Modellbildung und Simulation

Sommersemester 2011

4. 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

Organisation

- Moodle-Server: http://lernen.h-da.de/
- Kurs: Informatik/Informatik (M.Sc.)/Sommersemester 2011/Modellbildung und Simulation – Kasper - SS 2011
- Schlüssel: MuS_KAS_SS2011
- Vorlesungen, Aufgabenblätter und weitere Materialien werden auf dem Moodle-Server zur Verfügung gestellt.
- Fragen und Anmerkungen zur Veranstaltung bitte immer über das Diskussionsforum der Veranstaltung, so dass immer alle informiert sind.
- Wenn Sie interessante Informationen finden, stellen Sie diese bitte auch allen zur Verfügung.

Inhalt

- Wiederholung (VaR)
  - Multivariate Normalverteilung
  - Historische Simulation
  - Monte Carlo Simulation
- Künstliche Neuronale Netze (KNN)
  - Einführung
  - Mehrschichtiges Perzeptron (MLP)
  - Back-Propagation
  - Implementierung MLP

Value at Risk (VaR)

Erwartungsvektor und empirische Kovarianzmatrix

$$\widehat{\underline{\mu_x}} = \sum_{n=1}^{N} \underline{x}(n)/N$$

$$\widehat{\underline{\underline{\mathrm{cov}}}}_{\underline{x}} = \begin{pmatrix} \mathrm{cov}(x_1,x_1) & \cdot & \cdot & \cdot & \mathrm{cov}(x_1,x_M) \\ \cdot & & \cdot & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & \cdot & & \cdot \\ \mathrm{cov}(x_M,x_1) & \cdot & \cdot & \cdot & \mathrm{cov}(x_M,x_M) \end{pmatrix}$$

$$\mathrm{cov}(x_i,x_j) = \sum_{n=1}^{N} (x_i - \mu_i)(x_j - \mu_j)/(N-1) = \mathrm{cov}(x_j,x_i)$$

Korrelation

Die Korrelation ist ein Maß für die Beziehung zwischen zwei Zufallsvariablen. Hinweis: Eine starke Korrelation lässt keine Aussage über den kausalen Zusammenhang zwischen Zufallsvariablen zu.

Die Komponente $\mathrm{cov}(x_i, x_j)$ der Kovarianzmatrix der multivariaten Normalverteilung einer Modellgröße ist ein Maß für die Korrelation der *i*-ten Komponente der Modellgröße mit der *j*-ten Komponente der Modellgröße.

Berechnung des Korrelationskoeffizienten

$$\rho(x_1, x_2) = \frac{\mathrm{cov}(x_1, x_2)}{\sigma(x_1) * \sigma(x_2)}$$

$\rho(x_1, x_2) = 0$: Unabhängigkeit der beiden Variablen

$\rho(x_1, x_2) = 1$: exakte positive lineare Abhängigkeit

$\rho(x_1, x_2) = -1$: exakte negative lineare Abhängigkeit

Mehrdimensionale Gaußdichte

$$f(\underline{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\underline{\underline{\mathrm{cov}}}_{\underline{x}})}} e^{-\frac{1}{2}(\underline{x}-\underline{\mu}_x)^T (\underline{\underline{\mathrm{cov}}}_{\underline{x}})^{-1}(\underline{x}-\underline{\mu}_x)}$$

Multivariate Normalverteilung (2-dimensional)

Interaktive Demo:
http://www.uni-konstanz.de/FuF/wiwi/heiler/os/vt-binorm3d.html
http://lstat.kuleuven.be/java

Glossar

Modellgröße: Die zu modellierende Größe. Beispiel: Wert bzw. Preis der Bestandteile des Portfolios (Rohdaten). eine sinnvolle Parametrisierung ist die Berechnung relativer Änderungen.

empirisch: Die Prognose wird aus historischen Beobachtungen, Simulationen oder subjektiven Einschätzungen gewonnen.

parametrisch: Die Prognose wird auf der Basis der Annahme des Typs der zu Grunde liegenden Verteilung und der Schätzung der erforderlichen Parameter der Verteilung gewonnen.

Konzepte zur Berechnung des VaR

- Historische Simulation
- Monte Carlo Simulation

Historische Simulation

- Verfahren: historisch/empirisch
- Historische Beobachtungen der Modellgrößen werden gesammelt.
- Aus den historischen Beobachtungen werden Wertänderungen des aktuellen Portfolio berechnet.
- Aus der geordneten (empirischen) Messreihe der berechneten Änderungen des Portfolio wird das Quantil bestimmt.
- Hinweis: Für die Anwendung der historischen Simulation muss keine Annahme über die zu Grunde liegende Verteilung getroffen werden.

Monte Carlo Simulation

- Verfahren: parametrisch/empirisch
- An Hand empirischer Daten werden die Parameter einer zu Grunde gelegten Verteilung der Modellgrößen geschätzt.
- Es werden zufällig Werte (Wertänderungen) erzeugt, die der geschätzten Verteilung gehorchen.
- Für jede zufällig erzeugte Wertänderung wird die Auswirkung auf das aktuelle Portfolio berechnet.
- Aus der geordneten (empirischen) Messreihe der simulierten Änderungen des Portfolio wird das Quantil bestimmt.

Zufallsprozesse

- Radioaktiver atomarer Zerfall
- Thermisches Rauschen (Widerstand)

- Zufallsprozesse sind reale physikalische Prozesse.
- Zur Durchführung von Simulationen werden häufig Zufallszahlen benötigt.
- Nur sehr selten werden zur Erzeugung von Zufallszahlen reale Zufallsprozesse beobachtet.

Zufallsgeneratoren

- Meist werden für Simulationen Zufallsgeneratoren eingesetzt, die algorithmisch Zufallszahlen erzeugen.
- Hierbei handelt es sich um deterministische Pseudo-Zufallszahlen, die sich periodisch wiederholen.
- Es werden also immer die selben Zufallszahlen aufeinander folgen.

Beispiel

rand() unter BSD-Unix (Linear Kongruenter Generator LCG)

$$x(n) = (a * x(n-1) + c) \bmod m$$

$$a = 1103515245$$

$$c = 12345$$

$$m = 2^{31}$$

Periode: 2^{31}

x ist die aktuell berechnete Zufallszahl, die von der vorher berechneten eindeutig abhängt. Daher ist die Initialisierung von *x* von großer Bedeutung.

Anwendung

Initialisierung erfolgt über einen Seed (Saat)-Wert. Häufig wird die aktuelle Systemzeit verwendet.

```
srand( time(0))
```

Der von *rand()* generierte Wert liegt im Bereich:

$$0 \leq x \leq m$$

Für die Erzeugung von Zufallszahlen aus einem speziellen Wertebereich sollte das Ergebnis von *rand()* mit *m* und dem gewünschten Wertebereich skaliert werden. Üblicherweise kann *m* als Konstante abgefragt werden.

Erzeugung standardisierter normalverteilter Zufallszahlen

1. Erzeugung zweier gleichverteilter unabhängiger Zufallszahlen x_1 und x_2 zwischen 0 und 1.

2. Umformungen: $v_1 = 2x_1 - 1$, $v_2 = 2x_2 - 1$, $s = v_1^2 + v_2^2$

3. Falls $s \geq 1$, zurück zu 1.

4. $u_1 = v_1\sqrt{-(2/s)\ln(s)}$, $u_2 = v_2\sqrt{-(2/s)\ln(s)}$

u_1 und u_2 sind zwei unabhängige Zufallszahlen, die der Standardnormalverteilung folgen.

Erzeugung normalverteilter Zufallszahlen

Eine Normalverteilung *N* ist mit einer Kovarianzmatrix

$\widehat{\underline{\underline{\mathrm{cov}}}}_{\underline{x}}$ und einem Erwartungsvektor $\underline{\hat{\mu}}_{\underline{x}}$ vollständig definiert.

Zufallsvektoren $\underline{y}$, die der Normalverteilung *N* genügen, können durch folgende Operation erzeugt werden:

$$\underline{y} = \underline{\underline{D}}\,\underline{u} + \underline{\hat{\mu}}_{\underline{x}} \quad ,$$

wobei $\underline{u}$ ein Zufallsvektor ist, der einer Standardnormal-verteilung folgt, und $\underline{\underline{D}}$ die untere Dreiecksmatrix der Cholesky-Zerlegung der Kovarianzmatrix ist.

Cholesky-Zerlegung

$\underline{\underline{B}}$ sei die Kovarianzmatrix $\underline{\underline{\widehat{cov}}}_{\underline{x}}$.

$\underline{\underline{B}} = \underline{\underline{D}}\underline{\underline{D}}^T$ wird als Cholesky-Zerlegung bezeichnet.

Die Koeffizienten d_{ij} der Matrix $\underline{\underline{D}}$ können in folgender Weise berechnet werden:

$$d_{ij} = \begin{cases} 0 & \text{für } i < j \\ \sqrt{b_{ii} - \sum_{k=1}^{i-1} d_{ik}^2} & \text{für } i = j \\ \frac{1}{d_{jj}} \left(b_{ij} - \sum_{k=1}^{j-1} d_{ik} d_{jk} \right) & \text{für } i > j \end{cases}$$

Cholesky-Zerlegung

$\underline{\underline{B}}$ sei die Kovarianzmatrix $\underline{\underline{\widehat{cov}}}_{\underline{x}}$.

$\underline{\underline{D}}$ ist die untere Dreiecksmatrix der Cholesky-Zerlegung.

$$d_{ij} = \begin{cases} 0 & \text{für } i < j \\ \sqrt{b_{ii} - \sum_{k=1}^{i-1} d_{ik}^2} & \text{für } i = j \\ \frac{1}{d_{jj}} \left(b_{ij} - \sum_{k=1}^{j-1} d_{ik} d_{jk} \right) & \text{für } i > j \end{cases}$$

d_{11}	0	0	0
d_{21}	d_{22}	0	0
d_{31}	d_{32}	d_{33}	0
d_{41}	d_{42}	d_{43}	d_{44}

Cholesky-Zerlegung

$\underline{\underline{B}}$ sei die Kovarianzmatrix $\underline{\underline{\widehat{cov}}}_{\underline{x}}$.

$\underline{\underline{D}}$ ist die untere Dreiecksmatrix der Cholesky-Zerlegung.

$$d_{ij} = \begin{cases} 0 & \text{für } i < j \\ \sqrt{b_{ii} - \sum_{k=1}^{i-1} d_{ik}^2} & \text{für } i = j \\ \frac{1}{d_{jj}} \left(b_{ij} - \sum_{k=1}^{j-1} d_{ik} d_{jk} \right) & \text{für } i > j \end{cases}$$

d_{11}	0	0	0
d_{21}	d_{22}	0	0
d_{31}	d_{32}	d_{33}	0
d_{41}	d_{42}	d_{43}	d_{44}

Zusammenfassung

1. Berechnung von Kovarianzmatrix und Erwartungsvektor.
2. Berechnung der Cholesky-Zerlegung der Kovarianzmatrix.
3. Erzeugung von Zufallsvektoren, die der Standardnormalverteilung folgen.
4. Auf Basis des Erwartungsvektors und der Cholesky-Zerlegung der Kovarianzmatrix kann für jeden erzeugten Zufallsvektor aus 3. ein Zufallsvektor generiert werden, der der Normalverteilung folgt, die durch die in 1. berechneten Parameter definiert wird.

Parametrisierung

- Kurse liegen als absolute Werte vor.
- Man möchte die Veränderung erfassen und hierbei eine gewisse Unabhängigkeit von den absoluten Größen erreichen.
- Es ist naheliegend die Kursveränderung als relativen Wert zu erfassen.

Deutsche Bank: 41,70 € (17.04.09), 39,26 € (16.04.09)

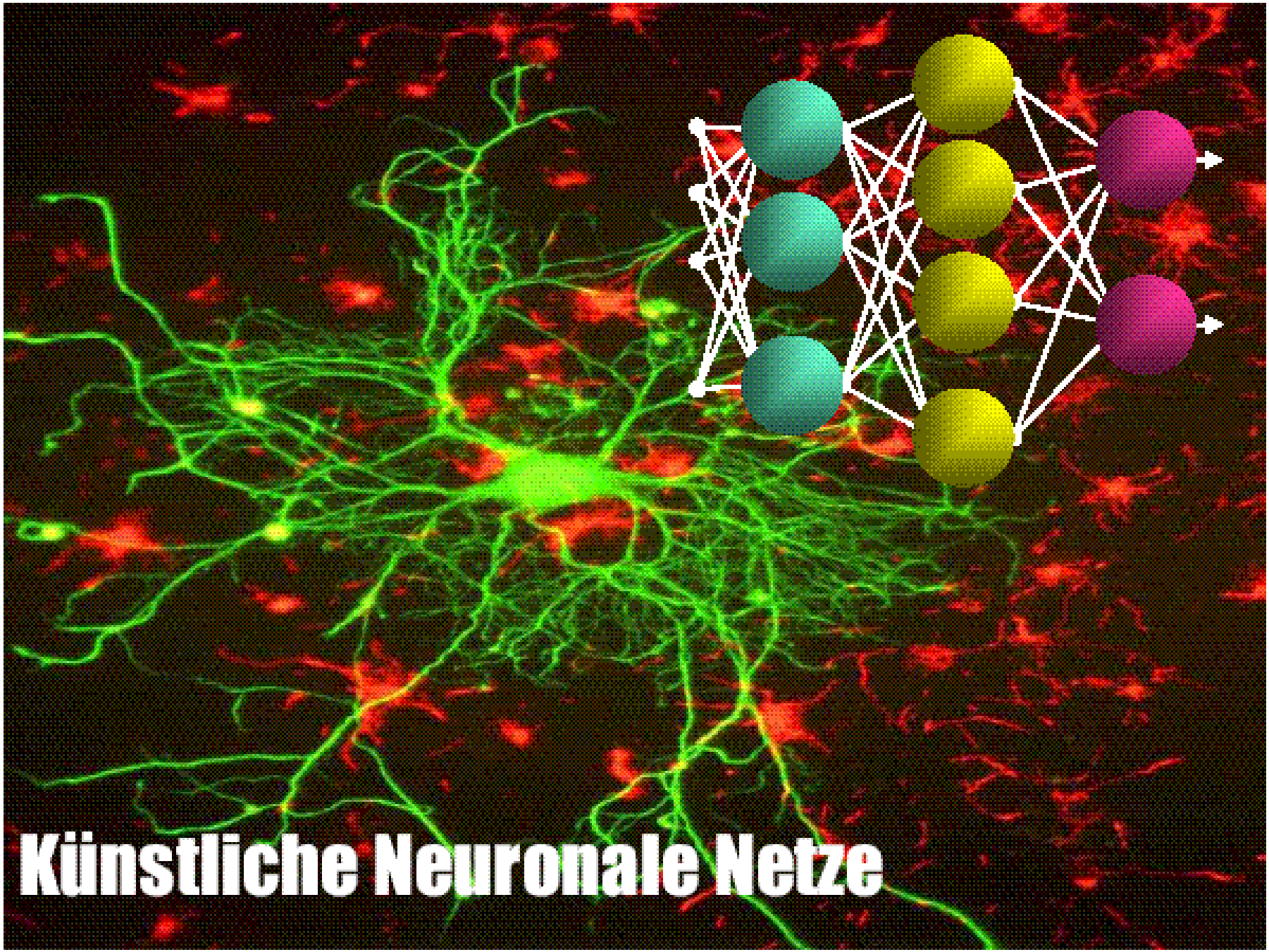
$$\frac{41,70 - 39,26}{39,26} = 0,062 = 6,2\%$$

Berechnung des VaR mit einer Monte Carlo Simulation

- An Hand historischer Daten werden der Erwartungs-vektor und die empirische Kovarianzmatrix für den Vektor der Kursänderungen geschätzt. Wobei der Vektor die Kurse der Papiere enthält, die im aktuellen Portfolio enthalten sind.
- Es wird eine große Zahl von Zufallsvektoren erzeugt, die der geschätzten Normalverteilung der Kursänder-ungen der Werte des Portfolios folgen.
- Für jede zufällig erzeugte Wertänderung wird die Auswirkung auf das aktuelle Portfolio berechnet.
- Aus der geordneten (empirischen) Messreihe der simulierten Änderungen des Portfolio wird das Quantil bestimmt.

Berechnung des VaR

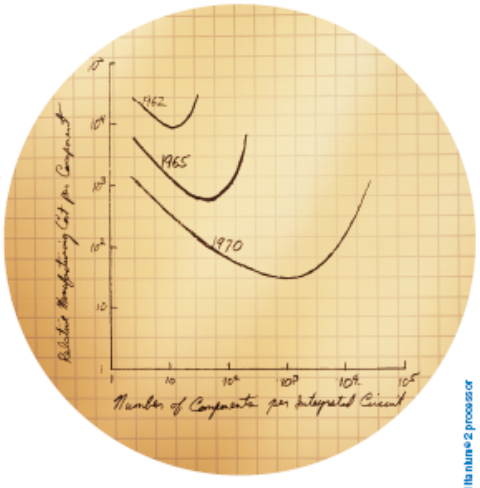
- **Monte Carlo Simulation**
- Sammlung und Analyse historischer Daten.
- Modellierung der Verteilung.
- Simulation von Kursentwicklungen gemäß der modellierten Verteilung.
- Große Zahl an Simulationen.
- Prognose von Kursentwicklungen.
- Berechnung des monetären Risikos für die Entwicklung des Portfolios.


Künstliche Neuronale Netze

Moore's Law

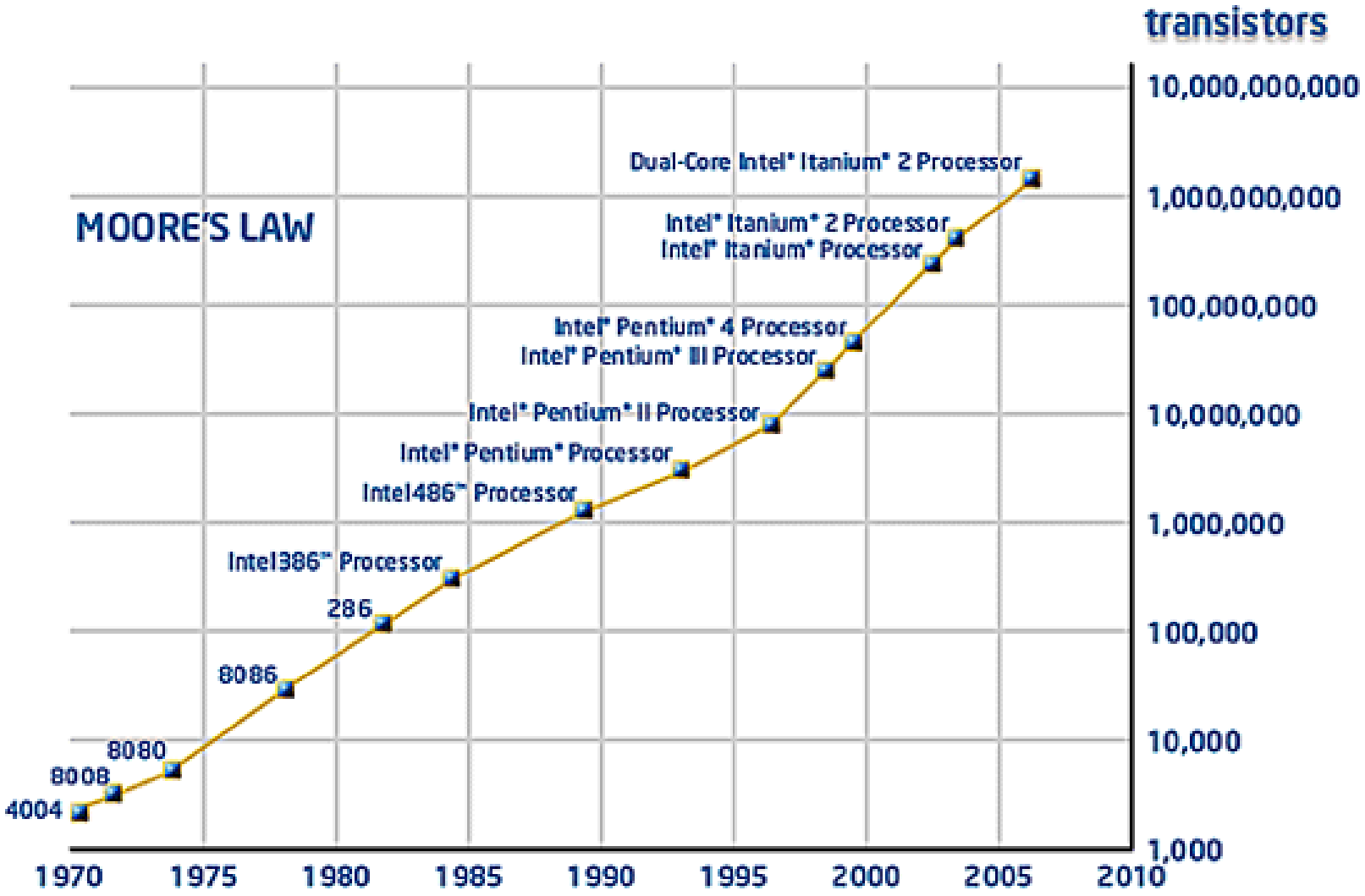
"The number of transistors incorporated in a chip will approximately double every 24 months."

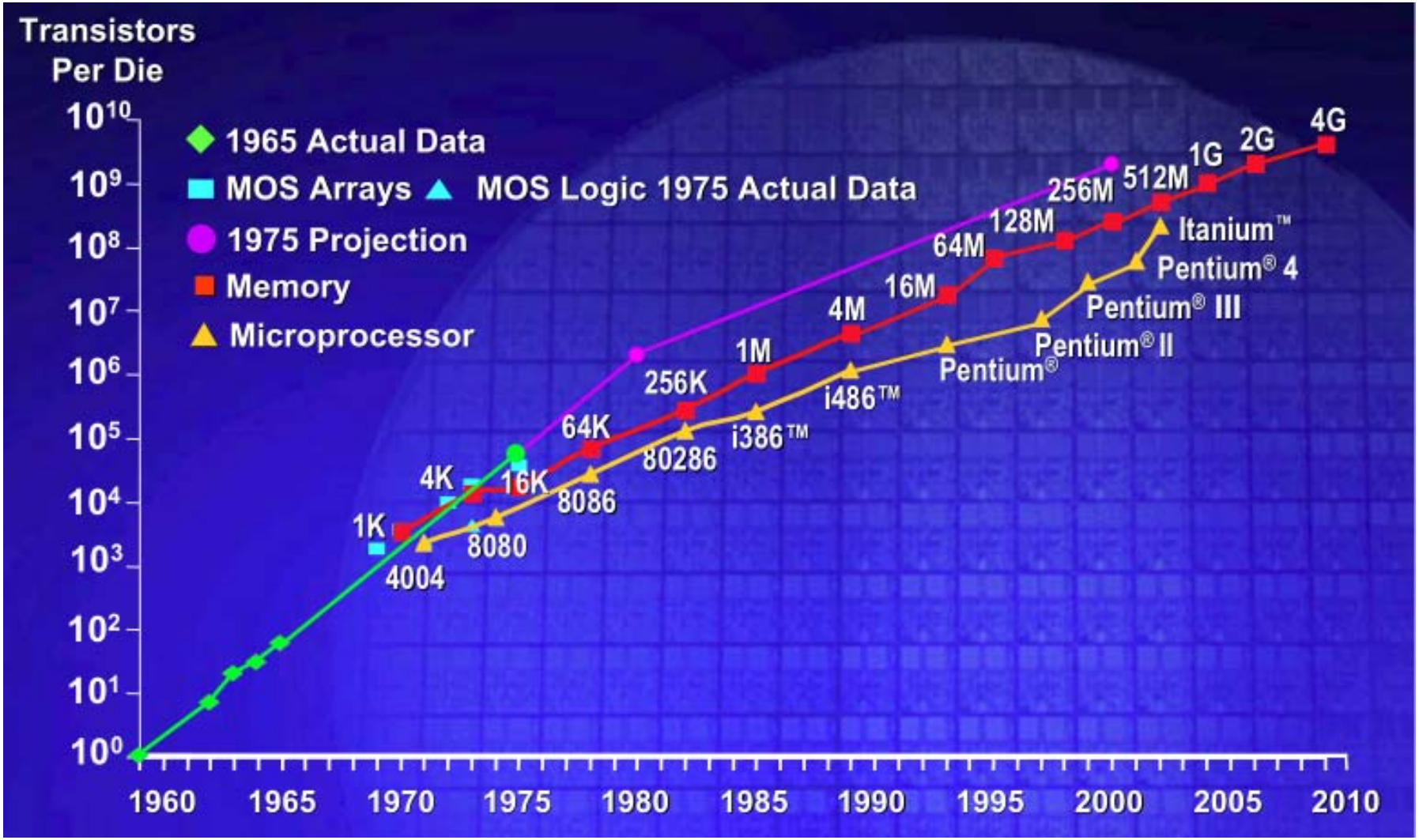
Gordon Moore, Intel Co-founder

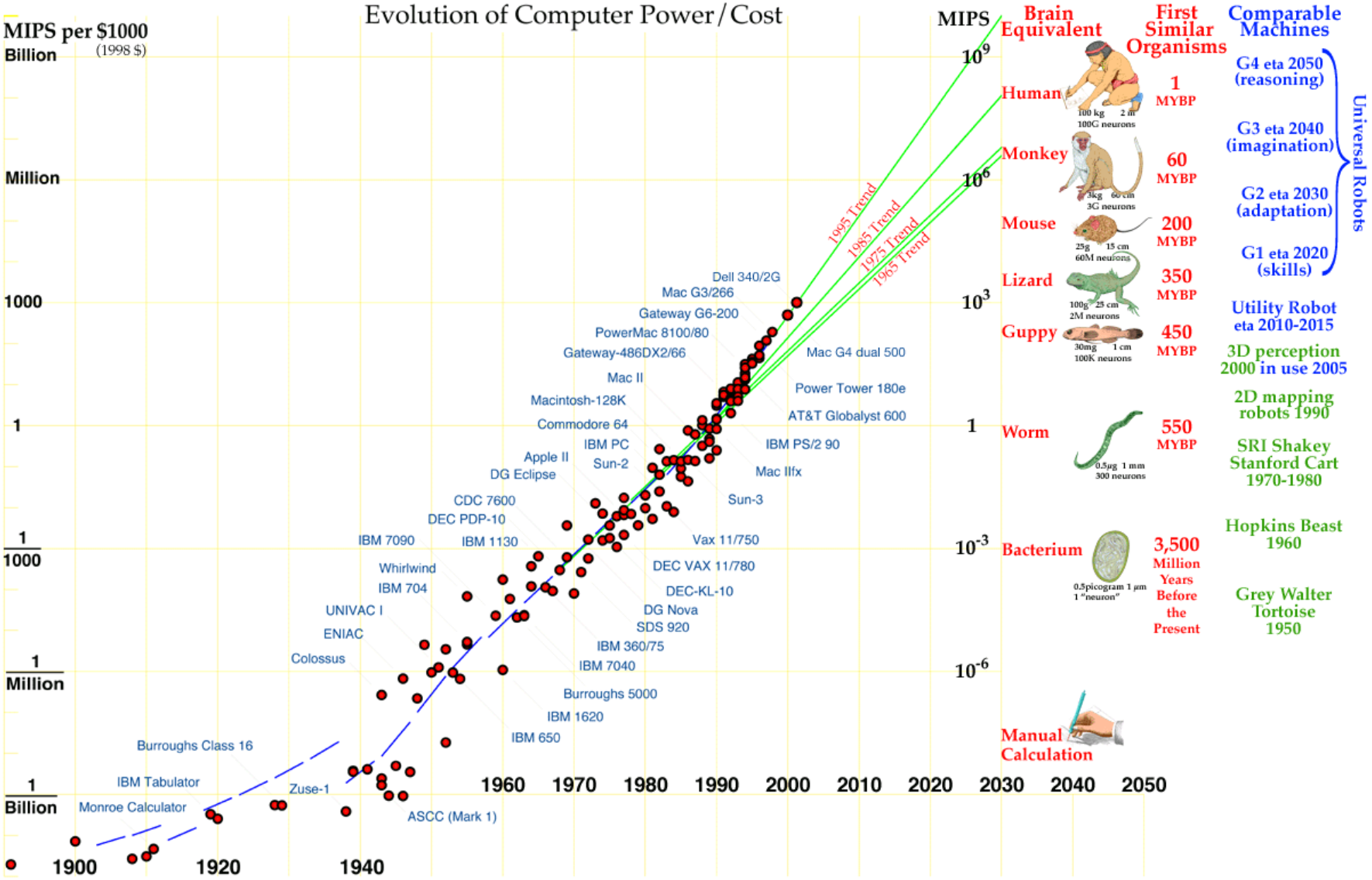


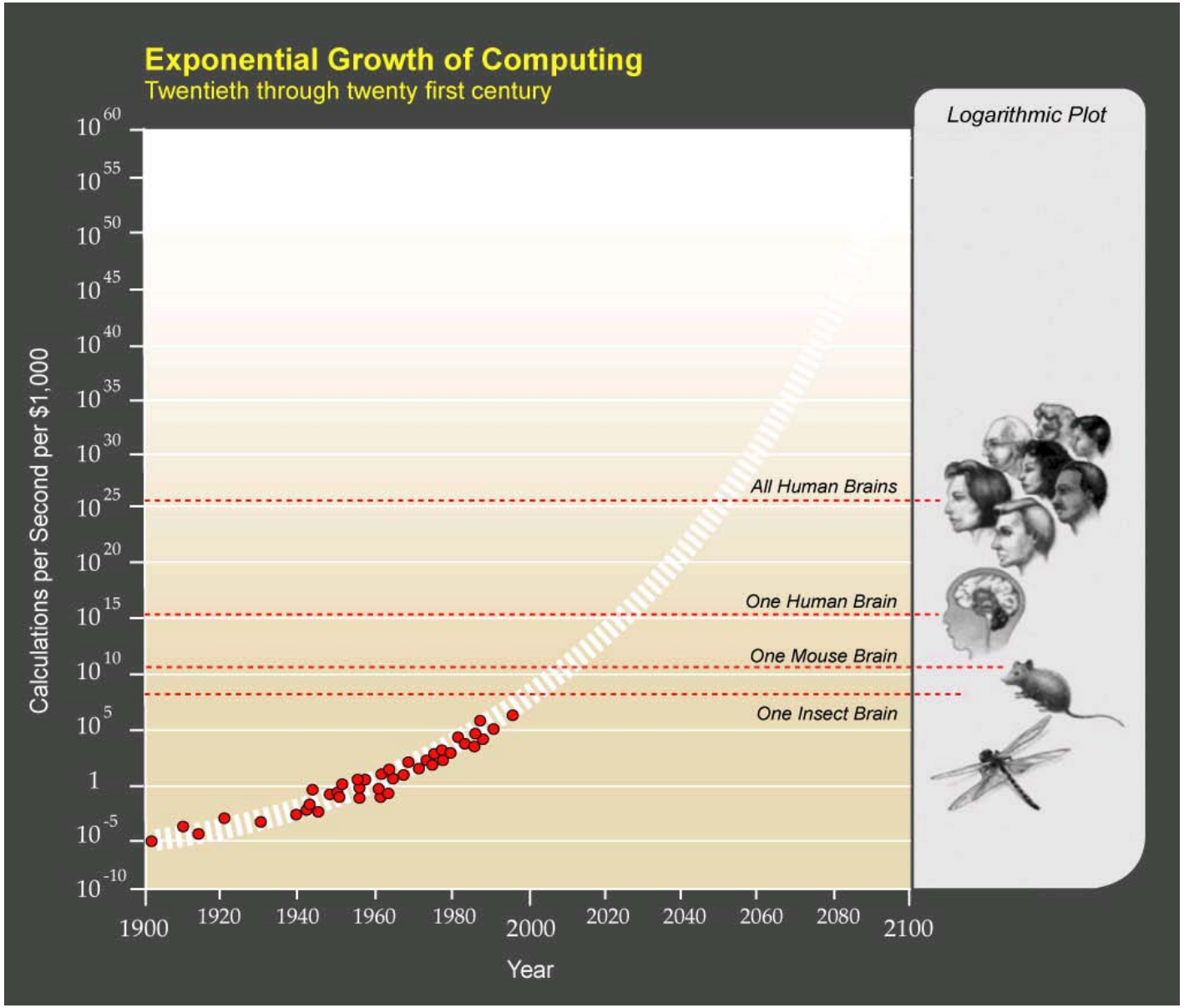
In 1965, Gordon Moore sketched out his prediction of the pace of silicon technology. Decades later, Moore's Law remains true, driven largely by Intel's unparalleled silicon expertise.

Silicon technology is transforming the lives of billions of people around the globe. By leading the advancement of Moore's Law, Intel is a major driver and enabler of that transformation. Intel will continue to define the future of silicon technology, fueling architecture and platform innovation and improving how we work, play, learn, and communicate.


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
10^10
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1965 Actual Data
MOS Arrays
MOS Logic 1975 Actual Data
1975 Projection
Memory
Microprocessor
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™
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010

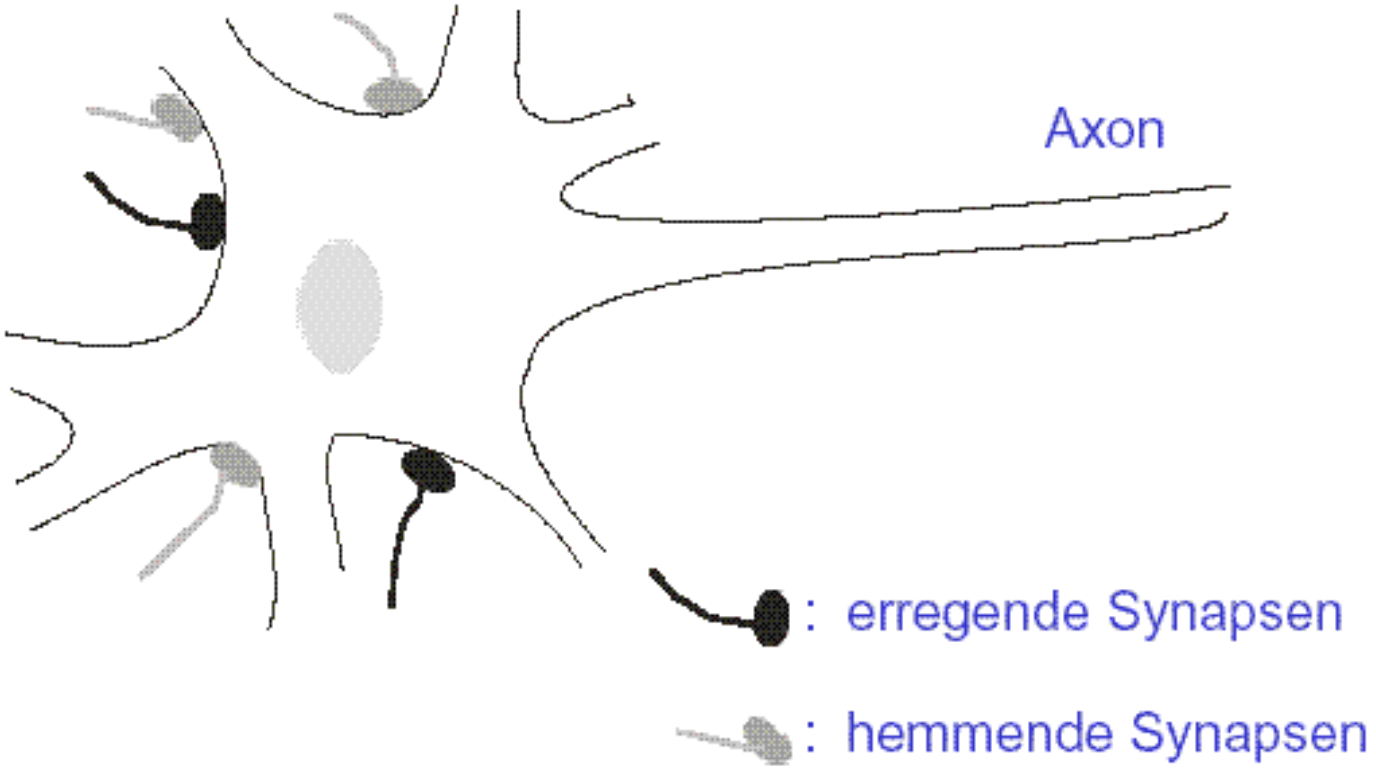

Evolution of Computer Power/Cost
MIPS per $1000
(1998 $)
Billion
Million
1000
1
1/1000
1/Million
1/Billion
MIPS
10^9
10^6
10^3
1
10^-3
10^-6
1900
1920
1940
1960
1970
1980
1990
2000
2010
2020
2030
2040
2050
1995 Trend
1985 Trend
1975 Trend
1965 Trend
Dell 340/2G
Mac G3/266
Gateway G6-200
PowerMac 8100/80
Gateway-486DX2/66
Mac G4 dual 500
Mac II
Power Tower 180e
Macintosh-128K
AT&T Globalyst 600
Commodore 64
IBM PC
IBM PS/2 90
Apple II
Sun-2
Mac IIfx
DG Eclipse
Sun-3
CDC 7600
DEC PDP-10
IBM 7090
IBM 1130
Vax 11/750
Whirlwind
IBM 704
DEC VAX 11/780
DEC-KL-10
UNIVAC I
DG Nova
ENIAC
SDS 920
IBM 360/75
Colossus
IBM 7040
Burroughs 5000
IBM 1620
IBM 650
Burroughs Class 16
IBM Tabulator
Zuse-1
Monroe Calculator
ASCC (Mark 1)
Brain Equivalent
First Similar Organisms
Comparable Machines
Human
1 MYBP
100 kg 2 m
100G neurons
Monkey
60 MYBP
3kg 60 cm
3G neurons
Mouse
200 MYBP
25g 15 cm
60M neurons
Lizard
350 MYBP
100g 25 cm
2M neurons
Guppy
450 MYBP
30mg 1 cm
100K neurons
Worm
550 MYBP
0.5µg 1 mm
300 neurons
Bacterium
3,500 Million Years Before the Present
0.5picogram 1 µm
1 "neuron"
Manual Calculation
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

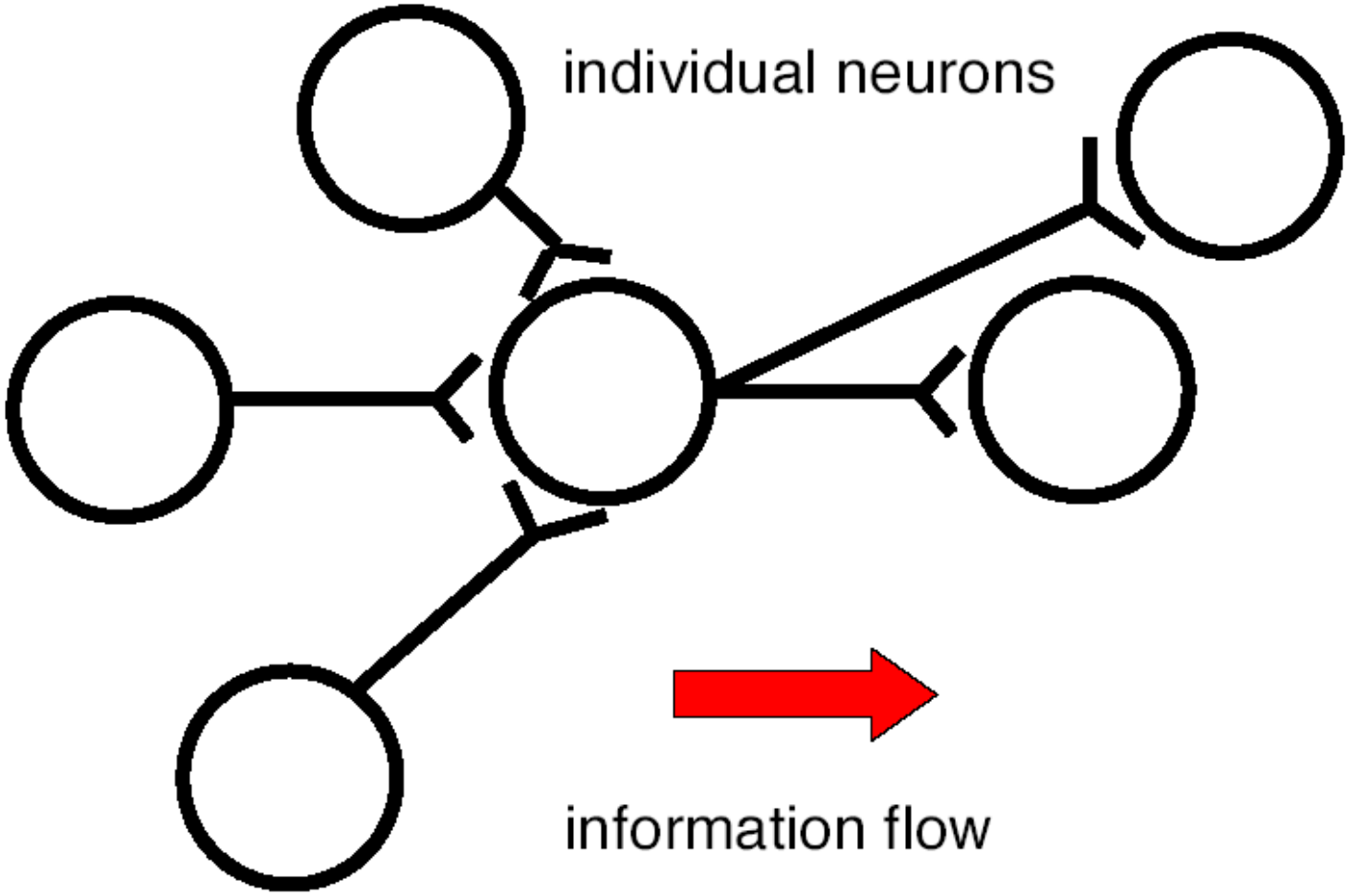
Gehirn - Mikroprozessor

	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

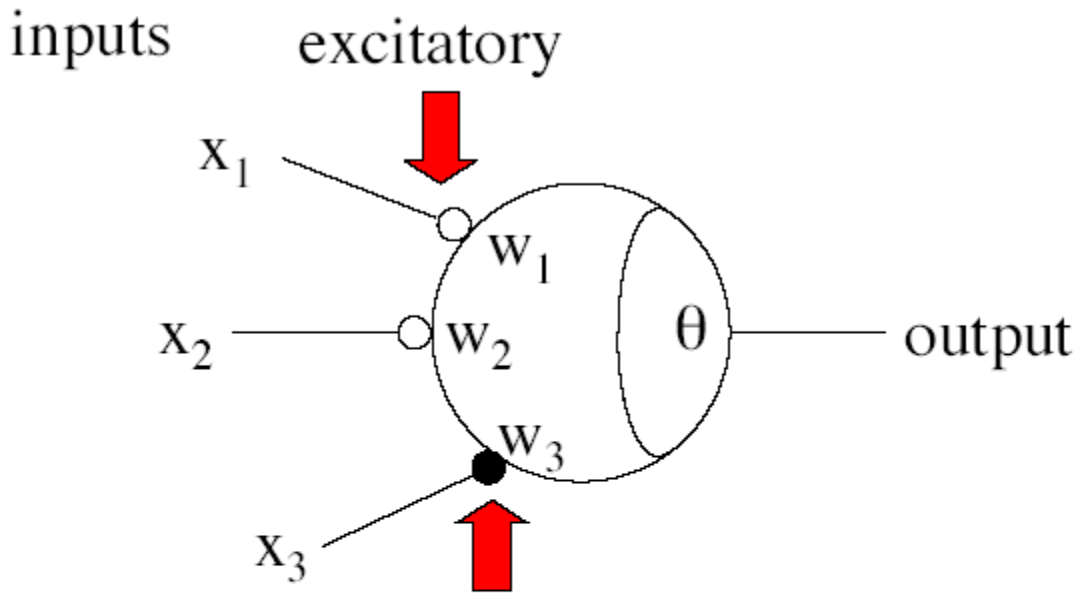
Aufbau eines Neurons



Künstliche Neuronen



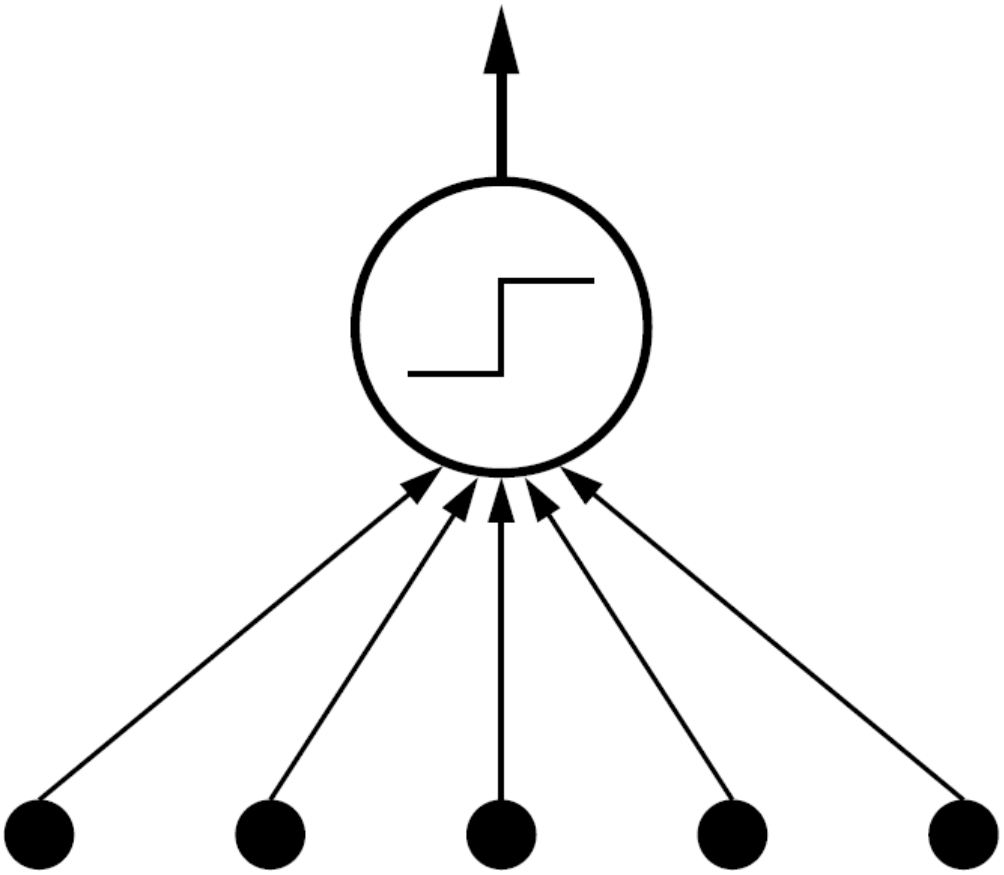
McCulloch-Pitts Neuron



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

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

Perzeptron



Output

Gewichte

Input

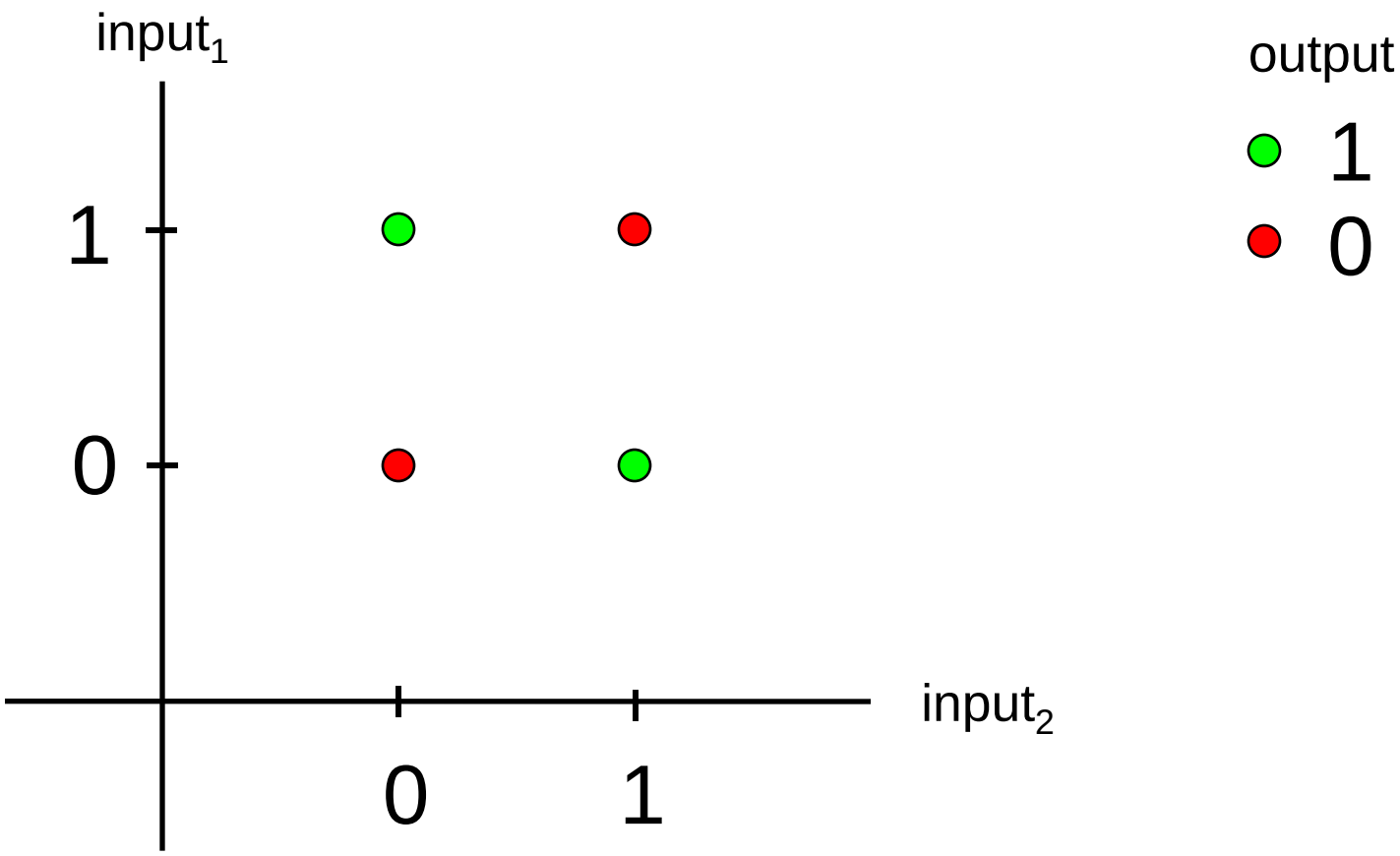
Perzeptron (Demo)

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)

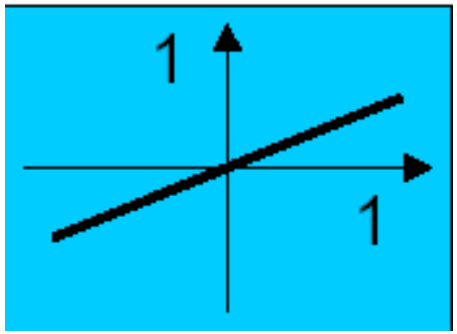
Berechnung der Aktivität

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

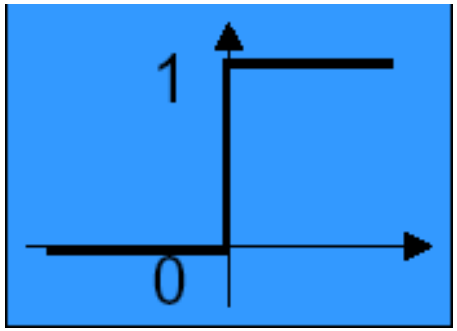
Aktivität: $x_i = f(a_i)$

Transferfunktion: $f(a_i)$

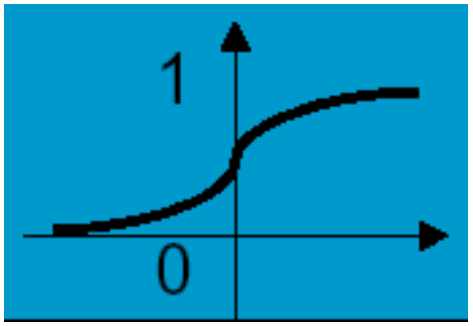
Transferfunktionen



linear

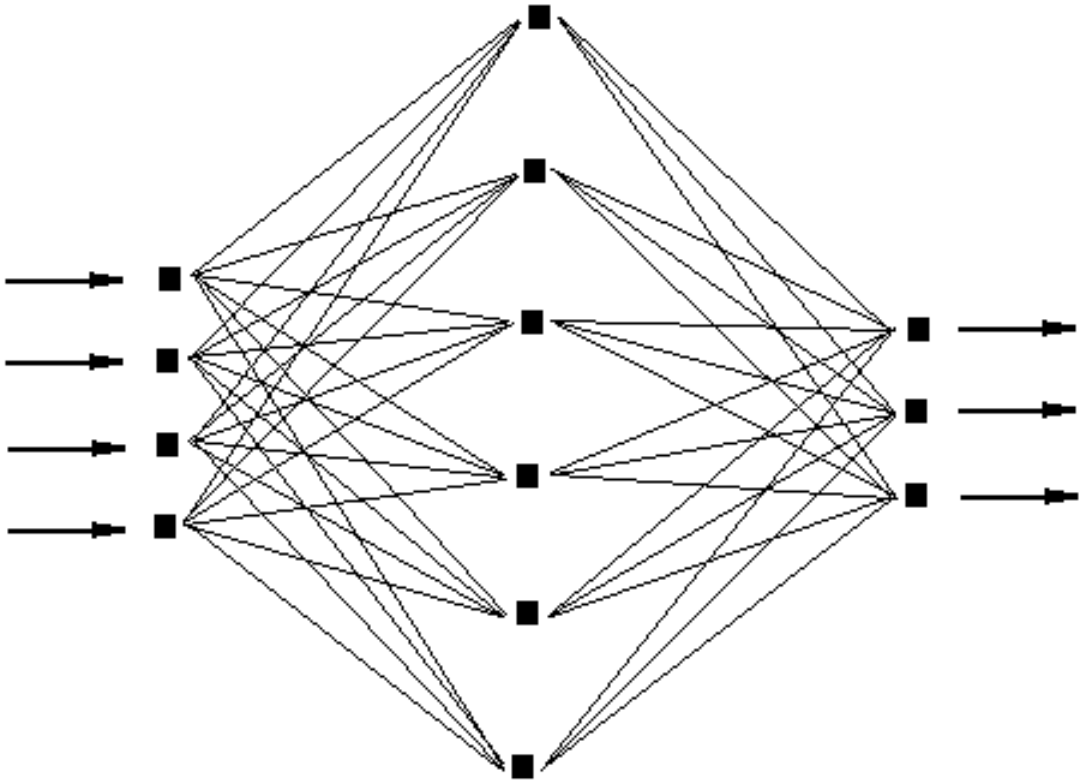


Sprungfunktion

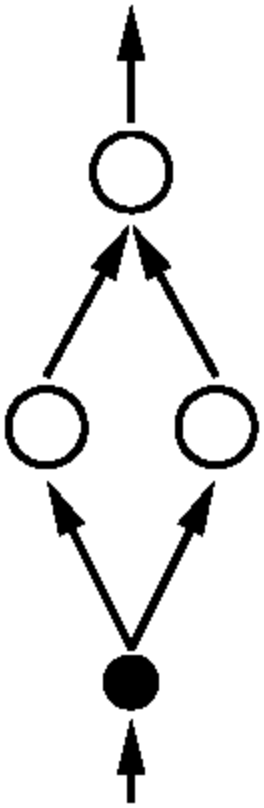


Sigmoidfunktion

Mehrschichtiges Perzeptron (MLP)



Mehrschichtiges Perzeptron



Output

Hidden Layer

Input (keine Verarbeitung!)

Training (Back-Propagation-Algorithmus)

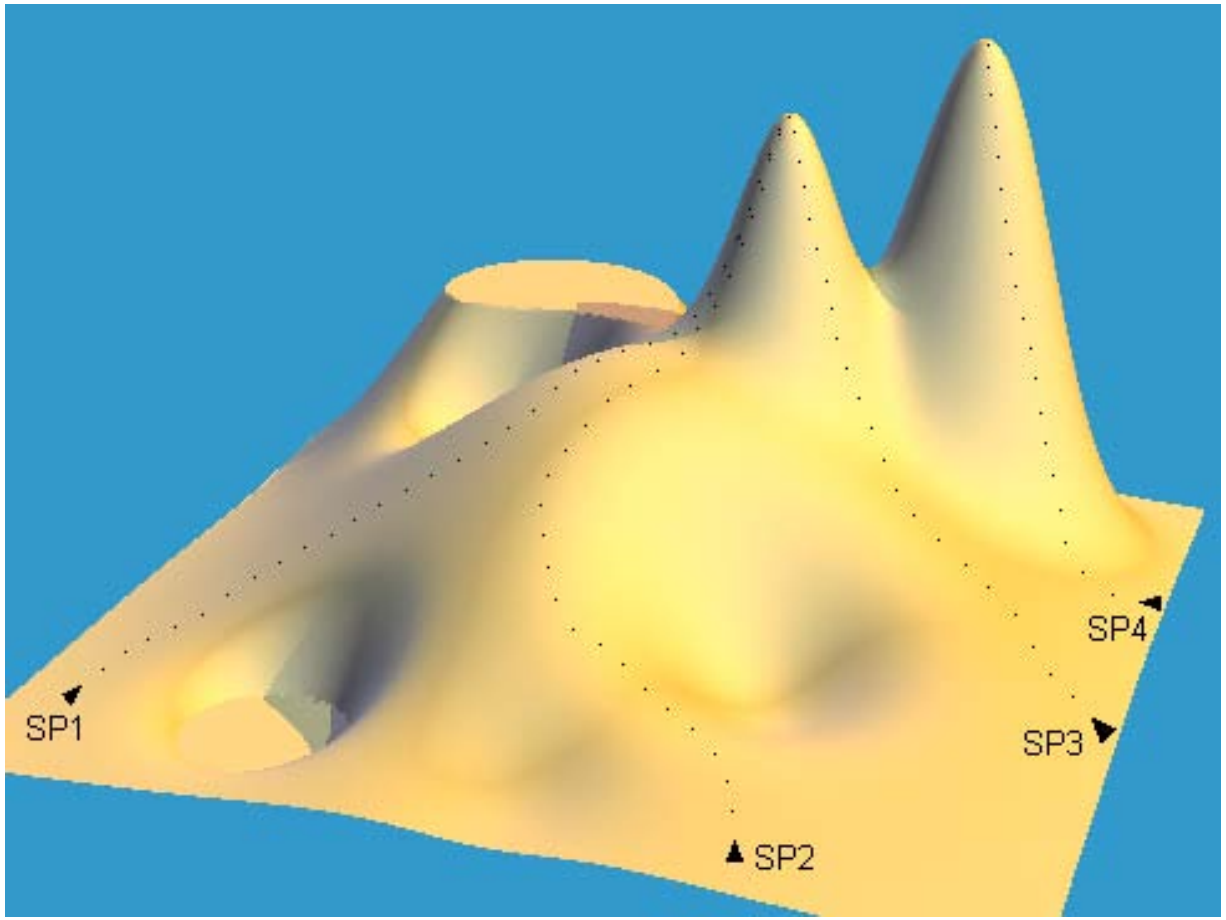
Partielle Ableitung

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

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

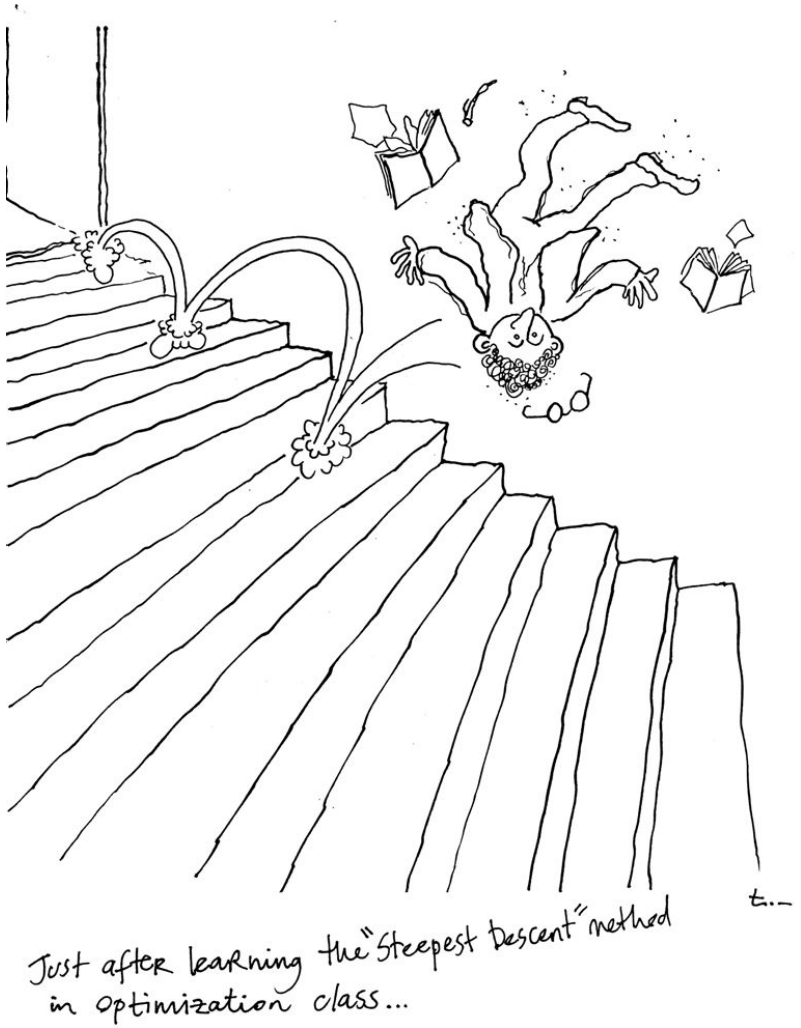
Ein **Gradient** gibt die Änderung einer Größe zwischen räumlich oder zeitlich definierten Punkten an, er ist ein Vektor und zeigt in Richtung des größten Gefälles.

Gradientenverfahren



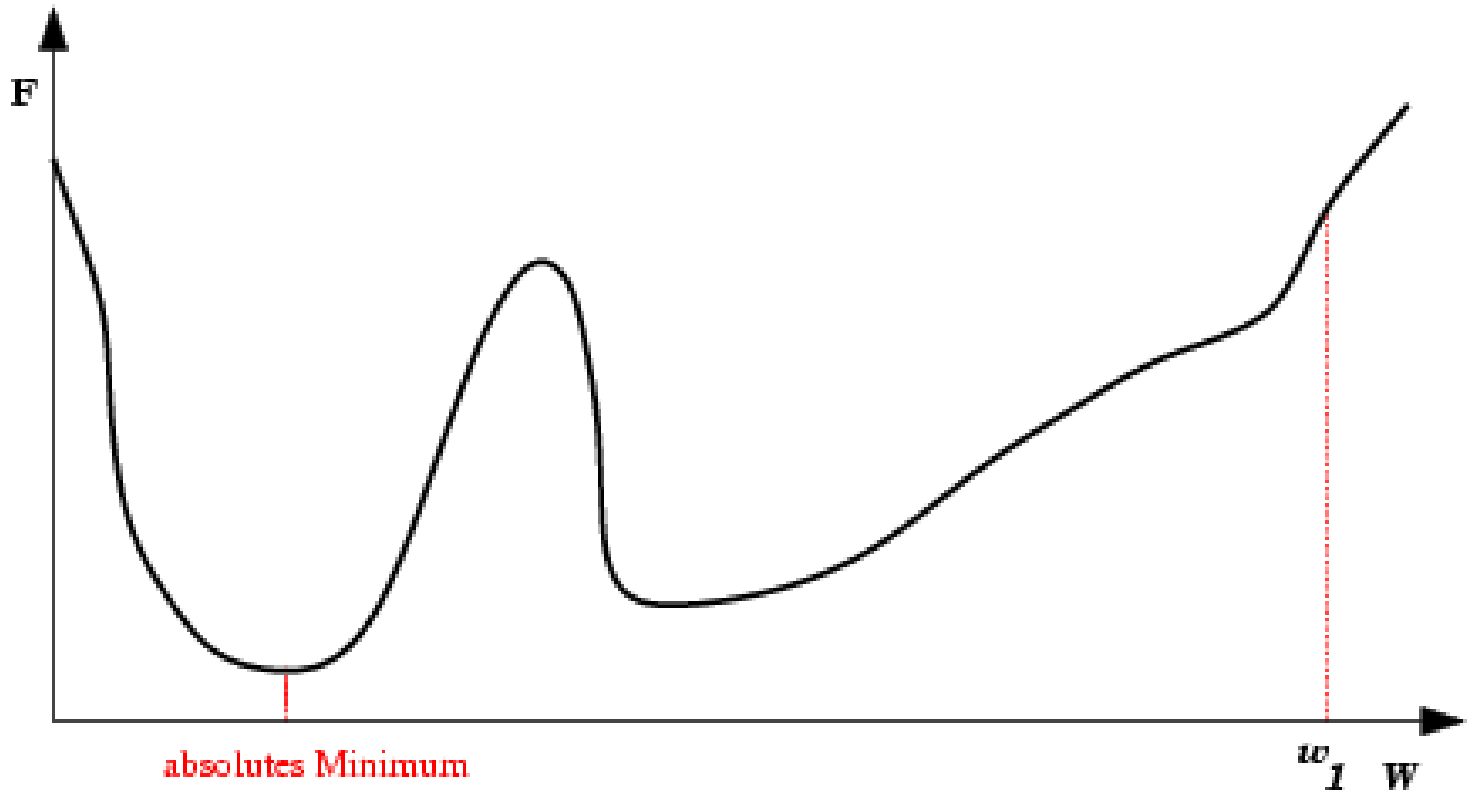
Quelle: http://www.statistics4u.info/fundstat_germ/cc_optim_meth_gradient.html

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

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

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

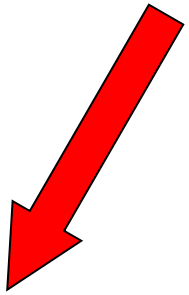
$$\frac{df}{dx} = 6g^2 \cdot (-2x) = 6(3 - x^2)^2(-2x)$$

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

$$a_i = \sum_{j=1}^{N} w_{ji} x_j - \theta$$

$$= \sum_{j=1}^{N} w_{ji} x_j + w_{0i} x_0 \text{ , wobei } x_0 \equiv 1$$

$$= \sum_{j=0}^{N} w_{ji} x_j$$

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

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 δ_j^{imp} : implizite Fehler

Injizierte Fehler

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

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

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

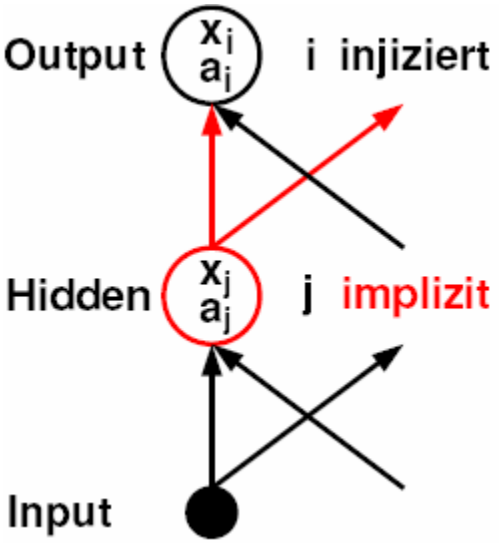
$$\delta_i^{inj} = -f'(a_i)\cdot(z_i - x_i)$$

Implizite Fehler

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

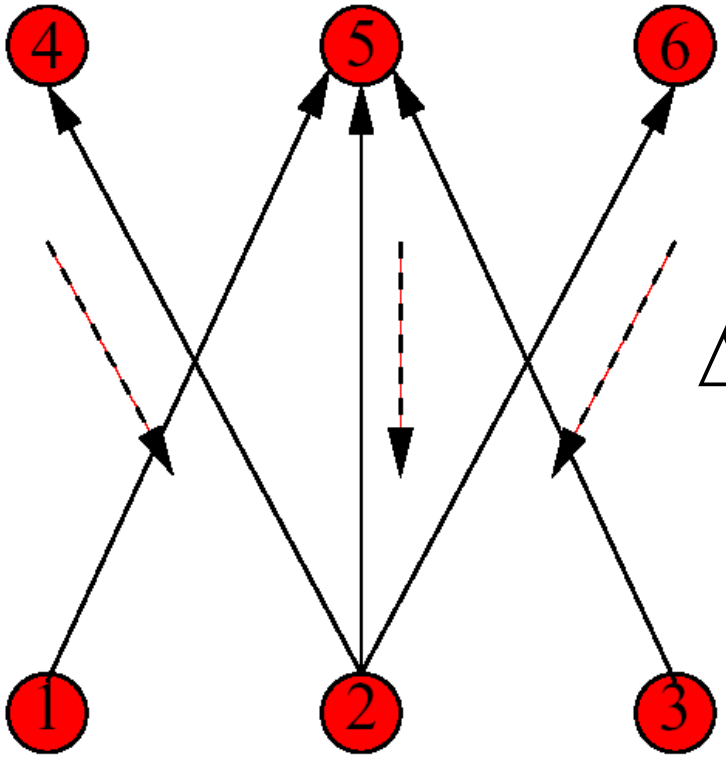
$$= \sum_{i \in \mathbb{N}_j} \delta_i \frac{\partial (w_{ji} f(a_j))}{\partial a_j}$$

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



Zusammenfassung

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



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

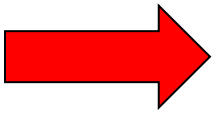
$$\delta_2 = f'(a_2) \cdot (w_{24}\delta_4 + w_{25}\delta_5 + w_{26}\delta_6)$$

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.

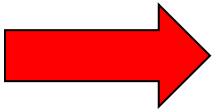
Ableitung Transferfunktion

Was fehlt noch für die vollständige Berechnung des Gradienten?



$f'(a)$

Transferfunktion



Hier: Ableitung der Sigmoidfunktion

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

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

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

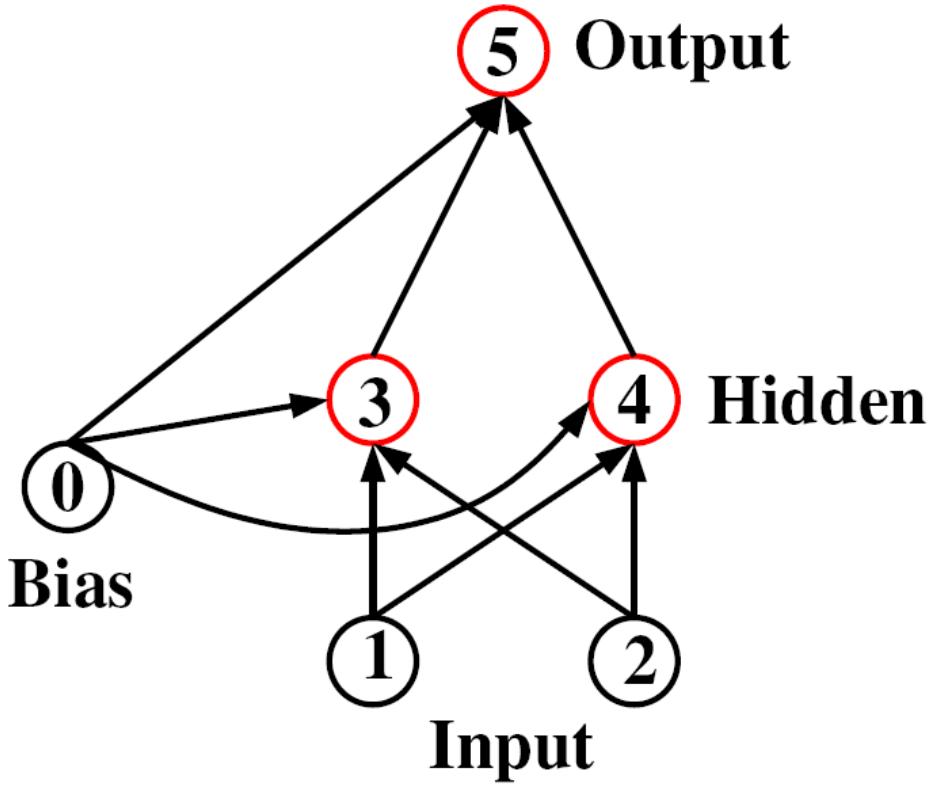
$$\delta_i^{inj} = -f'(a_i) \cdot (z_i - x_i)$$

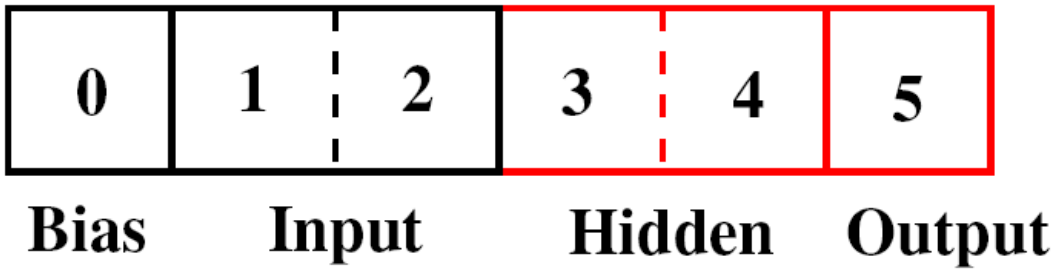
$$\delta_j^{imp} = f'(a_j) \sum_{i \in \mathbb{N}_j} w_{ji} \delta_i$$

Sigmoidfunktion:

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

Implementierung MLP



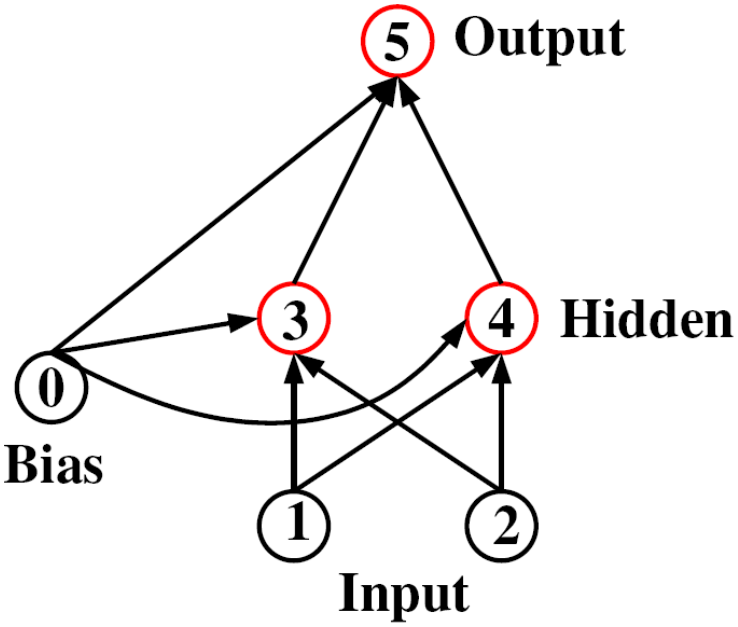


Berechnung Forward (propagate())

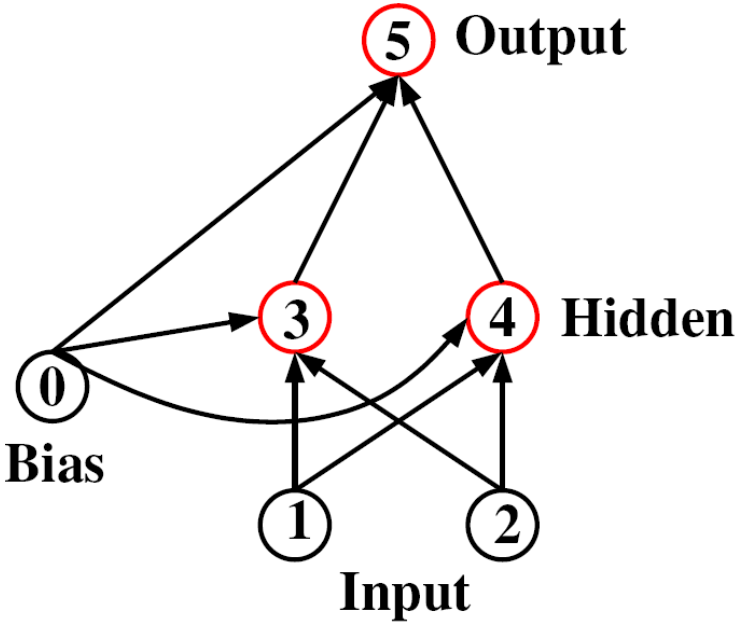
```
// set Input
for(i=1;i < StartHidden;i++)
    Activity[i] = Pattern[ActPattern][i]; // !!i

// 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
}
```



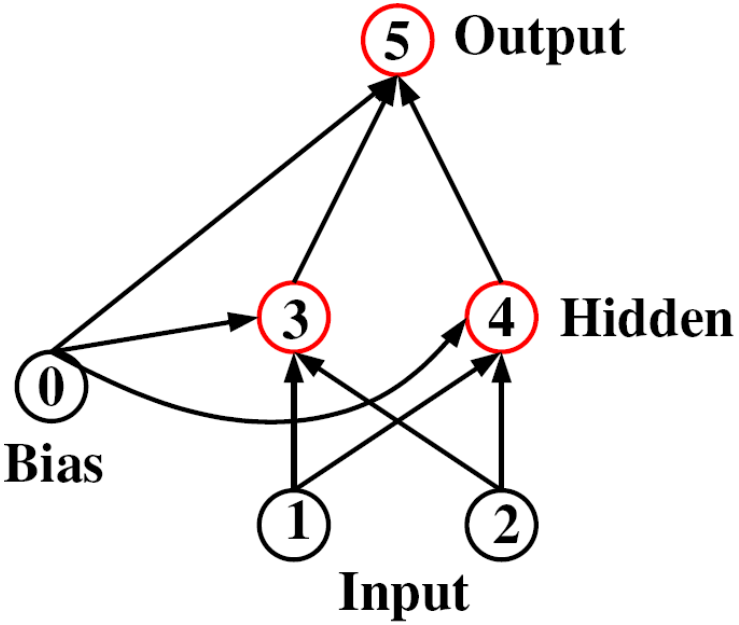
back_propagate - injizierter Fehler



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

```
// 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
```