Neuronale Netze
Mehrschichtige Perzeptren

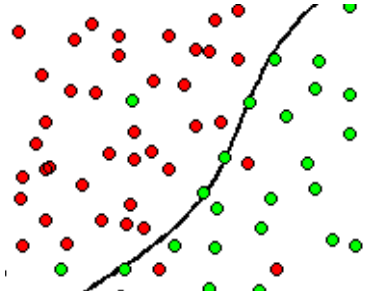
Prof. Dr.-Ing. Sebastian Stober

Artificial Intelligence Lab
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
stober@ovgu.de

FACULTY OF
COMPUTER SCIENCE

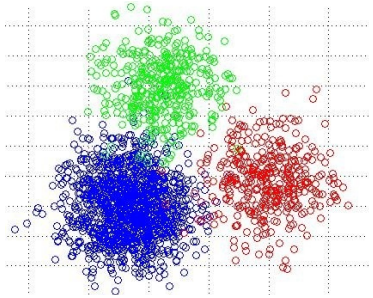
Machine Learning Grundlagen

ML Problemklassen



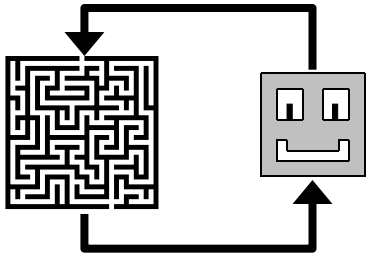
Überwachtes Lernen

(supervised learning)



Unüberwachtes Lernen

(unsupervised learning)

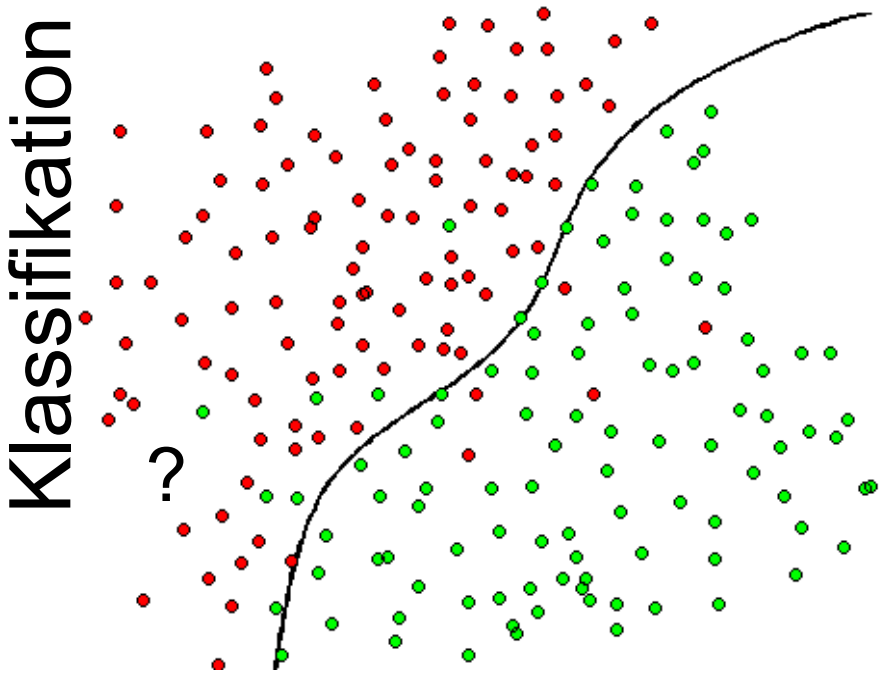


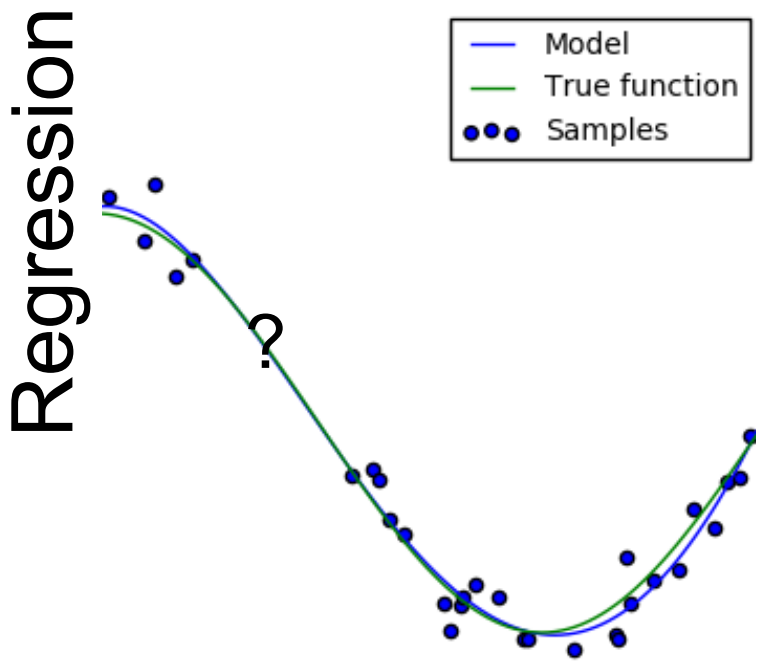
Bestärkendes Lernen

(reinforcement learning)

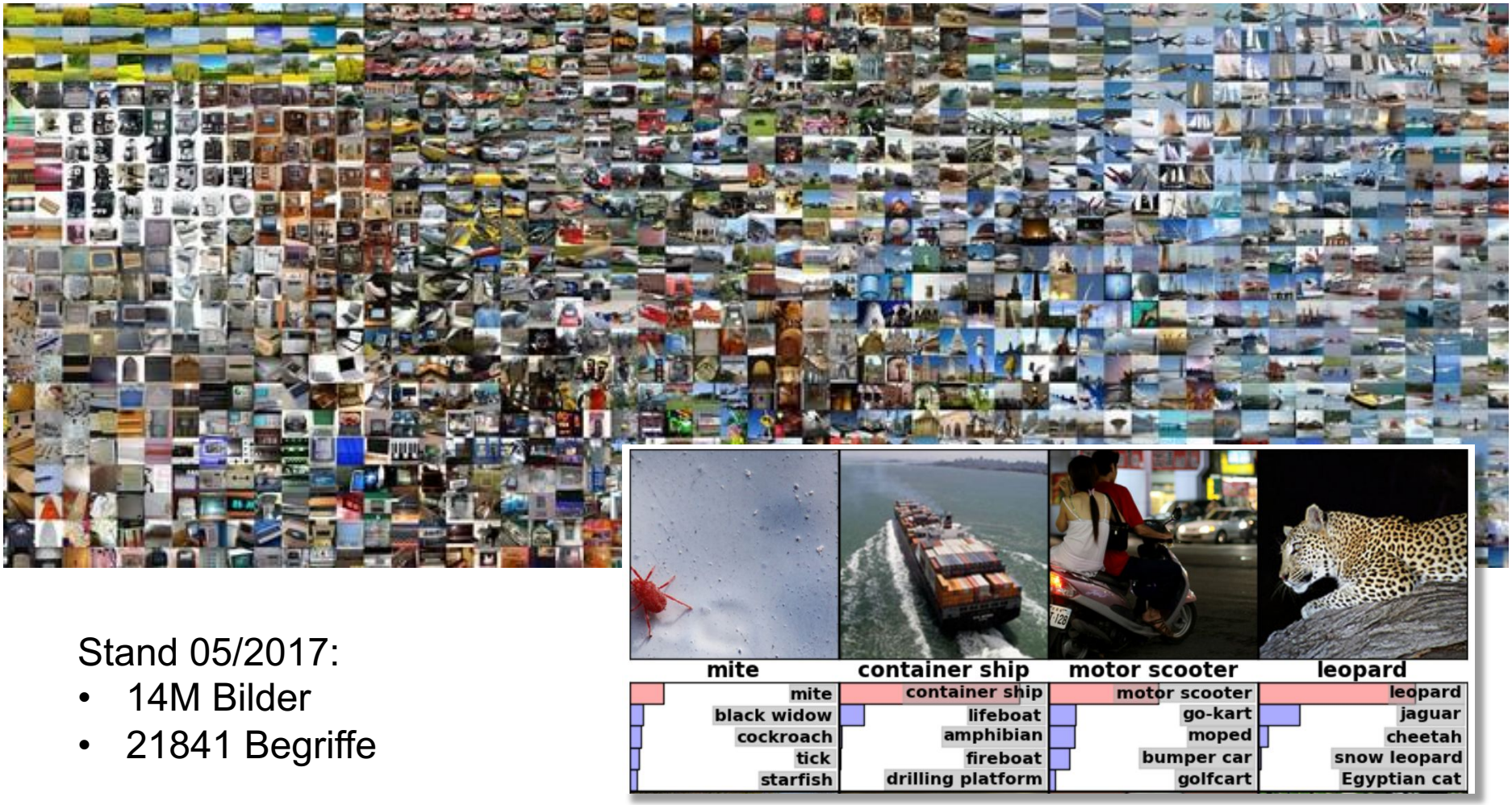
Überwachtes Lernen

- annotierte Trainingsdaten (Beispielein- und Ausgaben)
- lerne Vorhersagemodell





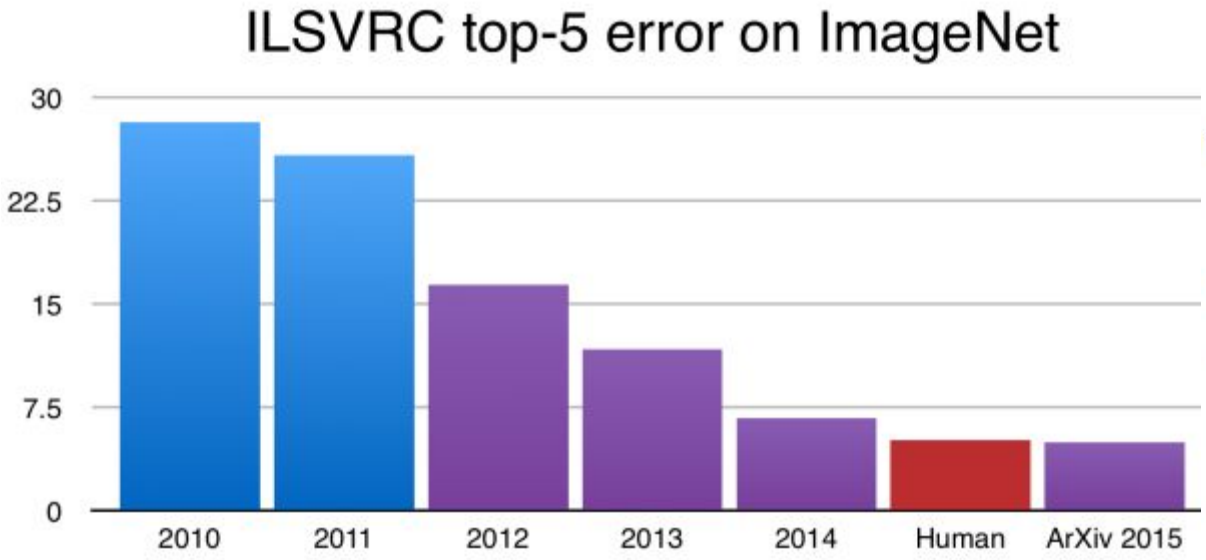
IMAGENET



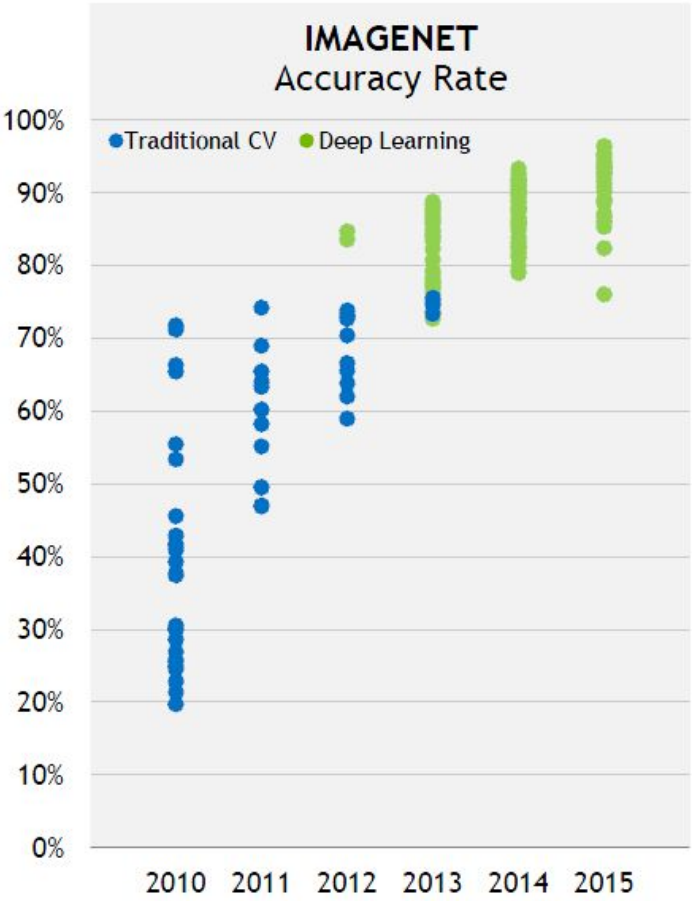
Stand 05/2017:

- 14M Bilder
- 21841 Begriffe

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

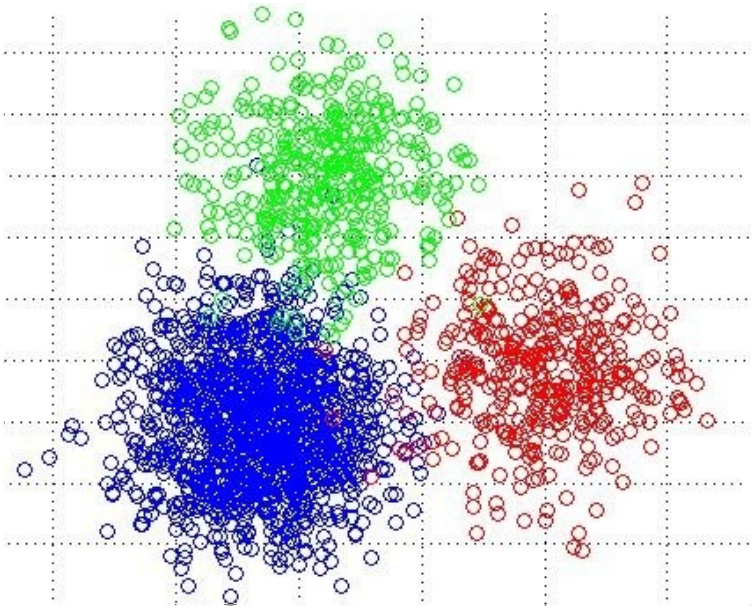


- Blue: Traditional CV
- Purple: Deep Learning
- Red: Human



Unüberwachtes Lernen

- Trainingdaten **ohne Annotationen**
- Lerne Struktur der Daten



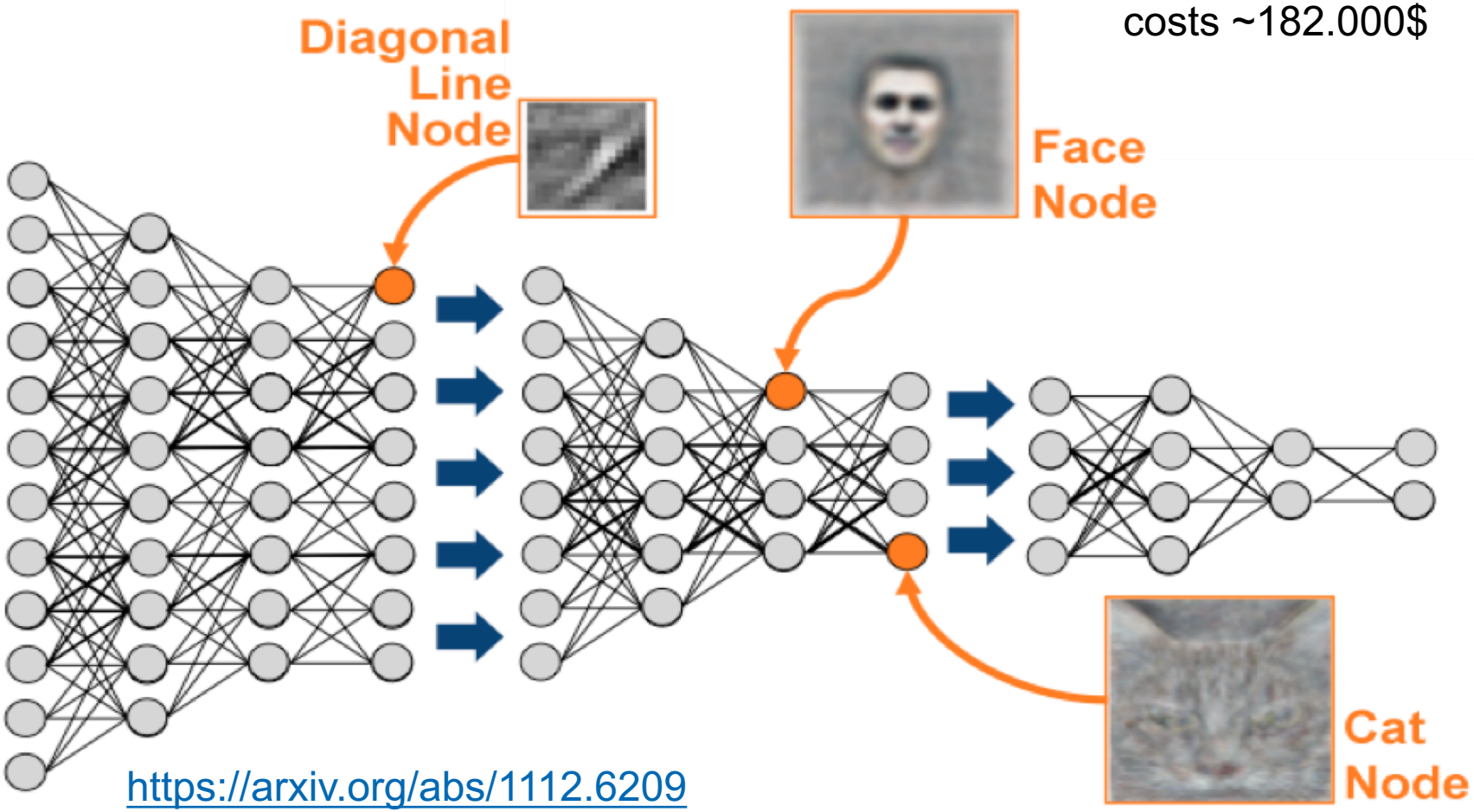
a) eigenständiges Ziel (Muster entdecken)

b) Zwischenschritt der Datenverarbeitung

Google Brain Experiment (2012)

2000 x 8 CPUs look at images from Youtube videos for 1 week

costs ~182.000$

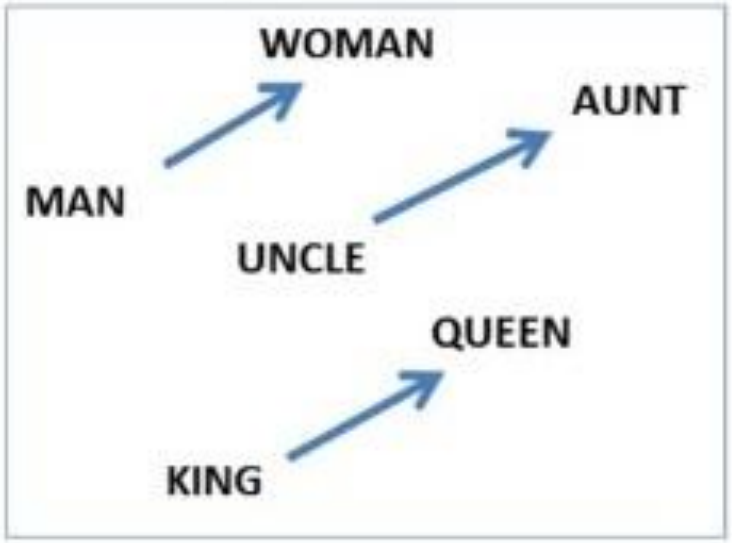


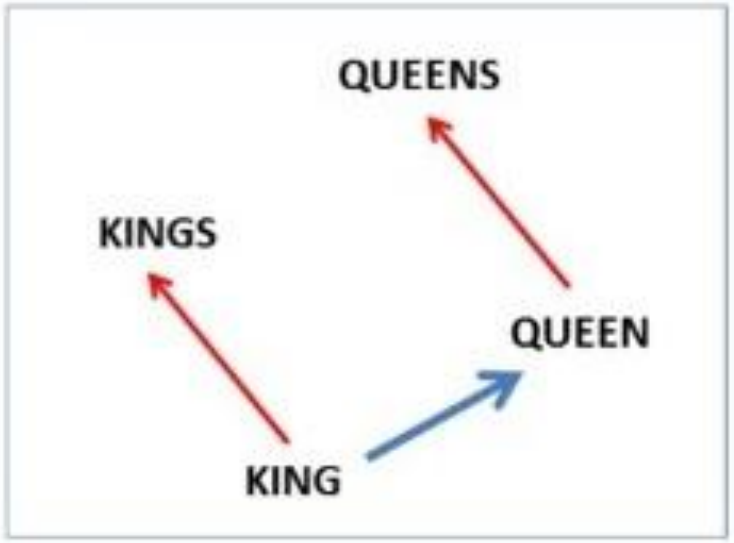
https://arxiv.org/abs/1112.6209

Wort-Mathematik

(Gelernte Wort-Semantik nur durch Anschauen großer Textmengen)

vec(“king”) – vec(“man”) + vec(“woman”) = vec(“queen”)

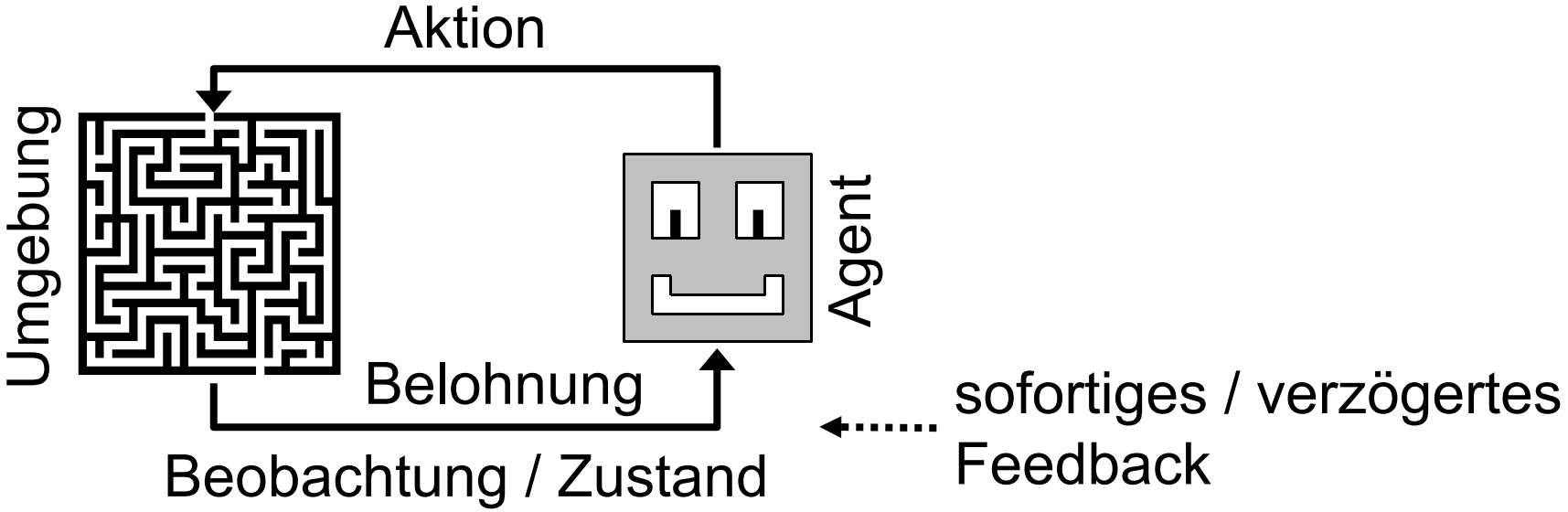




https://code.google.com/archive/p/word2vec/

Bestärkendes Lernen

- Trainingdaten durch Interaktion mit Umgebung
- Lerne Verhalten, welches die kumulative (verzögerte!) Belohnung über die Zeit maximiert

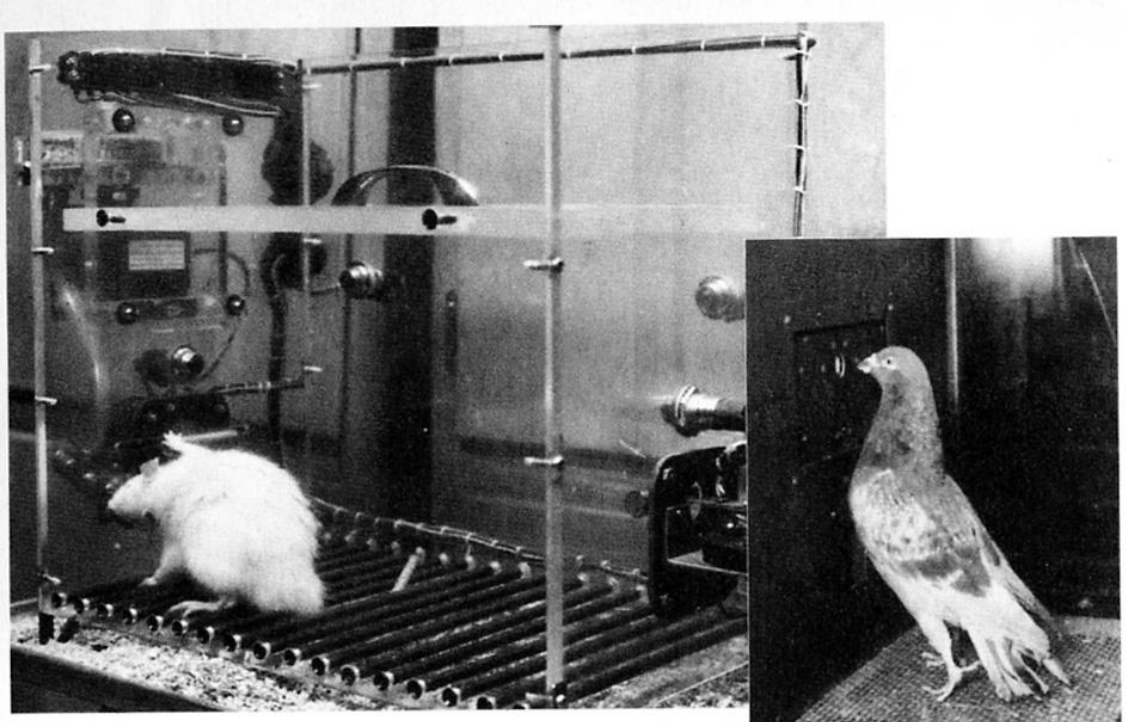


Reinforcement Learning in the Real World:

Operant Conditioning

- Rewarding „free“ behaviours
- Controlling the outcomes

Burrhus F. Skinnner (1904-1990)



A rat pressing a bar in a Skinner box.

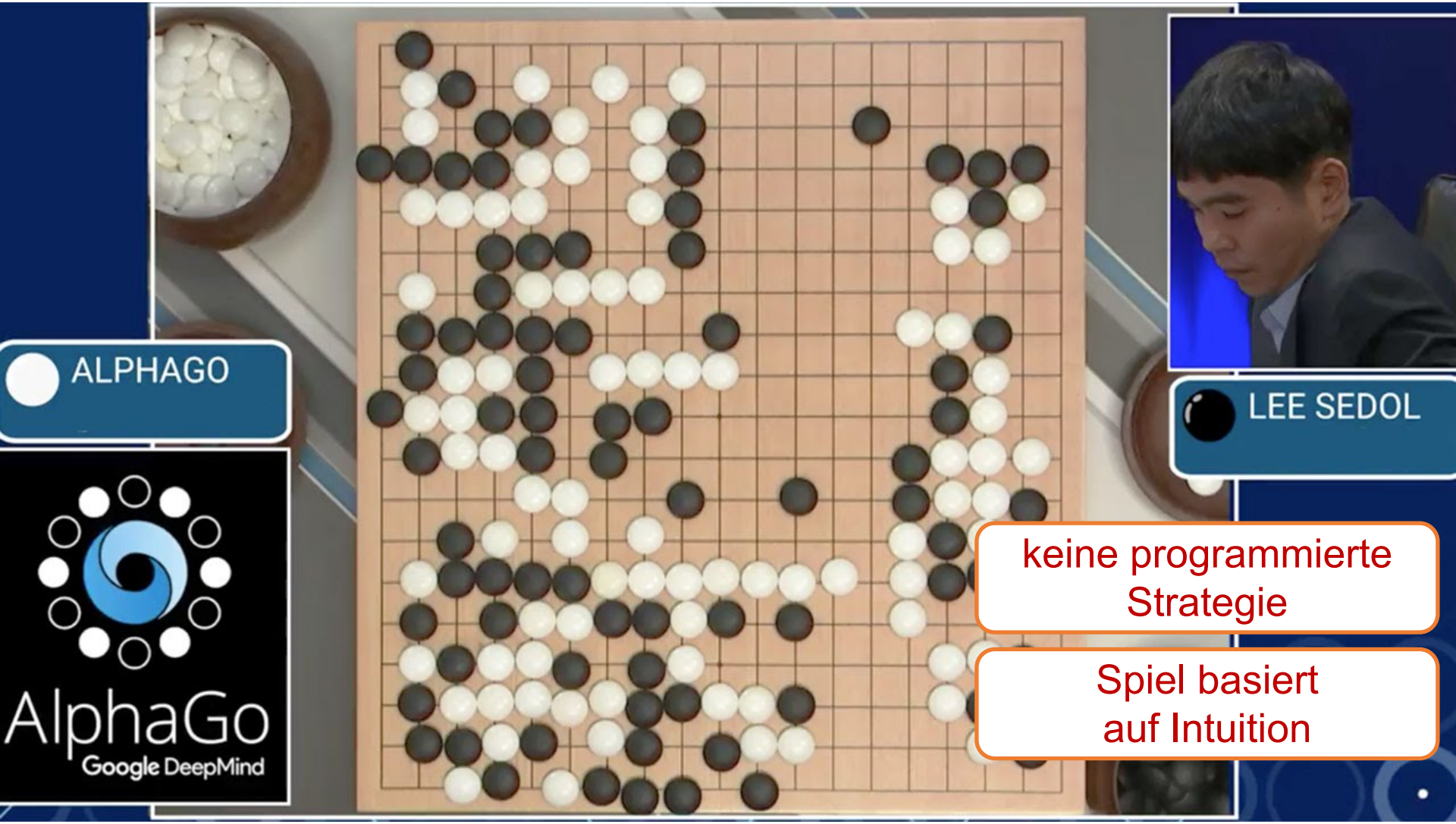
A pigeon pecking a key in an operant chamber.

from: Anderson, 2000

Atari Spielen Lernen

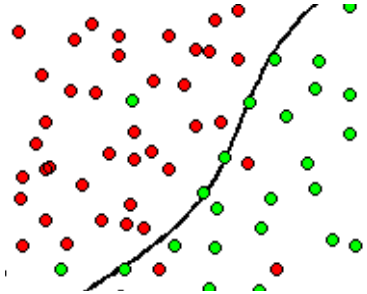
https://gym.openai.com/envs/#atari

AlphaGo (2016) / AlphaGo Zero (2017)

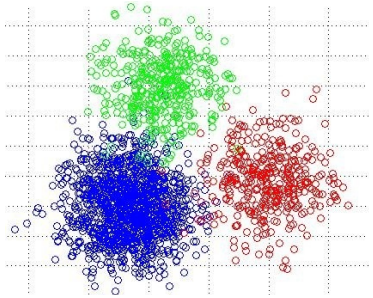


Dame spielen lernen

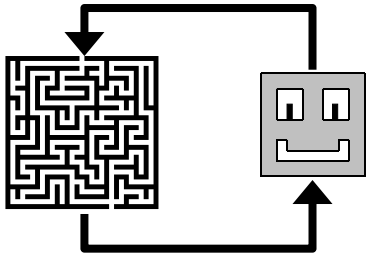
60



a) Überwachtes Lernen



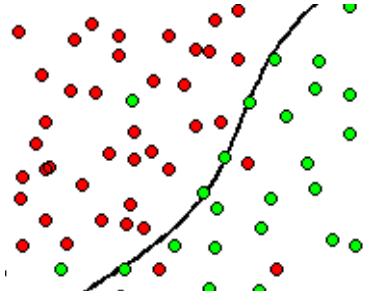
b) Unüberwachtes Lernen



c) Bestärkendes Lernen

Dame spielen lernen

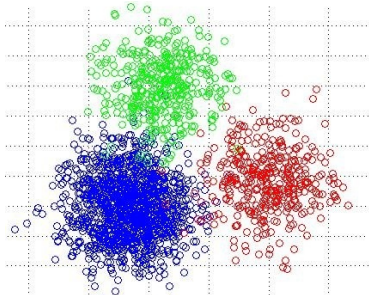
60



a) Überwachtes Lernen

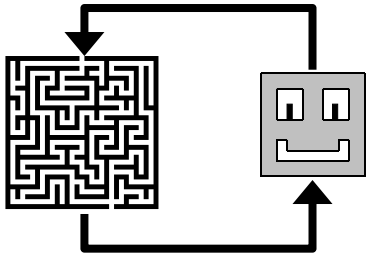
a) lerne goodness(board, move)*
b) lerne goodness(board)

*aus dem Buch “Lees’ Guide to Checkers”



b) Unüberwachtes Lernen

a) lerne, wie ein Board+Steine aussieht
b) lerne typische Züge



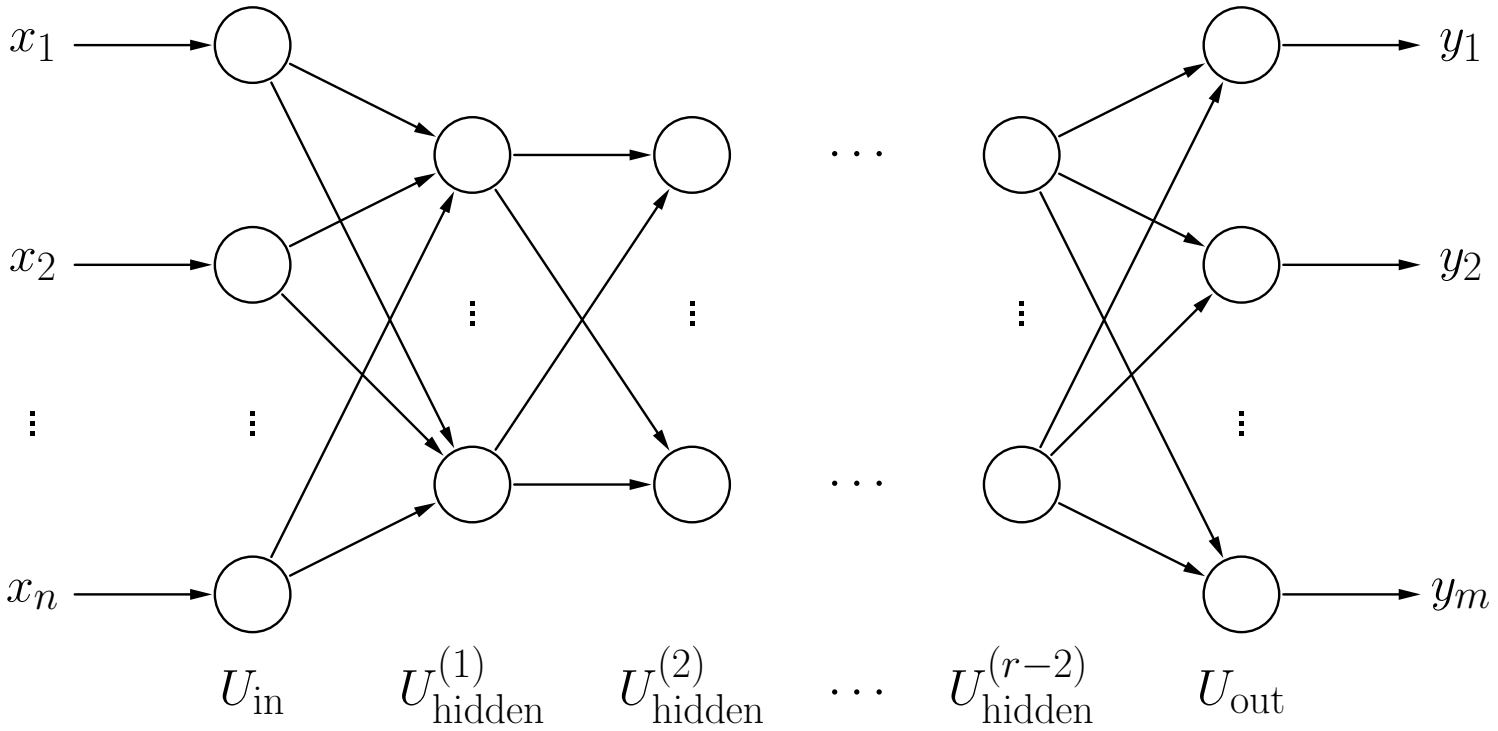
c) Bestärkendes Lernen

a) Spiele gegen Experten
b) Spiele gegen sich selbst

Mehrschichtge Perzeptren
(Multi-Layer Perceptrons, MLPs)

Mehrschichtige Perzeptren (MLPs)

Allgemeine Struktur eines mehrschichtigen Perzeptrons



Mehrschichtige Perzeptren (MLPs)

Ein **r-schichtiges Perzeptron** ist ein neuronales Netz mit einem Graph $G = (U, C)$ das die folgenden Bedingungen erfüllt:

(i) $U_{\text{in}} \cap U_{\text{out}} = \emptyset$,

(ii) $U_{\text{hidden}} = U_{\text{hidden}}^{(1)} \cup \cdots \cup U_{\text{hidden}}^{(r-2)}$,

$\forall 1 \leq i < j \leq r-2: \quad U_{\text{hidden}}^{(i)} \cap U_{\text{hidden}}^{(j)} = \emptyset$,

(iii) $C \subseteq \left(U_{\text{in}} \times U_{\text{hidden}}^{(1)}\right) \cup \left(\bigcup_{i=1}^{r-3} U_{\text{hidden}}^{(i)} \times U_{\text{hidden}}^{(i+1)}\right) \cup \left(U_{\text{hidden}}^{(r-2)} \times U_{\text{out}}\right)$

oder, falls keine versteckten Neuronen vorhanden ($r = 2, U_{\text{hidden}} = \emptyset$),

$C \subseteq U_{\text{in}} \times U_{\text{out}}$.

Vorwärtsgerichtetes Netz mit streng geschichteter Struktur.

Mehrschichtige Perzeptren (MLPs)

Die Netzwerkeingabe jedes versteckten Neurons und jedes Ausgabeneurons ist die **gewichtete Summe** seiner Eingaben, d.h.

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}: \qquad f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = \vec{w}_u \vec{\text{in}}_u = \sum_{v \in \text{pred}(u)} w_{uv} \, \text{out}_v \, .$$

Die Aktivierungsfunktion jedes versteckten Neurons ist eine sogenannte **Sigmoide**, d.h. eine monoton steigende Funktion

$$f : \mathbb{R} \to [0,1] \quad \text{with} \quad \lim_{x \to -\infty} f(x) = 0 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 1.$$

Die Aktivierungsfunktion jedes Ausgabeneurons ist ebenfalls entweder eine Sigmoide oder eine **lineare Funktion**, d.h.

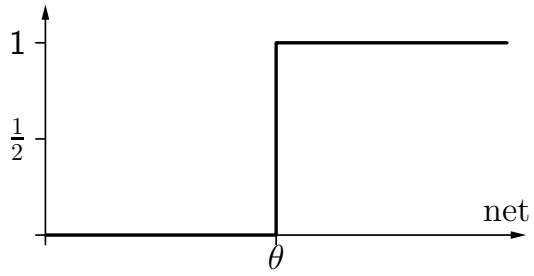
$$f_{\text{act}}(\text{net}, \theta) = \alpha \, \text{net} - \theta.$$

Nur die Stufenfunktion ist eine neurobiologisch plausible Aktivierungsfunktion.

Sigmoide Aktivierungsfunktionen

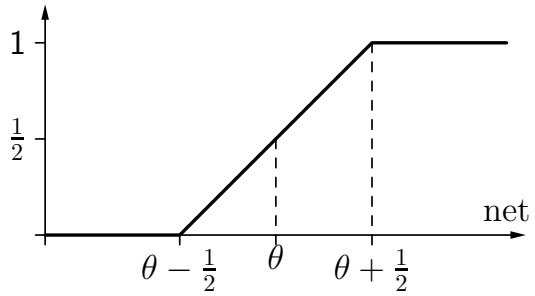
Stufenfunktion:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{falls net} \geq \theta, \\ 0, & \text{sonst.} \end{cases}$$



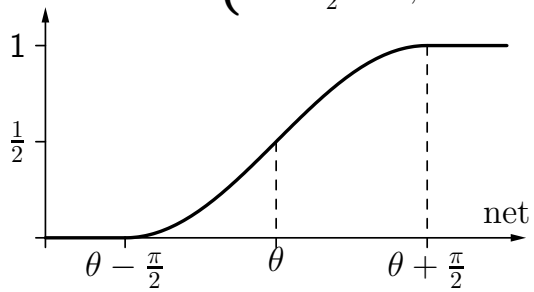
Semilineare Funktion:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{falls net} > \theta + \frac{1}{2}, \\ 0, & \text{falls net} < \theta - \frac{1}{2}, \\ (\text{net} - \theta) + \frac{1}{2}, & \text{sonst.} \end{cases}$$



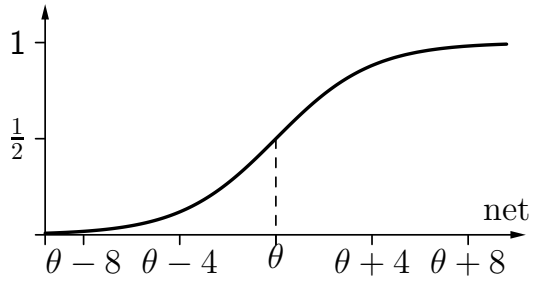
Sinus bis Sättigung:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{falls net} > \theta + \frac{\pi}{2}, \\ 0, & \text{falls net} < \theta - \frac{\pi}{2}, \\ \frac{\sin(\text{net} - \theta) + 1}{2}, & \text{sonst.} \end{cases}$$



Logistische Funktion:

$$f_{\text{act}}(\text{net}, \theta) = \frac{1}{1 + e^{-(\text{net} - \theta)}}$$



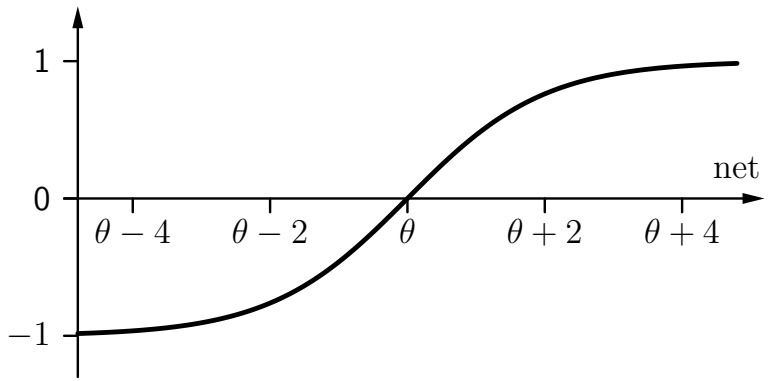
Sigmoide Aktivierungsfunktionen

Alle Sigmoiden auf der vorherigen Folie sind **unipolar**,
d.h. sie reichen von 0 bis 1.

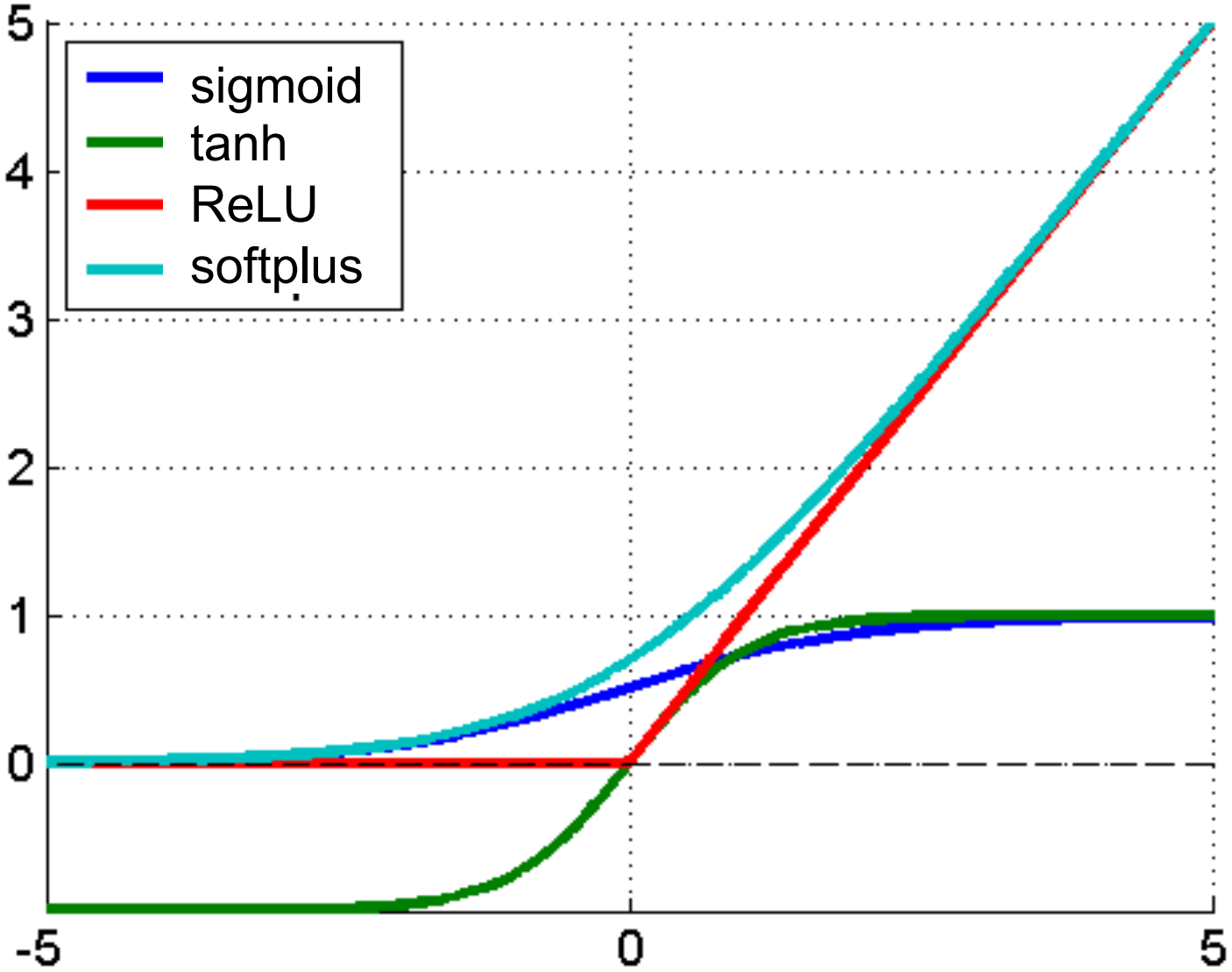
Manchmal werden **bipolare** sigmoidale Funktionen verwendet,
wie beispielsweise der tangens hyperbolicus.

tangens hyperbolicus:

$$f_{\text{act}}(\text{net}, \theta) = \tanh(\text{net} - \theta)$$
$$= \frac{2}{1 + e^{-2(\text{net} - \theta)}} - 1$$



Populäre Aktivierungsfunktionen



Softmax (eigentlich eher “softargmax”)

- zur Klassifikation mit mehreren Klassen
  - 1 Ausgabeneuron pro Klasse
  - betrachtet act aller (!) Ausgabeneuronen
  - interpretiert act als unnormalisierte bedingte log Wahrscheinlichkeit, d.h. z_i = log P’(y=i | x)

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Normalisierung (Summe = 1)

  - ergibt Wahrscheinlichkeitsverteilung der Klassen (im Ggs. zu Winner-take-all Prinzip)

Gewichtsmatrizen

Seien $U_1 = \{v_1, \ldots, v_m\}$ and $U_2 = \{u_1, \ldots, u_n\}$ die Neuronen zwei aufeinanderfolgender Schichten eines MLP.

Ihre Verbindungsgewichte werden dargestellt als eine $n \times m$-Matrix

$$\mathbf{W} = \begin{pmatrix} w_{u_1 v_1} & w_{u_1 v_2} & \ldots & w_{u_1 v_m} \\ w_{u_2 v_1} & w_{u_2 v_2} & \ldots & w_{u_2 v_m} \\ \vdots & \vdots & & \vdots \\ w_{u_n v_1} & w_{u_n v_2} & \ldots & w_{u_n v_m} \end{pmatrix},$$

wobei $w_{u_i v_j} = 0$, falls es keine Verbindung von Neuron v_j zu Neuron u_i gibt.

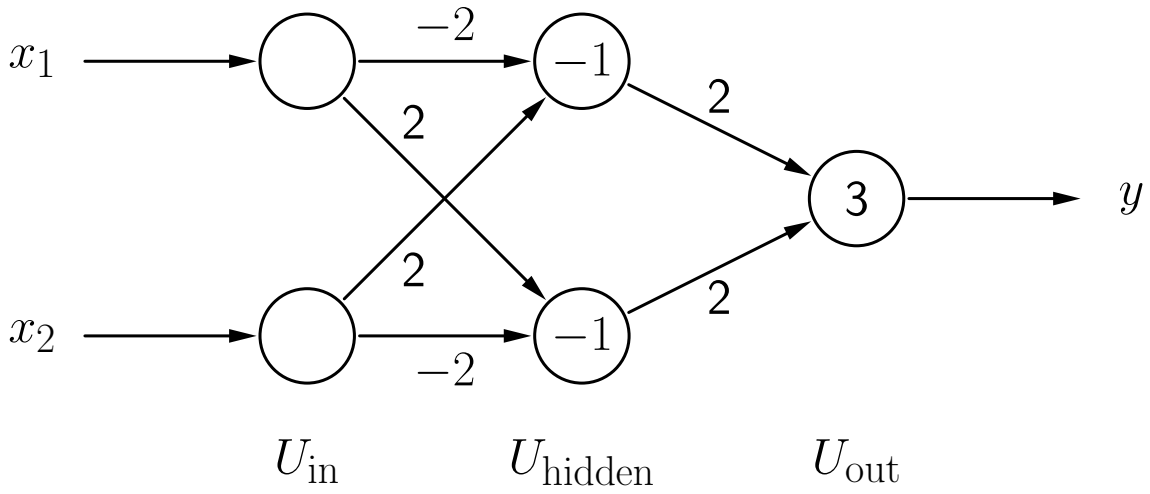
Vorteil: Die Berechnung der Netzwerkeingabe kann geschrieben werden als

$$\vec{\mathrm{net}}_{U_2} = \mathbf{W} \cdot \vec{\mathrm{in}}_{U_2} = \mathbf{W} \cdot \vec{\mathrm{out}}_{U_1}$$

wobei $\vec{\mathrm{net}}_{U_2} = (\mathrm{net}_{u_1}, \ldots, \mathrm{net}_{u_n})^\top$ und $\vec{\mathrm{in}}_{U_2} = \vec{\mathrm{out}}_{U_1} = (\mathrm{out}_{v_1}, \ldots, \mathrm{out}_{v_m})^\top$.

Biimplikation

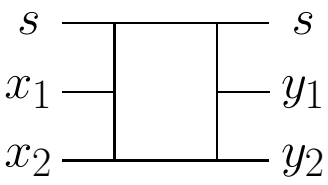
Lösen des Biimplikationsproblems mit einem MLP.

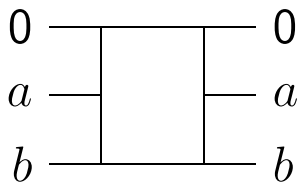


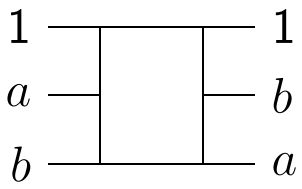
Man beachte die zusätzlichen Eingabeneuronen im Vergleich zur Lösung mit Schwellenwertelementen.

$$\mathbf{W}_1 = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix} \quad \text{und} \quad \mathbf{W}_2 = \begin{pmatrix} 2 & 2 \end{pmatrix}$$

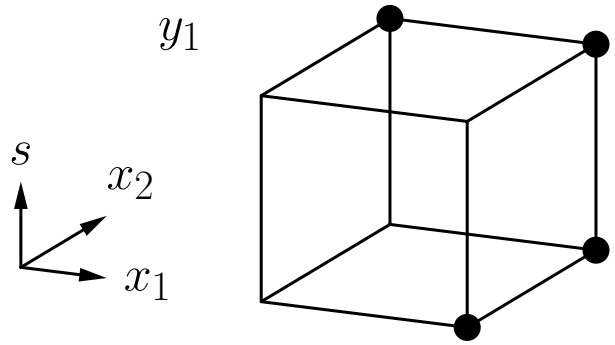
Fredkin-Gatter

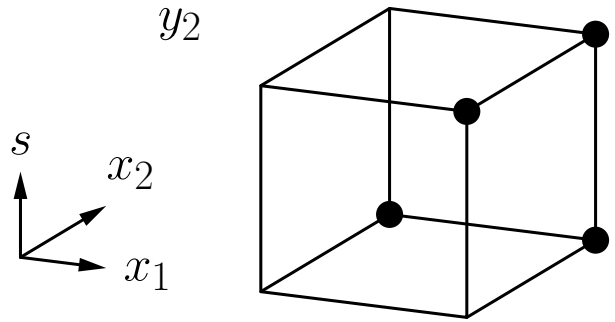




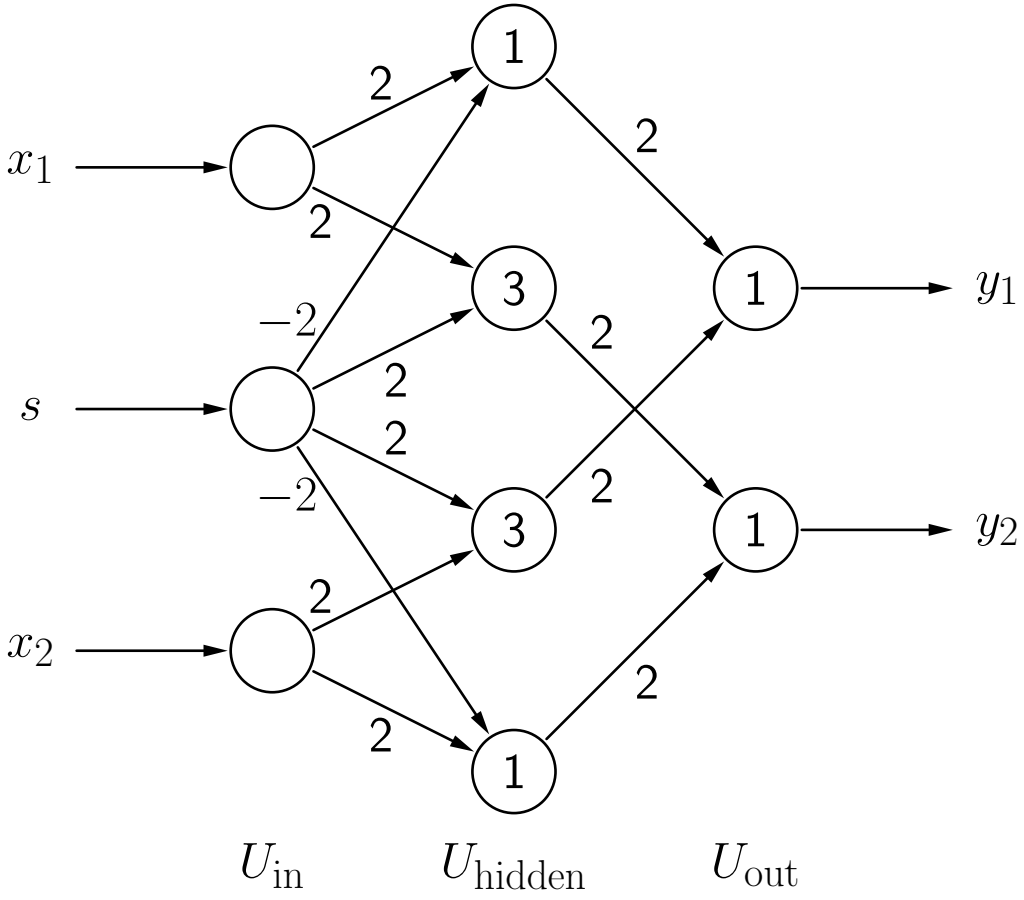


s	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1
x_2	0	1	0	1	0	1	0	1
y_1	0	0	1	1	0	1	0	1
y_2	0	1	0	1	0	0	1	1





Fredkin-Gatter



$$\mathbf{W}_1 = \begin{pmatrix} 2 & -2 & 0 \\ 2 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & -2 & 2 \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{pmatrix}$$

Warum nicht-lineare Aktivierungsfunktionen?

In Matrixschreibweise ergibt sich für zwei aufeinanderfolgende Schichten U_1 und U_2

$$\vec{\text{net}}_{U_2} = \mathbf{W} \cdot \vec{\text{in}}_{U_2} = \mathbf{W} \cdot \vec{\text{out}}_{U_1}.$$

Wenn die Aktivierungsfunktionen linear sind, d.h.

$$f_{\text{act}}(\text{net}, \theta) = \alpha \, \text{net} - \theta.$$

können die Aktivierungen der Neuronen in der Schicht U_2 wie folgt berechnet werden:

$$\vec{\text{act}}_{U_2} = \mathbf{D}_{\text{act}} \cdot \vec{\text{net}}_{U_2} - \vec{\theta},$$

wobei

- $\vec{\text{act}}_{U_2} = (\text{act}_{u_1}, \ldots, \text{act}_{u_n})^\top$ der Aktivierungsvektor ist,
- $\mathbf{D}_{\text{act}}$ eine $n \times n$ Diagonalmatrix aus den Faktorenα_{u_i}, $i = 1, \ldots, n$, ist und
- $\vec{\theta} = (\theta_{u_1}, \ldots, \theta_{u_n})^\top$ der Bias-Vektor ist.

Warum nicht-lineare Aktivierungsfunktionen?

Falls die Ausgabefunktion auch linear ist, gilt analog

$$\vec{\mathrm{out}}_{U_2} = \mathbf{D}_{\mathrm{out}} \cdot \vec{\mathrm{act}}_{U_2} - \vec{\xi},$$

wobei

$\vec{\mathrm{out}}_{U_2} = (\mathrm{out}_{u_1}, \ldots, \mathrm{out}_{u_n})^\top$ der Ausgabevektor ist,

$\mathbf{D}_{\mathrm{out}}$ wiederum eine $n \times n$ Diagonalmatrix aus Faktoren ist und

$\vec{\xi} = (\xi_{u_1}, \ldots, \xi_{u_n})^\top$ ein Biasvektor ist.

In Kombination ergibt sich

$$\vec{\mathrm{out}}_{U_2} = \mathbf{D}_{\mathrm{out}} \cdot \left(\mathbf{D}_{\mathrm{act}} \cdot \left(\mathbf{W} \cdot \vec{\mathrm{out}}_{U_1} \right) - \vec{\theta} \right) - \vec{\xi}$$

und daher

$$\vec{\mathrm{out}}_{U_2} = \mathbf{A}_{12} \cdot \vec{\mathrm{out}}_{U_1} + \vec{b}_{12}$$

mit einer $n \times m$-Matrix $\mathbf{A}_{12}$ und einem n-dimensionalen Vektor $\vec{b}_{12}$.

Warum nicht-lineare Aktivierungsfunktionen?

Daher ergibt sich

$$\vec{\text{out}}_{U_2} = \mathbf{A}_{12} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{12}$$

und

$$\vec{\text{out}}_{U_3} = \mathbf{A}_{23} \cdot \vec{\text{out}}_{U_2} + \vec{b}_{23}$$

für die Berechnungen zwei aufeinanderfolgender Schichten U_2 und U_3.

Diese beiden Berechnungen können kombiniert werden zu

$$\vec{\text{out}}_{U_3} = \mathbf{A}_{13} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{13},$$

wobei $\mathbf{A}_{13} = \mathbf{A}_{23} \cdot \mathbf{A}_{12}$ und $\vec{b}_{13} = \mathbf{A}_{23} \cdot \vec{b}_{12} + \vec{b}_{23}$.

Ergebnis: Mit linearen Aktivierungs- und Ausgabefunktionen können beliebige mehrschichtige Perzeptren auf zweischichtige Perzeptren reduziert werden.

Funktionsapproximation

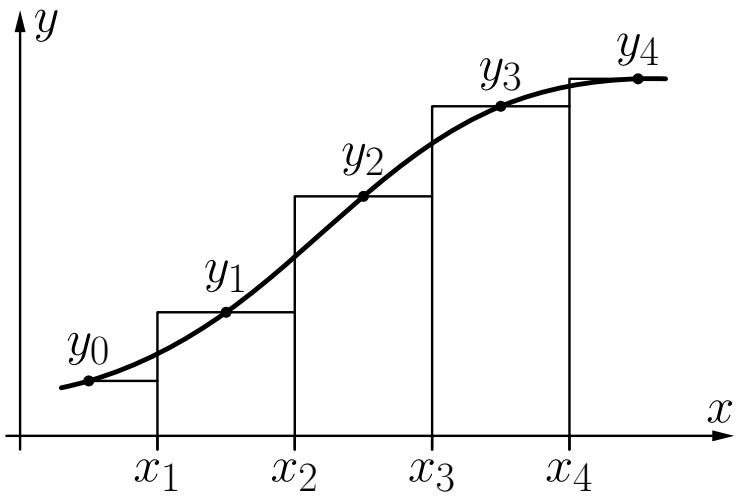
bisher: Erlernen von Boole'schen Funktionen $f : \{0,1\}^n \to \{0,1\}$

nun: Erlernen von reellwertigen Funktionen $f : \mathbb{R}^n \to \mathbb{R}$

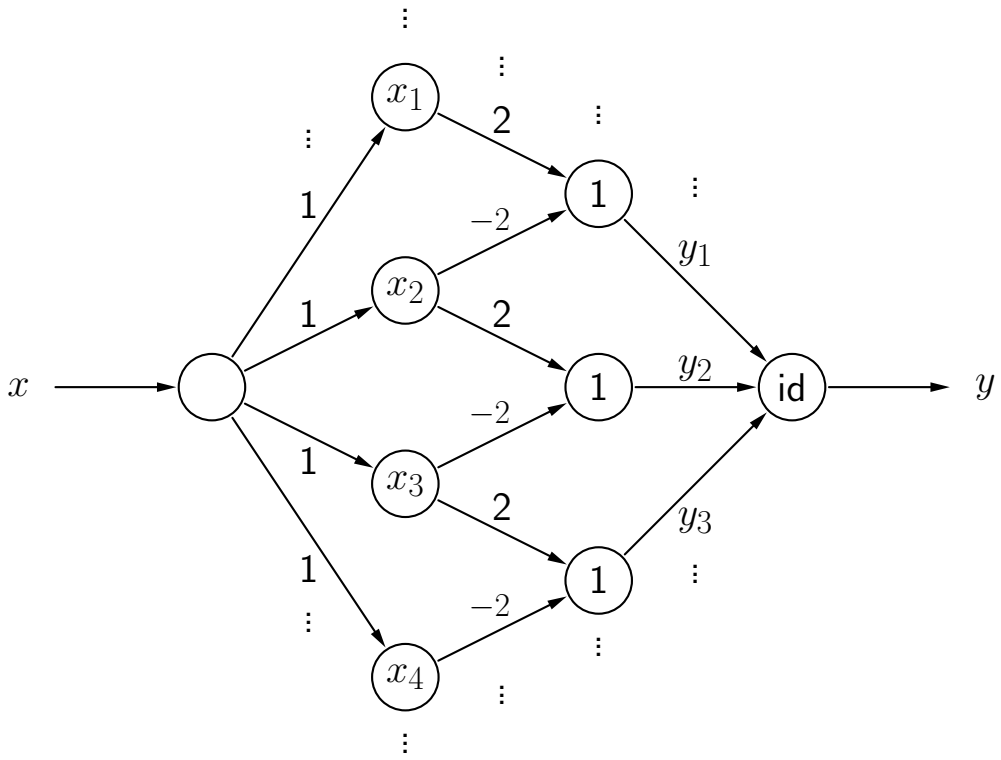
Idee der Funktionsapproximation

Nähere eine gegebene Funktion durch eine Stufenfunktion an.

Konstruiere ein neuronales Netz, das die Stufenfunktion berechnet.



Funktionsapproximation

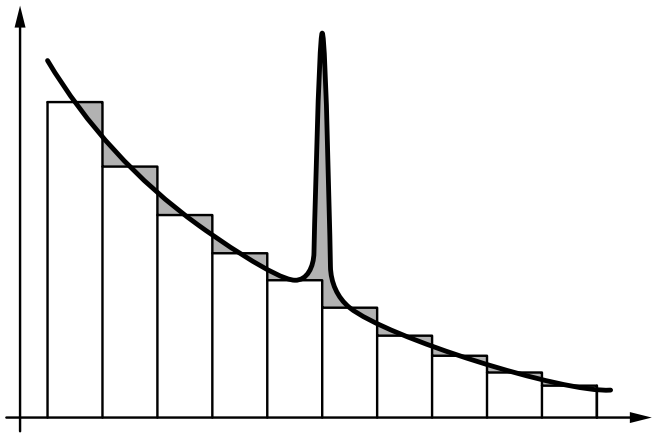


Ein neuronales Netz, das die Treppenfunktion von der vorherigen Folie berechnet. Es ist immer nur eine Stufe passend zum Eingabewert aktiv und die Stufenhöhe wird ausgegeben. Das Ausgabeneuron ist kein Schwellenwertelement: Seine Ausgabe ist die Identität der Neuroneingaben.

Funktionsapproximation

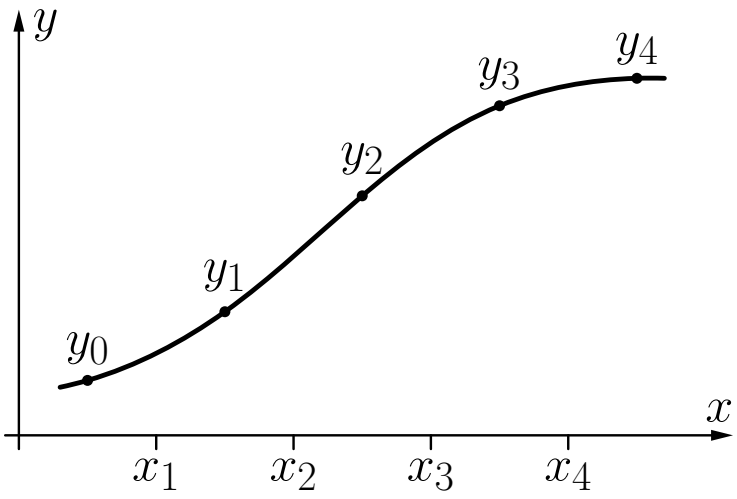
Theorem: Jede Riemann-integrierbare Funktion kann mit beliebiger Genauigkeit durch ein vierschichtes MLP berechnet werden.

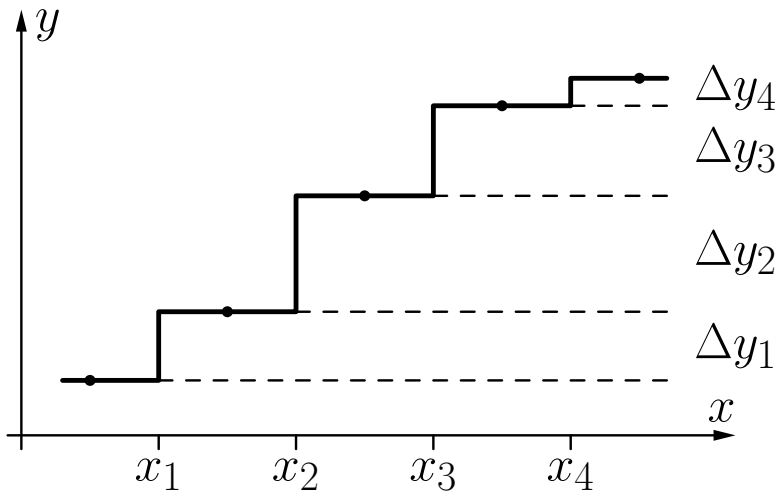
Aber: Fehler wird bestimmt als die **Fläche** zwischen Funktionen.

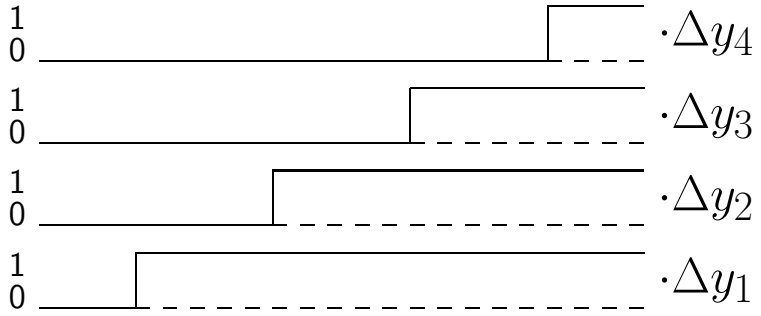


Weitere mathematische Untersuchungen zeigen, dass sogar gilt: Mit einem dreischichtigen Perzeptron kann jede stetige Funktion mit beliebiger Genauigkeit angenähert werden (Fehlerbestimmung: maximale Differenz der Funktionswerte).

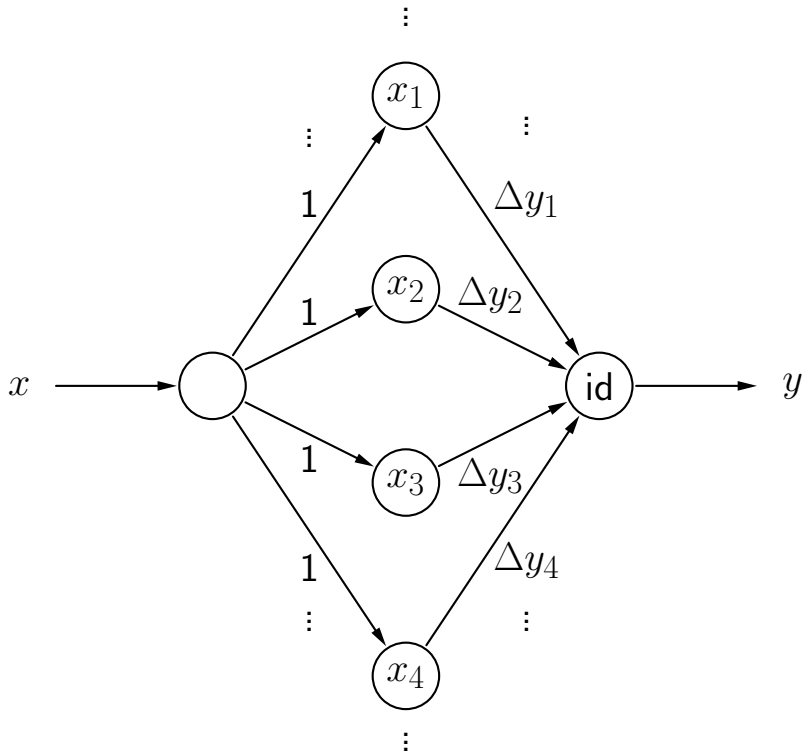
Funktionsapproximation





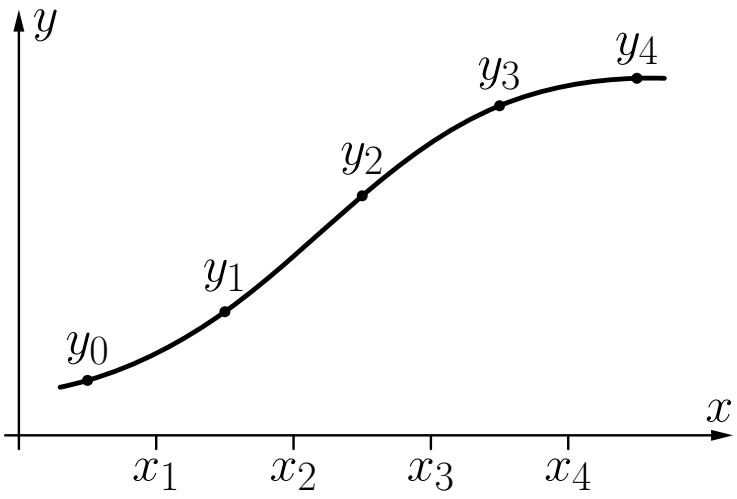


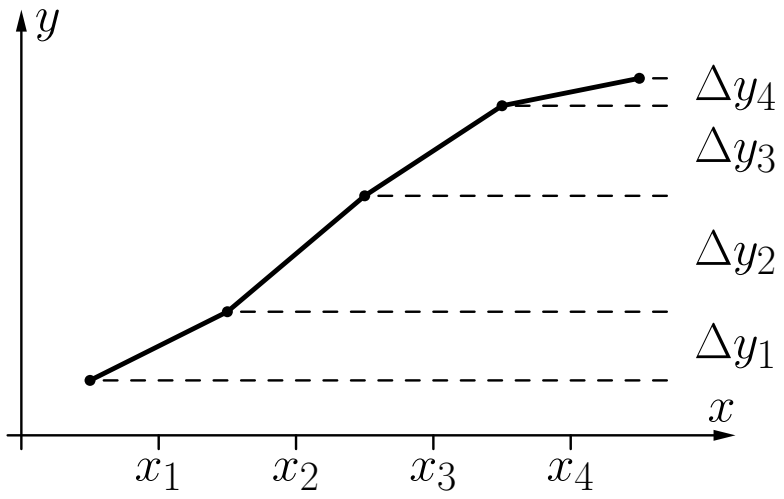
Funktionsapproximation

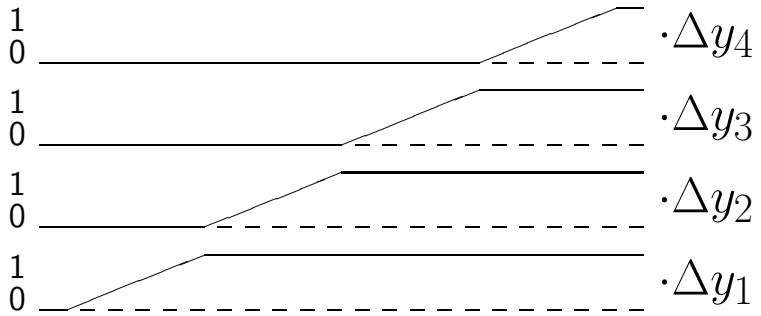


Ein neuronales Netz, das die Treppenfunktion von der vorherigen Folie als gewichtete Summe von Sprungfunktionen berechnet.

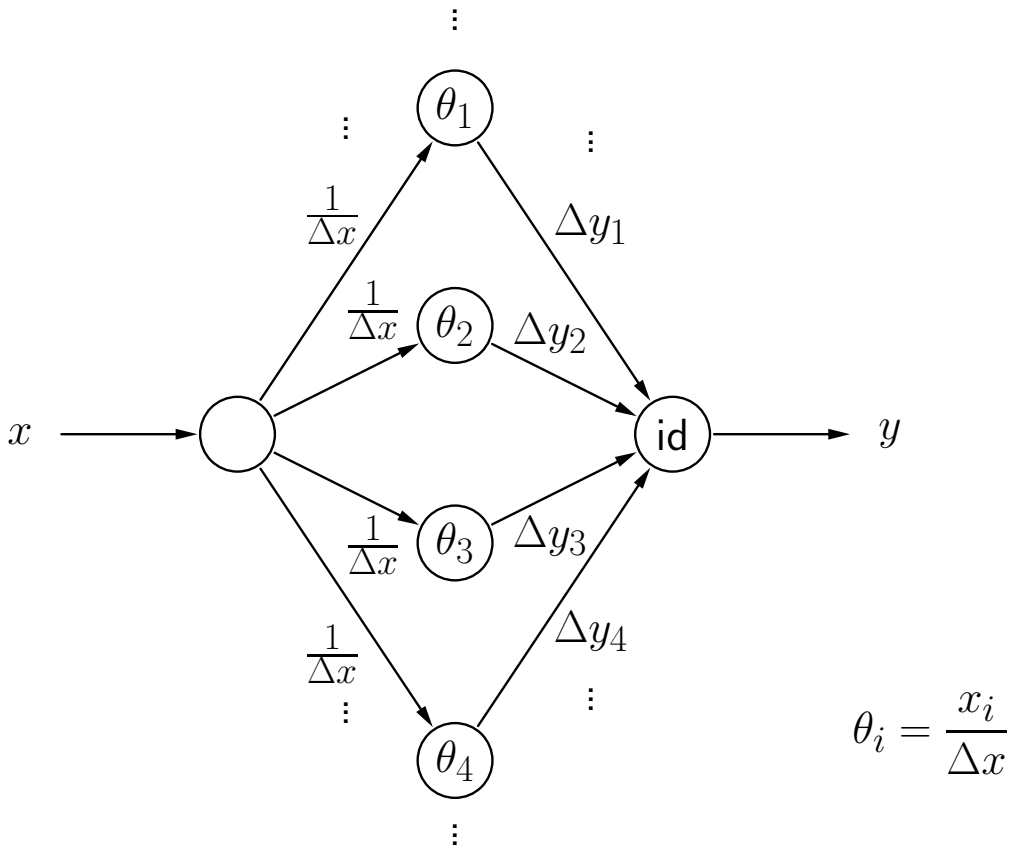
Funktionsapproximation







Funktionsapproximation



Ein neuronales Netz, das die stückweise lineare Funktion von der vorherigen Folie durch eine gewichtete Summe von semi-linearen Funktionen berechnet, wobei $\Delta x = x_{i+1} - x_i$.

Trainieren von MLPs

Gradientenabstieg

Allgemeinerer Ansatz:**Gradientenabstieg**.

Notwendige Bedingung: **differenzierbare Aktivierungs- und Ausgabefunktionen**.

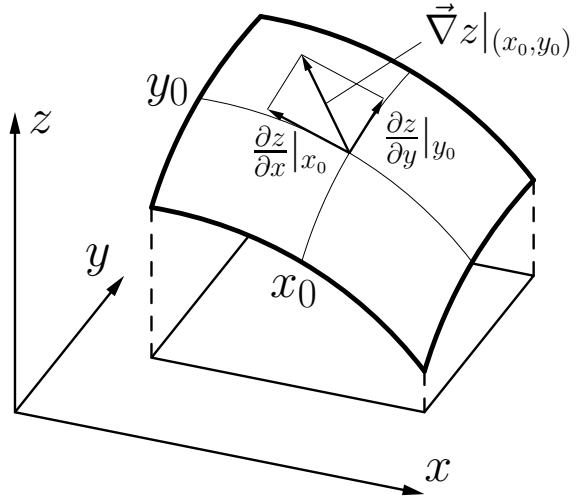
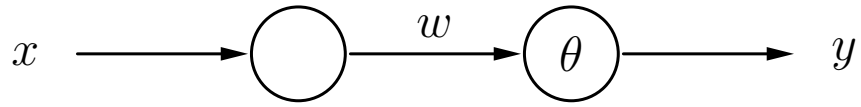


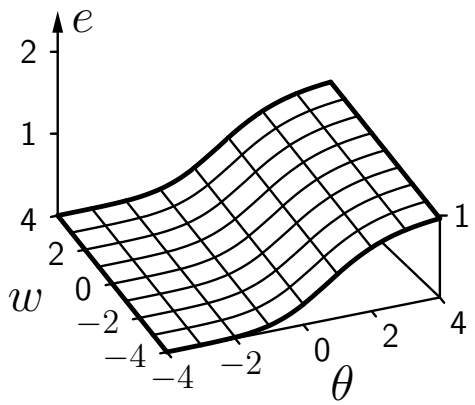
Illustration des Gradienten einer reellwertigen Funktion $z = f(x, y)$ am Punkt (x_0, y_0). Dabei ist $\vec{\nabla} z|_{(x_0,y_0)} = \left(\frac{\partial z}{\partial x}|_{x_0}, \frac{\partial z}{\partial y}|_{y_0}\right)$.

Gradientenabstieg: Beispiele

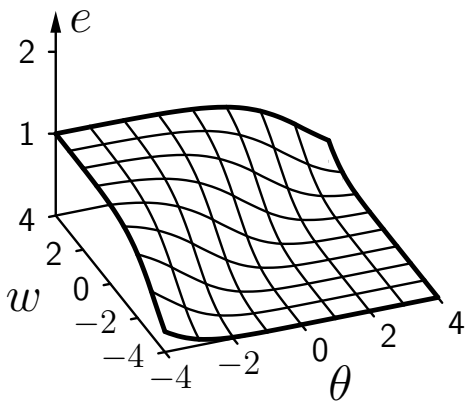
Gradientenabstieg für die Negation $\neg x$



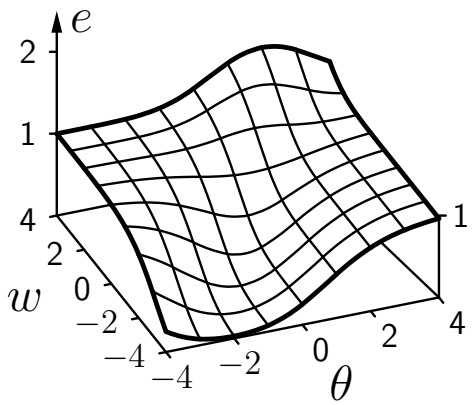
x	y
0	1
1	0



Fehler für $x = 0$



Fehler für $x = 1$



Summe der Fehler

Gradientenabstieg: Beispiele

Epoche	θ	w	Fehler
0	3.00	3.50	1.307
20	3.77	2.19	0.986
40	3.71	1.81	0.970
60	3.50	1.53	0.958
80	3.15	1.24	0.937
100	2.57	0.88	0.890
120	1.48	0.25	0.725
140	-0.06	-0.98	0.331
160	-0.80	-2.07	0.149
180	-1.19	-2.74	0.087
200	-1.44	-3.20	0.059
220	-1.62	-3.54	0.044

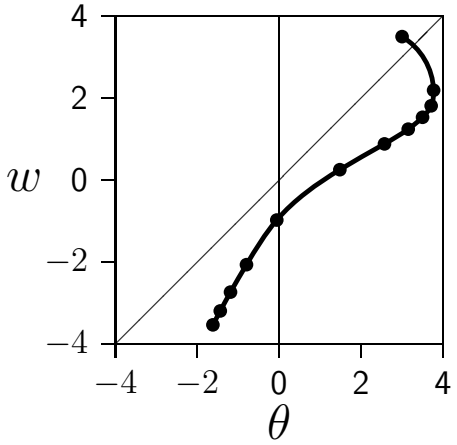
Online-Training

Epoche	θ	w	Fehler
0	3.00	3.50	1.295
20	3.76	2.20	0.985
40	3.70	1.82	0.970
60	3.48	1.53	0.957
80	3.11	1.25	0.934
100	2.49	0.88	0.880
120	1.27	0.22	0.676
140	-0.21	-1.04	0.292
160	-0.86	-2.08	0.140
180	-1.21	-2.74	0.084
200	-1.45	-3.19	0.058
220	-1.63	-3.53	0.044

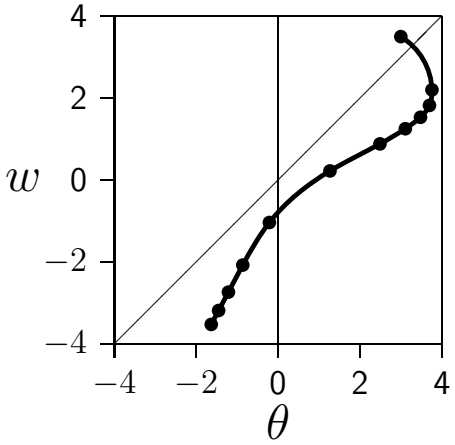
Batch-Training

Gradientenabstieg: Beispiele

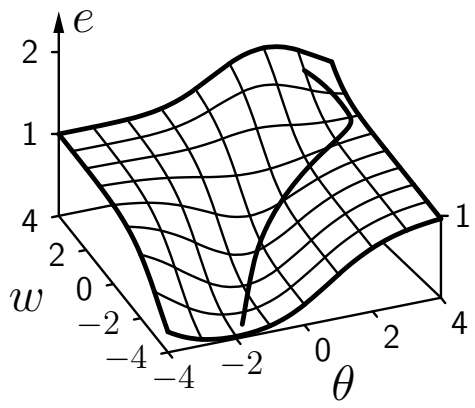
Visualisierung des Gradientenabstiegs für die Negation $\neg x$



Online-Training



Batch-Training

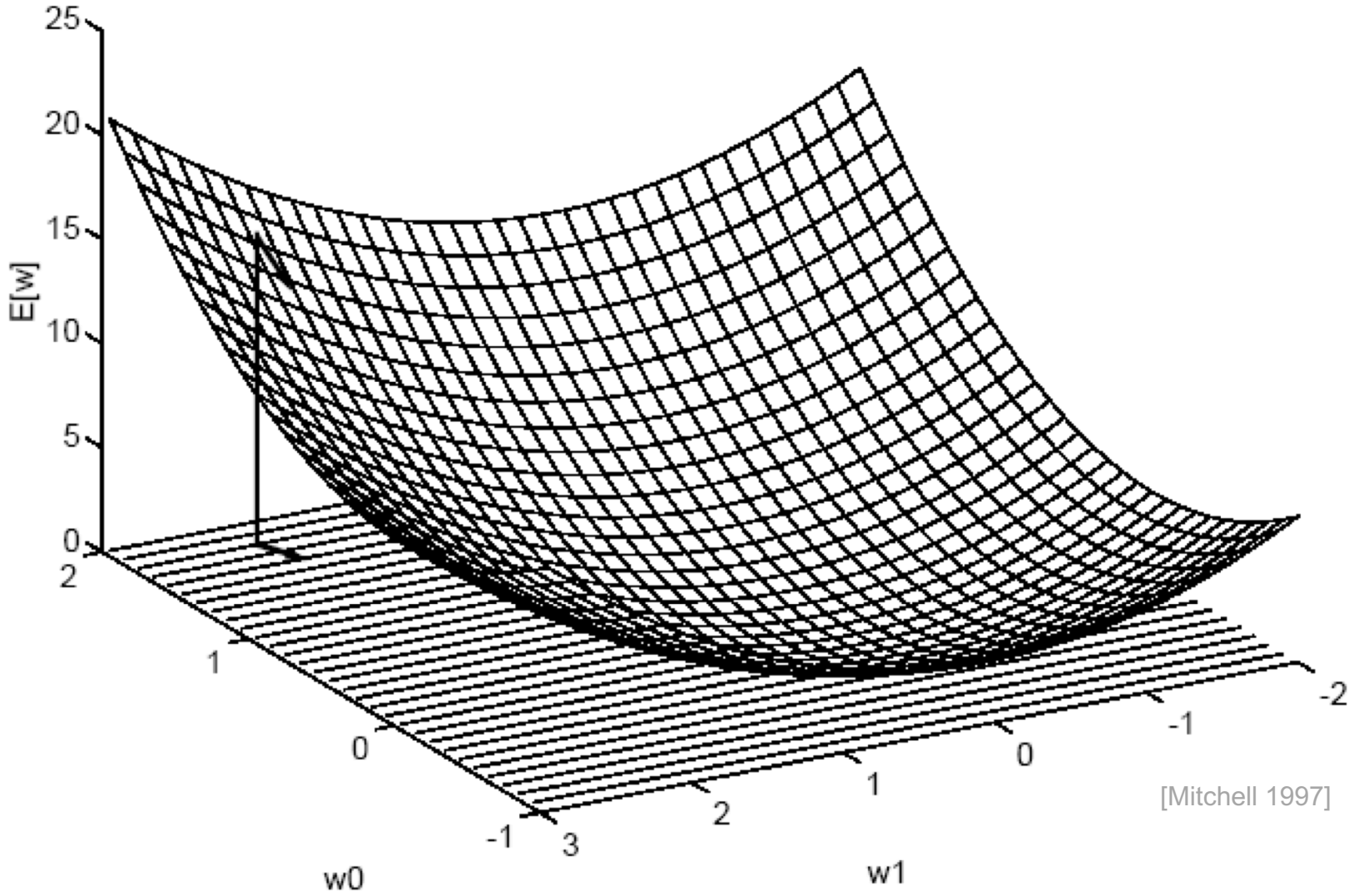


Batch-Training

Das Training ist offensichtlich erfolgreich.

Der Fehler kann nicht vollständig verschwinden, bedingt durch die Eigenschaften der logistischen Funktion.

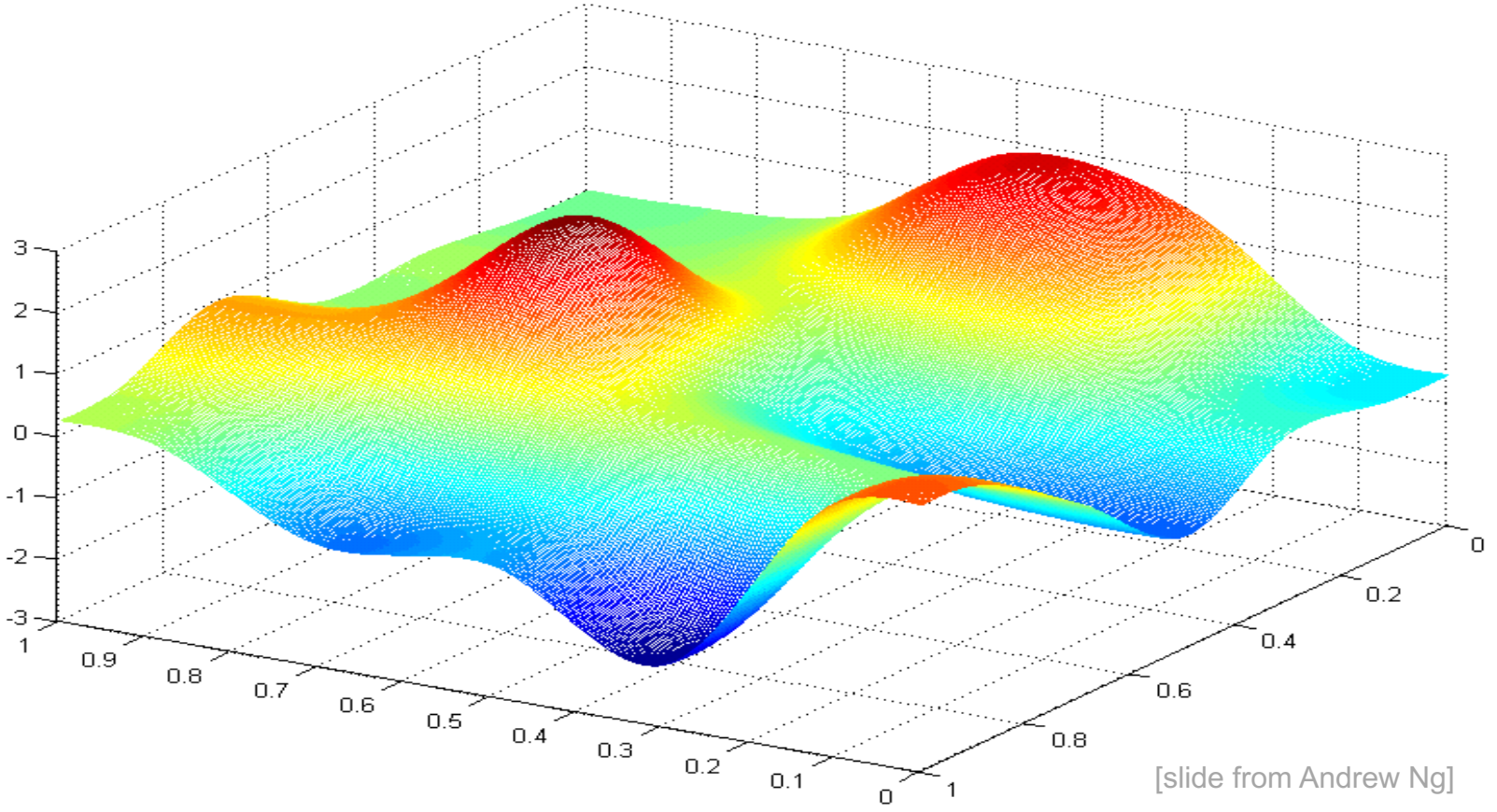
Gradientenabstieg (konvex)



[Mitchell 1997]

Gradientenabstieg

realistischeres Fehlergebirge (für 2 Gewichte)



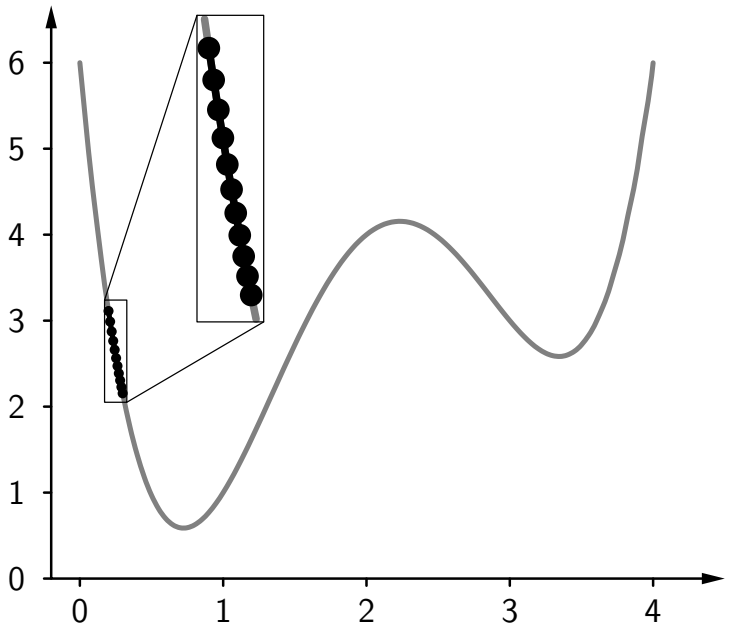
[slide from Andrew Ng]

=> viele lokale Minima!

Gradientenabstieg: Beispiele

Beispielfunktion: $$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	−11.147	0.011
1	0.211	2.990	−10.811	0.011
2	0.222	2.874	−10.490	0.010
3	0.232	2.766	−10.182	0.010
4	0.243	2.664	−9.888	0.010
5	0.253	2.568	−9.606	0.010
6	0.262	2.477	−9.335	0.009
7	0.271	2.391	−9.075	0.009
8	0.281	2.309	−8.825	0.009
9	0.289	2.233	−8.585	0.009
10	0.298	2.160		

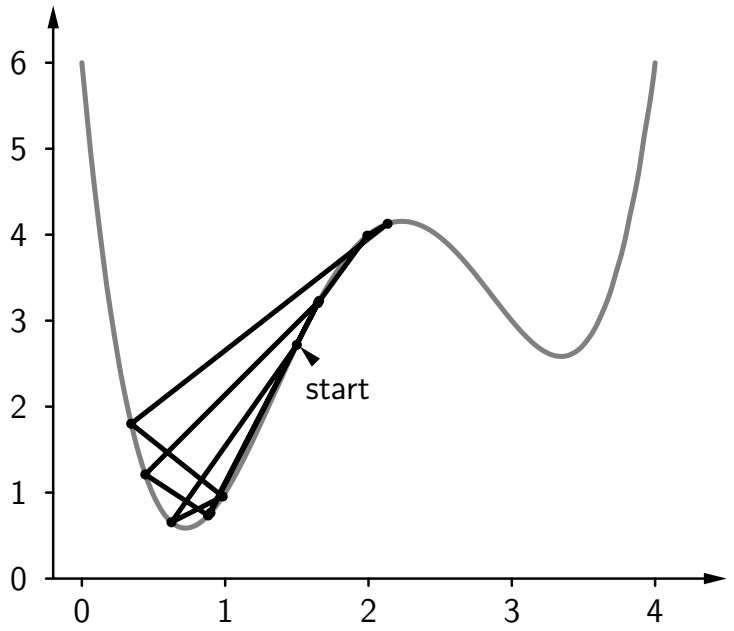


Gradientenabstieg mit Startwert 0.2 und Lernrate 0.001.

Gradientenabstieg: Beispiele

Beispielfunktion: $$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1.500	2.719	3.500	−0.875
1	0.625	0.655	−1.431	0.358
2	0.983	0.955	2.554	−0.639
3	0.344	1.801	−7.157	1.789
4	2.134	4.127	0.567	−0.142
5	1.992	3.989	1.380	−0.345
6	1.647	3.203	3.063	−0.766
7	0.881	0.734	1.753	−0.438
8	0.443	1.211	−4.851	1.213
9	1.656	3.231	3.029	−0.757
10	0.898	0.766		

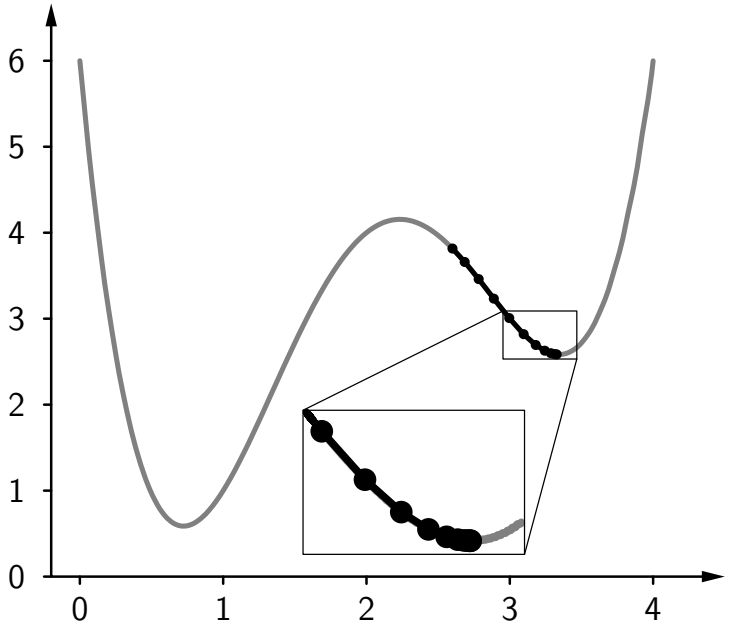


Gradientenabstieg mit Startwert 1.5 und Lernrate 0.25.

Gradientenabstieg: Beispiele

Beispielfunktion: $$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

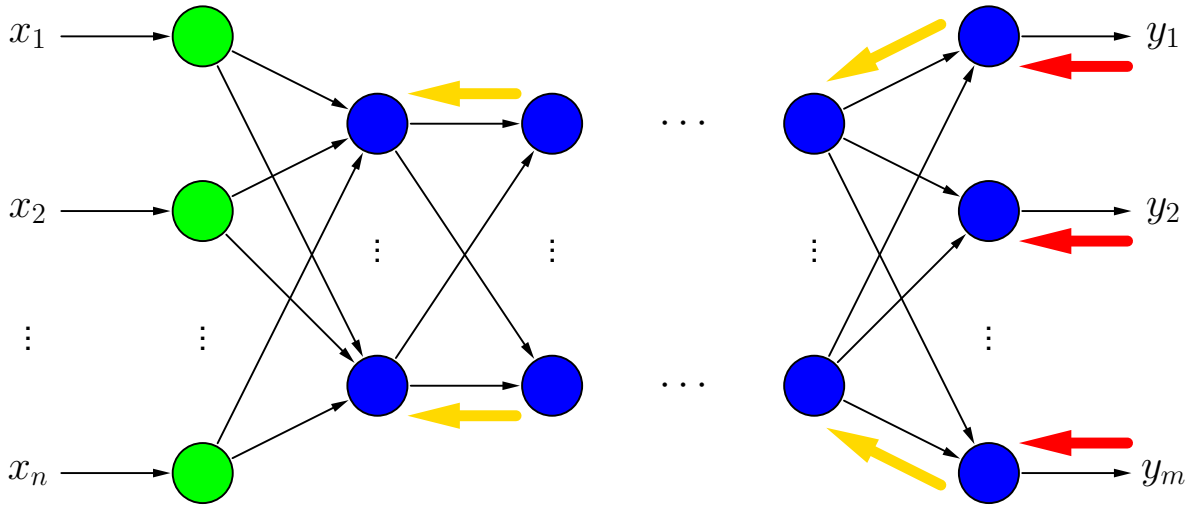
i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	2.600	3.816	−1.707	0.085
1	2.685	3.660	−1.947	0.097
2	2.783	3.461	−2.116	0.106
3	2.888	3.233	−2.153	0.108
4	2.996	3.008	−2.009	0.100
5	3.097	2.820	−1.688	0.084
6	3.181	2.695	−1.263	0.063
7	3.244	2.628	−0.845	0.042
8	3.286	2.599	−0.515	0.026
9	3.312	2.589	−0.293	0.015
10	3.327	2.585		



Gradientenabstieg mit Startwert 2.6 und Lernrate 0.05.

Error Backpropagation

Setzen der Eingabe Vorwärtsweitergabe der Eingabe



Fehlerrückübertragung Fehlerbestimmung

Error Backpropagation

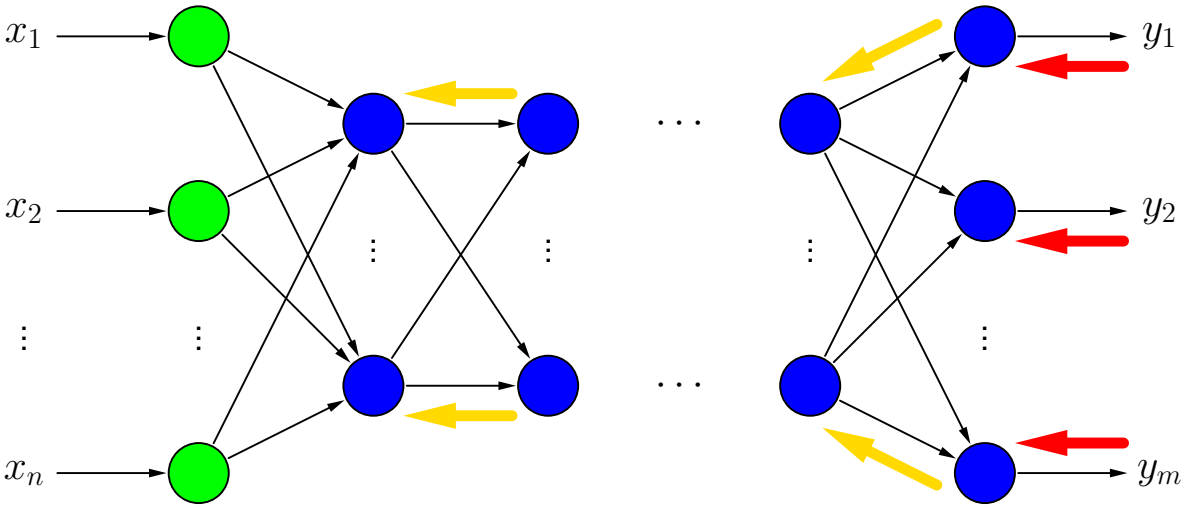
Aktivierungsfunktion: logistisch
Ausgabefunktion: Identität
impliziter Biaswert

$$\forall u \in U_{\text{in}}: \quad \text{out}_u^{(l)} = i_u^{(l)}$$

Setzen der Eingabe

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}: \quad \text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$$

Vorwärtsweitergabe der Eingabe



Fehlerrückübertragung

Fehlerbestimmung

Rückwärts-propagation:

$$\forall u \in U_{\text{hidden}}: \quad \delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$$

$$\forall u \in U_{\text{out}}: \quad \delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$$

Aktivierungs-ableitung:

$$\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$$

Gewichts-änderung:

$$\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$$

Algorithmus-Skizze (online)

gegeben: MLP mit $G = (U, C)$, Lernrate η, Trainingsbeispiele L_{fixed}

Initialisierung aller Gewichte (Zufallswerte)

wiederhole:

für jedes Trainingsbeispiel $l = (\vec{\imath}^{(l)}, \vec{o}^{(l)}) \in L_{\text{fixed}}$

Eingabe, Vorwärtsberechnung der Aktivierungen und Ausgabe:

$$\forall u \in U_{\text{in}}: \quad \text{out}_u^{(l)} = i_u^{(l)}$$

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}: \quad \text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$$

Fehlerberechnung und Rückübertragung (Backpropagation):

$$\forall u \in U_{\text{out}}: \quad \delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$$

$$\forall u \in U_{\text{hidden}}: \quad \delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$$

mit Ableitung der Aktivierungsfunktion $\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$

Berechnung der Gewichtsänderung $\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$ und Update

bis Stopkriterium erreicht