Einführung in Web- und Data-Science

Differenzierbare Programmierung

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Differenzierbare Programmierung

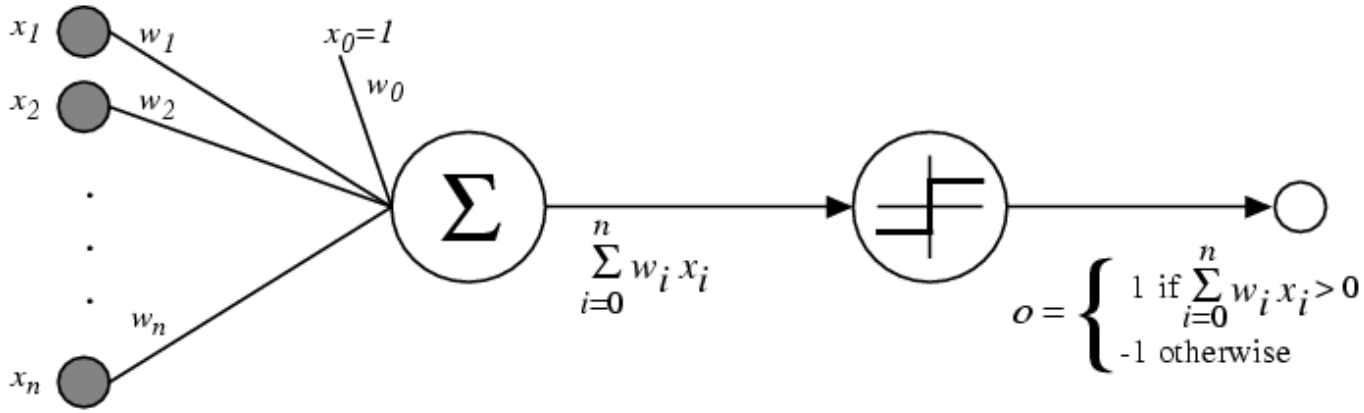
PERZEPTRONS

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**

Perzeptron

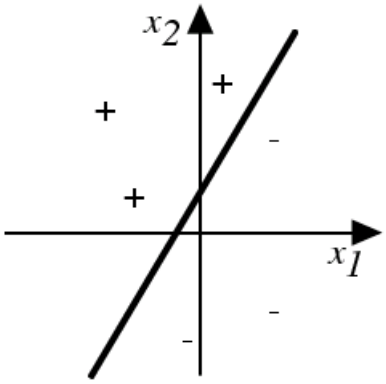


$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{sonst} \end{cases}$$

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**

Trennlinien

- Wenn $w_2x_2 + w_1x_1 + w_0 > 0$, dann wird für (x_1, x_2) ein (+) vergeben.
- $w_2x_2 + w_1x_1 + w_0 = 0$ ist Trennlinie. Warum?
- Es wird die Gerade $x_2 = -w_1/w_2\ x_1 - w_0/w_2$ als Trennlinie verwendet.
- Wir betrachten $y = mx + b$
- Bedingung lautet: $y - mx - b > 0$
- Wähle $m=1$ und b positiv. Dann betrachte $(x, y) = (0, 2b)$. Das liegt über der Geraden.
- $2b - 0 - b > 0$
- Also wird $(0, 2b)$ mit 1 bewertet (+)

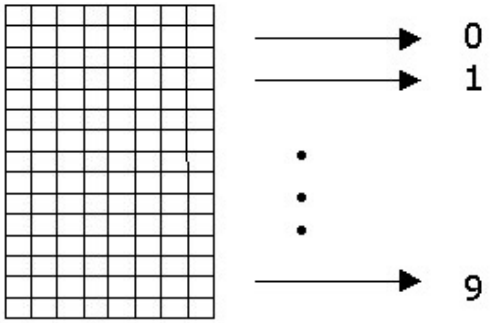
Lassen sich alle Funktionen repräsentatieren?

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

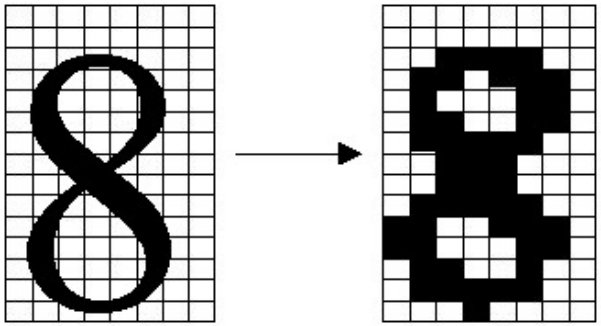
Marvin Minsky and Seymour Papert (2nd edition with corrections, first edition 1969) *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, Cambridge MA, **1972**

Ein Anwendungsbeispiel

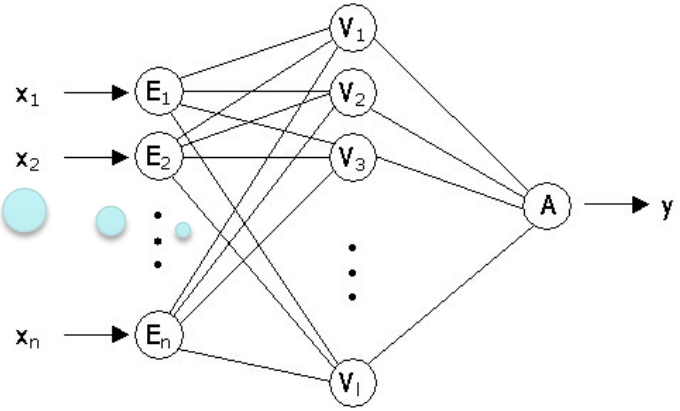
Das folgende Netz soll Ziffern von *0* bis *9* erkennen. Dafür wird zunächst das *Eingabefeld* in 8x15 Elemente aufgeteilt:



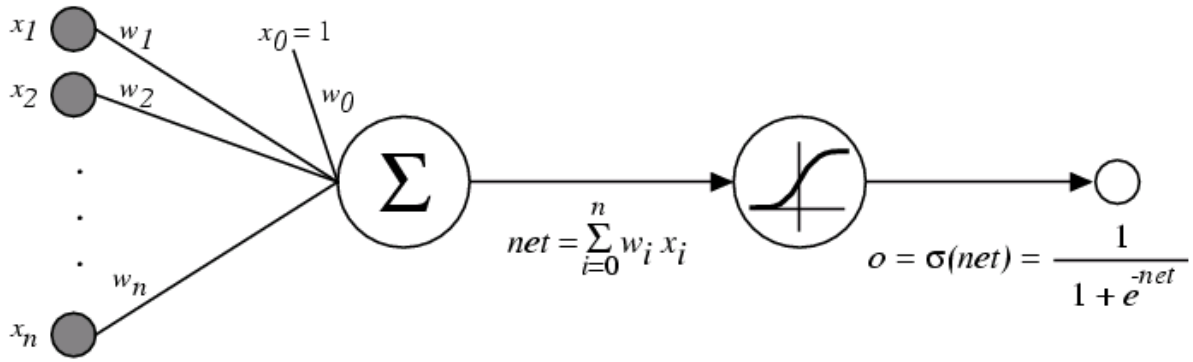
Die geschriebene Ziffer wird in eine Folge von Nullen und Einsen umgewandelt, wobei *0* für leere und *1* für übermalte Rasterpunkte steht:



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



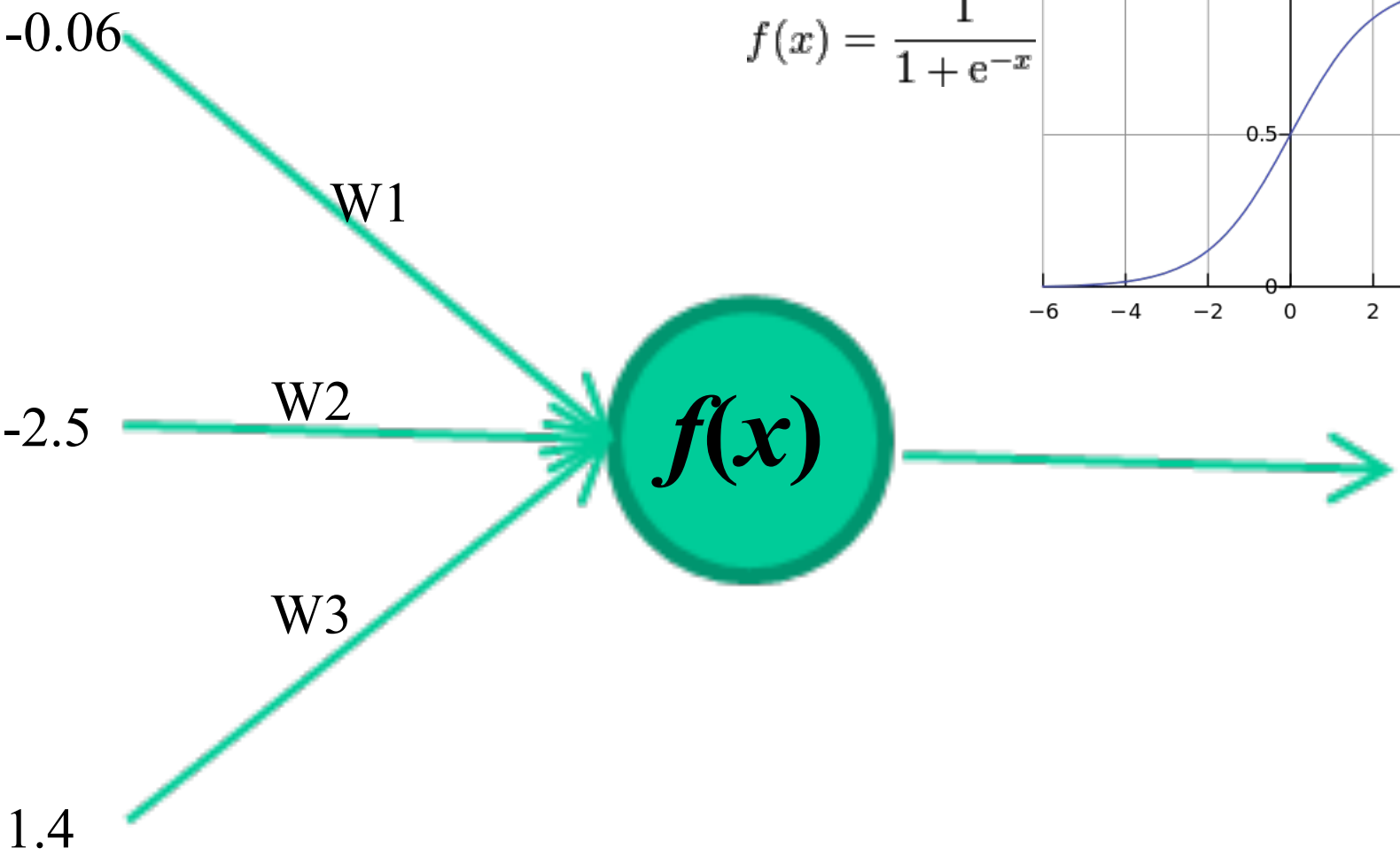
Verwendung kontinuierlicher Funktionen



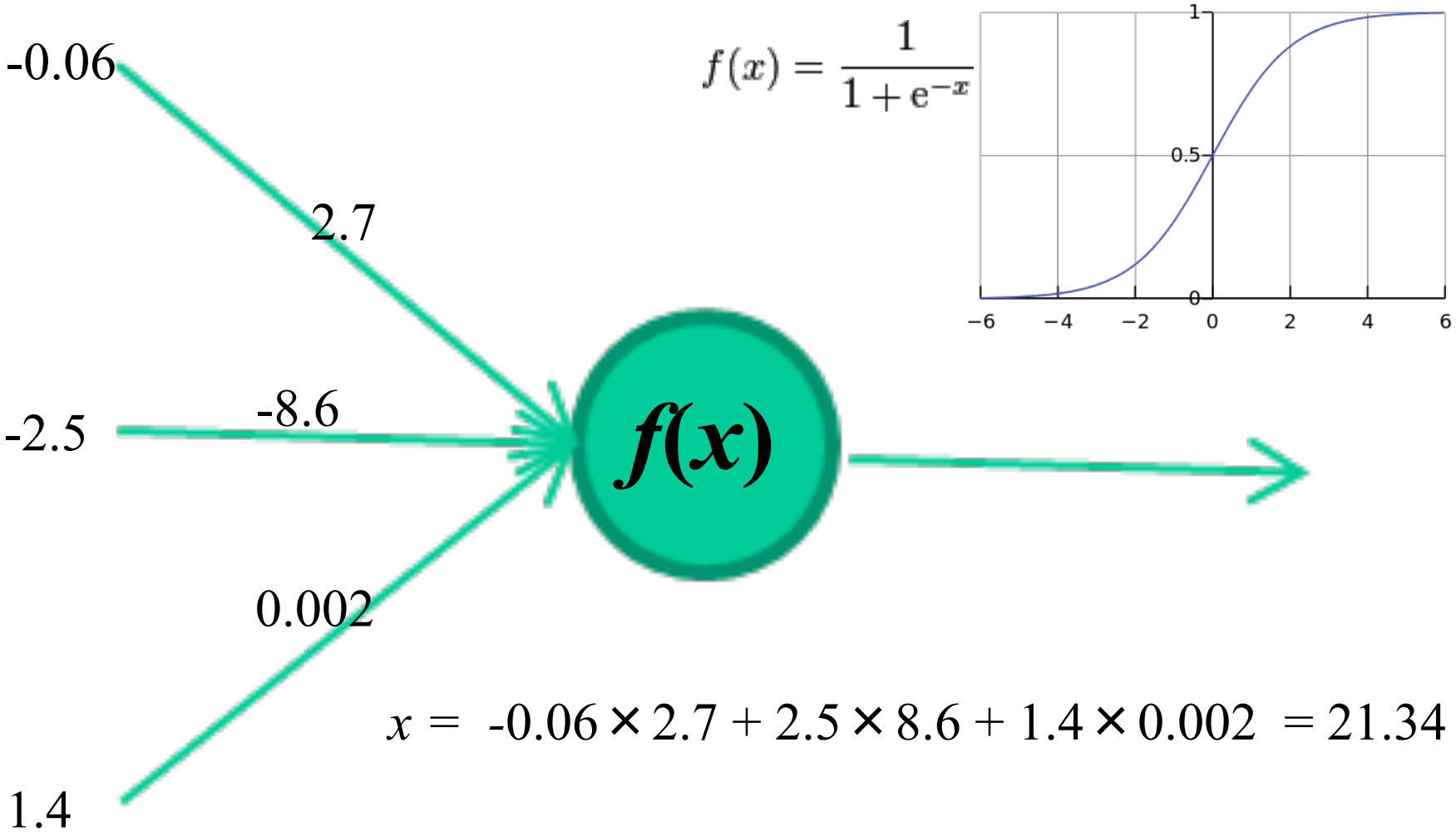
$\sigma(x)$ ist die Sigmoid-Funktion

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

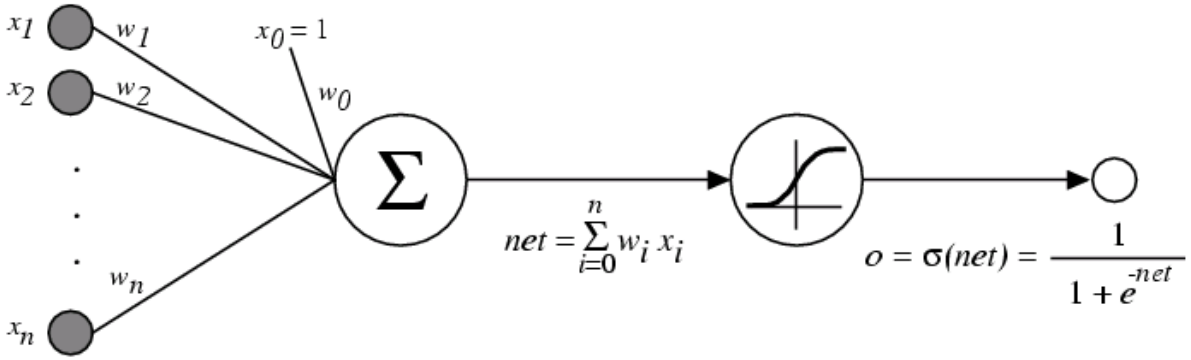
Beispiel einer Sigmoid-Funktion



Beispiel einer Sigmoid-Funktion



Sigmoid-Einheit



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

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

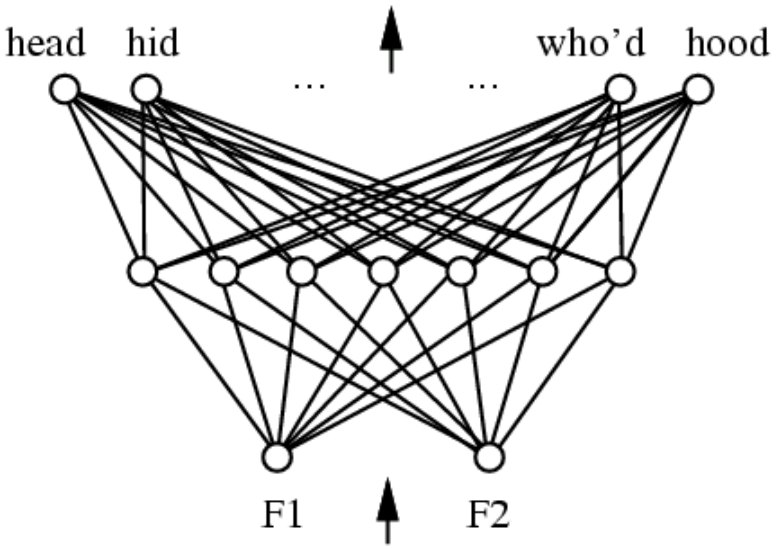
Eigenschaft: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$ (Gradient)

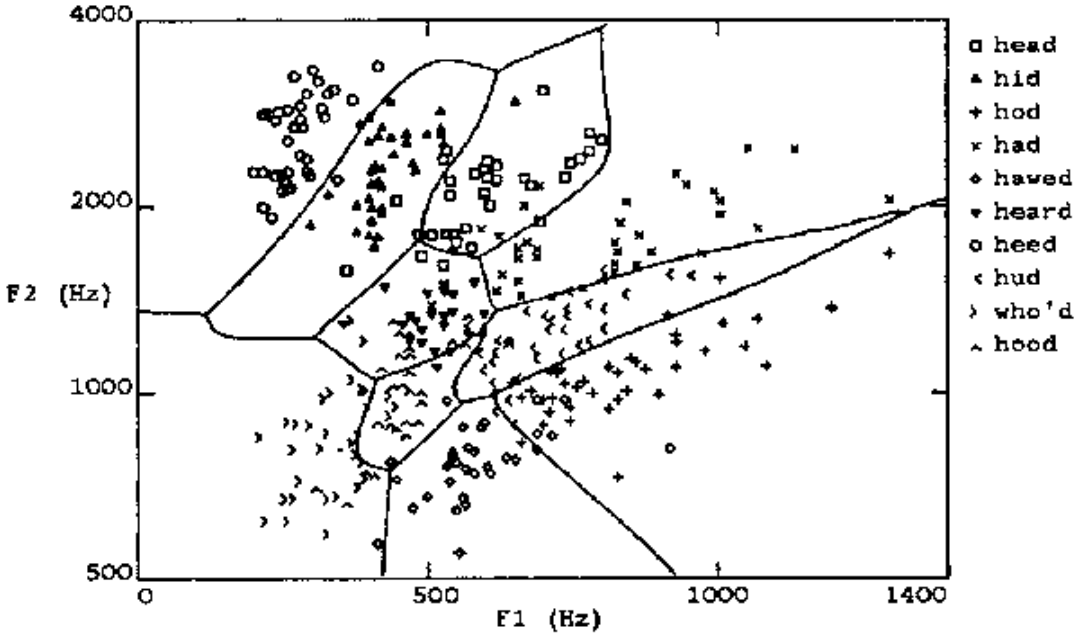
- Wir können den Gradienten verwenden, um die Einheit anzupassen
- Wir kommen gleich darauf zurück

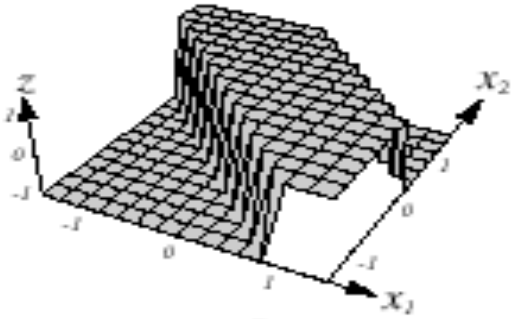
Differenzierbare Programmierung

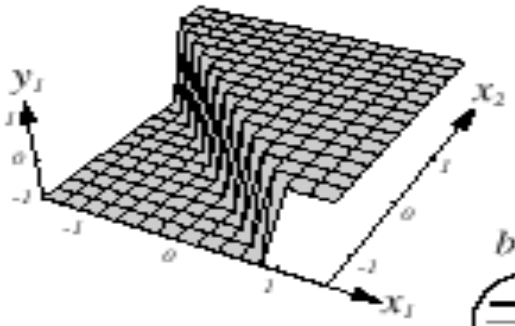
MEHR SCHICHTEN PERZEPTRONS

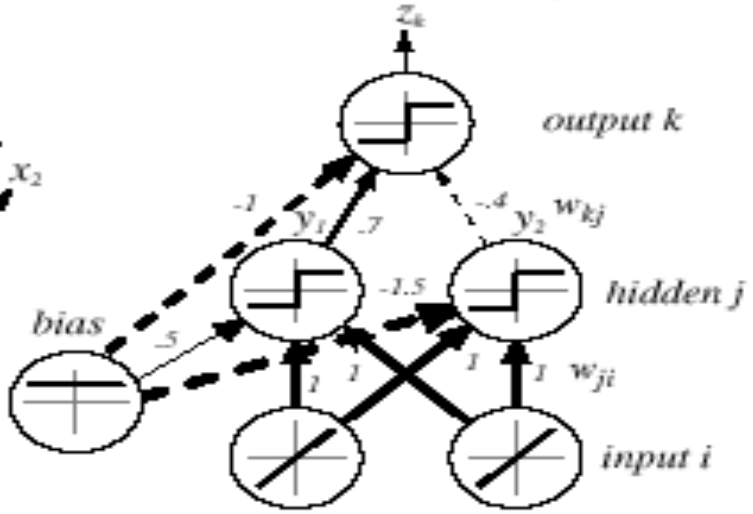
Mehr-Ebenen Netze von Sigmoid-Einheiten

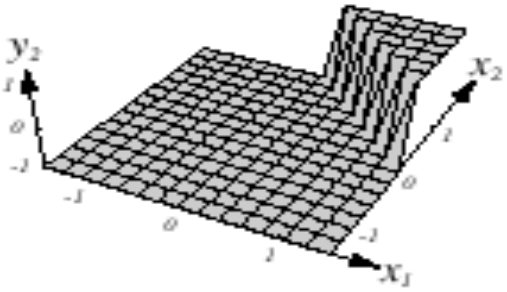




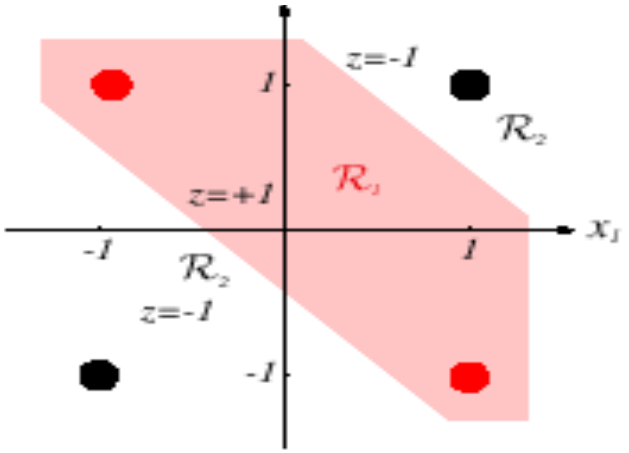








Z = x1 XOR x2 = (x1 OR x2) AND NOT(x1 AND x2)



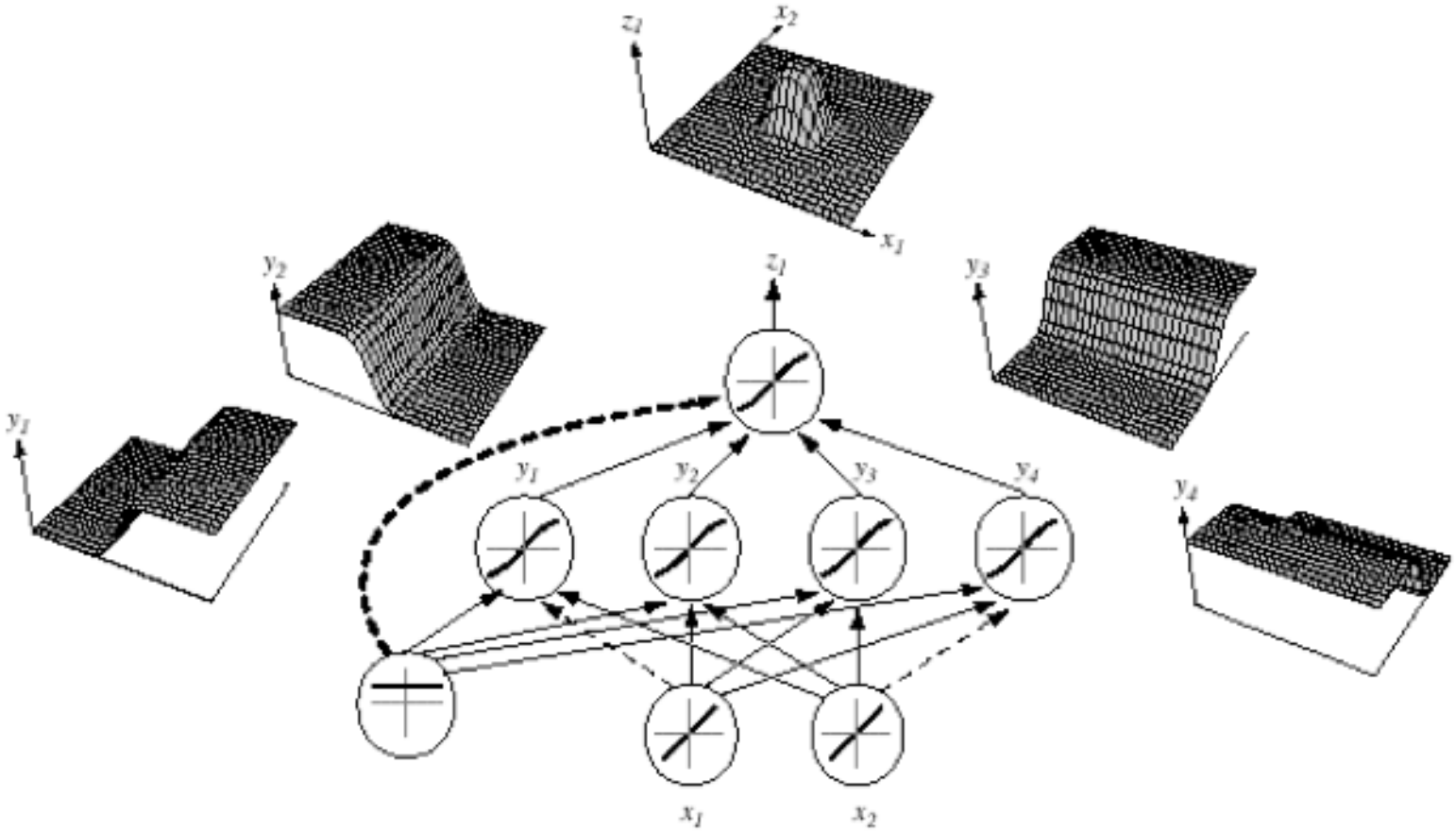


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large 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.

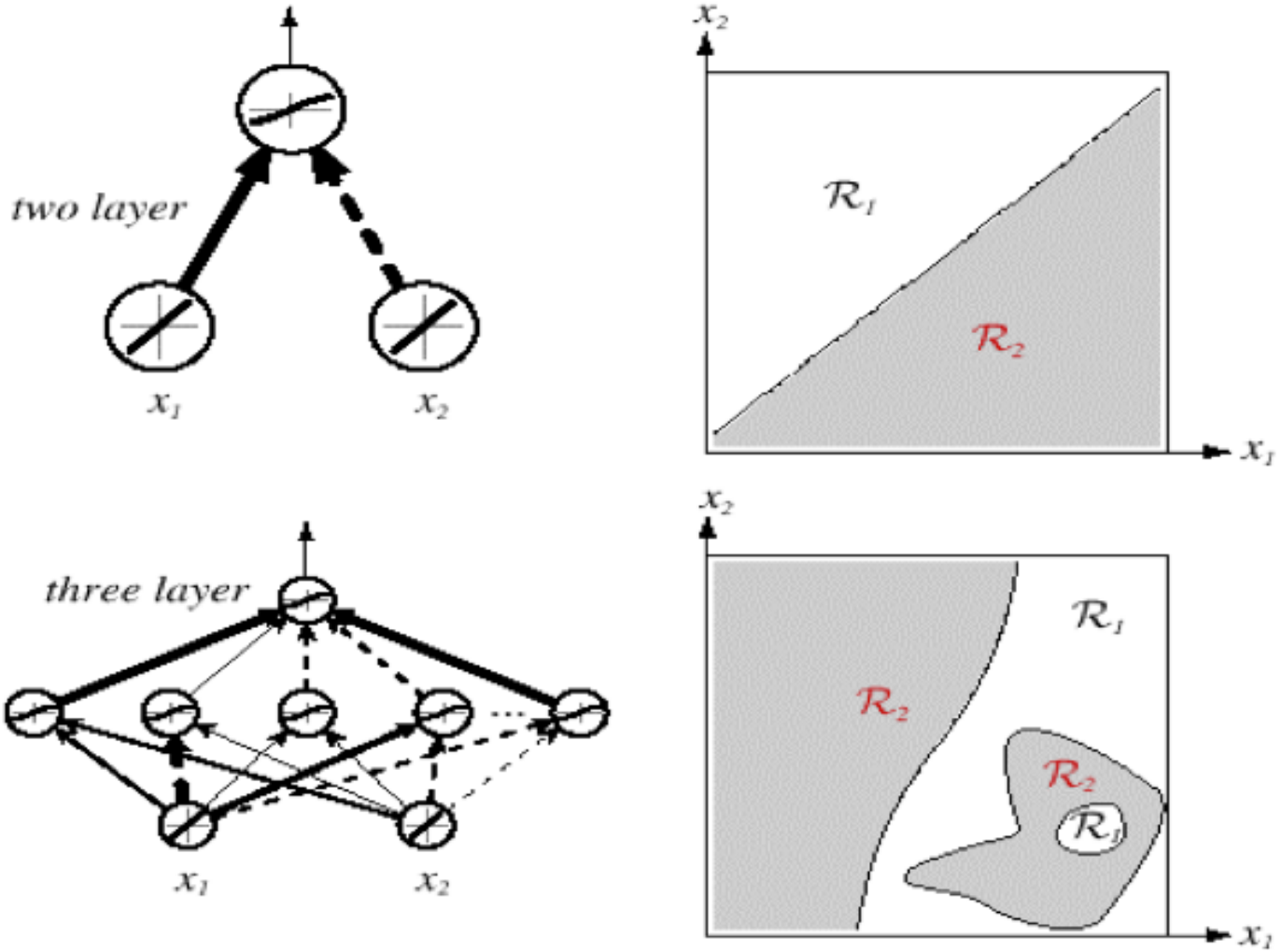
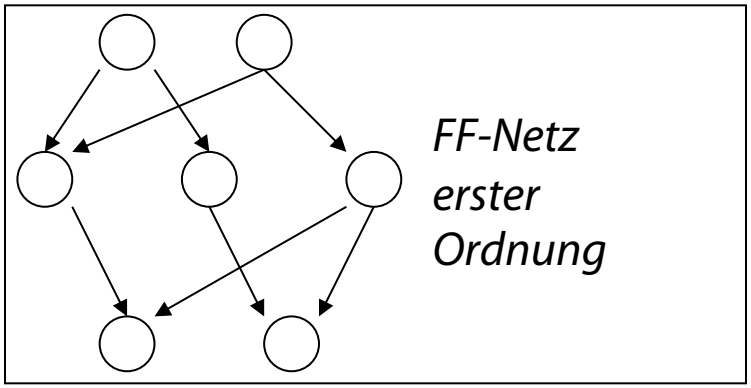


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O.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

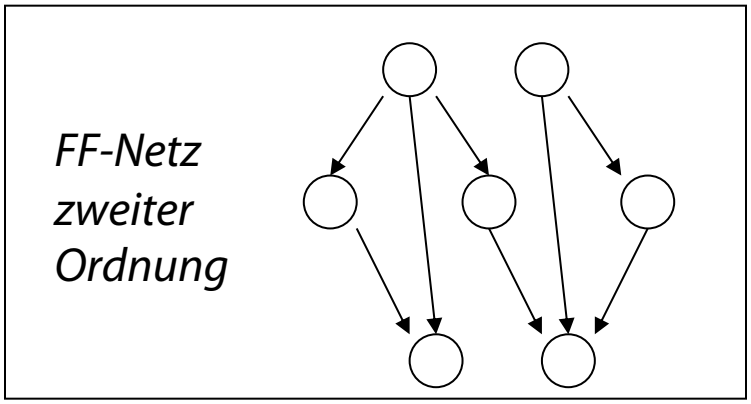
Netztopologien

FeedForward-Netze:

Gerichtete Verbindungen nur von niedrigen zu höheren Schichten



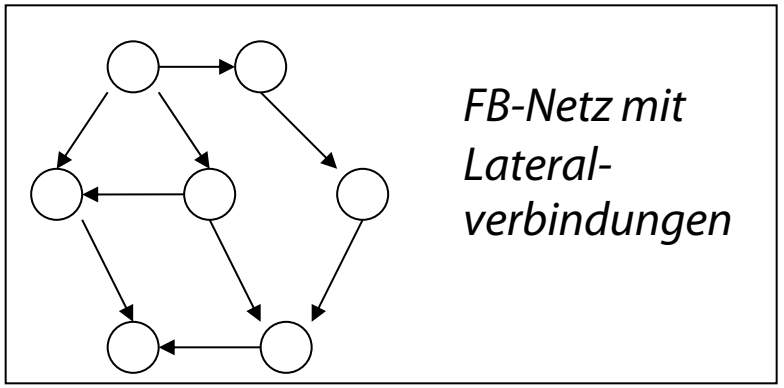
FF-Netz erster Ordnung



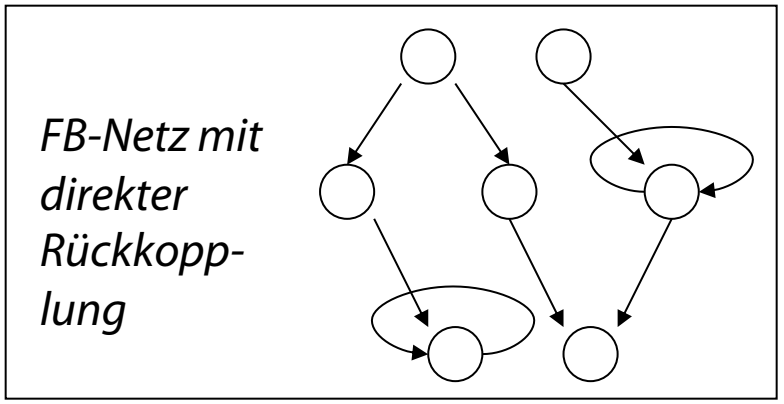
FF-Netz zweiter Ordnung

FeedBack-Netze (*rekurrente* Netze):

Verbindungen zwischen allen Schichten möglich (Abrollen)



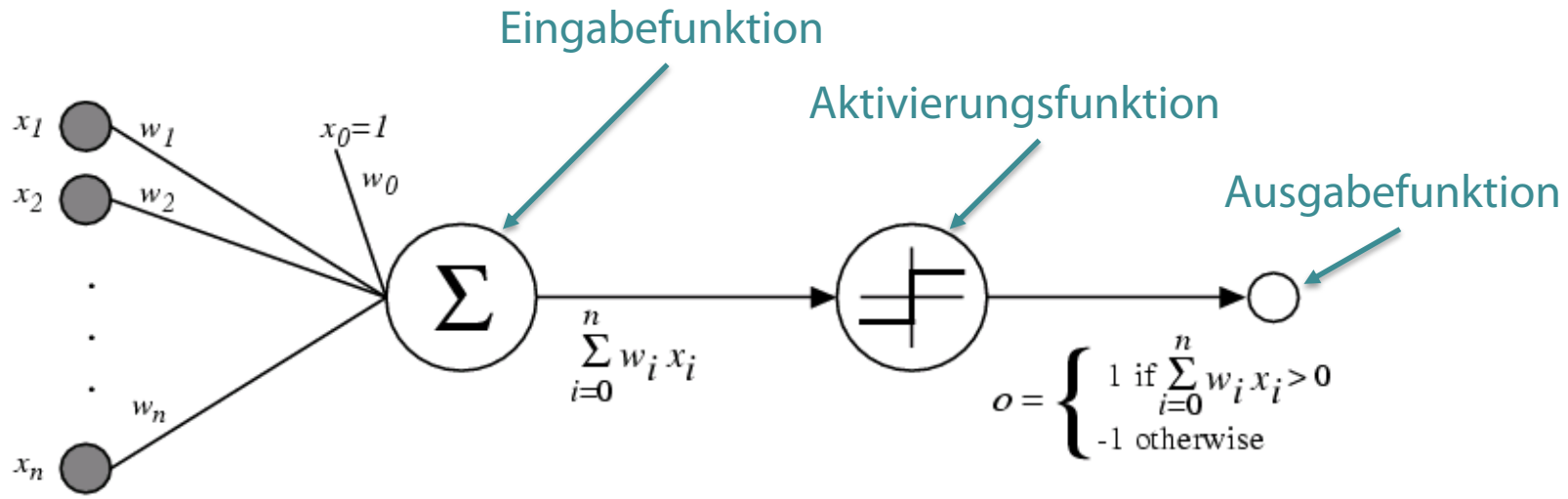
FB-Netz mit Lateral-verbindungen



FB-Netz mit direkter Rückkopp-lung

Netzwerk von Perzeptrons

- Perzeptron:



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

Differenzierbare Programmierung

BEGRIFFSBESTIMMUNG

Die Eingabefunktion

Die Eingabe- oder Propagierungsfunktion berechnet aus dem Eingabevektor $\vec{x} = (x_1, \ldots, x_m)$ und dem Gewichtsvektor $\vec{w}_k = (w_{1,k}, \ldots, w_{m,k})$ 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

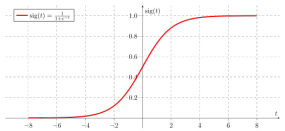
Mit der Aktivierungsfunktion (auch: Transferfunktion) wird aus dem Nettoinput net_k der Aktivierungszustand a_k eines Knotens berechnet.

Folgende Aktivierungsfunktionen sind gebräuchlich:

- Lineare Aktivierungsfkt.: $a_k = f_{act}(net_k) = c_k \cdot net_k$

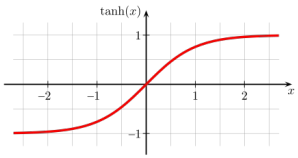
Schwellenwert, häufig 0

- 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.): $a_k = f_{act}(net_k) = \frac{1}{1+e^{-\frac{net_k}{T}}}$



Spezialfall: Sigmoid: T=1

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

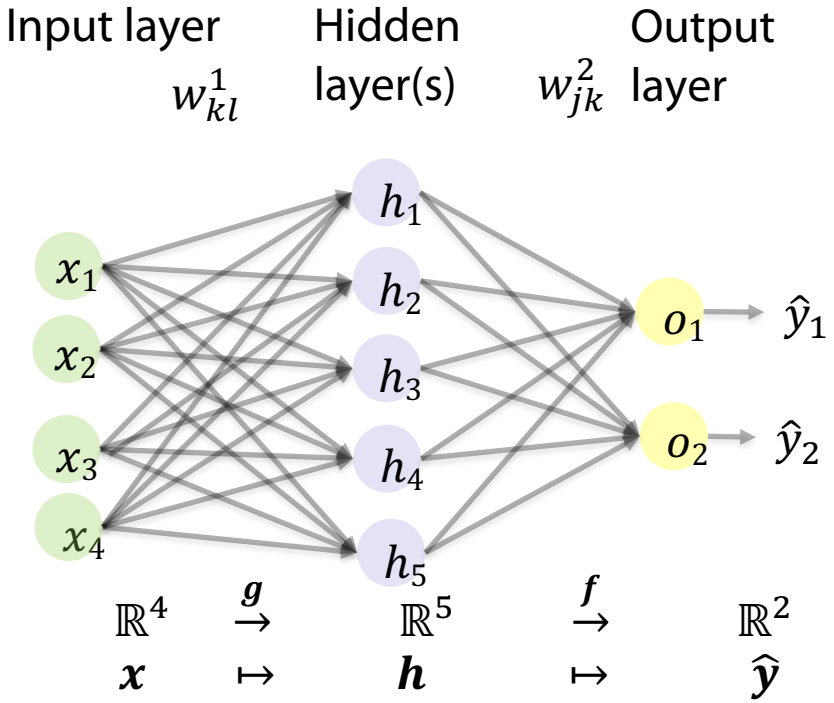


Die Ausgabefunktion

- 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:

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

Beispiel



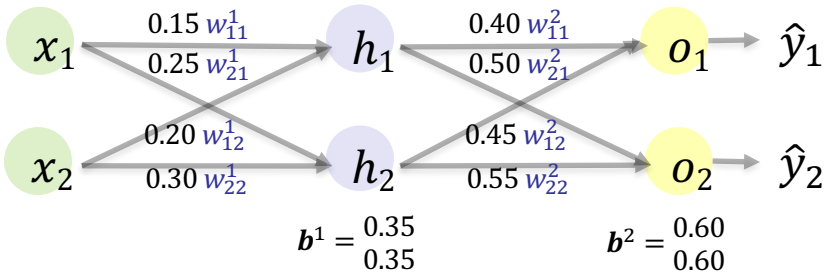
i:	Ebene i
$\boldsymbol{b}^i$:	Verzerrung für i
$\boldsymbol{W}^i$:	Gewichtsmatrix für i
$\boldsymbol{\sigma}^i$:	Aktivierungsfkt. für i

$\boldsymbol{out}^i$: $\boldsymbol{\sigma}^i(\boldsymbol{W}^i\boldsymbol{out}^{i-1} + \boldsymbol{b}^i)$
Aktivierung in i

$\boldsymbol{net}^i$: $\boldsymbol{W}^i\boldsymbol{out}^{i-1} + \boldsymbol{b}^i$
Lineare Ausgabe in Ebene i

$$\hat{\boldsymbol{y}} = \boldsymbol{f}\,(\boldsymbol{g}\,(\boldsymbol{x}\,;\,\boldsymbol{W}^1\,,\boldsymbol{b}^1\,);\,\boldsymbol{W}^2\,,\boldsymbol{b}^2) = \boldsymbol{\sigma}^2\,(\boldsymbol{W}^2\,\boldsymbol{\sigma}^1(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1\,) + \boldsymbol{b}^2)$$

Bespiel: Berechnung



Aktivierungsfkt:

$$\boldsymbol{\sigma_1, \sigma_2} = \frac{1}{1 + e^{-x}}$$

Eingabe:

$$\boldsymbol{x} = \begin{matrix}0.05\\0.10\end{matrix}$$

- $net_{h_1} = w_{11}^1 \cdot x_1 + w_{12}^1 \cdot x_2 + b_1^1$
- $net_{h_1} = 0.15 \cdot 0.05 + 0.20 \cdot 0.10 + 0.35 = 0.3775$
- $out_{h_1} = \frac{1}{1+e^{-net_{h_1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$
- $out_{h_2} = 0.596884378$

- $\hat{y}_1 = out_{o_1} = 0.75136507$
- $\hat{y}_2 = out_{o_2} = 0.772928465$

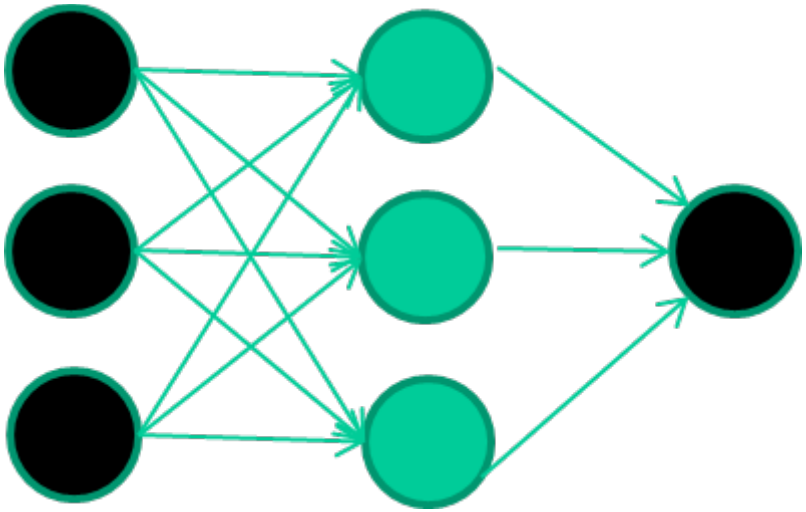
- $target_{o_1} = 0.01$
- $target_{o_2} = 0.99$

Differenzierbare Programmierung

LERNEN VON GEWICHTEN (IDEE)

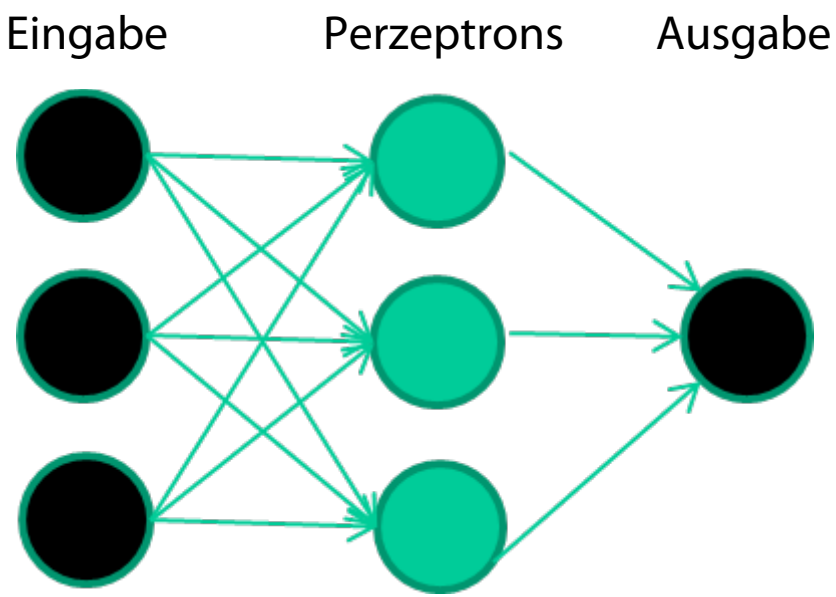
Einstellen der Gewichte mit Trainingsdaten…

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 …			



Anlernen des Netzwerks

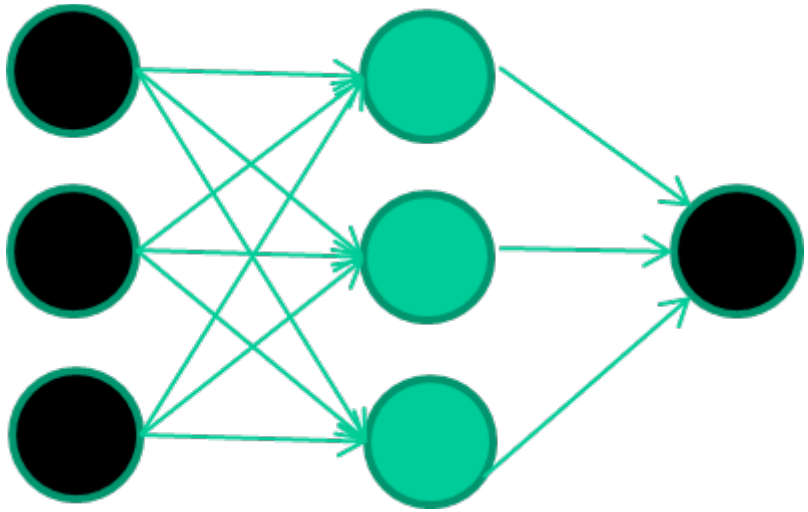
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



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 …			

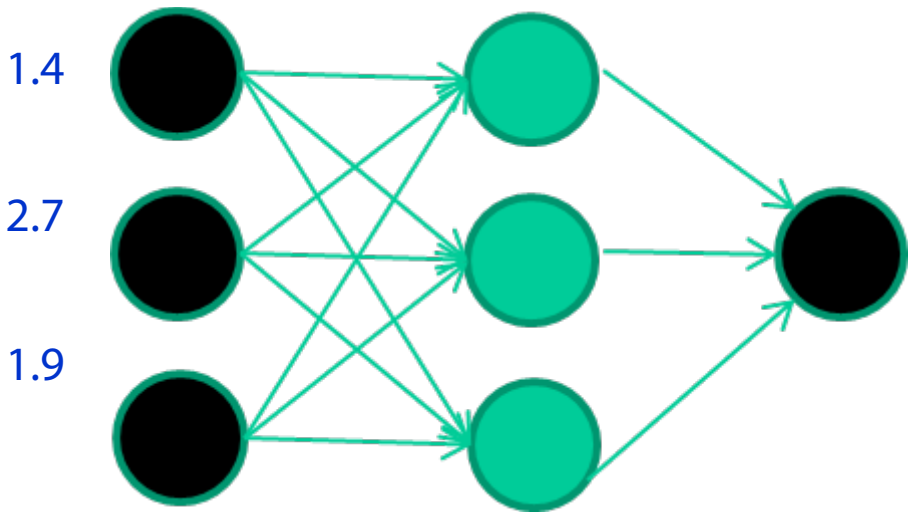
Initialisierung mit zufälligen Gewichten



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 …			

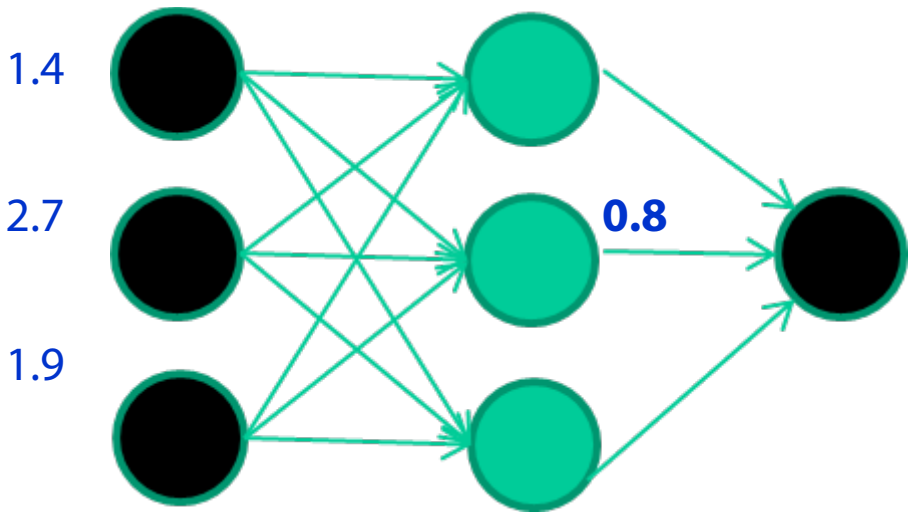
Präsentierung eines Trainingsdatensatzes



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 …			

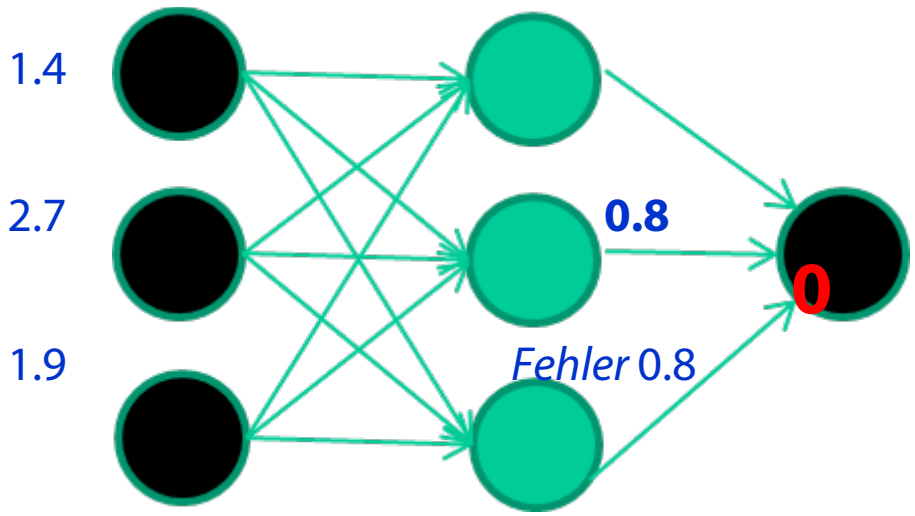
Durchpropagierung zur Ausgabe



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 …			

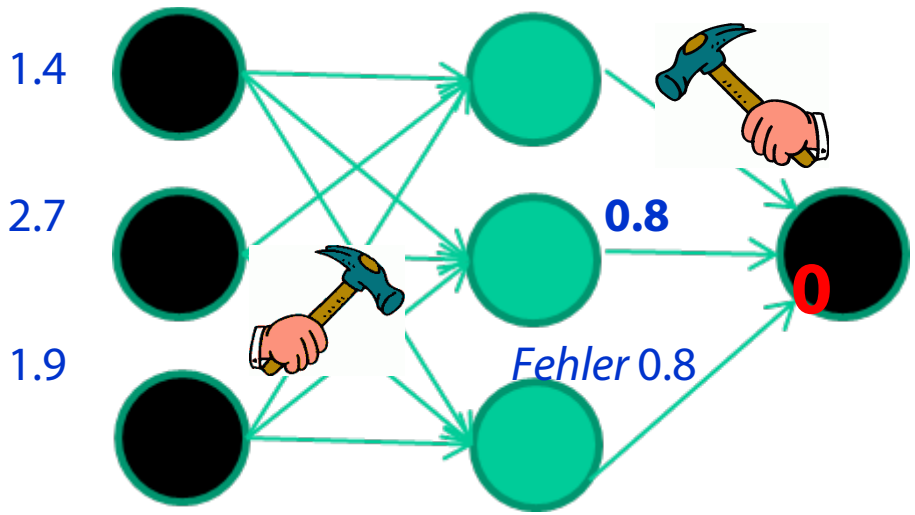
Vergleich mit der Zielausgabe



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 …			

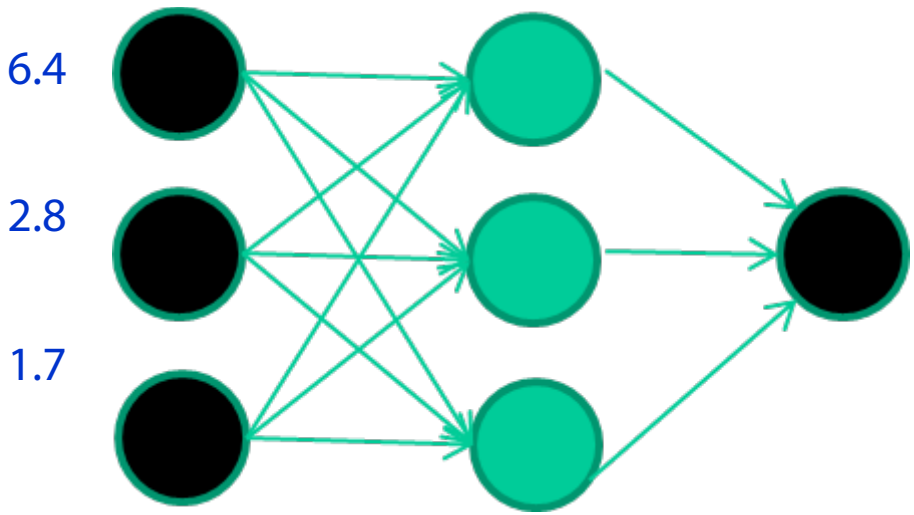
Anpassen der Gewichte gemäß Fehler



Trainingsdaten

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 …			

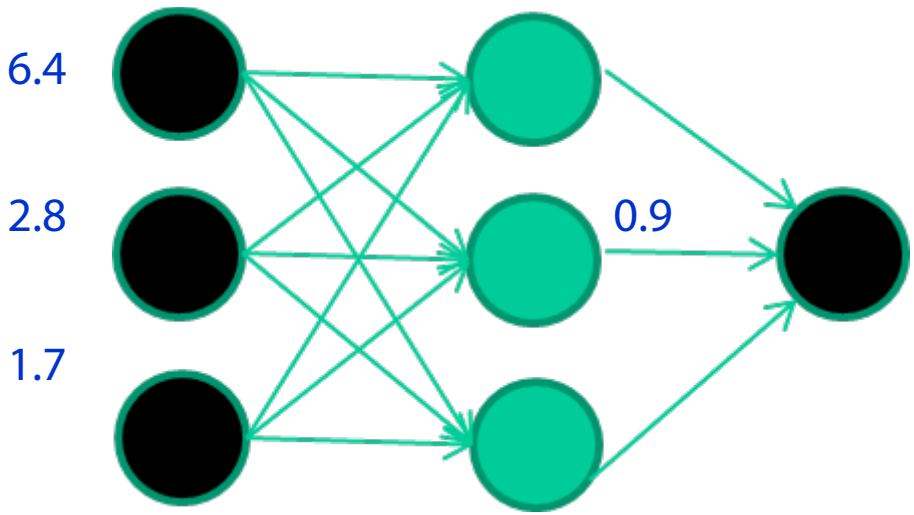
Präsentierung eines Trainingsdatensatzes



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 …			

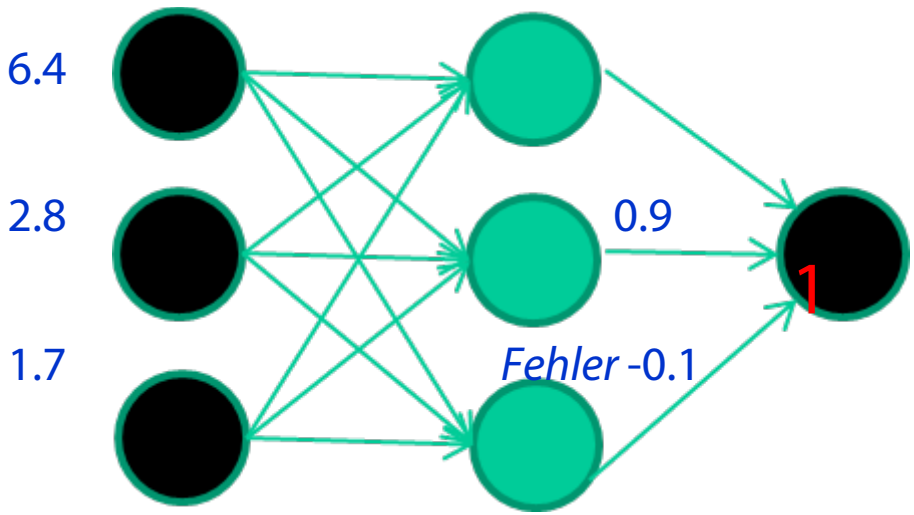
Durchpropagierung zur Ausgabe



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 …			

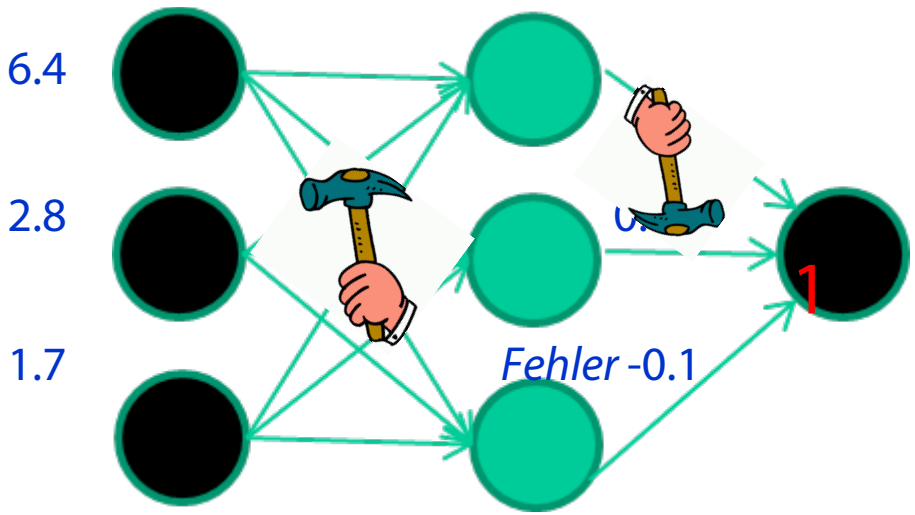
Vergleich mit der Zielausgabe



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 …			

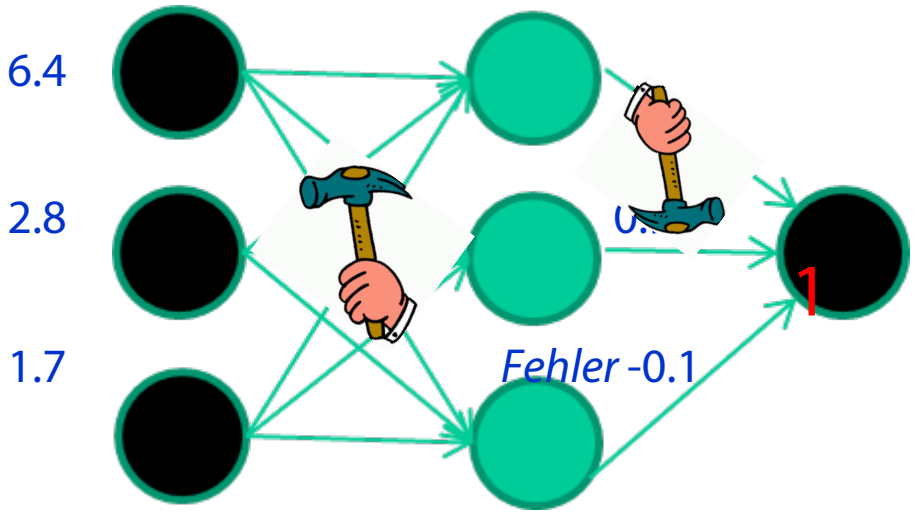
Anpassen der Gewichte gemäß Fehler



Trainingsdaten

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 …			

Und so weiter ….



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

Verfahren zur Gewichtsanpassung müssen Fehler minimieren

Differenzierbare Programmierung

LERNEN VON GEWICHTEN (EINE EBENE)

Eine Ebene: Perzeptron-Lernregel (Delta-Lernregel)

$$w_i \leftarrow w_i + \Delta \mathrm{w_i}$$

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

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

Begründung für die Delta-Regel

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

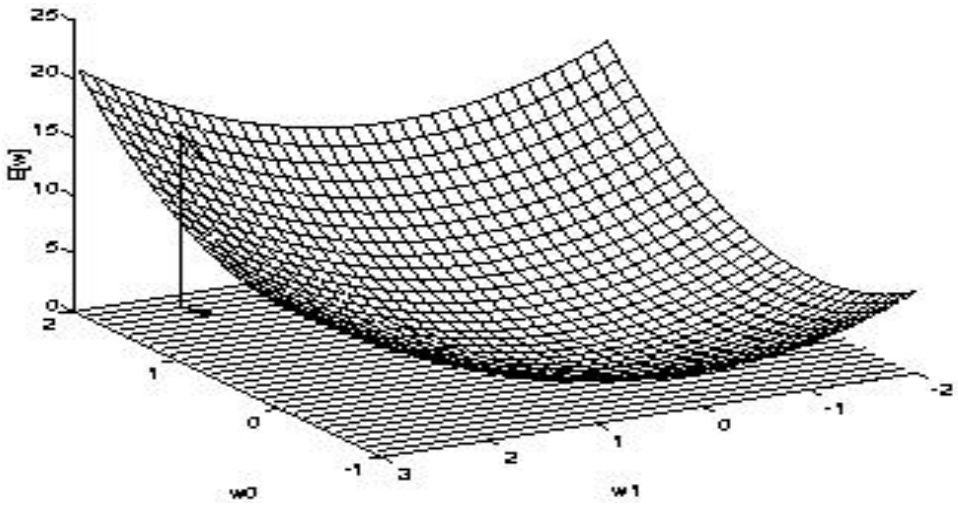
$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Minimumbestimmung mit 1. Ableitung

Absteigender Gradient

- Gradient

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

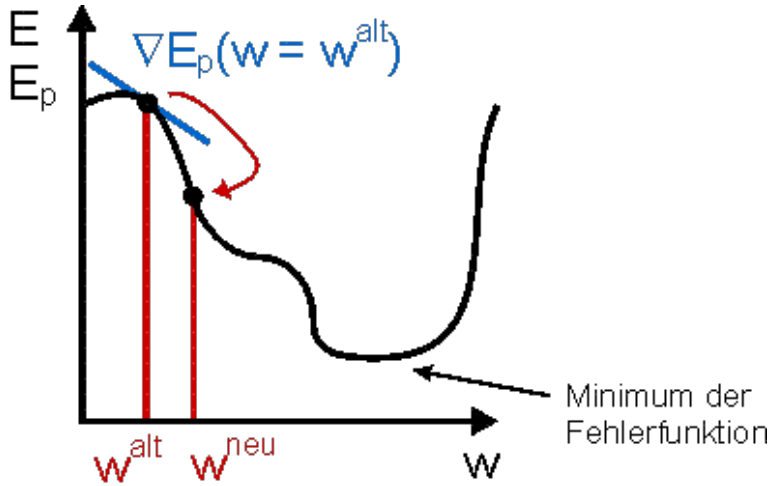


- Lernregel

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

- i.e.

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



Gradient

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$

$$= \sum_{d\in D}(t_d - o_d)\left(-x_{i,d}\right) = -\sum_{d\in D}(t_d - o_d)x_{i,d}$$

Absteigender Gradient (Forts.)

- Gradient

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

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

- 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* Gewichtsanpassung

- Jedes Trainingsbeispiel sei ein Paar $\langle \vec{x}, t \rangle$
  - $\vec{x}$ ist ein Inputvektor (Vektor mit Feature-Werten)
  - t is 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**

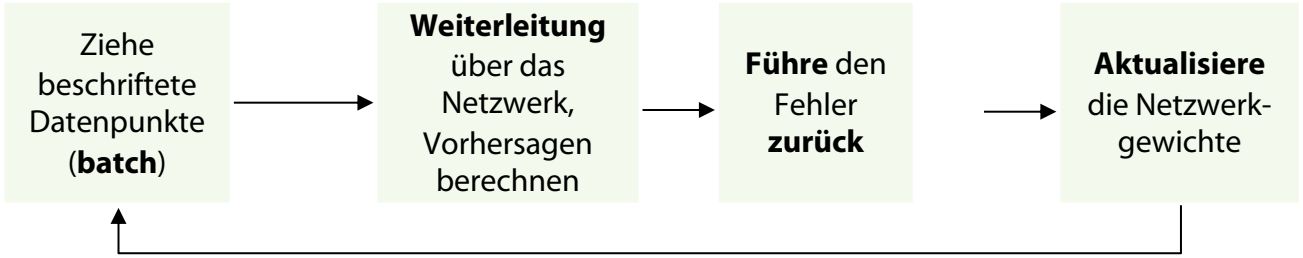
Netzwerke daher von manchen
als künstliche neuronale Netze bezeichnet

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

Differenzierbare Programmierung

LERNEN VON GEWICHTEN (MEHRE EBENEN)

Backpropagating Idee



Backpropagation idee

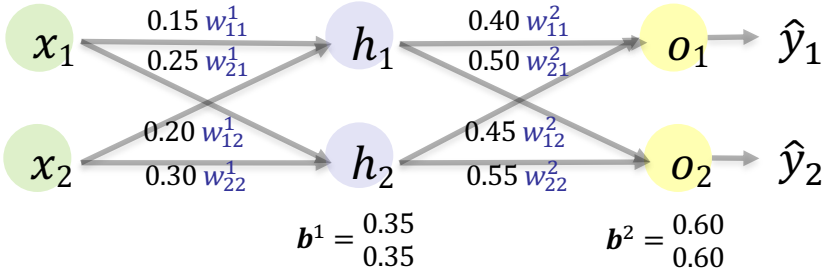
- Erzeuge ein **Fehlersignal**, das die Differenz zwischen Prognosen und Zielwerten misst
- Verwenden Sie das Fehlersignal, um die Gewichte zu ändern und genauere Vorhersagen rückwärts zu erhalten
- Zugrunde liegende Mathematik: Kettenregel

Kettenregel (1-dim)

$$\frac{dh}{dx} = \frac{df}{dg}\frac{dg}{dx} \quad (\text{für } h(x) = f(g(x)))$$

Idee: D. Hebb: The organization of behavior. A neuropsychological theory. Erlbaum Books, Mahwah, N.J., **1949**
Für neuronale Netze: D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors, 1986

Beispiel: Backward Pass für E_{total}



- $\hat{y}_1 = out_{o_1} = 0.75136507$ vs $target_{o_1} = 0.01$
- $\hat{y}_2 = out_{o_2} = 0.772928465$ vs $target_{o_2} = 0.99$

Aktivierungsfkt.:

$$\boldsymbol{\sigma_1, \sigma_2} = \frac{1}{1 + e^{-\boldsymbol{x}}}$$

- $E_{o_1} = \frac{1}{2}\left(target_{o_1} - out_{o_1}\right)^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.27481108$
- $E_{o_2} = 0.023560026$
- $E_{total} = E_{o_1} + E_{o_2} = 0.298371109$

Fehlerfkt.:

$$E_{total} = \sum \frac{1}{2}\,(target - out)^2$$

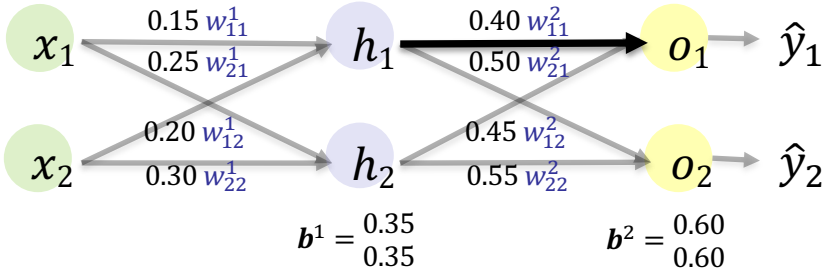
Eingabe:

$$\boldsymbol{x} = \begin{matrix}0.05\\0.10\end{matrix}$$

Zielausgabe:

$$target = \begin{matrix}0.01\\0.99\end{matrix}$$

Beispiel: Backward Pass für w_{11}^2



Aktivierungsfkt

$$\boldsymbol{\sigma_1, \sigma_2} = \frac{1}{1 + e^{-x}}$$

Eingabe:

$$\boldsymbol{x} = \begin{matrix} 0.05 \\ 0.10 \end{matrix}$$

Fehlerfkt:

$$E_{total}$$

$$= \sum \frac{1}{2} (target - out)^2$$

Zielausgabe:

$$target = \begin{matrix} 0.01 \\ 0.99 \end{matrix}$$

- $\frac{\partial E_{total}}{\partial E_{w_{11}^2}} = \frac{\partial E_{total}}{\partial out_{o_1}} \cdot \frac{\partial out_{o_1}}{\partial net_{o_1}} \cdot \frac{\partial net_{o_1}}{\partial E_{w_{11}^2}}$
- $\frac{\partial E_{total}}{\partial out_{o_1}} = 2 \cdot \frac{1}{2} \left(target_{o_1} - out_{o_1}\right)^{2-1} \cdot (-1) + 0$
  $= out_{o_1} - target_{o_1} = 0.75136507 - 0.01$
  $= 0.74136507$
- $\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1} \cdot \left(1 - out_{o_1}\right) = 0.186815602$
- $\frac{\partial net_{o_1}}{\partial E_{w_{11}^2}} = 1 \cdot out$ (…)
- $\frac{\partial E_{total}}{\partial E_{w_{11}^2}} =$ (…) 41

Welcher Fehler sollte als nächstes propagiert werden?

δ_{o_1} out_{h_1}

- $new\_w_{11}^2 = w_{11}^2 - \frac{\partial E_{total}}{\partial E_{w_{11}^2}} = 0.317832959$

Backpropagation Algorithmus (Eine Runde)

1. Eingabe: Initialisieren des Eingabevektors $\boldsymbol{x} = \boldsymbol{out}^0$
2. Feedforward: Für i = 1,2, … , M

$$\boldsymbol{net^i} = \boldsymbol{W^{(i)}out^{i-1}} + \boldsymbol{b_i} \text{ and } \boldsymbol{out^i} = \boldsymbol{\sigma_i}(\boldsymbol{net^i})$$

3. Berechnen Sie den Fehler für die letzte Ebene

$$\boldsymbol{\delta}^M = \nabla_{\hat{\boldsymbol{y}}} E \odot \boldsymbol{\sigma}'(\boldsymbol{net}^M)$$

4. Fehler durch propagieren: Für i = M-1, M-2, …,

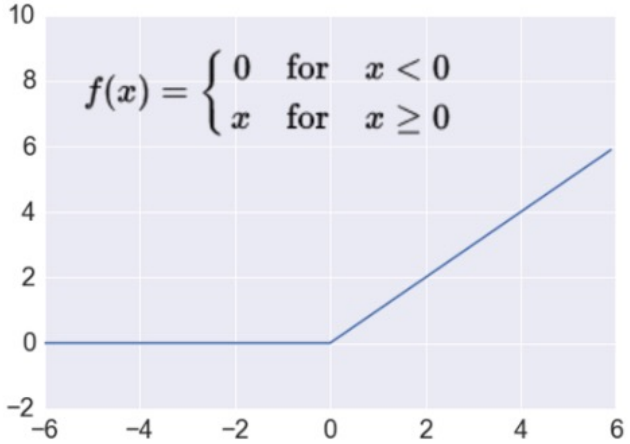
$$\boldsymbol{\delta}^i = (\boldsymbol{w^{i+1}})^\top \boldsymbol{\delta}^{i+1} \odot \boldsymbol{\sigma}'(\boldsymbol{net}^i)$$

5. Berechnen der Gradienten:

$$\frac{\partial E}{\partial w_{jk}^i} = out_k^{i-1}\delta_j^i \quad \text{und} \quad \frac{\partial E}{\partial b_j^i} = \delta_j^i$$

Hadamard product $\odot$: $\binom{a}{b} \odot \binom{c}{d} = \binom{ac}{bd}$

Aktivierung: ReLU (rectified linear unit)



http://adilmoujahid.com/images/activation.png

Nimmt eine reellwertige Zahl und legt einen Schwellenwert bei Null fest $f(x) = \max(0, x)$

$$R^n \rightarrow R^n_+$$

Die meisten Deep Netzwerke verwenden heutzutage ReLU

Trainiert viel **schneller**

- beschleunigt die Konvergenz der SGD
- aufgrund linearer, nicht beschränkter Form

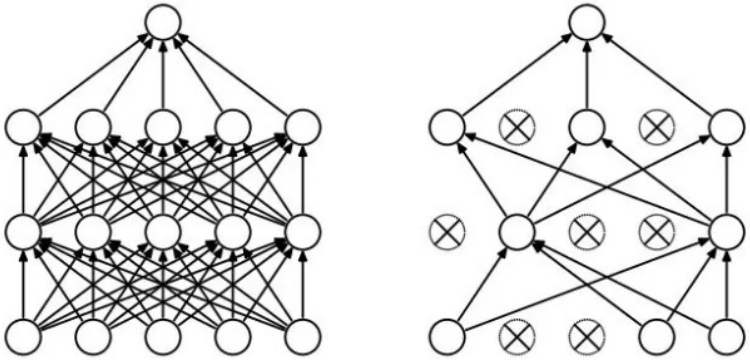
Weniger aufwändige Berechnungen

- im Vergleich zu Sigmoid/Tanh (Exponentiale etc.)
- Implementierung durch einfaches Schwellenwert einer Matrix bei Null

Ausdrucksstarker

Beugt dem **gradient vanishing problem** vor

Regularization



Dropout

- Zufälliges Ablegen von Einheiten (zusammen mit ihren Verbindungen) während des Trainings
- Jeder Knoten wird mit fester Wahrscheinlichkeit p beibehalten, unabhängig von anderen Knoten
- Hyperparameter p zu wählen (abgestimmt)

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research (2014)

L2 = Gewichtszerfall (weight decay)

- Regularisierungsterm, der große Gewichte bestraft, werden hinzugefügt
- Der Gewichtszerfallswert bestimmt, wie dominant die Regularisierung während der Gradientenberechnung ist
- Großer Gewichtszerfallswert → Große Strafe für große Gewichte

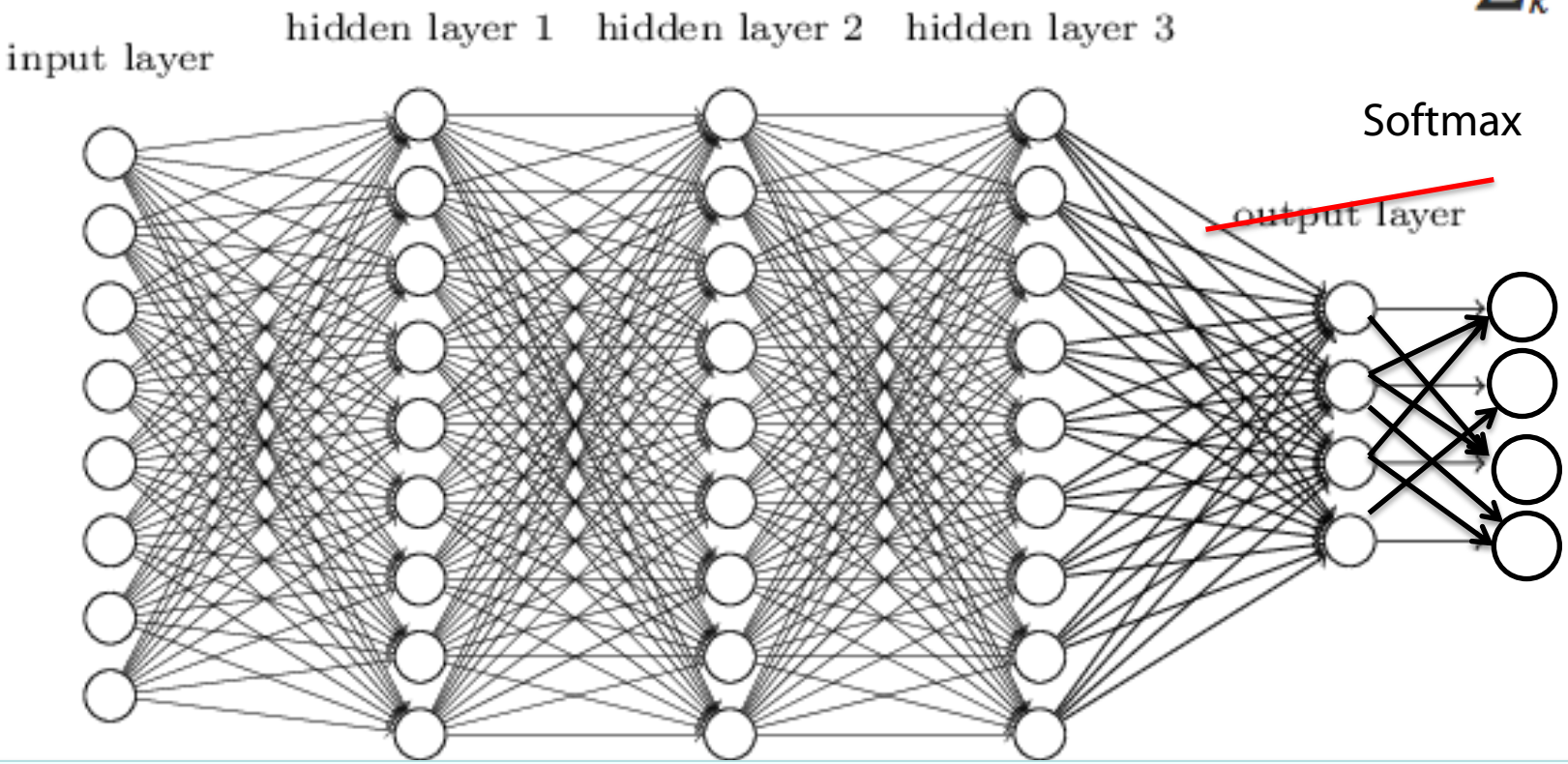
$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

Early-stopping

- Verwenden eines Validierungsfehlers, um zu entscheiden, wann das Training beendet warden soll
- Stoppe, wenn sich die überwachte Menge nach n aufeinanderfolgenden Epochen nicht verbessert hat
- n wird auch patience (Geduld?) genannt

Softmax Ausgabeschicht

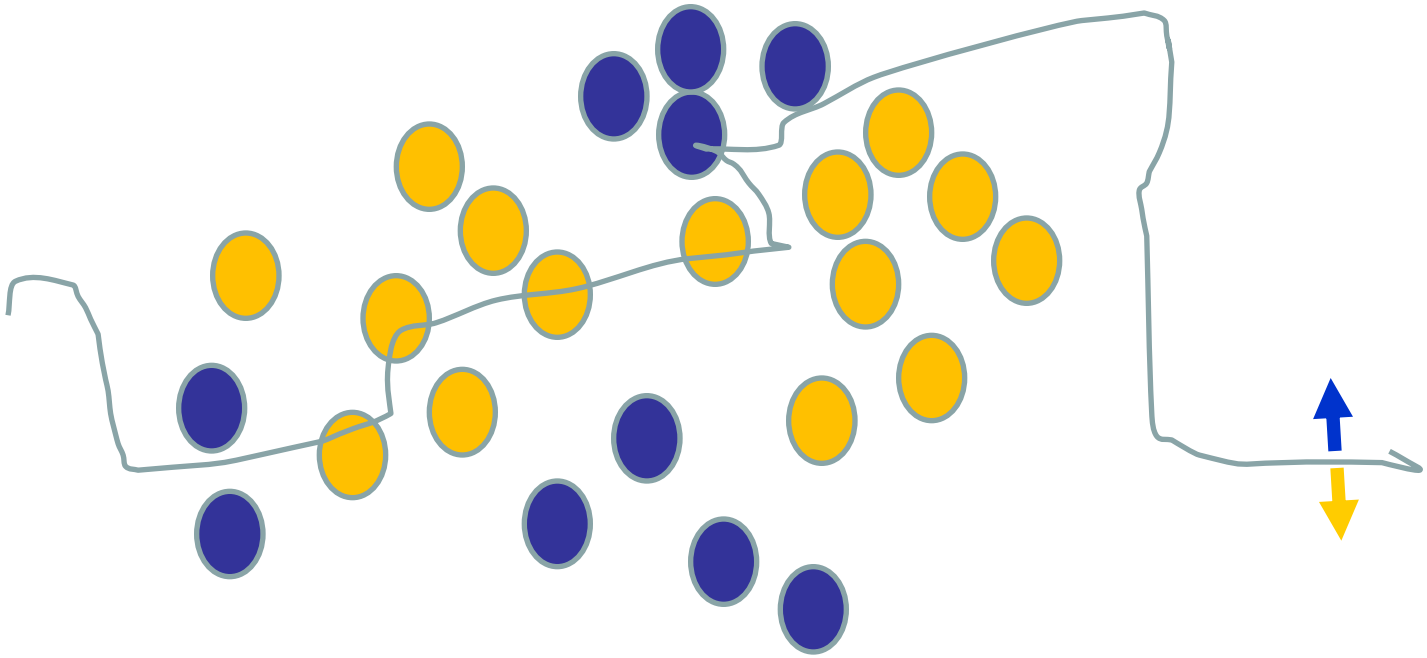
$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$



Netzwerk gibt eine Wahrscheinlichkeitsverteilung aus!

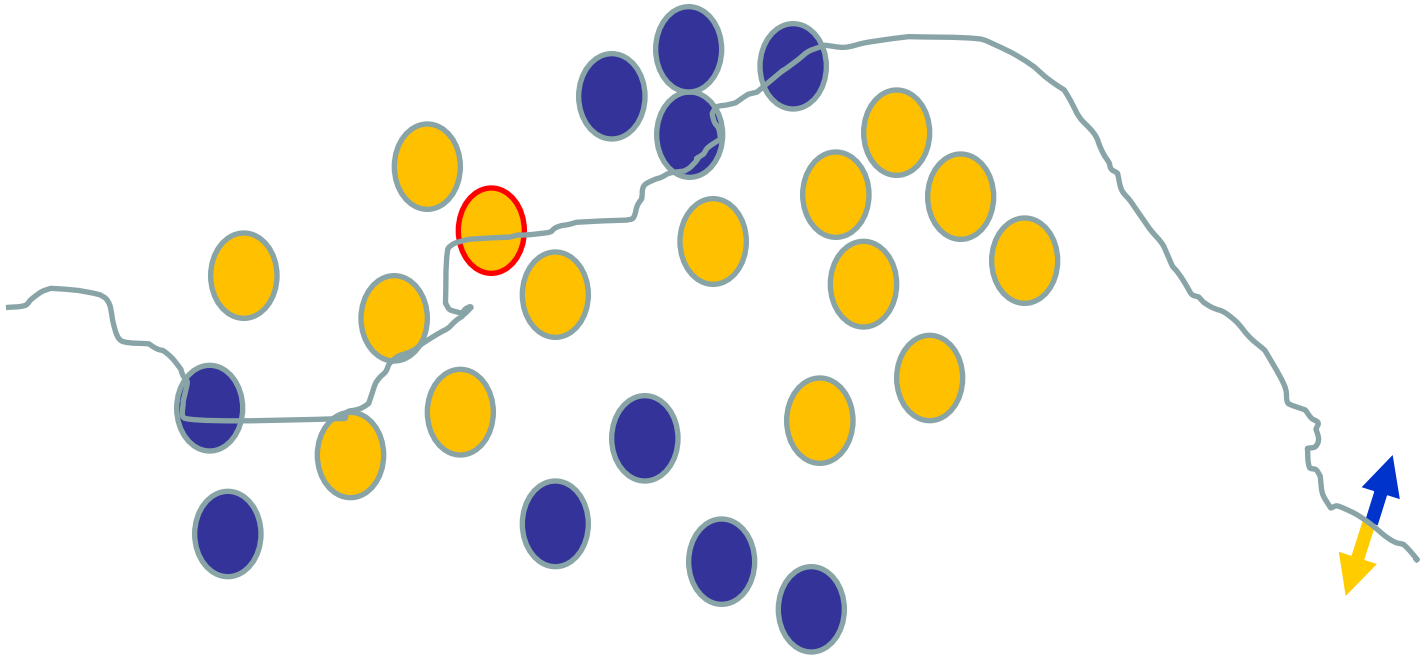
Perspektive der Entscheidungsgrenzenanpassung

Zufällige Initialgewichte



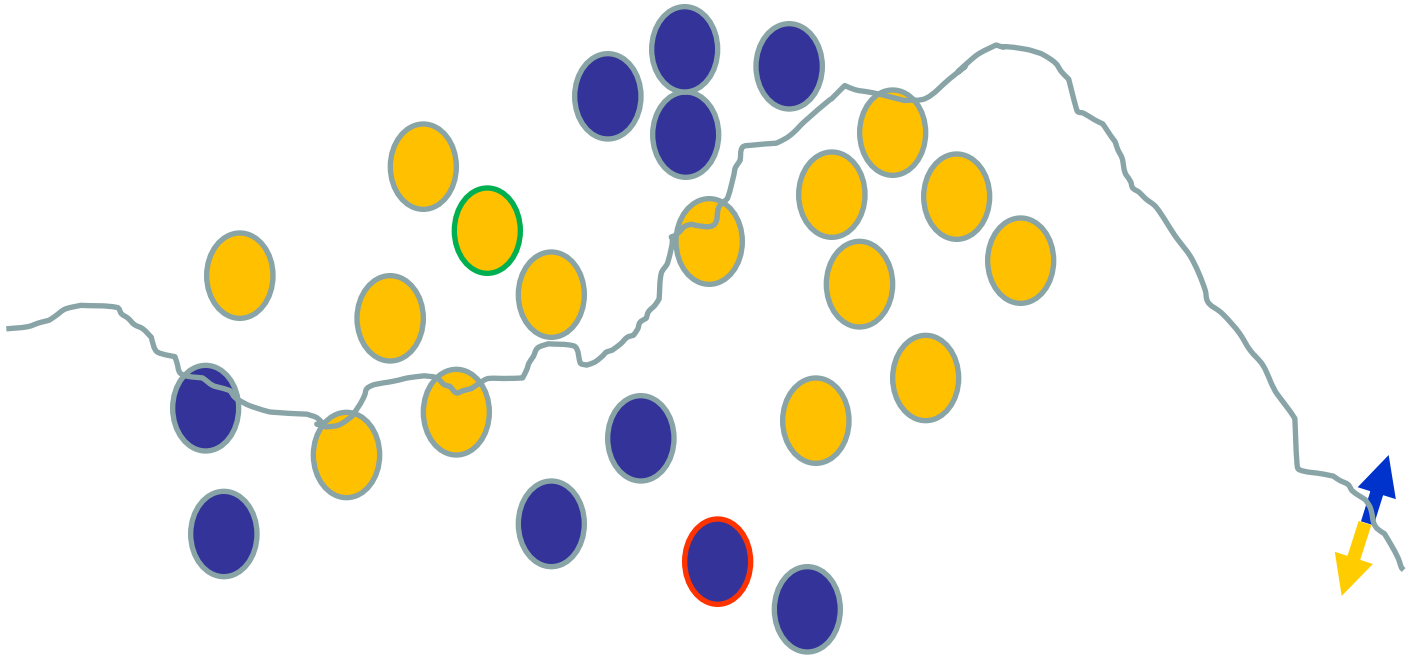
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte



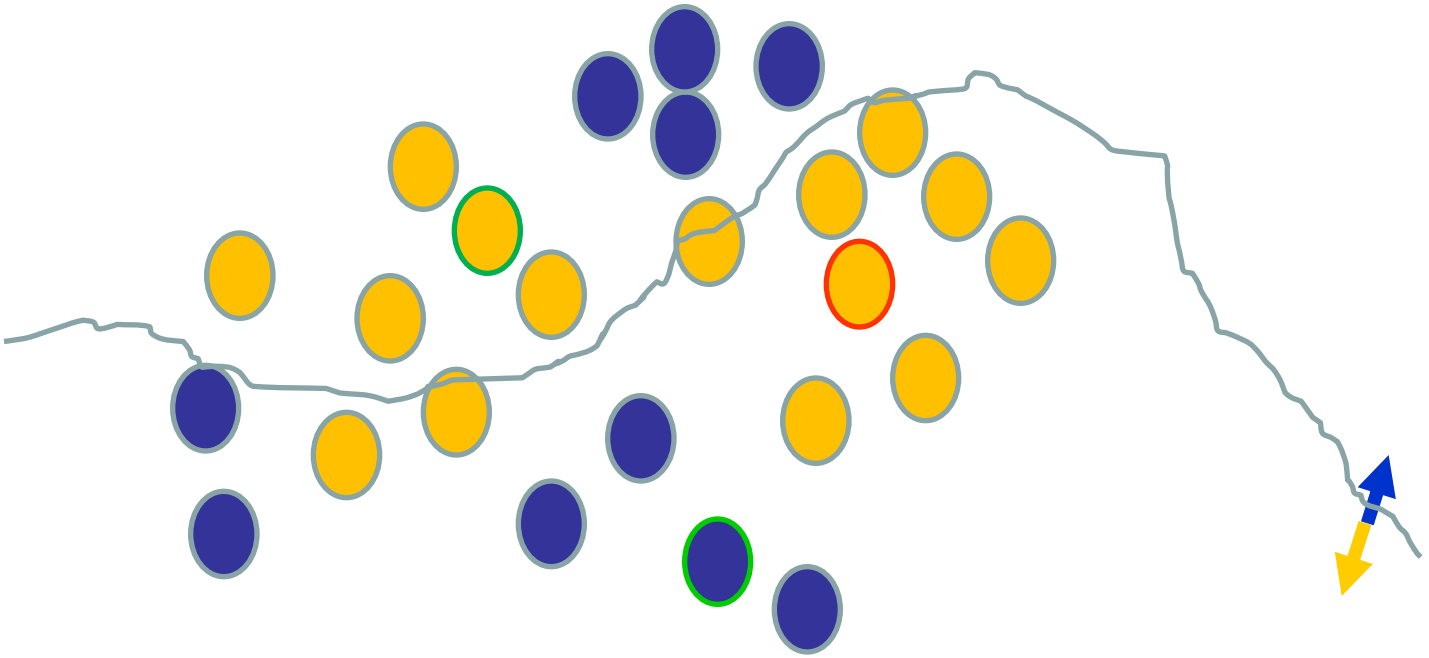
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte



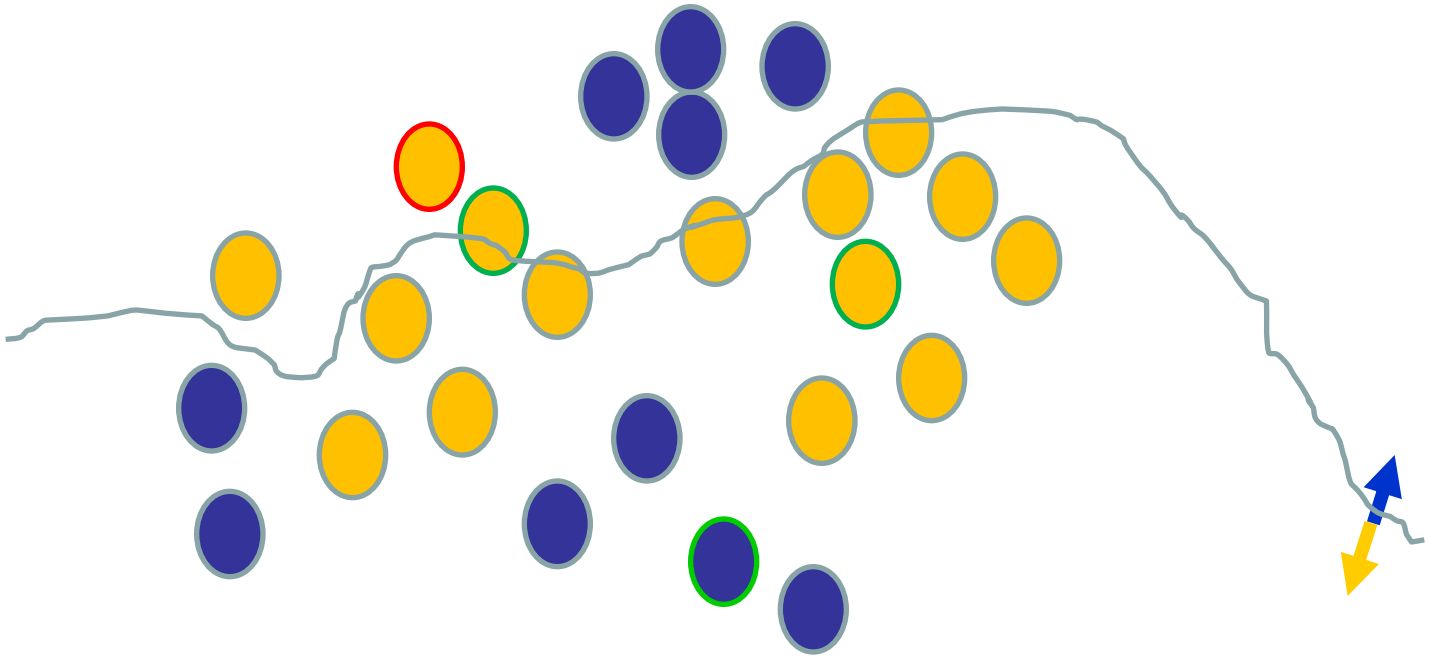
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte



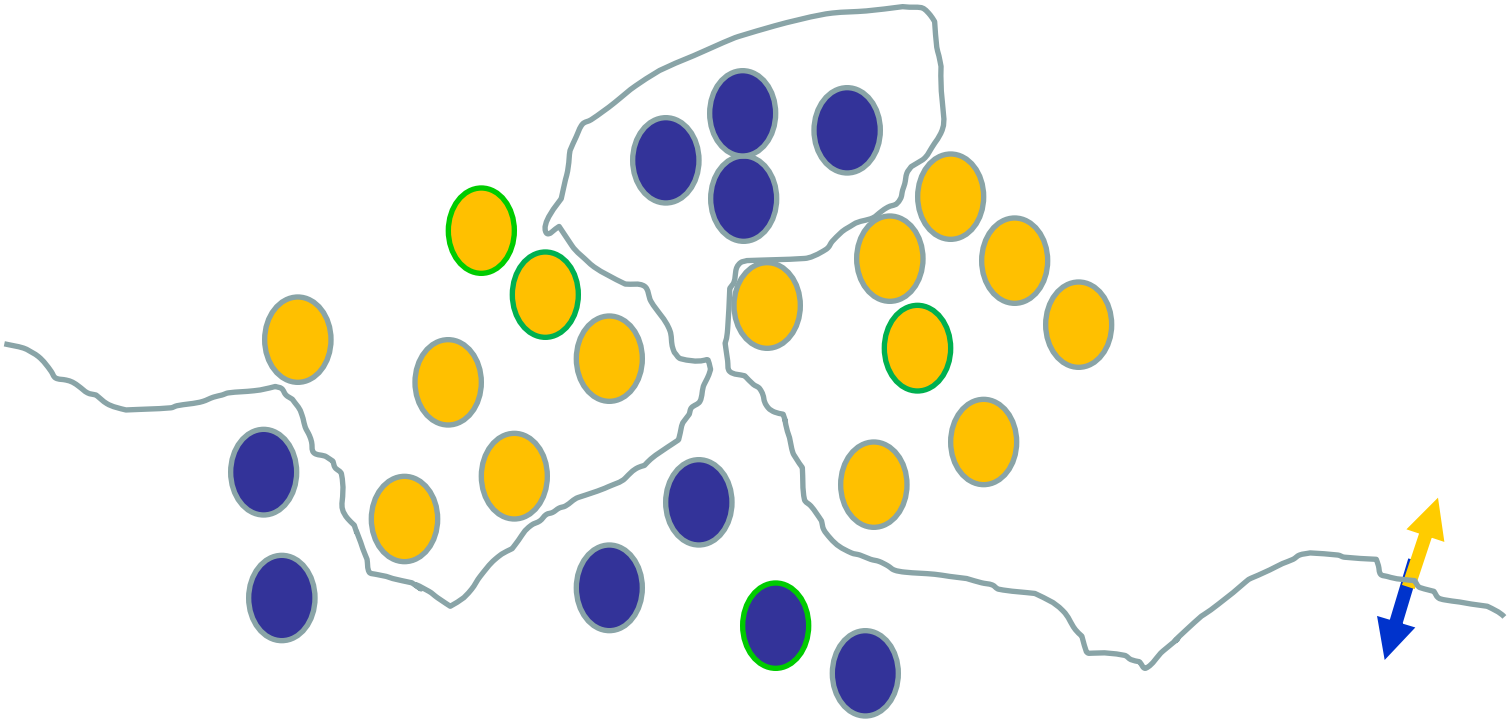
Perspektive der Entscheidungsgrenzenanpassung

Verwenden einer Trainingsinstanz / Anpassung der Gewichte

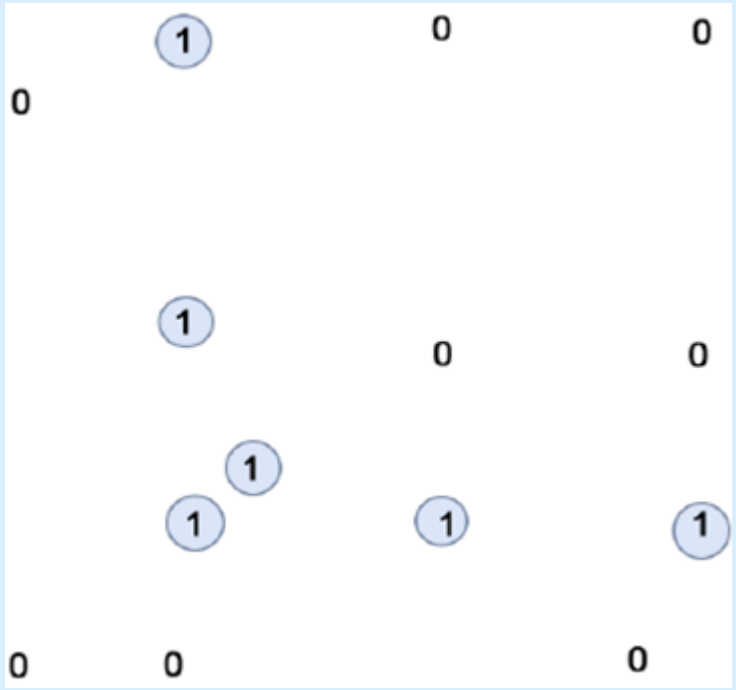


Perspektive der Entscheidungsgrenzenanpassung

Und schließlich….

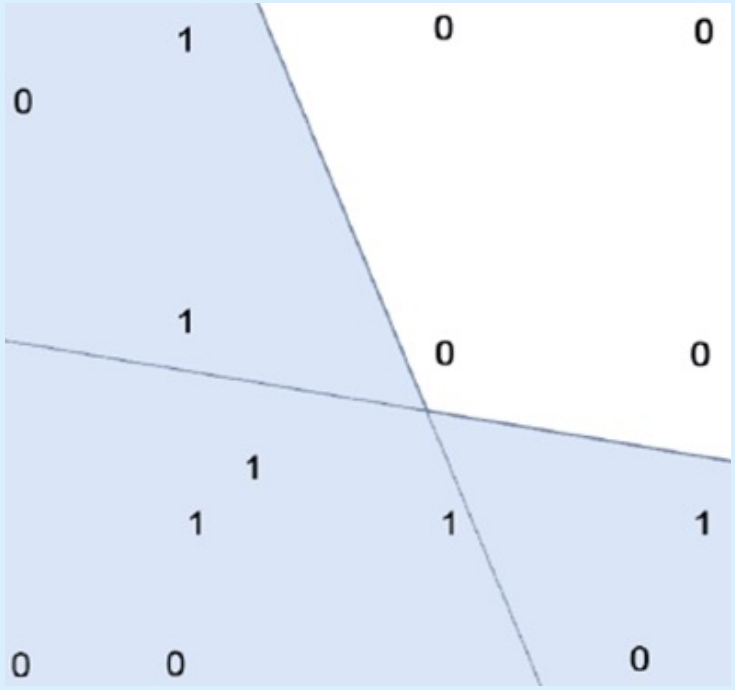


Fehler bei einer ungeeigneten Trainingsmenge



Zu viele Trainingsbeispiele:

das Netz hat „auswendig gelernt“

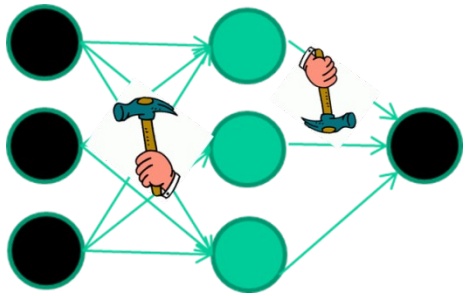


Zu wenig Trainingsbeispiele:

keine richtige Klassifikation

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 verdammtes Glück kommt oft eine Funktion heraus, die gut genug in Anwendungen funktioniert



Differenzierbare Programmierung

APPLICATIONS

Word-Word Associations in Document Retrieval

Recap bag-of-words approaches

- Client profiles, TF-IDF

Words are not independent of each other

Need to represent some aspects of word semantics

Church, K.W., Hanks, P.: Word association norms mutual information, and lexicography. Comput. Linguist. 1(1), 22–29, **1990**

Point(wise) Mutual Information: PMI

- Measure of association used in information theory and statistics

$$\operatorname{pmi}(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)}$$

- Positive PMI: PPMI(x, y) = max(pmi(x, y), 0)
- Quantifies the discrepancy between the probability of their coincidence given their joint distribution and their individual distributions, assuming independence
- Finding collocations and associations between words
- Countings of occurrences and co-occurrences of words in a text corpus can be used to approximate the probabilities p(x) or p(y) and p(x,y) respectively

[Wikipedia]

PMI – Example

word 1	*word 2*	*count word 1*	*count word 2*	count of co-occurrences	PMI
puerto	rico	1938	1311	1159	10.0349081703
hong	kong	2438	2694	2205	9.72831972408
los	angeles	3501	2808	2791	9.56067615065
carbon	dioxide	4265	1353	1032	9.09852946116
prize	laureate	5131	1676	1210	8.85870710982
san	francisco	5237	2477	1779	8.83305176711
nobel	prize	4098	5131	2498	8.68948811416
ice	hockey	5607	3002	1933	8.6555759741
star	trek	8264	1594	1489	8.63974676575
car	driver	5578	2749	1384	8.41470768304
it	the	283891	3293296	3347	-1.72037278119
are	of	234458	1761436	1019	-2.09254205335
this	the	199882	3293296	1211	-2.38612756961
is	of	565679	1761436	1562	-2.54614706831
and	of	1375396	1761436	2949	-2.79911817902
a	and	984442	1375396	1457	-2.92239510038
in	and	1187652	1375396	1537	-3.05660070757
to	and	1025659	1375396	1286	-3.08825363041
to	in	1025659	1187652	1066	-3.12911348956
of	and	1761436	1375396	1190	-3.70663100173

- Counts of pairs of words getting the most and the least PMI scores in the first 50 millions of words in Wikipedia (dump of October 2015)
- Filtering by 1,000 or more co-occurrences.
- The frequency of each count can be obtained by dividing its value by 50,000,952. (Note: natural log is used to calculate the PMI values in this example, instead of log base 2)

[Wikipedia]

PMI – Co-occurrence Matrix

Add-2 Smoothed Count(w,context)

	computer	data	pinch	result	sugar
apricot	2	2	3	2	3
pineapple	2	2	3	2	3
digital	4	3	2	3	2
information	3	8	2	6	2

PPMI(w,context)

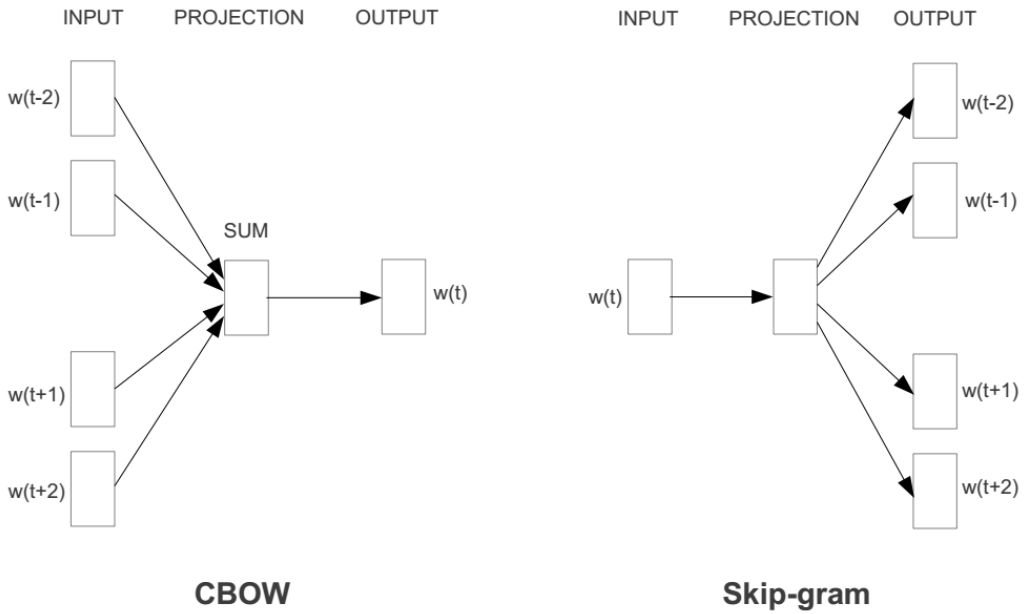
	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

Embedding Approaches to Word Semantics

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word

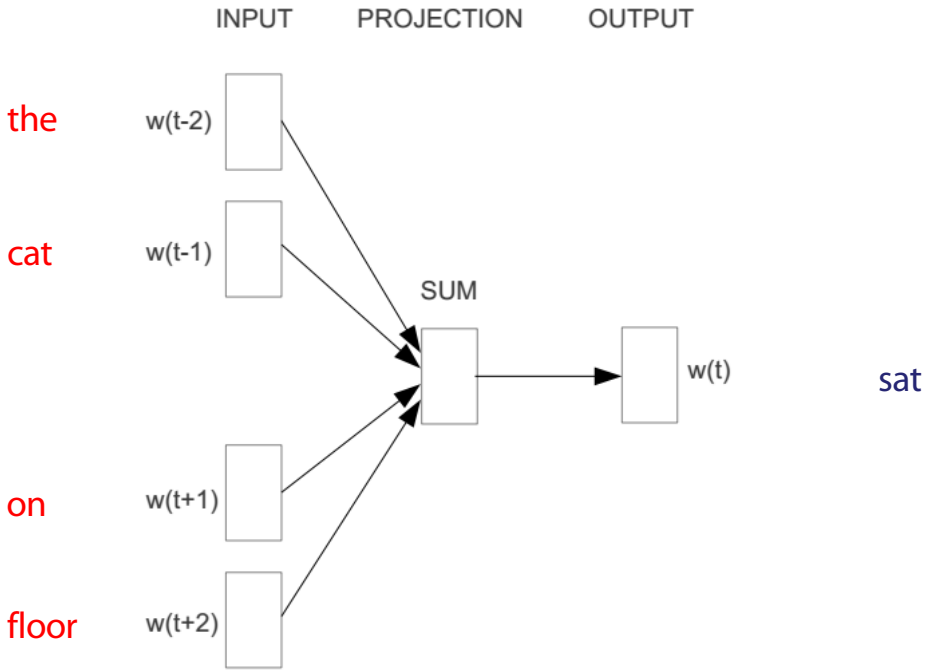
Represent the meaning of **words** – word2vec

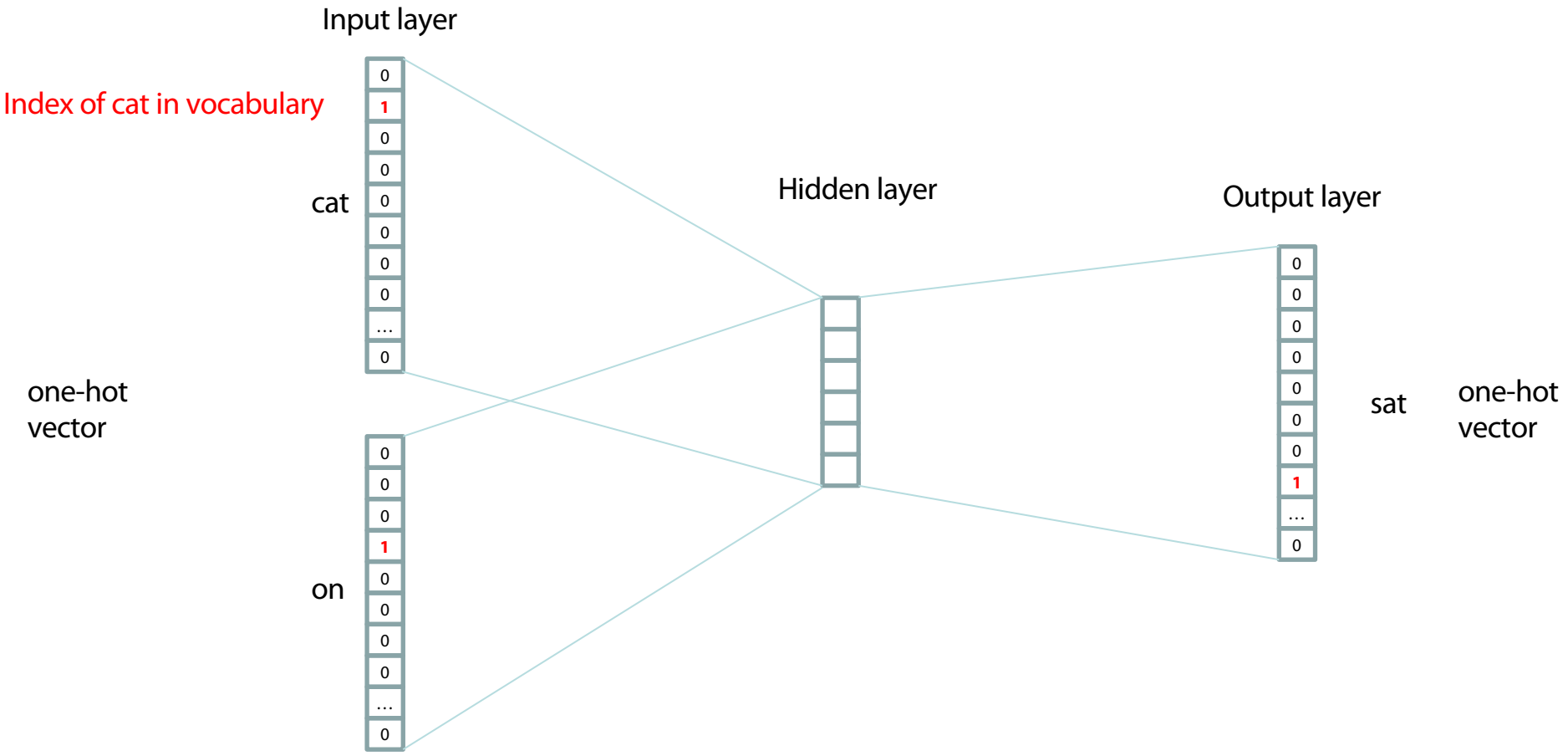
- 2 basic structural models:
  - Continuous Bag of Words (CBOW): use a window of words to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in window.

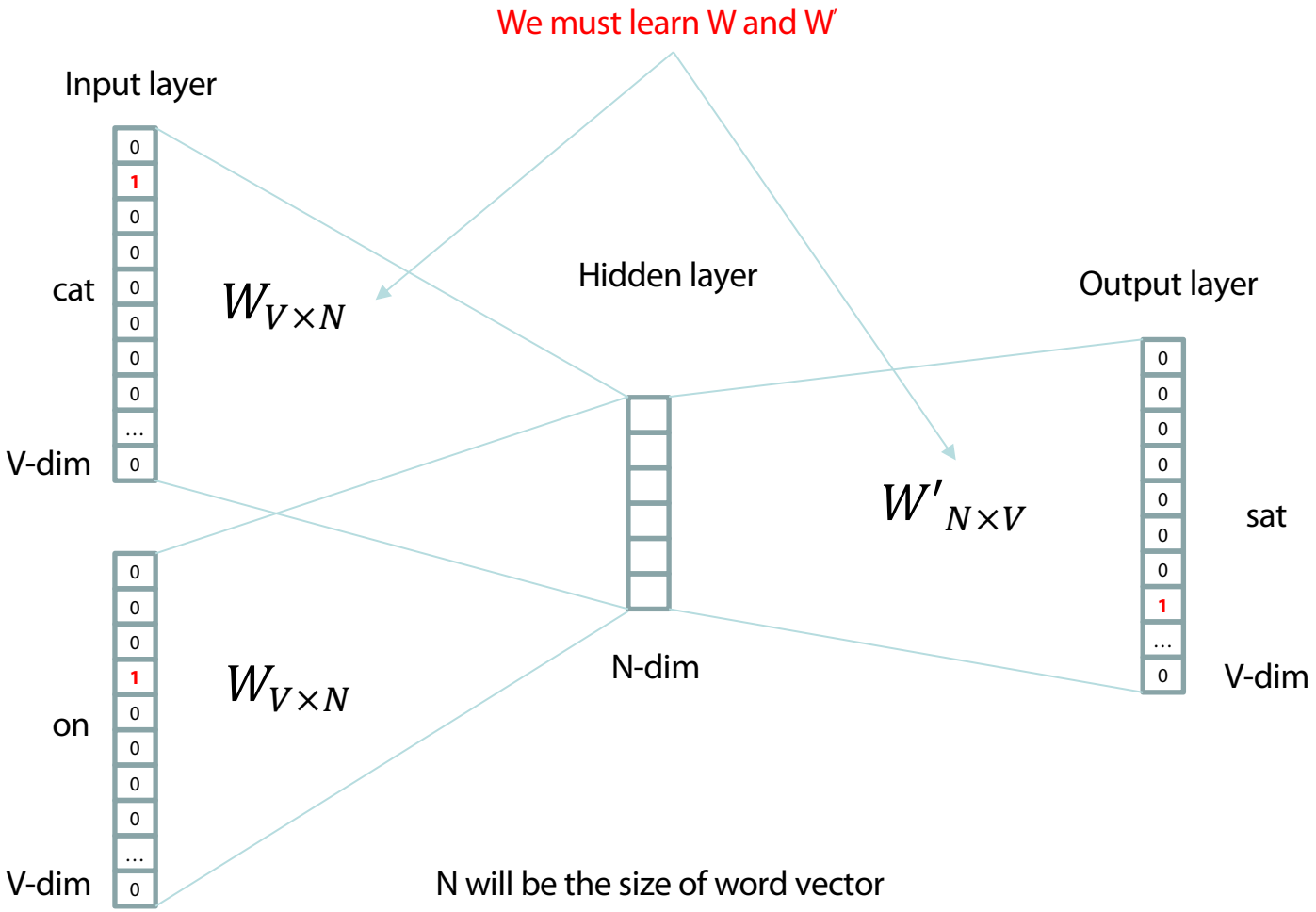


Word2vec – Continuous Bag of Word

- E.g. “The cat <sat> on floor”
  - Window size = 2

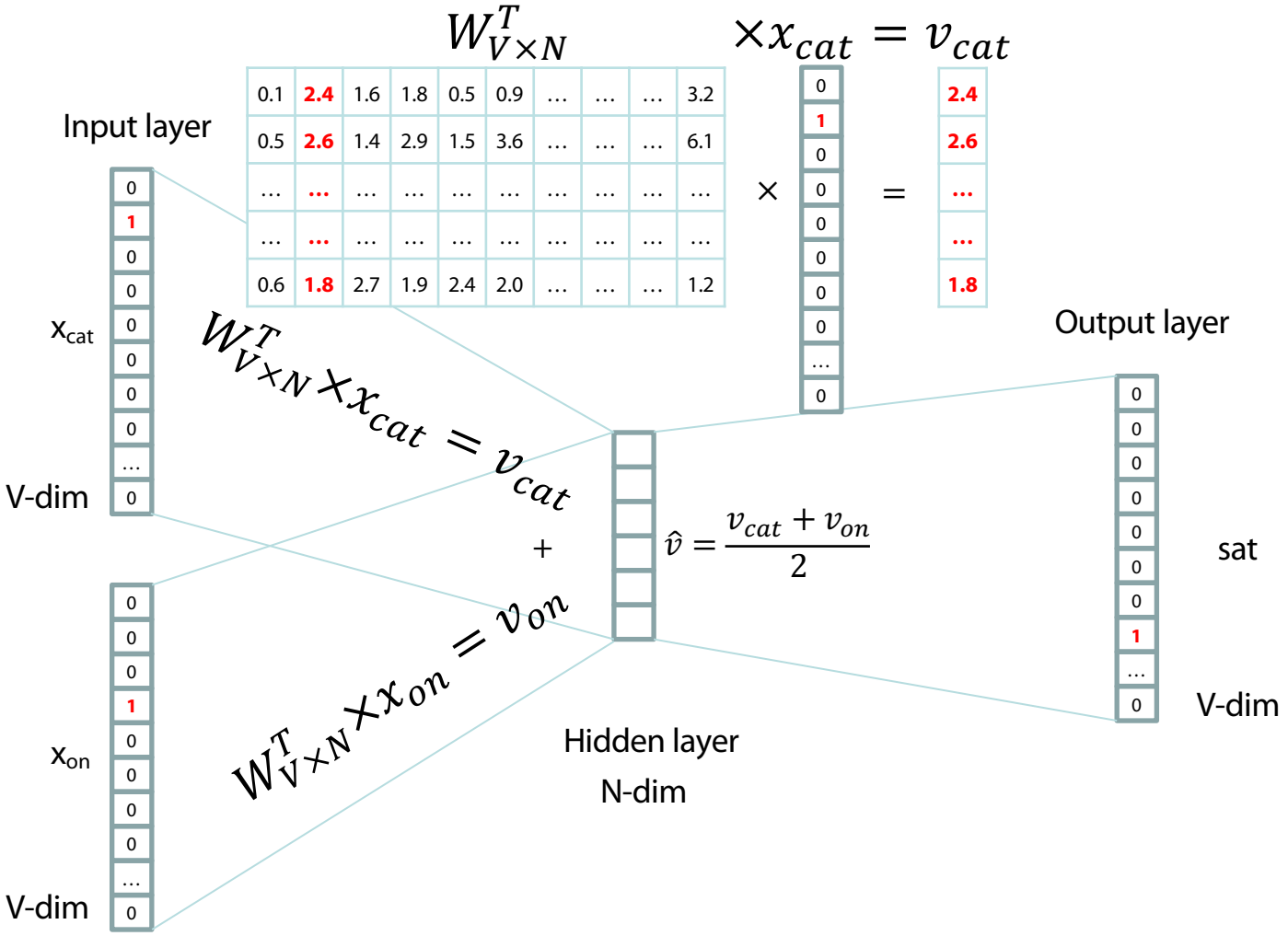


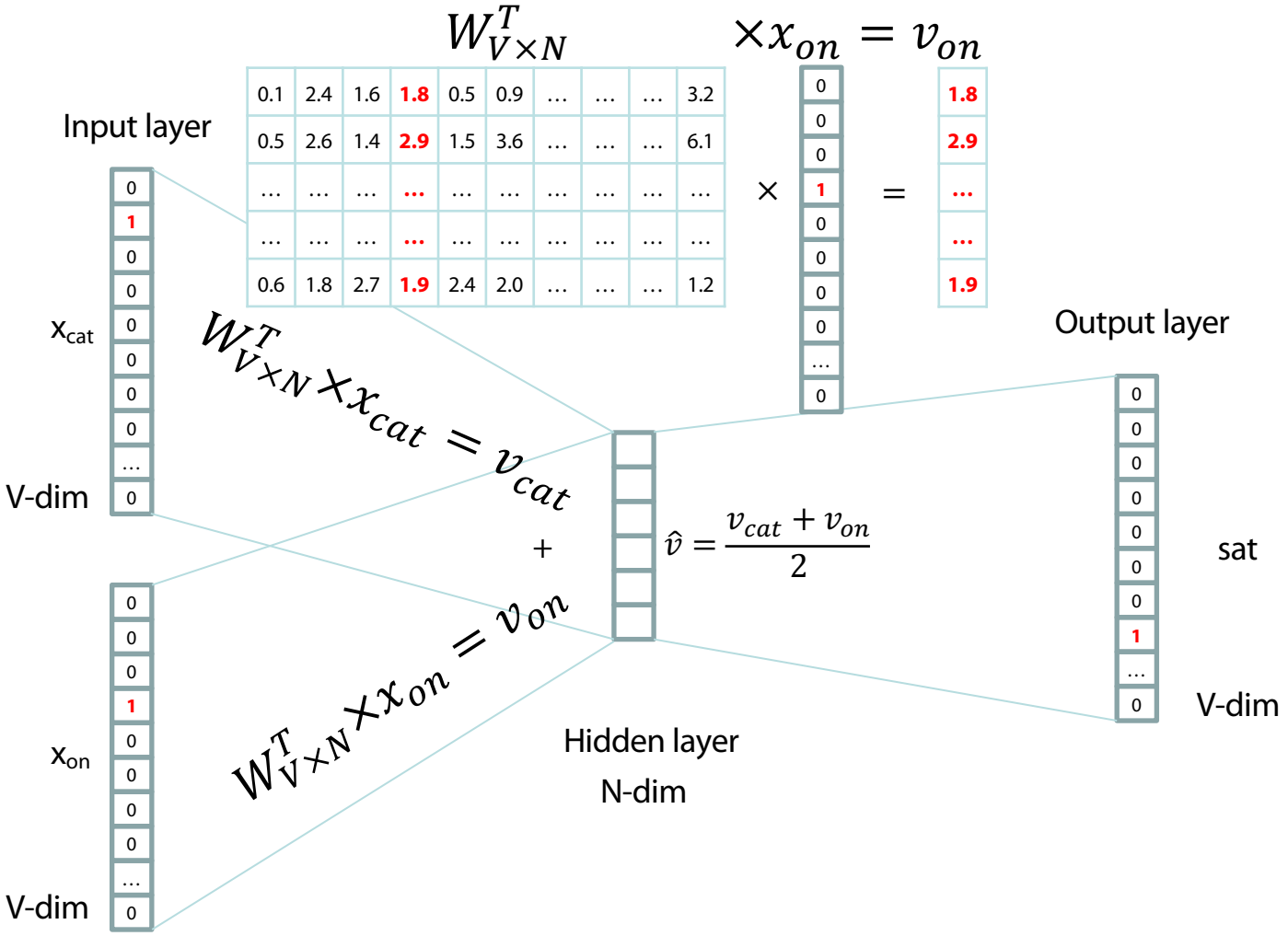

Input layer
Index of cat in vocabulary
cat
one-hot vector
on
Hidden layer
Output layer
sat
one-hot vector

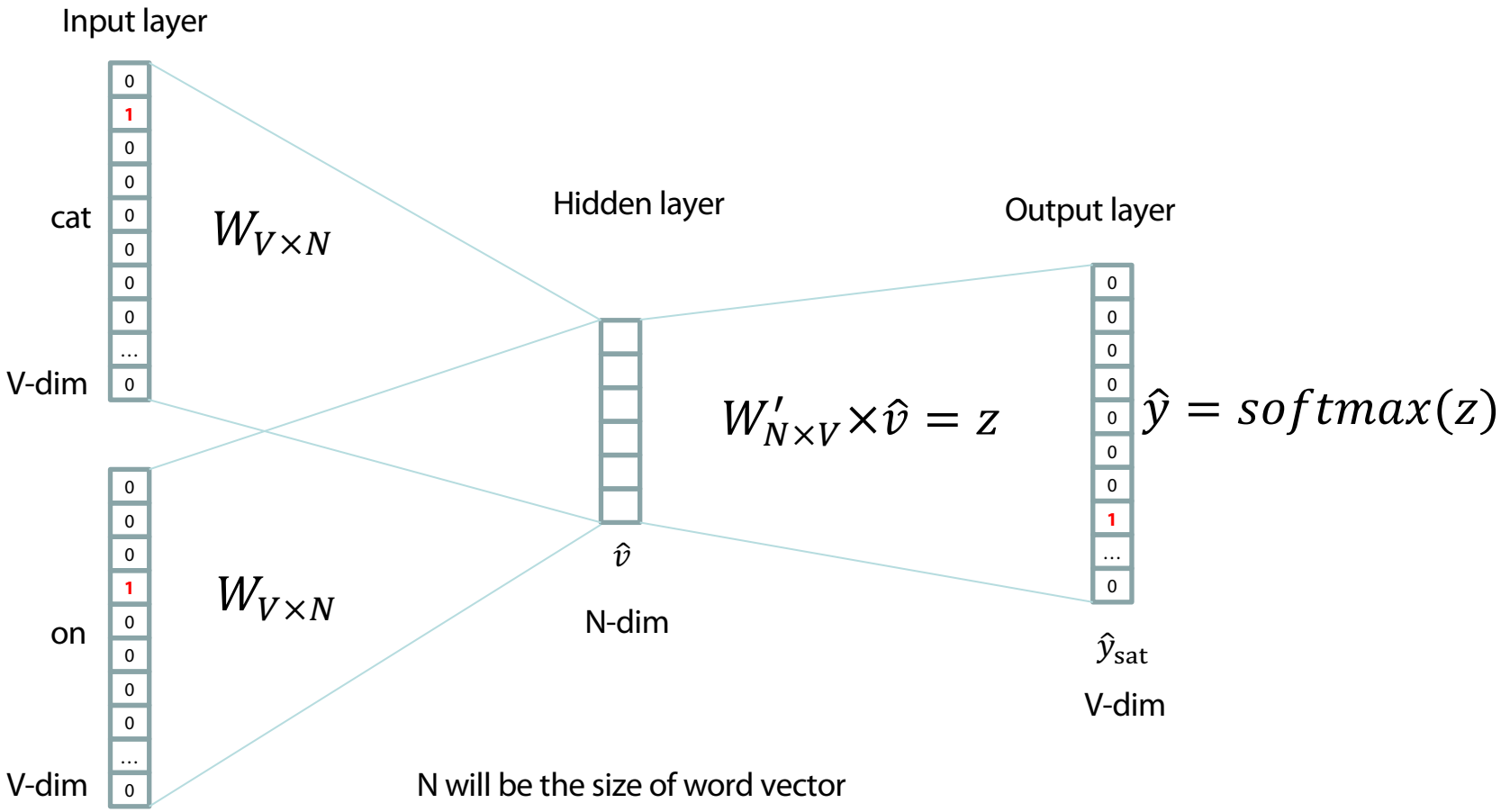

We must learn W and W'
Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
N-dim
$W'_{N\times V}$
Output layer
sat
V-dim
N will be the size of word vector

Deep Learning

- Hidden layer represents feature space
  - Making explicit features in the data…
  - … that are relevant for a certain task
- Determine features automatically
  - Learning suitable mappings into feature space
- Deep learning also known as representation learning


$W^T_{V\times N}$
$\times x_{cat} = v_{cat}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
… … … … … … … … … …
… … … … … … … … … …
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
×
=
2.4
2.6
…
…
1.8
Output layer
x_{cat}
V-dim
$W^T_{V\times N} x_{cat} = v_{cat}$
+
$W^T_{V\times N} x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
sat
x_{on}
V-dim
Hidden layer
N-dim


$W^T_{V\times N}$
$\times x_{on} = v_{on}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
… … … … … … … … … …
… … … … … … … … … …
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
× = 1.8 2.9 … … 1.9
Output layer
x_{cat}
$W^T_{V\times N} x_{cat} = v_{cat}$
+
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
sat
V-dim
x_{on}
$W^T_{V\times N} x_{on} = v_{on}$
Hidden layer
N-dim
V-dim


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{N\times V}\times\hat{v}=z$
Output layer
$\hat{y}=softmax(z)$
$\hat{y}_{\text{sat}}$
V-dim
N will be the size of word vector

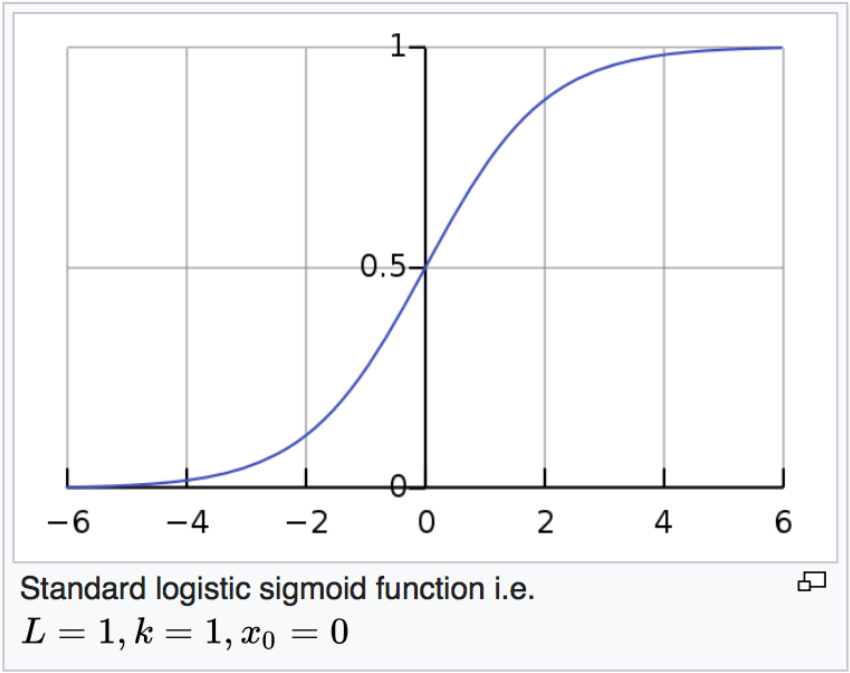
Logistic function

A **logistic function** or **logistic curve** is a common "S" shape (sigmoid curve), with equation:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

where

- e = the natural logarithm base (also known as Euler's number),
- x_0 = the x-value of the sigmoid's midpoint,
- L = the curve's maximum value, and
- k = the steepness of the curve.[1]



Standard logistic sigmoid function i.e. $L = 1, k = 1, x_0 = 0$

[Wikipedia]

softmax(**z**)

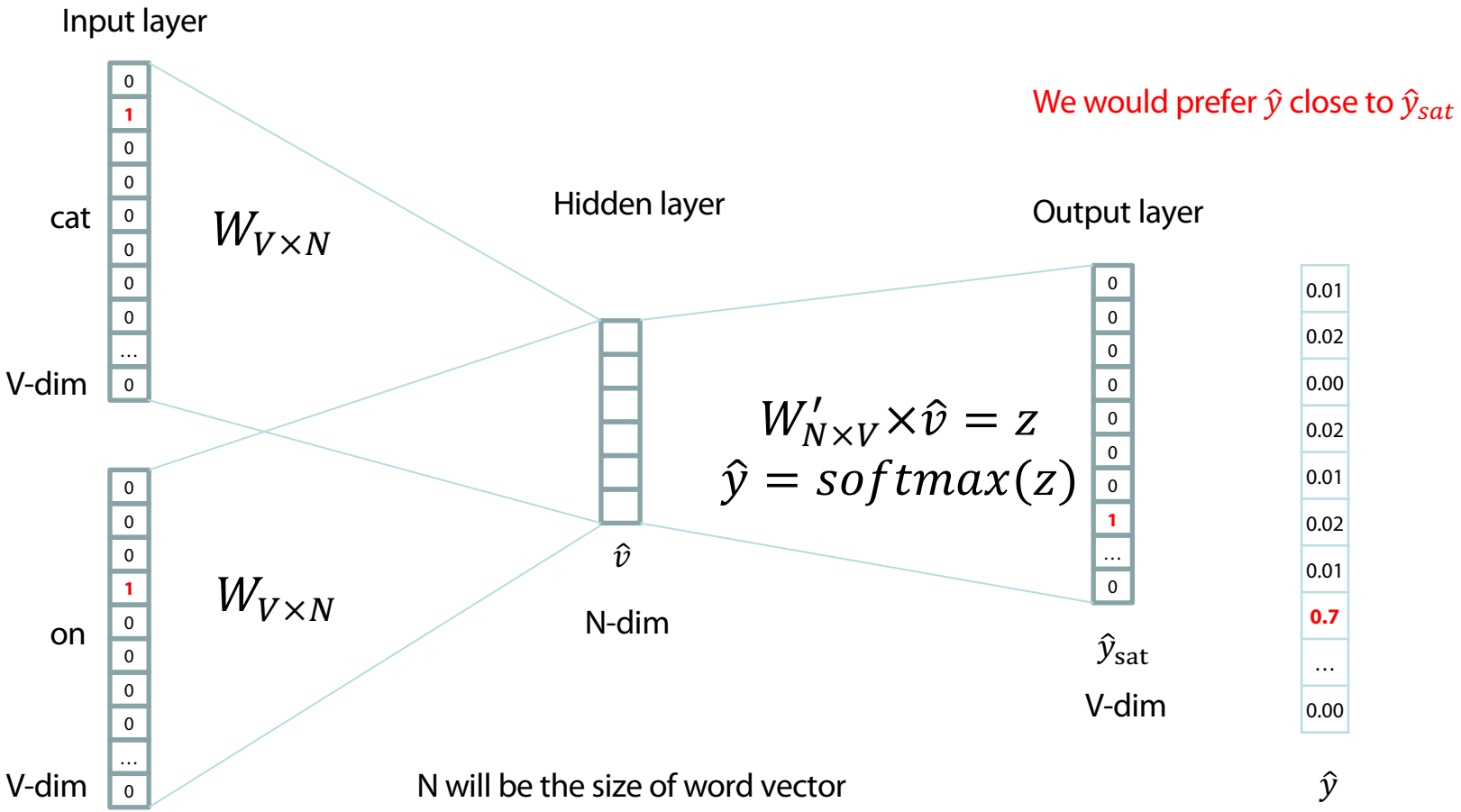
The **softmax function**, or **normalized exponential function**, is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range [0, 1] that add up to 1. The function is given by

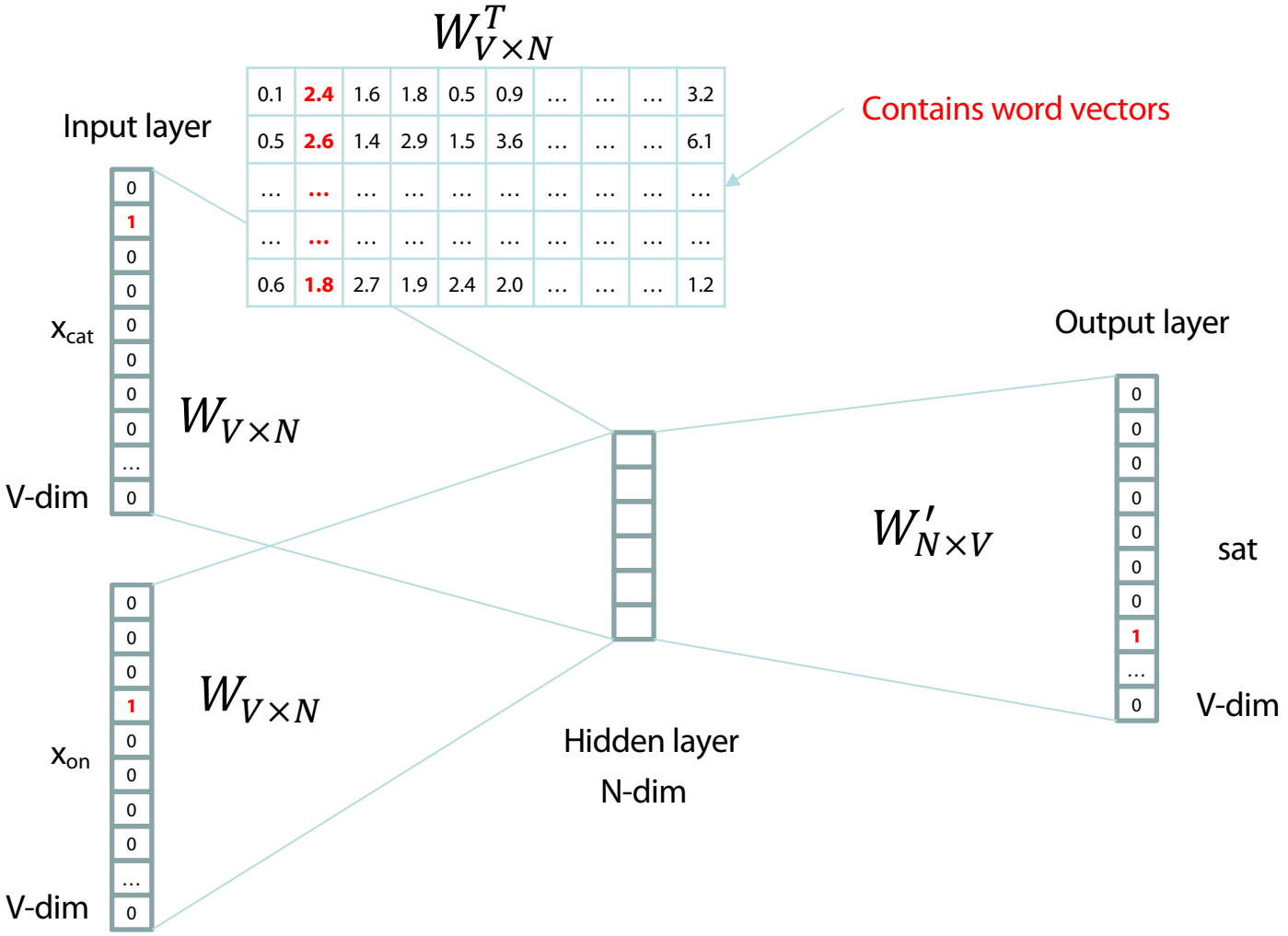
$$\sigma : \mathbb{R}^K \to [0,1]^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

In probability theory, the output of the softmax function can be used to represent a categorical distribution – that is, a probability distribution over K different possible outcomes.

[Wikipedia]


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{N\times V}\times\hat{v}=z$
$\hat{y}=softmax(z)$
Output layer
$\hat{y}_{sat}$
V-dim
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
$\hat{y}$
N will be the size of word vector



Consider either W or W' as the word's representation.

Word Analogies

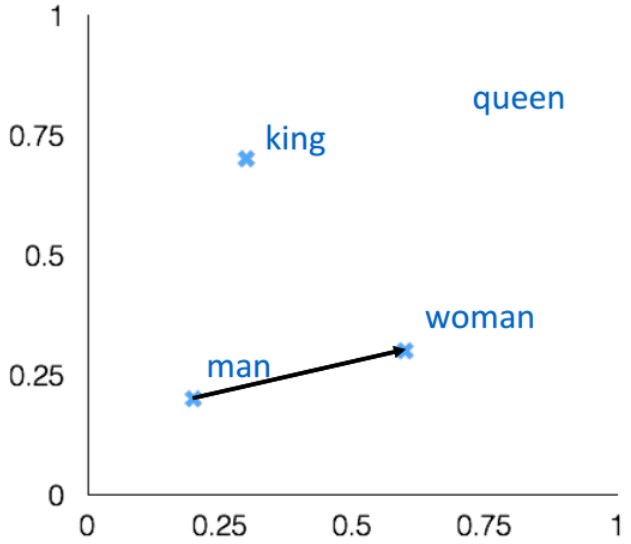
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:?

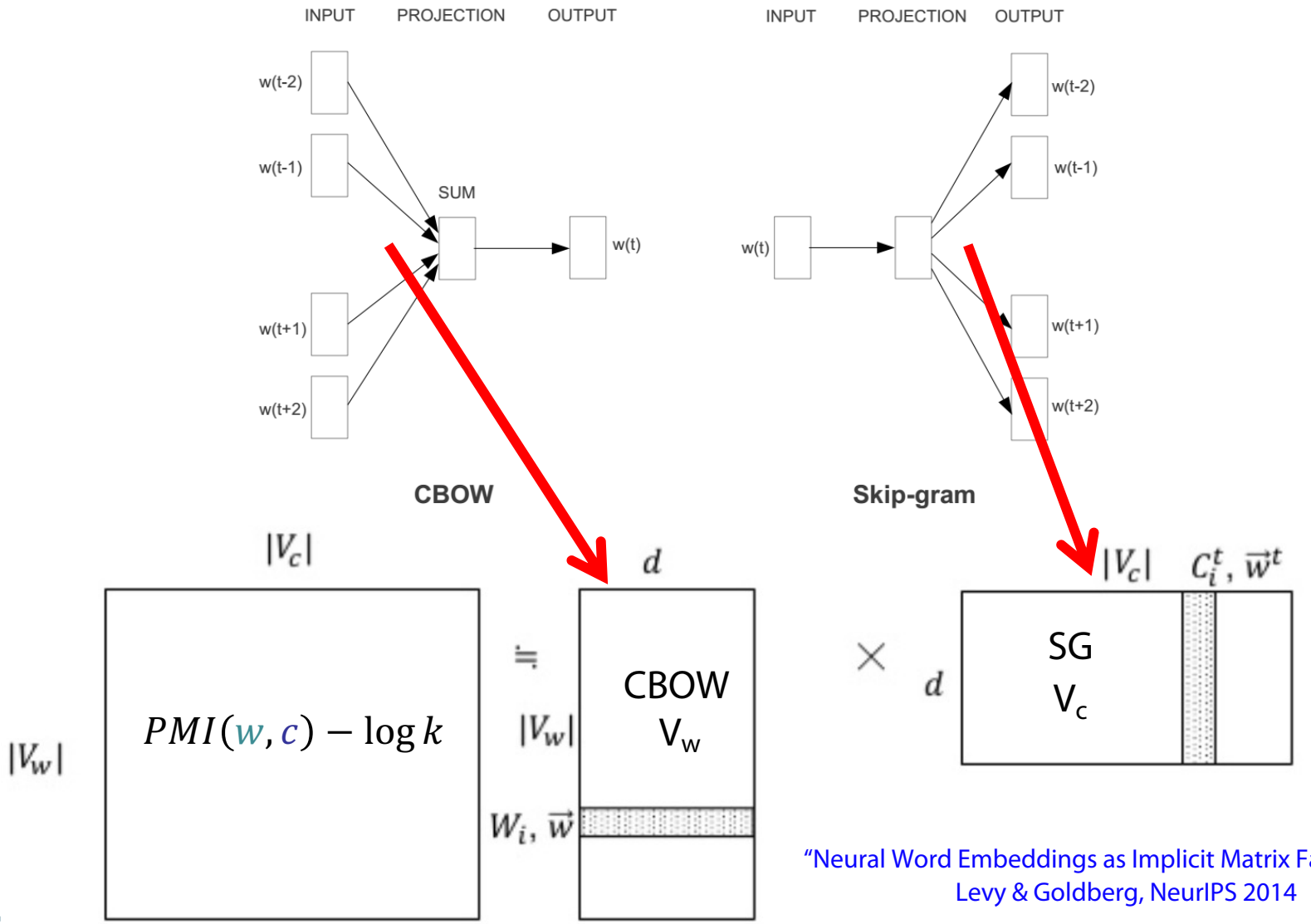
$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c|| ||w_x||}$$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



The Picture: CBOW and Skip-Gram (SG)



"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NeurIPS 2014

Convolution

Input image

Convolution Kernel

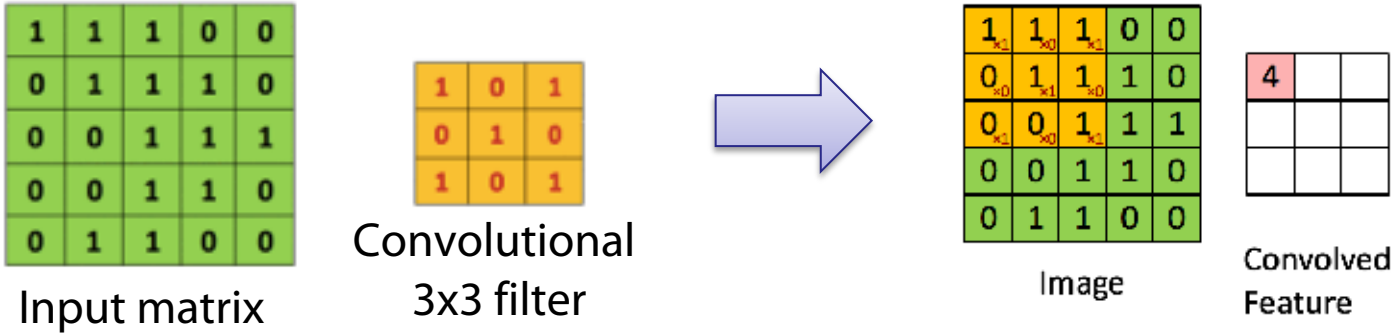
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and group them afterwards

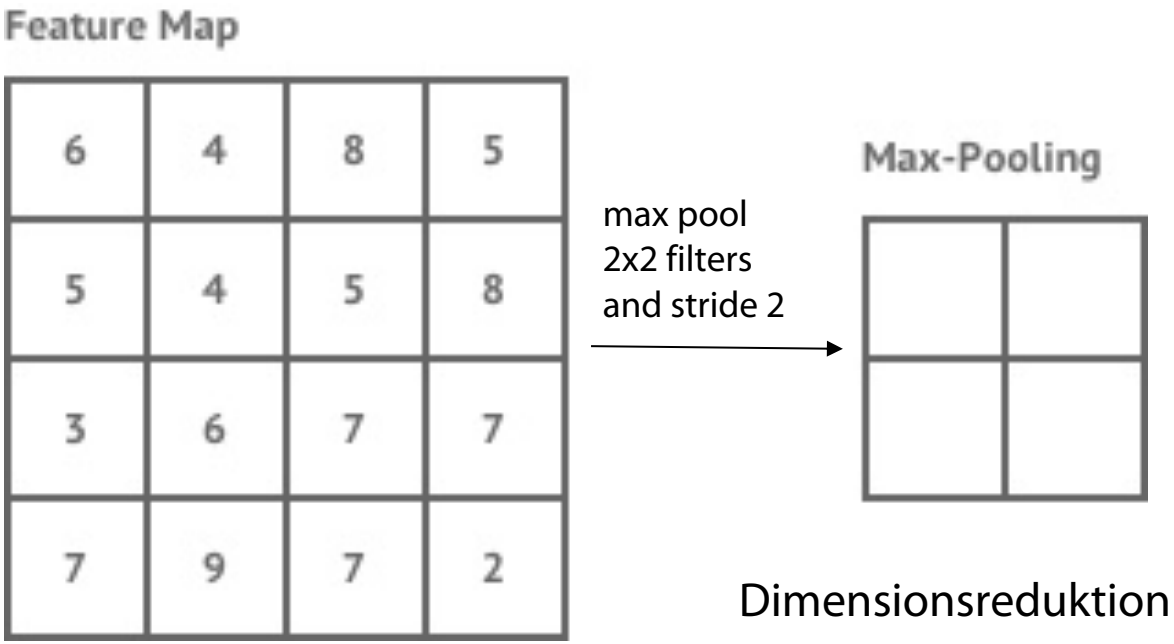
Example: “this takes too long” compute vectors for:
This takes, takes too, too long, this takes too, takes too long, this takes too long



http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

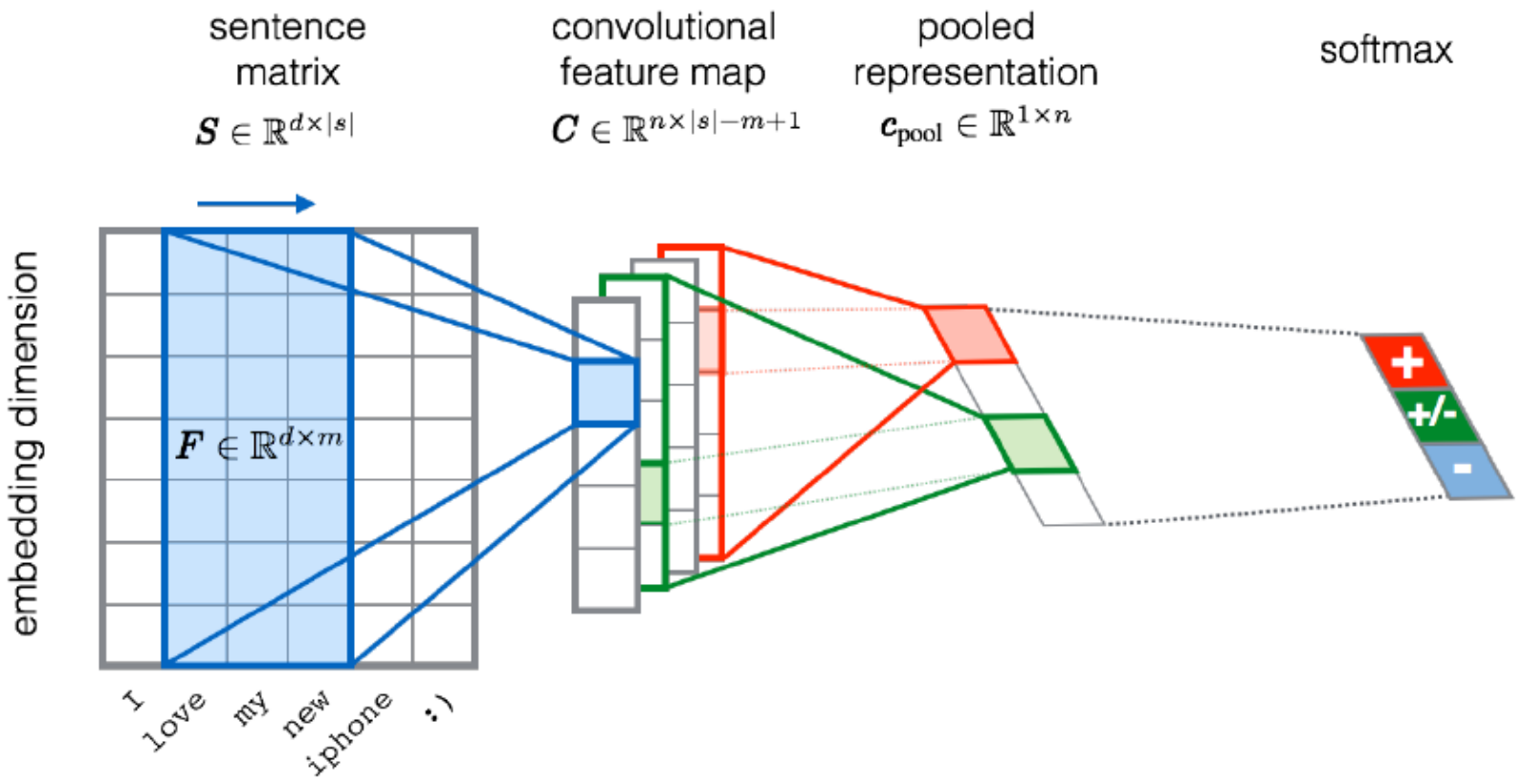
Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and **group them afterwards**



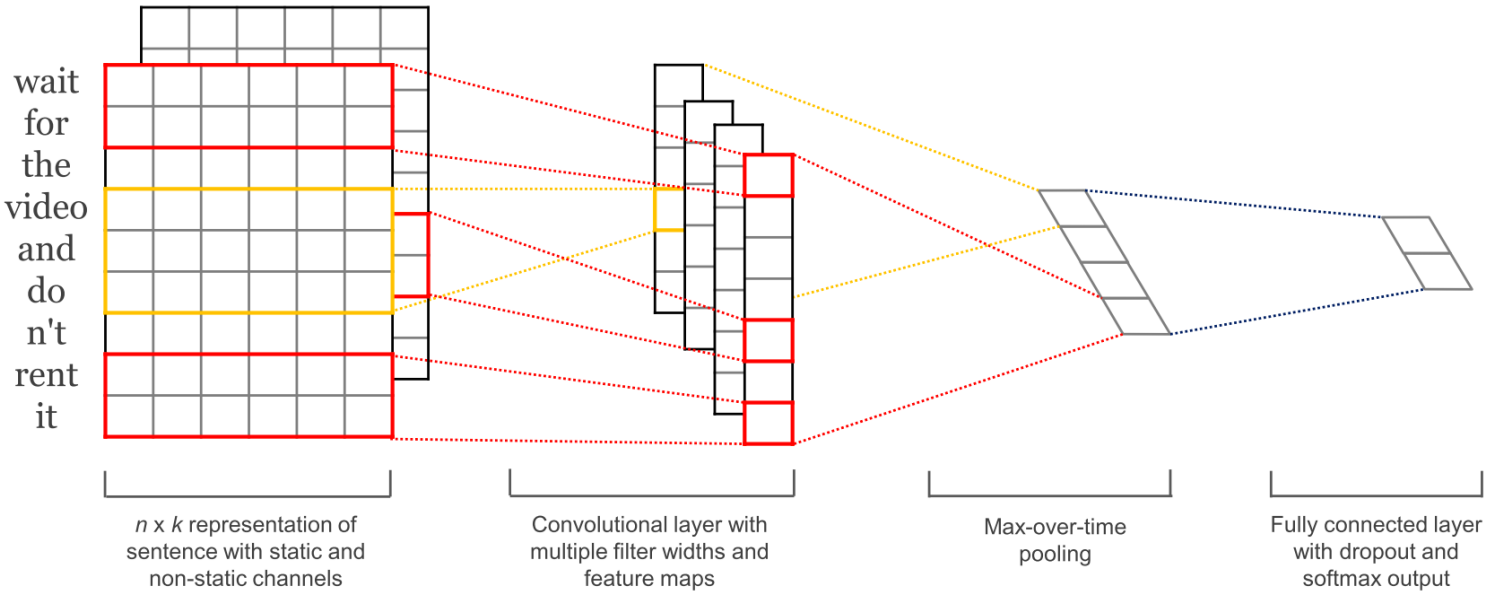
https://shafeentejani.github.io/assets/images/pooling.gif

CNN for text classification



Severyn, Aliaksei, and Alessandro Moschitti. "UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification." *SemEval@ NAACL-HLT*. 2015.

CNN with multiple filters

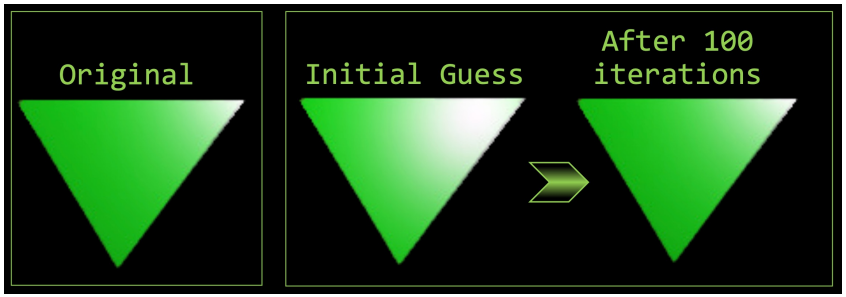


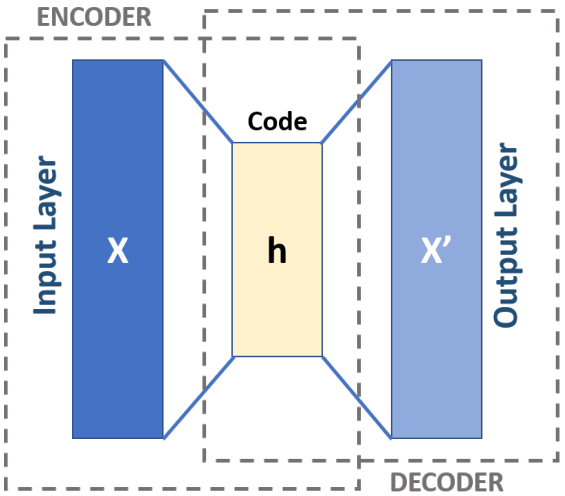
Kim, Y. "Convolutional Neural Networks for Sentence Classification", EMNLP (2014)

sliding over 3, 4 or 5 words at a time

Differential Programming: Basic Idea

- Example: Computer Vision
  - Estimate position of light source
  - Use standard learning approach





- Develop render in appropriate programming language (e.g., Julia)
- Differentiate renderer (e.g., Zygote)
  - Use differentiated render for backpropagation

Geschichtlicher Überblick

Anfänge

Ernüchterung

Renaissance

Boom

- ***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

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

- ***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.
- ***Ab 2000***: Deep Learning (Hinton, LeCun, Bengio, Ng, et al.)
- ***Ab 2020***: Differentiable Programming (Lecun et al.)