Neuronale Netze

Eine Einführung

Druckversion der Internetseite
www.neuronalesnetz.de

Inhaltsverzeichnis

Einleitung

Vorbild: Menschliches Gehirn

Neuronale Netze beziehen sich auf das Neuronennetz des menschlichen Gehirns. Dieses dient als Analogie und Inspiration für in Computern simulierte **künstliche neuronale Netze**. Diese Analogie steht bei heutigen Arbeiten zu neuronalen Netzen jedoch häufig nicht mehr im Vordergrund.

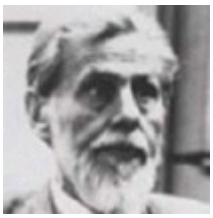


Abbildung 1: Warren McCulloch (1898 - 1972)

Abbildung 2: Walter Pitts (1924 - 1969)

Die Ersten, die sich mit dem Thema neuronale Netze beschäftigten, waren **Warren McCulloch und Walter Pitts** im Jahr 1943 mit ihrem Formalmodell des Neurons. Die Arbeiten mit und zu neuronalen Netzen haben seit ca. 1986 sehr stark zugenommen. Es existieren inzwischen zahlreiche wissenschaftliche Zeitschriften, die sich primär mit diesem Thema auseinandersetzen, z. B. "Neurocomputing", "Neural Computation" oder "Neural Networks".

Zweiteilung der Themengebiete

Mittlerweile lässt sich der Anwendungsbereich der Methode in zwei große Bereiche unterteilen:

- Künstliche neuronale Netze, die modelliert werden, um menschliches Verhalten und Erleben bzw. die **Funktionsweise des menschlichen Gehirns besser zu verstehen**.
- Künstliche neuronale Netze, die dazu dienen **konkrete Anwendungsprobleme** aus Bereichen wie z. B. Statistik, Wirtschaftswissenschaften, Technik und vielen anderen Gebieten zu lösen.

Matrizenberechnungen

Insgesamt übt die Idee der neuronalen Netze auf viele Menschen eine sehr hohe Faszinationskraft aus. Diese Lernhilfe soll auf der einen Seite versuchen, diesen Enthusiasmus auf Sie zu übertragen. Zugleich soll jedoch auch gezeigt werden, dass hinter dem Thema neuronale Netze nichts weiter als Matrizenberechnungen stehen.

Doch zunächst zu den Grundeinheiten neuronaler Netze, den Neuronen...

Units

Definition: Units

Neuronale Netze bestehen aus mehreren Neuronen. Diese Neuronen werden auch als Units, Einheiten oder Knoten bezeichnet. Sie dienen dazu, Informationen aus der Umwelt oder von anderen Neuronen aufzunehmen und an andere Units oder die Umwelt in modifizierter Form weiterzuleiten.

unterschiedliche Unitarten

Man unterscheidet zwischen 3 verschiedenen Arten von Neuronen:

- **Input-Units**: Units, die von der Außenwelt Signale (Reize, Muster) empfangen können.
- **Hidden-Units**: Units, die sich zwischen Input- und Output-Units befinden und eine interne Repräsentation der Außenwelt beinhalten.
- **Output-Units**: Units, die Signale an die Außenwelt weitergeben.

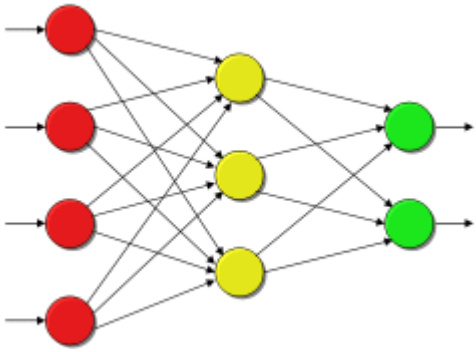


Abbildung 3: Schematische Darstellung eines neuronalen Netzes

"Übereinander" angeordnete Knoten (z.B. die beiden rechts befindlichen Output-Units in Abbildung 3) fasst man als Schicht bzw. Layer zusammen.

Verbindungen zwischen Units

Darstellung der Verbindungen durch Gewichte

Units sind miteinander durch Kanten verbunden. Die Stärke der Verbindung zwischen zwei Neuronen wird durch ein Gewicht ausgedrückt. Je größer der Absolutbetrag des Gewichtes ist, desto größer ist der Einfluss einer Unit auf eine andere Unit.

- Ein **positives Gewicht** bringt zum Ausdruck, dass ein Neuron auf ein anderes Neuron einen exzitatorischen, erregenden Einfluss ausübt.
- Ein **negatives Gewicht** bedeutet, dass der Einfluss inhibitorisch, also hemmender Natur ist.
- Ein **Gewicht von Null** besagt, dass ein Neuron auf ein anderes Neuron derzeit keinen Einfluss ausübt.

Definition: Wissen und Lernen

Das **Wissen** eines neuronalen Netzes ist in seinen Gewichten gespeichert.

Lernen wird bei neuronalen Netzen zumeist als Gewichtsveränderungen zwischen den Einheiten definiert. Wie die Gewichtsveränderung genau erfolgt ist abhängig von der verwendeten **Lernregel**.

Input und Netzinput

Definition: Input

Der Input (bzw. die Eingabe), den ein Neuron von einer anderen Unit empfängt, hängt von zwei Werten ab, die zumeist multiplikativ miteinander verknüpft sind:

- Output (bzw. Aktivitätslevel) der sendenden Einheit
- Gewicht zwischen den beiden Neuronen

Je stärker also der Aktivitätslevel der sendenden Einheit und je höher das Gewicht zwischen den beiden Units, desto größer ist der Einfluss (Input) auf die empfangende Einheit. Ist einer der beiden Terme gleich Null, so ist kein Einfluss vorhanden.

Definition: Netzinput

Der gesamte Input einer Unit wird Netzinput (auch: Netto-Input, Netzeingabe, Netinput) genannt. Dieser wird über die sog. Propagierungsfunktion bestimmt. Die verbreitetste Propagierungsfunktion ist eine Linearkombination, bei der sich der Netzinput additiv aus sämtlichen einzelnen Inputs zusammensetzt, die das Neuron von anderen Neuronen erhält.

Formeldarstellung

Input und Netzinput einer Unit lassen sich auch als Formeln darstellen.

Formel: Input

Input der Unit i: $input_{ij} = a_j \, w_{ij}$

dabei bezieht sich "i" auf die empfangende Unit, während "j" die sendenden Units meint.

Konvention: Die erste Indexstelle kennzeichnet immer die empfangende Einheit, der zweite Index betrifft die sendende Einheit.

a_j = Aktivitätslevel der sendenden Unit j

w_{ij} = Gewicht zwischen der sendenden (j) und der empfangenden (i) Unit

Formel: Netzinput

Netzinput der Unit i: $netinput_i = \sum_j input_{ij} = \sum_j a_j \, w_{ij}$

Die bereitgestellte interaktive Visualisierung dient dazu, den Zusammenhang zwischen Aktivitätslevel, Gewichten und Netzinput noch besser zu verstehen. Probieren Sie diese doch einfach einmal aus!


Aktivitätslevel
Gewichte
Netzinput
0
0
0
0
0
0
Animation starten
Durch Anklicken starten Sie ein Java-Applet.
Bei Problemen aktivieren Sie JavaScript in Ihrem Browser und laden Sie die neueste Java-Version auf http://www.java.com herunter.

(0 · 0) + (0 · 0) + (0 · 0) = 0

Aktivitätsfunktion, Aktivitätslevel und Output

Definition: Aktivitätsfunktion

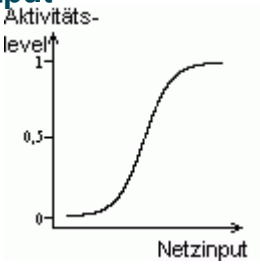


Abbildung 4: Zweidimensionales Liniendiagramm mit sigmoider Aktivitätsfunktion

Die Aktivitätsfunktion (Transferfunktion, Aktivierungsfunktion) stellt den Zusammenhang zwischen dem Netzinput und dem Aktivitätslevel eines Neurons dar. Die Aktivitätsfunktion wird in einem 2-dimensionalen Diagramm visualisiert, wobei auf der Abszisse (x-Achse) der Netzinput der Einheit und auf der Ordinate (y-Achse) der entsprechende Aktivitätslevel abgetragen wird. Der Aktivitätslevel wird durch eine sog. Ausgabefunktion dann in den Output transformiert, den das Neuron an andere Neuronen weitersendet. Häufig wird als Ausgabefunktion die Identitätsfunktion verwendet, d.h. der Output ist gleich dem Aktivitätslevel. Dies wird auch hier im folgenden angenommen.

verschiedene Aktivitätsfunktionen

Man unterscheidet zwischen verschiedenen Aktivitätsfunktionen:

- **Lineare Aktivitätsfunktion**: Hier ist der Zusammenhang zwischen Netzinput und Aktivitätslevel linear.
- **Lineare Aktivitätsfunktion mit Schwelle**: Bevor der Zusammenhang zwischen den beiden Größen linear wird, muss eine zuvor festgelegte Schwelle überschritten werden. Dies kann sinnvoll sein, wenn ein zu niedriger Netzinput (z. B. ein Rauschen) nicht als Signal weitergeleitet werden soll.
- **Binäre Schwellenfunktion**: Hier gibt es nur zwei Zustände des Aktivitätslevels, 0 (bzw. manchmal auch -1) oder 1.
- **Sigmoide Aktivitätsfunktion**: Diese Art von Aktivitätsfunktion wird in den meisten Modellen verwendet, die kognitive Prozesse simulieren. Man kann dabei die logistische Funktion und die Tangens-Hyperbolicus-Funktion unterscheiden. Beide Funktionen verhalten sich jedoch relativ ähnlich: Ist der Netzinput (vom Betrag her) groß und negativ, dann ist der Aktivitätslevel nahe 0 (logistische Funktion) bzw. -1 (Tangens-Hyperbolicus-Funktion), steigt dann zunächst langsam an (eine Art Schwelle), danach wird der Anstieg steiler und gleicht einer linearen Funktion. Bei einem hohen Netzinput nähert sich der Wert dann asymptotisch der 1 an (siehe Abbildung 4). Sigmoide Aktivitätsfunktionen bieten demnach zwei wesentliche Vorteile:
  - **Begrenzung des Aktivitätslevels**: Im Gegensatz zu den linearen Aktivitätsfunktionen ist der Aktivitätslevel hier sowohl nach oben als auch nach unten begrenzt. Dies deutet nicht nur auf eine höhere biologische Plausibilität hin (vgl. die begrenzte Intensität des Aktionspotentials biologischer Neuronen), sondern hat auch den Vorteil, daß die Aktivität im Netz (bedingt durch rekurrente Verbindungen) nicht ungewollt "überschwappen" kann und dadurch nur noch Fehlerwerte produziert werden.
  - **Mögliche Differenzierbarkeit**: Im Gegensatz zu der binären Schwellenfunktion ist die Funktion an allen Stellen differenzierbar, was beispielsweise eine notwendige Voraussetzung für das noch vorzustellende Gradientenabstiegsverfahren ist.

Exkurs: Bias Unit

Definition: Bias-Unit

Die Bias-Unit erhält selbst keinen Input, ihr Aktivitätslevel beträgt immer +1 (siehe Abbildung 5). Das Gewicht von der Bias-Unit zu einer anderen Unit kann positiv oder negativ sein. Wenn kein starker Input von anderen Einheiten erfolgt, dann stellt die Bias-Unit sicher, dass die Einheit bei positivem Gewicht aktiv bleibt. Bei negativem Gewicht sorgt die Bias-Einheit hingegen dafür, dass die Unit in ihrem inaktiven Zustand verharrt.

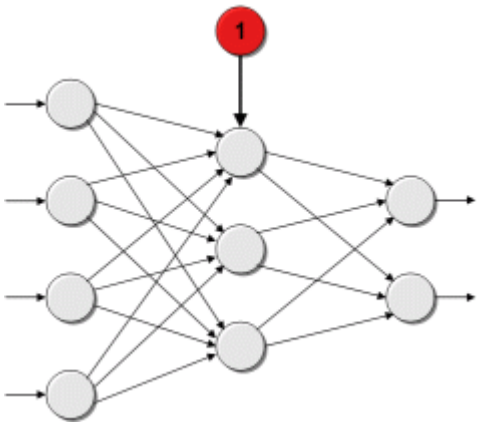


Abbildung 5: Schematische Darstellung eines neuronalen Netzes mit Bias-Unit (in rot)

Funktion der Bias-Unit

Dies kann nützlich sein, wenn man eine Schwelle benötigt (beim negativen Bias), die andere Input-Units erst überschreiten müssen. Diese Schwelle ist übrigens anders als die Schwelle einer Aktivierungsfunktion veränderbar, da das Gewicht zwischen Bias- und Empfängerunit wie alle anderen Gewichte durch Lernen modifizierbar ist. Umgekehrt kann das Ziel auch sein, dass die Einheit sehr häufig feuern, also gewöhnlich aktiv sein soll. Dazu verwendet man einen positiven Bias.

Trainings- und Testphase

Bei neuronalen Netzen unterscheidet man typischerweise zwischen einer Trainingsphase und einer Testphase (auch Ausbreitungsphase genannt).

Trainingsphase

- **Trainingsphase**: In dieser Phase lernt das neuronale Netz anhand des vorgegebenen Lernmaterials. Dementsprechend werden in der Regel die Gewichte zwischen den einzelnen Neuronen modifiziert. Lernregeln (siehe nächste Seiten) geben dabei die Art und Weise an, wie das neuronale Netz diese Veränderungen vornimmt. Viele Lernregeln lassen sich in die folgenden beiden Kategorien einordnen (weitere Möglichkeiten wie das sog. reinforcement learning werden hier nicht behandelt):
  - **supervised learning** (überwachtes bzw. beaufsichtigtes Lernen): Der korrekte Output wird (als "teaching vector") vorgeben und daran werden die Gewichte optimiert.
  - **unsupervised learning** (nicht überwachtes bzw. unbeaufsichtigtes Lernen): Es wird kein Output vorgegeben. Die Gewichtsveränderungen erfolgen in Abhängigkeit der Ähnlichkeit der Gewichte mit den Inputreizen.

Testphase

- **Testphase**: In der Testphase werden hingegen keine Gewichte verändert. Statt dessen wird hier auf Grundlage der bereits modifizierten Gewichte aus der Trainingsphase untersucht, ob das Netz etwas gelernt hat. Dazu präsentiert man den Inputneuronen Reize und prüft, welchen Output das neuronale Netz berechnet. Zwei verschiedene Arten von Reizen können unterschieden werden:
  - **Ausgangsreize**: Durch erneute Präsentation der zu lernenden Ausgangsreize wird geprüft, ob das neuronale Netz das Trainingsmaterial erfasst hat.
  - **Neue Reize**: Durch Präsentation neuer Reize kann man feststellen, ob das Netz über die zu lernenden Reize hinaus in der Lage ist, Aufgaben zu lösen. Anders formuliert: Findet eine Generalisierung auf neue Reize statt?

Exkurs: Matrizendarstellung

Definition: Matrix

Neuronale Netze lassen sich auch als Matrizen darstellen (siehe Abbildung 6). Dies hat den Vorteil, dass die zuvor erläuterten Berechnungen mathematisch relativ einfach und zusammenfassend vorgenommen werden können. Eine Matrix ist eine mathematische Einheit, genau wie eine einzelne Zahl. Solch eine Matrix **W** besteht aus einer Menge von Elementen $\mathbf{w_{ij}}$. Der erste Index i gibt dabei die Zeile in der Matrix an, der zweite Index j hingegen die Spalte in der das Element steht. Da das Lernen in neuronalen Netzen in den Gewichten stattfindet und diese das gelernte Wissen des Netzes speichern, werden die Netzgewichte als Matrix dargestellt.

Gewichtsmatrix

Ein neuronales Netz kann man durch *eine* Gewichtsmatrix darstellen, sofern keine Hidden-Schicht existiert. Bei einer Hidden-Schicht würde man zwei Gewichtsmatrizen benötigen, bei zwei Hidden-Schichten drei Matrizen usw.

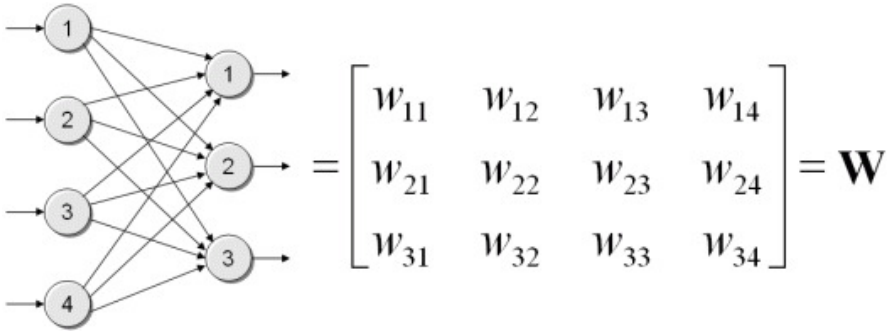


Abbildung 6: Darstellung der Äquivalenz zwischen schematischer Illustration und Matrizenschreibweise bei neuronalen Netzen. Der jeweils erste Index der Gewichte bezieht sich auf die Output-Unit (1, 2 oder 3), der zweite Index gibt die entsprechende Input-Unit (1 bis 4) des neuronalen Netzes an.

Concept map: Grundlagen

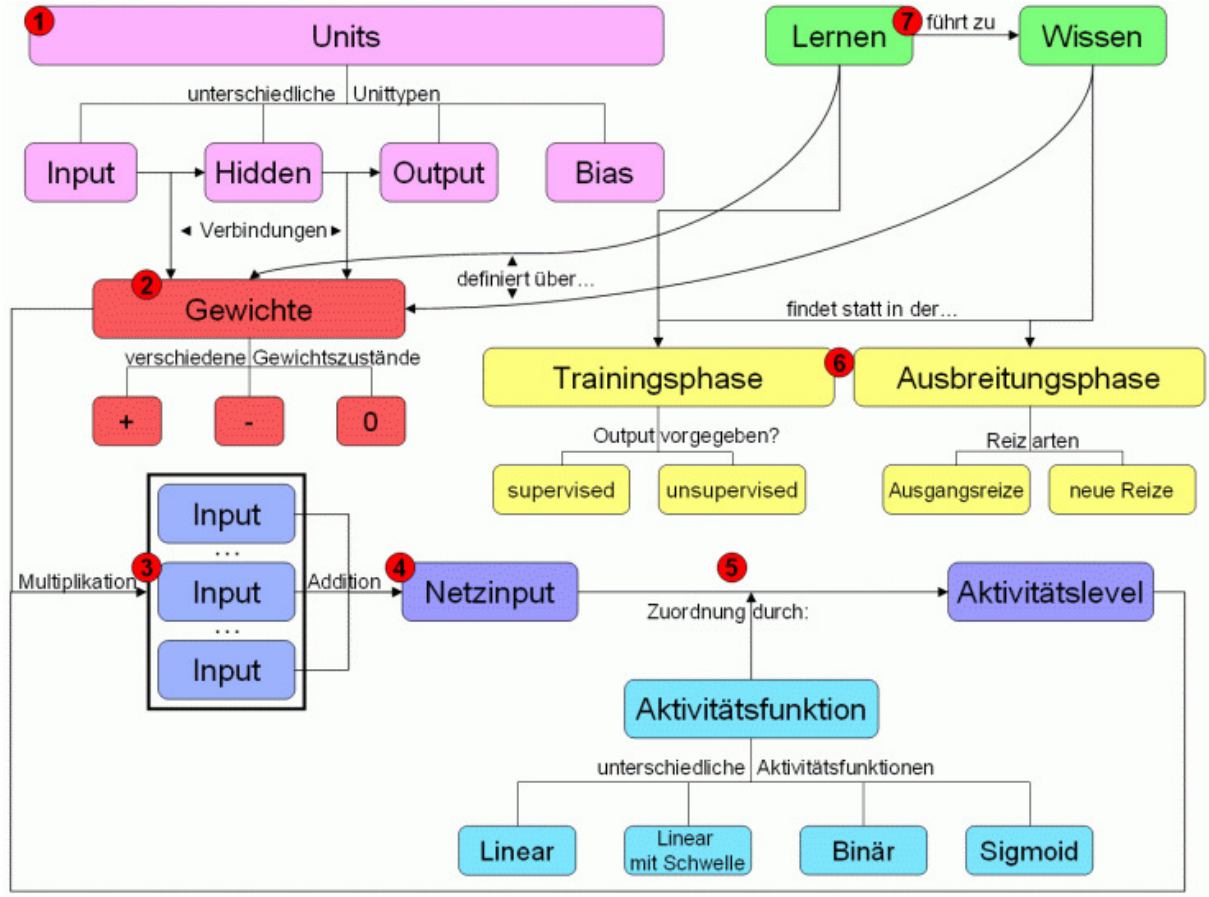


Abbildung 7: Concept map zu dem Kapitel Grundlagen. Erläuterung: siehe Text.

Dieser visuelle Strukturüberblick fasst die zentralen Konzepte des Kapitels "Grundlagen" als Concept map zusammen. (Eine Concept map ist eine Begriffstrukturdarstellung, die aus Knoten, welche Personen, Objekte oder Konstrukte repräsentieren und gerichteten Kanten, die die Beziehungen zwischen den Knoten darstellen, besteht.)

Die 7 rot markierten Kreise kennzeichnen Teilbereiche in der Grafik, die im folgenden erläutert werden:

1. Oben links im Bild sehen Sie 4 verschiedene **Unittypen**, wobei Input- und Hidden-Units sowie Hidden- und Output-Units durch Verbindungen miteinander verknüpft sind.
2. Diese Verbindungen können als positive, negative oder neutrale **Gewichte** (auch in Form einer Gewichtsmatrix!) dargestellt werden.
3. Ein Gewicht wird mit dem jeweiligen **Output** multipliziert. Das Ergebnis wird als **Input** einer Unit (nicht zu verwechseln mit einer Input-Unit!) bezeichnet.
4. Sämtliche Inputs eines Neurons werden addiert und ergeben den **Netzinput**.
5. Durch die **Aktivitätsfunktion** erfolgt die Zuordnung des **Netzinputs** einer Einheit zu ihrem **Aktivitätslevel**. Dieser wird wiederum einem Output zugeordnet, der an andere Neuronen oder die Umwelt weitergegeben wird. Im Text werden vier verschiedene Aktivitätsfunktionen besprochen.
6. Man unterscheidet grundsätzlich zwischen einer **Trainings-** und einer **Testphase**. Die Trainingsphase kann man zudem danach unterteilen, ob der korrekte Output vorgegeben wird oder nicht (supervised vs. unsupervised learning). Bei der Testphase stellt sich die Frage, welche Reize vorgegeben werden (Ausgangsreize vs. neue Reize).

7. **Lernen** kann zu **Wissen** führen. Während das Lernen ausschließlich in der Trainingsphase stattfindet, ist das bereits erworbene Wissen sowohl für die Trainings- als auch für die Testphase relevant. Dass sich in neuronalen Netzen die Begriffe Lernen und Wissen über die Gewichte definieren lassen, unterstreicht nochmals deren zentrale Stellung.

Lernregeln: Überblick

Um die Gewichte in der Trainingsphase zu modifizieren, benötigt man eine Lernregel, die angibt, wie die Veränderungen vorgenommen werden sollen.

Definition: Lernregel

Eine Lernregel stellt dabei einen Algorithmus dar, der darüber Auskunft gibt, welche Gewichte des neuronalen Netzes wie stark erhöht oder reduziert werden sollen.

verschiedene Lernregeln

Auf den folgenden Seiten werden folgende Lernregeln besprochen:

- **Hebb-Regel**
- **Delta-Regel**
- **Backpropagation**
- **Competitive Learning**

Hebb-Regel

Donald O. Hebb

Abbildung 8: Donald O. Hebb (1904 - 1985)

Einer der einfachsten Lernregeln mit großer biologischer Plausibilität stammt vom Psychologen **Donald Olding Hebb**. In seinem Buch "The Organization of Behavior" aus dem Jahr 1949 formulierte der Kanadier die hebbsche Lernregel:

"Wenn ein Axon der Zelle A ... Zelle B erregt und wiederholt und dauerhaft zur Erzeugung von Aktionspotentialen in Zelle B beiträgt, so resultiert dies in Wachstumsprozessen oder metabolischen Veränderungen in einer oder in beiden Zellen, die bewirken, dass die Effizienz von Zelle A in bezug auf die Erzeugung eines Aktionspotentials in B größer wird." (S. 62, Übersetzung nach Kandel et al., 1995, S. 700)

Für unser Thema lässt sich die Hebb-Regel folgendermaßen formulieren:

Definition: Hebb-Regel

Das Gewicht zwischen zwei Einheiten wird dann verändert, wenn beide Units gleichzeitig aktiv sind.

Die Größe der Gewichtsveränderung bemisst sich aufgrund dreier Werte:

- dem Aktivitätslevel (bzw. Output) der sendenden Unit a_j
- dem Aktivitätslevel (bzw. Output) der empfangenden Unit a_i
- einem vorher festgelegten, positiven Lernparameter ε

Formel: Hebb-Regel

Als Formel ausgedrückt lautet die Hebb-Regel wie folgt: $\Delta w_{ij} = \varepsilon a_i a_j$

Delta-Regel

Grundlage der Delta-Regel

Die Delta-Regel beruht auf einem Vergleich zwischen dem gewünschten und dem tatsächlich beobachteten Output einer Outputunit i. Als Formel kann dieser Vergleich wie folgt dargestellt werden: δ = a_i(gewünscht) - a_i (beobachtet)

Man kann drei Möglichkeiten unterscheiden:

- **Die beobachtete Aktivität ist zu niedrig**. Um die Aktivität zu steigern werden die Gewichte zwischen den sendenden und der empfangenden Unit erhöht, sofern von den sendenden Einheiten ein positiver Input ausgeht. Die Gewichte zu den sendenden Units mit negativem Input werden hingegen gesenkt.
- **Die beobachtete Aktivität ist zu groß**. Da die Aktivität hier reduziert werden soll, schwächt man alle Verbindungen bei denen der Input positiv ist und stärkt diejenigen Verbindungen mit negativem Input.
- **Die beobachtete und gewünschte Aktivität sind gleich groß (= gewünschtes Resultat)**. In diesem Fall erfolgen keinerlei Veränderungen.

Formel: Delta-Regel

Diese drei Möglichkeiten sind in der Formel zur Delta-Regel enthalten: $\Delta w_{ij} = \varepsilon \delta_i a_j$

Dieser Term stellt außerdem sicher, dass die Größe der Gewichtsveränderung proportional zur Größe des Fehlers ist. Zudem werden durch die Multiplikation mit a_j diejenigen Gewichte zu den sendenden Units stärker verändert, die einen größeren Einfluss auf den Fehlerterm ausüben. Der Lernparameter ε wird vor der Trainingsphase festgelegt und gibt an, wie stark die Gewichtsveränderung pro Lerndurchgang ausfallen soll.

Backpropagation

Einleitung

Lernregel für neuronale Netze mit Hidden-Units

Die bisher dargestellten Lernregeln funktionieren nur bei neuronalen Netzen ohne Hidden-Units. Auch wenn zahlreiche Probleme ohne die Verwendung einer oder mehrerer Hidden-Schichten gelöst werden können, muss man gelegentlich auf neuronale Netze mit Hidden-Units zurückgreifen. Beispielsweise kann das sogenannte **XOR-Problem** (= exklusives Oder, z. B. die Entscheidung entweder ins Kino oder aber ins Theater zu gehen, nicht aber beides) nicht ohne weiteres ohne Hidden-Units gelöst werden.

Abbildung 9: Paul Werbos

Das Backpropagation Verfahren stellt eine Rechenvorschrift dar, mit der die Gewichte zu den Hidden-Units modifiziert werden können. Entwickelt wurde das Verfahren bereits in den 70er Jahren (u. a. von **Paul Werbos**, 1974), allerdings geriet es zunächst für über ein Jahrzehnt in Vergessenheit.

Abbildung 10: David Rumelhart

Besonders bekannt wurde der Backpropagation Ansatz von **Rumelhart, Hinton und Williams (1986)**, deren Verfahren eine Verallgemeinerung der Delta-Regel darstellt.

Heutige neuronale Netze, die konkrete Anwendungsprobleme lösen sollen, greifen typischerweise auf das Backpropagation-Verfahren zurück. Ein Problem im Zusammenhang mit diesem Ansatz ist die **fragwürdige biologische Plausibilität** bei der Erklärung der Funktionsweise des menschlichen Gehirns (siehe hierzu Probleme neuronaler Netze).

Backpropagation

Problemstellung und Algorithmus

Problem: Fehlerterme der Hidden-Units unbekannt

Bei Netzen mit Hidden-Units steht man vor dem Problem, dass man keinen direkten Fehler für Neuronen der Hidden-Schicht bestimmen kann. Dieses Problem entsteht, weil man nur für die Output-Schicht, nicht aber für die Hidden-Schicht den gewünschten Output kennt und hieraus (zusammen mit dem tatsächlichen Output) einen Fehlerterm ermitteln kann.

Lösung: Gewichtsmodifikation durch Rückwärtsausbreitung der Fehlerterme

Um dennoch eine Modifikation der Gewichte über die entstehenden Fehlerterme vornehmen zu können wird in der Trainingsphase jede Gewichtsveränderung in drei Schritte unterteilt:

1. **Forward-Pass**: Zunächst werden - wie in der Trainings- und der Testphase üblich - den Input-Neuronen Reize präsentiert und sodann der Output des neuronalen Netzes berechnet.
2. **Fehlerbestimmung**: In einem zweiten Schritt erfolgt die Fehlerbestimmung für die Output-Units, indem die gewünschten Output-Werte mit den im forward-pass tatsächlich ermittelten Werten verglichen werden. Wenn die Fehler eine vorgegebene Güteschwelle überschreiten, folgt der dritte Schritt. Sind die Fehler klein genug und überschreiten die Güteschwelle nicht, kann die Trainingsphase abgebrochen werden.
3. **Backward-Pass**: Der dritte Schritt ist der innovative Kern des Backpropagation Verfahrens. Die Fehlerterme breiten sich nun in entgegengesetzter Richtung bis zur Input-Schicht aus. Mit Hilfe dieser Fehlerterme werden nun nach und nach (d.h. zunächst zwischen Output und letzter Hidden-Schicht, dann zwischen letzter und vorletzter Hidden-Schicht usw.) die Gewichte des Netzes modifiziert so dass die Fehlerterme kleiner werden.

Backpropagation

Gewichtsanpassung

Problem: Fehlerminimierung beim Backward-Pass

Wie kann man nun im **Backward-Pass** bei Backpropagation die Gewichte so anpassen, dass der resultierende Gesamtfehler möglichst klein ausfällt?

Eine Möglichkeit bestünde darin, zu allen möglichen Kombinationen von Gewichten im neuronalen Netz einen Gesamtfehlerterm (**F**) zu bestimmen. Die Gewichtskombination (**w**) mit dem kleinsten Gesamtfehlerterm ($\mathbf{F_{min}}$) wäre die optimale Lösung ($\mathbf{w_{min}}$), ein **absolutes Minimum** hinsichtlich des Fehlers.

Was im zweidimensionalen Raum (siehe Abbildung 11), sprich mit nur einem einzigen Gewicht (!) noch vergleichsweise einfach wäre, gestaltet sich im **n-dimensionalen Raum** (d.h. bei n-1 Gewichten) ungleich schwerer. Hier würde der Fehlerterm keiner Kurve, sondern einer so genannten **Hyperebene** entsprechen. Der Rechenaufwand zur Bestimmung der gesamten Hyperebene, um ein absolutes Minimum innerhalb dieses Raumes zu finden, wäre viel zu groß.

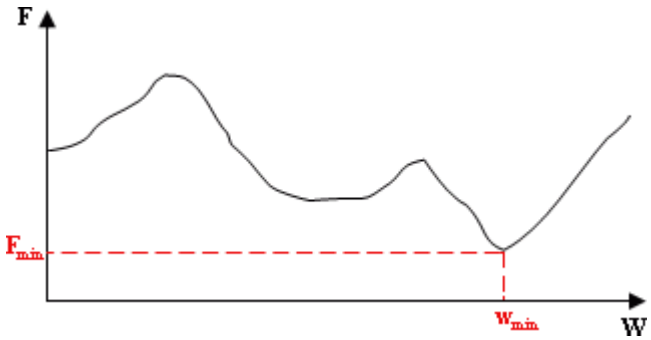


Abbildung 11: Zweidimensionales Liniendiagramm mit Gradientenabstiegskurve. Auf der Abszisse (x-Achse) ist das Gewicht (W) abgetragen, auf der Ordinate (y-Achse) der Fehlerterm (F). In rot: das Gewicht (w) mit der optimalen Lösung (absolutes Minimum hinsichtlich des Fehlers).

Lösung: Gradientenabstiegsverfahren

Stattdessen werden die Gewichte mit dem **Gradientenabstiegsverfahren** modifiziert. Bei diesem Verfahren muss man nicht die gesamte Hyperebene kennen.

Backpropagation

Gradientenabstiegsverfahren

Start: zufällig gewählte Gewichtskombination

Das **Gradientenabstiegsverfahren** startet mit einer zufällig gewählten Gewichtskombination. Für diese wird der **Gradient** bestimmt und um eine vorgegebene Länge - der Lernrate - hinabgestiegen, sprich die Gewichte werden entsprechend verändert (siehe Abbildung 12). Der Gradient kann dabei definiert werden als eine Funktion eines Skalarfeldes, welche die Änderungsrate und Richtung der größten Änderung in Form eines Vektorfeldes angibt.

Ende: Erreichen eines lokalen Minimums oder der maximalen Anzahl von Zyklen

Für die neu erhaltene Gewichtskombination wird wiederum der Gradient bestimmt und abermals eine Modifikation der Gewichte vorgenommen. Dieses Verfahren wiederholt sich solange, bis ein **lokales Minimum** (bzw. globales Minimum) erreicht ist (siehe Abbildung 12) oder eine zuvor festgelegte maximale Anzahl von Wiederholungen erreicht worden ist.

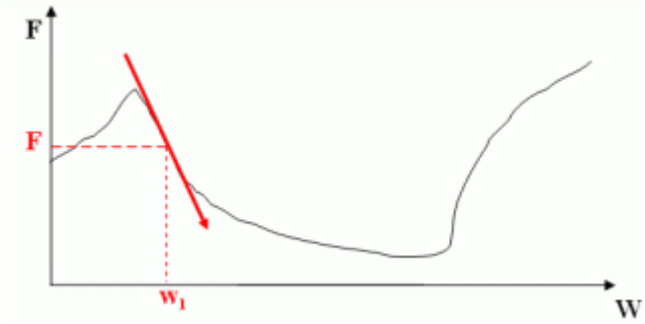


Abbildung 12: Zweidimensionales, animiertes Liniendiagramm mit Gradientenabstiegskurve. Auf der Abszisse (x-Achse) ist das Gewicht (W) abgetragen, auf der Ordinate (y-Achse) der Fehlerterm (F). Der rote Pfeil markiert den Gradienten, der blaue gibt die Richtung und Stärke der Positionsveränderung der Gewichte an. Er bildet sich mit Hilfe des Gradienten und der Lernrate. In diesem Beispiel findet das Gradientenabstiegsverfahren nach 3 Zyklen das globale Minimum.

Backpropagation

Probleme des Gradientenabstiegsverfahrens

Problem: lediglich Kenntnis der lokalen Umgebung

Leider steht dem Vorteil des geringeren Rechenaufwands der Nachteil gegenüber, dass dem Verfahren lediglich die lokale Umgebung (der Gradient) bekannt ist. Daraus resultieren folgende Probleme:

- **Lokale Minima**: Man weiß beim Gradientenabstiegsverfahren nie, ob man nach der Durchführung ein lokales oder absolutes Minimum gefunden hat. Dieses Problem tritt verstärkt bei höherer **Dimension des Netzes** (= Anzahl der Verbindungen zwischen den Neuronen) auf. Eine höhere Dimension führt dazu, dass die Hyperebene des Fehlerterms stärker zerklüftet und sich somit die Anzahl der lokalen Minima erhöht.
- **Flache Plateaus**: Im Grunde genommen besteht hier genau das umgekehrte Problem. Statt einer (zu) starken Zerklüftung existieren - zumindest in Teilen der Hyperebene - kaum "Berge und Täler", sondern ein relativ flaches "Plateau". Dadurch wird der Gradient beim Gradientenabstiegsverfahren sehr klein, so dass das nächste "Tal" gar nicht mehr erreicht wird. Das Verfahren stagniert.
- **Verlassen guter Minima**: Auch dieses Problem lässt sich als Gegenstück zum Problem lokaler Minima auffassen. Statt ein globales Minimum gar nicht zu erreichen, wird hier das globale Minimum "übersprungen". Dies passiert vornehmlich dann, wenn solch ein "tiefes Tal" mit relativ geringer Ausdehnung in der Hyperebene liegt. In der Folge findet das Gradientenabstiegsverfahren nur ein lokales Minimum (s. o.).
- **Direkte Oszillation**: Im Falle der direkten Oszillation entdeckt das Gradientenabstiegsverfahren weder ein globales noch ein lokales Minimum. Dies passiert dann, wenn der Gradient von einem "Abhang" eines Tals zum gegenüberliegenden "Abhang" springt und von dort wieder zur selben Stelle zurück. In diesem Fall sind die Beträge der Gradienten gleich, lediglich die Vorzeichen wechseln hin und her. Dem Gradientenabstiegsverfahren gelingt es nicht in die "Tiefe der Hyperebene hervorzustoßen". Das Verfahren oszilliert.
- **Indirekte Oszillation**: Im Gegensatz zur direkten Oszillation kann es auch passieren, dass das Verfahren nicht direkt zurückspringt, sondern mehrere Schritte benötigt, um wieder zum Ausgangspunkt zurückkehren.

Backpropagation

Lösungen zu den Problemen des Gradientenabstiegsverfahrens

Lösung: Initialisierung der Gewichte und Lernrate verändern

Es stehen eine Vielzahl Lösungsmöglichkeiten bei unbefriedigendem Lernerfolg des neuronalen Netzes zur Verfügung. Zwei häufig angewandte Methoden sind:

1. Die **Veränderung der Initialisierung der Gewichte**: Zuerst versucht man die Initialisierung der Gewichte zu verändern, um den Lernerfolg zu verbessern. Dabei kann man zwischen zwei Aspekten unterscheiden:
   - **Startpunkt des Gradientenabstiegsverfahrens**: Der Startpunkt hat einen zentralen Einfluss darauf, welche Werte die Gewichte im Verlauf des Verfahrens annehmen und ob sich schlussendlich ein lokales oder globales Minimum findet.
   - **Art der Initialisierung**: Ebenso ist die Art der Initialisierung für das Auffinden eines lokalen bzw. globalen Minimums von Bedeutung. Damit ist u. a. die Frage gemeint, wie unterschiedlich die Werte der einzelnen Gewichte gewählt wurden. Beispielsweise können diese nur sehr gering um Null herum streuen oder aber sehr stark.
2. Die **Veränderung der Lernrate**: Wenn die Neu-Initialisierung der Gewichte nicht erfolgreich war, sollte man versuchen die Lernrate zu verändern. Grundsätzlich gilt, dass es keine optimale Lernrate für alle Arten neuronaler Netze gibt. Es lassen sich hier zwei Möglichkeiten voneinander unterscheiden:
   - **Lernrate erhöhen**: Eine höhere Lernrate bewirkt, dass die Sprünge in der Hyperebene größer werden. Dies kann folgende **Vorteile** nach sich ziehen:
     - flache Plateaus werden schneller durchlaufen bzw. überwunden
     - vom Startpunkt weit entfernte Minima werden schneller erreicht

     Mit einer höheren Lernrate können sich aber auch **Nachteile** ergeben:
     - gute (globale) Minima werden häufiger übersprungen
     - die Gefahr der Oszillation steigt an
   - **Lernrate reduzieren**: Eine niedrigere Lernrate führt dazu, dass kleinere Schritte beim Gradientenabstiegsverfahren vorgenommen werden. Dies führt zu folgenden **Vorteilen**:
     - gute (globale) Minima werden nicht mehr so leicht übersprungen
     - die Gefahr der Oszillation sinkt
     - komplexe Daten sowie eine große Datendichte werden besser bewältigt

     Den Vorteilen stehen jedoch auch **Nachteile** gegenüber:
     - die Trainingszeit bis zum Erreichen eines Minimums kann inakzeptabel groß werden

- flache Plateaus werden langsamer durchlaufen bzw. nicht mehr überwunden
- lokale Minima werden häufiger nicht mehr verlassen

Eine mögliche Strategie ist die schrittweise Verringerung einer anfangs hohen Lernrate im Laufe des Verfahrens. Als Empfehlung zur Lernratenveränderung wird beispielsweise von der Neuro-Fuzzy-AG der Universität Münster vorgeschlagen, mit einer Lernrate von 0.7 zu beginnen und diese dann *bei unbefriedigendem Lernerfolg* schrittweise um 0.1 zu verringern.

Competitive Learning

unsupervised learning

Im Gegensatz zum Backpropagation-Verfahren kommt das kompetitive Lernen ohne Vorgabe eines korrekten, externen Output-Reizes aus, an dem die Gewichte adjustiert werden. Stattdessen ist das Lernen nach dieser Lernregel **unsupervised**, da das neuronale Netz hier anhand der Ähnlichkeit der präsentierten Input-Reize eine Kategorisierung vornimmt.

Prinzip: "The Winner takes it all."

Beim Competitive Learning können drei verschiedene Phasen unterschieden werden:

1. **Erregung** (excitation): Zunächst wird wie gewohnt für alle Output-Units der Netzinput durch folgende Formel bestimmt: $netinput_i = \sum_j a_j w_{ij}$
2. **Wettbewerb** (Competition): Anschließend werden die Netzinputs sämtlicher Output-Units miteinander verglichen. Diejenige Unit mit dem höchsten Netzinput ist der Gewinner.
3. **Adjustierung der Gewichte** (weight adjustment): Im letzten Schritt werden die Gewichte verändert und zwar bei allen Verbindungen, die zur Gewinner-Unit führen. Alle anderen Gewichte werden nicht verändert ("The Winner takes it all." - Prinzip). Die Gewichte zum Gewinner werden so modifiziert, dass sie dem Input ähnlicher gemacht werden ($a_j - w_{ij}$). Auch dies lässt sich wiederum als Formel darstellen, wobei ε wie gewohnt ein zuvor festgelegter Lernparameter ist: $\Delta w_{ij} = \varepsilon (a_j - w_{ij})$

Übersichtstabelle: Lernregeln

Tabelle 1: Tabellarische Darstellung der Kernkonzepte, Eigenschaften sowie Vor- und Nachteile der Lernregeln: Hebb-Regel, Delta-Regel, Backpropagation und Competitive learning.

	Hebb-Regel	**Delta-Regel**	**Backpropagation**	**Competitive Learning**
Kernkozept	Gleichzeitige Aktivierung	Vergleich: gewünscht vs. beobachtet; Gradientenverfahren	Backward-pass; Gradientenverfahren	"The winner takes it all."
Art der Lernregel	Als supervised, unsupervised und reinforcement learning möglich	Supervised learning	Supervised learning	Unsupervised learning
Biologische Plausibilität?	Teilweise	Eher nicht	Eher nicht	Teilweise
Netztypen, die auf diese Lernregel zurückgreifen (u.a.)	Pattern Associator; Auto Associator	Pattern Associator; Auto Associator	Simple Recurrent Networks, Jordan Netze	Kompetitive Netze; konzeptuell auch in Kohonennetzen
Vorteile	Einfachheit, biologische Plausibilität	Einfachheit, relativ leicht zu implementieren	Auch bei Netzen mit Hidden-Units einsetzbar; größere Mächtigkeit im Vergleich zur Delta-Regel	Unsupervised learning; biologische Plausibilität
Nachteile	In der "klassischen" Form: Überlaufen der Werte der Gewichte und geringe Mächtigkeit des Systems	Nicht bei Netzen mit Hidden-Units einsetzbar; fragwürdige biologische Plausibilität; geringe Mächtigkeit des Systems	Fragwürdige biologische Plausibilität; lokale Minima	Einzelne Output-Unit kann alle Inputmuster "an sich reißen" --> keine Kategorisierung mehr

Netztypen: Überblick

Problem: Klassifikation neuronaler Netze

Neuronale Netze lassen sich nach unterschiedlichen Gesichtspunkten klassifizieren. Eine Möglichkeit verschiedene Netze voneinander zu unterscheiden sind die Lernregeln, die diese Netze verwenden. Dabei ist aber zumeist keine klare Zuordnung zwischen Netztyp und Lernregel möglich, da einige Netztypen auf dieselbe Lernregel zurückgreifen, während andere sich mit verschiedenen Lernregeln realisieren lassen.

Andere Aspekte der Klassifikation betreffen die Frage, ob:

- Hidden-Units vorhanden sind oder nicht
- die Trainingsphase supervised oder unsupervised learning ist
- Rückkopplungen von Neuronen zu anderen Neuronen derselben oder einer vorangegangenen Schicht existieren
- welchem Anwendungszweck ein Netz dient (z.B. Vorhersage, Klassifikation und Erkennen von Mustern, assoziative Speicherung von Informationen oder Optimierung)

verschiedene Netztypen

Jedoch führen auch diese Aspekte zumeist nicht zu einer klaren Zuordnung zu verschiedenen Netztypen. Auf den kommenden Seiten werden folgende Netztypen vorgestellt:

- **Pattern Associator**
- **Rekurrente Netze bzw. Simple Rekurrent Networks (SRNs)**
- **Kompetitive Netze**
- **Kohonennetze bzw. Selforganizing Maps (SOM's)**

Pattern Associator

Definition: Pattern Associator

Der Pattern Associator ist ein Netztyp, der Muster erkennen kann, die er zuvor gelernt hat. Dabei lernt das neuronale Netz Assoziationen zwischen verschiedenen Reizpaaren zu bilden (Stichwort "Klassische Konditionierung").

Beim Pattern Associator existieren keine Hidden-Units, d. h. das neuronale Netz besteht hier lediglich aus zwei Schichten, einer Input-Schicht und einer Output-Schicht (siehe Abbildung 13).

Die Trainingsphase - das Lernen verschiedener Reize - kann entweder mit der Hebb-Regel vorgenommen werden oder aber mit der Delta-Regel.

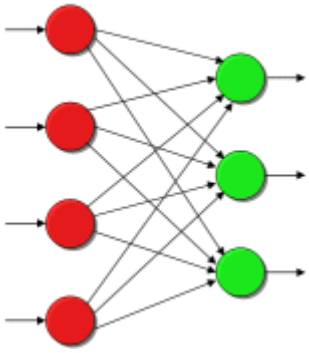


Abbildung 13: Schematische Darstellung eines Pattern Associators mit 4 Input- (rot) und 3 Output-Units (grün).

Eigenschaften: Pattern Associator

Eigenschaften

Der Pattern Associator verfügt über eine Reihe wünschenswerter **Eigenschaften** (vgl. auch das Kapitel **Eigenschaften neuronaler Netze**):

- **Generalisierung**: Wie bei der klassischen Konditionierung erfolgt eine Generalisierung (aber auch eine Diskrimination) verschiedener Reize. Dabei werden ähnliche Reize zur selben Reizgruppe kategorisiert. Beispielsweise kann man verschiedene Hunderassen als Patterns zur Gruppe "Hunde" klassifizieren. Ein Problem bei der Generalisierung kann die entstehende "Übergeneralisierung" sein. Dabei werden ähnliche Reize, die jedoch unterschiedlichen Kategorien angehören, einer einzigen Kategorie zugeordnet. Zum Beispiel ist es falsch, Forellen, Delphine, Barsche und Heringe der Kategorie "Fisch" zuzuordnen.
- **Toleranz gegenüber internen Schäden**: Hiermit ist gemeint, dass trotz innerer Schäden des neuronalen Netzes (z. B. durch das Absterben einzelner Neuronen oder Verbindungen zwischen Neuronen) dieses oftmals dennoch den richtigen Output produziert.
- **Toleranz gegenüber externen Schäden**: Auch bei unvollständigem oder fehlerhaftem Input gelingt es dem neuronalen Netz, das Muster zu identifizieren (z. B. kann man ein Gesicht meistens auch erkennen selbst wenn die Nase verdeckt ist).
- **Output der zentralen Tendenz bzw. des Prototypen der Kategorie**: Bei mehreren gelernten Inputmustern bildet der Pattern Associator einen Prototypen der verschiedenen Muster aus. Eine Eigenschaft, die auch für Menschen typisch ist.

Rekurrente Netze

Definition: Rekurrente Netze

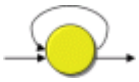
Rekurrente Netze sind dadurch gekennzeichnet, dass **Rückkopplungen** von Neuronen einer Schicht zu anderen Neuronen derselben oder einer vorangegangenen Schicht existieren. Damit sollen zumeist zeitlich codierte Informationen in den Daten entdeckt werden.

Existieren solche Rückkopplungen nicht (wie z. B. beim Pattern Associator oder den noch vorzustellenden Kompetitiven Netzen und Kohonennetzen), spricht man auch von Feedforward-Netzen.

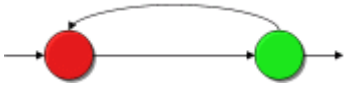
Unterteilung rekurrenter Netze

Rekurrente Netze lassen sich unterteilen in neuronale Netze mit:

- **direkten Rückkopplungen** (direct feedback): Hier existieren Verbindungen vom Output zum Input derselben Unit zurück. Das bedeutet, dass der Aktivitätslevel bzw. der Output der Einheit zu einem Input der gleichen Einheit wird.

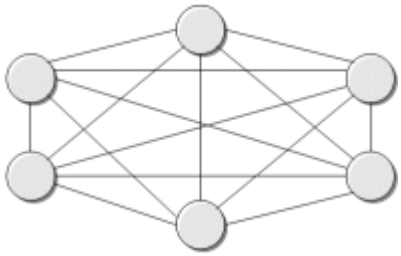


- **indirekten Rückkopplungen** (indirect feedback): In diesem Fall wird die Aktivität an vorangegangene Schichten des neuronalen Netzes zurückgesandt.



- **seitlichen Rückkopplungen** (lateral feedback): Hier erfolgt die Rückmeldung der Informationen einer Unit an Neuronen, die sich in derselben Schicht befinden. Ein Beispiel für solche seitlichen Rückkopplungen sind die Horizontalzellen im menschlichen Auge.

- **vollständigen Verbindungen**: Diese Netze besitzen Verbindungen zwischen sämtlichen Neuronen. (In der Abbildung unterhalb sind jedoch keine direkte Rückkopplungen eingezeichnet.)



Im Folgenden werden die so genannten **"Simple Recurrent Networks" (SRNs)** - rekurrente Netze mit indirekten Rückkopplungen - vorgestellt.

Rekurrente Netze

Simple Recurrent Networks (SRNs)

Kontext-Einheiten

SRNs zeichnen sich durch ihre **Kontext-Einheiten** (diese sind in Abbildung 14 in blau dargestellt) aus. Kontext-Einheiten sind Neuronen, die sich quasi auf gleicher Ebene wie die der Input-Schicht befinden und von Units der Hidden-Schicht Informationen erhalten. Diese Informationen werden dann dort "verarbeitet" (= der Input wird wie gewohnt in einen Output umgewandelt) und sodann **um einen Schritt verzögert** an die Hidden-Units der betreffenden Schicht zurückgesendet.

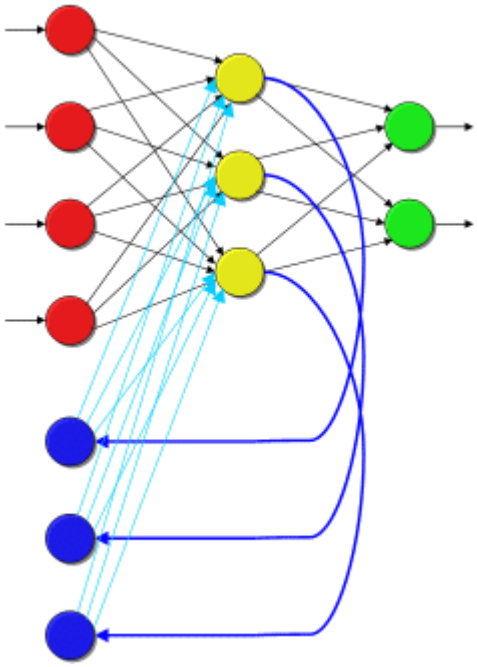


Abbildung 14: Schematische Darstellung eines Simple Recurrent Network (SRN) mit 3 Kontext-Units (in blau).

Die Anzahl der Kontext-Einheiten ist genauso groß wie die Anzahl der Einheiten der Hidden-Schicht, mit welcher die Kontext-Units verbunden sind. Ferner erhält jede Kontext-Einheit nur von genau einer Hidden-Unit Informationen. Deswegen wird in den Kontext-Units auch der Input und nicht der Netzinput einem Aktivitätslevel zugeordnet. Zudem werden **sämtliche Gewichte von den Hidden-Units zu den Kontext-Einheiten permanent auf +1 fixiert**. Dadurch erhalten die Kontext-Units in jedem Durchlauf eine **exakte Kopie** des Outputs der Hidden-Schicht, mit der sie verknüpft sind.

Einen Schritt später geben die Kontext-Einheiten diesen Output dann wieder zurück an die Hidden-Units, wobei jede Kontext-Unit mit sämtlichen Neuronen der betreffenden Hidden-Schicht verbunden ist. Die Gewichte dieser Verbindungen werden nicht fixiert, sondern können während der Lernphase wie alle anderen Gewichte des Netzes auch mit Hilfe (für rekurrente Verbindungen modifizierter Varianten) des **Backpropagation-Verfahrens** angepasst werden.

Durch die Kontext-Einheiten besitzt ein Simple Recurrent Network Teilinformationen aus **sämtlichen** (!) vorangegangen Zeitpunkten bzw. Durchgängen (t). Dies liegt daran, weil zu einem beliebigen Durchgang die Kontext-Units eine Kopie der Aktivität der Hidden-Units aus dem vorangegangenen Durchgang (t-1) besitzen. Zu diesem vorangegangen Zeitpunkt (t-1) haben die Kontext-Einheiten aber wiederum eine Kopie des vorangegangen Zeitpunktes (t-2) besessen und den Hidden-Units in modifizierter Form zur Verfügung gestellt usw. (t-3, t-4 ... bis zum ersten Durchgang). Dadurch enthalten die Kontext-Units **indirekt** Teilinformationen über **alle** vorangegangen Zeitpunkte.

Kontext-Einheiten mit den dazugehörigen modifizierbaren Gewichten kann man auch als **dynamisches Gedächtnis** des neuronalen Netzes betrachten. Identischer Input im Netz wird durch die Existenz der Kontext-Units **kontextabhängig modifiziert** (daher der Name).

Lernen arbiträrer

Abbildungen

weitere rekurrente Netze

Im Gegensatz zu anderen Netzlösungen mit Rückkopplungen besitzen SRNs den Vorteil, dass sie sehr flexibel **arbiträre** (=willkürliche) **Sequenzen** lernen können.

Stichpunktartig seien zwei andere rekurrente Netze genannt:

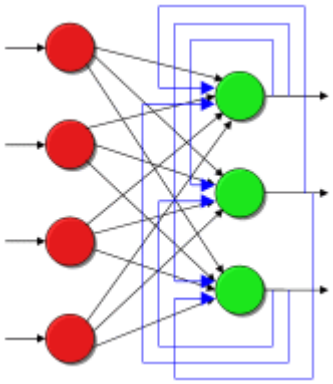


Abbildung 15: Schematische Darstellung eines Autoassociators. In blau: rekurrente Verbindungen der Output-Units.

- **Autoassociator**: Der Autoassociator weist sehr große Ähnlichkeiten zum Pattern Associator auf. Im Gegensatz zu diesem besitzt er jedoch modifizierbare, rekurrente Verbindungen von sämtlichen Output-Neuronen zu allen anderen Output-Units, ausgenommen sich selbst (siehe Abbildung 15).

- **Jordan-Netze**: Diese sind sehr ähnlich aufgebaut wie SRNs mit dem einzigen Unterschied, dass hier die Aktivität des Outputs (und nicht die der Hidden-Schicht) an die Kontext-Einheiten zurückgemeldet wird. Zudem sind für die Kontext-Einheiten direkte Rückkopplungen mit einer zuvor festgelegten, nicht veränderbaren Stärke vorgesehen. In den meisten Anwendungen wird jedoch auf diese weiteren rekurrenten Verbindungen verzichtet.

Rekurrente Netze

Anwendungen

Anwendungen rekurrenter Netze

Folgende **Anwendungen** können durch rekurrente Netze realisiert werden:

- **Prognosen über die Zukunft treffen**: Grundsätzlich können rekurrente Netze Vorhersagen über die Zukunft machen, z. B. den nächsten Input einer Sequenz prognostizieren. Das Treffen von Vorhersagen spielt eine zentrale Rolle bei menschlichen Kognitionen, insbesondere beim Lernen.
- **Simulation von menschlichen Verhaltenssequenzen**: Rekurrente Netze können menschliche Verhaltensweisen simulieren, z. B. die Steuerung der Motorik, vor allem der Sprache. Hierbei müssen die artikulatorischen Bewegungen genau stimmen, da sonst die Phonemfolgen nicht mehr korrekt sind und das Gesagte somit unverständlich wird.
- **Attraktorennetze**: Diese sollen auf der folgenden Seite genauer erörtert werden.

Rekurrente Netze

Attraktorennetze

Definition: Attraktorennetze

Attraktorennetze stellen u.a. ein Anwendungsgebiet von rekurrenten Netzen dar. Dabei erhält das Netz einen Input und arbeitet dann in Zyklen weiter, bis ein **stabiler (Output-)Zustand** erreicht worden ist.

Attraktorennetze besitzen eine bestimmte Zahl solcher stabilen Zustände (so genannte **Attraktoren**), auf die sich der Output im Laufe mehrerer Zyklen zu bewegt. Welcher Attraktor im Laufe der Zeit erreicht wird, hängt davon ab, in welches "Einzugsgebiet" (auch "Becken" oder "Schale" genannt) der einzelnen Attraktoren der Input hineinfällt. Man spricht in diesem Zusammenhang auch von **Bassins der Attraktion.**

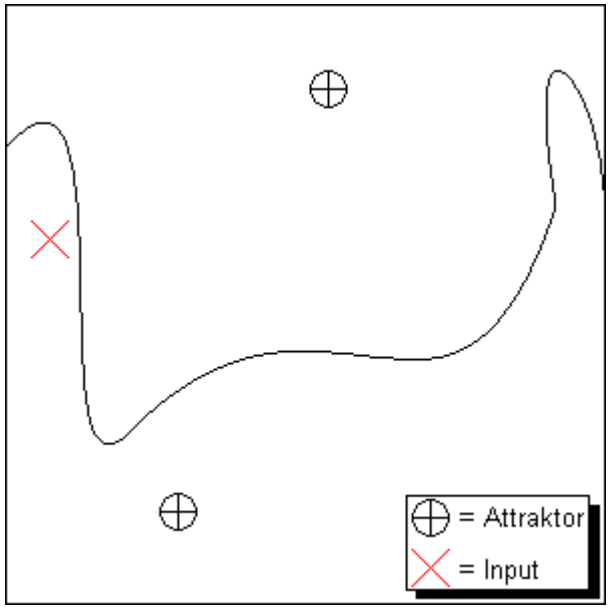


Abbildung 16: Schematische Darstellung der Funktionsweise eines Attraktorennetzes. Die zwei Attraktorenbassins werden durch eine non-lineare Trennlinie voneinander abgetrennt.

Attraktorenbassins

Je weiter der Input in einem Bassin vom jeweiligen Attraktor entfernt ist, desto mehr Zyklen sind notwendig bis dieser erreicht ist. Dass ein Input exakt auf eine Grenze zwischen zwei Bassins fällt und keine eindeutige Zuordnung vorgenommen werden kann, welchem Attraktor der Input zugeordnet wird, kann beispielsweise durch Hinzunahme eines kleinen Zufallsterms verhindert werden. Dies wäre insofern biologisch plausibel, da vermutlich jedes zufällige Rauschen den Input von der Bassingrenze "kickt" und in eine der beiden Schalen wirft.

Rekurrente Netze

Eigenschaften von Attraktorennetzen

Eigenschaften von Attraktorennetzen

Folgende Eigenschaften machen Attraktorennetze nützlich für die Simulation von kognitiven Prozessen:

- **Simulation von Latenzzeiten**: Die Reaktionszeit eines Attraktorennetzes kann durch die Anzahl an Zyklen operationalisiert werden, die erforderlich sind, um zu einem stabilen Zustand zu gelangen. Die Zeit ist dabei abhängig von der Gestalt der Schale und der Position innerhalb des Beckens, wo der Anfangsinput ursprünglich "landete".
- **Resistenz gegenüber fehlerhaften Input**: Im Attraktionsbassin ist die exakte Position eines Inputs bezüglich des Erreichens eines Attraktors nicht ausschlaggebend.
- **Lernen arbiträrer Abbildungen**: Dies ist vermutlich die wichtigste Eigenschaft von Attraktorennetzen, da viele kognitive Aufgaben aber auch statistische Auswertungen (z. B. die Diskriminanzanalyse) solche willkürlichen Abbildungen (mappings) betreffen. Das Netz kann beliebige Trennlinien zwischen Attraktorenbassins erlernen. Das Erlernen von arbiträren Abbildungen - man könnte auch sagen: das Clustern eines Inputraumes - wird auch von Kompetitiven Netzen und Kohonennetzen vorgenommen.

Kompetitive Netze

Definition: Kompetitive Netze

Kompetitive Netze sind neuronale Netze, typischerweise mit einer Input- und einer Outputschicht. Wie beim Pattern Associator und Kohonennetzen sind hier keine Hidden-Units vorhanden, wenngleich die Verwendung von Hidden-Units mit der von kompetitiven Netzen benutzten kompetitiven Lernregel möglich ist. Damit erfolgt die Trainingsphase kompetitiver Netze in 3 Schritten:

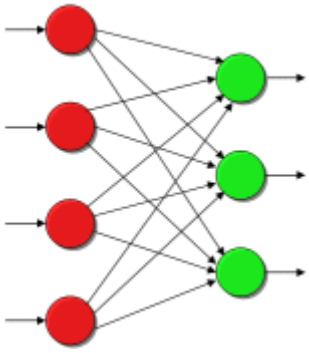


Abbildung 17: Schematische Darstellung eines kompetitiven Netzes mit 4 Input- (rot) und 3 Output-Units (grün). Vergleichen Sie bitte die Abbildung mit der zum Pattern Associator!

1. **Erregung**
2. **Wettbewerb**
3. **Adjustierung der Gewichte**

unsupervised learning

Im Gegensatz zu den bisher vorgestellten Netzen sind kompetitive Netze **unsupervised**, d. h. sie kommen ohne Vorgabe eines korrekten, externen Output-Reizes aus.

Begrenzung der Gewichtsvektoren

Bei kompetitiven Netzen kann es vorkommen, dass die Gewichte zu einer einzigen oder einigen wenigen Outputunits so groß werden, dass diese den Wettbewerb unabhängig vom Muster des Inputs gewinnen. In diesem Fall leistet ein kompetitives Netz keine "sinnvolle" Kategorienbildung mehr. Um dies zu verhindern kann man z. B. die Größe (d.h. den Absolutbetrag) aller einzelnen Gewichtsvektoren einer Schicht (Gewichtsvektor = alle Gewichte einer bestimmten Outputunit) auf einen konstanten Wert festlegen.

Anwendungen

Anwendungen kompetitiver Netze

Kompetitive Netze eignen sich zu folgenden **Anwendungen**:

- **Filtern von Redundanzen** und **Alternative zur Faktorenanalyse** (z. B. Erzeugung von Outputmustern, die weniger korreliert sind als der Input)
- **vorgeschaltetes Netz für andere Netztypen** (Beispielsweise kann man ein kompetitives Netz vor einen Pattern Associator setzen, damit die einzelnen "Patterns" weniger korreliert sind. Dadurch wird die Arbeitsweise des Pattern Associators verbessert.)
- **Musterklassifikation** (ähnlich wie beim Pattern Associator, jedoch im Gegensatz zu diesem als "unsupervised learning") -> z. B. Buchstabenerkennung
- Im Rahmen der Musterklassifikation: **Korreliertes Lernen** (correlated teaching). Beispiel: Nehmen wir an, es sollen die Inputmuster "AA", "BA", "SB" und "EB" mit 2 Outputunits kategorisiert werden. Eine Möglichkeit für das neuronale Netz bestünde darin, dass die zweite Stelle ("A" oder "B") festlegt, zu welcher Outputunit ein Inputmuster zugeordnet wird. Damit würde nun "A" und "B", sowie "S" und "E" zusammen geclustert, obwohl diese Buchstaben sich relativ unähnlich sind (=eine zum Teil "unnatürliche Klassifikation" durch korreliertes Lernen). Das korrelierte Lernen kann unter anderem bei linear nicht separierbaren Problemen eingesetzt werden.

Kohonennetze

Was sind Kohonennetze?

Definition: Kohonennetze

Kohonennetze (auch als Kohonen Feature Maps oder Self-Organizing Maps (SOMs) bezeichnet) stellen eine Erweiterung kompetitiver Netze dar. Auch bei ihnen wird der korrekte Output nicht festgelegt und dem neuronalen Netz zurückgemeldet, sondern sie agieren ohne einen externen Lehrer (=unsupervised learning).

unsupervised learning

Kohonennetze können in selbstorganisierender Weise lernen, Karten (maps) von einem Inputraum zu erstellen (man kann auch sagen: Kohonennetze clustern den Inputraum).

Abbildung 18: Teuvo Kohonen (1934 -)

Der Begriff "Kohonen" geht auf den finnischen Ingenieur **Teuvo Kohonen** (1982) zurück, der ein sehr bekanntes Kohonennetz konzipierte. Allerdings wurden bereits zuvor durch **Stephen Grossberg** (1972) und **Christoph von der Malsburg** (1973) Ansätze in dieser Richtung entwickelt.

Abbildung 19: Christoph von der Malsburg

Ein wesentlicher Vorteil von Kohonennetzen im Vergleich zu konventionellen neuronalen Netzen liegt in der biologischen Plausibilität, da Menschen vermutlich Probleme in der Regel ohne externen Lehrer lösen.

Beispiele für selbstorganisiertes Lernen im menschlichen Gehirn finden sich beim Sehsinn im Okzipitallappen (siehe hierzu auch die Arbeiten von **David Hubel und Torsten Wiesel** (1962)) oder aber im somatosensorischen und motorischen Kortex.

Bei diesen sind die topographischen Anordnungen nicht vollständig genetisch festgelegt, sondern sensorische Erfahrung ist eine notwendige Bedingung, um solche Karten auszubilden.

Kohonennetze / Selforganizing Maps

Aufbau

Aufbau: Kohonennetze

Kohonennetze bestehen üblicherweise aus zwei Schichten von Neuronen, einer Input- und einer Output-Schicht. Es existieren wie beim Pattern Associator und den kompetitiven Netzen keine Hidden-Units, wenngleich diese wie bei den kompetitiven Netzen grundsätzlich auch hier implementiert werden können.

Von jedem Input-Neuron führen Verbindungen zu sämtlichen Outputneuronen. Häufig ist bei Kohonennetzen die Outputschicht 2-dimensional aufgebaut (siehe Abbildung 20). Dabei spielt die Distanz zwischen den einzelnen Output-Neuronen eine wichtige Rolle.

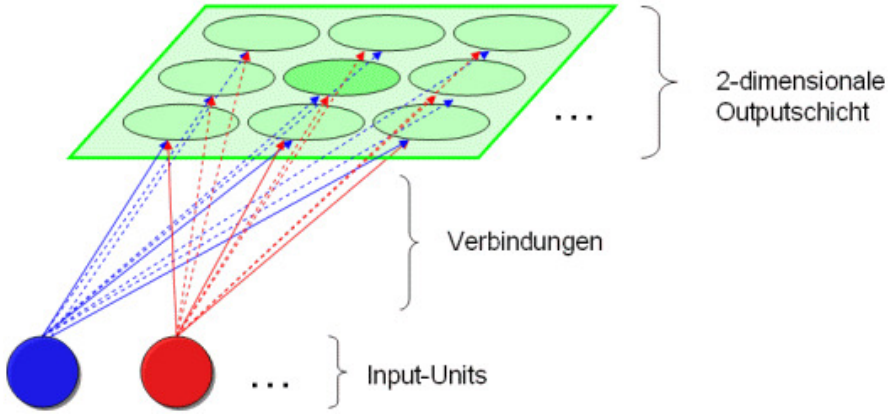


Abbildung 20: Schematische Darstellung eines 2-dimensionalen Kohonennetzes. In blau und rot: 2 Input-Units mit ihren Verbindungen zu sämtlichen Output-Units (in grün), die 2-dimensional angeordnet sind. Versuchen Sie bitte die Abbildung mental um 90 Grad nach rechts zu rotieren. Vergleichen Sie nun dieses rotierte Bild mit den schematischen Darstellungen zum Pattern Associator und zu den kompetitiven Netzen!

Kohonennetze / Selforganizing Maps

Berechnung

Berechnung: Kohonennetze

Die Berechnung von Kohonennetzen erfolgt in der Trainingsphase in folgenden Schritten:

1. **Startwerte festlegen**: Die Gewichte werden zufällig generiert. Außerdem legt man die Lernkonstante, den Radius, die Nachbarschaftsfunktion sowie die maximale Anzahl an Durchläufen fest (s. auch folgender Abschnitt).
2. **Auswahl eines Inputvektors**: Ein Inputvektor wird ausgewählt bzw. zufällig generiert.
3. **Aktivitätsberechnung und Auswahl**: Die Aktivität der Output-Neuronen wird berechnet. Die Unit mit der maximalen Erregung wird ausgewählt. Das ist jene, deren Gewichtsvektor die geringste Distanz zum Inputmuster aufweist, sprich dem Inputvektor am ähnlichsten ist.
4. **Gewichtsmodifikation**: Die Gewichte zur Gewinner-Unit werden so verändert, dass sie dem Input-Vektor ein wenig ähnlicher werden. Ebenso, allerdings in abgeschwächter Form, die Gewichte aus der Nachbarschaft dieser Unit. Zudem wird der Lernparameter (alpha) reduziert und gegebenenfalls der Radius für die Nachbarschaft eingegrenzt. Danach geht es wieder mit dem ersten Schritt von vorne los.
5. **Abbruch**: Der Abbruch erfolgt, wenn die maximale Anzahl der Durchläufe erreicht ist. Durch die ständigen Gewichtsveränderungen werden einzelne Gewichte bestimmten Inputvektoren immer ähnlicher (siehe Kompetitive Netze). Die Lernkonstante wird kontinuierlich reduziert, wodurch die Korrekturen der Gewichte zu Beginn größer ausfallen als am Ende des Trainings. Somit stellt sich allmählich ein stabiler Zustand ein. Zusätzlich kann noch der Radius der Nachbarschaft reduziert werden, um den Einfluss auf der Gewinner-Einheit benachbarte Units kontinuierlich zu reduzieren.

Kohonennetze / Selforganizing Maps

Wichtige Parameter

Wichtige Parameter: Kohonennetze

Damit das Netz einen stabilen Zustand erreichen kann (also etwas lernt), sind folgende Aspekte wichtig:

- **Zähler**: Anzahl der Durchläufe (also, wie viele Inputvektoren dem Netz dargeboten und damit, wie oft Gewichtsmodifikationen vorgenommen werden)
- **Radius der Nachbarschaft**: Größe der bei Gewichtsanpassungen berücksichtigten Nachbarschaft
- **Lernparameter**: bestimmt, wie stark die Gewichte zwischen den Input-Units und den betroffenen Output-Units verändert werden
- **Matrixgröße**: Anzahl der Outputneuronen: Je mehr Output-Units das Kohonennetz umfasst, desto genauer kann eine Clusterung des Inputraumes vorgenommen werden
- **Form der Nachbarschaftsfunktion**: Die Nachbarschaftsfunktion bestimmt, in welcher Weise benachbarte Neuronen von Gewichtsveränderungen betroffen sind. Beispielsweise kann die Stärke der Veränderung mit zunehmender Distanz vom Gewinner linear oder exponentiell abnehmen
- Art und Weise, wie die oben genannten Paramter (Lernparameter, Radiusgröße, Nachbarschaftsfunktion) im Lerndurchgang verändert werden
- Dimension (2-dimensional, 3-dimensional, ..., n-dimensional) des Kohonennetzes

Kohonennetze / Selforganizing Maps

Anwendungsmöglichkeiten

Wozu kann man solche Kohonennetze verwenden?

Anwendungsmöglichkeiten: Kohonennetze

Unter anderem existieren folgende **Anwendungsmöglichkeiten** (neben den bereits genannten Anwendungen bei den kompetitiven Netzen):

- **Approximation von Funktionen** (u. a. bei der keine analytische Lösung existiert)
- **Inverse Kinematik**, z. B. bei mechanischen Armen bei Robotern im 2-dimensionalen Raum
- Im Rahmen der inversen Kinematik: Kohonennetze können den **kürzesten Weg zwischen 2 Punkten finden** auch wenn sich auf dem direkten Weg Hindernisse befinden (vorausgesetzt, diese Hindernisse verändern ihre Position nicht)
- **n-dimensionale Figuren** kann man in 2- (oder mehr) dimensionale Figuren **transformieren** ohne die kritischen Informationen dabei zu verlieren (für den Menschen von Nutzen, da dieser sich nur 2- oder 3-dimensionale Objekte vorstellen und visualisieren kann)
- **Traveling Salesman Problem** mit dem sogenannten Elastischen Netzalgorithmus
- **Spracherkennung**, **Unterschriftenerkennung**, **Gesichtererkennung** (also verschiedene Formen der Mustererkennung)

Übersichtstabelle: Netztypen

Tabelle 2: Tabellarische Darstellung der Kernkonzepte, Eigenschaften sowie Vor- und Nachteile der Netztypen: Pattern Associator, Rekurrente Netze, Kompetitive Netze und Kohonennetze.

	Pattern Associator	**Rekurrente Netze**	**Kompetitive Netze**	**Kohonennetze**
Kernkonzept	Assoziationen zwischen verschiedenen Reizpaaren bilden	Rückkopplungen zu derselben oder einer vorherigen Schicht	1. Erregung 2. Wettbewerb 3. Gewichts-modifikation	Wie Kompetitive Netze, nur mit mehrdimensionaler Output-Schicht
Lernregel	Hebb-Regel; Delta-Regel	Backpropagation	Competitive Learning	Konzeptuell: Competitive Learning
Rückkopplungen?	Nein	Ja	Nein	Nein
Hidden-Units?	Nein	Können vorhanden sein	Können vorhanden sein	In der Regel nicht
Art der Lernregel?	Supervised learning	Supervised learning	Unsupervised learning	Unsupervised learning
Vorteile	Einfachheit	Entdeckung zeitlich codierter Informationen	Biologische Plausibilität	Biologische Plausibilität
Nachteile	Keine Hidden-Units --> biologisch eher unplausibel	"Überlaufen" der Aktivität	"Erstarken" einzelner Output-Units verhindert "sinnvolle" Kategorisierung	Wahl zahlreicher Parameter entscheidend für adäquate Clusterung

Eigenschaften: Einleitung

Eigenschaften neuronaler Netze

Auf den vorangegangenen Folien wurde bereits auf einige Eigenschaften neuronaler Netze eingegangen. Die folgenden Seiten sollen einen systematischen Überblick über diese Eigenschaften liefern, wobei hier sowohl Parallelverarbeitung als auch verteilte Speicherung von zentraler Bedeutung sind.

Parallelverarbeitung und verteilte Speicherung

- **Parallelverarbeitung**: Parallelverarbeitung bedeutet, dass ein neuronales Netz die notwendigen Berechnungen (z. B. des Inputs oder des Netzinputs) nicht nacheinander durchführt, sondern gleichzeitig. Allerdings muss man beachten, dass diese Parallelverarbeitung bei der Simulation neuronaler Netze am PC zur Zeit nur theoretischer Natur ist, da die meisten heutigen Computer nur eine Berechnung nach der anderen durchführen können (=seriell arbeiten). Von der zunehmenden Parallelisierung auf dem PC-Markt dürften daher in Zukunft u. a. Programme profitieren, die auf neuronale Netze zurückgreifen.

- **Verteilte Speicherung**: Auch die verteilte Speicherung bei neuronalen Netzen ähnelt viel mehr der Arbeitsweise menschlicher Gehirne. Während beispielsweise auf einer CD, DVD oder einer Festplatte bestimmte Informationen an einer bestimmten Stelle gespeichert sind (=lokale Speicherung), ist die Speicherung von Informationen in neuronalen Netzen verteilt. Dies heißt, dass eine Information (z. B. das Alter einer Person) über das gesamte Netz oder zumindest über einen Teil davon gelagert wird. Genauer formuliert: Das Wissen ist in vielen verschiedenen Gewichten zusammen abgespeichert.

Weitere Eigenschaften neuronaler Netze

Eigenschaften neuronaler Netze unterstreichen die biologische Plausibilität

- **zum Teil biologische Plausibilität**: Neuronale Netze besitzen allein aufgrund der Parallelverarbeitung und verteilten Speicherung eine gewisse Nähe zum menschlichen Gehirn. Außerdem konnte man mit Hilfe neuronaler Netzsimulationen menschliches Verhalten oftmals sehr gut simulieren und aus ihnen weitere Hypothesen ableiten, die sich in der Folge beim Menschen empirisch bestätigen ließen. Allerdings wurden neuronale Netze auch hinsichtlich ihrer fehlenden biologischen Plausibilität kritisiert (siehe Seite "Probleme neuronaler Netze").

Die biologische Plausibilität zeigt sich in folgenden Eigenschaften:

- **Toleranz gegenüber internen Schäden**: Wie bereits erwähnt wird trotz innerer Schäden des neuronalen Netzes (z. B. durch das Absterben einzelner Neuronen oder Verbindungen zwischen Neuronen) oftmals dennoch der richtige Output produziert.
- **Toleranz gegenüber externen Fehlern**: Auch bei unvollständigem oder fehlerhaftem Input gelingt es dem neuronalen Netz häufig, den richtigen Output herzustellen.
- **Generalisation / Kategorienbildung**: Ähnliche Reize werden von den Netzen zur selben Reizgruppe kategorisiert. Auf das Problem der "Übergeneralisierung" wurde bereits hingewiesen.
- **Output der zentralen Tendenz bzw. des Prototypen der Kategorie**: Eine weitere Eigenschaft neuronaler Netze bezieht sich auf den gebildeten Output eines solchen Netzes. Dieses formt häufig bei mehreren gelernten Inputvektoren einen gemeinsamen Outputvektor aus, den man als Prototypen einer Kategorie auffassen kann. Auch dies ist eine für Menschen typische Eigenschaft.
- **Inhaltsabruf (content addressability)**: Der Inhaltsabruf besagt, dass man für das Abrufen von Informationen dem Netz lediglich Inhalte präsentieren muss, die mit dem gesuchtem Inhalt in Verbindung stehen. Beispielsweise kann es ausreichen, wenn man den Namen (an den man sich gerade nicht erinnert) einer Person in Erfahrung bringen will, sich das Aussehen, den Wohnort oder den Freundeskreis der Person zu vergegenwärtigen. Dieses Prinzip gilt für Menschen wie für neuronale Netze.

Darüber hinaus weisen künstliche neuronale Netze auf:

- **hohe Lernfähigkeit**: Neuronale Netze haben im Gegensatz zu vielen herkömmlichen statistischen Methoden keine Probleme auch Zusammenhänge nonlinearer Art oder Interaktionseffekte zwischen mehreren Variablen zu erfassen. Man spricht in diesem Zusammenhang auch vom Erlernen beliebiger (=arbiträrer) Abbildungen.
- **große Anzahl an Freiheitsgraden**: Grundsätzlich existieren zahlreiche Parameter, die die Arbeitsweise eines neuronalen Netzes beeinflussen können. Diese Variationsmöglichkeiten bedingen die hohe Lernfähigkeit und Erfassung nonlinearer Zusammenhänge neuronaler Netze mit. Dass diese große Anzahl an Freiheitsgraden auch Probleme mit sich bringt, zeigt die nächste Seite.

Probleme neuronaler Netze

Abbildung 21: Karl R. Popper (1902 - 1994)

- **Große Anzahl an Freiheitsgraden**: Viele frei variierbare Parameter können dazu führen, dass ein neuronales Netz grundsätzlich jede menschliche Verhaltensweise simulieren kann. Die Gefahr besteht darin, dass neuronale Netze als Konzept zur Erklärung menschlichen Verhaltens nicht falsifizierbar (=nicht widerlegbar) sind, sondern durch Variation verschiedener Parameter immer vor der Falsifikation geschützt werden können (man spricht in diesem Zusammenhang auch von einer Immunisierungsstrategie). Das auf **Karl R. Popper** (z. B. 1996) zurückgehende Wissenschaftskriterium der Falsifizierbarkeit betrifft allerdings nur diejenigen neuronalen Netze, die modelliert werden, um die Funktionsweise des menschlichen Gehirns besser zu verstehen. Neuronale Netze, die konkrete Anwendungsprobleme lösen sollen, sind von diesem Kritikpunkt nicht betroffen.

- **Fragwürdige biologische Plausibilität**: Auch der zweite Kritikpunkt bezieht sich nur auf diejenigen Netze zur Erklärung der Funktionsweise des menschlichen Gehirns. Viele neuronale Netze widersprechen biologischen Grundannahmen und sind somit als Modell zur Erklärung menschlichen Verhaltens nur bedingt geeignet. Ein Beispiel hierzu wäre die Rückwärtsausbreitung bei der "Backpropagation"-Lernregel.

- **Großer Rechenaufwand**: Herkömmliche Methoden zur Lösung eines Problems besitzen oftmals einen geringeren Rechenaufwand und können mitunter eine exaktere Lösung produzieren als neuronale Netze. Im Gegensatz zu den beiden anderen aufgeführten Problemen betrifft dieser Kritikpunkt nur neuronale Netze, die dazu dienen konkrete Anwendungsprobleme zu lösen.

Concept map: Eigenschaften

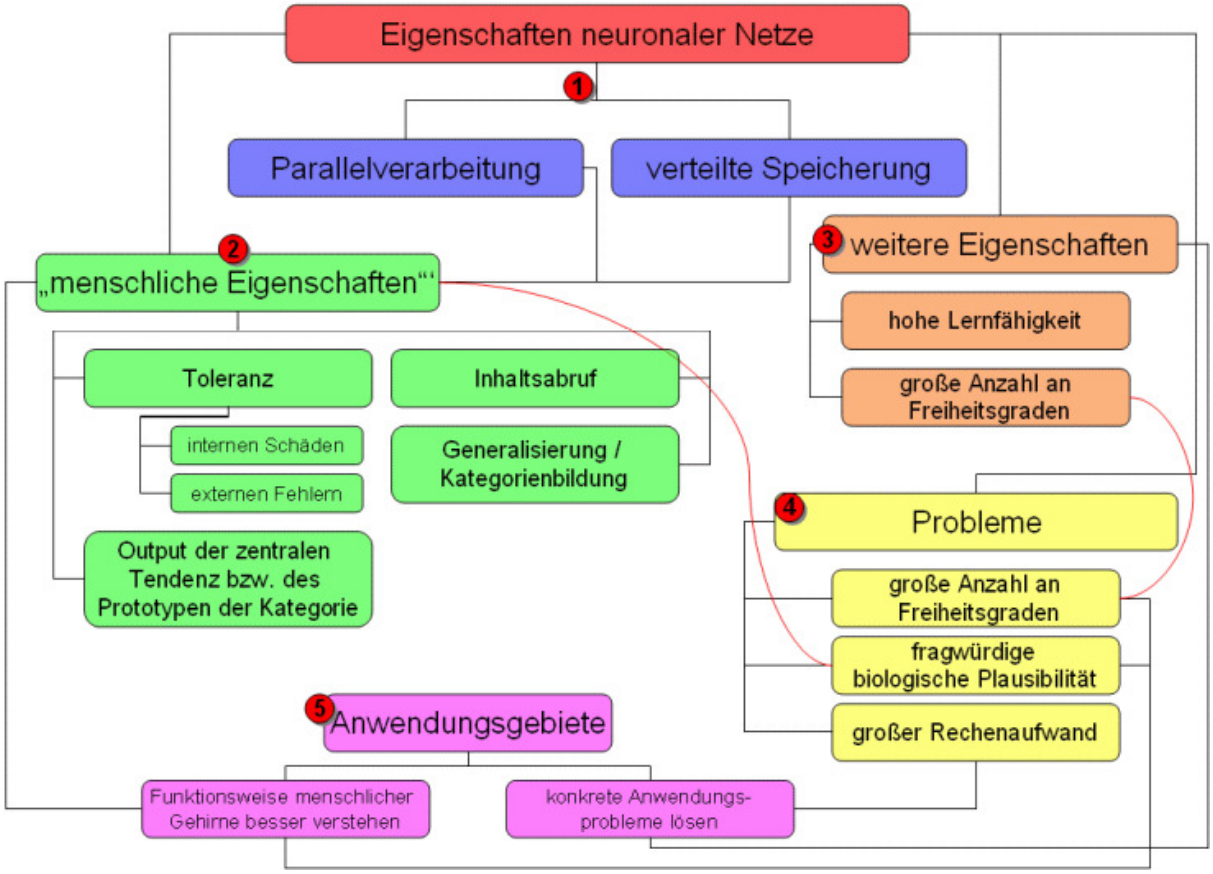


Abbildung 22: Concept map zu dem Kapitel Eigenschaften. Erläuterung: siehe Text.

Dieser visuelle Strukturüberblick fasst die zentralen Konzepte des Kapitels "Eigenschaften" als Concept map zusammen.

Die 5 rot markierten Kreise stellen Teilbereiche in der Grafik dar, die im folgenden erläutert werden:

1. Die beiden zentralen Eigenschaften neuronaler Netze sind: **Parallelverarbeitung** und **verteilte Speicherung**. Man beachte, dass diese auch als typische menschliche Eigenschaften aufgefasst werden können.

2. Zu den **"menschlichen Eigenschaften"** (im Text als biologische Plausibilität bezeichnet) zählen die Toleranz gegenüber internen Schäden und externen Fehlern, der Output der zentralen Tendenz bzw. des Prototypen der Kategorie, der Inhaltsabruf (content addressable) sowie die Generalisierung bzw. Kategorienbildung. Bei bestimmten Netzen ist die biologische Plausibilität jedoch nicht gewährleistet (z. B. bei Netzen, die das Backpropagation-Verfahren als Lernregel verwenden). Daher wurde in rot eine Verbindung zur fragwürdigen biologischen Plausibilität neuronaler Netze eingezeichnet.

3. Neben den "menschlichen Eigenschaften" lassen sich auch noch **weitere Eigenschaften** neuronaler Netze aufführen. U. a. ist hier die hohe Lernfähigkeit und die große Anzahl an Freiheitsgraden zu nennen. Auch hier stellt die rote Verbindung zu den Problemen dar, dass die große Freiheitsgradanzahl auch zu Schwierigkeiten (Stichwort: Falsifizierbarkeit) führen kann.

4. Bei den **Problemen** neuronaler Netze kann man neben den bereits angesprochenen vielen Freiheitsgraden und der fragwürdigen biologischen Plausibilität auch den großen Rechenaufwand nennen.

5. Nicht unmittelbar zu den Eigenschaften zählend finden sich die beiden **Anwendungsgebiete neuronaler Netze** in dieser Begriffsstrukturdarstellung.

- Der Aspekt, die **Funktionsweise menschlicher Gehirne besser zu verstehen**, lässt sich unmittelbar mit "menschliche Eigenschaften" verknüpfen. Es ergeben sich jedoch auch Probleme aufgrund der großen Anzahl an Freiheitsgraden und der fragwürdigen biologischen Plausibilität.
- Im Gegensatz dazu zeichnen sich neuronale Netze zur **Lösung konkreter Anwendungsprobleme** durch ihre hohe Lernfähigkeit und der großen Anzahl an Freiheitsgraden aus, besitzen aber den Nachteil des großen Rechenaufwandes.

Anwendungen: Überblick

Anwendungsbeispiele zur Funktionsweise des menschlichen Gehirns

Auf den folgenden Seiten werden verschiedene Anwendungen exemplarisch dargestellt. Die vorgestellten Beispiele beziehen sich ausschließlich auf neuronale Netze, die modelliert wurden, um die Funktionsweise des menschlichen Gehirns besser zu verstehen.

Folgende Anwendungsbeispiele werden nun besprochen:

- **Farbkonstanz** -> Hier wird das Phänomen der Farbkonstanz erörtert, das bei der Wahrnehmung von Farben auftritt.
- **Routinetätigkeiten** -> In diesem Anwendungsbeispiel wird die Repräsentation und Ausführung von Routinetätigkeiten thematisiert.
- **Autismus** -> Das Unterkapitel beschäftigt sich mit der tiefgreifenden Entwicklungsstörung "Autismus".

Diese Themen werden jeweils in 3 Schritten besprochen:

1. **Ausgangssituation:** Zunächst führt eine kurze Einleitung in die Thematik des Anwendungsbeispiels ein. Hierbei wird die Ausgangsbasis der Simulationsstudie geschildert.
2. **Netzaufbau:** In einem zweiten Schritt wird der Aufbau des neuronalen Netzes dargestellt. Dabei werden die verwendeten Input-, Hidden- und Output-Schichten eingehend erörtert sowie die konkrete Durchführung der Simulation thematisiert.
3. **Ergebnisse und Fazit:** Die Darstellung der Ergebnisse der Untersuchung bildet gemeinsam mit einem kurzen Fazit den Abschluss eines jeden Unterkapitels. Hier wird – neben der Würdigung der Simulationsstudie – auch kurz auf weiterführende Fragestellungen und Kritik zur Untersuchung eingegangen.

Farbkonstanz

Ausgangssituation

Farbkonstanz

Dieses Anwendungsbeispiel bezieht sich auf das Phänomen der Farbkonstanz, das bei der Wahrnehmung von Farben auftritt.

Als Farbkonstanz bezeichnet man die wahrgenommene Stabilität einer Objektfarbe unter verschiedenen Beleuchtungsumgebungen. Beispielsweise erscheint uns eine Zitrone gelb, unabhängig davon, ob wir diese bei Sonnenschein, unter einer herkömmlichen Glühbirne oder dem Licht einer Fluoreszenzlampe betrachten (Kandel et al., 1995). Stünden uns keine Informationen aus der Umgebung der Zitrone zur Verfügung, so müsste diese aufgrund des von ihr reflektierten Lichtes genauer gesagt der Wellenlängenzusammensetzung dieses Lichtes –unterschiedliche Farben annehmen. Bei Sonnenlicht erschiene uns die Zitrone weißlich, unter der Glühbirne rötlich und unter der Fluoreszenzlampe bläulich.

Abbildung 23: Beispiel für Farbkonstanz. Man nimmt die Erdbeeren als rot wahr, sei es im Sonnenlicht (links, Original-Aufnahme), im Licht einer Glübirne statt eigentlich orange (mittig, simuliert durch Farbfilter) oder im Licht einer Fluoreszenzlampe statt eigentlich violett (rechts, simuliert durch Farbfilter), die weißen Plastikschalen dienen dabei als Vergleichsobjekt.

Dass ein Objekt – trotz weniger Ausnahmen – in aller Regel (in etwa) seine Farbe behält, ist deshalb erstaunlich, weil z.T. enorme Schwankungen in der Spektralverteilung der Umgebungsbeleuchtung (Sonnenschein vs. Glühbirne) auftreten. Diese Schwankungen führen dazu, dass auch das Objekt ganz unterschiedliche Wellenlängen des Lichtes reflektiert. Dennoch erscheint uns – bis auf wenige Ausnahmen – die Zitrone konstant in gelber Farbe (Kandel et al., 1995).

Informationsverarbeitung

Das Phänomen der Farbkonstanz kann allgemein damit erklärt werden, dass das menschliche Gehirn die Informationen, die es aus der Wellenlängenzusammensetzung des reflektierten Lichtes des Objektes und seiner Umgebung erhält, weiterverarbeitet. Am Ende dieses Informationsverarbeitungsprozesses sollten sich Neuronen finden lassen, deren Antwortverhalten mit dem (subjektiven) Farbeindruck eines Objektes korrespondiert und nicht mehr mit der spezifischen Wellenlängenzusammensetzung des vom Objekt reflektierten Lichtes. Derartige Neuronen wurden tatsächlich entdeckt und zwar von Semir Zeki (1993) im visuellen Areal V4 des Cortex.

Auch wenn davon ausgegangen werden kann, dass Neuronen im Areal V4 den vorläufigen Endpunkt des Informationsverarbeitungsprozesses darstellen, so scheint bis heute ungeklärt, wie dieser Prozess auf neuronaler Ebene genau abläuft. Stanikunas, Vaitkevicius und Kulikowski (2004) haben daher den Versuch unternommen, den Prozess mit Hilfe eines künstlichen neuronalen Netzes zu simulieren, um dadurch ein tieferes Verständnis über das Phänomen der Farbkonstanz zu erlangen.

Farbkonstanz

Netzaufbau

Netzaufbau

Das von Stanikunas, Vaitkevicius und Kulikowski (2004) verwendete neuronale Netz stellte ein Feedforward-Netz dar, wobei neben der Input- und Output-Schicht zwei weitere Hidden-Schichten zum Einsatz kamen (siehe Abbildung 24):

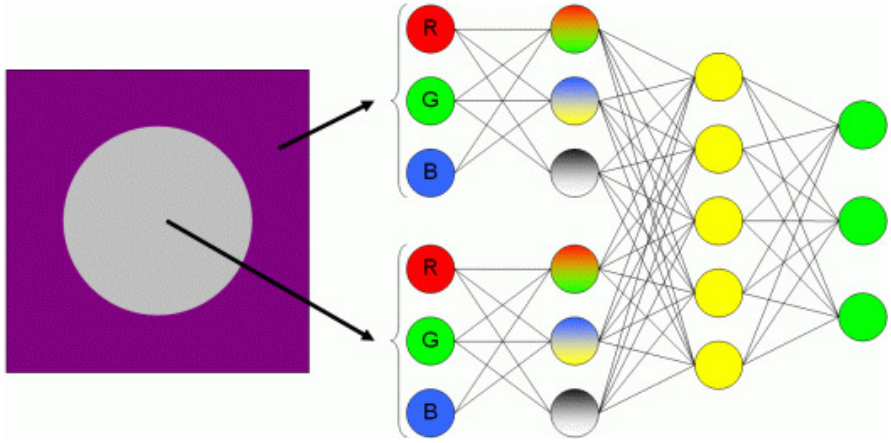


Abbildung 24: Schematische Darstellung des Netzaufbaus zur Simulation der Farbkonstanz. Die Beschreibung der einzelnen Units ist dem Text zu entnehmen.

Input-Schicht

- **Input-Schicht**: Die Input-Schicht umfasste insgesamt sechs Input-Units und ließ sich in zwei Farbkanäle unterteilen, die jeweils drei Units umfassten:
  - **R-Zapfen**: Diese Unit produzierte im künstlichen neuronalen Netz auf Zahlen, die Licht einer langen Wellenlänge repräsentierten, einen besonders starken Output und reagierte im Bereich langer Wellenlängen besonders sensitiv. Analog dazu gibt es auch im menschlichen Auge einen solchen Zapfen, der ebenfalls auf sichtbares Licht mit einer längeren Wellenlänge reagiert und somit hauptsächlich für die Wahrnehmung von Rot verantwortlich ist.
  - **G-Zapfen**: G-Zapfen (G für Grün) sprachen besonders auf den mittleren Wellenbereich des sichtbaren Lichtes an, so dass diese Zapfen sowohl im menschlichen Auge als auch in dem künstlichen neuronalen Netz besonders sensitiv auf eine solche mittlere Wellenlänge reagieren.
  - **B-Zapfen**: Die kürzeste Wellenlänge wurde durch die B-Zapfen abgedeckt. Die Zapfen waren folglich besonders sensitiv für die Farbe Blau.

  Die Unterscheidung dreier Zapfen beruht auf dem britischen Physiker Sir Thomas Young, der zu Beginn des 19. Jahrhunderts drei Arten von Lichtsinneszellen für die Farbwahrnehmung postulierte.

Hidden-Schicht

- **Erste Hidden-Schicht**: Die erste Hidden-Schicht war durch zuvor festgelegte, unveränderbare Gewichte mit den Input-Units verbunden. Es wurden drei verschiedene Typen von Gegenfarbenzellen unterschieden:
  - **R-G**: Bei dem Typ "R-G" standen sich die Farben Rot und Grün gegenüber. Im künstlichen neuronalen Netz wurden drei Zahlwerte, die die kurzen, mittleren und langen Wellenlängen

repräsentierten so verrechnet, dass diese Unit durch die Farbe Rot (lange Wellenlängen) aktiviert und durch die Farbe Grün (mittlere Wellenlängen) gehemmt wurde. Die kurze Wellenlänge ("Blau") hatte bei der Berechnung nur einen äußerst geringfügigen Einfluss.

- **B-G**: Die Unit, die das Gegensatzpaar Blau-Gelb (G steht hier nicht für Grün, sondern für Gelb!) symbolisierte, wurde ebenfalls von den R-, G- und B-Zapfen gespeist. Der B-Zapfen, d.h. die kurze Wellenlänge, besaß im künstlichen Netz auf diese Gegenfarbenzelle einen stark hemmenden Einfluss, während R- und G-Zapfen (lange und mittlere Wellenlängen) jeweils einen moderaten aktivierenden Einfluss aufwiesen. Wurden beide Wellenlängen aktiviert, so addierte sich dieser Einfluss entsprechend. Dies war vor allem bei der Farbe Gelb der Fall, die sich ungefähr zwischen (bezüglich der Wellenlänge!) den Farben Rot und Grün befand.

- **S-W**: Das Gegensatzpaar Schwarz-Weiß war nur mit den R- und G-Zapfen durch positive Gewichte verbunden (siehe Abbildung 24). Weißes Licht, welches die R- und G-Zapfen in etwa in gleichem Maße aktivierte (Kandel et al., 1995), führte zu einer Aktivierung dieser Unit. Wurde kein Licht dargeboten, so blieb eine Aktivierung aus.

Die Annahme dieser aufgeführten Gegenfarbenzellen griff auf die Gegenfarbentheorie des deutschen Physiologen Ewald Hering aus dem Jahr 1877 zurück.

- **Zweite Hidden-Schicht**: Die zweite Hidden-Schicht umfasste fünf Hidden-Units, die vollständig mit den sechs Gegenfarbenzellen der ersten Hidden-Schicht verbunden waren (siehe Abbildung 24). Sämtliche Gewichte zwischen diesen beiden Schichten konnten in der **Trainingsphase** modifiziert werden.

Outputschicht

- **Output-Schicht**: Auch die Output-Schicht war vollständig mit der zweiten Hidden-Schicht verbunden (siehe Abbildung 24), wobei hier ebenfalls alle Gewichte veränderbar waren. Der Output-Vektor wurde mit Hilfe der drei Output-Units gebildet, der – stark vereinfacht dargestellt – den Farbeindruck eines Objektes als Zahlenwert beschreiben und somit Neuronen im visuellen Areal V4 repräsentieren sollte.

Wie in Abbildung 24 erkennbar, wurde das Netz in zwei Verarbeitungspfade unterteilt. Der obere Pfad der Abbildung griff auf Farbinformationen des Hintergrundes zurück, während der untere Pfad die Farbe des betrachteten Objektes analysierte.

Dem neuronalen Netz wurde das Objekt in der Trainingsphase unter 40 verschiedenen Farben (nach dem Farbsystem von Munsell) dargeboten, während die Farbe des Hintergrundes nicht variierte. Jedoch konnte auch das vom Hintergrund reflektierte Licht in Abhängigkeit der verwendeten Beleuchtung unterschiedliche Wellenlängen annehmen. Insgesamt kamen in verschiedenen Trainingsdurchläufen maximal neun verschiedene Beleuchtungen zum Einsatz. Ziel des Trainings war die korrekte Identifikation der Objektfarbe unter den unterschiedlichen Beleuchtungsbedingungen.

In der Trainingsphase wurden die korrekten Output-Werte vorgegeben (**supervised learning**) und die variablen Gewichte mit Hilfe der **Backpropagation** Lernregel angepasst.

Farbkonstanz

Ergebnisse und Fazit

Ergebnisse

Von den umfangreichen Befunden der Arbeit von Stanikunas, Vaitkevicius und Kulikowski (2004) soll an dieser Stelle nur Bezug zu einigen ausgewählten Ergebnissen genommen werden:

- Sofern mindestens fünf verschiedene Beleuchtungsbedingungen in der Trainingsphase zur Verfügung stehen und diese relativ gleichmäßig über den Farbraum verteilt sind, ist eine Generalisierung des Gelernten festzustellen. In diesem Fall zeigt sich Farbkonstanz auch unter neuen, dem Netz unbekannten Beleuchtungsbedingungen.
- Analysen trainierter Netze mit Objekten verschiedener Farbsättigung in der Trainingsphase zeigen wie erwartet, dass bessere Lernleistungen erzielt werden (d.h. Farbkonstanz in stärkerem Maße erreicht wird), wenn das Objekt eine hohe Farbsättigung aufweist. Dies liegt vermutlich daran, dass Objekte hoher Farbsättigung einen größeren Bereich des Farbraums abdecken.
- Vergleicht man die Ergebnisse des künstlichen neuronalen Netzes mit Untersuchungsbefunden zur Farbkonstanz bei menschlichen Probanden (Kulikowski und Vaitkevicius, 1997), so zeigen sich trotz vereinzelter Unterschiede große Gemeinsamkeiten bei der Farbeinschätzung von Objekten unter verschiedenen Beleuchtungsbedingungen. Auf Basis der Befunde des künstlichen Netzes kann vermutet werden, dass das menschliche Sehsystem zwei separate, jedoch gleichzeitig ablaufende Berechnungen durchführen muss, damit das Phänomen der Farbkonstanz auftritt: Zum einen muss die Farbe des Hintergrundes eines Objektes näherungsweise geschätzt werden und zum anderen ist die Farbdifferenz zwischen dem Objekt und seinem Hintergrund zu bestimmen.

Fazit

Als Fazit lässt sich festhalten, dass das neuronale Netz von Stanikunas, Vaitkevicius und Kulikowski (2004) einen interessanten Ansatz zum Thema Farbkonstanz darstellt. Zudem scheint der Ansatz, Farbkonstanz mittels neuronaler Netze zu erforschen, vielversprechend. Ein tieferes Verständnis dieses Phänomens hat zudem unmittelbare Implikationen u.a. auf die Simulation des menschlichen Auges bei Robotern, aber auch beispielsweise bezüglich der Möglichkeit verschiedener Beleuchtungen in Umkleidekabinen von Modehäusern.

Kritisch anzumerken ist an der Arbeit von Stanikunas, Vaitkevicius und Kulikowski (2004), dass es sich bei der Lernregel um **supervised learning** handelt. Möglicherweise tritt bei Menschen auch dann das Phänomen der Farbkonstanz in Erscheinung, wenn ihnen niemals jemand zuvor gesagt hat, welches der "richtige" Farbeindruck eines Objektes ist.

Routinetätigkeiten

Ausgangssituation

Routinetätigkeiten

Das nun folgende Anwendungsbeispiel befasst sich mit der Repräsentation und Ausführung von Routinetätigkeiten.

Morgens aufstehen, mit dem Auto zur Arbeit fahren, einen Tee kochen – all diese Tätigkeiten werden von uns (zumeist) mühelos und ohne größere Konzentration ausgeführt. Bei genauerer Analyse stellt sich jedoch heraus, dass derartige Tätigkeiten komplex sind und sie ein kompliziertes Wechselspiel zwischen Wahrnehmungs-, Aufmerksamkeits-, Gedächtnis- und Bewegungsprozessen sowie vielen weiteren Prozessen erfordern. Insofern erscheint es sinnvoll, sich unter einer psychologischen Perspektive mit derartigen Tätigkeiten zu beschäftigen.

Hierarchische Modelle

Zur Beschreibung von Routinetätigkeiten wurde und wird sehr häufig auf (sequentiell-)hierarchische Modelle zurückgegriffen. Hierarchische Modelle, beschreiben eine Tätigkeit mittels einer Baumstruktur (siehe Abbildung 25). An der Spitze dieser pyramidenartigen Struktur findet sich das oberste Ziel einer Tätigkeit. Dieses Ziel lässt sich in weitere Unterziele untergliedern, die sich ihrerseits wieder in Subziele differenzieren lassen. Auf der untersten Ebene dieser Verästelung finden sich Basiseinheiten, die beispielsweise einfachen Bewegungsabläufen entsprechen. Ihnen wird die geringste Komplexität zugeordnet, wobei sie zumeist zur Erreichung des höherstufigen Ziels unbewusst in sequentieller Weise abgearbeitet werden (siehe das nicht ganz ernst gemeinte Beispiel in Abbildung 25).

Erst wenn alle Subeinheiten eines Ziels vollständig abgearbeitet wurden, gilt das Ziel als erfüllt und es kann das nachfolgende Ziel in Angriff genommen werden.

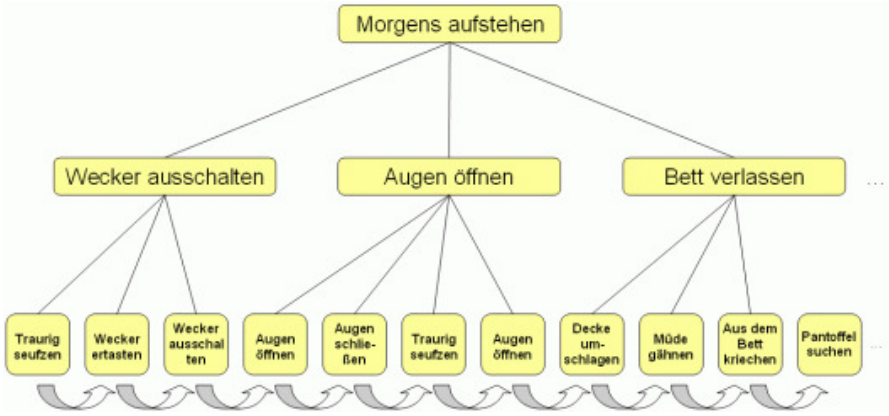


Abbildung 25: Beispielhafte, hierarchische Repräsentation der Routinetätigkeit 'Morgens aufstehen'.

Hierarchische Modelle weisen zahlreiche Vorteile bei der Darstellung von Routinetätigkeiten auf.

Allerdings besitzen diese Modelle auch diverse Probleme bei der Erklärung von Routinetätigkeiten. Um diese Probleme zu überwinden, haben Botvinick und Plaut (2004) vorgeschlagen, bei der Beschreibung von Routinetätigkeiten anstelle hierarchischer Modelle auf **rekurrrente Netze** zurückzugreifen.

Routinetätigkeiten

Netzaufbau

Netzaufbau

Das rekurrente Netz von Botvinick und Plaut (2004) stellte ein **Simple Recurrent Network (SRN)**. dar, wobei dieses aus drei verschiedenen Schichten bestand (siehe Abbildung 26):

- **Input-Schicht:**: Die Input-Schicht umfasste 39 Einheiten, wovon 20 Units verwendet wurden, um ein bestimmtes Objekt (z.B. Tasse oder Teebeutel) zu betrachten, während die verbleibenden 19 Einheiten angaben, welches Objekt aktuell mit der Hand festgehalten wurde.
- **Hidden-Schicht:**: Es kamen insgesamt 50 Hidden-Neuronen zum Einsatz, die vollständig mit den Einheiten der Input- und Output-Schicht verbunden waren.
- **Output-Schicht:**: In der Output-Schicht fanden sich 19 Units, die verschiedene Handlungen wie beispielsweise das Eingießen oder Umrühren einer Flüssigkeit repräsentierten. Die Tätigkeiten sollten an dem Gegenstand durchgeführt werden, der sich im Zentrum der Aufmerksamkeit befand (siehe Input-Schicht).

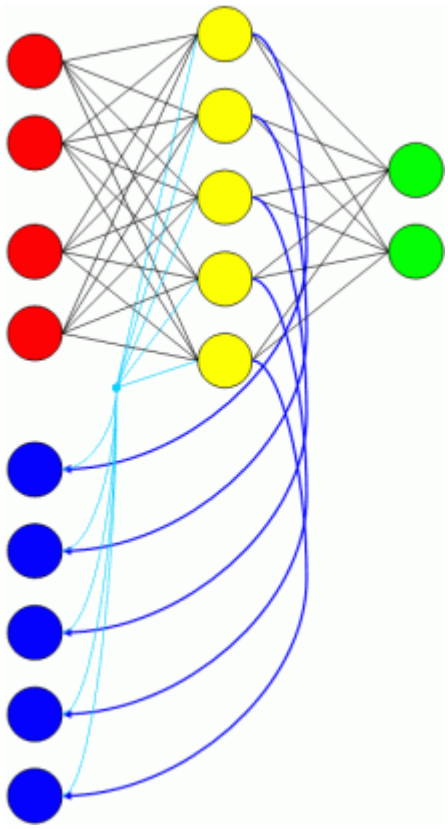


Abbildung 26: Schematische Darstellung eines SRNs zur Simulation von Routinetätigkeiten. Aus Gründen der Übersichtlichkeit werden Verbindungen von den Kontext- zu den Hidden-Units zurück 'gebündelt' dargestellt.

Dieses Netz (siehe Abbildung 26) sollte den Erwerb und die Durchführung der beiden Tätigkeiten "Kaffee und Tee kochen" simulieren. Damit das Netz wusste, welche der beiden Handlungen durchgeführt werden sollte, verwendete man zwei Input-Units als "Instruktionseinheiten" (Kaffee kochen vs. Tee kochen). In Abhängigkeit der durchzuführenden Tätigkeit wurde zu Beginn des Verarbeitungsprozesses eine der beiden Instruktionseinheiten einmalig aktiviert und verharrte danach in einem inaktiven Zustand, bis die gesamte Tätigkeit beendet und eine Neue begonnen wurde.

In der Trainingsphase wurden die berechneten Output-Werte mit den Korrekten verglichen (**supervised learning**) und die Gewichte mit Hilfe der **"Backpropagation-through-time"** Lernregel angepasst (Botvinick und Plaut, 2004). Zu beachten ist, dass der generierte Output teilweise als Eingabe der Input-Units für den nachfolgenden Schritt fungierte. Somit hätte man in Abbildung 26 auch die Output-Units noch mit den Input-Units verbinden können.

Routinetätigkeiten

Ergebnisse und Fazit

Ergebnisse

In den fünf verschiedenen Simulationsphasen ergeben sich zahlreiche Ergebnisse, wovon einige ausgewählte näher skizziert werden sollen:

- Dem neuronalen Netz gelingt es, die beiden Tätigkeiten, Kaffee und Tee kochen, zu erlernen und in der **Testphase** in Abhängigkeit der gewünschten Tätigkeit, die durch die entsprechende "Instruktionseinheit" vorgegeben wird, fehlerfrei zu reproduzieren. Folglich kann eine hierarchisch anordbare Aufgabenstruktur auch mit Hilfe eines nicht hierarchischen Modells wie dem vorgestellten rekurrenten Netz abgebildet werden.
- Wie beim menschlichen Verhalten auch produziert das Netz besonders während der Übergänge zweier Subaufgaben, d.h. bei Beendigung einer Teilhandlung und dem Beginn einer neuen Teiltätigkeit, besonders häufig Fehler. Detailliertere Analysen zeigen, dass die Fehlerentstehung intern zumeist einige Schritte vor dem Übergang entsteht (zumeist im mittleren Bereich einer Teiltätigkeit), sich jedoch aufgrund der Akkumulation der Fehlerwerte erst später im "Verhalten" bemerkbar macht.
- Des Weiteren werden – ebenfalls wie beim menschlichen Verhalten – vom künstlichen Netz vor allem Fehler generiert, bei denen die Teilhandlung zwar korrekt reproduziert wird, aber zum falschen Zeitpunkt beginnt. Außerdem stellt sich eine höhere Fehlerhäufigkeit ein, wenn die abzurufende Tätigkeit (z.B. Kaffee kochen) in der **Trainingsphase** im Vergleich zur "Konkurrenztätigkeit" (z.B. Tee kochen) seltener ausgeführt wurde.

Fazit

Insgesamt kann das rekurrente Netz von Botvinick und Plaut (2004) für Routinetätigkeiten als alternativer Erklärungsansatz zu den herkömmlich verwendeten hierarchischen Modellen betrachtet werden. Dabei decken sich die Simulationsergebnisse des künstlichen Netzes z.T. besser mit den empirischen Befunden zu Routinetätigkeiten und dort enthaltenem Fehlverhalten als hierarchische Ansätze. Des Weiteren können auf Basis der Simulationsergebnisse auch neue Hypothesen abgeleitet werden, die sich in Untersuchungen überprüfen lassen.

Neben einer weiterführenden Analyse menschlichen Fehlverhaltens mittels Simulation rekurrenter Netze böte sich auch die Anwendung dieser Netze auf nicht routinemäßig durchgeführte Tätigkeiten an (vgl. Botvinick und Plaut, 2004). So ist beispielsweise an Handlungen zu denken, die die Problemlösung betreffen. Möglicherweise greifen Menschen auch bei derartigen Tätigkeiten teilweise auf ihnen bekannte Routinehandlungen zurück (siehe z.B. Anderson und Lebiere, 1998; siehe z.B. Chapman und Agre, 1987).

Kritisch anzumerken ist, dass auch bei diesem Anwendungsbeispiel die überwachte (**supervised learning**) **Backpropagation** Lernregel zum Einsatz kommt, deren **biologische Plausibilität** zumindest fragwürdig erscheint. Positiv hervorzuheben ist in diesem Zusammenhang, dass die Autoren diesen Einwand selbst kritisch diskutieren und auf sequentielle Netze von Dominey (1998; Dominey und Ramus, 2000) verweisen, die mit Hilfe einer bestimmten Variante des verstärkenden Lernens (reinforcement learning) trainiert wurden.

Autismus

Ausgangssituation

Dieses Anwendungsbeispiel bezieht sich auf die autistische Entwicklungsstörung und erörtert den neuronale Netze Ansatz von Gustafsson und Paplinski (2004), der einen tieferen Einblick in diese tiefgreifende Entwicklungsstörung ermöglichen soll.

Der Film Rain Man

Der Film „Rain Man“ aus dem Jahr 1988 handelt von dem Autisten Raymond, gespielt von Dustin Hoffman. Während einer Reise durch die USA zeigt sich, welche Schwierigkeiten Raymond, der zuvor in einer Klinik lebte, tagtäglich bereitet. So besteht er beispielsweise auf das Ansehen bestimmter, täglich ausgestrahlter Fernsehsendungen sowie der absolut pünktlichen Einnahme seiner ausgewählten Mahlzeiten, da die Reise Raymond aufgrund des ungewohnten Tagesablaufs und der ständig wechselnden Umgebung ansonsten zu überfordern scheint. Neben diesen Schwierigkeiten werden aber auch die besonderen Fähigkeiten des Autisten dargestellt, der sich beispielsweise in kürzester Zeit umfangreiche Informationsbestände, wie Telefonbücher aneignen oder in Sekundenbruchteilen große Zahlen miteinander multiplizieren kann.

Autismus

Nicht alle Autisten wie Raymond in „Rain Man“ besitzen derart außergewöhnliche Fähigkeiten in einem eng umgrenzten Bereich wie dem Behalten von ganzen Telefonbüchern oder dem blitzartigen "Berechnen" großer Zahlen. Derartige extreme Inselbegabungen (auch Savant-Syndrom genannt) sind sogar äußerst selten anzutreffen (derzeit weltweit etwa 100 bekannte Personen, während Autismus deutlich höher, nämlich bei 0.05% der Geburten auftritt). Wenn sie jedoch auftreten, dann geht diese Begabung häufig mit einer autistischen Entwicklungsstörung einher. So leiden schätzungsweise 50% der Savants auch unter Autismus. Trotz der herausragenden Fähigkeiten besitzen Savants ebenso wie Autisten einen IQ-Wert, der zumeist unter 70 Punkten liegt (bei Autisten etwa 80%). Eine allgemein anerkannte Erklärung des Phänomens der Inselbegabung existiert bis heute nicht.

Autismus stellt nach den gängigen Klassifikationssystemen für Krankheiten bzw. psychischen Störungen (DSM-IV) und dem Internationalen Klassifikationssystem für Krankheiten in der zehnten Revision (ICD-10) eine tiefgreifende Entwicklungsstörung dar, mit folgenden Symptomen:

- Extrem autistische Einsamkeit
- Kommunikationsdefizite
- Zwanghafte und rituelle Handlungen

Bezüglich der Ursachen für Autismus existieren verschiedene, z.T. kontrovers diskutierte Ansätze (Davison et al., 2007; Gustafsson und Paplinski, 2004). Einigkeit herrscht darüber, dass Aufmerksamkeitsbeeinträchtigungen eine wichtige Rolle bei der autistischen Entwicklungsstörung spielen. Ungeklärt ist jedoch, ob derartige Defizite nur bei sozialen und/oder neuen Reizen in Erscheinung treten. Bezüglich der Vermeidung neuer Reize hatte bereits Leo Kanner (1943) bei der Darstellung von elf, unter Autismus leidenden Kindern auf eine geradezu zwanghafte Beharrlichkeit, vertraute Reize aufzusuchen, hingewiesen.

Neben Aufmerksamkeitsbeeinträchtigungen wurde auch postuliert, dass sogenannte Merkmalskarten im Kortex (cortical feature maps) autistischer Patienten unfähig zur Erstellung abstrakter Codes und Repräsentationen

von Objekten und Ereignissen seien (Gustafsson und Paplinski, 2004). Merkmalskarten repräsentieren bestimmte grundlegende und nutzbare Eigenschaften von Objekten, z.B. Farbe, Orientierung, Größe oder Entfernung. In Abhängigkeit dieser Objekteigenschaften sind die verschiedenen Merkmalskarten in unterschiedlichen Hirnarealen lokalisiert (siehe z.B. Kandel et al., 1995; Treisman, 1988). Die Simulation solcher kortikaler Merkmalskarten kann mit Hilfe von **Kohonennetzen** erfolgen. Im vorliegenden Beispiel untersuchen Gustafsson und Paplinski (2004), wie Aufmerksamkeitsbeeinträchtigungen und Bevorzugung von vertrauten gegenüber neuen Reizen bei Autisten die adäquate Ausbildung der kortikalen Merkmalskarten (in den Simulationen in Form von Kohonennetzen realisiert) behindern können

Autismus

Netzaufbau

Netzaufbau

In insgesamt vier verschiedenen Simulationsreihen verwendeten Gustafsson und Paplinski (2004) Kohonennetze mit vier (2 x 2) oder neun (3 x 3) Output-Units:

- **Input-Schicht:**: In allen Simulationen kamen zwei Input-Units zum Einsatz, die den zweidimensionalen Inputreiz bildeten. Die beiden Units wurden in fast allen Simulationsreihen mit insgesamt 60 verschiedenen Reizen gefüttert. Diese 60 zweidimensionalen Vektoren ließen sich in sechs verschiedene Reizgruppen unterteilen, wobei die zehn Reize in jeder Gruppe untereinander sehr hohe Ähnlichkeiten aufwiesen, d.h. im zweidimensionalen Raum nahe beieinander lagen. Je zwei Zehnergruppen waren darüber hinaus im zweidimensionalen Raum relativ nahe beieinander platziert. Folglich wurden zwei Dreiergruppen, d.h. insgesamt sechs Reizuntergruppen dargeboten, die aus jeweils zehn zweidimensionalen Reizen bestanden.
- **Output-Schicht:**: Die Output-Schicht umfasste vier (2 x 2) bzw. in der zweiten Simulation neun (3 x 3) Output-Units, die in rechteckiger Form angeordnet waren (siehe Abbildung 27).

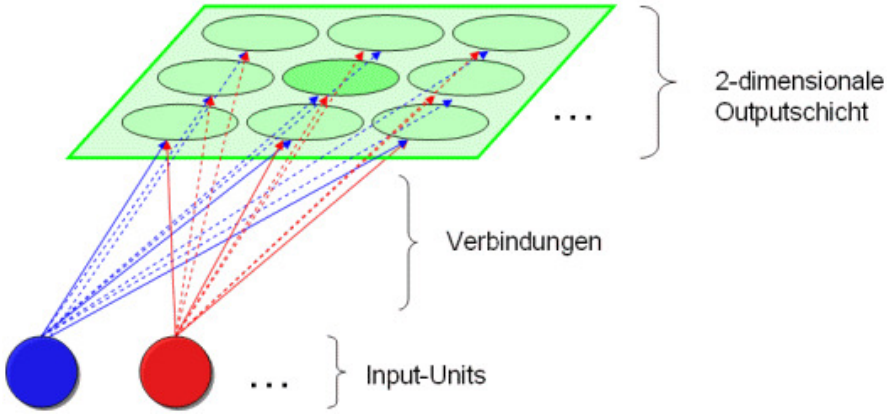


Abbildung 27: Schematische Darstellung des zweidimensionalen Kohonennetzes von Gustafsson und Paplinski (2004), welches von einem zweidimensionalen Input gespeist wird und vier (2 x 2) bzw. in der vorliegenden Abbildung neun (3 x 3), kreuzförmig verbundene Output-Units enthält.

In den Simulationen wurden insgesamt vier verschiedene "Lernmodi" miteinander verglichen:

- **Normaler Modus**: Es wurden die zwei Dreiergruppen (siehe oben) in etwa gleicher Häufigkeit dargeboten, d.h. es wurde mit einer Wahrscheinlichkeit von 50% nach Präsentation eines Inputreizes aus einer der beiden Gruppen nachfolgend ein Reiz aus der jeweils anderen Gruppe vorgegeben.
- **Modus mit Aufmerksamkeitsbeeinträchtigung**: In diesem Modus betrug das Verhältnis zwischen den beiden Dreiergruppen nicht 50% zu 50%, sondern etwa 99.5% zu 0.5%. Dies bedeutet, dass ein Aufmerksamkeitswechsel zur anderen Reizgruppe nach jedem Inputreiz durchschnittlich nur in einem von 200 Fällen auftrat. Zusätzliche Simulationen, die im Vorfeld der eigentlichen Studie durchgeführt wurden, zeigten, dass größere oder kleinere

Wahrscheinlichkeitsverhältnisse (anstelle der hier verwendeten 99.5% zu 0.5%) zu sehr ähnlichen Ergebnissen führten.

- **Modus mit Präferenz für vertraute Reize**: Dieser Modus begann wie der normale Modus, d.h. ein Aufmerksamkeitswechsel zur anderen Dreier-Reizgruppe erfolgte zu Beginn der Trainingsphase mit einer Wahrscheinlichkeit von 50%. Im Verlauf des Trainings bildete das Kohonennetz jedoch eine Präferenz für eine der beiden Reizgruppen aus und zwar in Abhängigkeit von der Vertrautheit mit diesen. Je vertrauter die Reizgruppe, d.h. je geringer die Distanz zwischen den Inputreizen einer Reizgruppe und den Gewichtsvektoren, desto eher verblieb die "Aufmerksamkeit" auf der jeweiligen Reizgruppe und ein Aufmerksamkeitswechsel zur anderen Reizgruppe fand umso seltener statt.
- **Modus mit Aufmerksamkeitsbeeinträchtigung und Präferenz für vertraute Reize**: Hier wurden die beiden zuvor aufgeführten Modi miteinander kombiniert.

Da es sich bei dem verwendeten **Kohonennetz** von Gustafsson und Paplinski (2004) um **unsupervised learning** handelt, wurde dem Netz keine korrekte Ausgabe vorgegeben, sondern die Reize wurden in Abhängigkeit ihrer Ähnlichkeit geclustert bzw. kartiert. Die Startgewichte wurden zufällig in einem kleinen Bereich in der Mitte des zweidimensionalen Inputraumes initialisiert.

Autismus

Ergebnisse und Fazit

Ergebnisse

Unter anderem können in den Simulationsstudien von Gustafsson und Paplinski (2004) folgende Ergebnisse erzielt werden:

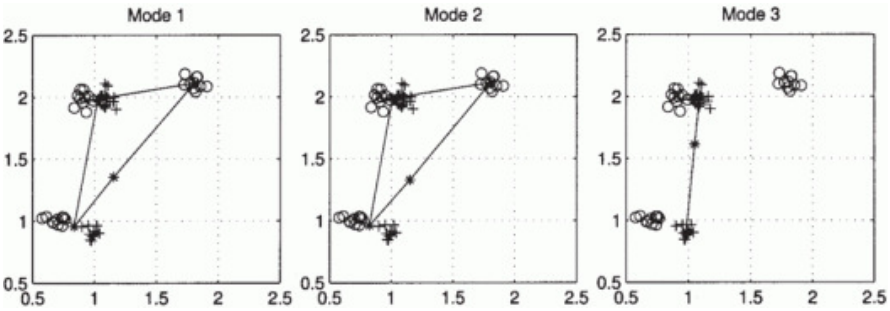


Abbildung 28: Schematische Darstellung des zweidimensionalen Input- und Outputraumes für drei verschiedene Lernmodi der dritten Simulation (aus Gustafsson und Paplinski, 2004).

- Im normalen Lernmodus als auch im Modus mit Aufmerksamkeitsbeeinträchtigung zeigt sich, dass die zwei verschiedenen Reizgruppen hinreichend erfasst werden. In Abbildung 28 ist dies an der Ähnlichkeit der beiden Linienverläufe abzulesen. Diese Linien werden mit Hilfe der vier (2 x 2) Output-Units gebildet, die in Abbildung 28 als "Stern" anhand ihrer beiden zugehörigen Gewichte im zweidimensionalen Raum eingezeichnet sind. Erkennbar ist, dass die zwei Reizgruppen, für deren Einzelreize die Symbole "+" und "Kreis" Verwendung fanden, im ersten und zweiten Modus durch das Kohonennetz "adäquat" repräsentiert werden. Im dritten Lernmodus, bei dem vertraute Reize präferiert werden, zeigt sich hingegen, dass das Netz, dessen vier Units in Abbildung 28 auf einer Linie liegen, nur eine der beiden Reizgruppen erfasst und die andere außer Acht lässt. Genauer gesagt werden die durch das "+"-Symbol dargestellten, insgesamt 40 Reize (2 x 20) sehr gut geclustert, während die "Kreis"-Symbole unberücksichtigt bleiben. Dieses Ergebnis scheint auch nicht zufällig zustande gekommen zu sein, da es in mehreren hundert Simulationen repliziert werden kann. Sofern man einen Bezug der Simulationsergebnisse auf Autisten und Savants wagt, könnte man argumentieren, dass diese – ähnlich wie im dritten Lernmodus – zwar bestimmte Reizgruppen außer Acht lassen bzw. dort deutliche Defizite aufweisen, jedoch in anderen, eng umgrenzten Teilbereichen eine sehr hohe Differenzierungsfähigkeit, bis hin zu extremen Inselbegabungen besitzen. Die Analysen mit Kohonennetzen legen zudem nahe, dass diese extremen Begabungen mit großen Defiziten in anderen Bereichen einhergehen. Gestützt wird diese Vermutung durch den Befund, dass sich bei Inselbegabten, die beispielsweise im Laufe ihres Lebens soziale Kompetenzen erwerben und ihre dortigen Defizite reduzieren, ihre extreme Begabung abschwächen kann. Bezüglich des zweiten Modus zeigen die Simulationsergebnisse, dass sich Aufmerksamkeitsbeeinträchtigungen im Vergleich zur Bevorzugung bekannter Reize nicht in der genannten Form auswirken. Insgesamt stützen die Befunde die Hypothese, dass autistische Symptome durch starke Präferenz vertrauter Reize hervorgerufen werden.
- Auch wenn das Ergebnis des Erwerbsprozesses im normalen Lernmodus und im Modus mit Aufmerksamkeitsbeeinträchtigung große Ähnlichkeiten aufweist (siehe Abbildung 29), so zeigen sich doch im Lernprozess selbst große Unterschiede. Während im

normalen Lernmodus die Gewichte der zugehörigen Output-Units relativ kontinuierlich zu ihren finalen Positionen im zweidimensionalen Raum "wandern", oszillieren die Gewichte des zweiten Lernmodus phasenweise dramatisch, bevor sie sich an ähnlicher Stelle wie im ersten Modus stabilisieren. Abbildung 29 visualisiert neben den Positionen der Inputreize und der vier Output-Units im zweidimensionalen Raum auch den Verlauf einer ausgewählten Output-Unit (genauer gesagt deren Position auf Basis ihrer beiden Gewichte). Deren Startpunkt wird durch eine Raute ("Raute") symbolisiert. Erkennbar sind dabei die deutlich höheren, bisweilen chaotisch wirkenden Schwankungen im Verlauf beim zweiten im Vergleich zum ersten Lernmodus. Haben sich die Positionen der Output-Units jedoch im Modus mit Aufmerksamkeitsbeeinträchtigung stabilisiert, so decken sie die Inputreize häufig sogar geringfügig besser ab als im normalen Lernmodus und können besser zwischen diesen diskriminieren (in der Abbildung nicht zu erkennen). Auch dies lässt sich mit empirischen Befunden zu Autisten in Beziehung setzen. So besitzen diese häufig gute Diskriminationsfähigkeiten, z.B. in Form eines absolutes Gehörs (z.B. Frith, 1989).

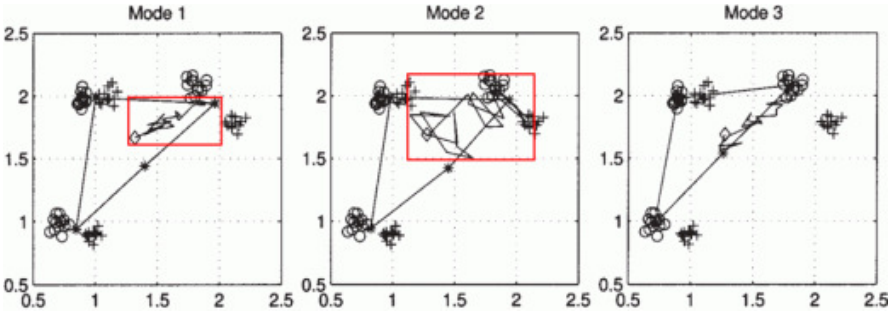


Abbildung 29: Schematische Darstellung des zweidimensionalen Input- und Outputraumes für drei verschiedene Lernmodi der ersten Simulation (aus Gustafsson und Paplinski, 2004).

Fazit

Die Simulationsstudien von Gustafsson und Paplinski (2004) stellen einen hochinteressanten Ansatz zur Erforschung autistischer Symptome mit Hilfe neuronaler Netze dar. Hervorzuheben sind u.a. die biologische Plausibilität aufgrund des Einsatzes einer nicht überwachten Lernregel (unsupervised learning) und die Übereinstimmung der Simulationsergebnisse mit zahlreichen Befunden zu Autisten und Inselbegabten.

Auch weiterführende Fragen ergeben sich aus den Simulationen. So könnte man versuchen, mit Hilfe von Kohonennetzen die Frage zu klären, unter welchen Umständen extreme Inselbegabungen bei Autisten zustande kommen und wann dies nicht der Fall ist. Möglicherweise könnten auch verschiedene Therapieansätze zur tiefgreifenden Entwicklungsstörung Autismus (Davison et al., 2007) in derartigen Simulationsstudien "überprüft" werden, wenngleich ein in der Simulation erfolgreicher Ansatz nicht zwangsläufig bei den Patienten selbst Symptomlinderung verspricht.

Trotz des Einsatzes von unsupervised learning können hinsichtlich der biologischen Plausibilität auch Bedenken angeführt werden. So handelt es sich bei den verwendeten Inputreizen und dem eingesetzten Kohonennetz um extrem starke Vereinfachungen kognitiver Prozesse im menschlichen Gehirn. Weitere Simulationsstudien, z.B. mit komplexeren Inputreizen wären hier sicherlich von Vorteil, wenngleich dies den derzeitigen Vorteil der guten Visualisierungsmöglichkeiten im zweidimensionalen Raum zunichte machen würde. Ein weiterer Kritikpunkt bezieht sich auf die Generierung neuer Prognosen auf Basis der Simulationsergebnisse, wobei diese Vorhersagen in neuen empirischen Untersuchungen überprüft werden könnten (siehe z.B.

das Anwendungsbeispiel "Routinetätigkeiten"). Derartige Hypothesen werden von den Autoren leider nicht vorgenommen.

Zusammenfassung: Anwendungen

Als zusammenfassendes Fazit zu den aufgeführten Anwendungsbeispielen lassen sich folgende Aspekte festhalten:

- Die Simulation der kompletten Breite menschlichen Verhaltens kann als Anwendungsgebiet neuronaler Netze herangezogen werden.
- Häufig funktioniert die Simulation menschlichen Verhaltens durch neuronale Netze sehr gut.
- Mitunter stellen neuronale Netze Ansätze sparsamere Modelle als traditionelle Ansätze bereit.
- Neben der Überprüfung vorhandener Hypothesen in Simulationsstudien ergeben sich auch neue Hypothesen durch die Simulationsbefunde. Diese können sich forschungsanregend auswirken.
- Die biologische Plausibilität der Modelle ist häufig fragwürdig oder überhaupt nicht gegeben.

Neben den vorgestellten Anwendungen zur Simulation menschlichen Verhaltens (bzw. die diesem zugrunde liegenden Gehirnprozesse) können neuronale Netze auch bei der Lösung von Anwendungsproblemen (z.B. bei der Datenauswertung) sinnvoll eingesetzt werden.

Exkurs: Einstiegshilfe in MemBrain

Einstiegshilfe in MemBrain

Dieser Exkurs soll Ihnen als erste Einstiegshilfe in den neuronale Netze Simulator "MemBrain" dienen. Ziel ist es, Ihnen den Einstieg in MemBrain zu erleichtern und somit die Grundlage zu schaffen, damit Sie zukünftig selbstständig neuronale Netze entwickeln können.

Downloadmöglichkeit

MemBrain ist ein in C++ verfasstes Programm von Thomas Jetter. Dieser bietet den Simulator auf seiner Homepage zum kostenlosen Download an. Das Programm ist zudem mit einer umfangreichen Hilfefunktion ausgestattet. Im Gegensatz zu anderen Simulatoren (z. B. dem neuronale Netze Simulator in Matlab) ist das Programm nicht nur kostenfrei zu beziehen, sondern ermöglicht auch Anfängern einen relativ leichten Einstieg in die komplexe Thematik. Des Weiteren steht Herr Jetter freundlicherweise bei Fragen, Anregungen und Kritik via Email zur Verfügung.

Gliederung der Einstiegshilfe

Der Exkurs ist ähnlich gegliedert wie das Kapitel "Grundlagen" dieser Lernhilfe. Zunächst wird erläutert, wie man mit MemBrain Units und Verbindungen erstellen kann. Anschließend wird die Wahl einer Aktivitätsfunktion dargestellt. Zur Illustration für die dann folgende Trainings- und Testphase dient ein einfaches "OR-Gatter".

Voreinstellungen anpassen

Voreinstellungen anpassen

Um für das im Exkurs gewählte Beispiel eine optimale Darstellung zu erhalten, sollten Sie die Folgenden drei Einstellungen anpassen bzw. überprüfen:

- **Menüpunkt "View", Untermenüpunkt "Show Activation Spikes on Links":** Hier sollte kein Häkchen gesetzt sein.
- **Menüpunkt "View", Untermenüpunkt "Show Fire Indicators":** Hier sollte ebenfalls kein Häkchen gesetzt sein.
- **Menüpunkt "Teach", Untermenüpunkt "Set Teach Speed...":** Tragen Sie im daraufhin erscheinenden Dialog einen Wert von "1" ein (Vorgabewert ist hier "0").

Alle anderen Einstellungen sollten nach der Erstinstallation von MemBrain bereits entsprechend gesetzt sein.

Anmerkung: Sie müssen diese Einstellungen nur beim ersten Start von MemBrain anpassen. Wenn Sie MemBrain beenden, werden die Einstellungen gespeichert und beim nächsten Start automatisch wieder geladen.

Units erstellen

Units erstellen

Nachdem Sie MemBrain auf Ihrem Rechner gestartet haben, erscheint zunächst neben einer Werkzeugleiste am oberen Bildschirmrand ein schwarzer Bildschirm.

Auf dieser schwarzen Oberfläche können Sie Units platzieren, indem Sie entweder in der Werkzeugleiste folgendes Symbol anklicken oder aber aus dem Hauptmenü "Insert" und dann "New Neurons" auswählen. Positionieren Sie nun wie in Abbildung 30 eine Unit auf dem schwarzen Bildschirm.

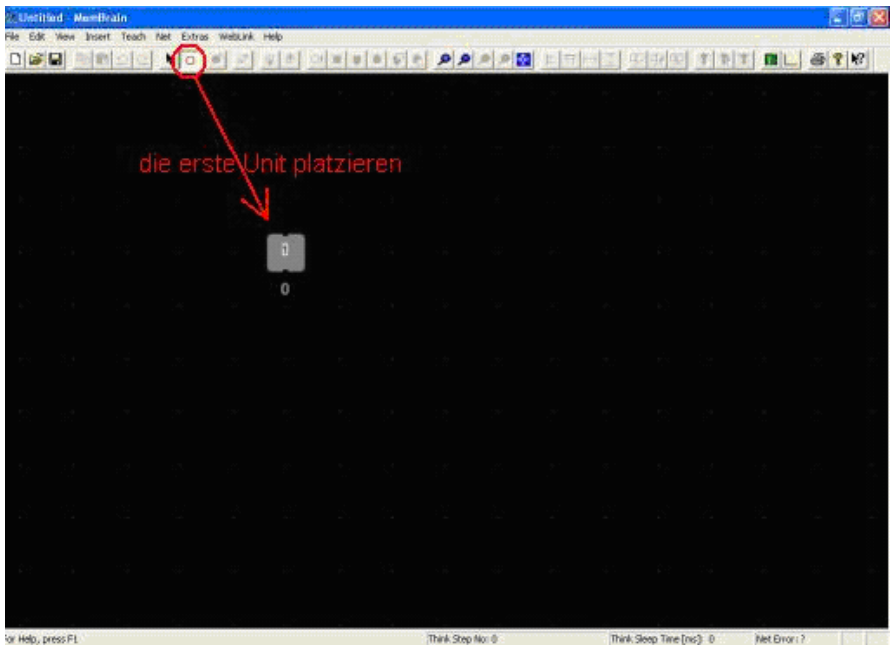


Abbildung 30: Erstellung einer Unit mittels MemBrain.

Anschließend sollten Sie mittels des Symbols oder dem Betätigen der "Esc"-Taste wieder den herkömmlichen Mauscursor bewegen können. Betätigen Sie nun die rechte Maustaste über Ihrer ersten Unit und klicken im Anschluss auf den Unterpunkt "Properties" (der vierte Unterpunkt von unten). Alternativ gelangen Sie auch durch einen einfachen Doppelklick der Unit zu dem entsprechenden Eigenschaftsmenü.

Objekteigenschaften der Unit

Hier werden die Objekteigenschaften unserer ersten Unit angezeigt. Im Moment soll uns lediglich die Art der Unit interessieren. Klicken Sie dazu auf das Pull-down Menü "Type" (oben rechts, dort steht "HIDDEN"). Nun können Sie - wie in den Grundlagen dieser Einführungshilfe genauer erläutert - zwischen drei verschiedenen Arten von Neuronen auswählen: Input-, Hidden- und Output-Units. Wählen Sie an dieser Stelle einmal "INPUT" aus und klicken Sie anschließend auf den "OK-Button".

Ihre erste Input-Unit sollte nun einen hellblauen Pfeil oberhalb des Unit-Namens (in unserem Fall "1") enthalten.

Platzieren Sie nun zwei weitere Einheiten und weisen Sie einer Unit den Typ "INPUT", der anderen den Typ "OUTPUT" zu. Positionieren Sie die Output-Unit etwas unterhalb der beiden Input-Einheiten, so dass Sie folgende Darstellung erhalten (siehe Abbildung 31).

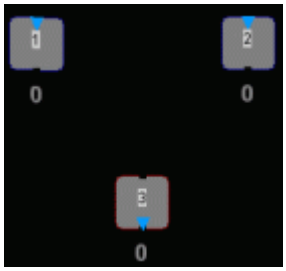


Abbildung 31: 3 Units in MemBrain.

Verbindungen erstellen

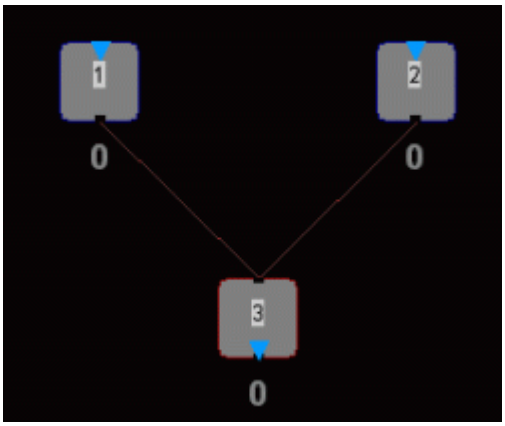


Abbildung 32: 3 miteinander verbundene Units in MemBrain.

Um zwischen zwei Units eine Verbindung zu erstellen, müssen Sie lediglich mit dem Mauspfeil auf die untere, kleine "Lücke" einer Input-Unit fahren, so dass ein dunkelblaues Quadrat erscheint. Drücken Sie dann mit der linken Maustaste auf dieses Quadrat und fahren von dort aus auf die obere, kleine "Lücke" der Output-Unit. Jetzt sollte eine Input-Unit mit einer Output-Unit verbunden sein (die Verbindung wird in rot dargestellt). Verfahren Sie analog für die zweite Input-Unit, so dass beide Input-Neuronen mit dem Output-Neuron verbunden sind. Ihr erstes neuronales Netz müsste dann so aussehen wie in Abbildung 32.

Aktivitätsfunktion auswählen

Aktivitätsfunktion auswählen

Die Auswahl der Aktivitätsfunktion für die einzelnen Units erfolgt wie die Typauswahl (Input, Hidden, Output) der Einheiten im Untermenü "Properties". Doppelklicken Sie dazu auf ihre erstellte Output-Unit. Im mittleren Bereich können Sie im Pull-down-Menü "Activation Function" (dort ist "LOGISTIC" ausgewählt) eine von insgesamt 5 Aktivitätsfunktionen auswählen.

Wenn Sie sich die einzelnen Aktivitätsfunktionen näher ansehen möchten, dann empfiehlt es sich, im "Properties"-Untermenü den Reiter "Customize Activation Function" auszuwählen. Die 5 Aktivitätsfunktionen werden hier auch graphisch visualisiert.

An dieser Stelle behalten wir die ausgewählte Aktivitätsfunktion "LOGISTIC" bei und widmen uns nun der Trainingsphase für das genannte "OR-Gatter".

Trainingsphase

Trainingsphase

Um in der Trainingsphase die Input- und Output-Daten für das "OR-Gatter" einzugeben, wählen Sie im Menü unter "Teach" den "Lesson Editor" aus. Sie erreichen diesen auch alternativ durch folgendes Symbol:  .

Beim "OR-Gatter" soll das Netz als Output eine 0 produzieren, wenn beide Input-Units keine Aktivität aufweisen. In den 3 verbleibenden Fällen (Input-Unit 1 ist aktiv, Input-Unit 2 ist aktiv, beide Input-Units sind gleichzeitig aktiv) soll der Output hingegen aktiv sein (bzw. eine 1 als Aktivitätslevel produzieren). Die rechts stehende Tabelle fasst die 4 verschiedenen Zustände zusammen.

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Abbildung 33: Wahrheitstabelle für das OR-Gatter.

Da also insgesamt 4 "Pattern" vom neuronalen Netz verarbeitet werden sollen, generieren wir zunächst drei weitere Pattern durch Anklicken des "New Pattern"-Buttons auf der rechten Seite. Im mittleren Bereich des "Lesson Editors" sollte bei Ihnen nun "Pattern No: 4 of 4" stehen.

Sie können die einzelnen "Pattern" nun durch Anklicken der Pfeil-Buttons (oben und unten) am rechten Rand auswählen.

Wählen Sie nun Pattern "No. 1 of 4" aus und tragen für die beiden Input-Units ebenso wie für die Output-Unit eine 0 ein. Da sich diese Zahlen schon im ersten Pattern befinden sollten können Sie alternativ auch direkt das zweite Pattern modifizieren. Setzen Sie hier die erste Input-Unit auf 1 ebenso wie die Output-Unit. Für Pattern No. 3 wird die zweite Input-Unit auf 1 gesetzt ebenso wie die Output-Unit. Für das letzte verbleibende, vierte Pattern gilt es, alle Units auf 1 zu setzen.

Wenn Sie die eingegebenen Daten für spätere Zwecke sichern möchten, so können Sie dies im Lesson Editor über dessen Menüpunkt "File" - "Save Lesson" tun. Für das weitere Vorgehen in diesem Exkurs ist dies aber nicht notwendig.

Die restlichen Funktionen im "Lesson Editor" werden an dieser Stelle nicht weiter erläutert, Sie können diesen nun wieder schließen.

Initialisierung der Gewichte mit Zufallswerten

Nun sollte das Netz noch mit Zufallswerten für die Gewichte der Verbindungen und des Aktivierungs-Schwellwerts der Ausgabe-Unit initialisiert werden. Klicken Sie hierzu auf den Menüpunkt "Net" - "Randomize Net". Wenn Sie sich die Verbindungen zwischen den Neuronen einmal genauer anschauen, werden Sie bemerken, dass diese danach die Farbe mehr oder weniger stark verändert haben. Entsprechend der Zufallswerte für die Verbindungsstärke können die Farben von bläulich über grau zu rötlich variieren.

Öffnen Sie nun den "Net Error Viewer" (im Menü unter "Teach" zu finden oder durch Betätigen

des -Buttons).

Wenn Sie nun das Netz durch den Start-Button (oder im Menü "Teach" "Start Teacher (Auto)") trainieren lassen, dann sollte ihr "Net Error Viewer" nach ca. 100 Durchläufen so aussehen, wie auf Abbildung 34.

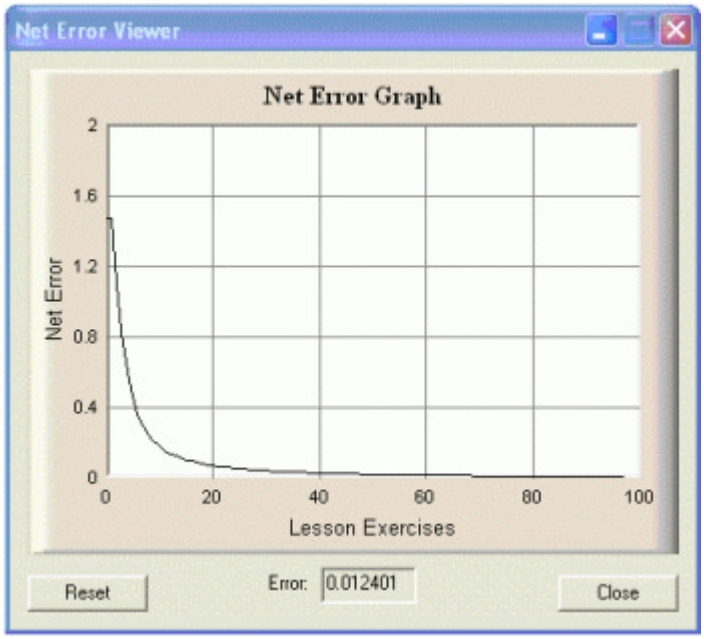


Abbildung 34: Fehlerkurve des neuronalen Netzes für die ersten ca. 100 Durchläufe.

Deutlich erkennbar ist, dass der Fehlerterm des Netzes bereits relativ klein ausfällt und sich im weiteren Verlauf asymptotisch der Null annähern wird. Das neuronale Netz kann also problemlos ein "OR-Gatter" simulieren.

Stoppen Sie nun die Trainingsphase des neuronalen Netzes (oder unter "Teach" "Stop Teacher (Auto)"). In der nun folgenden Testphase soll überprüft werden, ob sich das neuronale Netz tatsächlich wie ein "OR-Gatter" verhält.

Testphase

Testphase

Um zu überprüfen, ob das Netz tatsächlich den richtigen Output für die 4 verschiedenen Input-Muster produziert, öffnen Sie den "Lesson Editor" erneut.

Im "Lesson Editor" können Sie unten links im Bereich "Data to Net" dem neuronalen Netz ein beliebiges Reizmuster ("Pattern") präsentieren. Wählen Sie mit Hilfe der Pfeil-Buttons am rechten Rand ein gewünschtes Reizmuster aus und klicken anschließend auf "Think on Input". Im Hauptbildschirm sollte nun unter ihrer Output-Unit eine Zahl erscheinen. Haben Sie ein Reizmuster gewählt, bei dem eine Input-Unit aktiv ist, dann sollte die Zahl unter der Output-Unit nahe 1 liegen. Handelt es sich hingegen um das Pattern No. 1 (beide Input-Units sind inaktiv), so müsste der Aktivitätslevel der Output-Unit nur geringfügig über 0 liegen. Überprüfen Sie nun alle 4 Reizmuster. Alternativ zum Button "Think on Input" können Sie dazu auch den Button "Think on Next Input" wählen. Dieser präsentiert dem Netz automatisch das jeweils nächste Pattern, so dass Sie nicht manuell mit den Pfeiltasten weiterschalten müssen.

Ggf. können Sie auch in die Trainingsphase zurückspringen und das Netz erneut trainieren lassen.

An dieser Stelle ist die Einstiegshilfe in MemBrain zu Ende. MemBrain bietet jedoch noch zahlreiche weitere Möglichkeiten, die an dieser Stelle nicht erörtert werden konnten. Ich kann Sie nur dazu ermuntern, MemBrain weiter zu verwenden und mit diesem Simulator komplexere neuronale Netze zu generieren. Neben der umfangreichen Hilfefunktion in MemBrain steht Ihnen bei Fragen, Kritik und Anmerkungen zu MemBrain auch Herr Jetter selbst zur Verfügung.

Literaturverzeichnis

- Botvinick, M. M. and Plaut, D. C. (2004). Doing without schema hierarchies: a recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review, 111*, 395-429
- Grossberg, S. (1972). Neural expectation: cerebellar and retinal analogs of cells fired by learnable or unlearned pattern classes. *Kybernetik, 10*, 49-57.
- Gustafsson, L. and Paplinski, A. P. (2004). Self-organization of an artificial neural network subjected to attention shift impairments and familiarity preference, characteristics studied in autism. *Journal of Autism and Developmental Disorders, 34*, 189-198.
- Hebb, D. O. (1949). *The Organization of Behavior.* New York: Wiley.
- Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology (London), 160*, 106-154.
- Kandel, E. R., Schwartz, J. H. und Jessell, T. M. (Hrsg.). (1995). *Neurowissenschaften. Eine Einführung.* Heidelberg: Spektrum Akademischer Verlag.
- Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43*, 59-69.
- Locke, E. A. and Latham, G. P. (1990). *A theory of goal setting and task performance.* Englewood Cliffs: Prentice-Hall.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.
- Popper, K. R. (1996). *Alles Leben ist Problemlösen.* München: Piper.
- Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing* (vol. 2). Cambridge, MA: MIT Press.
- Stanikunas, R., Vaitkevicius, H. and Kulikowski, J. J. (2004). Investigation of color constancy with a neural network. *Neural Networks, 17*, 327-337.
- Von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik, 14*, 85-100.
- Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Harvard: Harvard University.
- Xu, F. and Carey, S. (1996). Infants' Metaphysics: The case of numerical identity. *Cognitive Psychology, 30*, 111-153.