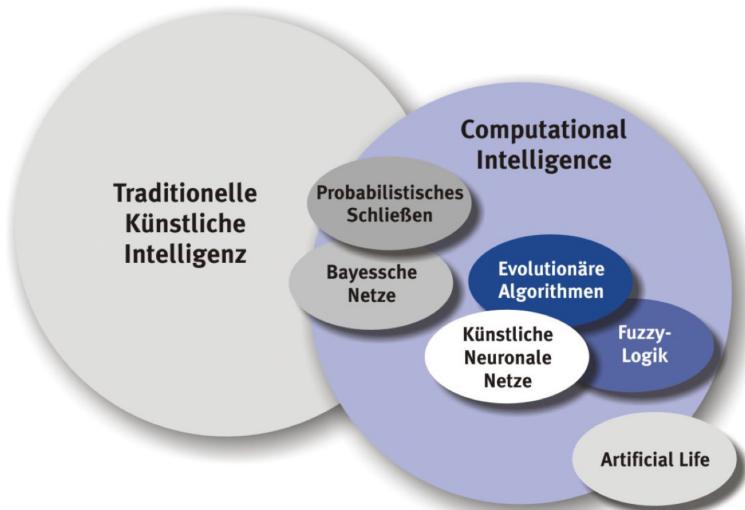




# I. Einführung

## Bausteine der Computational Intelligence



### Schwache künstliche Intelligenz:

ein Algorithmus gilt als intelligent, wenn er zur Problemlösung menschen ähnliche Leistungen, wie Lernen, Anpassung oder Schlussfolgerung, vollbringt

### Starke künstliche Intelligenz:

Es soll menschenähnliche Kognition, wie Bewusstsein, Emotionen oder Kreativität, nachgebaut werden

### Physical Symbol System Hypothesis:

"A physical symbol system has the **necessary and sufficient** means for general intelligent action."

[Alan Newell and Herbert Simon, 1976]

## Traditionelle Künstliche Intelligenz

Lösen von Problemen, die mit höheren kognitiven Funktionen verwandt sind

**Prinzipien:** Abstraktion, Adaption, Lernen, Kommunikation

## 2. Neuronale Netze

### Biologischer Hintergrund

#### Gehirn vs. Computer:

	Gehirn	Computer
Anz. Recheneinheiten	$\approx 10^{11}$	$\approx 10^9$
Art Recheneinheiten	Neuronen	Transistoren
Art der Berechnung	Massiv parallel	i.d.R. seriell
Datenspeicherung	assoziativ	adressbasiert
Schaltzeit	$\approx 10^{-3}\text{s}$	$\approx 10^{-9}\text{s}$
Theoretische Schaltvorgänge	$\approx 10^{14}\text{s}^{-1}$	$\approx 10^{18}\text{s}^{-1}$
Tatsächliche Schaltvorgänge	$\approx 10^{12}\text{s}^{-1}$	$\approx 10^{10}\text{s}^{-1}$

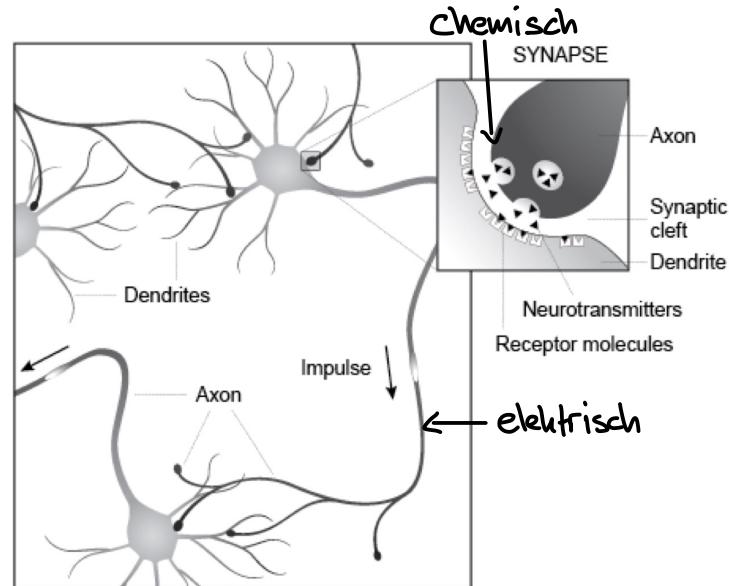
### Biologische Neuronen

Axon: übertragen Signale (Daten)

Dendrites: Empfangen Signale

Synapse: Verbindungsplatz von Axon und Neuronen

→ Neuronen sind meist gleich aufgebaut aber unterscheiden sich hauptsächlich in der Konnektivität



### Signalübertragung an Synapsen

Zwischen Axon und Dendrit: elektrischer Impuls → chemisches Signal  
→ hier wird gelernt

Höhe Flexibilität bei der Übertragung durch Neuromodulatoren

Excitation: Verstärkung

Inhibition: Hemmung

Bei zwei kurz aufeinander folgenden Impulsen:

Facilitation: 2. Puls hat eine stärkere Wirkung

Depression: 2. Puls hat eine schwächere Wirkung

## Informationsverarbeitung im Neuron

Ruhepotential eines inaktiven Neurons: ca. -70 mV

Feuern eines Neurons:

→ erregende Reize

Schwellenwert: ca. -50 mV

Selbstverstärkender Prozess wird gestartet

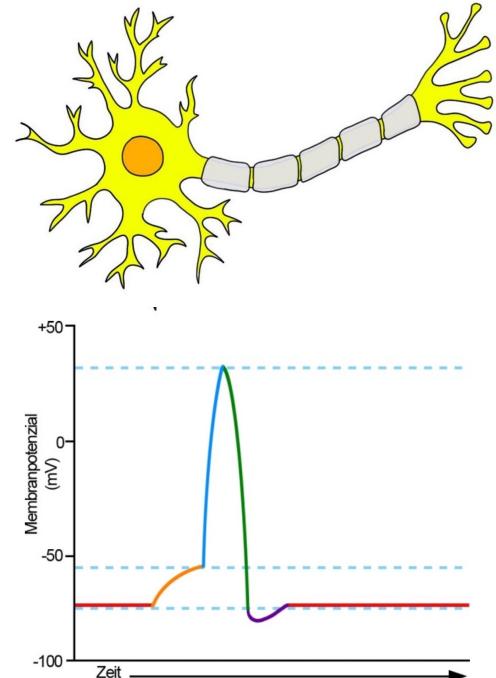
→ Potentialspitze: ca. 30 mV

→ Dauer: 1 ms

Refraktärzeit

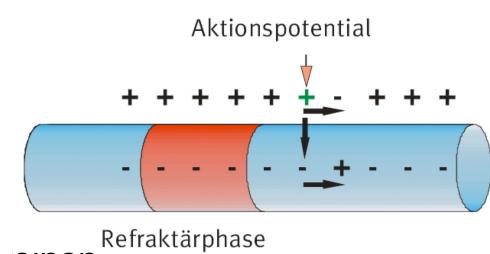
→ Neuron kann 5ms nicht angeregt werden

→ Potential sinkt unter Ruhepotential ab



## Ausbreitung von Aktionspotentialen

Weiterleitung: Lokale Depolarisation führt zu Depolarisation benachbarter Membranbereiche  
→ verlustfrei

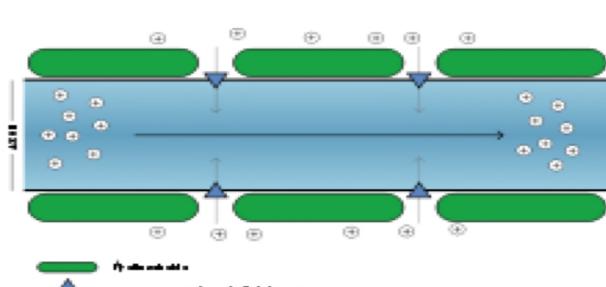


Aktionspotential wandert entlang des Axons

## Signalausbreitung

Spannung längs zum Axon, Strom quer zum Axon

Ausbreitungsgeschwindigkeit: 100  $\frac{m}{s}$

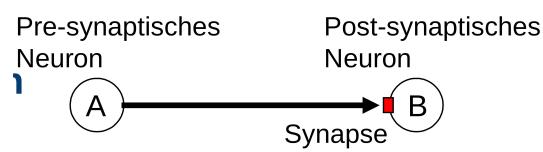


Ranviersche Schnürringe  
→ elektrische Isolierung

## Neuronales Netz

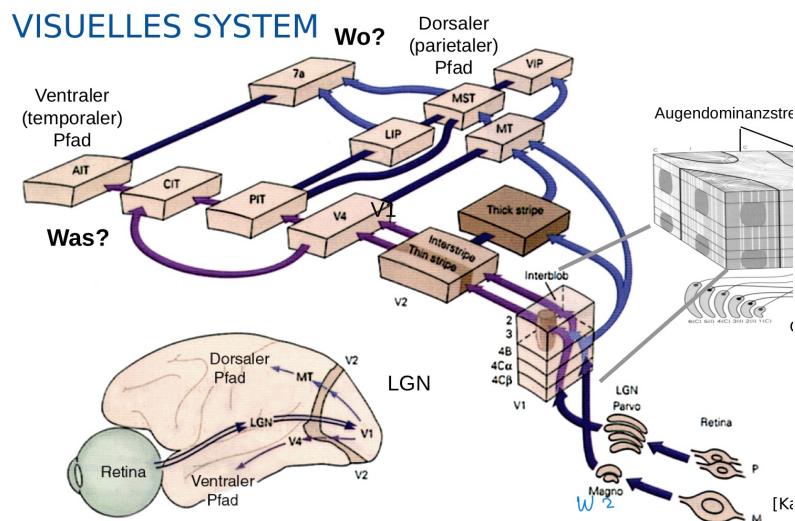
Hebb'sche Lernregel:

Synaptische Stärke wird erhöht, wenn Zelle A wiederholt zum Feuern von Zelle B beiträgt



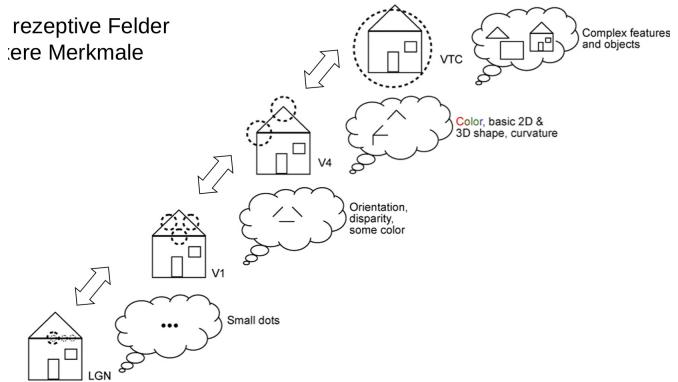
## Visuelles System

Dorsaler Pfad schneller als  
Ventraler Pfad  
→ Wo vor Was



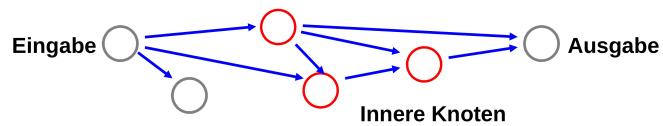
## Visuelle Verarbeitungshierarchie

Immer komplexere Merkmale



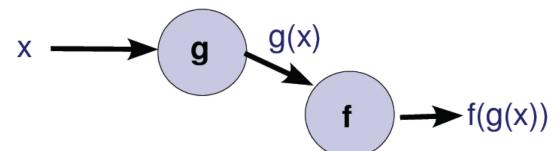
## Neuronale Netze

**Struktur:** gerichteter Graph  
→ Knoten (Neuronen)  
→ Kanten



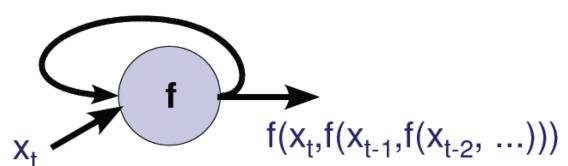
**Vorwärtsgerichtet:**

- keine Zyklen
- Ausgabe hängt nur von Eingabe zu einem Zeitpunkt ab

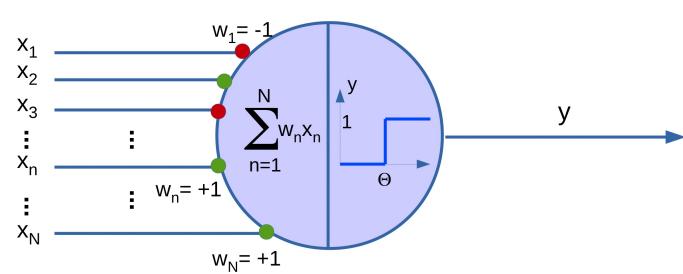
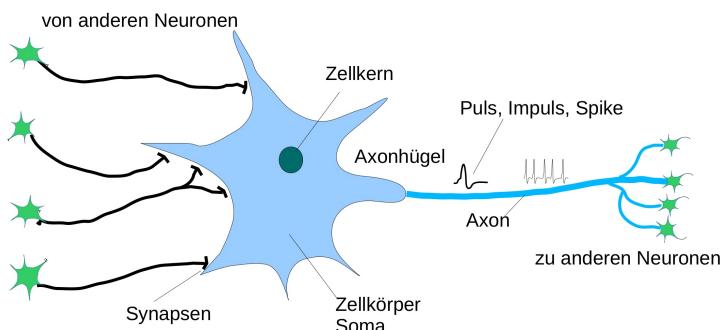


**rekurrent:**

- Zyklen
- Verbindungen nach hinten gerichtet
- Zeitverlust durch weitere zeitabhängigkeit



## Biologische vs künstliche Neuronen



## Integrationsfunktionen:

**Summe:**  $a = \sum_{i=1}^N w_i x_i$

**Distanz:**  $a = \sum_{i=1}^N (w_i - x_i)^2$

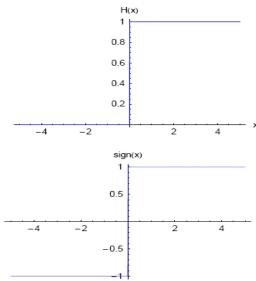
**Produkt:**  $a = \prod_{i=1}^N x_i^{w_i}$

**Maximum:**  $a = \max_i w_i x_i$

## Aktivierungsfunktionen

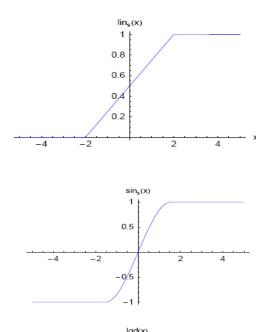
Unipolare Schwellwertfunktion

$$H(x) = \begin{cases} 0 & \text{falls } x < 0 \\ 1 & \text{sonst} \end{cases}$$



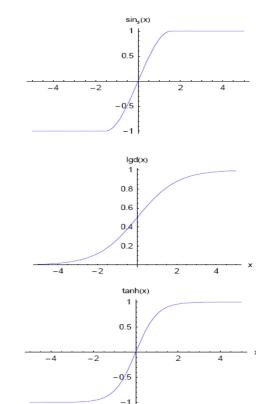
Bipolare Schwellwertfunktion

$$\text{sign}(x) = \begin{cases} -1 & \text{falls } x < 0 \\ 1 & \text{sonst} \end{cases}$$



Linear bis Sättigung

$$\text{lin}_s(x) = \begin{cases} 0 & \text{falls } x < \frac{a}{s} \\ s \cdot x - a & \text{falls } \frac{a}{s} \leq x \leq \frac{1+a}{s} \\ 1 & \text{sonst} \end{cases}$$



Sinus bis Sättigung

$$\sin_s(x) = \begin{cases} -1 & \text{falls } x < -\frac{\pi}{2} \\ \sin(x) & \text{falls } -\frac{\pi}{2} \leq x \leq \frac{\pi}{2} \\ 1 & \text{sonst} \end{cases}$$

Logistische Funktion

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

Tangens Hyperbolicus

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

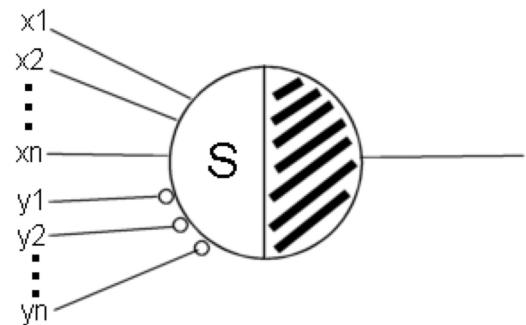
## McCulloch-Pitts-Netze

Anschlusskanten:  $(x_1, \dots, x_n)$  normal

$(y_1, \dots, y_n)$  hemmend

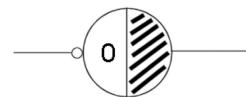
→ sobald ein hemmender Eingang anliegt, ist die Ausgabe 0

S: Schwellenwert

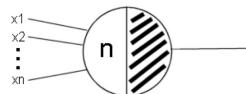


McCulloch-Pitts:

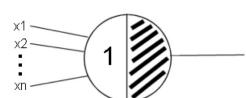
Negation:



Konjunktion:  
(und)



Disjunktion:  
(oder)

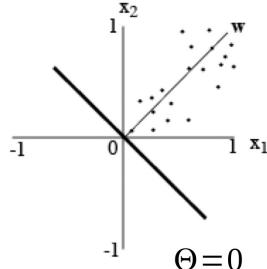


### 3. Perzeptron, MLP, Backpropagation of Error

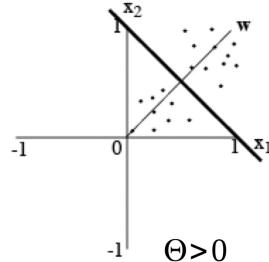
#### Trennung von Eingabemustern

Eingaberaum in zwei Regionen trennen durch eine Trennlinie, die durch die Gewichte definiert wird.

$$w_1 x_1 + w_2 x_2 - \Theta = 0$$



$$x_2 = \frac{\Theta}{w_2} - \frac{w_1}{w_2} x_1$$

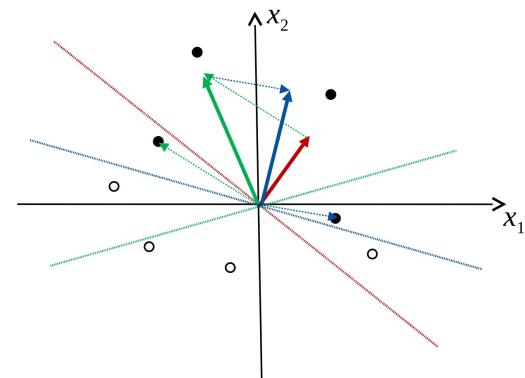


**Bias:** Konstante, die die Trenngerade verschiebt

$$y = \sum_{i=1}^N w_i x_i + \boxed{w_0}$$

#### Perzeptron-Lernalgorithmus

1. Identifizierte falsch klassifizierte Punkte
2. Addiere zum Gewichtsvektor
3. Wiederhole so lange bis die Trenngerade richtig klassifiziert



#### Perzeptron-Konvergenz-Satz

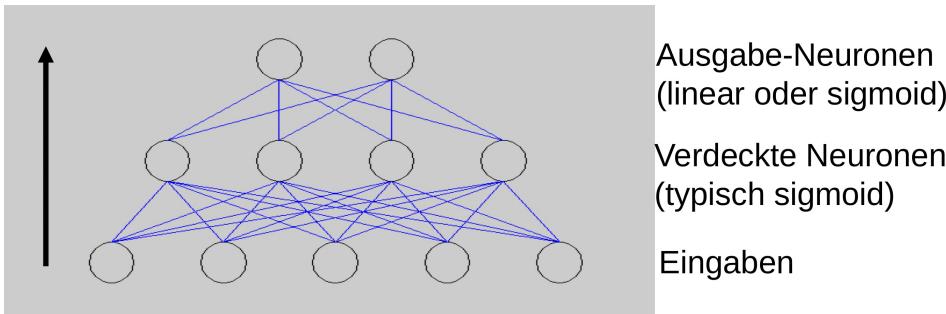
Wenn das Perzeptron eine Klasseneinteilung überhaupt lernen kann, dann lernt es diese mit der Perzeptron-Lernregel in endlich vielen Schritten.

#### Transferfunktion: Fermi-Funktion

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

## Multi-Layer - Perceptron (MLP)

### Perceptrons auf mehreren Schichten



kann beliebige  
Funktionen annähern

### Backpropagation

**Fehlerfunktion:** quadratischer Fehler:  $E_l = \sum_{k=1}^m (y_k - t_k)^2$   
 → soll minimiert werden

**Gradientenabstieg:** die Fehlerfunktion wird entgegen des größten Anstiegs abgestiegen und die Gewichte entsprechend angepasst

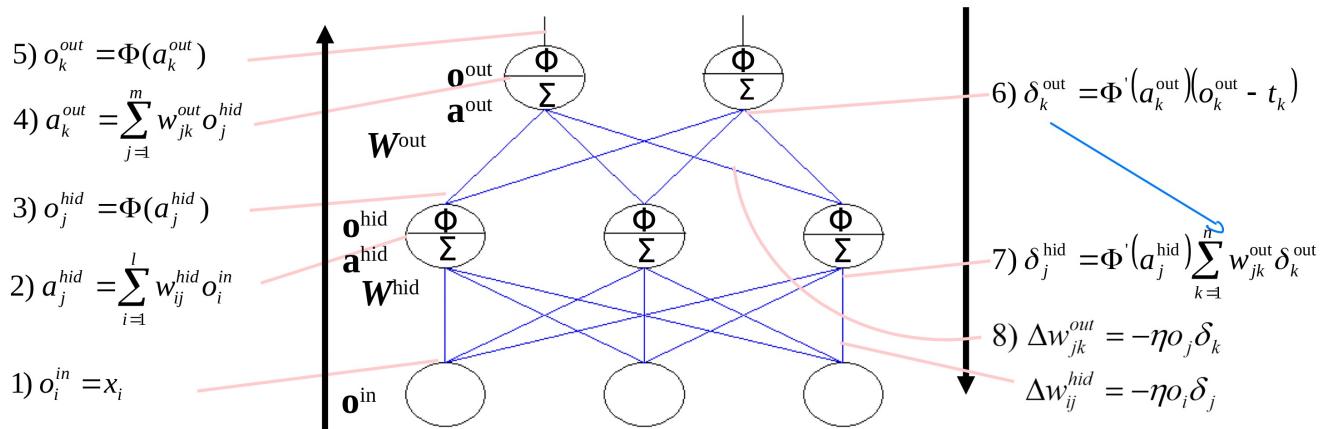
$$\nabla E = \left\| \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_m} \right\|$$

$$w_i^{new} = w_i + \Delta w_i$$

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

**Ablauf:** Schritt 1: Feed-Forward  
 Schritt 2: Backpropagation

### Implementierung:



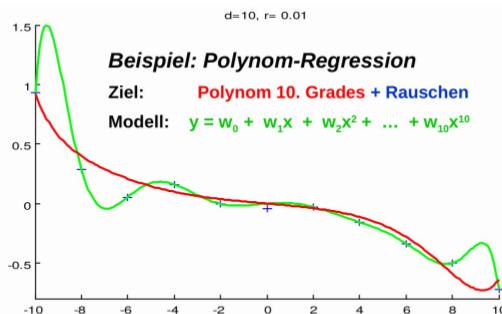
**Delta-Regel:**  $\Delta w_{ij} = \eta \cdot \delta_j \cdot \widetilde{o^{out}_i}$

**Output-Neuron:**  $\delta_m = (\hat{y}_m - y_m) \cdot f'(net_m)$

**Hidden-Neuron:**  $\delta_h = (\sum_{k=1}^K \underline{\delta}_k w_{hk}) \cdot f'(net_h)$

## 4. Regularisierung

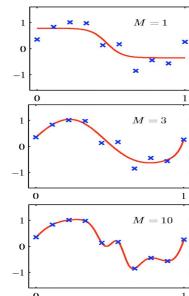
**Overfitting:** Trainingsdaten werden zu stark auswendig gelernt, was keine gute Generalisierung zulässt



**Regularisierung:** Kapazität der Lernmaschine wird beschränkt

### Begrenzung der Hidden Units

- mehr Units heißt stärkere Anpassung an die Datenpunkte
- weniger Units begrenzt die Anpassung an Testdaten
- auch weniger Gewichte

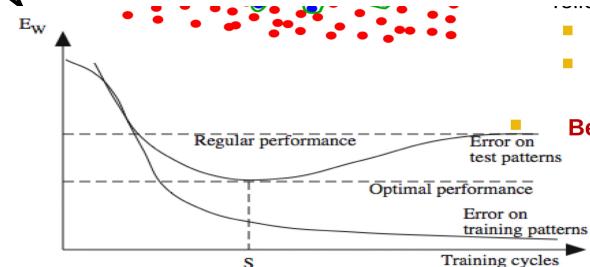


### Early Stopping

- Teile Daten in Testdaten und Trainingsdaten

Beende sobald der Fehler auf den Testdaten wieder steigt

- Wenn die Gewichte klein sind, sind alle Hidden-Units linear



### Weight Decay

**Hebb'sche Regel:**  $W_{ij} \leftarrow W_{ij} + y_i X_j$

**Weight Decay:**  $W_{ij} \leftarrow (1-\gamma) W_{ij} + y_i X_j$   $\gamma \in [0, 1]$ , Decay-Parameter

- Gewichte werden kleiner, wenn sie nicht verstärkt werden

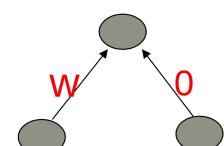
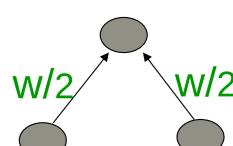
Für MLP:  $W_{ij} \leftarrow (1-\gamma) W_{ij} + \text{back\_prop}(i,j)$

### Begrenzung der Gewichtswerte

Kostenfunktion + Term der hohe Gewichte bestraft;

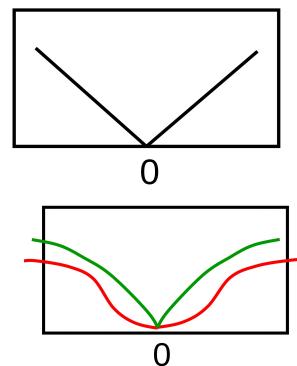
$$C = E + \frac{\lambda}{2} \sum_i W_i^2$$

- weniger abhängig von Rauschen
- hängt glatter von der Eingabe ab



## Andere Strafterme:

- Bestrafte Absolutwerte  
→ reduziert viele Gewichte auf 0
- Bestrafte große Gewichte nicht so sehr



## Selektion von Hyper-Parameter

Parameter: Gewichtsvektor  $w$

Hyper-Parameter:  $\gamma, \theta, k$

## K-fache Kreuzvalidierung:

- Lerne aus  $(k-1)$   $(k-1)$  Trainingsbeispielen  
→ Validiere auf  $1/k$  Beispielen
- Mittlerer Validierungsfehler über alle Validierungsteilmengen
- Wähle Hyperparameter, sodass der Kreuzvalidierungsfehler minimal ist und trainiere damit

## Bias - Varianz - Dilemma

→ Varianz kann oft nur verringert werden, wenn das Bias erhöht wird

$$\underbrace{E_D[f(\mathbf{x})-y]^2}_{\text{Erwarteter Fehler f\"ur Trainingsmenge } D \text{ der Gr\"o\se m}} = \underbrace{[E_D f(\mathbf{x})-y]^2}_{\text{Bias}^2} + \underbrace{E_D [f(\mathbf{x})-E_D f(\mathbf{x})]^2}_{\text{Varianz}}$$

- Varianz kann h\"aufig nur verringert werden,

## 5. Radiale Basis-Funktionen Netze

### Radiale Basis-Funktionen Netze

**RBF-Neuronen:** 1. Berechnet Abstände der Eingabe  $x$  zu dem jeweiligen Zentrum

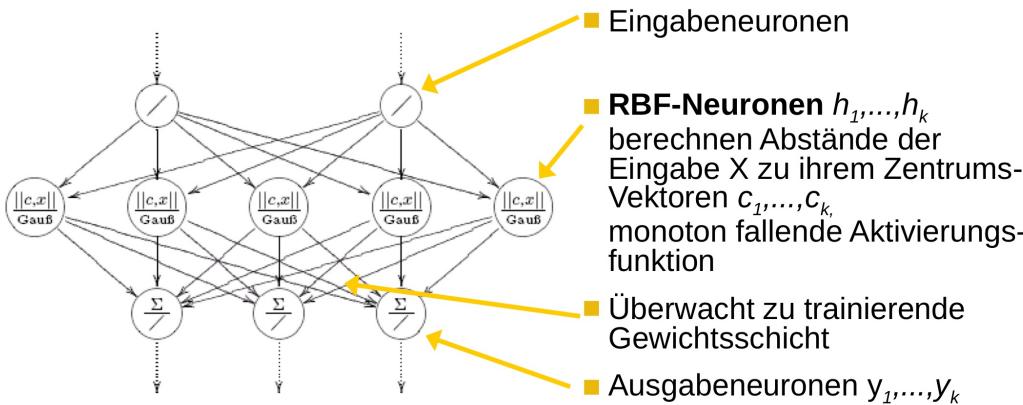
$$r_h = f_{in}(x) = \|x - c_h\| = \sqrt{\sum_{i \in I} (x_i - c_{h,i})^2}$$

2. Aktivierungsfunktion: Gaußfunktion

$$f_{act}(r_h) = e^{\frac{-r_h^2}{2\sigma_h^2}}$$

Ausgabeschicht: linear

nicht-linear  $\rightarrow$  Klassifikation möglich



### Training von RBF-Netzen

Überwachtes Training der Ausgabeschicht:

$$\begin{aligned}\Delta w_{h,\Omega} &= \eta \cdot \delta_\Omega \cdot o_h \\ &= \eta \cdot (t_\Omega - o_\Omega) \cdot f_{act}(\|x - c_h\|)\end{aligned}$$

Kein Backpropagating durch die verdeckte Schicht

### Bestimmung der RBF-Zentren

1. komplett uninformativ

zufällig oder in einem systematischen Raster im Eingaberaum

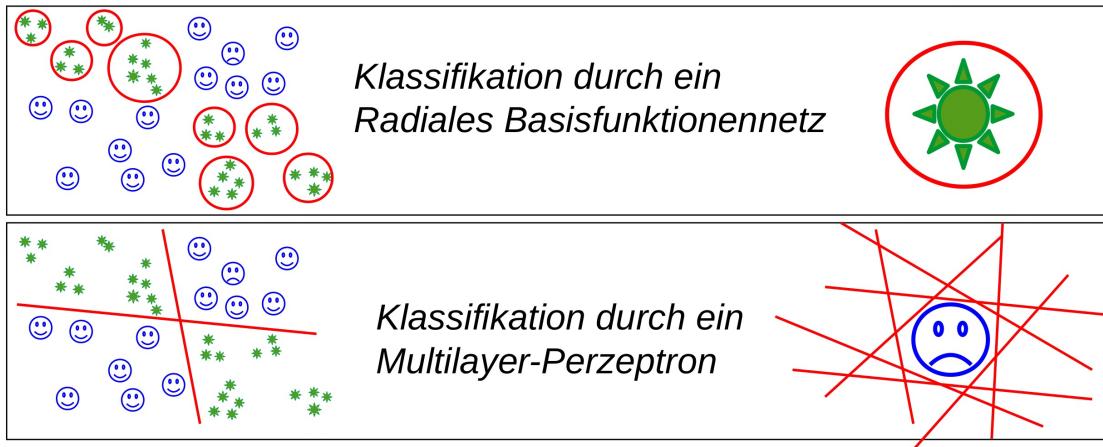
2. abhängig von den Eingabedaten

- Zentren sind Teilmenge der Datenpunkte
- Zentren werden per Clusteranalyse bestimmt (K-means)
- Zentren werden gelernt (unüberwacht, selbst-organisiert)

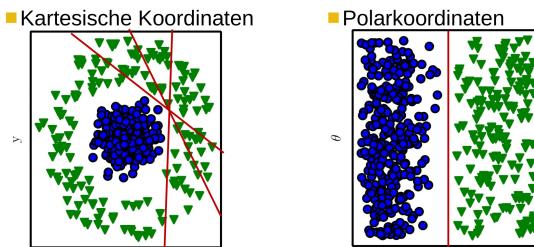
### 3. abhängig vom Fehler

Zentren werden iterativ anhand des Fehlers und Gradientenabstiegs berechnet

### Vergleich RBF-Netze und MLPs



Präsentation ist wichtig:



## 6. Konvolutionale Netze

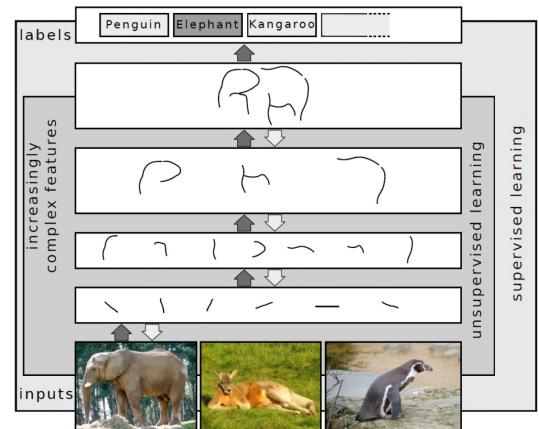
### Deep Learning.

Menge von Algorithmen des maschinellen Lernens, die versuchen geschickte Repräsentationen von Eingabedaten zu lernen

#### Konzept-Ebenen

Konzepte höherer Ebenen basieren auf Konzepten niedrigerer Ebenen

Konzepte niedrigerer Ebenen können zur Definition vieler Konzepte höherer Ebenen verwendet werden



### Flache vs. tiefe Netze

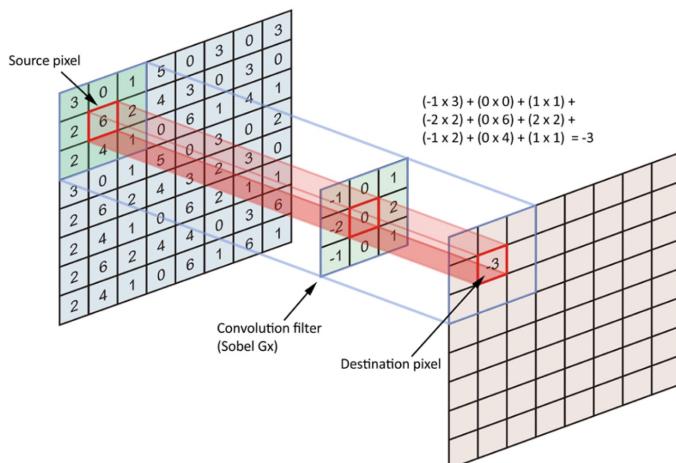
Tiefe Netzwerke können exponentiell effizienter sein

Flache Netze (1 Schicht) können jede Funktion berechnen. Manche Funktionen benötigen exponentiell viele Hidden Units

### Konvolutionale Netze:

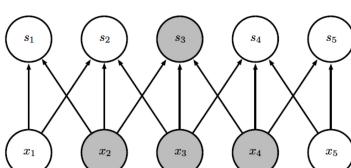
Nutzen in jeder Schicht Sequenzen von räumlich begrenzten Konvolutionen / Faltungen

#### Konvolution:

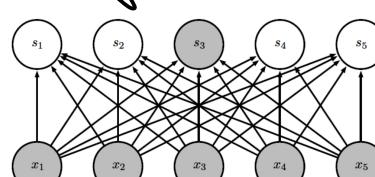


### Sporadische, Lokale Konnektivität; Weight sharing

#### 1D-Konvolution:

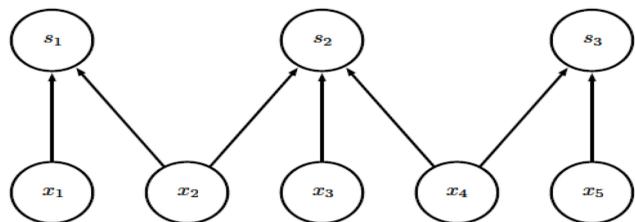


#### Vollständig vernetzt.

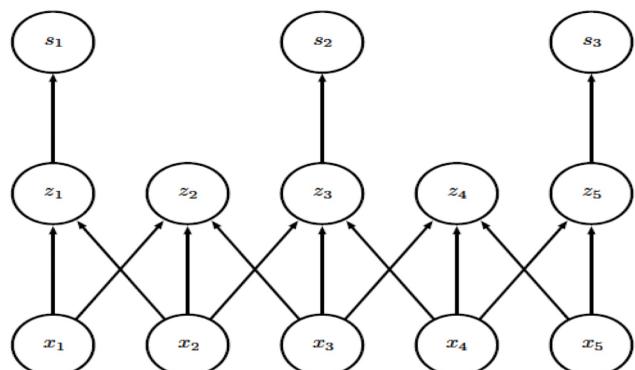


## Schrittweite (Stride)

Konvolution mit Schrittweite 2:



Konvolution mit Subsampling:



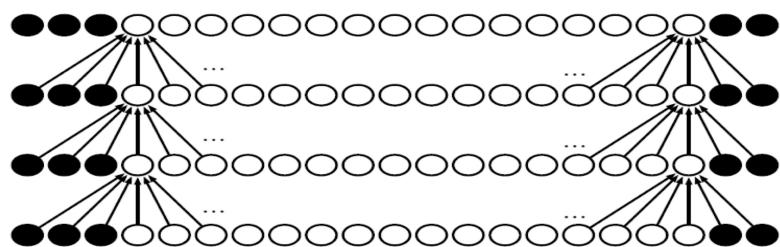
## Border Padding

→ Auffüllen des Rands

1. Zero Padding: Rand wird mit 0 aufgefüllt

2. Spiegeln von Aktivitäten am Rand

3. Kopieren von Aktivitäten

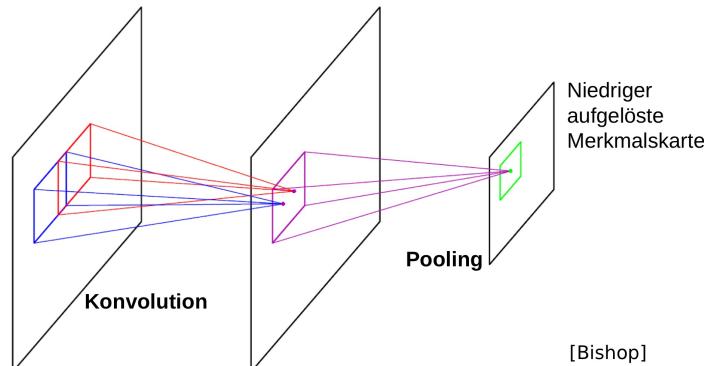


## Pooling

Abilden eines Bereiches auf eine niedriger aufgelöste Merkmalskarte

Max-Pooling: Nimm den maximalen Wert aus dem Bereich

$$o'_{i,j} = \max(o_{2i,2j}, o_{2i+1,2j}, o_{2i,2j+1}, o_{2i+1,2j+1})$$



## Einige wichtige Prinzipien

**Occam's Razor:** Gibt es zwei gleichwichtige Modelle, dann nimm das einfacherere

**Fluch der Dimensionen:** Der Bedarf an Beispielen steigt exponentiell mit der Zahl der Merkmale  
→ nur Merkmale die wichtig sind

**No free lunch:** kein Lerner kann für alle Probleme die beste Lösung liefern. Man braucht Wissen über die Daten

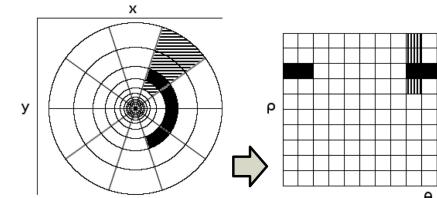
## Beschränkung konvolutionaler Verarbeitung

1. alle Bildpositionen werden auf gleiche Weise behandelt  
 → Netzwerk Ortsinformation geben

Kodierung der Bildkoordinate



2. keine Invarianz gegenüber Skalierung, Drehung  
 → log-polare Vorverarbeitung



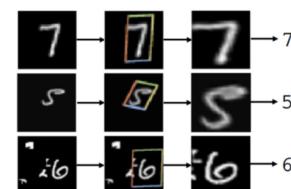
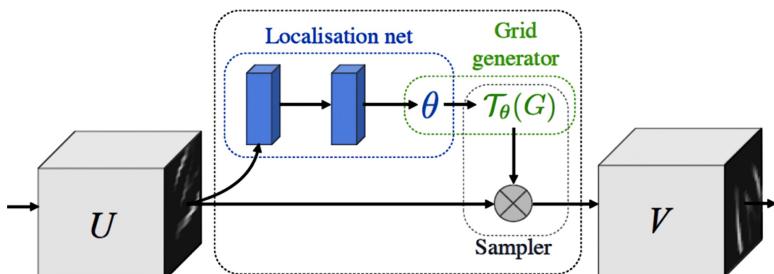
3. kein Fokus der Aufmerksamkeit  
 → Ausschneiden von Regionen



## Spatial Transformer Networks

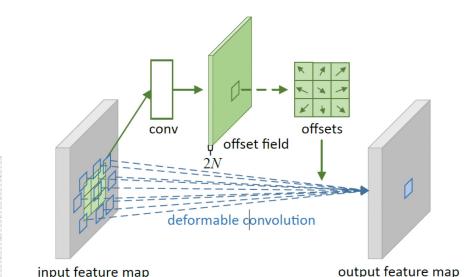
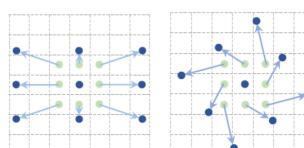
Soll räumliche Verzerrungen & Drehung rausrechnen

- parametrisierte Bildtransformation  
 → Lokalisierungsnetzwerk schätzt Transformationsparameter
- Grid-Generator berechnet Ursprungskoordinaten
- Sampler führt Transformation aus



## Deformierbare konvolutionale Netzwerke

Verzerrungen o.ä. sollen in einer Schicht rausgerechnet werden



## Semantische Segmentierung

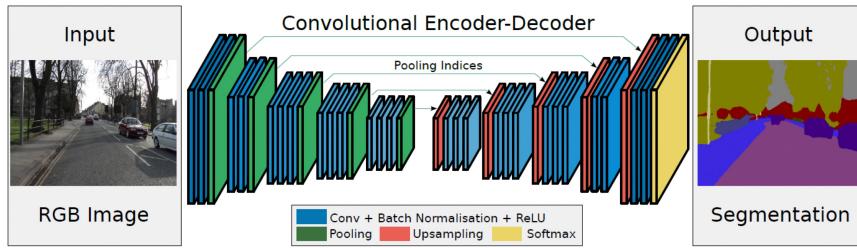
Jeder Bildpunkt wird einer Klasse zugordnet



## Fully Convolutional Networks

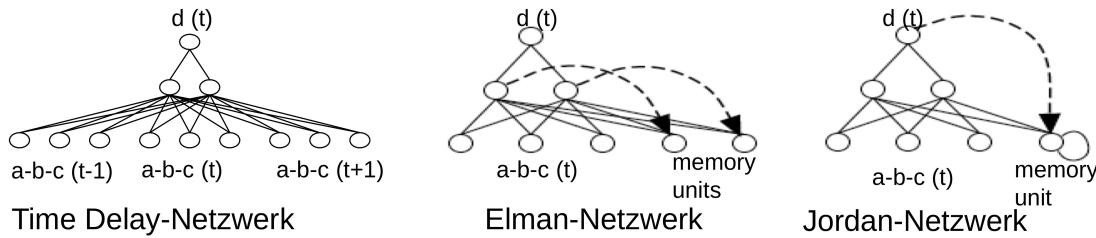
große Auflösung von CNN. Jeder Pixel muss klassifiziert werden.

→ Pooling durch Upsampling rückgängig machen



## 7. Rekurrente Netze

### Verarbeitung von Zeitserien:



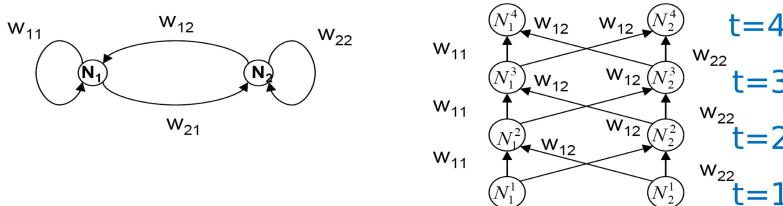
### Eigenschaften

- es kann oszillieren
- kann gegen Attraktoren konvergieren
- kann sich chaotisch verhalten
- Dingen können geworkt werden
- sequentielle Daten können einfach geworkt werden

### Backpropagation through time (BPTT)

→ Methode zum Training rekurrenter Netze

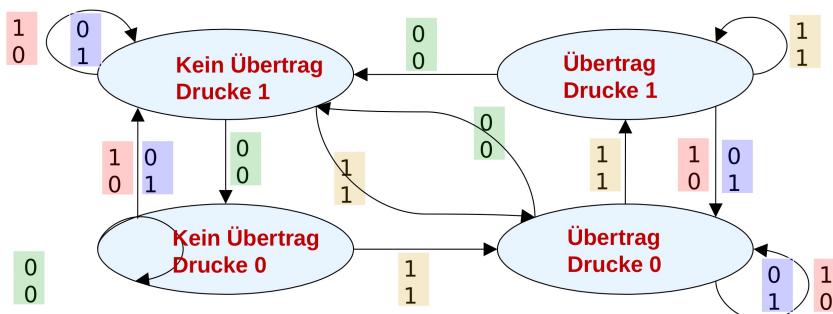
Netz wird entlang der Zeitachse entfaltet und so in ein vorwärtsgerichtetes Netz umgewandelt



$$o_i(t+1) = f(\text{net}_i = f(\sum_j w_{ij} o_j(t) + x_i(t)) )$$

Die Updates der geteilten Gewichte werden gemittelt

### Beispiel: Binäre Addition

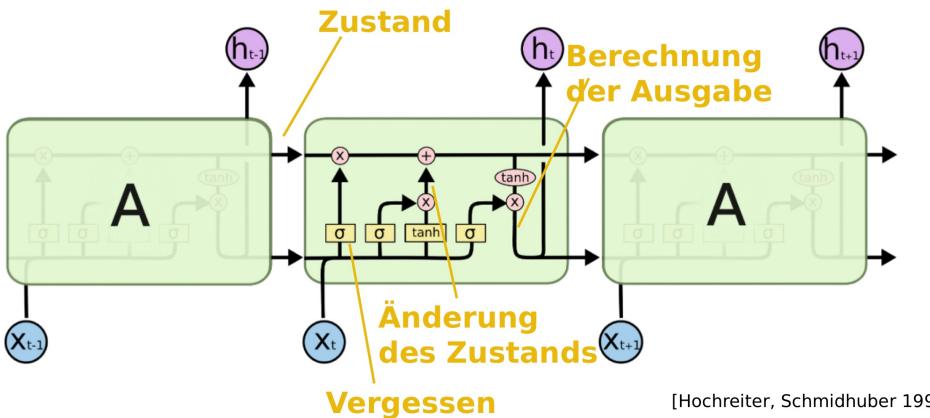


→ Verarbeitung soll an jeder Stelle der Eingabe gleich sein

## Long short-term Memory (LSTM)

Modellierung längerfristiger Abhängigkeiten

Idee: Verdrahtete Speicherung des Zustands, kontrollierte Änderungen



## Hopfield-Netze

Hopfield-Netze realisieren Assoziativspeicher

→ physikalisch motiviert durch Elementarmagnete, die sich nach dem energetisch günstigsten Zustand ausrichten

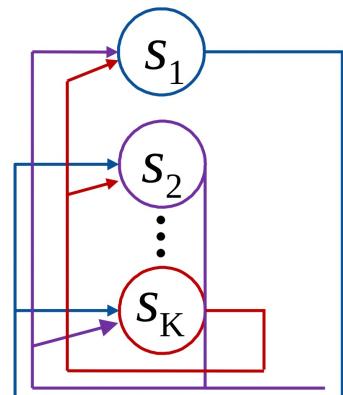
Eigenschaften:

- vollständig vernetzte binäre Units
- symmetrische Gewichte:

$$w_{ij} = w_{ji}, \quad w_{ii} = 0$$

Update Regel:

$$s_i \leftarrow \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$



## Berechnungsreihenfolge

Asynchrone Aktualisierung:

in jedem Zeitpunkt ändert sich nur ein zufällig ausgewähltes Neuron

→ ursprüngliche Propagierungsregel

Synchrone Aktualisierung:

alle Neuronen ändern ihren Zustand gleichzeitig

Zustandsraum:  $k$  Neuronen  $\rightarrow 2^k$

## Einprägen der Muster

Wahl der Gewichte richtet sich nach dem zu speichernden Muster:  $\vec{x}_1, \dots, \vec{x}_p$

Gewichte werden auf die durchschnittlichen gemeinsamen Einse[n] der Muster gesetzt:

$$w_{ij} = \frac{1}{n} \sum_{m=1}^p x_{mi} x_{mj}$$

$x_{mi}$  ist die  $i$ -te Komponente des Eingabemusters  $\vec{x}_m$

Maximale Speicherkapazität:  $|P|_{MAX} \approx 0.14 \cdot |K|$

## 8. Unüberwachtes Lernen

### Unterschiede:

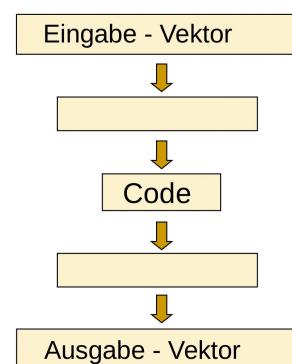
**Überwachtes Lernen:** Lerne mit Teacher-Wert aus einer Eingabe eine Ausgabe vorherzusagen

**Unüberwachtes Lernen:**

- Lerne generatives Modell, was Merkmale erkennt
- erkenne nützliche Strukturen
- erzeuge andere Repräsentationen für überwachtes Lernen
- beschleunigtes Lernen in hochdimensionalen Räumen

### Auto-Encoder

Eingabe wird durch einen Flaschenhals gepresst, wodurch häufig unabhängige Merkmale extrahiert werden

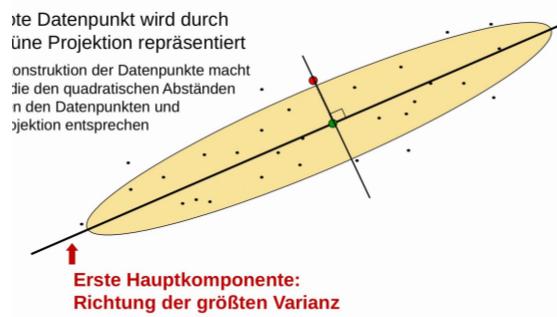


### Principal Components Analysis (PCA)

Extrahiert aus  $N$ -dimensionalen Datenpunkten die  $N$ -Hauptkomponenten. Das sind  $M \leq N$  orthogonale Richtungen, in die die Daten die größte Varianz haben.

Diese Komponenten sind also am "charakteristischsten" für die Daten

Beispiel:  $N=2$   $M=1$

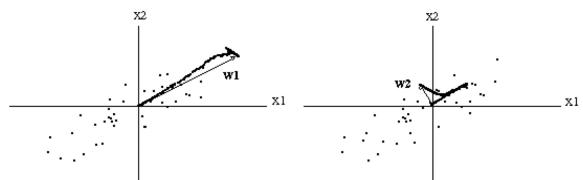
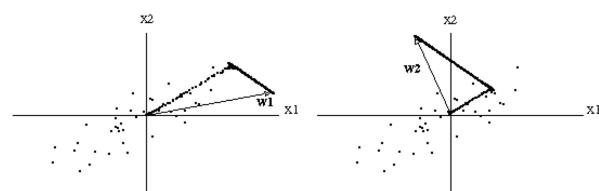


$$\text{Oja-Regel: } \Delta w_{ij} = \eta y_i \frac{\partial}{\partial x_j} - \sum_{k=1}^N w_{kj} y_k \frac{\partial}{\partial w_k}$$

Führt zu Gewichten, die den Unterraum der  $N$  Hauptachsen der Eingabeverteilung ausspannen

$$\text{Sanger-Regel: } \Delta w_{ij} = \eta y_i \frac{\partial}{\partial x_j} - \sum_{k=1}^i w_{kj} y_k \frac{\partial}{\partial w_k}$$

Führt zu Gewichten, die den  $N$  Hauptachsen entsprechen

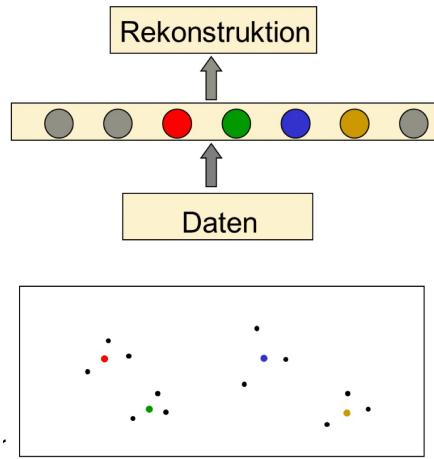


## Gruppierung und Backpropagation

Input → Hidden-Gewicht und  
Hidden → Output-Gewichte sind gleich

⇒ Gradient von Input → Hidden-Gewichte Null

⇒ Backpropagation durch Ausgabeschicht möglich



## Verschiedene Repräsentationen

PCA: gut für verteilte, lineare Daten

Gruppierung: gut für lokal beschränkte nicht-lineare Daten

	Lokal	Verteilt
Linear		PCA
nicht-linear	Gruppierung	Gewünscht

## Hebb'sche und Oja's Lernregel

Hebb'sche Lernregel:  $\Delta w_{ij} = x_j y_i$

$$w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}$$

Problem: unbegrenztes Gewichtswachstum

Oja's Lernregel:  $\Delta w_j = \eta y(x_j - w_j y)$

→ selbsthemmend, nicht-beschränkt

## Gewinner-Neuron

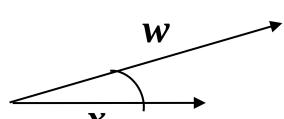
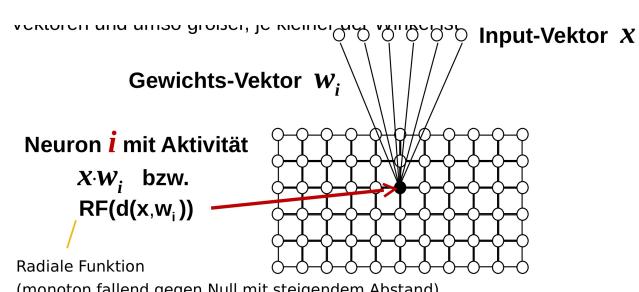
Nur Neuron mit kleinster Distanz wird trainiert

## Abstandsfunktionen:

Minimale Euklidische Distanz.  $\|x-w\|_2 = \sqrt{(x-w)^2}$

Minimale Manhattan Distanz:  $\|x-w\|_1 = \sum_i |x_i - w_i|$

Maximales Skalarprodukt:  $x \cdot w = |x| \cdot |w| \cdot \cos(x, w)$



→ alle Repräsentanten sollten die gleiche Größenordnung haben

→ es wird der Winkel zwischen x und w gemessen

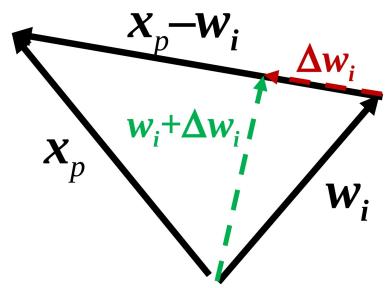
→ Normierung des Gewichtsvektors macht Sinn

## Kompetitive Lernregel

$$\Delta w_i = \eta(x_p - w_i)$$

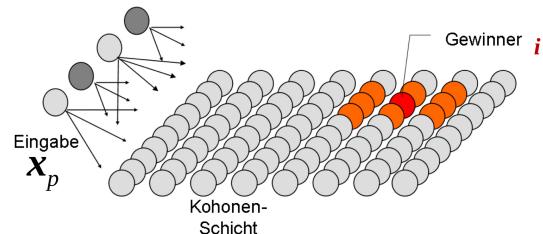
Bewegt die Gewichtsvektoren auf den Eingabevektor zu

→ Gewichte stellen sich auf den Schwerpunkt der Gruppe von Eingabevektoren ein



## Self-Organizing (Feature) Maps

Das SOM ist ein zwei-Schichten-Netz mit  $k$  Neuronen, die in einem Raster über den Eingäberaum verteilt sind



**Gewinner-Neuron:** Das Neuron mit geringster Distanz  $|x_p - w_i|_2$  wird trainiert. Die umliegenden Neuronen werden leicht mittrainiert.

**Gewichtsänderung:** die entsprechenden Gewichte werden in Richtung der Eingabe gezogen:

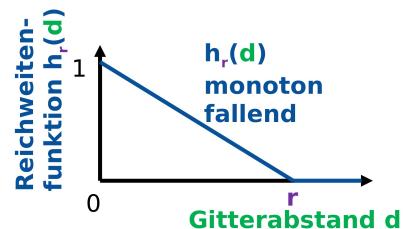
$$\Delta w_k = \eta \cdot (x_p - w_k) \cdot h_r(d(i, k))$$

$\eta$ : Lernrate

$d(i, k)$ : Abstandsmaß im Neuronengitter (ungleich dem Abstand im Eingäberaum)

$h_r(d)$ : Reichweitenfunktion mit Gitterabstand  $r$

→ gibt an wie stark benachbarte Neuronen mittrainiert werden



## 9. Evolutionäre Algorithmen

### Biologischer Hintergrund:

**Phänotyp:** Ausprägung des Organismus

→ Selektion geschieht anhand des Phänotyps

**Genotyp:** Genmaterial eines Organismus

→ wird durch Fortpflanzung übertragen

### Zellteilung (Replikation):

**Mitose:** Replikation während der Entwicklung eines Organismus

**Meiose:** Replikation bei sexueller Fortpflanzung

### Genomgröße:

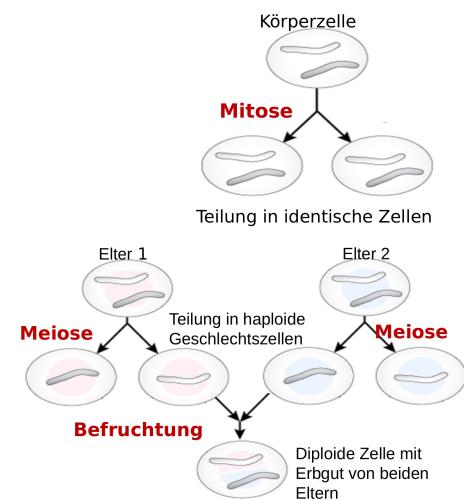
→ kann stark von Art zu Art variieren

**Bestandteile:**

- kodierende DNA
- nichtkodierende DNA

### Typen nichtkodierender DNA:

Repetitive Sequenzen  
Satelliten-DNA  
Genduplikation



### Problemklasse von Evolutionären Algorithmen

#### Optimierungsprobleme:

Suchraum  $S$ ,  $f: S \rightarrow \mathbb{R}$

Gesucht:  $s \in S$ , dass  $f$  optimiert

**Globale Optimierung:** Finde das global beste Optimum

→ schwierig, wenn Suchraum nicht konvex

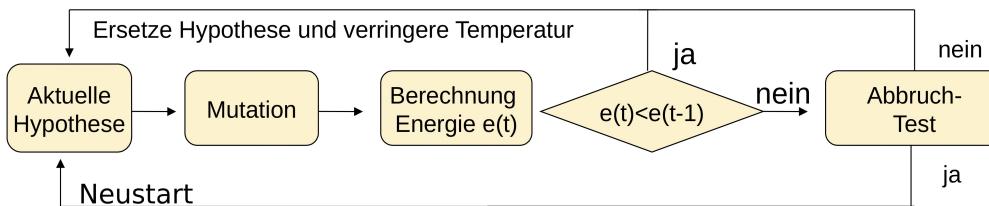
→ neben globalen Optimum kann es viele lokale Optima geben

**Lokale Optimierung:** Finde ein möglichst gutes Extremum in der Nähe des Startpunktes

## Simulated Annealing

→ inspiriert durch Abkühlung von Metallen

Mutiere ein Individuum bis eins mit minimaler Energie gefunden wird.



## Evolutionäre Algorithmen - Grundelemente

Begriff	Biologie	Evolutionärer Algorithmus
Individuum	Lebewesen / Organismus	Punkt im Suchraum
Genotyp	Genetische Kodierung eines Lebewesens	Kodierung eines Punkts im Suchraum
Phänotyp	Äußere Erscheinung eines Lebewesens	Dekodierung / Implementierung eines Punkts des Suchraums
Chromosom	DNS-Strang aus Nukleotiden	Vektor von Informationseinheiten
Allel	Ausprägung eines Gens	Ausprägung einer Informationseinheit
Locus	Position eines Gens	Position einer Informationseinheit
Population	Menge von Lebewesen	Menge von Punkten im Suchraum
Generation	Population zu einem Zeitpunkt	Population zu einem Zeitpunkt
Reproduktion	Erzeugung von Nachkommen durch ein oder mehrere Eltern	Erzeugung von Nachkommen durch ein oder mehrere Eltern
Fitness	Tauglichkeit eines Lebewesens	Güte eines Punktes im Suchraum

**Kodierungsvorschrift:** Repräsentation der Kodierungsvorschrift ist problemspezifisch

**Initialisierungsmethode:**

- erzeugt Ausgangspopulation
- üblicherweise zufällig, bei komplexeren Problemen ggf. spezielle Initialisierung

**Fitnessfunktion:**

- bewertet die Individuum, stellt also die "Umwelt" dar
- häufig identisch mit der Zielfunktion
- Sollte immer maximiert werden

**Selektionsoperator:** Wählt auf Basis der Fitnessfunktion Individuen zur Reproduktion aus

**Reproduktionsoperator:** Erzeugt Nachkommen durch Variation der genetischen Information eines oder mehrere Eltern

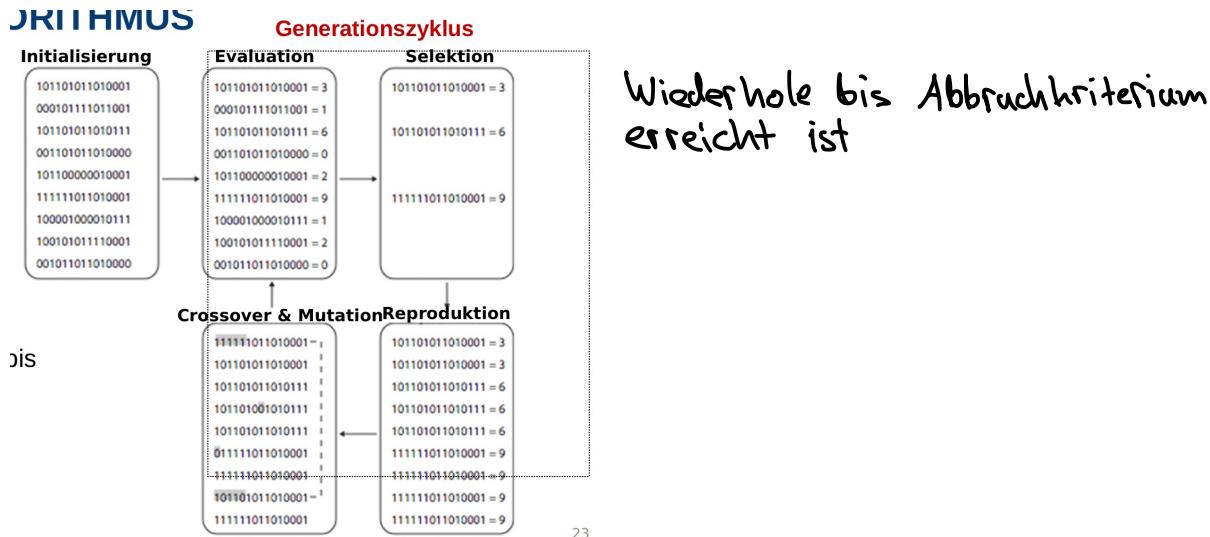
**Crossover:** Rekombination von Chromosomen

**Mutation:** Zufällig Variation einzelner Gene

## Abbruchkriterium: Bestimmt das Ende des EA

- maximale Anzahl an Generationen
- Mindestgröße der Lösung erreicht
- Stagnation der Optimierung

## Schema EA:



## Genetische Operatoren:

→ müssen entsprechend der Kodierung gewählt werden

### Ein-Elter-Operatoren: (werden als Mutation bezeichnet)

**Standardmutation:** Variiere Ausprägung eines Gens des Chromosoms

2	4	3	2	6	1
2	4	3	2	5	1

**Mutationswahrscheinlichkeit:**  $p_m, 0 < p_m \ll 1$

Für Bit-String:  $p_m = 1/\text{length}(x)$

**Zweiertausch:** Tausche Ausprägung zweier Gene des Chromosoms

2	4	3	2	6	1
3	4	2	2	6	1

→ parallel für  $3, \dots, n$  Gene

→ Menge getauschter Gene muss gleich sein

**Permutation eines Teilstücks:** Mische Gene eines Teilstücks

2	4	3	2	6	1
2	4	2	3	1	6

**Inversion eines Teilstücks:**

2	4	3	2	6	1
2	4	3	1	6	2

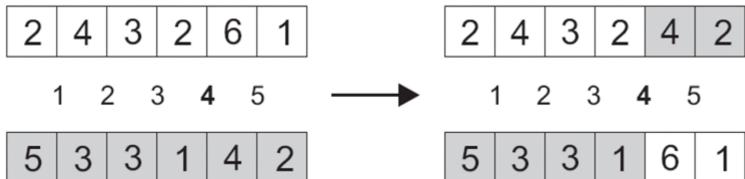
## Verschiebung eines Teilstücks:



## Zwei-Elter-Operatoren

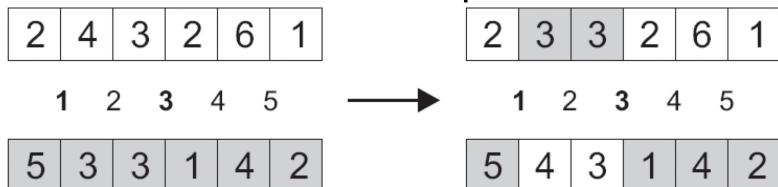
### Ein-Punkt Crossover:

- Wähle zufälligen Schnittpunkt und tausche ab dort die Gene



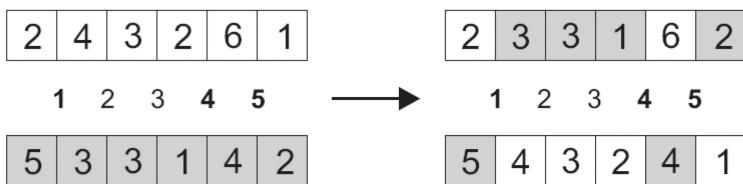
### Zwei-Punkt Crossover:

- Wähle zufällig 2 Schnittpunkte und tausch Gene dazwischen



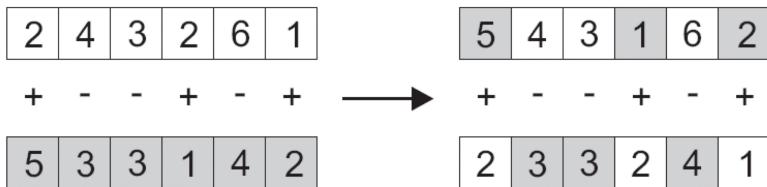
### n-Punkt Crossover:

- Wähle n zufällige Schnittpunkte und tausche Gene nach jedem zweiten Schnittpunkt



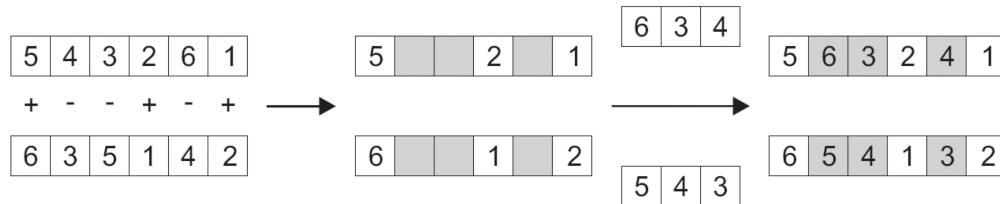
### Uniformes Crossover

- Für jedes Gen wird zufällig gewählt, ob es getauscht wird



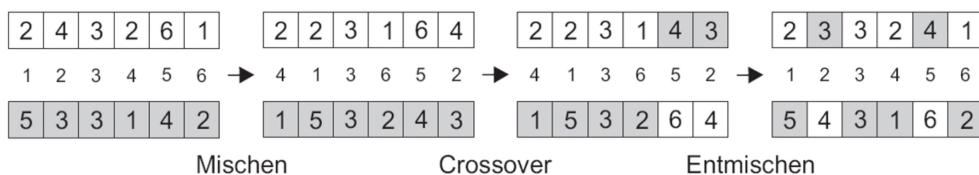
## Uniformes ordnungsbasiertes Crossover

- entscheide für jedes Gen zufällig, ob es getauscht wird
- setze es in der Reihenfolge wieder ein, wie es im anderen Chromosom vorkommt



## Shuffle Crossover

- Chromosomen werden gewischt, dann wird ein 1-Punkt Crossover vollzogen und anschließend wieder entmischt



## Ortsabhängige Verzerrung:

Wahrscheinlichkeit zwei Gene zusammen ist abhängig von der relativen Lage

→ unerwünschter Effekt, da der Erfolg des EA abhängig von der Anordnung der Gene ist

## Kodierung:

→ kein Verfahren für eine gute Kodierung

## Eigenschaften einer guten Kodierung

- ähnlich Lösungshandikaten sollten ähnliche Fitness haben
- ähnliche Genotypen sollten ähnliche Phänotypen repräsentieren
- Suchraum sollte unter den verwendeten genetischen Operatoren abgeschlossen sein

## Starke Kausalität der Fitness

Ähnliche Lösungshandikaten sollen ähnliche Fitness haben

→ Grundlage für die Anwendbarkeit EA

## Epistase: Wechselwirkung zwischen Genen eines Chromosoms

→ Allel eines epistatischen Gens unterdrückt die Wirkung aller möglichen Ausprägungen eines oder mehrerer anderer Gene

### Kodierung mit hoher Epistase

- Genetische Operatoren führen zu fast zufälligen Fitnessänderungen
- Schwierigkeiten Zusammenhang zwischen Genom und Fitness

### Kodierung mit sehr geringer Epistase

- einfache Zusammenhänge zwischen Genom und Fitness
- benötigt kein EA; Problem kann durch Zufallsanstieg gelöst werden

## Ahnlichkeit Genotyp - Phänotyp

Ahnliche Genotypen sollten ähnliche Phänotypen entsprechen

Mutationen stellen kleine Änderungen des Genoms dar → ähnliche Genotypen

Sonst: es können keine nahe liegenden Verbesserungen des Phänotyps vorgenommen werden

## Hamming-Ketten

**Problem:** Benachbarte Zahlen können sich in der Kodierung stark kodieren

**Hamming-Abstand:** Anzahl verschiedener Bits

→ große Abstände können schwer überwunden werden

**Lösung:** Kodierung zweier benachbarter Zahlen unterscheidet sich um ein Bit

dezimal	binär	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

**Gray Code:**  $g = b \oplus \lfloor \frac{b}{2} \rfloor$

$$b = \bigoplus_{i=0}^{k-1} \lfloor \frac{g}{2^i} \rfloor$$

## Abgeschlossener Suchraum

Suchraum sollte unter den genetischen Operatoren abgeschlossen sein

**Suchraum gilt als verlossen:**

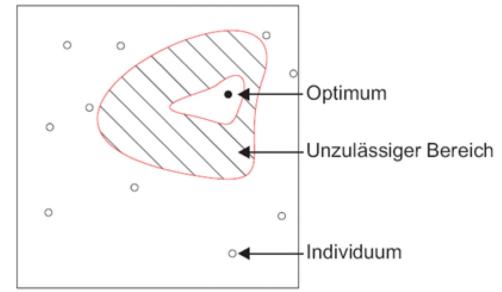
1. Chromosom kann nicht sinnvoll interpretiert / dekodiert werden
2. Lösungskandidat verletzt Nebenbedingung
3. Lösungskandidat kann nicht durch Fitnessfunktion bewertet werden

Reparaturmaßnahmen zum Verhindern

## Nicht zusammenhängende Suchräume

Reparaturmaßnahmen können das Erreichen des Optimums verhindern

→ Strafsterne können sinnvoll sein



## Anwendbarkeit

EA können für vielfältige Probleme genutzt werden

→ Wissen über Problem entscheidet über Design des EA

## Selektion

Bessere Individuen sollen mit höherer Wahrscheinlichkeit Nachkommen haben

Selektionsdruck: Intensität des Konkurrenzhaftes

Konsequenzen:

- reduziert überschüssige Lösungshandikatoren
- Verringert die Diversität der Population
- gibt Optimierung eine Richtung

## Selektionsdruck

Exploration: Erkundung des Suchraums

Suchraum soll möglichst weitläufig abgesucht werden

→ breite Verteilung der Individuen

→ geringer Selektionsdruck

Exploitation: Ausbeutung guter Individuen

Suchraum wird möglichst zielstrebig durchschritten werden, hin zu einem Optimum

→ Individuen sollten möglichst dem Gradienten der Fitness folgen

→ hoher Selektionsdruck

Konsequenzen:

- Zu niedriger Selektionsdruck:
- gute Individuen vermehren sich kaum
  - Zufallsuch
  - langsame oder keine Konvergenz

**zu hoher Selektionsdruck:**

- Variabilität nimmt stark ab
- Superindividuen dominieren

### Zeitabhängiger Selektionsdruck

anfänglich geringer Selektionsdruck, der mit der Zeit zunimmt

→ Selektionsdruck durch Skalierung der Fitnessfunktion oder Art des Selektionsverfahrens

### Dominanzproblem:

Individuum mit hoher Fitness dominieren

Population besteht nur noch aus gleichen oder ähnlichen Individuen

→ Crowding



### Crowding

Vorzeitige Konvergenz gegen lokale Optima

### Varianzproblem

Auswahl der Individuen ist zwar fitnessproportional, aber dennoch zufällig

→ Beste Individuen kommen ggf. nicht in der nächsten Generation vor

### Lösung: Diskretisierung des Fitnesswertbereichs

- |                    |  |
|--------------------|--|
| 0 Nachkommen falls | $f(i) < \mu_f(t) - \sigma_f(t)$                                |
| 1 Nachkomme falls  | $\mu_f(t) - \sigma_f(t) \leq f(i) \leq \mu_f(t) + \sigma_f(t)$ |
| 2 Nachkommen falls | $f(i) > \mu_f(t) + \sigma_f(t)$                                |

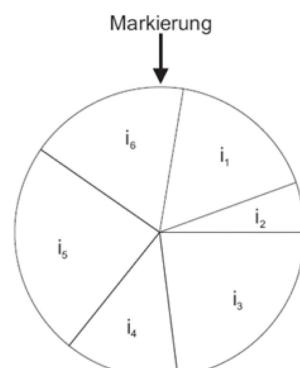
### Glückradsauswahl

Rad wird gedreht für jeden benötigten Nachkommen. Individuum dessen Sektor gedreht wird, wird zur Reproduktion ausgewählt.

**Sektorgröße entspricht relativen Fitnesswerten:**

$$f_{\text{rel}}(i) = \frac{f_{\text{abs}}(i)}{\sum_{i' \in \text{pop}(t)} f_{\text{abs}}(i')}$$

- Fitnessproportionale Selektion
- Fitnesswerte nicht negativ



## Erwartungswertmodell

Erzeuge für jeden Lösungskandidaten  $\lfloor f_{\text{rel}}(i) \cdot \text{popsize} \rfloor$   
und füll den Rest durch Glücksradauswahl  
→ löst Varianzproblem

## Modifizierte Glücksradauswahl

Pro Nachkommen wird die Fitness des Individuums um  $\Delta f$  verringert

Negative Fitness → keine Nachkommen

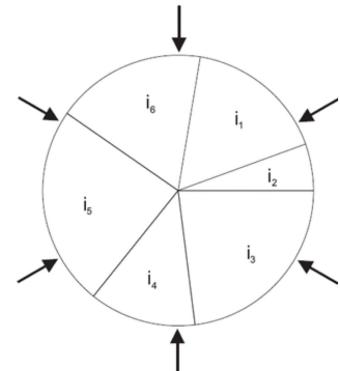
Wahl von  $\Delta f$ , s.d. das beste Individuum höchstens  $k$  Nachkommen hat

$$\Delta f = \frac{1}{k} \max\{f(i) \mid i \in \text{pop}(t)\}$$

## Stochastic Universal Sampling

Glücksrad mit gleichverteilten Markierungen wird einmal gedreht und die markierten Individuen gewählt.

→ überdurchschnittliche Individuen kommen sicher in die Auswahl



## Rangauswahl

- 1 Sortiere Individuen absteigend nach Fitness
2. Rangliste definiert Auswahlwahrscheinlichkeit
- 3 Glücksradauswahl mit der Verteilung

**Vorteile:** • Dominanzproblem verhindern  
• Selektionsdruck steuerbar

**Nachteile:** zusätzlicher Aufwand für Sortierung

## Turnierwahl

1. Wähle  $n$  Individuen als Teilnehmer eines Turniers zufällig
2. Bestes Individuum erzeugt Nachkommen

$$2 \leq n \leq \text{Populationsgröße}$$

$n=2$ : geringer Selektionsdruck

$n=\text{Populationsgröße}$ : maximaler Selektionsdruck

## Elitismus:

**Problem:** Bestes Individuum wird nicht sicher in die nächste Generation übernommen bzw. durch genetische Operatoren modifiziert.  
→ gute Lösungen gehen verloren

**Lösung:** übernehme immer die Elite der Population

## lokaler Elitismus

Elitismus nur zwischen Eltern und ihren Nachkommen

**Mutation:** das Bessere aus Elter und Nachkommen wird gewählt  
**Crossover:** Nur die zwei besten Individuen aus dem Crossover werden gewählt

→ hohe Konvergenzgeschwindigkeit, jedoch eventuell frühzeitig gegen lokale Optima

## Deterministisches Crowding

1. Erzeuge pro Elternpaar ein Nachkommenpaar mittels Crossover und Mutation
2. Bildet Paare aus je einem Elter und dem ihm ähnlichsten Nachkommen
3. Das fittere wird in die nächste Generation übernommen

→ umfasst Selektion

→ Diversität nimmt nur langsam ab

## Shalierungsmethoden

**Zielfunktion:**  $f(x) : \mathcal{X} \rightarrow \mathbb{R}$

→ gibt an wie gut ein Chromosom vordefinierte Ziele erfüllt

**Fitnessfunktion:**  $\Phi(x) : \mathcal{X} \rightarrow \mathbb{R}^+ \quad \Phi(x) = g(f(x))$

→ Chance zur Reproduktion

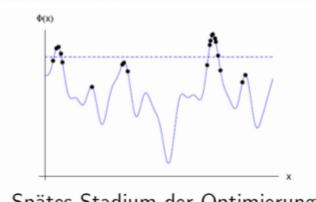
**Minimierung der Zielfunktion:**  $\Phi(x) = \begin{cases} c_{\max_{\text{est}}} - f(x) & \text{falls } c_{\max_{\text{est}}} - f(x) > 0 \\ 0 & \text{sonst} \end{cases}$

→ Selektionsdruck lässt nach, da alle Fitnesswerte hoch sind  
→ Shalierung

Maximierung der Funktion  $\Phi' : \mathcal{X} \rightarrow \mathbb{R}$  ist äquivalent zur Maximierung der Funktion  $\Phi : \mathcal{X} \rightarrow \mathbb{R}$ , da gilt  $\Phi'(x) = \Phi(x) + c, c \in \mathbb{R}$



Beginn der Optimierung



Spätes Stadium der Optimierung

## Linear-dynamische Skalierung

Fitnessfunktion benutzt das Minimum der letzten  $b$  Generationen

$$\Phi_{lds}(x) = \alpha f(x) - \min\{f(x') \mid x' \in \bigcup_{i=t-b}^t \text{pop}(i)\}, \quad \alpha > 0, \quad \alpha > 1$$

## $\Sigma$ -Skalierung

Ziehe unterdurchschnittliche Fitness ab.

$$\Phi_\sigma(x) = f(x) - (\mu_f(x) - \beta \cdot \sigma_f(x)), \quad \beta > 0$$

Mittelwert:  $\mu_f(t) = \frac{1}{\text{popsize}} \sum_{x \in \text{pop}(t)} f(x)$

Standardabweichung:  $\sigma_f(t) = \sqrt{\frac{1}{\text{popsize} - 1} \sum_{x \in \text{pop}(t)} (f(x) - \mu_f(t))^2}$

Wahl von  $\beta$ :  $\beta = 1/\nu^*$        $\nu$  häufig  $\approx 1$

$$\nu = \frac{\sigma_f}{\mu_f} = \frac{\sqrt{\frac{1}{|\mathcal{X}|-1} \sum_{x' \in \mathcal{X}} \left( f(x') - \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} f(x) \right)^2}}{\frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} f(x)}$$

## Boltzmann-Selektion

Exponentiation der Fitness:  $g(x) = \exp\left(\frac{f(x)}{kT}\right)$

Temperatur  $T$  steuert den Selektionsdruck und nimmt ab während der Optimierung

# Evolutionsstrategien (ES)

## Repräsentation

- Vektor reeller Zahlen
- jede Komponente wird als Koordinate eines n-dimensionalen Optimierungsproblems interpretiert

## Genetische Operatoren

- Mutation durch Addition normalverteilter Zufallszahlen
- Rekombination mehrerer Vektoren (kein Crossover)

## Selektion.

Übernahme nur die besten Individuen in die nächste Generation  
→ elitistisch, deterministisch

## Varianten:

„ $+\lambda$ -Strategie“: Wähle aus den  $\mu$  Eltern und  $\lambda$  Nachkommen die  $\mu$  besten Individuen ( $\mu + \lambda$ )  
→ Population kann sich nicht verschlechtern, aber vorzeitig konvergieren

„ $1+1$ -Strategie“: Wähle nur aus den  $\lambda$  Nachkommen die  $\mu$  besten Individuen aus

## Allgemeiner Ablauf:

### Algorithmus

- ① Erzeuge eine zufällige Anfangspopulation von  $\mu$  Individuen
- ② Erzeuge  $\lambda$  Nachkommen. Dazu jeweils:
  - ① Wähle  $\rho$  Eltern zur Erzeugung eines Nachkommen aus
  - ② Rekombiniere diese Eltern zu einem neuen Individuum
  - ③ Mutiere das Individuum
- ④ Bewerte die Individuen gemäß ihrer Fitness
- ⑤ Selektiere die  $\mu$  Eltern der nächsten Population
- ⑥ Gehe zu Schritt 2 bis ein Abbruchkriterium erfüllt ist

## (1+1)-Evolutionsstrategien

Ein Elter erzeugt ein Nachkommen → Zufallsabstieg

## Ablauf:

### Algorithmus

- ① Erzeuge eine zufällige „Anfangspopulation“, d.h. Startpunkt  $\vec{x}^0$
- ② Erzeuge einen normalverteilten Zufallsvektor  $\vec{z}$  und bestimme den Nachkommen als  $\vec{x}' = \vec{x}^t + \vec{z}$
- ③ Berechne die Fitness der Individuen mit der Zielfunktion
- ④ Selektiere den Elter der nächsten Population

$$\vec{x}^{(t+1)} = \begin{cases} \vec{x}^{(t)} & \text{falls } f(\vec{x}^{(t)}) \leq f(\vec{x}') \\ \vec{x}' & \text{sonst} \end{cases}$$

- ⑤ Gehe zu Schritt 2 bis ein Abbruchkriterium erfüllt ist

## Reproduktion

Wähle uniform  $1 \leq \rho \leq M$  Eltern aus der Population, die durch Rekombination Nachkommen erzeugen

$\rho > 2$ : Multirekombination

→ Eltern können mehrfach gewählt werden

## Rekombination:

Mische die genetischen Informationen der  $\rho$  gewählten Eltern

### Intermediäre Rekombination

Nachkomme ist der Schwerpunkt der Eltern

$$\vec{r} := \frac{1}{\rho} \sum_{i=1}^{\rho} \vec{v}_i$$

### Discrete Rekombination

Wähle jede Komponente zufällig aus einem der Eltern

$$\vec{r} := \sum_{i=1}^{\dim \vec{x}} (\vec{e}_i^T \vec{v}_{m_i}) \vec{e}_i \quad m_i := \text{rand}\{1, \dots, \rho\} \text{ gleichverteilte Zufallszahlen, } \vec{e}_i \text{ Einheitsvektor in } i\text{-ter Dimension}$$

## Mutation

Variiere genetische Information durch Addition normalverteilter Zufallszahlen

→ kleine Veränderungen mit variabler Stärke der Mutation

## Normalverteilte Mutation

kleine Varianz: kleine Veränderungen  
⇒ lokale Suche

große Varianz: große Änderungen  
⇒ globale Suche

## Anpassen der Mutationsstärke

1/5 Erfolgsrate:

Erfolgsrate  $p_s$ : Wahrscheinlichkeit, dass ein Nachkommen ein Elter ersetzt

→  $p_s = 1/5$  für gute Konvergenzgeschwindigkeit

Globale Mutationsstärke (Standardabweichung) wird so gewählt, dass die Konvergenzgeschwindigkeit maximal ist

## Definition

$$\sigma^{(g+1)} = \begin{cases} c \sigma^{(g)}, & \text{if } p_s < 1/5 \\ \sigma^{(g)}, & \text{if } p_s = 1/5 \\ \sigma^{(g)}/c, & \text{if } p_s > 1/5, \end{cases}$$

- Die Wahl von  $c$  hängt von der Zielfunktion, der Dimension und der Anzahl der Generationen ab
- Empfehlung von Rechenberg:  $0.85 \leq c \leq 1$

## Selbstadaptation

Alle Individuen besitzen eigene Strategieparameter, die mit optimiert werden ( $\vec{x}, \vec{s}$ )

→ Strategieparameter werden im Chromosom kodiert und beim Erzeugen von Nachkommen zuerst erzeugt und damit die Objektparameter erzeugt

## Varianten von Mutationsoperatoren

### Isotropische Mutation:

Eine einzige Schrittweite für alle Dimensionen des Optimierungsproblems

### Skalierte Mutation:

Eine eigene Schrittweite für jede Dimension

### Korrelierte Mutation:

Variiere neben den Mutationsschrittweiten auch die Korrelation der Zufallsvariablen

### Gerichtete Mutation:

Bevorzugung von bestimmten Richtungen durch Einsatz schiefsymmetrischer Verteilungen

## Unterschiede Evolutionsstrategien vs. Genetische Algorithmen

	ES	GA
Darstellung	reellwertig	binär
Selektion	deterministisch, elitär	stochastisch
Reproduktion	Rekombination, Mutation	Crossover, Mutation
Mutation	häufig	selten
Selbstadaptation	+	-

## Genetische Programmierung

### Repräsentation

- Individuen repräsentieren Funktionen oder Programme
- Komplexe Chromosomen variabler Länge

### Genetische Operatoren

- Mutation durch Ersätzen von Teilbäumen
- Crossover durch Austausch von Teilstücken verschiedener Individuen

### Selektion: wie bei GA

## Repräsentation

### Grundlage: Formale Grammatiken

$T$  : Menge der Terminalsymbole

$F$  : Menge der Funktionssymbole

→ Wahl problemspezifisch

### Terminalsymbole

- Benutzereingaben

- Konstanten

→ Zufallszahlen, Veränderliche Konstanten

### Funktionssymbole

- Funktionen

- Operatoren

- Kontrollstrukturen

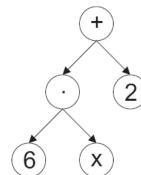
## Kodierung

Gene der Chromosome enthalten Elemente der Mengen

### Darstellung durch symbolische Ausdrücke in Präfixnotation:

Beispiele:

- $+ \cdot 6 \times 2$  ist ein symbolischer Ausdruck, der  $6 \cdot x + 2$  repräsentiert.
- $4 \cdot 1 x -$  ist kein symbolischer Ausdruck

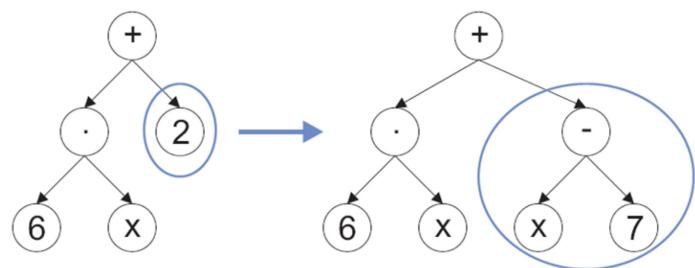


## Mutation

Erssetze Teilausdruck durch einen zufällig erzeugten Neuen

→ Mutation sollte selten sein

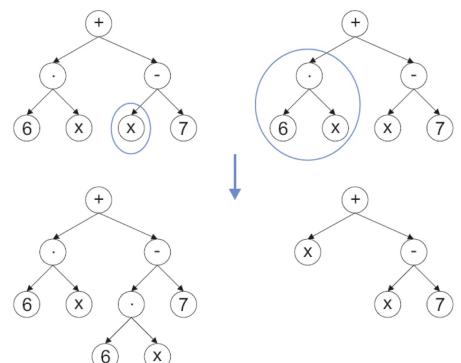
→ ersetzenende Teilbäume möglichst klein



## Crossover

Tausche Teilausdrücke verschiedener Individuen

→ Variantenreicher



## Ablauf:

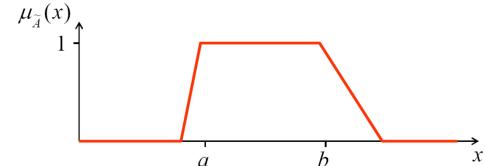
### Algorithmus

- ① Erzeuge eine zufällige Anfangspopulation symbolischer Ausdrücke Parameter:
  - Maximale Verschachtelungstiefe
  - Wahrscheinlichkeiten der Terminalsymbole
- ② Bewerte die Ausdrücke gemäß ihrer Fitness
  - Symbolische Regression: Summe der Fehlerquadrate zu allen Datenpunkten
  - Boolesche Funktionen: Anteil korrekter Ausgaben für die Menge der Eingaben
- ③ Selektiere die Individuen der Zwischenpopulation
- ④ Wende die genetischen Operatoren an
- ⑤ Gehe zu Schritt 2 bis ein Abbruchkriterium erfüllt ist

# 10 Fuzzy Logic, Fuzzy Control

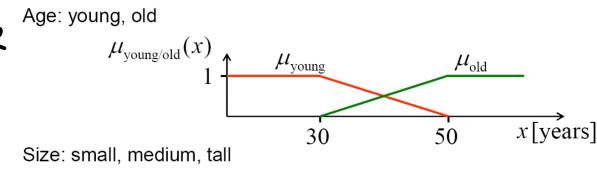
## Modellierung unscharfer Konzepte

Fuzzy-Mengen: werden durch Mitgliedschaftsfunktionen beschrieben  $\mu_{\tilde{A}}(x) \in [0,1]$



Linguistische Variablen: Domäne der Fuzzy-Menge

Linguistische Werte: „Diskretisierung“ der Domäne durch Fuzzy-Mengen



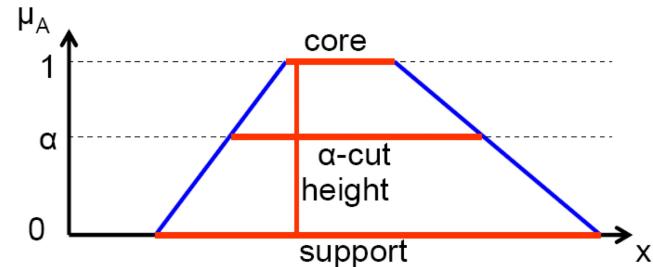
## Fuzzy-Mitgliedschaftsfunktionen $\mu(x)$

Support: Menge mit  $\mu(x) > 0$

Core: Menge mit  $\mu(x) = 1$

$\alpha$ -Cut: Menge mit  $\mu(x) \geq \alpha$

Height: Maximale Mitgliedschaft



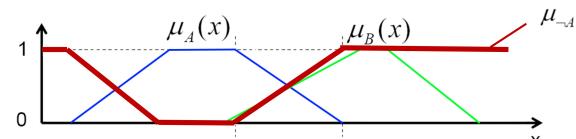
## Fuzzy-Operatoren

### Min/Max - Norm

Konjunktion  $\mu_{A \wedge B}(x) := \min\{\mu_A(x), \mu_B(x)\}$

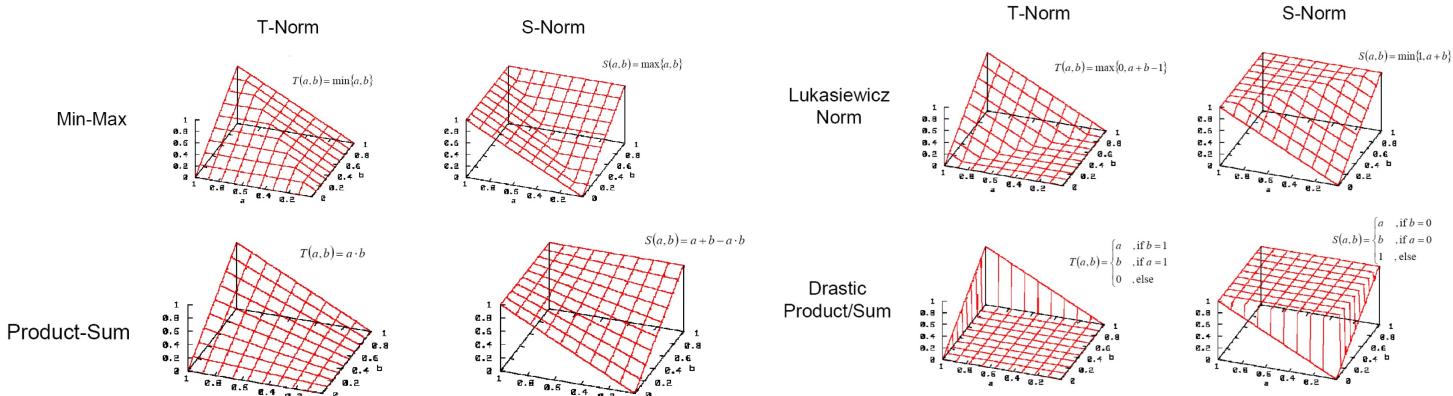
Disjunktion  $\mu_{A \vee B}(x) := \max\{\mu_A(x), \mu_B(x)\}$

Negation:  $\mu_{\neg A}(x) := 1 - \mu_A(x)$



T-Norm: Operatoren für Konjunktion

S-Norm: Operatoren für Disjunktion



## Fuzzy-Implikation

$$\mu_{A \rightarrow B}(x) := \max_{*v^*} \frac{1 - \mu_A(x)}{\neg A} \frac{\min\{\mu_A(x), \mu_B(x)\}}{A \wedge B}$$

$$\mu_{A \rightarrow B}(x) := \min\{1, 1 - \mu_A(x) + \mu_B(x)\}$$

## Fuzzy-Folgerung:

**Modus Ponens:**

$$\begin{array}{c} A \rightarrow B \\ A \quad (\text{gegeben}) \\ \hline B \quad (\text{folgern}) \end{array}$$

$$\begin{array}{c} A \rightarrow B \\ \neg A \\ \hline ? \end{array} \quad \begin{array}{c} A \rightarrow B \\ \mu_{A^c}(x) \\ \hline \mu_{B^c}(y) \end{array}$$

$$\begin{array}{c} A \rightarrow B \\ \mu_{\neg A^c}(x) \\ \hline ? \end{array}$$

**Modus Tollens:**

$$\begin{array}{c} A \rightarrow B \\ \neg B \\ \hline \neg A \end{array} \quad \begin{array}{c} A \rightarrow B \\ \mu_{\neg B^c}(y) \\ \hline ? \end{array}$$

## Fuzzy-Regeln

**Regel:** IF <Prämissen> THEN <Konklusion>

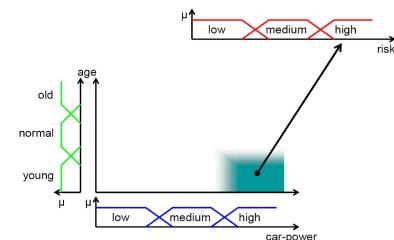
**Mamdani-Regeln:**

Prämissen: Konjunktion von Fuzzy-Mitgliedschaften  
Konklusion: Fuzzy-Menge

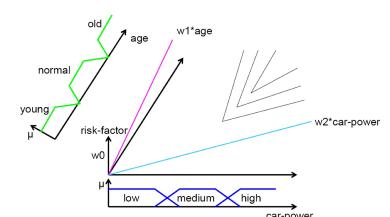
**Takagi-Sugeno-Regeln:**

Prämissen: Konjunktion von Fuzzy-Mitgliedschaften  
Konklusion: Reellwertige Funktion vom Grad 0-2

IF age IS young AND car-power IS high THEN risk IS high

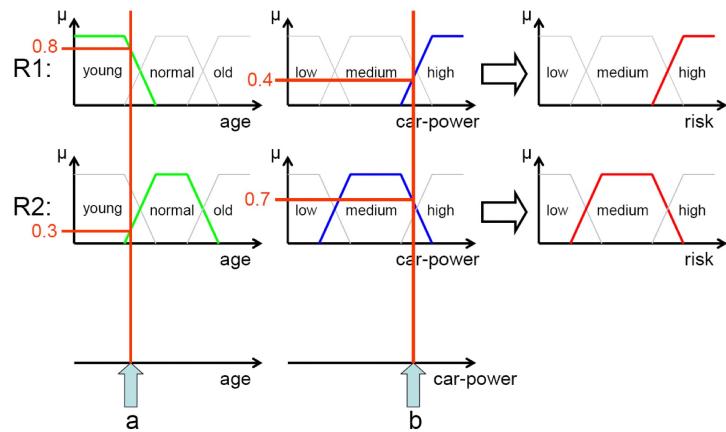


IF age IS young AND car-power IS high  
THEN risk-factor = w0 + w1 \* age + w2 \* car-power

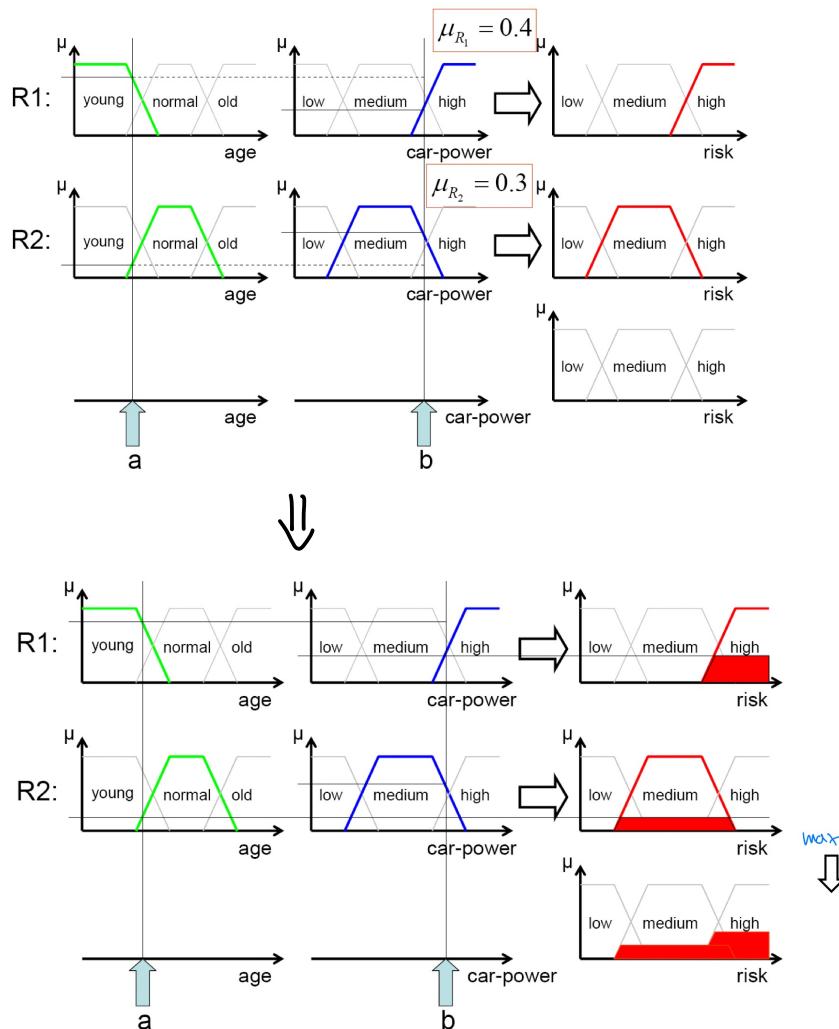


# Fuzzyfizierung scharfer Eingaben

## Einzelne Punkte raus suchen



## Inferenz mit Min/Max-Norm



## Defuzzifizierung:

Schwerpunkt:

$$y = \frac{\int y \cdot \mu_{\text{risk}}(y) dy}{\int \mu_{\text{risk}}(y) dy}$$

Approximation: Gewichtete Summe

$$y = \frac{\sum_{j=1}^r \mu_j \cdot s_j}{\sum_{j=1}^r \mu_j}$$

## Fuzzy - Inferenz

