 0$: Klasse 1

Optimierung

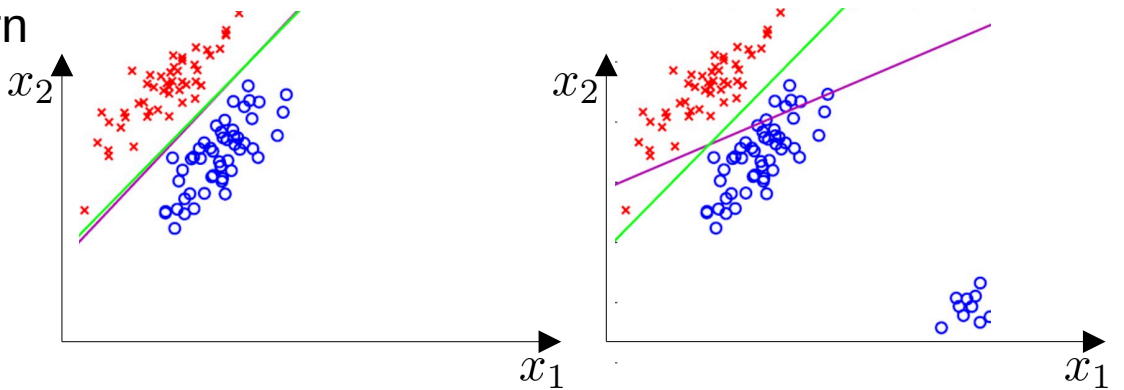
- Finde $\boldsymbol{w}$, sodass die Zahl der falsch klassifizierten Punkte minimal ist
- Quantifizierung: $L_0(\boldsymbol{w}) = \sum \mathbb{I}\left(\text{step}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right) \neq c_i\right)$
- Sehr schwer zu optimieren! NP-schwer!

Lineare Klassifizierung: Zweiter Versuch

Können wir die gleiche Loss-Funktion wie für Regression benutzen?

$$L_{\text{reg}}(\boldsymbol{w}) = \sum_i \left(f\left(\boldsymbol{x}_i\right) - c_i\right)^2$$

- Minimierung der Loss-Funktion: Einfach!
- Aber: Die Labels sind nur 0 oder 1, der Output von $f\left(\boldsymbol{x}\right)$ ist kontinuierlich
- → Nicht robust gegenüber Ausreißern



Lineare Klassifizierung: Dritter Versuch

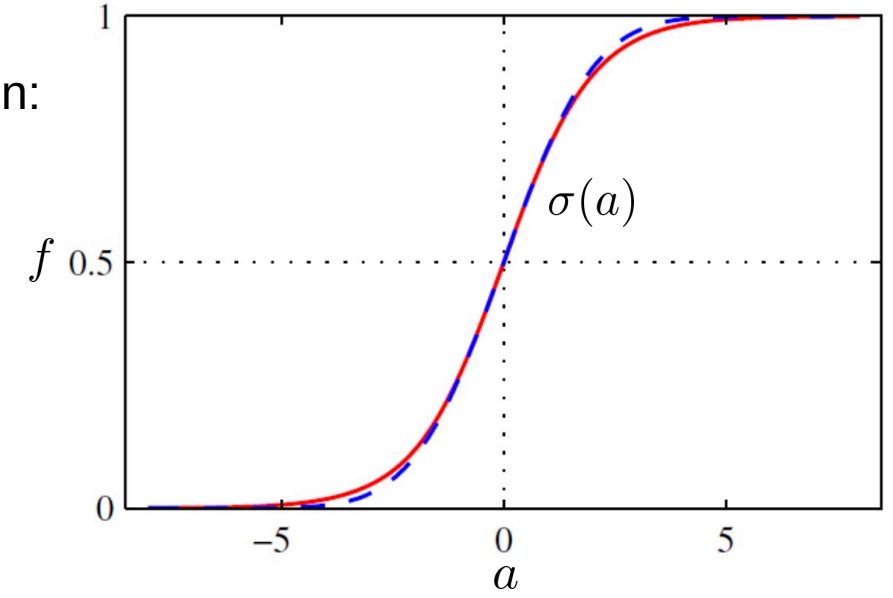
Können wir $f(x)$ einschränken?

- Benutze Sigmoid Funktion / logistische Funktion:

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

→ Output ist zwischen 0 und 1

→ Keine Step-Funktion, sondern glatt



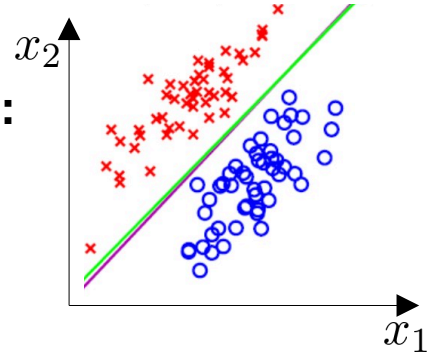
Benutzung für Klassifikation

- Wende Sigmoid auf lineare Funktion an

$$L(\boldsymbol{w}) = \sum_i \left(\sigma\left(f\left(\boldsymbol{x}_i\right)\right) - c_i\right)^2 = \sum_i \left(\sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) - c_i\right)^2$$

Besser: Probabilistische Betrachtung

Interpretation von $\sigma\left(f\left(\boldsymbol{x}\right)\right)$ als Wahrscheinlichkeitsfunktion:



$$p(c = 1 \mid \boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$$

$$p(c = 0 \mid \boldsymbol{x}) = 1 - \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$$

Trick um dies in einer Formel zusammenzufassen: Label/Klasse *c* im Exponent

$$p(c \mid \boldsymbol{x}) = p(c = 1 \mid \boldsymbol{x})^c p(c = 0 \mid \boldsymbol{x})^{1-c} = \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)^c \left(1 - \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)\right)^{1-c}$$

(„Exponential-Trick“)

Log-Likelihood

Maximierung der Wahrscheinlichkeit → Maximierung des Logarithmus

$$\sum_i p\left(c_i \mid \boldsymbol{x}_i\right) \rightarrow \sum_i \log p\left(c_i \mid \boldsymbol{x}_i\right)$$

Wir können direkt die log-likelihood optimieren (maximieren)

$$\log \operatorname{lik}(\tilde{\boldsymbol{w}}, D) = \sum_i \log p\left(c_i \mid \boldsymbol{x}_i\right) = \sum_i \log \left(p\left(c=1 \mid \boldsymbol{x}_i\right)^{c_i} p\left(c=0 \mid \boldsymbol{x}_i\right)^{1-c_i} \right)$$

$$= \sum_i c_i \log p\left(c=1 \mid \boldsymbol{x}_i\right) + \left(1-c_i\right) \log p\left(c=0 \mid \boldsymbol{x}_i\right)$$

$$\tilde{\boldsymbol{x}}_i = [1, x_{i,0}, x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$$

$$\tilde{\boldsymbol{w}}_i = [b, w_0, w_1, w_2, \ldots, w_n]$$

$$= \sum_i c_i \log \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right) + \left(1-c_i\right) \log \left(1-\sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right)\right)$$

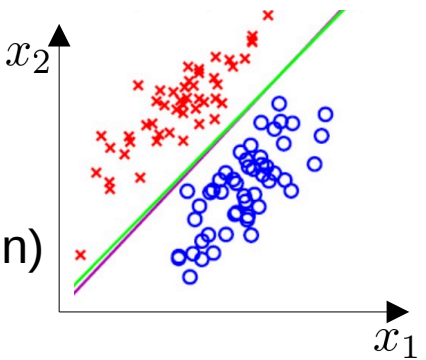
→ Negative log-likelihood wird oft als „**cross-entropy loss**“ bezeichnet

Logistische Regression

Optimierung der log-likelihood der Sigmoid-Funktion: „logistische Regression“

$$\text{argmax}_{\tilde{\boldsymbol{w}}} \log \text{lik}(\tilde{\boldsymbol{w}}, D) = \text{argmax}_{\tilde{\boldsymbol{w}}} \sum_i c_i \log \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right) + (1 - c_i) \log\left(1 - \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right)\right)$$

- … obwohl wir es für Klassifikation (nicht Regression) benutzen
- Die Loss-Funktion ist konvex → Nur ein Optimum
- Es gibt keine geschlossene Lösung (anders als bei linearer Regression)



Wie finden wir das Maximum? → **Gradient descent** (Gradientenabstiegsverfahren)

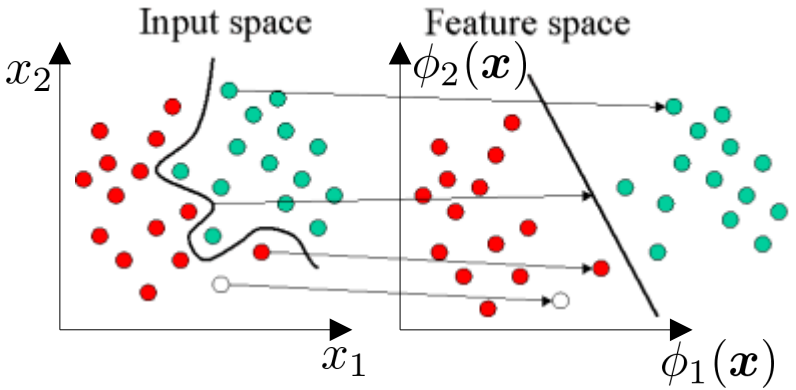
→ **Später!**

Verallgemeinerte logistische Regression

Linearer Klassifikator mit nicht-linearen Features $\boldsymbol{x}_i \rightarrow \phi(\boldsymbol{x}_i)$

$$\text{argmax}_{\boldsymbol{w}} \log \text{lik}(\boldsymbol{w}, D) = \text{argmax}_{\boldsymbol{w}} \sum_i c_i \log \sigma\left(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)\right) + (1 - c_i) \log\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)\right)\right)$$

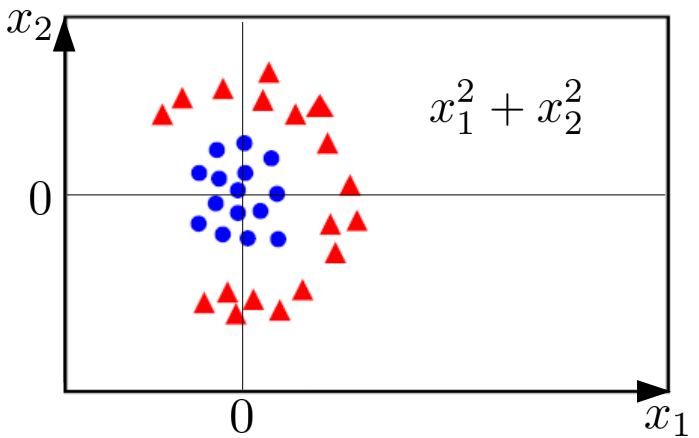
Ziel: Transformation der Punkte im Input-Raum (nicht linear separabel) zu Punkten im Feature-Raum die linear separabel sind



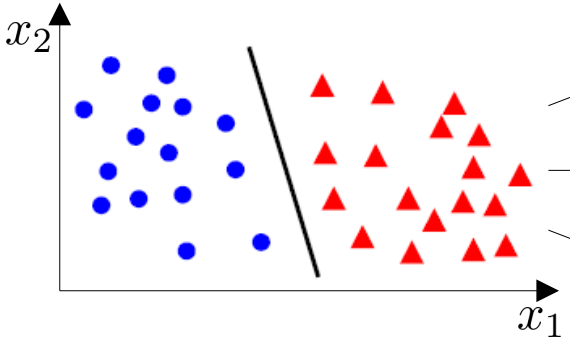
Verallgemeinerte logistische Regression

Ziel: Transformation der Punkte im Input-Raum (nicht linear separabel) zu Punkten im Feature-Raum die linear separabel sind

Beispiel



Zusammenfassung: Logistische Regression



Lineare Separierbarkeit

Sigmoidfunktion und logistische Regression

Log Likelihood Optimierung, Cross Entropy Loss

Zusammenfassung

1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

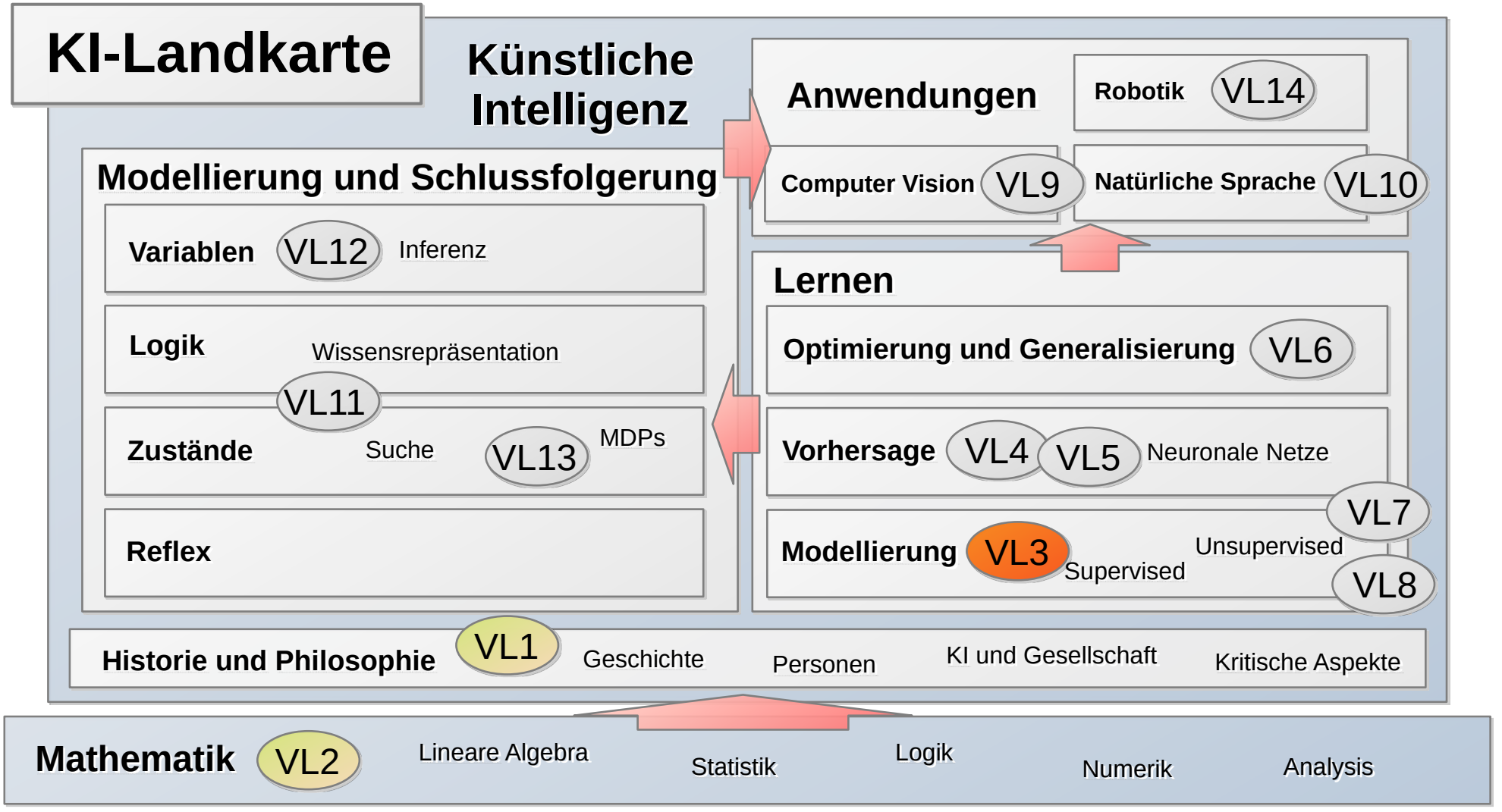
Grundlagen der Künstlichen Intelligenz

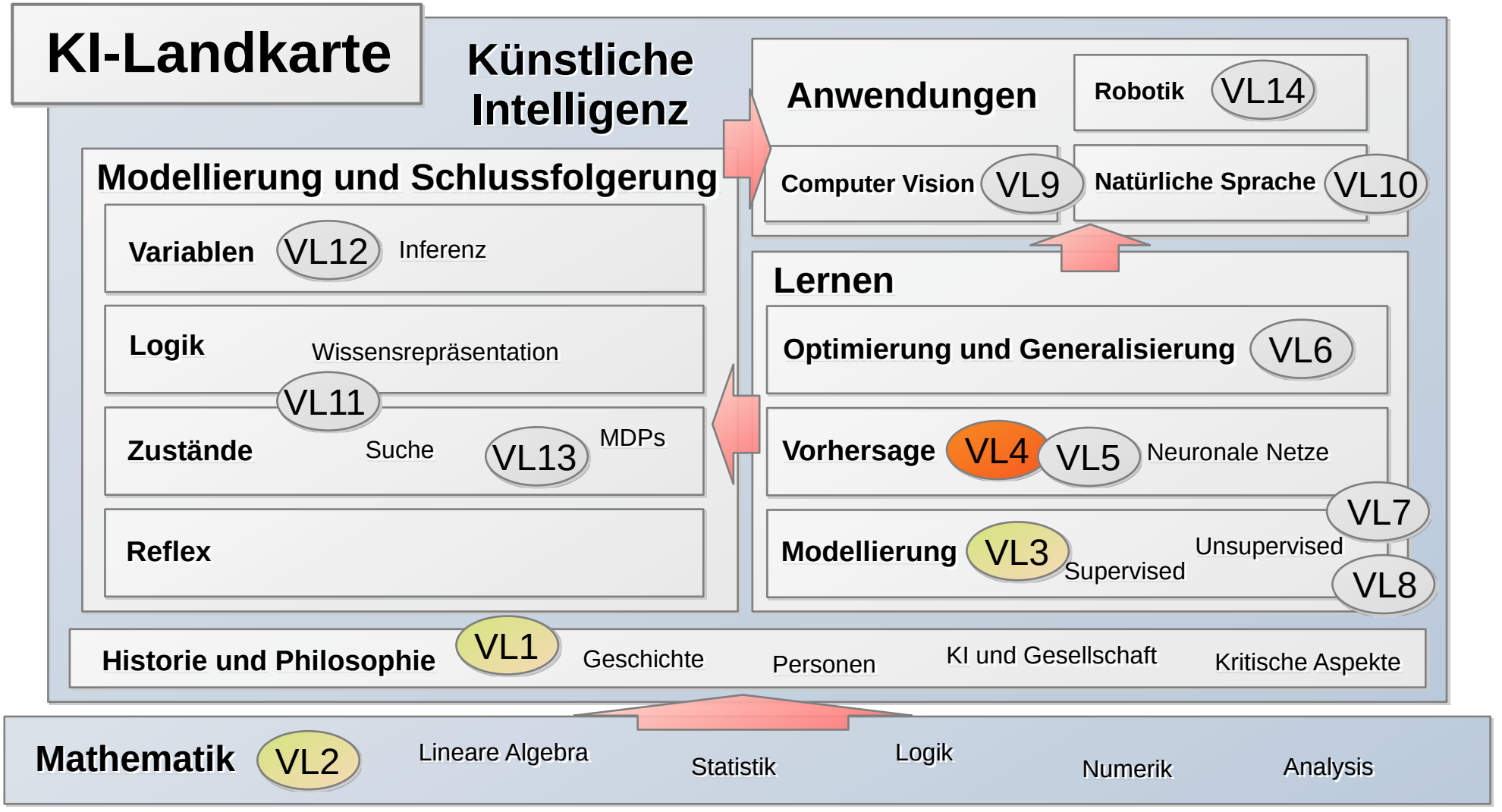
Wintersemester 25/26

Vorlesung 4

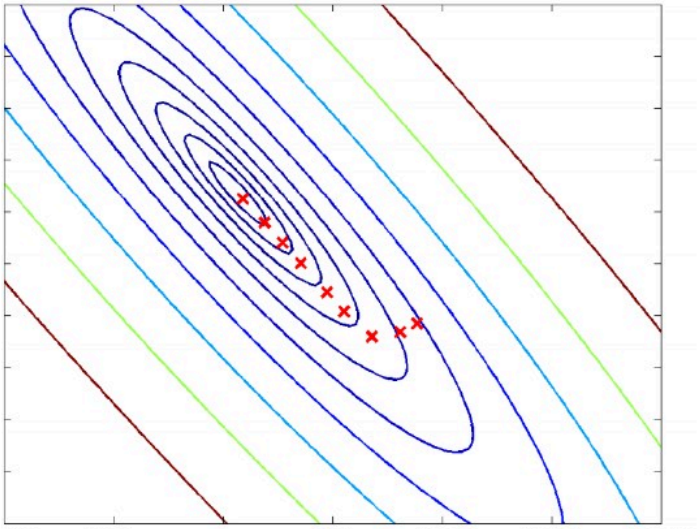
Neuronale Netze, Backpropagation

Prof. Dr. Pascal Friederich
T.T.-Prof. Dr. Peer Nowack


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL10
Modellierung und Schlussfolgerung
Variablen
VL12
Inferenz
Logik
Wissensrepräsentation
VL11
Zustände
Suche
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL6
Vorhersage
VL4
VL5
Neuronale Netze
VL7
Modellierung
VL3
Supervised
Unsupervised
VL8
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL10
Modellierung und Schlussfolgerung
Variablen
VL12
Inferenz
Logik
Wissensrepräsentation
VL11
Zustände
Suche
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL6
Vorhersage
VL4
VL5
Neuronale Netze
VL7
Modellierung
VL3
Supervised
Unsupervised
VL8
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis

Teil 1



1) Optimierung: Gradient descent

2) Neuronale Netze: Motivation

3) Neuronale Netze: Details

4) Neuronale Netze: Back-propagation

Optimierung

Für viele Machine Learning Methoden:

→ Finde das Modell (die Parameter), das die Daten am besten beschreibt

Beispiele

- Mean-Squared-Error Loss $\operatorname{argmin}_{\boldsymbol{w}} \operatorname{MSE}(\boldsymbol{w}, D)$
- Maximum-Likelihood: $\operatorname{argmax}_{\boldsymbol{w}} \operatorname{loglik}(\boldsymbol{w}, D)$

Optimierung: Allgemeine Form

Loss-Funktion für jeden Datenpunkt plus Bestrafung/Penalty (Regularisierung)

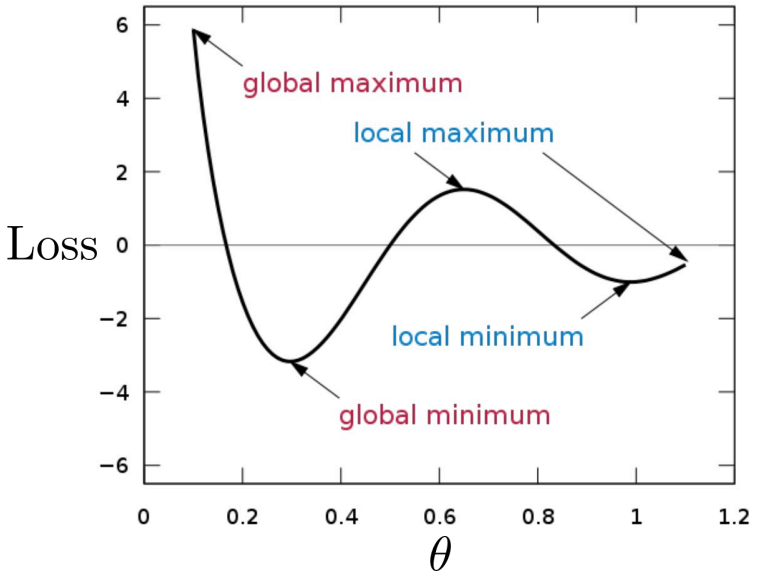
$$\underset{\text{parameters } \boldsymbol{\theta}}{\arg\min} \left(\sum_{i=1}^{N} l\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right) + \lambda \, \text{penalty}(\boldsymbol{\theta}) \right)$$

Wie finden wir die optimalen Parameter θ ?

→ (numerische) **Optimierung!**

Globale und lokale Optima

Ziel der Optimierung



- Für **konvexe Funktionen**: Globales Optimum kann gefunden werden
- Für **nicht-konvexe Funktionen**: Im Allgemeinen limitiert auf lokale Optima

Gradient descent (Gradientenabstiegsverfahren)

- Konvexe Funktionen: Suche nach **globalem Minimum**
- Nicht-konvexe Funktionen: Suche nach **lokalen Minima**
- Viele Anwendungen im **maschinellen Lernen**:
  - Lineare Regression (für große Inputdimensionen)
  - Logistische Regression
  - Neuronale Netze
  - ...

Gradient descent: Algorithmus

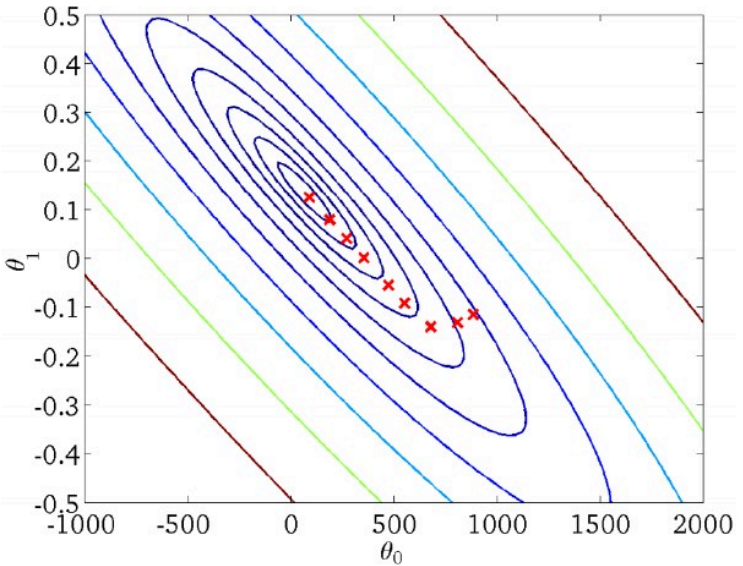
- Beginne mit (zufälligem) Punkt
- Folge dem Gradienten

$$\boldsymbol{\theta}_0 \leftarrow \text{init}, t = 0$$

$$\textbf{while } \text{not converged } \textbf{do}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_\theta L\left(\boldsymbol{\theta}_t\right), \quad t = t + 1$$

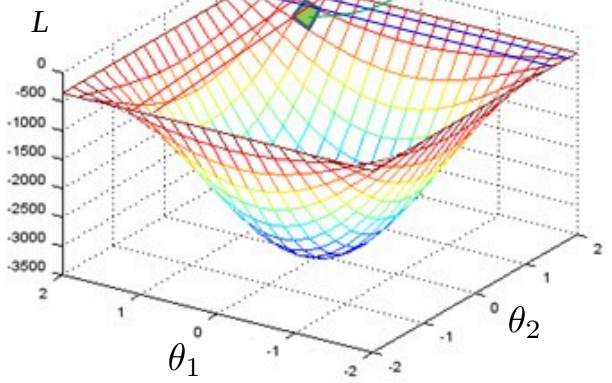
$$\textbf{end while}$$

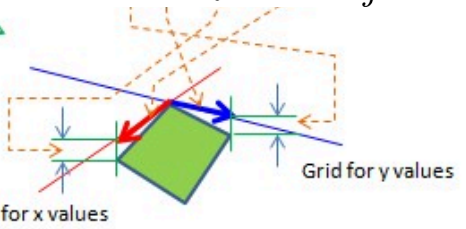


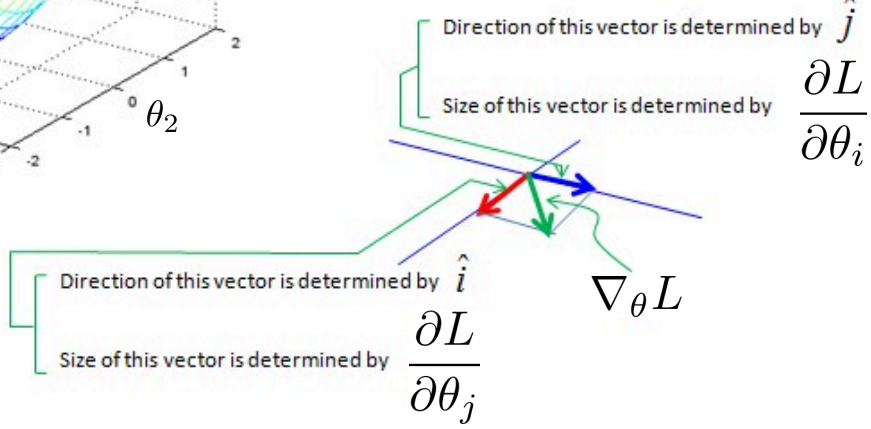
- η : Schrittweite / Lernrate
- Gradient zeigt immer in die Richtung des steilsten Anstiegs

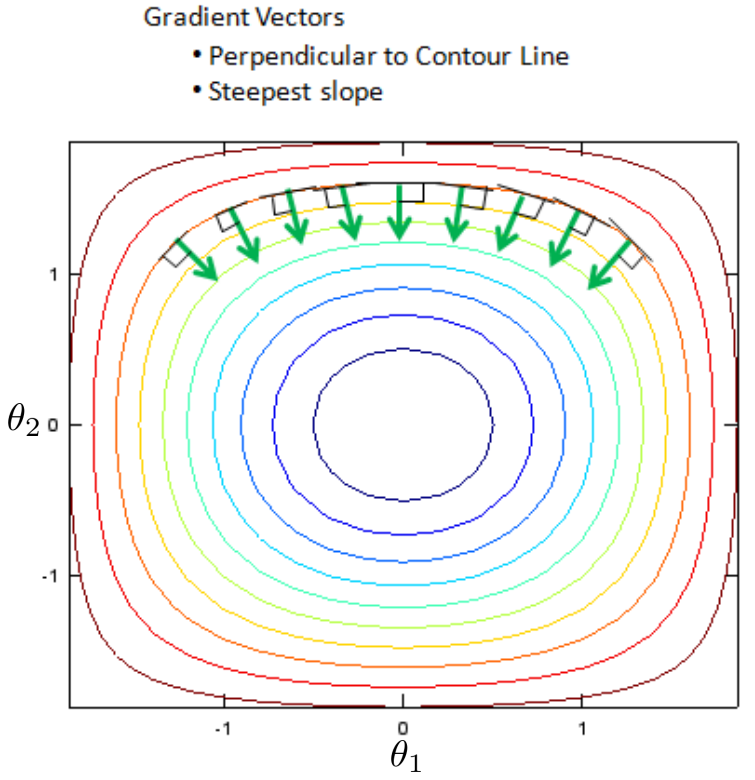
Visualisierung Gradient

$$\nabla_\theta L = \frac{\partial L}{\partial \theta_i}\hat{i} + \frac{\partial L}{\partial \theta_j}\hat{j}$$





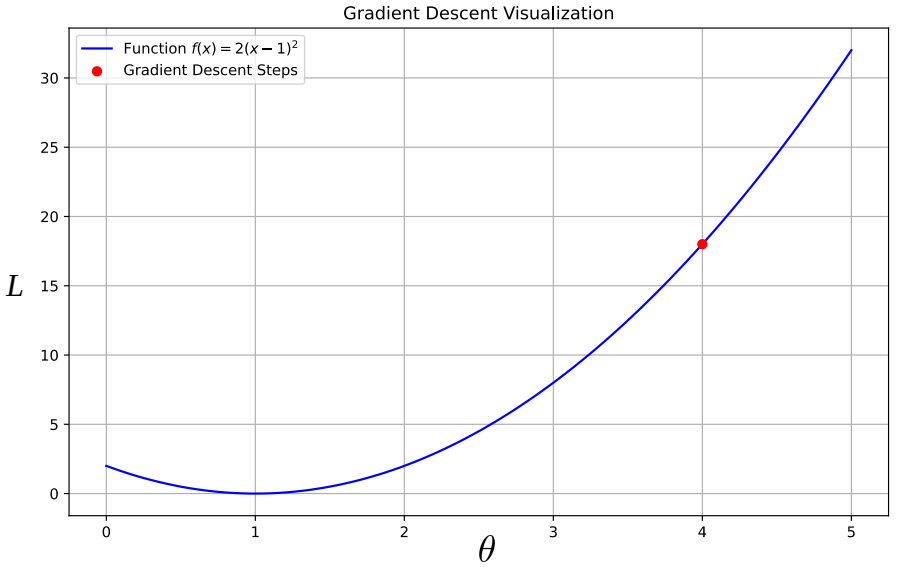




https://www.sharetechnote.com/html/Calculus_Gradient.html

Gradient descent: Algorithmus

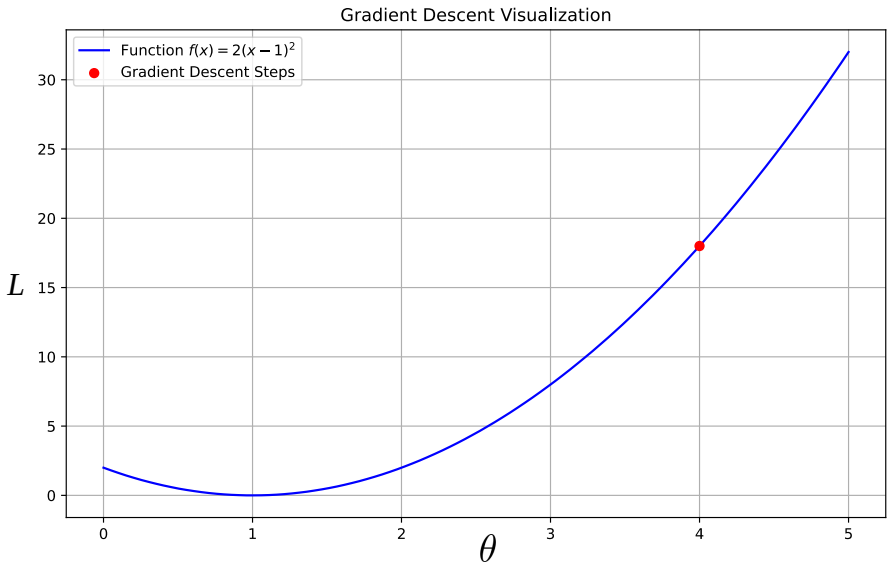
- Gegeben die Funktion $f(\theta)=2(\theta-1)^2$ und ihre Ableitung $\frac{dL}{d\theta}=4(\theta-1)$, Lernrate $\eta=0.1$
- Wir suchen nun das Minimum dieser Funktion:
- Wir starten (als initialen Punkt) bei $x_0=4$



Gradient descent: Algorithmus

- Gegeben die Funktion $f(\theta)=2(\theta-1)^2$ und ihre Ableitung $\frac{dL}{d\theta}=4(\theta-1)$, Lernrate $\eta=0.1$
- Wir suchen nun das Minimum dieser Funktion:
- Wir starten (als initialen Punkt) bei $\theta_0=4$

Ableitung: $\frac{dL}{d\theta}(\theta_0=4)=4(4-1)=4\cdot 3=12$

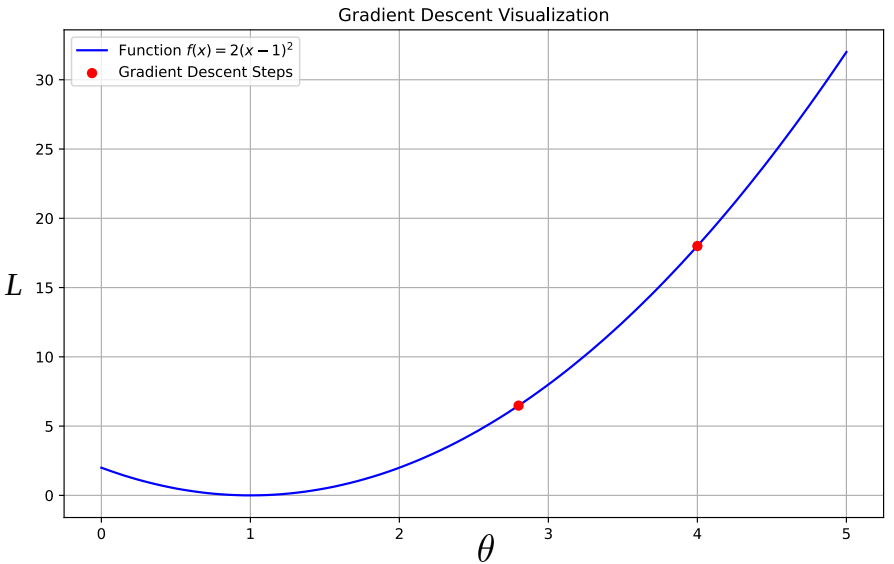


Gradient descent: Algorithmus

- Gegeben die Funktion $f(\theta)=2(\theta-1)^2$ und ihre Ableitung $\frac{dL}{d\theta}=4(\theta-1)$, Lernrate $\eta=0.1$
- Wir suchen nun das Minimum dieser Funktion:
- Wir starten (als initialen Punkt) bei $\theta_0=4$

Ableitung: $\frac{dL}{d\theta}(\theta_0=4)=4(4-1)=4\cdot 3=12$

Update: $\theta_1=\theta_0-\eta\frac{dL}{d\theta}(\theta_0=4)=4-0.1\cdot 12=2.8$



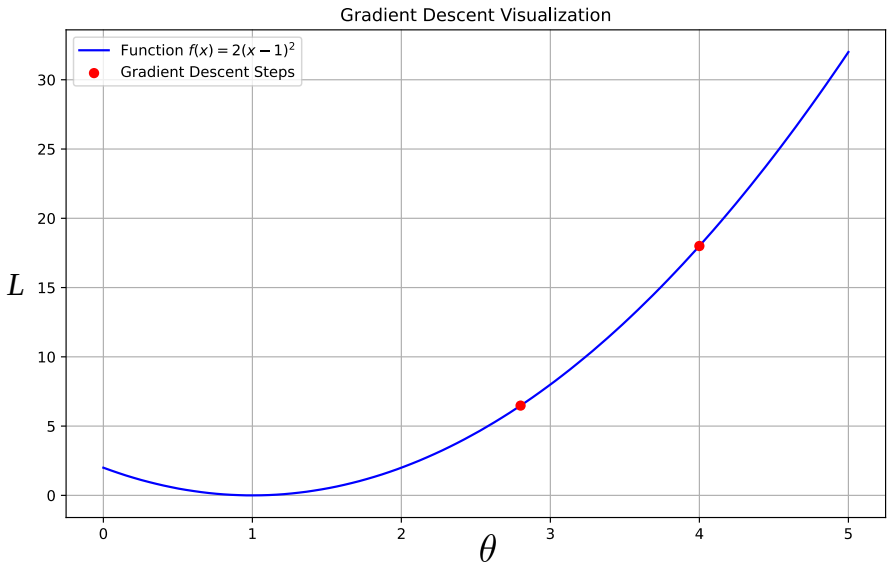
Gradient descent: Algorithmus

- Gegeben die Funktion $f(\theta)=2(\theta-1)^2$ und ihre Ableitung $\frac{dL}{d\theta}=4(\theta-1)$, Lernrate $\eta=0.1$
- Wir suchen nun das Minimum dieser Funktion:
- Wir starten (als initialen Punkt) bei $\theta_0=4$

Ableitung: $\frac{dL}{d\theta}(\theta_0=4)=4(4-1)=4\cdot 3=12$

Update: $\theta_1=\theta_0-\eta\frac{dL}{d\theta}(\theta_0=4)=4-0.1\cdot 12=2.8$

Ableitung: $\frac{dL}{d\theta}(x_1=2.8)=7.2$



Gradient descent: Algorithmus

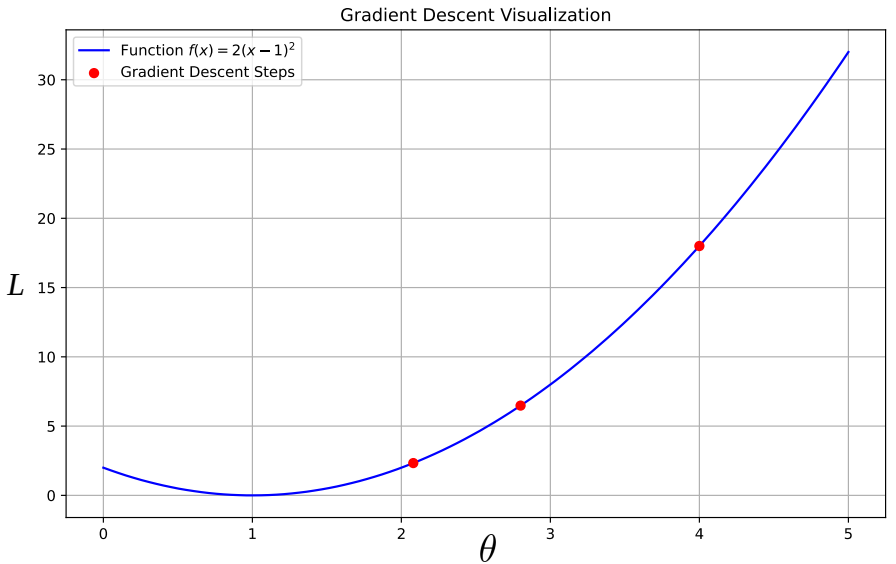
- Gegeben die Funktion $f(\theta)=2(\theta-1)^2$ und ihre Ableitung $\frac{dL}{d\theta}=4(\theta-1)$, Lernrate $\eta=0.1$
- Wir suchen nun das Minimum dieser Funktion:
- Wir starten (als initialen Punkt) bei $\theta_0=4$

Ableitung: $\frac{dL}{d\theta}(\theta_0=4)=4(4-1)=4\cdot 3=12$

Update: $\theta_1=\theta_0-\eta\frac{dL}{d\theta}(\theta_0=4)=4-0.1\cdot 12=2.8$

Ableitung: $\frac{dL}{d\theta}(x_1=2.8)=7.2$

Update: $\theta_2=2.8-0.1\cdot 7.2=2.08$



Gradient descent: Algorithmus

- Gegeben die Funktion $f(\theta)=2(\theta-1)^2$ und ihre Ableitung $\frac{dL}{d\theta}=4(\theta-1)$, Lernrate $\eta=0.1$
- Wir suchen nun das Minimum dieser Funktion:
- Wir starten (als initialen Punkt) bei $\theta_0=4$

Ableitung: $\frac{dL}{d\theta}(\theta_0=4)=4(4-1)=4\cdot 3=12$

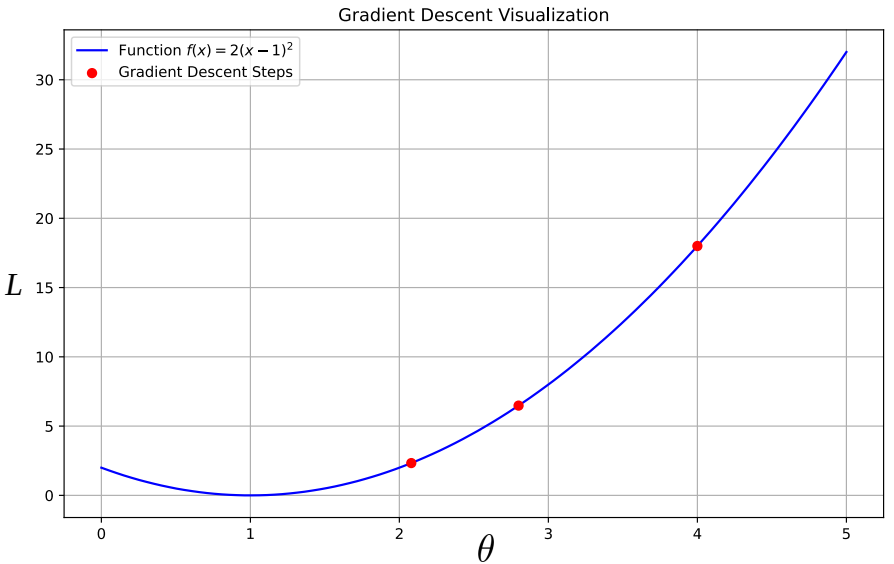
Update: $\theta_1=\theta_0-\eta\frac{dL}{d\theta}(\theta_0=4)=4-0.1\cdot 12=2.8$

Ableitung: $\frac{dL}{d\theta}(x_1=2.8)=7.2$

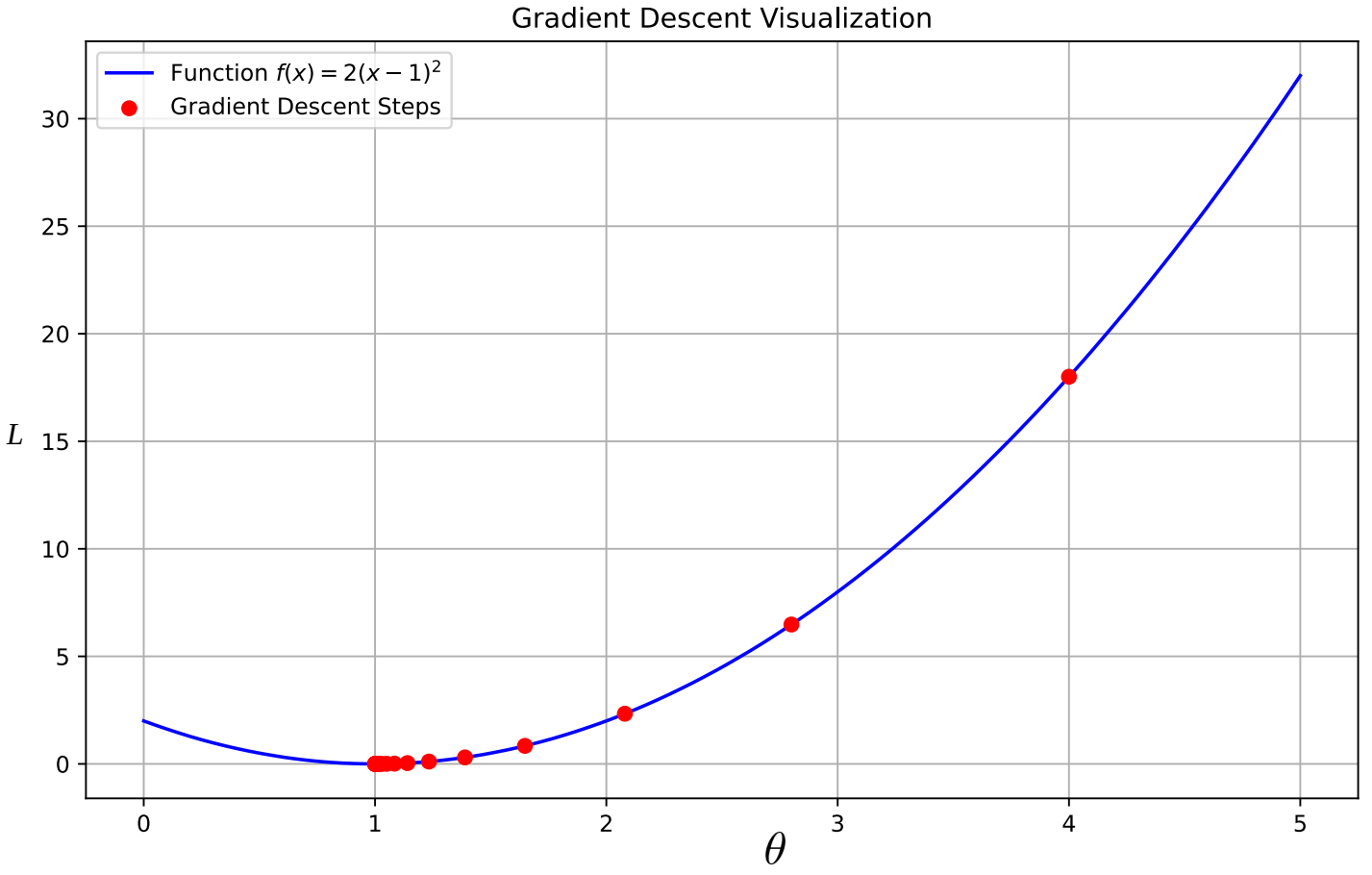
Update: $\theta_2=2.8-0.1\cdot 7.2=2.08$

Ableitung: $\frac{dL}{d\theta}(x_1=2.08)=...$

Update: $\theta_3\approx 1.648$

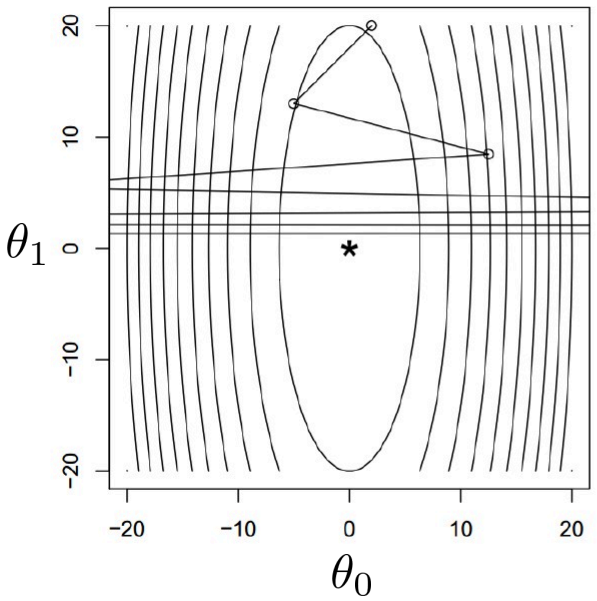


Gradient descent: Algorithmus

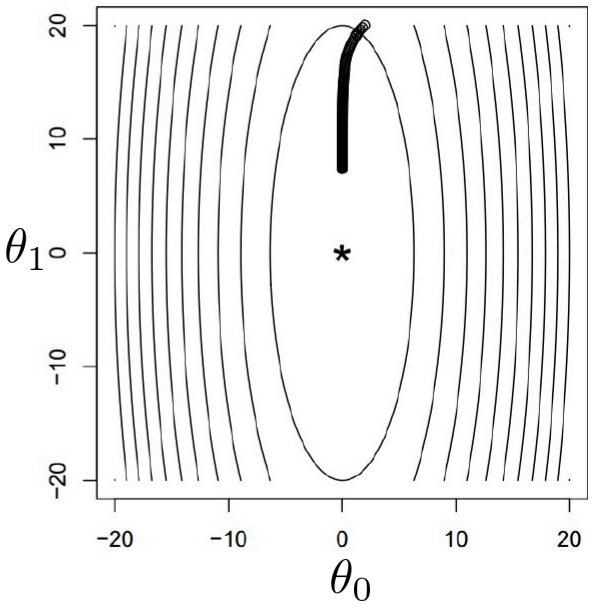


„Richtige“ Lernrate

Lernrate zu groß
Oszillation, Divergenz



Lernrate zu klein
Langsame Konvergenz



Konvergenz, Abbruchkriterium

Änderung als Kriterium

- Gradient unterschreitet kritischen Wert („Threshold“)
- Änderung des Funktionswertes unterschreitet kritischen Wert

Budget als Kriterium

- Zeitschritt/Budget überschreitet kritischen Wert

Stochastic Gradient Descent

Batch Gradient Descent

Loss: $\frac{1}{n}\sum_i l\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)$ Update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{n}\sum_i \nabla_{\boldsymbol{\theta}} l\left(\boldsymbol{x}_i;\boldsymbol{\theta}_t\right)$

→ Summe der Gradienten aller Datenpunkte

Stochastic Gradient Descent

Loss: $l\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)$ Update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\nabla_{\boldsymbol{\theta}} l\left(\boldsymbol{x}_i;\boldsymbol{\theta}_t\right)$

→ Update nach jedem einzelnen Datenpunkt

Vergleich: Batch vs. Stochastic Gradient Descent

Grobe Daumenregel

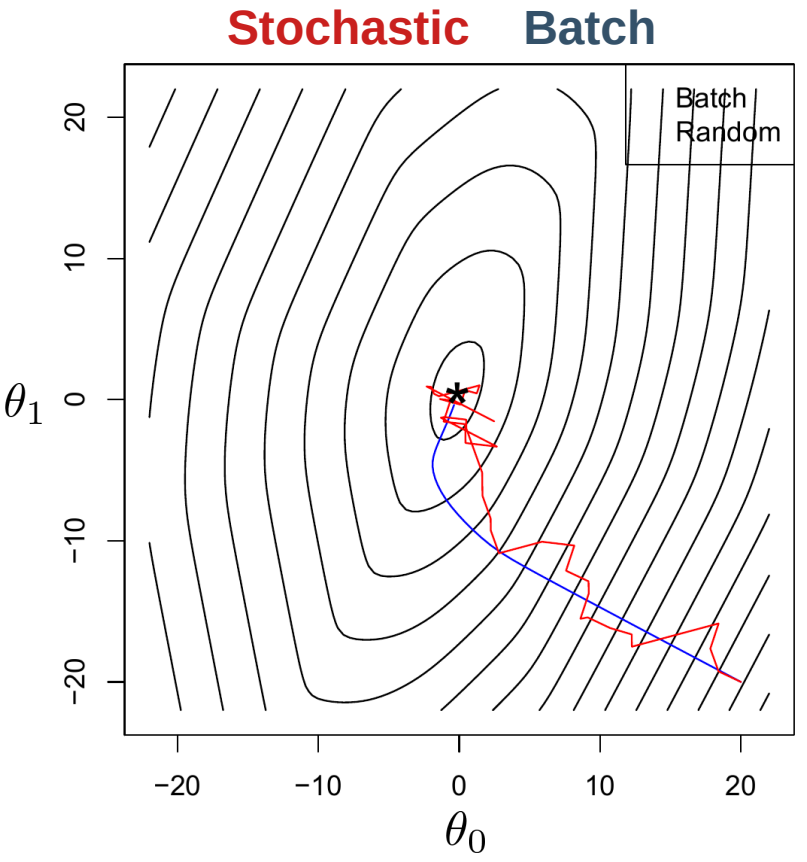
- Stochastic: Gut weit weg vom Optimum
- Batch: Gut nahe am Optimum

Verbesserung

- Zeit-abhängige Lernrate: $\eta_t = \frac{1}{t}$

  → Konvergiert asymptotisch zum Optimum

  → Verhindert Oszillationen um das Optimum

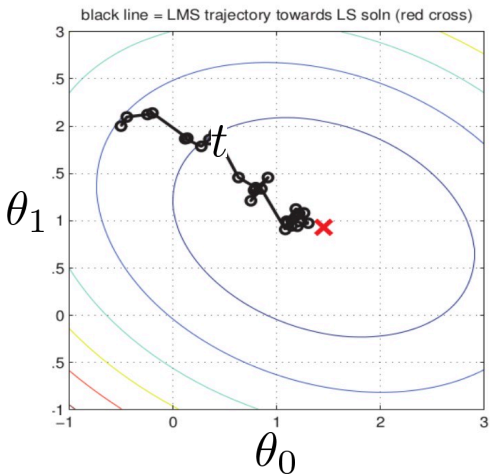


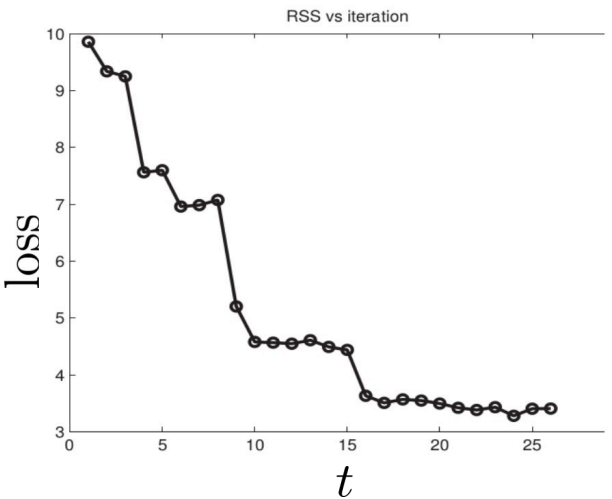
Stochastic Gradient Descent

Ein Datenpunkt pro Gradient Descent Schritt

- Keine Garantie, dass der Gesamtloss in jedem Schritt sinkt
- Iterationen sind schneller
- Braucht mehr Iterationsschritte

  … und kleinere Lernrate





Vergleich: Batch vs. Stochastic Gradient Descent

Intuition: Warum Stochastic Gradient Descent? Wo liegt der Vorteil?

- Typischerweise: Datensatz beinhaltet Redundanz

  → Einige Berechnungen im Batch-Gradienten fast identisch

  → Gradienten für ähnliche Datenpunkte werden mit denselben Parametervektoren berechnet

- Stochastic Gradient Descent: **Parametervektoren werden nach jedem Gradienten ersetzt**

  → Retundante Berechnungen werden verhindert

  → Empirisch: SGD benötigt weniger Berechnungen (in den meisten Fällen)

Sweetspot: Mini-Batches

Idee

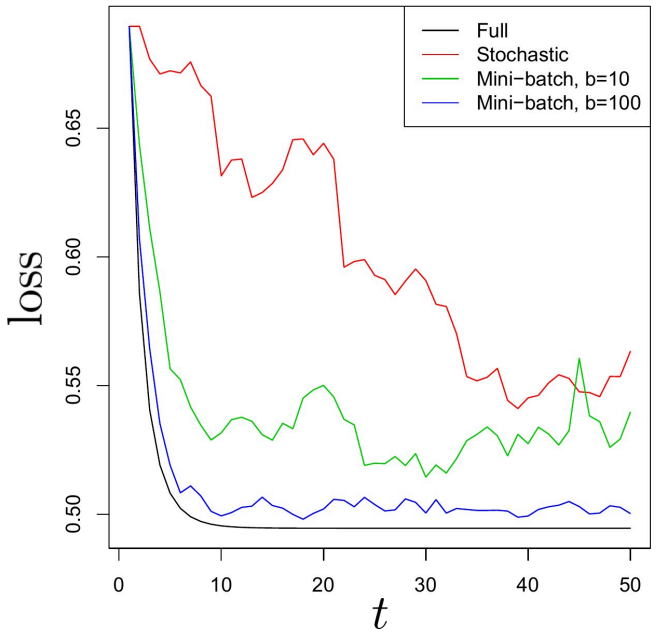
- Benutze Subsets des Trainingssets: $I_t \subset \{1, \ldots, m\},\ |I_t| = b,\ b \ll m$

Loss: $\frac{1}{b} \sum_{i \in I_t} l\left(\boldsymbol{x}_i; \boldsymbol{\theta}\right)$ Update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{b} \sum_{i \in I_t} \nabla_{\boldsymbol{\theta}} l\left(\boldsymbol{x}_i; \boldsymbol{\theta}_t\right)$

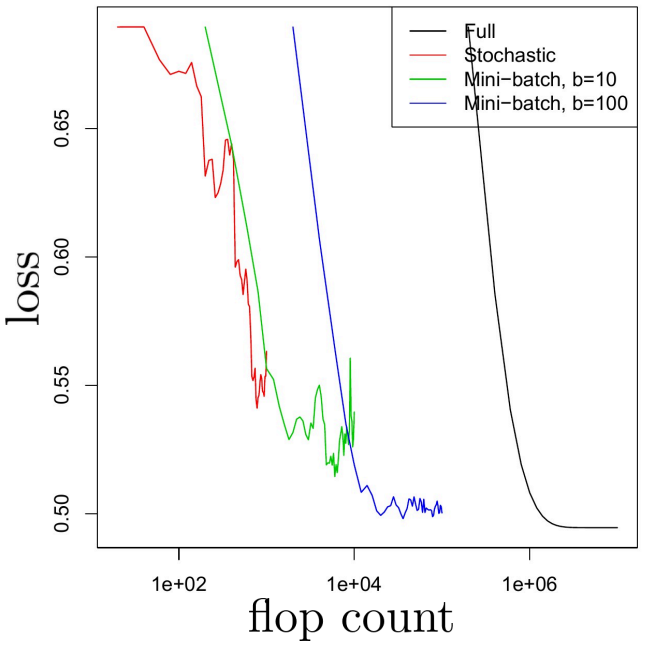
- Mischung aus Stochastic und Batch Gradient Descent
- Weniger Noise/Sprünge als SGD
- Trotzdem weniger Redundanz als BGD
- Sehr effizient für GPU-Implementierungen

Praktisches Beispiel

Logistische Regression, 10000 Datenpunkte, log-likelihood Maximierung



Pro Iteration
(also pro Parameter-Update)

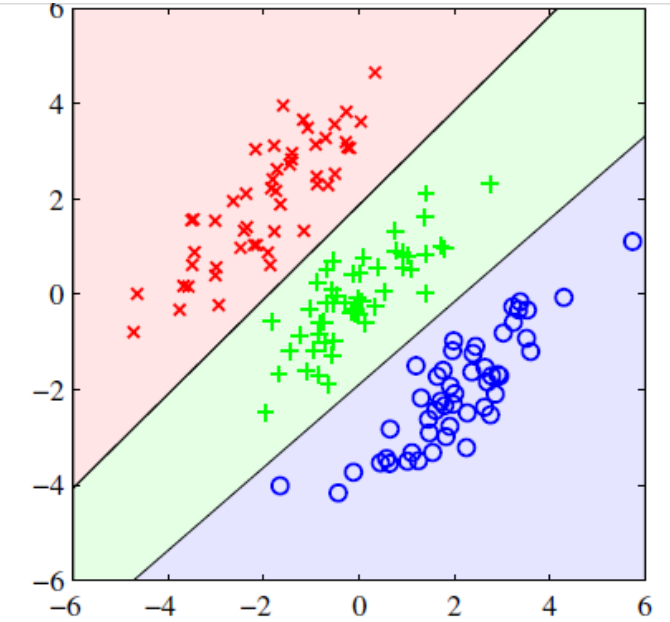


Pro FLOP

Mehr als zwei Klassen: Multiklassen Klassifizierung

Softmax Funktion

$$p(c = i \mid \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x})\right)}{\sum_{k=1}^{K} \exp\left(\boldsymbol{w}_k^T \boldsymbol{\phi}(\boldsymbol{x})\right)}$$

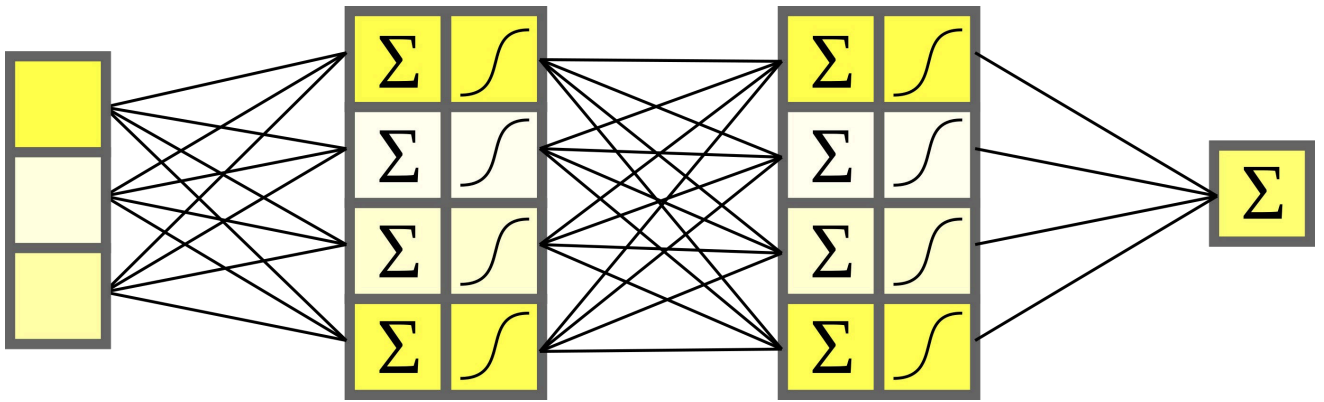


Erklärung

- Ein Gewichtsvektor pro Klasse
- Klassenwahrscheinlichkeit skaliert mit $\boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x})$
- Wahrscheinlichkeit: Normalisierung notwendig
- Für zwei Klassen wäre ein zweiter Gewichtsvektor redundant

  → Optimierung analog zu logistischer Regression (mit Exponential-Trick)

Teil 2



1) Optimierung: Gradient descent

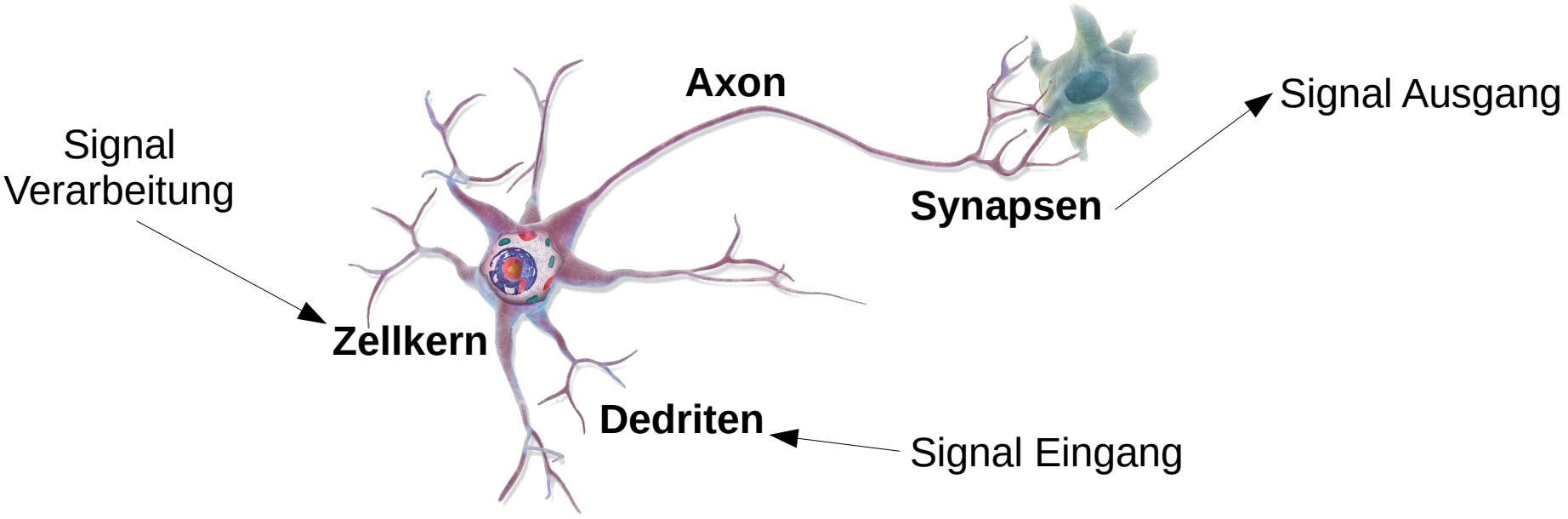
2) Neuronale Netze: Motivation

3) Neuronale Netze: Details

4) Neuronale Netze: Back-propagation

Woher kommt der Name: Biologische Inspiration

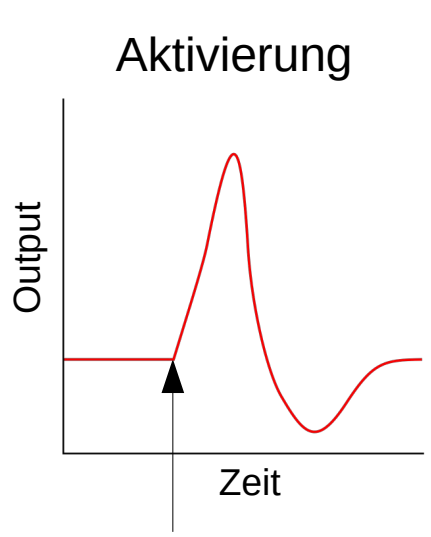
Ein **Neuron** ist die **grundlegende Einheit des Gehirns**



- Unser **Gehirn** hat ungefähr **10^{11} Neuronen**
- Jedes Neuron ist mit ca. 10^4 anderen Neuronen über **Synapsen** verbunden

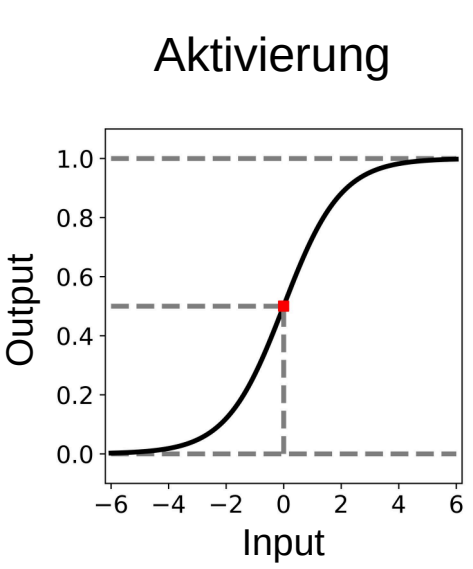
Woher kommt der Name: Biologische Inspiration

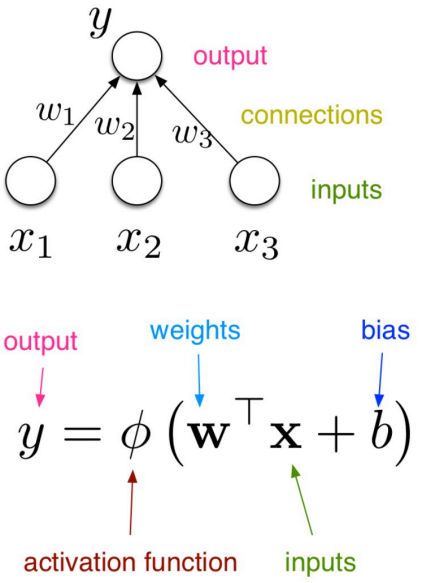
Natürliches Neuron



Hoher Input → Hohe „Feuerrate“

Künstliches Neuron

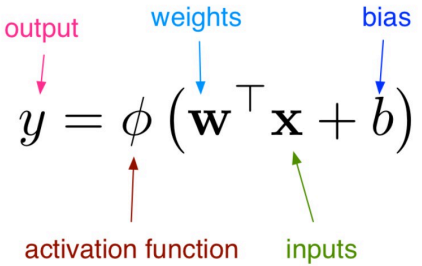


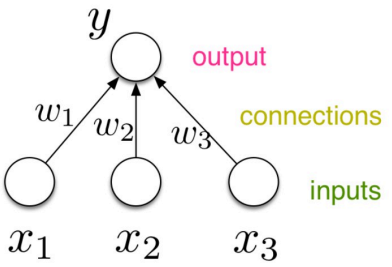


Hoher Input → Hoher Output

Perzeptron/Perceptron

- Historische Einordnung: „Perzeptron“ (perceptron) von Rosenblatt:



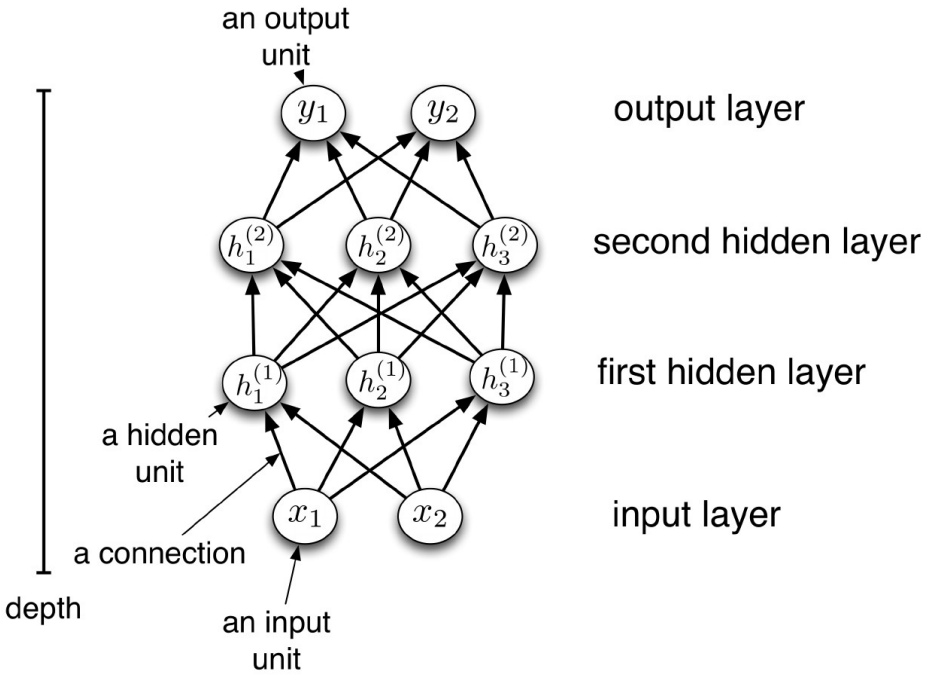


- Wichtige Einschränkung: Als Aktivierungs-Funktion wird eine Stufen-Funktion genutzt → das macht sie schwer zu trainieren/optimieren
- Das „single layer perceptron“ kann
  - Linear trennbare Probleme lösen (z.B. AND)
  - Nicht linear trennbare Probleme (z.B. XOR) **nicht** lösen
- ABER: Fügen wir „layer“ hinzu, dann erhalten wir ein „multi-layer perceptron (MLP)“. Dieses kann auch XOR und andere komplexere Funktionen fitten

Definition eines neuronalen Netzes

In den meisten Fällen

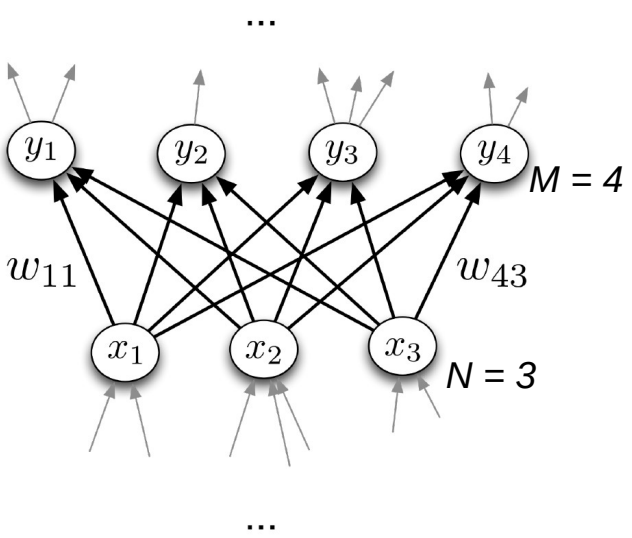
- (Künstliche) Neuronen werden in einen **gerichteten, azyklischen Graphen** angeordnet
- Dies wird „**feed-forward neural network**“ bezeichnet (vorwärts gerichtetes Netzwerk)
- Später werden wir rekurrente neuronale Netze kennenlernen, die Zyklen enthalten
- Typischerweise werden die **Neuronen in Schichten** („layers“) angeordnet: Input, hidden, output



Definition eines neuronalen Netzes

Einzelschicht

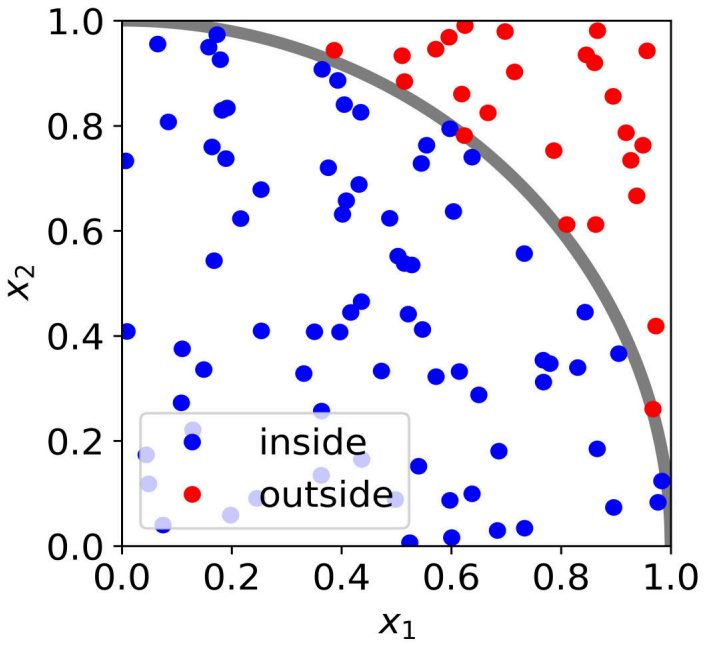
- **Jeder Layer:** *N* Inputs und *M* Outputs
- Im einfachsten Fall sind alle Inputs mit allen Outputs verbunden: „**Fully connected layers**“ (vollständig verbundene Schichten)
- Bemerkung: Input und Output einer Schicht ist nicht das selbe wie Input und Output des Netzwerks
- Jede Schicht: $M \times N$ Gewichtsmatrix θ
- **Berechnung** in jeder Schicht: $y = \phi(\theta x + b)$
- Feed forward neural networks werden häufig auch Multilayer-Perzeptronen genannt („multi-layer perceptrons“)



Einfaches Klassifizierungsproblem

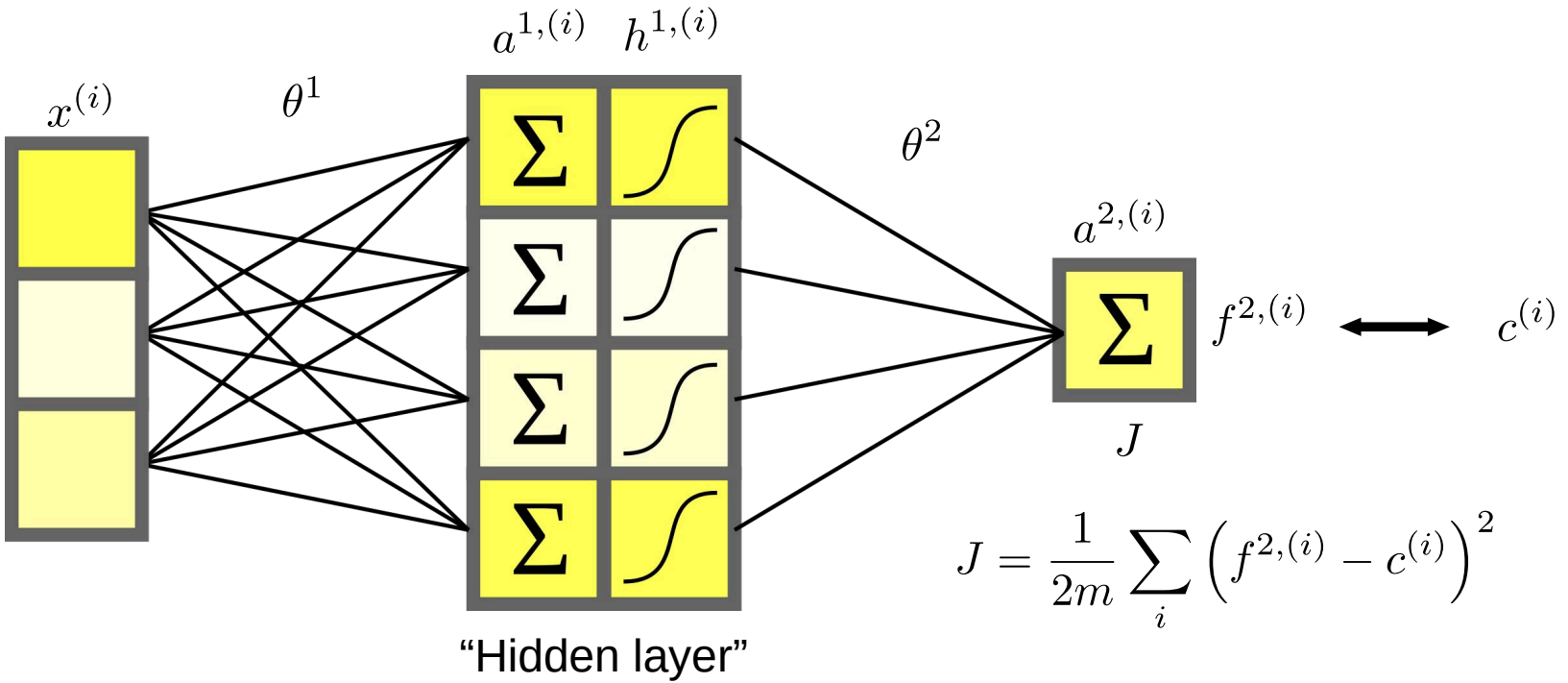
Welche Punkte liegen im Kreis?

Antwort: $x_1^2 + x_2^2 \leq 1$



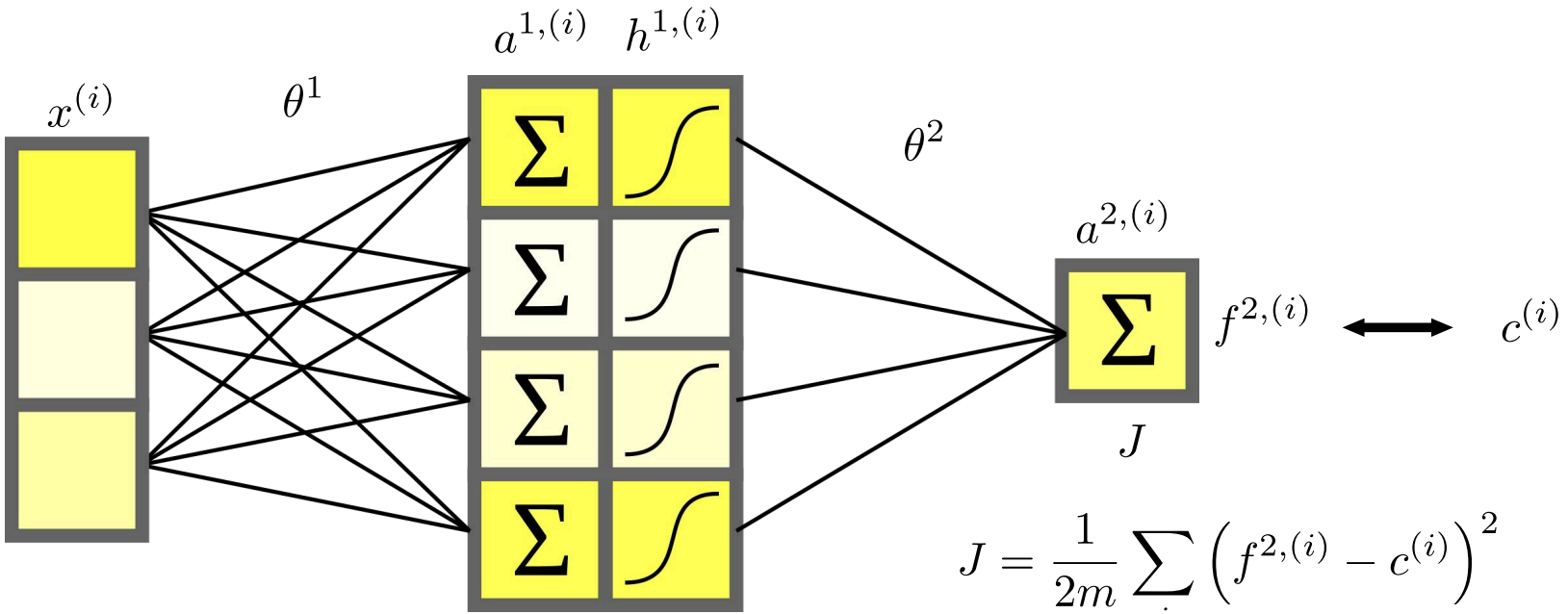
Neuronale Netze

$$x^{(i)} \xrightarrow{\theta^1} a^{1,(i)} = \left(\theta^1\right)^T x^{(i)} \longrightarrow h^{1,(i)} = \sigma\left(a^{1,(i)}\right) \xrightarrow{\theta^2} a^{2,(i)} = \left(\theta^2\right)^T h^{1,(i)}$$



Neuronale Netze: Backpropagation

$$x^{(i)} \xrightarrow{\theta^1} a^{1,(i)} = \left(\theta^1\right)^T x^{(i)} \longrightarrow h^{1,(i)} = \sigma\left(a^{1,(i)}\right) \xrightarrow{\theta^2} a^{2,(i)} = \left(\theta^2\right)^T h^{1,(i)}$$



$$J = \frac{1}{2m} \sum_i \left(f^{2,(i)} - c^{(i)}\right)^2$$

“Hidden layer”

$$\theta^1 \to \theta^1 - \alpha \frac{\partial J}{\partial \theta^1} \text{ ?}$$

$$\theta^2 \to \theta^2 - \alpha \frac{\partial J}{\partial \theta^2} \text{ ?}$$

Erinnerung: Ableitungen

- **Kettenregel**
  - Falls $f(x)$ und $x(t)$ univariate Funktionen

$$\frac{d}{dt} f(x(t)) = \frac{df}{dx}\frac{dx}{dt}$$

- **Beispiel**

$$a = wx + b$$
$$y = \sigma(a)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{L} = \frac{1}{2}(\sigma(wx + b) - t)^2$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w}\frac{1}{2}(\sigma(wx + b) - t)^2 \\
&= (\sigma(wx + b) - t)\frac{\partial}{\partial w}\sigma(wx + b) \\
&= (\sigma(wx + b) - t)\sigma'(wx + b)\frac{\partial}{\partial w}(wx + b) \\
&= (\sigma(wx + b) - t)\sigma'(wx + b)x
\end{aligned}$$

Erinnerung: Ableitungen

- **Ableitungen einer skalaren Funktion nach Vektoren**

$$\nabla_{\boldsymbol{x}} f = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_d}\right]^T$$

Gradient

$$\nabla_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{x} = 2\boldsymbol{x}$$

$$\nabla_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} = 2\boldsymbol{A}\boldsymbol{x}$$

Quadratische Funktionen

- **Ableitungen einer vektorwertigen Funktion nach Vektoren**

$$\nabla_{\boldsymbol{x}} \boldsymbol{f} = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_k(\boldsymbol{x})}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1(\boldsymbol{x})}{\partial x_d} & \cdots & \frac{\partial f_k(\boldsymbol{x})}{\partial x_d} \end{bmatrix}$$

Jacobi Matrix

$$\nabla_{\boldsymbol{x}} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T$$

Lineare Funktion

Erinnerung: Ableitungen

- **Ableitungen einer skalaren Funktion nach Matrizen**

$$\nabla_{\boldsymbol{W}} f = \frac{\partial f(\boldsymbol{W})}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial f(\boldsymbol{W})}{\partial W_{11}} & \cdots & \frac{\partial f(\boldsymbol{W})}{\partial W_{1d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\boldsymbol{W})}{\partial W_{k1}} & \cdots & \frac{\partial f(\boldsymbol{W})}{\partial W_{kd}} \end{bmatrix}$$

Ergebnis: Matrix

- **Ableitungen einer vektorwertigen Funktion nach Matrizen**

Ergebnis: 3D tensor

  - **Hier:** Meistens Ableitung nach Gewichtsmatrix

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \quad \text{und} \quad \boldsymbol{f} = \boldsymbol{f}(\boldsymbol{z})$$

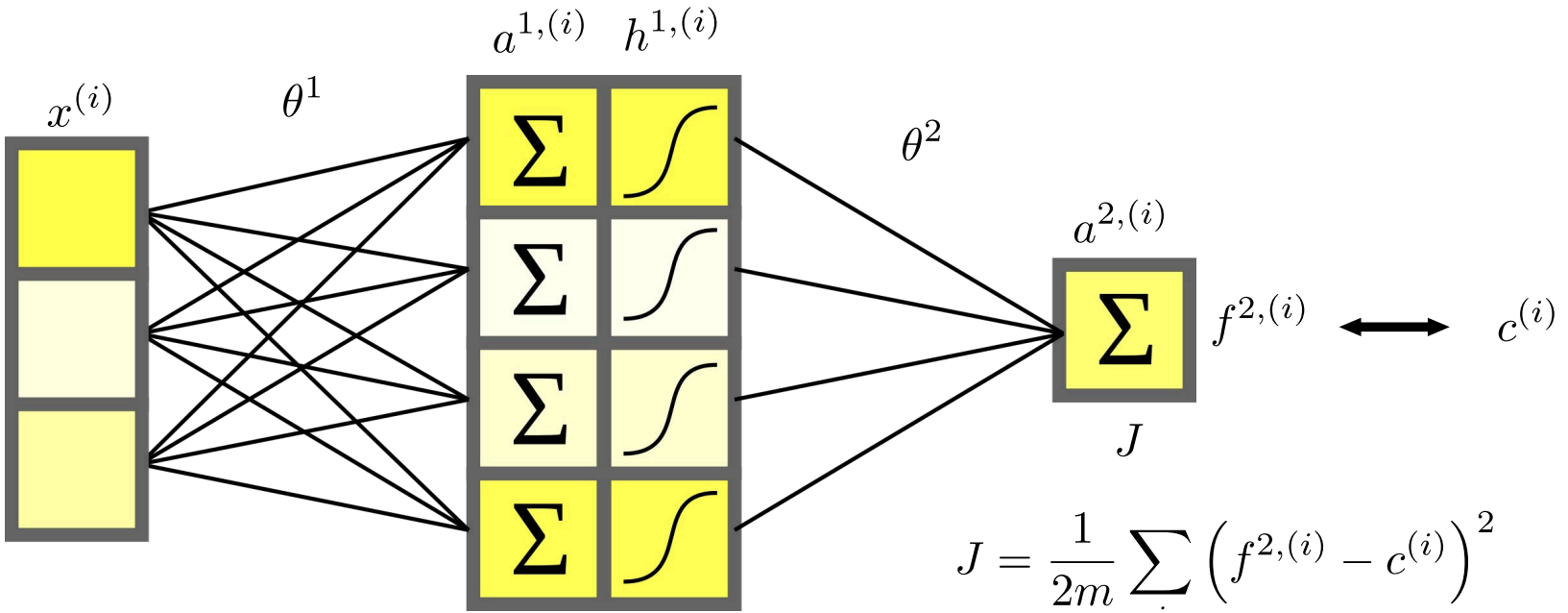
ohne Herleitung →

$$\nabla_{\boldsymbol{W}} \boldsymbol{f} = \frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{W}} = \frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{z}} \frac{\partial}{\partial \boldsymbol{W}} (\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{z}} \boldsymbol{x}^T$$

Matrix-Vektor Produkt

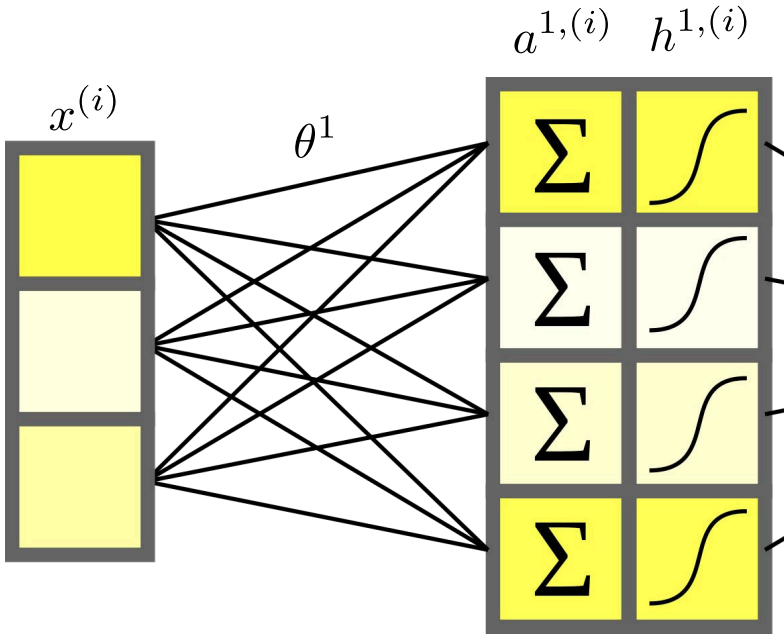
Neuronale Netze: Backpropagation

$$x^{(i)} \xrightarrow{\theta^1} a^{1,(i)} = \left(\theta^1\right)^T x^{(i)} \longrightarrow h^{1,(i)} = \sigma\left(a^{1,(i)}\right) \xrightarrow{\theta^2} a^{2,(i)} = \left(\theta^2\right)^T h^{1,(i)}$$



$$\theta^1 \to \theta^1 - \alpha \frac{\partial J}{\partial \theta^1} \qquad \theta^2 \to \theta^2 - \alpha \frac{\partial J}{\partial \theta^2}$$

Neuronale Netze: Backpropagation



$$J = \frac{1}{2m} \sum_i \left(f^{2,(i)} - c^{(i)} \right)^2$$

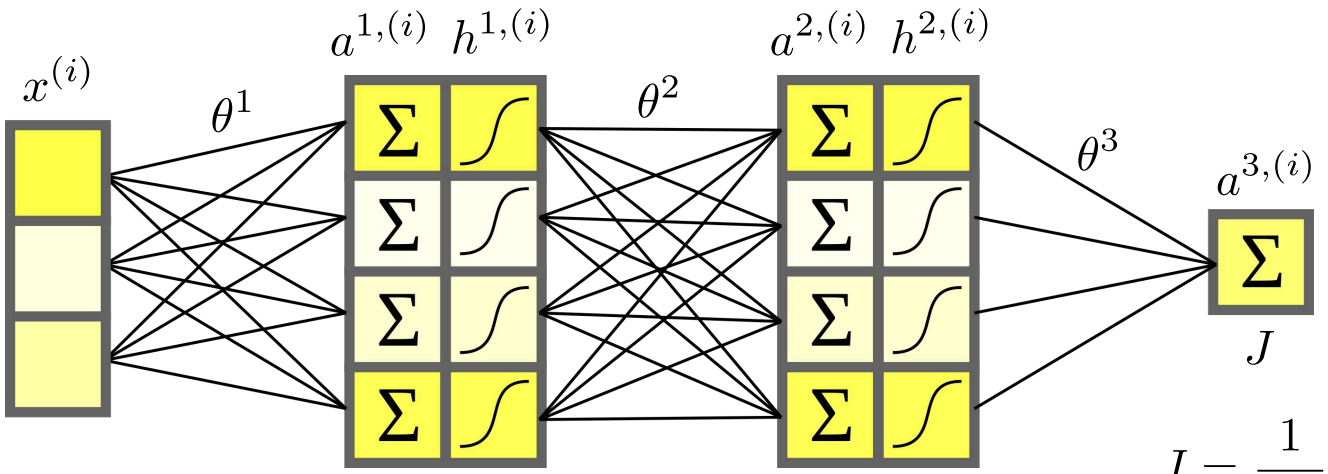
$$\frac{\partial J}{\partial \theta^1} = \frac{\partial J}{\partial a^2} \frac{\partial a^2}{\partial h^1} \frac{\partial h^1}{\partial a^1} \frac{\partial a^1}{\partial \theta^1}$$

$$\frac{\partial J}{\partial \theta^2} = \frac{\partial J}{\partial a^2} \frac{\partial a^2}{\partial \theta^2}$$

$$\theta^1 \rightarrow \theta^1 - \alpha \frac{\partial J}{\partial \theta^1}$$

$$\theta^2 \rightarrow \theta^2 - \alpha \frac{\partial J}{\partial \theta^2}$$

Neuronale Netze: Backpropagation



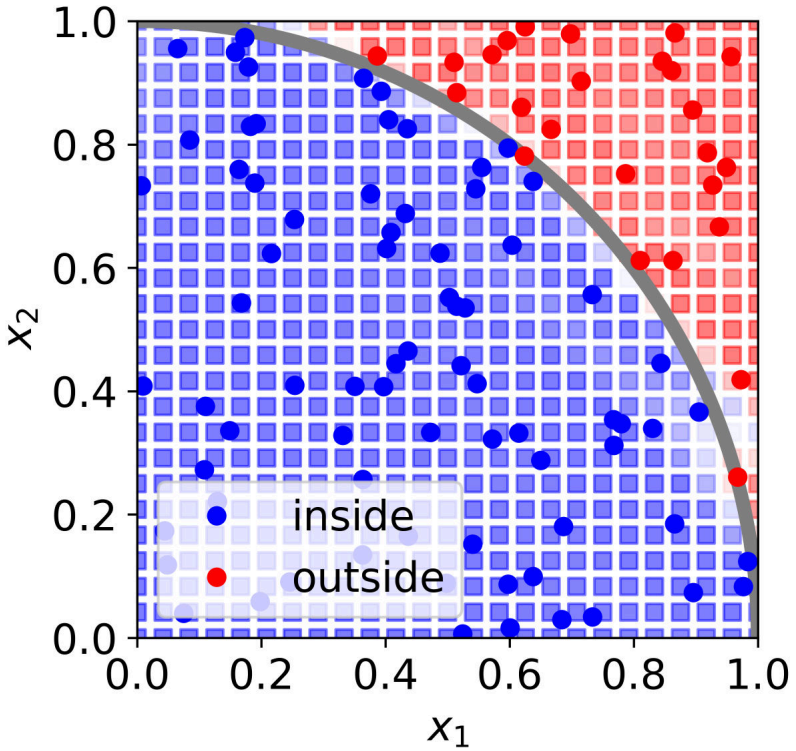
$f^{3,(i)} \longleftrightarrow c^{(i)}$

$$J = \frac{1}{2m} \sum_i \left(f^{3,(i)} - c^{(i)} \right)^2$$

$$\frac{\partial J}{\partial \theta^1} = \frac{\partial J}{\partial a^3} \frac{\partial a^3}{\partial h^2} \frac{\partial h^2}{\partial a^2} \frac{\partial a^2}{\partial h^1} \frac{\partial h^1}{\partial a^1} \frac{\partial a^1}{\partial \theta^1} \leftarrow \frac{\partial J}{\partial \theta^2} = \frac{\partial J}{\partial a^3} \frac{\partial a^3}{\partial h^2} \frac{\partial h^2}{\partial a^2} \frac{\partial a^2}{\partial \theta^2} \leftarrow \frac{\partial J}{\partial \theta^3} = \frac{\partial J}{\partial a^3} \frac{\partial a^3}{\partial \theta^3}$$

$$\theta^1 \rightarrow \theta^1 - \alpha \frac{\partial J}{\partial \theta^1} \qquad \theta^2 \rightarrow \theta^2 - \alpha \frac{\partial J}{\partial \theta^2} \qquad \theta^3 \rightarrow \theta^3 - \alpha \frac{\partial J}{\partial \theta^3}$$

Neuronale Netze: Ergebnisse



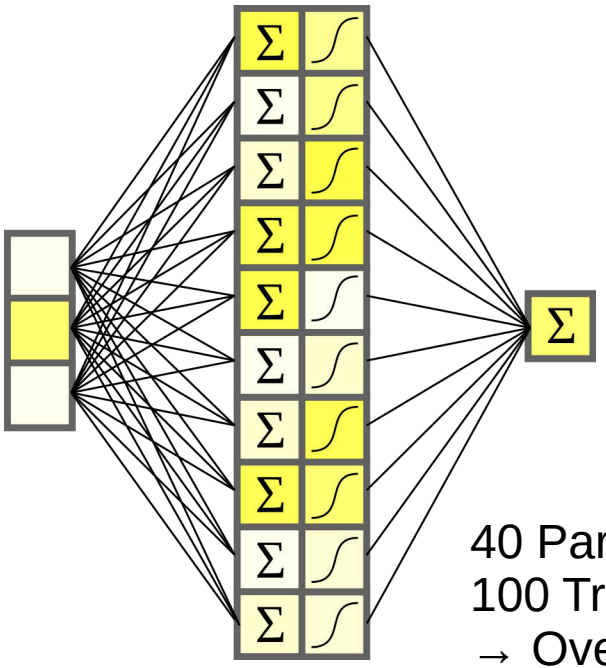
Genauigkeit
Training: 100 %
Test: >98 %

Neuronales Netz

$$a^1(x) = \left(\theta^1\right)^T x$$

$$h^1(x) = \sigma\left(a^1(x)\right)$$

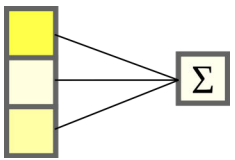
$$f(x) = a^2(x) = \left(\theta^2\right)^T h^1(x)$$

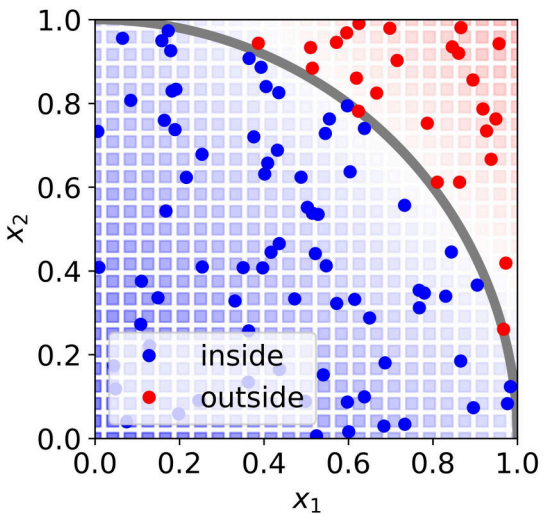


40 Parameter
100 Trainingspunkte
→ Overfitting?

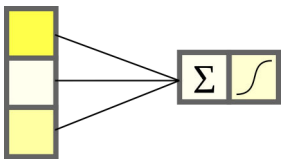
Vergleich mit letzter Vorlesung: Lineare und logistische Regression

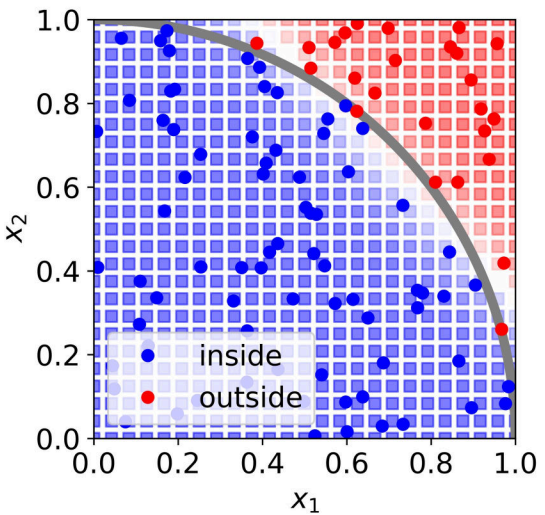
Lineare Regression



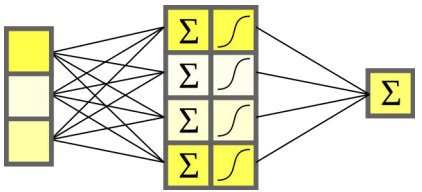


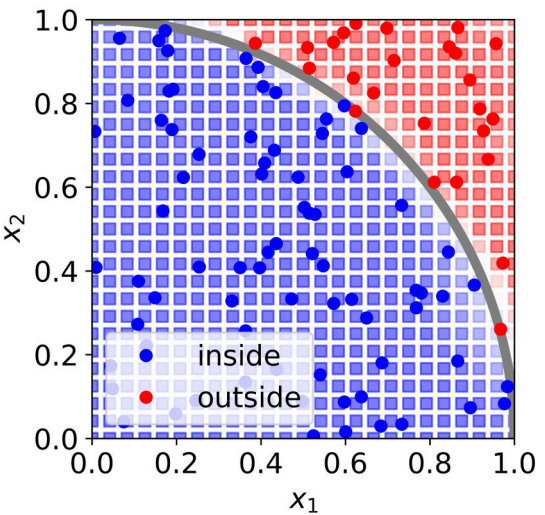
Logistische Regression





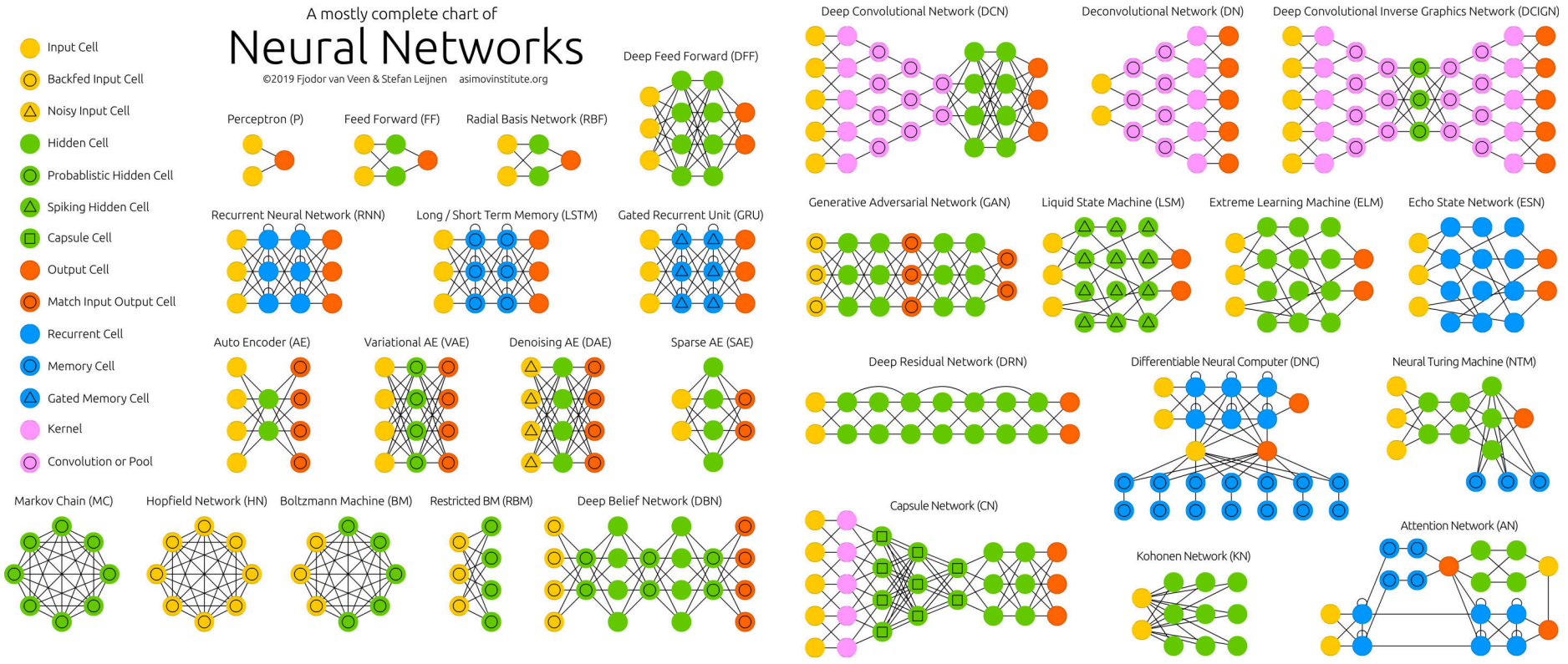
Neuronales Netz





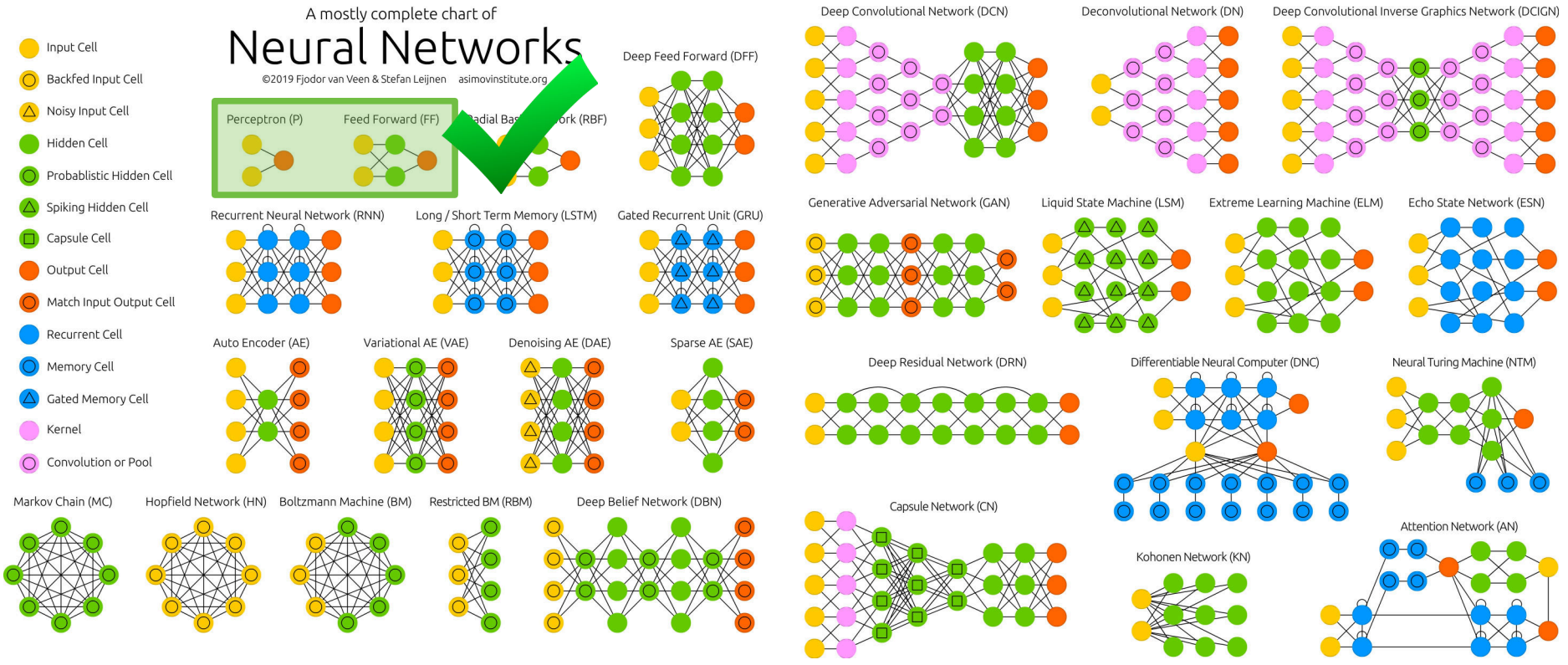
Und jetzt?

A “mostly” complete chart of neural networks *(Asimov institute)*

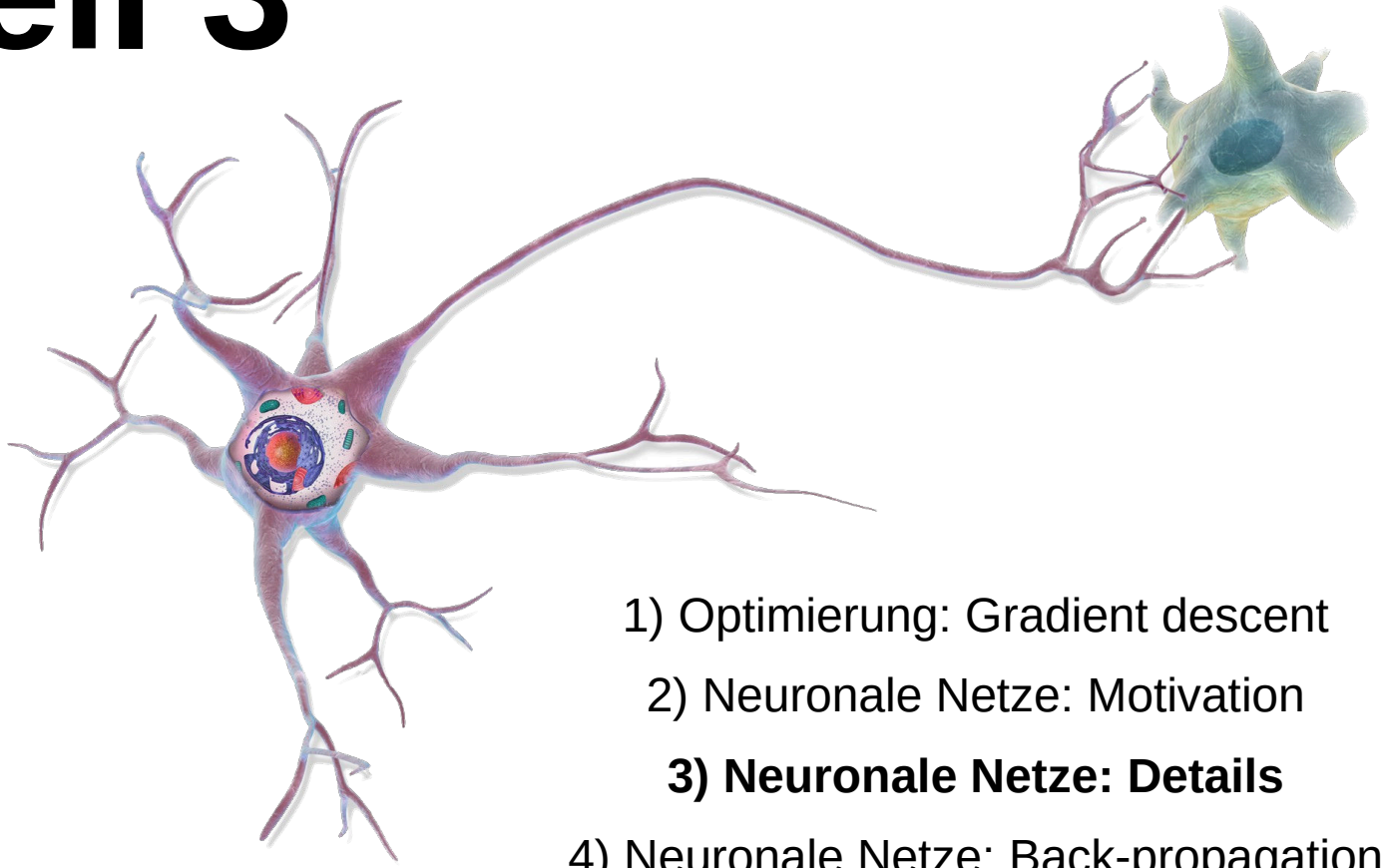


Und jetzt?

A “mostly” complete chart of neural networks *(Asimov institute)*



Teil 3



1) Optimierung: Gradient descent

2) Neuronale Netze: Motivation

3) Neuronale Netze: Details

4) Neuronale Netze: Back-propagation

Warum benötigt man die Aktivierungsfunktion?

Neuronales Netz mit einem hidden layer

$$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}) \qquad y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$$

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)$$

Beide Funktionen sind linear: $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^\top \boldsymbol{x}$

(ohne Bias) $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^\top \boldsymbol{w}$

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{W} \boldsymbol{w}$$

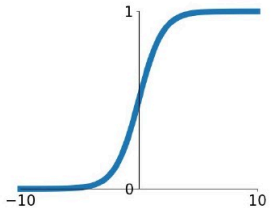
Das ist äquivalent zu einfachem linearen Modell: $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}'$ mit $\boldsymbol{w}' = \boldsymbol{W}\boldsymbol{w}$

Aktivierungsfunktionen

Viele verschiedene Aktivierungsfunktionen können verwendet werden

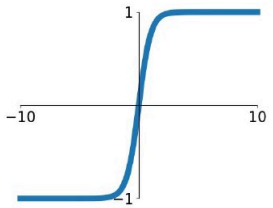
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



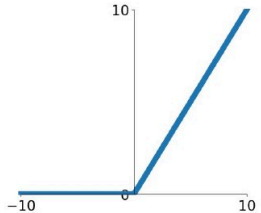
tanh

$$\tanh(x)$$



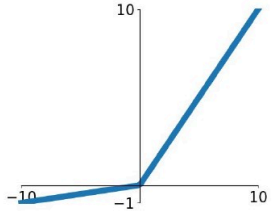
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

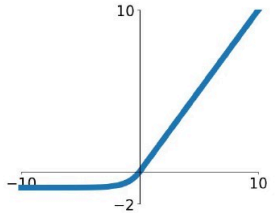


Maxout

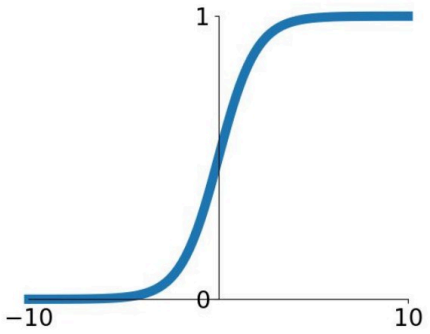
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Aktivierungsfunktionen: Sigmoid



Sigmoid

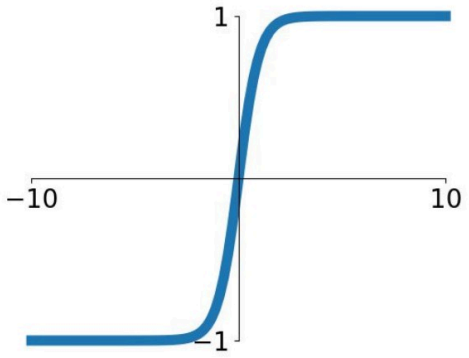
Berechnet: $f(a) = \sigma(a) = \frac{1}{1 + \exp(-a)}$

- Wertebereich [0,1]
- Historisch, Interpretation als saturierende Aktivierungsrate der Neuronen

Probleme

- Wenn Input groß/klein: Flach → Kein Gradient
- Output nicht um 0 zentriert: Macht Initialisierung schwer
- $\exp()$ ist aufwändig zu berechnen

Aktivierungsfunktionen: Tangens hyperbolicus (Tanh)



tanh(x)

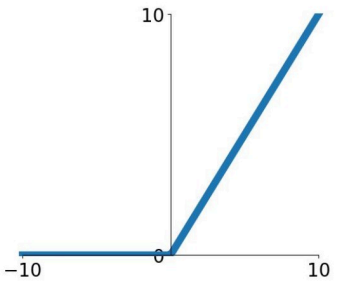
Berechnet: $h(a) = \tanh(a)$

- Um 0 zentriert

Probleme

- Wenn Input groß/klein: Flach → Kein Gradient

Aktivierungsfunktionen: Tangens hyperbolicus (Tanh)



ReLU
(Rectified Linear Unit)

Berechnet: $h(a) = \max(0, a)$

- Saturiert nicht bei großem Input
- Sehr effizient zu berechnen
- Konvergiert typischerweise schneller als sigmoid/tanh

Probleme

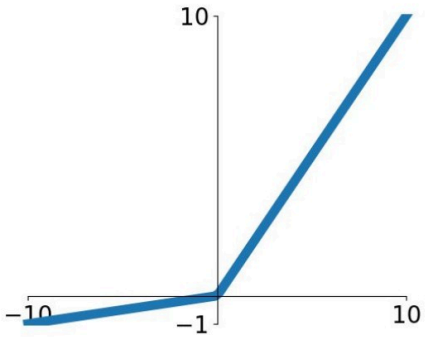
- Wenn Input klein: Flach → Kein Gradient
- Output nicht um 0 zentriert: Macht Initialisierung schwer

Nebenbemerkung: ReLU

- **ReLU** ist ein Akronym für „**Re**ctified **L**inear **U**nit“
- Der Name geht aber zurück auf **Relu** Patrascu, Sysadmin an der Universität von Toronto

Image: Twitter, Relu Patrascu, @ikoflexer

Aktivierungsfunktionen: Leaky ReLU



Leaky ReLU

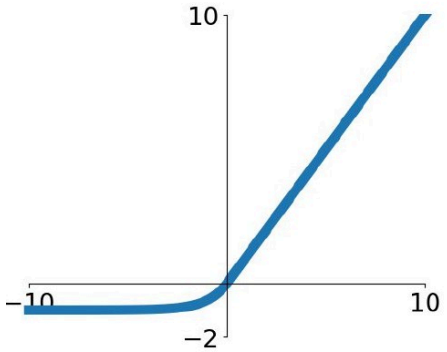
Berechnet: $h(a) = \max(0.1a, a)$

- Saturiert nicht → Hat immer einen Gradienten
- Sehr effizient zu berechnen
- Konvergiert typischerweise schneller als sigmoid/tanh

Erweiterung: Parametric Rectifier (PReLu)

- Berechnet: $h(a) = \max(\alpha a, a)$
- Parameter α muss/kann gelernt werden

Aktivierungsfunktionen: Exponential Linear Unit



Exponential Linear Units (ELU)

Berechnet: $h(x) = \begin{cases} a & \text{if } a > 0 \\ \alpha(\exp(a) - 1) & \text{if } a \leq 0 \end{cases}$

- Alle Vorteile von ReLU
- Outputs im Mittel näher bei 0
- Benötigt die Berechnung von $\exp()$

Aktivierungsfunktionen: Auswahl

Praktisches Vorgehen

- Benutze ReLU, aber justiere die Lernrate und die Initialisierung der Parameter
- Teste Leaky ReLU und ELU
- Teste tanh, aber erwarte nicht zu viel
- Benutze keine sigmoid Funktion (außer als Output für Klassifizierungsprobleme)

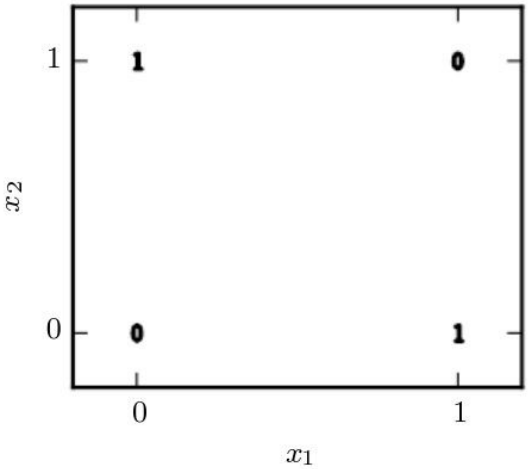
Fazit

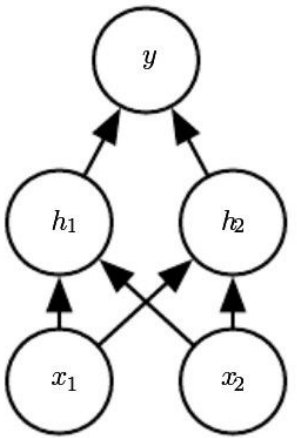
- Aktivierungsfunktion als Hyperparameter: Nächste Woche mehr!

Beispiel: XOR

- Sehr einfaches Problem
- Vier Input Vektoren, vier Labels (0, 1)
- **Nicht lösbar** mit **linearem Modell**
- **Lösbar mit**
  - Neuronalem Netz mit einem hidden layer mit zwei Neuronen
  - Aktivierung: step (nur hier als Beispiel)

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \text{step}\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\right) + b$$





Beispiel: XOR

Eine mögliche Lösung: $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \text{step}\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\right) + b$

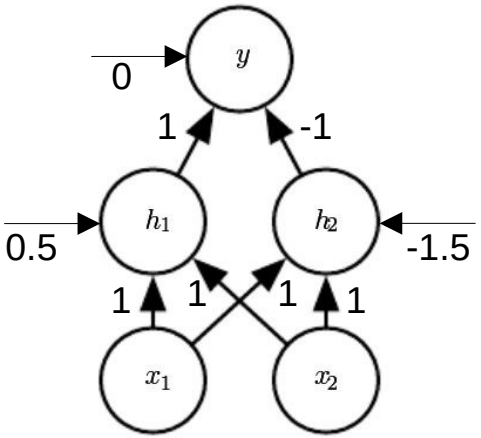
Schritt für Schritt:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{c} = \begin{bmatrix} -0.5 \\ -1.5 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad b = 0$$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \quad \boldsymbol{XW} + \boldsymbol{c} = \begin{bmatrix} -0.5 & -1.5 \\ 0.5 & -0.5 \\ 0.5 & -0.5 \\ 1.5 & 0.5 \end{bmatrix}$$

Für jede Zeile

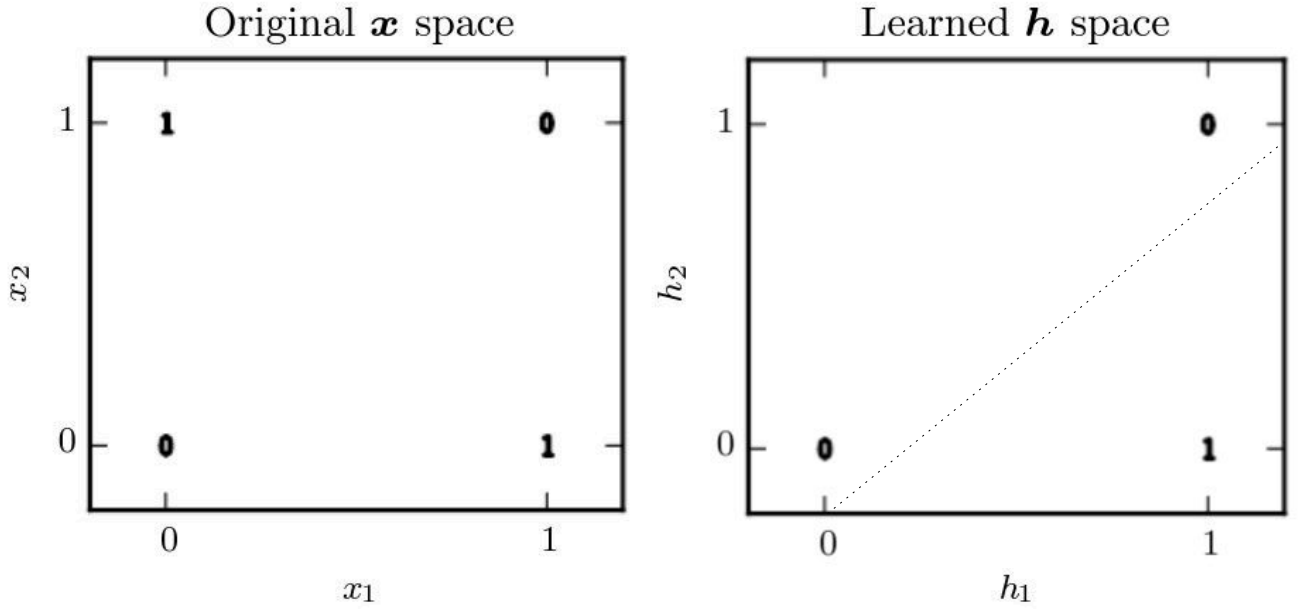
$$\text{step}\left(\boldsymbol{XW} + \boldsymbol{c}\right) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{w}^\top \text{step}\left(\boldsymbol{XW} + \boldsymbol{c}\right) + b = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



Interpretation 1: Lernen einer linear separierbaren Repräsentation

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \text{step}\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\right) + b = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Erklärung



$$\max\{0, \boldsymbol{X}\boldsymbol{W} + \boldsymbol{c}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Interpretation 1: Lernen der Logik

Aufgabe

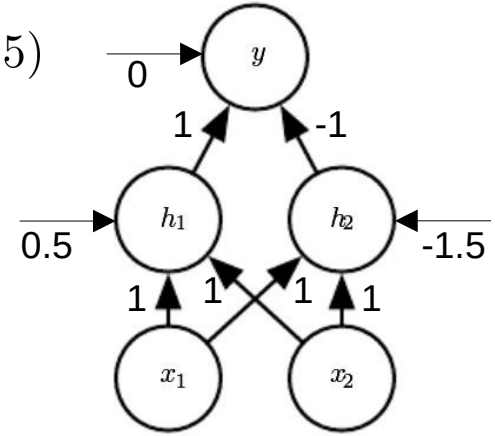
$$\mathrm{XOR}(x_1, x_2) = (x_1 \mathrm{OR}\ x_2)\ \mathrm{AND\ NOT}\ (x_1 \mathrm{AND}\ x_2)$$

Erklärung

$$h_1 = (x_1 \mathrm{OR}\ x_2) \qquad h_1 = \mathrm{step}(1 \cdot x_1 + 1 \cdot x_2 - 0.5)$$

$$h_2 = (x_1 \mathrm{AND}\ x_2) \qquad h_2 = \mathrm{step}(1 \cdot x_1 + 1 \cdot x_2 - 1.5)$$

$$y = h_1\ \mathrm{AND\ NOT}\ h_2 \qquad y = (h_1 - h_2)$$



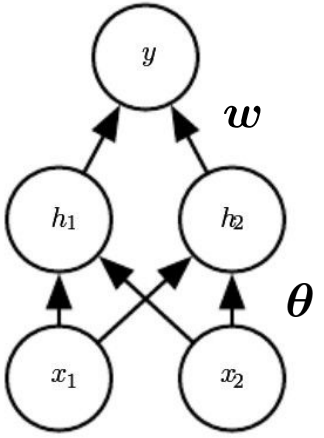
Allgemeiner: Representation Learning

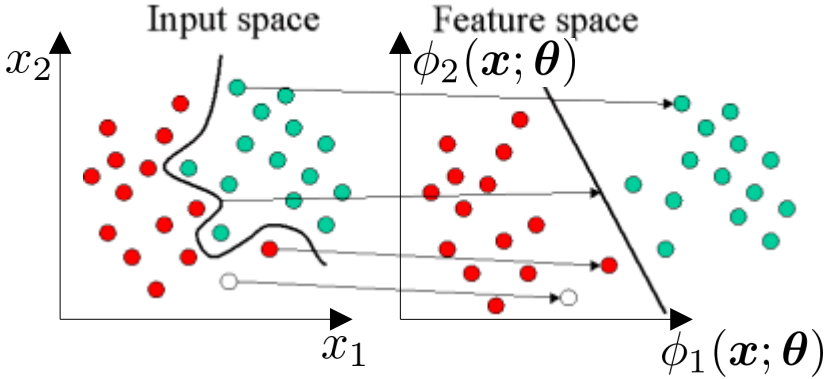
Interpretation von neuronalen Netzen

- Kombination aus:
  - Nicht-lineare Transformation des Inputs (Gewichte $\boldsymbol{\theta}$)
  - Lineares Modell (Gewichte $\boldsymbol{w}$)

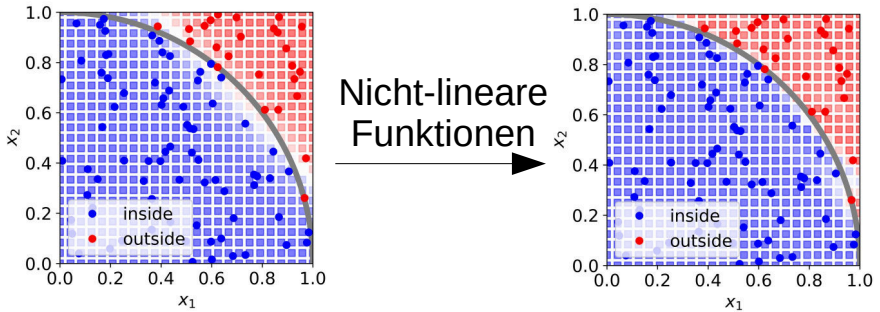
$$y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^\top \boldsymbol{w}$$

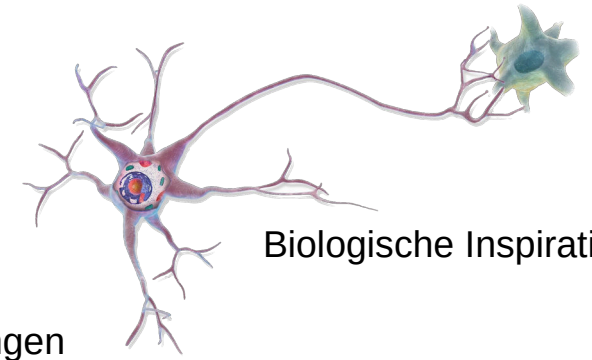
($\boldsymbol{h} = \phi(\boldsymbol{x}; \boldsymbol{\theta})$)





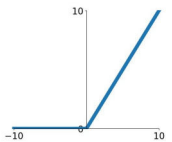
Zusammenfassung





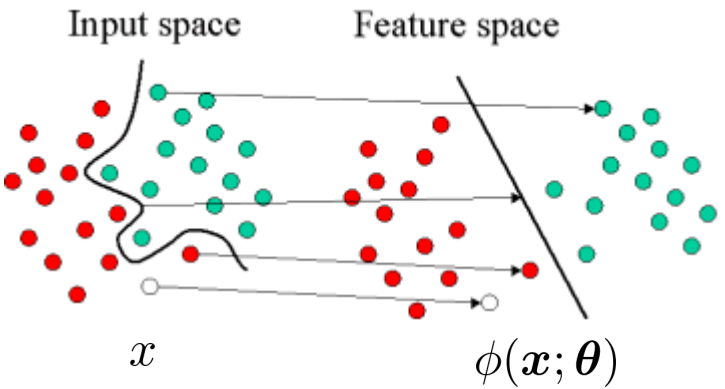
Biologische Inspiration

Aktivierungen



ReLU
(Rectified Linear Unit)

Representation learning



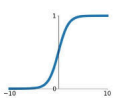
1) Optimierung: Gradient descent

2) Neuronale Netze: Motivation

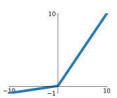
3) Neuronale Netze: Details

Nächste Woche: Details zu Back-Propagation

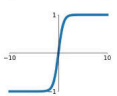
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



Leaky ReLU
$\max(0.1x, x)$

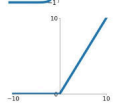


tanh
$\tanh(x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU
$\max(0, x)$



ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$