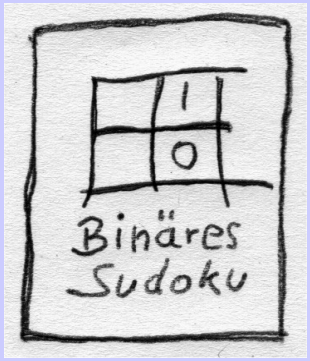
hochschule mannheim

Jörn Fischer
j.fischer@hs-mannheim.de

Willkommen zur Vorlesung

Maschinelles Lernen



Vorstellung

Zu meiner Person...

Überblick

Inhalt

- Organisatorisches
- Einführung
- Optimierung
- Evolutionäre Algorithmen
- Reinforcement Learning
- Vektorquantisierung
- Neuronale Netze

Organisatorisches

Fragen…

- Fragen können gerne auch per Email an mich gerichtet werden oder aber bei Diskussionsbedarf einfach per Mail einen Termin vereinbaren...
- Rückkopplung ist ausdrücklich erwünscht !!!!!

Einführung

Ziel der Vorlesung

- ...ist, dass ihr maximal dabei lernt
- Dass ihr die Algorithmen versteht
- Dass ihr sie selbst programmieren könnt
- Dass Ihr sie selbst erklären könnt

Einführung

Mein Ansatz

- Ich werde Euch ausgesuchte Algorithmen und deren Grenzen erklären
- Ihr werdet die Algorithmen in elementarer Form programmieren...
  auch zu Hause!
- Dazu könnt ihr Euch zu Zweier- oder höchstens Dreiergruppen zusammenschließen
- Die Algorithmen werden testiert und gelten als

  Zulassung für die Prüfung
- Neuer RGS: mindestens 5 von 6 Programmen müssen verstanden sein und elementar funktionieren!
- Alter RGS: alle 6 Programme müssen verstanden sein und elementar funktionieren!

Organisatorisches

Fragen zu organisatorischen Dingen?

Organisatorisches

Frage an Sie...

Haben Sie schon mal was von

- Optimierungsalgorithmen
- Evolutionäre Algorithmen
- Reinforcement Learning
- Neuronale Netze

gehört?

Einführung

Einführung

Literatur

- I. Goodfellow, Y.Bengio, A. Courville : Deep Learning, MIT Press, ISBN: 978-0262035613, 2016
- Russel, Stuard; Norvig, Peter: Künstliche Intelligenz. Prentice Hall, New Jersey, 1995
- Mitchell, Tom: Machine Learning. McGraw-Hill, 1997
- Zell, Andreas: Simulation Neuronaler Netze. Oldenbourg Verlag, München, 1997
- Sutton, Richard; Barto, Andrew G.: Reinforcement Learning. MIT Press, 1998

Einführung

Gedankenexperiment

- Wenn man das menschliche Gehirn in seiner technischen Leistungsfähigkeit mit einem Computer vergleicht…

	technisch (Computer)	biologisch(Gehirn)
Speicher	1 Tbyte = 10^{12} Byte (Festplatte)	10^{13} synapsen ~ 10^{13} Byte
Multiplikationen	1 Tflops = 10^{12} multipl. (Grafikkarte)	10^{13} multipl. (stark vereinfachtes Modell)

Einführung

Frage?

- Was hält uns also davon ab eine Maschine zu bauen, die in allen Belangen die Leistungsfähigkeit eines Menschen hat?

Antwort:

?

Einführung

Frage?

- Was hält uns also davon ab eine Maschine zu bauen, die in allen Belangen die Leistungsfähigkeit eines Menschen hat?

Antwort:

- Die Möglichkeit nicht nur Lösungen zu finden sondern auch neue Lösungsmethoden erlernen oder entwickeln zu können
- Problemlösungen auf andere Probleme übertragen zu können
- Die Angepasstheit an unsere Umwelt durch eine Jahrtausende währende Evolution
- Die Fähigkeit Lernmethoden zu erlernen (Methalernen)
- …

Einführung

Was gibt es schon? Wie weit sind wir?

- Das beste Programm für Gensequenzierung ist ein Neuronales Netz
- Texte, Handschriften, Sprache werden bereits vielfach von Automaten erkannt
- Das beste Backgammon Programm basiert auf einem Reinforcement Learning Algorithmus. Das Programm enthält im Wesentlichen die Spielregeln und hat so oft gegen sich selbst gespielt, dass Backgammon-Großmeister sich den ein oder anderen Zug abschauen.
- Es gibt Chatbots, die sich mit Menschen unterhalten und ständig hinzulernen

Einführung

Was gibt es schon? Wie weit sind wir?

- Es gibt Antivirenprogramme, welche Neuronale Netze zur Erkennung mutierender Schädlinge einsetzen
- Roboter können Gegenstände durch ansehen und nachfragen einordnen und wiedererkennen
- Jürgen Schmidhubers „Gödel Machines“ lernen nicht nur ein Ziel zu erreichen, sondern lernen auch seine Lernmechanismen optimal zu verbessern
- Durch Evolution werden Roboter-“Gehirne“ entwickelt, die optimal auf Ihre Umwelt angepasst sind
- Dabei kann selbst die Vererbung durch weitergeben der Gene auf realen Robotern stattfinden
- ……

Einführung

Was bedeutet ein System lernt

Die Änderung von Informationen im System mit dem Ziel effizienter zu sein.

- besser generalisieren zu können
- besser agieren zu können
- besser vorhersagen zu können

Einführung

Was ist ein Agent

Als Agent kann alles angesehen werden, was seine Umwelt durch Sensoren (sensors) wahrnimmt und durch Effektoren (effectors) beeinflußt.

Zitat: Russel und Norvig

Einführung

Was für Problemkategorien kann man unterscheiden?

vollständig wahrnehmbar <-> teilweise wahrnehmbar

deterministisch <-> stochastisch

(Episoden sind unabhängig voneinander)

episodisch <-> sequentiell

statisch <-> dynamisch

diskret <-> kontinuierlich

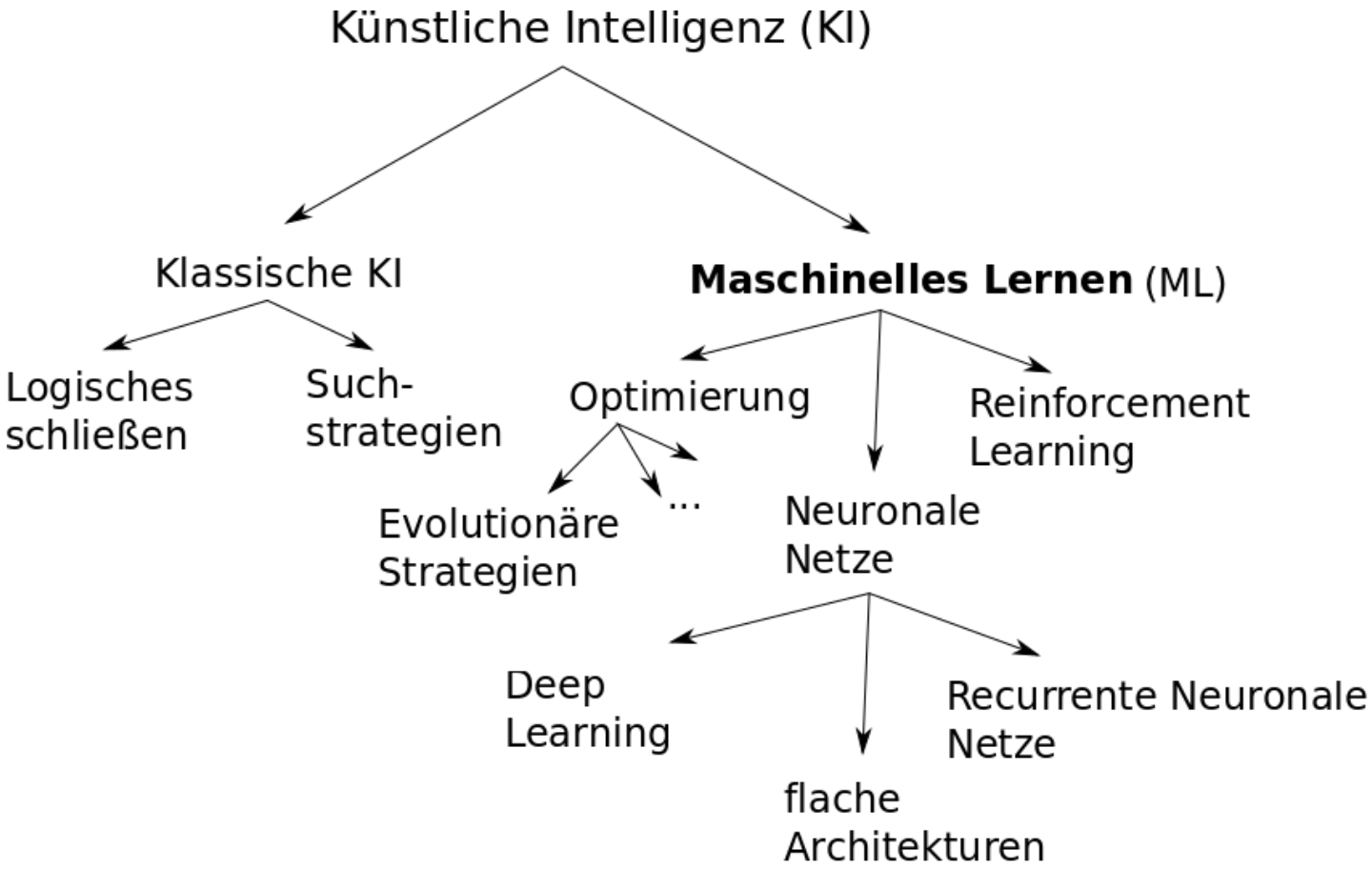
Einführung

Was für Methoden kann man unterscheiden?

- Überwachtes lernen (supervised learning)
  eine Funktion wird anhand Ein-Ausgabe Paaren gelernt
- Unüberwachtes Lernen (unsupervised learning)
  der Algorithmus findet anhand von Eingabedaten ein Modell, welches die Eingaben beschreibt und Vorhersagen ermöglicht
- Bekräftigungs Lernen (reinforcement learning)
  der Algorithmus belohnt oder bestraft eine Taktik wie in potentiell auftretenden Situationen zu handeln ist

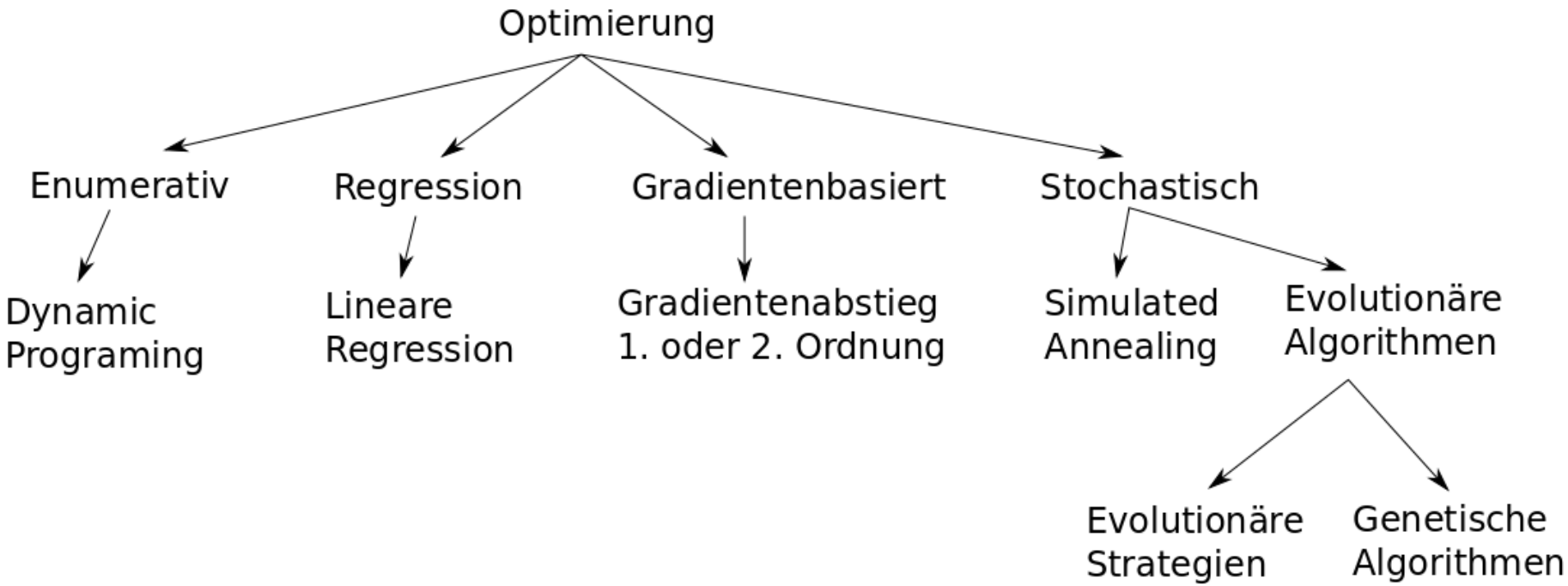
Einführung

KI, ML, Deep Learning… wie hängt das zusammen?



Optimierung

Optimierung



Optimierung

Hill Climbing

Hill-Climbing

- Bergsteigeralgorithmus (englisch hill climbing) ist ein einfaches, heuristisches Optimierungsverfahren.
- Von einer gegebenen Hypothese (z.B. Satz von Startparameterns) aus wird zu einer Hypothese aus der Nachbarschaft der aktuellen Hypothese gegangen. Ist die Fitness der neuen Hypothese höher, als die der vorherigen Hypothese, so wird von dort aus weitergesucht. Ist die Fitness kleiner, so wird der Schritt rückgängig gemacht.
- Da das Verfahren sehr leicht in lokalen Optima stecken bleibt, wird das Verfahren meist mit zufällig ausgewählten Startpunkten wiederholt.

Optimierung

Hill-Climbing (Pseudocode)

```
threshold is stopping criterion
hypothesis is startingPoint in the hyotheses space
lastFitness = fitness(hypothesis)
do
  copy(hypothesis) // copy hypothesis into saveState
  moveOneStepAtRandom(hypothesis)
  if fitness(hypothesis)>lastFitness then
     lastFitness = fitness(hypothesis)
  else
     undoMove(hyothesis) // copy saveState back into hypothesis
  endif
While lastFitness < threshold
```

Optimierung

Aufgabe 1)

- Schreibt einen Hill-Climber, der eine gute Rundreise zwischen 100 Städten mit zufälligen Distanzen findet.

Optimierung

Simulated Annealing

Optimierung

Simulated Annealing

- Simulated Annealing ist ähnlich dem Hill-Climbing
- Motiviert durch den langsamen Abkühlungsprozess bei Metallen, bei dem die Atome ausreichend Zeit haben sich zu ordnen und stabile Kristalle zu bilden. Es resultiert ein energiearmer Zustand nahe am Optimum.
- Bergaufschritte werden immer vollzogen
- Bergabschritte nur mit einer Wahrscheinlichkeit w(x,y), die vom Fitnessunterschied zwischen Startpunt x und Hypothese y und von einer stets sinkenden Temperatur T abhängt:

$$w(x, y) = e^{\frac{(f(y) - f(x))}{T_t}}$$

- Ist die neue Fitness F(y) geringer als die vorherige Fitness F(x), so ergibt sich eine Wahrscheinlichkeit $w(x, y) \in [0..1]$

Optimierung

Simulated Annealing (Pseudocode)

```
hypo is startingPoint in the hyotheses space
lastFitness = fitness(hypothesis);
temp = startTemperature;
do
  copy(hypo) // copy hypothesis into saveState
  moveOneStepAtRandom(hypo)
  if ((fitness(hypo)>lastFitness) then
     lastFitness = fitness(hypo)
  else if (randomValue[0..1]) < exp((fitness(hypo)-lastFitness)/temp)) then
     lastFitness = fitness(hypo)
  else
     undoMove(hypo) // copy saveState back into hypothesis
  endif
  temp = temp - epsilon  // epsilon is small!
while(temperature > epsilon)
```

Optimierung

Aufgabe 2)

- Ändert den Hill-Climber von Aufgabe 1 in einen Simulated Annealing Algorithmus.

Optimierung

Raum und Dimensionen

Optimierung

Nachbarzustände (Travelling Salesman)

- Wie viele Nachbarzustände k gibt es, wenn man im Traveling Salesman Problem Städte vertauscht um eine n Städte Rundreise zu optimieren?

$$k = n*(n-1)$$

- Dabei sucht man sich zunächst eine der n Städte aus und vertauscht sie mit einer der übriggebliebenen (n-1) Städte.
- → Nachbarn können i.a. alle geprüft werden

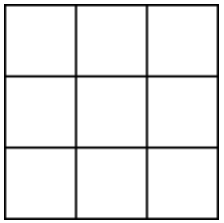
Optimierung

Nachbarzustände (Hill Climber)

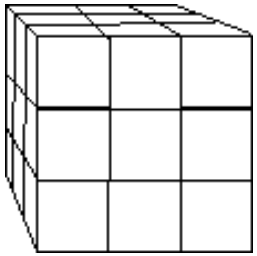
- Wie viele Nachbarzustände k hat ein HillClimber, wenn der Parameterraum n Dimensionen hat?

$k=3^n -1$

n=1 k=2



n=2 k=8



n=3 k=26

n	k
1	2
2	8
4	80
8	6560
16	43046720
32	$18*10^{14}$
64	$34*10^{29}$

→ Nachbarn können i.a. nicht alle geprüft werden

Evolutionäre Algorithmen

Evolutionäre Algorithmen

In der Biologie

- Mutation: Es kommen bei der Teilung einer Zelle kleine Fehler bei der Replikation des genetischen Codes vor
- Rekombination: Es werden neue Gene durch Kombination von Gensequenzen aus der Mutter- und andere aus der Vaterzelle gebildet
- Selektion: Besser angepasste Individuen setzen sich gegenüber anderen Individuen durch.

→keine Suche nach

Globalem Optimum!

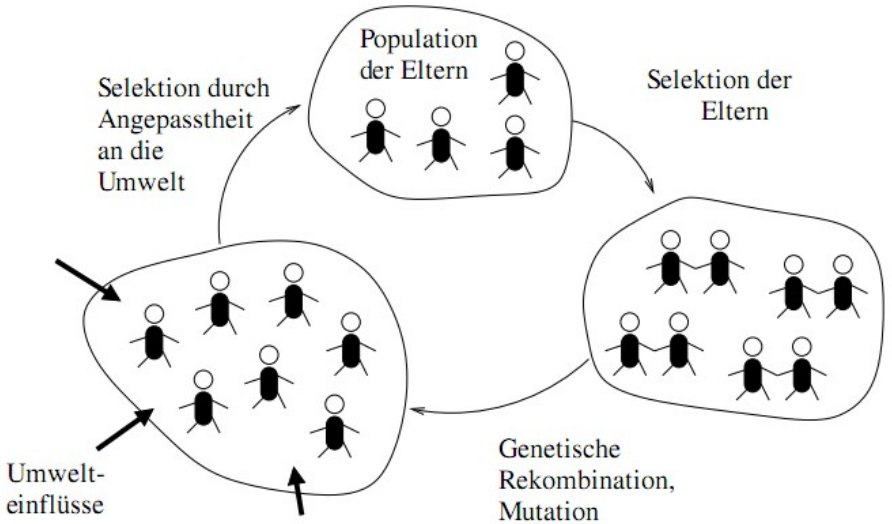
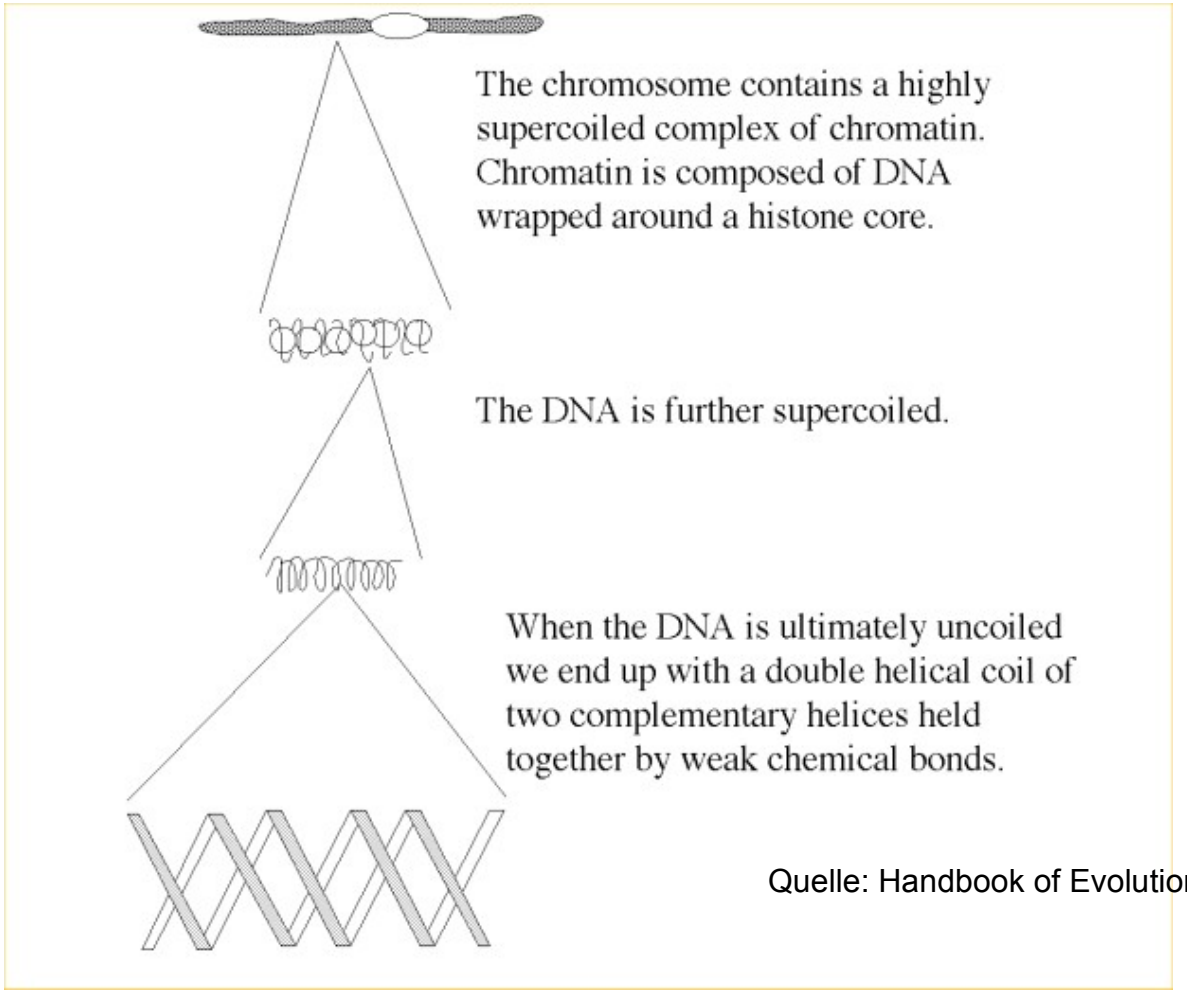


Abbildung 1: Evolution in der Biologie (vereinfachte Darstellung)

Evolutionäre Algorithmen

In der Biologie



Quelle: Handbook of Evolutionary Computation

Evolutionäre Algorithmen

In der Biologie

Die DNA besteht aus Genen, die jeweils für sich eine abgeschlossene Kodierung enthalten

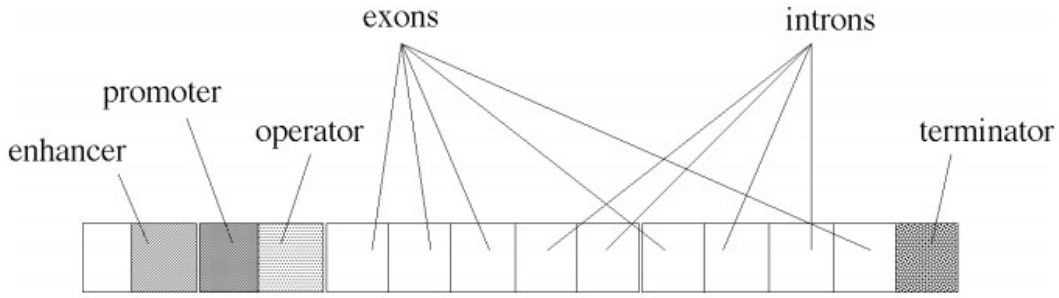


Figure A2.2.11. A simplified diagram of a structural gene.

Quelle Handbook of Evolutionary Computation

Der Mensch hat 24 Chromosomen mit insgesamt etwa $2.5*10^5$ Genen à $1.2*10^4$ Bausteinen bei 4 unterschiedlichen Basen (CG TA)

→ $3*10^9$ Bausteine à 2 Bit = $6*10^9$ Bit = 750 Mbyte

Evolutionäre Algorithmen

In der Biologie

- Beim Teilen der DNA passieren zufällig Fehler, die man Mutation nennt. Dies geschieht durch Austausch, Einfügen oder Verlust von Basen.
- Die Wahrscheinlichkeit einer Mutation ist 10^{-10}
- Die Wahrscheinlichkeit, dass eine Zelle einen Fehler beim Teilen hat ist bei $3*10^{9}$ Basenpaaren etwa $3*10^{-1}$

Evolutionäre Algorithmen

In der Biologie

- Mittels Enzymen wird die DNA (deoxyribonucleic acid) in Messanger RNA (Ribonucleic acid) aufgespalten
- Diese wird dann in Proteinmoleküle übersetzt, die verschiedene Aufgaben einer Zelle übernehmen
- Durch Teilung mittels Enzymen kann sich die DNA auch selbst replizieren. Dabei werden die aufgespaltenen Stränge durch Komplementärstränge ergänzt
- Biologische Gene enthalten nicht nur zu lesende Informationen, sondern auch die Informationen, wie gelesen wird und was gelesen werden soll.

Evolutionäre Algorithmen

Baldwin Effekt

- Lamarck nahm an, dass erlernte Fähigkeiten die Gene so beeinflussen, dass die Fähigkeiten an die Nachkommen vererbt werden können

  → Diese Hypothese ist aber FALSCH!!!

- Baldwin zeigte, dass Evolutionsdruck dazu führen kann die Lernfähigkeit zu verbessern, da z.B. Tiere, die die Lernfähigkeit nicht haben auch geringere Überlebenschancen haben.

Evolutionäre Algorithmen

Genotyp und Phänotyp

- Der **Genotyp** ist die Erbinformation eines Individuums, also seine genetische Ausstattung.
- Der **Phänotyp** oder das Erscheinungsbild ist die Summe aller äußerlich feststellbaren Merkmale eines Individuums. Dazu zählen auch die erworbenen Fähigkeiten, die nicht weitervererbt werden.

Evolutionäre Algorithmen

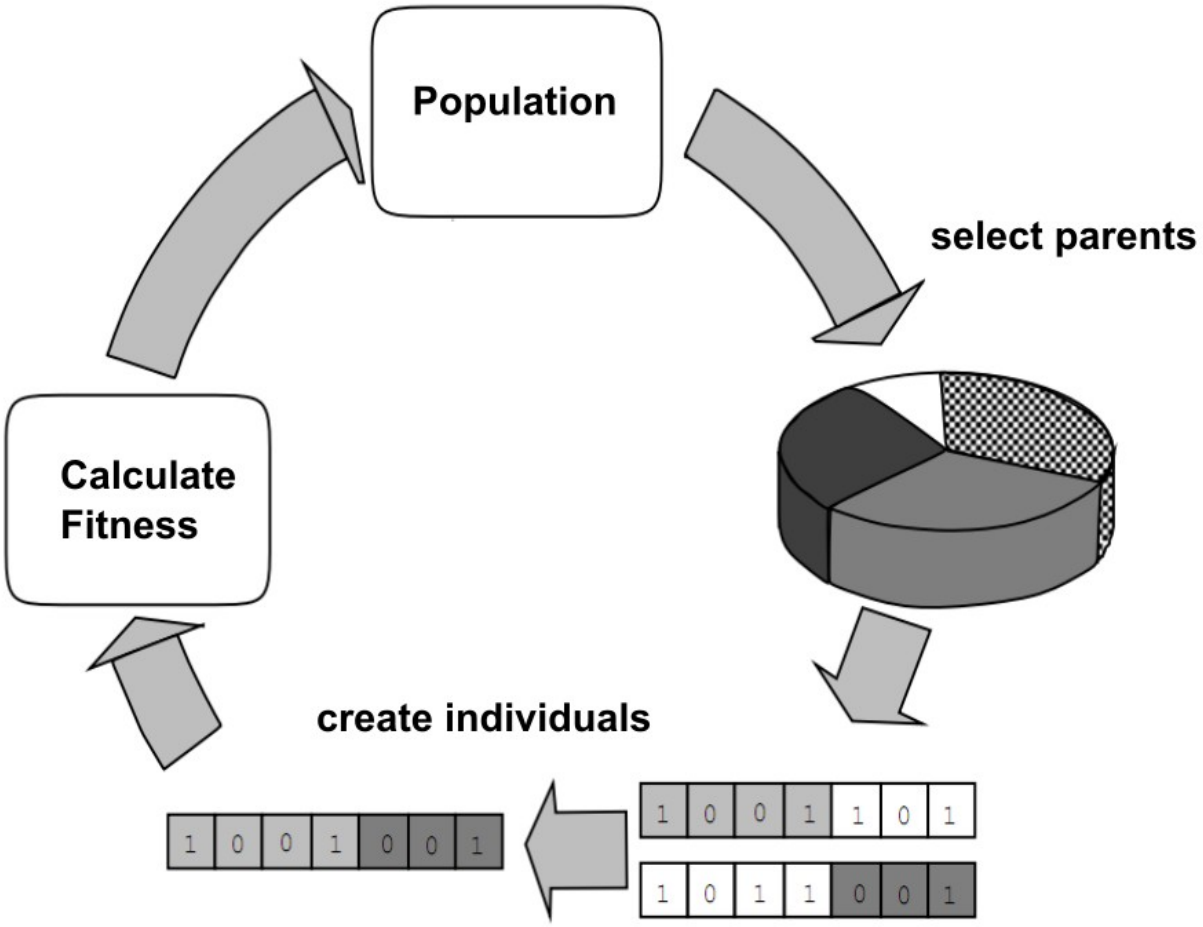
Was gibt es für evolutionäre Algorithmen

- Evolutionäre Algorithmen
  - Genetische Algorithmen
  - Memetische Algorithmen (gen. Algorithmus kombiniert mit lokaler Suche z.B. gradientenbasiert)
  - Strukturevolution
  - Evolutionäre Programmierung
    - Genetische Programmierung
    - etc.

Evolutionäre Algorithmen

Genetische Algorithmen

Evolutionäre Algorithmen



Evolutionäre Algorithmen

In der Informatik

- Mutation: In der Gensequenz werden mit gegebener Wahrscheinlichkeit zufällig Bits „gekippt“
- Rekombination (Crossover): Aus den Genen zweier effizienter Individuen werden durch Wählen eines oder mehrerer Crossover Punkte zwei neue Gene (Individuen) produziert. Dabei wird der erste Teil des einen Gens (bis zum Crossoverpunkt) und der zweite Teil des anderen Gens zu einem neuen Gen zusammengefügt. Zusätzlich wird der zweite Teil des einen Gens und der erste Teil des anderen Gens zu einem zweiten neuen Gen zusammenfügen.
- Selektion: Bei der Selektion werden die Individuen bewertet und nur die effizientesten werden in die darauffolgende Generation übernommen

Evolutionäre Algorithmen

p sei die Anzahl der Hypothesen(=Individuen) in der Population P
r sei der Anteil, der in jedem Schritt durch Crossover ersetzt wird
m sei Mutationsrate
Initialisiere die Population (erzeuge Zufallshypothesen)
Errechne die Fitness für alle Hypothesen
while maxFitness<fitnessThreshold
   Selektion: wähle (1-r)*p Individuen aus P mit der Wahrscheinlichkeit:

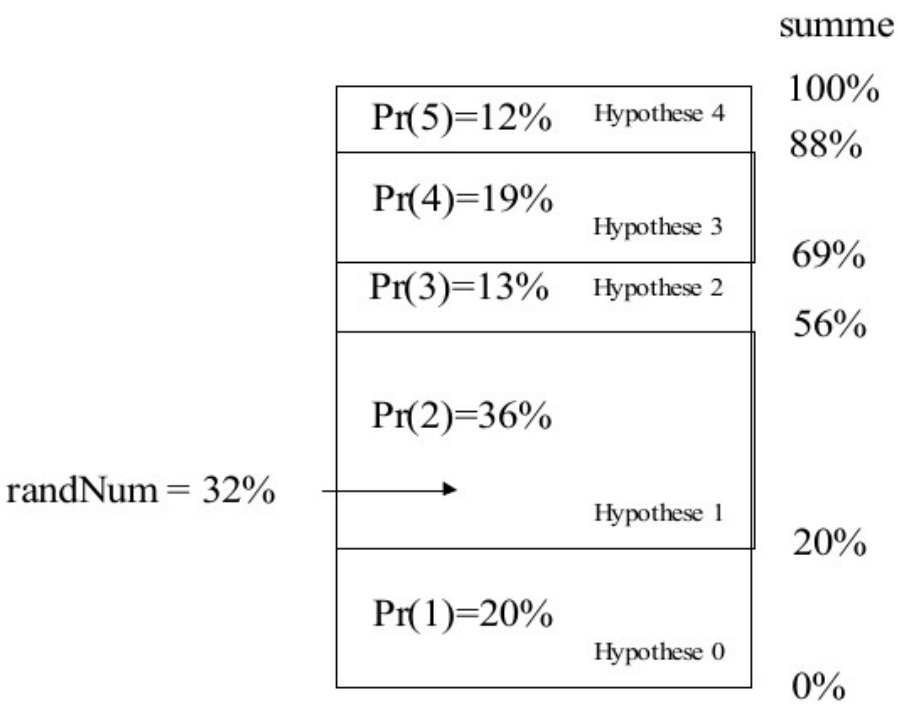
$$Pr(h_i) = \frac{fitness(h_i)}{\sum_{j=1}^{p} fitness(h_j)}$$

und füge sie der neuen Generation P_s hinzu

   Crossover: wähle r*p/2 Hypothesenpaare von P mit der oben stehenden Wahrcheinlichkeit $Pr(h_i)$. Erzeuge unter Anwendung des Crossover Operators 2 Nachkommen für jedes Paar und füge sie P_s hinzu.
   Mutation: Wähle m Kandidaten aus Ps mit gleichverteilter Wahrscheinlichkeit und kippe jeweils ein zufällig gewähltes Bit
   update: P ← P_s
   errechne die Fitness für alle Hypothesen
endwhile

Evolutionäre Algorithmen

```
int selectHypothesis()
{
 randNum sei Zufallszahl [0..1]
 summe = 0
 index = Zufallszahl [0..p]
 do
   index = index + 1
   index = index modulo p
   summe := summe + Pr(index)
 while summe<randNum
 return index
}
```



Da die Wahrscheinlichkeiten im Rechner nur begrenzt genau dargestellt werden, ist die Summe der Wahrcheinlichkeiten i.a. etwas kleiner als eins. Um zu verhindern, dass der Index dann zu groß wird, weil z.B. randNum sehr nahe bei eins ist, sollte der index modulo p berechnet werden. Um dann den Fehler der Wahrscheinlichkeiten auf alle Hypothesen gleichmäßig zu verteilen startet man mit einem zufälligen index.

Evolutionäre Algorithmen

Problem bei der Mutation

- Problem: Ein genetischer Algorithmus findet einen Bitstring „011111“ = 31 der noch nicht optimal ist, denn „100000“ = 32 währe der optimale Bitstring. Dann müsste der genetische Algorithmus alle Bits invertieren um dahin zu gelangen. Leider führt jede Invertierung eines Bits zunächst zu einem schlechteren Ergebnis…
- Lösung: Nach Möglichkeit kodiert man die zu optimierenden Parameter im Graycode. Damit ist garantiert, dass Nachbarzustände immer durch Änderung eines der Bits erreicht werden können!

Evolutionäre Algorithmen

Graycode Kodierung

Sei b={Bn..B1} eine n stellige Binärzahl und g={Gn..G1} der dazu passende Graycode, dann kann g aus b mit Hilfe folgender Vorschrift gewonnen werden:

$G_n = B_n$ (Das oberste Bit ist gleich)

$G_i = XOR(B_{i+1},B_i)$ (für alle $1 < i < n-1$)

genauso gilt:

$B_n = G_n$ (Das oberste Bit ist gleich)

$B_i = XOR(B_{i+1},G_i)$ (für alle $1 < i < n-1$)

Evolutionäre Algorithmen

Genetische Algorithmen

Vorteile:

- GA‘s sind nicht so anfällig dafür in lokalen Minima stecken zu bleiben (durch Operatoren wie Crossover können die Nachkommen z.B. weit auseinander liegen)
- Der Algorithmus ist einer der universellsten und lässt sich auch auf Probleme mit nicht stetiger Fitnessfunktion (im Prinzip sogar auf Probleme zeitlich veränderlicher Fitnessfunktion) anwenden

Nachteile

- Individuen mit hoher Fitness übernehmen schnell durch Kopien eine der folgenden Generationen, so dass weniger Diversität auftritt (crowding).
- Man findet nicht so konsequent und schnell die Minima wie beim Gradientenabstieg

Evolutionäre Algorithmen

Genetische Algorithmen

Was kann man tun um die Nachteile des Crowding zu überwinden?

- Man kann beispielsweise die Fitness verringern, wenn es viele ähnlich Individuen gibt : „fitness sharing“
- Man kann die Rekombination auf ähnliche Individuen beschränken, so erhält man Gruppen von „subspecies“
- Man kann die Selektionsstrategie ändern:

  z.B. durch Tournament Selection oder Rank Selection

Evolutionäre Algorithmen

Genetische Algorithmen

- Tournament selection (Goldberg, Deb 1991)

  Aus einer Gruppe von s zufällig aus p ausgewählten Individuen wird, im einfachsten Fall, immer das Beste selektiert. Wählt man s=2, so ist der Evolutionsdruck niedrig. Je größer s, desto größer der Evolutionsdruck. Bei s=p und einer Vermeidung, dass Individuen mehrfach gewählt werden, erhält man eine nach Fitness sortierte Liste aller Individuen.

  → führt zu mehr Diversität in der Population

- Rank selection

  Die Hypothesen werden nach Fitness sortiert und die Wahrscheinlichkeit, dass eine Hypothese ausgewählt wird, hängt vom Rang statt von der Fitness ab.

Evolutionäre Algorithmen

Aufgabe 3)

- Der Weihnachtsmann soll einen Sack mit Geschenken voll packen, der 100 Liter Fassungsvermögen hat. Nun hat er eine Auswahl von 100 Geschenken, die er einpacken kann (1) oder nicht einpackt (0). Der Weihnachtsmann ist geübt und stapelt die Geschenke immer lückenlos. Jedes der Geschenke g verbraucht im Sack ein zuvor zufällig festgelegtes Volumen v[g] zwischen 0,1 und 2 Liter. Um dem Weihnachtsmann zu helfen den Sack möglichst gut mit den Geschenken zu füllen möchte er, von euch wissen, welche Geschenke er einpacken soll und welche nicht. Schreibt einen genetischen Algorithmus mit einer Population von 1000 Binären Genen, die 100 Bits umfassen bg[1000][100] und beinhalten, ob ein Geschenk eingepackt wird (1) oder nicht (0). Legt zuvor fest, welche Geschenke welche Volumina haben. Wie nahe seid ihr nach 50 Generationen an den 100 Litern.

$$f[i] = e^{-c\left(100 - \sum_g bg[i][g]\,v[g]\right)^2}$$

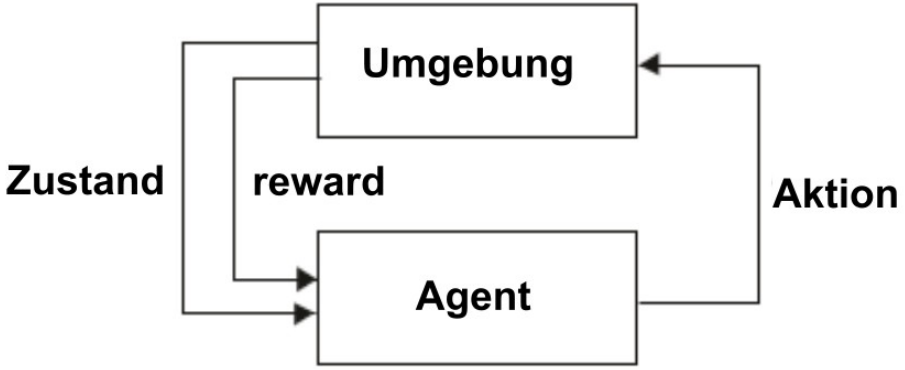
f(i) ist die Fitness von Individuum i
v[g] ist das Volumen des Geschenkes g
c z.B. c=0.001 ist Konstante, die die Breite der Gaußglocke bestimmt. Bitte etwas experimentieren

Reinforcement Learning

Reinforcement Learning

Was ist Reinforcement Learning

Reinforcement Learning (bzw. bestärkendes Lernen) ist ein Lernen, bei dem ein Agent lediglich aufgrund von Belohnung (positivem „reward“) und Bestrafung (negativem „reward“) lernt seinen Nutzen zu optimieren



Zustand s kann sein:
- Sensorwerte der Umwelt
- eigene errechnete Position
- Innere Zustände wie Speicherinhalte

Aktion a kann sein:
- Manipulator bewegen
- Abwarten
- Speicher beschreiben

Reward r ist:
- Belohnung, die positiv oder negativ sein kann

Reinforcement Learning

Bewertungsfunktion

Value Funktion $V(s)$ oder $Q(s,a)$ (Bewertungsfunktion / Value-Funktion):

- Die Bewertungsfunktion spiegelt die zu erwartende Belohnung (reward) wieder. Ist sie gelernt, so kann der Agent in jedem Zustand die Aktion aussuchen, die die größte zu erwartende Belohnung widerspiegelt.
- Sei Q* die optimale Bewertungsfunktion, dann gilt:

$$Q_t^* = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Dabei ist $\gamma = [0..1]$ Discount Faktor, der dafür sorgt, dass weit in der Zukunft liegende mögliche Rewards geringer gewichtet werden.

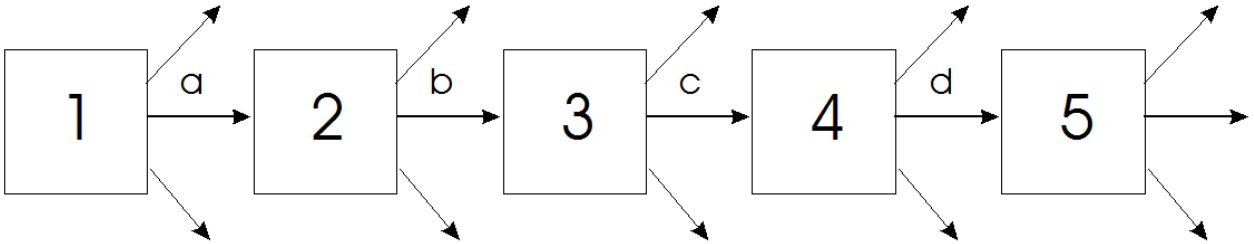
(t=0 ist der aktuelle Zeitpunkt, t>0 ist die Anzahl der Schritte in die Zukunft)

Reinforcement Learning

Bellmansches Optimalitätsprinzip

„An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.“ (Bellman 1957, Chap II.3)

Beispiel:



Wenn die Zustandsfolge 1,2,3,4,5 (mit Aktionen a,b,c,d) optimal ist, so ist auch jede Teilfolge (z.B. 2,3,4,5 mit Aktionen b,c,d) optimal.

Reinforcement Learning

Herleitung der Q-Learning Update-Formel

Sei Q* die optimale Value-Funktion:

$$Q_t^* = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$$Q_t^* = r_t + \sum_{i=1}^{\infty} \gamma^i r_{t+i}$$

$$Q_t^* = r_t + \sum_{i=0}^{\infty} \gamma^{i+1} r_{t+i+1}$$

$$Q_t^* = r_t + \gamma \sum_{i=0}^{\infty} \gamma^i r_{i+(t+1)}$$

$$Q_t^* = r_t + \gamma Q_{t+1}^*$$

$$0 = r_t + \gamma Q_{t+1}^* - Q_t^*$$

$$r_t + \gamma \, max_{a_t}(Q_{t+1}) - Q_t = \begin{cases} \text{negativ,wenn } Q_t \text{ zu groß} \\ \text{0, wenn } Q_t \text{ optimal} \\ \text{positiv, wenn } Q_t \text{ zu klein} \end{cases}$$

$$Q_t = Q_t + \alpha \left(r_t + \gamma \, max_{a_t}(Q_{t+1}) - Q_t \right)$$

$\alpha \in [0..1]$ *ist Lernrate z.B.* $\alpha = 0.01$
$\gamma \in [0..1]$ *ist Discountfaktor*
r_t *ist Reward (Belohnung)*

Reinforcement Learning

Aktionsauswahl

- Immer die Aktion mit der größten zu erwartenden Belohnung zu wählen nennt man Greedy-Strategie
- Fast immer die Aktion mit der größten Bewertung zu wählen, aber mit einer ε-Wahrscheinlichkeit eine der anderen Aktionen zu wählen nennt man ε-greedy
- Alle Aktionen mit gewichteten Wahrscheinlichkeiten zu wählen nennt man Softmax Action Selection:

$$p[a] = \frac{e^{Q_t[a]/\tau}}{\sum_{j=1}^{n} e^{Q_t[j]/\tau}}$$

Reinforcement Learning

Softmax action Selection

$$p[a] = \frac{e^{Q_t[a]/\tau}}{\sum_{j=1}^{n} e^{Q_t[j]/\tau}}$$

Die Verteilung nennt man Gibbs oder Boltzmann Verteilung

tau nennt man Temperatur

Für T → unendlich ist p[a] für alle a etwa gleich groß

Für T → 0 ist $p[a_{greedy}]=1$ und für alle anderen $p[a_{not\_greedy}]=0$

Reinforcement Learning

Markow Entscheidungs Prozess

Reinforcement Learning wird häufig als Markow Entscheidungsprozess aufgefasst und es wird vorrausgesetzt, dass die Markoweigenschaft erfüllt ist.

Markoweigenschaft

Die Markoweigenschaft garantiert, dass das Verhalten des Systems und damit die Wahrscheinlichkeiten, in welche Folgezustände das System gelangt nur vom aktuellen Zustands-Aktionspaar abhängt.

Reinforcement Learning

Q-Learning(Watkins, 1989)

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \, max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

α = [0..1] ist die Lernrate

Der Algorithmus:

```
Initialisiere Q(s,a) zufällig mit kleinen Zahlen
repeat (für jede Episode)
     Initialisiere st
     Repeat (für jeden Schritt in der Episode)
          Wähle eine Aktion a aus (z.B. mit Epsilon Greedy)
          Führe Aktion a aus und erhalte r und st+1
          Q(st, at) = Q(st, at) + α[rt + γ max_a Q(st+1, a) − Q(st, at)]
          st = st+1
     until st is terminal
```

Reinforcement Learning

Zustand s aus Koordinaten xi berechnen

Seien $X = \{ x_0, x_1 \ldots x_n \}$ diskretisierte Koordinaten mit $x_i \in [0..(max_i - 1)]$ eines Zusandes, dann kann man daraus folgendermaßen einen gesamten Zustand s berechnen:

```
int getState (int x[], int max[])

   s = x[0]

   for (i=1; i< x.size; i++)

        s = s*max[i] + x[i]

return s
```

Reinforcement Learning

Aufgabe 4)

Implementieren Sie einen Q-Learning Agenten der lernt ein PingPong Spiel zu spielen! Das Spiel wird vorgegeben.

- Reward = 1 bekommt der Agent, wenn der Schläger den Ball trifft.
- Reward = -1 bekommt der Agent, wenn er den Ball nicht trifft

Vektorquantisierung

Vektorquantisierung

Überblick

- Einführung
- Lernende Vektorquantisierer (LVQ)
- Selbst organisierende Karte (Kohonen Maps)
- K-Nearest Neighbor (KNN)

Vektorquantisierung

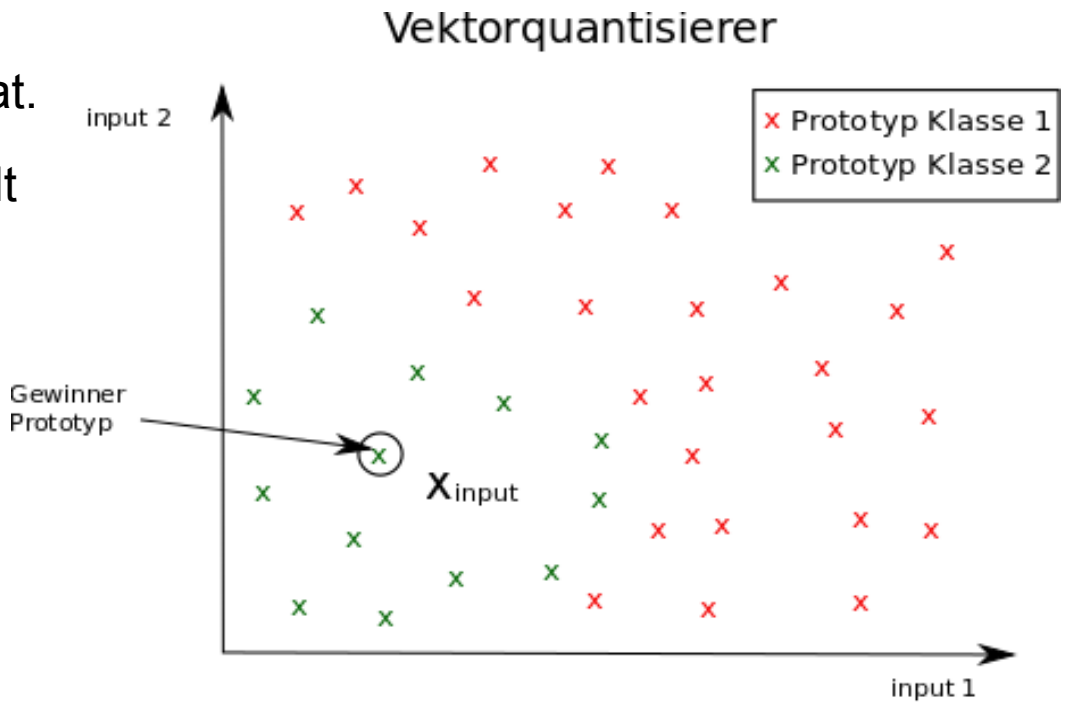
Einführung

- Eine Aufgabe, die häufig mit Machine Learning Algorithmen gelöst wird, ist die Klassifizierung, also die Zuordnung von Daten zu einer von mehreren Klassen
- Je nach Komplexität der Daten braucht man dafür z.B. ein tiefes neuronales Netz, um die Daten über mehrere Abstraktionsebenen einer Klasse zuordnen zu können.
- Bei Daten mit einfachen Zusammenhängen kann aber auch schon ein Vektorquantisierer oder ein K-Nearest Neighbor Algorithmus passable Ergebnisse erzeugen.

Vektorquantisierung

Vektorquantisierer

- Ein Vektorquantisierer beinhaltet Prototypvektoren der Dimension **n**.
- Es ist ein Abstandsmass definiert (z.B. euklidischer Abstand)
- Ein Eingabevektor wird nun dem Prototypen zugeordnet, der den kleinsten Abstand hat.
- Jeder Prototyp enthält enthält auch die Klasse, die seinem Bereich zugeordnet ist.



Vektorquantisierung

Lernender Vektorquantisierer

- Ein lernender Vektorquantisierer bewegt den nächsten Prototypenvektor jedes mal, wenn ein Eingabevektor präsentiert wird, der zur gleichen Klasse gehört, einen Schritt in die Richtung dieses Eingabevektors.
- Gehört ein Prototyp allerdings einer anderen Klasse an, so bewegt er sich vom Eingabevektor weg.
- Dafür wird eine Lernrate (z.B. 0.1) mit dem Differenzvektor zwischen Prototypen- und Eingabevektor multipliziert und auf den Prototypenvektor aufaddiert. Das Vorzeichen der Lernrate bestimmt dann, ob er sich auf den Eingabevektor zu, oder von ihm weg bewegt.
- Die Prototypen des Vektorquantisierers decken so den Bereich präsentierter Muster besonders gut ab.

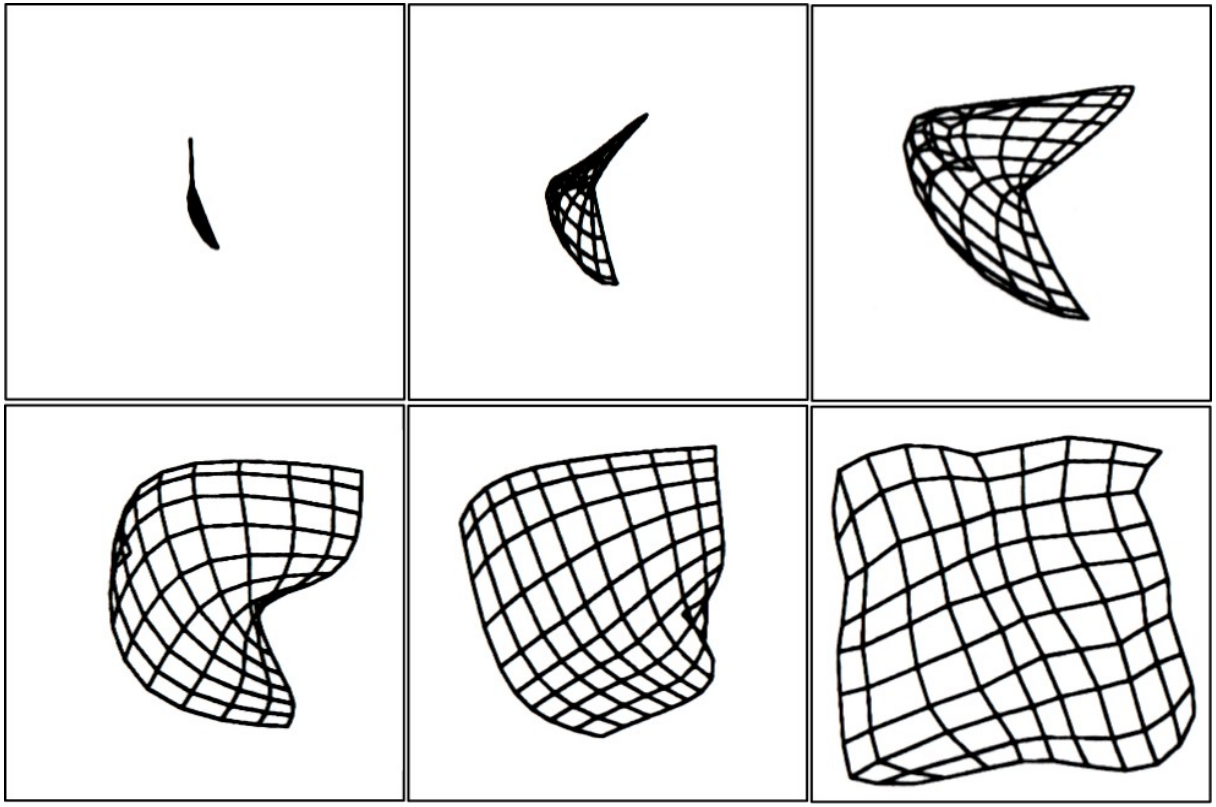
Vektorquantisierung

Selbstorganisierende Karte (Kohonens Map)

- Kohonens selbstorganisierende Karten (SOM) sind mit dem Vektorquantisierer verwandt.
- Auch hier gibt es Prototypen und es werden Eingabevektoren präsentiert.
- Das Lernverfahren ist allerdings, anders als beim LVQ unüberwacht.
- Beim SOM haben die Prototypen eine Nachbarschaft (z.B. 2 dimensional)
- Der Prototyp mit dem kleinsten euklidischen Abstand bewegt sich immer auf den präsentierten Eingabevektor zu.
- Die SOM-Nachbarprototypen bewegen sich einen kleineren Schritt auf den präsentierten Eingabevektor zu
- Ähnlich wie bei den Sensorischen Karten im Gehirn sind benachbarte Neurone auch üblicherweise für Reize aus einer benachbarten sensorischen Region zuständig

Vektorquantisierung

Selbstorganisierende Karte (Kohonens Map)



Die Prototypen befinden sich auf den Kreuzungspunkten des Gitters. Die Nachbarschaft ist zweidimensional und das Gitter entfaltet sich nach Präsentation von 3000 Eingabemustern.(Karten erzeugt mit KNet)

Vektorquantisierung

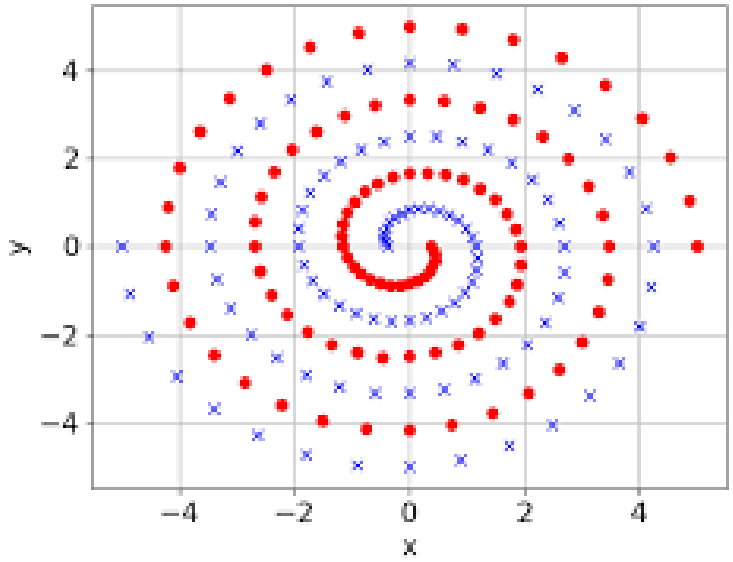
K-Nearest-Neigbor Algorithmus

- Auch K-Nearest Neigbor (KNN) ist mit dem Vektorquantisierer verwandt.
- Beim lazy learning werden die Prototypen an den zu lernenden Eingabevektoren plaziert (ohne sich zu bewegen) und den entsprechenden Klassen zugeordnet
- Nun lässt man via Mehrheitsentscheid, bei Präsentation eines Eingabevektors, die Klasse aus den k Prototypen bestimmen, die dem Eingabevektor am nächsten sind.
- Damit es immer eine Mehrheit gibt, sollte die Zahl k ungerade sein.
- Ist k zu groß, so kann eine stark vertretene Klasse überhand gewinnen.
- Ist k zu klein, so können Ausreißer zu Fehlklassifizierungen führen.

Vektorquantisierung

Aufgabe 5)

- Schreiben Sie ein Programm, dass mit Hilfe des K-Nearest-Neigbor Algorithmus zwei ineinander verwobene Spiralen voneinander trennen und jeder Spirale einer Klasse zuordnen kann.



Neuronale Netze

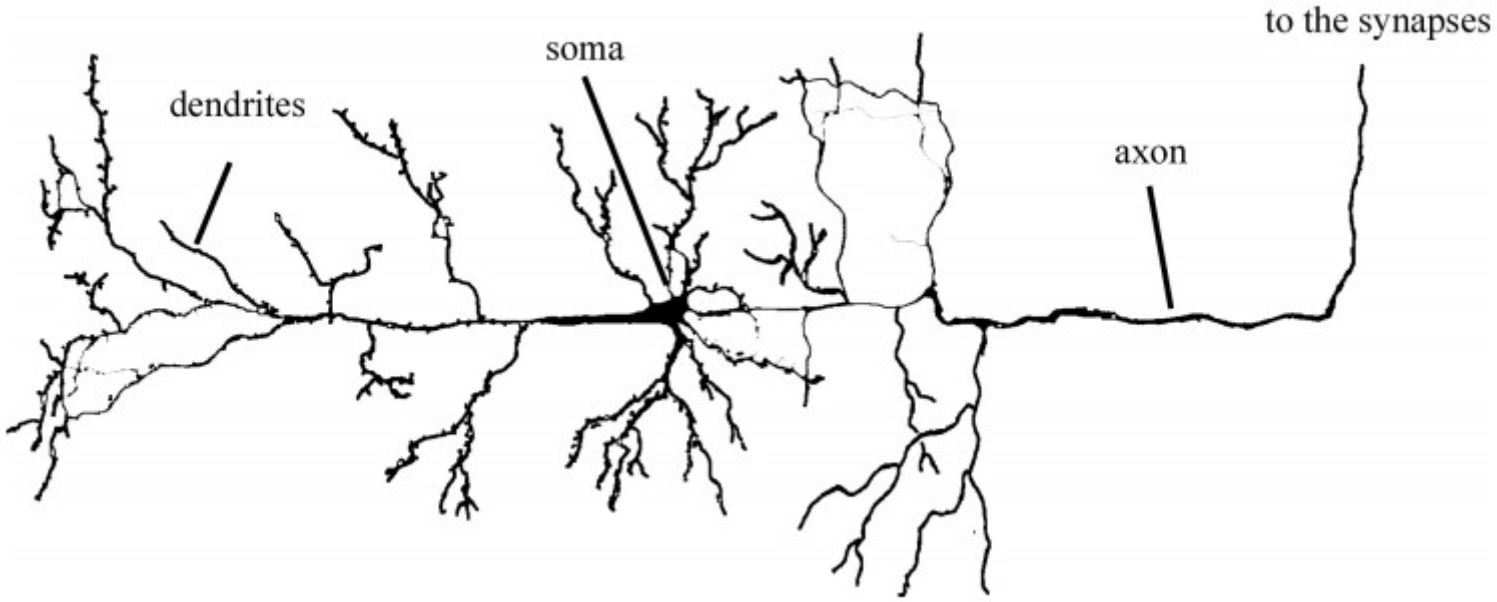
Neuronale Netze

Überblick

- Das Neuron
- Das Perzeptron
- Unsupervised Training
  - Bolzmann Maschine
  - Deep Belief Netzwerke
- Supervised Training
  - Delta-Lernregel
  - Backpropagation
- State of the Art
  - Deep Networks

Neuronale Netze

Das Neuron



- Eine Pyramidenzelle (Cajal, 1911) Die Information fließt von den Dendriten über den Soma zum Axon und zu den Synapsen, welche das Neuron mit anderen Neuronen verbindet

Neuronale Netze

Neuronen: Aktivität und Output

- Die **Aktivität** eines Neurons j ist gleich die Summe der gewichteten Eingänge:

$$\varphi_j = \sum_i W_{ji} o_i$$

- Der **Output** des Neurons ist die über eine Transferfunktion abgebildete Aktivität:

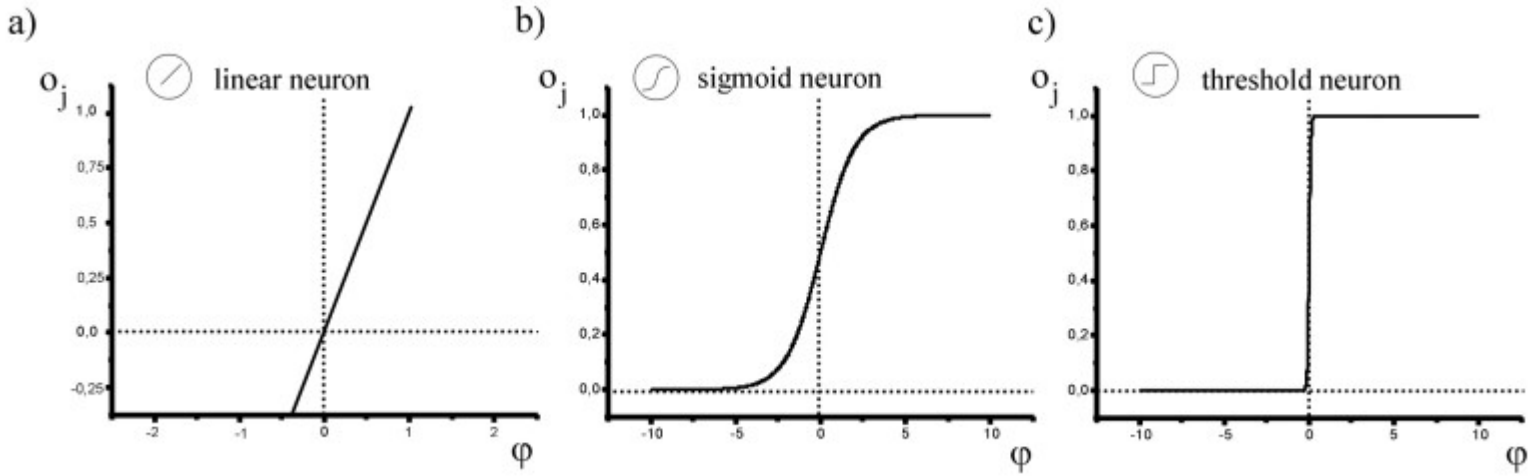
$$o_j = F(\varphi_j)$$

Neuronale Netze

- Eine übliche Transferfunktion ist die Standardsigmoide:

$$F(\varphi_j) = \frac{1}{(1 + e^{(-\varphi_j)})}$$

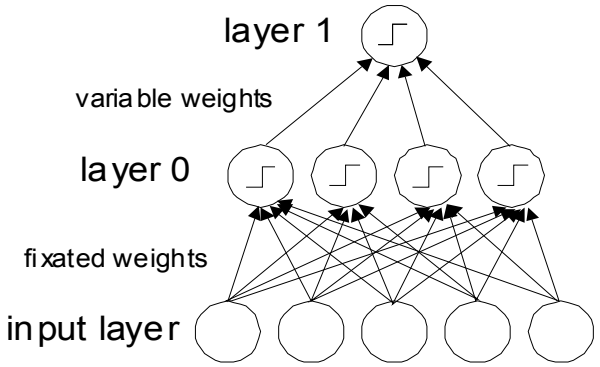
- Neben der Standardsigmoiden gibt es aber verschiedene Transferfunktionen mit unterschiedlichen Einsatzbereichen



Neuronale Netze

Das Perzeptron: Aufbau in Layer

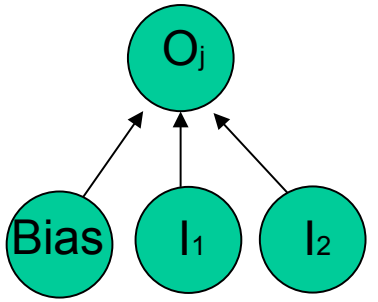
- Aus dem von 1943 von W.McCulloch und W.Pitts eingeführten Neuron als logisches Schwellwertelement folgte 1958 das von F. Rosenblatt beschriebene ein oder mehrlagige Perzeptron
- Das Perzeptron bildet die Grundlage für alle heutigen Feed Forward Neuronalen Netze
- F. Rosenblatt zeigte, dass ein Neuron AND, OR, NOT Operationen ausführen und erlernen kann
- M.Minsky und S. Papert zeigten, dass mit einer Schicht von Neuronen jedoch kein XOR erlernt werden kann.



Neuronale Netze

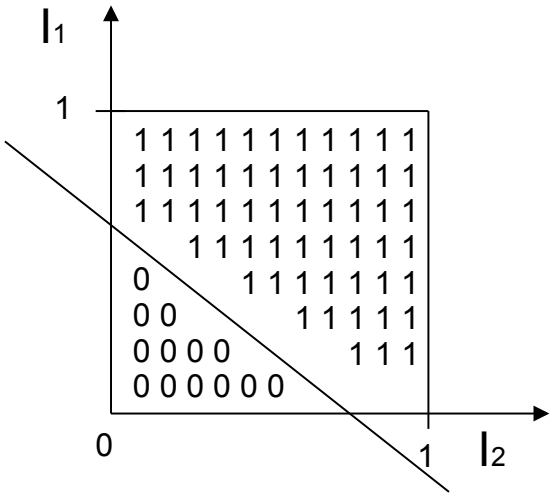
Das Perzeptron (1 Layer) und lineare Separierbarkeit

- Ein Neuron O_j mit 2 Eingabeneuronen I1 und I2 und einem Bias kann eine OR Verknüpfung darstellen!
- Schiebt man die Gerade (Abb. unten) nach rechts-oben, so wird aus dem OR ein AND



OR Problem

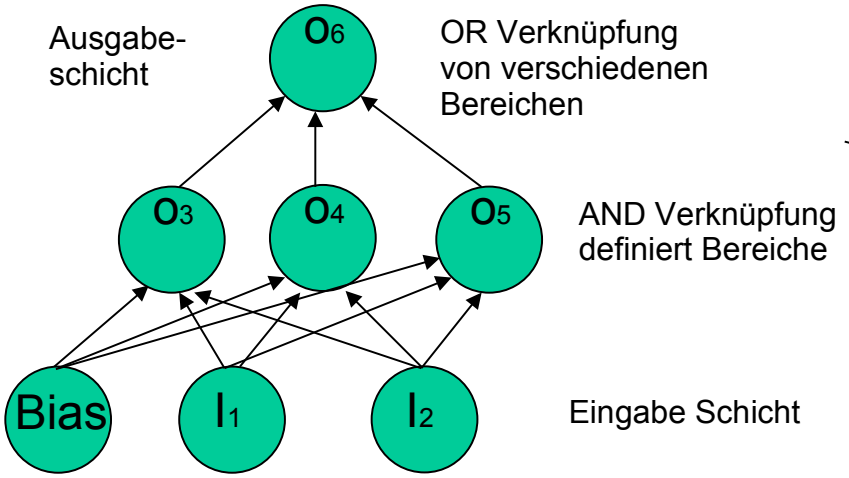
I1	I2	O
0	0	0
0	1	1
1	0	1
1	1	1

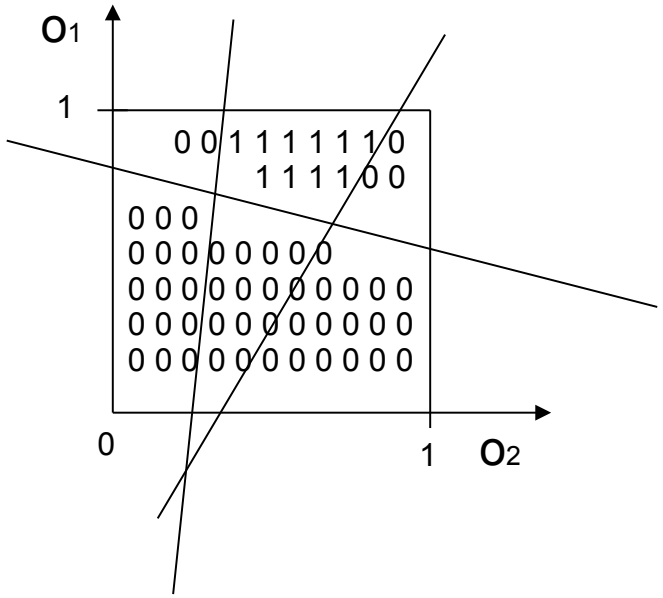


Neuronale Netze

Das Perzeptron (2 Layer)

- Die erste Schicht bildet durch AND Verknüpfung verschiedene Bereiche
- Die zweite (hier Ausgabeschicht) verknüpft die Bereiche zu Sammlungen von Bereichen z.B. für eine Klassifikation





Neuronale Netze

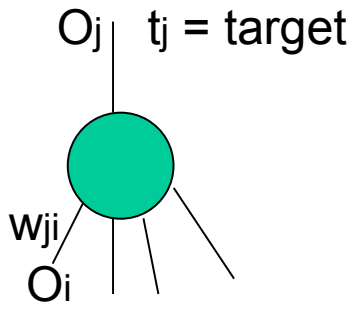
Das Perzeptron (Lernregel)

If $O_j=0$ and $t_j=1$ and $O_i=1$ then

$W_{ji} = W_{ji} + O_i$

If $O_j=1$ and $t_j=0$ and $O_i=1$ then

$W_{ji} = W_{ji} – O_i$



- Wenn der Eingang gleich Null ist, dann nützt eine Gewichtsänderung nichts!
- Ist der Ausgabewert kleiner als das Target, dann vergrößere das Gewicht
- Ist der Ausgabewert größer als das Target, dann verkleinere das Gewicht

Backpropagation

Neuronale Netze (Backprop)

Fehlerrückführung

Eine Fehlerrückführung erfolgt in zwei Phasen:

1) Vorwärtsaktivierung von der Eingangs- bis zur Ausgabeschicht

2) Fehler Propagation rückwärts von der Ausgabe bis zur Eingabeschicht

Zu beachten ist, dass jede Schicht ein Bias Neuron hat, welches immer 1 ausgibt. Alle Gewichte vom Bias Neuron zu den anderen Neuronen werden wie üblich mit Backpropagation trainiert!

Neuronale Netze (Backprop)

Delta Lernregel

Gradientenabstieg für lineare Netzwerke:

zu minimieren ist der

quadratischer Fehler: $E = \frac{1}{2}(O_j - t_j)^2$ mit $O_j = \sum_i W_{ji} O_i$

die zu optimierenden Parameter sind die Gewicht W_{ji}

der Gradient ist die Ableitung des Fehlers nach den Gewichten

$$\Delta W_{ji} = -\eta \frac{\partial E}{\partial W_{ji}} = -\eta \frac{\partial \frac{1}{2}(O_j - t_j)^2}{\partial W_{ji}} = -\eta O_i (O_j - t_j)$$

w = Gewicht

η = Lernrate

E = square error

Neuronale Netze (Backprop)

Delta Lernregel

Die Deltalernregel für Output Neuronen lautet also

$$\Delta W_{ji} = -\eta O_i (O_j - t_j)$$

w = Gewicht

η = Lernrate

E = square error

Mehrschichtige lineare Netze machen keinen Sinn, weil mehrere Schichten mittels Matrixmultiplikation der Gewichtsmatritzen zu einer zusammengefasst werden können!

Neuronale Netze (Backprop)

Backpropagation Lernregel (Ausgabeschicht)

Gradientenabstieg auf quadratischen Fehler des nichtlinearen Ausgabeneurons

$$\Delta w_{kj} = -\eta \frac{\partial E_k}{\partial w_{kj}} = -\eta \frac{\partial E_k}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{kj}}$$

$$= -\eta (o_k - t_k) F'\left(\sum_j w_{kj} o_j\right) o_j$$

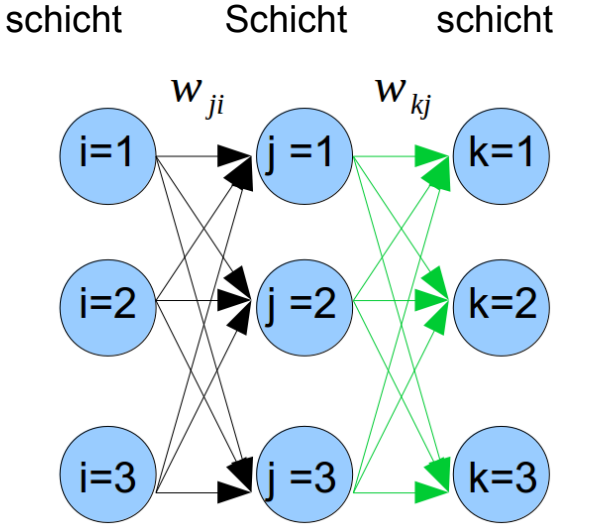
$$= -\eta e_k o_j$$

mit $E_k = \frac{1}{2}(o_k - t_k)^2$

mit $o_k = F\left(\sum_j w_{kj} o_j\right)$

- Wobei e_k der Fehlergradient der Ausgabeschicht ist
- Für eine Standardsigmoide F gilt:

$$F'(x) = \frac{\partial F(x)}{\partial x} = F(x)(1 - F(x))$$



Neuronale Netze (Backprop)

Backpropagation Lernregel (versteckte Schicht)

Gradientenabstieg nichtlineares verstecktes Neuron

$$\Delta w_{ji} = -\eta \frac{\partial \sum_k E_k}{\partial w_{ji}} = -\eta \sum_k \frac{\partial E_k}{\partial o_k} \frac{\partial o_k}{\partial o_j} \frac{\partial o_j}{\partial w_{ji}}$$

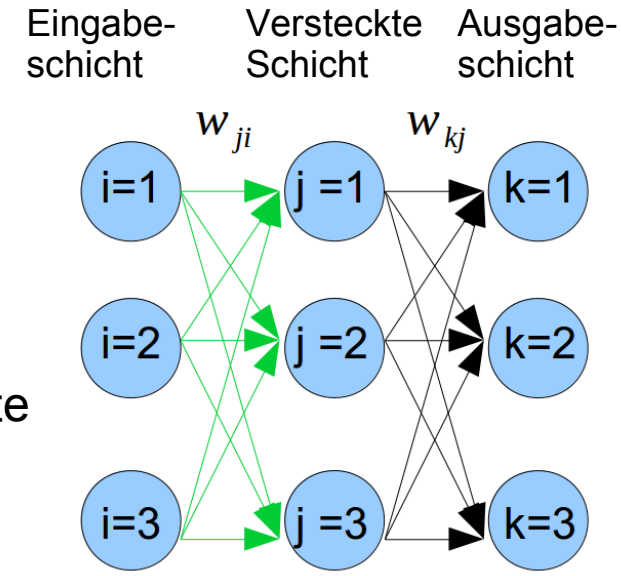
$$= -\eta \sum_k (o_k - t_k) \frac{\partial o_k}{\partial o_j} \frac{\partial o_j}{\partial w_{ji}}$$

$$= -\eta \sum_k (o_k - t_k) F'\left(\sum_j w_{kj} o_j\right) w_{kj} \frac{\partial o_j}{\partial w_{ji}}$$

$$= -\eta \sum_k (o_k - t_k) F'\left(\sum_j w_{kj} o_j\right) w_{kj} F'\left(\sum_i w_{ji} o_i\right) o_i$$

$$\Delta w_{ji} = -\eta e_j o_i \qquad e_j = \sum_k e_k w_{kj} F'\left(\sum_i w_{ji} o_i\right)$$

Der Fehlergradient e_k wird rückwärts über die Gewichte w_{kj} verteilt!



Neuronale Netze (Backprop)

Probleme mit Backpropagation

- Symmetry breaking: wenn man mit gleichen Gewichten in einem Layer startet, ist die Aktivität und der Output der Netze identisch. Damit sind auch die Gewichtsänderungen identisch…
- Local minima: beim Gradientenabstieg landet man mit großer Sicherheit in einem lokalen Minimum der Fehlerfunktion
- Flat plateau: Die Gewichtsänderung hängt direkt von dem Gradienten der Fehlerfunktion ab. Ist die Ableitung gering, wie bei flachen Ebenen, so ist es auch die Gewichtsänderung.
- Oscillation in steep gaps: Die Gewichtsänderung in tiefen Lücken kann sehr groß sein. Das kann zu oszillatorischem Verhalten führen, bei dem die Gewichte von einer Seite der Lücke zur anderen springen.
- Leaving good minima: Aus dem gleichen Grund können die Gewichte gute Minima sogar wieder verlassen.

Neuronale Netze (Backprop)

Lösungen:

- Momentum-term

$$\Delta w_{ji}^{t+1} = -\eta e_j o_i + \alpha \Delta W_{ji}^{t} \qquad \alpha \in [0..1]$$

- Flat-Spot Elimination

  Die Ableitung der Aktivierungsfunktion nahe bei 0 und nahe bei 1 ist nahezu 0 → für Backprop. Ist es schwer die Gewichte aus dem Zustand rauszuholen, weil die Gewichtsänderung sehr klein ist.

  Lösung: Eine Konstante z.B. 0.1 zur Ableitung addieren

- Weight Decay

  Um divergierende Gewichte zu verhindern führt man einen Term ein, der große Gewichte stärker reduziert als kleine:

$$\Delta w_{ji}^{t+1} = -\eta e_j o_i - d W_{ji}^{t} \qquad d \in [0..1]$$

Neuronale Netze (Backprop)

Weiterentwicklungen von Backpropagation

- Backprop. zweiter Ordnung

  Die zweite Ableitung hilft die korrekte Gewichtsänderung durchzuführen→ schnellere Konvergenz bei höherem Rechenaufwand

- Quickpropagation

  Iterative Nutzung zweiter Ordnung, ähnlich wie Newton Verfahren. Im Durchschnitt sehr viel schneller als Backpropagation.

- Backperculation

  Berechnet für jedes Neuron einen Aktivierungsfehler, indem das Muster im Netz zurückpropagiert wird. Dieser Fehler wird dann minimiert. Backpropagation minimiert den Aufsummierten Fehler der Ausgabeschicht.

Neuronale Netze (Backprop)

Neuere Verfahren

Bei RMS-Prop wird der Gradient durch die Wurzel des schleppenden Mittels der Gradienten geteilt und so normiert.

$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$

$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

Im Wesentlichen wird die Schrittweite nur noch durch die Lernrate vorgegeben, nicht durch den Gradienten selber. Der Gradient gibt nur noch die Richtung vor.

Damit werden Schwingungen in Richtung des stärksten Gradienten reduziert.

Neuronale Netze (Backprop)

Neuere Verfahren

ADAMS kombiniert die Heuristik von RMS-Prop und Momentum Term, so dass die Schwingungen in Richtung des stärksten Gradienten reduziert werden und die Suche in Richtung Minimum beschleunigt wird.

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

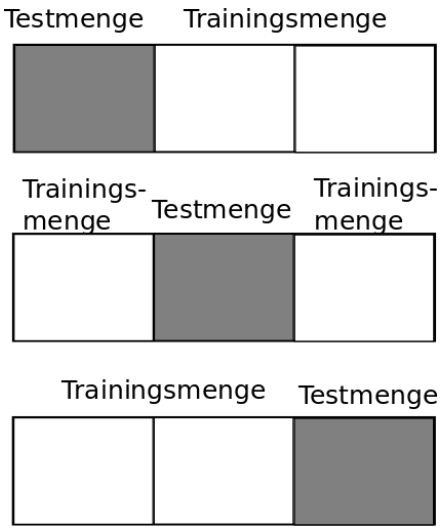
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

Neuronale Netze (Backprop)

K-fache Kreuzvalidierung

- Ein neuronales Netz, welches auf trainierten Daten einen kleinen Fehler erreicht kann auf Testdaten einen deutlich größeren Fehler zeigen (overfitting).
- Um den Fehler auf Testdaten zu evaluieren wendet man die Kreuzvalidierung an.
- Teilt man die Daten in K gleich große Teile, so kann man z.B. einen Teil als Testmenge nehmen und die restlichen Teile als Trainingsmenge. (z.B. 1/3 Testdaten, 2/3 Trainingsdaten) und ermittelt z.B. den quadratischen Fehler
- Dann wählt man das nächste Teil als Testmenge und den Rest als Traininigsmenge etc.
- Der Gesammtfehler errechnet sich als Mittelwert aus den Fehlern aller Testmengen.

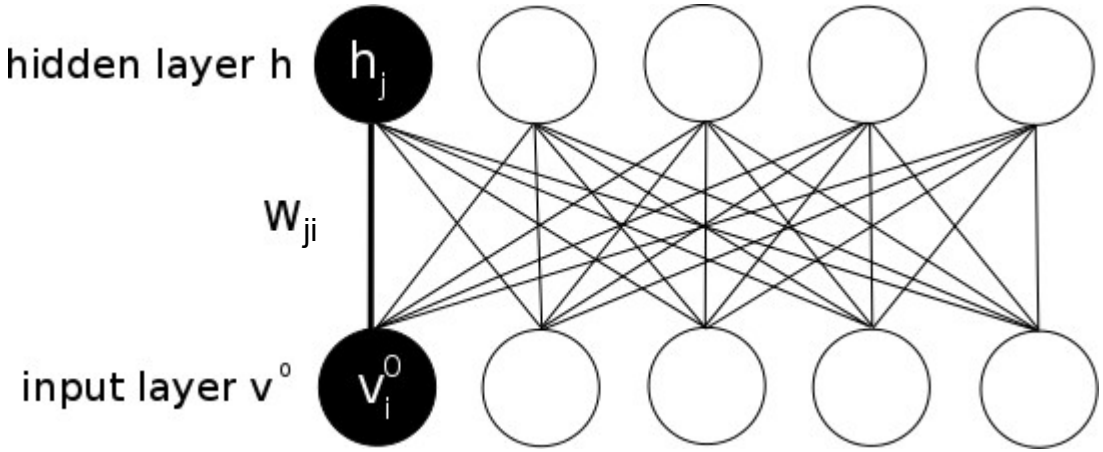


Deep Learning

Neuronale Netze (Deep Learning)

Restricted Bolzmann Machines

- Ziel der Bolzmann Maschine ist die Rekonstruktion des Input
- Dafür wird in einer Vorwärtsaktivierung die Hidden Schicht und dann in einem Rückwärtsschritt mit den gleichen Gewichten eine Rekonstruktion des Input berechnet



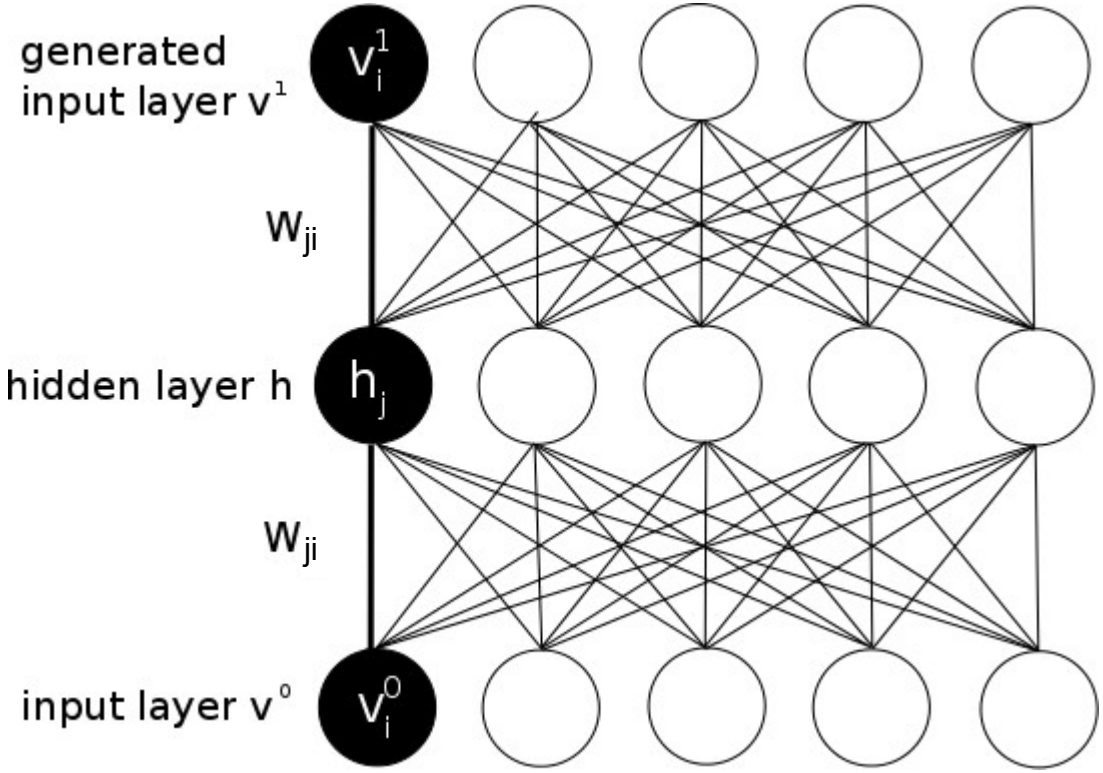
vorwärts Aktivierung

Rekonstruktion

Neuronale Netze (Deep Learning)

Restricted Bolzmann Machines

- Zur Veranschaulichung: hier die Rekonstruktion des Input gespiegelt

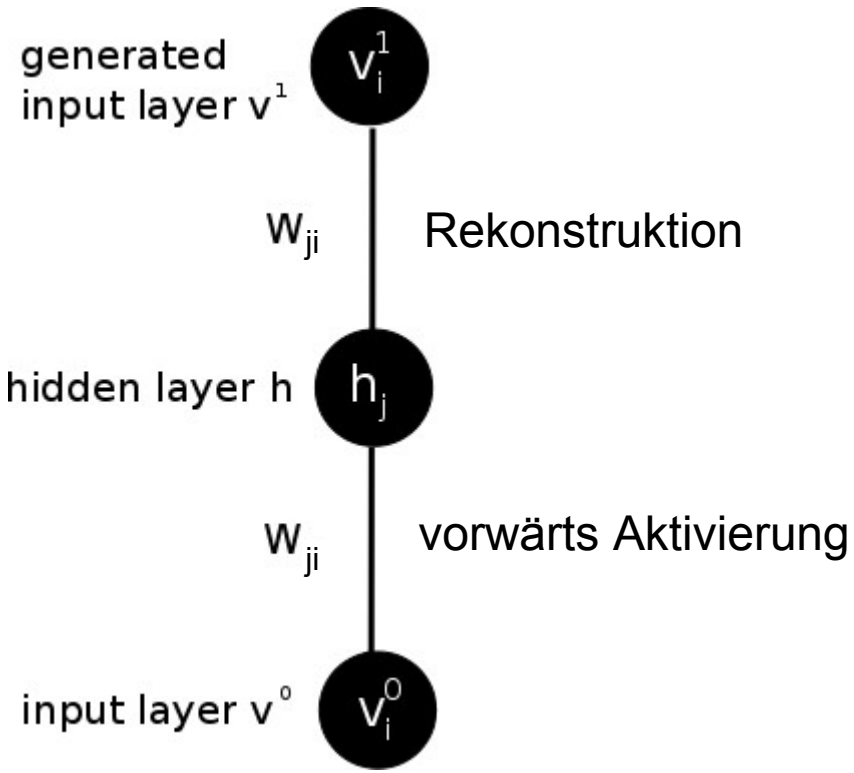


Rekonstruktion

vorwärts Aktivierung

Neuronale Netze (Deep Learning)

Restricted Bolzmann Machines



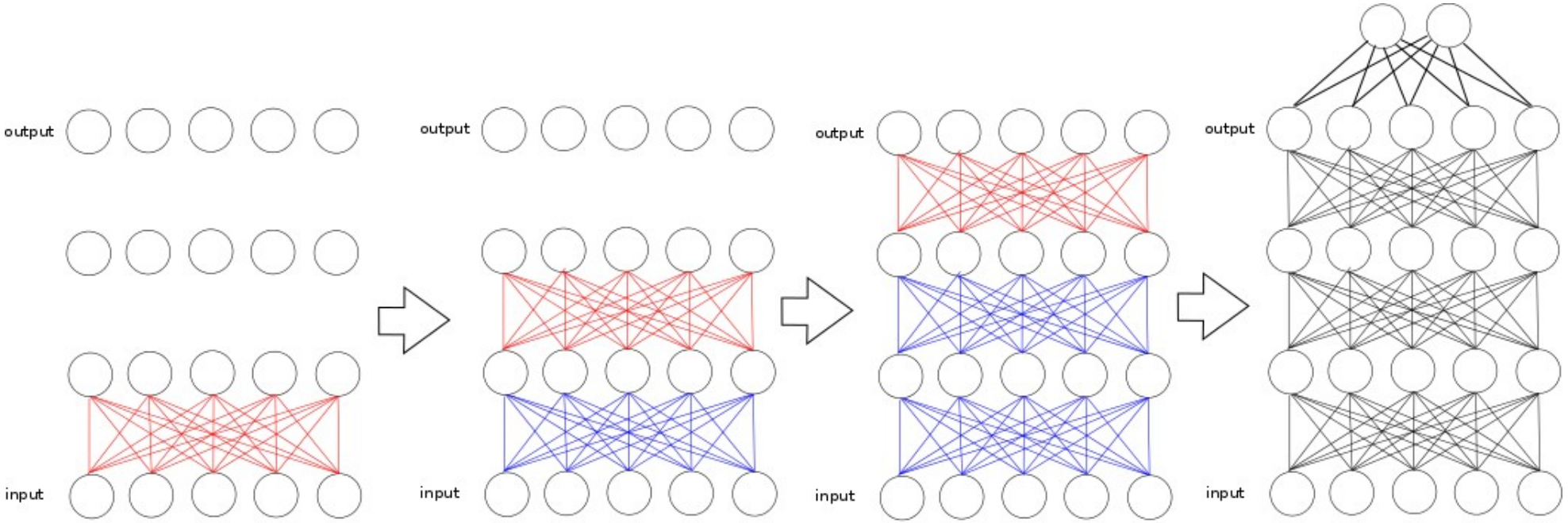
Lernregel

$$W_{ji}^{t+1} = W_{ji}^{t} + \eta\left(h_j\, v_i^0 - h_j\, v_i^1\right)$$

- Ist h_j=0, so hat dieses hidden Neuron keinen Einfluss auf die Rekonstruktion
- Ist h_j=1 und $v_i^0 > v_i^1$ dann vergrößert sich das Gewicht (v_i^1 zu klein)
- Ist h_j=1 und $v_i^0 < v_i^1$ dann verkleinert sich das Gewicht (v_i^1 zu groß)
- Ist h_j=1 und $v_i^0 = v_i^1$ dann verändert sich das Gewicht nicht mehr, denn die Rekonstruktion ist erfolgreich!

Neuronale Netze (Deep Learning)

Deep Network

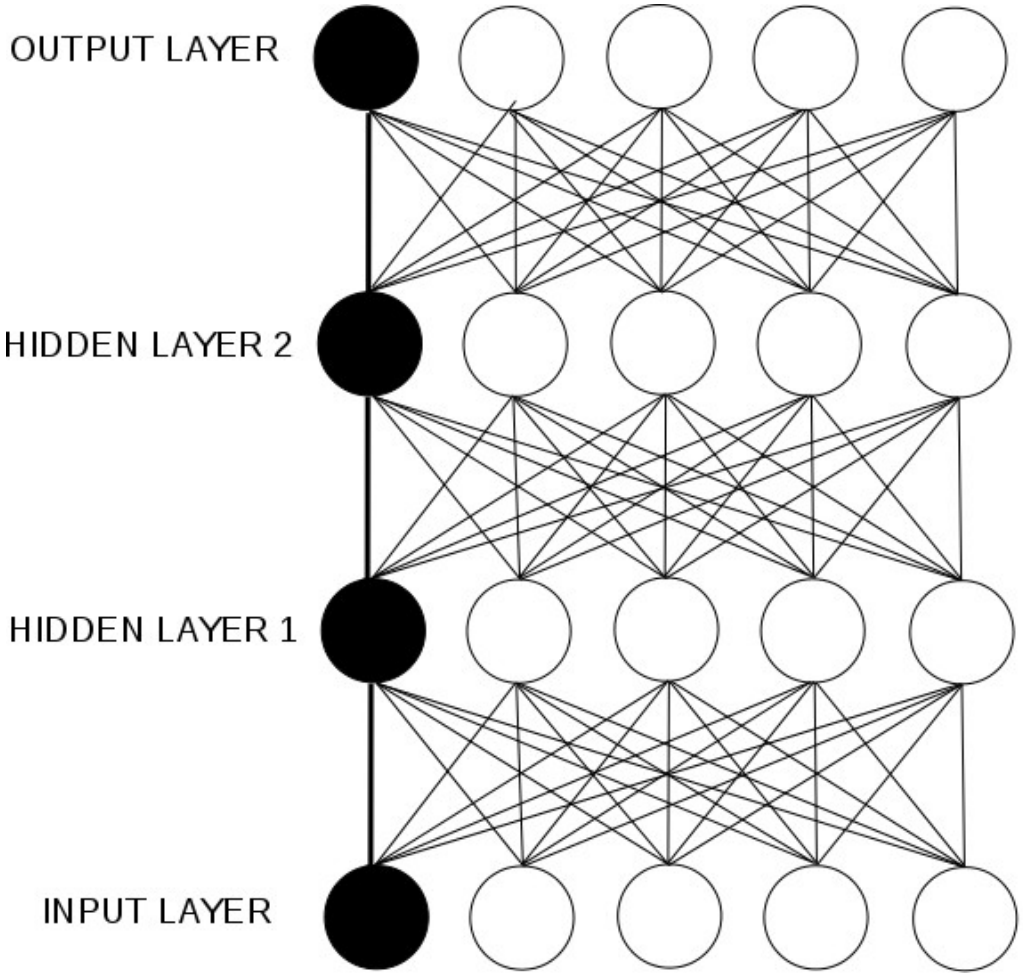


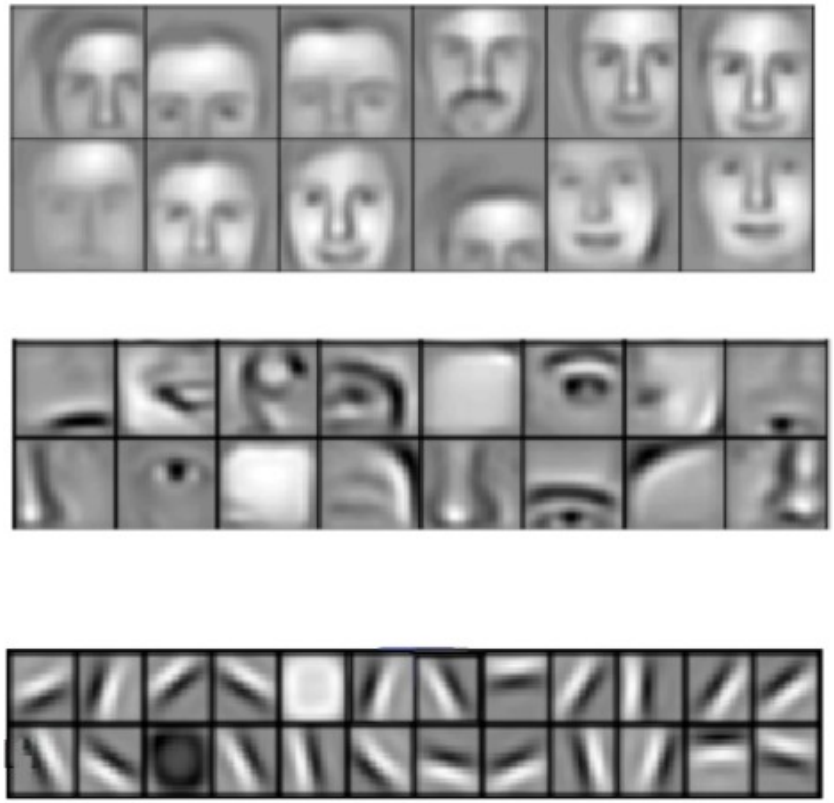
Train with Contrastive Divergence

Leave as is

Fine tune with Backpropagation

Neuronale Netze (Deep Learning)





Neuronale Netze

Aufgabe 6)

Programmieren Sie eine Restricted Bolzmann Maschine, die die handgeschriebenen Ziffern aus der MNIST Datenbank lernt. Zusätzlich soll am Eingang durch 10 Neuronen (immer nur eines aktiv) das Label mit eingespeichert werden. Bei der Rekonstruktion sollte dann auch die erkannte Ziffer mit rekonstruiert werden!