Modellbildung und Simulation

Sommersemester 2011

5. Vorlesung

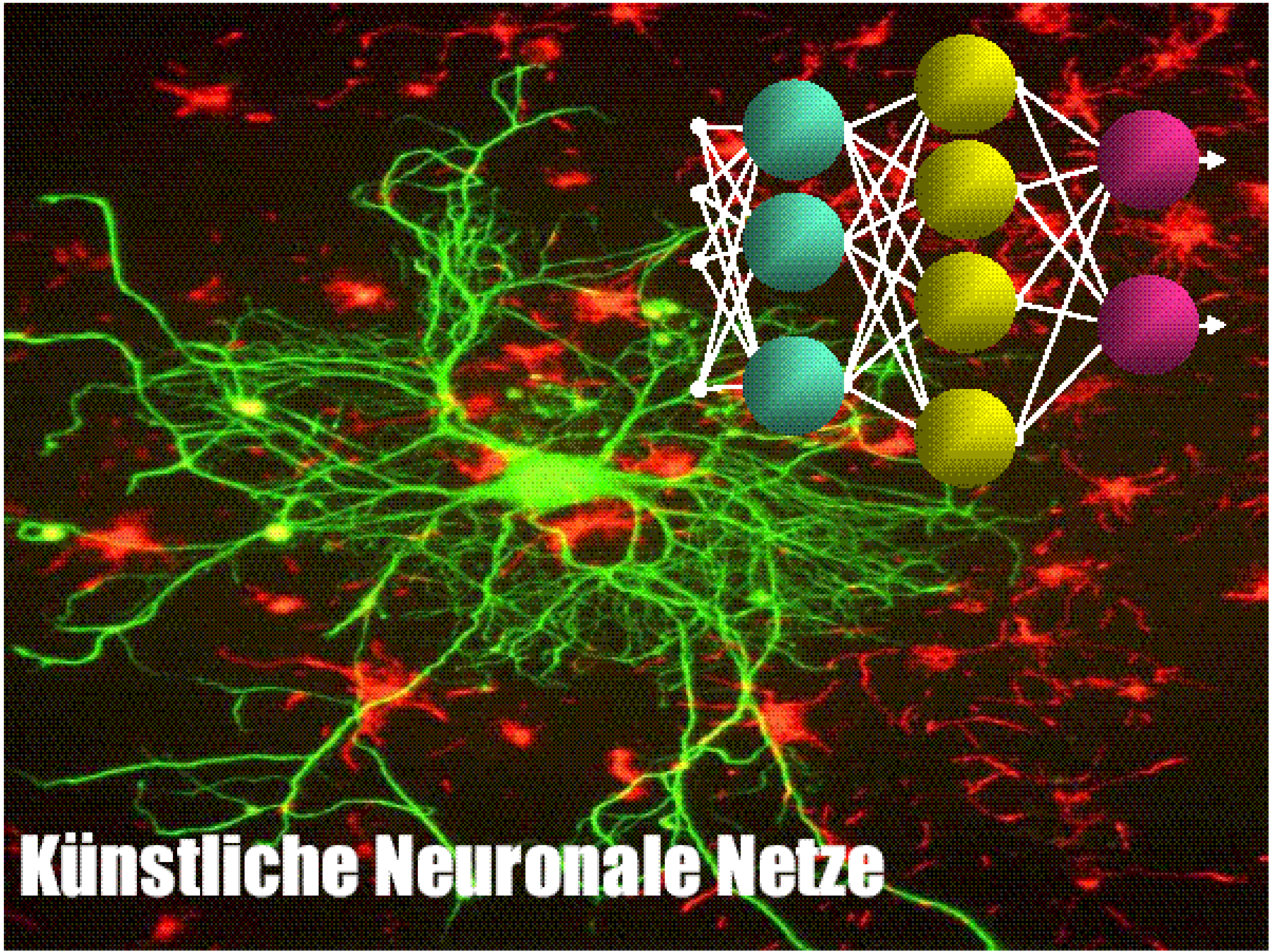
Klaus Kasper

Termine

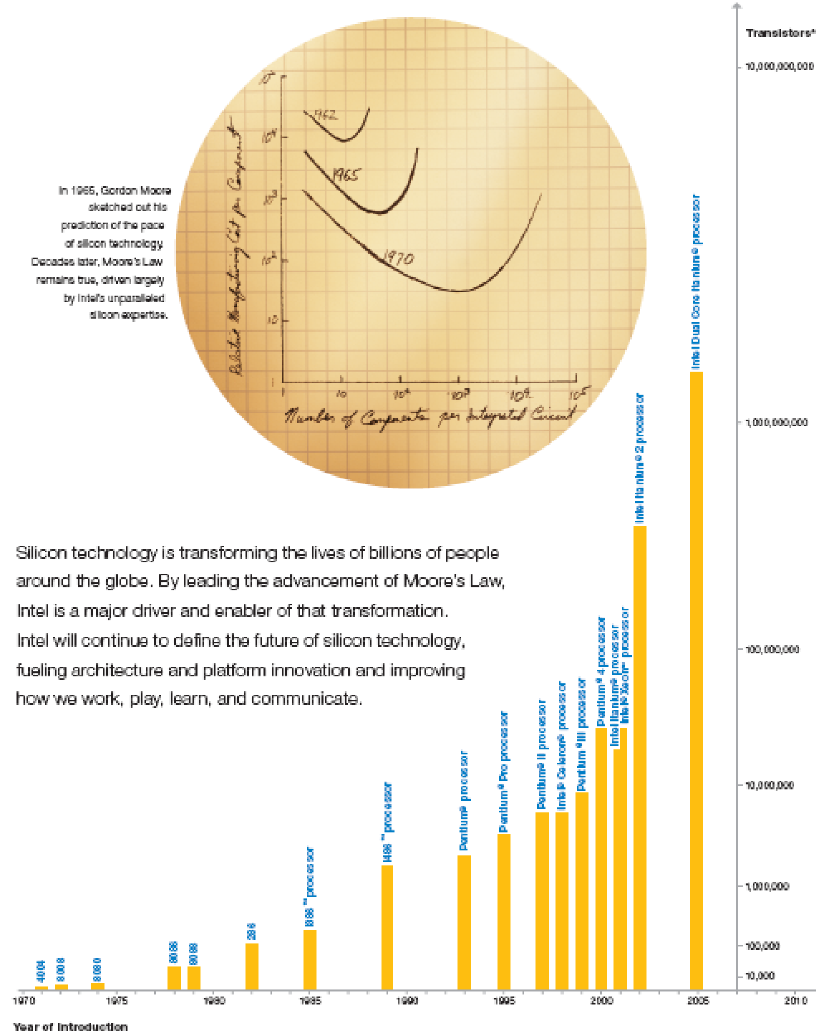
Veranstaltung	Vorlesung Mo 18-21 (X) D14/104	Praktikum Mo 18-21 (Y) D15/107	Praktikum Fr 12-15 (X) D15/107
1. Termin	28.03.11	04.04.11	08.04.11
2. Termin	11.04.11	18.04.11	29.04.11
3. Termin	02.05.11	09.05.11	13.05.11
4. Termin	16.05.11	23.05.11	27.05.11
5. Termin	30.05.11	06.06.11	17.06.11
6. Termin	20.06.11	27.06.11	01.07.11

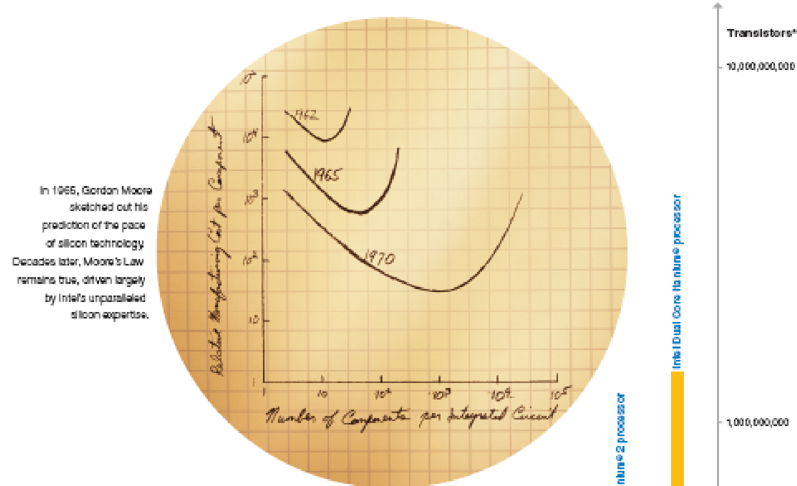
Inhalt

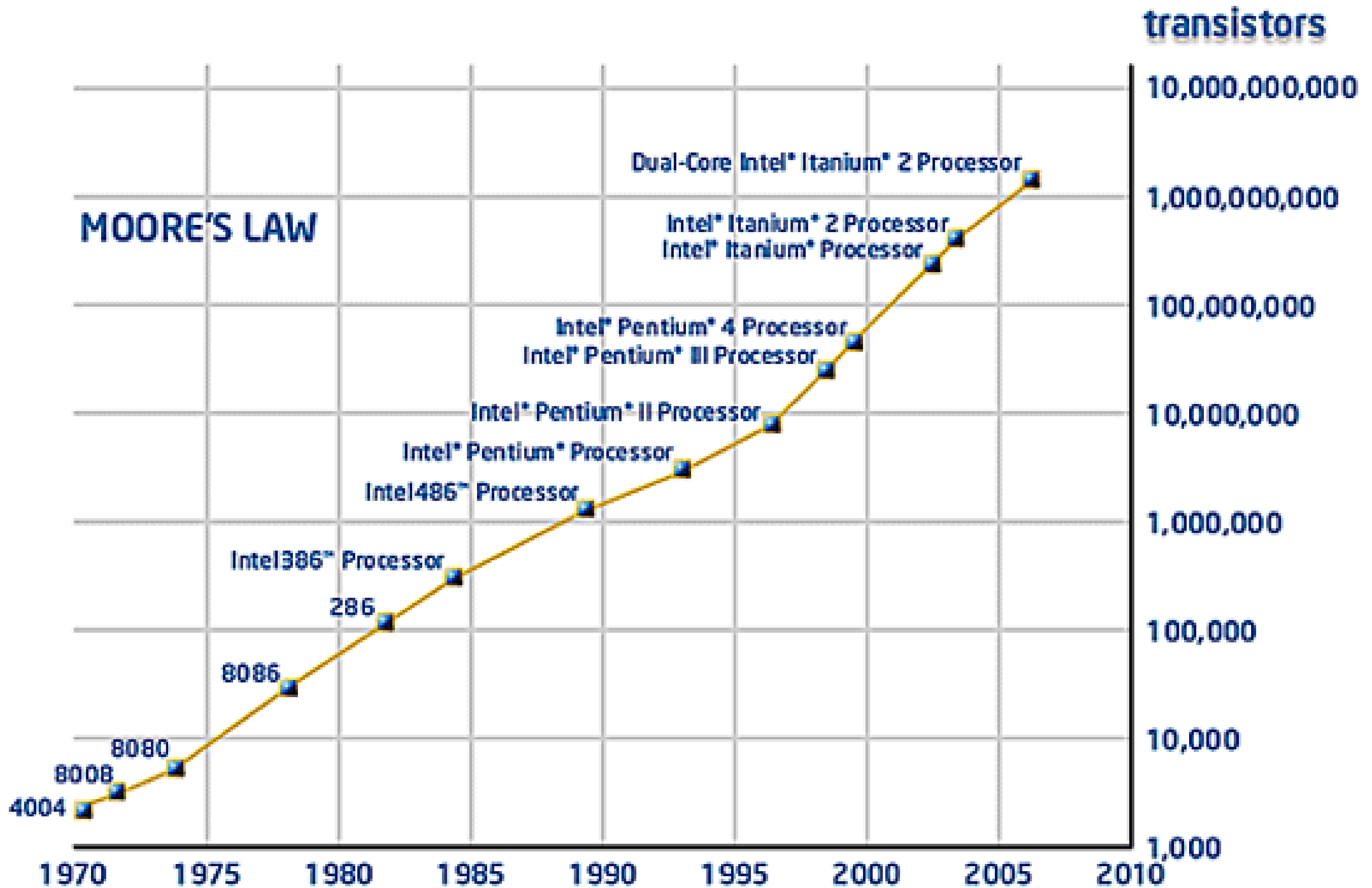
- Wiederholung
  - Mehrschichtiges Perzeptron (MLP)
  - Back-Propagation
  - Implementierung MLP
- Beschleunigung des Trainings
- Lösungen des XOR-Problems
- Generalisierung
- Gedächtnis
- Rekurrente Neuronale Netze

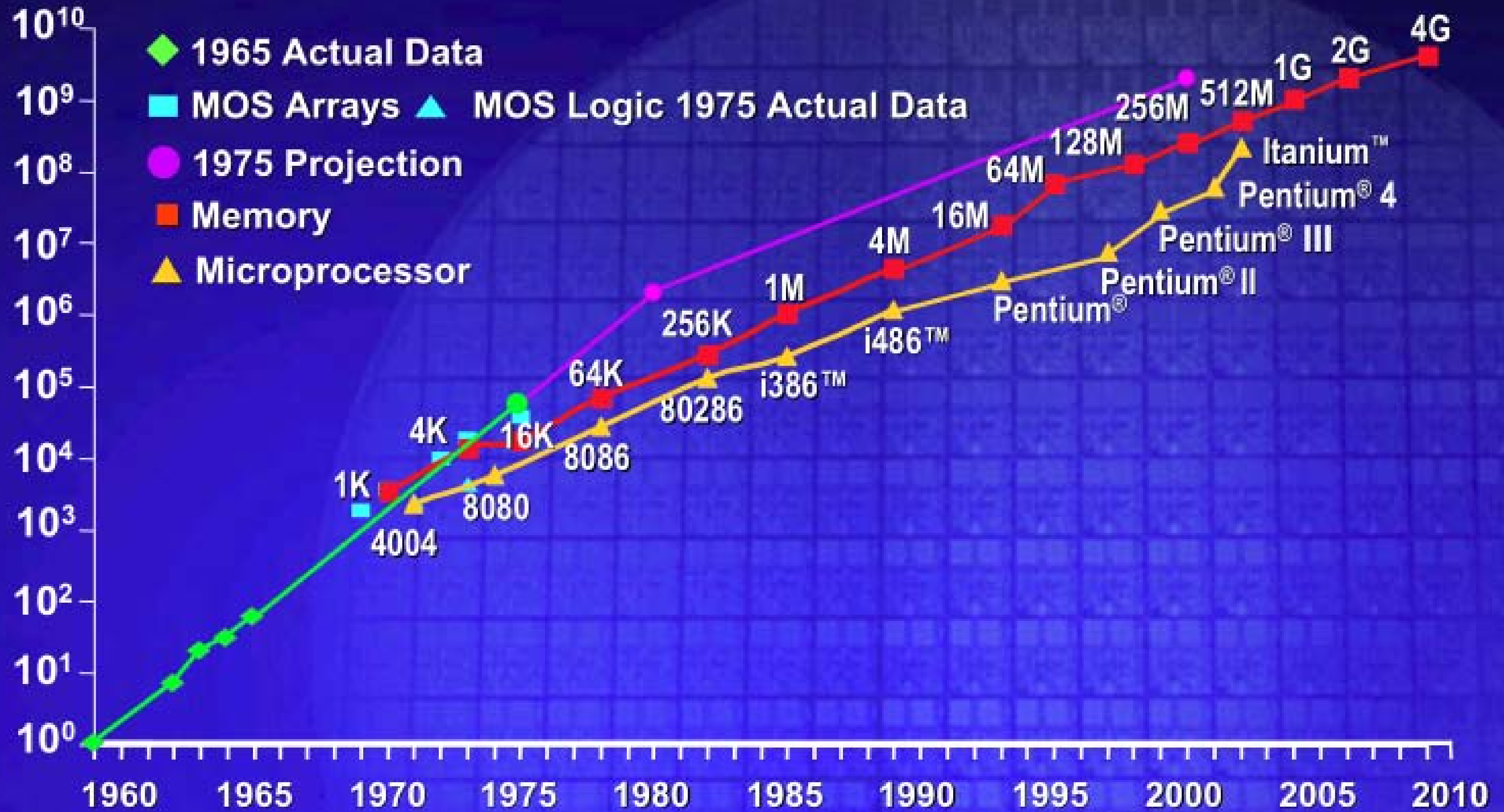

Künstliche Neuronale Netze

Moore's Law






transistors
MOORE'S LAW
Dual-Core Intel® Itanium® 2 Processor
Intel® Itanium® 2 Processor
Intel® Itanium® Processor
Intel® Pentium® 4 Processor
Intel® Pentium® III Processor
Intel® Pentium® II Processor
Intel® Pentium® Processor
Intel486™ Processor
Intel386™ Processor
286
8086
8080
8008
4004
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010


Transistors Per Die
1965 Actual Data
MOS Arrays
MOS Logic 1975 Actual Data
1975 Projection
Memory
Microprocessor
10^{10}
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
1K
4K
16K
64K
256K
1M
4M
16M
64M
128M
256M
512M
1G
2G
4G
4004
8080
8086
80286
i386™
i486™
Pentium®
Pentium® II
Pentium® III
Pentium® 4
Itanium™

Evolution of Computer Power/Cost

MIPS per $1000 (1998 $)

Billion

Million

1000

1

1/1000

1/Million

1/Billion

1900 1920 1940 1960 1970 1980 1990 2000 2010 2020 2030 2040 2050

MIPS

10^9

10^6

10^3

1

10^{-3}

10^{-6}

1995 Trend
1985 Trend
1975 Trend
1965 Trend

Dell 340/2G
Mac G3/266
Gateway G6-200
PowerMac 8100/80
Gateway-486DX2/66
Mac II
Macintosh-128K
Commodore 64
IBM PC
Apple II
Sun-2
DG Eclipse
CDC 7600
DEC PDP-10
IBM 7090
IBM 1130
Whirlwind
IBM 704
UNIVAC I
ENIAC
Colossus
Burroughs Class 16
IBM Tabulator
Zuse-1
Monroe Calculator
ASCC (Mark 1)
IBM 650
IBM 1620
Burroughs 5000
IBM 7040
IBM 360/75
SDS 920
DG Nova
DEC-KL-10
DEC VAX 11/780
Vax 11/750
Sun-3
Mac IIfx
IBM PS/2 90
AT&T Globalyst 600
Power Tower 180e
Mac G4 dual 500

Brain Equivalent

Human
100 kg 2 m
100G neurons

Monkey
3kg 60 cm
3G neurons

Mouse
25g 15 cm
60M neurons

Lizard
100g 25 cm
2M neurons

Guppy
30mg 1 cm
100K neurons

Worm
0.5μg 1 mm
300 neurons

Bacterium
0.5picogram 1 μm
1 "neuron"

Manual Calculation

First Similar Organisms

1 MYBP

60 MYBP

200 MYBP

350 MYBP

450 MYBP

550 MYBP

3,500 Million Years Before the Present

Comparable Machines

G4 eta 2050 (reasoning)

G3 eta 2040 (imagination)

G2 eta 2030 (adaptation)

G1 eta 2020 (skills)

Universal Robots

Utility Robot eta 2010-2015

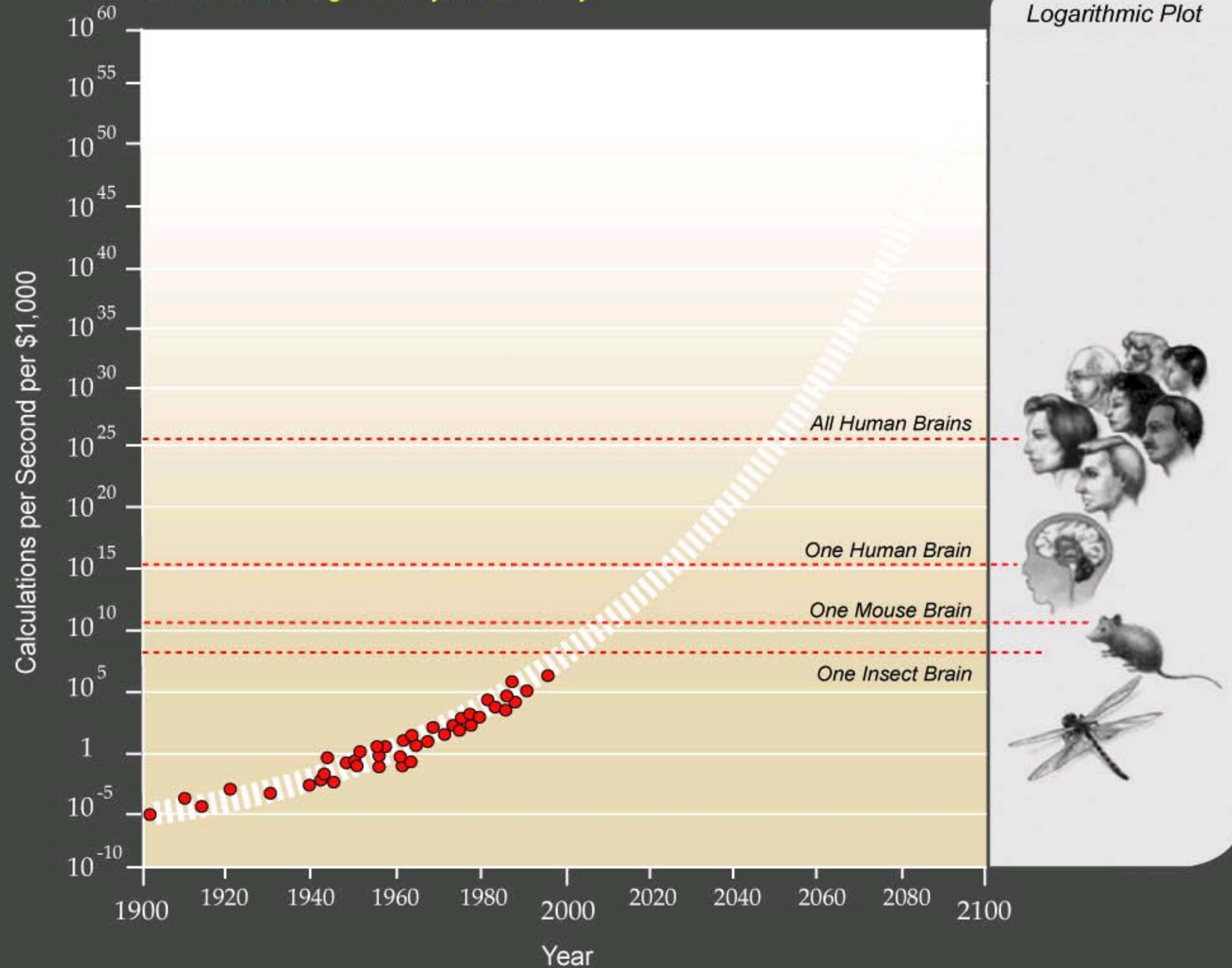
3D perception 2000 in use 2005

2D mapping robots 1990

SRI Shakey Stanford Cart 1970-1980

Hopkins Beast 1960

Grey Walter Tortoise 1950

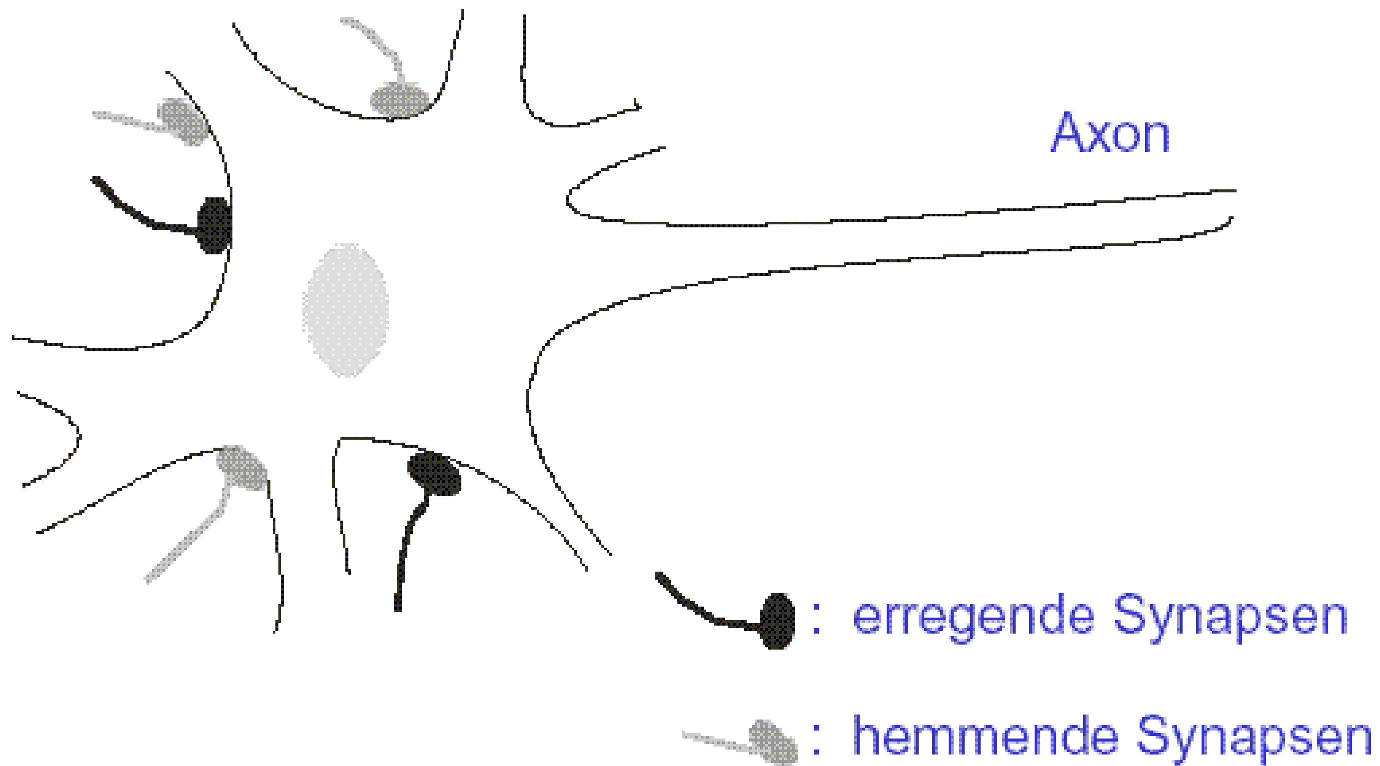


Exponential Growth of Computing
Twentieth through twenty first century
Logarithmic Plot
Calculations per Second per $1,000
10^{60}
10^{55}
10^{50}
10^{45}
10^{40}
10^{35}
10^{30}
10^{25}
10^{20}
10^{15}
10^{10}
10^{5}
1
10^{-5}
10^{-10}
All Human Brains
One Human Brain
One Mouse Brain
One Insect Brain
1900
1920
1940
1960
1980
2000
2020
2040
2060
2080
2100
Year

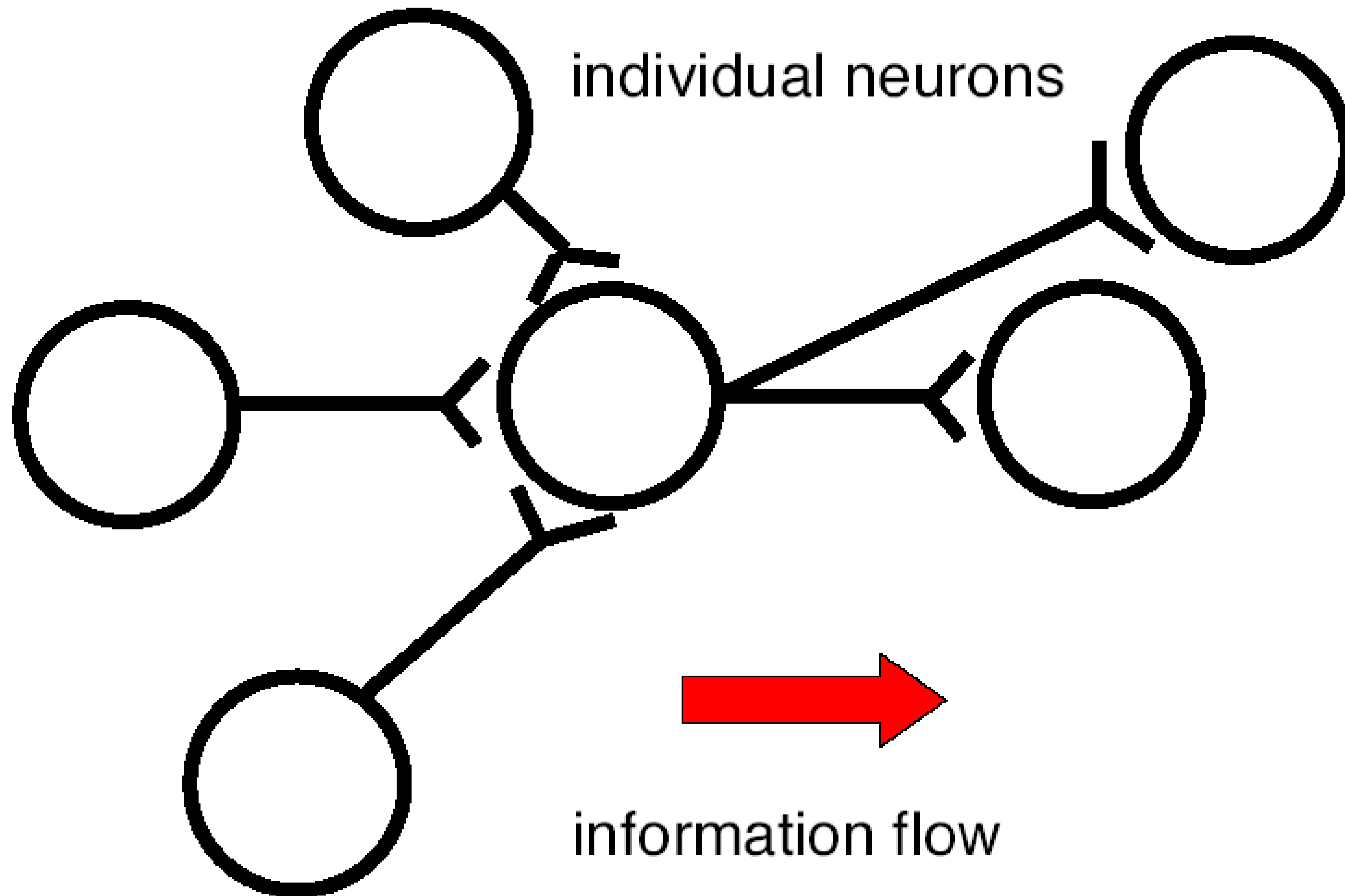
Gehirn - Mikroprozessor

	Gehirn	Mikroprozessor
Zeitskala	ms (10^{-3}s)	ns (10^{-9}s)
Anzahl Prozessoren	10^{10}-10^{14} Neuronen	~10^{9} Transistoren
Parallelität	fein	grob
Konnektivität	10^{3}-10^{5} Synapsen	< 10 direkte Verbindungen
Repräsentation	verteilt	lokal
Zuverlässigkeit	einzelne Neurone sterben, wenig Einfluss auf das System	Transistoren fallen selten aus, Ausfall hat großen Einfluss auf das System
Leistung	<<10^{-6} W/Neuron 10^{2} Watt/Mensch	10^{-1} W low power CPU 10^{4} W Supercomputer

Aufbau eines Neurons



Künstliche Neuronen



McCulloch-Pitts Neuron

inputs

excitatory

x_1 w_1

x_2 w_2 θ output

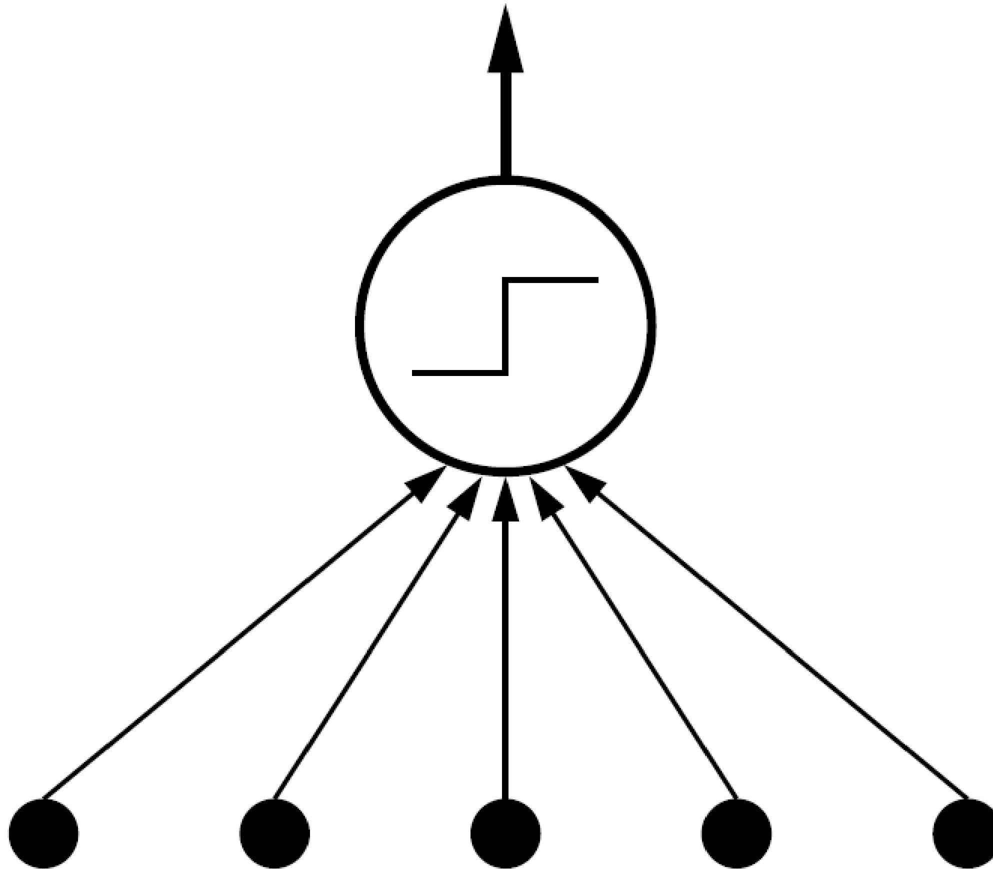
x_3 w_3

inhibitory

$$\text{output} = \begin{cases} 1 \text{ if } w_1 x_1 + w_2 x_2 - w_3 x_3 > \theta \\ 0 \text{ otherwise} \end{cases}$$

$\sum w_i x_i > \theta$
if we allow weights to be *signed*

Perzeptron



Output

Gewichte

Input

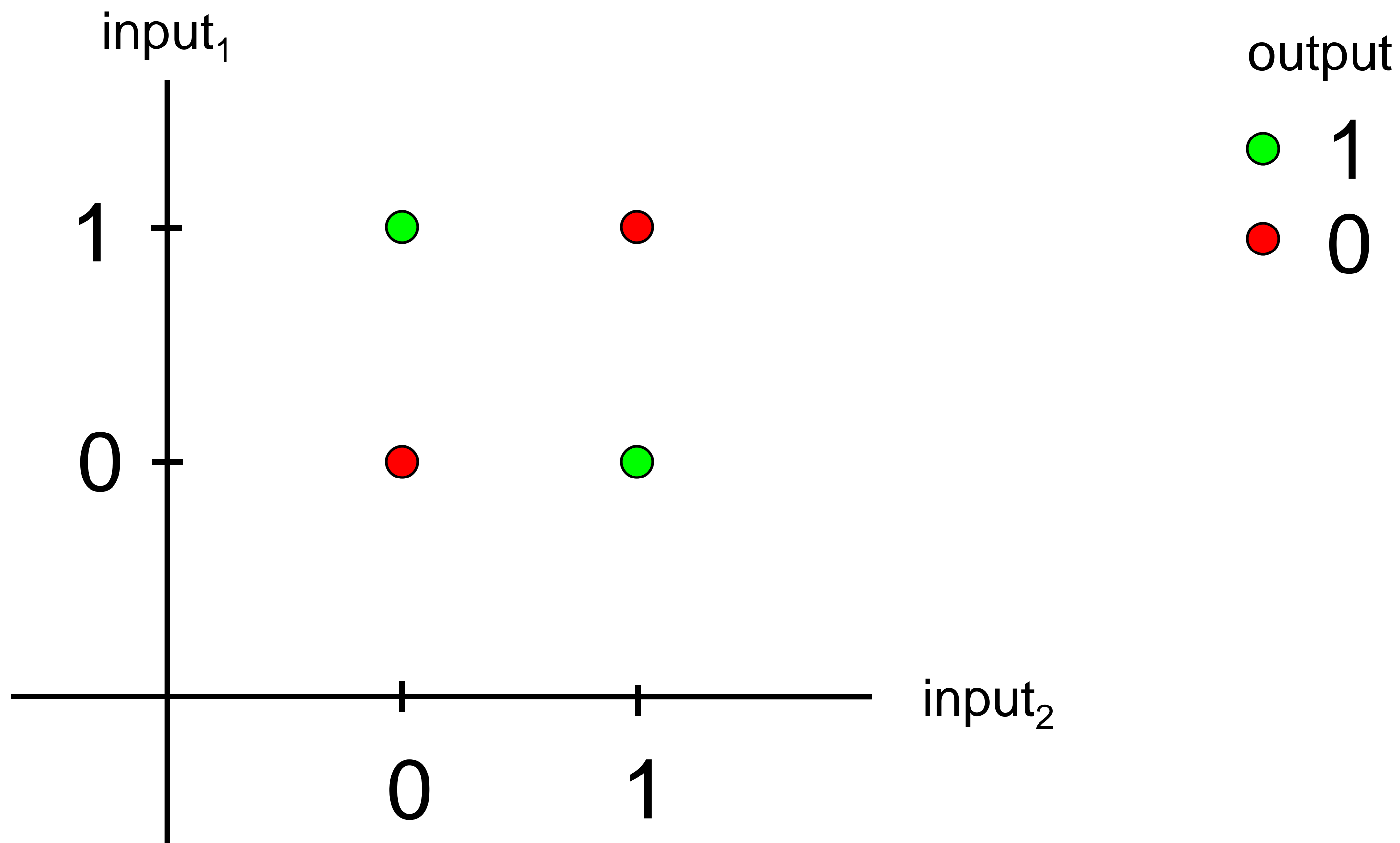
Perzeptron (Demo)

http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletPerceptron.html

Exklusiv-Oder (XOR)

$input_1$	$input_2$	output
1	1	0
1	0	1
0	1	1
0	0	0

XOR-Problem



Mehrschichtiges Perzeptron (Demo)

http://www.eee.metu.edu.tr/~alatan/Courses/Demo/BackPropagation.htm
http://www.sund.de/netze/applets/BPN/bpn2/ochre.html

Historie

1943	McCulloch & Pitts, erste Modelle
1949	Hebb, Postulat des Lernens
1957	Rosenblatt, Perzeptron
1969	Minsky & Papert, Limitierungen des Perzeptrons
1972	Kohonen, selbstorganisierende KNN
1974	Werbos, Lernregel für mehrschichtige Perzeptrons (nicht beachtet)
1986	Rumelhart & McClelland, Popularisierung der Lernregel für MLP (Beginn des Revival)

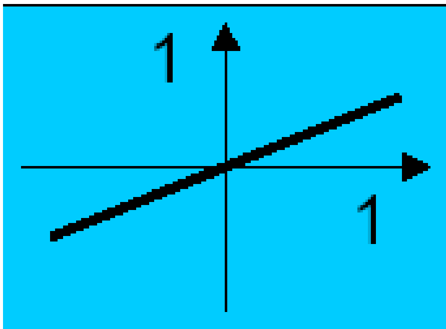
Berechnung der Aktivität

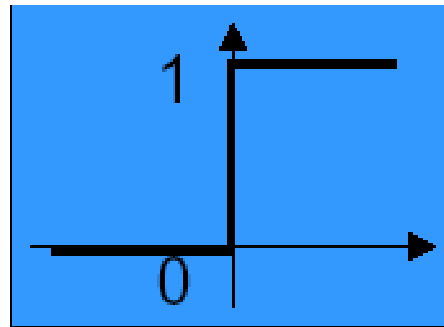
Aktivierung: $a_i = \sum_j w_{ji} x_j - \theta$

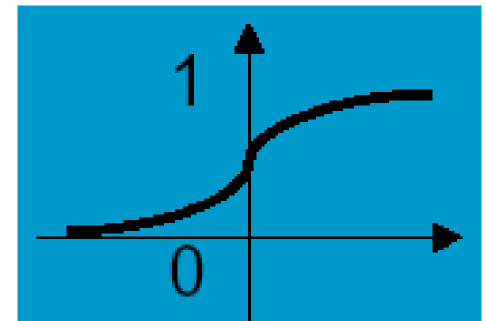
Aktivität: $x_i = f(a_i)$

Transferfunktion: $f(a_i)$

Transferfunktionen





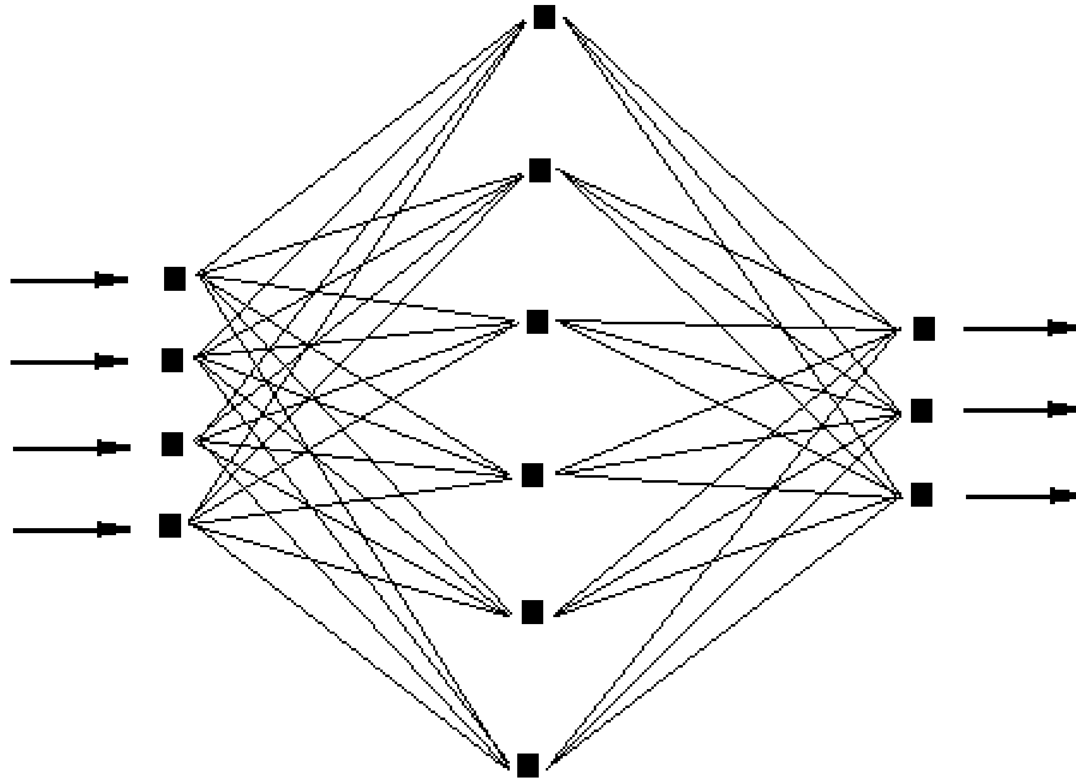


linear

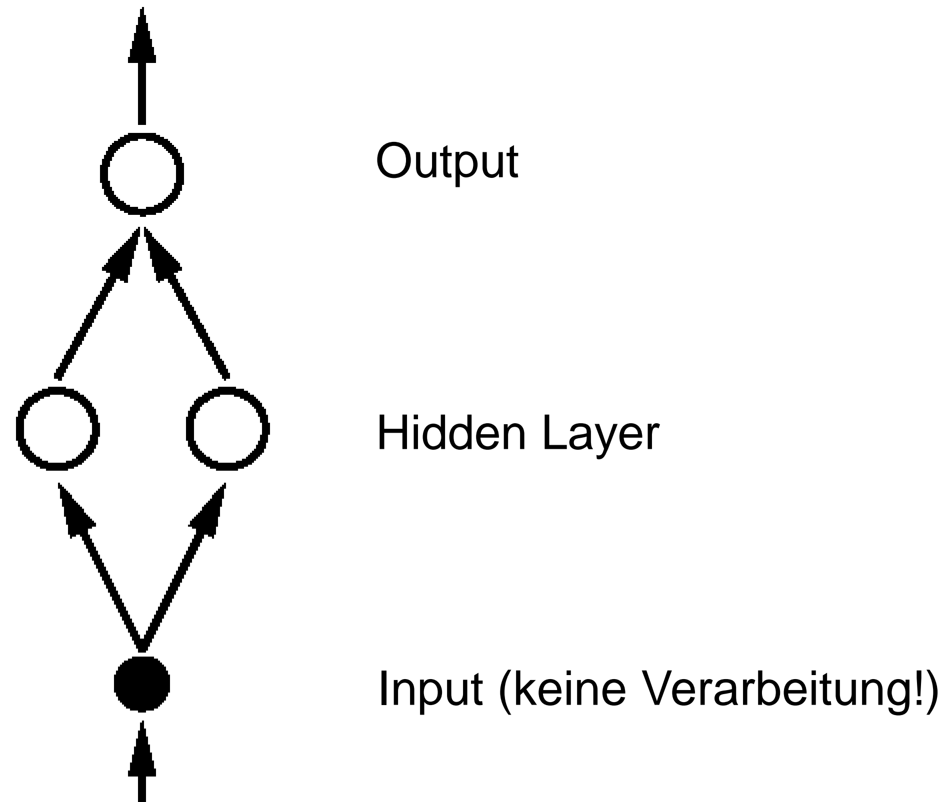
Sprungfunktion

Sigmoidfunktion

Mehrschichtiges Perzeptron (MLP)



Mehrschichtiges Perzeptron



Training
(Back-Propagation-Algorithmus)

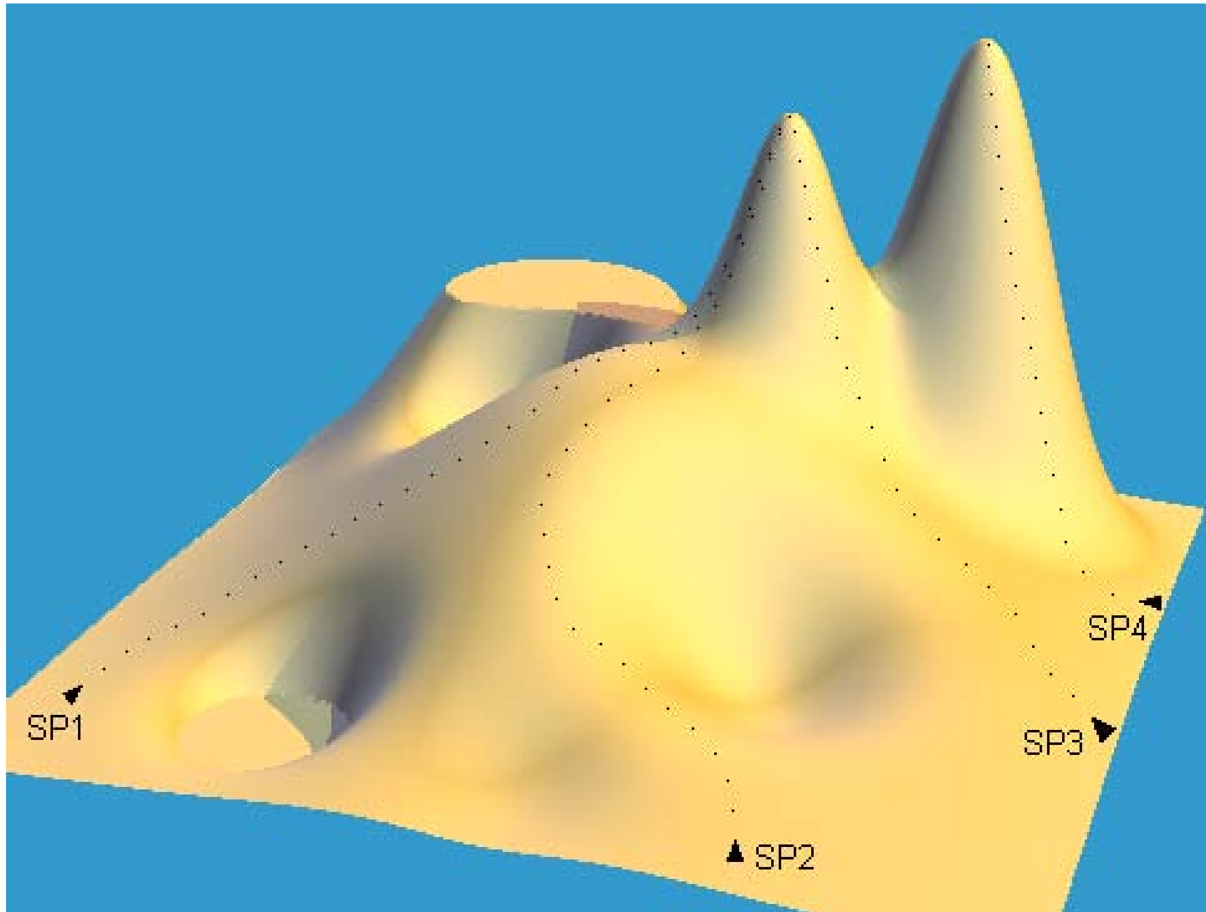
Partielle Ableitung

Partielle Ableitungen ermöglichen die Berechnung einer Lösung für Probleme, die von mehreren Parametern abhängen.

Partielle Ableitungen bilden die Komponenten des **Gradienten**.

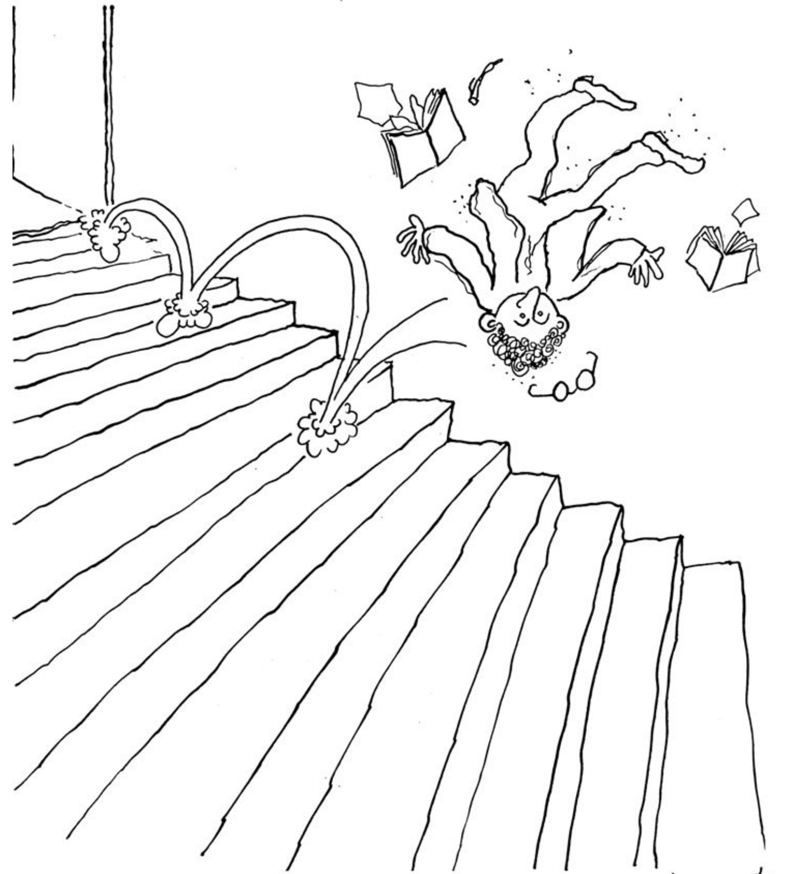
Ein **Gradient** gibt die Änderung einer Größe zwischen räumlich oder zeitlich definierten Punkten an, er ist ein Vektor und zeigt in Richtung des größten Gefälles.

Gradientenverfahren



Quelle: http://www.statistics4u.info/fundstat_germ/cc_optim_meth_gradient.html

Gradientenverfahren



Quelle: http://www.eurasip.org/DSPHumour/steepest-descent.jpg

Gradientenabstieg

Kettenregel

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

Hängt f nur *über* g von x ab, so ist die Ableitung von f nach x gleich dem Produkt aus der Ableitung von f nach g (wobei g als Variable behandelt wird) und der Ableitung von g nach x.

Anwendung Kettenregel

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

$$f(x) = 2(3 - x^2)^3 \qquad g(x) = 3 - x^2 \qquad f(g) = 2g^3$$

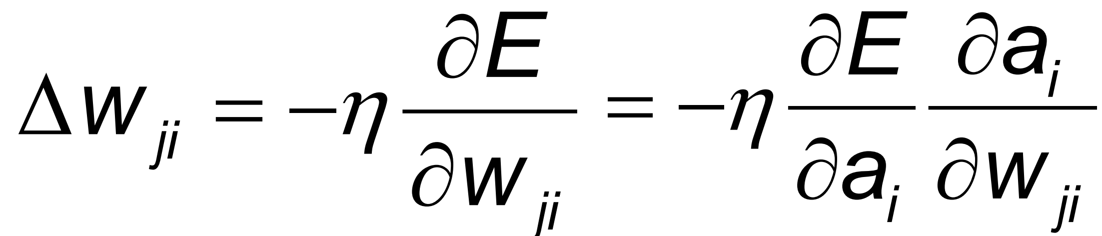
$$\frac{df}{dx} = 6g^2 \cdot (-2x) = 6(3 - x^2)^2(-2x)$$

Back-Propagation I

$$E = \frac{1}{2} \sum_i e_i^2$$

$$e_i = \begin{cases} z_i - x_i, i \in \mathbb{Z} \\ 0, sonst \end{cases}$$

Anwendung der Kettenregel



$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial w_{ji}}$$

Back-Propagation II

$$
\begin{aligned}
a_i &= \sum_{j=1}^{N} w_{ji} x_j - \theta \\
&= \sum_{j=1}^{N} w_{ji} x_j + w_{0i} x_0 \ , \text{ wobei } \quad x_0 \equiv 1 \\
&= \sum_{j=0}^{N} w_{ji} x_j
\end{aligned}
$$

$$\frac{\partial a_i}{\partial w_{ji}} = x_j$$

Back-Propagation III

$$\delta_i \equiv \frac{\partial E}{\partial a_i} \qquad \frac{\partial a_i}{\partial w_{ji}} = x_j$$

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial w_{ji}} = -\eta \cdot \delta_i \cdot x_j$$

δ_i^{inj} : injizierte Fehler $\qquad \delta_j^{imp}$: implizite Fehler

Injizierte Fehler

$$\delta_i^{inj} = \frac{\partial E}{\partial a_i} = \frac{\partial E}{\partial x_i}\frac{\partial x_i}{\partial a_i}$$

$$\frac{\partial E}{\partial x_i} = \frac{\partial(1/2\sum_i (z_i - x_i)^2)}{\partial x_i} = -1 \cdot (z_i - x_i)$$

$$\frac{\partial x_i}{\partial a_i} = \frac{\partial(f(a_i))}{\partial a_i} = f'(a_i)$$

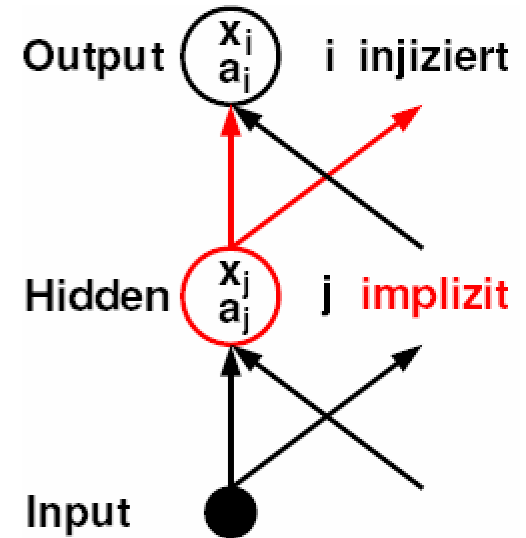
$$\delta_i^{inj} = -f'(a_i) \cdot (z_i - x_i)$$

Implizite Fehler

$$\delta_j^{imp} = \frac{\partial E}{\partial a_j} = \sum_{i \in \mathbb{N}_j} \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial a_j} = \sum_{i \in \mathbb{N}_j} \delta_i \frac{\partial (w_{ji} x_j)}{\partial a_j}$$

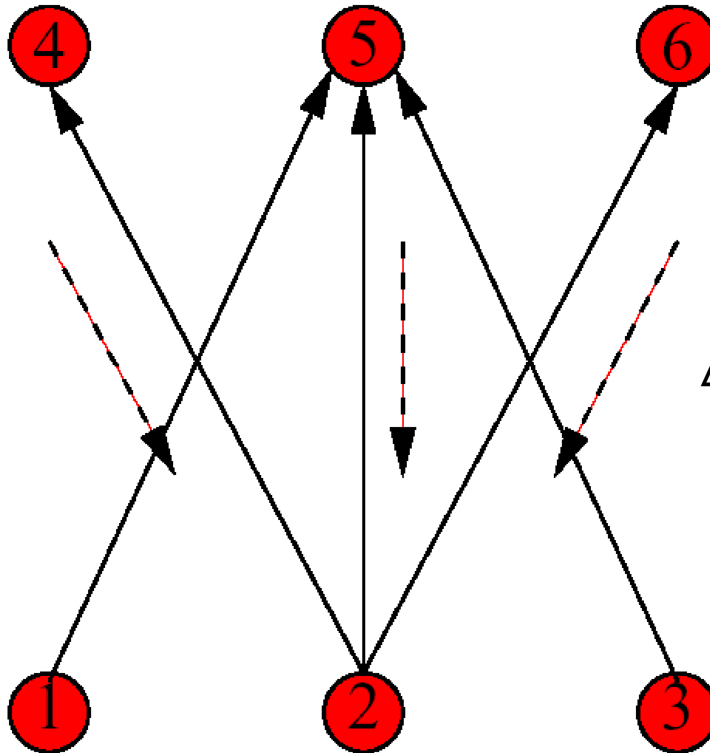
$$= \sum_{i \in \mathbb{N}_j} \delta_i \frac{\partial (w_{ji} f(a_j))}{\partial a_j}$$

$$= f'(a_j) \sum_{i \in \mathbb{N}_j} w_{ji} \delta_i$$



Zusammenfassung

$$x_5 = f(w_{15}x_1 + w_{25}x_2 + w_{35}x_3)$$



$$\Delta w_{25} = -\eta \cdot \delta_5 \cdot x_2$$

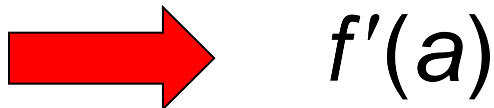
$$\delta_2 = f'(a_2) \cdot (w_{24}\delta_4 + w_{25}\delta_5 + w_{26}\delta_6)$$

Hebbsche Lernregel

Das Grundprinzip der meisten gebräuchlichen Lernverfahren geht auf eine Beobachtung von biologischen Neuronen durch Hebb (1949) zurück. Hebb stellte fest, dass Adaption im Modell dadurch erreicht werden kann, wenn man das Gewicht zwischen zwei stark feuernden Units vergrößert. Das gleichzeitige Feuern deutet auf eine Korrelation zwischen den beiden Units hin, daher wird die gegenseitige Beeinflussung vergrößert.

Ableitung Transferfunktion

Was fehlt noch für die vollständige Berechnung des Gradienten?



$$f'(a)$$

Transferfunktion

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

Ableitung Sigmoidfunktion

$$f(a) = \frac{1}{1+e^{-a}} \qquad \left(\frac{f}{g}\right)' = \frac{gf' - fg'}{g^2} \text{ (Quotientenregel)}$$

$$\left(e^{-a}\right)' = \left(\left(e^{a}\right)^{-1}\right)' = -1 \bullet \left(e^{a}\right)^{-2} \bullet e^{a} = -e^{-a} \text{ (Kettenregel)}$$

$$f'(a) = \frac{0-(-e^{-a})}{(1+e^{-a})^2} = \frac{1+e^{-a}-1}{(1+e^{-a})^2}$$

$$= \frac{1+e^{-a}}{(1+e^{-a})^2} - \frac{1}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\left(1-\frac{1}{1+e^{-a}}\right)$$

$$= f(a) \bullet (1-f(a)) = x(1-x)$$

Zusammenfassung

$$\Delta w_{ji} = -\eta \cdot \delta_i \cdot x_j$$

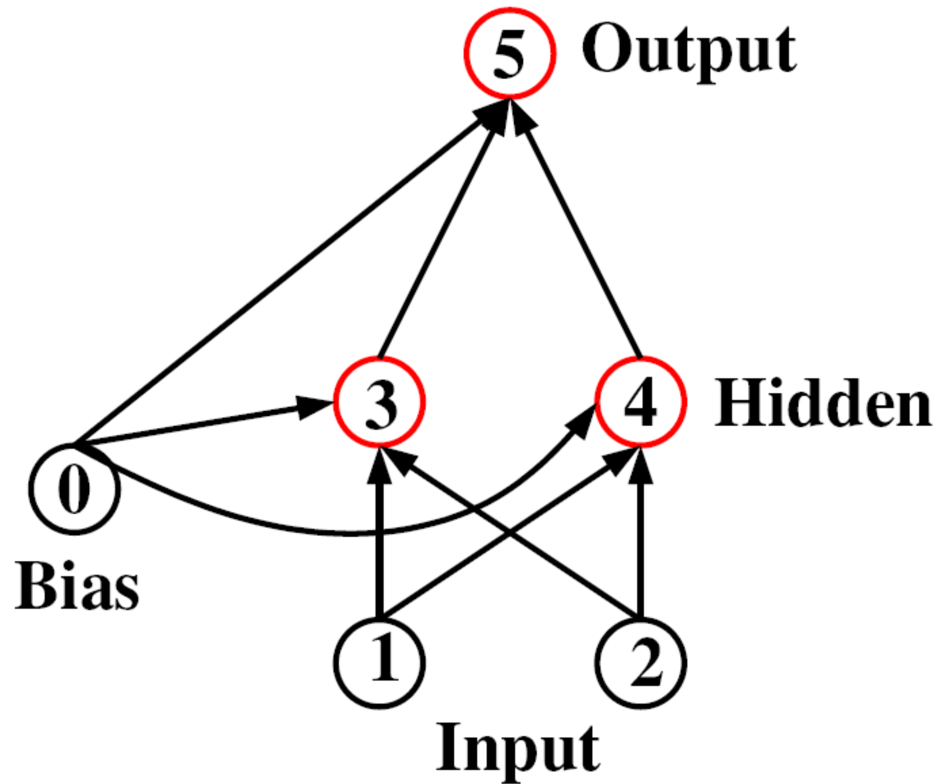
$$\delta_i^{inj} = -f'(a_i) \cdot (z_i - x_i)$$

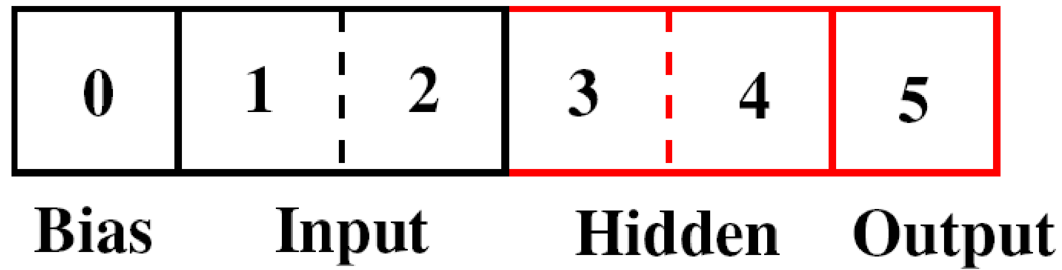
$$\delta_j^{imp} = f'(a_j) \sum_{i \in \mathbb{N}_j} w_{ji} \delta_i$$

Sigmoidfunktion:

$$f'(a_i) = x_i(1 - x_i)$$

Implementierung MLP



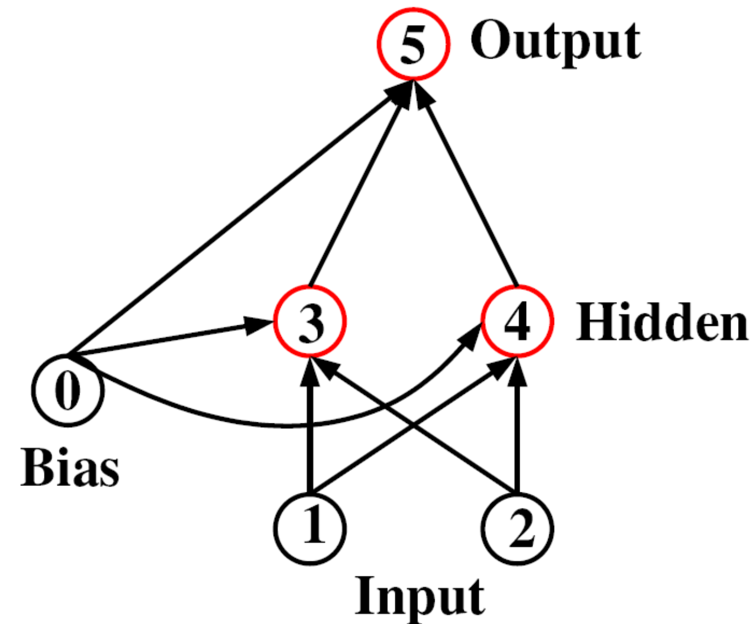


Berechnung Forward (propagate())

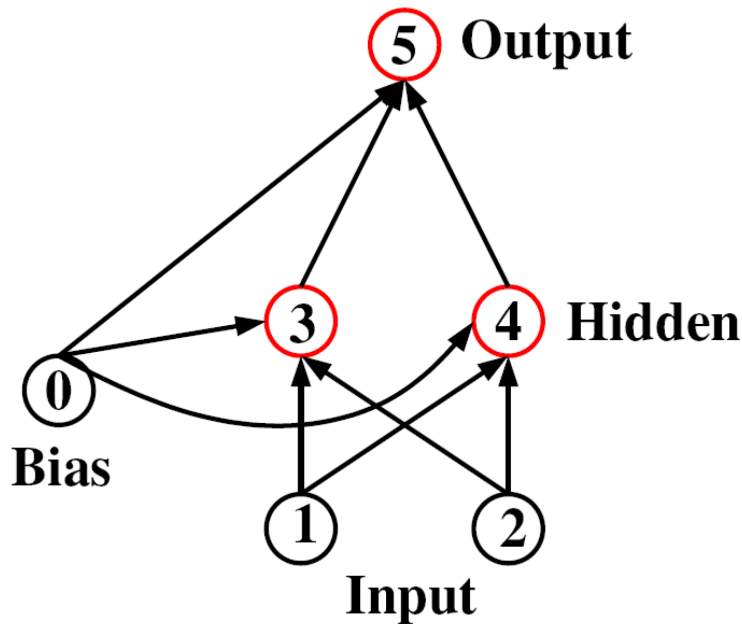
```
// set Input
for(i=1;i < StartHidden;i++)
    Activity[i] = Pattern[ActPattern][i]; // !!i

// compute Input ⇒ Hidden
for(i=StartHidden;i < StartOutput;i++) {
    Activation = Activity[0] * Weight[0][i]; // Threshold
    for(j=1;j < StartHidden;j++)
        Activation += Activity[j] * Weight[j][i];
    Activity[i] = sigmoid(Activation); // Transfer Function
}

// compute Hidden ⇒ Output
for(i=StartOutput;i < NoNeuron;i++) {
    Activation = Activity[0] * Weight[0][i]; // Threshold
    for(j=StartHidden;j < StartOutput;j++)
        Activation += Activity[j] * Weight[j][i];
    Activity[i] = sigmoid(Activation); // Transfer Function
}
```



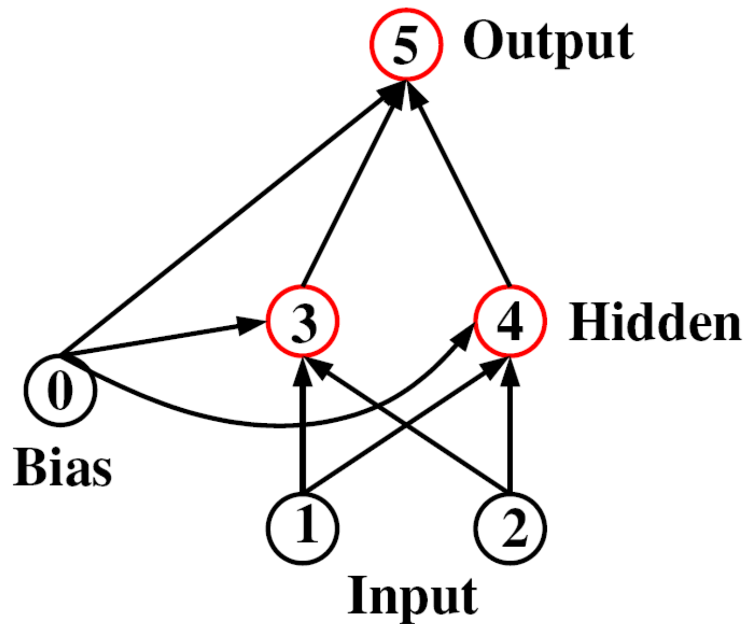
back_propagate - injizierter Fehler



$$\delta_i^{inj} = -f'(a_i)\cdot(z_i - x_i)$$

```
// Delta Output (injizierter Fehler)
for(i=StartOutput;i < NoNeuron;i++) {
    Delta[i]  = (-1.) * (Target[Pattern][i] - Activity[i]);  // !!i
    Delta[i] *= Activity[i] * (1. - Activity[i]);  // Ableitung Sigmoid
}
```

back_propagate() – impliziter Fehler



$$\delta_i^{imp} = f'(a_i) \sum_{j \in \mathbb{N}_i} w_{ij} \delta_j$$

```
// Delta Hidden (impliziter Fehler)
for(i=StartHidden;i < StartOutput;i++) {
    for(j=StartOutput, Delta[i] = 0.;j < NoNeuron;j++)
        Delta[i] += Weight[i][j] * Delta[j];
    Delta[i] *= Activity[i] * (1. - Activity[i]);  // Ableitung Sigmoid
```

back_propagate() - Gradient

$$\Delta w_{ij} = -\eta \cdot \delta_j \cdot x_i$$

```
// Bias
for(j=StartHidden;j < NoNeuron;j++)
    DeltaWeight[0][j] += (Activity[0] * Delta[j]) // Hebb
// Input ⇒ Hidden
for(i=1;i < StartHidden;i++)
    for(j=StartHidden;j < StartOutput;j++)
        DeltaWeight[i][j] += (Activity[i] * Delta[j]); // Hebb
// Hidden ⇒ Output
for(i=StartHidden;i < StartOutput;i++)
    for(j=StartOutput;j < NoNeuron;j++)
        DeltaWeight[i][j] += (Activity[i] * Delta[j]); // Hebb
```

Initialisierung der Gewichtsmatrix

Bei Verwendung einer quadratischen Gewichtsmatrix werden alle nicht existierenden Gewichtswerte auf Null gesetzt.

Alle existierenden Gewichte werden mit zufälligen Werten initialisiert. Hierbei hat sich eine Initialisierung im Wertebereich [-0.5,0.5] bewährt.

Beschleunigung des Trainings

- Die Durchführung eines Trainingszyklus ist für große Netzwerke und große Datenmengen sehr aufwendig.
- Diverse Ansätze zur Beschleunigung des Trainings.
- Verwendung von Näherungen der zweiten Ableitung und Adaption der Lernrate.

Momentum Term I

Mit den Ziel die Konvergenz des Trainings zu steigern wird häufig ein sogenannter Momentum Term eingeführt.

$$\Delta w_{ji}^{n} = -\eta \cdot \frac{\partial E^{n}}{\partial w_{ji}} + \alpha \cdot \Delta w_{ji}^{n-1}$$

wobei $\alpha < 1$ sein muss und n den Update indiziert

Momentum Term II

konstanter Gradient für viele Updates:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} (1 + \alpha + \alpha^2 + ...)$$

$$= -\frac{\eta}{1-\alpha} \frac{\partial E}{\partial w_{ji}}$$

In monotonen Fehlerebenen kann das Training stark beschleunigt werden.

Durchführung Update
update_weight()

```
for(i=0;i < NoNeuron;i++) {
    for(j=0;j < NoNeuron;j++) {
        Update   = (-1.) * Eta * DeltaWeight[i][j]; // Schritt gemäß Gradient
        Update += Alpha * OldUpdate[i][j]; // Momentum Term
        Weight[i][j] += Update; // Gewicht wird geändert
        OldUpdate[i][j] = Update; // Speicherung des aktuellen Update
        DeltaWeight[i][j] = 0.; // Reset Gradient
    }
}
```

Durchführung des Trainings

- **Batch Update:** Update der Gewichte nach der Berechnung und Addition der Gewichtsänderungen für alle Trainingsmuster (hohe Stabilität)
- **Single Update:** Update der Gewichte nach jedem Muster (Beschleunigung)
- **Validierung:** Überprüfung des Lernerfolgs an einer Validierungsmenge (ungleich Testmenge)

Batch Update / Single Update

```
for(i=0;i < NbrCycle;i++) {
    for(j=0;j < NbrPattern;j++) {
        propagate(j);
        back_propagate(j);
        if(SingleUpdate) {
            update_weight();
            reset_delta();
        }
    }
    if(BatchUpdate) {
        update_weight();
        reset_delta();
    }
}
```

Berechnung Fehler

```
// Schleife über alle Pattern der Validierungsmenge
for (i=0, Error=0.0;i < NoPattern;i++) {
    propagate(Pattern[i]); // forward Berechnung
    for(j=StartOutput;j < NoNeuron;j++)
        Error += pow((Target[i][j]-Activity[j]),2); // !!j
}
```

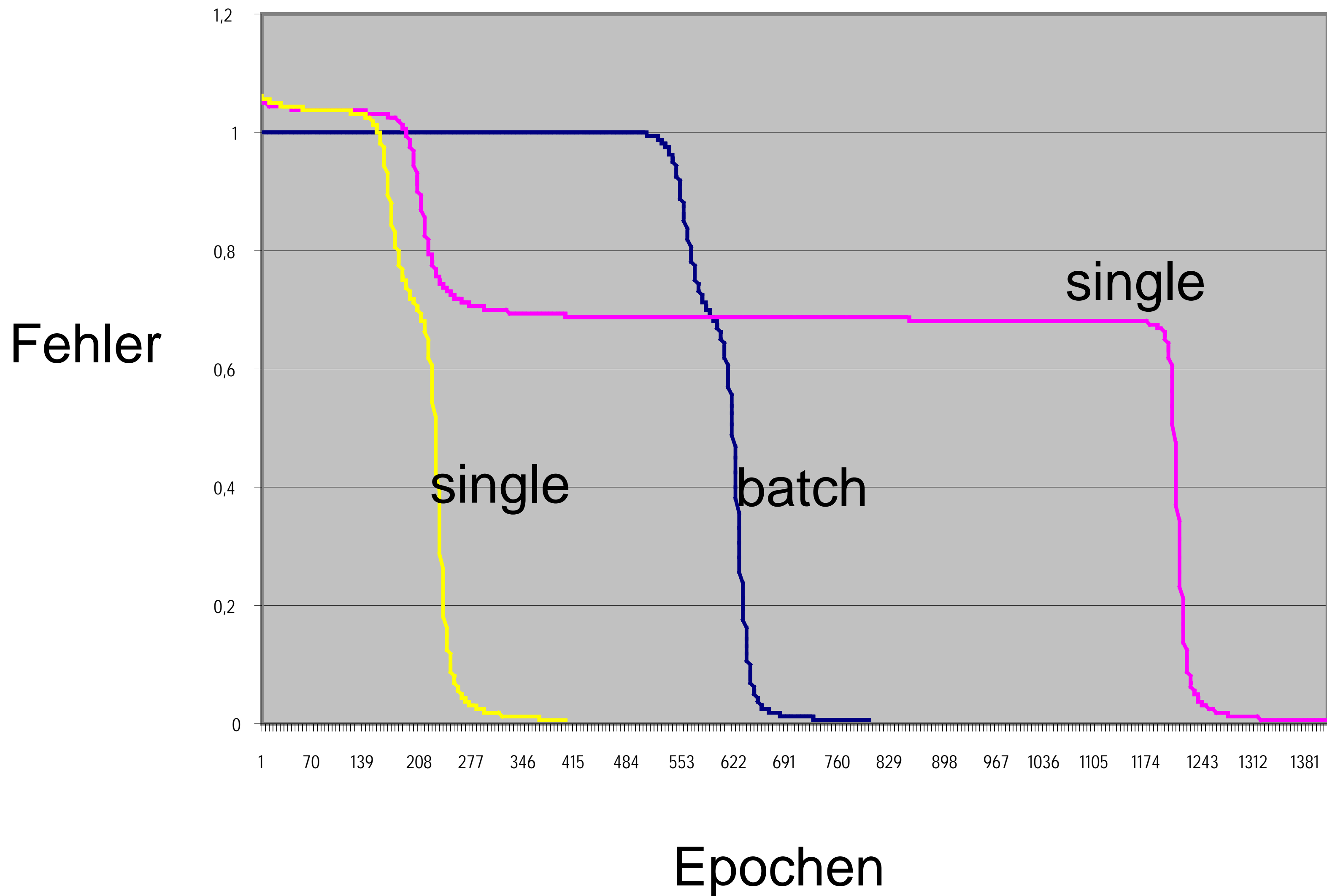
Lösung des XOR-Problems mit MLP

Hidden	2	2	2	2	4	4	2	2
Threshold	ja	ja	ja	nein	nein	nein	ja	ja
Eta	0.5	0.05	0.9	0.8	0.8	0.8	0.5	0.9
Alpha	0.9	0.9	0.0	0.9	0.9	0.9	0.9	0.0
Update	batch	batch	batch	batch	batch	single	single	single
Zyklen	500	4000	2000	1000 (10%)	500	300 (90%)	300 (95%)	1000 (95%)

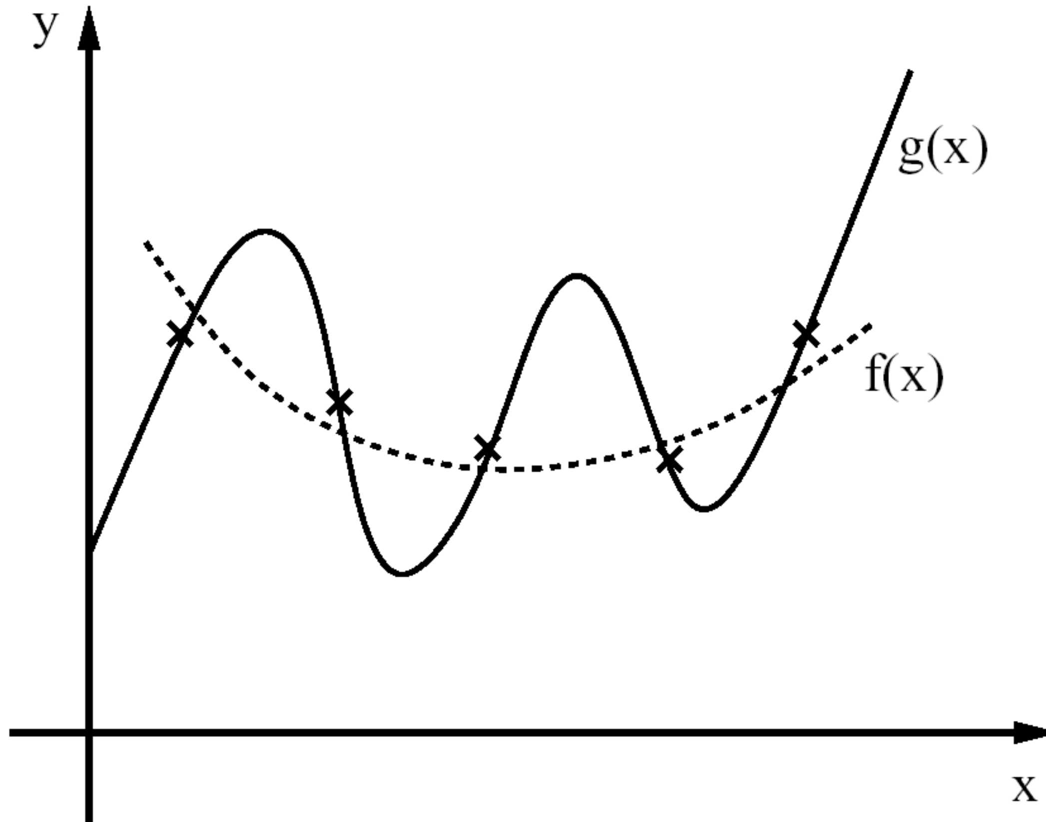
Abbruchbedingung: $E^2 < 0.1$ (Summe der Fehler für alle Muster)

Maximale Anzahl Epochen: 20.000

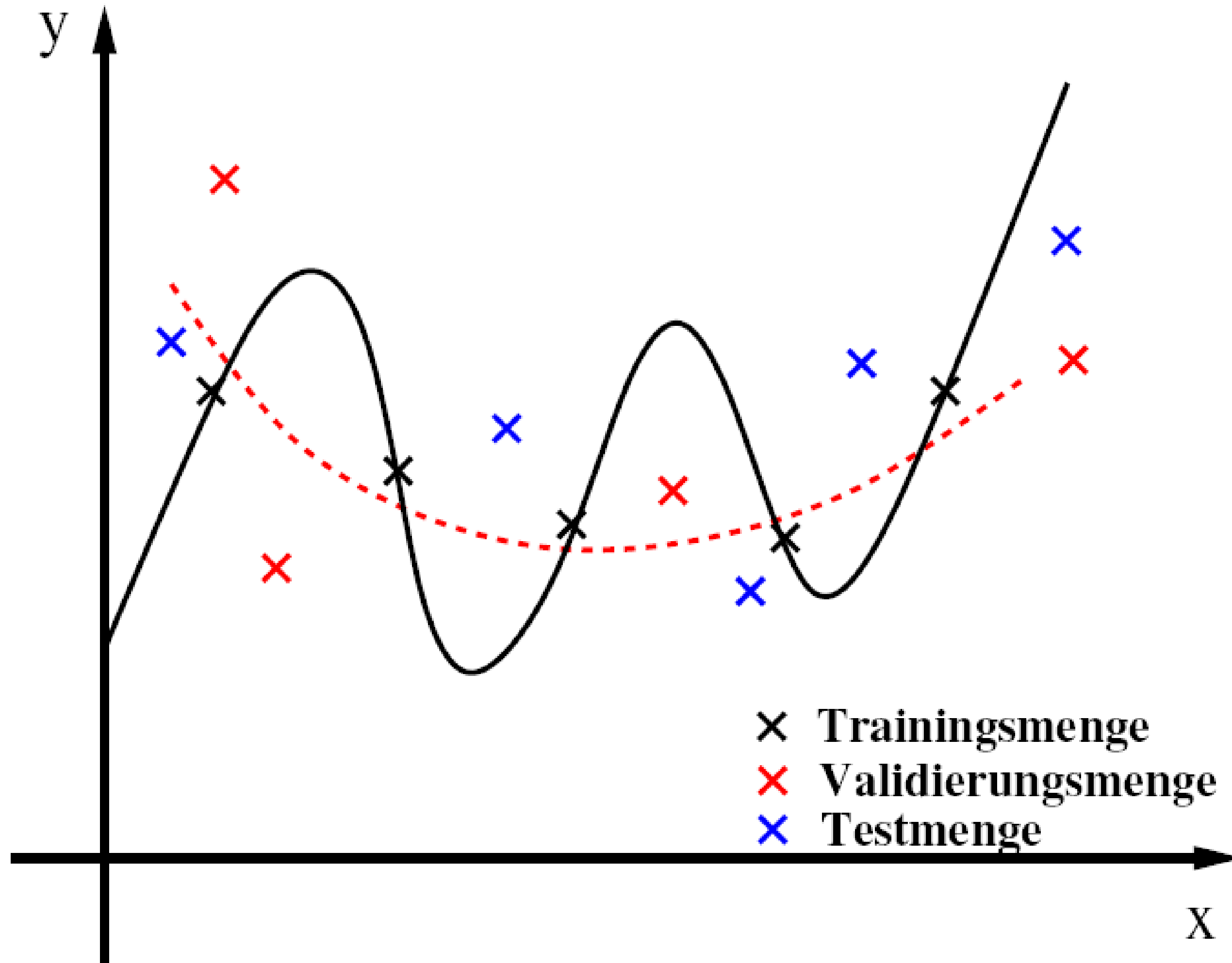
Training XOR



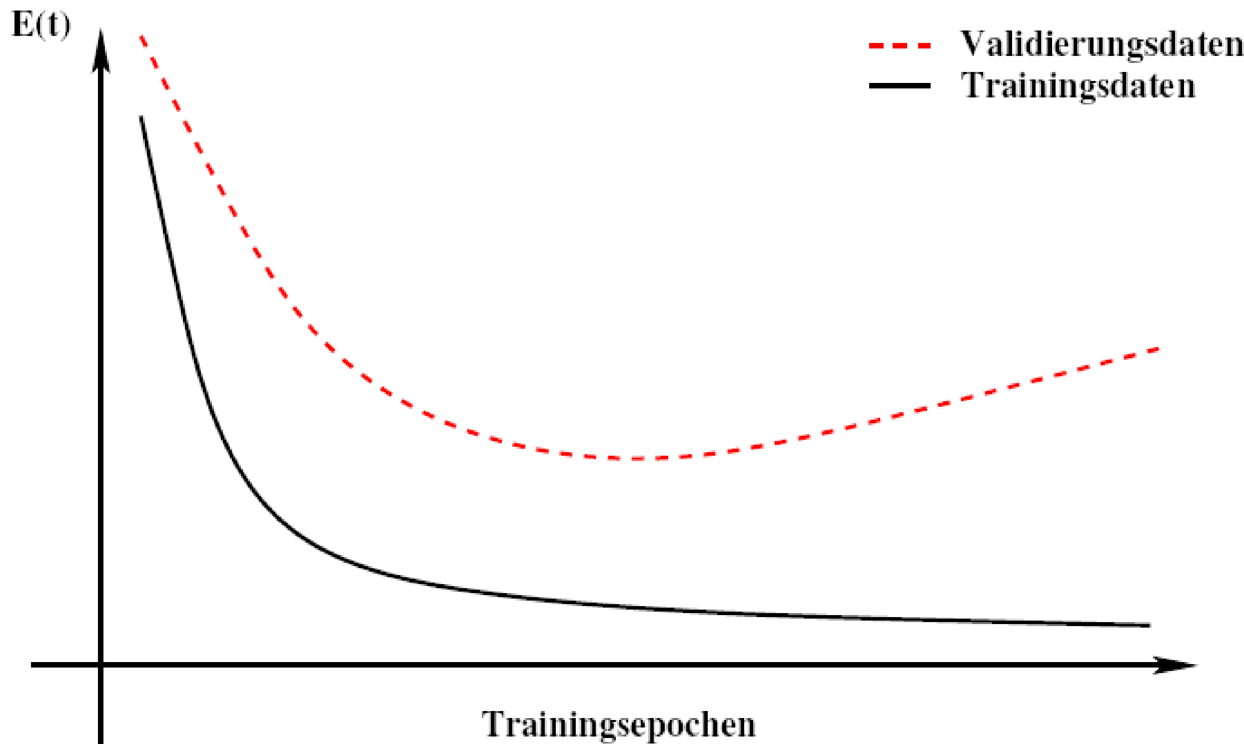
Overfitting

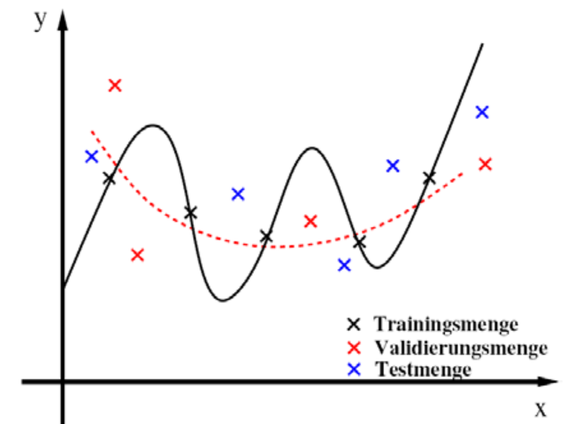


Validierung



Überwachung des Lernerfolgs





Drei Datensätze

- Trainingsdaten: Die Muster der Datenmenge werden für die Adaption der Gewichte verwendet.
- Validierungsdaten: Der berechnete Fehler für die Muster der Datenmenge wird für die Auswahl des besten Netzes verwendet.
- Testdaten: Die Muster der Datenmenge werden zur Qualitätskontrolle verwendet. Der Erfolg für die Testdaten ist ein Indikator für die Leistungsfähigkeit im realen Einsatz.

Potential MLP

- Datengetriebenes Verfahren, d.h. zur Lösung des Problems genügt die Sammlung und Parametrisierung hinreichend vieler repräsentativer Daten.
- Insbesondere ist keine Algorithmisierung der spezifischen Problemlösung erforderlich.
- Approximation beliebiger funktionaler Zusammenhänge.
- Realisierung eines optimalen Klassifikators.

Einsatz von MLP

- Musterklassifikation (z.B. XOR) (http://www.sund.de/netze/applets/BPN/bpn2/ochre.html)
  - Auf der Basis von Mustern wird die Klassenzugehörigkeit ermittelt (1 aus N-Codierung)
- Funktionsapproximation (http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html)
  - Funktionswerte werden auf der Basis von Funktionsargumenten ermittelt.
- Prädiktion (http://diwww.epfl.ch/mantra/tutorial/english/predic/html/index.html)
  - Zustände (z.B. Funktionswerte) werden auf der Basis vergangener Zustände prädiziert.

Einschätzung MLP

- Ist der Lernerfolg eines MLP (Bsp. XOR) von der Reihenfolge der Präsentation der Muster abhängig?
- Ist die Bewertung des *n-ten* Musters abhängig von der Bewertung des (*n-1*)-ten Musters?
- Kann ein MLP ein Gedächtnis ausbilden?
- Welche Struktur müsste ein Netz prinzipiell haben damit zeitliche Abhängigkeiten modelliert werden können?

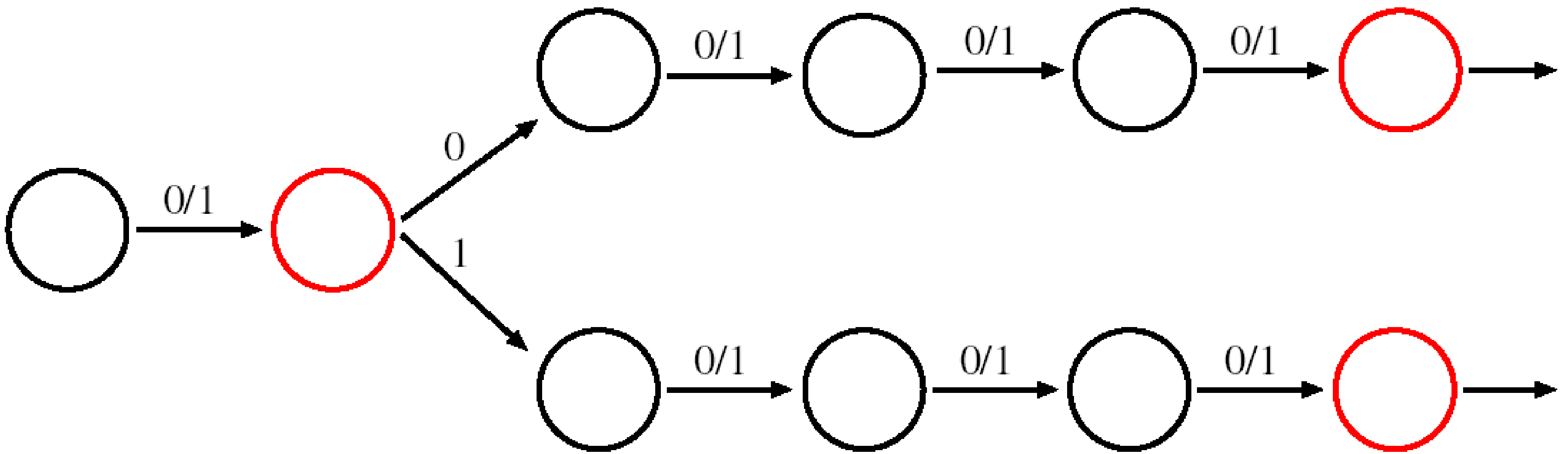
Gedächtnis

- Arbeitsgedächtnis (Kurzzeitgedächtnis)
  - begrenzte Kapazität (7 +/-2 Einheiten)
  - vergessen erfolgt durch Überschreiben
  - zur längerfristigen Speicherung Überführung ins Langzeitgedächtnis
- Langzeitgedächtnis
  - dauerhaftes Speichersystem
  - neues Einspeichern von Informationen wird als Lernen bezeichnet
  - Üben ist für Lernen grundlegend

Lernen

- Memorieren
  - auswendig lernen
  - Informationen abspeichern und bei Bedarf abrufen
  - kleine Abweichungen führen zu einer falschen Zuordnung
- Generalisieren
  - Regelhaftigkeit wird extrahiert
  - Muster werden auf Grund bestimmter Eigenschaften zugeordnet
  - gestörte Muster können mit Hilfe der gelernten strukturellen Zusammenhänge korrekt zugeordnet werden

Automat



Ereignis

Klassifikation

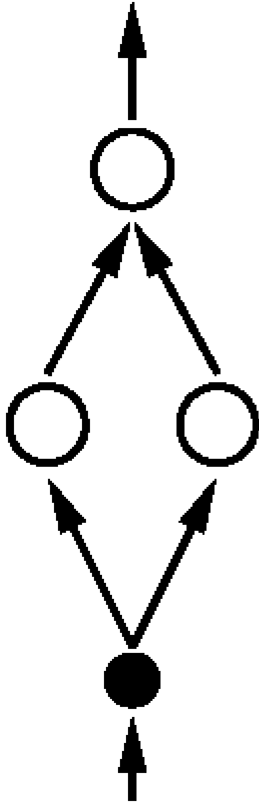
Limitationen MLP

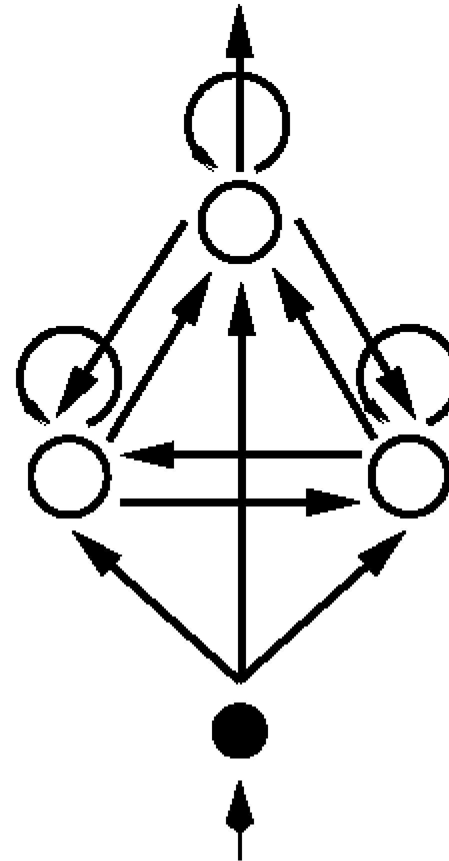
- Zeitliche Abhängigkeiten können nicht modelliert werden.
- Muster unterschiedlicher Länge können nicht verarbeitet werden.
- Die Länge des Arbeitsgedächtnisses muss explizit festgelegt werden.

Motivation für rekurrente KNN

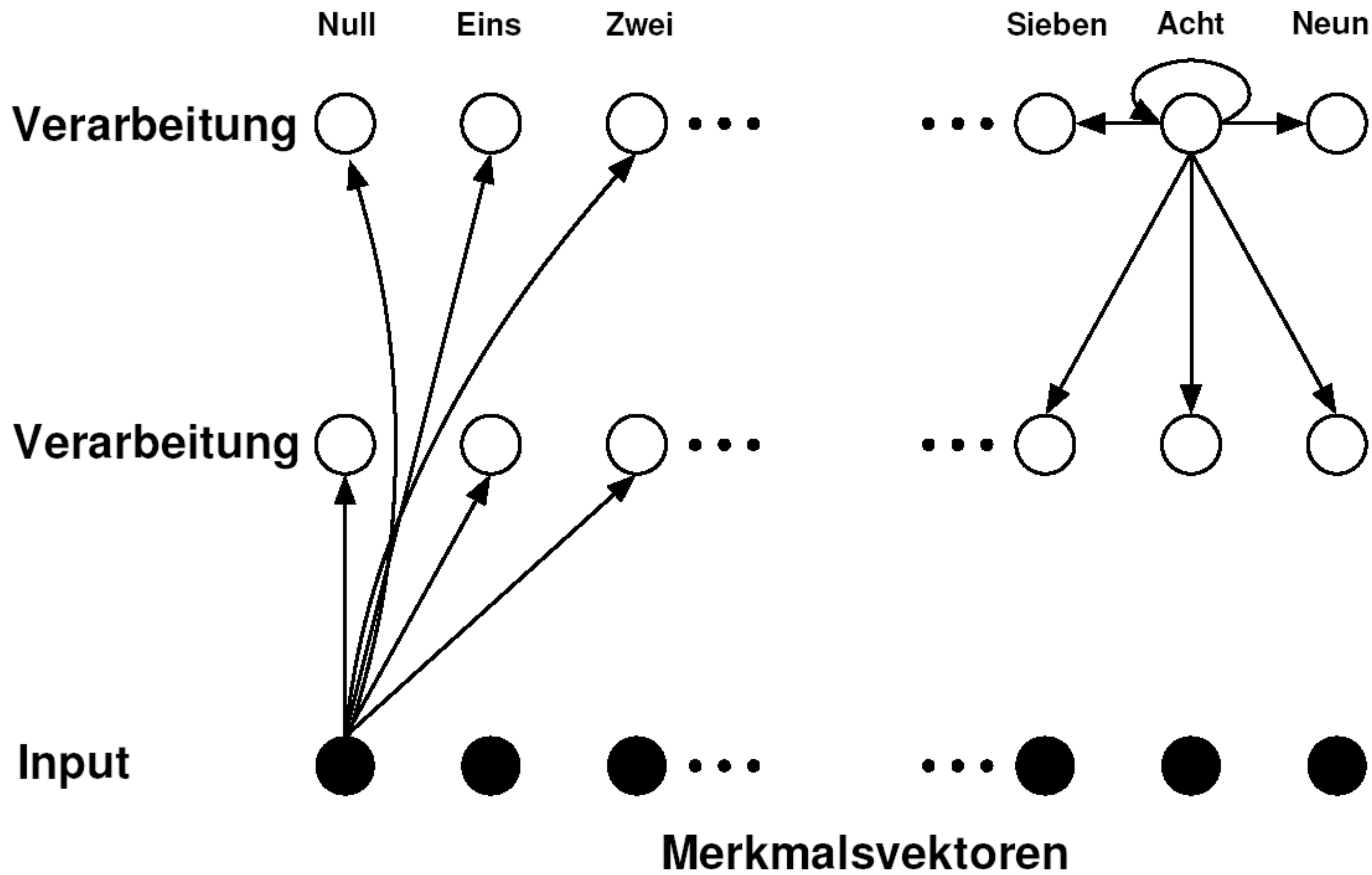
- Realisierung von Modellen, die zeitliche Abhängigkeiten berücksichtigen
- Insbesondere soll ein Arbeitsgedächtnis modelliert werden
- Muster unterschiedlicher Länge können für die Klassifikation oder Prognose verwendet werden.

KNN mit rekurrenten Verbindungen





RNN



Formale Erweiterung

Aktivierung: $a_i(t) = \sum_j w_{ji} x_j(t-1) - \theta$

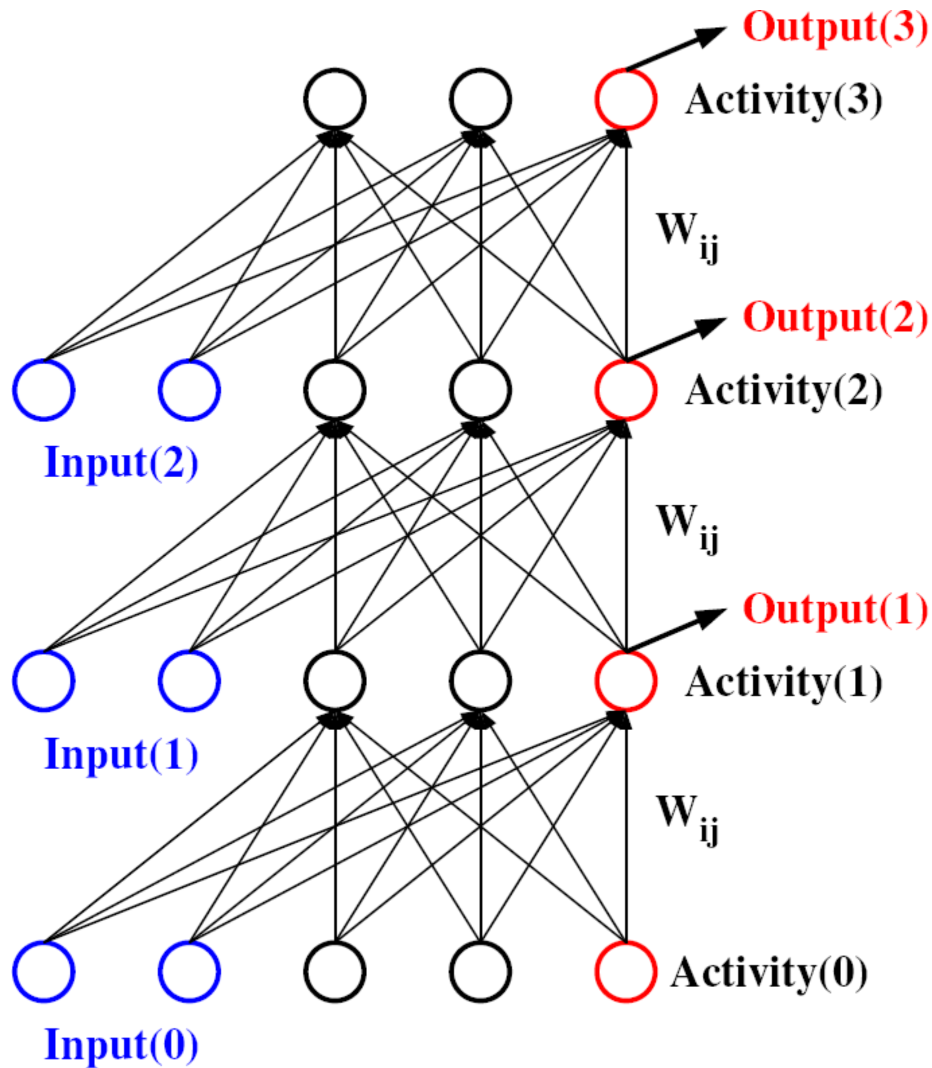
Aktivität: $x_i(t) = f(a_i(t))$

Transferfunktion: $f(a_i(t))$

Back Propagation Through Time (BPTT)

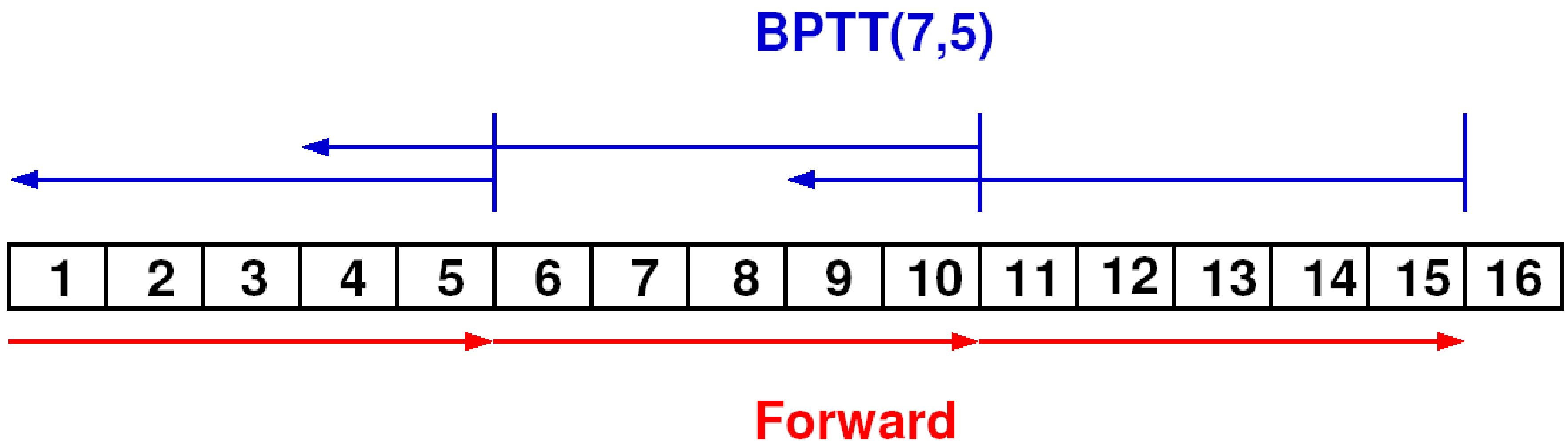
- Das Netzwerk wird in der Zeit entfaltet
- Die einzelnen Zeittakte können wie Schichten eines MLP behandelt werden
- Back Propagation Algorithmus kann angewendet werden

Entfaltung in der Zeit



BPTT(h,h´)

Nach h´ Zeittakten wird der Fehler h Zeittakte zurück propagiert (Williams & Zipser, 1990)



Formeldarstellung BPTT

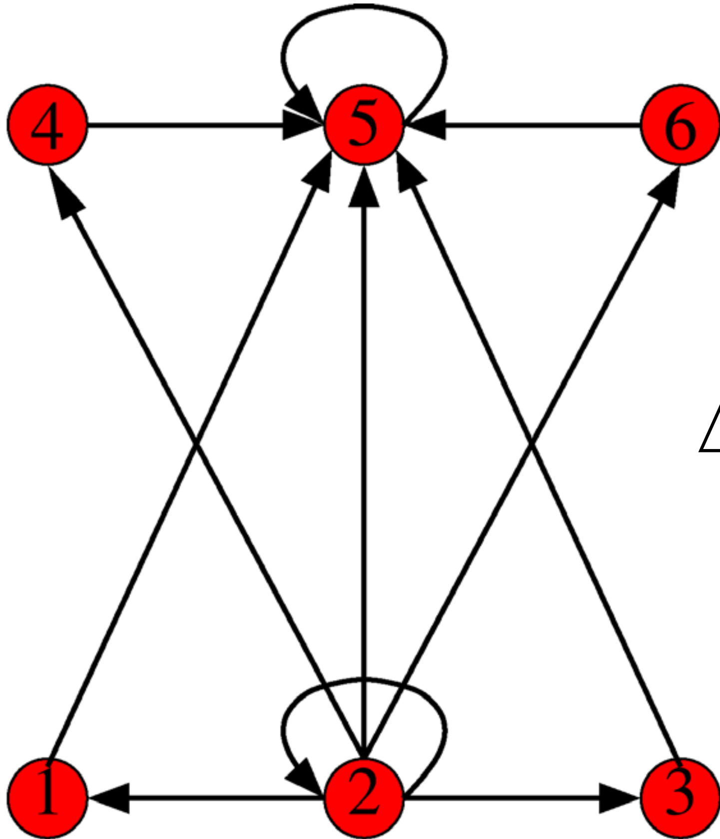
$$\delta_i^{inj}(t) = -f'(a_i(t)) \cdot (z_i(t) - x_i(t))$$

$$\delta_i^{imp}(t) = f'(a_i(t)) \cdot \sum_j w_{ij} \delta_j(t+1)$$

$$\Delta w_{ij} = -\eta \sum_t x_i(t-1) \cdot \delta_j(t)$$

Zusammenfassung BPTT

$$x_5(t) = f(w_{15}x_1(t-1) + w_{25}x_2(t-1) + w_{35}x_3(t-1)$$
$$+w_{45}x_4(t-1) + w_{55}x_5(t-1) + w_{65}x_6(t-1))$$



$$\Delta w_{25} = -\eta \sum_t x_2(t-1) \cdot \delta_5(t)$$

$$\delta_2(t) = f'(a_2(t)) \cdot (w_{21}\delta_1(t+1) + w_{22}\delta_2(t+1) + w_{23}\delta_3(t+1)$$
$$w_{24}\delta_4(t+1) + w_{25}\delta_5(t+1) + w_{26}\delta_6(t+1))$$