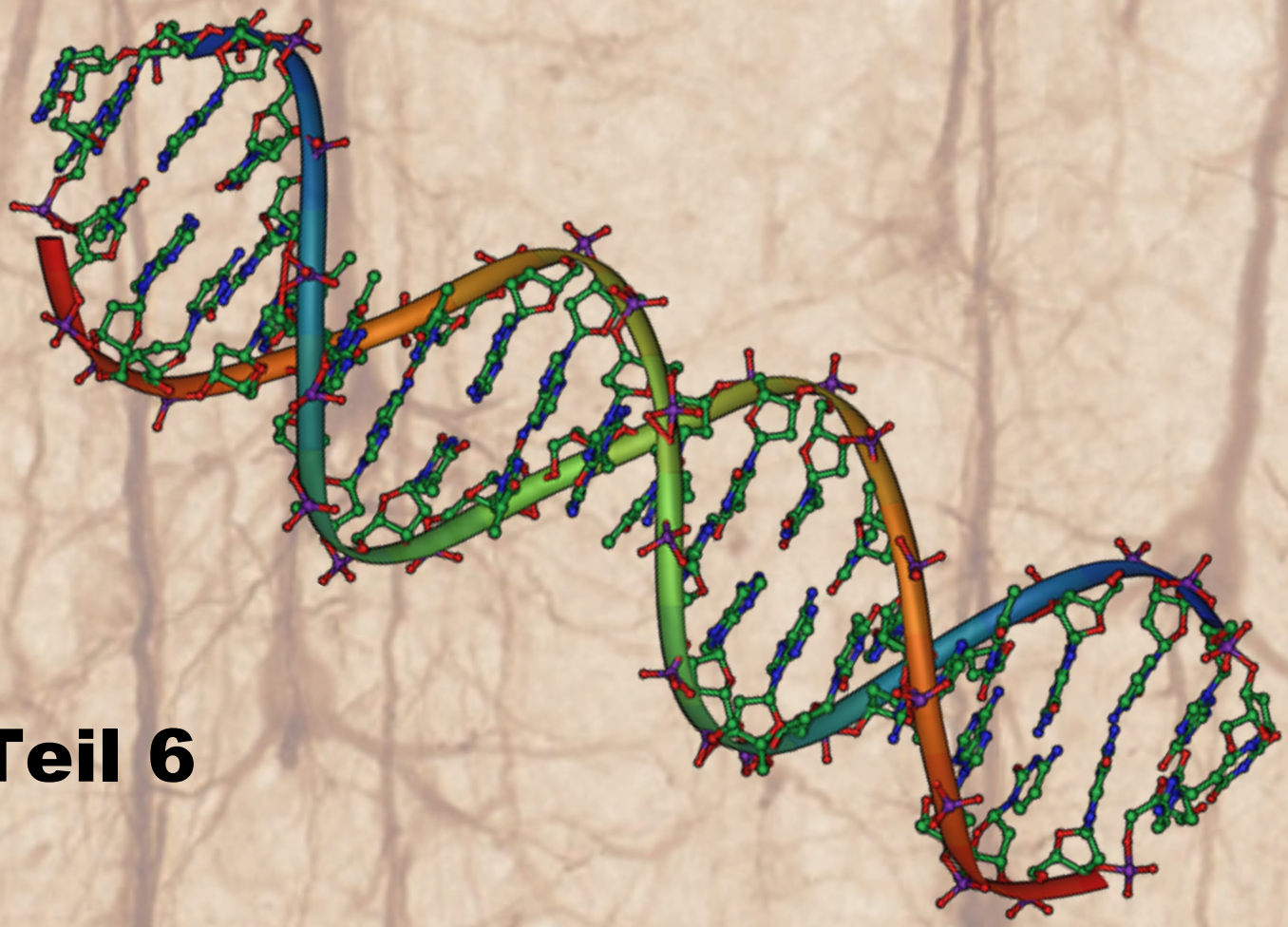


Karlsruhe Institute of Technology

AIFB

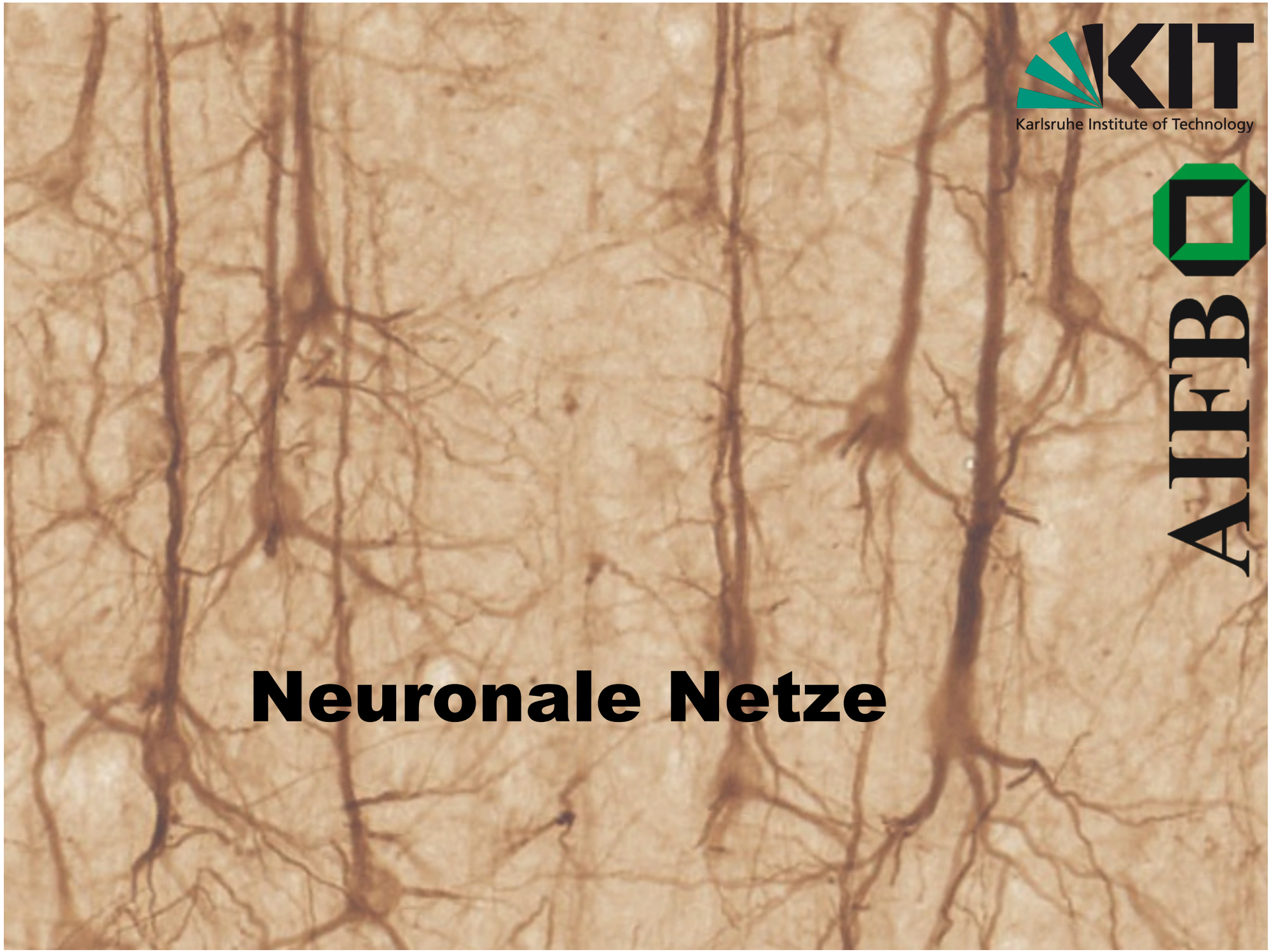
Teil 6

Neuronale Netze & Genetische Algorithmen

Biologie-inspirierte Lernverfahren

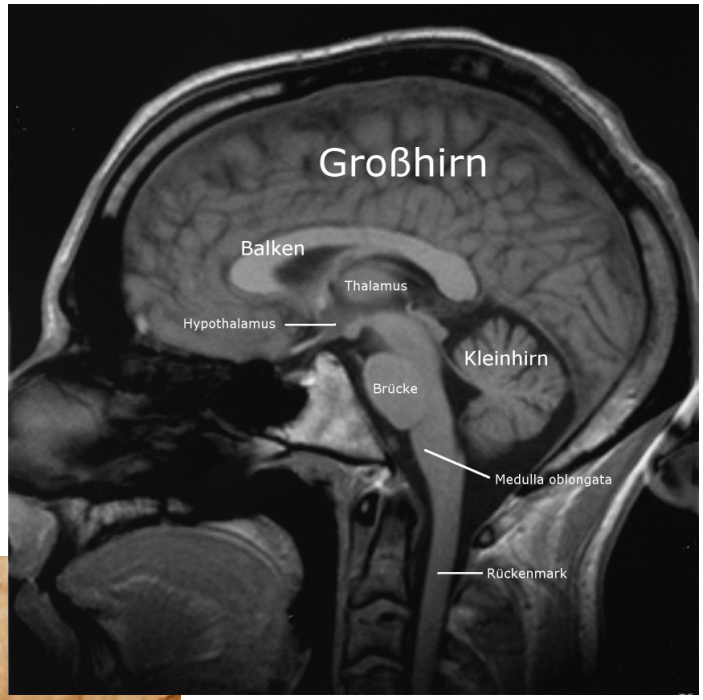
2 verschiedene Standpunkte:

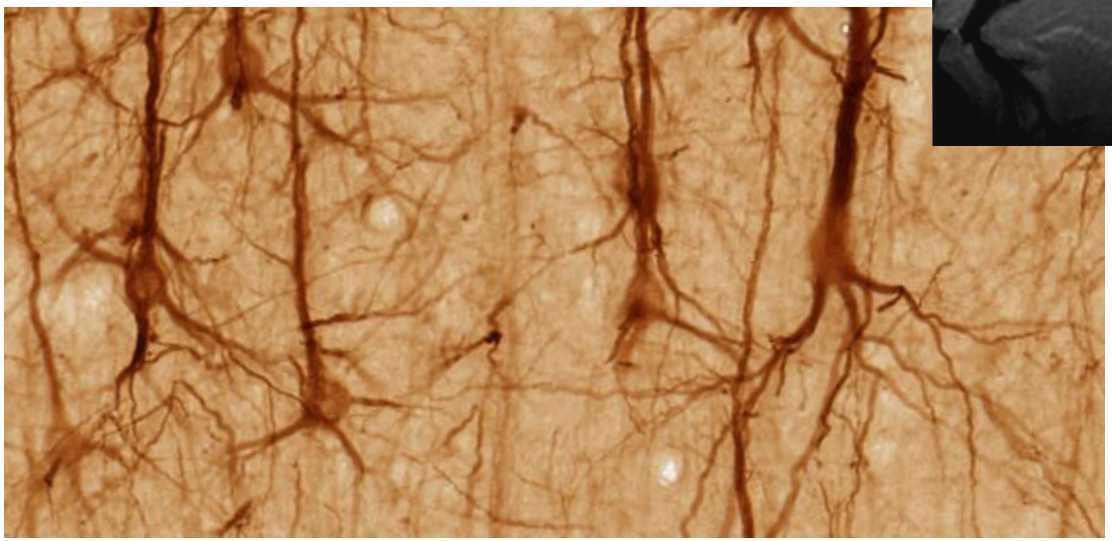
- KI durch Ausnutzung spezifischer Stärken der verwendeten Systeme (z.B. Rechenleistung, Speicherkapazität, etc.)
  - ⇨ führt meist zu symbolischen logikbasierten Ansätzen
  - ⇨ z.B. Entscheidungsbäume, ILP (s.a. letzte Vorlesung)
- KI durch Nachempfinden (simulieren, emulieren) von Strukturen und Prozessen, die man in der Natur vorfindet
  - ⇨ meist subsymbolisch
  - ⇨ z.B. neuronale Netze und genetische Algorithmen
  - ⇨ diese Vorlesung


KIT
Karlsruhe Institute of Technology
AIFB
Neuronale Netze

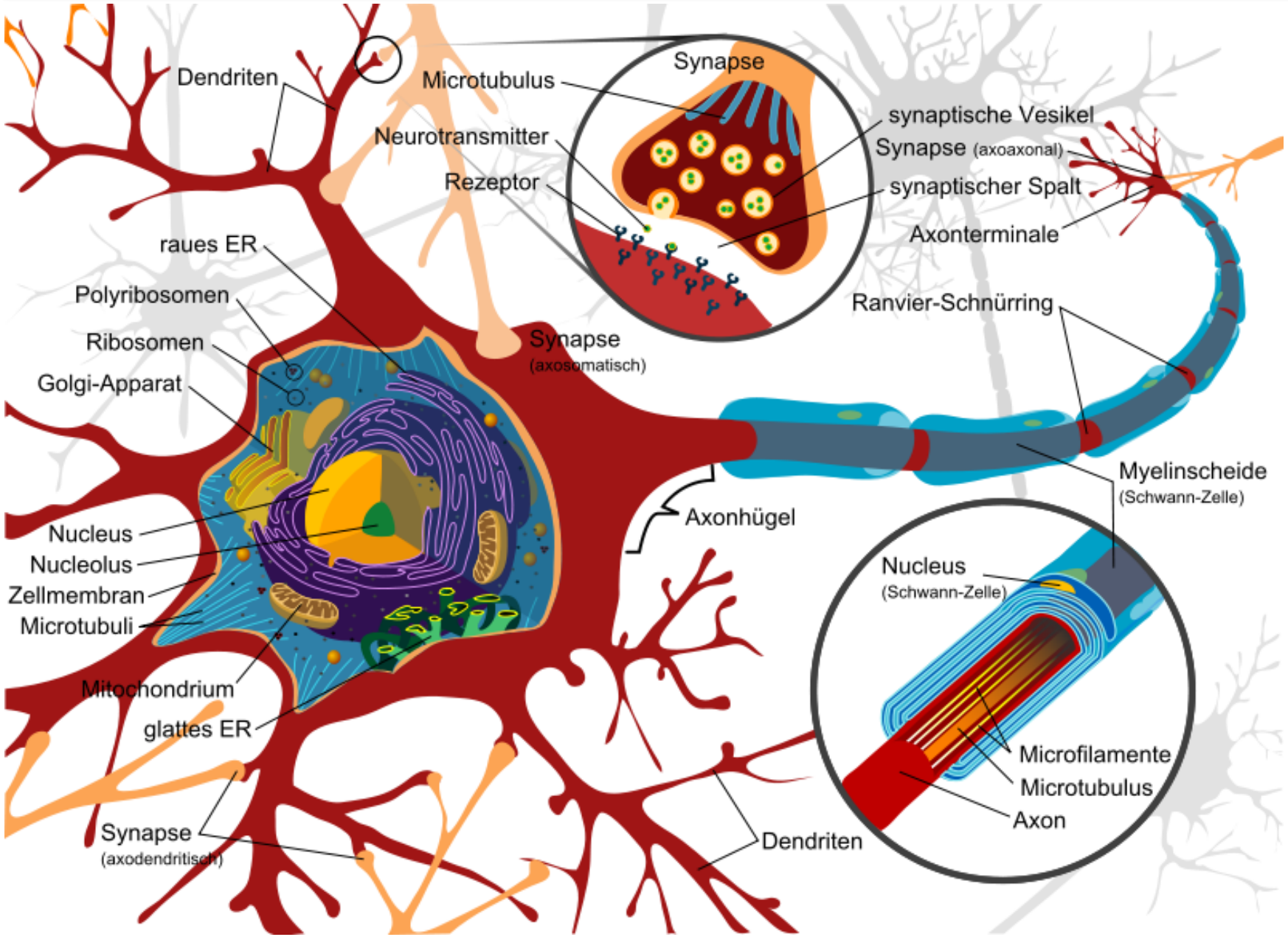
Vorbild: Natur

- Wie funktioniert Wissensverarbeitung beim Menschen?
- Gehirn
  - ⇨ca. 10^{11} Neuronen, die mit
  - ⇨ca. 10^{4} anderen Neuronen
  - ⇨durch ca. 10^{13} Synapsen verschaltet sind.

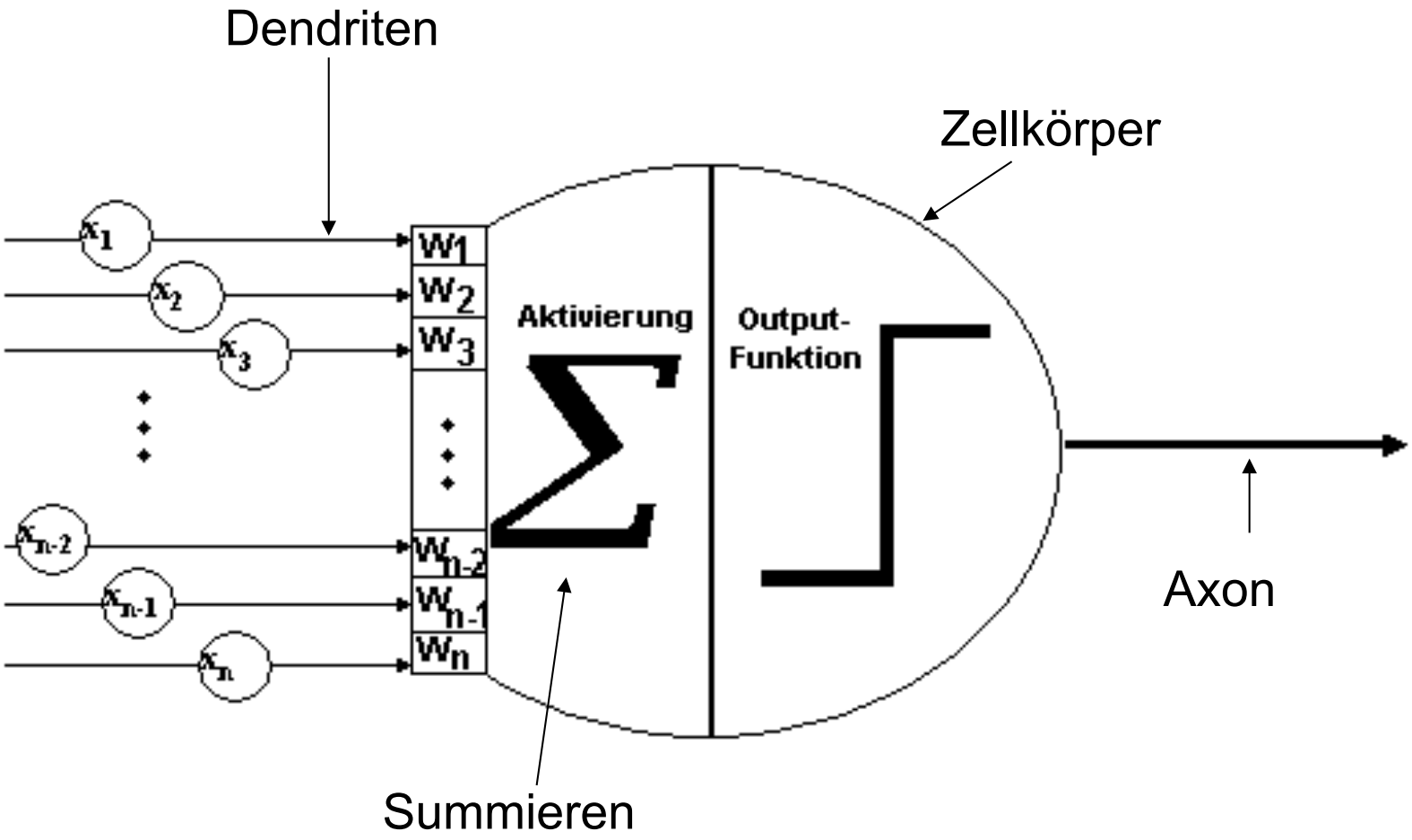




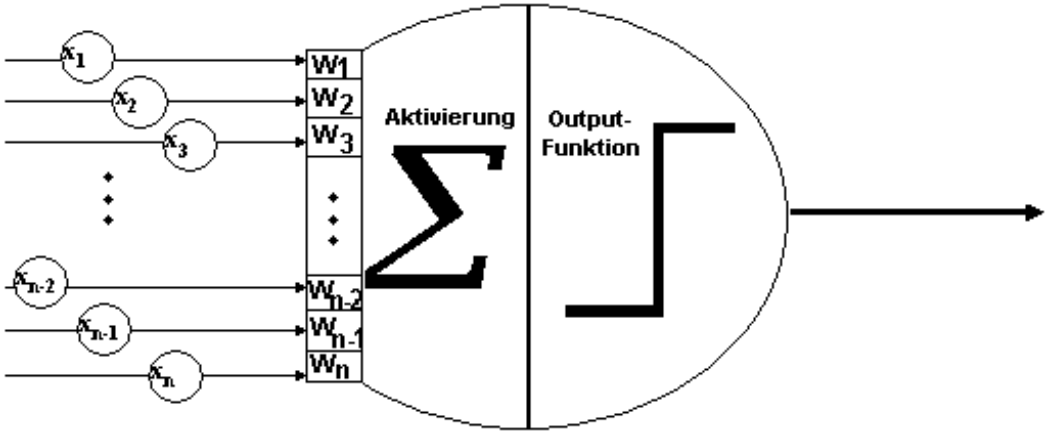
Neuronen in der Biologie



Neuronen abstrahiert

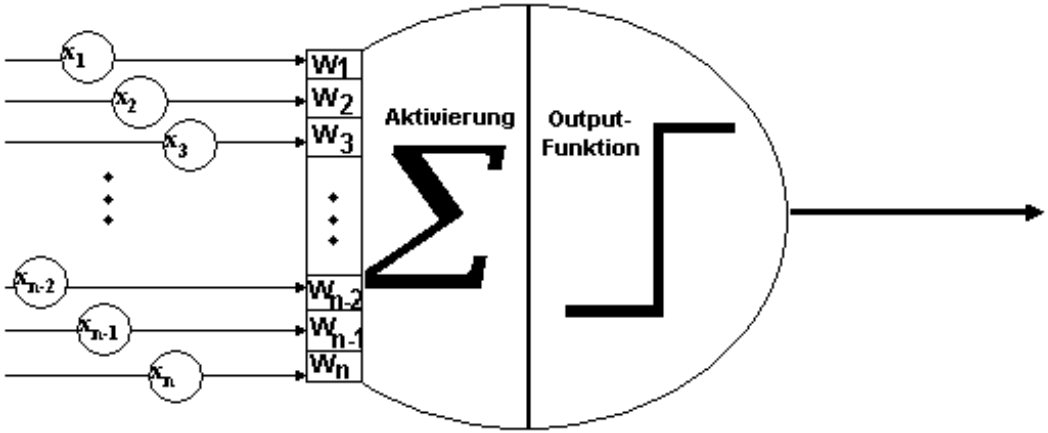


Gewichtung der Inputs



- Es gibt mehrere Arten der Aktivierung von Neuronen:
  - Allen gemeinsam ist die **Gewichtung** der Inputs. Die Inputs $x_1,x_2,...,x_n$ werden stets mit ihren (Synapsen-) Gewichten $w_1,w_2,...,w_n$ multipliziert: $a_1=x_1*w_1$, $a_2=x_2*w_2,...$, $a_n=x_n*w_n$.

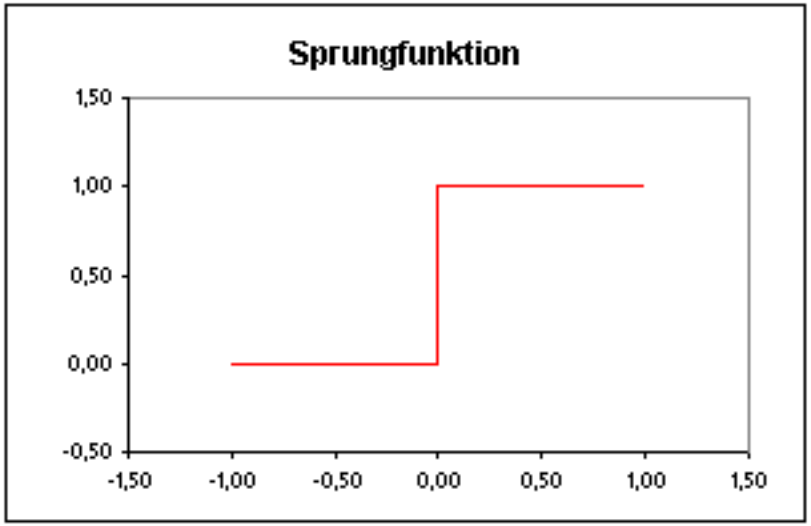
Akkumulation der Inputs

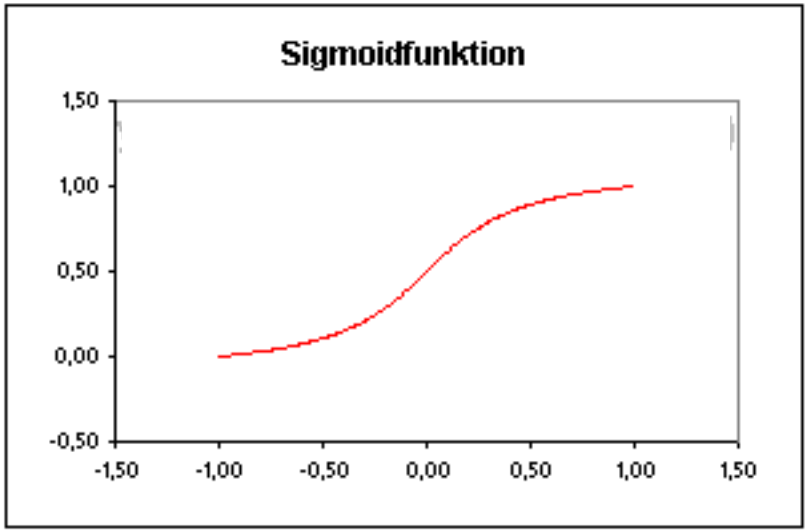


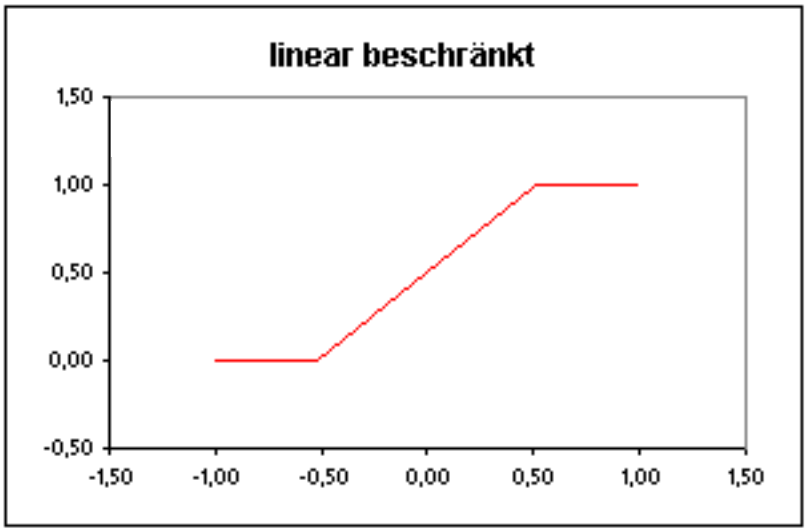
- Die am häufigsten angewandte Regel ist die **Skalarprodukt-Regel:** Die gewichteten Inputs a_1,a_2, ...,a_n werden zur Aktivität des Neurons aufaddiert: $a = a_1+a_2+...+a_n$

- Sehr häufig ist ebenfalls die **Winner-take-all-Regel:** bei der die Aktivität a zunächst nach der Skalarproduktregel ermittelt wird, dann aber mit allen Aktivitäten in derselben Schicht verglichen wird und auf 0 herabgesetzt wird, wenn ein anderes Neuron höhere Aktivität hat.

Outputfunktionen



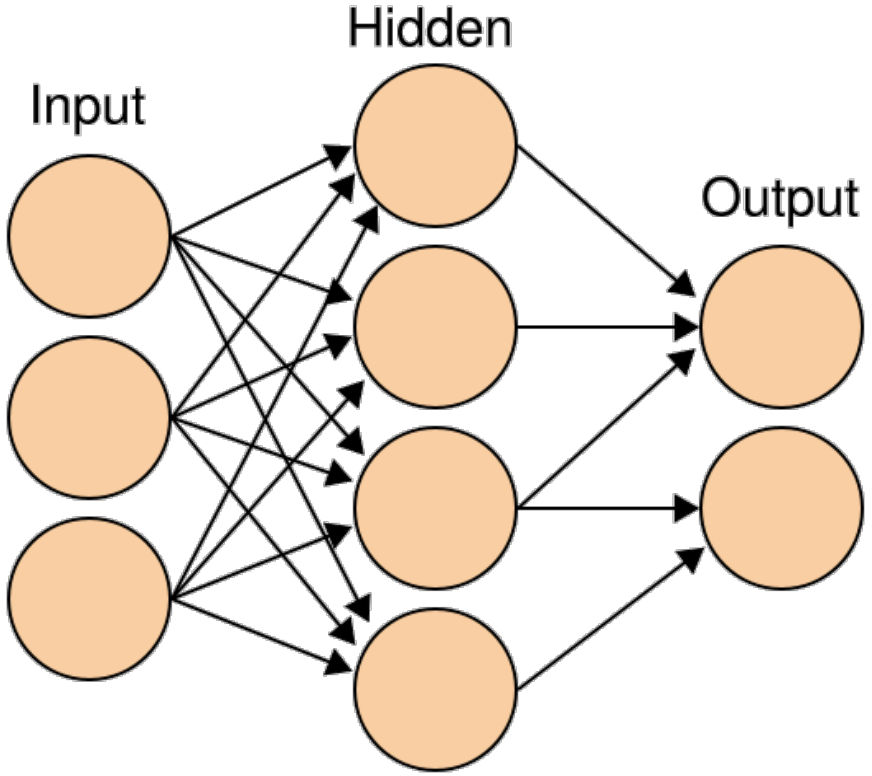




Alle Funktionen könnten wie die Sprungfunktion mit einem Schwellwert verschoben werden. Wie wir aber sehen werden kann auf diese Verschiebung verzichtet werden.

Netzwerktopologie

- Um ein Netz zu erhalten, müssen die Neuronen verschaltet werden (Verbindung von Outputs mit Inputs)
- häufig: Anordnung in Schichten („layers“) mit gerichtetem Informationsfluss



Definition

- Künstliche Neuronale Netze sind
  - ⇨ massiv parallel verbundene Netzwerke aus
  - ⇨ einfachen (üblicherweise adaptiven) Elementen in
  - ⇨ hierarchischer Anordnung oder Organisation,
  - ⇨ die mit der Welt in der selben Art wie biologische Nervensysteme interagieren sollen.

(Kohonen 1984)

Arbeitsweise

- gekennzeichnet durch:
  - massiv parallele Informationsverarbeitung
  - Propagierung der Informationen über Verbindungsstellen (Synapsen)
  - verteilte Informationsspeicherung
  - Black-Box-Charakter
- unterschieden werden:
  - Aufbauphase (Topologie),
  - Trainingsphase (Lernen) und
  - Arbeitsphase (Propagation).

(Die Phasen können auch überlappen.)

Beispiel: einfaches Perzeptron

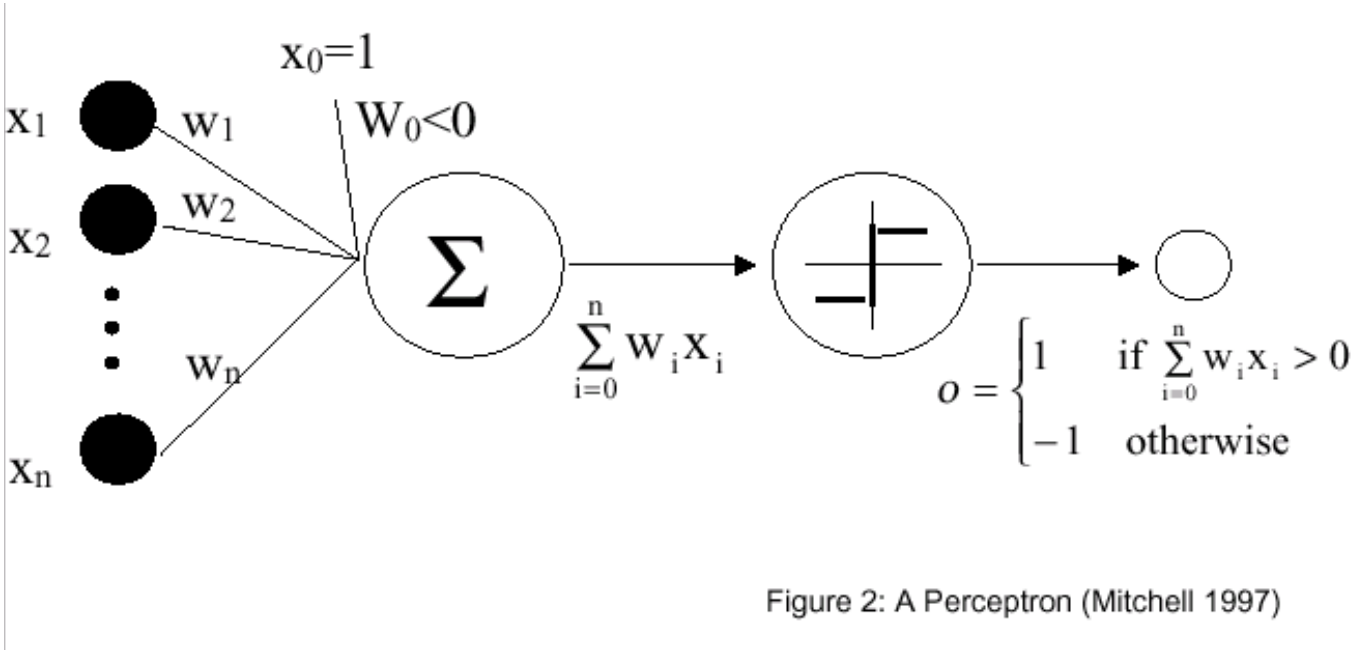


Figure 2: A Perceptron (Mitchell 1997)

- Input: $\mathbf{x} = (x_1, \ldots, x_n)$
- Gewichte: $\mathbf{w} = (w_1, \ldots, w_n)$
- Output: $o = \text{sgn}(\mathbf{xw})$

Geometrische Interpretation:

- Gleichung ***xw*** = 0 beschreibt eine Hyperebene im n-dimensionalen Raum, die diesen Raum in zwei Halbräume teilt.

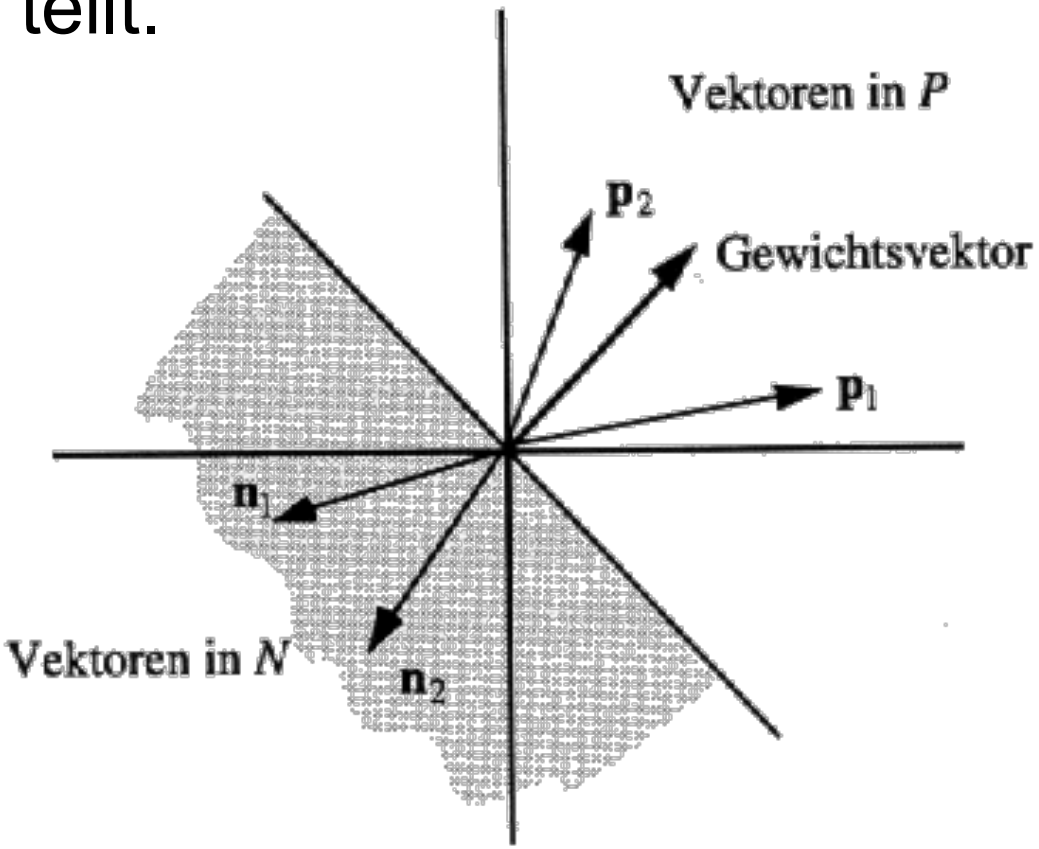


Abb. 4.8 Positiver und negativer Halbraum im Eingaberaum

entnommen Rojas, S. 84

Perzeptron: Lernaufgabe

- Gegeben ist eine Menge von Beispielen, bestehend aus Input-Vektoren ***x***.
- Überwachte Lernaufgabe: Daten sind disjunkt in zwei Mengen X,Y geteilt.
- Gesucht ist der Gewichtsvektor ***w*** $(w_1, \ldots, w_n)$, so dass eine Hyperebene spezifiziert wird, die die Mengen X und Y voneinander trennt.
- Mengen müssen linear trennbar sein, damit die Aufgabe lösbar ist.

Perzeptron: Delta-Regel

- Beim Training werden die Beispiele dem Netz als Input präsentiert.
- Output ist für die Beispiele bekannt
  → überwachte Lernaufgabe (supervised)
- (hier: liegt Beispiel in X oder Y?)
- Soll und Ist-Output werden verglichen.
  Bei Diskrepanz werden Schwellenwert und Gewichte nach folgender Delta-Regel angepasst:

$$w_{i,neu} = w_{i,alt} + \eta x_i * (Output_{soll} - Output_{ist})$$

Lernrate

aktueller Fehler

Perzeptron: Delta-Regel

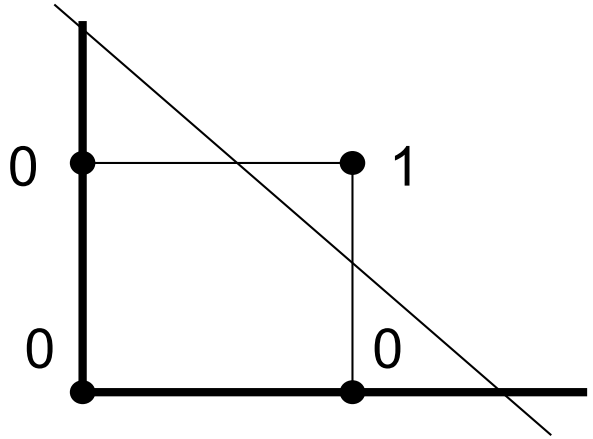
- Zwei Varianten für Gewichtsänderungen
  - Online Training: jedes Gewicht wird sofort angepasst (folgt nur im Mittel dem Gradienten)
  - Batch-Verfahren: es werden alle Datensätze präsentiert, die Gewichtsänderung des Gewichtes berechnet, summiert und dann erst angepasst (entspricht dem Gradienten über dem Datensatz)

Satz: Wenn das Perzeptron eine Klasseneinteilung überhaupt realisieren kann, dann lernt es diese mit der Delta-Regel in endlich vielen Schritten.

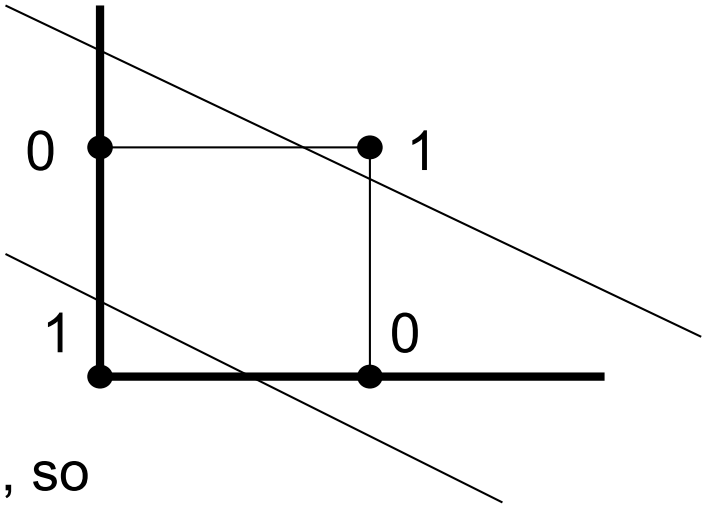
Problem: Falls das Perzeptron nicht lernt, kann nicht unterschieden werden, ob nur noch nicht genügend Schritte vollzogen wurden oder ob das Problem nicht lernbar ist. (Es gibt keine obere Schranke für die Lerndauer.)

Perzeptron: XOR-Problem

Logisches AND ist
linear separierbar



Logisches XOR ist nicht
linear separierbar



Führt man weitere Hyperebenen ein, so kann man auch hier die Klassen unterscheiden. → Realisierung durch Zwischenschichten

2-layer Perzeptron

Input-Vektor $\underline{\mathbf{x}}=(x_1,\ldots,x_m)$

Gewichtsvektor $\underline{\mathbf{v}}=(v_{11},\ldots,v_{nm})$

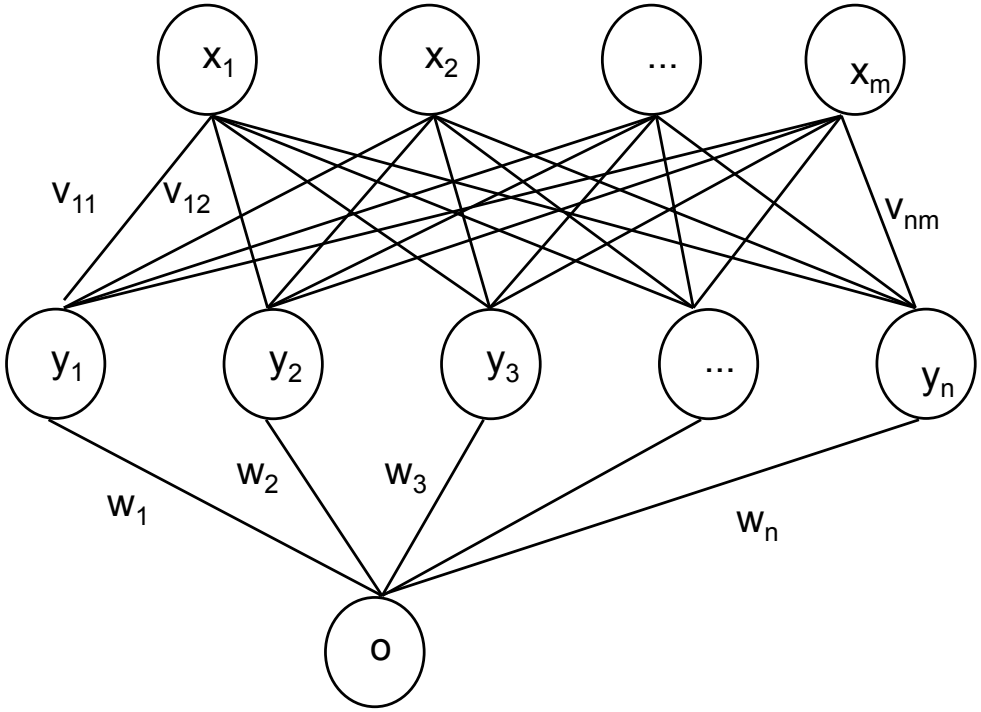
Aktivitätsvektor $\underline{\mathbf{y}}=(y_1,\ldots,y_n)$

Gewichtsvektor $\underline{\mathbf{w}}$

Output o

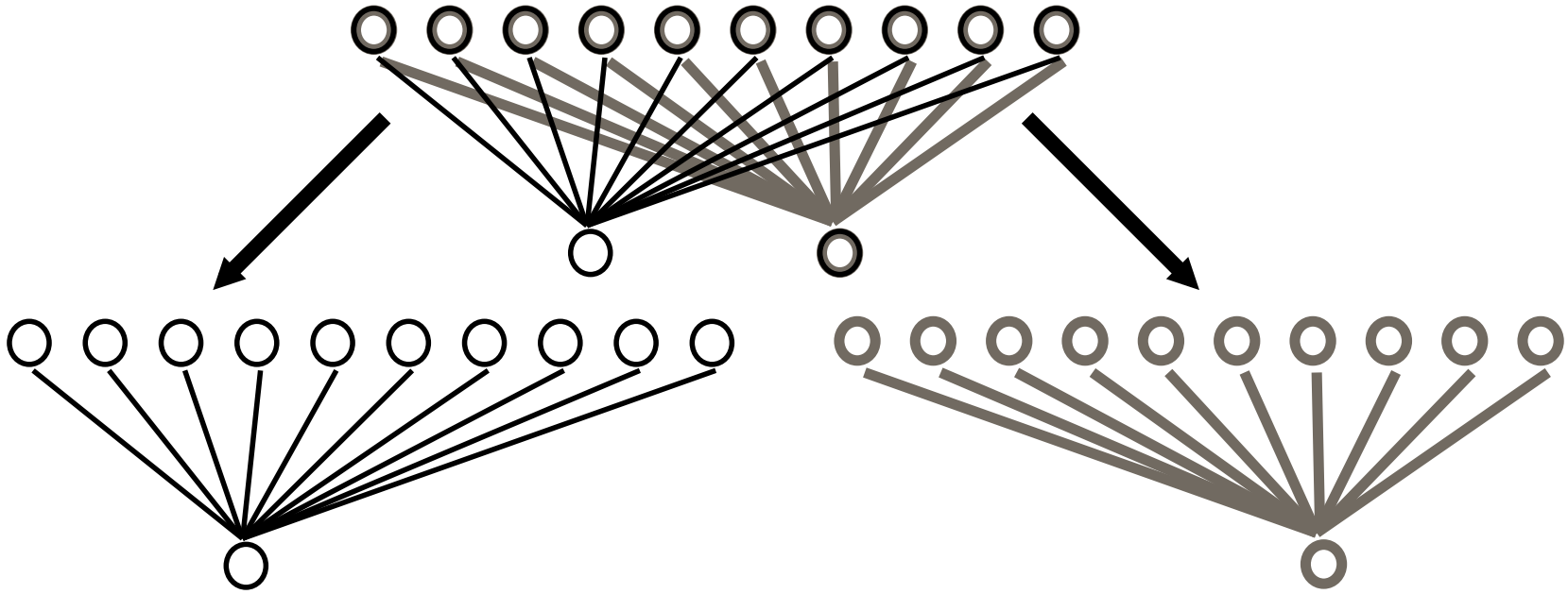
$$\underline{\mathbf{y}} = \theta(\underline{\mathbf{v}}\cdot\underline{\mathbf{x}})$$

$$o = \theta(\underline{\mathbf{w}}\cdot\underline{\mathbf{y}})$$



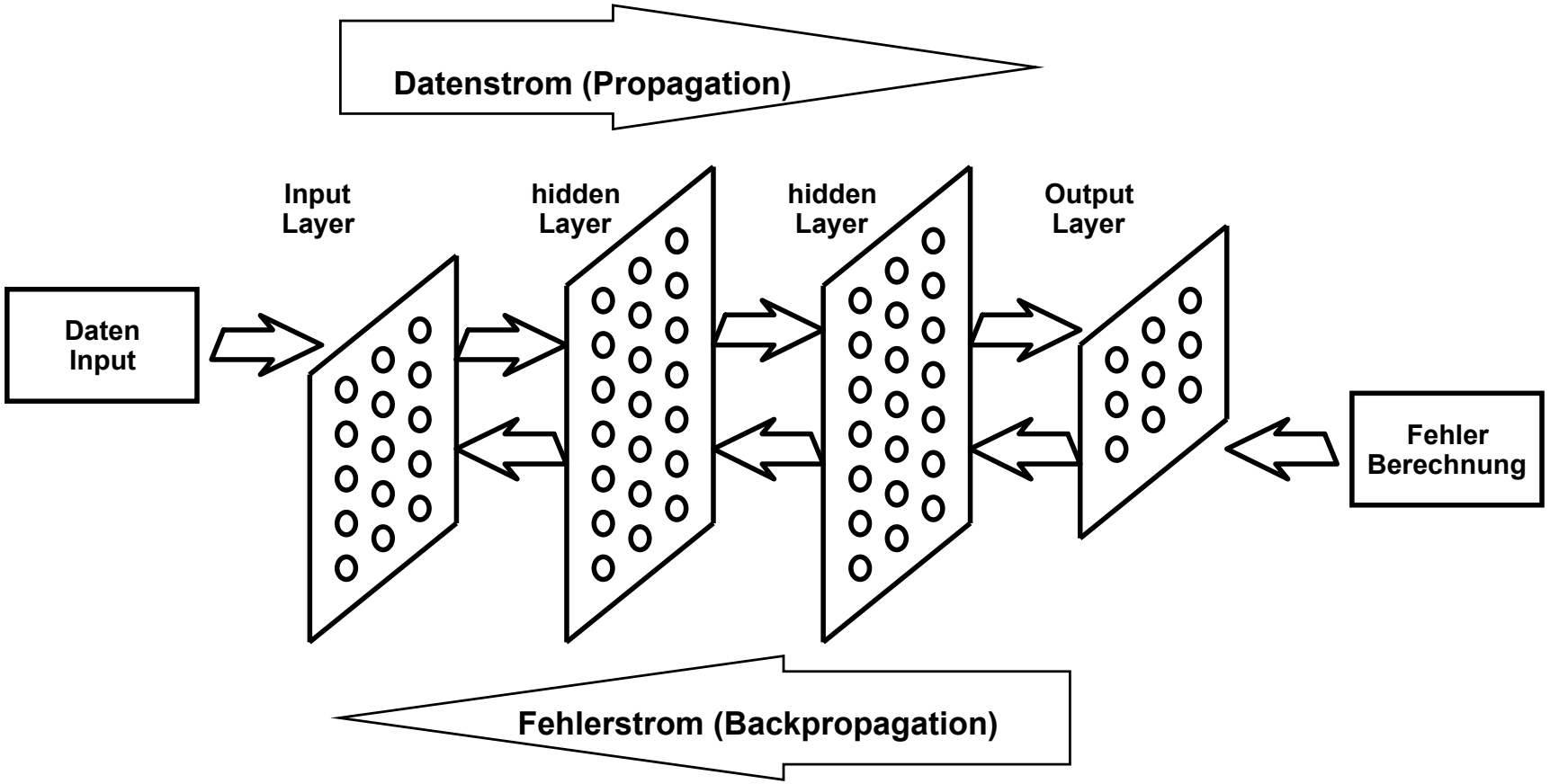
2-layer Perzeptron

In einem 2-Schichten-Netz beeinflussen die Neuronen der 2. Schicht einander nicht, deshalb können sie voneinander getrennt betrachtet werden.



Bei mehrschichtigen Netzen geht die Unabhängigkeit verloren, d.h. sie können nicht so getrennt werden.

Backpropagation - Idee



Backpropagation - Algorithmus

1. Wähle ein Muster x aus der Menge der Trainingsbeispiele D aus.

2. Präsentiere das Muster dem Netz und berechne Output (Propagation).

3. Der Fehler F wird als Differenz von errechnetem und gewünschten Output ermittelt.

4. Die Fehlerinformationen werden durch das Netz zurückpropagiert (Backpropagation).

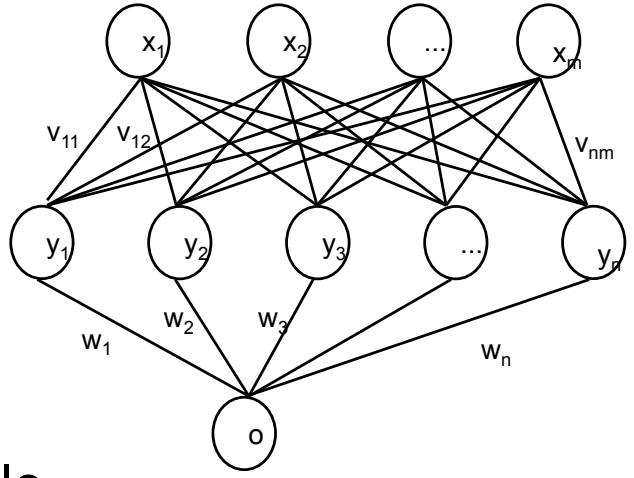
5. Die Gewichte zwischen den einzelnen Schichten werden so angepasst (Δw_{ik}), dass der mittlere Ausgabefehler für das Muster sinkt.

6. Abbruch, wenn Fehler auf Validierungsmenge unter gegebenem Schwellwert liegt. (Siehe spätere Folien.)
   Sonst gehe zu 1.

Anpassung der Gewichte

Fehlerfunktion F (mittlerer quadratischer Fehler) für das Lernen:

$$F_D = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$



wobei gilt:

D Menge der Trainingsbeispiele
t_d korrekter Output für $d \in D$
o_d berechneter Output für $d \in D$

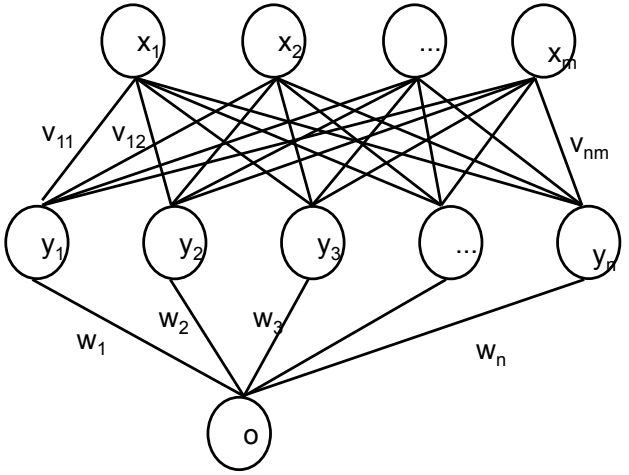
Die Gewichte müssen so angepasst werden, dass der Fehler minimiert wird. Dazu bietet sich wieder das Gradientenabstiegsverfahren an. (D.h.: Bergsteigerverfahren mit Vektorraum der Gewichtsvektoren als Suchraum!)

Anpassung der Gewichte

Sei nun ein $d \in D$ gegeben.
Anders geschrieben ist

$F_d = (o - t)^2 \qquad = (\theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) - t)^2$

Der Fehlergradient für w_i lautet für dieses d:



$\partial F/\partial\, w_i \qquad = \partial(o-t)^2/\partial w_i$

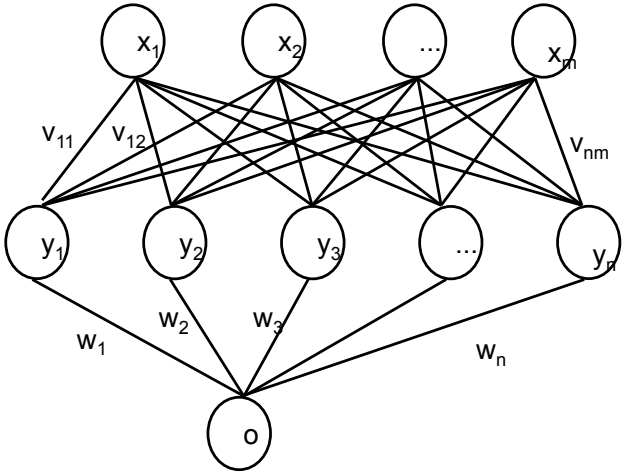
$= \partial(\theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) - t)^2/\partial w_i$

$= 2 \cdot (o-t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) \cdot \partial(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}})/\partial w_i$

$= 2 \cdot (o-t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) y_i$

Anpassung der Gewichte

Fehlergradient für v_{ij} lautet:



$$\partial F/\partial v_{ij} = \partial F/\partial y_i \cdot \partial y_i/\partial v_{ij}$$

$$= \partial F/\partial y_i \cdot \partial\, \theta(\underline{v}_i \cdot \underline{x})/\partial v_{ij}$$

(Fehler von Neuron i)

$$= \partial F/\partial y_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= \partial F/\partial o \cdot \partial o/\partial y_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= \partial F/\partial o \cdot \partial\theta(\underline{w} \cdot \underline{y})/\partial y_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= \partial F/\partial o \cdot \theta'(\underline{w} \cdot \underline{y}) \cdot w_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= 2 \cdot (o-t) \cdot \theta'(\underline{w} \cdot \underline{y}) \cdot w_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

Fehler bei der Ausgabe

Info von Zwischenschicht

Gewicht

Input

Info von Inputschicht

Anpassung der Gewichte

- Die schichtweise Berechnung der Gradienten ist auch für mehr als zwei Schichten möglich.
- Der Fehler wird dann schichtenweise nach oben propagiert (Back-Propagation).

- Allgemein gilt für den Output-Fehler e eines Neurons j
  $e_j = \partial F / \partial x_j$

- Gewichtsänderung
  $\Delta w_{ik} = a \cdot e_k \cdot \theta'(a_k) \cdot x_i$

Anpassung der Gewichte

- Betrachtet man den Fehler des Netzes als Funktion aller Gewichte **w**, so kann man zeigen, dass bei jedem Schritt der Fehler kleiner wird. Dies erfolgt unabhängig vom gewählten Netz, also unabhängig von **w**.

- $e(\mathbf{w}) = (o - t)^2 = (t - \theta\,(\mathbf{w} \cdot x))^2$

- Es gilt bei jedem Schritt:

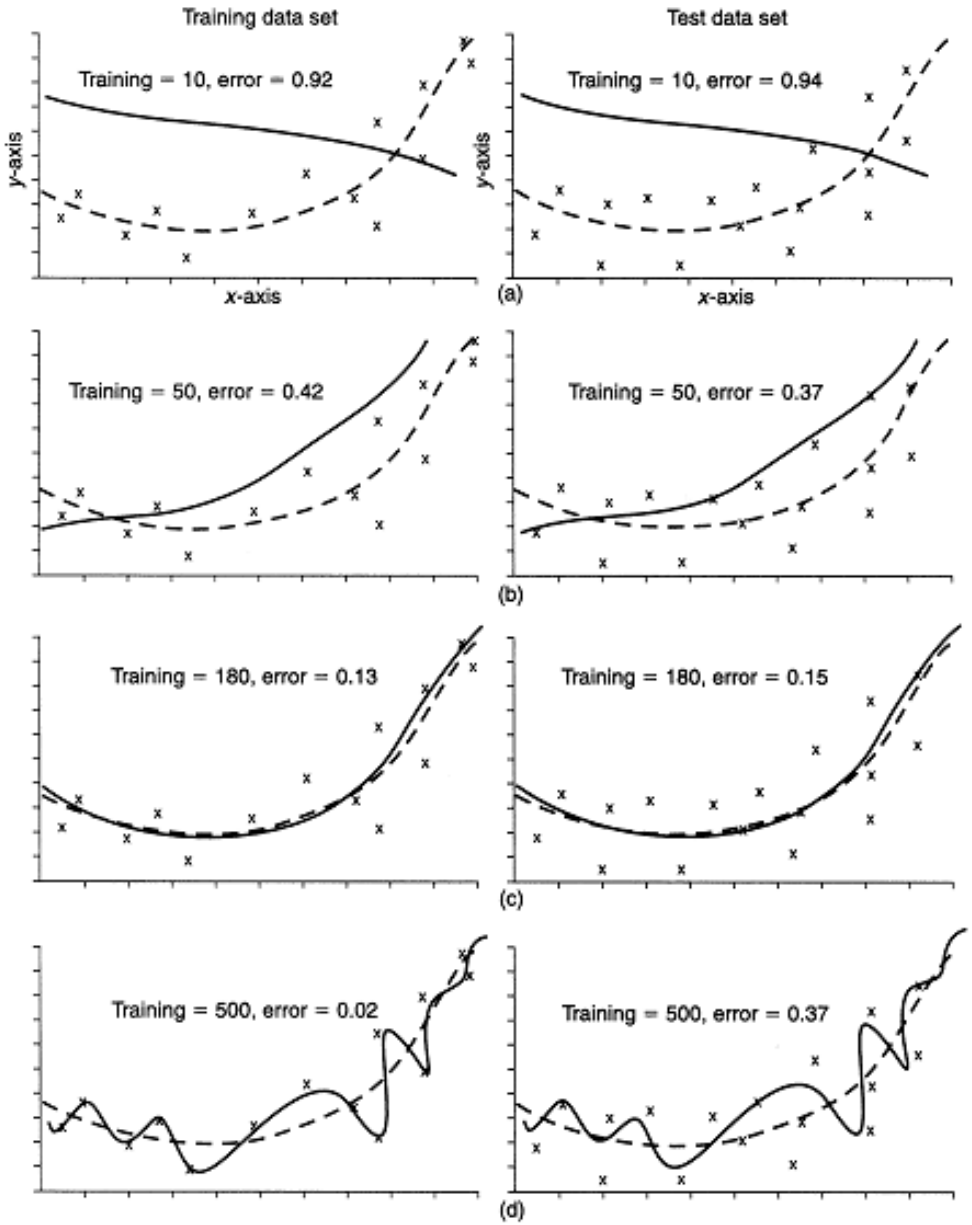
$$e(\mathbf{w} + \Delta\mathbf{w})^2 < e(\mathbf{w})^2$$

Bemerkungen

- Jede boolesche Funktion kann durch derartiges Netz repräsentiert werden.
- Beliebige Funktionen können durch ein ANN mit drei Schichten beliebig genau approximiert werden.
- Hypothesenraum ist kontinuierlich im Gegensatz z.B. zum diskreten Hypothesenraum von Entscheidungsbaumverfahren.
- ANN kann für interne Schicht sinnvolle Repräsentationen lernen, die im vorhinein nicht bekannt sind;
- dies ist Gegensatz zu z.B. ILP (ein Verfahren mit vorgegebenem Hintergrundwissen).

Overfitting

- Problem der Überanpassung eines Modells an Trainings-Datensatz. Gute Anpassung an kleinen Weltausschnitt, aber fehlende Generalisierung → schlechte Reaktion auf neue Situationen.
- Kontrolle der Generalisierung während Trainings durch Teilen des Datensatzes:
  - Trainings-Datensatz (Output bekannt)
  - Validierungs-Datensatz (Output bekannt)
  - Anwendungsdaten (Output unbekannt)
- Evaluierung mit Validierungs-Datensatz



Abbruchbedingung

- Beschreibung des Backpropagation-Algorithmus lässt Abbruchbedingung offen.
- schlechte Strategie: solange iterieren, bis Fehler für Trainingsbeispiele unter vorgegebenem Schwellwert liegt (Gefahr: Overfitting)
- besser: separate Menge von Beispielen zur Validierung (validation set); iteriere solange, bis Fehler für Validierungsmenge minimal ist
- Aber: Fehler auf der Validierungsmenge muss nicht monoton fallen (im Gegensatz zu Fehler auf der Trainingsmenge, siehe nächste Folien)!

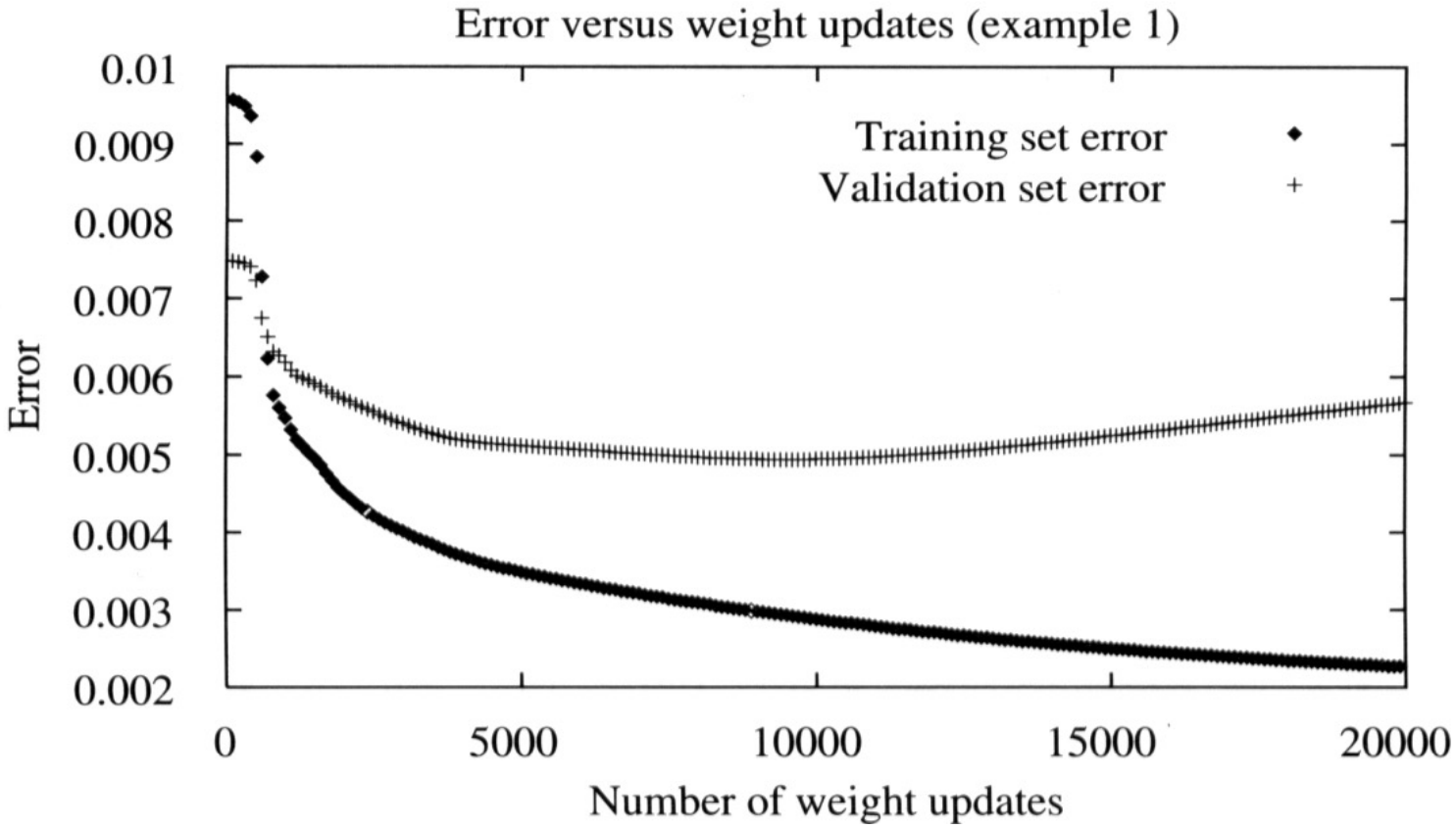


Figure 8a: Plots of error E as a function of the number of weight updates, for two different robot perception tasks (Mitchell 1997)

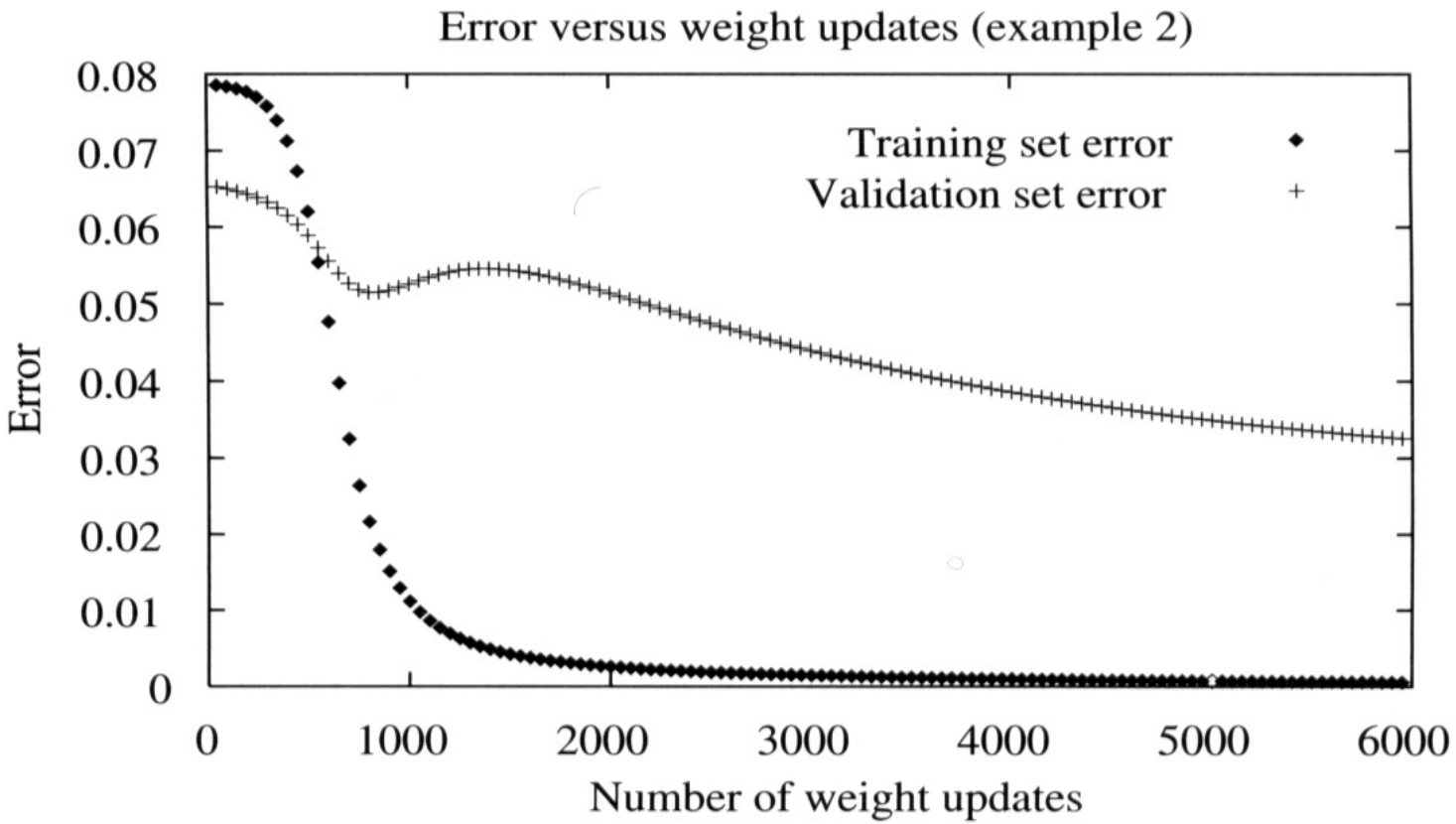


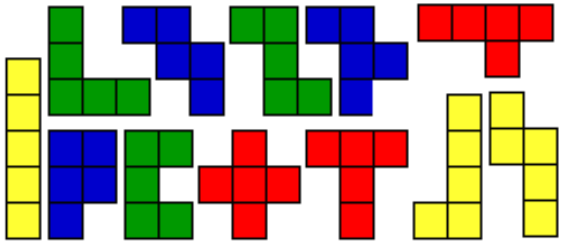
Figure 8b: Plots of error E as a function of the number of weight updates, for two different robot perception tasks (Mitchell 1997)

Kreuzvalidierung

- Alternativ (insbes. bei wenigen Trainingsdaten) kann k-fache Kreuzvalidierung (k-fold cross-validation) verwendet werden:
  - Unterteile Menge der Trainingsbeispiele in k gleich große disjunkte Teilmengen.
  - Verwende der Reihe nach jeweils eine andere Teilmenge als Validierungsmenge und die restlichen (k - 1) Teilmengen als Trainingsmenge.
  - Für jede Validierungsmenge wird „optimale“ Anzahl i von Iterationen bestimmt (d.h. mit kleinstem mittleren Fehler für die Daten aus der Validierungsmenge).
  - Der Mittelwert von i über alle k Trainingsphasen wird letztendlich verwendet, um das gegebene ANN mit allen Trainingsbeispielen zu trainieren.

Vorteile neuronaler Netze

- sehr gute Mustererkenner



- verarbeiten verrauschte, unvollständige und widersprüchliche Inputs
- verarbeiten multisensorischen Input (Zahlen, Farben, Töne, ...)
- erzeugen implizites Modell für Eingaben (ohne Hypothesen des Anwenders)
- fehlertolerant auch gegenüber Hardwarefehlern
- schnelle Berechnung der gelernten Funktion
- leicht zu handhaben

Nachteile neuronaler Netze

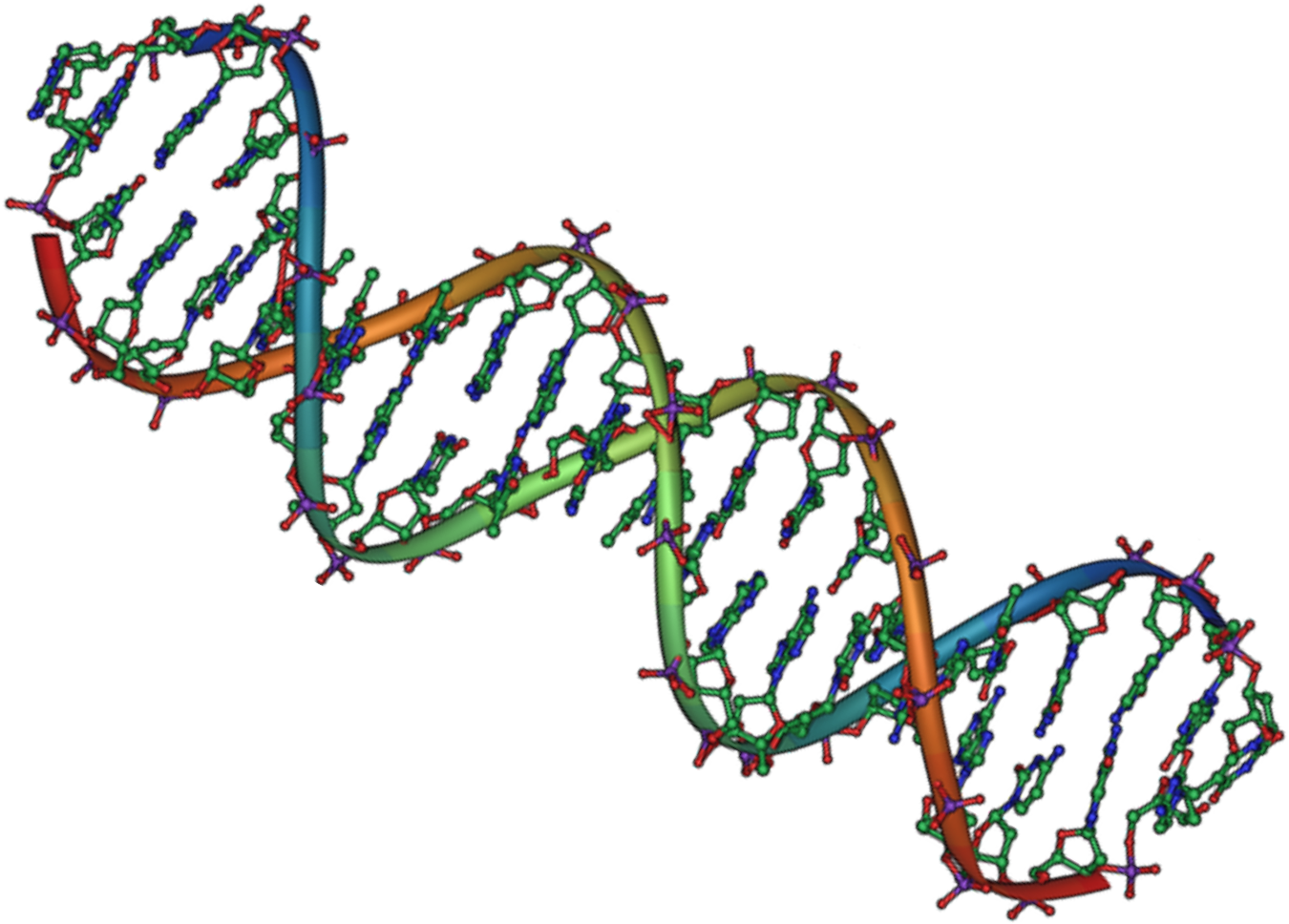
- lange Trainingszeiten
- Lernerfolg kann nicht garantiert werden
- Generalisierungsfähigkeit kann nicht garantiert werden (Overfitting)
- keine Möglichkeit, die Begründung einer Antwort zu erhalten (Blackbox)

Anwendung neuronaler Netze

- Interpretation von Sensordaten
- Prozesssteuerung
- Medizin
- Elektronische Nase
- Schrifterkennung
- Risikomanagement
- Zeitreihenanalyse und -prognose
- Robotersteuerung

Was gibt es noch...

- hier nicht behandelt: dynamische Aspekte:
  - Spiking neurons
    Reaktion eines Neurons auf einen Input (das „Feuern“) ist nicht ein fester Wert, sondern hat einen bestimmten Verlauf, auch das kann simuliert werden
  - Rekurrente Netze
    Bei Analyse von zeitlichen Verläufen ist es sinnvoll, wenn sich das Netz etwas „merken“ kann. Erreicht wird dies, indem ein Teil des outputs wieder als input in das Netz eingespeist wird.
  - Dynamische Anpassung der Netzwerktopologie
    Anpassung der Netzstruktur (z.B. Anzahl von hidden neurons und Verbindungen) während des Trainings



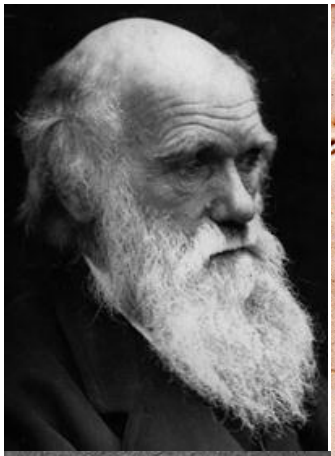
Genetische Algorithmen

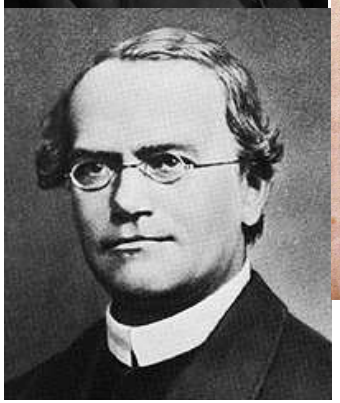
Karlsruhe Institute of Technology

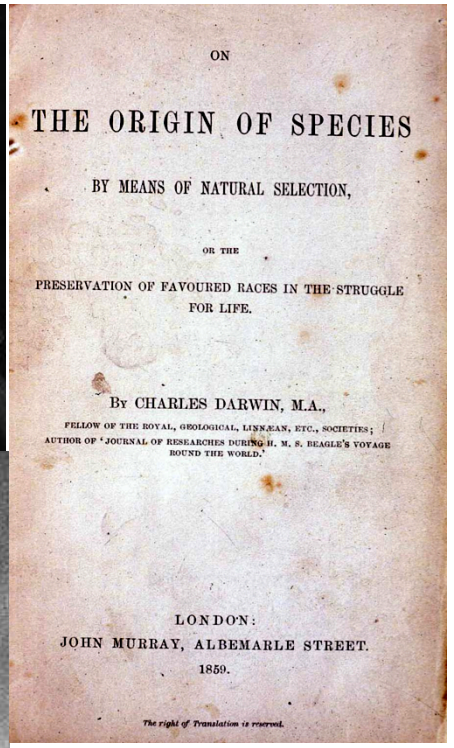
AIFB

Evolution und Genetische Optimierung

- Vorbild: Mechanismen der biologischen Evolution
  - ⇨ Darwin:
    - ✧ Natürliche Selektion
    - ✧ Variationen in einer Population
  - ⇨ genetische Vererbung
    - ✧ Rekombination
    - ✧ Mutation
  - ⇨ Mendel:
    - ✧ Genotyp → Phänotyp

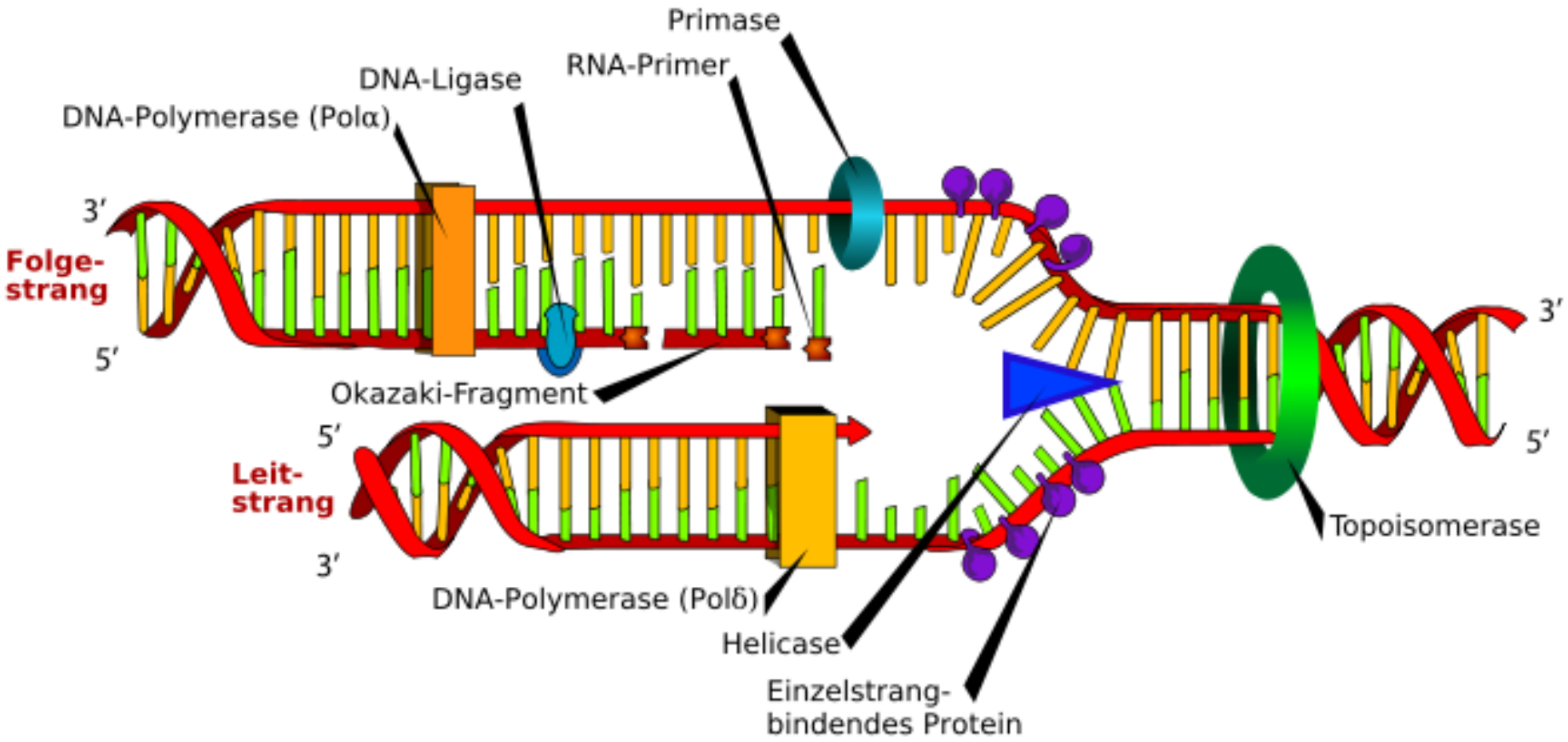




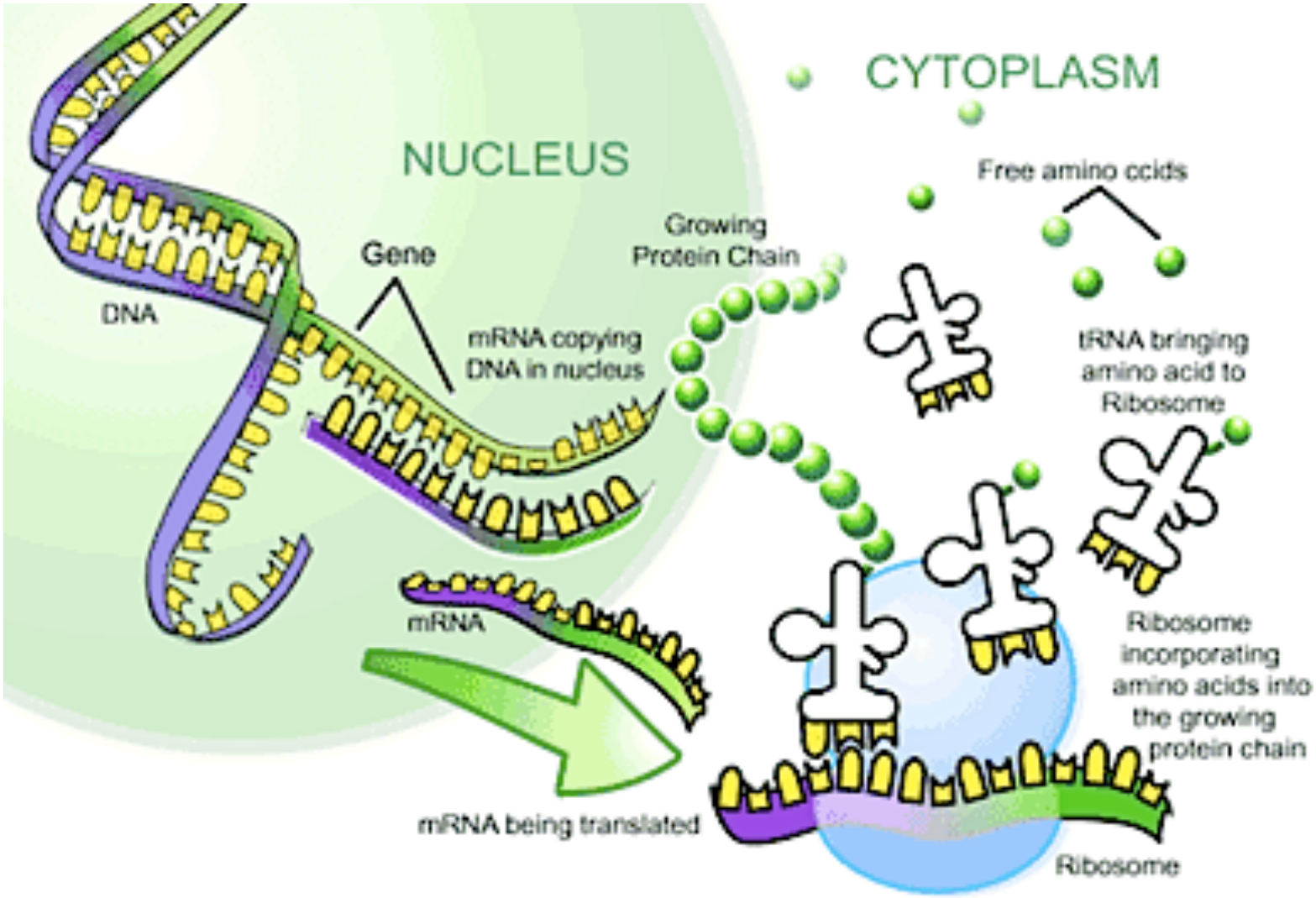


- Die Natur hat über die Evolution die Fitness der Individuen so optimiert, daß sie in Ihrer Umgebung am besten überleben können.

Genetik: Replikation

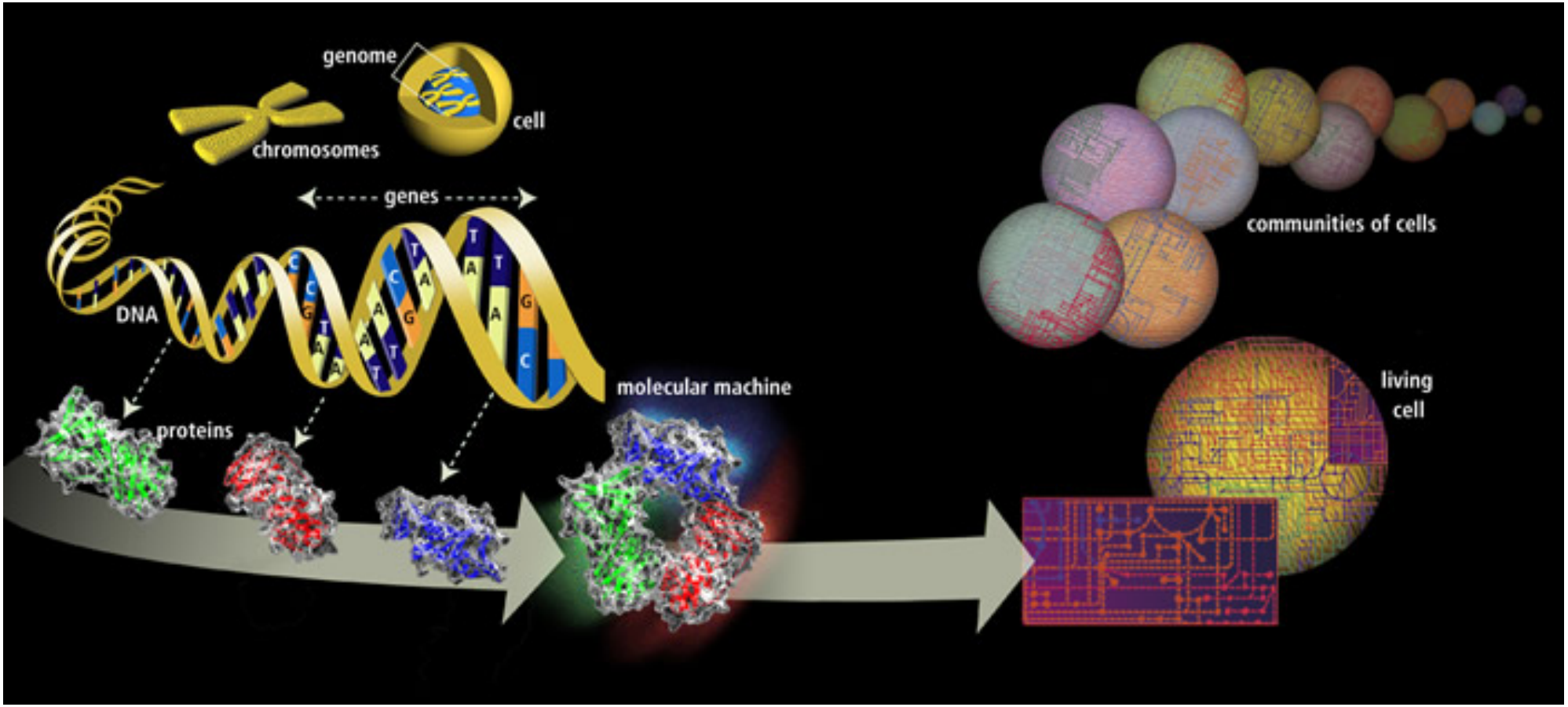


Genetik: Translation



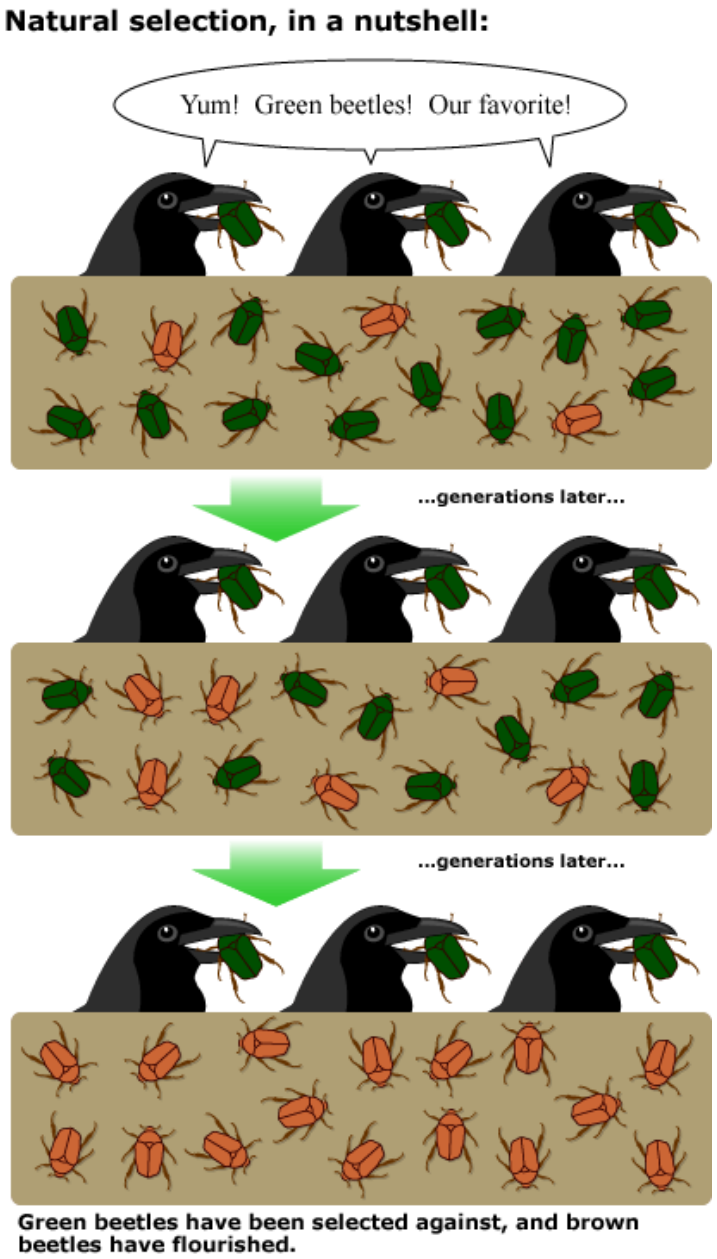
Genotyp → Phänotyp

Im Genom gespeicherte Information führt zur Ausprägung bestimmter Merkmale und Verhaltensweisen.



Natürliche Auslese (Selektion)

- Merkmale und Verhaltensweisen bewähren sich im Kampf ums Dasein – oder eben nicht.
- Gene mit positiven Auswirkungen werden mit höherer Wahrscheinlichkeit weitergegeben.
- Gesamtheit der überlebenden Individuen ist genetisch besser an die Erfordernisse der Umgebung angepasst.

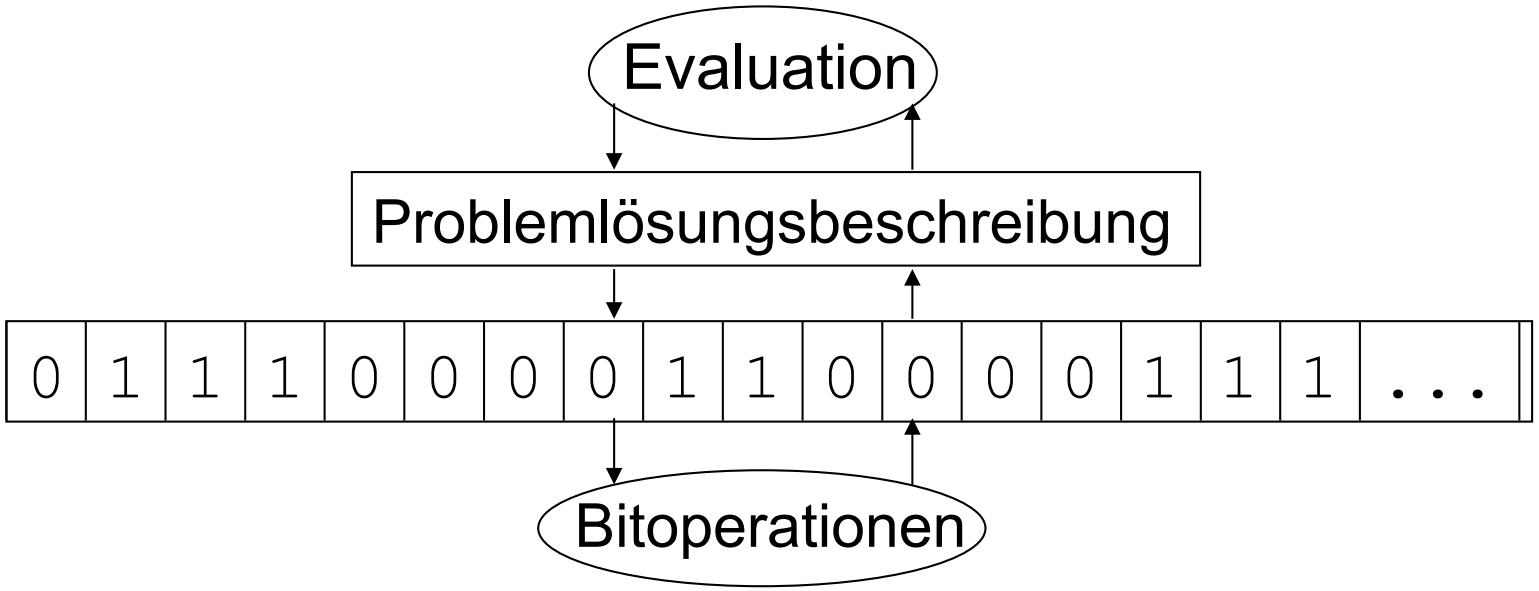


Genetische Algorithmen

- Idee: diesen Prozess künstlich simulieren zum Zwecke der Optimierung
- Varianten:
  - ⇨ Genetische Algorithmen i.e.S.: festes Genom
  - ⇨ Genetisches Programmieren: komplexes Genom variabler Länge (z.B. Programmcode)

Genetische Algorithmen

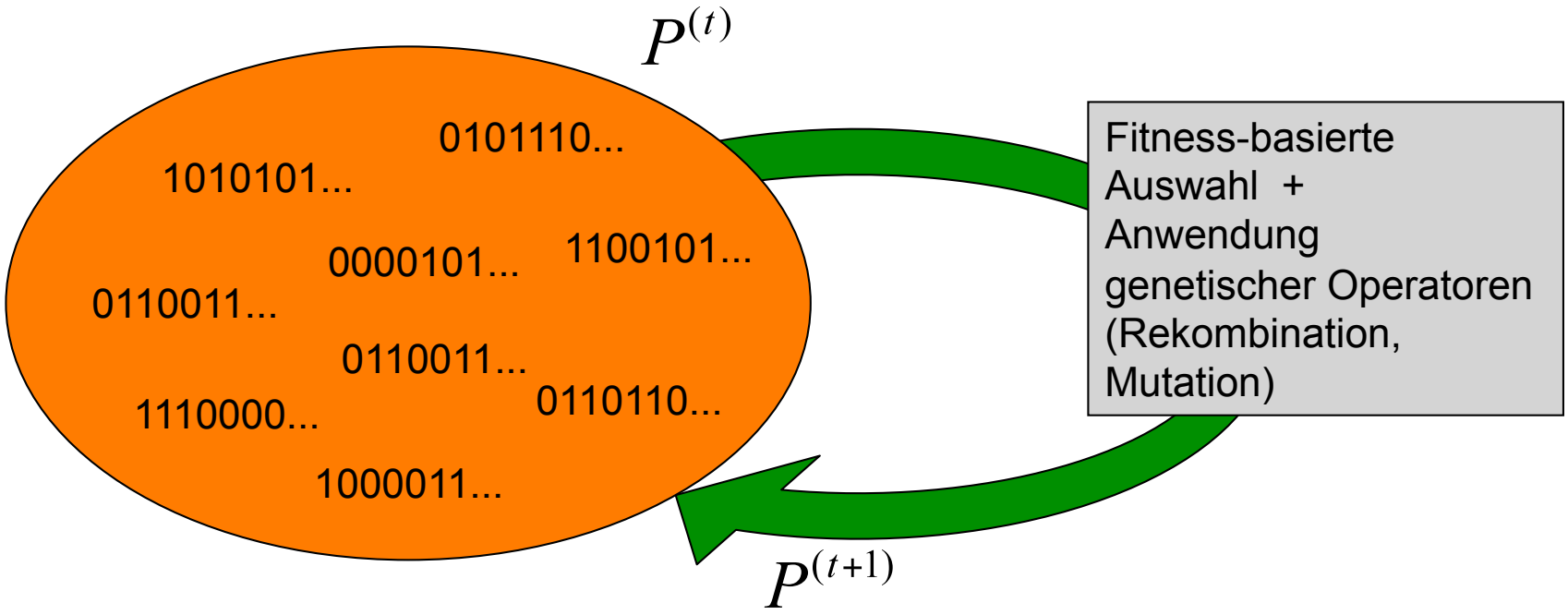
- Von John Holland in den 70er Jahren als Problemlösungsmethode entwickelt, welche sich evolutionärer Mechanismen bedient.
  - ⇨ Codierung einer Lösung durch Bitstrings
  - ⇨ Mögliche Lösungen unterschiedlicher Qualität bilden eine Population, wobei die besten Individuen eine höhere Chance erhalten, sich fortzupflanzen
  - ⇨ neue Lösungen entstehen durch Kreuzung und Mutation



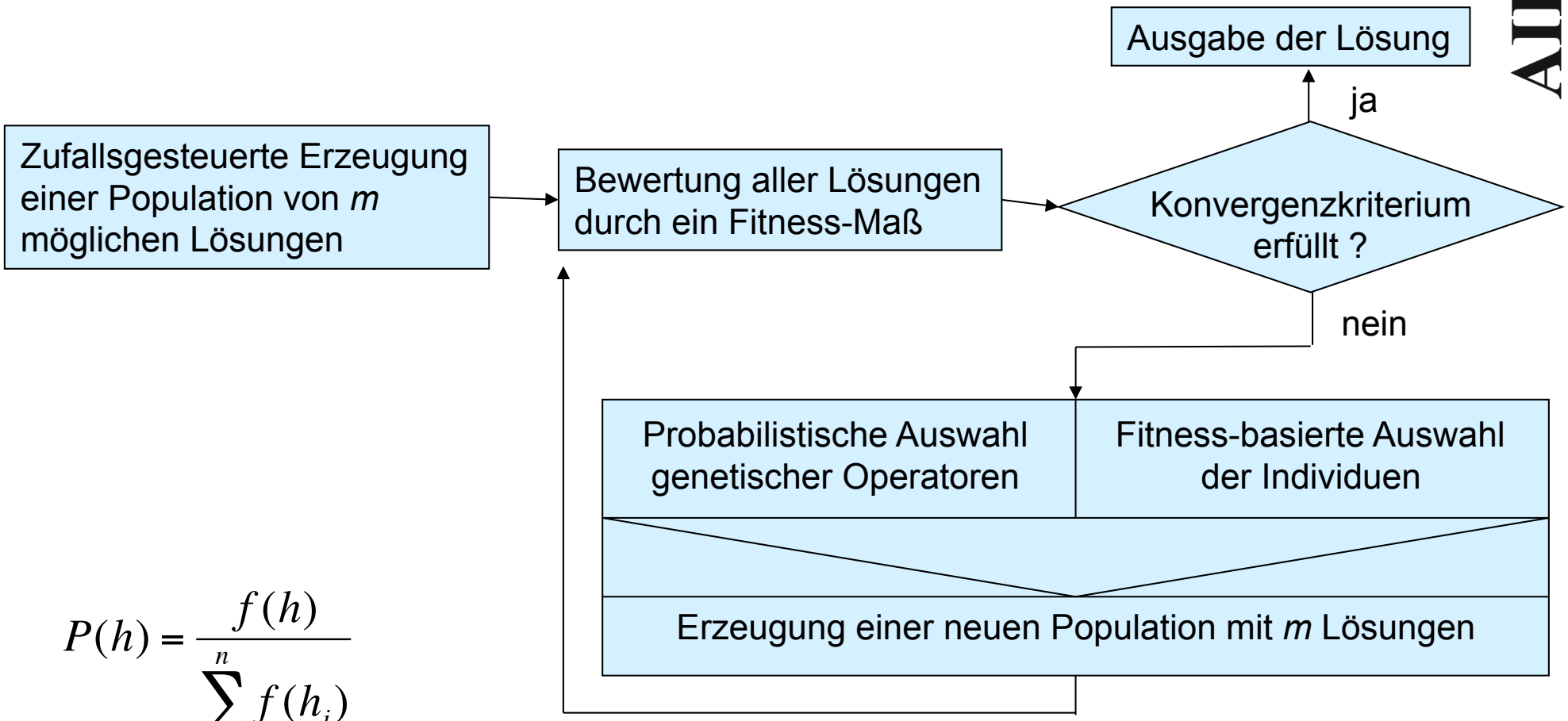
Genetische Algorithmen

Input: eine Menge von Lösungen unterschiedlicher Qualität

Ziel: die beste Lösung



Genetische Algorithmen



$$P(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)}$$

$f(h)$ = Fitness des Individuums h

Wahrscheinlichkeit das Individuum h in einem Selektionsschritt auszuwählen: $P(h)$

Bestandteile

- Repräsentation
- Genetische Operatoren
- Initialisierung
- Fitness-Funktion
- Selektion

- Problemstellung erfordert oft Abweichungen vom klassischen Ansatz

Repräsentation

- Bit-Strings: $S \in \{0,1\}^n$, $n \in \mathbb{N}$

0	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1

- Strings über einem endlichen Alphabet $S \in \Sigma^n$, $n \in \mathbb{N}$

  z.B. $\Sigma = \{G,A,C,T\}$

G	A	G	C	C	A	T	A	T

- Strings, die aus reellen Zahlen bestehen $S \in \mathbb{R}^n$, oder $S \in M \subset \mathbb{R}^n$, $n \in \mathbb{N}$

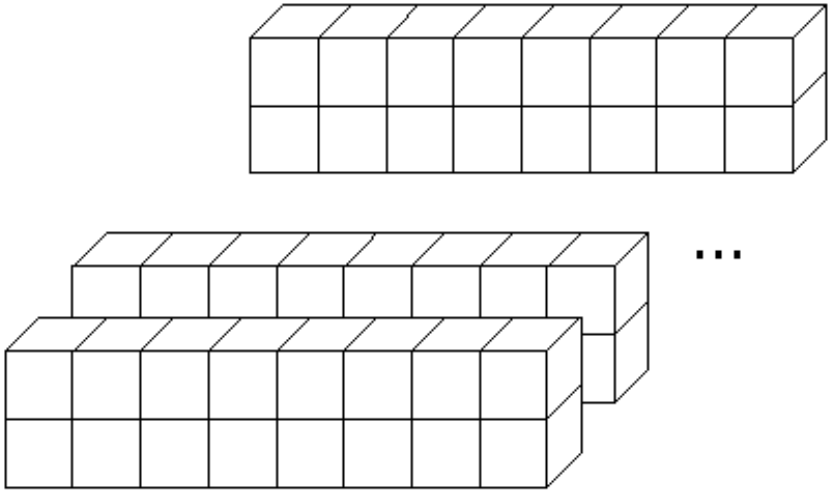
0.1	0.3	0.2	0.2	0.1	0.5	0.9	0.1	0.7

Repräsentation

- Diploide Struktur

0	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1
0	0	1	1	1	1	0	1	0	1	1	0	1	0	0	1	0

- Multiple Chromosen



Repräsentation

- Strings mit variabler Länge

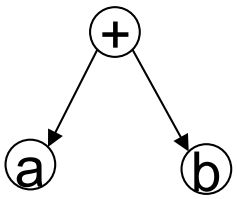
3	177	251	112	-17	-32	-91	999	-25

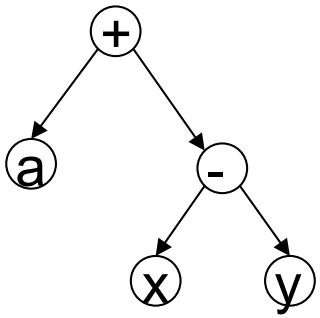
3	177	251	-17	-32	100	-12	-91	999	-25	5	-12

- Permutationen der Menge {1, ..., *n*} , $n \in \mathbb{N}$ (z.B. für TSP)

2	5	4	8	3	1	6	9	7

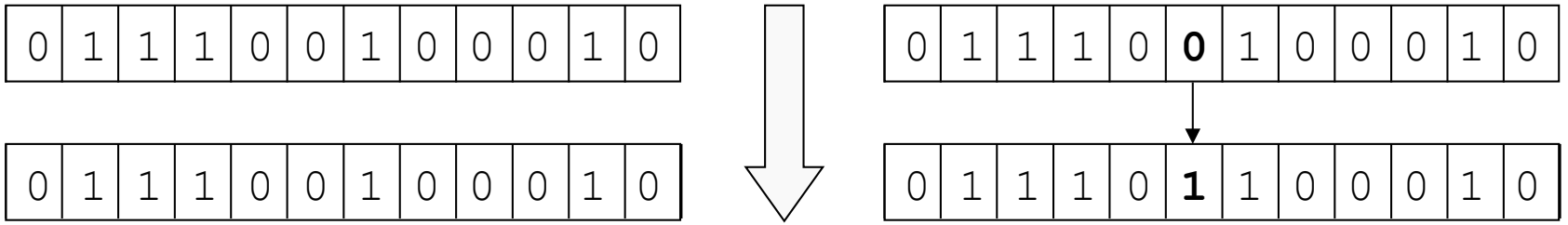
- Strukturen aller Art: Bäume, S-expressions (Genetic Programming), (Neuronale) Netze, ...



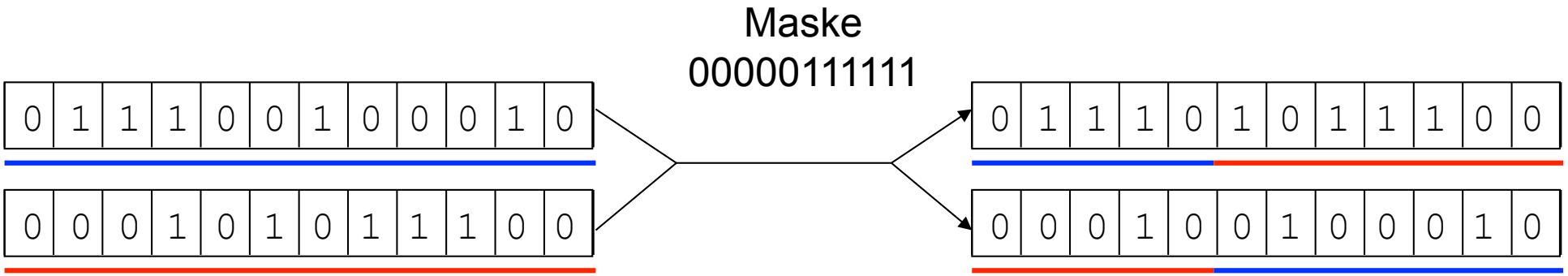


Genetische Operatoren

- Reproduktion, Mutation, Crossover
  - Mutation

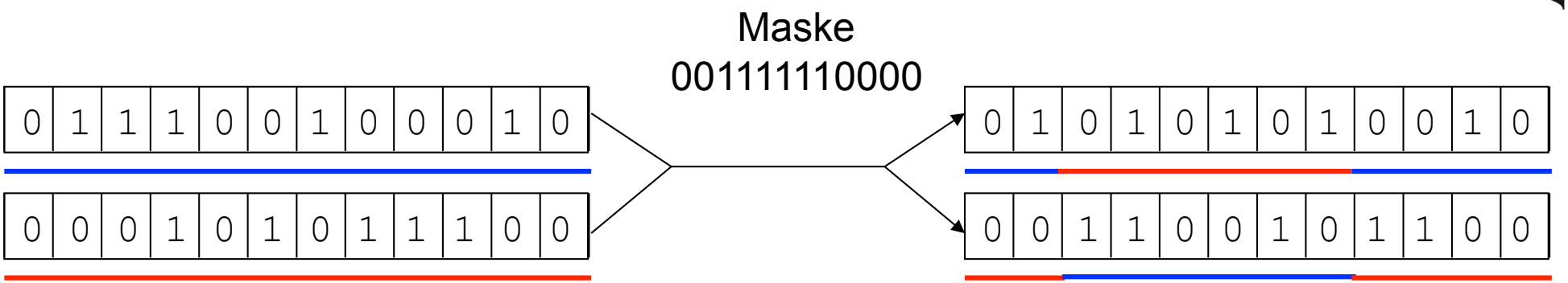


  - Single-Point-Crossover

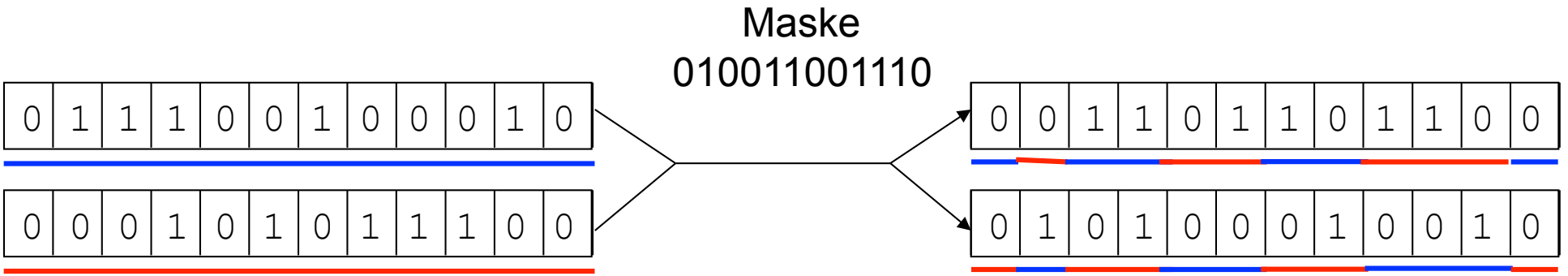


Genetische Operatoren

⇨ Two-Point-Crossover



⇨ Uniform Crossover



Genetische Operatoren

- Mittelwertbildung (für reelle Zahlen)

0.1	0.3	0.2	0.2	0.1	0.5	0.9	0.1	0.7
0.9	0.3	1.8	0.6	1.7	0.7	0.1	0.3	0.5

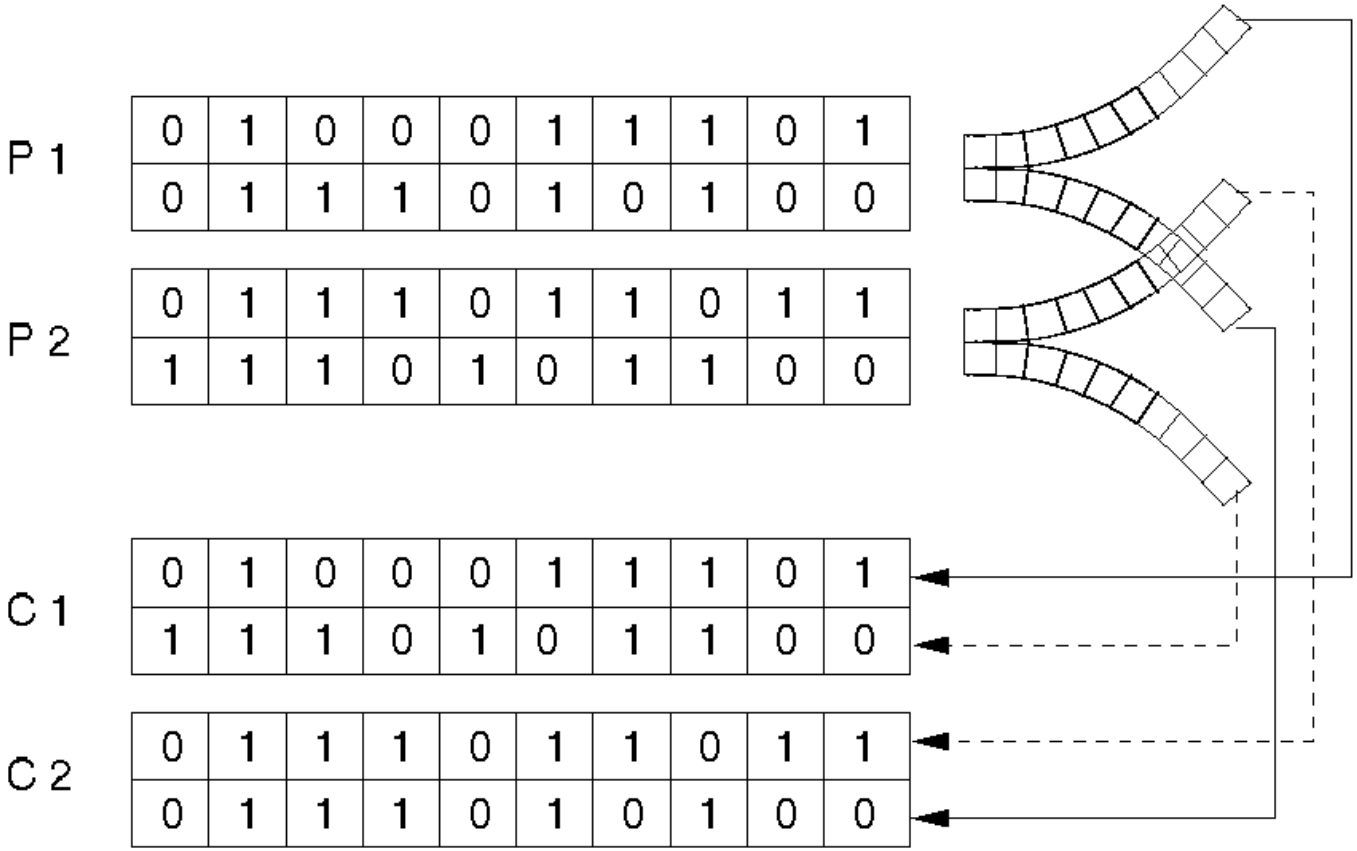
⇒

⇒

0.5	0.3	1.0	0.4	1.4	0.6	0.5	0.2	0.6

Genetische Operatoren

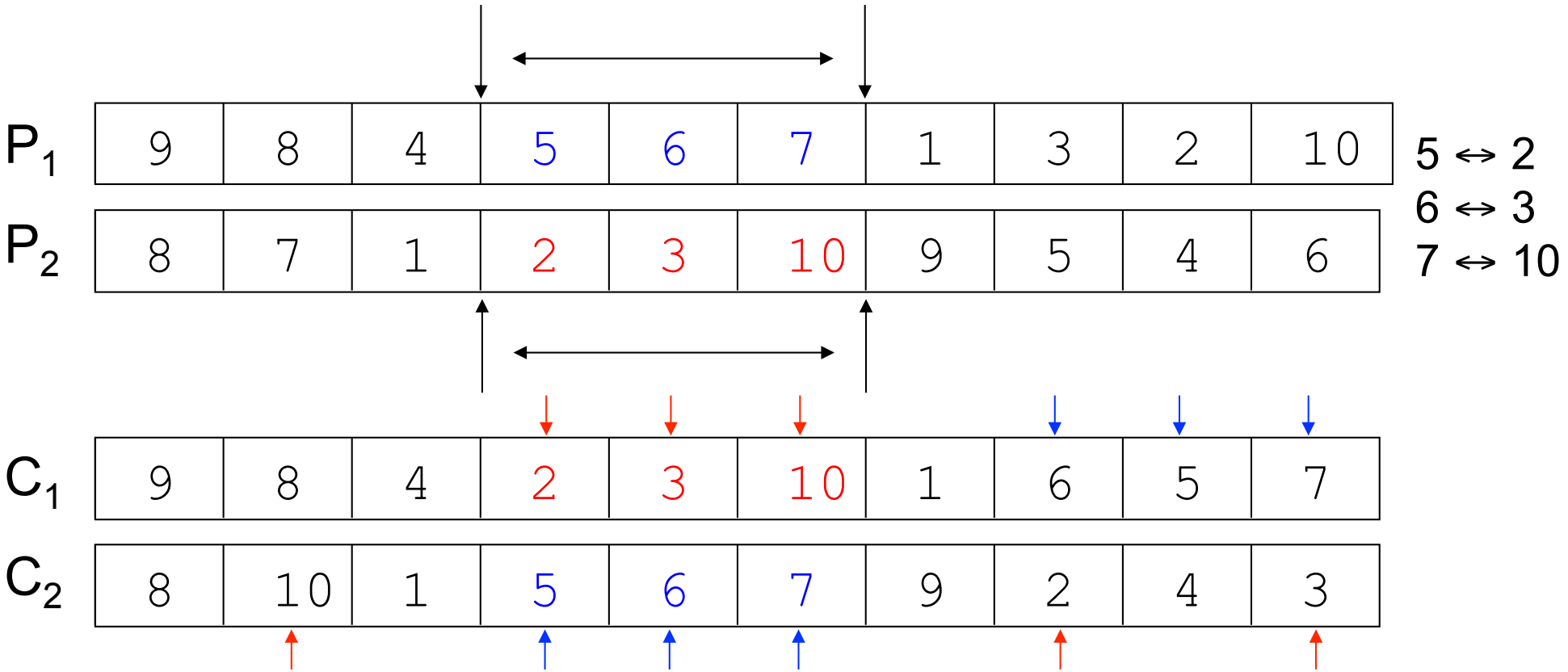
- Segregation (für diploide Chromosomen):



- Translokation (multichromosomales Crossover)
  - ⇨ Austausch von Teilstrings über verschiedene Chromosomen hinweg, wobei die Bedeutung einzelner Gene bewahrt werden muss

Genetische Operatoren

- Umordnung: Partiell matchendes Crossover (z.B. für TSP)
  - ⇨ 2 Crossover-Punkte zufallsbasiert auswählen
  - ⇨ positionsweisen Austausch für alle betroffenen Positionen vornehmen (erst zwischen ausgewählten Punkten, dann Rest konsistent machen).



Initialisierung

- “Intelligente” Initialisierung der Population kann
  - ⇨ ungünstigen Bias vermeiden und dementsprechend
  - ⇨ vorzeitige Konvergenz verhindern und
  - ⇨ Konvergenzverhalten insgesamt verbessern
- Mögliche Initialisierungen:
  - ⇨ die *n* besten einer Menge zufällig gewählter Individuen
  - ⇨ heuristische, bereichsspezifische Regeln
  - ⇨ mit einem anderen Verfahren erlernte Individuen

Selektion

- Auswahl der Individuen
  - proportional zur Fitness (ursprünglicher Ansatz)
  - Tournament Selection: aus *n* Gruppen mit je *t* Individuen jeweils die besten auswählen
  - Truncation Selection: aus der Gruppe T der besten Individuen alle mit gleicher Wahrscheinlichkeit auswählen
  - Linear Ranking Selection: Individuen linear ordnen und dann mit Wahrscheinlichkeit proportional zu ihrem Rang auswählen
  - Exponential Ranking: wie Linear Ranking, jedoch mit exponentiell gewichteten Rangfolgewerten

Selektion

- Auswahl der Operatoren: meist reduziert auf die Optimierung folgender Parameter
  - ⇨ Wahrscheinlichkeit von Crossover
  - ⇨ Wahrscheinlichkeit von Reproduktion
  - ⇨ Wahrscheinlichkeit von Mutation
- z.B. Senkung der Mutationsrate mit der Zeit

Bewertung

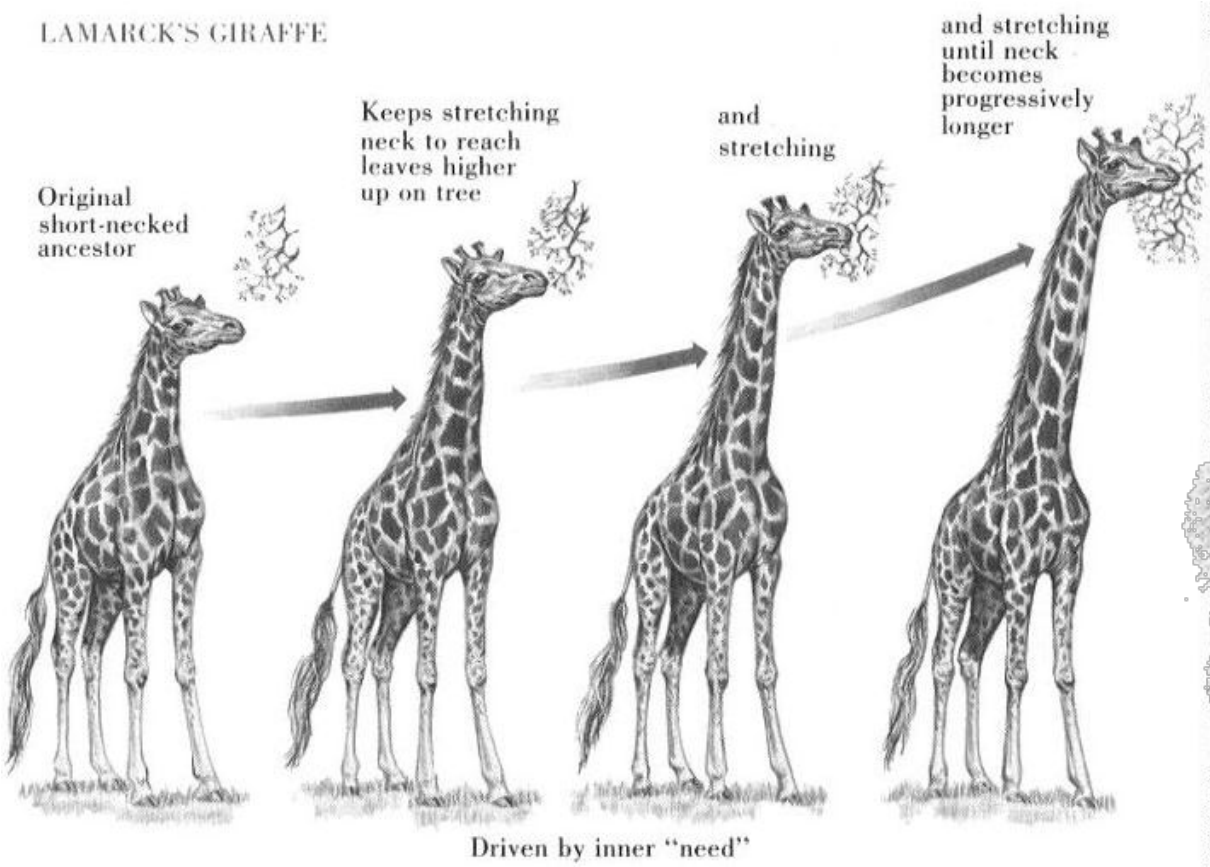
- \+ Globale Suche in riesigen Lösungsräumen möglich
- \+ Konvergenz zu lokalen Minima selten
- \+ bereichsunabhängig (aber nicht problemunabhängig)
- − Repräsentation mit Bitstrings oft unangemessen
- − Schwächen bei lokaler Suche
- − üblicherweise nur eine Lösung als Ergebnis
- − Korrelation zwischen Teilen der Problemlösung wird weitgehend ignoriert und kann zu Konvergenzproblemen führen
  - − nichtlinearer Einfluss auf den Gesamtfitnesswert durch einzelne Gene
  - − Beitrag eines Gens zur Gesamtfitness hängt von anderen Genen ab

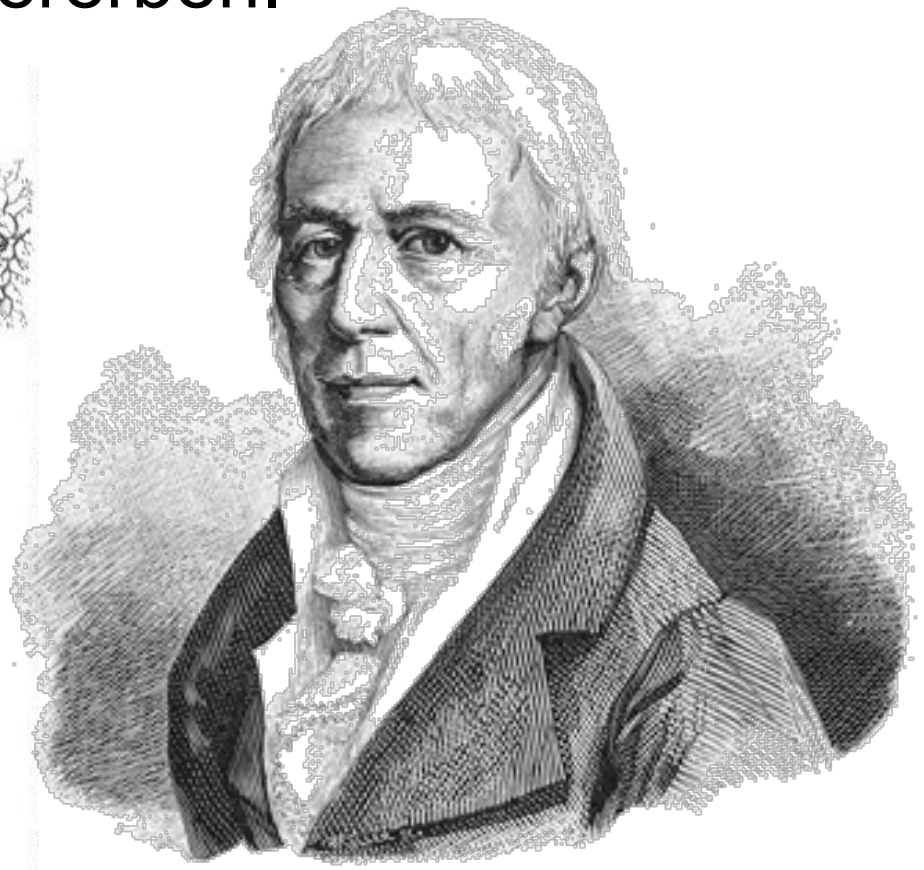
No Free Lunch

- Ohne Berücksichtigung problemspezifischer Randbedingungen kommt der kanonische genetische Algorithmus nicht schneller zu einer optimalen Lösung als eine zufallsbasierte Suche.
  - ⇨ im Mittel ist die Performance über alle Probleme gleich
  - ⇨ wenn bestimmte Probleme besser gelöst werden, werden andere dafür schlechter gelöst
- Das gilt für alle Lösungsverfahren für kombinatorische Optimierungsprobleme

Lamarck + Baldwin

- **Lamarck'sche Hypothese**: Individuen können die während ihres Lebens erlernten/erworbenen Fähigkeiten/Eigenschaften vererben.





Lamarck + Baldwin

- **Baldwin-Effekt**: Lernfähige (und damit besonders anpassungsfähige) Individuen haben eine höhere Reproduktionsrate. Erlernte Verhaltensweisen befördern Selektion entsprechender Instinkte.
- Interaktion zwischen (individuellem) Lernen und Evolution.