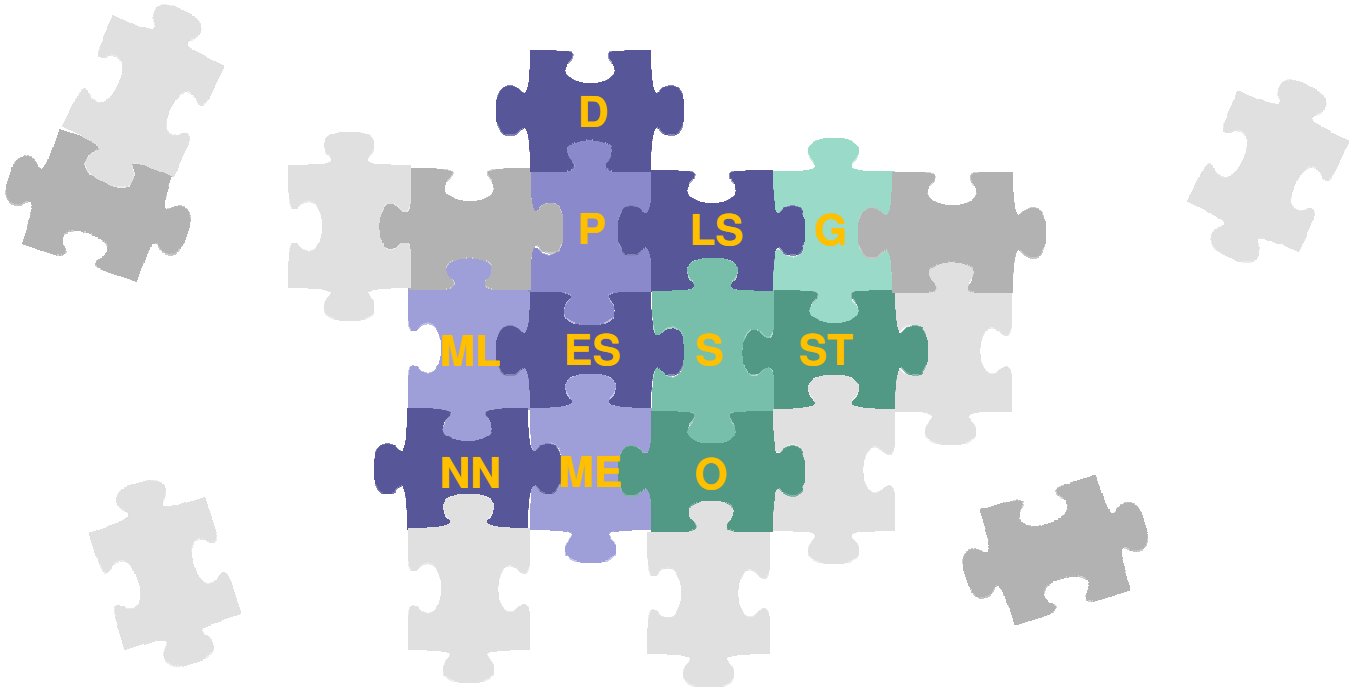
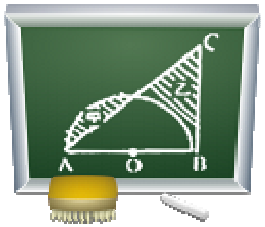
Einführung in die Künstliche Intelligenz

- Maschinelles Lernen -



Übersicht zum Vorlesungsinhalt

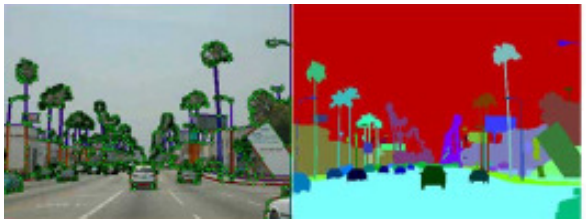
- ✓ 1) Definition und Geschichte der KI, PROLOG
- ✓ 2) Expertensysteme
- ✓ 3) Logisches Schließen, Resolution
- ✓ 4) Suche und Spieltheorie
- ✓ 5) Optimierungen und Heuristiken (Spieleprogrammierung)
- ✓ 6) Mustererkennung
- ✓ 7) Neuronale Netze
- ✓ 8) General Game Playing
- ➡ 9) **Maschinelles Lernen**
- 10) Genetische und evolutionäre Algorithmen
- 11) Bildverarbeitung
- 12) Robotik, Pathfinding



Maschinelles Lernen

Was assoziieren wir mit dem Begriff „Maschinelles Lernen“(ML)?

Muster und Objekte erkennen, Datenbanken von Wissen erstellen, Abstraktion, ...



[18]

ML-Kategorien

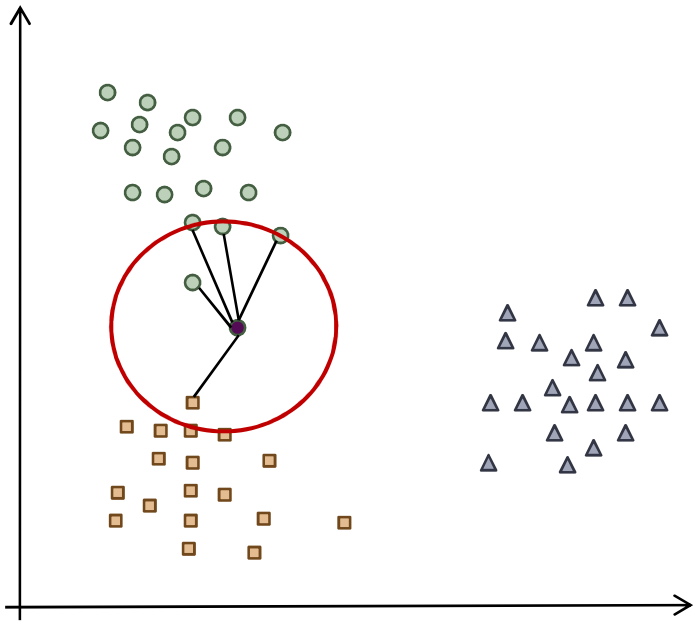
überwachtes Lernen: Es gibt einen Lehrer, der eine Menge von Eingabe- und Ausgabebeispielen vorgibt und somit aktiv am Lernvorgang des Systems beteiligt ist.

unüberwachtes Lernen: Für eine Menge von Eingaben erzeugt das System selbstständig eine Repräsentation (ein Modell), die es ihm ermöglicht, eine Klassifizierung vorzunehmen. Das System kennt dabei nicht notwendigerweise die semantische Bedeutung der Ausgabe.

verstärkendes Lernen: Das System lernt durch Belohnung oder /und Bestrafung, seine Handlungen /Aktionen bestimmten Situationen anzupassen, um eine gegebene Nutzenfunktion zu optimieren.

Mustererkennung – k-nn

- Repräsentation des Datenraumes durch viele (gute) Beispiele
- Mehrheitsentscheidung auf Grund der **k**-nächstgelegenen Nachbarn (**k** klein und ungerade), keine Trainingsphase (eher ein Auswendiglernen)
- verschiedene Metriken können zum Einsatz kommen



Euklidische Distanz

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

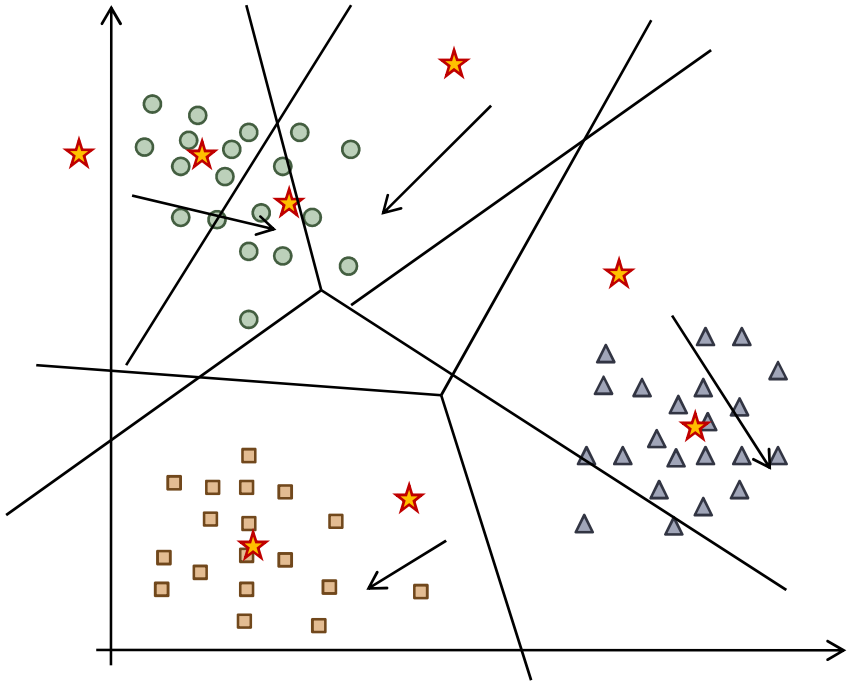
Manhattan Distanz

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

- sehr gute Erkennungsraten, wenn viele Beispiele gut gewählt und keine Rauschelemente vorhanden sind
- langsam, da Abstand zu jedem Beispiel der Datenbank berechnet wird, keine Generalisierung

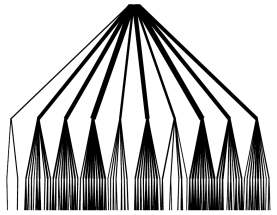
Mustererkennung – k-means

- **k** Prototypen werden zufällig im Raum verteilt, dann werden alle Beispiele über eine Abstandsfunktion dem jeweils nächstgelegenen Prototypen zugeordnet (lineare Trennung des Datenraumes). Nach der Zuordnung werden die Prototypen auf die jeweiligen Zentren verschoben. Wiederholung bis Abbruchkritierium, dann labeln der Prototypen.
- Trainingsphase kann viele Iterationen benötigen und dabei die Anzahl der Prototypen varriieren



- gute Erkennungsraten, gute Generalisierung, Rauschelemente können vorhanden sein
- schnell, da Abstand nur zu den Prototypen berechnet wird, konvergiert nicht immer

Mustererkennung – rekursiver k-means



- Lineare Trennung (rekursiv) des Datenraumes durch Prototypen
- Reduktion von unterschiedlich auf gleichdimensionales Problem

Exklusiviätsmaß

$$w_i = ln(\frac{N}{N_i})$$

Fingerprint

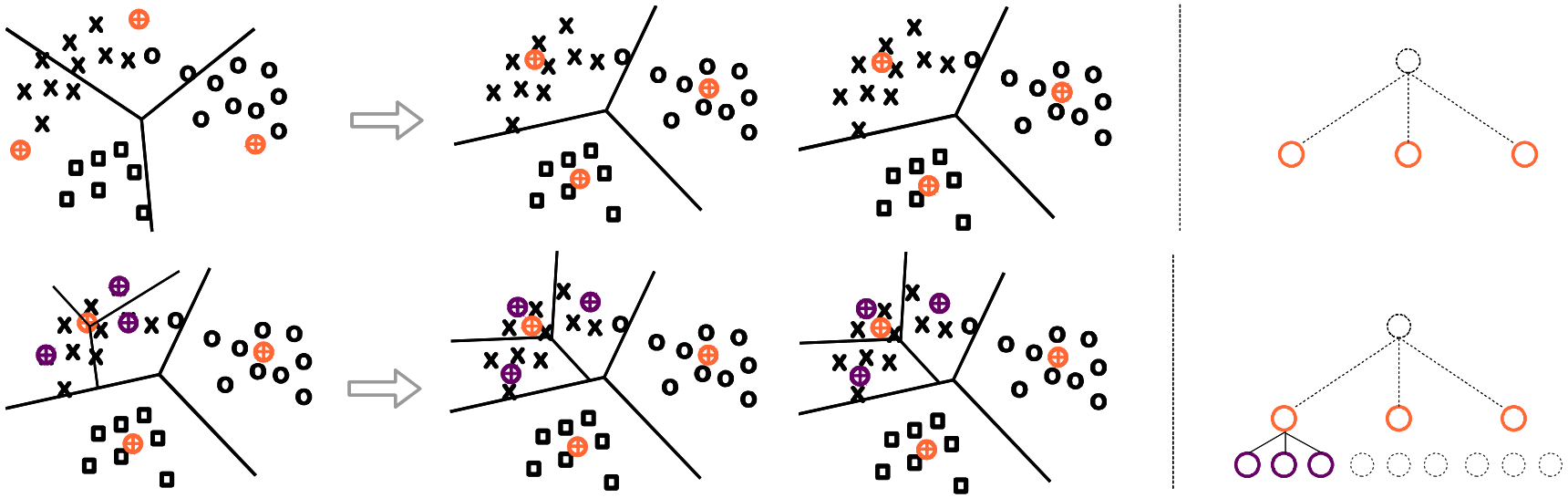
$$d_i = w_i m_i$$
$$q_i = w_i n_i$$

Normierung und Abstand

$$s(\vec{q}, \vec{d}) = || \frac{\vec{q}}{||\vec{q}||} - \frac{\vec{d}}{||\vec{d}||} ||$$

Erkennung

$$argmin_i s(\vec{q}, \vec{d_i})$$



Max. des Skalarprodukts

$$max < \vec{q}, \vec{d_i} >$$

Beschleunigung

$$max \sum_j q_j d_{ij}$$

Paper zum Nachlesen [12]

Mustererkennung – ein weites Forschungsfeld

Gauß-Klassifikator

Online-Clustering

Fisher-Linear-Discriminant

Lineare Regression

FeedForward-Netze

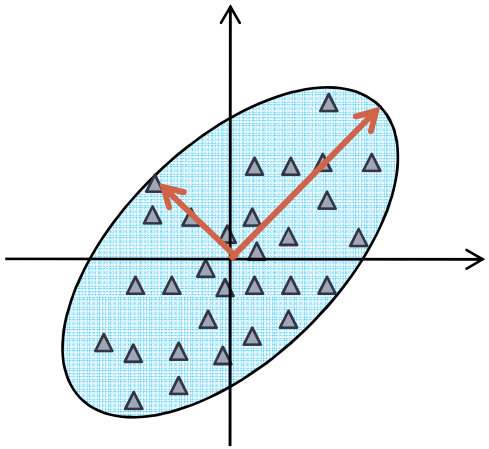
Support-Vector-Machines

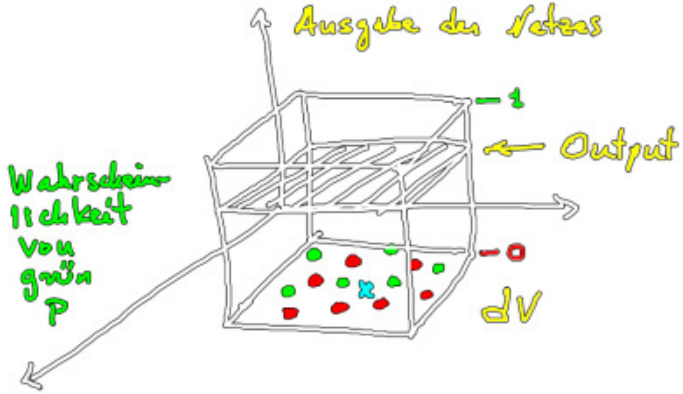
Principal Component Analysis

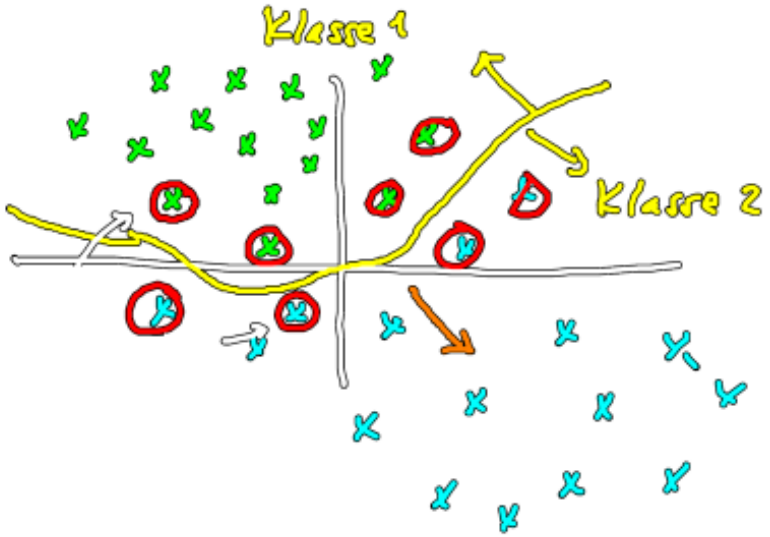
Independent Component Analysis

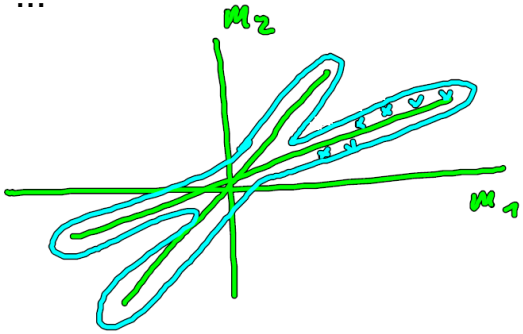
Neuronale Netze

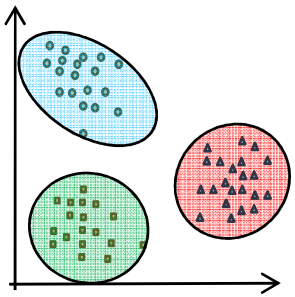
...





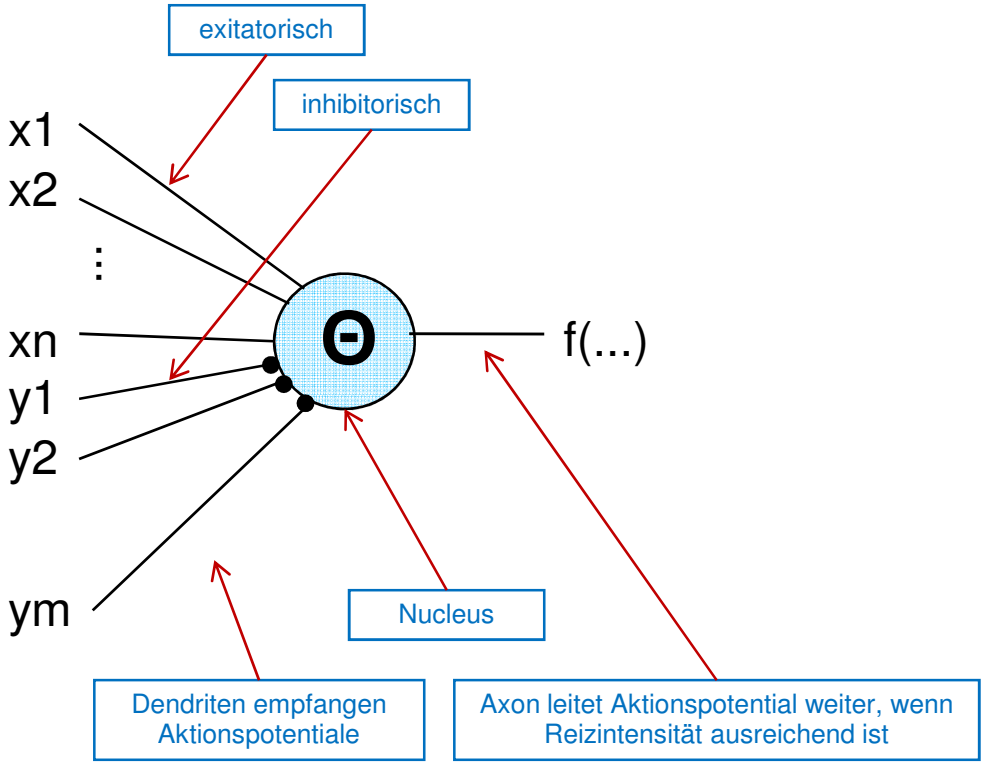






Neuronale Netze – McCulloch/Pitts

Modell eines Neurons von McCulloch/Pitts (1943) als kleinste Einheit.

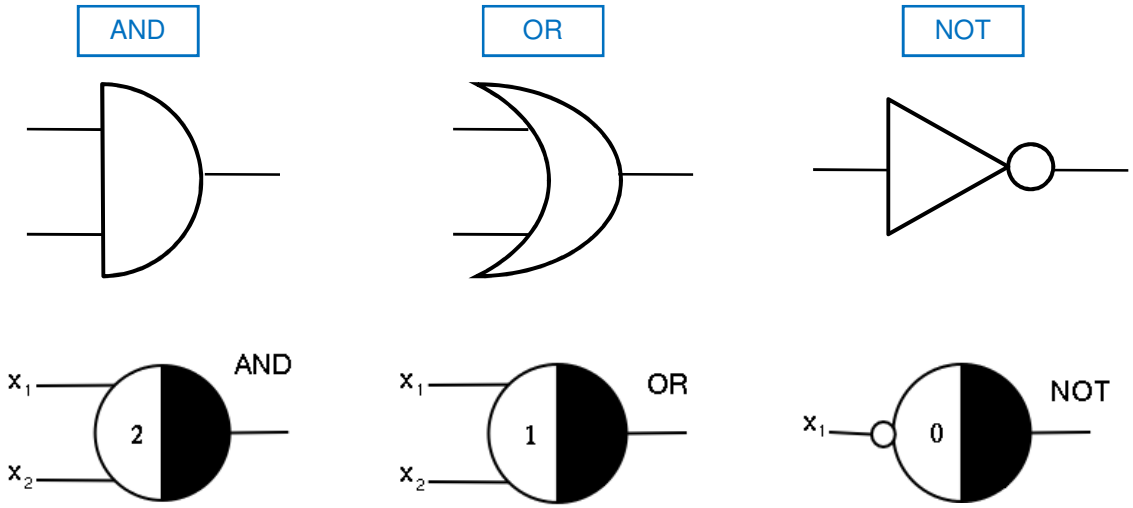


$$f(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m) = \begin{cases} 0 & , falls\, y_i = 1\, mit\, i \in \{1, ..., m\} \vee \sum_{i=1}^{n} x_i < \Theta \\ 1 & , falls\, y_1 = y_2 = ... = y_m = 0 \wedge \sum_{i=1}^{n} x_i \geq \Theta \end{cases}$$

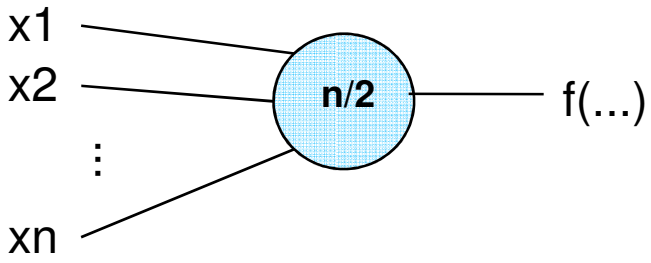
Paper zum Nachlesen [13]

Neuronale Netze – McCulloch/Pitts

Die McCulloch/Pittszelle stellt eine vollständige Basis dar. Demzufolge lassen sich Computer mit diesem Baustein realisieren.

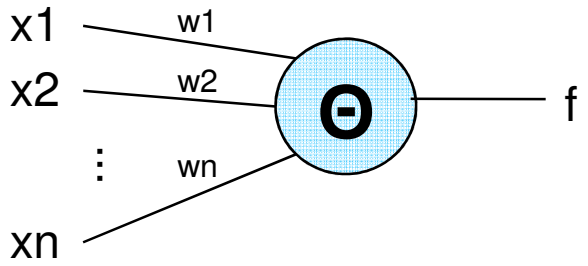


McCulloch-Pitts-Netz (Threshold-Gatter) sind aber allgemeiner als klassische logische Gatter. So zum Beispiel der Mehrheitsentscheider:



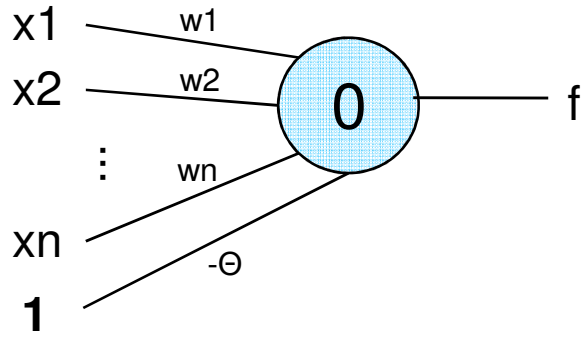
Neuronale Netze – Erweiterung zum Perceptron

Seit 1958 ist das Perceptron (nach Rosenblatt) Grundlage für die künstlichen neuronalen Netze. Anstatt exitatorische und inhibitorische Kanten zu verwenden, gibt es Gewichte (pos/neg), mit deren Hilfe einem gegebenen Netz ohne Architekturänderungen unterschiedliche Aufgaben antrainiert werden können.



$$f = \begin{cases} 1 & , falls \sum_{i=1}^{n} w_i x_i \geq \Theta \\ 0 & , sonst \end{cases}$$

Äquivalent dazu ist:



$$f = \begin{cases} 1 & , falls \sum_{i=1}^{n} w_i x_i - \Theta \geq 0 \\ 0 & , sonst \end{cases}$$

Die zweite Darstellung hat den Vorteil, dass alle „lernbaren“ Parameter auf den Kanten stehen und somit Theta mitgelernt werden kann.

Neuronale Netze - Backpropagation

Optimierung einer abstrakten Funktion, die in der Lage sein soll, einen Inputvektor **x** in einen Outputvektor **y** derart zu überführen, dass eine Generalisierung der Eingabe möglich ist und ähnliche Eingaben zu x ebenfalls als y erkannt werden. Gegeben ist eine Trainingsmenge T={(**x1**,**y1**), (**x2**,**y2**), ... , (**xN**,**yN**)}

Vorverarbeitung

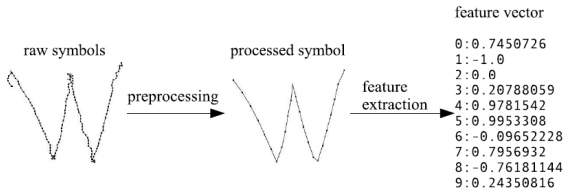
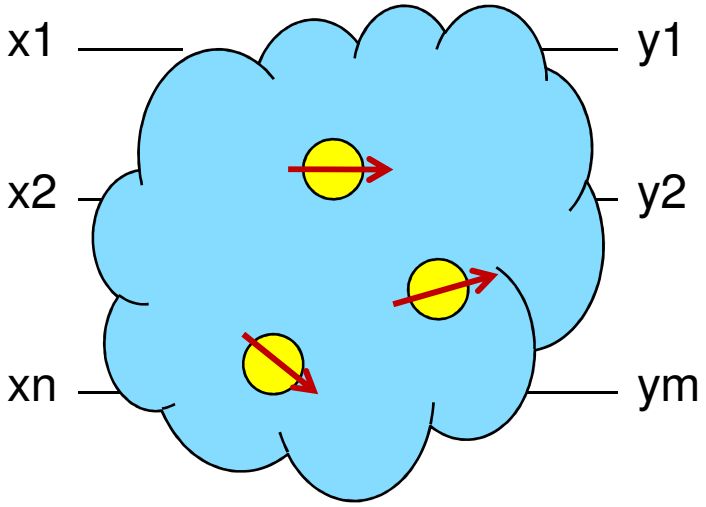


Abbildung aus [14]

Input n-dim | BLACKBOX | Output m-dim



Klassifizierung

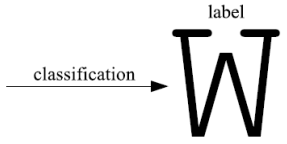


Abbildung aus [14]

In der **Trainingsphase** wird eine Fehlerschicht an das Netz gepackt, um damit die Änderungen der Gewichte zu ermitteln.

Neuronale Netze - Backpropagation

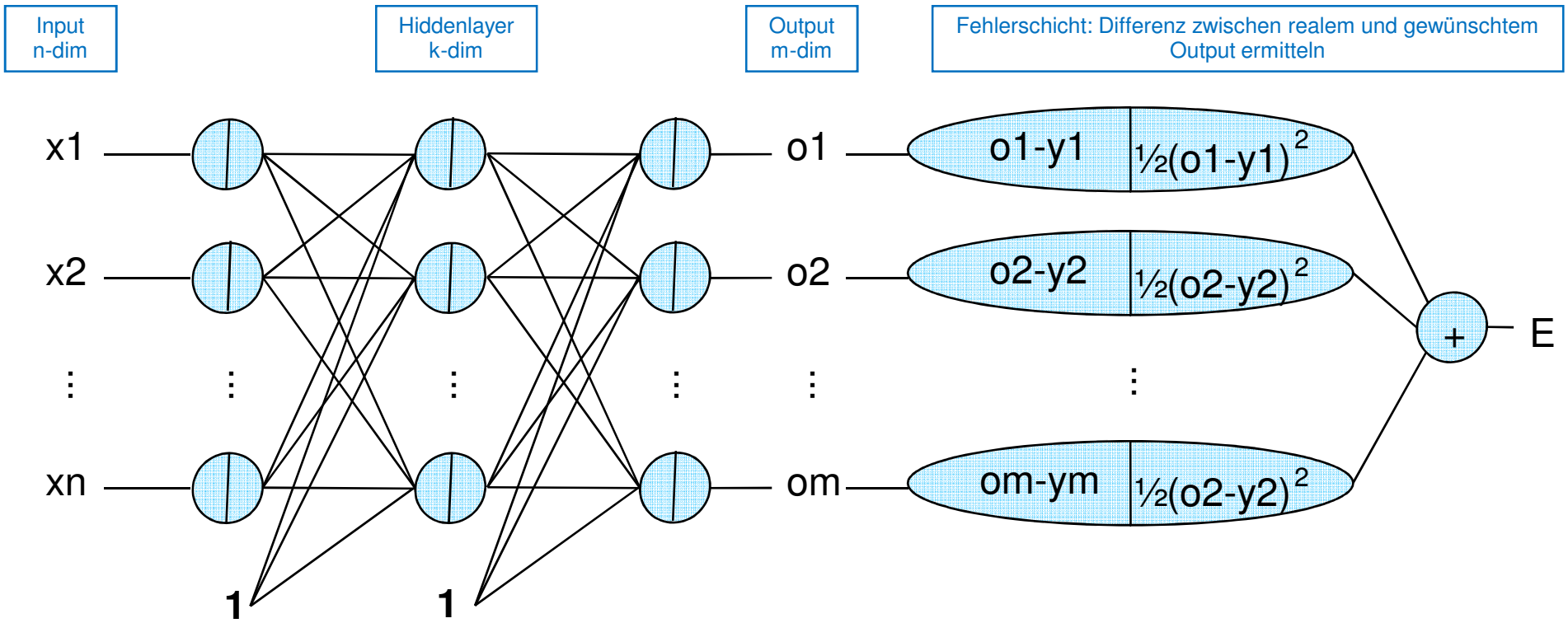
Sigmoide und Ableitung

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

$$\frac{ds(x)}{dx} = s(x)(1-s(x))$$

Architektur: eine verdeckte Schicht und Sigmoid-Funktion in den Neuronen

Trainingsphase:



FF-Schritt: Zuf. x an das Netz, von links nach rechts durchlaufen, Ableitungen speichern

BP-Schritt: Konstante 1 anlegen, von rechts nach links durchlaufen, Korrekturen speichern

Neuronale Netze – Backpropagation online/offline

Offline-Backpropagation

In der Offline-Variante wird der Fehler für alle Vektoren der Trainingsbasis ermittelt und die Gewichte anschließend in die Richtung der globalen Gradientenrichtung (negiert) korrigiert.

$$\frac{1}{N} \sum_{i=1}^{N} \vec{\nabla} E_i = \vec{\nabla} E_{global}$$

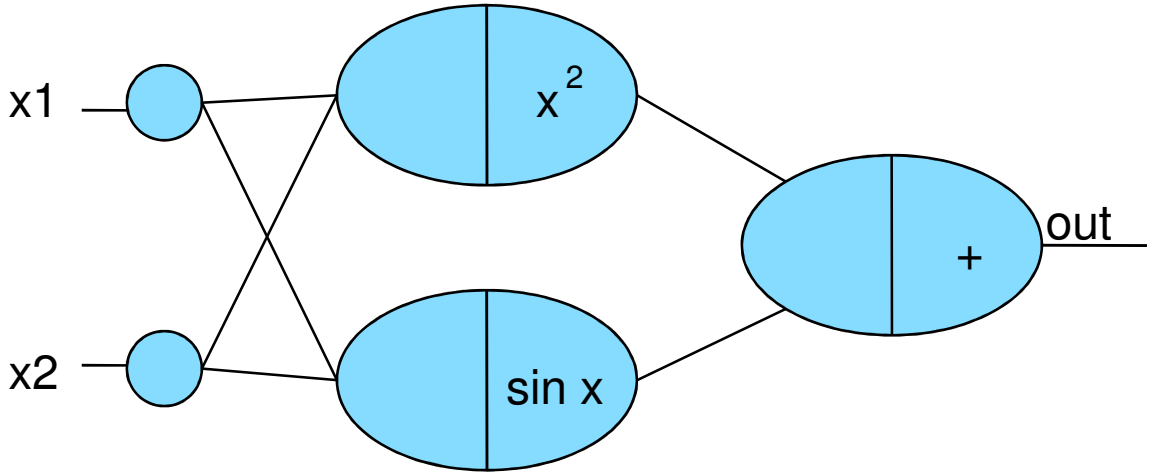
Online-Backpropagation

In der Online-Variante wird ein zufälliger Vektor gewählt, anschließend die Gewichte korrigert und Gamma verkleinert. Ist schneller als die offline-Variante, führt aber oft zu lokalen Minima.

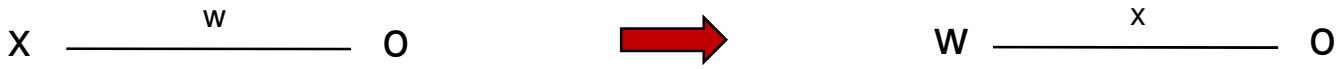
$$\vec{\nabla} E_{local}$$

Konkretes Beispiel für Berechnung der Ableitungen

Wir konstruieren eine einfache Netzarchitektur und berechnen anschließend die Gewichte



Wir erinnern uns dabei an den folgenden (sehr wichtigen!) Trick. Im Backpropagationschritt wird die Tatsache verwendet, dass **x** und **w** vertauscht werden können. Wir wollen nach **w** und nicht nach **x** ableiten!

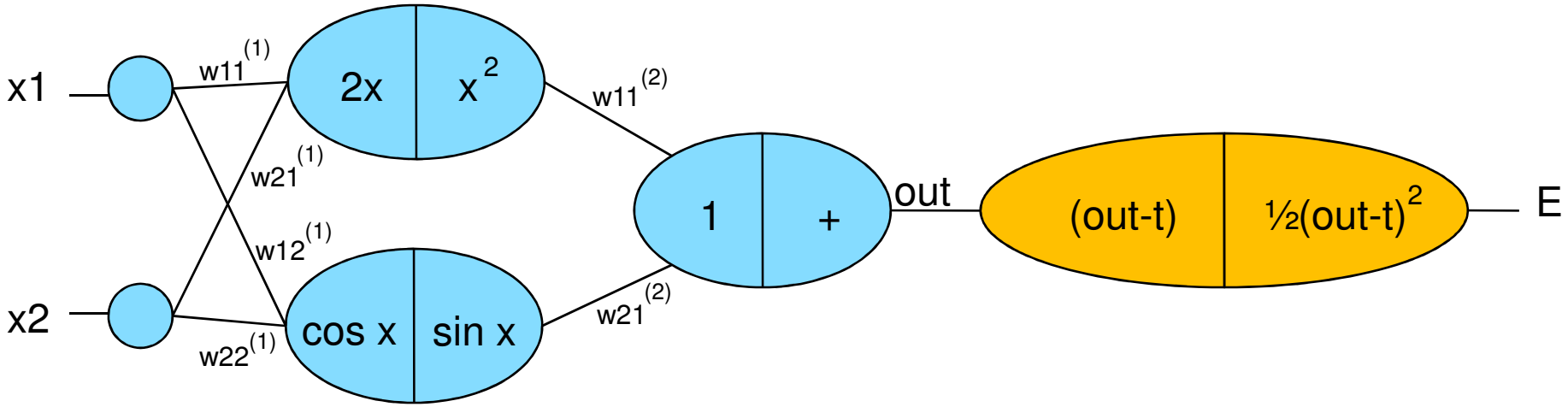


Konkretes Beispiel für Berechnung der Ableitungen

Wir konstruieren eine einfache Netzarchitektur und berechnen anschließend die Änderungen der einzelnen Gewichte.

Ableitungen werden berechnet

Fehlerschicht kommt hinzu



negative Gradientenrichtung

$$w_i = w_i + (-\gamma) \cdot \triangle w_i$$

„variable Lernkonstante“ (c) Rojas

Update für Gewichte ermitteln

$$\triangle w_{11}^{(2)} = 1 \cdot (out - t) \cdot (x_1 \cdot w_{11}^{(1)} + x_2 \cdot w_{21}^{(1)})^2$$

$$\triangle w_{21}^{(2)} = 1 \cdot (out - t) \cdot sin(x_1 \cdot w_{12}^{(1)} + x_2 \cdot w_{22}^{(1)})$$

$$\triangle w_{11}^{(1)} = 1 \cdot (out - t) \cdot w_{11}^{(2)} \cdot 2(x_1 \cdot w_{11}^{(1)} + x_2 \cdot w_{21}^{(1)}) \cdot x_1$$

Neuronale Netze - BP-Varianten

Zur Beschleunigung von Backpropagation gibt es viele Verfahren. Dabei ist die Änderung des Gewichtes abhängig von Gamma:

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

Momentum

$$\Delta w_k(i) = -\gamma \frac{\partial E}{\partial w_k} + \alpha \Delta w_k(i-1),$$

Anstatt nur in die aktuell negative Gradientenrichtung zu laufen, orientieren wir uns ebenfalls an der vorhergehenden.

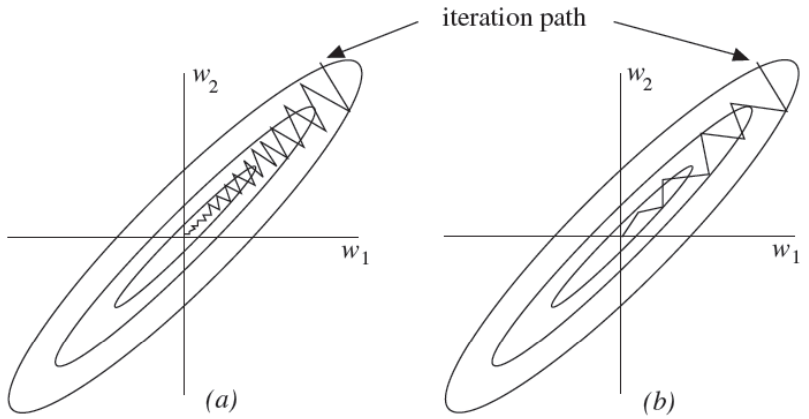


Fig. 8.1. Backpropagation without (a) or with (b) momentum term

Silva-Almeida

$$\Delta^{(k)} w_i = -\gamma_i^{(k)} \nabla_i E^{(k)}$$

Wenn die Richtung gleich bleibt, vergrößert sich die Schrittgröße.

$$\gamma_i^{(k+1)} = \begin{cases} \gamma_i^{(k)} u \text{ if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} \geq 0 \\ \gamma_i^{(k)} d \text{ if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} < 0 \end{cases}$$

Rprop

$$\Delta^{(k)} w_i = -\gamma_i^{(k)} \operatorname{sgn}(\nabla_i E^{(k)})$$

Ähnlich zu Silva-Almeida, Schrittgrößen sind nach oben begrenzt.

$$\gamma_i^{(k+1)} = \begin{cases} \min(\gamma_i^{(k)} u, \gamma_{max}) \text{ if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} > 0 \\ \max(\gamma_i^{(k)} d, \gamma_{min}) \text{ if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} < 0 \\ \gamma_i^{(k)} \qquad \text{otherwise,} \end{cases}$$

Abbildungen aus [15]

Neuronale Netze - Assoziativspeicher

- Netze mit Rückkopplung (rekurrent)
- Ausgabe wird wieder als Eingabe verwendet

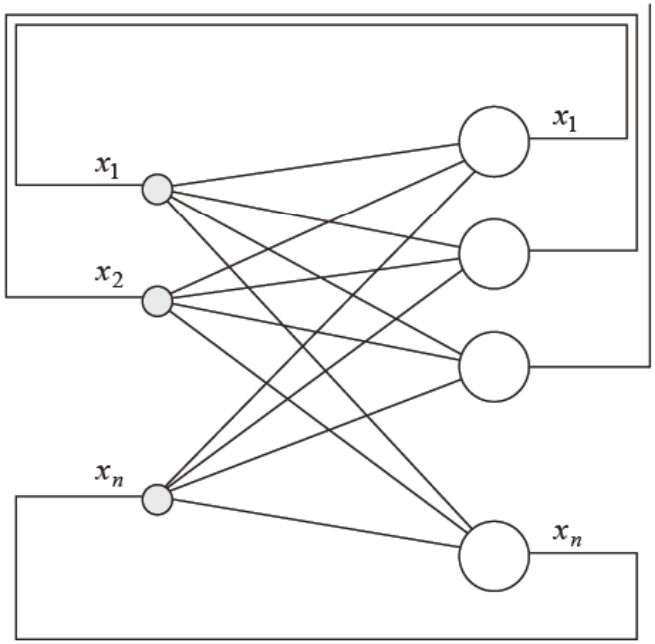


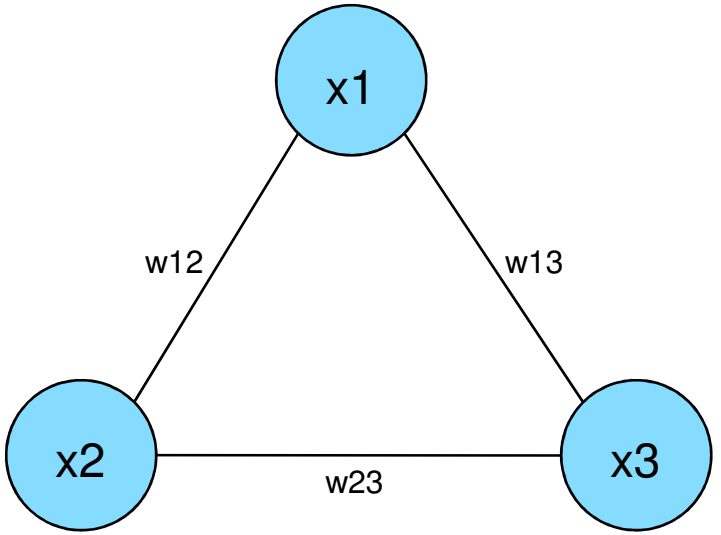
Fig. 12.3. Autoassociative network with feedback

Abbildungen aus [15]

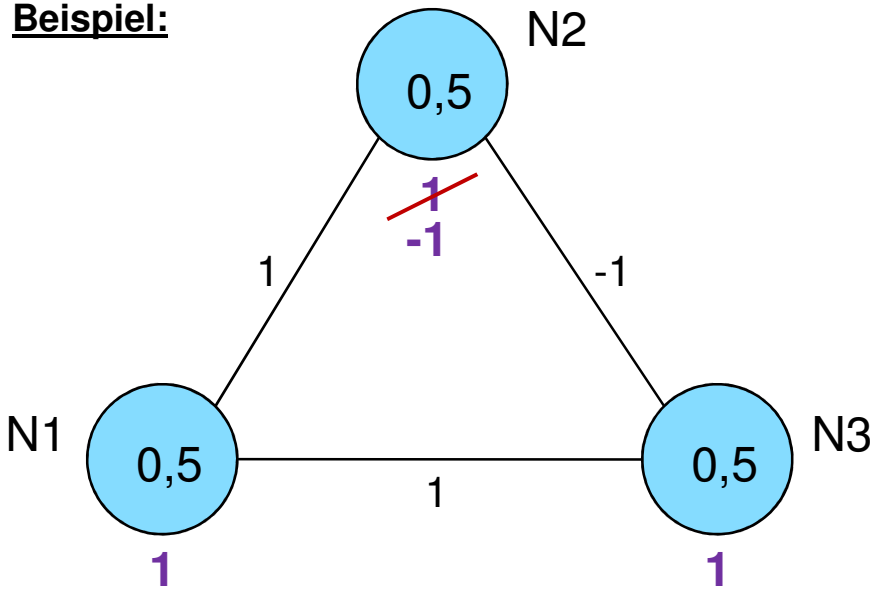
Neuronale Netze - Hopfieldnetz

Modell wurde 1982 von John Hopfield entwickelt und vorgestellt.

- Asynchrones, rekurrentes Netz
- Zustände und Kanten haben bipolare Gewichte {1, -1}
- symmetrisch, Kanten sind bidirektional
- Zustände der Neuronen beschreiben einen abstraktes Muster



Beispiel:



Netzkonfiguration

N1	N2	N3	
1	1	1	← Startkonfiguration
1	-1	1	
	⋮		
-1	-1	-1	

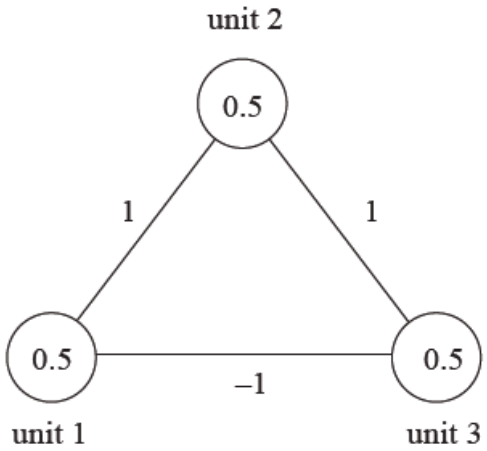
Konvergiert das Netz in einen stabilen Zustand?

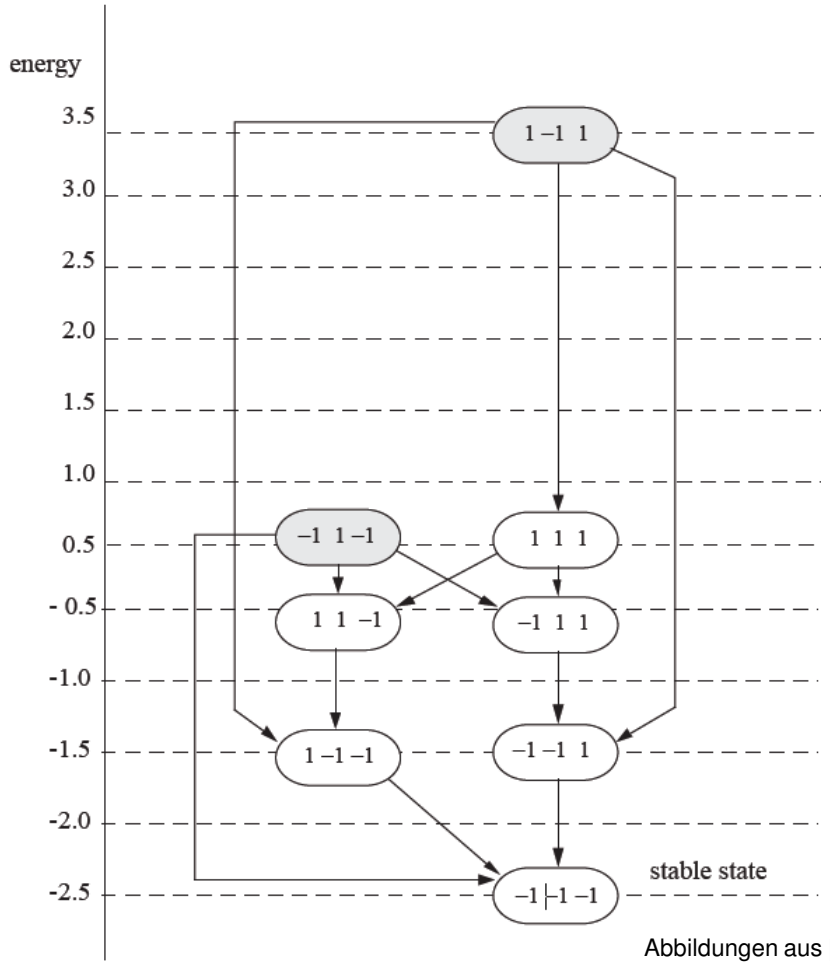
Abbildungen aus [15]

Neuronale Netze - Hopfieldnetz

Energie eines Netzzustandes ist definiert als $$E(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i x_i,$$

Schauen wir uns die Energieniveaus aller Netzzustände für das folgende Beispiel an





Abbildungen aus [15]

Neuronale Netze - Konvergenzbeweis

Energie eines Netzzustandes ist definiert als $$E(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i x_i,$$

Konvergenzbeweis:

Der Folgezustand, falls eine Änderung des Zustandes x eintritt, ist energieärmer, also:

$$E(\mathbf{x}) - E(\mathbf{x}') = (-\sum_{j=1}^{n} w_{kj}x_k x_j + \theta_k x_k) - (-\sum_{j=1}^{n} w_{kj}x'_k x_j + \theta_k x'_k)$$

Etwas umformen und es erscheint die uns bereits bekannte Aktivierungsfunktion eines Neurons:

$$E(\mathbf{x}) - E(\mathbf{x}') = -(x_k - x'_k)\sum_{j=1}^{n} w_{kj}x_j + \theta_k(x_k - x'_k)$$

$$= -(x_k - x'_k)(\sum_{j=1}^{n} w_{kj}x_j - \theta_k),$$

$$E(\mathbf{x}) - E(\mathbf{x}') = -(x_k - x'_k)e_k$$

Aktivierungsfunktion eines Neurons

Über die Vorzeichen von x und x‘ lassen sich beide Fälle abdecken und es folgt

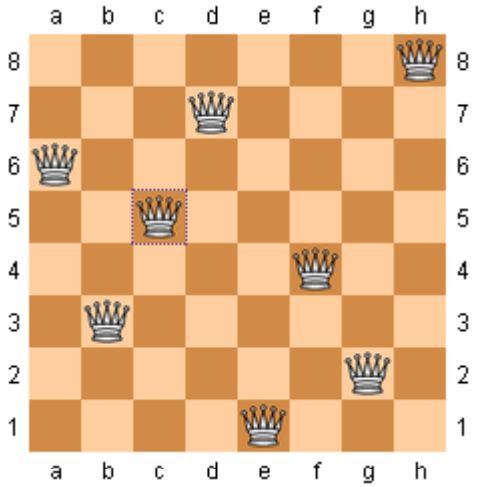
$$E(\mathbf{x}) - E(\mathbf{x}') > 0.$$

Abbildungen aus [15]

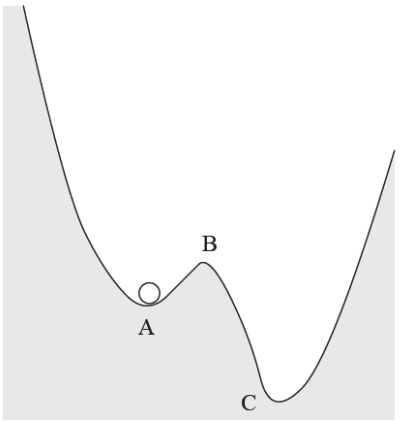
Neuronale Netze - Boltzmannmaschine

Modell wurde 1986 von Hinton/Sejnowski entwickelt und vorgestellt.

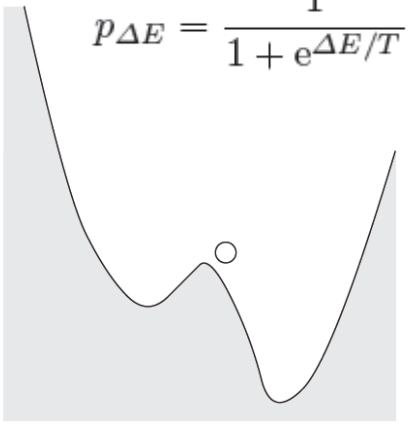
- Asynchrones, rekurrentes Netz
- Gewichte der Zustände {0, 1}
- symmetrisch, Kanten sind bidirektional
- Keine Rückkopplung wii=0
- Zustände der Neuronen beschreiben einen abstraktes Muster
- Änderung der Zustände findet stochastisch statt



Simulated annealing



$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E/T}}$$



Boltzmann-Maschine

$$x_i = \begin{cases} 1 & with\ probability\ \ p_i, \\ 0 & with\ probability\ 1 - p_i, \end{cases}$$

$$p_i = \frac{1}{1 + \exp(-(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)/T)}$$

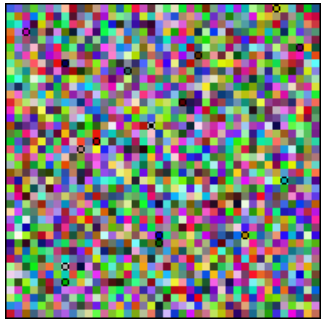
➡ **Markov-Modelle**

Abbildungen aus [15]

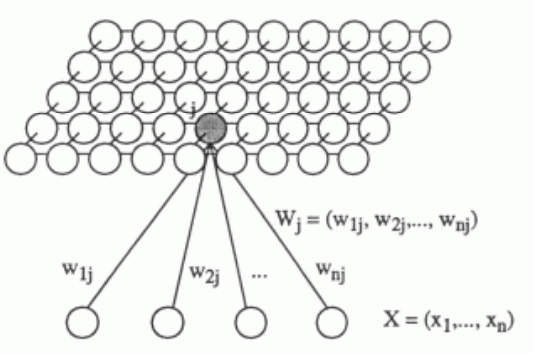
Neuronale Netze - Kohonennetze

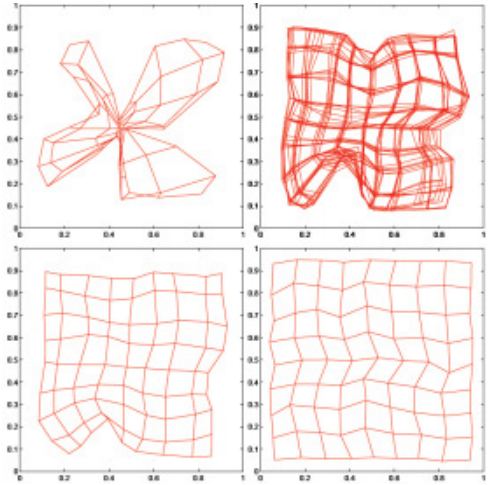
- Selbstorganisierende Netze (SOMs)
- Gegeben ist eine Netztopologie 1D, 2D, 3D, …
  Gewichte der Knoten werden zufällig gewählt
  Kanten repräsentieren die logische Nachbarschaft

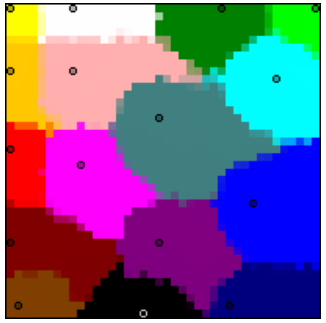
Zielraum



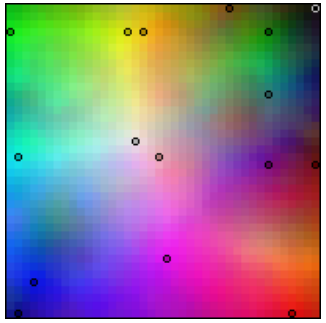
Zufällige Startkonstellation







limitiert auf 16 Farben



kompletter RGB.-Farbraum

- Datum x aus Zielraum wird zufällig gewählt und das nächstliegende Datum n im SOM ermittelt.
- Im Updateschritt wird der Knotenwert n im SOM in Richtung x gezogen und die Nachbarn ebenfalls ein wenig mit

Abbildungen aus [11]

Neuronale Netze – weites Forschungsfeld

Perzeptron-Lernen

Assoziativspeicher

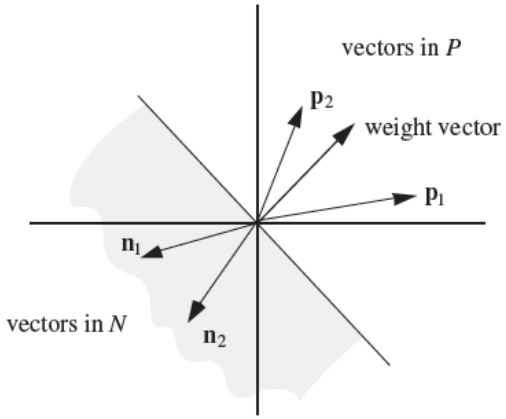
Markov-Modelle

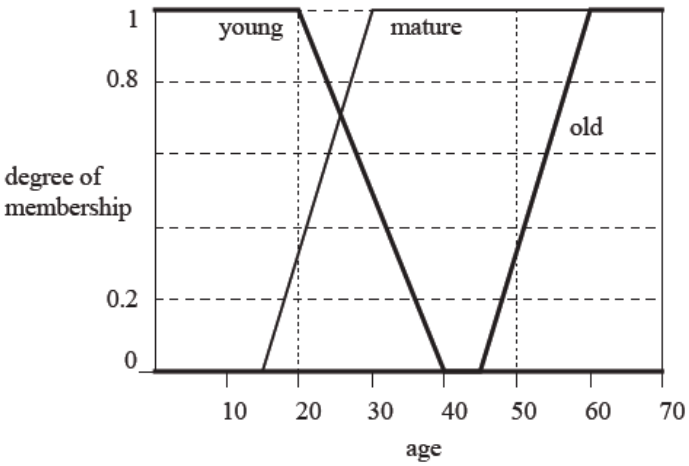
Genetische Algorithmen

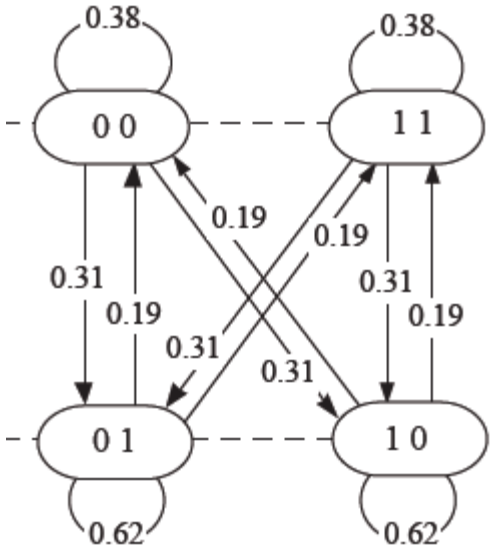
Fuzzy-Logik

Hybride Modelle

...



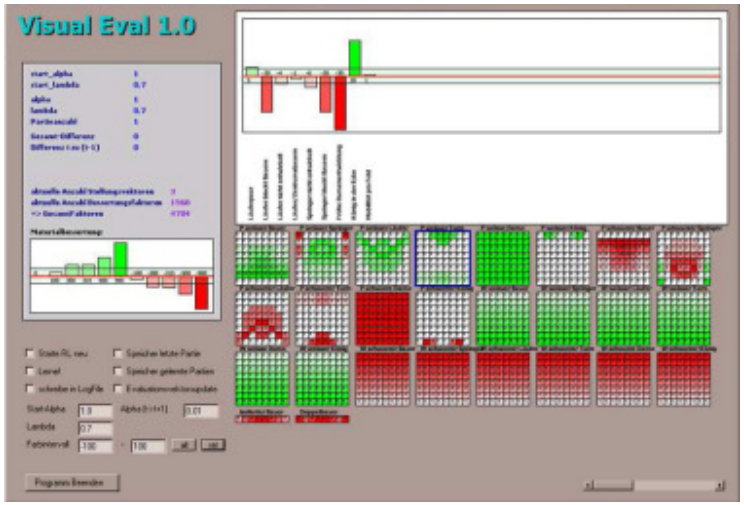




Reinforcement Learning – Temporale Differenz

Kalibrierung und Optimierung von Gewichten

$$\sum_{i=1}^{n} \alpha_i f_i(x)$$



In welchem gewichteten Verhältniss sollen die Heuristiken arbeiten?

$$f_1(S) = material(W) - material(S)$$

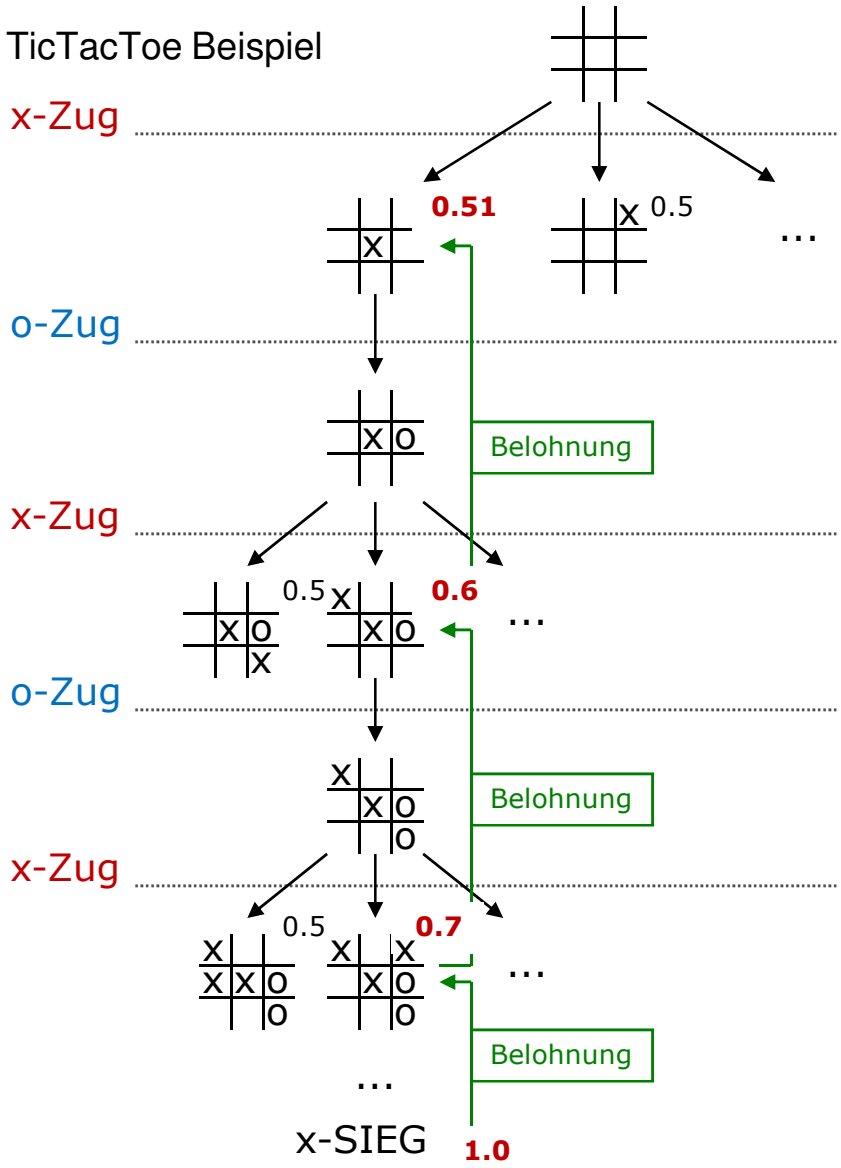
$$f_2(S) = mobilität(W) - mobilität(S)$$

Reinforcement Learning – Temporale Differenz

Einsatz von Temporaler Differenz in den Schachmotoren KnightCap und FUSc#, ein Leitsatz der uns noch von der letzen Vorlesung im Zusammenhang mit der Monte-Carlo-Suche (UCT) bekannt sein sollte:

> Agenten, die immer oder niemals weiterforschen, werden stets scheitern.

Reinforcement Learning – Temporale Differenz



1. Tabelle V(s) erzeugen mit (Zustand s → Bewertung)

Wahrscheinlichkeit zu gewinnen

Gewinnstrategie

1 Sieg, 0 Niederlage, 0.5 Remis

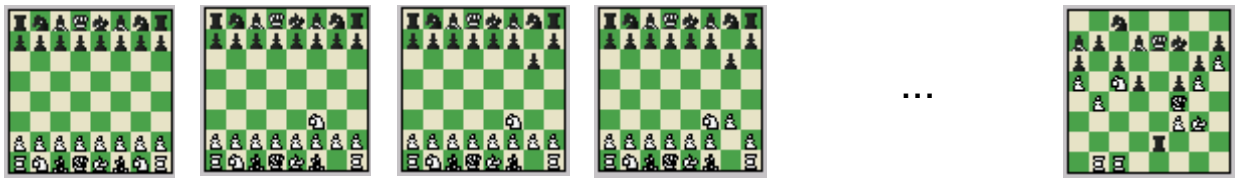
2. Es werden viele Partien gespielt

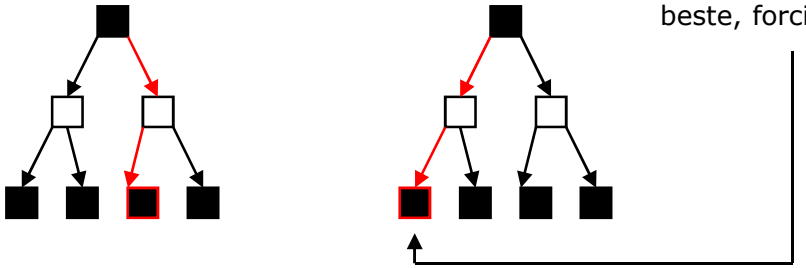
- meistens besten Zug wählen
- gelegentlich aber experimentieren! (zufällige Auswahl)
- während des Spiels wird Bewertungs-funktion verändert (Belohnung)
- vorheriger Zustand verschiebt sich etwas in Richtung des nachfolgenden

$$V(s_t) \rightarrow V(s_t) + alpha * [V(s_{t+1} - V(s_t))]$$

Reinforcement Learning – Temporale Differenz

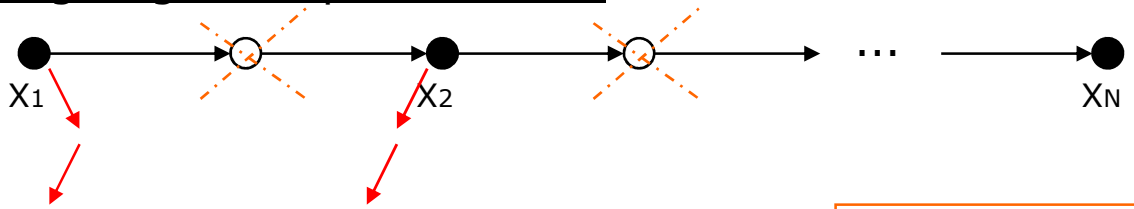
Partie: FUSc# – Internetgegner 0-1 (Evaluationsvektor w wird benutzt)





Alpha-Beta als Zugwahlalgorithmus, Hauptvariante wird gespeichert

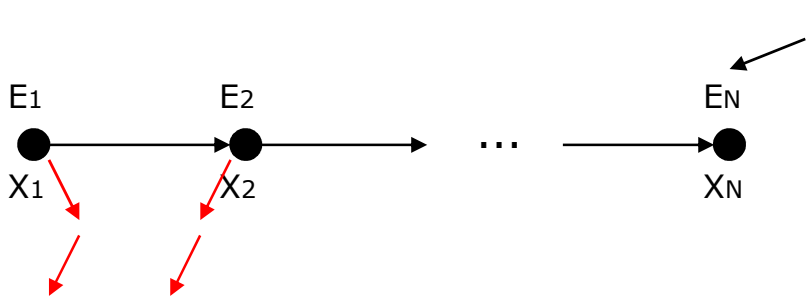
Stellungsfolge+HauptVarianten :



für Evaluation nur eigene Stellungen

Reinforcement Learning – Temporale Differenz

Stellungsfolge+HauptVarianten(HV) :



Evaluationswerte auch gespeichert

Reward {-400, 0, 400}

4 Bauern im Vorteil als Sieg
{-1,0,1}

Lernalgorithmus:

```
E[N] := -400
for i:=1 to N-1 do
    td[i] := E[i+1] - E[i]
    go in HV[i]
        compute pd[i]
    go out HV[i]
end i
vectorUpdate
```

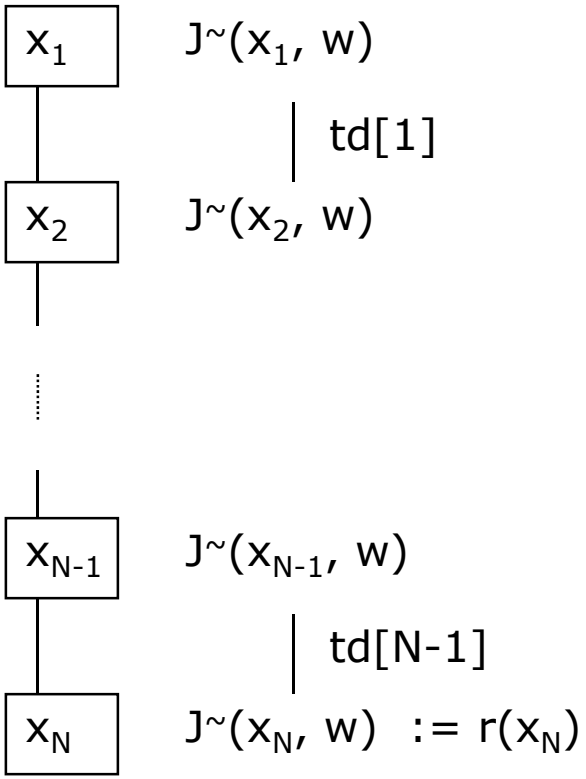
E[x] = Stellungsbewertung der Stellung x
mit Evaluationsvektor w bestehend aus n Stellungsfaktoren
lineare Funktion: w1f1(x) + w2f2(x) + w3f3(x) + ... + wnfn(x)

$$pd[i] = \begin{bmatrix} w1f1(xi) + 0*f2(xi) + 0*f3(xi) + \ldots + 0*fn(xi) \\ 0*f1(xi) + w2f2(xi) + 0*f3(xi) + \ldots + 0*fn(xi) \\ 0*f1(xi) + 0*f2(xi) + w3f3(xi) + \ldots + 0*fn(xi) \\ \ldots \\ 0*f1(xi) + 0*f2(xi) + 0*f3(xi) + \ldots + wnfn(xi) \end{bmatrix}$$

Evaluationsvektor w auf diesen Berechnungen basierend erneuern
und anschliessend normalisieren (auf Bauernwert 100)

Reinforcement Learning – Temporale Differenz

x_1 , ..., x_{N-1}, x_N eigene Stellungen unserer Partie.



$$E_{x_{t+1}|x_t}\left[J^*(x_{t+1}) - J^*(x_t)\right] = 0$$

„ich stehe besser, ergo erwarte ich einen Sieg...“

<u>Modifizierung :</u>

positive td[i]
- auf 0 setzen, es sei denn das Schachprogramm sagt den richtigen Zug des Gegners voraus

negative td[i]
- bleiben unverändert (eigene Fehler)

Reinforcement Learning – Temporale Differenz

Aktualisierung des Evaluationsvectors **w** nach TD(λ):

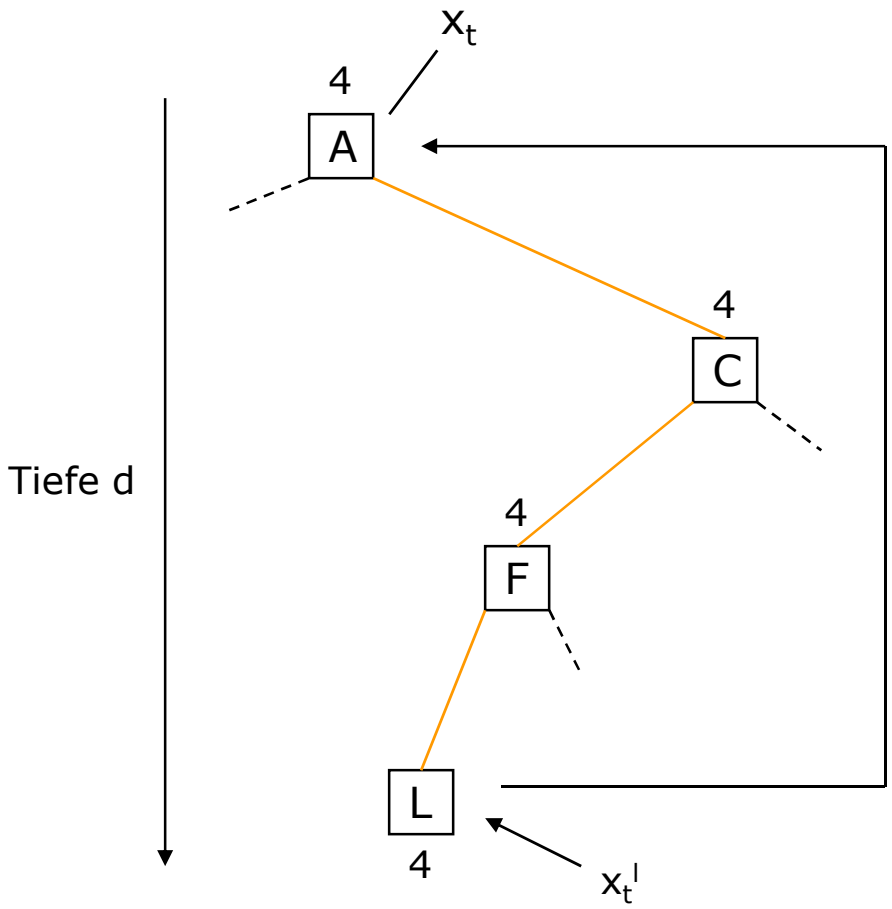
$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

Lernrate

Gradient

gewichtete Summe der Differenzen im Rest der Partie

Reinforcement Learning – Temporale Differenz



MinMax + TD(λ) = TD-Leaf(λ)

Reinforcement Learning – Temporale Differenz

Aktualisierung des Evaluationsvectors **w** nach TD-Leaf(λ):

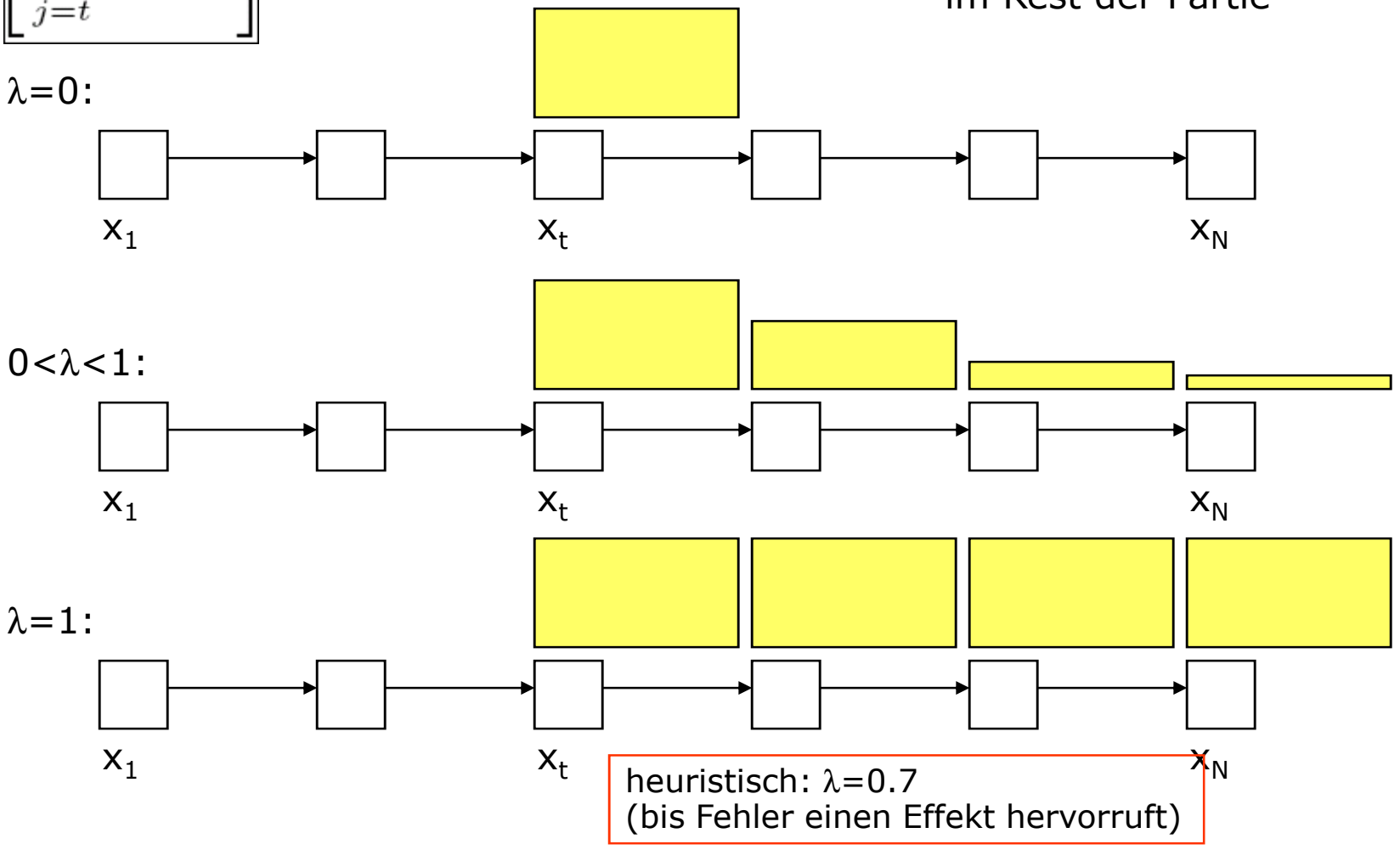
$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

Reinforcement Learning – Temporale Differenz

$$\left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t\right]$$

td[t]

Gewichtung der Differenzen im Rest der Partie



heuristisch: λ=0.7
(bis Fehler einen Effekt hervorruft)

Reinforcement Learning – Temporale Differenz

Aktualisierung des Evaluationsvectors w :

kurz: Δt

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

um wieviel

was muss ich ändern

$\Delta t > 0$:

$\Rightarrow$ Stellung x_t wurde vermutlich unterbewertet
$\Rightarrow$ w‘ = w + pos. Vielfaches des Gradienten
$\Rightarrow$ $J^{\sim}(x_t, w') > J^{\sim}(x_t, w)$

$\Delta t < 0$:

$\Rightarrow$ Stellung x_t wurde vermutlich überbewertet
$\Rightarrow$ w‘ = w + neg. Vielfaches des Gradienten
$\Rightarrow$ $J^{\sim}(x_t, w') < J^{\sim}(x_t, w)$

Reinforcement Learning – Temporale Differenz

TD(λ):

$$d_t := J'(x_t, \omega) - J'(x_t, \omega)$$

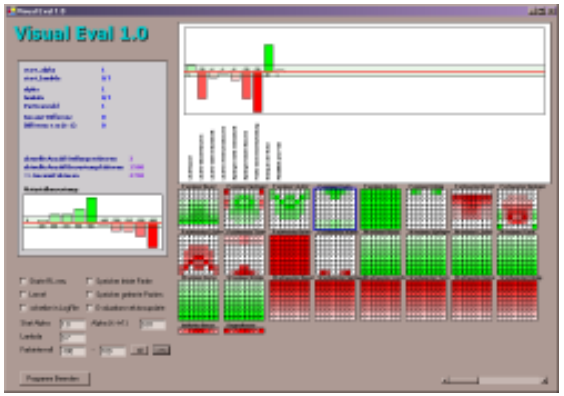
$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

TD-Leaf(λ):

$$d_t := J'_d(x_{t+1}, \omega) - J'_d(x_t, \omega)$$

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

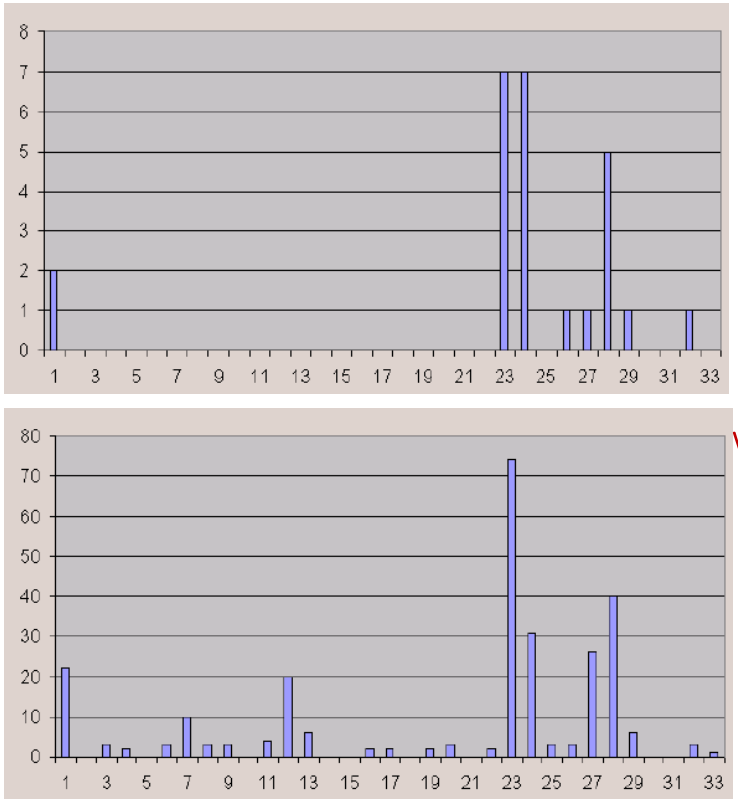
Reinforcement Learning – Temporale Differenz

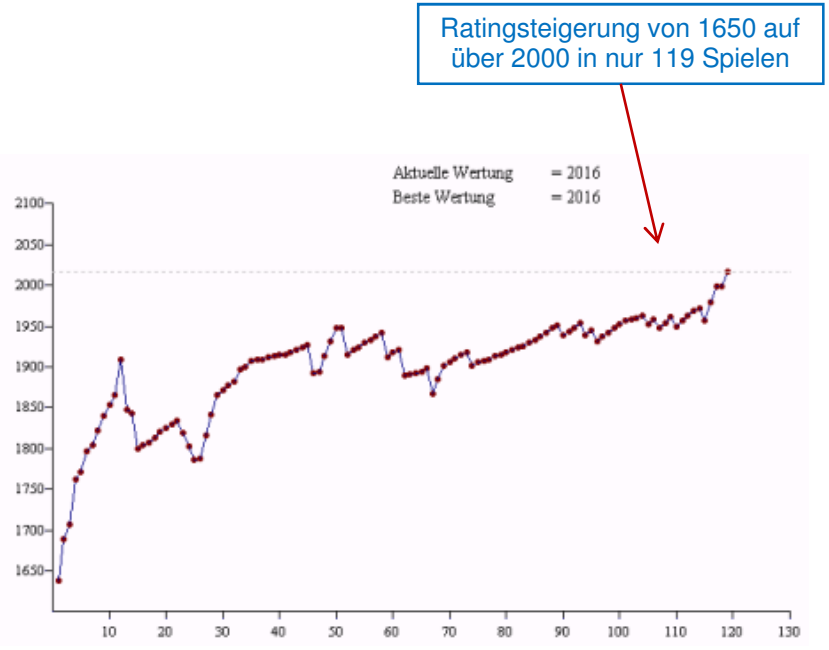


Einsatz bei FUSc#: TD-Leaf(λ)+Stellungsklassifikation [16]

$$\omega_k := \omega_k + \alpha_k \sum_{t=1}^{N-1} \nabla J'_d(x_t, \omega_k) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_j \right]$$

Stellungsklassifikation: 33 Stellungstypen mit insgesamt 56298 Koeffizienten

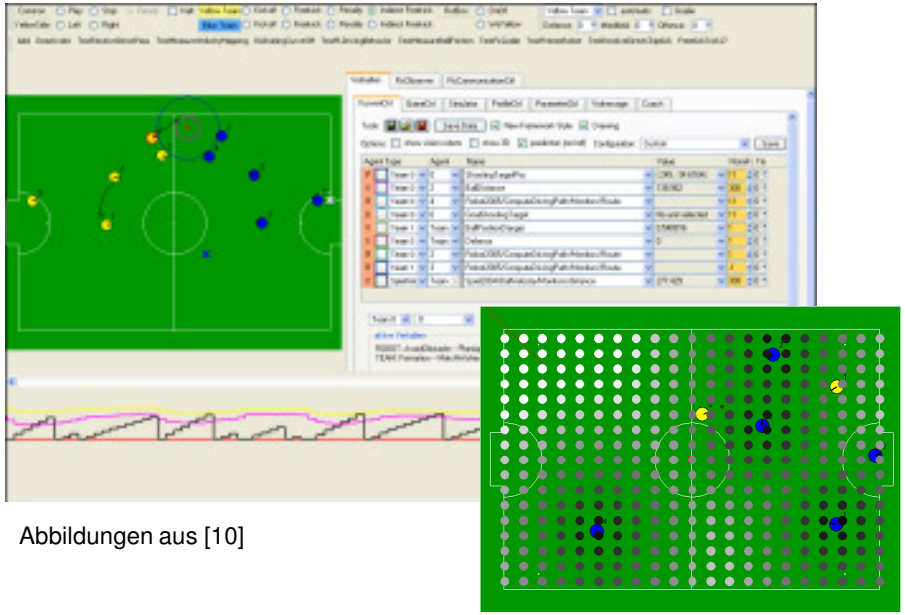




Einsatzgebiete

- Backgammon-Programm TD-Gammon setzt erfolgreich TD ein [17]
- Endspieldatenbank von Deep(er) Blue wurde teilweise durch TD optimiert
- Optimierung des Verhaltens bei den FU-Manoiden [10]
- und viele weitere

Kasparov – Deep(er) Blue 1997 (2.5-3.5)



Abbildungen aus [10]

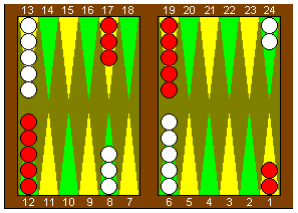


Abbildung aus [1]

Literatur- und Abbildungsquellen

[1] Michulke D., Schiffel S.: „*Matt bei Vier gewinnt*“, C't: Computer und Technik Magazin, pp.174, 1/2009

[2] Love N., Hinrichs T., Haley D., Schkufza E., Genesereth M.: „*General Game Playing: Game Description Language Specification*“, Stanford Logic Group Computer Science Department Stanford University, Technical Report LG-2006-01, 2008

[3] Quenault M., Cazenave T.: „*Extended General Gaming Model*“ Computer Games Workshop, pp. 195-204, 2007

[4] Genesereth M.: Vorlesungsfolien der Stanford-Universität, CS227B: General Game Playing, siehe dazu: *http://logic.stanford.edu/classes/cs227/index.html*

[5] Palamedes-IDE(Eclipse-Plugin): *http://palamedes-ide.sourceforge.net/*

[6] AAAI-GGP-Competitions: *http://www.aaai.org/Conferences/AAAI/2008/aaai08generalgame.php*

[7] Homepage von Frank Nießen: *http://www.frank-niessen.de/hnefatafl/index.html*

[8] Coulom R.: „*The Monte-Carlo Revolution in Go*“, Präsentation bei dem JFFoS'2008

[9] Möller F.M., Schneider M., Wegner M.: „*Centurio 2.0 - Monte Carlo Tree Search für General Game Playing*”, Dokumentation der Engine Centurio, Universität Potsdam, 2008

[10] Hecht B.: „*Neuronaler Coach - Lernmechanismus für Roboter mittels menschlicher Beispieldiagramme*“, Diplomarbeit, Freie Universität Berlin, 2008

[11] Matthews J.: „*Applying Kohonen Networks*“, Online-Artikel http://www.generation5.org/content/2004/kohonenApplications.asp

Literatur- und Abbildungsquellen

[12] Nistér D., Stewénius H.: „Scalable Recognition with a Vocabulary Tree“, Conference on Computer Vision and Pattern Recognition (CVPR 2006), New York, 2006

[13] McCulloch W., Pitts W.: “*A logical calculus of the ideas immanent in nervous activity”,* Bulletin of Mathematical Biophysics, 5:115-133, 1943

[14] Tapia E.: “*Understanding Mathematics: A System for the Recognition of On-Line Handwritten Mathematical Expressions*”, Dissertation, Freie Universität Berlin, 2005

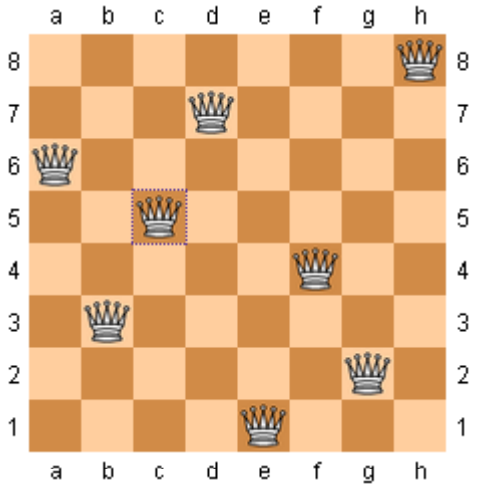
[15] Rojas R: “*Neural Networks - A Systematic Introduction*”, Springer-Verlag, 1996

[16] Block M., Bader M., Tapia E., Ramírez M., Gunnarsson K., Cuevas E., Zaldivar D., Rojas R.: „Using Reinforcement Learning in Chess Engines”, CONCIBE SCIENCE 2008, In Journal Research in Computing Science: Special Issue in Electronics and Biomedical Engineering, Computer Science and Informatics, ISSN:1870-4069, Vol.35, pp.31-40, Guadalajara/Mexico, 2008

[17] Tesauro G.: “*Temporal Difference Learning and TD-Gammon*”, in *Communications of the ACM,* Vol. 38, No. 3., 1995

[18] Brooks A., Gao L.: „*Face Recognition: Eigenface and Fisherface Performance Across Pose*”, In Proceedings of the Fourth conference on IASTED International Conference: Signal Processing, Pattern Recognition, and Applications, pp.321-327, 2007

Übungsaufgaben



Aufgabe 1 – Boltzmannmaschine (10 Punkte)
Implementieren Sie eine Boltzmannmaschine in Java, die das 8-Dame-Problem löst. Visualisieren Sie dabei die aktuelle Brettkonstallation und zeigen Sie die aktuellen Parameter.

Experimentieren Sie mit der Fallrate der Temperatur und dokumentieren Sie dieses entsprechend

Abgabe: 03.07.2009 (zwei Wochen)

Zusatzaufgabe – Temporale Differenz (10 Punkte + KI-Skills)
Sie haben bereits auf einem der vorhergehenden Übungszettel eine KI für ein Spiel entworfen. Überlegen Sie sich ein Konzept, wie Sie TD(λ) oder TD-Leaf(λ) zur Optimierung der Gewichte verwenden können. Nach der Erstellung eines Konzeptes soll dieses implementiert und ausführlich getestet werden.

Alternativ können Sie auch Ihre ultimative GDL-KI mit Temporaler Differenz optimieren. Denken Sie aber auch dabei an ein schlüssiges Konzept.

Abgabe: 03.07.2009 (zwei Wochen)