Karlsruhe Institute of Technology

Neuronale Netze

Grundlagen und Anwendungen

Prof. Dr.-Ing. J. Marius Zöllner

„Not the neural devices themselves contain the secret of thought. It is, rather, the organizing principle by which vast numbers of these elementary devices work in concert. Neural computation is an emergent property of the system.“

Carver Mead (1989)

Professor am California Institute of Technology (CalTech) forderte 1968 alle gängigen Überzeugungen heraus, indem er berechnete, wie klein ein funktionierender Transistor sein könnte. (0,15 Micron)

Ein biologischen Vorbild

- Der Mensch, sein Gehirn:
  - Neuron Schaltzeit: > 0.001 sec
  - Anzahl Neuronen: 10^{10}
  - Verbindungen (Synapsen) pro Neuron: 10^4-10^5
  - Szenenerkennung: 0.1 sec

  - hochparallele Berechnung
  - verteilte Repräsentation von Wissen

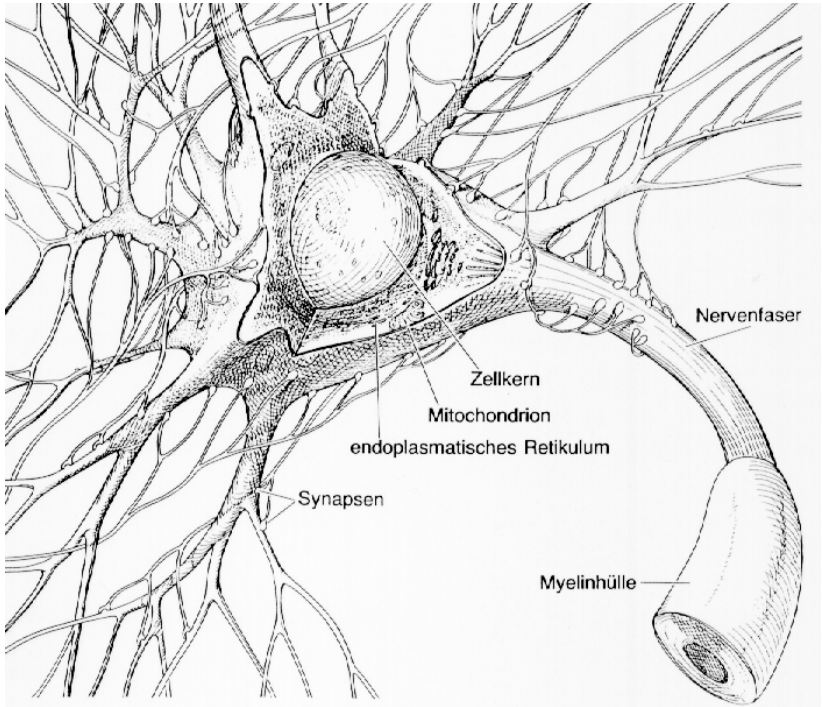
Vergleich zw. Gehirn und seriellem Rechner

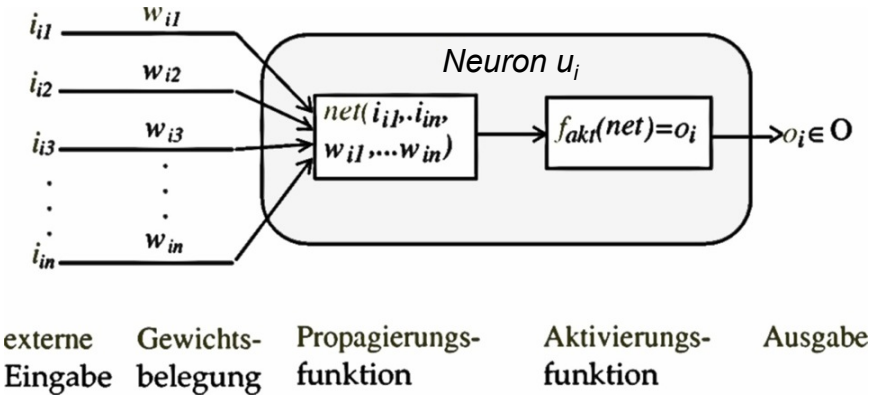
Eigenschaft	Parallelität	Präzision	Fehler-toleranz	Speicher-zugriff	Erkennen v. Mustern u. Ähnlichkeiten
Gehirn	hoch	mäßig	hoch	global	gut
ser. Rechner	noch mäßig	hoch	niedrig	lokal	mäßig

Eigenschaft	Numerische präzise Be-rechnungen	Fehlerloses Speichern v. Daten	Rekonstrukt. teilw. zerst. Daten	Verallgem. v. Bsp. auf implizite Regeln	Selbst-organisation
Gehirn	schlecht	schlecht	gut	gut	ja
ser. Rechner	gut	gut	schlecht	schlecht	bisher nicht

„Konnektionistische“ Verfahren

Künstliche Neuronale Netze (*KNN*) - Nachbildung des Gehirns ?





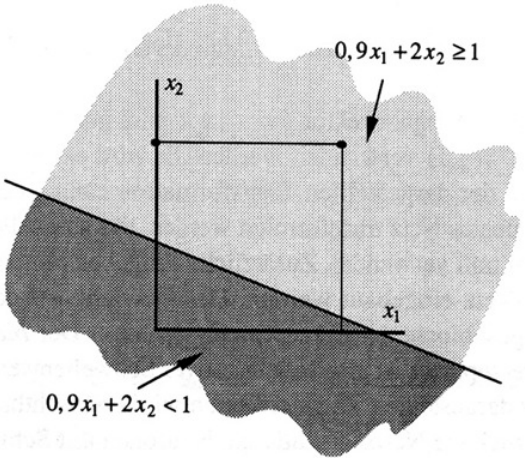
Große Anzahl einfacher Recheneinheiten (Neuronen)
Durch gewichtete Kanäle verbunden (Netz)
Kein Rechnen mit symbolisch kodierten Nachrichten
Wissen in der Struktur/Verbindungen repräsentiert

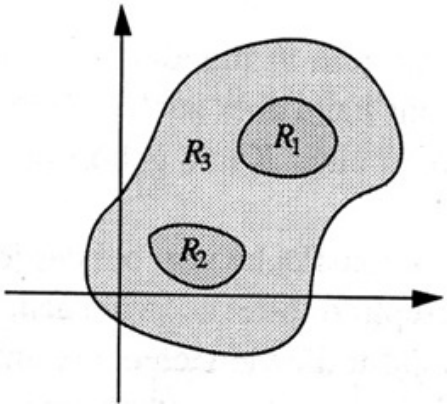
Gemeinsame Eigenschaften:

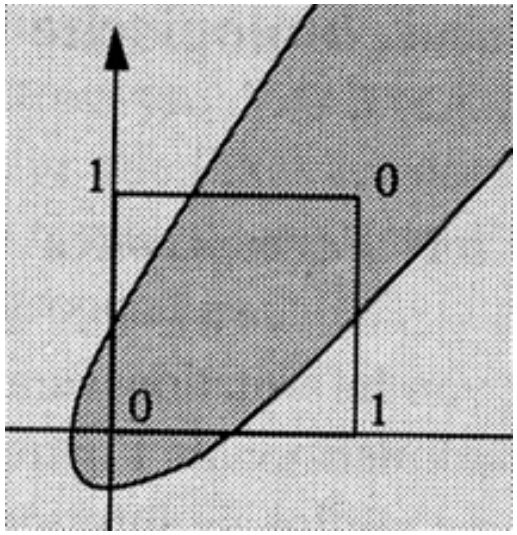
- Parallele Informationsverarbeitung
- Lernfähigkeit
- Generalisierungsfähigkeit
- Fehlertoleranz

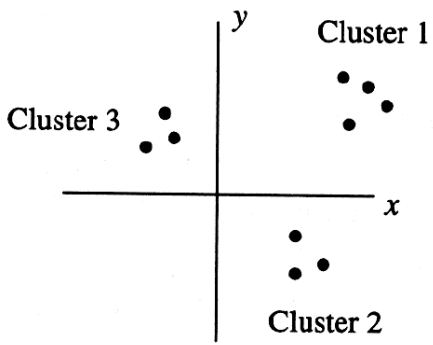
Motivation - Problemstellungen

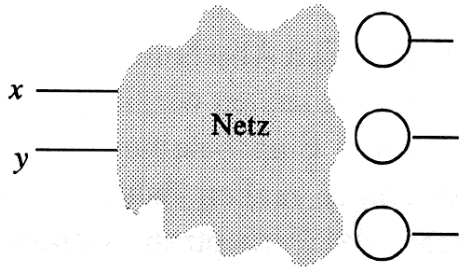
Entscheidungsregionen (Raumaufteilung), Clustering, Regression, Sequenzen, ...











Typische Einsatzfelder künstlicher Neuronaler Netze

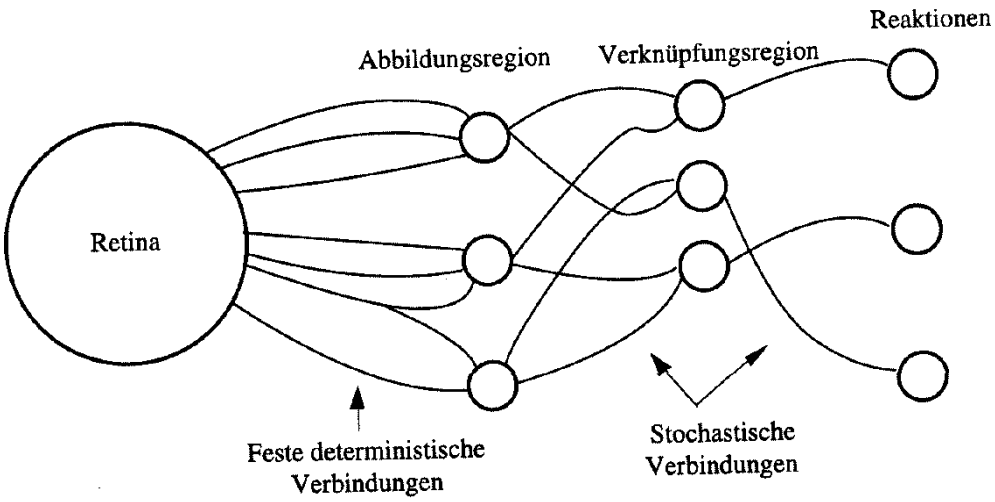
- Klassifikation und Mustererkennung
  - Diagnose, Spracherkennung, Schrifterkennung
- Funktionsapproximierung / Regression
  - Kontinuierliche Abbildung
  - Steuerung
  - Vorhersage
- Mustervervollständigung
  - Bilderkennung
  - Kodierung
- etc...

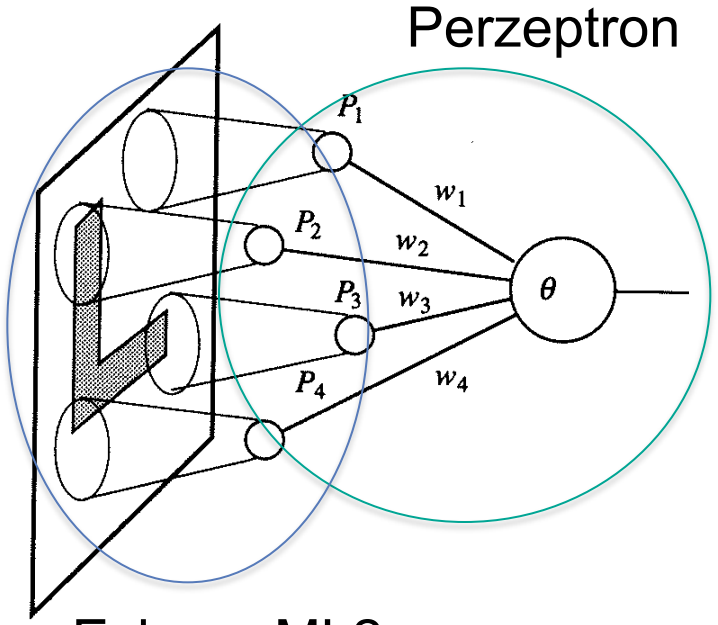
Frühe Realisierung: Perzeptron

- Grundidee
- Aufbau
- Lernverfahren
- Geometrische Interpretation

Perzeptron [Rosenblatt 1960]

Grundidee:
Anlehnung an das Funktionsprinzip der natürlichen Wahrnehmung – Reaktion im Tierbereich





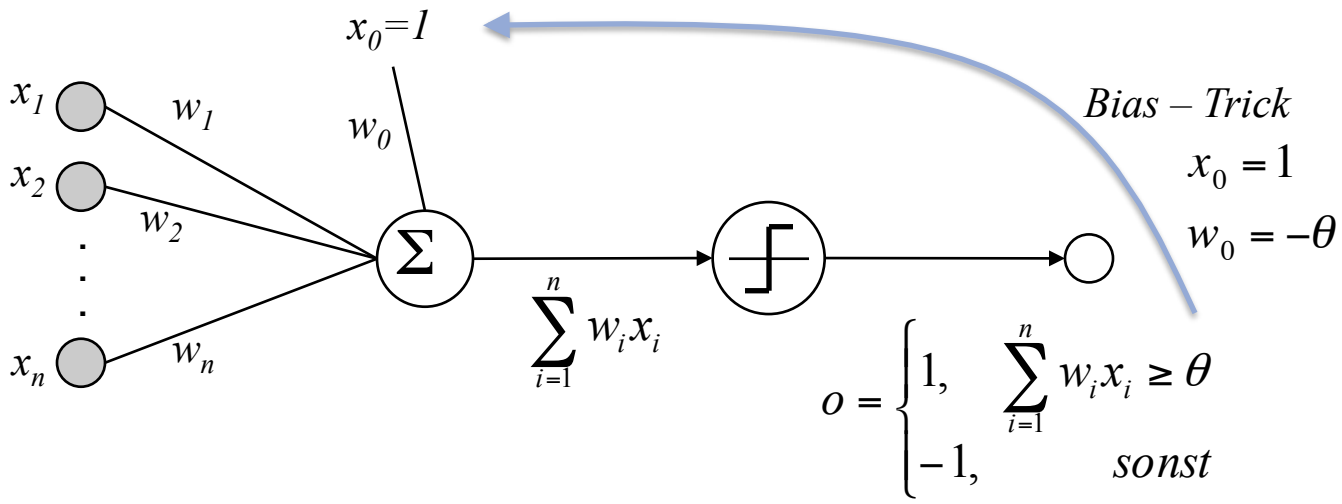
Exkurs: ML2
Deep Learning

Perzeptron [Rosenblatt 1960]

Aufbau eines Perzeptrons:

x – Eingabevektor

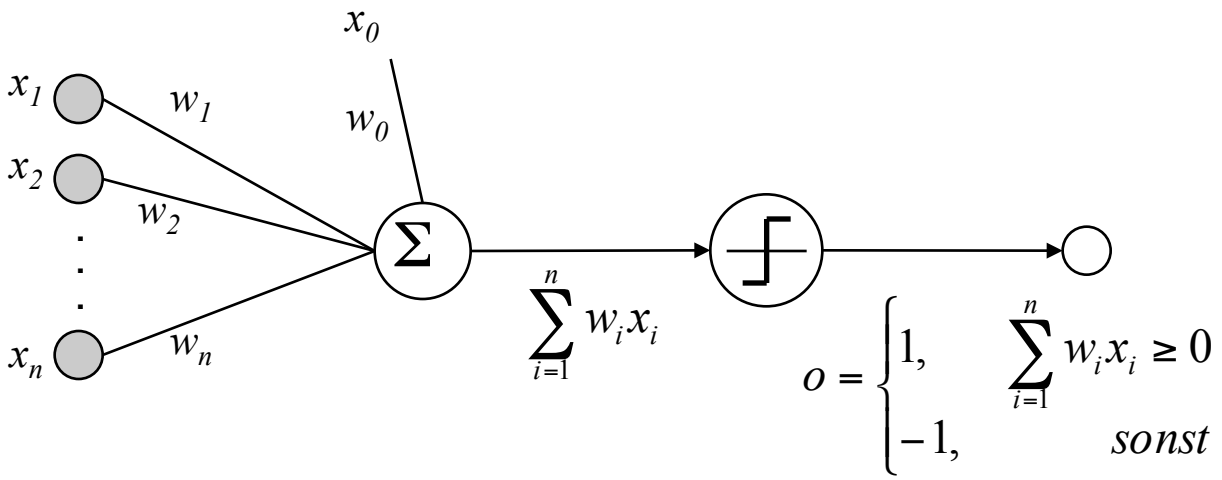
t – Target (Soll-Ausgabe)



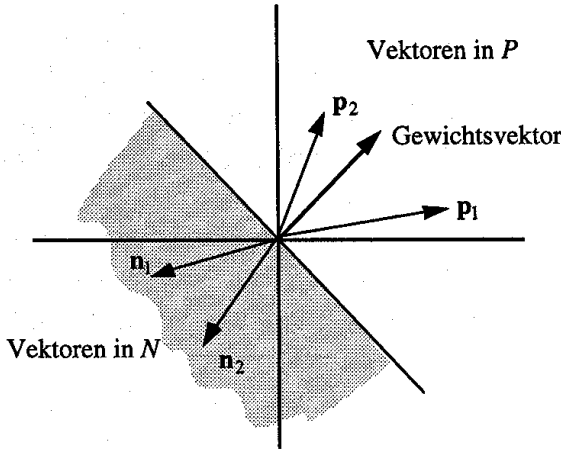
w – Gewichtsvektor

o – Output (Ist-Ausgabe)

Perzeptron: Geometrische Interpretation

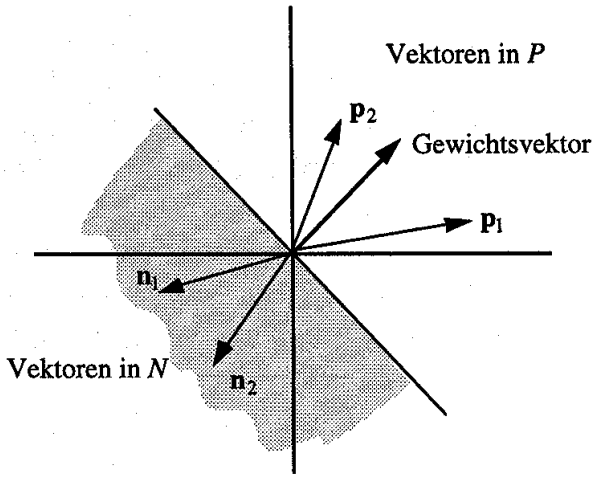


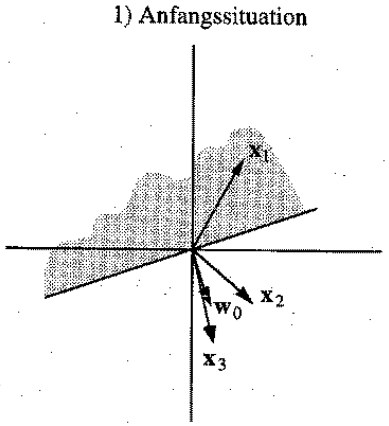
„Positive und Negative“ Daten (P,N)
Erweiterung der Dimension durch x_0
Trennhyperebene (R^2: Gerade)
Gewichte definieren diese Ebene (Normale)
Entscheidung, gewichtete Summe → Skalarprodukt

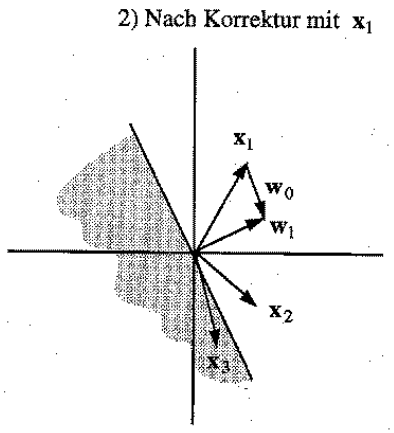


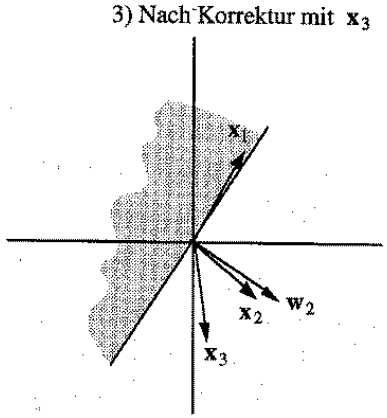
Lernen = Anpassen der Gewichte
➔ Gesucht wird die beste Trennhyperebene

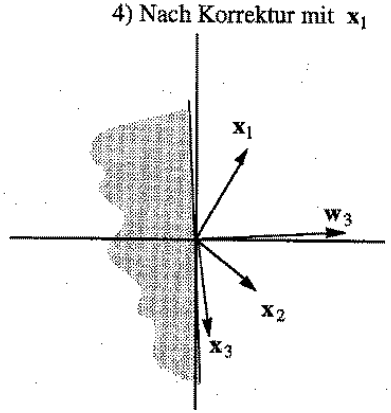
Lernen - Geometrische Interpretation











Hilfsmenge

$$N' = \{x' \mid x' = -x, \forall x \in N\}$$

Neues Lernproblem

$$xw > 0 \quad , \forall x \in N' \cup P$$

➔ Im Beispiel: alle x_i aus P

Perzeptron – Lernalgorithmus

Start: Gegeben Lerndatenmenge $P \cup N$

Der Gewichtsvektor $w(0)$ wird zufällig generiert.

Setze $t:=0$.

Testen: Ein Punkt x in $P \cup N$ wird zufällig gewählt.

Falls $x \in P$ und $w(t) \cdot x > 0$ gehe zu *Testen*

Falls $x \in P$ und $w(t) \cdot x \leq 0$ gehe zu *Addieren*

Falls $x \in N$ und $w(t) \cdot x < 0$ gehe zu *Testen*

Falls $x \in N$ und $w(t) \cdot x \geq 0$ gehe zu *Subtrahieren*

Addieren: Setze $w(t+1) = w(t)+x$.

Setze $t:= t+1$. Gehe zu *Testen.*

Subtrahieren: Setze $w(t+1) = w(t)-x$.

Setze $t:=t+1$. Gehe zu *Testen*.

Multi Layer Feedforward Neural Network

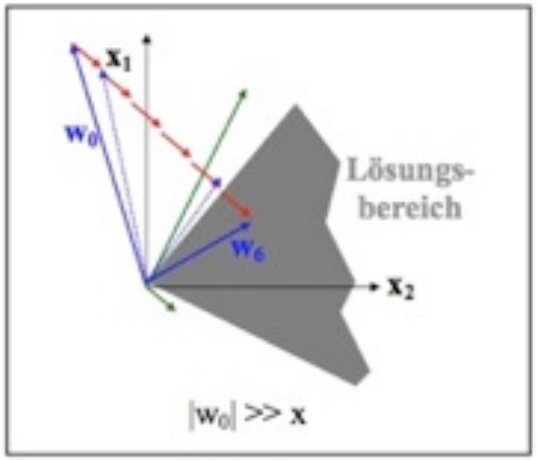
- Motivation
  - Grenzen des Perzeptrons
- Aufbau des MLNN
  - MLNN - Neuronen
  - Lernverfahren:
    - Backpropagation (allgemeine Deltaregel)
- Aufbau des RBF Netzes
  - Neuronen
  - Lernen
- Probleme / Optimierungen
  - Gradientenabstieg (z.B. RPROP)
  - Konstruktive Verfahren (DDA, Cascade Correlation)

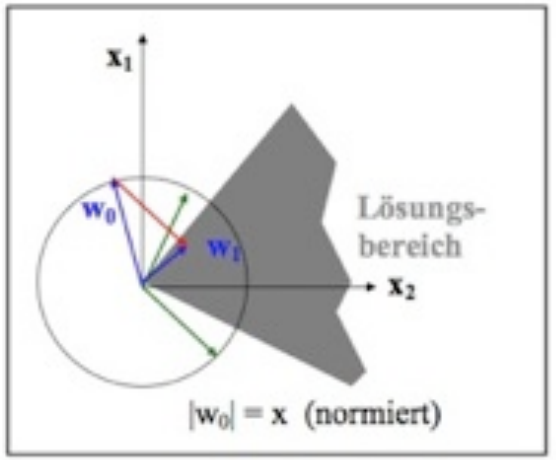
Perzeptron Lernalgorithmus

- $|w| >> |x|$

  Sehr langsame Anpassung

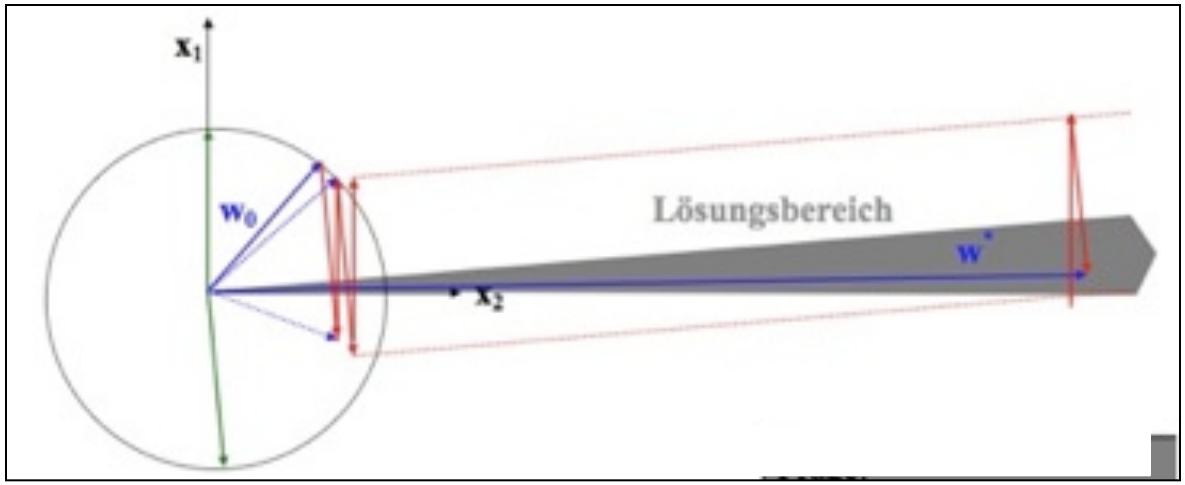
  ➔ Normierung





- Worst case:
  Fast anti-parallele Vektoren

  ➔ Gradientenabstieg
  Delta-Regel
  (idealerweise mit Optimierung)

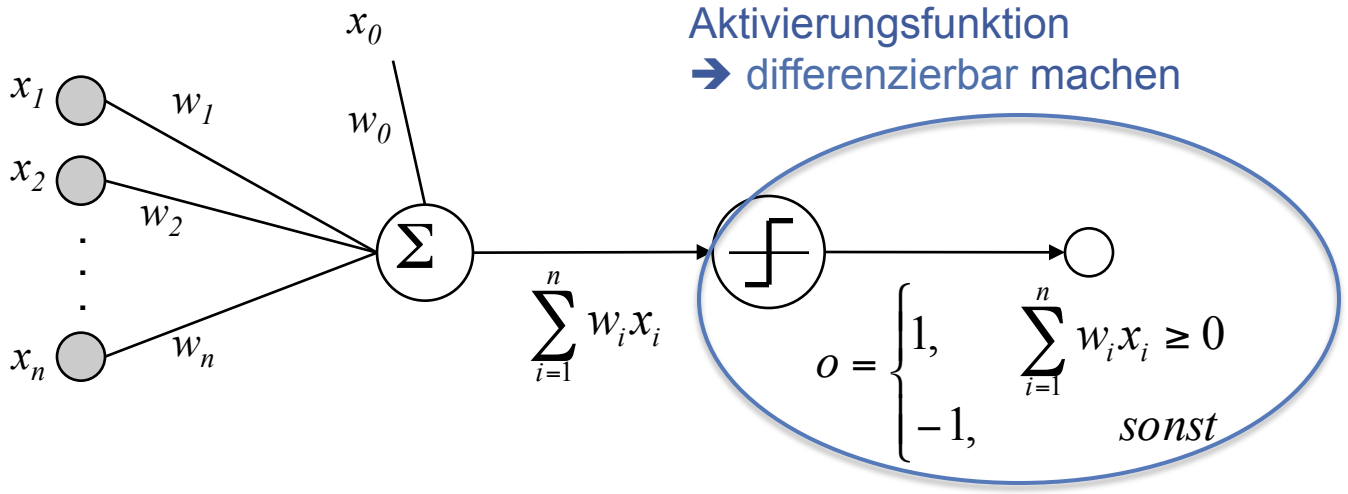


Perzeptron [Rosenblatt 1960]

Aufbau eines Perzeptrons:

x – Eingabevektor

t – Target (Soll-Ausgabe)



w – Gewichtsvektor

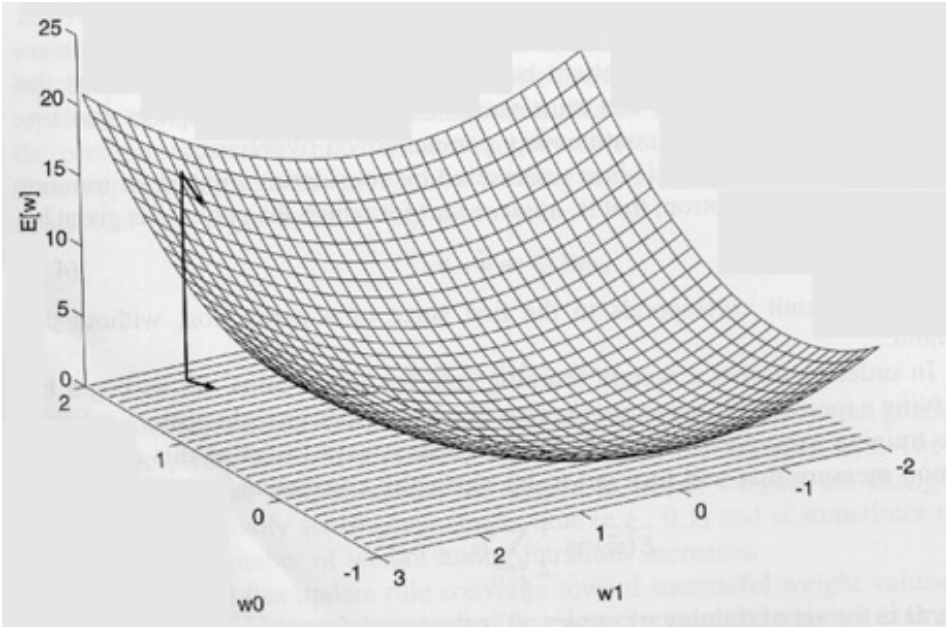
o – Output (Ist-Ausgabe)

Gradientenabstieg - Fehlerfunktion

Fehlerfunktion

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

D – *Lerndaten*



Lernen: Minimieren von E

Gradient

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

Anpassung des Gewichtsvektors

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

Lernrate η

oder

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

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

Gradientenabstieg - Deltaregel

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

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

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

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

Aktivierungsfunktion muss differenzierbar sein !!
z.B.

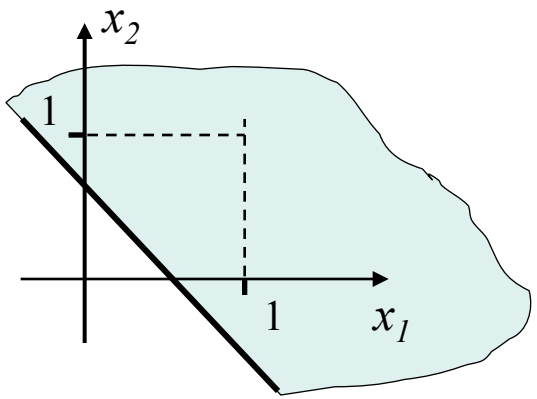
$$o(\vec{x}) = \sum \vec{w} \cdot \vec{x}$$

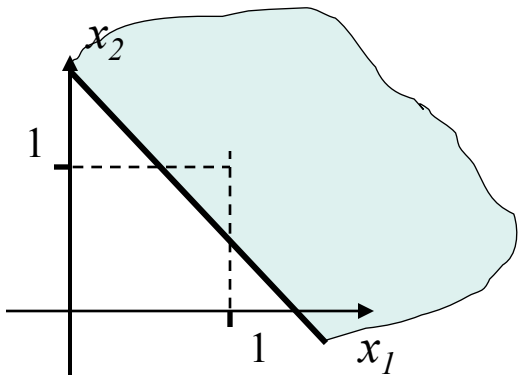
$$\frac{\partial E}{\partial w_i} \cong \sum_{d \in D} \left(t_d - o_d\right)\left(-x_{id}\right)$$

$$\Delta w_i = \eta \sum_{d \in D} \left(t_d - o_d\right) x_{id}$$

Perzeptron Kapazität

Bsp. Logik:





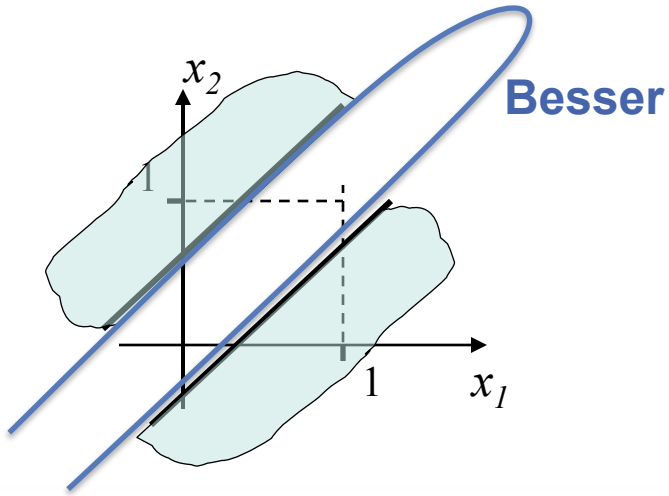
x_1 OR x_2: $0.5x_1 + 0.5\,x_2 - 0.3 > 0$

x_1 AND x_2 : $0.5x_1 + 0.5\,x_2 - 0.8 > 0$

➔ Durch Kombination von Perzeptronen sind viele Funktionen möglich

XOR: ???

NICHT MÖGLICH
(mit einem Perzeptron)



Kernel Methoden?

- Einfacher Algorithmus kann als Kernel Methode realisiert werden

- Gegeben Vektoren $\vec{x}, \vec{y}$ und eine Transformation $\phi(\vec{x}), \phi(\vec{y})$ in einen anderen Raum
- Dann ist ein Kernel:

$$K(\vec{x}, \vec{y}) = \left\langle \phi(\vec{x}), \phi(\vec{y}) \right\rangle$$

- Damit lassen sich oft mit einfachen Methoden im transformierten Raum komplexe Probleme im Ursprungsraum lösen

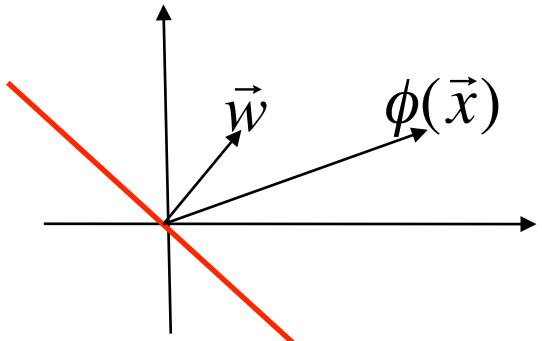
Kernel Methoden?

- Einfacher Algorithmus kann als Kernel Methode realisiert werden

Perzeptron Algorithmus

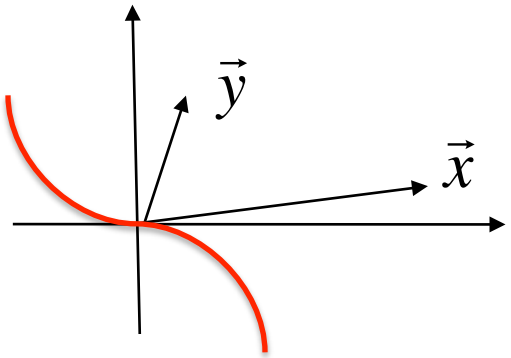
- Erinnerung: realisiert Trennung an Hyperebene

$$h(\vec{x}) = sign\langle \vec{w}, \phi(\vec{x}) \rangle$$



Transformierter Raum !!!

- Lineare Trennung im transformierten Raum führt zu komplexer Trennung im Ursprungsraum



Ursprungsraum !!!

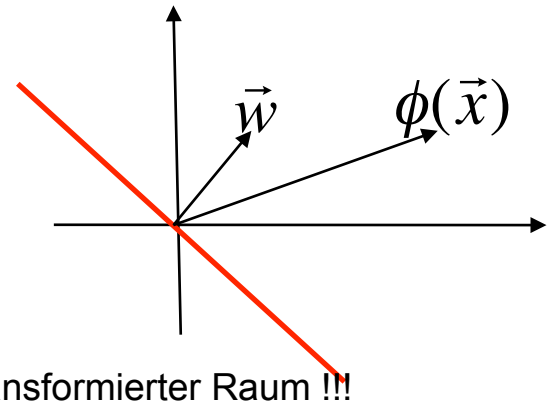
Kernel Methoden?

- Einfacher Algorithmus kann als Kernel Methode realisiert werden

Perzeptron Algorithmus

- Erinnerung: realisiert Trennung an Hyperebene

$$h(\vec{x}) = sign\langle \vec{w}, \phi(\vec{x}) \rangle$$



Transformierter Raum !!!

- Lernen (im transformierten Raum)
  - falsch klassif. Beispiele $(\vec{x}_i, y_i)$ ➔ update :

$$\vec{w}_{t+1} = \vec{w}_t + y_i \phi(\vec{x}_i) \Rightarrow \vec{w}_t = \sum_{i=1}^{l} \alpha_i y_i \phi(\vec{x}_i)$$

Kernel Perceptron

Gegeben eine Kernel - Funktion K ergibt sich der einfache Algorithmus: (ohne Transformation ϕ explizit zu kennen)

$$\vec{\alpha} = 0, i = 0$$

$$repeat$$

$$i = i + 1$$

$$if \;\; sign\langle \vec{w}, \phi(\vec{x}_i) \rangle = sign(\sum_{j=1}^{l} \alpha_j y_j K(\vec{x}_j, \vec{x}_i)) \neq y_i$$

$$\alpha_i = \alpha_i + 1$$

$$until \;\; finished$$

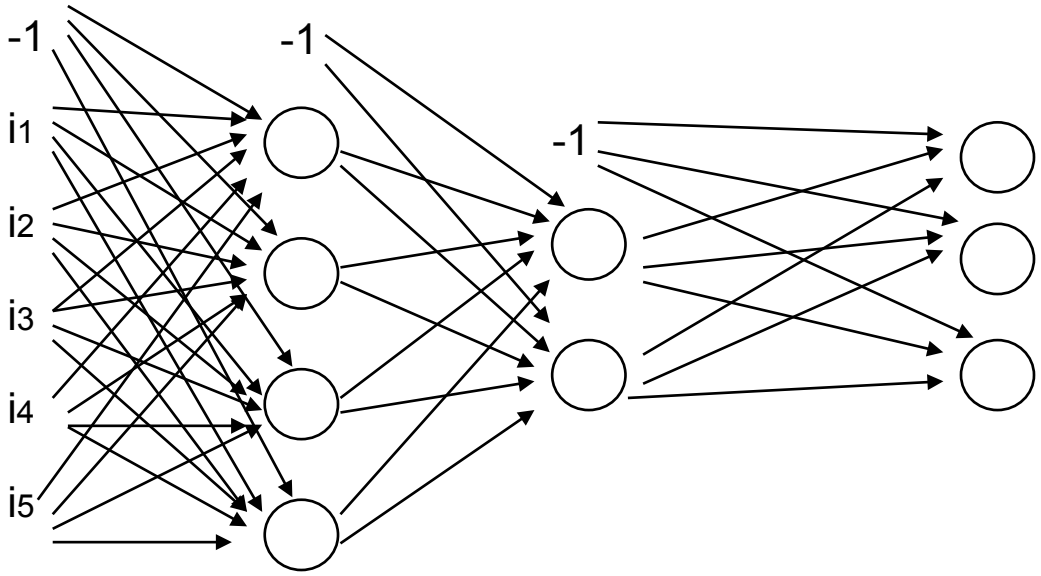
$$f(\vec{x}) = \sum_{j=1}^{l} \alpha_j y_j K(\vec{x}_j, \vec{x})$$

Multi Layer Feedforward Neural Network

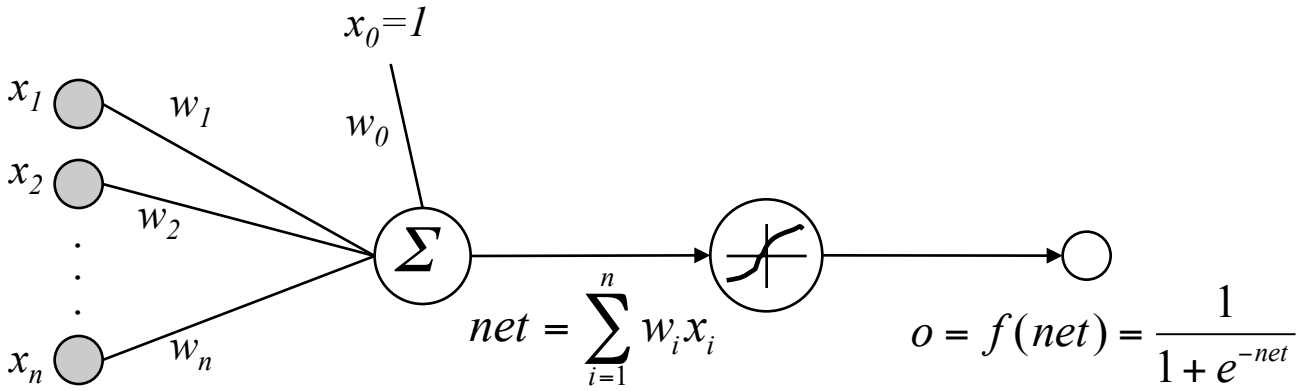
- Motivation
  - Grenzen des Perzeptrons
- Aufbau des MLNN
  - MLNN - Neuronen
  - Lernverfahren:
    - Backpropagation (allgemeine Deltaregel)
- Aufbau des RBF Netzes
  - Neuronen
  - Lernen
- Probleme / Optimierungen
  - Gradientenabstieg (z.B. RPROP)
  - Konstruktive Verfahren (DDA, Cascade Correlation)

Multi Layer Neural Network

- Netzaufbau: mehrere versteckte (innere) Schichten
- Lernverfahren: Backpropagation - Algorithmus
  (allgemeine Delta-Regel)
  [Rumelhart86, Werbos74]
- Neuronenaufbau: nichtlineare Aktivierungsfunktion



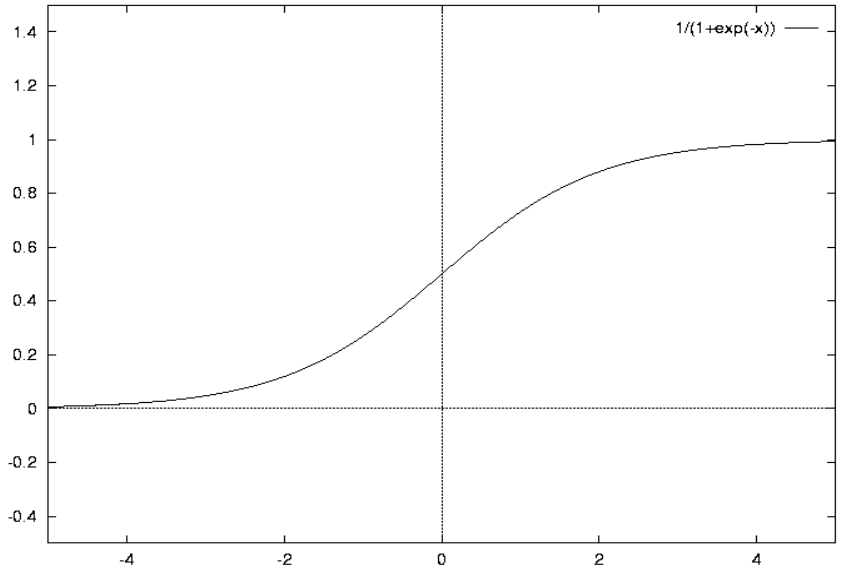
Aufbau der Neuronen

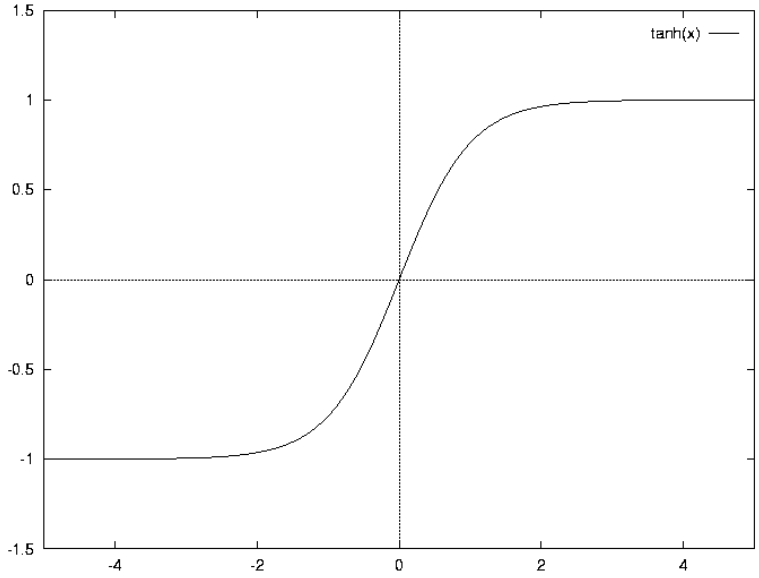


- x_{ij} = i-te Eingabe des Neurons j
- w_{ij} = das Gewicht zwischen Neuron i und Neuron j
- net_j = $\sum_i w_{ij} x_{ij}$ Propagierungsfunktion
- o_j = Ausgabe des Neurons j
- t_j = Zielausgabe (target) des Ausgabeneurons j
- $f(x)$ = Aktivierungsfunktion

- *outputs* = Menge der Ausgabeneuronen
- *Downstream* (j) = direkte Nachfolger des Neurons j

Nichtlineare Aktivierungsfunktionen





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

$$\frac{\partial f}{\partial x} = f(x)\,(1 - f(x))$$

$$f(x) = \tanh(x)$$

$$\frac{\partial f}{\partial x} = (1 + f(x))\,(1 - f(x))$$

Backpropagation Algorithmus I

Vorgaben:

Menge T von Trainingsbeispielen

Eingabevektor / Ausgabevektor (input / output)

Lernrate η

Netztopologie

Anzahl und Ausmaße der Zwischenschichten

Schichten sind vollständig vorwärts gerichtet verbunden

Lernziel:

Finden einer Gewichtsbelegung $\boldsymbol{W}$, die T korrekt wiedergibt

Vorgehen:

Gradientenabstieg ➔ allgemeine Deltaregel

Backpropagation Algorithmus II

Initialisieren der Gewichte mit kleinen zufälligen Werten

Wiederhole

- Auswahl eines Beispielmusters d
- Bestimmen der Netzausgabe
- Bestimmen des Ausgabefehlers (bzgl. Sollausgabe)
- Sukzessives Rückpropagieren des Fehlers auf die einzelnen Neuronen

$$\delta_j = \begin{cases} o_j(1-o_j) \sum_{k \in Downstream(j)} \delta_k w_{jk} & j \notin output \\ o_j(1-o_j)\ (t_j - o_j) & j \in output \end{cases}$$

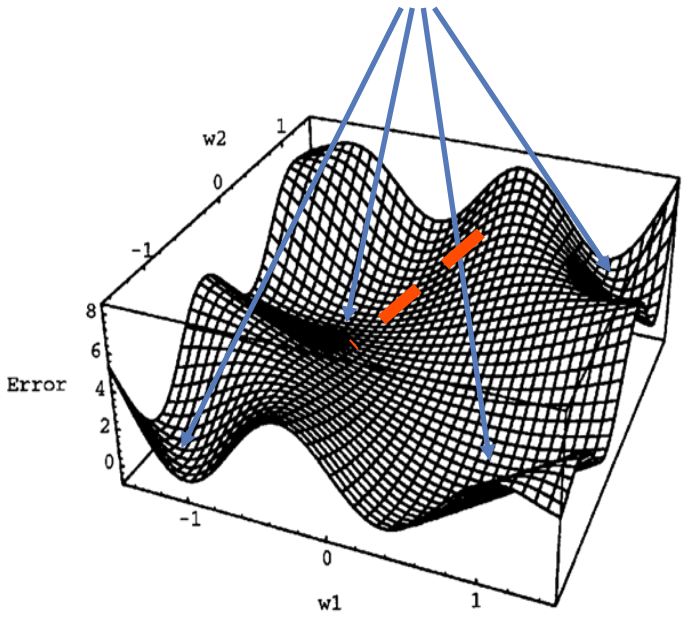
- Anpassung der Gewichtsbelegungen um

$$\Delta w_{ij} = \eta \delta_j x_{ij}$$

solange ein gewähltes Abbruchkriterien nicht erfüllt ist

Gradientenabstieg - Allgemeine Deltaregel

Minima



Fehlerfunktion

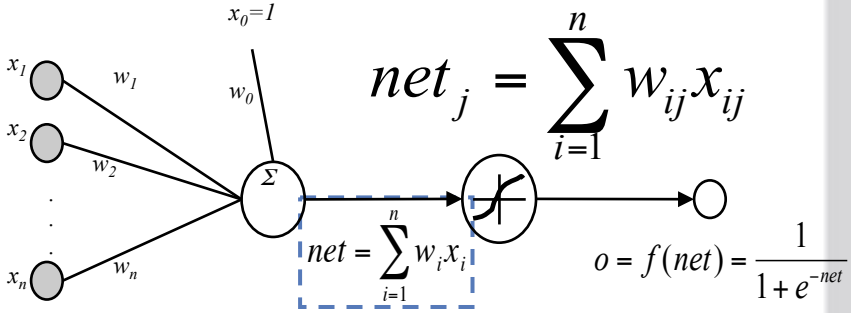
$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

Gewichtsänderung nach dem Gradienten

$$\Delta w_{ij} = -\eta \frac{\partial E_d}{\partial w_{ij}}$$

Kettenregel

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j} x_{ij}$$



Allgemeine Deltaregel

Ableitung von $\frac{\partial E}{\partial net_j}$ für Ausgabeschicht

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = o_j(1 - o_j)$$

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

$$\frac{\partial f}{\partial x} = f(x)(1 - f(x))$$

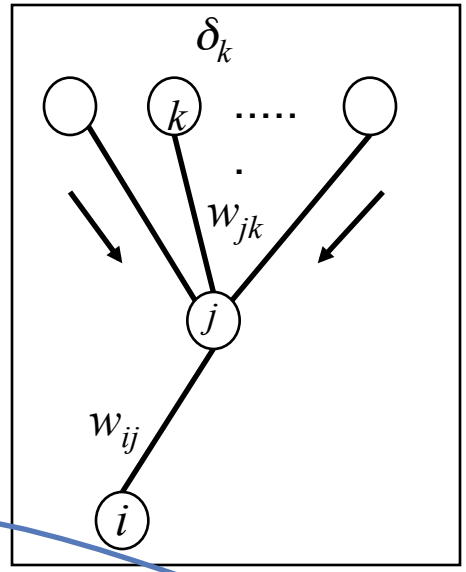
Allgemeine Deltaregel

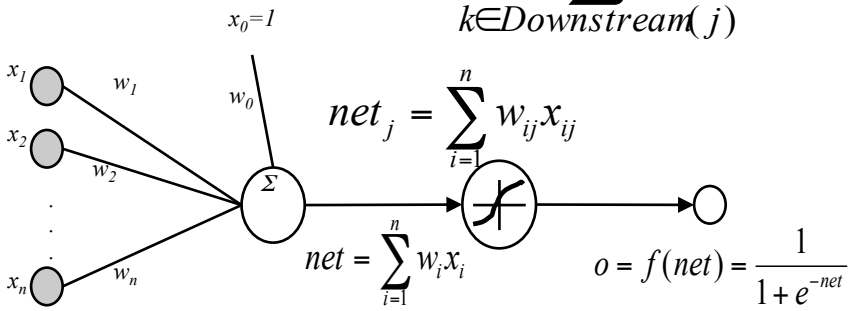
Ableitung von $\frac{\partial E}{\partial net_j}$ für Zwischenschichten

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$\delta_j = -\frac{\partial E_d}{\partial net_j} = -\sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= -\sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -\sum_{k \in Downstream(j)} -\delta_k w_{jk} \frac{\partial o_j}{\partial net_j}$$





$$= \sum_{k \in Downstream(j)} \delta_k w_{jk} o_j (1 - o_j)$$

$$= o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{jk}$$

Allgemeine Deltaregel

$$\Delta w_{ij} = -\eta \frac{\partial E_d}{\partial w_{ij}}$$

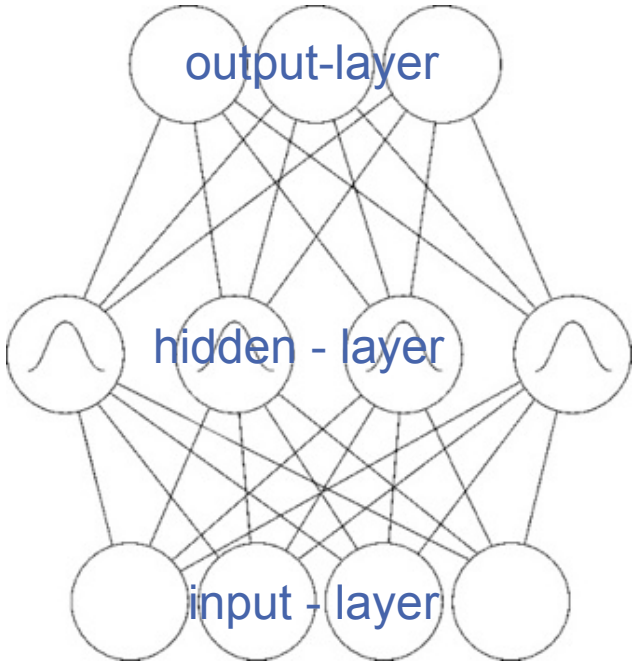
$$\Delta w_{ij} = \eta \delta_j x_{ij}$$

$$\delta_j = \begin{cases} o_j(1-o_j) \sum_{k \in Downstream(j)} \delta_k w_{jk} & j \notin output \\ o_j(1-o_j)\ (t_j - o_j) & j \in output \end{cases}$$

Multi Layer Feedforward Neural Network

- Motivation
  - Grenzen des Perzeptrons
- Aufbau des MLNN
  - MLNN - Neuronen
  - Lernverfahren:
    - Backpropagation (allgemeine Deltaregel)
- Aufbau des RBF Netzes
  - Neuronen
  - Lernen
- Probleme / Optimierungen
  - Gradientenabstieg (z.B. RPROP)
  - Konstruktive Verfahren (DDA, Cascade Correlation)

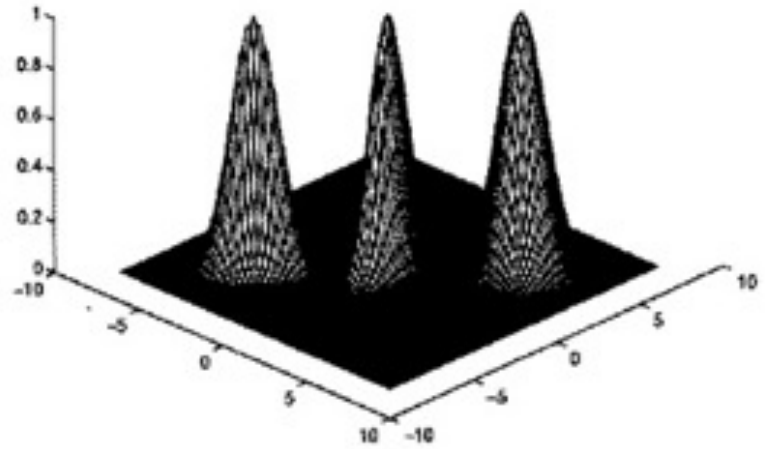
Radial Basis Function - Netz



Topologie und Aufbau:

- vorwärtsgerichtetes Netz
- 3-schichtig mit einer hidden - Schicht

- Neuronen des hidden layer :
  lokale rezeptive Felder

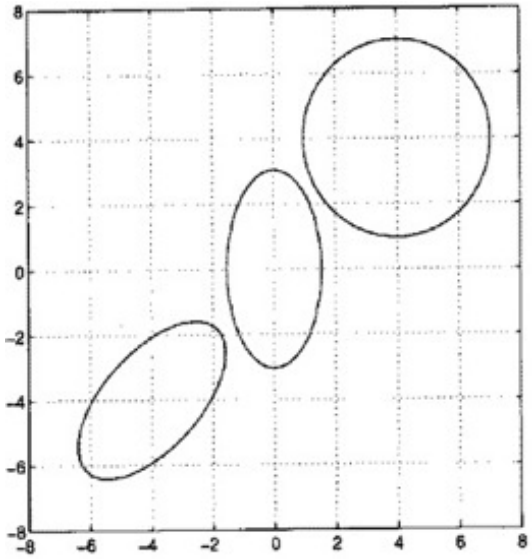


Radial Basis Function - Netz

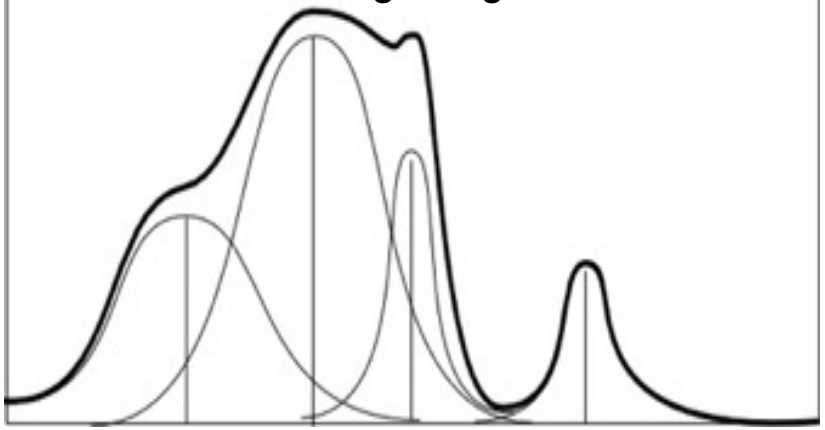
Gauß - Funktionen

$\mu - Zentrum$ (Mittelwert)

$\sigma - \mathrm{Re}ichweite$ (Std.abweichung)



Gewichtete Überlagerung: 4 Neuronen



- Neuronen des output layer:
  gewichtete Summe

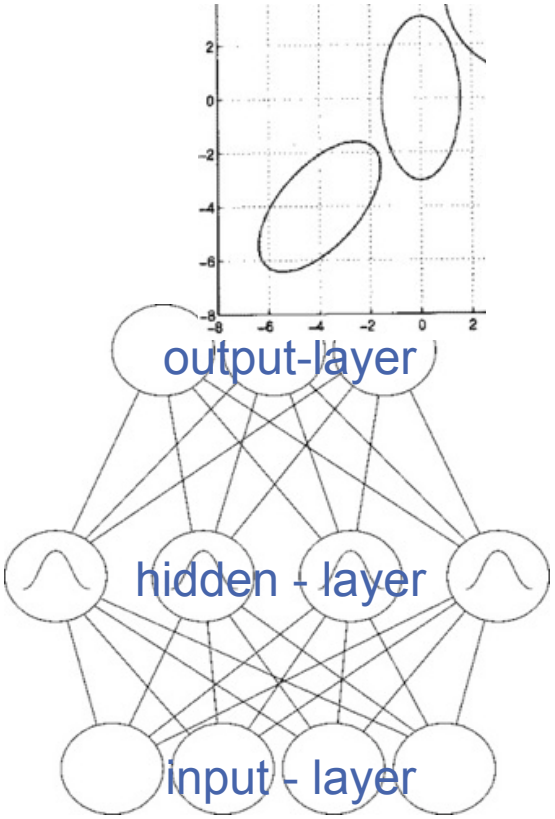
$$o_j = \sum_{i \in hidden} o_i w_{ij}, \quad j \in output$$

Optimierungen RBF - Netz

Erweiterungen des RBF- Netzes:

- Neuronen des hidden layer
  gewichtete Eingabedimensionen ➔ unsymmetrische Felder

$$o_i(\vec{x}) = e^{-\frac{\sum_k \xi_k (x_k - \mu_{ik})^2}{2\sigma_i^2}}$$



- Neuronen des output layer
  normalisierte RBFs

$$o_j = \frac{\sum_i o_i w_{ij}}{\sum_i o_i}$$

Lernen mit RBF - Netzen

BackProp: Adaption nach Gradientenabstieg

Zentren der Felder

$$\frac{\partial E}{\partial \mu_{ji}} = \sum_n \sum_k \left(o_k(x^n) - t_k^n\right) w_{kj} \exp\left(-\frac{\left\|x^n - \mu_j\right\|^2}{2\sigma_j^2}\right) \frac{\left\|x^n - \mu_{ji}\right\|}{\sigma_j^2}$$

Reichweite der Felder

$$\frac{\partial E}{\partial \sigma_j} = \sum_n \sum_k \left(o_k(x^n) - t_k^n\right) w_{kj} \exp\left(-\frac{\left\|x^n - \mu_j\right\|^2}{2\sigma_j^2}\right) \frac{\left\|x^n - \mu_j\right\|^2}{\sigma_j^3}$$

Gewichte der Ausgabeschicht

.....

Erweiterung des BackProp für RBF - Netzen

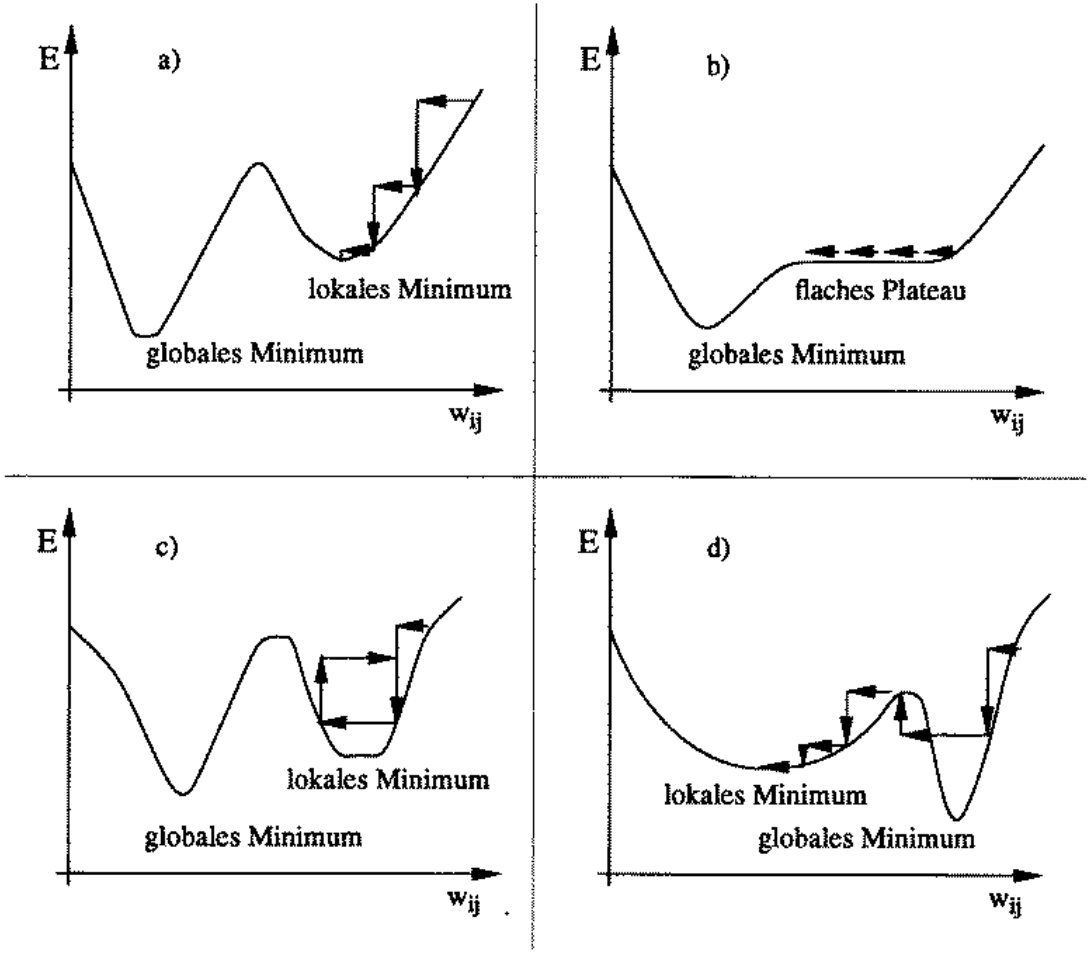
- BackProp Nachteile:
  - nichtlineare Funktionen ➔ hoher Rechenaufwand
  - lokale Minima
  - Reichweite kann sehr groß werden ➔ keine lokalen Felder

- Hybride Lernverfahren (als Idee)
  - Finden lokaler Felder: Zentrum und/oder Reichweite – unüberwacht, d.h. z.B. ohne die Sollausgabe zu betrachten (oder durch andere Verfahren)
  - Anpassen der übrigen Gewichte: Ausgabeschicht - durch Gradientenabstieg

Multi Layer Feedforward Neural Network

- Motivation
  - Grenzen des Perzeptrons
- Aufbau des MLNN
  - MLNN - Neuronen
  - Lernverfahren:
    - Backpropagation (allgemeine Deltaregel)
- Aufbau eines RBF Netze
  - Neuronen
  - Lernen
- Probleme / Optimierungen
  - Empirischer Fehler - Gradientenabstieg (z.B. RPROP)
  - Kapazität - Konstruktive Verfahren (DDA, Cascade Correlation)

Fehlerflächen



Lernen abhängig von:

- Steigung der Fehlerfläche
- Vorkommen lokaler Minima
- Ausprägung der lok. Min.
- Lernrate

Beispiele in Matlab

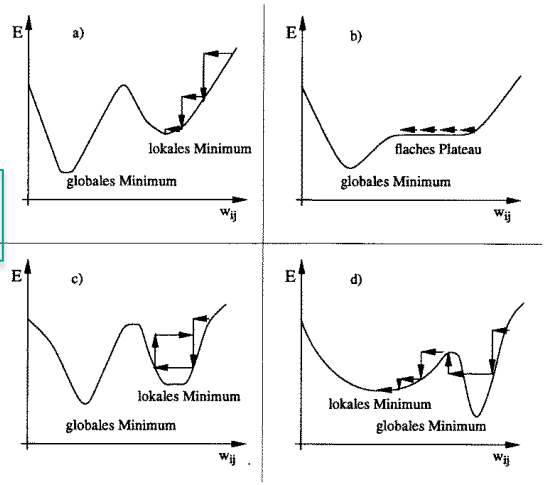
Linear fit	→	Fehlerkurve, Lernen
Too large	→	Lernrate zu groß, keine Konvergenz
Undetermined Problem	→	Großes Tal, kein Minimum

Optimierungen des Gradientenabstiegs

Momentum Term

$$\Delta w_{ij}(t) = \eta\,\delta_j(t)\,x_{ij}(t) + \alpha\,\Delta w_{ij}(t-1)$$

(α im Intervall [0.2,0.9])



Normierung der Schrittweite

$$\Delta w_{ij} = \eta\,sign(\delta_j)\,x_{ij}$$ (Manhattan-Training)

Lernratenanpassung

Wie ist ideale Schrittweite in hochdimensionaler Fehlerfläche ?

→ individuelle Lernrate

Modifikation der Lernrate mittels Gradientenvergleichs

$$\eta(t+1) > \eta(t) \quad \textit{wenn} \quad sign\left(\frac{\partial E}{\partial w}(t+1)\right) = sign\left(\frac{\partial E}{\partial w}(t)\right)$$

$$\eta(t+1) < \eta(t) \quad \textit{sonst}$$

(RPROP)

RPROP (Resilient Propagation)

Fehlerfunktion

$$E = \frac{1}{2} \sum_{i \in outputs} (t_i - o_i)^2$$

Gewichtsänderung

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t), & wenn \ \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\Delta_{ij}(t), & wenn \ \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0, & sonst \end{cases}$$

RPROP

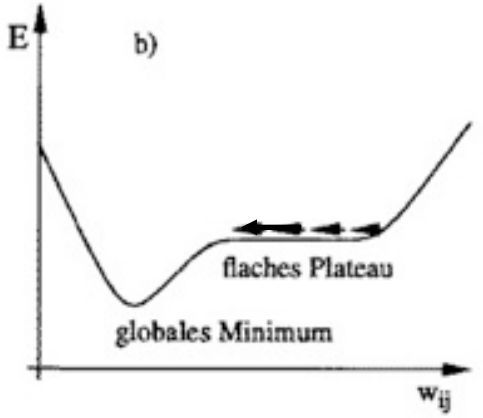
Lernrate abhängig vom Gradientenvergleich

$$\Delta_{ij}(t) = \begin{cases} \Delta_{ij}(t-1)\cdot\eta^{+}, & wenn \ \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \Delta_{ij}(t-1)\cdot\eta^{-}, & wenn \ \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1), & sonst \end{cases}$$

$$\eta^{+} > 1,\ \eta^{-} < 1$$

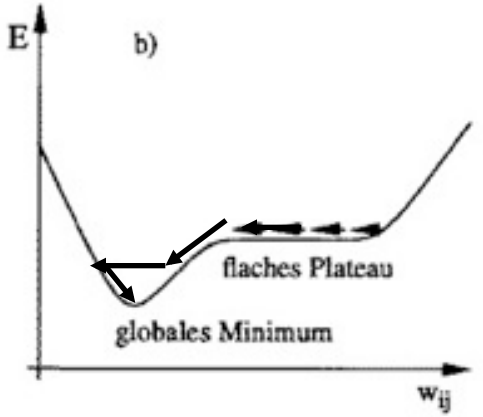
RPROP – Interpretation

Beschleunigen auf flachem Plateau



Langsam Anpassen im Minimum

➔ Schnelle Konvergenz



Multi Layer Feedforward Neural Network

- Motivation
  - Grenzen des Perzeptrons
- Aufbau des MLNN
  - MLNN - Neuronen
  - Lernverfahren:
    - Backpropagation (allgemeine Deltaregel)
- Aufbau eines RBF Netze
  - Neuronen
  - Lernen
- Probleme / Optimierungen
  - Empirischer Fehler - Gradientenabstieg (z.B. RPROP)
  - Kapazität - Konstruktive Verfahren (DDA, Cascade Correlation)

Topologieauswahl MLNN

Anzahl der (hidden) layer ←→ Zielfunktion

- 3 Layer (1 hidden Layer - sigmoid):
  - jede Boolsche Funktion abbildbar
  - jede kontinuierliche beschränkte Funktion
    [Cybenko 1989, Hornik et al. 1989]
- 4 Layer (2 hidden Layer -sigmoid)
  - beliebige Funktionen mit beliebiger Genauigkeit
    [Cybenko 1988]

➔ Schon eine geringe Tiefe ist ausreichend

Lernverhalten - Topologieauswahl

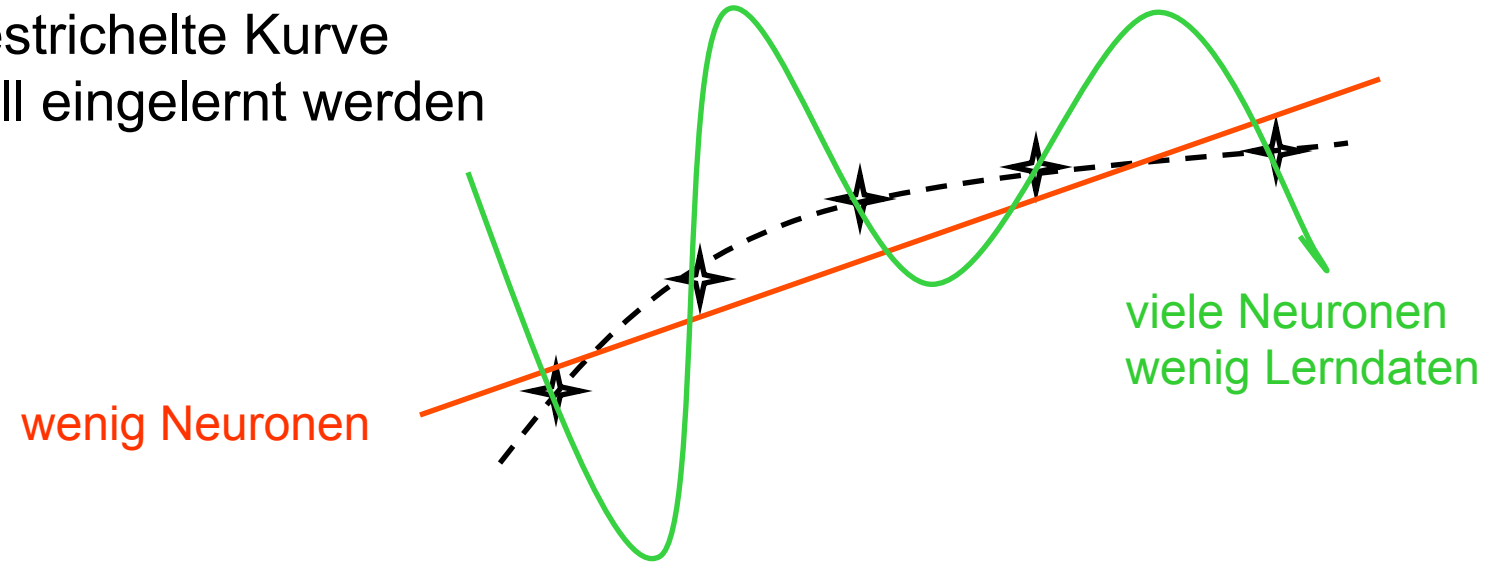
Anzahl der Neuronen pro Schicht im Bezug zu der Anzahl von (stochastisch unabhängigen) Lerndaten ist wichtig

- allgemeine Aussage zur Topologie nicht möglich

Topologie ←→ Kapazität (VC –Dimension)

Beispiel:

gestrichelte Kurve
soll eingelernt werden



Verbesserung der Generalisierung (~realer Fehler) von MLNN

Ziel: Gleichzeitiges Trainieren eines NN und Finden der optimalen Topologie

- Anpassung der Kapazität, 2 Ansätze :
  - großes Netzwerk wird verkleinert und so daran gehindert, die Daten auswendig zu lernen
  - kleines Netzwerk wird solange erweitert bis es die Daten lernen kann

- Methoden der Reduzierung
  - Weight Decay, Weight Elimination = Bestrafen von großen w durch Verwendung erweiterter Fehlerfunktion:

$$E(w) = E_D(w) + \frac{1}{2}\lambda \sum w^2,$$

  - Optimal Brain Damage
    - Lernen + anschließend Löschen von Verbindungen,
    - zB. w mit kleinen Beträge oder wenig Einfluss auf den Fehler E (Verwendung der zweiten Ableitung. Taylor-Entwicklung)

Verbesserung der Generalisierung (~realer Fehler) von MLNN

Ziel: Gleichzeitiges Trainieren eines NN und Finden der optimalen Topologie

- Anpassung der Kapazität, 2 Ansätze :
  - großes Netzwerk wird verkleinert und so daran gehindert, die Daten auswendig zu lernen
  - kleines Netzwerk wird solange erweitert bis es die Daten lernen kann

- Schrittweise Vergrößern (konstruktiv)
  - Cascade Correlation

  - Meiosis Netzwerke (Unsicherheit der Gewichte wird berücksichtigt: relative Standardabweichung = Standardabweichung/Mittelwert >1 ➔ Aufspalten)

Konstruktive Lernverfahren für MLNN

Beispiel: Cascade – Correlation – Verfahren für MLNN (Fahlman, Lebiere)

Algorithmus:

Initialisierung:

2- schichtiges Netz,

Abbruchkriterien: Fehlerschranke $E(w)$, # Neuronen ….

Trainieren – Anpassen aller Gewichte

while $E(w) >$ *Fehlerschranke* und *nicht weiteres Abbruchkriterium do*

Füge Neuron ein: (s.g. Kandidat-Neuron)

- hidden, vor die Ausgabeschicht
- vollständig verbunden,

(mit Eingabeschicht)

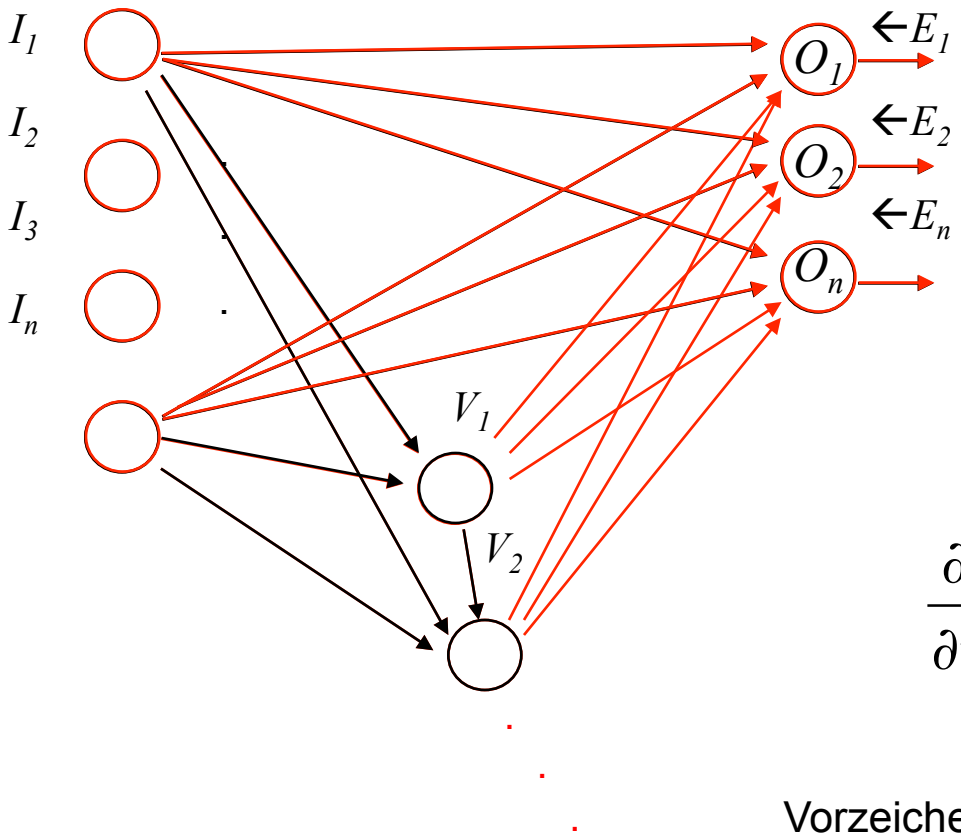
(z.B. Gewicht -1 zu vorher eingefügten Neuronen)

Trainiere neues Neuron (einmalig)

Trainiere Netz

done

Cascade Correlation I (Fahlman)



Anpassung der Kandidat-Neuronen (maximale Ausprägung für Fehler für alle Pattern p und alle output Neuronen o) durch Maximierung von:

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}) \right|$$

Ausgabe Kandidatneuron

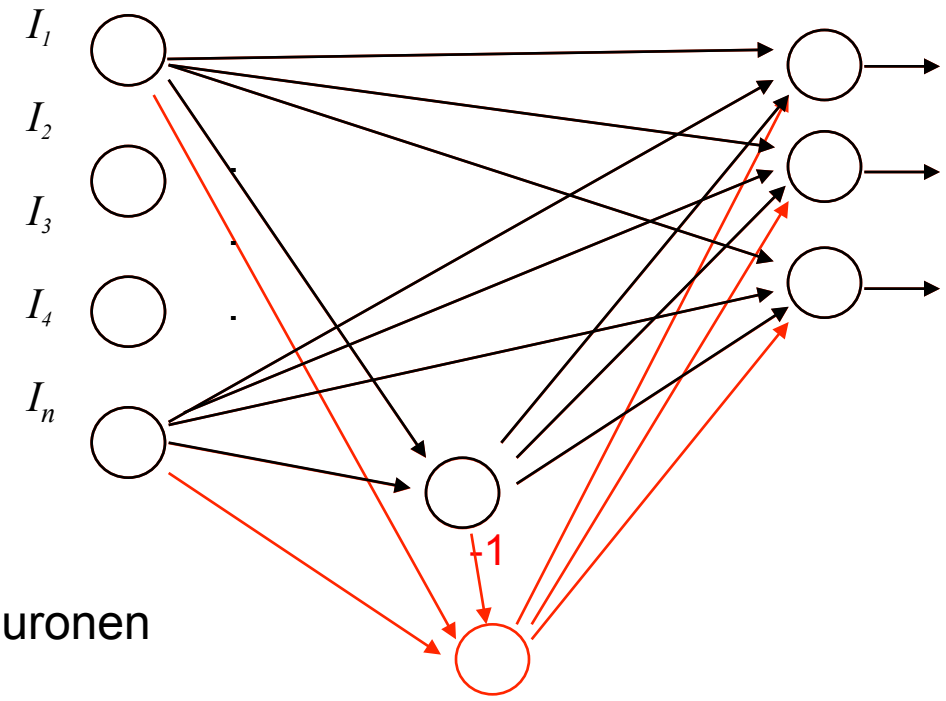
$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o (E_{p,o} - \bar{E}) f'_p I_{i,p},$$

Vorzeichen, Korrelation zw. Aktivierung Kandidatneuron und output o

Ableitung Sigmoid

Input Kandidat-neuron

Cascade Correlation II (Duda & Hart)



Vereinfachte Version:
Anpassung der neuen Neuronen

Kandidat-Neuronen wird mit -1 zu den anderen Neuronen der Zwischenschicht verbunden (maximal unterschiedliche Ausprägung)

Cascade Correlation

- Zu jedem Zeitpunkt wird „nur eine Ebene“ von Verbindungen trainiert
- Lernt sehr schnell
- Inkrementelles Training (d.h. bei neuen Daten vom letzten Stand aus **weiter** Lernen)
- Iterative Anpassung der Kapazität des Netzes

Dynamic Decay Adjustment- DDA

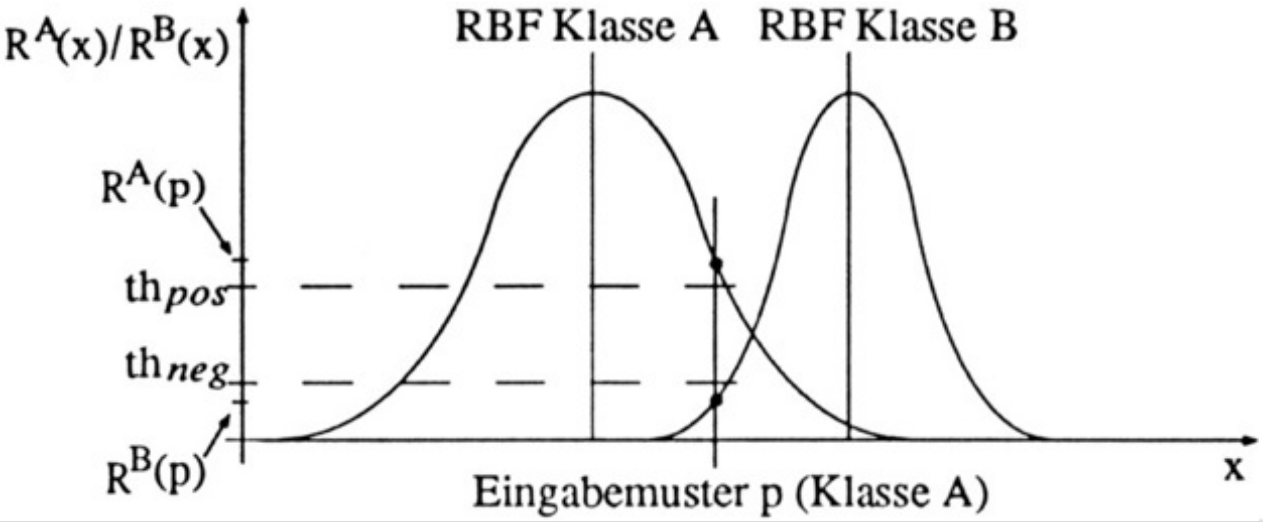
Konstruktiver Algorithmus

- Ausbilden einer RBF-Topologie für Klassifikation (Berthold94..
- Finden aller Parameter – überwacht .... Ricci06)
- Jedes Neuron ist einer Klasse zugeordnet

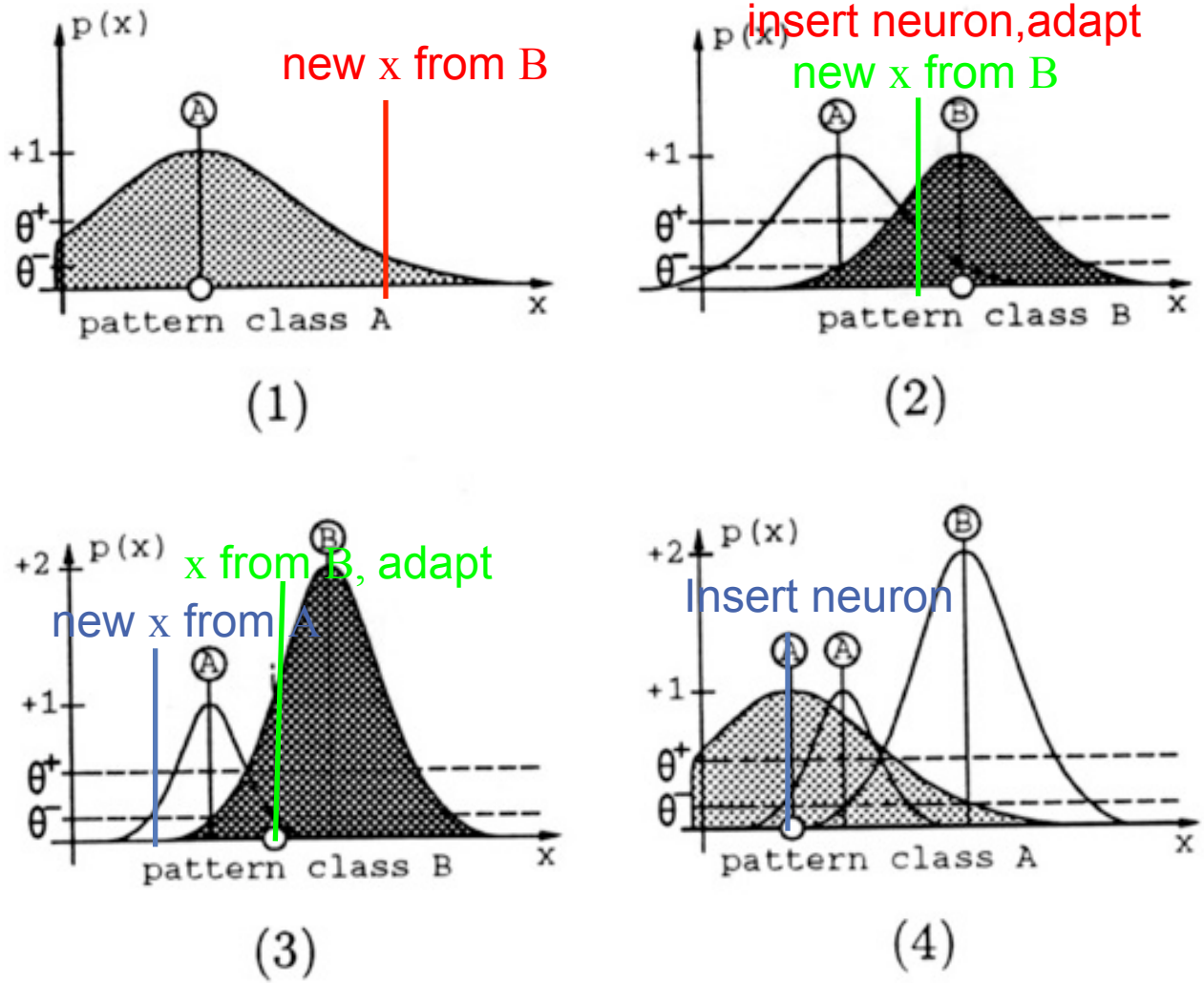
Lernen:

- Einfügen neuer Neuronen
- Getriggert über zwei Schwellwerte θ_{pos} und θ_{neg}

  ➔ Anpassen der Reichweiten, je nach Aktivität R der Neuronen



DDA-Algorithmus Beispiel



DDA - Algorithmus

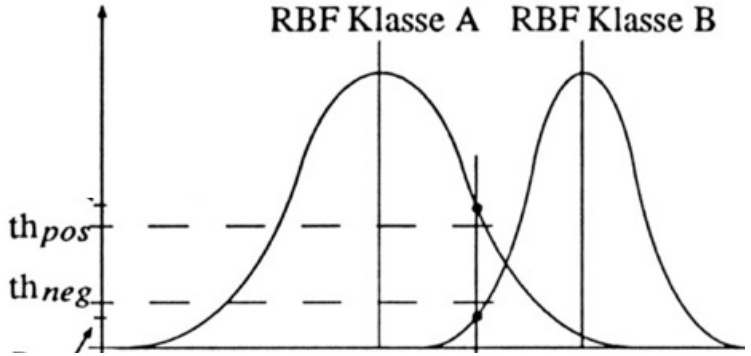
Bezeichnungen:

A,B Klassen; n_j^A Neuron aus A; j_A = Anzahl Neuronen Klasse A

$$\forall\ \vec{x} \in Lerndaten\ (oBdA\ \vec{x} \in A)\ \ do$$

if $\exists n_i^A :\ o_i^A(\vec{x}) \geq \theta_{pos}$

$$w_i^A + = 1.0$$



else erzeuge Prototypen (Neuron) $n_{j_A+1}^A$

$$\vec{\mu}_{j_A+1}^A = \vec{x}$$

$$\sigma_{j_A+1}^A = \max\left\{\sigma \mid \forall B \neq A, 1 \leq k \leq j_B\ \ gilt\ \ o_{j_A+1}^A\left(\vec{\mu}_k^B\right) < \theta_{neg}\right\}$$

$$w_{j_A+1}^A = 1.0$$

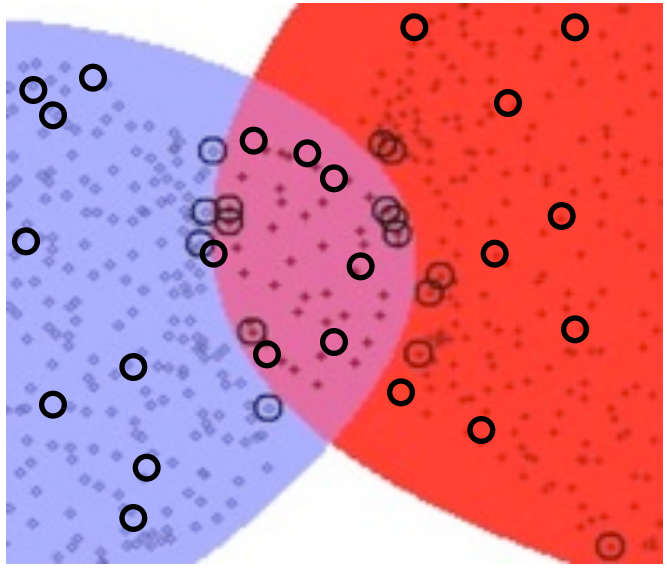
$$\forall B \neq A \wedge 1 \leq k \leq j_B : \sigma_k^B = \max\left\{\sigma \mid o_k^B(\vec{x}) < \theta_{neg}\right\}$$

Einsatz von MLNN

- Entwurf
  - Subsymbolische Repräsentation der Ein- und Ausgabe
  - Auswahl Trainingsdaten
  - Auswahl des Verfahren
  - Auswahl der Topologie
  - Parametereinstellung
- Auswahl des Lernverfahrens
  - z.B. Optimierungsmethode (Momentum,...., RPROP)
- Lernen
  - Initialisierung
  - Lernschritt (Gewichtsanpassung) Overfitting
  - Training & Verifikation (Test)

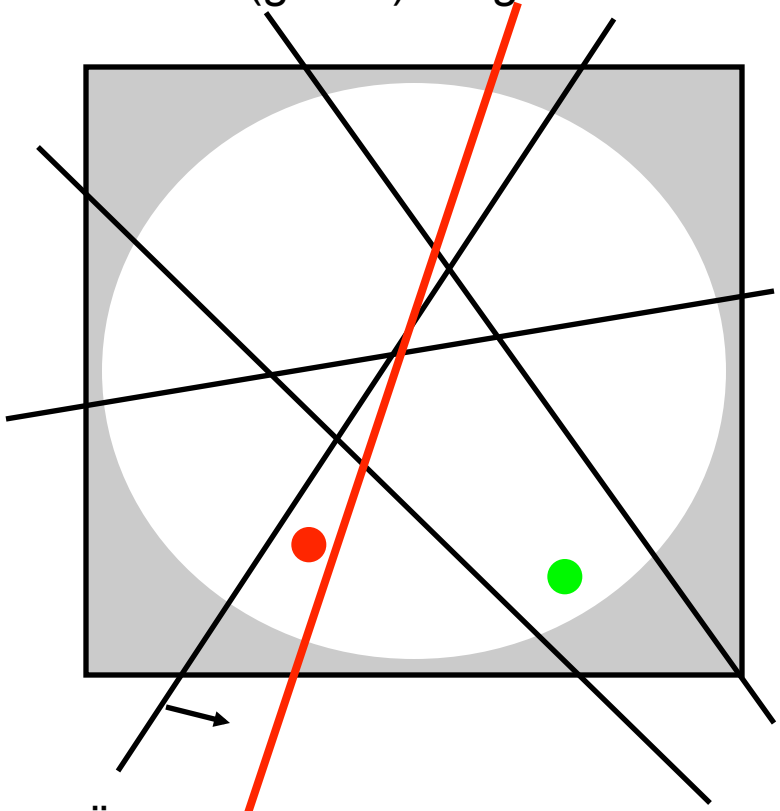
Auswahl repräsentativer Trainingsbeispiele

- Lerndaten
  - für die Anpassung der Gewichte
- Testdaten
  - für das Testen des Fehlers und auf Overfitting
- Verifikationsdaten
  - für das Feststellen der Generalisierung
- gute Verteilung
  - Klassifikation: Daten aus allen Klassen
  - Regression: gesamter Definitionsbereich
- komplexe Regionen
  - Randregionen zw. Klassen
  - Verlaufsänderungen



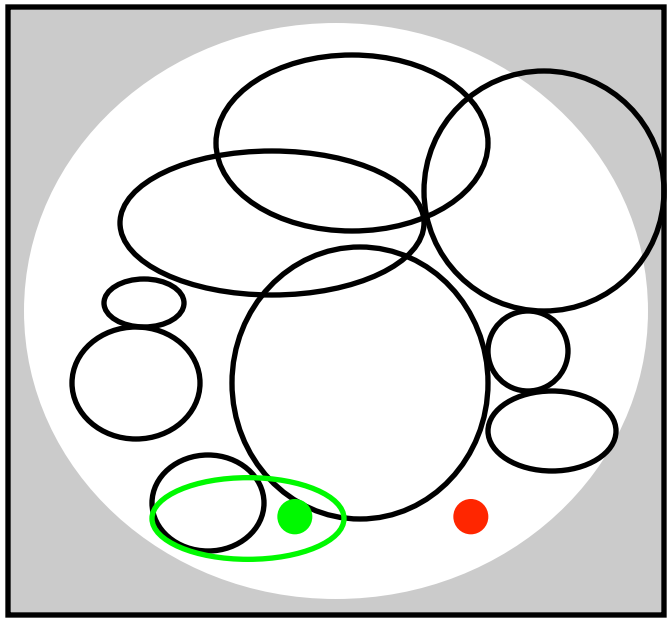
Ansätze der Wissensrepräsentation

Verteilt (global) - Sigmoid MLNN



- Änderungen wirken sich global aus
- Generalisierung ist gut

Lokal RBF - NN



- Änderungen nur lokal
- Generalisierung schlechter

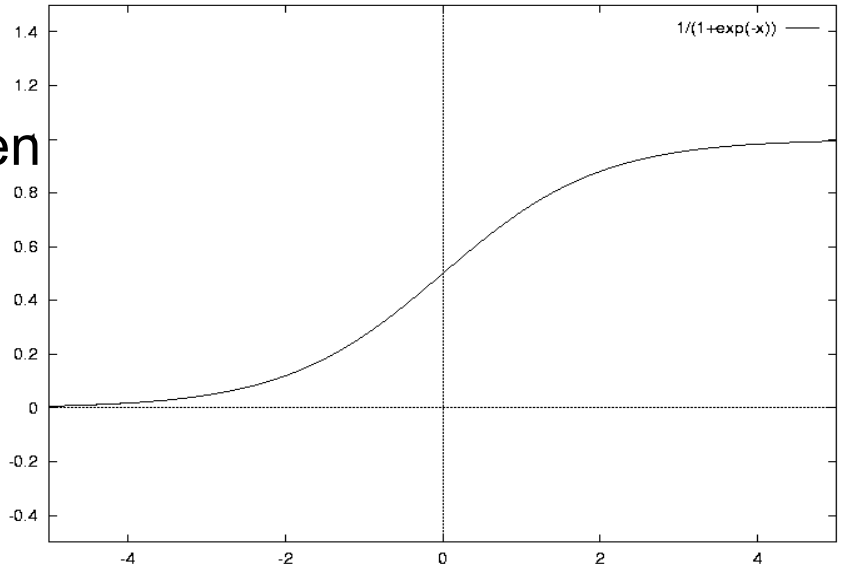
Verteilte vs. Lokale Wissensrepräsentation

- Vergleichskriterien:
  - Zielsetzung: Generalisierungsverhalten (verteilt)
  - Inkrementelles Lernen (Beibehalten von Wissen: eher lokal)
  - Interpretierbarkeit (lokal)
  - Einsatz in hybriden Lernarchitekturen (beide)
    - Umsetzbarkeit von bzw. in Fuzzy-Wissensbasen (lokal)

Initialisierung der Gewichte

Gewichte:
verschieden voneinander
sonst funktionsgleiche Neuronen

zufällig, gleichverteilt und klein
⇒ keine anfängliche Ausrichtung



weil $\Delta w_{ij} \Leftrightarrow \frac{\partial f(net_j)}{\partial net_j}$ und $net_j = \sum_{i=1}^{n} w_{ij} x_{ij}$

Anpassen der Gewichte

Patternlearning

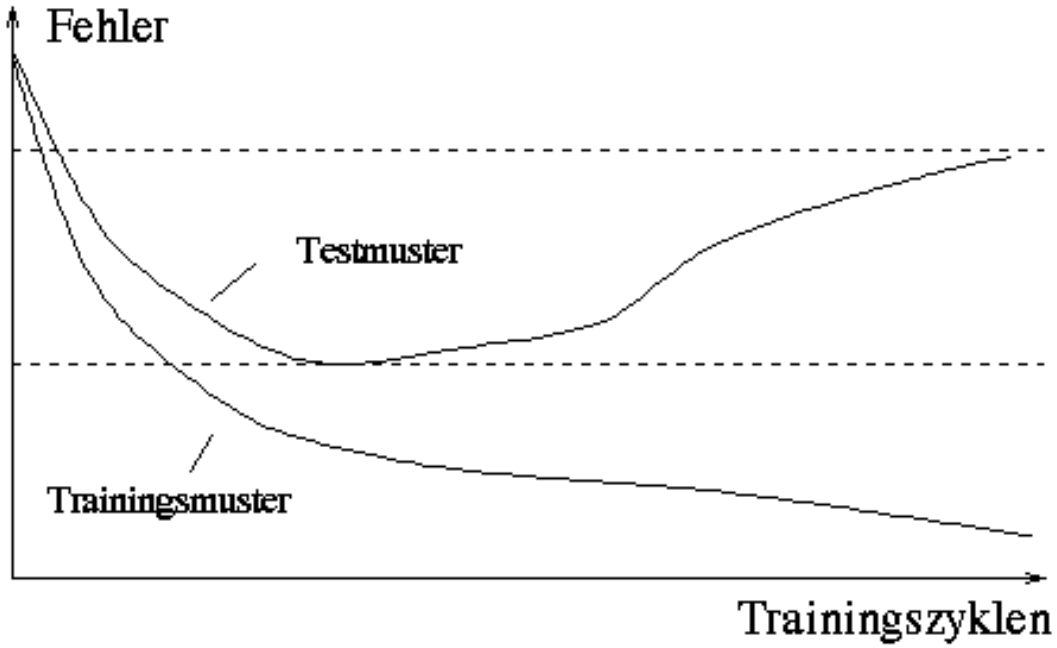
Anpassung nach jedem Lernbeispiel
schnelles Lernen
kein „echter“ Gradientenabstieg
zufällige Reihenfolge ➔ gute Approximation

Epochlearning

Mittelung der Gewichtsänderung über alle Beispiele
Anpassung nachdem alle Beispiele propagiert wurden
„echter“ Gradientenabstieg
nicht anfällig für Ausreißer / falsche Lerndaten

Overfitting

Fehler auf Verifikationsdaten steigt
ab einer Anzahl von Lernzyklen



Mögliches Abbruchkriterium

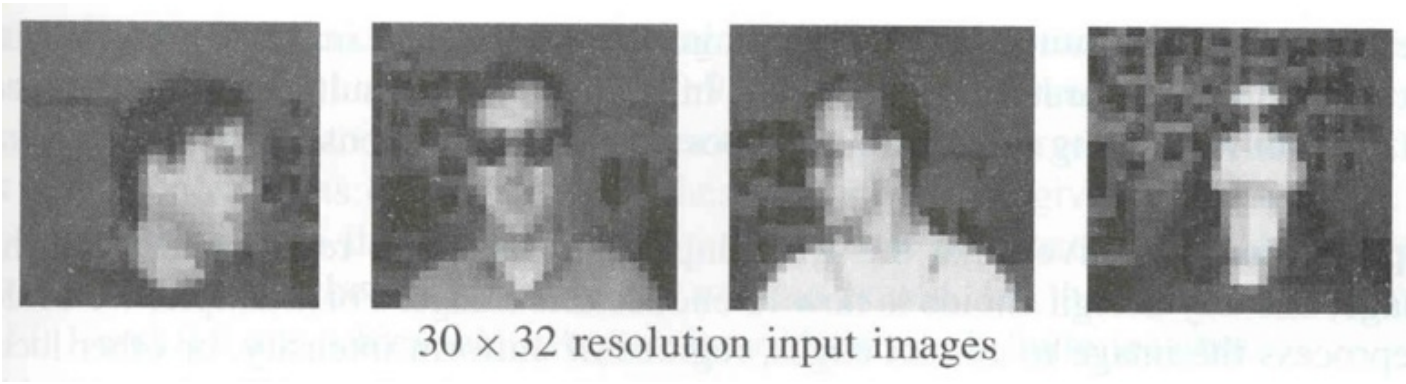
Ältere Anwendungen von MLNN

Einige klassische Beispiele und typische Probleme:

- Gesichtserkennung
- PdV
- Alvinn ←→ Grand Challenge
- Lauron

Vorsicht: Wohldefinierte Anwendungen !

Gesichtserkennung - MLNN



Vorgabe:

Grauwertbilder (255 Stufen, 128x128 pixel)

Ziel:

Erkennen ob ein Gesicht vorhanden ist
Welches Geschlecht
Blickrichtung (links, rechts, hoch, geradeaus)

Lernbeispiele

20 Personen
32 verschiedene Bilder/Person

Gesichtserkennung - MLNN

Input:

Rohdaten (Pixel), z.B.: 30x32 – Bild

➔ langsames Lernen

Merkmale wie Kanten, Ecken usw.

➔ Anwendungsabh., unterschiedliche Anzahl von Merkmalen ist problematisch

Output

- z.B: eine Ausgabe für Blickrichtung (0.2, 0.4, 0.6, 0.8)
- z.B: je eine Ausgabe für jede Blickrichtung ({0,1}, {0,1}, {0,1}, {0,1})
- Schwellwerte für die Interpretation der Ausgaben (z.B: [0,0.3] ➔ 0)
- analog für die anderen Ausgaben

Netz, Lernverfahren:

- z.B: Blickrichtung: Topologie: 960 x 3 x 4
- Lernrate z.B: $\eta = 0.3$
- Patternlearning
- Verifikation nach 50 Anpassungsschritten

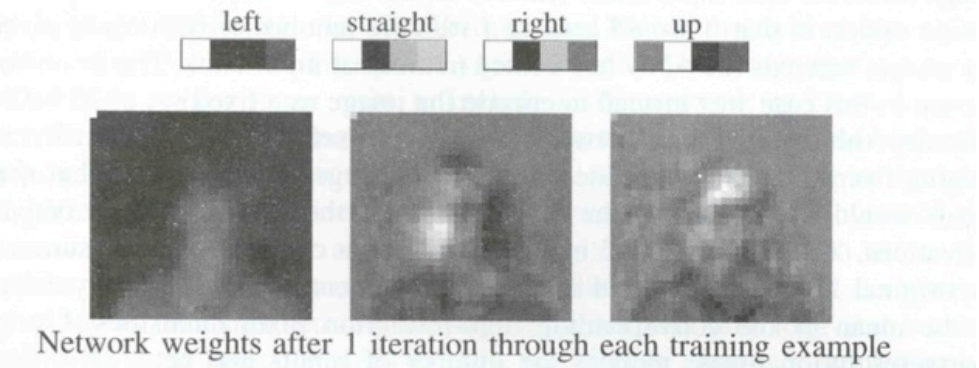
Gesichtserkennung - MLNN

Ergebnis:

260 Beispiele ➔ 90% Erkennungsrate für die Blickrichtung

Analyse der output und hidden Gewichte:

- nach 1 Iteration

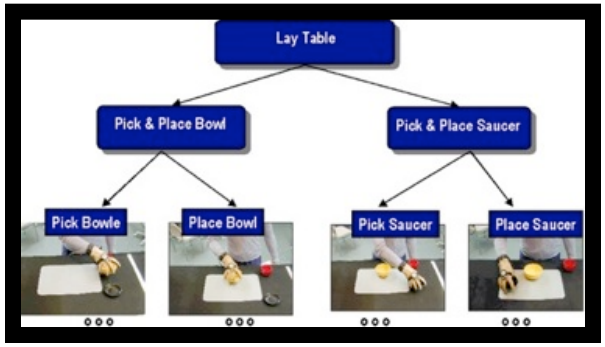


- nach 100 Iterationen

Hier keine Relevanz!!! Weil keine lokale Wissensrepräsentation

Programmieren durch Vormachen (PdV) - Skill Transfer (2000-2005)

Abstract Action Program "Macro Operator"



Learning

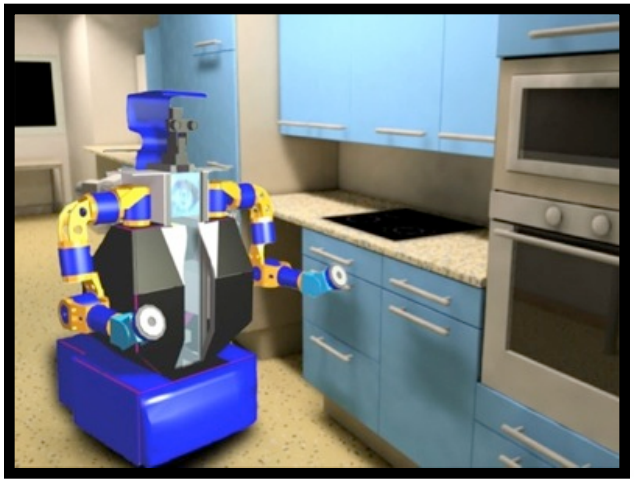
Adaption

Cognitive reasoning (explicit)

Sensor dependent Knowledge (implicit)

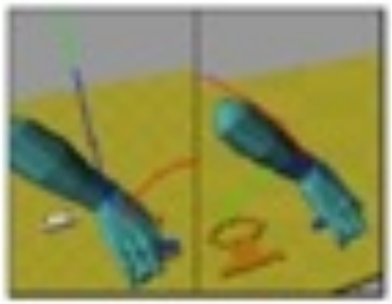
Instruction synthesis (explicit)

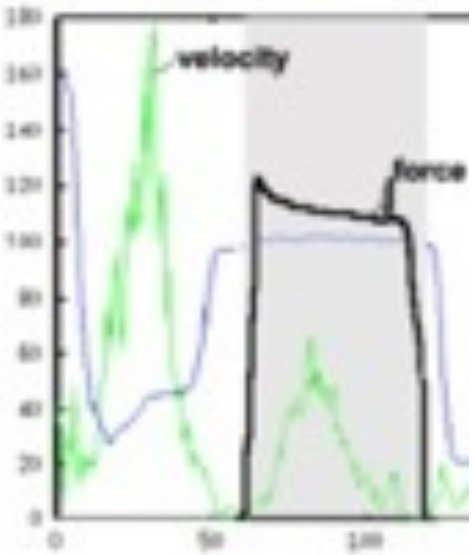
Sensor & Actor dependent Knowledge (implicit)



PdV – Klassifikation von statischen Griffen

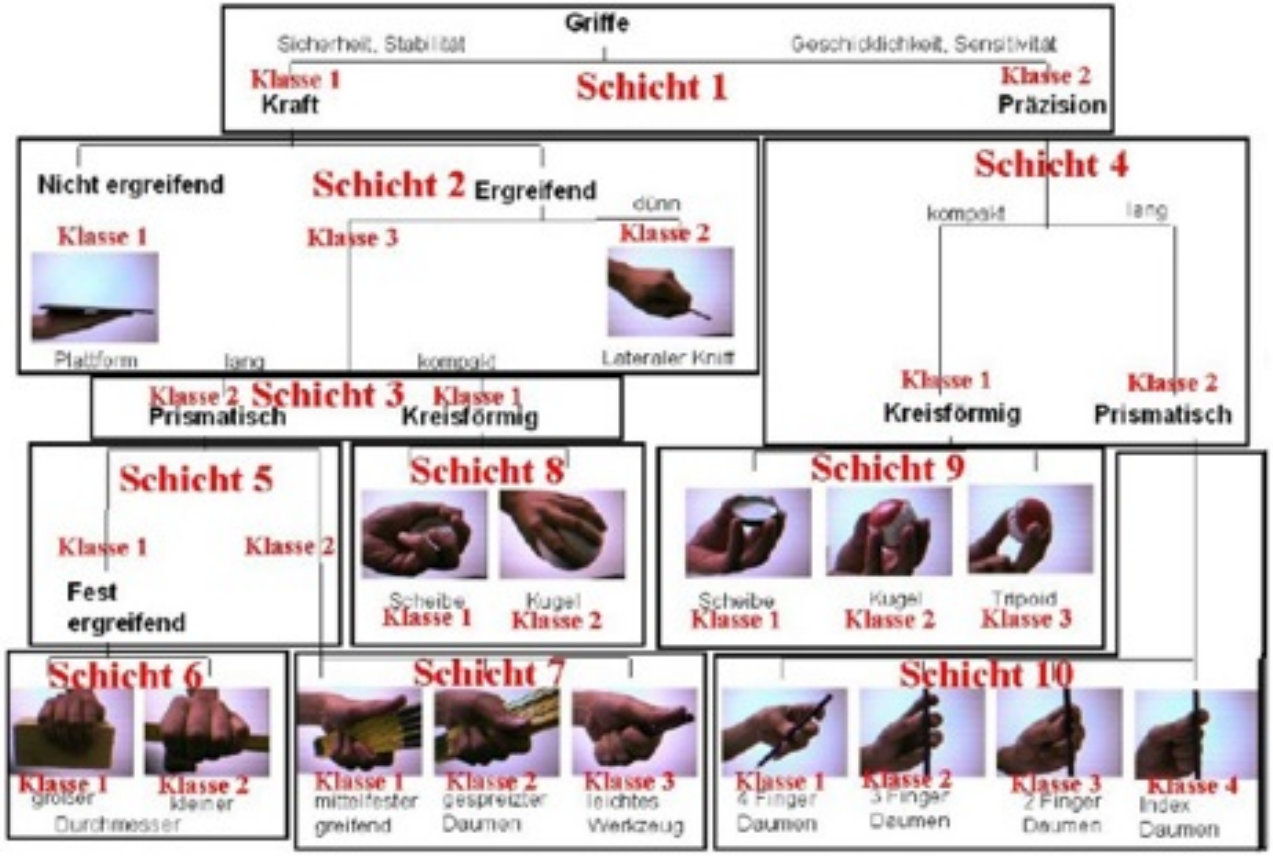
Ziel:
Erkennung von
Greifphasen beim
Vormachen





Daten:
Gelenkwinkel
Geschwindigkeit
Kräfte in der Hand

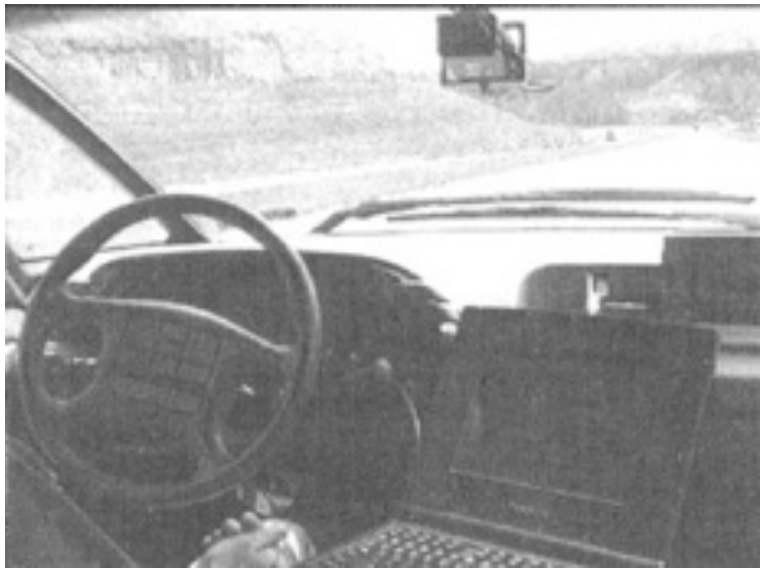
PdV – Klassifikation von statischen Griffen



Klassen: Cutkosky – Taxonomie

Hierarchisches Netz von RBF Netzten

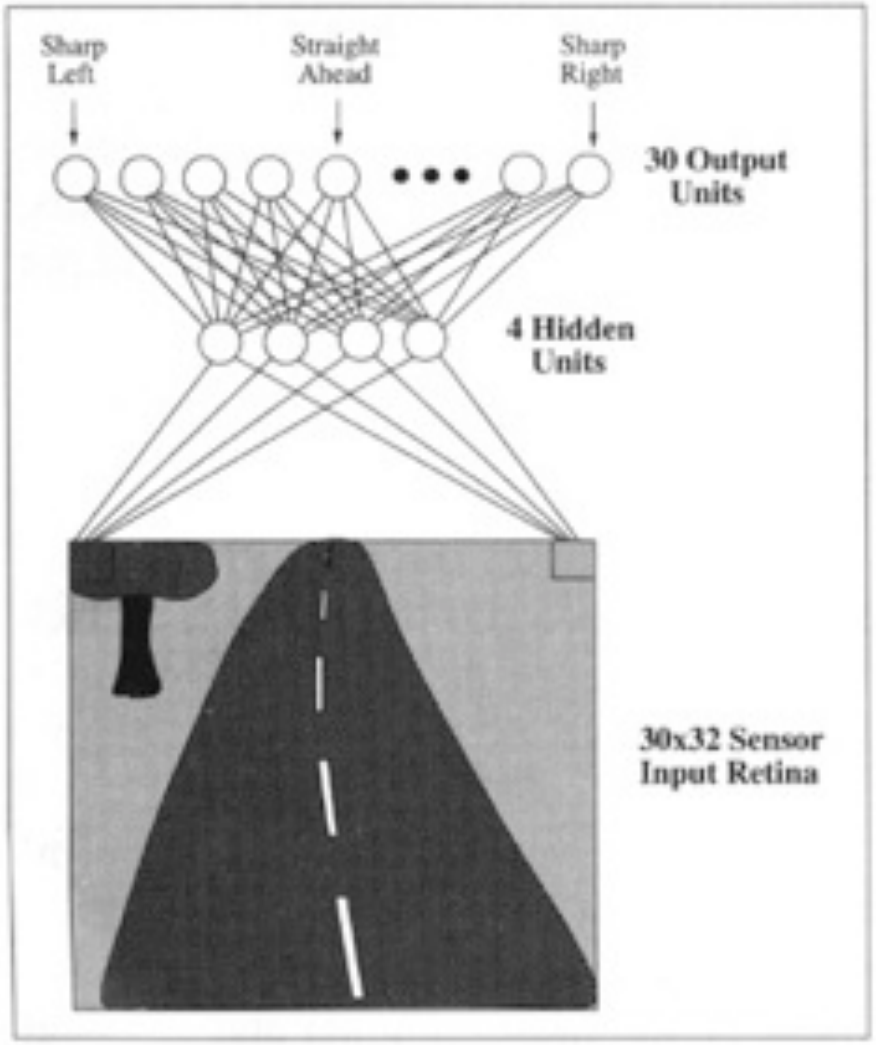
Autonomes Fahren mit Neuronalen Netzen
ALVINN (Pomerleau ,CMU, 1996)



Ziel:

Steuerung eines Fahrzeugs mit Hilfe von Neuronalen Netzen

Autonomes Fahren mit Neuronalen Netzen
ALVINN



Beispiel: Realisierung der Lenkung

Input: Rohdaten (30x32 Pixel)

Output: 30 Lenkpositionen {0,1}

Topologie

- 960 Input Neuronen
- 4 Hidden Neuronen
- 30 Output Neuronen

Autonomes Fahren mit Neuronalen Netzen ALVINN

- Problem: Beispieldatengenerierung
  - Fahrer macht keine Fehler
    ⇒ Ausnahmesituationen fehlen
  - Viele verschiedene Situationen werden benötigt
    - Lichtverhältnisse
    - Straßenformen
  - Sammeln ähnlicher Daten während einer Fahrt

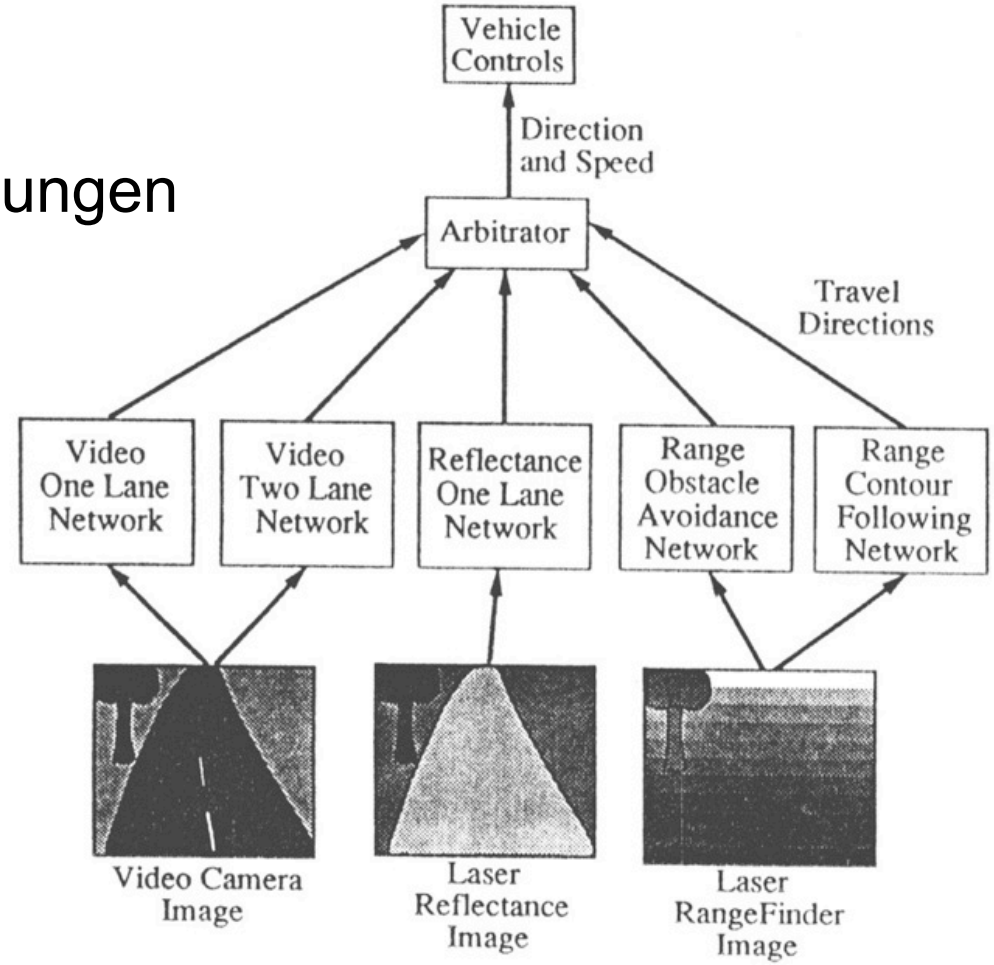
  → Gefahr einseitigen Lernens

- Lösungsansätze
  - Generierung künstlicher Daten
    - Rauschen, Verschieben (immer noch gängig☺)
  - Generierung ausgewogener Datensätze
  - Multi - Network Ansatz

Autonomes Fahren mit Neuronalen Netzen ALVINN

Unterschiedliche Bedingungen

➔ Vielfältige Sensorik
➔ mehrere Netze

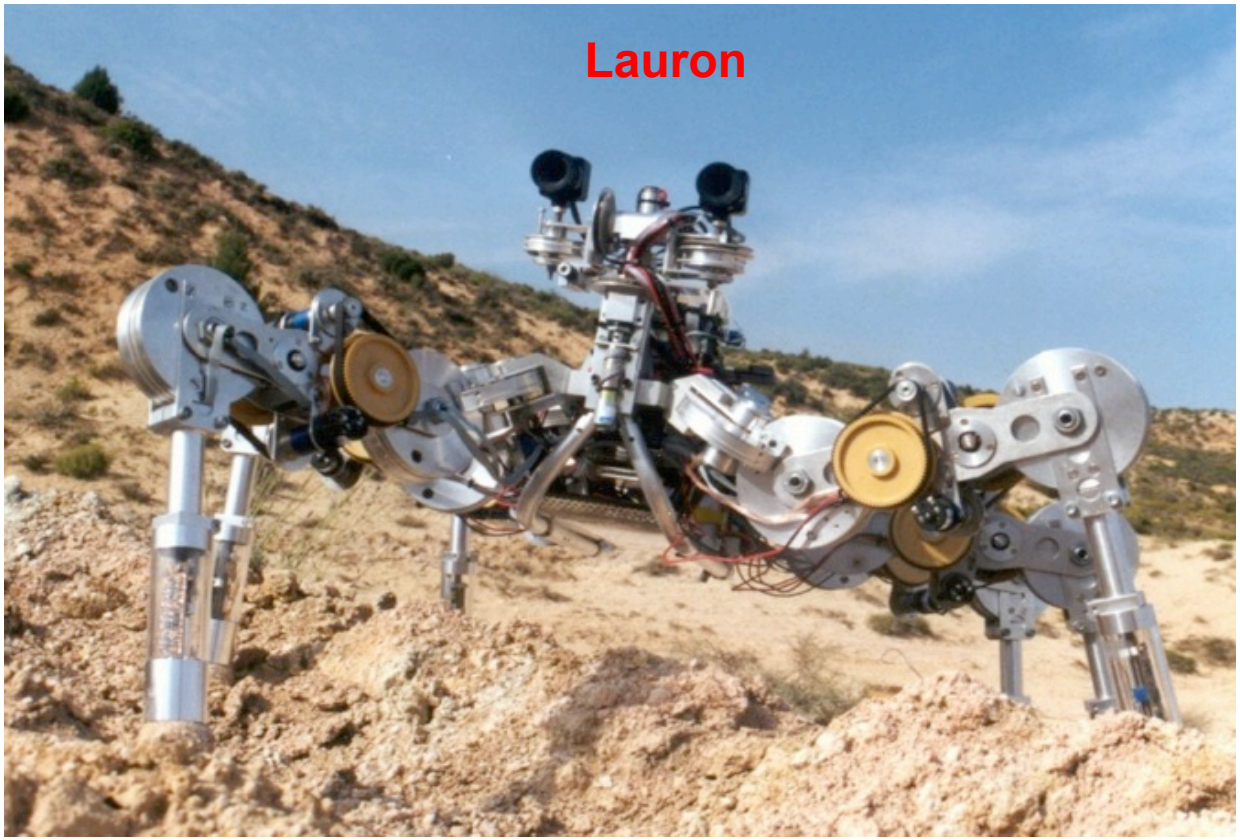


Im Vergleich dazu: Autonomes Fahren – (seit) Grand Challenge 2005

http://video.google.com/videoplay?docid=8594517128412883394

- Keine NN !!

Lauron (Laufmaschine neuronal gesteuert)



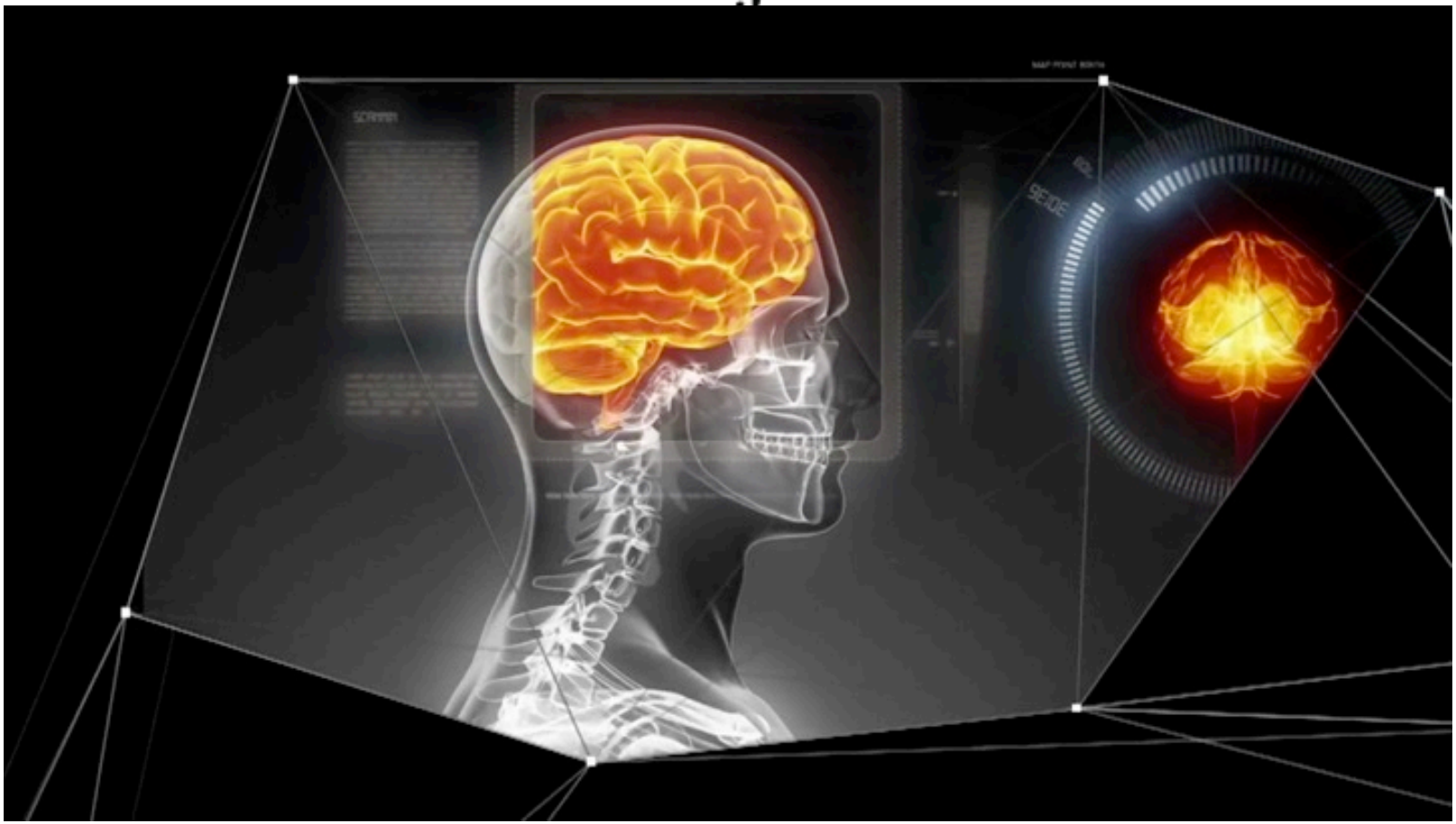
Frühe Anwendung NN: Bewegungssteuerung, Steuerung der einzelnen Beine (Gelenkwinkel)

Heute – Verhaltensbasierte Steuerung ohne NN

Bald ☺ – Verhaltensbasierte Steuerung mit Spiking NN (3. Generation)

EU – Flagship Projekt (seit 2013)

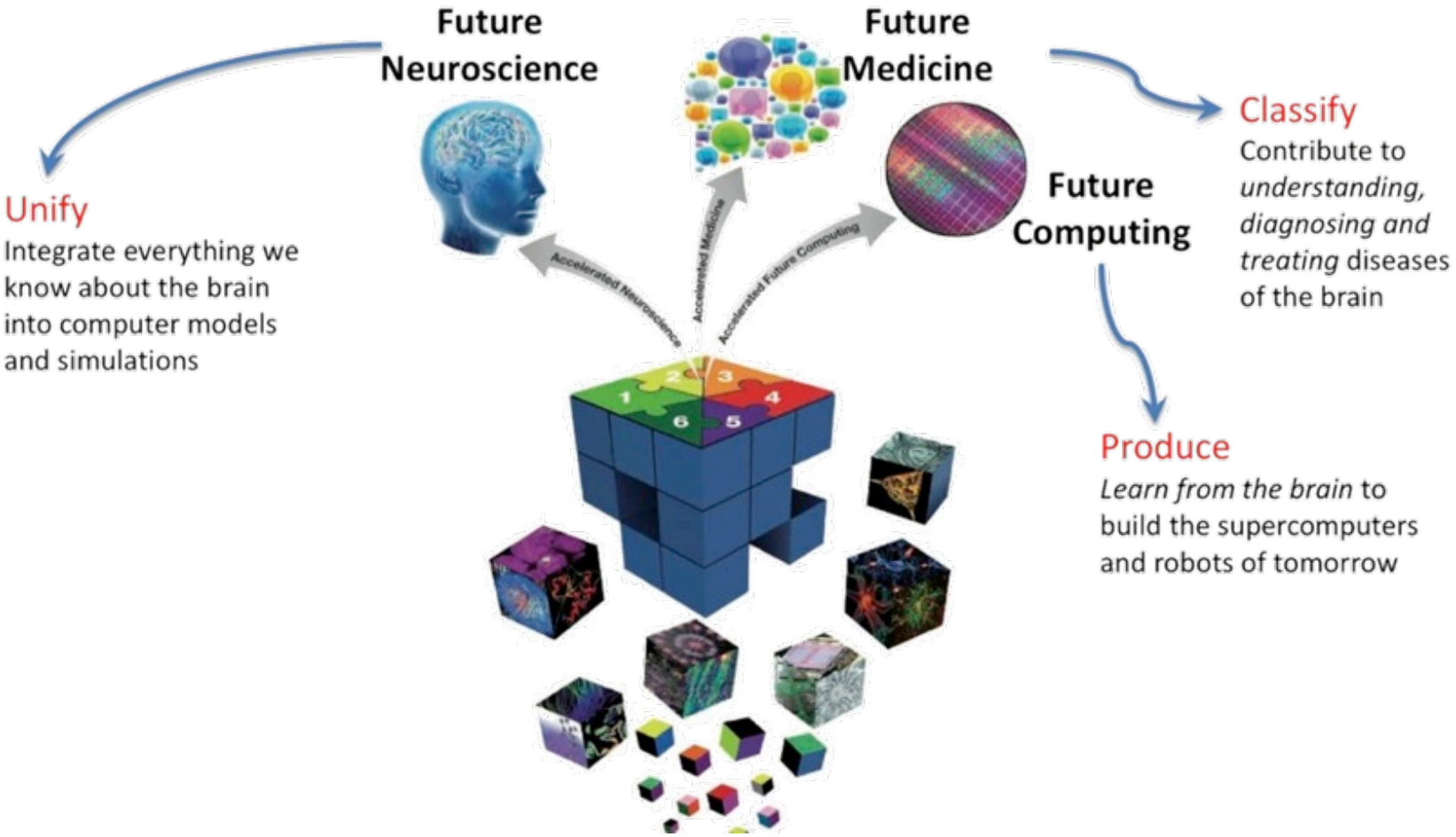
Human Brain Project

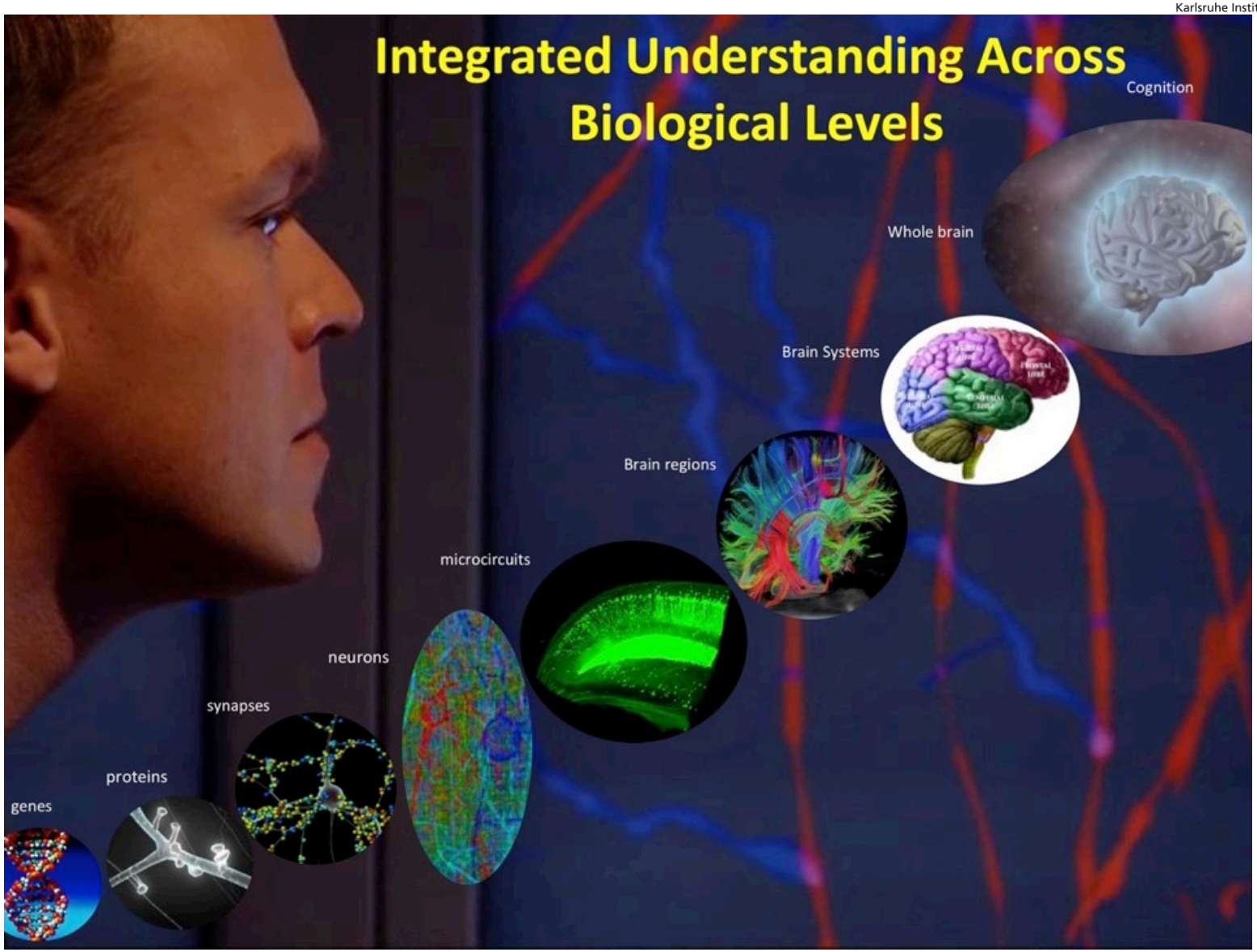


https://www.humanbrainproject.eu
http://www.youtube.com/watch?v=JqMpGrM5ECo

The HBP research areas and long-term goals

Goal: to *develop technology to unify our understanding of the human brain and to transfer this knowledge into products*

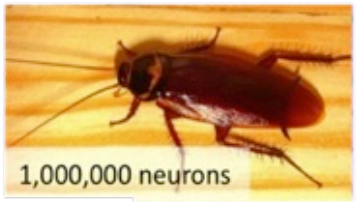



Integrated Understanding Across Biological Levels
Cognition
Whole brain
Brain Systems
Brain regions
microcircuits
neurons
synapses
proteins
genes

Motivation: Why is the brain interesting for us, anyway?

10,000 neurons

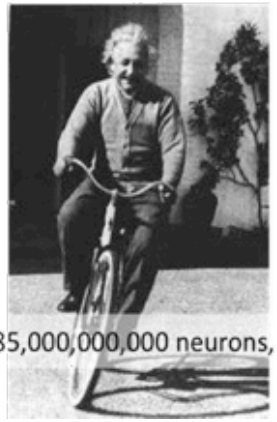
850,000 neurons



1,000,000 neurons

75,000,000 neurons

200,000,000 neurons



~85,000,000,000 neurons, 10^{15} synapses

- Algorithms are embedded in hardware
- Sensors and effectors operate in real-time
- The brain is massively parallel, but does not suffer from the problems of parallel computing: dead-locks, non-determinism, race-conditions, …
- … and decomposition into parallel tasks is self-organized/evolved
- Architecture is scalable from thousands to billions of "processors"
- Performance is robust – with graceful degradation
- Brains are extremely power and space efficient ("peta-flop computer on 20 Watt")
- Calls for a "neurorobotics approach"

Human Brain Project

Neurorobotics – what is it about?

- **Building robot bodies** with an **embedded brain** and **embedded control systems** → mimicking the structure and function of the nervous systems of creatures
- Pre-condition: development of **biologically accurate models** of **brains and bodies**
- Research aspect: let these robots live in both **real** and **high-fidelity virtual environments**, and observe their (developing/growing) skills to **reverse-engineer** the brain

➢ Contribution to theory: HBP develops a **"Hilbert Programme towards the Ultimate (Cognitive) Robot"** for Neurorobotics

➢ Contribution to practice: HBP implements a tool chain and technology to design **"Modular Brains for Modular Bodies"**

Human Brain Project

Ein Ansatz: Spiking Neural Networks SNN

- Gepulste neuronale Netze
- Derzeit am häufigsten/häufig diskutierte KNN Modelle
- Hintergrund: Möglichst reale Abbildung natürlicher NN
- Berücksichtigen auch den zeitlichen Effekt der Aktivierung von Neuronen

- Idee:
  Neuronen feuern nicht wie bei MLP in jedem Propagationszyklus sondern abhängig von Kodierungsmodellen, Aktionspotential (als Folge der Frequenz eingehender Spikes oder der Zeitspanne zw. Spikes)

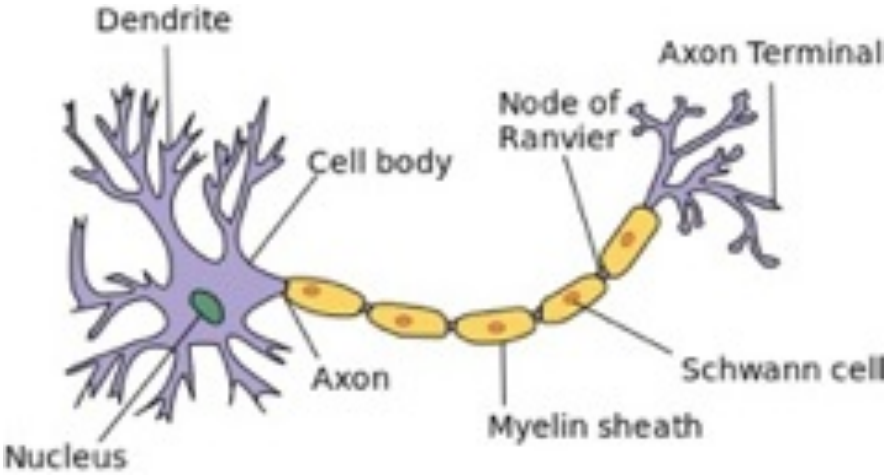
Spiking Neural Networks SNN

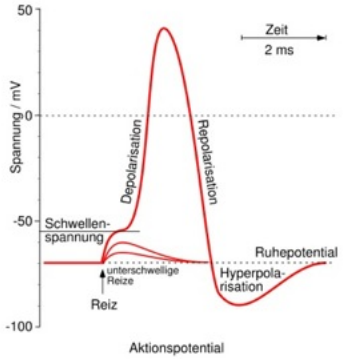
- Signal besteht aus kurzen elektr. Pulsen
- Pulse (Aktionspotentiale/Spikes) ~100mV der Dauer v. 1-2 ms
- Puls ändert sich bei Propagation entlang des Axons nicht

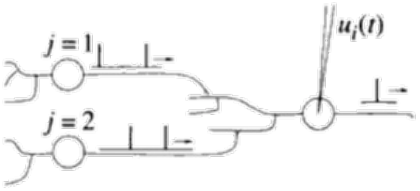
- Da alle Spikes eines Neurons gleich sind tragen sie keine Information

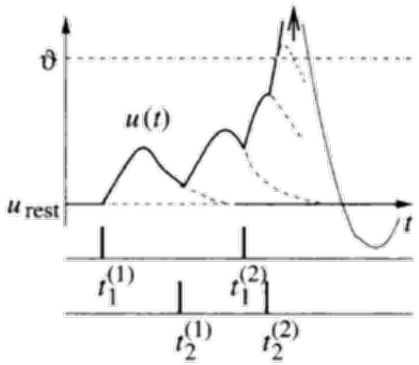
→

- Anzahl und Zeitpunkte der Spikes spielen eine Rolle
  z.B.: 20-50 Spikes in kurzer Zeit aktivieren Folgeneuron





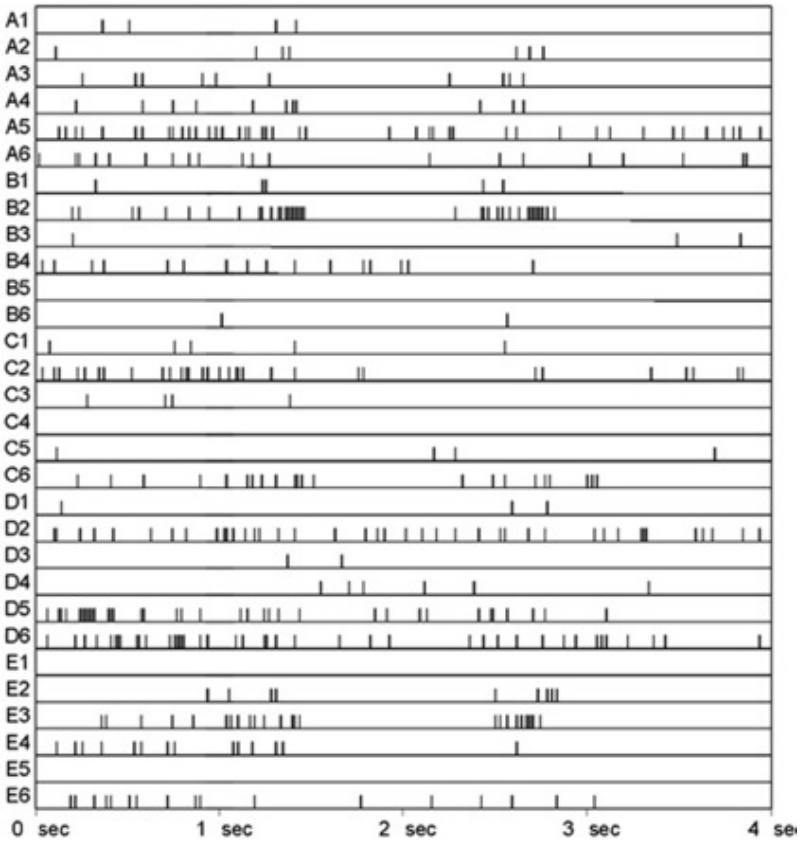




Spiking Neural Networks SNN

- In jedem kleinen Volumen des nat. Kortex werden Tausende von Spikes pro. ms emittiert

- Viele Forschungsfragen:
  - Aufbau Gehirn?
  - Teilsysteme - Aquivalenz zu anderen (bekannten) Algorithmen
  - (Unüberwachtes) Lernen?
  - Wie entsteht daraus Kognition?
  - HW - Realisierung?
  - ....

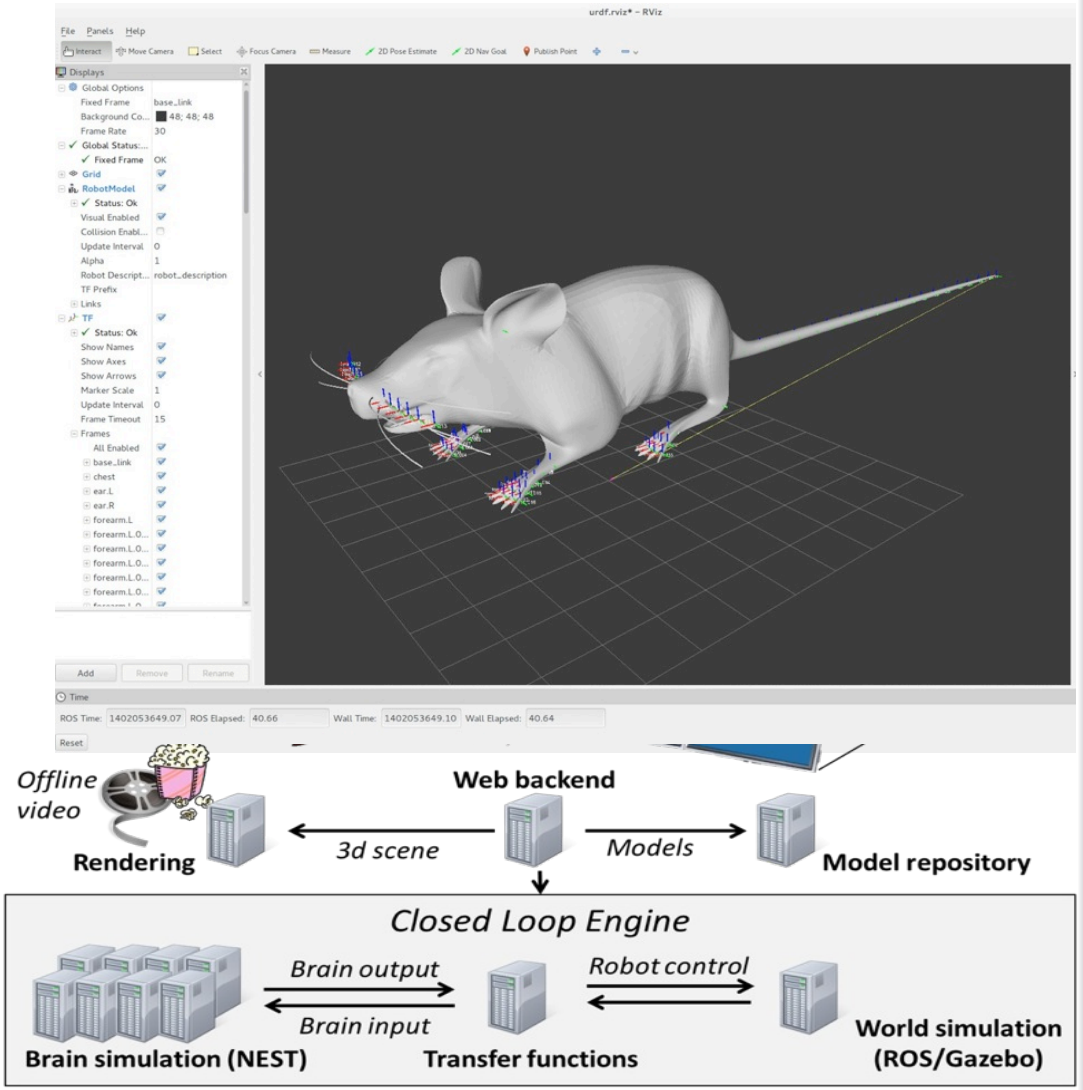


30 Neuronen im Affenkortex?

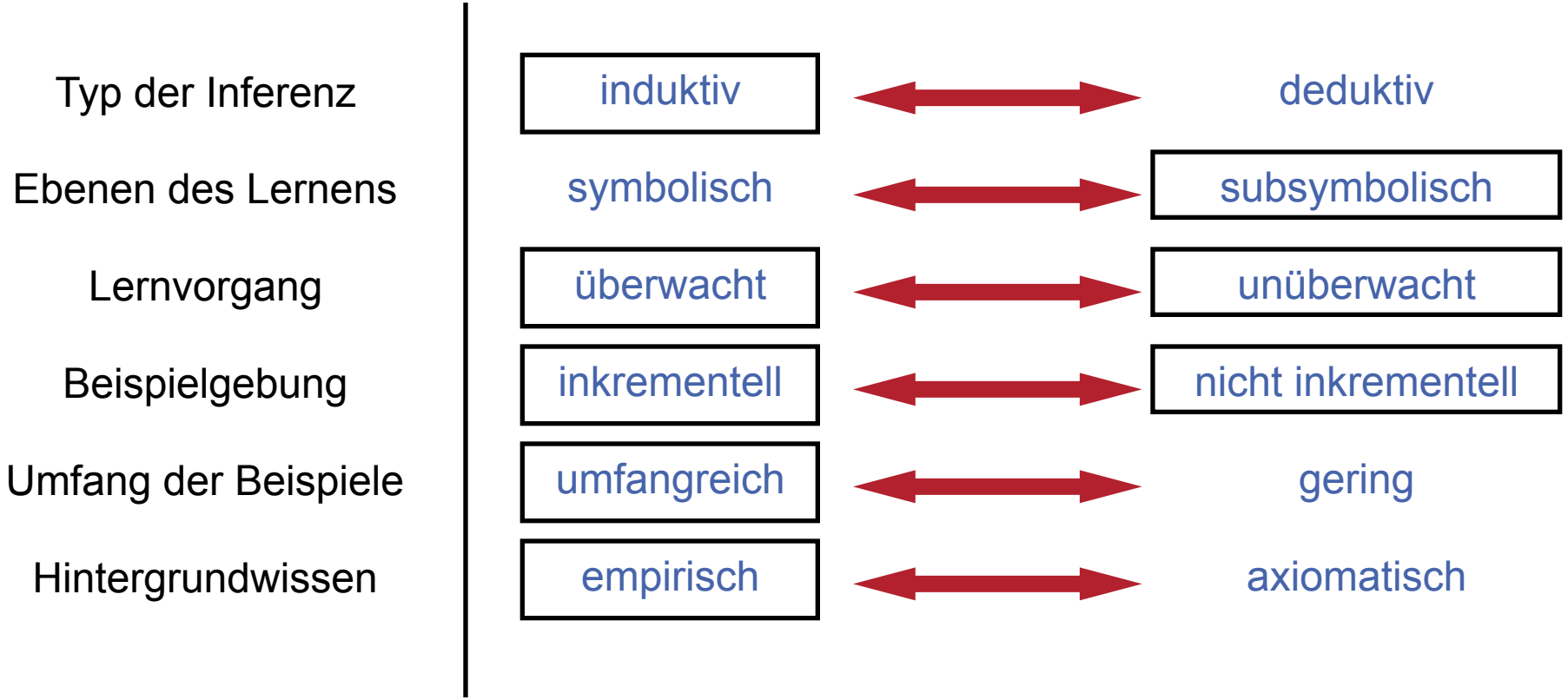
Forschungsfragen am FZI

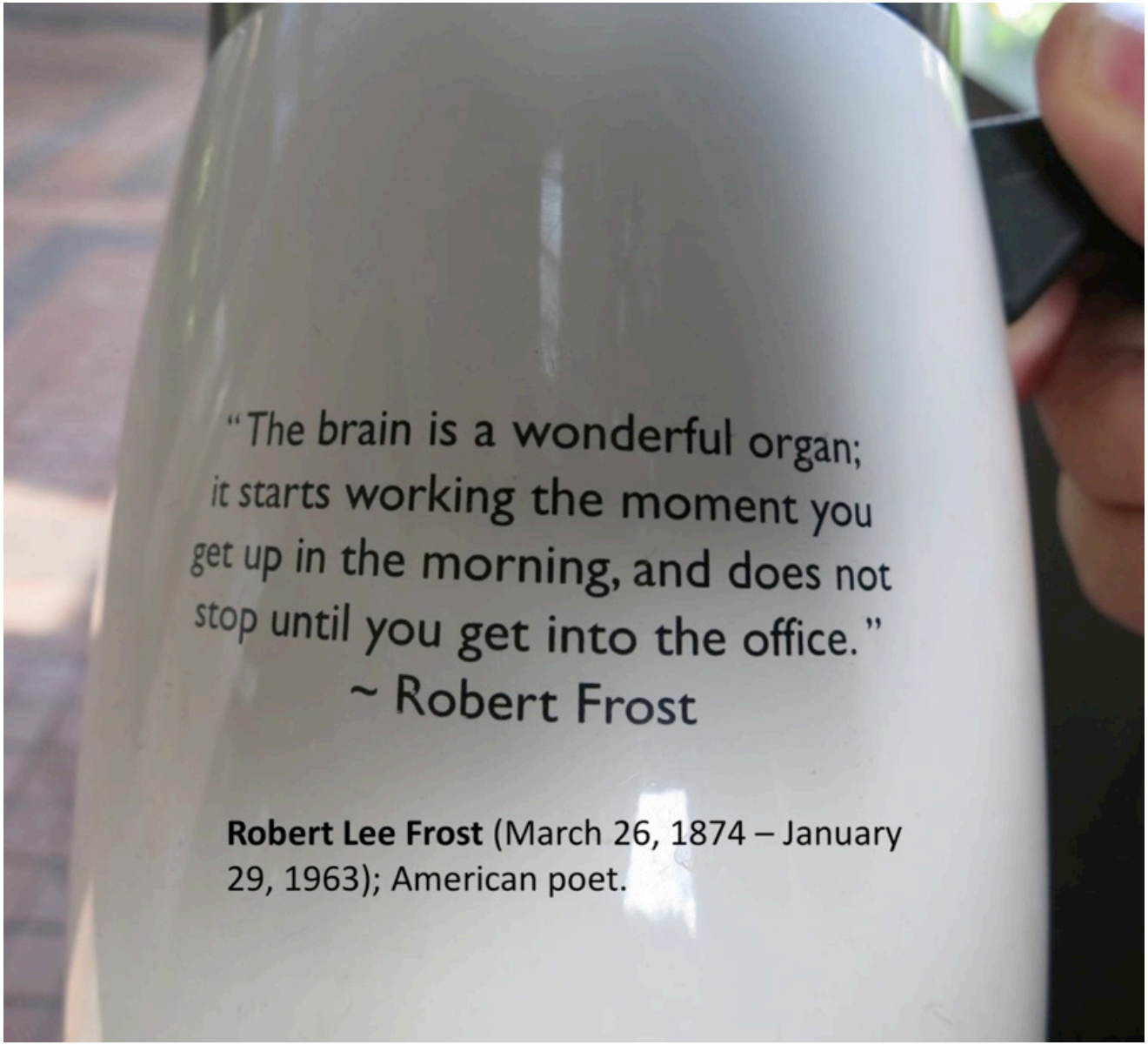
- Was lässt sich dadurch in der Robotik erreichen? Wie? Wieso?
- Gibt es Modularität?
- Welche? Wie?
- Umsetzung in Verhalten?
- Wie lernen solche NN? (Auf Basis vorhandener NN- Modelle)

- Interesse?
  → FZI



NN: Einordnung




"The brain is a wonderful organ;
it starts working the moment you
get up in the morning, and does not
stop until you get into the office."
~ Robert Frost
Robert Lee Frost (March 26, 1874 – January 29, 1963); American poet.

Literatur

- *Tom Mitchell*: ***Machine Learning***. McGraw-Hill, New York, 1997.
- *M. Berthold, D.J. Hand*: ***Intelligent Data Analysis.***
- *P. Rojas*: ***Theorie der Neuronalen Netze – Eine systematische Einführung***. Springer Verlag, 1993.
- *C. Bishop*: ***Neural Networks for Pattern Recognition.*** Oxford University Press, 1995. Vorlesung „Neuronale Netze 2006“ http://isl.ira.uka.de/
- (siehe auch „Ein kleiner Überblick über Neuronale Netze“, http://www.dkriesel.com)