Grundlagen der Künstlichen Intelligenz

15 Lernen in Künstlich Neuronalen Netzen

Grundlagen, Netzwerkstrukturen, Perzeptron, Backpropagation

Volker Steinhage

Inhalt

- Motivation
- Grundlegende Elemente neuronaler Netze
- Netzwerkstrukturen
- Lernen in neuronalen Netzen
- Perzeptron
- Backpropagation
- Beispielanwendung
- Zusammenfassung & Ausblick

Motivation (1)

Vom mathematischen Standpunkt:

Künstlich Neuronale Netze (*KNNs*) als *Methode*, Funktionen zu repräsentieren

- durch Netzwerke von einfachen Berechnungselementen (vergleichbar mit logischen Schaltkreisen),
- die aus Beispielen gelernt werden können;

⇝ in dieser Vorlesung!

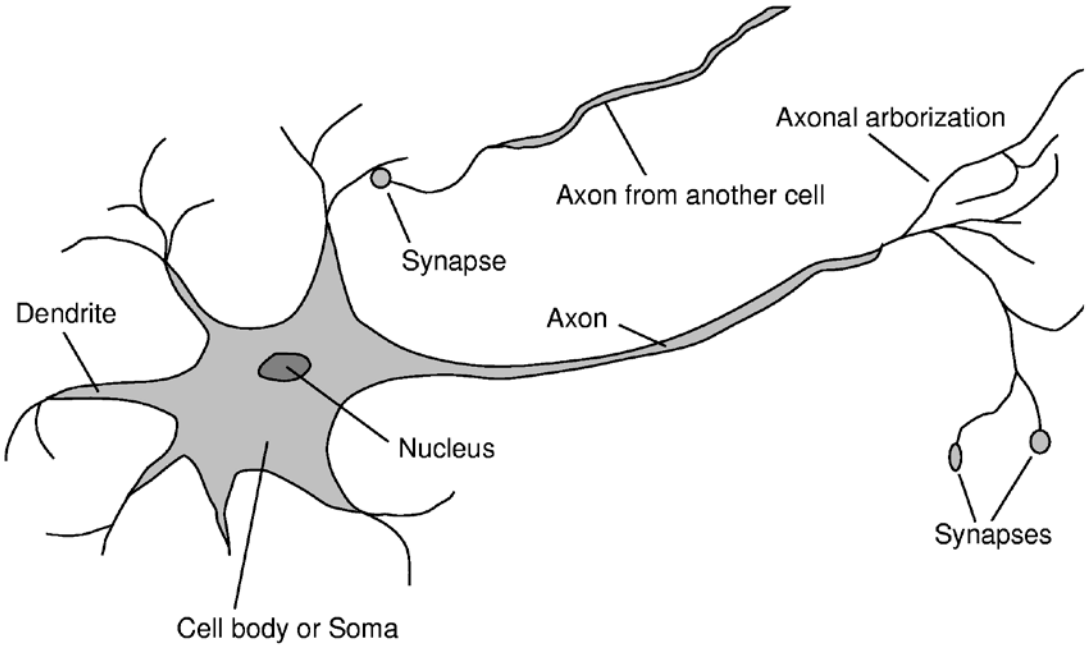
Vom biologischen Standpunkt:

Künstlich Neuronale Netze als *Modell* des Gehirns und seiner Funktionsweise;

⇝ nicht in dieser Vorlesung!

Motivation (2)

Ausgangspunkt: Abstrahierende Nachbildung von Struktur und Verarbeitungsmechanismen des Gehirns:



Ziel: *Viele Prozessoren (Neuronen) und Verbindungen (Synapsen), die parallel und verteilt Informationen verarbeiten.*

Forschung auf dem Gebiet der KNNs (1)

1943: McCulloch und Pitts führen die Idee eines binären Neurons ein.

1943-1969: Forschung an neuronalen Netzen meist in Form von einschichtigen Netzen, den sog. *Perzeptrons*

1969: Minsky und Papert zeigen, dass *Perzeptrons* sehr beschränkt sind.

1969-1980: Kaum noch Arbeiten auf dem Gebiet.

Seit 1981: Erste Renaissance neuronaler Netze.

- In der KI als *Werkzeug* benutzt zum Approximieren von Funktionen.
- Insbesondere für sensomotorische Aufgaben gut geeignet.
- In der Biologie und Physiologie als Modell.

Forschung auf dem Gebiet der KNNs (2)

Seit ca. 2009: Zweite Renaissance neuronaler Netze.

- KNNs liefern bei herausfordernden Anwendungen wie internationalen Mustererkennungswettbewerben oft bessere Ergebnisse als konkurrierende Lernverfahren.

- Zwischen 2009 und 2012 gewannen rekurrente bzw. tiefen vorwärts-gerichteten neuronalen Netzwerke aus dem Schweizer KI Labor IDSIA gleich acht internat. Wettbewerbe in den Bereichen Mustererkennung und masch. Lernen.(1)

- Tiefe Netze vorwärtsgerichtete KNNs auf der Basis effizienter GPU-Implementierungen und in Verbindung mit „Big Data“ erzielen die bisher besten Ergebnisse auf dem *ImageNet* Benchmark. (2),(3)

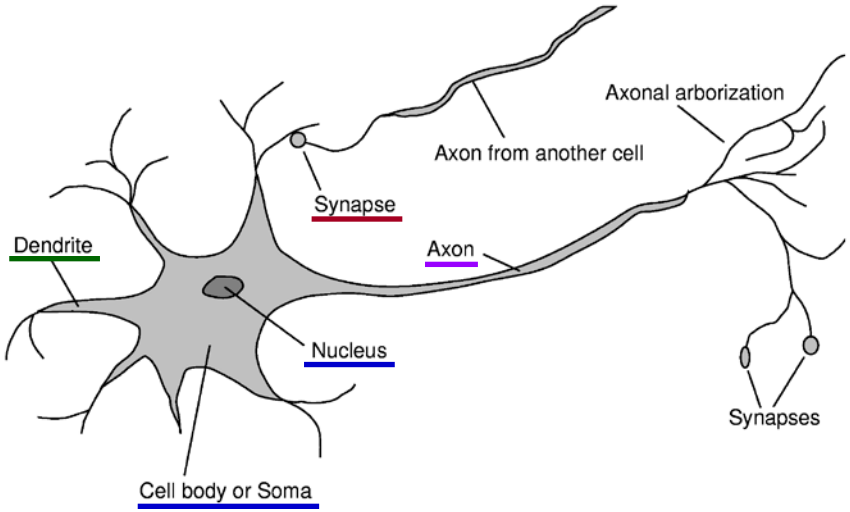
---

(1) http://www.kurzweilai.net/how-bio-inspired-deep-learning-keeps-winning-competitions 2012 Kurzweil AI Interview mit Jürgen Schmidhuber zu den acht Wettbewerben, die sein Deep Learning Team zwischen 2009 und 2012 gewann.

(2) A. Krizhevsky, I. Sutskever, G. E. Hinton: *ImageNet Classification with Deep Convolutional Neural Networks.* NIPS 25, MIT Press, 2012. http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf

(3) M. D. Zeiler, R. Fergus: *Visualizing and Understanding Convolutional Networks.* TR arXiv:1311.2901 [cs.CV], 2013. http://arxiv-web3.library.cornell.edu/abs/1311.2901

Grundbegriffe natürlicher neuronaler Netze (1) *



Neuron: Nervenzelle, bestehend aus Zellkörper (Soma) und Zellkern (Nucleus).

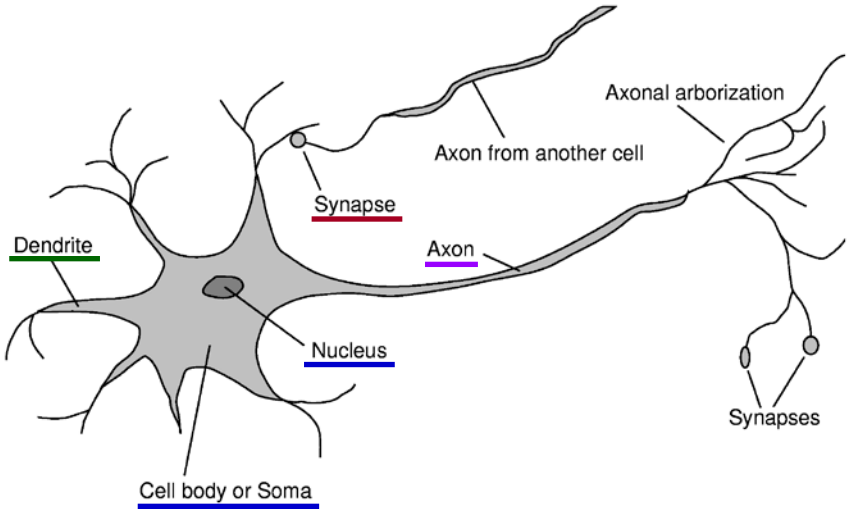
Dendriten: Stachelartige Fortsätze eines Neurons, auf denen mehrere Synapsen enden.

Synapse: Verbindung zwischen Dendrit eines Neurons und Axon eines anderen Neurons.

Axon: Verbindungsfaser eines Neurons, die am Ende oft verzweigt in Vielzahl synaptischer Endungen an anderen Neuronen.

* Diese Darstellung dient nicht einer hinreichenden Erklärung neurophysiologischer Zusammenhänge, sondern der Erläuterung einiger der wichtigsten Begrifflichkeiten und Zusammenhänge aus der Neurophysiologie, auf deren Grundlage Begriffe und Funktionalitäten von künstliche Neuronalen Netze (KNN) abgeleitet wurden.

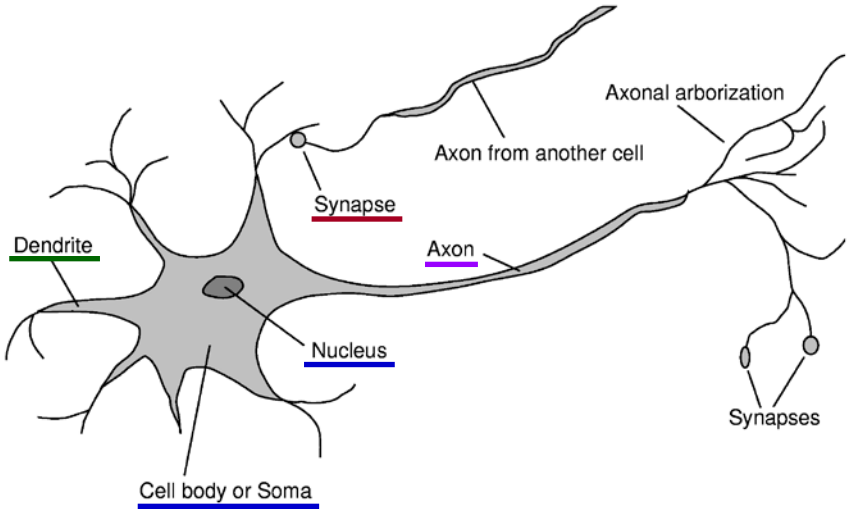
Grundbegriffe natürlicher neuronaler Netze (2)



Signalübertragung 1: Chem. Neurotransmittersubstanzen (Natrium- und Kaliumionen) werden von Synapsen auf Dendriten übertragen und verändern das elekrostatische Potential des Neurons.

Signalübertragung 2: Ab bestimmten sog. Aktionspotential des Neurons wird dieses Potential als Nervensignal entlang des Axons fortgepflanzt bis zu den Synapsen anderer Neuronen.

Grundbegriffe natürlicher neuronaler Netze (3)



Erregende Synapsen (*Excitatory Synapses*): Erhöhen das Potential des Neurons und animieren damit das Aussenden von Nervensignalen.

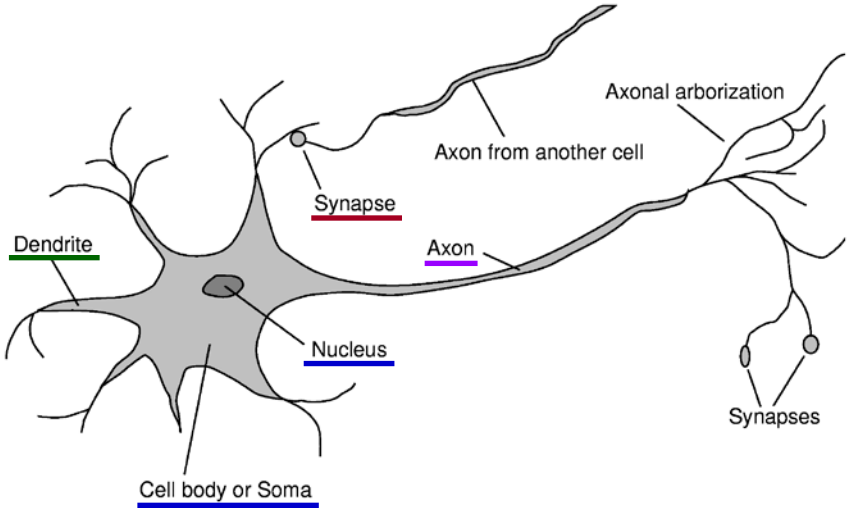
Hemmende Synapsen (*Inhibitory Synapses*): Verringern das Potential des Neurons und behindern damit das Aussenden von Nervensignalen.

Plastizität von Synapsen: Die Stärke von Erregung bzw. Hemmung hängt von der Transmitterausschüttung der Synapse ab. Diese ändert sich über die Zeit aufgrund chemischer Veränderungen in der Synapse, die wiederum von der Geschichte der empfangenen Nervensignale abhängen ⇝ Speichern und Lernen von Information!

Grundbegriffe künstlicher neuronaler Netze

Einheiten (*Units*): Stellen die Knoten (Neuronen) im Netz dar.

Verbindungen (*Links*): Kanten zwischen den Knoten des Netzes. Jeder Knoten hat Ein- und Ausgabekanten.



Gewichte (*Weights*): Jede Kante hat eine Gewichtung, in der Regel eine reelle Zahl.

Ein-/Ausgabeknoten (*Input and Output Units*): Besonders gekennzeichnete Knoten, die mit der Außenwelt verbunden sind.

Aktivierungsniveau (*Activation Level*): Der von einem Knoten aus seinen Eingabekanten zu jedem Zeitpunkt berechnete Wert. Dieser Wert wird über Ausgabekanten an die Nachbarknoten weitergeleitet.

Funktionsweise einer Einheit

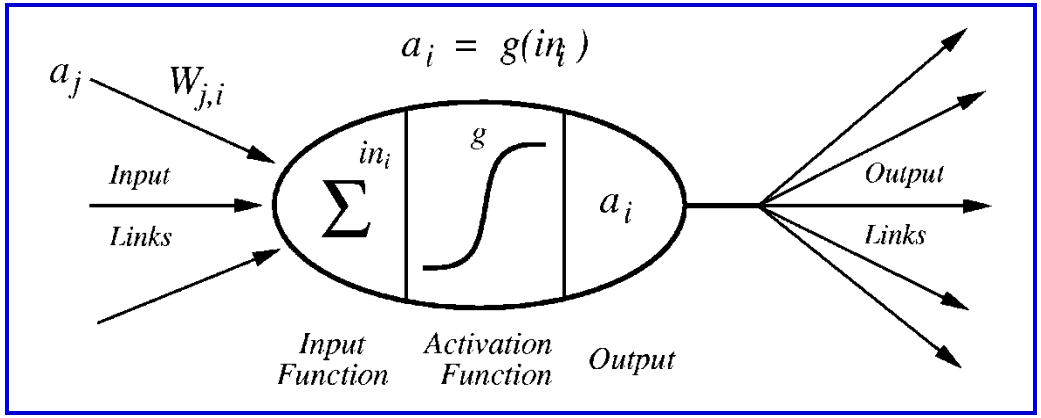
Eingabefunktion in_i berechnet die Stärke der Eingabe für Einheit *i* als *Linear-kombination* von Eingabeaktivierung a_j und Gewichten $w_{j,i}$ über alle Knoten *j*, die direkt mit *i* verbunden sind:

$$in_i = \sum_{j=0,\ldots,n} w_{j,i} \cdot a_j = \boldsymbol{w}_i \cdot \boldsymbol{a}_j$$

mit Vektoren $\boldsymbol{a}_i$ und $\boldsymbol{w}_i$ der eingehenden Aktivierungswerte bzw. entspr. Gewichte.

Eine i.A. *nicht-lineare* Aktivierungsfunktion *g* berechnet das Aktivierungsniveau a_i:

$$a_i = g(in_i) = g\left(\sum_{j=0,\ldots,n} w_{j,i} \cdot a_j\right).$$

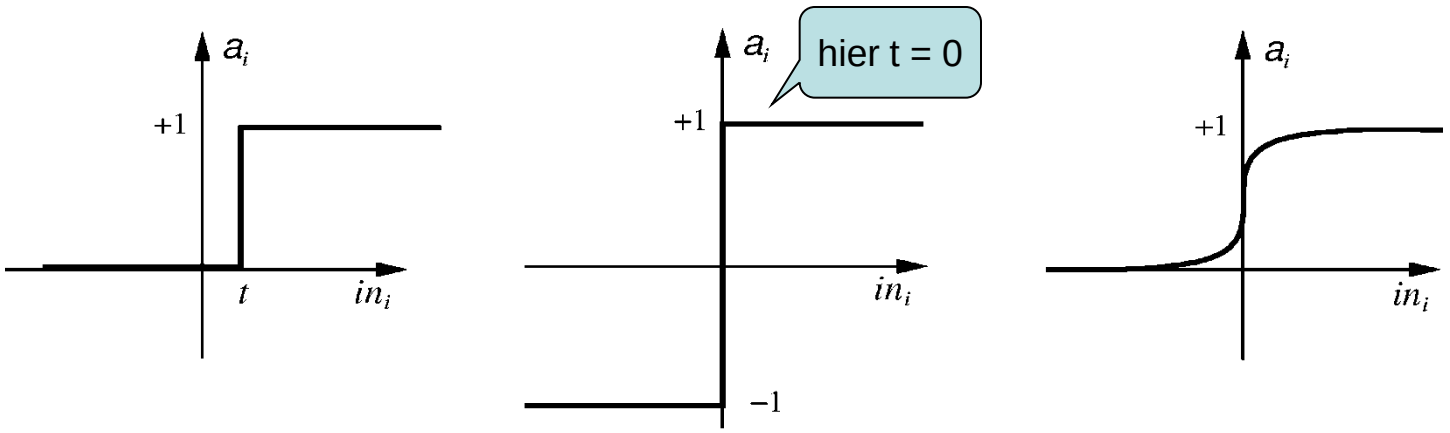


Aktivierungsfunktion

$$step_t(x) = \begin{cases} 1 & \text{if } x \geq t \\ 0 & \text{if } x < t \end{cases}$$

$$sign(x) = \begin{cases} +1 & \text{if } x \geq t \\ -1 & \text{if } x < t \end{cases}$$

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



(a) Step function **(b) Sign function** **(c) Sigmoid function**

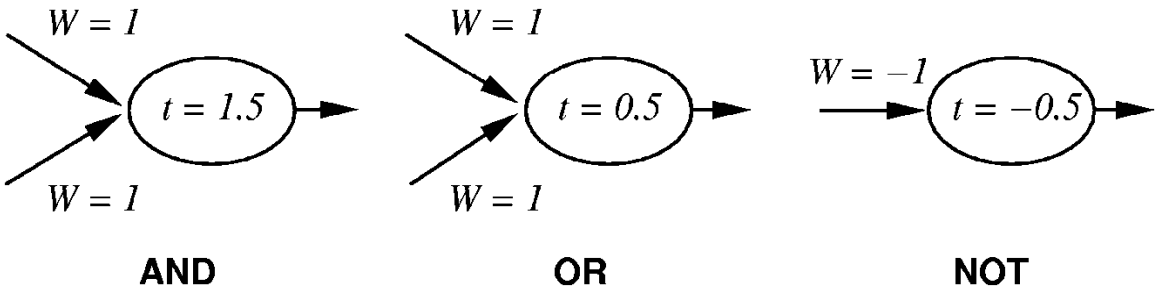
Beachte: *t* in $step_t$ stellt einen Schwellwert (*threshold*) dar – in Analogie zum Aktionspotential beim natürl. Nervensystem (s. Signalübertragung 2 auf Folie 8).

Mathematisch ist Schwellwert *t* äquivalent zu einer zusätzlichen Eingabe mit Aktivierungsniveau $a_0 = -1$ und Gewicht $w_{0,i} = t$:

$$a_i = step_t\left(\sum_{j=1,\ldots,n} w_{j,i} \cdot a_j\right) = step_0\left(\sum_{j=0,\ldots,n} w_{j,i} \cdot a_j\right).$$

Was kann man mit neuronalen Netzen darstellen?

- z.B. *Boolesche Funktionen* (mit Hilfe der $step_t$ Funktion):



… und durch Verschaltung vieler Einheiten beliebige Boolesche Funktionen

⇝ Schaltkreistheorie

⇝ Aber neuronale Netze können noch viel mehr …

Netzwerktopologien

- Rekurrente Netze oder zyklische Netze zeigen i.A. bidirektionale Verbindungen, aus denen beliebige Topologien gebildet werden können.

  nicht in dieser Vorlesung

  – Ausgaben des Netzes können dabei als Eingaben zurückgegeben werden.

  – Die Aktivierungslevel des Netzes bilden ein dynam. System, das einen stabilen Zustand erreichen kann, schwingt oder sogar chaotisches Verhalten zeigen kann.

  – Die Netzantwort für eine bestimmte Eingabe kann so vom Ausgangszustand abhängig sein, der wiederum von vorherigen Eingaben abhängig sein kann (↝ Art von Kurzzeitgedächtnis ↝ interessant für Gehirnmodellierung)

- Feed-forward-Netze oder azyklische Netze zeigen unidirektionale Verbindungen.

  Gegenstand dieser Vorlesung

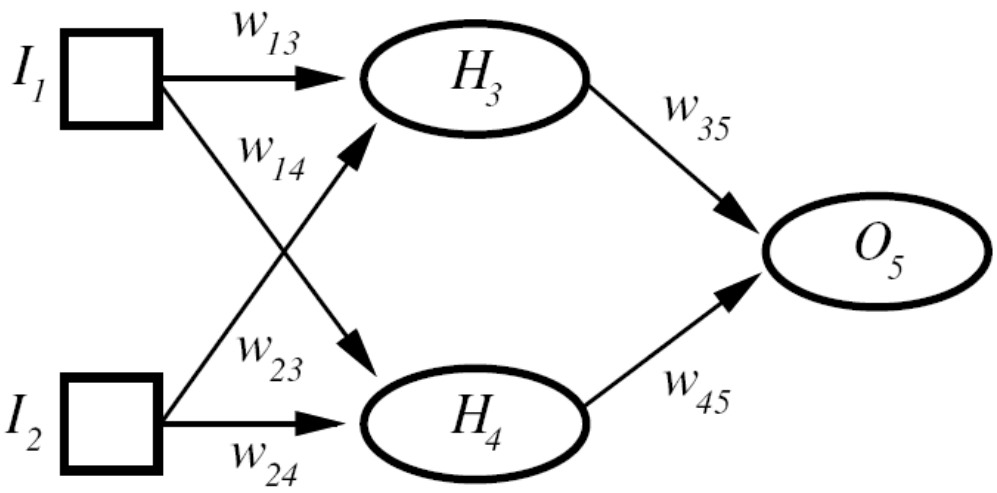
- Hybrid: Modular aus azyklischen und rückgekoppelten Komponenten aufgebaut. Beispiel: *Winner-takes-all-Netze* als unidirektionale Feed-forward-Netze mit lateraler bidirektionaler Inhibition in Wettbewerbsschichten.

  nicht in dieser Vorlesung

Feed-forward-Netze (FF-Netze)

Feed-forward-Netze werden i. A. in Schichten (Layer) konstruiert. Dabei existieren keine Verbindungen zwischen Einheiten einer Schicht, sondern immer nur zur jeweils nächsten Schicht (gerichtete Netze).

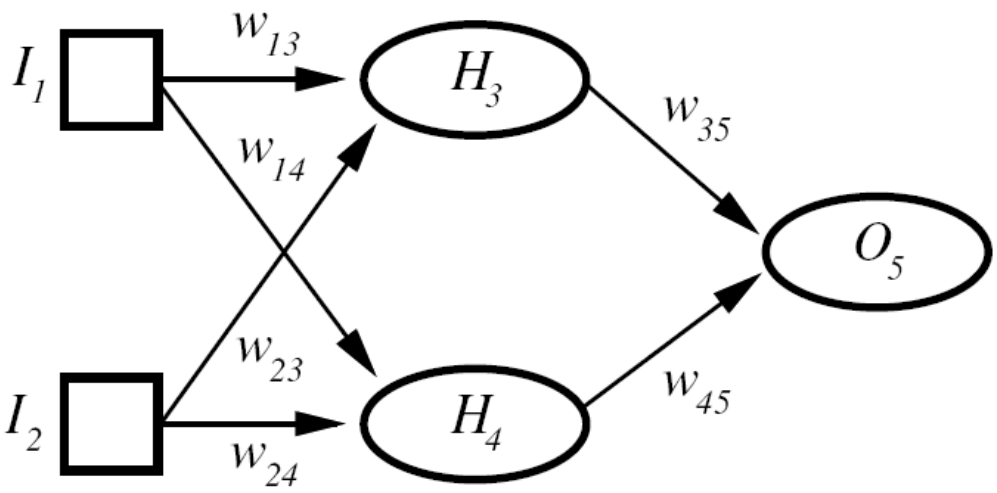
Beispiel:



Feed-forward-Netze sind am besten verstanden:

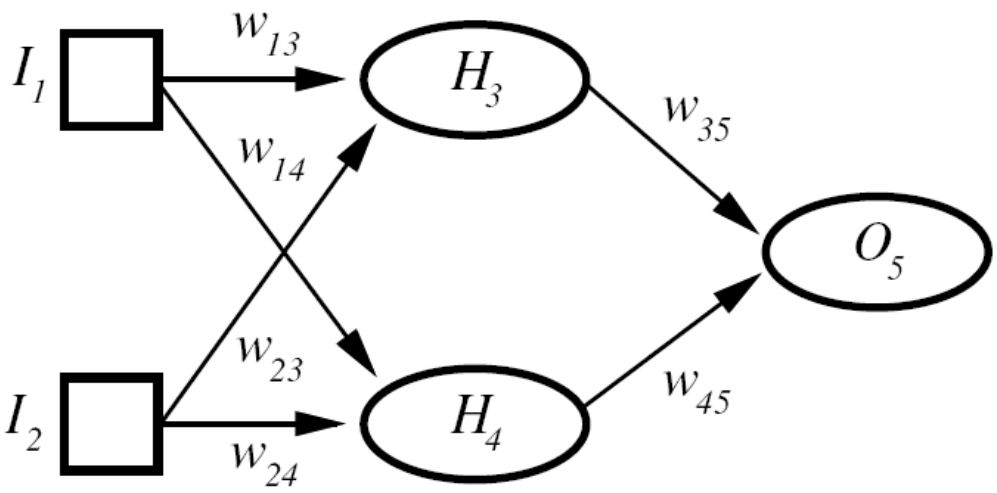
Die Ausgabe ist allein eine Funktion der Eingaben und der Gewichte.

Geschichtete Feed-forward-Netze (GFFs) (1)



- **Eingabeeinheiten** (hier: I_1, I_2) erhalten ihr Aktivierungsniveau von der Umwelt.
- **Ausgabeeinheiten** (hier: O_5) teilen ihr Ergebnis der Umwelt mit.
- **Verborgene Einheiten** (*Hidden Units*) (hier: H_3, H_4)
  - sind Einheiten ohne direkte Verbindungen zur Umwelt,
  - sind in verborgenen Schichten (*Hidden Layers*) angeordnet.

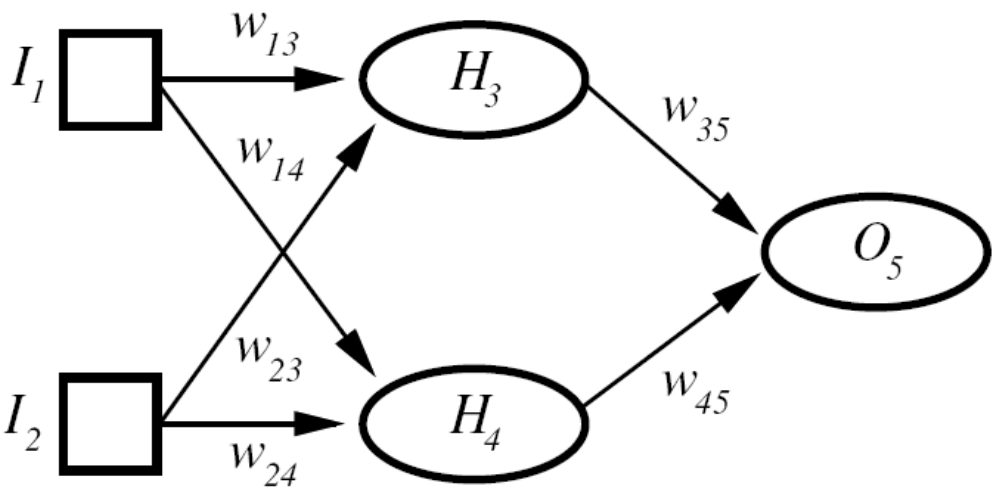
Geschichtete Feed-forward-Netze (GFFs) (2)



- **Anzahl der Schichten**: wird angegeben *unter Ignorierung der Eingabeschicht*.
- **Perzeptron**: einschichtiges FF-Netz ohne verborgene Schicht.
- **Mehrschichtige Netzwerke** (*Multilayer Networks*):

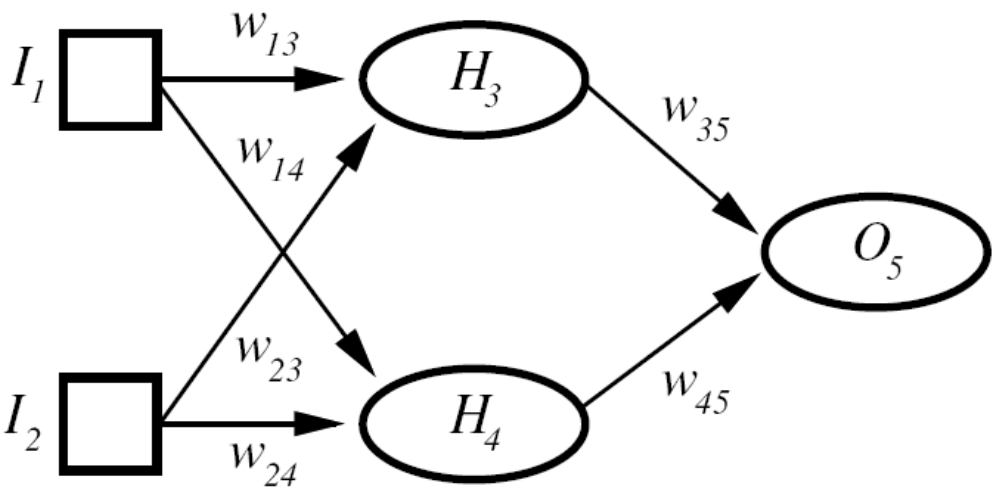
  mindestens eine verborgene Schicht.

Darstellungskraft von GFF-Netzen



- GFF-Netze *mit einer verborgenen Schicht* können beliebige *stetige Funktionen* darstellen.
- GFF-Netze *mit zwei verborgenen Schichten* können zusätzlich auch *diskontinuierliche Funktionen* darstellen.

Beispiel



Bei fester Topologie und Aktivierungsfunktion *g* sind darstellbare Funktionen charakterisierbar in parametrisierter Form (*Parameter* = *Gewichte*):

$$a_5 = g(w_{35} \cdot a_3 + \cdot w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot g(w_{13} \cdot I_1 + w_{23} \cdot I_2) + w_{45} \cdot g(w_{14} \cdot I_1 + w_{24} \cdot I_2))$$

$\Rightarrow$ Lernen in NNs = Suche nach richtigen Parameterwerten

$\Rightarrow$ aus statistischer Sicht: nichtlineare Regression

Lernen mit neuronalen Netzen

Gegeben eine Menge von Beispielen $E = \{e_1, \ldots, e_n\}$, wobei jedes Beispiel aus *Eingabewerten* und *Ausgabewerten* besteht.

Ziel: Netzwerk soll die Funktion lernen, die die Beispiele erzeugt hat, dabei ist die Netzwerktopologie vorgegeben und die Gewichte sollen angepasst werden.

Lernen in Epochen: man legt dem Netz die Beispiele mehrfach (in Epochen) vor.

```
function NEURAL-NETWORK-LEARNING(examples) returns network

  network ← a network with randomly assigned weights
  repeat
      for each e in examples do
          O ← NEURAL-NETWORK-OUTPUT(network, e)
          T ← the observed output values from e
          update the weights in network based on e, O, and T
      end
  until all examples correctly predicted or stopping criterion is reached
  return network
```

Optimale Netzwerkstruktur?

1) Die Wahl des richtigen Netzes ist ein schwieriges Problem!

2) Zudem kann das optimale Netz exponentiell groß relativ zur Eingabe werden.*

Probleme:

- Netz zu klein: gewünschte Funktion nicht darstellbar
- Netz zu groß: beim Trainieren lernt das Netzwerk die Beispiele „auswendig“, ohne zu generalisieren ⇝ *Overfitting*

Es gibt keine gute Theorie zur Wahl des richtigen Netzes!

---

* Z.B. Netze für Boolesche Funktionen: Ein Ergebnis aus der Schaltkreistheorie besagt, dass die *meisten* Booleschen Funktionen mit $O(2^n/n)$ Gattern dargestellt werden müssen.

Heuristik des *Optimal brain damage*

Optimal Brain Damage:

1. Wähle Netz mit maximaler Zahl an Verbindungen und trainiere das Netz.
2. Reduziere Zahl der Verbindungen mit Hilfe von Informationstheorie und trainiere das Netz erneut.
3. Wenn Performanz gleich oder besser, dann gehe zu 2.

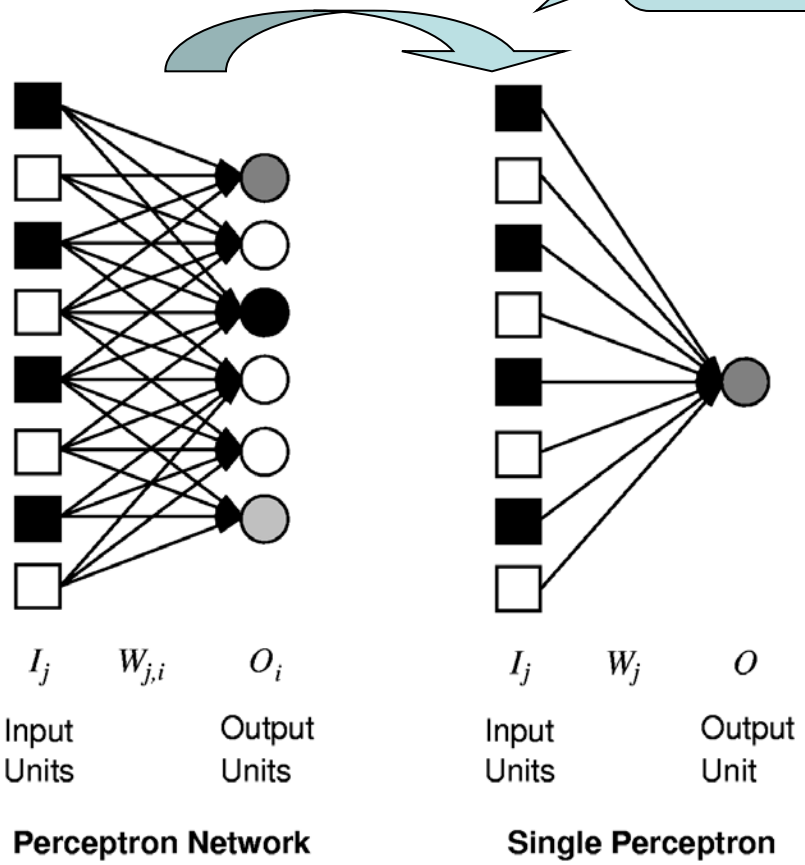
Beispiel: Netz zum Erkennen von handgeschriebenen Postleitzahlen. 3/4 der anfänglichen Verbindungen wurden eingespart!

Es gibt auch Verfahren, um von einem kleinen Netz zu einem besseren, größeren Netz zu kommen: Füge sukzessive Knoten für alle nicht erlernten Beispiele ein bis Trainingsmenge korrekt klassifiziert.

Perzeptron

Einschichten-FF-Netze oder *Perzeptrons* wurden insbes. in den 50er Jahren studiert. Sie waren die einzigen GFFs mit bekannten *effektiven* Lernverfahren.

Da alle Ausgabeknoten unabhängig voneinander sind, ist Betrachtung auf einen einzigen Ausgabeknoten *O* reduzierbar:



Mit einer Schwellwert-Aktivierungsfunktion ist z.B. eine Boolesche Funktion (s. Folie 13) darstellbar:

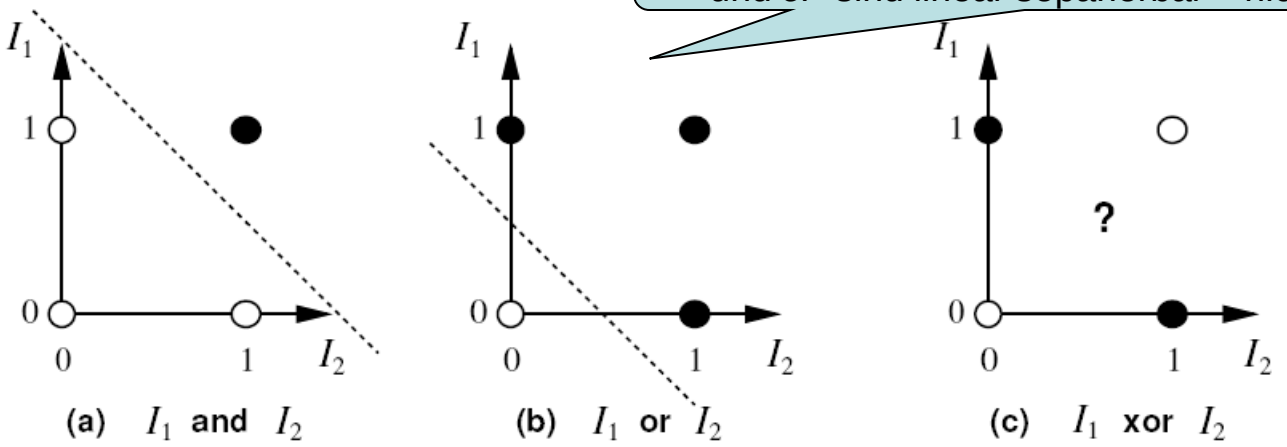
$$O = step_0\left(\sum_{j=0,\ldots,n} W_j \cdot I_j\right) = \boldsymbol{W}\cdot\boldsymbol{I}$$

mit $I_0 = -1$ als Schwellwert.

Was können Perzeptrons darstellen?

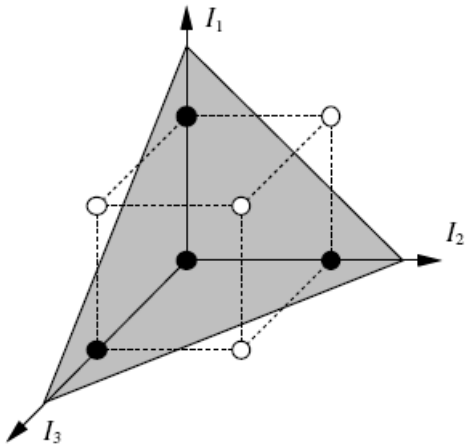
Aber: Perzeptrons können (nur) die Klasse aller *linear separierbaren Funktionen* darstellen:

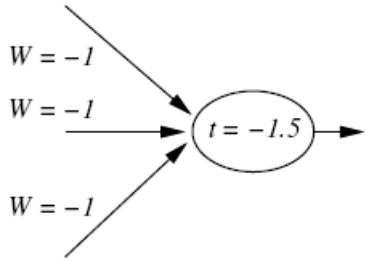
Urbilder von 0 und 1 der Booleschen Funktionen *and* und *or* sind linear separierbar – nicht so bei *xor*



Grund: Die Gleichung $\boldsymbol{W} \cdot \boldsymbol{I} = 0$ teilt den gesamten Raum entlang einer Geraden (2D), einer Ebene (3D), einer Hyperebene (allg.) genau in zwei Bereiche:

- Ausgabe 1, wenn $\boldsymbol{W} \cdot \boldsymbol{I} > 0$
- Ausgabe 0, wenn $\boldsymbol{W} \cdot \boldsymbol{I} \leq 0$





Was können Perzeptrons lernen?

↝ Alle linear separierbaren Funktionen.

Die meisten Lernverfahren folgen dem Ansatz der *Current Best Hypothesis*: es wird immer nur eine Hypothese betrachtet, die optimiert wird.

- Dabei gilt: Hypothese = Netz = aktuelle Werte der Gewichte.
- Start: Das Netzwerk wird mit zufällig gewählten Gewichten, in der Regel aus dem Intervall [-0.5, 0.5], initialisiert.
- Iterative Optimierung: Durch Ändern der Gewichte wird die Konsistenz mit den Trainingsbeispielen hergestellt
  - durch Reduzieren der Differenz zwischen berechnetem und vorgegebenem Wert,
  - definiert in der sog. *Perzeptron-Lernregel*.

Die Perzeptron-Lernregel

Sei *T* die vorgegebene korrekte Ausgabe und sei *O* die berechnete Ausgabe des aktuellen Netzes, dann ist der Fehler $Err = T - O$.

Lernziel:

- wenn *Err* positiv ist, sollte *O* erhöht werden
- wenn *Err* negativ ist, sollte *O* erniedrigt werden

Da jede Eingabeeinheit $W_j \cdot I_j$ zu *O* beiträgt, sollte zur Erhöhung von *O* das Gewicht W_j bei positivem I_j größer und bei negativem I_j kleiner werden:

$$W_j \leftarrow W_j + \alpha \cdot I_j \cdot Err$$

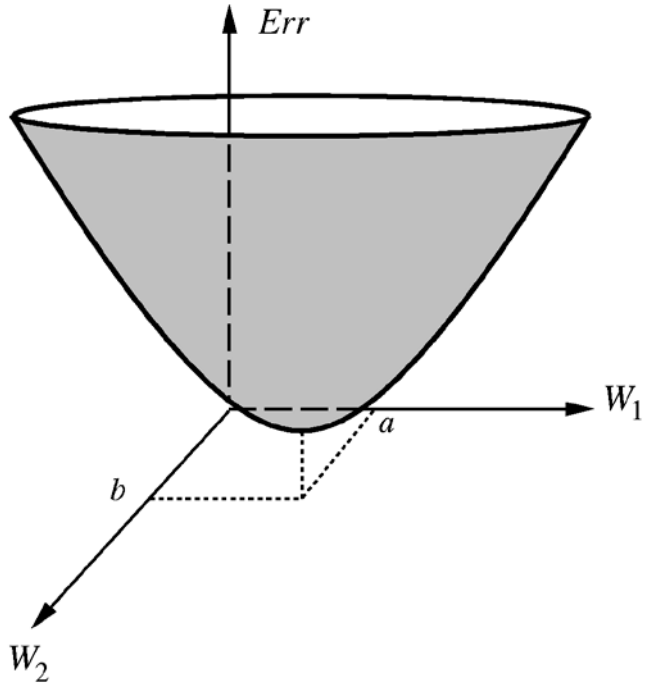
mit positiver Konstante α, die Lernrate genannt wird.

Satz [Rosenblatt 1960]: Lernen mit der Perzeptron-Lernregel konvergiert immer zu korrekten Werten bei linear separierbaren Funktionen.

Wieso funktioniert die Perzeptron-Lernregel?

- Perzeptron-Lernen entspricht *Gradientenabstieg* im Raum aller Gewichte, wobei der Gradient durch die *Fehlerfläche* bestimmt wird.
- Jede Gewichtskonfiguration bestimmt einen Punkt auf der Fläche.

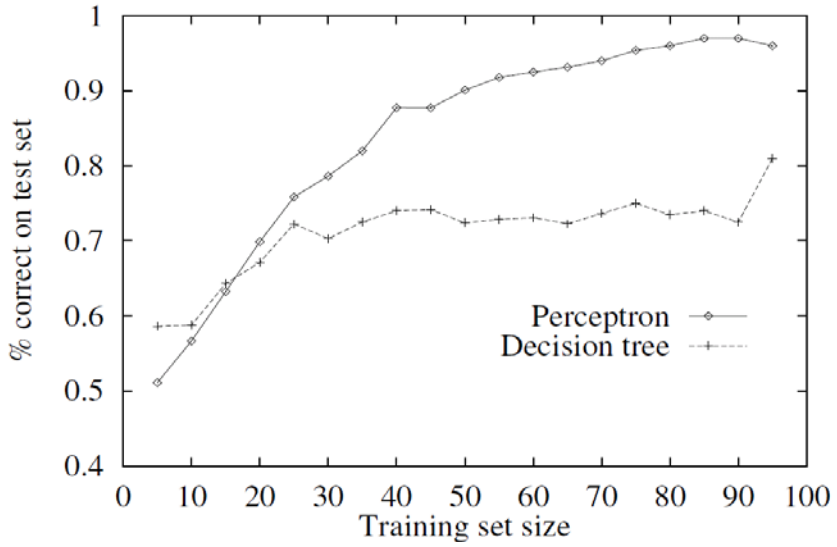
⇝ *Partielle Ableitung* bzgl. der einzelnen Gewichte.



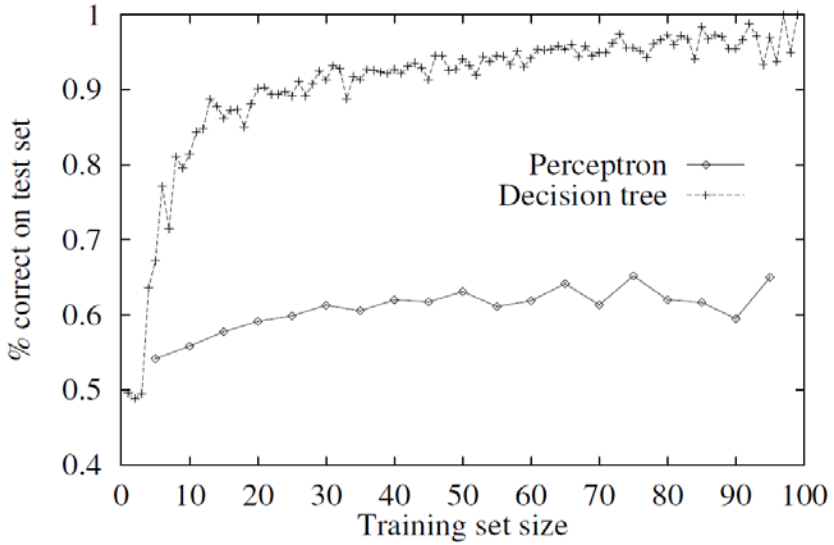
Da es bei linear separierbaren Funktionen keine lokalen Minima gibt, folgt der Perzeptron-Konvergenzsatz.

Lernkurven: Perzeptron vs. Entscheidungsbaum

Majoritätsproblem



Restaurantproblem



Majoritätsproblem (links): $O = 1$, wenn mind. $n/2$ von n Booleschen Eingaben wahr sind

⇝ kompakte Darstellung bei Perzeptron mit $t = n/2$ und $w_j = 1$ für $j = 1, \ldots, n$.

⇝ ungünstige Darstellung bei DT: Knotenzahl exponentiell in der Zahl der Attribute

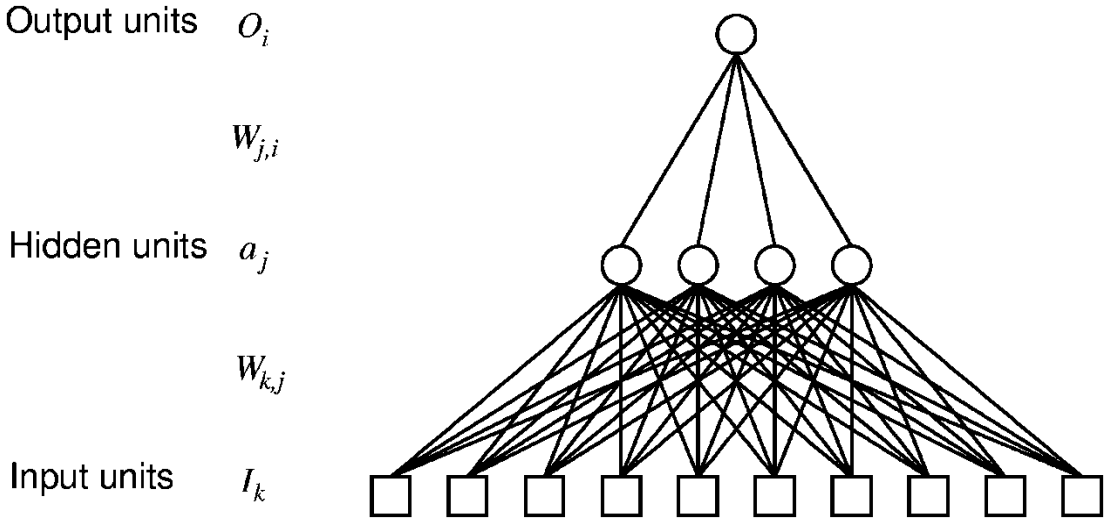
Restaurantproblem (rechts): nicht linear separierbar

⇝ beste Trennungsebene bei Perzeptron mit maximaler Korrektheit von 65%.

⇝ Multilayer-Feed-Forward-Netze als Ausweg?

Mehrschichten-FF-Netze

Beispiel:



Das Restaurantproblem als 2-Schichten-FF-Netz:

- 10 Eingaben I_k für die 10 Attribute
- 4 innere Knoten mit Aktivierungsniveaus a_i
- 1 Ausgabe O_i für das „*will wait*“ - Zielprädikat

Bei Perzeptron: Ausgabeabweichung führt zu Korrektur des „verantwortlichen Gewichtes“ zwischen entsprechenden Eingabeknoten und dem Ausgabeknoten.

Bei Mehrschichten-Netz: Verteilte Fehlerverantwortlichkeit ⇝ Fehlerpropagation!

Herleitung

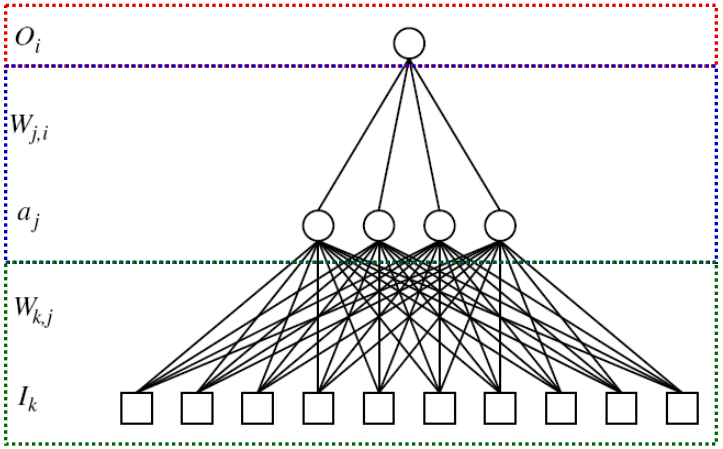
Seien ***W*** der Gewichtsvektor und

E die folgende Fehlerfunktion:

$$E(\boldsymbol{W}) = \tfrac{1}{2} \sum_i (T_i - O_i)^2$$

$$= \tfrac{1}{2} \sum_i (T_i - g(\sum_j (W_{j,i} \cdot a_j)))^2 \quad (*)$$

$$= \tfrac{1}{2} \sum_i (T_i - g(\sum_j (W_{j,i} \cdot g(\sum_k (W_{k,j} \cdot I_k)))))^2$$



„Verantwortlichkeit der 1. Stufe“ (*): partielle Ableitung von *E* nach $W_{j,i}$: $\partial E / \partial W_{j,i}$

↝ Abzuleiten also: $\tfrac{1}{2} \sum_i (T_i - g(\sum_j (W_{j,i} \cdot a_j)))^2$

Nur jeweils ein Summand in den Summen über *i* und *j* ist von einem bestimmten W_{ji} abhängig. Die anderen Summanden verschwinden also als Konstante beim Differenzieren nach W_{ji}.

$$\partial E / \partial W_{j,i} = 2 \cdot \tfrac{1}{2} (T_i - g(\sum_j (W_{j,i} \cdot a_j))) \cdot - g'(\sum_j (W_{j,i} \cdot a_j)) \cdot a_j$$

$$= - (T_i - O_i) \cdot g'(in_i) \cdot a_j$$

$$= - a_j \cdot \Delta_i \qquad \text{mit } \Delta_i = (T_i - O_i) \cdot g'(in_i).$$

Analog kann man für $W_{k,j}$ zeigen: $\partial E / \partial W_{j,i} = - I_k \cdot \Delta_j$ mit $\Delta_j = g'(in_j) \cdot \sum_i g(W_{j,i} \cdot \Delta_i)$.

Lernen in Mehrschichten-Netzen: *Backpropagation* (1)

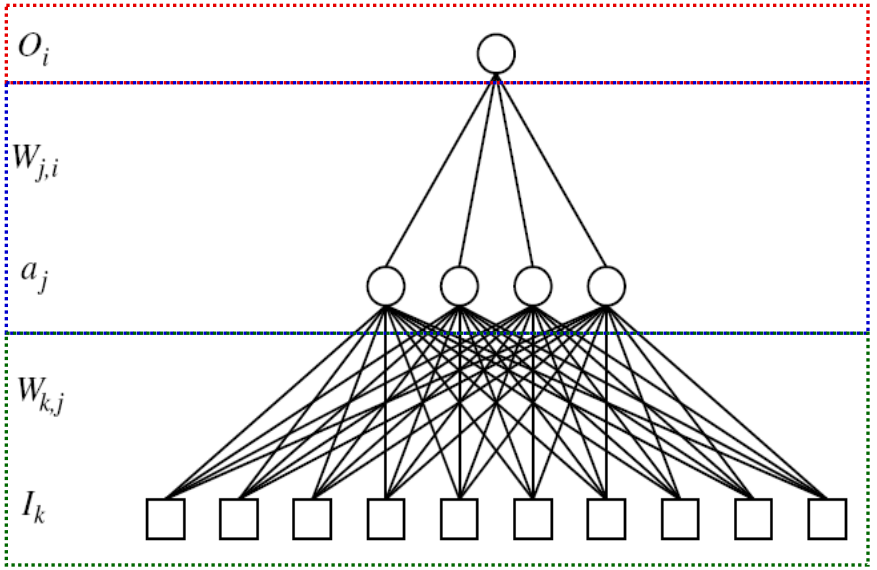
Für alle Gewichte wird ihr Anteil am Fehler bestimmt und die Gewichte entsprechend geändert, d.h. der Fehler wird rückwärts durchs Netz propagiert.

Für Gewichte $W_{j,i}$ zu Ausgabeknoten i:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \cdot a_j \cdot \Delta_i$$

mit $\Delta_i = Err_i \cdot g'(in_i)$

und $Err_i = T_i - O_i$



Für Gewichte $W_{k,j}$ zu inneren Knoten j:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \cdot I_k \cdot \Delta_j \quad \text{mit } \Delta_j = g'(in_j) \cdot \Sigma_i \, g(W_{j,i} \cdot \Delta_i)$$

Da g differenzierbar sein muss, wählt man für g meist die *Sigmoid*-Funktion.

Der Backpropagation-Algorithmus

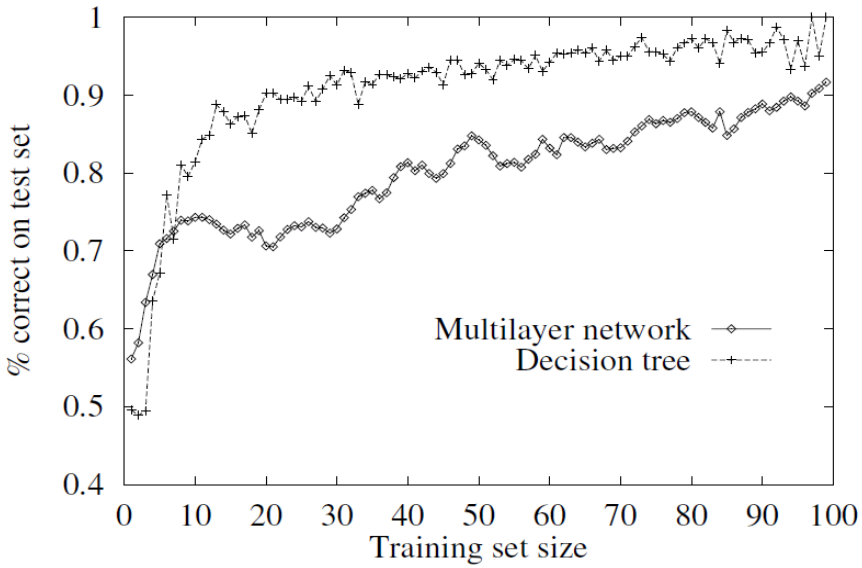
function BACK-PROP-LEARNING($examples$, $network$) **returns** a neural network
inputs: $examples$, a set of examples, each with input vector **x** and output vector **y**
$network$, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

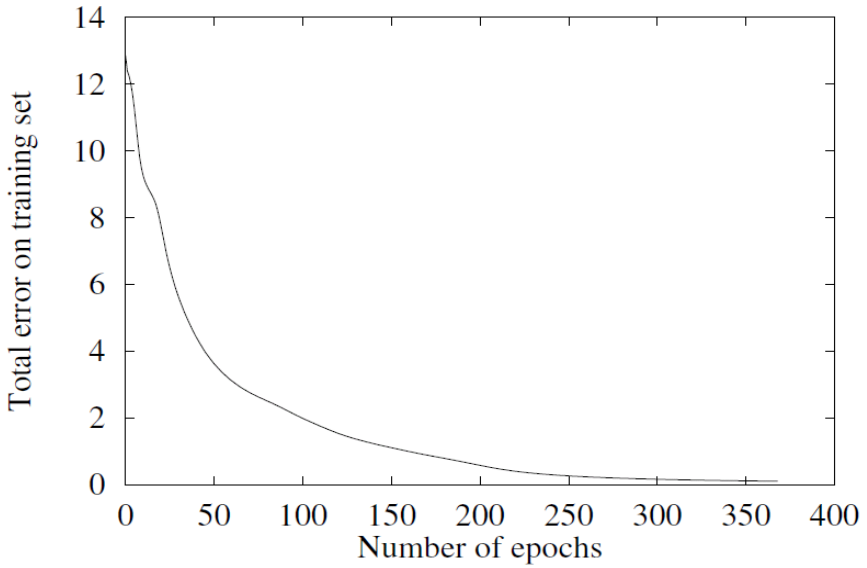
repeat
for each example (**x**, **y**) **in** $examples$ **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j} \; a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \; \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in $network$ **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return $network$

Lernkurven

Lernkurve für das Restaurantbeispiel mit dem 2-Schichtennetz:

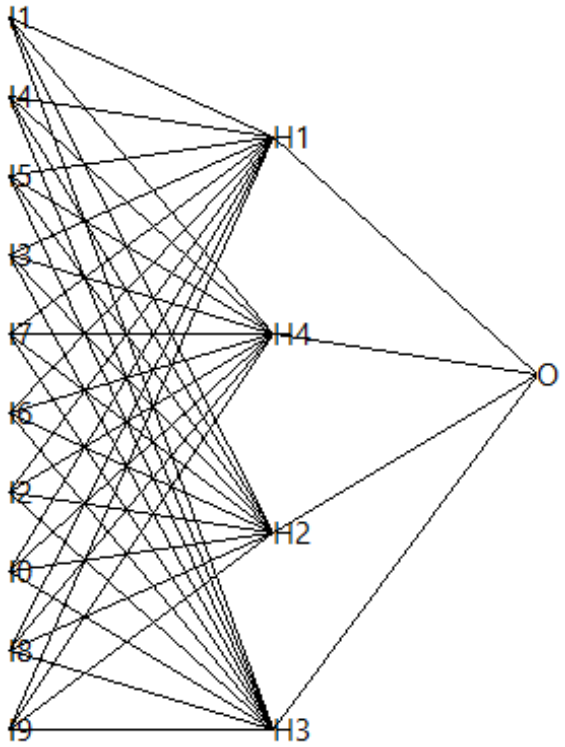
- links: im Vergleich mit der Lernkurve für das Entscheidungsbaum-Lernen,
- rechts: Fehler in Abhängigkeit von der Anzahl der Lernepochen bei Trainingsmenge mit 100 Beispielen.

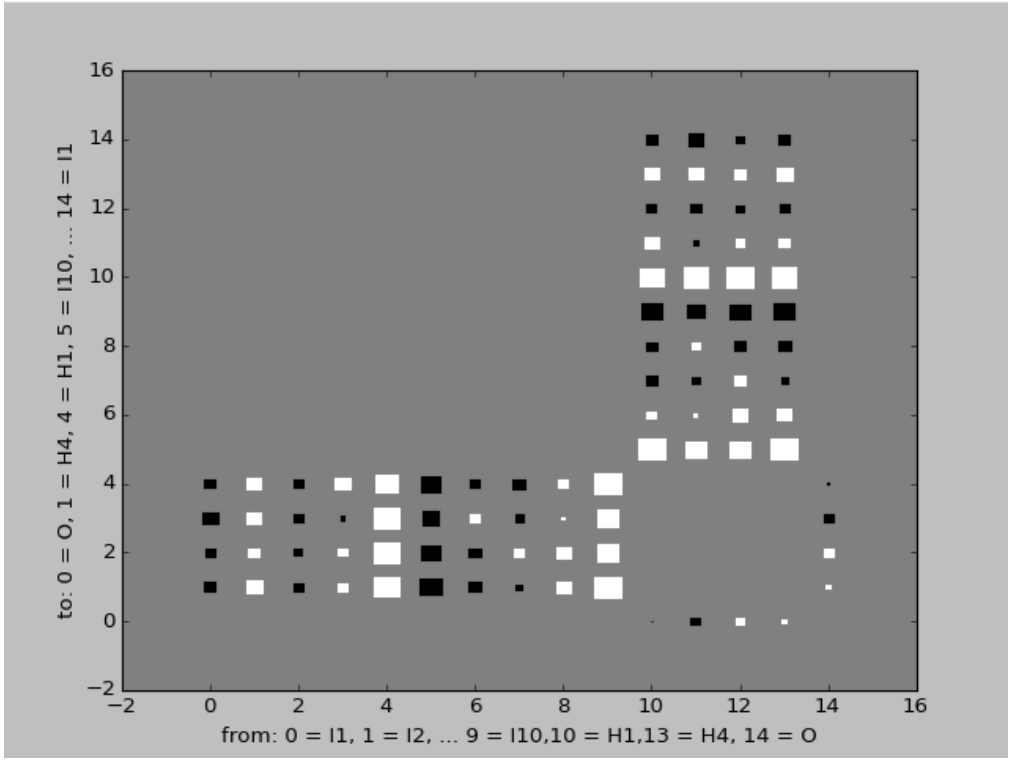




Hinton-Diagramme

Hinton-Diagramme dienen der Wertevisualisierung der Kantengewichte eines KNNs. Jedes Feld steht für ein Kantengewicht. Seine Größe ist proportional zum Wert des Kantengewichts. Seine Farbe (hier schwarz oder weiß) kodiert das Vorzeichen (weiß = positiv, schwarz = negativ).





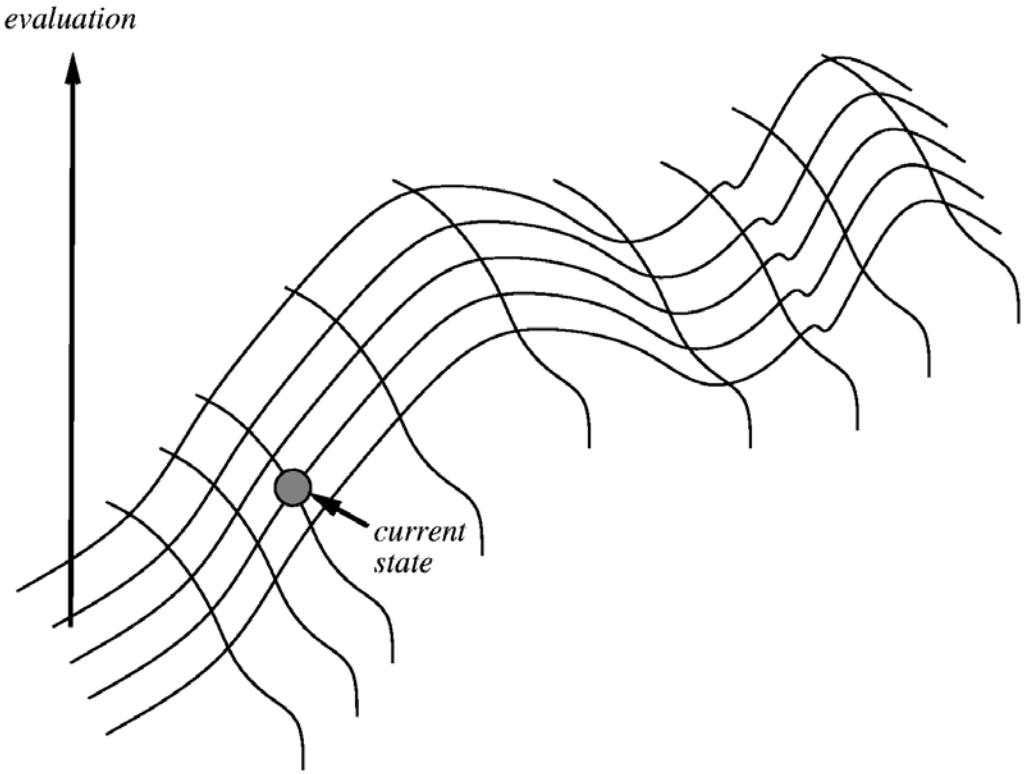
Screenshots von AIMA-Py, s. Manual zu AIMA-Py, Institut für Informatik 4, Univ. Bonn, 2015.

Backpropagation = Gradientenabstieg

Backpropagation kann wieder als *Gradientenabstieg* im Raum aller Gewichte aufgefasst werden, wobei der Gradient durch die *Fehlerfläche* bestimmt wird.

Allerdings treten nun nicht mehr rein konvexe Fehlerflächen auf, sondern der Lernalgorithmus ist mit dem Problem *lokaler Minima* konfrontiert.

- ⇝ Neustarten
- ⇝ Tabusuche
- ⇝ Simulated Annealing



Komplexität, Generalisierung, Robustheit und Transparenz

- Wofür sind sie gut? Neuronale Netze eignen sich zum Lernen von Funktionen über Attributen, die auch *kontinuierlich* sein können.
- Größe des Netzes: bis zu $2^n/n$ verborgene Einheiten mit 2^n Gewichten, um beliebige Boolesche Funktionen darzustellen, meist aber weniger.
- Effizienz des Lernens: Zeitkomplexität pro Epoche: $O(m|\mathbf{W}|)$ für m Beispiele und $|\mathbf{W}|$ Gewichte. Worst-Case-Zahl der benötigten Epochen kann exponentiell in der Zahl n der Eingaben sein. Problem der lokalen Minima.
- Generalisierung: gut, wenn man das richtige Netz gewählt hat … dafür aber keine Theorie … aber Heuristiken wie *Optimal Brain Damage*.
- Rauschen: nichtlineare Regression ist sehr tolerant gegenüber Rauschen.
- Transparenz: schlecht! Oft weiß man nicht, warum das Netz gut funktioniert ⇝ Black-Box-Ansatz.

Beispielanwendung: Aussprache lernen (1)

Ein Klassiker: das NETtalk-Programm [Sejnowski & Rosenberg 87] lernt aus transkribierten englischsprachigen Texten Ausspracheregeln, also die Abbildung von Text zu Phonemen.

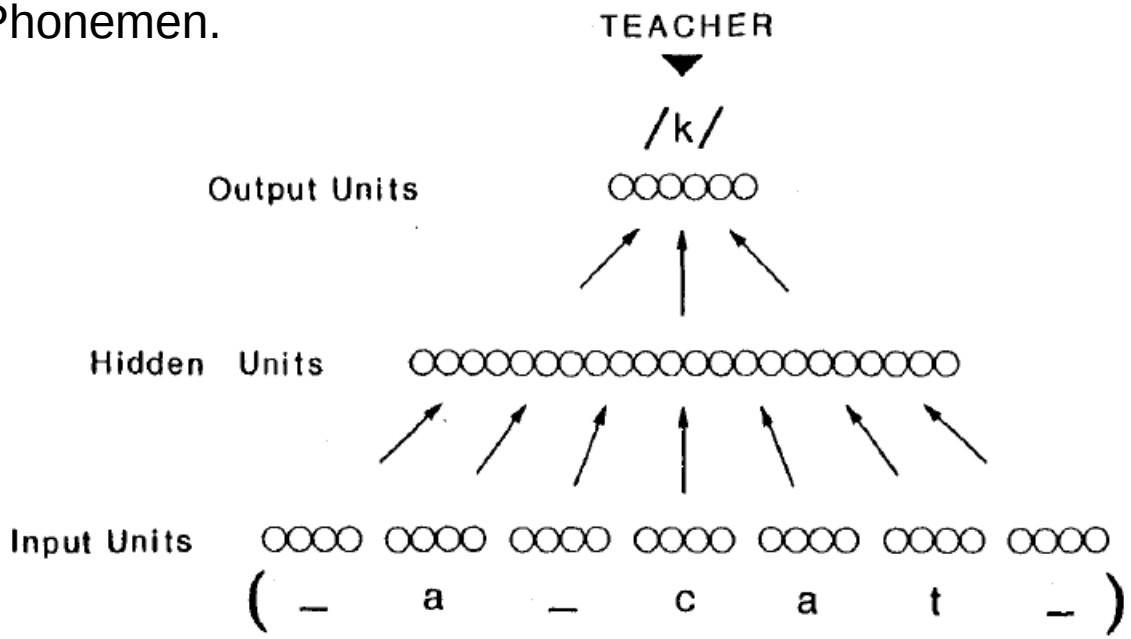


Figure 1: Schematic drawing of the NETtalk network architecture. A window of letters in an English text is fed to an array of 203 input units. Information from these units is transformed by an intermediate layer of 80 "hidden" units to produce patterns of activity in 26 output units. The connections in the network are specified by a total of 18629 weight parameters (including a variable threshold for each unit).

Sejnowski, *T*., *Rosenberg*, *C*. (*1987*) *Parallel networks that learn to pronounce English text*. Complex Systems, 1, 145-168

Beispielanwendung: Aussprache lernen (2)

Aufbau:

- 7 Gruppen zu 29 Eingabeeinheiten: Zeichen + drei Vorgänger- und Nachfolgezeichen; 29 Eingabeeinheiten pro Zeichen (26 Buchstaben + Zeichensetzung);
- 80 verborgene Einheiten (hidden units)
- 26 Ausgabeeinheiten zur Steuerung von Merkmalen des zu produzierenden Lautes wie betont/unbetont, hoch/tief, etc.

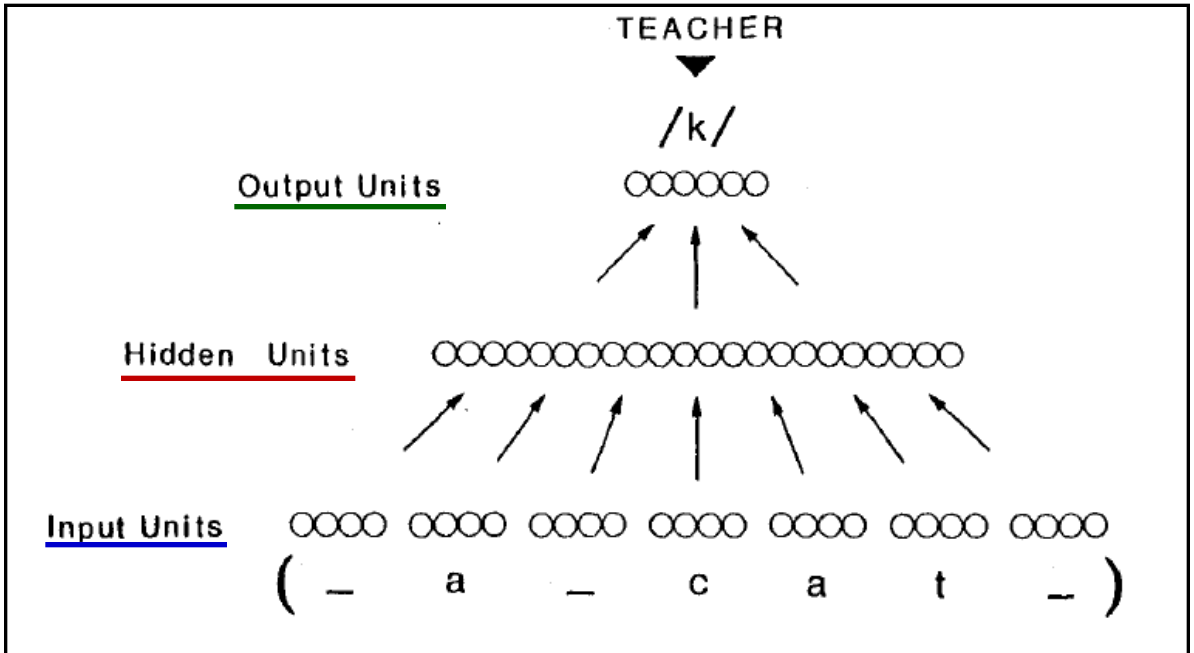


Figure 1: Schematic drawing of the NETtalk network architecture. A window of letters in an English text is fed to an array of 203 input units. Information from these units is transformed by an intermediate layer of 80 "hidden" units to produce patterns of activity in 26 output units. The connections in the network are specified by a total of 18629 weight parameters (including a variable threshold for each unit).

Beispielanwendung: Aussprache lernen (3)

Ergebnis:

- Nach 50 Episoden auf einem Text mit 1024 Worten hat NETtalk 95% Korrektheit auf der Trainingsmenge erreicht.
- Allerdings nur 78% auf der Testmenge.

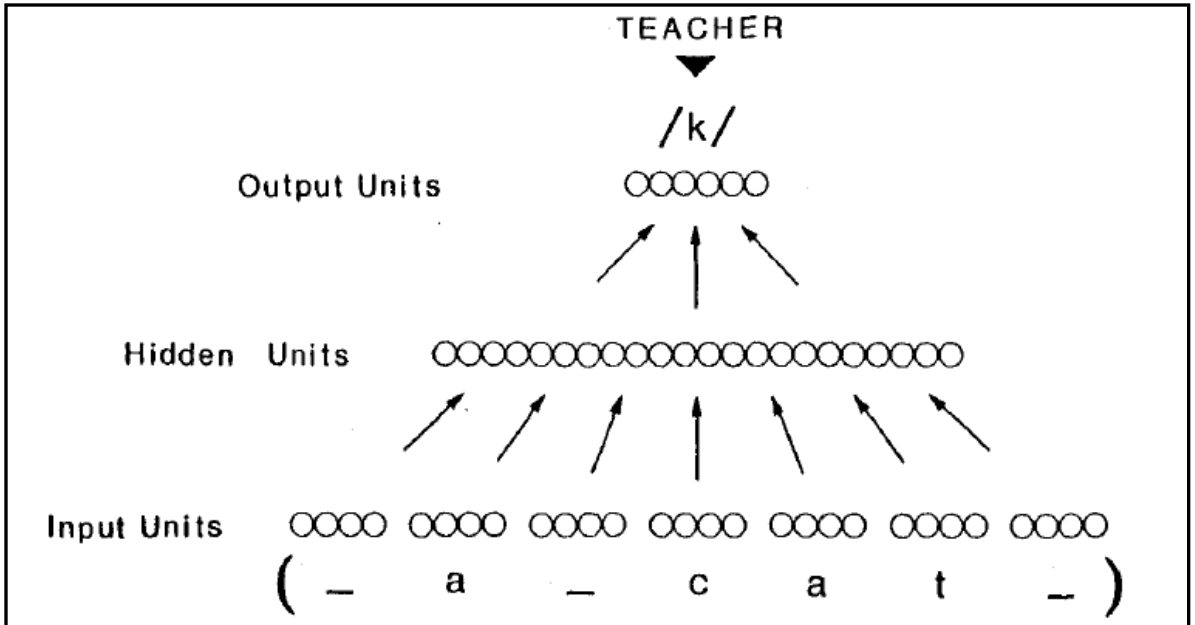
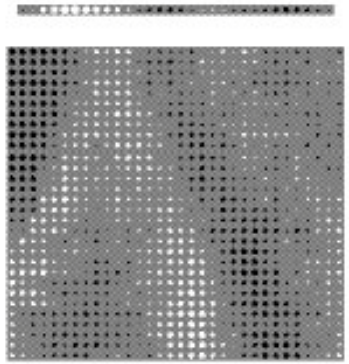
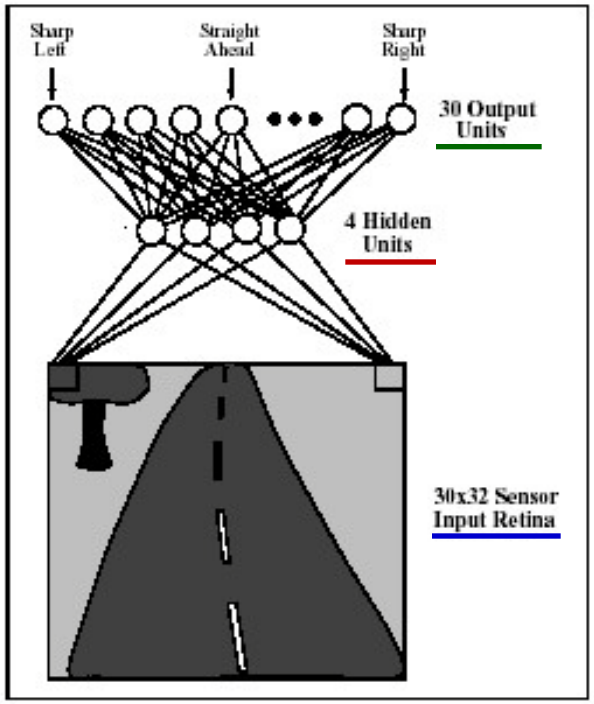


Figure 1: Schematic drawing of the NETtalk network architecture. A window of letters in an English text is fed to an array of 203 input units. Information from these units is transformed by an intermediate layer of 80 "hidden" units to produce patterns of activity in 26 output units. The connections in the network are specified by a total of 18629 weight parameters (including a variable threshold for each unit).

Beispielanwendung: Auto fahren (1)

- ALVINN [Pomerleau 93] steuert ein Fahrzeug auf einer Straße.
- Eingabe von Videokamerabild, das zu einem 30x32-Bild für 960 Eingabeeinheiten transformiert wird.
- Vier verborgene Einheiten sowie 30 Ausgabeeinheiten für die Steuerungskommandos.

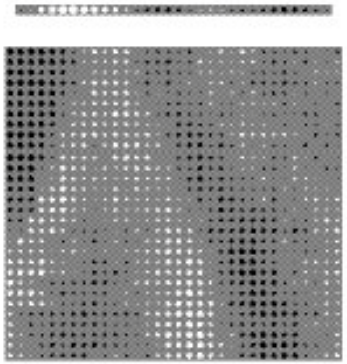


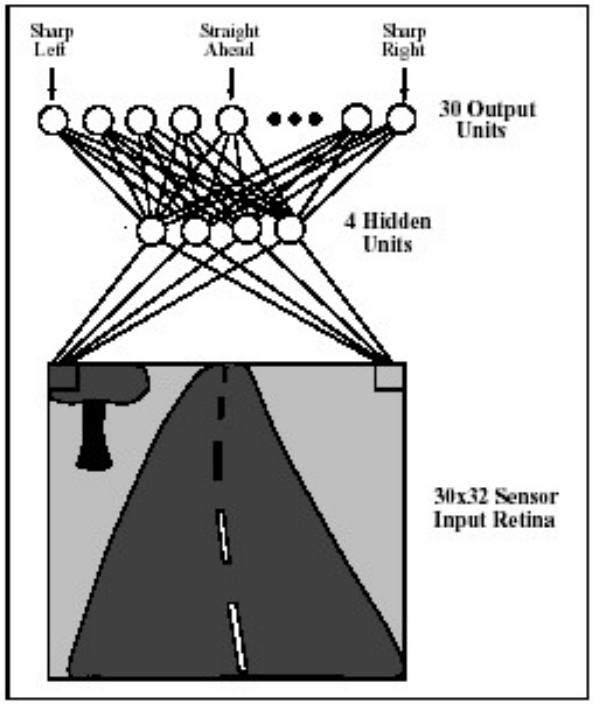


Dean Pomerleau: Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving. Robot Learning, J. Connell and S. Mahadevan, ed., 1993.

Beispielanwendung: Auto fahren (2)

- Trainingsdaten werden durch menschlichen Fahrer gewonnen.
- Nach fünf Minuten Fahrt als Trainingsdaten kann ALVINN auf den trainierten Straßen bis zu 70 mph schnell fahren.
- Probleme: andere Straßentypen und andere Beleuchtungen.
- MANIAC [Jochem et al 93] ist ein Metasystem, welches das „richtige“ ALVINN-Netz für die aktuelle Straße auswählt.





Todd M. Jochem , Dean A. Pomerleau , Charles E. Thorpe: MANIAC: A Next Generation Neurally Based Autonomous Road Follower. Proc. Intern. Conf. on Intelligent Autonomous Systems (IAS), 1993.

Beispielanwendung: Deep Learning

BIOGRAPHICAL INFORMATION
.....Curriculum Vitae .pdf
.....Biographical sketch
.....Photograph1 (.jpg)
.....Photograph2 (.jpg)

PUBLICATIONS
.....Publications by year
.....Slides of public talks

CURRENT & RECENT COURSES
....csc321 Spring 2013(undergrad)
....csc2535 Spring 2013(graduate)
....csc2515 Fall 2008(graduate)

VIDEO TALKS & TUTORIALS
....YouTube (2012) Brains, Sex and Machine Learning (1hr)
....YouTube (2007) The Next Generation of Neural Networks (1hr)
....YouTube (2010) Recent Developments in Deep Learning (1hr)
.... Interview on CBC radio "Quirks and Quarks" Feb 11 2011
.... Interview on CBC radio "The Current", May 5 2015
....Tutorial (2009) Deep Belief Nets (3hrs) ppt pdf readings
....Workshop Talk (2007) How to do backpropagation in a brain (20mins) ppt2007 pdf2007 ppt2014 pdf2014

OLD TUTORIAL SLIDES
....2011 NIPS workshop talk pdf ppt
.....paper on Transforming Autoencoders
....2007 NIPS tutorial html ppt ps pdf
.....Readings: 2007 NIPS tutorial
....CIFAR Summer School 2007
....CIAR Summer School 2006
....CIFAR Summer School 2005
....List of Past Tutorials

MOVIES
....generating digits

Geoffrey E. Hinton

I now work part-time for Google as a Distinguished Researcher and part-time for the University of Toronto as a Distinguished Emeritus Professor. For much of the year, I work at the University from 9.30am to 1.30pm and at the Google Toronto office at 111 Richmond Street from 2.00pm to 6.00pm. I also spend several months per year working full-time for Google in Mountain View, California.

Department of Computer Science
University of Toronto
6 King's College Rd.
Toronto, Ontario
M5S 3G4, CANADA

email: hinton [at] cs [dot] toronto [dot] edu
voice: 416-978-7564
fax: 416-978-1455
office: Pratt 290G
Directions for Visitors

Check out the new web page for Machine Learning at Toronto

Information for prospective students:
I will not be taking any more graduate students, visiting students, summer students or visitors, so please do not apply to work with me.

News
Results of the 2012 competition to recognize 1000 different types of object
How George Dahl won the competition to predict the activity of potential drugs
How Vlad Mnih won the competition to predict job salaries from job advertisements
How Laurens van der Maaten won the competition to visualize a dataset of potential drugs

Basic papers on deep learning

Hinton, G. E., Osindero, S. and Teh, Y. (2006)
A fast learning algorithm for deep belief nets.
Neural Computation, **18**, pp 1527-1554. [pdf]
Movies of the neural network generating and recognizing digits

Hinton, G. E. and Salakhutdinov, R. R. (2006)
Reducing the dimensionality of data with neural networks.
Science, Vol. 313. no. 5786, pp. 504 - 507, 28 July 2006.
[full paper] [supporting online material (pdf)] [Matlab code]

NEW LeCun, Y., Bengio, Y. and Hinton, G. E. (2015)
Deep Learning
Nature, Vol. 521, pp 436-444. [pdf]

Papers on deep learning without much math

Hinton, G. E. (2007)
To recognize shapes, first learn to generate images
In P. Cisek, T. Drew and J. Kalaska (Eds.)
Computational Neuroscience: Theoretical Insights into Brain Function. Elsevier. [pdf of final draft]

Homepage von Prof. G. Hinton an der Toronto Univ: http://www.cs.toronto.edu/~hinton/ (Abruf: 13. Juni 2016)

Beispielanwendung: ImageNet 2012

IMAGENET **Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)**

Held in conjunction with PASCAL Visual Object Classes Challenge 2012 (VOC2012)

Back to Main page

All results

- Task 1 (classification)
- Task 2 (localization)
- Task 3 (fine-grained classification)
- Team information and abstracts

Task 1

Team name	Filename	Error (5 guesses)	Description
SuperVision	test-preds-141-146.2009-131-137-145-146.2011-145f.	0.15315	Using extra training data from ImageNet Fall 2011 release
SuperVision	test-preds-131-137-145-135-145f.txt	0.16422	Using only supplied training data
ISI	pred_FVs_wLACs_weighted.txt	0.26172	Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.
ISI	pred_FVs_weighted.txt	0.26602	Weighted sum of scores from classifiers using each FV.
ISI	pred_FVs_summed.txt	0.26646	Naive sum of scores from classifiers using each FV.
ISI	pred_FVs_wLACs_summed.txt	0.26952	Naive sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and

http://www.image-net.org/challenges/LSVRC/2012/results.html (Abruf: 13. Juni 2016)

Beispielanwendung: ImageNet 2016

IMAGENET **Large Scale Visual Recognition Challenge 2016 (ILSVRC2016)**

News History Timetable Introduction Challenges FAQ Citation Contact

News

- May 31, 2016: Register your team and download data at here.
- May 26, 2016: Tentative time table is announced.
- May 3, 2016: Stay tuned. Coming soon.

History

2015, 2014, 2013, 2012, 2011, 2010

Tentative Timetable

- May 31, 2016: Development kit, data, and registration made available.
- September 9, 2016, 5pm PDT: Submission deadline.
- September 16, 2016: Challenge results released.
- October, 2016: Most successful and innovative teams present at ECCV 2016 workshop.

Introduction

This challenge evaluates algorithms for object localization/detection from images/videos and scene classification/segmentation at scale.

I. Object localization for 1000 categories.
II. Object detection for 200 fully labeled categories.
III. Object detection from video for 30 fully labeled categories.
IV. Scene classification for 365 scene categories (Joint with MIT Places team) on Places2 Database http://places2.csail.mit.edu.
V. Scene segmentation[New] for 150 stuff and discrete object categories (Joint with MIT Places team).

Challenges

I: Object localization

The data for the classification and localization tasks will remain unchanged from ILSVRC 2012 . The validation and test data will consist of 150,000 photographs, collected from flickr and other search engines, hand labeled with the presence or absence of 1000 object categories. The 1000 object categories contain both internal nodes and leaf nodes of ImageNet, but do not overlap with each other. A random subset of 50,000 of the images with labels will be released as validation data included in the development kit along with a list of the 1000 categories. The remaining images will be used for evaluation and will be released without labels at test time. The training data, the subset of ImageNet containing the 1000 categories and 1.2 million images, will be packaged for easy downloading. The validation and test data for this competition are not contained in the ImageNet training data.

http://image-net.org/challenges/LSVRC/2016/ (Abruf: 13. Juni 2016)

Zusammenfassung & Ausblick

- Neuronale Netze sind ein Berechnungsmodell, das einige Aspekte des Gehirns nachbildet.
- Feed-forward-Netze sind die am einfachsten zu analysierenden Netze.
- Ein Perzeptron ist ein einschichtiges FF-Netz, mit dem man aber nur *linear separierbare* Funktionen repräsentieren kann. Mit Hilfe der Perzeptron-Lern-regel können solche Funktionen erlernt werden.
- Mehrschichtige Netze können beliebige Funktionen darstellen.
- Backpropagation ist ein Verfahren, um Funktionen in solchen Netzen zu lernen, und das auf Gradientenabstieg beruht. In der Praxis kann man damit viele Probleme lösen, allerdings gibt es keine Garantien.

Anhang: Zusammenfassung der Notation

Notation	Meaning
a_i $\mathbf{a}_i$	Activation value of unit i (also the output of the unit) Vector of activation values for the inputs to unit i
g g'	Activation function Derivative of the activation function
Err_i Err^e	Error (difference between output and target) for unit i Error for example e
I_i $\mathbf{I}$ $\mathbf{I}^e$	Activation of a unit i in the input layer Vector of activations of all input units Vector of inputs for example e
in_i	Weighted sum of inputs to unit i
N	Total number of units in the network
O O_i $\mathbf{O}$	Activation of the single output unit of a perceptron Activation of a unit i in the output layer Vector of activations of all units in the output layer
t	Threshold for a step function
T $\mathbf{T}$ $\mathbf{T}^e$	Target (desired) output for a perceptron Target vector when there are several output units Target vector for example e
$W_{j,i}$ W_i $\mathbf{W}_i$ $\mathbf{W}$	Weight on the link from unit j to unit i Weight from unit i to the output in a perceptron Vector of weights leading into unit i Vector of all weights in the network