
Einführung in Web- und Data-Science

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Tanya Braun (Übungen)

Übersicht

- Einführung, Klassifikation vs. Regression, parametrisches und nicht-parametrisches überwachtes Lernen
- **Neuronale Netze und Support-Vektor-Maschinen**
- Häufungsanalysen, Warenkorbanalyse, Empfehlungen
- Statistische Grundlagen: Stichproben, Schätzer, Verteilung, Dichte, kumulative Verteilung, Skalen: Nominal-, Ordinal-, Intervall- und Verhältnisskala, Hypothesentests, Konfidenzintervalle, Reliabilität, Interne Konsistenz, Cronbach Alpha, Trennschärfe
- Bayessche Statistik, Bayessche Netze zur Spezifikation von diskreten Verteilungen, Anfragen, Anfragebeantwortung, Lernverfahren für Bayessche Netze
- Induktives Lernen: Versionsraum, Informationstheorie, Entscheidungsbäume, Lernen von Regeln
- Ensemble-Methoden, Bagging, Boosting, Random Forests
- Clusterbildung, K-Means, Analyse der Variation (Analysis of Variation, ANOVA), Inter-Cluster-Variation, Intra-Cluster-Variation, F-Statistik, Bonferroni-Korrektur, MANOVA, Discriminant Function Analysis
- Analyse Sozialer Strukturen

Repräsentation von Funktionen ...

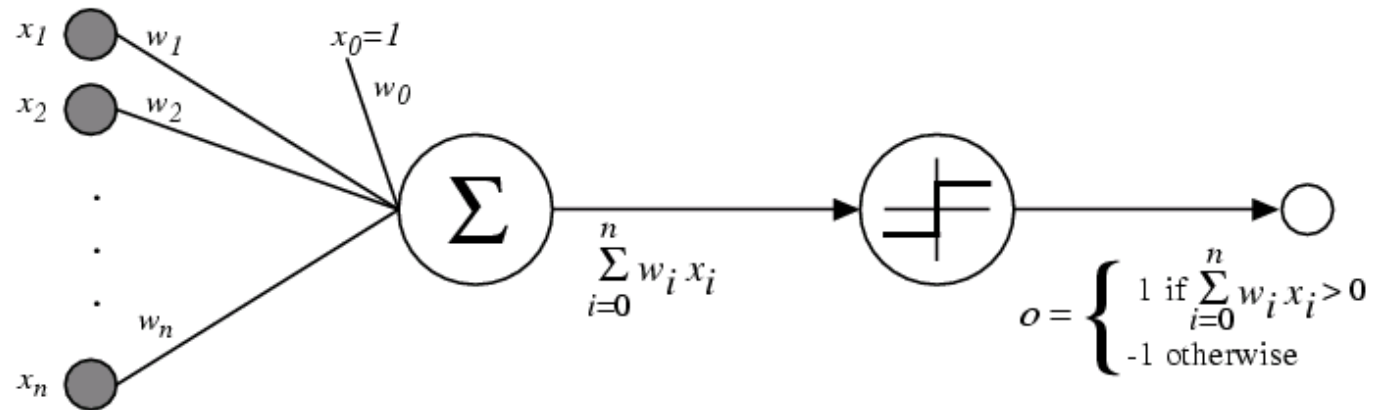
- ... durch Perzeptrons
 - Ein-Ebenen-Netzwerk (Linearer Klassifikator)
 - Mehrebenen-Netzwerke
 - Lernregel Fehlerrückführung (Backpropagation)

Frank Rosenblatt, The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory, 1957

- ... durch Support-Vektor-Maschinen (SVMs)
 - Unterteilt einer Menge von Objekten in Klassen, so dass um die Klassengrenzen herum ein möglichst breiter Bereich frei von Objekten bleibt
 - Geeignet für Regression und Klassifikation
 - Nichtlineare Klassifikation durch Transformation des Eingaberaums

V. Vapnik, A. Chervonenkis, A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964

Perzeptron

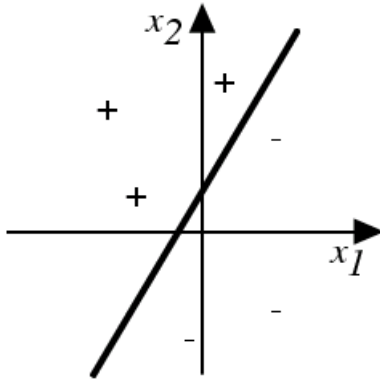


$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{sonst} \end{cases}$$

Manchmal einfacher geschrieben als (Ann.: $x_0 = 1$):

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{sonst} \end{cases}$$

Entscheidungslinie eines Perzeptrons



Repräsentiert lineare Funktion $y = mx + b$

- Was machen die Gewichte?
 $g(x_1, x_2) = AND(x_1, x_2)$?

Verallgemeinerung auf Entscheidungsebenen möglich

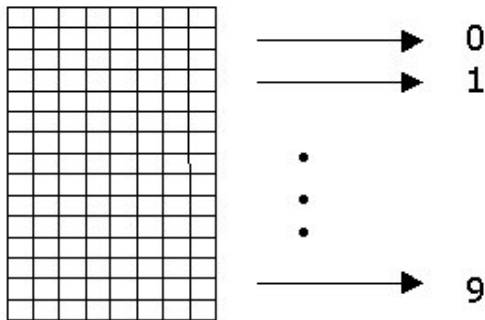
Vgl.: Warren McCulloch und William Pitts: *A logical calculus of the ideas immanent in nervous activity*. In: *Bulletin of Mathematical Biophysics*, Bd. 5, S. 115–133, **1943**

Lassen sich alle Funktionen repräsentieren?

- Kann die Fehlerfunktion immer sinnvoll minimiert werden?
- XOR-Problem
 - Einführung weiterer Dimensionen
 - Erweiterung der Daten?
 - Besser: Einführung weiterer Ebenen im Netz

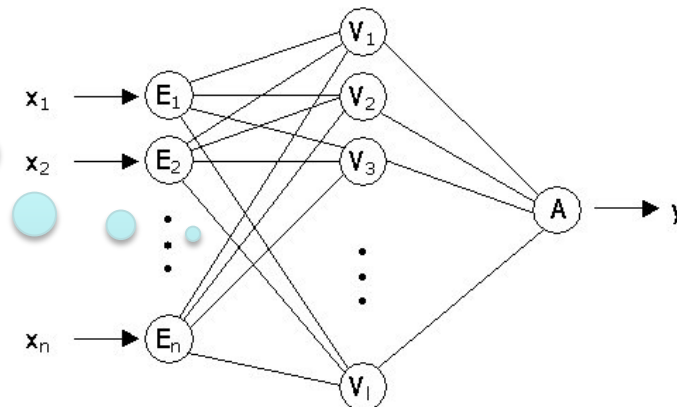
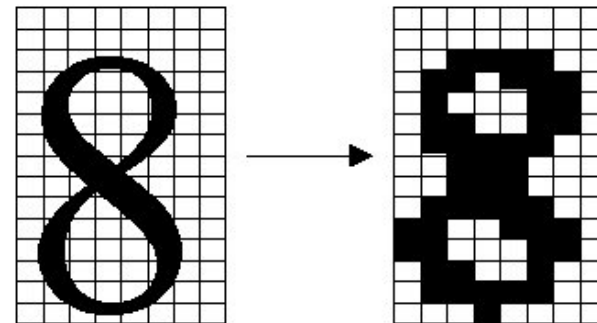
Ein Anwendungsbeispiel

Das folgende Netz soll Ziffern von 0 bis 9 erkennen. Dafür wird zunächst das *Eingabefeld* in 8x15 Elemente aufgeteilt:

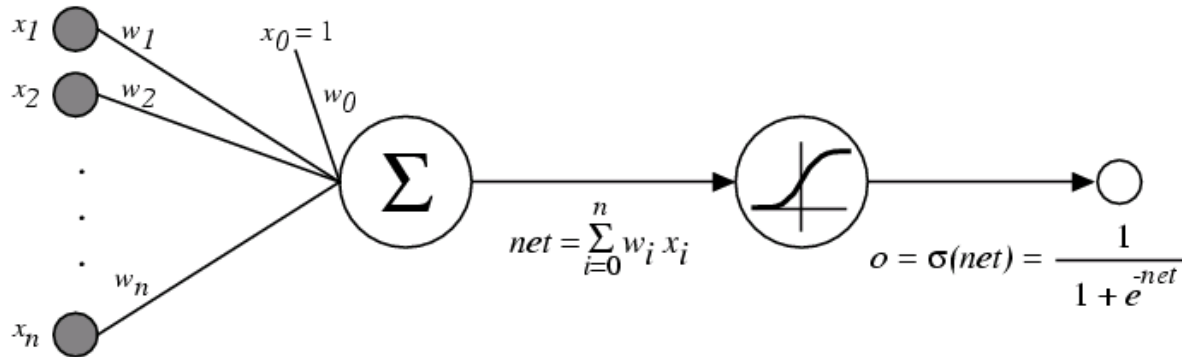


Wie können wir die Parameter automatisch so bestimmen, dass geschriebene Ziffern erkannt und als Ausgabe y ausgegeben werden?

Die geschriebene Ziffer wird in eine Folge von Nullen und Einsen umgewandelt, wobei 0 für leere und 1 für übermalte Rasterpunkte steht:



Verwendung kontinuierlicher Funktionen



$\sigma(x)$ ist die Sigmoid-Funktion

$$\frac{1}{1 + e^{-x}}$$

-0.06

W1

-2.5

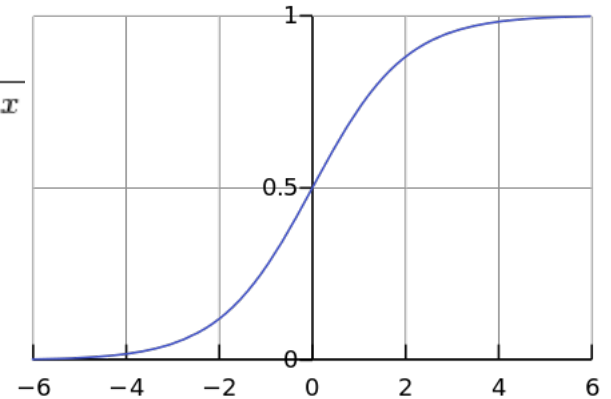
W2

W3

1.4

$f(x)$

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



-0.06

2.7

-2.5

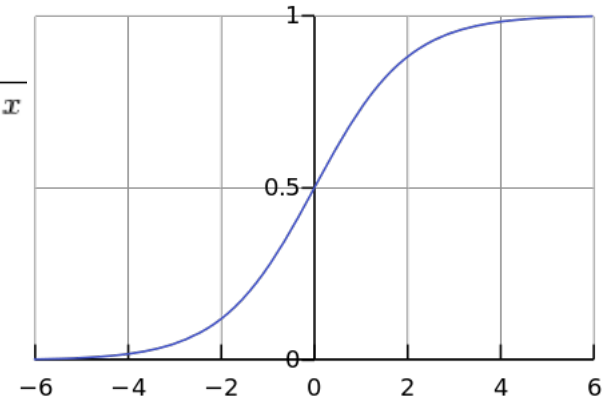
-8.6

0.002

1.4

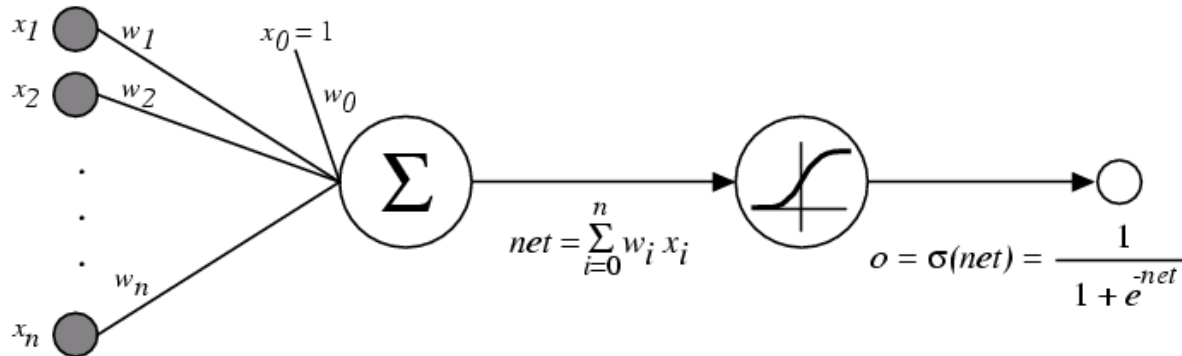
$f(x)$

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



$$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$$

Sigmoid-Einheit



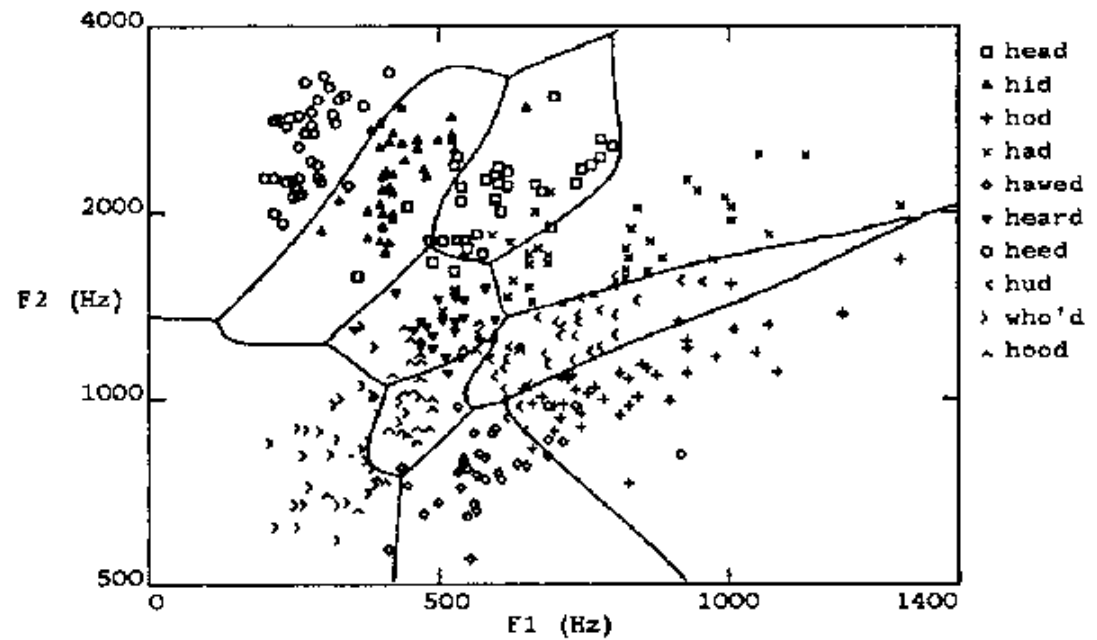
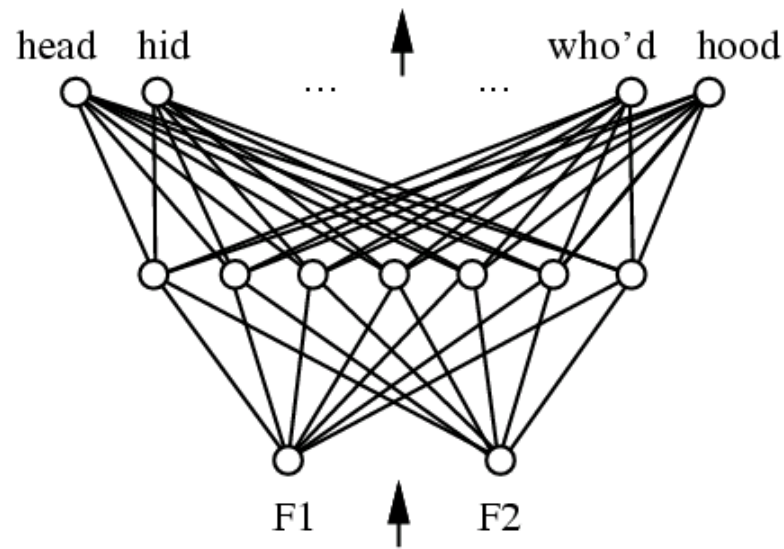
$\sigma(x)$ ist die Sigmoid-Funktion (auch: logistische Funktion)

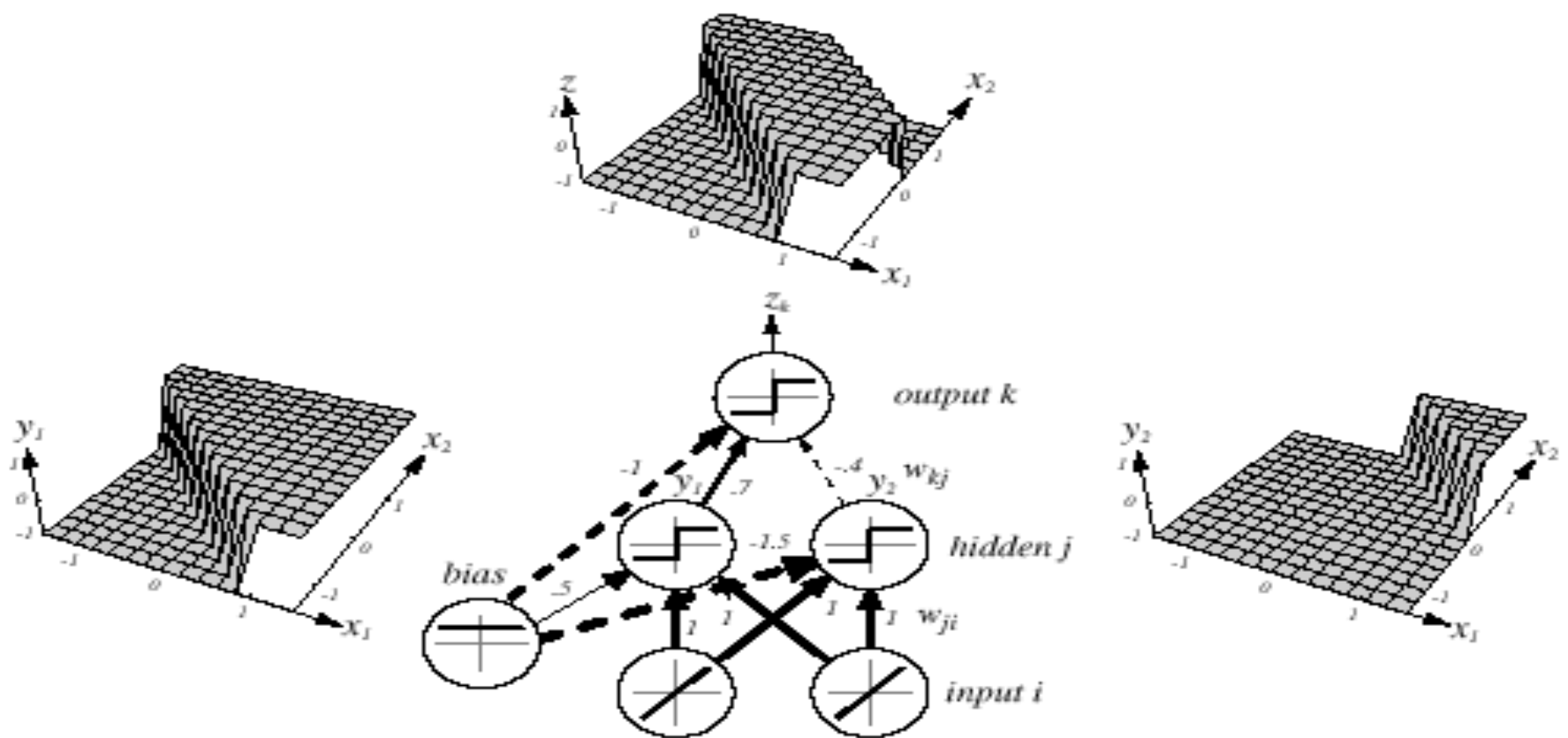
$$\frac{1}{1 + e^{-x}}$$

Eigenschaft: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$ (Gradient)

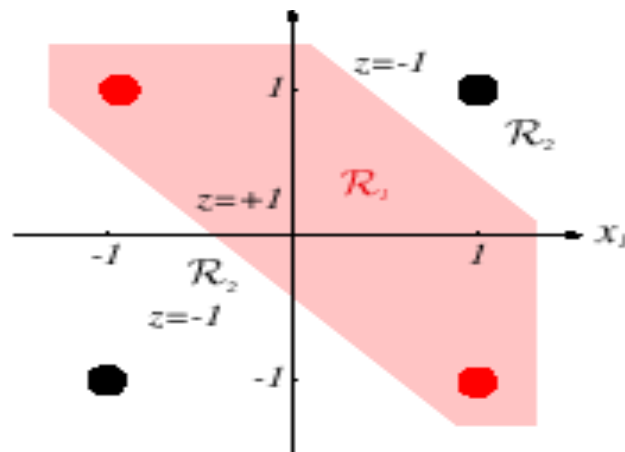
- Wir können den Gradienten verwenden, um die Einheit anzupassen
- Wir kommen gleich darauf zurück

Mehr-Ebenen Netze von Sigmoid-Einheiten





$$Z = x_1 \text{ XOR } x_2 = (x_1 \text{ OR } x_2) \text{ AND NOT}(x_1 \text{ AND } x_2)$$



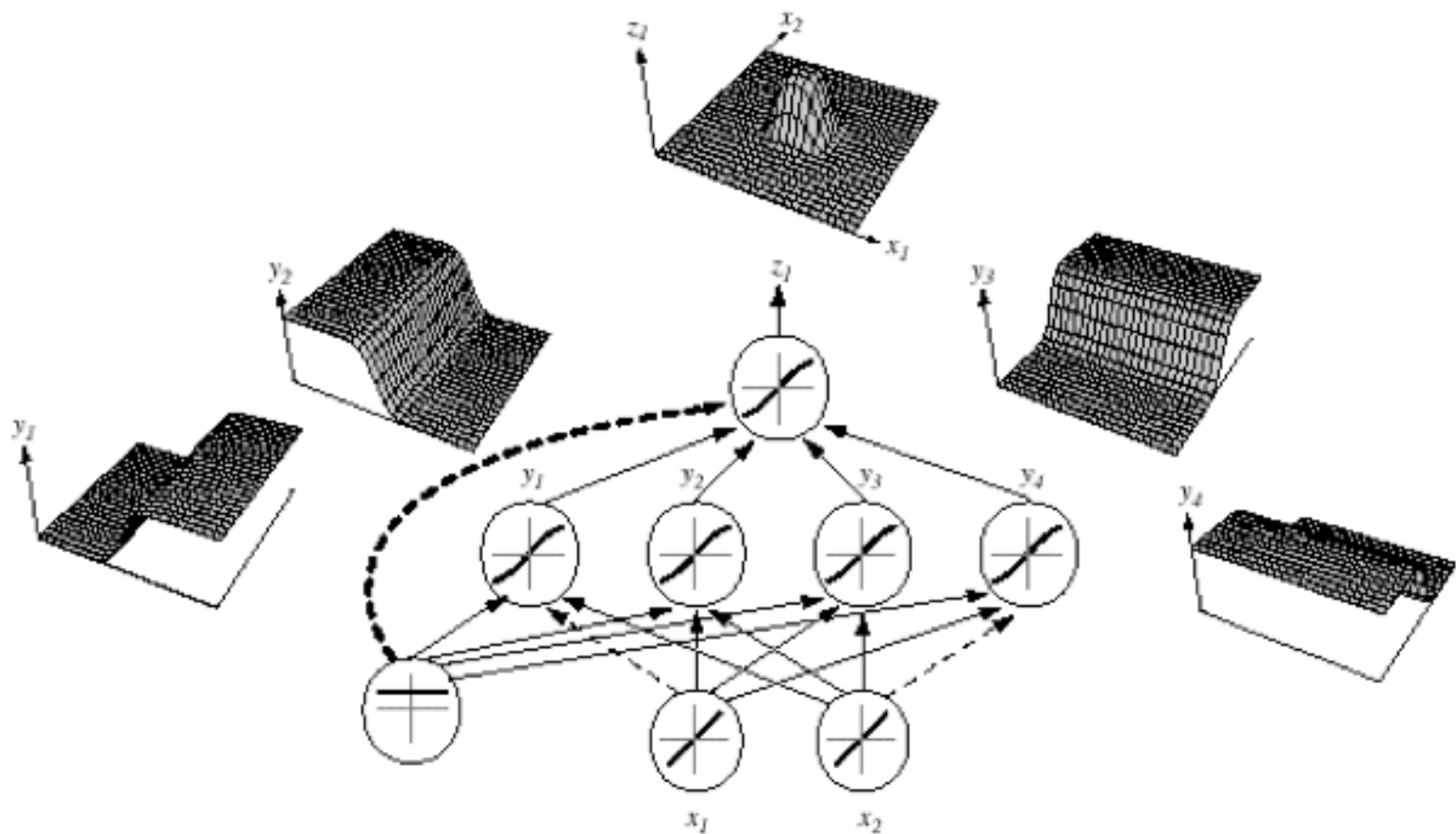


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a “bump” at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

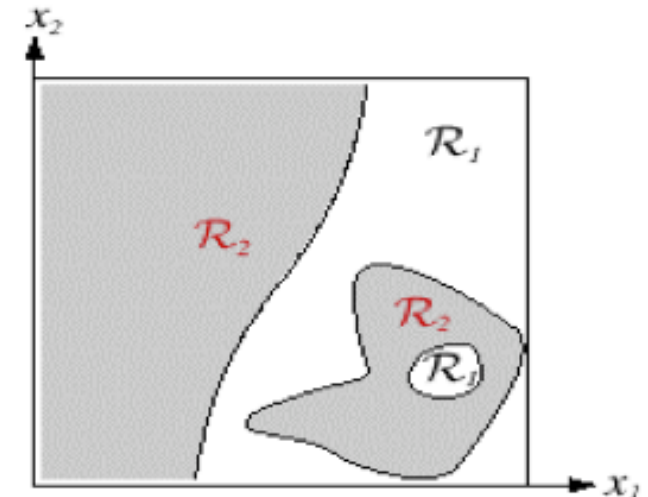
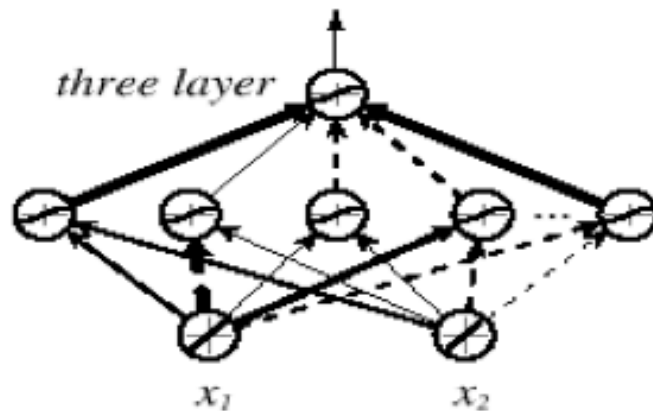
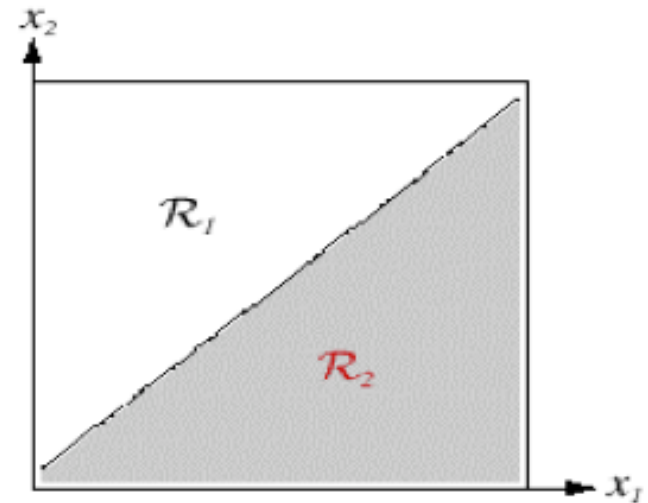
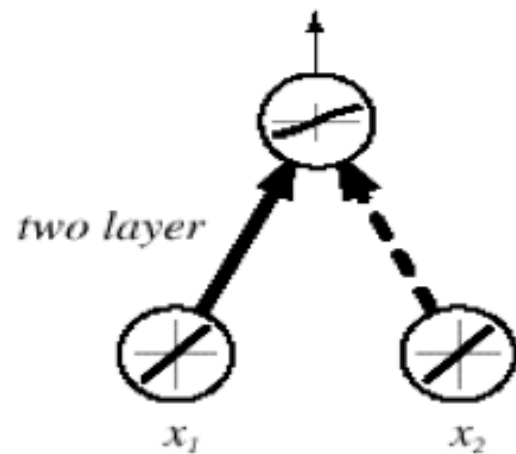
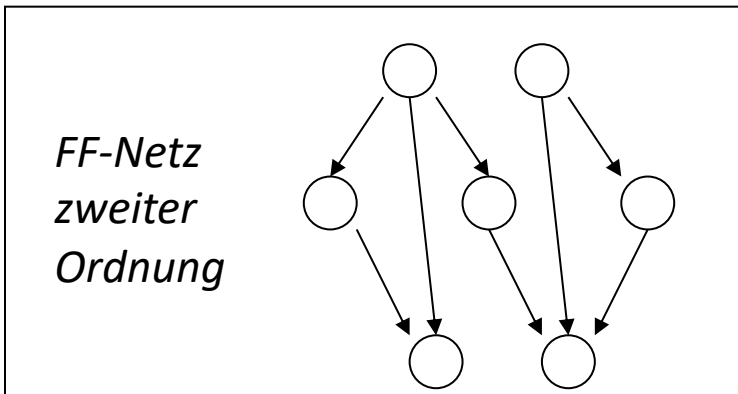
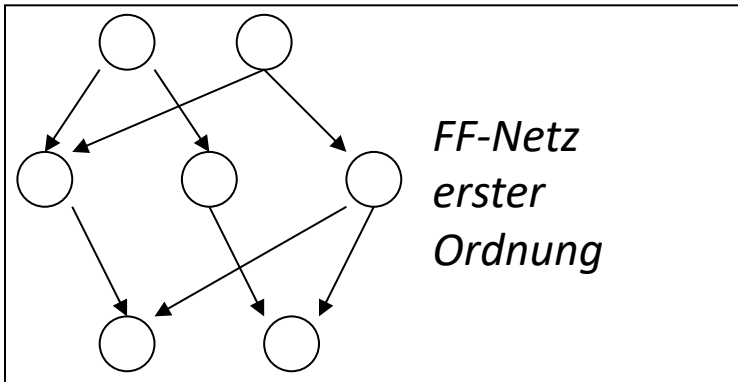


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Netztopologien

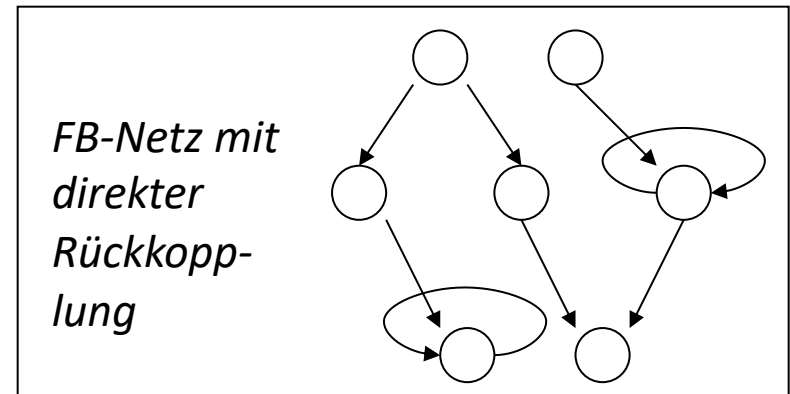
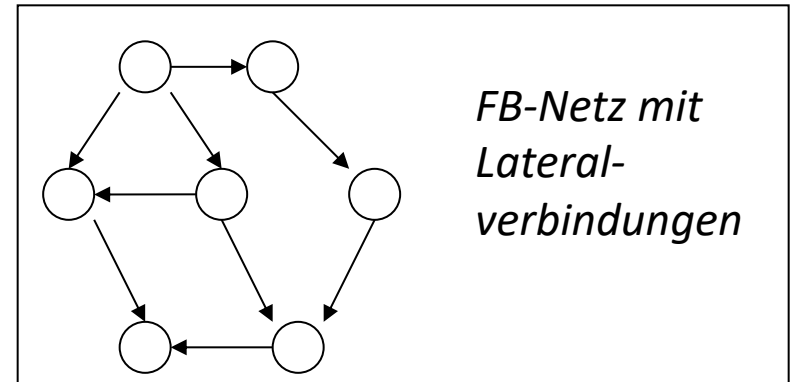
FeedForward-Netze:

Gerichtete Verbindungen nur von niedrigen zu höheren Schichten



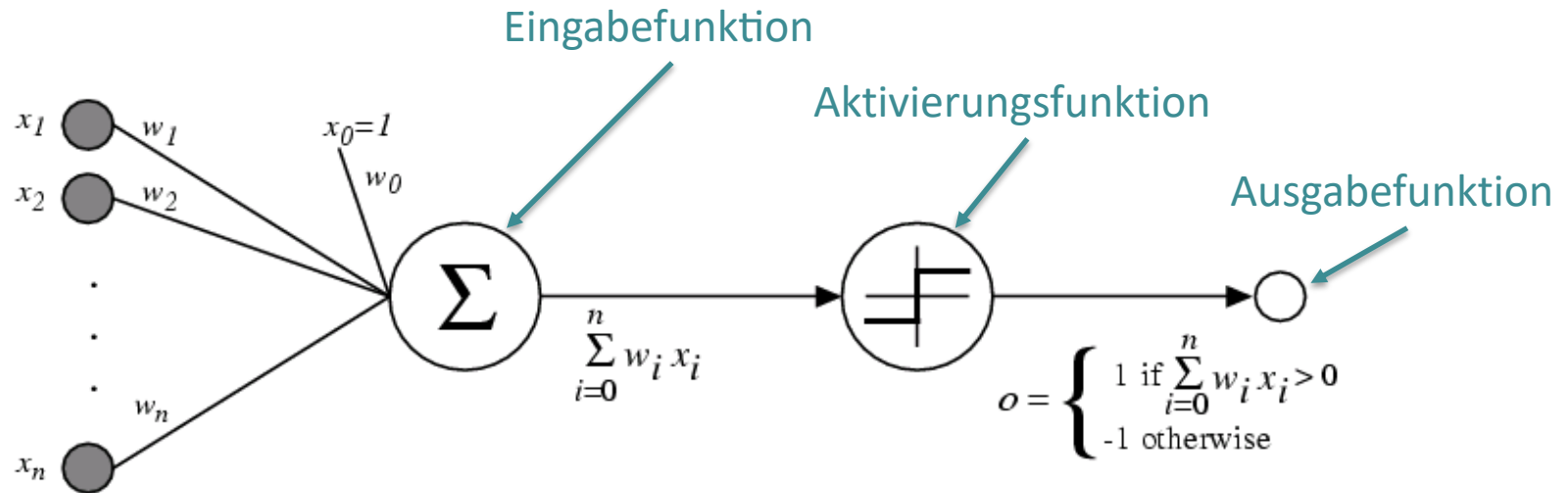
FeedBack-Netze (*rekurrente Netze*):

Verbindungen zwischen allen Schichten möglich (Abrollen)



Netzwerk von Perzeptrons

- Perzeptron:



- Was muss ich vor dem Lernen/Trainieren bestimmen?
 - Netzwerk-Topologie?
 - Perzeptron bezogen?

Die Eingabefunktion

Die Eingabe- oder Propagierungsfunktion berechnet aus dem Eingabevektor $\vec{x} = (x_1, \dots, x_m)$ und dem Gewichtsvektor $\vec{w}_k = (w_{1,k}, \dots, w_{m,k})$ den Nettoinput des k -ten Knotens. Es gibt folgende Inputfunktionen:

- Summe: $net_k = f_{in}(\vec{w}_k, \vec{x}) = \sum_{i=1}^m w_{i,k} \cdot x_i$
- Maximalwert: $net_k = f_{in}(\vec{w}_k, \vec{x}) = \max_i (w_{i,k} \cdot x_i)$
- Produkt: $net_k = f_{in}(\vec{w}_k, \vec{x}) = \prod_{i=1}^m w_{i,k} \cdot x_i$
- Minimalwert: $net_k = f_{in}(\vec{w}_k, \vec{x}) = \min_i (w_{i,k} \cdot x_i)$

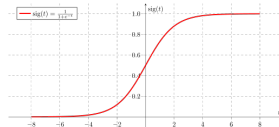
Die Aktivierungsfunktion

Mit der Aktivierungsfunktion (auch: Transferfunktion) wird aus dem Nettoinput net_k der Aktivierungszustand a_k eines Knotens berechnet.

Folgende Aktivierungsfunktionen sind gebräuchlich:

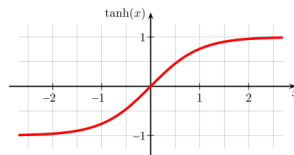
- Lineare Aktivierungsfkt.: $a_k = f_{act}(net_k) = c_k \cdot net_k$
- Binäre Schwellenwertfkt.: $a_k = f_{act}(net_k) = \begin{cases} 1, & \text{falls } net_k \geq \theta_k, \\ 0, & \text{sonst.} \end{cases}$
- Fermi-Fkt. (logistische Fkt.): $a_k = f_{act}(net_k) = \frac{1}{1 + e^{-\frac{net_k}{T}}}$

Schwellenwert,
häufig 0



Spezialfall: Sigmoid

- Tangens hyperbolicus: $a_k = f_{act}(net_k) = \frac{e^{net_k} - e^{-net_k}}{e^{net_k} + e^{-net_k}} = \frac{1 + \tanh(net_k)}{2}$



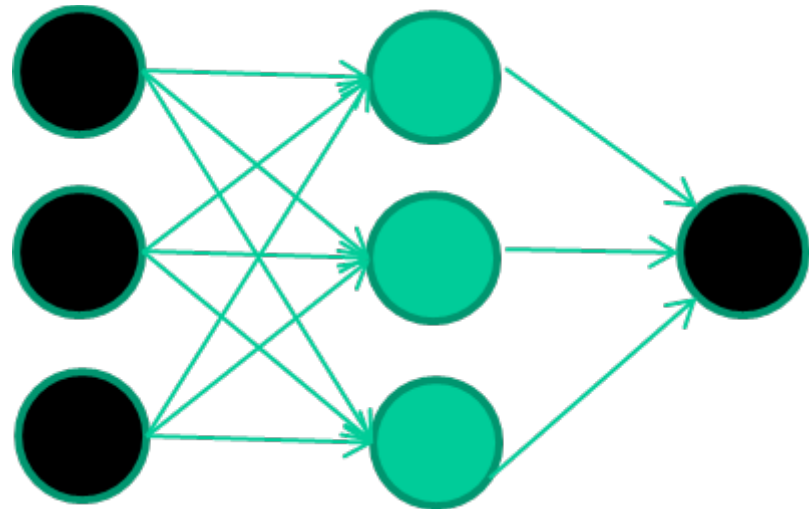
Die Ausgabefunktion

- Die Ausgabefunktion berechnet aus der Aktivierung a_k den Wert o_k , der als Ausgabe an die nächste Schicht weitergegeben wird.
- In den meisten Fällen ist die Ausgabefunktion die **Identität**, d.h. $o_k = a_k$.
- Für binäre Ausgaben wird manchmal auch eine Schwellenwertfunktion verwendet:

$$o_k = f_{out}(a_k) = \begin{cases} 1, & \text{falls } a_k \geq \theta_k, \\ 0, & \text{sonst.} \end{cases}$$

Einstellen der Gewichte mit Trainingsdaten...

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	



Anlernen des Netzwerks

<i>Fields</i>	<i>class</i>
---------------	--------------

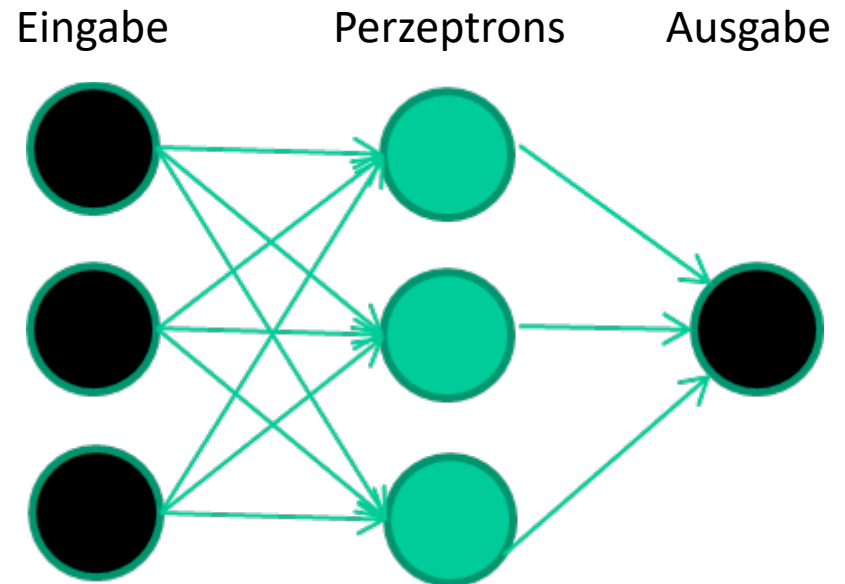
1.4	2.7	1.9	0
-----	-----	-----	---

3.8	3.4	3.2	0
-----	-----	-----	---

6.4	2.8	1.7	1
-----	-----	-----	---

4.1	0.1	0.2	0
-----	-----	-----	---

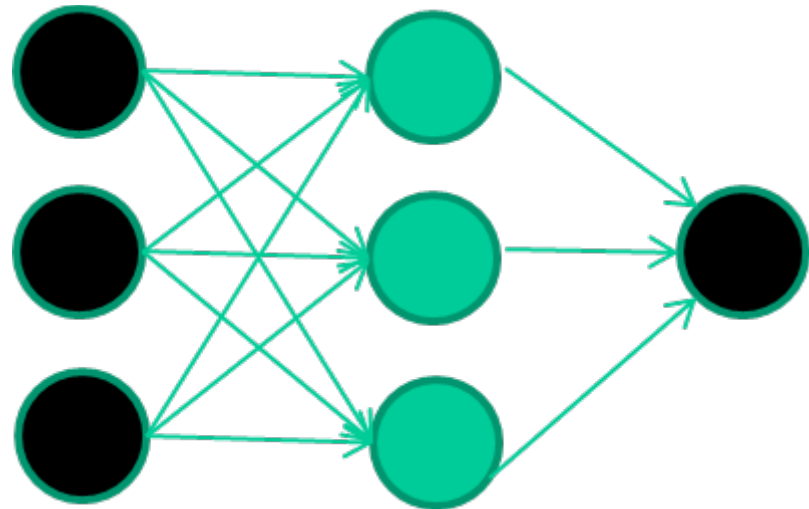
etc ...



Trainingsdaten

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Initialisierung mit zufälligen Gewichten



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

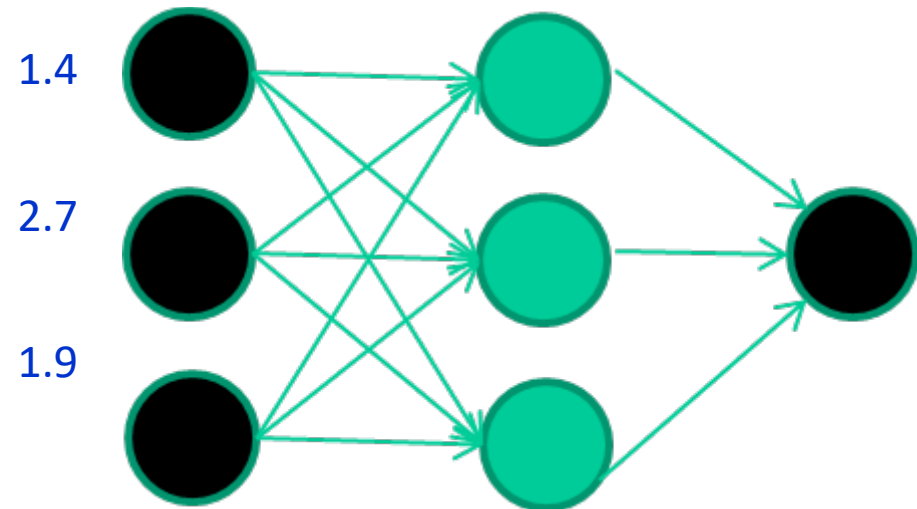
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Präsentierung eines Trainingsdatensatzes



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

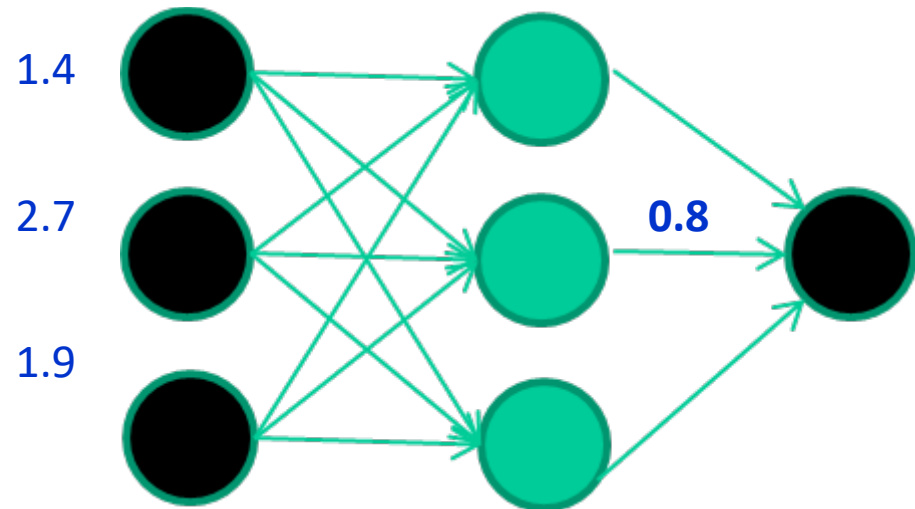
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Durchpropagierung zur Ausgabe



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

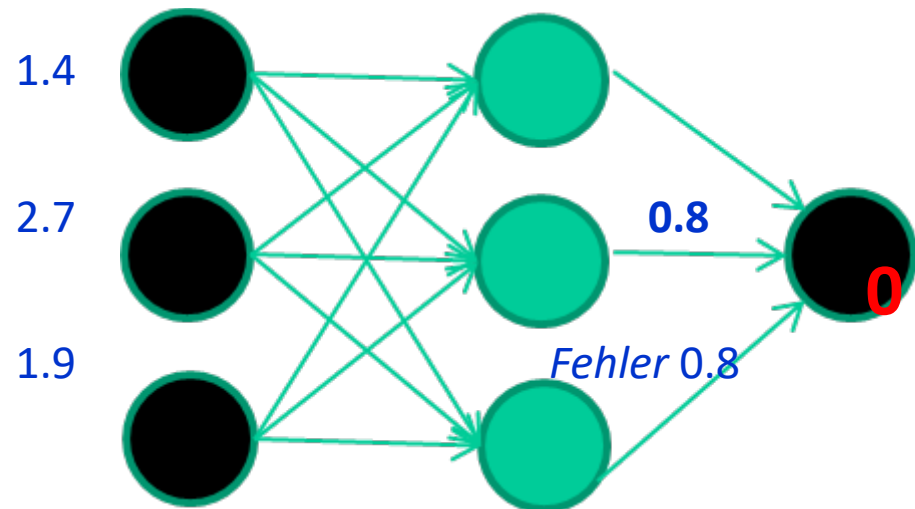
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Vergleich mit der Zielausgabe



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

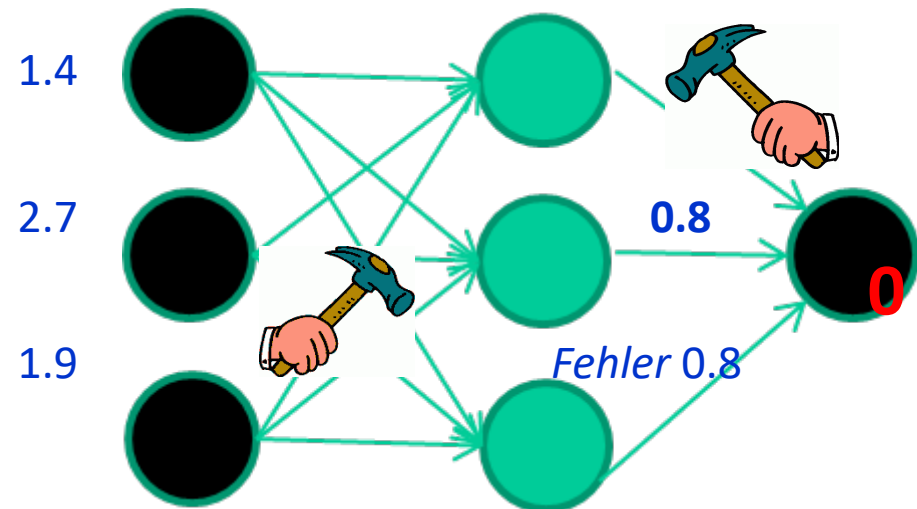
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

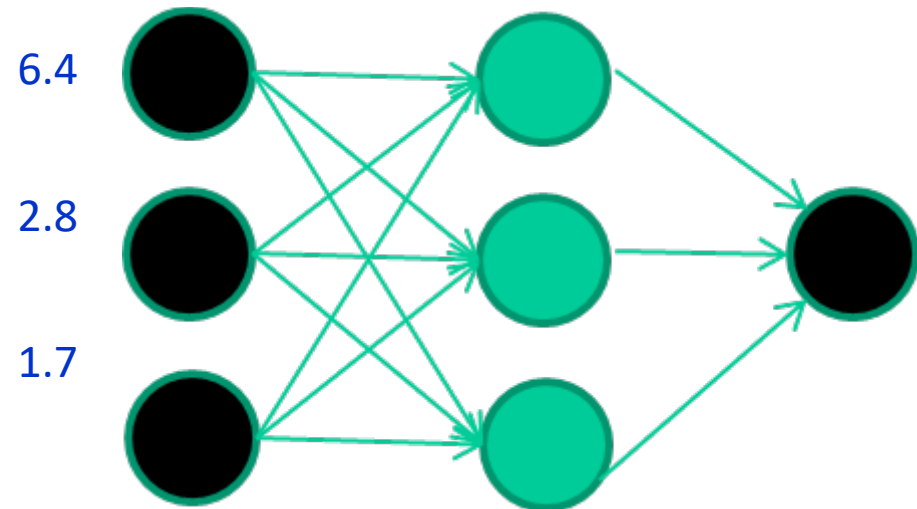
Anpassen der Gewichte gemäß Fehler



Trainingsdaten

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Präsentierung eines Trainingsdatensatzes



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

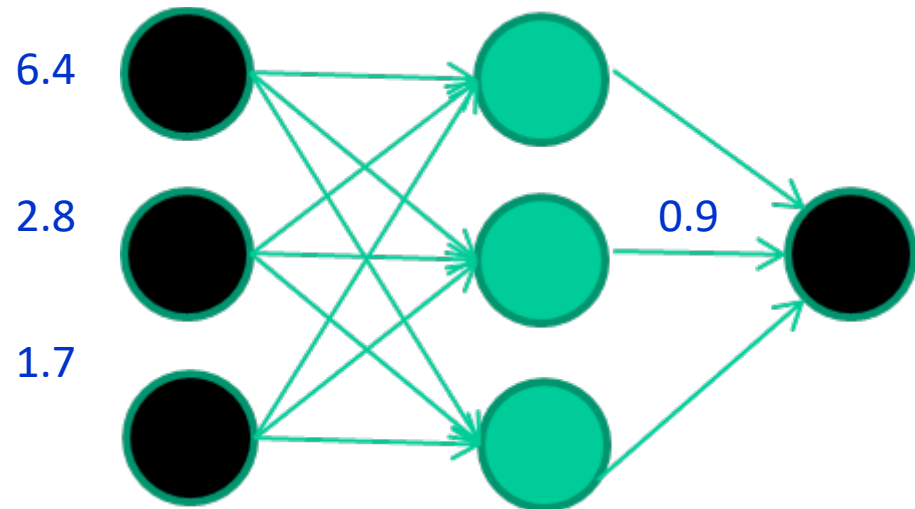
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Durchpropagierung zur Ausgabe



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

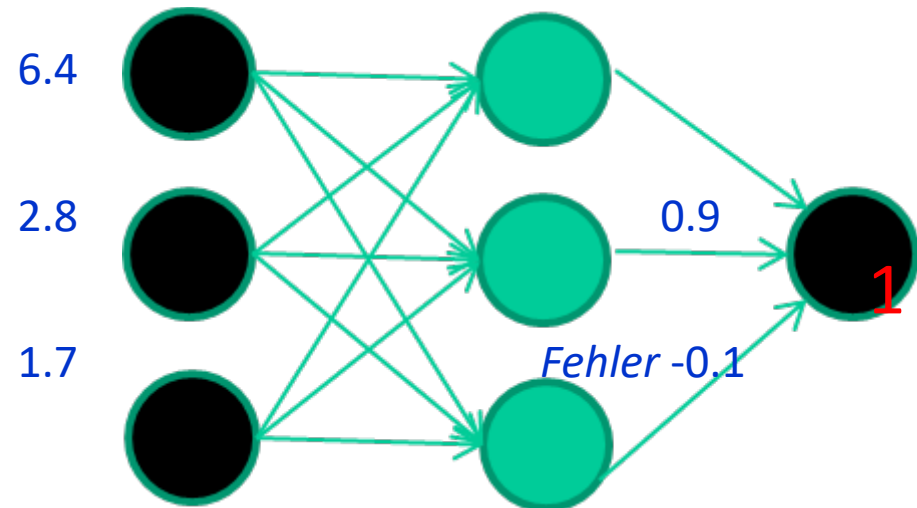
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Vergleich mit der Zielausgabe



Trainingsdaten

Fields *class*

1.4 2.7 1.9 0

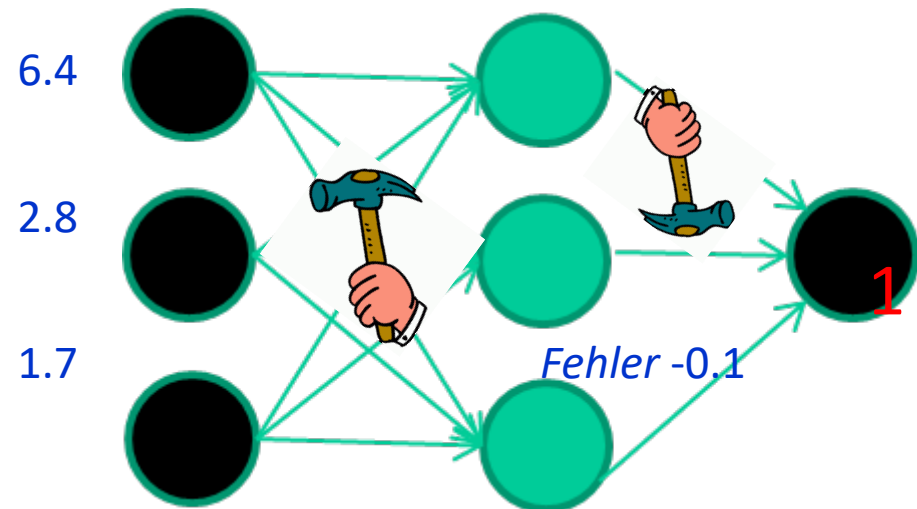
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

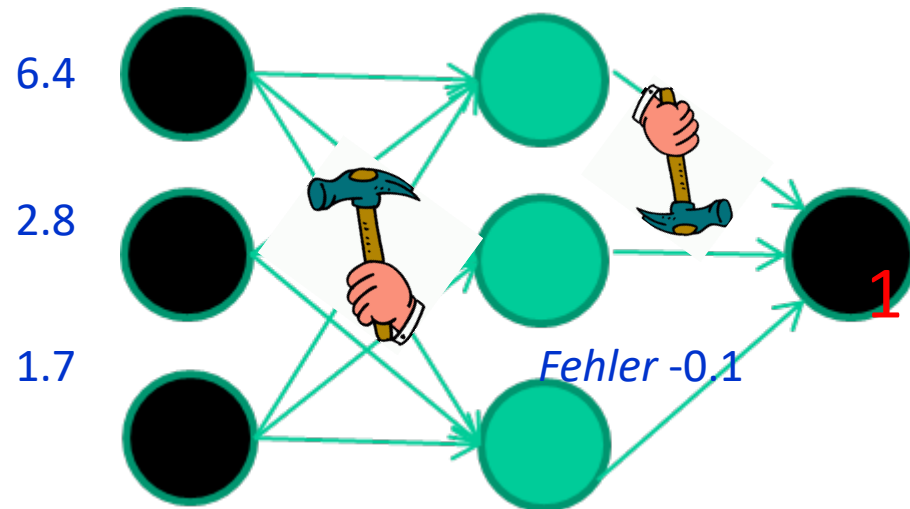
Anpassen der Gewichte gemäß Fehler



Trainingsdaten

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

Und so weiter



Wiederhole tausend-, vielleicht millionenmal – jedesmal mit einer zufälligen Trainingsinstanz und einer kleinen Anpassung der Gewichte

Verfahren zur Gewichtsangpassung müssen Fehler minimieren

Eine Ebene: Perzeptron-Lernregel (Delta-Lernregel)

$$w_i \leftarrow w_i + \Delta w_i$$

wobei

$$\Delta w_i \leftarrow \eta(t - o)x_i$$

und

- $t = c(\vec{x})$ der Zielwert ist
- o ist die Ausgabe des Perzeptrons
- η ist eine kleine Konstante (z.B. 0,1): die Lernrate

Gewichte häufig nur nach Verarbeitung eines ganzen Datensatzes D angepasst

Begründung für die Delta-Regel

- Idee: minimiere den quadratischen Fehler
 - D Trainingsmenge
 - t_d Wert für $d \in D$
 - o_d Ausgabe für d

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Minimumbestimmung mit 1. Ableitung

Absteigender Gradient

- Gradient

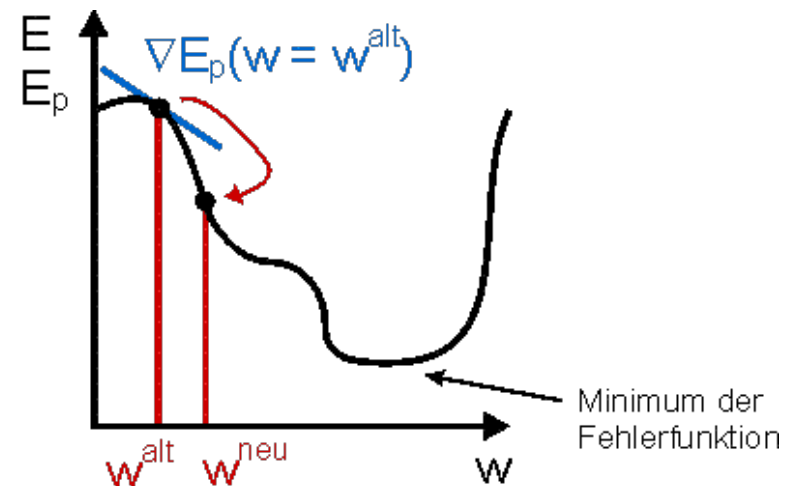
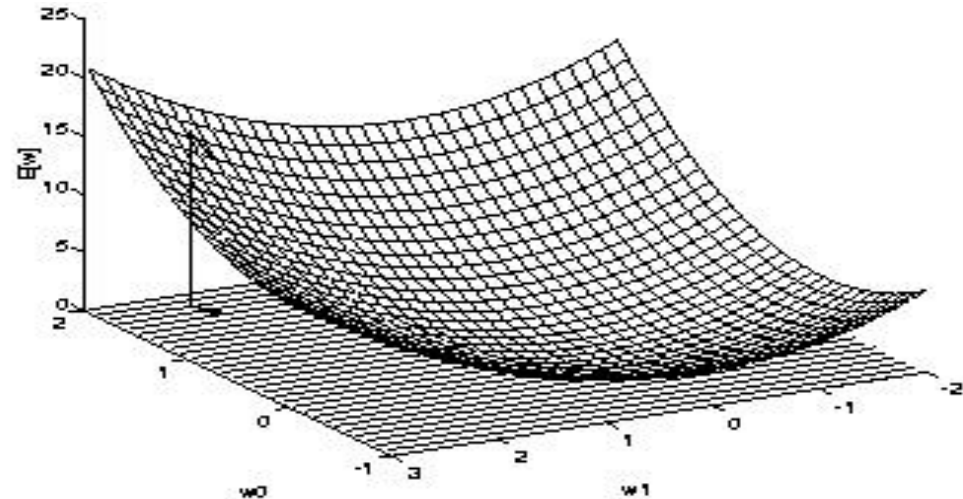
$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Lernregel

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

- i.e.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$



Gradient

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ &= \sum_{d \in D} (t_d - o_d) (-x_{i,d}) = - \sum_{d \in D} (t_d - o_d) x_{i,d}\end{aligned}$$

Absteigender Gradient (Forts.)

- Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

$$\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) x_{i,d}$$

- Lernregel

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

- i.e.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Iterativ pro Datenpunkt d

$$\Delta w_i = \eta (t - o) x_i$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{i,d}$$

Algorithmus für *eine* Gewichts Anpassung

- Jedes Trainingsbeispiel sei ein Paar $\langle \vec{x}, t \rangle$
 - \vec{x} ist ein Inputvektor (Vektor mit Feature-Werten)
 - t ist der Zielwert (Klassenlabel/Ausgabewert)
 - η ist die Lernrate
- Initialisiere jedes w_i mit einem beliebigen, kleinen Wert
- Bis die Abbruchbedingung erfüllt ist (z.B. Anzahl an verarbeiteten Trainingsbeispielen):
 - Initialisiere jedes Δw_i mit 0
 - Für jedes Trainingsbeispiel $\langle \vec{x}, t \rangle$
 - Berechne o_t
 - Für jedes Gewicht w_i : $\Delta w_i \leftarrow \Delta w_i + \eta(t - o_t)x_i$
 - Für jedes w_i : $w_i \leftarrow w_i + \Delta w_i$

Perzeptron-Lernregel

Man kann zeigen, dass der Vorgang konvergiert, ...

- ... wenn die Daten linear separierbar sind
- ... und η genügend klein gewählt wird

Schon früher untersucht:

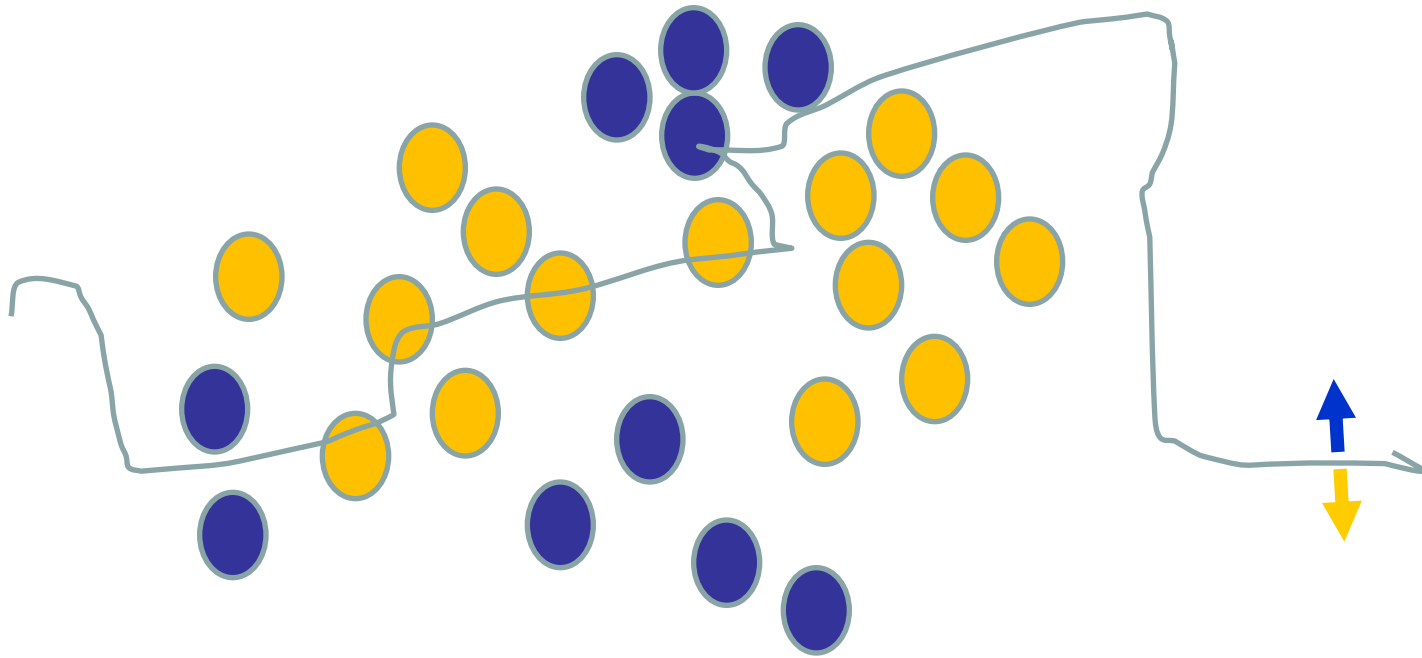
D. Hebb: The organization of behavior. A neuropsychological theory.
Erlbaum Books, Mahwah, N.J., 1949

Netzwerke daher von manchen
als künstliche neuronale Netze bezeichnet

Später für mehrschichtige Netze erweitert (Deep Learning):
Fehlerrückführung durch mehrere Ebenen (Backpropagation)

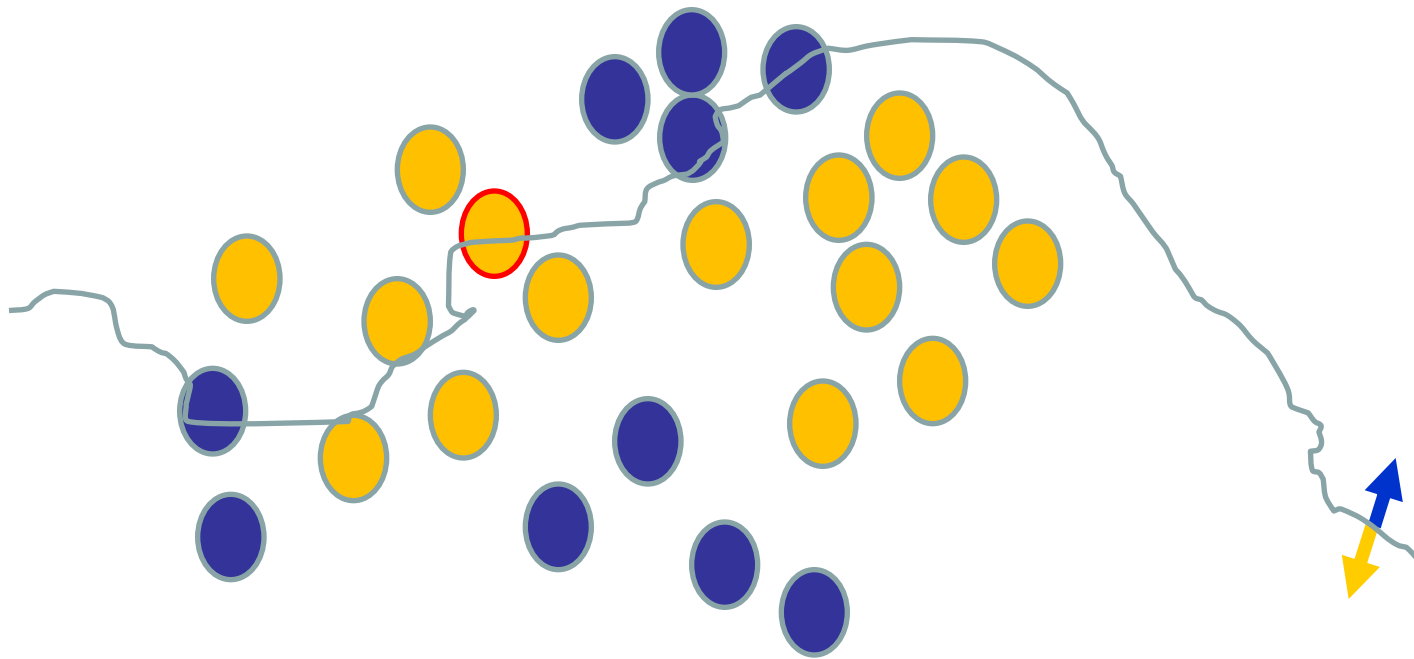
Perspektive der Entscheidungsgrenzenanpassung

Zufällige Initialgewichte



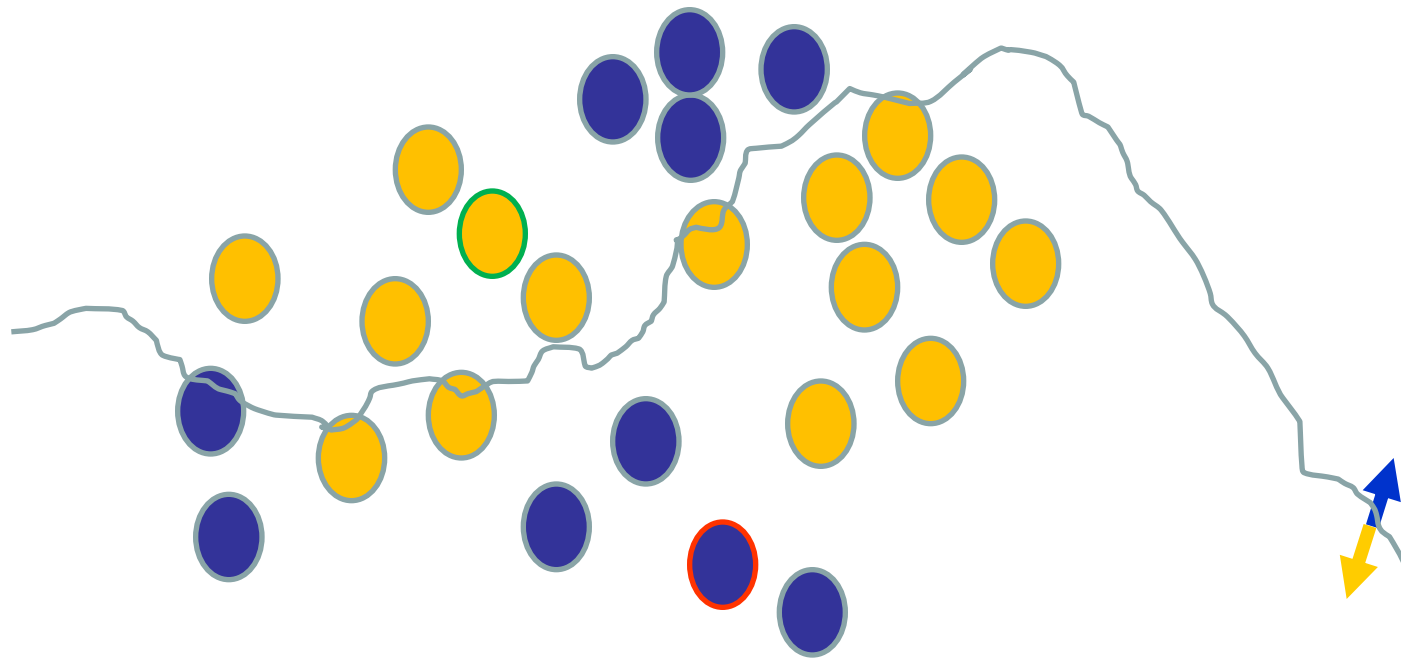
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte



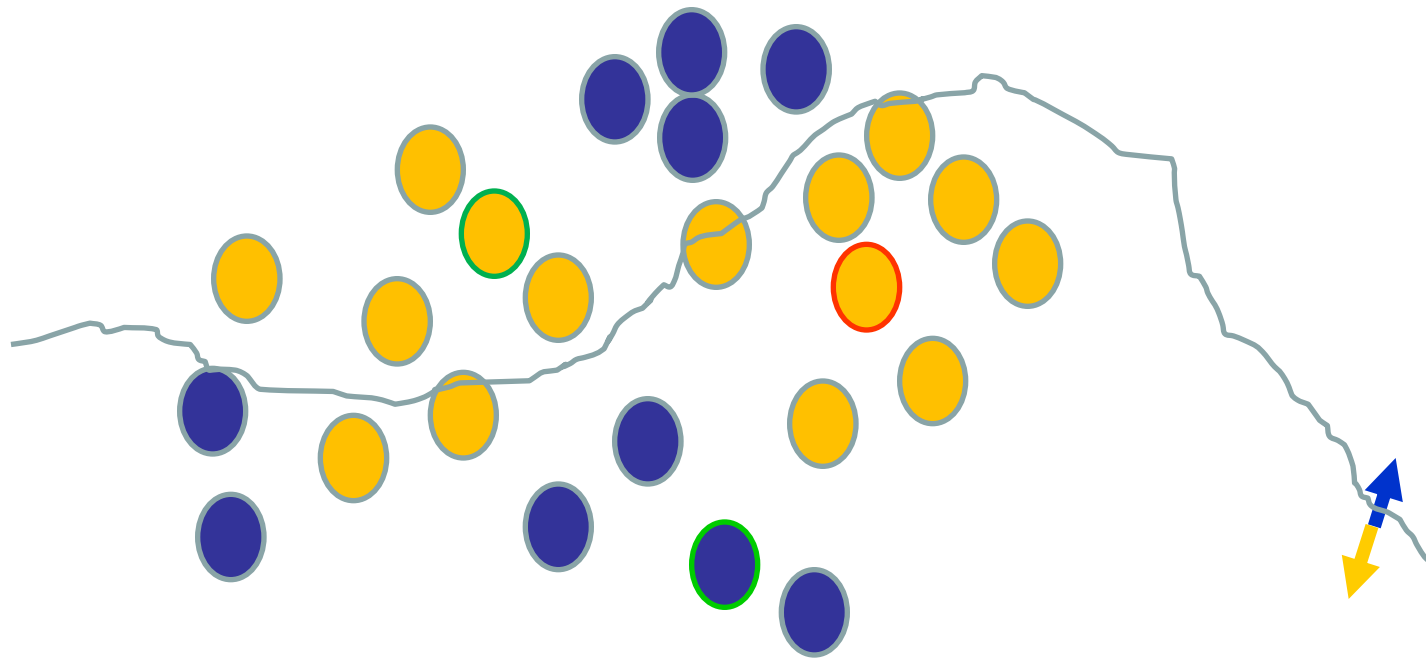
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte



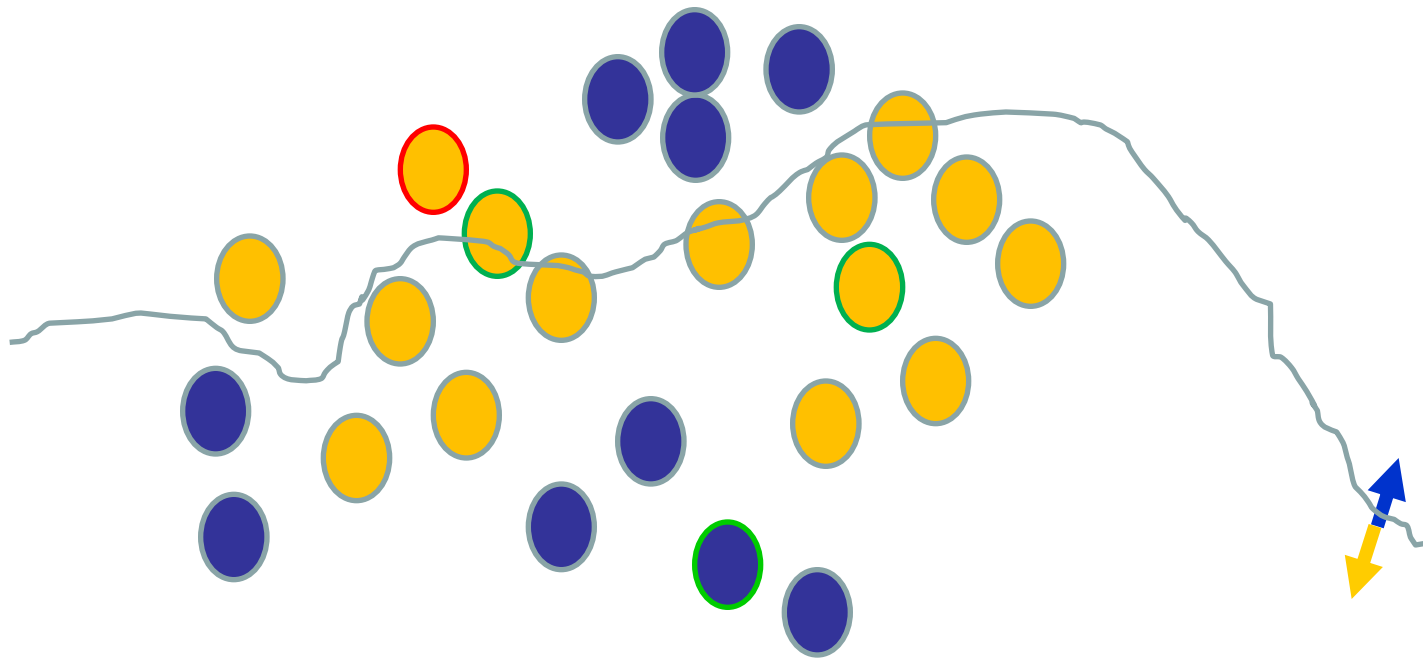
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte



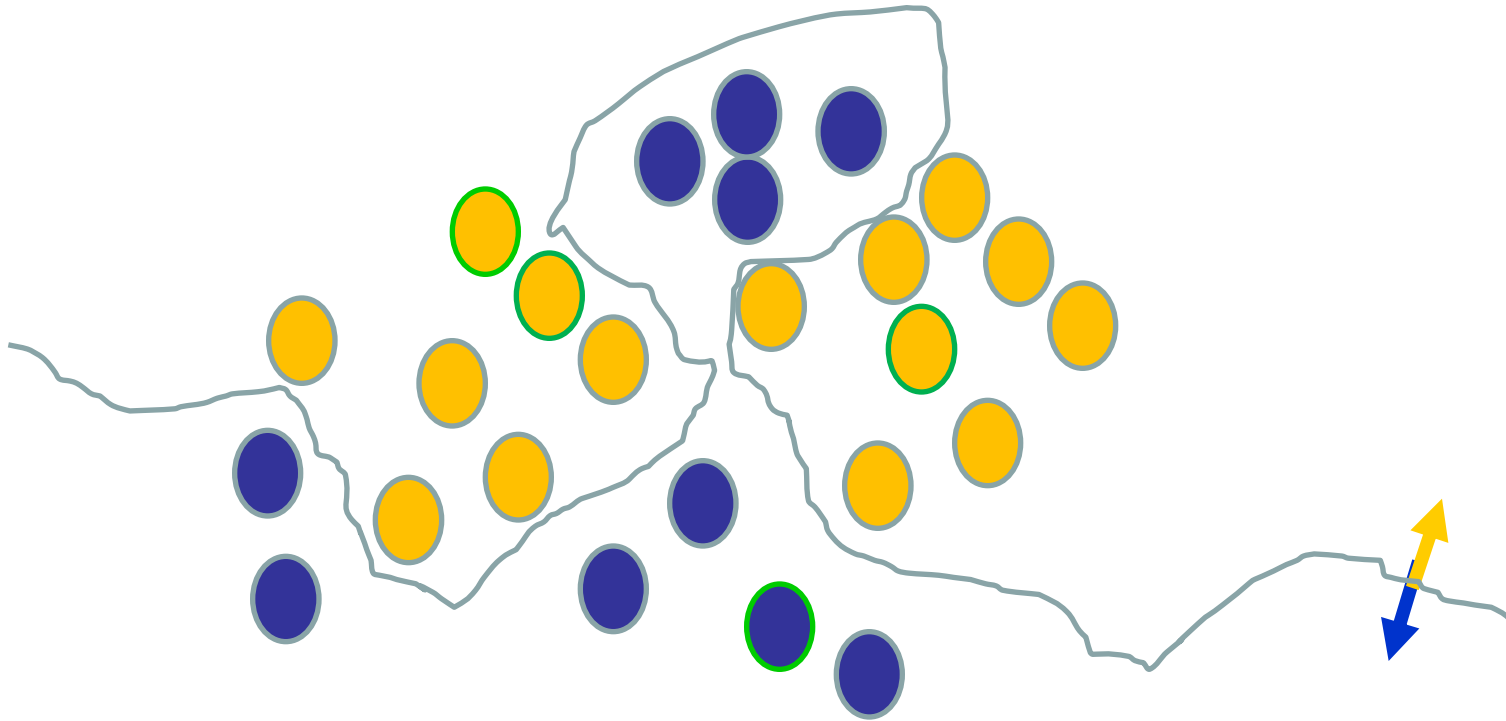
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte

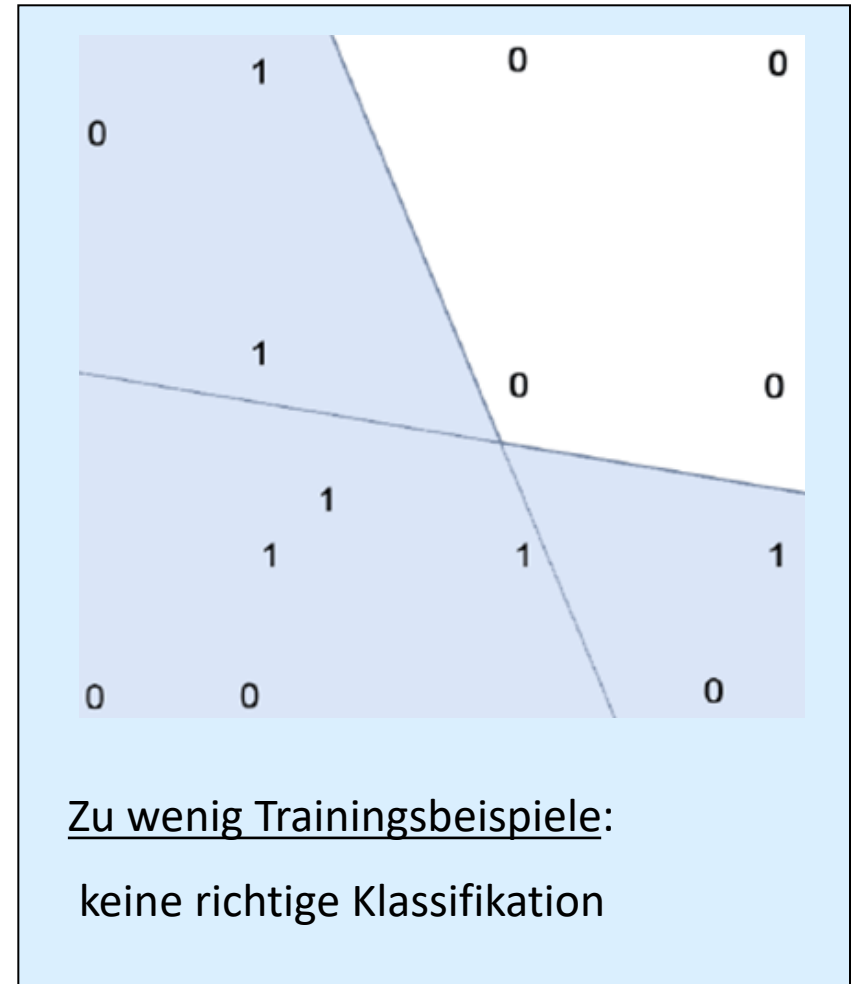
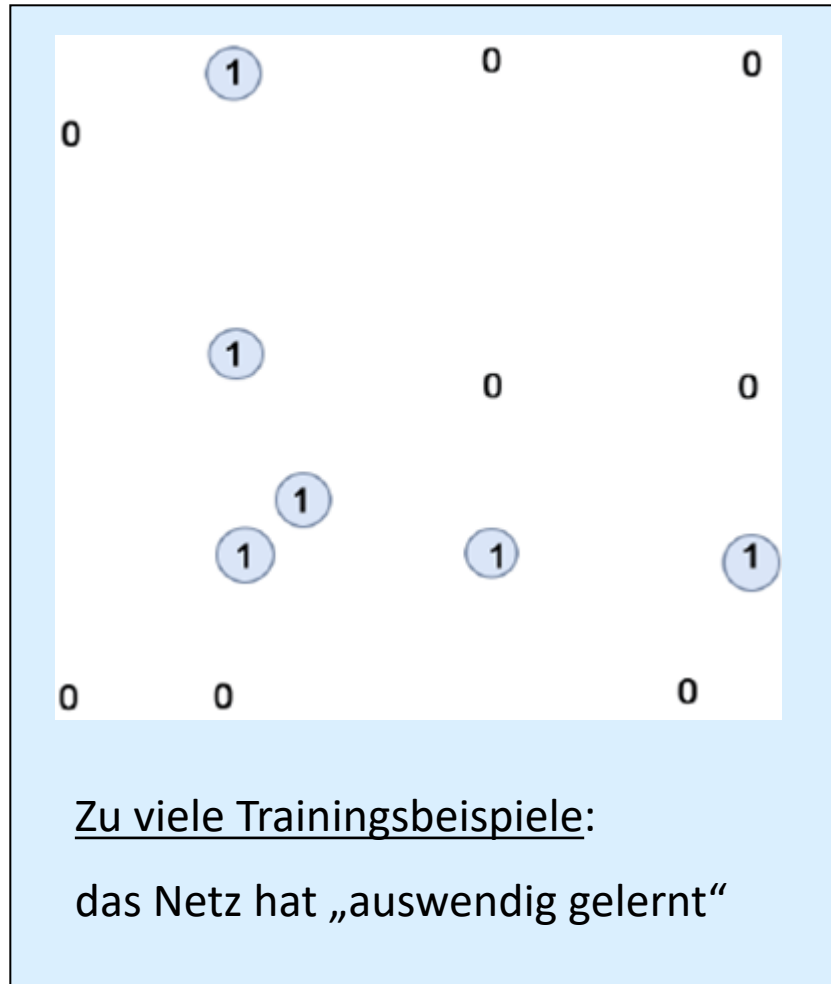


Perspektive der Entscheidungsgrenzenanpassung

Und schließlich....

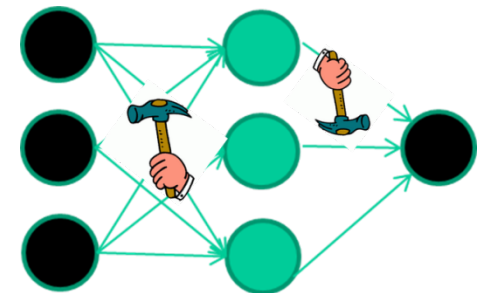


Fehler bei einer ungeeigneten Trainingsmenge



Lernverfahren: Resümee

- Epoche: einmalige Verwendung des gesamten Datensatzes
 - Manchmal zu groß
- Batch: Zerlegung eines Datensatzes in Teilmengen
 - Für jede Teilmenge: Passe Gewichte an
 - Anzahl der Teilmengen: #Iterationen
- Tausende von kleinen Anpassungen, jede macht das Netz besser für die letzte Eingabe (aber vielleicht schlechter für frühere Eingaben)
- Verwende mehrere Epochen
- Wieviele Epochen? Batches?
 - Ausprobieren
- Durch verdammt gutes Glück kommt oft eine Funktion heraus, die gut genug in Anwendungen funktioniert



Geschichtlicher Überblick

Anfänge

- **1943:** McCulloch und Pitts beschreiben und definieren eine Art erster neuronaler Netzwerke.
- **1949:** Formulierung der Hebb'schen Lernregel (nach Hebb)
- **1957:** Entwicklung des Perzeptrons durch Rosenblatt

Ernüchterung

- **1969:** Minsky und Papert untersuchen das Perzeptron mathematisch und zeigen dessen Grenzen, etwa beim XOR-Problem, auf.

Renaissance

- **1982:** Beschreibung der ersten selbstorganisierenden Netze (nach *biologischem* Vorbild) durch van der Malsburg und Kohonen und eines richtungweisenden Artikels von Hopfield, indem die ersten rückgekoppelten Netze (Hopfield-Netze, nach *physikalischen* Vorbild) beschrieben werden
- **1986:** Das Lernverfahren Backpropagation für mehrschichtige Perzeptrons wird entwickelt.

Boom

- **Ab 2000:** Deep Learning (Hinton, LeCun, Bengio, Ng, et al.)

Support-Vektor Maschinen

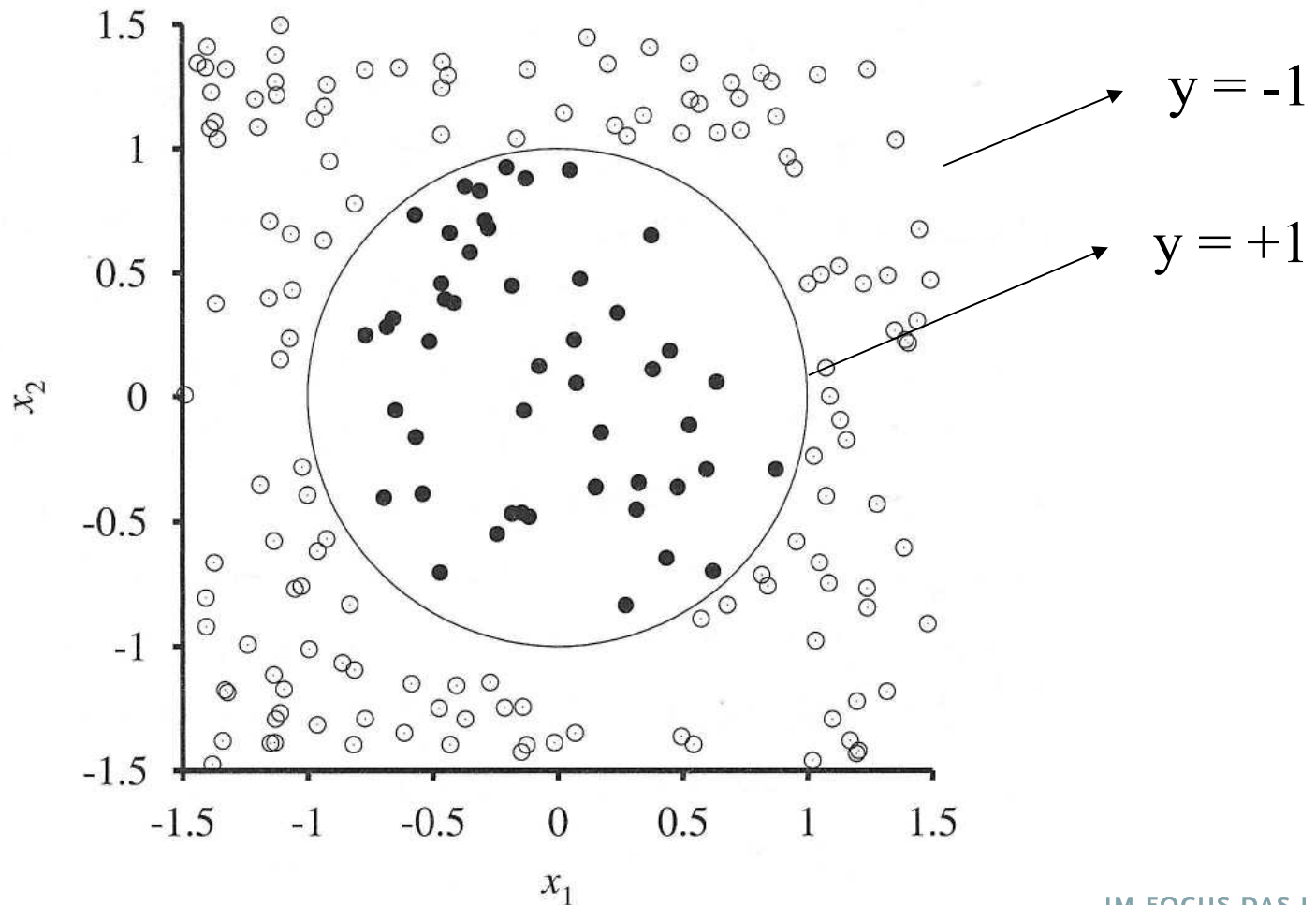
- **Abbildung** von Instanzen von zwei Klassen in einen Raum, in dem sie linear separierbar sind
 - Abbildungsfunktion heißt **Kernel-Funktion**
- Berechnung einer Trennfläche über **Optimierungsproblem** (und nicht iterativ wie bei Perzeptrons und mehrschichtigen Netzen)
 - Formulierung als **Problem nicht Verfahren!**

V. Vapnik, A. Chervonenkis, A note on one class of perceptrons.
Automation and Remote Control, **25**, **1964**

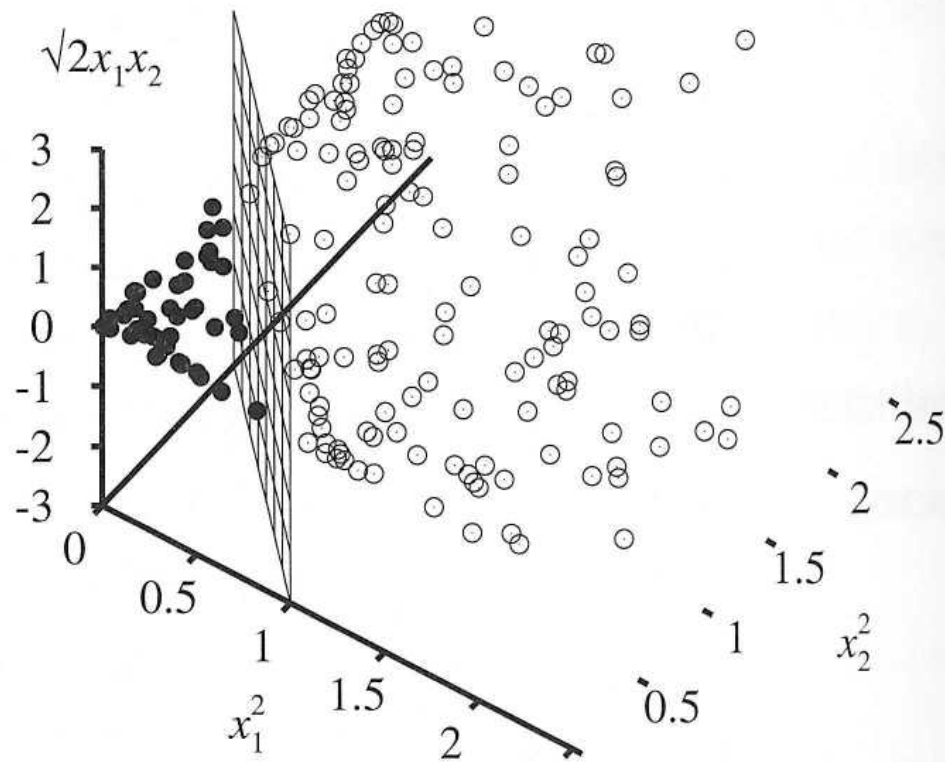
Boser, B. E.; Guyon, I. M.; Vapnik, V. N., A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory – COLT '92*. p. 144, **1992**

Vapnik, V., Support-vector networks,
Machine Learning. 20 (3): 273–297, **1995**

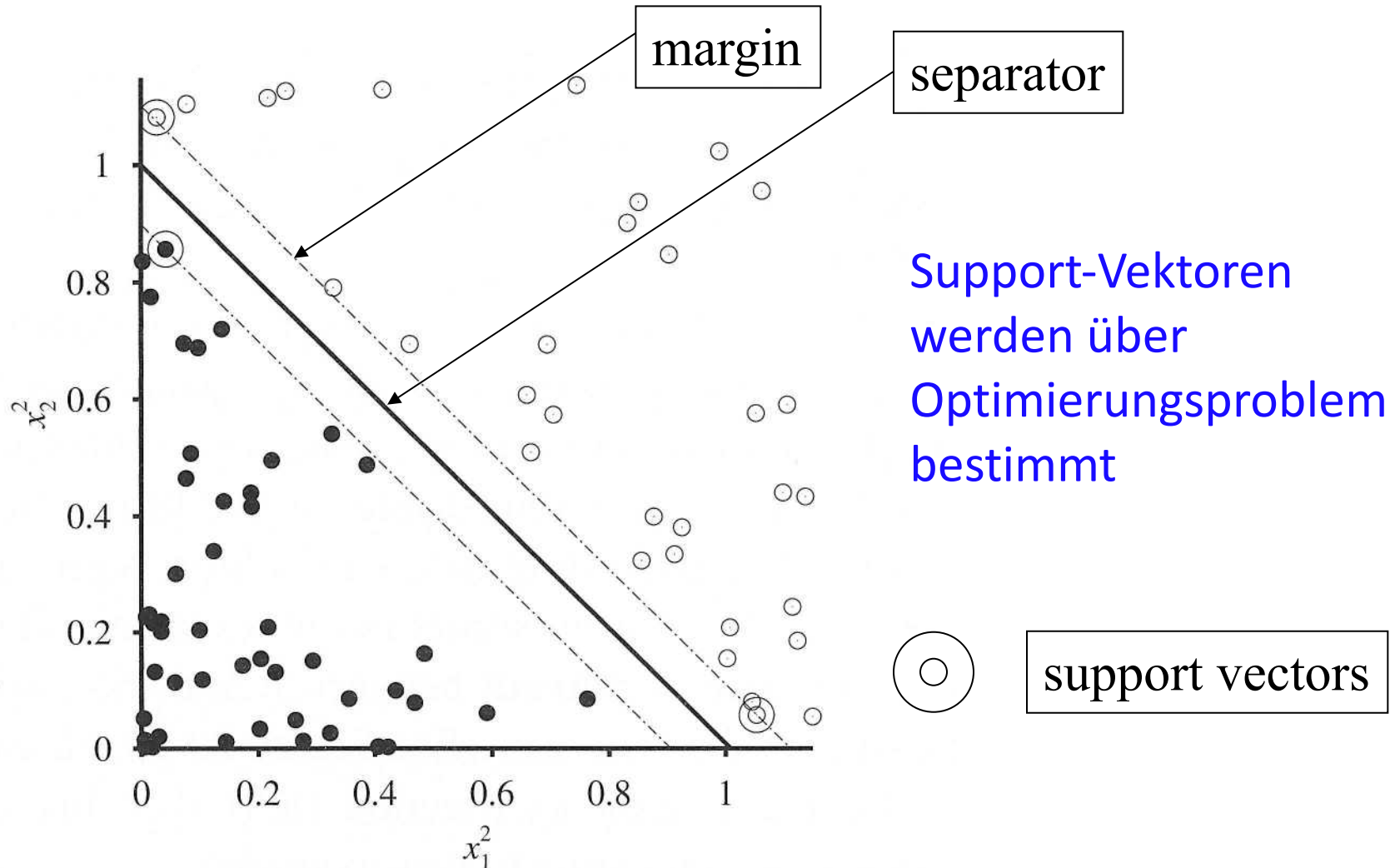
Nichtlineare Separierung



$$(x_1^2, x_2^2, \sqrt{2x_1x_2})$$



Support-Vektoren





Literatur (1)

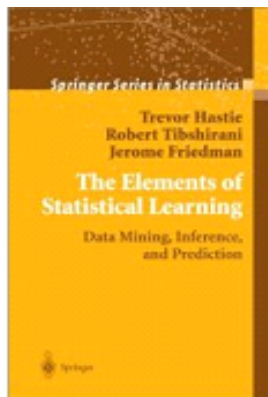
Mitchell (1989). Machine Learning.

<http://www.cs.cmu.edu/~tom/mlbook.html>



Duda, Hart, & Stork (2000). Pattern Classification.

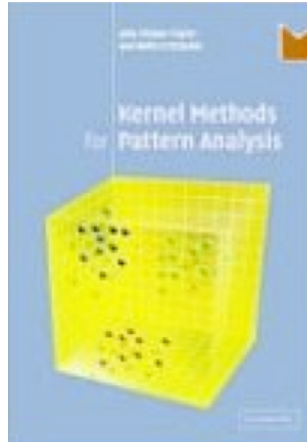
<http://rii.ricoh.com/~stork/DHS.html>



Hastie, Tibshirani, & Friedman (2001). The Elements of Statistical Learning.

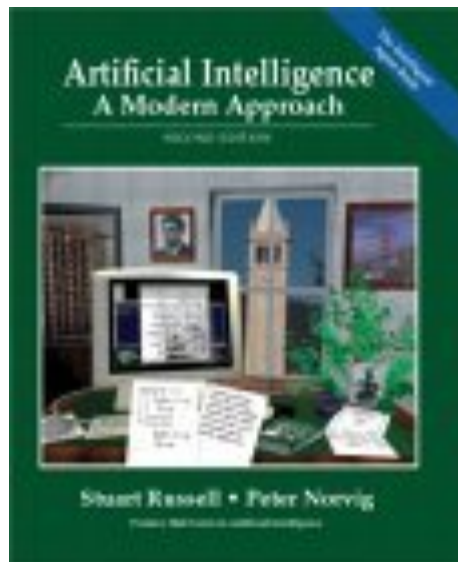
<http://www-stat.stanford.edu/~tibs/ElemStatLearn/>

Literatur (2)



Shawe-Taylor & Cristianini. Kernel Methods for Pattern Analysis.

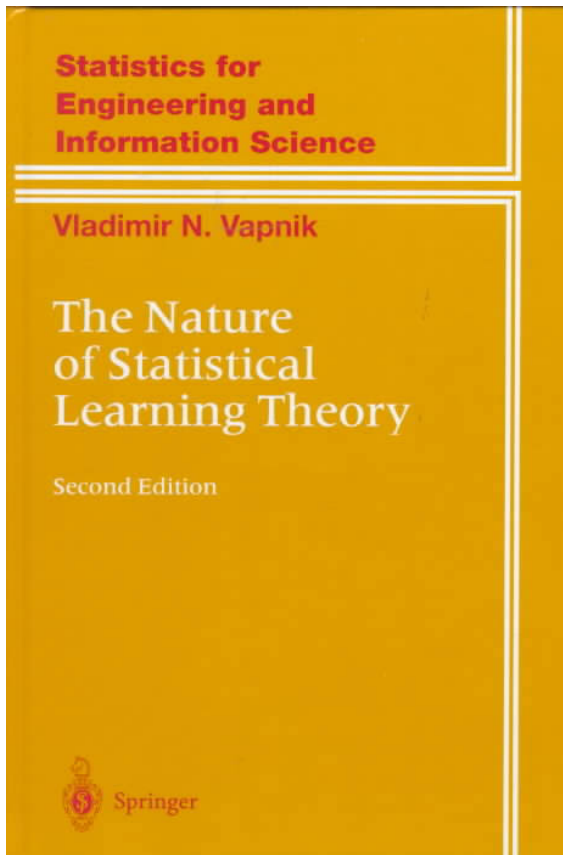
<http://www.kernel-methods.net/>



Russell & Norvig (2004). Artificial Intelligence.

<http://aima.cs.berkeley.edu/>

Originalliteratur SVM



VAPNIK, Vladimir N.,. The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc.,
1995

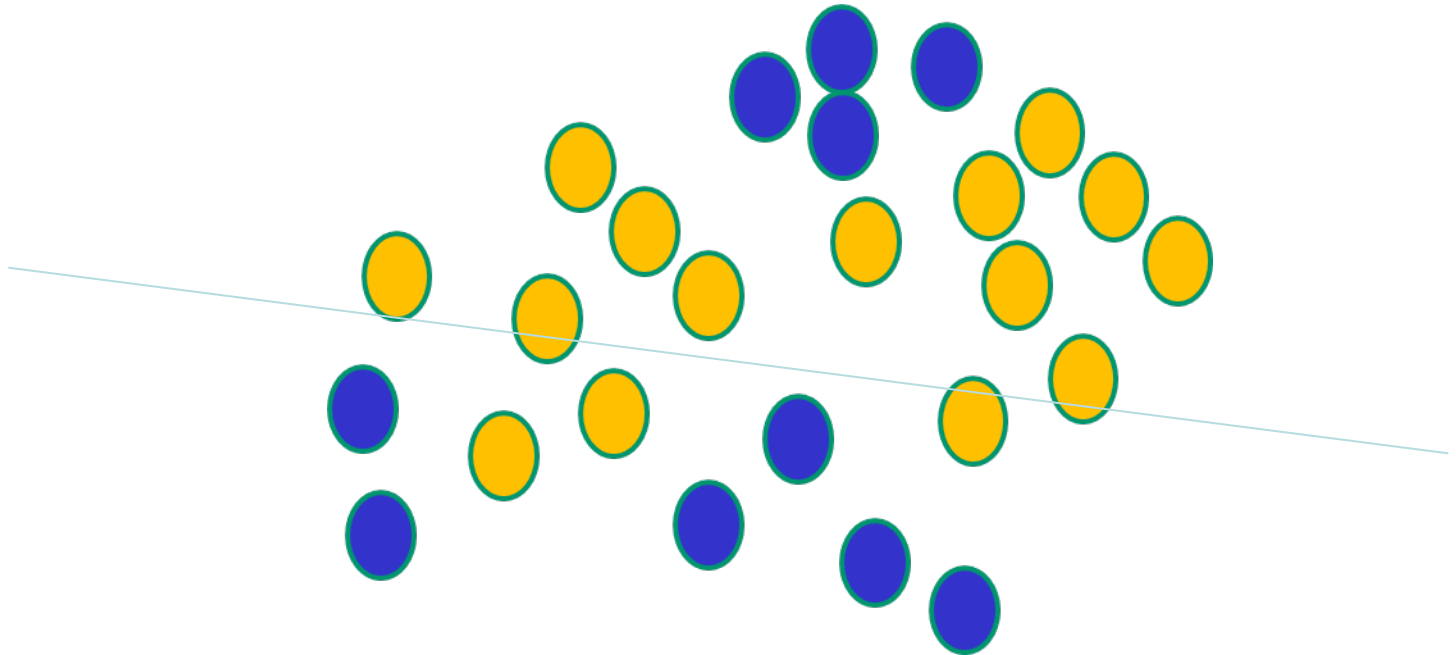
Zusammenfassung: Netze vs. SVMs (1)

$f(x)$ repräsentierte Funktion in Netzwerk

- Wenn $f(x)$ nichtlinear ist, gilt:
 - Ein Netzwerk mit einer internen Ebene (hidden layer) kann, theoretisch, eine Funktion für jedes Klassifikationsproblem lernen.
 - Es existiert Gewichtetesatz, so dass gewünschte Ausgaben produziert werden
- Das Problem ist, die Gewichte zu bestimmen

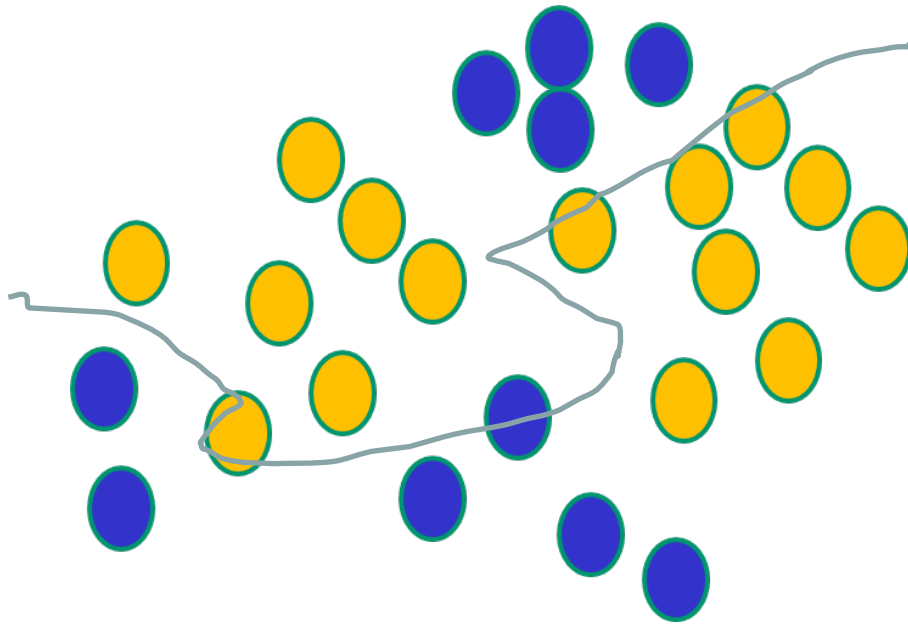
Zusammenfassung: Netze vs. SVMs (2)

Wenn $f(x)$ linear, können Netze nur gerade Linien (oder Ebenen) finden (selbst bei mehreren Schichten)



Zusammenfassung: Netze vs. SVMs (3)

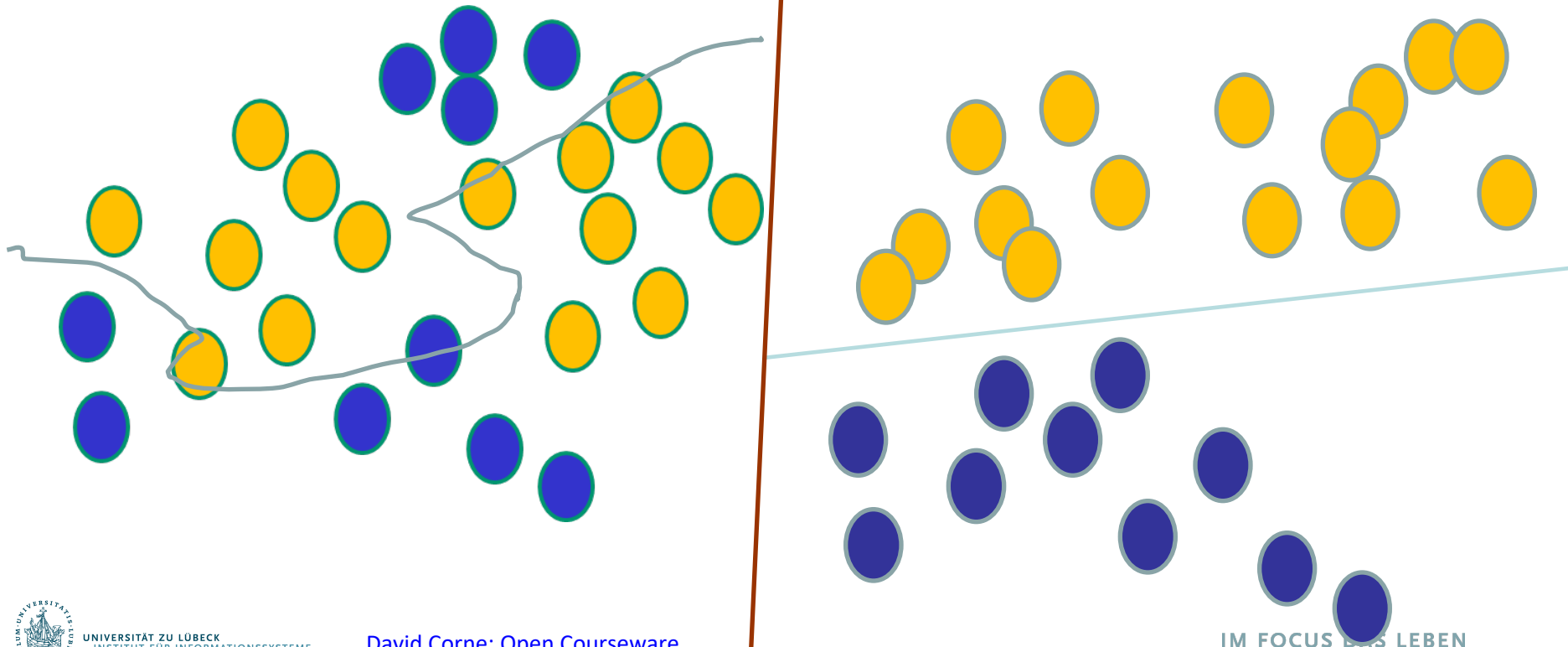
Mit nichtlinearen $f(x)$ können auch komplexere Entscheidungsgrenzen "eingestellt" werden
(Daten bleiben unverändert)



Zusammenfassung: Netze vs. SVMs (4)

Mit nichtlinearen $f(x)$ können auch komplexe Entscheidungsgrenzen "eingestellt" werden (Daten bleiben unverändert)

SVMs finden nur gerade Entscheidungsgrenzen, aber die Daten werden zuvor transformiert



Multi-class SVMs?

- SVMs für mehrere Klassenlabel
- Kombination mehrerer SVMs
 - Einer-gegen-alle
 - Für jede Klasse gibt es eine Entscheidungsgrenze zwischen den „eigenen“ Datenpunkten und allen anderen Datenpunkten
 - Alle-gegen-alle
 - Für jede mögliche Kombination von zwei Klassen gibt es eine Entscheidungsgrenze