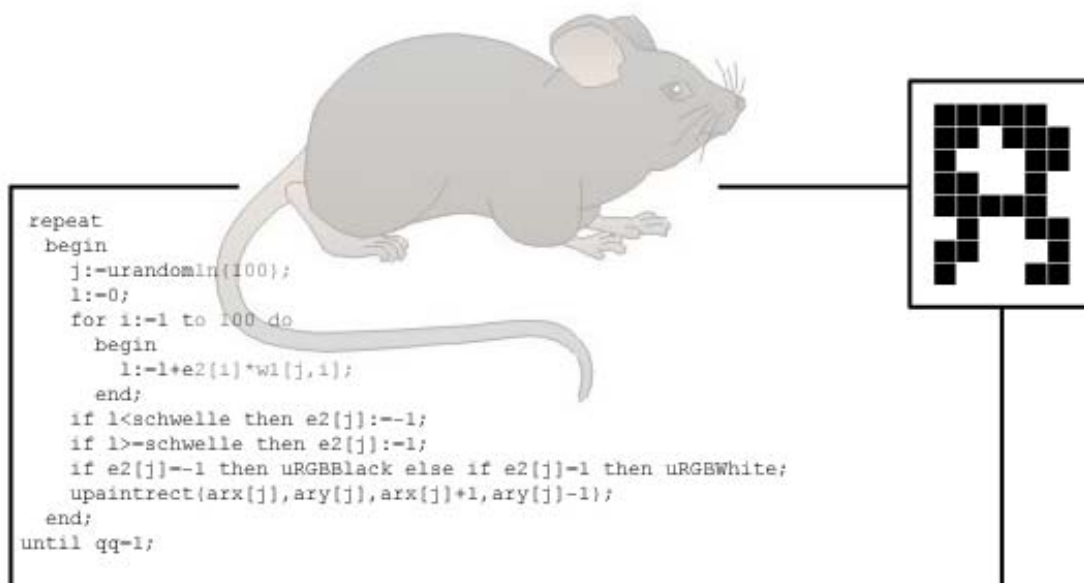


Mustererkennung eines Programms und einer Maus nach dem Modell von Hopfield



Maturaarbeit
Marco Hunziker (Jg. 1983)
Schuljahr 2002/03
Kantonsschule Zug

Betreuer: R. Peterhans, Per H. Antonsen

Inhaltsverzeichnis

1. Vorwort	3
2. Einleitung	4
2.1. Aufbau eines biologischen Neurons	4
2.2. Aufbau eines technischen Neurons	5
2.3. Funktionsweise eines Neurons: biologisch – technisch	5
2.4. Neuronale Netze	7
2.4.1. Mehrschichtige Neuronale Netze	7
2.4.2. Rückgekoppelte Neuronale Netze	7
2.5. Lernregeln/Lernmodelle	8
2.5.1. Delta Regel	8
2.5.2. Hebbsche Lernregel	8
2.5.3. Hopfield Modell	9
2.6. Möglichkeiten Neuronaler Netze	9
2.6.1. Randerkennung	9
2.6.2. Mustererkennung	10
2.6.3. Künstliche Intelligenz	12
2.6.3.1. Was ist künstliche Intelligenz?	12
2.6.3.2. Der Turing Test	12
2.6.3.3. Einsatzgebiete	13
3. Materialien/Methoden	14
3.1. Programm zur Mustererkennung	14
3.1.1. Was kann das Programm?	14
3.1.2. Funktionsweise in einem Flussdiagramm	15
3.1.3. Abstandsgrösse	17
3.2. Praktischer Versuch	17
3.2.1. Ziel des praktischen Versuchs	17
3.2.2. Versuchsaufbau	17
3.2.3. Störfaktoren und deren Behebung	18
3.2.4. Testen der Mäuse	18
3.2.5. Testen des Programms	19
3.2.6. Protokoll	19
4. Resultate	21
4.1. Praktischer Versuch	21
4.1.1. Trainingsresultate	21
4.1.2. Nullkontrolle	21
4.1.3. Testergebnisse	22
4.2. Programm	23
4.2.1. Testläufe	23
4.2.2. Mass Resultate	23

5. Diskussion/Schlussfolgerungen	26
5.1. Relevanz der Resultate	26
5.1.1. Mustererkennungsprogramm	26
5.1.2. Mäuse	26
5.2. Analyse der Resultate	26
5.2.1. Mustererkennungsprogramm	26
5.2.2. Mäuse	26
5.3. Vergleich der Wiedererkennungsmethoden	27
5.3.1. Gegenüberstellung der Resultate	27
5.3.2. Was haben die Mäuse gelernt?	28
5.4. Programmanalyse	28
5.4.1. Fehler im Programm	28
5.4.2. Ist es möglich, ein besseres Modell zu machen?	28
5.4.3. Gibt es ein Mass in Pixel, ab wann das Programm ein Muster wiederkennen kann?	29
5.5. Lernt das Modell von Hopfield?	30
5.6. Vergleich mit anderen Forschungsergebnissen	30
5.7. Ausblick	31
6. Zusammenfassung	32
7. Literaturverzeichnis	33
8. Tabellen – und Abbildungsverzeichnis	34
8.1. Abbildungsverzeichnis	34
8.2. Tabellenverzeichnis	36
9. Glossar/Abkürzungsverzeichnis	37
10. Anhang: Quellcode des Programms	38
11. Anhang: Tabellen	47
11.1. Praktischer Versuch	47
11.1.1. Trainingsresultate	47
11.1.2. Nullkontrolle	48
11.1.3. Testergebnisse	49
11.2. Programm	50
11.2.1. Testresultate	50
11.2.2. Mass Resultate	51

1. Vorwort

Sehr geehrte Damen und Herren

Ich möchte mich bei meinen Betreuern, Rolf Peterhans und Per H. Antonsen, bedanken, die mich während dieser Zeit unterstützt haben. Vielen Dank.

Die Faszination des Gehirns fesselt mich schon, seit ich mich erinnern kann. So stand für mich auch schon von Anfang an fest, dass meine Maturaarbeit sich mit neuronalen Netzen befassen wird; ob technisch oder biologisch stand bis dato noch offen. Anfangs hatte ich die Idee, das Lernen bei Tier und Mensch zu vergleichen. Es wurde der Vorschlag gemacht, das Lernen bei Tieren mit Hilfe der Biene zu untersuchen. Bienen können sehr gut und schnell lernen. Es sollte sich dabei um ein Experiment im Bereich der Mustererkennung handeln. Als die Bienen jedoch nicht mitmachen wollten, entschied ich mich, den gleichen Versuch mit Mäusen durchzuführen, da diese ebenfalls sehr gute Lerner sind. Sie sollten dabei zwei Buchstaben lernen, die sie später wiedererkennen sollten. Aufgrund meiner Programmiervorliebe setzte ich mir das Ziel, ein Programm zu schreiben, das Muster erkennen kann und legte somit den Schwerpunkt meiner Arbeit auf die Informatik.

Um eine Verbindung zwischen dem Programm und den Mäusen herzustellen, wollte ich überprüfen, ob die Mäuse nach dem gleichen Modell lernen, wie das Programm. Es musste eine Verbindung zwischen den beiden Elementen bestehen, da ich sie später vergleichen wollte.

Marco Hunziker

2. Einleitung

2.1. Aufbau eines biologischen Neurons

Das menschliche Gehirn besteht aus zwei halbkugelförmigen Hälften, der sogenannten linken und rechten Hemisphäre, die über einen Balken miteinander verbunden sind. Beide Hemisphären sind gleich aufgebaut und werden in gleiche Teilbereiche unterteilt: Grosshirn, Zwischenhirn, Kleinhirn, Mittelhirn, Nachhirn und das Rückenmark, wobei das Rückenmark nicht wirklich zum Gehirn gehört. Es ist am Gehirn angehängt und zieht sich linienförmig den Rücken hinunter. Das Rückenmark leitet Informationen vom Gehirn zu den Muskeln oder Signale von den Muskeln zum Gehirn. Es gehört nicht zur Informationsverarbeitung, sondern verteilt die Impulse im ganzen Körper. Jeder der Teilbereiche des Gehirns hat seine bestimmten Aufgaben. So ist zum Beispiel der Sitz der Informationsverarbeitung in der Grosshirnrinde. Trotzdem haben all diese Teilbereiche etwas gemeinsam: Sie bestehen letztendlich aus Nervenzellen, auch Neuronen genannt, welche die kleinste Komponente des Gehirns bilden. Das heutige Gehirn besteht aus etwa 10^{10} Neuronen. [Lindenmair 1995]

Neuronen kommen nicht einzeln vor, sondern sind untereinander über die Synapsen zu einem neuronalen Netz zusammengeschlossen, was die Kommunikation der Neuronen untereinander ermöglicht. Die Kommunikation äussert sich in elektrischen Impulsen, die durch eine unterschiedliche Ionenkonzentration im Neuron und extrazellulär entstehen. Die Kommunikation ermöglicht das Koordinieren einer Aktivität. Das erste Mal wurde ein Nervensystem, das aus verknüpften Neuronen bestand, vor über 500 Millionen Jahren bei Tieren wie Seeanemonen und Quallen entwickelt. Die Muskelbewegungen, die eine Qualle benötigt, um aktiv auf Nahrung zuzuschwimmen und diese zu erbeuten, müssen koordiniert sein. Doch dies ist erst möglich, wenn Nervenzellen vernetzt sind und so miteinander kommunizieren können. In den Jahrillionen seither wurde das Gehirn mehrfach weiterentwickelt, bis es schliesslich das Volumen des Gehirns des heutigen Menschen erreicht hatte. [Lindenmair 1995]

Ein Neuron besteht aus einem Zellkörper, auch Soma genannt, Axon, Synapse und Dendriten. Das Axon hat die Aufgabe, das Ausgabesignal des Neurons weiterzuleiten. Am Ende des langgezogenen Axons gibt es viele Verzweigungen, sie alle führen zu Synapsen. Die Synapse bildet die Verbindung zwischen zwei Neuronen. Es gibt dabei zwei Sorten von Synapsen: elektrische und chemische. Die elektrischen Synapsen leiten den elektrischen Impuls direkt weiter, da die zwei benachbarten Nervenzellen sehr engen Zellkontakt haben und somit eine direkte Verbindung besitzen. Die chemischen Synapsen bestehen aus zwei Teilen, dem postsynaptischen Teil und dem präsynaptischen Teil. Die Postsynapse liegt am Anfang eines Dendriten und die Präsynapse normalerweise am Ende des Axons. Der Impuls kann also nicht direkt übertragen werden, sondern muss durch sogenannte Neurotransmitter, chemische Botenstoffe, übertragen werden. Die Dendriten befinden sich am hinteren Teil der Nervenzelle und leiten das Ausgangssignal des vorhergehenden Neurons weiter zum Zellkörper. [Gundelfinger 2002]

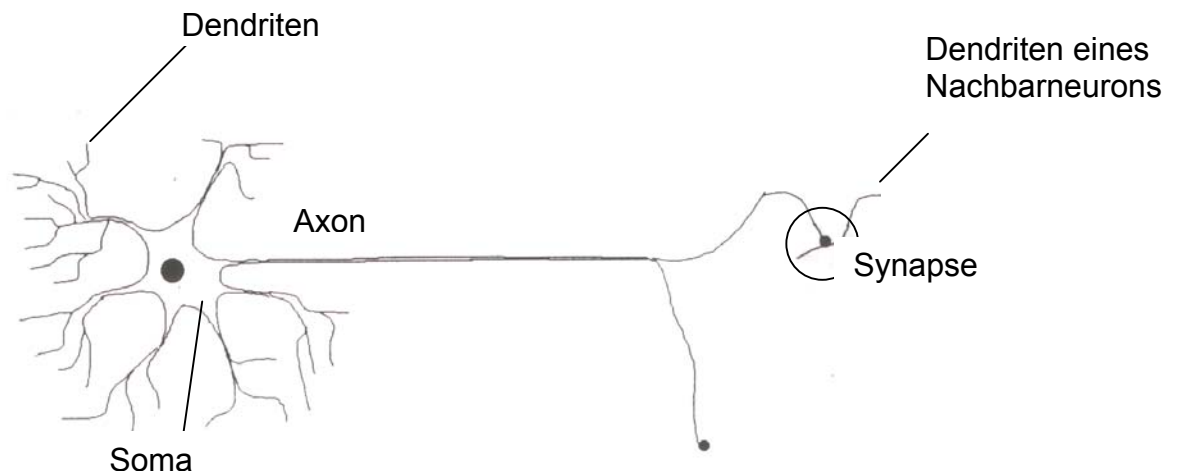


Abb. 1: Schematischer Aufbau eines Neurons

2.2. Aufbau eines technischen Neurons

In der Informatik wurde versucht, ein Modell zu entwickeln, das ein biologisches Neuron mathematisch beschreibt und somit auch die gleichen charakteristischen Eigenschaften übernimmt. Die Grundidee war, dass man eine künstliche Intelligenz herstellen wollte, die völlig unabhängig denken kann. Um eine künstliche Intelligenz herzustellen, musste jedoch zuerst das Modell des Neurons erstellt werden.

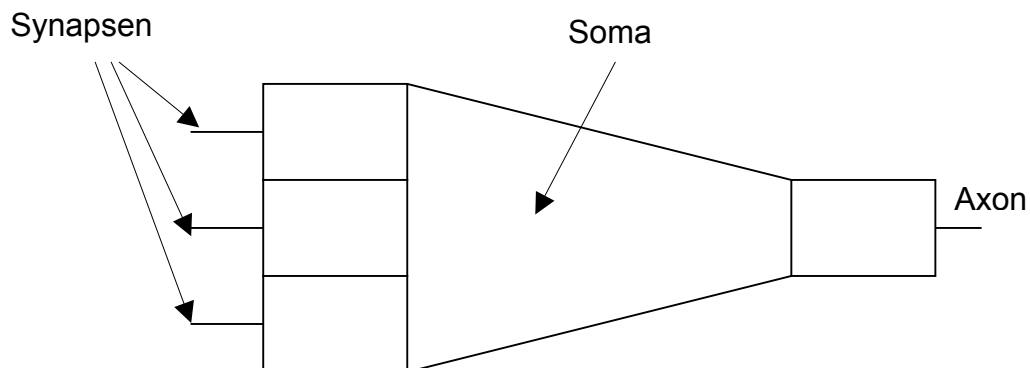


Abb.2: Mathematisches Modell eines Neurons im Vergleich mit einem biologischen Neuron.

Auch im Modell des Neurons lassen sich die verschiedenen Bereiche eines Neurons erkennen: Soma, Axon, Synapsen. Die Ausgabe des Neurons wird bestimmt durch die Ausgabefunktion. Die Inputs sind Eingabewerte, die neu dazukommen, oder Ausgabesignale von Nachbarneuronen. [Lindenmair 1995]

2.3. Funktionsweise eines Neurons: biologisch – technisch

Es wird davon ausgegangen, dass ein einzelnes biologisches Neuron im Durchschnitt 10'000 synaptische Verbindungen zu anderen Neuronen eingeht. Jedes der Nachbarneuronen sendet einen elektrischen Impuls aus.

Das Neuron empfängt theoretisch alle dieser 10'000 Impulsen und verarbeitet sie. Das bedeutet sie werden zu einem Summensignal aufaddiert. Erst wenn das Summensignal die Reizschwelle (ca. -70 mV) überschreitet, sendet das Neuron einen elektrischen Impuls aus. Man sagt auch, das Neuron feuert. Es ist klar, wenn jedes eintreffende Signal verstärkt weitergeleitet werden würde, wäre die Arbeitsweise unseres Gehirns instabil. Deshalb muss es einen Mechanismus geben, der die Signale unterdrückt oder abschwächt. Es wurde herausgefunden, dass es zwei Arten von Synapsen gibt: erregende und hemmende. Der Mensch hat etwa 70-80% erregende und etwa 20-30% hemmende Synapsen. Eine hemmende Synapse schwächt ein Signal so ab, dass die Reizschwelle nur sehr schwer erreicht wird. Wenn sie jedoch erregend wirkt, wird das Signal verstärkt und die Reizschwelle kann somit schneller erreicht werden. In die Mathematik übersetzt gibt die hemmende Synapse dem Signal ein negatives Vorzeichen, die erregende ein positives. Wird ein Signal mit einem negativen Vorzeichen addiert mit einem anderen Signal mit positivem Vorzeichen, so löschen sie sich gegenseitig aus, wenn ihre Summe 0 ergibt. Ausserdem werden sie nur aufaddiert, wenn sie gleichzeitig beim Neuron eintreffen. [Berger 2002]

Analog der biologischen Nervenzelle funktioniert auch das künstliche oder mathematische Neuron. Anstelle von hemmenden und erregenden Synapsen, werden beim künstlichen Neuron sogenannte Eingangsgewichte gesetzt, die negative (=hemmende) oder positive (=erregende) Zahlen annehmen können.

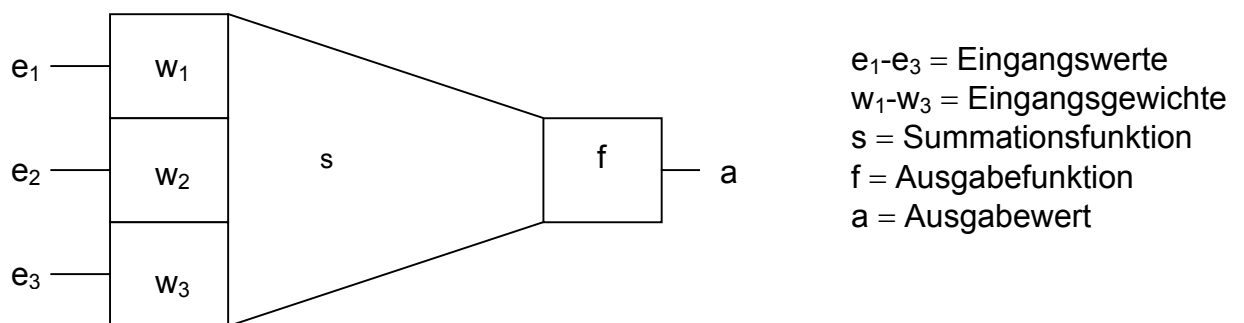


Abb. 3: Mathematisches Modell eines Neurons mit drei Eingabewerten.

Die Eingangswerte e_1-e_3 werden mit einer Summationsfunktion zu einer Summe aufaddiert. Die einfachste Summationsfunktion ist die Bildung einer gewichteten Summe: $\sum = e_1 * w_1 + e_2 * w_2 + e_3 * w_3$. Die Gewichte w_1-w_3 sind reelle Zahlen, welche die Stärke jeder Verbindung des neuronalen Netzes beschreiben. Durch die Ausgabefunktion f wird der Ausgabewert a dieses Neurons bestimmt. Dieser Ausgabewert kann weiter verwendet werden, als Eingabewert eines Nachbarneurons oder als Ausgabe des neuronalen Netzes. Am einfachsten wird der Ausgabewert mit der Schwellenwertfunktion bestimmt: Wenn die Summe den Schwellenwert erreicht oder ihn übersteigt, dann wird der Ausgabefunktion der Wert 1 (=aktiv) zugeordnet, ansonsten 0 (=inaktiv).

Natürlich gibt es noch viele andere Funktionen, die den Ausgabewert a bestimmen können, jedoch möchte ich nur auf die am häufigsten verwendeten eingehen. Das sind die Schwellenwert-, die Sigmoid- und die

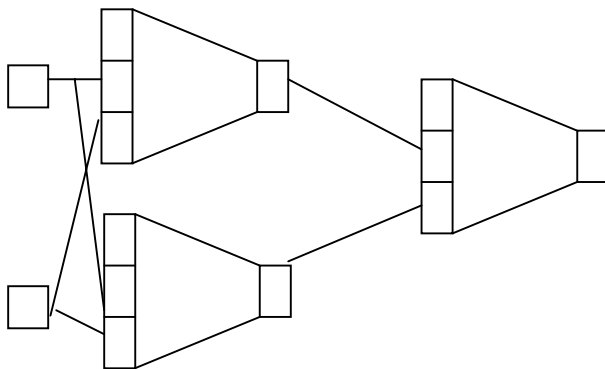
lineare Funktion. Die sigmoide Funktion gibt einen kontinuierlichen Ausgabewert zurück und lautet folgendermassen: $f(x)=1/(1+e^{-k*x})$, die lineare ist die einfachste und dadurch auch die am wenigsten mächtige Funktion: $f(x)=x$. [Lindenmair 1995]

2.4. Neuronale Netze

Neuronen kommen nie alleine vor, sie verbinden sich immer zu mehrschichtigen neuronalen Netzen.

2.4.1. Mehrschichtige neuronale Netze

Ein neuronales Netz besteht aus mehreren miteinander verbundenen Neuronen, die weiter in drei Schichten unterteilt werden: Input -, Verbindungs - und Outputschicht. Die Inputschicht liefert die Eingabewerte, die Outputschicht die Ausgabewerte und die Verbindungsschicht leitet die Eingabewerte weiter zu den anliegenden Neuronen. Diese Struktur eines neuronalen Netzes wird auch Perzeptron genannt. Mit dem Perzeptron stellten McCulloch und Pitts einen ersten Ansatz von neuronalen Netzwerken vor. Als Aktivierungsfunktion dient dem Perzeptron eine Schwellenfunktion. Der Informationsfluss des neuronalen Netzes muss dabei immer vorwärts gerichtet sein, d.h. die Daten fließen von der Eingabeschicht zur Ausgabeschicht, ohne dabei untereinander gegenseitige Verbindungen oder Rückkopplungen einzugehen.



Ein neuronales Netz wird immer der Aufgabenstellung angepasst. Zum Beispiel braucht man für die Randerkennung nur ein einzelnes Neuron, wobei man um das XOR – Problem (Exklusiv oder) zu lösen ein neuronales Netz mit der in Abbildung 4 gezeigten Struktur benötigt. [Reif 2002]

Abb. 4: Neuronales Netz mit der Struktur 2-2-1.

2.4.2. Rückgekoppelte neuronale Netze

Unser Gehirn ist so aufgebaut, dass die Neuronen untereinander kommunizieren können. In den bisher erläuterten Modellen funktioniert dies nicht, da sie alle eine vorwärtsgerichtete Netzstruktur aufweisen. Die Informations-

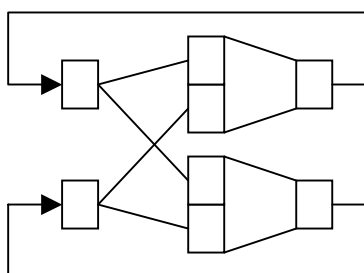


Abb. 5: Rückgekoppeltes neuronales Netz

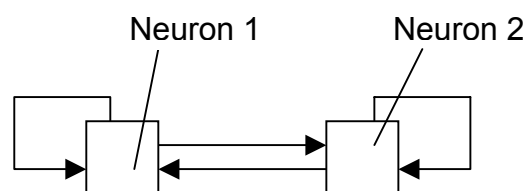


Abb. 6: Rückgekoppeltes neuronales Netz mit veränderter Anordnung

verarbeitung läuft daher nur von der Eingabe - zur Ausgabeschicht. Um eine wechselseitige Kommunikation zu ermöglichen, wird die Ausgabeschicht mit der Eingabeschicht verbunden, wodurch die Neuronen rückgekoppelt werden. Wenn man die Anordnung des neuronalen Netzes aus Abb. 5 verändert darstellt (Abb. 6), wird verdeutlicht, dass nun die gegenseitige und wechselhafte Kommunikation gewährleistet ist. [Lindenmair 1995]

2.5. Lernregeln/Lernmodelle

Biologische neuronale Netze können lernen, wenn man sie darauf trainiert. Lernen heisst, die Neuronen verstärken oder schwächen die Beziehungen zu anderen Neuronen. Das künstliche Neuron besitzt ebenfalls diese Eigenschaft, doch dazu wird eine Lernregel benötigt. Sie verändert die Gewichte der einzelnen Verbindungen, um so ein gewünschtes Ausgangssignal zu erhalten. Die Eingangssignale werden während der Lernphase mit den Gewichtungsfaktoren bewertet und weitergeleitet. Wie sie bewertet werden, ist abhängig von der angewendeten Lernregel.

2.5.1. Delta Regel

Die Delta Regel verwendet man für das überwachte Lernen. Das bedeutet das Lernen wird von einem virtuellen Lehrer überwacht. Nur er kennt die Ausgabe, die sich ergeben soll. Bei der Delta Regel werden der Ausgabewert a_{soll} und der aktuelle Ausgabewert a_{ist} verglichen. Wenn diese verschieden sind, sind die Eingangsgewichte so zu verändern, dass die beiden Ausgabewerte übereinstimmen. Das erreicht man mit den folgenden beiden Formeln:

$$1) \Delta = a_{\text{soll}} - a_{\text{ist}}$$

$$2) w_i^{\text{neu}} = w_i^{\text{alt}} + \alpha * \Delta * e_i$$

Zuerst wird die Differenz der beiden Ausgabewerten gebildet. Dann werden die neuen Gewichte mit der Lernrate α , der Differenz Δ und dem Eingabewert e_i neu errechnet. Das Neuron berechnet nun die Ausgabe wieder neu und die beiden Schritte zur Gewichtsänderung werden wiederholt, bis die Ausgabewerte übereinstimmen. [Lindenmair 1995]

2.5.2. Hebbsche Regel

Die Hebbsche Lernregel wird unüberwachtes oder selbstständiges lernen genannt. Sie stellt Beziehungen in Form von Synapsengewichten, zwischen zwei Neuronen x und y her. Wenn beide Neuronen gleichzeitig aktiv sind, erhöht sich die Stärke ihrer Verbindung. Sie verändern selbstständig die Gewichte mit der Formel:

$$w_{xy}(t+1) = w_{xy}(t) + k * a_x * a_y$$

Das neue Gewicht zum Zeitpunkt t+1 wird mit Hilfe einer Lernrate k, deren Wert immer zwischen 0 und 1 liegt, der Aktivität des Neurons X (a_x) und

der Aktivität des Neurons Y (a_y) berechnet. Diese Berechnung folgt bei jedem Lernschritt. [Lindenmair 1995]

2.5.3. Hopfield Modell

Hopfield ist gelernter Physiker und hat mit seinem Modell von 1982 eine Verbindung zwischen neuronalen Netzen und physikalischen Modellen geschaffen. Das Modell ist ein vollständig rückgekoppeltes, asynchrones neuronales Netz, das als Assoziativspeicher arbeitet.

Mit Hilfe des Hopfield Modells ist ein vorher gelerntes Muster, nach einer Veränderung rekonstruierbar. Für jeden Bildpunkt wird dabei ein Neuron erzeugt. Alle Neuronen sind miteinander verbunden um zu kommunizieren und schliesslich entscheiden zu können, ob ein Pixel weiss oder schwarz ist. Beim Lernen eines Musters werden mit Hilfe der Hebbschen Lernregel sogenannte Synapsengewichte festgelegt. Wird nun ein Muster verändert, so ändern sich nur die Aktivitätszustände der Neuronen, nicht aber die vorher festgelegten Beziehungen. Deshalb kann mit Hilfe dieser Beziehungen das zuvor gelernte Muster rekonstruiert werden. Das Modell von Hopfield wird unter Punkt 2.5.2. am Modell der Mustererkennung ausführlicher erklärt. [Lindenmair 1995]

2.6. Möglichkeiten Neuronaler Netze

Neuronale Netze können sehr vielfältig eingesetzt werden. Sie sind überall dort von Nutzen, wo Vorgänge überdacht werden müssen und immer gleich (systematisch) entschieden werden muss. Bei einfachen Anwendungen werden schon heute Computer eingesetzt. Anhand der folgenden Beispielanwendungen möchte ich zeigen, wie vielfältig neuronale Netze eingesetzt werden können.

2.6.1. Randerkennung

Bei der Randerkennung geht es darum, den Rand einer beliebigen Figur zu erkennen. Dabei werden bei jedem Pixel die acht umliegenden Pixel verglichen. Wenn einer davon weiss ist, dann liegt das Pixel am Rand einer Figur und bleibt somit erhalten. Sind jedoch alle umliegenden Pixel schwarz, wird das aktuelle Pixel herausgelöscht, da es sich im innern der Figur befindet. Für die Randerkennung ist nur ein einzelnes Neuron nötig. Jedem Pixel, das vor der Randerkennung die Farbe schwarz hat, wird der Wert 0 zugeordnet. Ebenso bekommen alle weissen Pixel den Wert -1 . Diese Werte bilden die Eingabewerte. Nun werden die Eingabewerte und die Gewichte w_1 - w_9 miteinander multipliziert und anschliessend aufaddiert. Die Ausgabefunktion ist eine Schwellenwertfunktion. Das heisst, wenn der erreichte Wert die Schwelle von -0.5 übersteigt, bekommt dieses Pixel den Wert 1 (=ausserhalb oder innerhalb der Figur). Wurde der Wert nicht überstiegen, bekommt das Pixel den Wert 0 (=am Rand der Figur). [Lindenmair 1995]

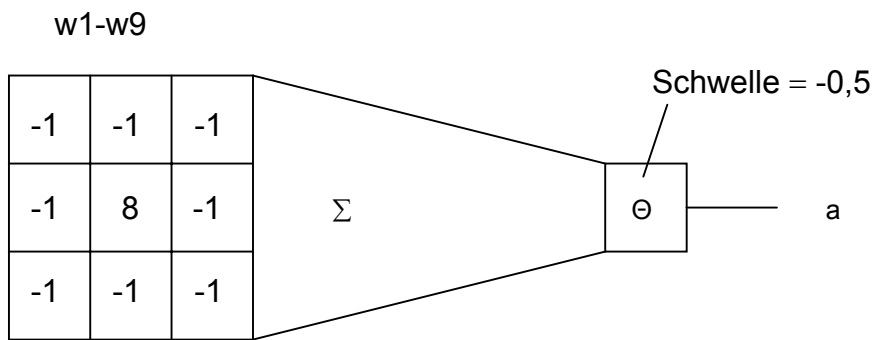


Abb. 7: Randerkennungsneuron

2.6.2. Mustererkennung

Bei der Mustererkennung geht es darum, ein verrauschtes oder unvollständiges Muster, das zuvor gelernt wurde, mit Hilfe des Modells von Hopfield wiederzuerkennen. Sein Modell funktioniert in zwei Phasen:

1. Lernphase

In der Lernphase werden mit Hilfe einer bipolaren Funktion die Aktivitätszustände der einzelnen Neuronen festgelegt. Ist ein Bildpunkt weiss, so erhält er den Aktivitätszustand +1 (=aktiv), ist er schwarz wird ihm der Wert -1 (=inaktiv) zugeordnet. Für jedes Pixel wird ein Neuron benötigt. Die bipolare Ausgabefunktion gibt nur die Werte -1 und +1 zurück. Die Schwelle hat den Wert 0. Ist die Summe grösser oder gleich gross wie die Schwelle, erhält das Neuron den Wert +1. Ist sie kleiner, wird ihr der Aktivitätszustand -1 zugeordnet.

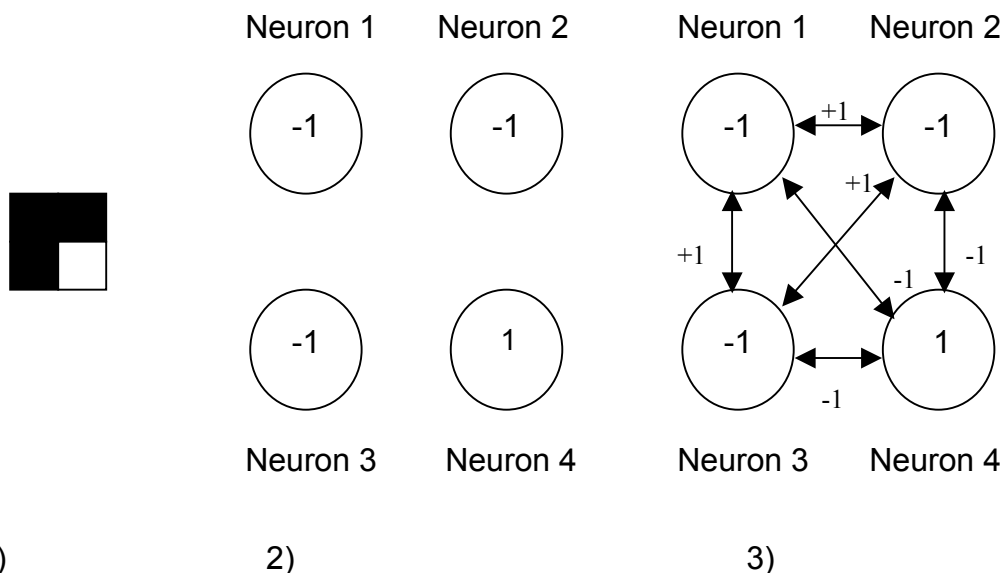


Abb. 8: Zum zu lernenden Muster (1) werden die Aktivitätszustände bestimmt (2) sowie deren dazugehörigen Synapsengewichte (3).

Nach dieser Zuordnung werden die Synapsengewichte festgelegt, welche die Beziehung zwischen zwei Neuronen beschreiben. Haben die beiden Neuronen den gleichen Aktivitätszustand, wird als Synapsengewicht der Wert $+1$ (=befriedigte Synapse) festgelegt, haben sie verschiedene, so wird ihnen das Gewicht -1 (=frustrierte Synapse) zugeordnet. Da das Neuron rückgekoppelt ist, erhält die Verbindung zu sich selbst den Wert 0 . Diese Zahlen werden in einer sogenannten Gewichtsmatrix, die symmetrisch aufgebaut ist, gespeichert. Das Lernen eines Musters bedeutet eigentlich nur, dass die Synapsengewichte festgelegt wurden.

2. Erkennungsphase

Wird das Muster verändert, ändern sich, wie bereits erwähnt, die Aktivitätszustände der Neuronen, nicht aber die Beziehungen zwischen den einzelnen Neuronen. Um zum gelernten Aktivitätszustand zurückzukommen, werden die neuen Aktivitätszustände eines zuvor mit einer Zufallszahl ermittelten Neurons ausgerechnet. Mit einer bipolaren Funktion werden bei jedem Durchlauf die Aktivitätszustände des aktuellen Neurons angepasst. Dieser Vorgang wird wiederholt, bis sich der Netzzustand stabilisiert hat. Abb. 9 zeigt, wie die Erkennungsphase verläuft und bezieht sich auf das Beispiel aus Abbildung 8. Das Muster, das vorher gelernt wurde, ist verändert worden; die Aktivitätszustände unterscheiden sich von den zuvor gelernten.

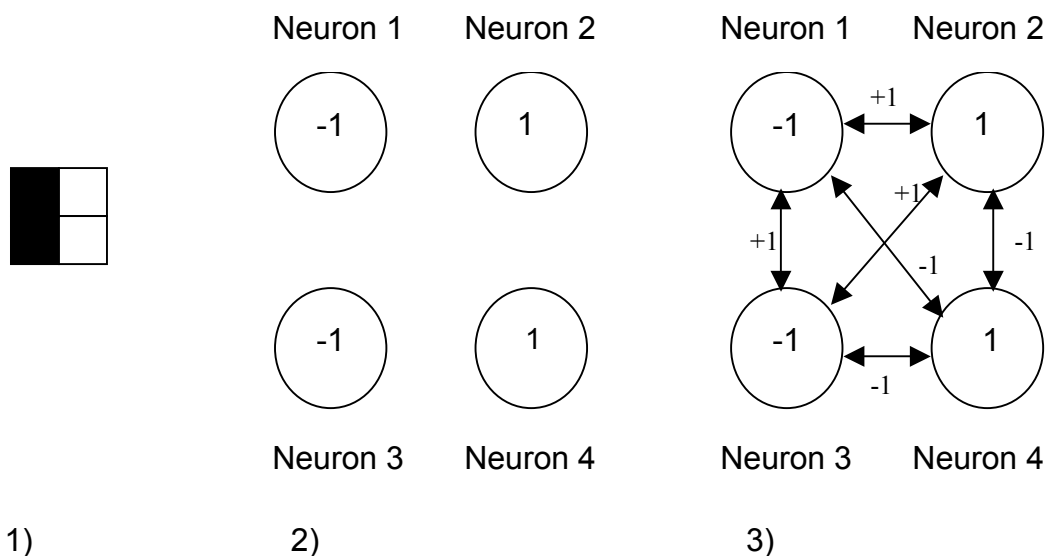


Abb. 9: 1) Das veränderte Neuron besitzt neue Aktivitätszustände (2). Die Gewichtsmatrix bleibt unverändert (3).

Als Beispiel rechnen wir einmal das ganze für die Zufallszahl 2 durch. Der Aktivitätszustand eines Neurons wird berechnet, indem man die Aktivitätszustände aller anderen Neuronen multipliziert mit ihrem Synapsengewicht zum Neuron, das neu berechnet werden soll. Anschließend werden alle Produkte aufsummiert und anhand der bipolaren Ausgabefunktion die Aktivitätszustände bestimmt.

$$a_2 = f(a_1 \cdot w_{21} + a_3 \cdot w_{23} + a_4 \cdot w_{24}) = f(-3) = -1$$

Der Aktivitätszustand von Neuron 2 sollte nach diesen Berechnungen -1 sein und nicht, $+1$, wie nach der Veränderung. Der Wert wird demzufolge auf den richtigen geändert. Da der Netzzustand somit stabil ist, werden die Berechnungen abgebrochen. Das Muster wurde wieder erkannt.

Wenn mehrere Muster gelernt werden sollen, werden die Gewichtsmatrizen aufsummiert. Die Synapsengewichte lösen sich gegenseitig auf oder sie gehen in ein Extrema über. Das heisst, wenn zwei Neuronen ein Synapsengewicht mit einem maximalen positiven oder negativen Wert besitzen, dann möchten diese beiden Neuronen umso stärker die gleichen oder verschiedenen Aktivitätszustände annehmen. [Lindenmair 1995]

2.6.3. Künstliche Intelligenz

2.6.3.1. Was ist künstliche Intelligenz

Der Mensch besitzt eine Intelligenz, die sich im Denken, Vorstellen, Wissen, Anwendung des Wissens, Logik, etc. äussert. Man versuchte mittels neuronalen Netzen eine künstliche Intelligenz zu erzeugen, jedoch gelang es bis heute nicht; nur einfachere Probleme konnten bisher damit gelöst werden.

Es gibt keine einheitliche Definition für sie - Winston versuchte 1993 eine beschreibende Annäherung zu machen:

„Artificial Intelligence is the study of ideas which enable computers to do things, that make people seem intelligent. [...] The central goals of Artificial Intelligence are to make computers more useful and to understand the principles which make intelligence possible.“ [Winston 1993].

Künstliche Intelligenz ist also im vereinfachten Sinne etwas von Menschen geschaffenes, das denken kann. Es ist zum Beispiel eine Maschine, die ein Problem lösen kann, ohne dazu menschliche Hilfe zu benötigen. [Reif 2002]

2.6.3.2. Der Turing Test

Der englische Mathematiker Alan Turing befasste sich mit der Frage, ob Maschinen denken könnten. Er schlug einen Test vor, der die Intelligenz einer Maschine nachweisen soll. Der Test funktioniert folgendermassen: Es gibt drei Spieler A,B,C. Der Spieler A ist der Richter, B ein Mensch und C ist die zu prüfende Maschine. Alle drei Spieler befinden sich in einem eigenen Raum und können mittels Terminals miteinander kommunizieren. Kein Spieler kennt die Identitäten der anderen. Die Aufgabe des Richters ist es anhand von geführten Dialogen herauszufinden, welcher Spieler die Person und welcher der Computer ist. Gelingt es dem Computer seine Identität zu verbergen, so dass der

Richter ihn nicht erkennt, wird der Computer als intelligent eingestuft.
[Reif 2002]

2.6.3.3. Einsatzgebiete

Das Spektrum der Einsatzgebiete dieses komplizierten Forschungsgebietes ist sehr vielfältig. Die Maschinen mit künstlicher Intelligenz können nicht allzu komplexe Aufgaben übernehmen. Hier zwei Beispiele für die Einsatzgebiete [Reif 2002]:

Sprachverstehen: Systeme, die unsere Sprache "verstehen", erlauben es, mit ihnen in der natürlichen Sprache zu kommunizieren.

Wahrnehmung: Die Forschung versucht unsere Sinne nachzubilden, wie sehen, hören, Bilderkennung sowie Spracherkennung.

3. Materialien/Methoden

In diesem Abschnitt möchte ich erklären, wie der praktische Versuche und das Programm aufgebaut sind.

3.1. Programm zur Mustererkennung

3.1.1. Was kann das Programm?

Das Programm dient der Mustererkennung. Ziel ist, dass der Computer ein Muster das gelernt wurde, wiedererkennt, wenn es verrauscht aufgeprägt wurde. Das Programm ist in folgende Phasen unterteilt:

Muster: Es müssen nicht die vorgegebenen Buchstaben verwendet werden, es können auch Muster oder Buchstaben selber entworfen werden. Die Buchstaben sind sich sehr ähnlich und deshalb ist die Wahrscheinlichkeit, dass der verrauschte Buchstabe vollständig wiedererkannt wird, viel kleiner. Aus meiner Erfahrung empfehle ich deshalb, selber gemachte Muster zu gebrauchen.

Lernen: Das Programm kann mehrere Muster nacheinander lernen. Nach einem erfolgreichen Lernschritt wird ein gelerntes Muster nicht mehr „vergessen“. Soll etwas dennoch „vergessen“ werden, muss das Programm neu gestartet werden. Bei einem Neustart sind alle gelernten Muster „vergessen“, sie können also nicht gespeichert werden. Das Programm kann nach folgenden Schritten wieder lernen: nach dem lernen, wenn ein Muster selber entworfen wurde und wenn ein Muster wiedererkannt wurde.

Verändern: Nachdem ein Buchstabe gelernt wurde, sollte er verändert werden und zwar so, dass seine Umrisse noch zu erkennen sind. Als Richtlinie gilt: Wenn die Person, die das Programm bedient das Muster noch erkennt, ist alles in Ordnung, da der Mensch die Norm ist.

Beispiel:

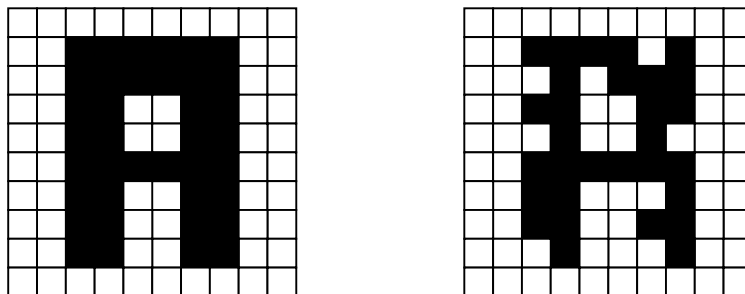


Abb. 10: Gegenüberstellung des Buchstabens A und dessen Konturen.

Erkennung: Nachdem das Muster verändert wurde, versucht das Programm dieses wieder zu erkennen. Es ist jedoch möglich, dass das Muster nicht vollständig wiedererkannt wird, da eine kleine Fehlertoleranz eingebaut wurde, die sich verstärkt, je länger es dauert, bis das Muster erkannt wird.

Soll das Programm ein Muster erkennen, obschon noch keines gelernt wurde, so kommt es zu einer Fehlermeldung

3.1.2. Funktionsweise in einem Flussdiagramm

Wie das Programm genau funktioniert, möchte ich anhand eines Flussdiagramms aufzeigen. Ich werde dabei nur auf das Lernen und auf das Wiedererkennen eingehen, da diese Programmteile zusammen das Kernstück meines Programms bilden.

Lernphase

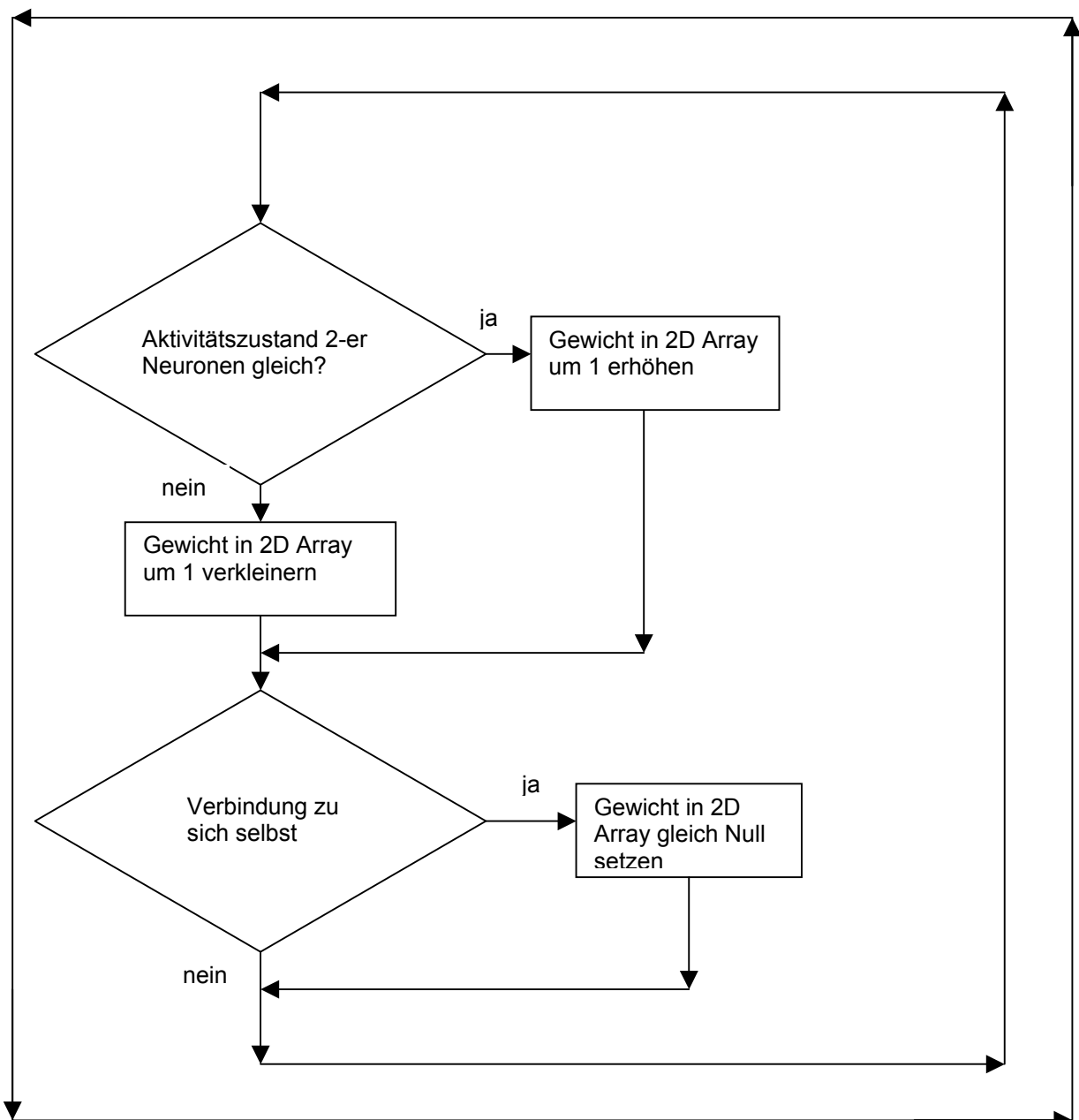


Abb. 11: Flussdiagramm, das den Algorithmus der Lernphase im Programm darstellt.

Die Lernphase besteht, wie hier dargestellt, aus zwei For-Schleifen. Die innere Schleife hat die Variable j , die äussere die Variable i . Für jeden

Durchlauf der äusseren Schleife, wird die innere n-mal ausgeführt, wobei die äussere ebenfalls n-mal durchlaufen wird. Dabei gibt es n-mal eine Abfrage, ob zwei Aktivitätszustände von zwei Neuronen gleich oder unterschiedlich sind. Sind sie gleich, so wird deren Wert an der Stelle ij im Array um 1 erhöht, ansonsten um 1 verringert. Die Verbindung eines Neurons zu sich selbst existiert nicht, daher wird der Wert 0 gespeichert.

Wiedererkennungsphase

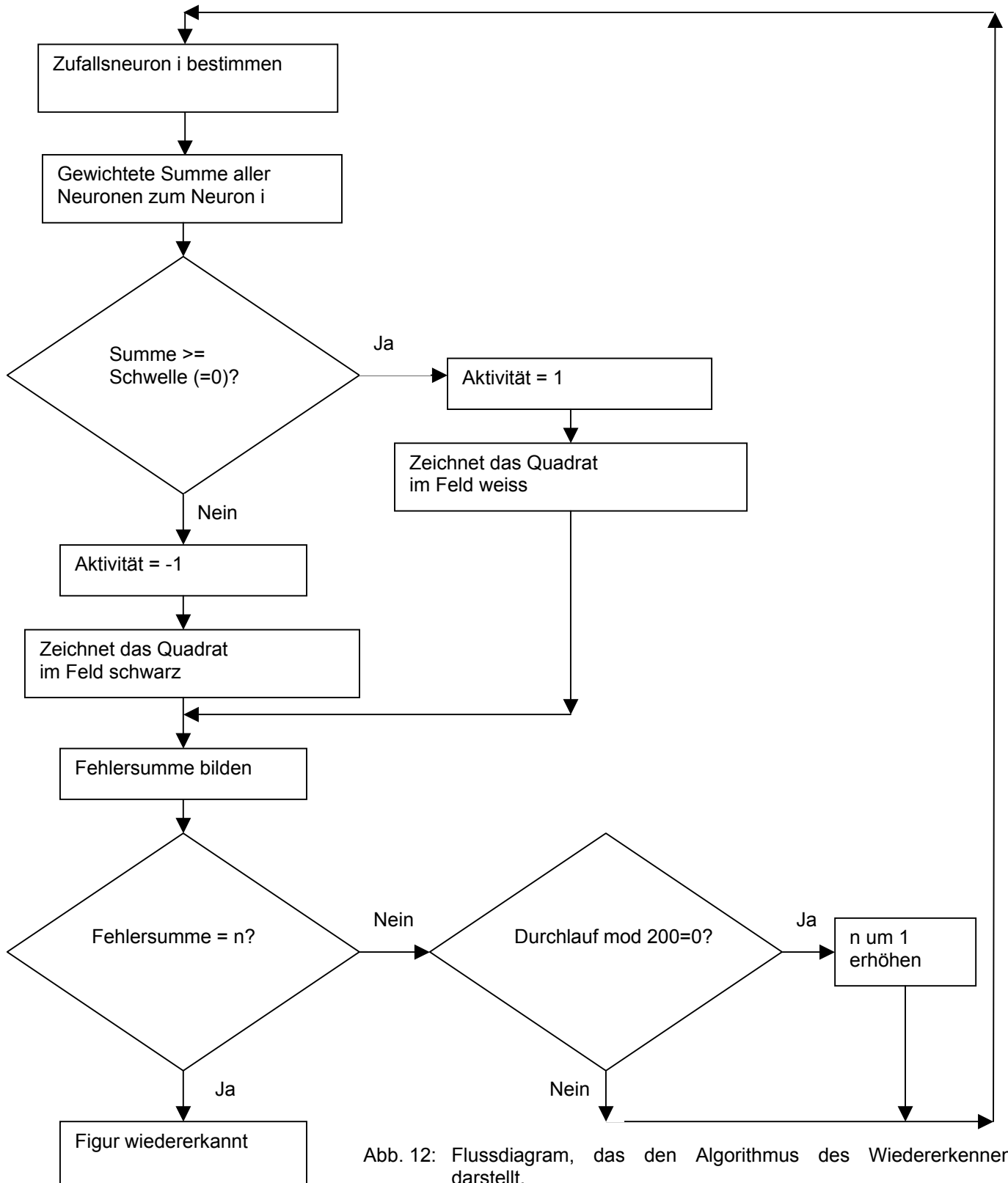


Abb. 12: Flussdiagramm, das den Algorithmus des Wiedererkennens darstellt.

Wie aus der Abb. 12 zu entnehmen ist, wird die Wiedererkennungsphase aus einer Repeat-Until-Schleife gebildet und wird so lange ausgeführt, bis die Figur wiedererkannt wurde.

Am Anfang wird ein Zufallsneuron ausgewählt und es wird eine gewichtete Summe gebildet (vgl. Kap. 2.5.2.). Mit einer bipolaren Funktion werden die neuen Aktivitätszustände bestimmt und dementsprechend die Pixel im Feld neu gefärbt. Anschliessend wird eine Fehlersumme gebildet. Wenn ein Aktivitätszustand im Vergleich zum gelernten unterschiedlich ist, wird ein Strafpunkt vergeben. Diese Strafpunkte werden aufsummiert und ergeben die Fehlersumme. Die Fehlertoleranz beträgt am Anfang 0. Nach jeweils 200 Schleifendurchläufen, wird sie (ab) um 1 erhöht. Das heisst, die Schleife muss nur noch so lange ausgeführt werden, bis eine Fehlertoleranz von ab erreicht ist, dann wird die Wiedererkennungsphase beendet.

3.1.3. Abstandsgrösse

Ich habe mir die Frage gestellt, ob es eine Grösse gibt, bis zu der das gelernte Muster noch erkannt werden kann und danach nicht mehr. Die Grösse bezeichnet die Anzahl veränderter Pixel nach dem Verändern. Ich mache mit dem Programm verschiedene Versuche, mit jeweils unterschiedlicher Anzahl Pixel, zuerst 16, dann 100. Beim 16 Pixel Versuch werde ich dem Programm unterschiedlich viele, zwischen 1 und 3, Muster aufprägen und anschliessend spielt das Programm alle möglichen Kombinationen von Veränderungen durch, insgesamt gibt es 65536 Möglichkeiten. Bei der 100 Pixel Variante verläuft das ganze genauso. Hierbei gäbe es 2^{100} Veränderungsmöglichkeiten, dies ist rechentechnisch zu aufwändig, deshalb werden nur einige zufällig veränderte Kombinationen untersucht.

3.2. Praktischer Versuch

Mit diesem Versuch will ich herausfinden, ob Labormäuse nach dem gleichen Modell ein Muster erkennen, wie das Programm.

3.2.1. Ziel des praktischen Versuchs

Das Ziel ist, dass alle Mäuse die Buchstaben A und C lernen. Um dies zu erreichen, brauchen sie eine Motivation, etwas, das sie veranlasst, die Buchstaben A und C zu unterscheiden; ich gebe sie ihnen in Form von Futter. Bei jeder Mahlzeit lasse ich sie im Versuchsaufbau essen, damit sie nach einiger Zeit lernen, bei welchem Buchstaben sich das Futter befindet.

3.2.2. Versuchsaufbau

Der Versuchsaufbau gleicht der Form eines Y. Der hintere Teil besteht aus 2 Kammern, die mit den Buchstaben O,C,X und A beschriftet sind. Welche Buchstaben in welcher Kombination angeordnet werden, ist zufällig. Es gibt zwei Bedingungen: Niemals dürfen A und C oder X und O gleichzeitig die Vorderseite der Kammern belegen, da sonst die zu lernenden Buchstaben

auf einmal gezeigt werden bzw keiner vorhanden sein würde, was für eine gewisse Verwirrung bei den Mäusen sorgen würde. Auf dem Foto (Abb. 13) ist eine Trennwand zu erkennen, welche die Buchstaben so trennt, dass die Maus nur gerade den Buchstaben sieht, vor dem sie steht. Der Versuchsaufbau ist symmetrisch aufgebaut.

Es wäre möglich den Mäusen alle 4 Buchstaben für die Kammern vorzugeben, aber ich hatte bedenken, dass die einen Mäuse nur ein C

lernen und die anderen nur ein A. Ich überlegte mir, wenn das Futter schon einmal gefunden wurde, besteht kein Anreiz mehr, noch weiter zu suchen. Deshalb bekommen sie während der Trainingsphase zweimal täglich hinter A und zweimal den Buchstaben C, jeweils natürlich in Kombination mit O oder X.

Im ungeteilten Abschnitt ist der Startpunkt, wohin die Mäuse einzeln gelegt werden. Von dort müssen sie den Weg zum Futter finden, indem sie den Buchstaben erkennen, hinter dem es sich befindet.



Abb. 13: Foto des Versuchsaufbaus

3.2.3. Störfaktoren und deren Behebung

Aus Beobachtungen ist bekannt, dass Mäuse unterschiedliche Strategien verfolgen, um Futter zu finden: Sie können den Weg markieren mit Gerüchen, sie lernen den Weg und nicht die Buchstaben, unterschiedliche Grössen der Vorkammern mit der Trennwand, Dicke der Buchstaben etc.

Um zu verhindern, dass sie den Weg in Form von Gerüchen markieren, reinige ich den Aufbau nach jedem Trainingslauf, sowie später in der Testphase mit 96% Alkohol, damit sie die Markierungen nicht mehr riechen. Die Mäuse können den Weg lernen und deshalb vertausche ich die Buchstaben jeden Morgen nach dem Zufallsprinzip mit einem Würfel, sowohl in der Trainingsphase als auch in der Testphase. Da die Grösse der Vorkammern nur minimal verschieden sind, könnte es sein, dass eine Maus tendenziell eher den grösseren Eingang benutzt, als den kleineren. Die Abstände der Trennwände sind nur minimal verschieden, weshalb sie nicht berücksichtigt werden.

3.2.4. Testen der Mäuse

Um einen direkten Vergleich vom Computerprogramm zum Versuch mit den Mäusen herzustellen, musste ich zuerst einen "Fehler" im Programm finden. Das heisst, ich musste eine Kombination finden, bei dem das Programm beim Wiedererkennen einen Fehler macht. Wenn das Programm A und C gelernt hat und der Umriss vom A gegeben wird, so kann der Buchstabe nicht mehr erkannt werden. Es gibt einen Fehler aus, eine Mischung zwischen den beiden Buchstaben (vgl. Abb. 14, weiter unten).

Im praktischen Versuch müssen die Mäuse A und C lernen. Die Trainingsphase verläuft folgendermassen: Während den ersten fünf Tagen des Ex-

perimentes benutze ich nur die Kombinationen A/X oder C/O, damit sie lernen, dass es immer in der linken Kammer Futter gibt. Nach diesen fünf Tagen wird, wie oben beschrieben, die zufällige Anordnung ausgelöst, um ihnen jetzt eine Verbindung vom Futter zu A und C zu geben.

Nachdem sie die Buchstaben gelernt haben, teste ich, ob die Mäuse die Zuordnung wirklich gelernt haben, indem ich eine Nullkontrolle durchführe, d.h. es werden fünf Durchläufe gemacht, in denen kein Futter in den Kammern liegt. Wenn sie in vier der fünf Durchläufen das Muster wiedererkennen, haben sie es gelernt. Es folgt die Testphase, jedoch nur für die beiden besten Mäuse. Ich werde 50 Testläufe durchgeführt, mit immer demselben Versuchsaufbau. Ich werde den Mäusen entweder die Konturen von C oder diejenigen von A und die Figur 4 aus Abbildung 14 zur Wiedererkennung vorgeben. Welcher Buchstabe getestet wird, stelle ich mit einer Zufallszahl zwischen 0 (=A) und 1 (=C) fest.

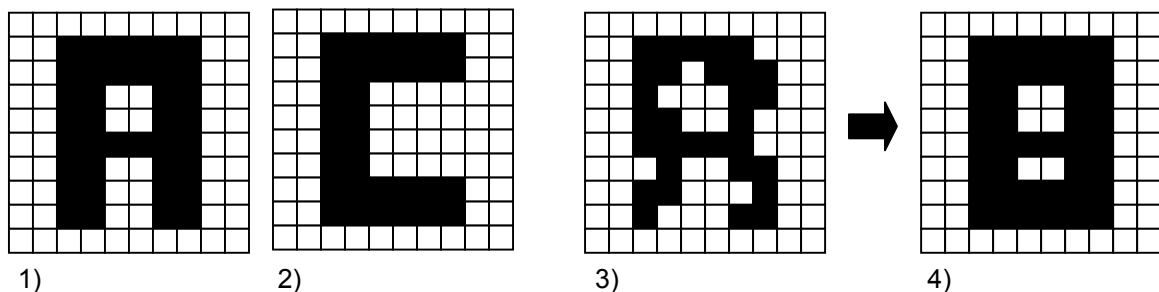


Abb.14: Die Buchstaben A und C werden gelernt (Figur 1 und 2). A wird verändert (Figur 3) und das Programm versucht, ihn wiederzuerkennen. Das funktioniert nicht und ergibt die Mischfigur 4.

3.2.5. Testen des Programms

Auch das Programm hat die Buchstaben A und C zu lernen. Nach der Lernphase, werde ich ihm die Konturen von A oder C zum wiedererkennen vorgeben (vgl. Abb. 15). Welcher Buchstabe schliesslich getestet wird, bestimme ich mit Hilfe einer Zufallszahl zwischen 0 und 1. Erhalte ich eine 0, werden die Konturen von A genommen, ansonsten diejenigen von C. Die Wiedererkennungsphase wiederhole ich 50-mal und zähle, wie oft der Buchstabe erkannt wurde.

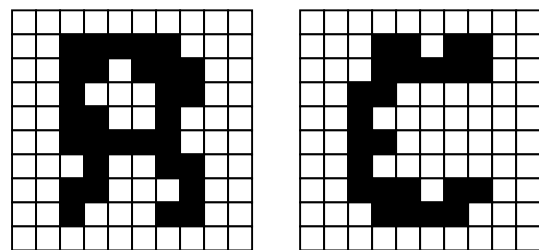


Abb. 15: Konturen von A und C

3.2.6. Protokoll

Im Versuch mit den Mäusen habe ich die Trainingsresultate, die Nullkontrolle und die Testresultate protokolliert. Während der Trainingsphase wurden die folgenden Kriterien und Fragen protokolliert: Datum, Uhrzeit, Futter gefunden nach x [Zeiteinheit] (Futter), Buchstabenkombination (BS), alle Mäuse gleichzeitig oder einzeln im Versuchsaufbau? (g/e), auffallendes Verhalten.

In der Nullkontrolle und während der Testphase protokolliere ich nur: Datum, Uhrzeit, Buchstabenkombination (BS), Ziel erreicht (J/N), Mausnummer (MN) und Laufnummer (LN). Der Zeitfaktor ist unwichtig, da die Mäuse alle sehr schnell sind. Das auffallende Verhalten, war nur in der Trainingsphase von Bedeutung, zur optimalen Zeitpunktbestimmung des Beginns der Testphase. Neu dazu kommt die Mausnummer, damit ich nach der Nullkontrolle die beiden besten Mäuse für die Tests selektionieren kann. In den Klammern sind jeweils die Abkürzungen, welche in den Tabellen stehen angegeben.

4. Resultate

4.1. Praktischer Versuch

4.1.1. Trainingsresultate

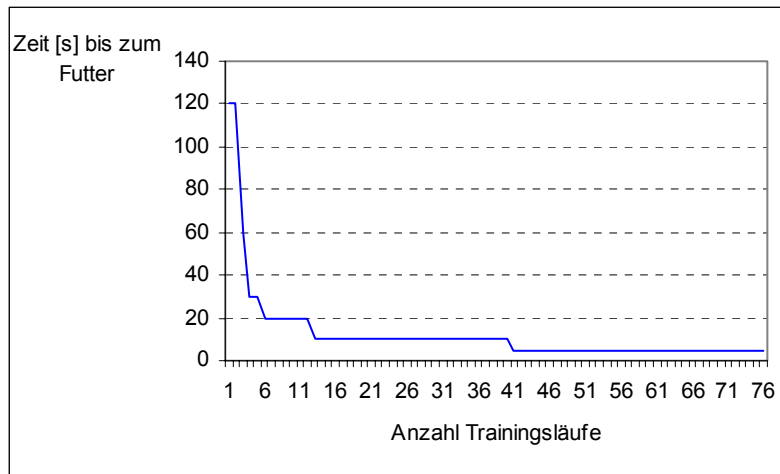


Abb. 16: Zeitverlauf der Trainingsphase

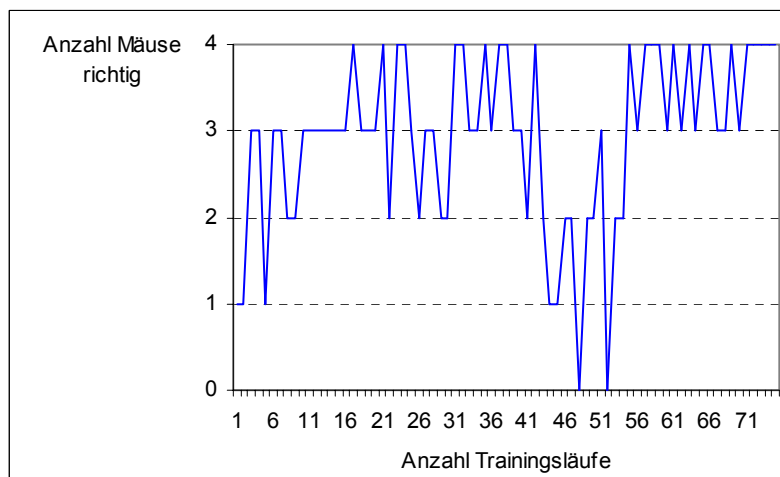


Abb. 17: Erfolgskurve

Am Anfang der Trainingsphase haben die Mäuse relativ lange gebraucht, um das Futter zu finden, wobei ihre Erfolgsquote noch nicht hoch lag. Die Zeit senkte sich sehr schnell und die Quote wurde immer höher, bis teilweise 100%. Bei dem 45. Testlauf hatten die Mäuse einen Absturz: Sie erkannten das Muster nicht mehr.

4.1.2. Nullkontrolle

Aus dieser Grafik (Abb. 18) ist ersichtlich, dass die Mäuse mit den Nummern 1, 2 und 3 das Muster je vier Mal erkannten und diejenige mit der Nummer 4 zwei Mal.

4.2. Programm

4.2.1. Testläufe

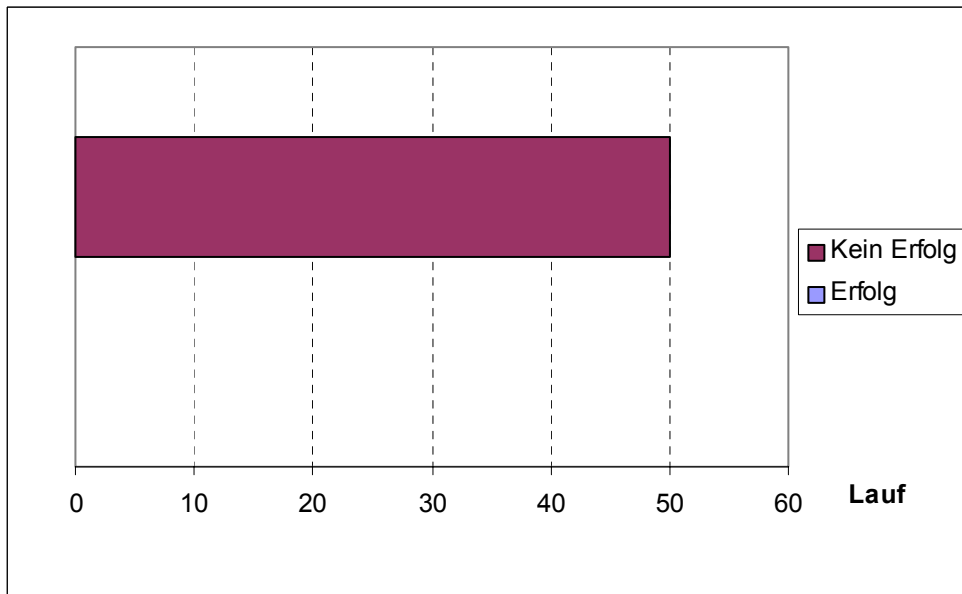


Abb. 20: Testläufe Programm

In der Grafik ist sehr gut zu erkennen, dass das Muster zu 0% erkannt bzw zu 100% nicht erkannt wurde.

4.2.2. Mass Resultate

16 Pixel

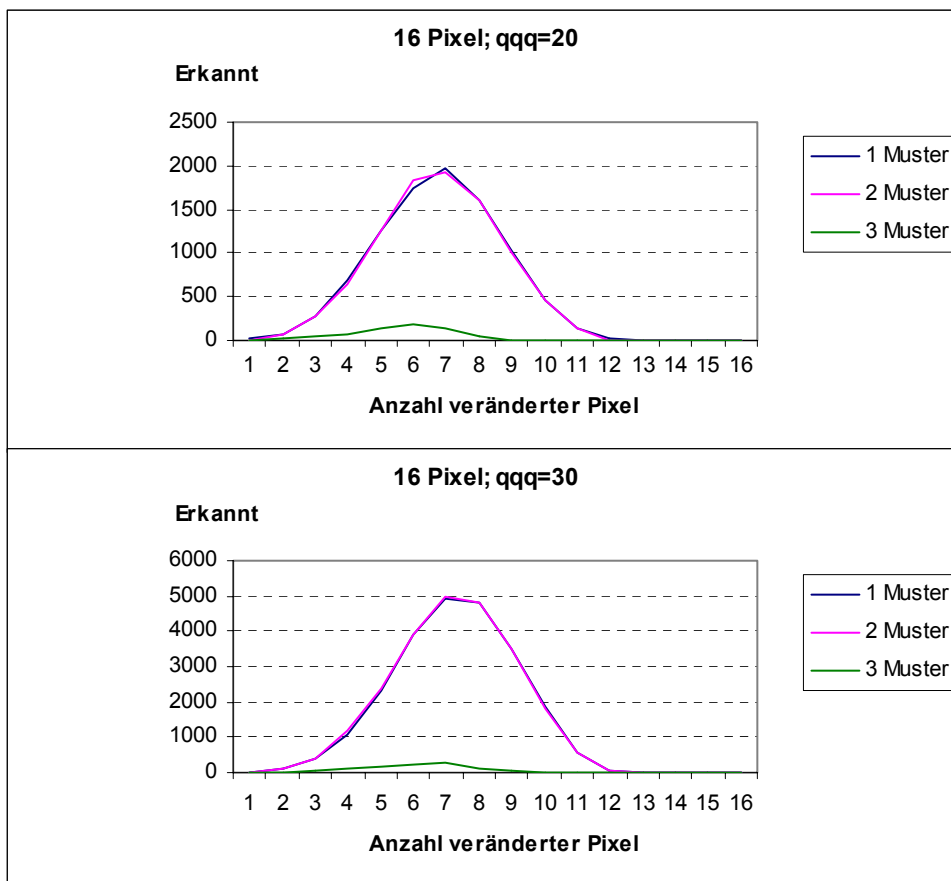


Abb. 21: 16 Pixel; qq=20

Abb. 22: 16 Pixel; qq=30

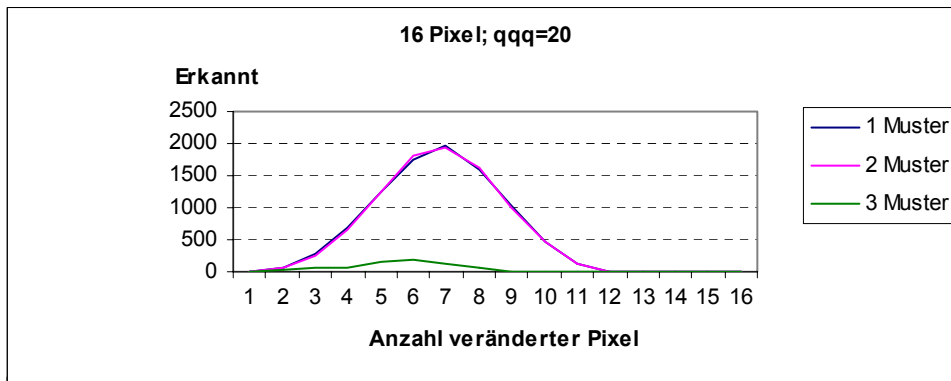


Abb. 23: 16 Pixel; qq=40

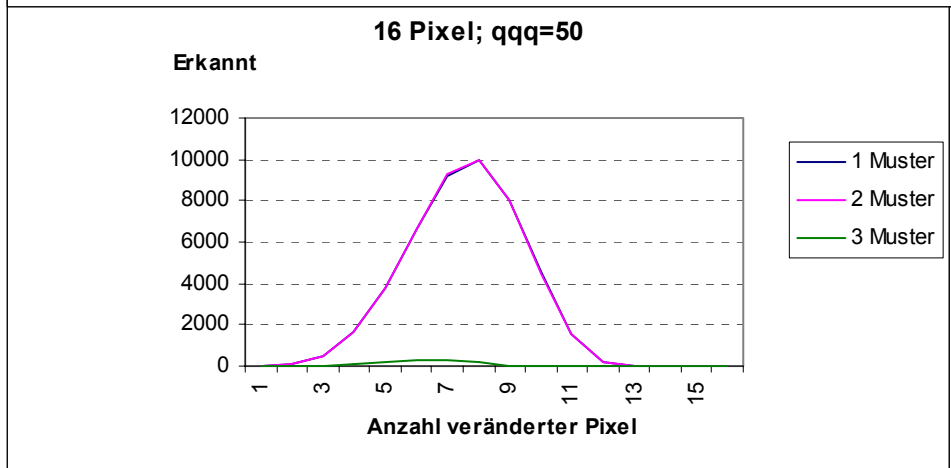


Abb. 24: 16 Pixel; qq=50

Die vier obenstehenden Abbildungen zeigen, wie viele Muster bei einer bestimmten Anzahl veränderter Pixel erkannt wurden. Aus der Grafik kann man nun also ablesen, wo sich die Abstandsgrosse befindet. Bei 16 Pixel beträgt sie für 1 und 2 gelernte Muster 13 und für 3 Muster, die gelernt wurden, 9. Das Resultat ist immer das gleiche, egal welcher Wert für qq gewählt wurde.

100 Pixel

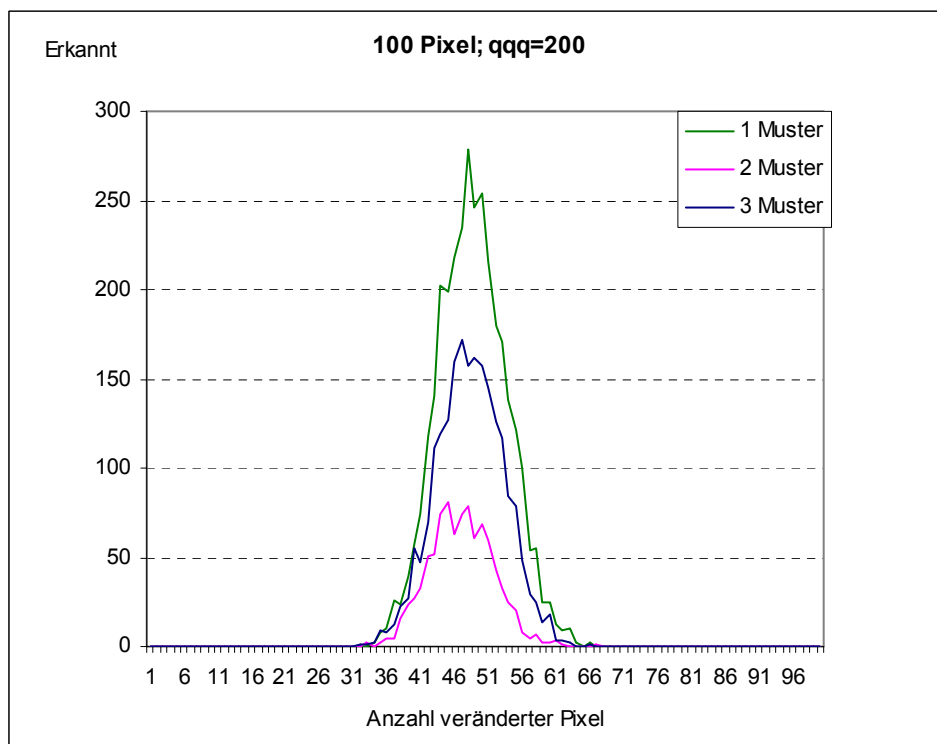


Abb. 25: 100 Pixel; qq=200

Aus Abb. 25 ist zu entnehmen, dass ein Muster nur im Bereich von etwa 32 bis ungefähr 68 Pixelveränderungen erkannt wird, vor- und nachher wird das Muster nicht mehr erkannt. Die Abstandgrösse beträgt bei einem 67, bei zwei 68 und bei drei gelernten Mustern 67 Pixelveränderungen.

5. Diskussion/Schlussfolgerungen

5.1. Relevanz der Resultate

5.1.1. Mustererkennungsprogramm

Die Resultate des Programms sagen nicht mehr aus, als dass es sehr konstant ist. Das vorgegebene Muster wird von einem Menschen sehr schnell erkannt und zwar immer richtig. Das Programm hat jedoch jedes Mal die Fehlerfigur als die richtige erkannt, obwohl dies bekanntlich unkorrekt ist.

5.1.2. Mäuse

Trotz der vielen Vorsichtsmassnahmen ist es möglich, dass ich immer noch einen Aspekt übersehen habe, der den Mäusen die Futtersuche vereinfacht hat.

Die Resultate, welche die Mäuse geliefert haben, sind aussagekräftiger als diejenigen des Programms. Die Mäuse haben während des Testlaufes sicherlich dazugelernt, dennoch sind ihre Resultate wesentlich realistischer. Auch sind sie nicht zu 100% konstant, so wie das Programm, ein weiterer positiver Beweis.

Die Testphase wurde nur mit zwei Mäusen durchgeführt, d.h. es dürfen keine Rückschlüsse auf die gesamte Tierart gemacht werden. Es könnte sich auch um zwei Ausnahmemaüse gehandelt haben.

5.2. Analyse der Resultate

5.2.1. Mustererkennungsprogramm

Die Resultate des Programms sind in 50 Durchläufen immer gleich ausgefallen, da es nach einem ganz bestimmten Algorithmus funktioniert und somit immer die fast (Zufallszahl in Erkennungsphase) gleichen Berechnungen anstellt. Deshalb fiel es leicht das Resultat vorherzusagen. Der Input, d.h. das Muster das vorgegeben wurde zur Wiedererkennung, blieb immer gleich. Als Negativpunkt kommt hinzu, dass das Feld in der Grösse von 100 Pixel viel zu klein für genaue Angaben ist. Eine Maus hat sicherlich mehr als 100 Neuronen und kann dadurch logischerweise zwei Muster unterscheiden. Das Programm würde besser funktionieren, wenn es 10'000 Pixel (=100*100 Pixel) wären, da dann 9'999 und nicht nur 99 etwas über den Zustand von dem einen Neuron aussagen. Diese Feldgrösse ist mit dem Programm Top Pascal jedoch nicht zu erreichen, da der benötigte Speicher für die Synapsengewichte zu gross wird.

Diese Faktoren zusammen bewirken, dass das Programm immer die gleiche Ausgabe lieferte.

5.2.2. Mäuse

1. Trainingsresultate

Gegen Ende der Trainingsphase ist immer mehr zu beobachten, dass die Mäuse zielgerichtet auf die Buchstaben losgehen, hinter denen sich das Futter befindet. Sie standen teilweise vor dem Buchstaben und

haben sich umgesehen, als ob das der richtige Buchstabe ist. Wenn nicht gingen sie zum anderen.

Als ich sie das erste Mal in den Versuchsaufbau gesetzt habe, rannten sie neugierig umher und erkundeten die neue Umgebung. Nach einiger Zeit waren sie sich daran gewöhnt und wussten, dass sie, um Futter zu bekommen, hinter einen der Buchstaben gehen müssen. Sie wussten auch, dass ich sie nur für die Fütterung in den Versuchsaufbau hineingebe, ansonsten nie. Als die Buchstabenkombination von O/C auf A/X änderte, gingen 75% der Mäuse wie gewohnt nach rechts. Sie hatten sich also den Weg gemerkt, bzw die Richtung, die sie gehen mussten, um das Futter zu erreichen (=positives Erlebnis). Diejenige Maus, die richtig war, ist vermutlich zufällig auf die andere Seite gelaufen. Später bemerkten sie, dass sich immer hinter den gleichen Buchstaben Futter befindet, alsoprägten sie sich schliesslich die Buchstabenformen ein und waren schliesslich am 24.11.02 soweit, dass sie vielfach die Buchstaben wiedererkannten. Als das Resultat einige Tage konstant blieb, wählte ich diesen Moment als den optimalen Zeitpunkt für die Nullkontrolle.

2. Nullkontrolle

Die Nullkontrolle hat ergeben, dass die Mäuse mit den Nummern 1, 2 und 3 die besten der insgesamt 4 Mäusen sind, da das Muster in jeweils 4 von 5 Versuchen richtig erkannt wurde. Da ich jedoch den Versuch nur mit zwei Mäusen machen möchte, habe ich mich entschlossen die Mäuse mit den Nummern 1 und 3 für den Versuch zu nehmen.

3. Testresultate

Die Mäuse haben die Buchstaben tatsächlich gelernt und konnten sie wiedererkennen, da die Mäuse sie mit sehr hohen Erfolgsquoten wiedererkannten, mit 84% und 86%. Bei diesen Resultaten muss aber beachtet werden, dass die Mäuse, während der Testphase auch noch dazugelernt haben könnten.

5.3. Vergleich der Wiedererkennungsmethoden

5.3.1. Gegenüberstellung der Resultate

Werden die Resultate verglichen, wird genau ersichtlich, dass die Mäuse nicht nach dem Algorithmus von Hopfield ein Muster erkennen, da sie mehrheitlich das richtige Muster erkannten, ansonsten hätten sie ebenfalls wie das Programm eine Erfolgsquote um die 0% herum haben müssen. Die Mäuse machen folglich keine Mischfigur aus A und C, sondern sie können die beiden Buchstaben unterscheiden.

Nach welchem Algorithmus funktioniert dann die Mustererkennung bei den Mäusen? Funktionieren sie tatsächlich mit einem Algorithmus? Diese fragen bleiben offen und müssten daher genauer untersucht werden. Die Mäuse funktionieren sicherlich mit einem Algorithmus, welcher das jedoch ist, bleibt offen.

Es ist bekannt, wie ein Auge funktioniert und wie wir Farbtöne wahrnehmen können, sogar auch wie der Nervenimpuls im Auge entsteht und zum Gehirn geleitet wird. Es ist jedoch nicht bekannt, wie das Gehirn ein Muster wiedererkennt. Wie setzt das Gehirn ein Muster aus Teilinformationen (Nervenimpulsen) zu einem Bild zusammen?

5.3.2. Was haben die Mäuse gelernt?

Haben die Mäuse wirklich das gelernt, was ich wollte, dass sie lernen? Ich wollte, dass sie A und C lernen. Die Mäuse haben diese mit einem positiven Ereignis verbunden, mit dem Futter. Ebenso haben sie O und X ein negatives Ereignis zugeordnet, da sie nichts fanden. Haben sie somit alle Buchstabenformen gelernt oder haben sie nur diejenigen mit dem positiven Ereignis gelernt? Meiner Meinung nach haben die Mäuse alle vier Formen gelernt. Sie wissen ja, dass sie O und X meiden sollen, also haben sie auch diese eingeprägt.

Anhand des Versuches kann ich nichts genaueres dazu sagen, da die Arbeit auch ein paar negative Einflüsse enthält. Ich habe nur vier Mäuse und kann nicht alle Einflussfaktoren ausschliessen, da mein Zimmer kein Labor ist, das zu 100% genau arbeiten kann und fast unbegrenzte Möglichkeiten aufweist. Es sind also genauere Versuche notwendig, um diesen Aspekt abzuklären.

5.4. Programmanalyse

5.4.1. Fehler im Programm

A und C befinden sich an genau der gleichen Position im Raster (vgl. Abb. 14), so dass, wenn beide Muster gelernt werden, sich die Gewichtsmatrizen aufsummieren. Es entsteht also eine neue Gesamtgewichtsmatrix, die einer der folgenden Werte als Gewicht enthält: 0, -2 oder 2. Ein Muster wird wiedererkannt, indem der Aktivitätszustand eines mit einer Zufallszahl ermittelten Neurons berechnet wird. Es ist möglich, dass anhand der veränderten Gewichte ein völlig anderer Zustand errechnet wird, als es tatsächlich sein sollte. Das Programm findet die Lösung natürlich mathematisch korrekt, d.h. das Programm erkennt nicht, ob es sich genau um das gleiche Muster handelt wie das gelernte, sondern für das Programm scheint alles richtig abzulaufen. Die Pixel, die schwarz gefärbt werden, obwohl sie bei einem A weiss wären, sind solche mathematisch korrekten Fehler, wo das Programm ein anderes Muster wiedererkennt.

Wenn nur wenige Pixel falsch gesetzt werden, ist es möglich, dass das Muster zwar wiedererkannt wird, aber nicht vollständig. Dies bedarf aber genaueren Untersuchungen, wie sie in Punkt 5.4.3. vorgenommen wurden. Dort wurde untersucht, wie viele Pixel verändert werden dürfen, damit ein Muster immer noch erkannt wird.

5.4.2. Ist es möglich, ein besseres Modell zu konzipieren?

Unser Gehirn ist sehr komplex gebaut und deshalb sehr schwierig nachzuahmen. Erschwerend kommt hinzu, dass die Entwicklung in der Neurobiologie noch nicht weit fortgeschritten ist. Selbst die Forscher sind

sich uneinig, wie weit man in der Erforschung des Gehirns überhaupt steht. Die einen sagen, man habe die Prinzipien des Gehirns erkannt und die Details würden in den nächsten 20 Jahren folgen. Die anderen sind sich noch nicht einmal sicher, ob die Prinzipien erkannt wurden, da man noch nicht genau weiss, wie Neuronen kommunizieren. Kommunizieren sie wirklich mittels elektrischen Impulsen? Wenn ja, wie übermitteln sie Informationen? Ist es die mittlere Entladungsrate oder das zeitliche Muster einer Salve von Aktionspotentialen? Diese Fragen stehen bis dato noch immer offen.

In der Neurobiologie versucht man möglichst viel über die Kommunikation der Neuronen herauszufinden. Je mehr darüber bekannt wird, desto genauere Modelle können erstellt werden, die immer weniger fehlerhaft sind. Das Modell von Hopfield ist zwar relativ alt (1982), da man jedoch in den letzten Jahren nicht viel weiter gekommen ist in der Neurobiologie, ist es immer noch aktuell.

Das Programm ist sehr konstant, d.h. es lernt in seinem weiteren Verlauf nicht, obwohl es zu Beginn das Muster erkennt. Das ist nicht realistisch, da das Gehirn auch in der Testphase noch dazulernt.

Die Funktionsweise des Neurons, bildet die Basis vom Modell von Hopfield. Auffallend ist aber, dass überall dort, wo das Gehirn sehr gut ist, das Programm sehr schlecht ist. Umgekehrt gilt das auch: Überall dort, wo der Computer sehr schnell ist, ist das Gehirn sehr langsam. Beispielsweise weiss das Gehirn eine hohe Fehlertoleranz auf, während die Mustererkennung sehr gut ist. Aber auch im Verallgemeinern von Beispielen ist das Gehirn sehr gut. Dagegen arbeitet der Computer sehr gut in numerisch exakten Berechnungen oder im fehlerlosen Datenspeichern.

Das Modell von Hopfield funktioniert in den meisten Situationen, aber nicht in allen. Ob es jemals möglich sein wird, ein Modell zu entwickeln, das perfekt Muster erkennen kann, bleibt fraglich.

5.4.3. Gibt es ein Mass in Pixel, ab wann das Programm ein Muster wiedererkennen kann?

Der Versuch mit 16 Pixel, bei dem alle möglichen Veränderungen durchgespielt wurden, ergab tatsächlich eine Grenze in Anzahl Pixel, die verändert werden dürfen, damit ein Muster wiedererkannt wird. Wenn ein oder zwei Muster gelernt wurden, erkannte das Programm ab einer Veränderung von 13 Pixel das Muster nicht mehr. Sobald jedoch drei Muster gelernt waren, wurde schon eine Veränderung von 10 Pixel nicht mehr erkannt. Je mehr Muster gelernt werden, desto kleiner wird folglich die Abstandsgrösse. Das qqq, das ich ebenfalls variiert habe, ist die Anzahl Schlaufendurchläufe durch die Wiedererkennungsphase. Es wird solange wiederholt, bis ein bestimmter Wert von qqq erreicht ist oder bis das Muster vollständig wiedererkannt wurde.

Die Resultate für den Versuch mit 100 Pixel sind sehr verwirrend auf den ersten Blick. Werden die Anzahl erkannter Figuren verglichen, so stellt man fest, dass bei einem gelerntem Muster am meisten erkannt wurden, am zweit meisten wurde jedoch bei drei gelernten Mustern die Figuren erkannt.

Normalerweise sollte die Zahl erkannter Figuren abnehmen mit zunehmender Anzahl gelernter Muster.

In einem Vergleich von der Verteilungen der erkannten Figuren fällt auf, dass sie sich nur innerhalb von etwa 36 Pixelveränderungen von ca. 32 bis etwa 68 befinden. Es wurden keine Muster erkannt, bei denen nur 1-31 Pixel verändert wurden. Die Pixel, die zu verändern sind, werden mit einer Zufallszahl im Programm bestimmt. Es wird die Zufallszahl 0 oder 1 ermittelt und dies 100 Mal, wobei eine 1 eine Änderung bedeutet. Die Wahrscheinlichkeit, dass das Programm dabei 99 Nullen und nur eine Eins herstellt, ist sehr klein. Auch das Gegenereignis, 99 Einsen und 1 Null, hat eine sehr kleine Wahrscheinlichkeit. Folglich kann die Grenze nicht genau bestimmt werden; aus den Resultaten ist eine Abstandsgrösse von 67 Pixelveränderungen ersichtlich. Da jedoch nicht ausgeschlossen werden kann, dass bei dem Versuch ein Muster hergestellt wurde, das mehr als 70 Pixelveränderungen zur Folge hatte, kann die Grenze nicht genau bestimmt werden. Der Algorithmus sollte jedoch korrekt sein, da ich ihn am 16 Pixel Programm getestet und dabei das gleiche Resultate erhalten habe, wie beim exakten Durchrechnen aller Zahlen. Das Programm hat eine Grenze, was auch logisch bzw realistisch ist, da der Mensch auch Grenzen hat. Das Gehirn erkennt schliesslich bei 13 von 16 möglichen Pixelveränderungen das Muster auch nicht mehr, es ist einfach zu ungenau vorgegeben.

5.5. Lernt das Modell von Hopfield?

Eines der Kernstücke des Modells von Hopfield bildet die Lernphase, in welcher die Synapsengewichte zwischen den einzelnen Neuronen festgelegt werden. Das Programm lernt nur an dieser Stelle und danach nicht mehr. Ein "richtiges" neuronales Netz macht jedoch nach jeder Handlung einen Lernschritt, jedes Mal, wenn ein Muster gesehen und nach einer Veränderung nicht mehr erkannt wird, wird die Lösung wieder gelernt. Als Beispiel für diesen Vorgang könnte man chinesische Zeichen geben, sie werden nicht gleich im ersten Anlauf gelernt und wiedererkannt. Genau dieser Vorgang geschieht beim Modell von Hopfield nicht. Das Muster kann nicht erkannt werden und die Lösung wird nicht erneut gelernt. Warum wird es dann trotzdem als Modell für ein neuronales Netz betrachtet, wenn es nur einmal und nicht fortlaufend lernt wie unser Gehirn?

Die Mustererkennung von Hopfield basiert auf einem vollständig rückgekoppelten, asynchronen neuronalen Netz. Es funktioniert wie ein neuronales Netz, weil es aufgrund von Information selbstständig entscheiden kann, ob ein Ereignis eintritt oder nicht. Das gefragte Neuron, vergleicht alle Beziehungen zu sich und entscheidet dann, ob es seinen Aktivierungszustand zu ändern hat. Eine Charaktereigenschaft von Neuronen ist, dass sie selbstständig entscheiden können, aufgrund von Informationen, die ständig erneuert werden.

5.6. Vergleich mit anderen Forschungsergebnissen

Ich kann meine Ergebnisse nicht mit anderen Testergebnissen vergleichen, da ich keine Studie über die Mustererkennung der Maus mit dem Modell von Hopfield gefunden haben.

5.7. Ausblick

In den letzten Jahren hat sich in der Mustererkennung mit Computern die Quantenmechanik zu einer neuartigen Informationstechnologie entwickelt, die eine Verbesserung der Informationsverarbeitung und der Informationsübertragung bringt. Es wäre möglich, dass sogenannte Quantencomputer zu neuen Speichern führen, die gewisse Parallelen mit unserem assoziativen Gedächtnis aufweisen. Das Gehirn würde dadurch besser angenähert werden und könnte, auch wenn nur unvollständige Informationen vorhanden sind, ein Muster wiedererkennen.

Die Quanten-Mustererkennung besteht aus zwei Phasen: Erkennung und Identifizierung. Der Mensch hat die Eigenschaft, je mehr Gesichter er kennt, desto öfters kommt es vor, dass er ein Gesicht erkennt, aber keinem Namen zuordnen kann. Genauso soll auch der assoziative Speicher funktionieren. Diesem Ziel, einen Assoziativspeicher mit Hilfe der Quantenmechanik nachzubilden, ist man noch sehr weit entfernt.

Es bleiben weiterhin diverse Fragen offen, so z.B. wäre es interessant zu untersuchen, welche Pixel verändert werden dürfen und welche nicht, damit ein Muster wiedererkannt werden kann.

6. Zusammenfassung

In meiner Arbeit habe ich eine Versuchsreihe mit Labormäusen durchgeführt und ein Programm geschrieben, das Muster erkennen kann. Ich wollte überprüfen, ob die Mäuse nach dem Modell von Hopfield, nach dem gleichen Modell wie das Programm, ein Muster wiedererkennen, wenn sie nur die Konturen des Musters sehen.

Die Mäuse mussten zwei Buchstaben A und C lernen. Der Y förmige Versuchsaufbau hat zwei Kammern, die sich hinter den Buchstaben mit Futter befinden. Als die Mäuse das Futter in einer der Kammern gefunden hatten, verbanden sie die Buchstaben mit einem positiven Ereignis, da sie ein Erfolgserlebnis hatten. Mit dem Futter habe ich ihnen zusätzlich einen Lernreiz gegeben. Insgesamt gibt es vier Buchstaben: A, C, X und O. Die Buchstaben X und O kamen jeweils in Kombination mit einem A oder einem C, es gab nur eine Bedingung: A und C oder X und O durften nicht zur gleichen Zeit zur Auswahl gegeben werden.

Das Modell von Hopfield hat einen Fehler: Wenn die beiden Buchstaben A und C gelernt wurden, wird eine Mischfigur als die richtige wiedererkannte Figur erkannt. Das machte ich mir zunutze indem die Mäuse in der Testphase die Konturen eines A's oder diejenigen eines C's und die Fehlerfigur des Programms erhalten. Wenn sie nun die Fehlerfigur als den richtigen Buchstaben wiedererkennen, ist bewiesen, dass sie nach dem Hopfield Modell ein Muster wiedererkennen.

Nach etlichen Neubeginnen des sehr langwierigen Experimentes, hat es endlich einmal geklappt; alle Mäuse konnten die Testphase lebend beenden und haben relativ gute Resultate geliefert. Die beiden Top Mäuse erkannten das Muster mit den Erfolgsquoten von 84% und 86% wieder.

Die Ergebnisse des Versuches mit dem Programm waren eindeutig: Es erkannte jedes Mal die Fehlerfigur als die richtige Ergänzung zur vorgegebenen Kontur. Das Programm ist zwar konstant, aber sein Erfolgsquote betrug 0%.

Die Mäuse erkennen die Mischfigur, die vom Programm erkannt wird, nicht an als Buchstabe mit Futter. Sie machen also keine Mischfigur aus A und C, sondern sie können die beiden Buchstaben unterscheiden und wissen, dass es dahinter Futter gibt. Folglich kann ich sagen, dass die Mäuse nicht nach dem Modell von Hopfield ein Muster wiedererkennen.

Ich habe zusätzlich versucht herauszufinden, ob es eine Abstandsgrösse gibt, bis zu der ein Muster vom Programm erkannt wird und dann nicht mehr. Ich habe dabei einen Versuch gemacht, mit zwei verschiedenen Pixelanzahlen; das erste Mal mit 16 Pixel, das zweite Mal mit 100 Pixel. Ist beim 16-Pixel-Versuch nur ein oder zwei Muster gelernt so beträgt sie 13 Pixel. Sind jedoch drei Muster gelernt, wird das Muster nach 10 Pixeländerungen nicht mehr erkannt.

Bei 100 Pixel ist die Grenze nicht so klar ersichtlich, wie bei 16 Pixel. Es wurde kein einziges Muster erzeugt, das eine Pixelveränderung zwischen 1 und 31 zur Folge hatte, so konnte auch keines in diesem Spektrum erkannt werden. Die Ursache dessen, liegt in der Bildung von Binärzahlen mittels Zufallsgenerator. Von 32 bis 68 Pixelveränderungen wurden diverse Muster erzeugt und auch wiedererkannt, wobei die Grenze bei 67 Pixel zu sein scheint. Es kann aber nicht gesagt werden, ob diese Grenze stimmt, da nicht bekannt ist, ob Veränderungen im Bereich von 69-100 Pixelveränderungen vorgenommen wurden.

7. Literaturverzeichnis

Winston, Patrick Henry: Artificial intelligence. Addison-Wesley Verlag Reading, 1993.

Roth, Gerhard: Das Gehirn und seine Wirklichkeit. Frankfurt am Main: Suhrkamp Verlag, 1994

Lindenmair, Wolfgang: Arbeitsheft Informatik - Neuronale Netze. Stuttgart: Ernst Klett Schulbuchverlag GmbH, 1995

Reif, Gerald: 3. Einführung in die künstliche Intelligenz.

<http://www.iicm.edu/greif/node5.html> (15.3.2002)

Reif, Gerald: 8. Neuronale Netze. <http://www.iicm.edu/greif/node10.html> (15.3.02)

Trugenberger, Carlo. NZZ: Ein Speicher mit assoziativen Fähigkeiten.

<http://www.nzz.ch/2002/03/13/ft/page-article7Z6L6.html> (13.11.02)

Gundelfinger, Prof. Dr. Eckart D. IFN – Neurone Chemie, Chemische Synapsen.

http://www.ifn-magdeburg.de/departments/dep2/dep2_prj1_de.jsp (16.12.2002)

Berger, Thomas. Hirn – Von der Nervenzelle zum Gehirn.

<http://pylwww.unibe.ch/sec/tafelnhirn.pdf> (16.12.2002)

8. Tabellen – und Abbildungsverzeichnis

8.1. Abbildungsverzeichnis

- Abb. 1: Schematischer Aufbau eines Neurons
Nach Lindenmair, 1995, S. 11
- Abb. 2: Mathematisches Modell eines Neurons im Vergleich mit einem biologischen Neuron.
Nach Lindenmair, 1995, S.12
- Abb. 3: Mathematisches Modell eines Neurons mit drei Eingabewerten.
Nach Lindenmair, 1995, S. 14
- Abb. 4: Neuronales Netz mit der Struktur 2-2-1
Nach Lindenmair, 1995, S. 28
- Abb. 5: Rückgekoppeltes neuronales Netz
Nach Lindenmair, 1995, S. 38
- Abb. 6: Rückgekoppeltes neuronales Netz mit veränderter Anordnung.
Nach Lindenmair, 1995, S. 39
- Abb. 7: Randerkennungsneuron
Nach Lindenmair, 1995, S. 17
- Abb. 8: Zu dem zu lernenden Muster (1) werden die Aktivitätszustände bestimmt (2) und deren dazugehörigen Synapsengewichte (3).
Nach Lindenmair, 1995, S. 41
- Abb. 9: 1) Das veränderte Neuron besitzt neue Aktivitätszustände (2). Die Gewichtsmatrix bleibt unverändert (3).
Nach Lindenmair, 1995, S. 42
- Abb. 10: Gegenüberstellung des Buchstabens A und dessen Konturen.
- Abb. 11: Flussdiagramm, das den Algorithmus der Lernphase im Programm darstellt.
- Abb. 12: Flussdiagramm, das den Algorithmus des Wiedererkennens darstellt.
- Abb. 13: Foto des Versuchsaufbaus
- Abb. 14: Die Buchstaben A und C werden gelernt (Figur 1 und 2). A wird verändert (Figur 3) und das Programm versucht, ihn wiederzuerkennen. Das funktioniert nicht und ergibt die Mischfigur 4.
- Abb. 15: Konturen von A und C
- Abb. 16: Zeitverlauf der Trainingsphase

Abb. 17: Erfolgskurve

Abb. 18: Grafik der Nullkontrolle

Abb. 19: Grafik der Mausresultate

Abb. 20: Testläufe Programm

Abb. 21: 16 Pixel; qqq=20

Abb. 22: 16 Pixel; qqq=30

Abb. 23: 16 Pixel; qqq=40

Abb. 24: 16 Pixel; qqq=50

Abb. 25: 100 Pixel; qqq=200

8.2. Tabellenverzeichnis

Tab. 1: Trainingsresultate der Mäuse

Tab. 2: Nullkontrolle

Tab. 3: Testergebnisse der Mäuse

Tab. 4: Programmresultate

Tab. 5: Anzahl erkannter Muster in Abhängigkeit von der Anzahl veränderter Pixel.

Tab. 6: Anzahl erkannter Muster in Abhängigkeit von der Anzahl veränderter Pixel.

9. Glossar/Abkürzungsverzeichnis

Assoziativspeicher = Herkömmlicher Speicher speichert Daten an einer bestimmten Adresse. Beim Assoziativspeicher ist keine Adresse nötig um Daten zu speichern, sondern er selber ist die Adresse. Die Daten werden gekoppelt mit anderen Daten. Der Begriff Ferien wird assoziativ gekoppelt mit den Begriffen Sonne, Strand etc.

Quantencomputer = Computer mit einer exponentiellen Rechenleistung gegenüber klassischen Rechnerarchitekturen

10. Anhang: Quellcode des Programms

```

program mustererkennung;

const e=100; k=10;
  {e=Pixel Anzahl (=Neuronen Anzahl), k=Wurzel aus Anzahl}
  teli=1; tere=8; teob=5; teun=4.5;
  {Ränder der Textboxen, te=text, li=links, re=rechts, ob=oben,
   un=unten}
  quadx=-10; quady=5;                               {Quadratfeld Randkoord.}
  delay=200;
  {Zeitliche Verzögerung in der Wiedererkennung, ist aber
   ausgeschaltet}
  rund=0.5;
  {Zahl, mit der die Koord. gerundet werden}
  nten=200;
  {Nach nten Durchlauf wird Fehlertoleranz erhöht um abb}
  abb=1;
  lauf=2000;
  {wiedererkennung==> so lange wird die Schaufe durchlaufen}
  on=1;                                               {Neuron aktiv=weiss}
  off=-1;                                             {Neuron inaktiv=schwazr}

var pt:point;
  {Bestimmt den Punkt der Mauskoordinate für die Textboxen}
  schwelle:integer;                                {Schwelle}
  i,jj,j,b,n:integer;                              {Zählvariablen}
  o,suche:integer;
  {suche:koord. gefunden?,o:schon ein Muster gelernt?}
  wahr:integer;
  {Muster wurde zu wahr% erkannt, am Schluss bei der Wiedererkennung
   benötigt}
  ab,q,qq,qqq,su:integer;
  {werden bei der Wiedererkennung gebraucht, repeat-Variablen,
   Fehlertoleranz-Variablen}
  q1,q2,q3,q4,q5,q6,q7,q8,q9,q10:rect; {Variablen für die Textboxen}
  xx,yy,l,u:real;
  {xx,yy:Variablen für Koordinaten runden, l:Gewichtete Summe,
   u:repeat-Variable}
  zeit:longint;
  {für Tickcount nötig, ist aber ausgeschalten==>wird nicht benötigt}

  e1:array[1..e] of real;
  {Zustandsmatrix Muster (schwarz, weiss?) vor dem verändern}
  e2:array[1..e] of real;                          {nach dem verändern}

  arx:array[1..e] of real;                          {x-Koordinate}
  ary:array[1..e] of real;                          {y-Koordinate}
  arz:array[1..e] of integer;                      {schwarz-weiss; 1-0}

  w1:array[1..e,1..e] of longint;                  {Gewichtmatrix}

procedure zeichnen2;
begin
  uRGBWhite;
  upaintrect(quadx,quady-k,quadx+k,quady);
  {Die Rasterfläche wird gelöscht=weiss übermalt}
  uRGBBlack;
end;

```

```
procedure zeichnen;
var x,y,i:integer;
begin
  x:=quadx;
  y:=-quady;
  for i:=1 to k+1 do
    {Raster zeichnen, Koord. (-10/-5) bis (0/5)}
    begin
      umoveto(x,quady-k);
      ulineto(x,quady);
      x:=x+1;
    end;
  for i:=1 to k+1 do
    begin
      umoveto(quadx+k,y);
      ulineto(quadx,y);
      y:=y+1;
    end;
end;

procedure anfang;
begin
  output('Herzlich Willkommen bei der Mustererkennungs-software,
        fertiggestellt am 6.8.02 von Marco Hunziker.
        Wie funktioniert das Programm?
        3 Muster sind vorgegeben. Man kann die Muster jedoch auch
        selber kreieren mit "Muster selber entwerfen...');
  output('...und dann "fertig entworfen. Jetzt klickt man auf "Lernen",
        dann wird das eben gesehene Muster gelernt. Als nächstes
        verändert man das Muster so, dass es verwechselt wird mit
        "verändern" und "fertig verändert.");
  output('Jetzt klickt man auf "Muster wiederherstellen" und das Muster
        sollte wiedererkannt werden. Man kann das Programm nun
        verlassen, oder die schritte mit dem verändern und dem
        wiederherstellen wiederholen.');
```

```
  output('Wenn das Muster wiedererkannt wurde, bleibt das gelernte
        Muster, oder die gelernten Muster bestehen. Wenn man ein
        Muster nicht mehr will, muss das Programm verlassen und neu
        gestartet werden');
end;

procedure set;
var r:rect;
begin
  usetxachse(quadx-quady,-(quadx-quady)); {Fenstereinstellungen}
  usetyachse(-quadx,quadx);
  getWindowRect(r);
  setDrawingRect(r);
end;

begin
  anfang; {Text wird eingeführt}
  set; {Fenster Eckpunkte werden gesetzt}
  o:=0; {Noch kein Muster gelernt ==> o=0}
  for i:=1 to e do
    {z-Werte=0 setzen. Zählvariable mod 2 = 0 then weiss, sonst schwarz}
    begin
      {feststellen, ob Pixel schwarz oder weiss}
      arz[i]:=0;
    end;
  end;
```

```
end;

b:=quadx;
for i:=1 to e do
  {x-Koord. festlegen, Eckpunkte, Werte von -10 bis -1 werden erstellt}
  begin
    {100 mal, da 100 Eckpunkte, die eine x-Koord. haben}
    arx[i]:=b;
    if (b=quadx+(k-1)) then b:=quadx-1;
    b:=b+1;
  end;

b:=quady;
for i:=1 to e do
  {y-Koord. festlegen, Eckpunkte, Werte von 5 bis -5}
  begin
    {100 mal, da 100 Eckpunkte, die eine y-Koord. haben}
    ary[i]:=b;
    if i mod k = 0 then b:=b-1;
    {x/y Koord.von Links oben nach Rechts oben, zeilenunterbr...bis
    Rechtsunten. }
  end;

for i:=1 to e do
  {Aktivitätszustände=inaktiv=1 setzen}
  begin
    el[i]:=on;
  end;
schwelle:=0;
zeichnen;                               {Raster zeichnen}

uRealToRect(teli,teob,tere,teun,q1);
{Textbox für diverse Anklickmögl., mit koord. teli,teob,tere,teun}
TextBox(q1,0,true,'Buchstabe A');        {A}

uRealToRect(teli,teob-1,tere,teun-1,q2);
TextBox(q2,0,true,'Buchstabe B');        {B}

uRealToRect(teli,teob-2,tere,teun-2,q3);
TextBox(q3,0,true,'Buchstabe C');        {C}

uRealToRect(teli,teob-3,tere,teun-3,q9);
TextBox(q9,0,true,'Selber Muster entwerfen');
{selber Muster entwerfen}

uRealToRect(teli,teob-4,tere,teun-4,q10);
TextBox(q10,0,true,'fertig entworfen');
{fertig entworfen}

uRealToRect(teli,teob-5,tere,teun-5,q4);
TextBox(q4,0,true,'Lernen');             {Lernen}

uRealToRect(teli,teob-6,tere,teun-6,q5);
TextBox(q5,0,true,'verändern');          {verändern}

uRealToRect(teli,teob-7,tere,teun-7,q6);
TextBox(q6,0,true,'fertig verändert');     {fertig verändert}

uRealToRect(teli,teob-8,tere,teun-8,q7);
TextBox(q7,0,true,'Muster wiedererkennen');
```

```
{Muster wiedererkennen}
```

```
uRealToRect(teli,teob-9,tere,teun-9,q8);
```

```
TextBox(q8,0,true,'verlassen');
```

```
{Programm verlassen}
```

```
repeat
```

```
  repeat until button;
```

```
  repeat until not button;
```

```
  getmouse(pt);
```

```
  {Mauskoordinaten: ist pt in Rechteck q1,q2...q8?=>überprüfung bei  
  jedem abgeschlossenen Schritt ==> gehe zu schleife, wo pt in Rechteck  
  ist}
```

```
  if ptinrect(pt,q1) then {BUCHSTABE A}
```

```
    begin
```

```
      zeichnen2;
```

```
      {Die Rasterfläche wird gelöscht=weiss übermalt}
```

```
      zeichnen; {Raster zeichnen}
```

```
      upaintrect(-8,4,-6,-4); {optische Oberfläche}
```

```
      upaintrect(-4,4,-2,-4);
```

```
      upaintrect(-6,4,-4,2);
```

```
      upaintrect(-6,-0,-4,-1);
```

```
    for i:=13 to 18 do
```

```
      {Aktivitätszustände (inaktiv=-1)}
```

```
      begin
```

```
        e1[i]:=off;
```

```
        arz[i]:=1;
```

```
        {Die aktiven Neuronen auf arz=1 setzen, damit schwarz  
        gezeichnet}
```

```
      end;
```

```
    for i:=23 to 28 do
```

```
      begin
```

```
        e1[i]:=off;
```

```
        arz[i]:=1;
```

```
      end;
```

```
    for i:=53 to 58 do
```

```
      begin
```

```
        e1[i]:=off;
```

```
        arz[i]:=1;
```

```
      end;
```

```
    e1[33]:=off;arz[33]:=1;
```

```
    e1[34]:=off;arz[34]:=1;
```

```
    e1[37]:=off;arz[37]:=1;
```

```
    e1[38]:=off;arz[38]:=1;
```

```
    e1[43]:=off;arz[43]:=1;
```

```
    e1[44]:=off;arz[44]:=1;
```

```
    e1[47]:=off;arz[47]:=1;
```

```
    e1[48]:=off;arz[48]:=1;
```

```
    e1[63]:=off;arz[63]:=1;
```

```
    e1[64]:=off;arz[64]:=1;
```

```
    e1[67]:=off;arz[67]:=1;
```

```
    e1[68]:=off;arz[68]:=1;
```

```
    e1[73]:=off;arz[73]:=1;
```

```
    e1[74]:=off;arz[74]:=1;
```

```
    e1[77]:=off;arz[77]:=1;
```

```
    e1[78]:=off;arz[78]:=1;
```

```
    e1[83]:=off;arz[83]:=1;
```

```
    e1[84]:=off;arz[84]:=1;
    e1[87]:=off;arz[87]:=1;
    e1[88]:=off;arz[88]:=1;
end;

if ptinrect(pt,q2) then {BUCHSTABE B}
begin
    zeichnen2;
    {Die Rasterfläche wird gelöscht=weiss übermalt}
    zeichnen; {Raster zeichnen}

    upaintrect(-8,4,-6,-4); {optische Oberfläche}
    upaintrect(-6,4,-3,3);
    upaintrect(-6,-2,-3,-4);
    upaintrect(-4,3,-2,1);
    upaintrect(-4,0,-2,-3);
    upaintrect(-6,1,-3,0);

    for i:=13 to 17 do
        {Aktivitätszustände (inaktiv=-1)}
        begin
            e1[i]:=off;
            arz[i]:=1;
            {Die aktiven Neuronen auf arz=1 setzen, damit schwarz
            gezeichnet}
        end;
    for i:=43 to 47 do
        begin
            e1[i]:=off;
            arz[i]:=1;
        end;
    for i:=73 to 78 do
        begin
            e1[i]:=off;
            arz[i]:=1;
        end;
    for i:=83 to 87 do
        begin
            e1[i]:=off;
            arz[i]:=1;
        end;

    e1[23]:=off;arz[23]:=1;
    e1[24]:=off;arz[24]:=1;
    e1[27]:=off;arz[27]:=1;
    e1[28]:=off;arz[28]:=1;
    e1[33]:=off;arz[33]:=1;
    e1[34]:=off;arz[34]:=1;
    e1[37]:=off;arz[37]:=1;
    e1[38]:=off;arz[38]:=1;
    e1[53]:=off;arz[53]:=1;
    e1[54]:=off;arz[54]:=1;
    e1[57]:=off;arz[57]:=1;
    e1[58]:=off;arz[58]:=1;
    e1[63]:=off;arz[63]:=1;
    e1[64]:=off;arz[64]:=1;
    e1[67]:=off;arz[67]:=1;
    e1[68]:=off;arz[68]:=1;
end;

if ptinrect(pt,q3) then {BUCHSTABE C}
```

```
begin
  zeichnen2;
  {Die Rasterfläche wird gelöscht=weiss übermalt}
  zeichnen;                               {Raster zeichnen}

  upaintrect(-8,4,-6,-4);                  {optische Oberfläche}
  upaintrect(-6,4,-2,2);
  upaintrect(-6,-2,-2,-4);

  for i:=13 to 18 do
    {Aktivitätszustände (inaktiv=-1)}
    begin
      e1[i]:=off;
      arz[i]:=1;
      {Die aktiven Neuronen auf arz=1 setzen, damit schwarz
      gezeichnet wird}
    end;

  for i:=23 to 28 do
    begin
      e1[i]:=off;
      arz[i]:=1;
    end;

  for i:=73 to 78 do
    begin
      e1[i]:=off;
      arz[i]:=1;
    end;

  for i:=83 to 88 do
    begin
      e1[i]:=off;
      arz[i]:=1;
    end;

  e1[33]:=off;arz[33]:=1;
  e1[34]:=off;arz[34]:=1;
  e1[43]:=off;arz[43]:=1;
  e1[44]:=off;arz[44]:=1;
  e1[53]:=off;arz[53]:=1;
  e1[54]:=off;arz[54]:=1;
  e1[63]:=off;arz[63]:=1;
  e1[64]:=off;arz[64]:=1;
end;

if ptinrect(pt,q9) then                    {SELBER MUSTER ENTWERFEN}
begin
  for i:=1 to e do
    {alle z-Werte werden auf weiss=0 gesetzt}
    begin
      arz[i]:=0;
    end;
  zeichnen2;
  zeichnen;
  for i:=1 to e do
    {alle Aktivitätszustände auf 1=aktiv setzen}
    begin
      e1[i]:=on;
    end;
end;
```

```

repeat
  repeat
    repeat until button;
    repeat until not button;
    ugetmouse(xx,yy);           {Mauskoordinaten erhalten}
    u:=0;
    if (xx>quadx+k) or (yy<-quady) or (xx<quadx) or (yy>quady)
    then output('Nicht im Feld') else u:=1;   {Klick im Feld?}

    xx:=round(xx-rund);
    yy:=round(yy+rund);
    {x,y Werte werden gerundet in den linken oberen Ecken, darum
     -0.5,+0.5}
    until u=1;
    {Wiederhole, bis klick im Feld, dann u=1}
  for n := 1 to e do
    begin
      if (xx=arz[n]) and (yy=ary[n]) then suche:=1 else suche:=0;
      {dazugehörende Koord. werden gesucht}
      if suche=1 then
        begin
          arz[n]:=arz[n]+1;
          {z-Wert um 1 erhöhen, damit nächstes Mal die Farbe
           wechselt}
          if arz[n] mod 2=0 then begin uRGBWhite; e1[n]:=on; end
          else begin uRGBBlack; e1[n]:=off; end;
          {z-Wert mod 2=0 then weiss else black}
        end;
      end;

      upaintrect(xx,yy,(xx+1),(yy-1));
      {das Pixel wird gezeichnet in entspr. farbe}
      uRGBBlack;
      uframerect(xx,yy,xx+1,yy-1);
      {Rand des Pixels immer schwarz, falls Pixel=weiss}
      i:=i+1;
      repeat until button;
      getmouse(pt);
      if ptinrect(pt,q10) then output('Entwerfen abgeschlossen');
      {wiederholen, bis Klick auf 'fertig entw.'}
      until ptinrect(pt,q10);
    end;

  if ptinrect(pt,q4) then
  {LERNEN}
  begin
    i:=1;
    for i:=1 to e do
      begin
        for jj:=1 to e do
          begin
            if (e1[i]+e1[jj]=-2) or (e1[i]+e1[jj]=2) then
              w1[i,jj]:=w1[i,jj]+1
              {neuron i, beziehung zu neuron jj}
            else if e1[i]+e1[jj]=0 then w1[i,jj]:=w1[i,jj]-1;
            {aktivitätszustände gleich? dann +1 sonst -1. summe=-2 od.
             2 dann gleich}
            if i=jj then w1[i,jj]:=0;
          end;
        end;
      end;
    end;
  end;

```

```

output('Lernphase beendet');
o:=o+1;
end;

if ptinrect(pt,q5) then                                {VERÄNDERN}
begin
for i:=1 to e do
  {alle Aktivitätszustände auf 1=inaktiv setzen}
begin
e2[i]:=e1[i];
end;
repeat
repeat
repeat until button;
repeat until not button;
ugetmouse(xx,yy);      {Mauskoordinaten erhalten}
u:=0;
if (xx>quadx+k) or (yy<-quady) or (xx<quadx) or (yy>quady)
then output('Nicht im Feld') else u:=1;      {Klick im Feld?}

xx:=round(xx-rund);
yy:=round(yy+rund);
{x,y Werte werden gerundet in den linken oberen Ecken, darum -
0.5,+0.5}
uRGBWhite;
until u=1;      {Wiederhole, bis klick im Feld, dann u=1}

for n:= 1 to e do
begin
if (xx=arx[n]) and (yy=ary[n]) then suche:=1 else suche:=0;
{dazugehörige Koord. werden gesucht}
if suche=1 then
begin
arz[n]:=arz[n]+1;
{z-Wert um 1 erhöhen, damit nächstes Mal die Farbe
wechselt}
if arz[n] mod 2=0 then begin uRGBWhite; e2[n]:=on; end
else begin uRGBBlack; e2[n]:=off; end;
{z-Wert mod 2=0 then weiss else black}
end;
end;

upaintrect(xx,yy,(xx+1),(yy-1));
{das Pixel wird gezeichnet in entspr. farbe}
uRGBBlack;
uframerec(xx,yy,xx+1,yy-1);
{Rand des Pixels, immer schwarz, falls Pixel=weiss}
repeat until button;
getmouse(pt);
if ptinrect(pt,q6) then output('fertig verändert');
until ptinrect(pt,q6);
{wiederholen, bis Klick auf 'fertig verändert'}
end;

if ptinrect(pt,q7) then
{WIEDERERKENNEN}
begin
ab:=0;
qqq:=0;
qq:=0;
repeat

```

```

begin

j:=urandomln(e);      {Zufallsneuron}
l:=0;                 {Vorhergehendes gewichtetes Resultat = 0}

for i:=1 to e do
  begin
    l:=l+e2[i]*w1[j,i];
    {Gewichtete Summe wird gebildet}
  end;

if l<schwelle then e2[j]:=off;
{iüberprüfung ob <,>= Schwelle, bei < Schwelle Wert=-1=n.A.}
if l>=schwelle then e2[j]:=on;
{bei >=Schwelle Wert=1=neuer Aktivitätszustand}
if e2[j]=-1 then uRGBBlack else if e2[j]=1 then uRGBWhite;
upaintrect(arx[j],ary[j],arx[j]+1,ary[j]-1);
{Farbe bestimmen+zeichnen mit neuem Aktivitätszustand}

uRGBBlack;
zeichnen;
su:=0;
for i:=1 to e do
  {Fehlerrsumme bilden, jedesmal eine Neue==> su=0, jedesmal}
  begin
    if e1[i]=e2[i] then q:=0 else q:=1;
    su:=su+q;
    {Summiert bei jedem ungleichen Zustand 1 dazu}
  end;
if (su=ab) and (qqq<lauf) then qq:=1 else qq:=0;
{erhaltene Summe=Fehlertoleranz? Dann beenden qq=0}
if o=0 then begin output('noch kein Muster gelernt!');qq:=1;
su:=-1; end;
{o=0 ==> noch kein Muster gelernt;wird erhöht im Lernen}
qqq:=qqq+1;      {zählt Durchläufe der Schleife}

if qqq mod nten=0 then ab:=ab+abb;
{bei jedem nten. Durchlauf wird ab um 1
erhöhte=>Fehlertoleranz}
{zeit:=Tickcount;
repeat until Tickcount>zeit+delay;}
{Verzögerung der Schlaufe}
end;
until (qq=1) or (qqq=lauf);
{wiederholen, bis erhaltene Fehlerrsumme=Fehlertoleranz}
wahr:=100-ab;
{Berechnung der Erkennungsw'keit des "wiedererkannten" Musters}
if su>0 then output('Muster wurde zu '+stringof(wahr),' Prozent
wiedererkannt');
if su=0 then output('Muster wurde zu 100 Prozent wiedererkannt');
end;
until ptinrect(pt,q8); {bis Klick auf 'verlassen'}
end.

```

11. Anhang: Tabellen

11.1. Praktischer Versuch

11.1.1. Trainingsresultate

Datum	Uhrzeit	Futter	BS	g/e	auffallendes Verhalten
10.11.02	14:00	alle < 2 min	A/X	g	1 richtig, 3 falsch
	18:00	alle < 2 min	A/X	g	
	20:00	alle < 1 min	A/X	g	3 richtig, 1 falsch
	22:00	alle < 30 sec	A/X	g	3 richtig, 1 falsch
11.11.02	13:30	alle < 30 sec	C/O	g	1 richtig, 3 zuerst falsch
	20:00	alle < 20 sec	C/O	g	3 richtig, 1 falsch
	22:00	alle < 20 sec	C/O	g	
	23:00	alle < 20 sec	C/O	g	alle richtig
12.11.02	13:30	alle < 20 sec	A/X	g	2 richtig, 2 falsch
	20:00	alle < 20 sec	A/X	g	3 richtig, 1 falsch
	22:00	alle < 20 sec	A/X	g	
	23:00	alle < 20 sec	A/X	g	
13.11.02	8:00	alle < 10 sec	C/O	g	3 richtig, 1 falsch
	10:00	alle < 10 sec	C/O	g	
	12:00	alle < 10 sec	C/O	g	3 richtig, 1 falsch
	16:00	alle < 10 sec	C/O	g	
14.11.02	7:00	alle < 10 sec	A/X	g	alle richtig
	12:00	alle < 10 sec	C/O	g	3 richtig, 1 falsch
	17:00	alle < 10 sec	A/X	g	
	21:00	alle < 10 sec	C/O	g	
15.11.02	6:30	alle < 10 sec	A/X	g	alle richtig
	12:00	alle < 10 sec	C/O	g	2 sofort, 2 später
	19:00	alle < 10 sec	A/X	g	alle richtig
	21:00	alle < 10 sec	C/O	g	
16.11.02	13:00	alle < 10 sec	O/A	e	3 richtig, 1 falsch, 2 sind zuerst zum falschen hingelaufen, haben geschaut, sind zum anderen gelaufen
	19:00	alle < 10 sec	X/C	e	2 richtig, 2 falsch
	23:00	alle < 10 sec	O/A	e	3 richtig, 1 falsch
	1:00	alle < 10 sec	X/C	e	3 richtig 1 falsch
17.11.02	12:00	alle < 10 sec	C/O	e	2 richtig, 2 falsch
	15:00	alle < 10 sec	X/A	e	
	18:00	alle < 10 sec	C/O	e	alle richtig
	21:00	alle < 10 sec	X/A	e	
18.11.02	7:30	alle < 10 sec	C/O	e	3 richtig, 1 falsch
	12:00	alle < 10 sec	X/A	e	
	16:00	alle < 10 sec	C/O	e	alle richtig
	21:00	alle < 10 sec	X/A	e	3 richtig
19.11.02	13:00	alle < 10 sec	C/X	g	alle richtig
	15:00	alle < 10 sec	O/A	g	alle richtig
	17:00	alle < 10 sec	C/X	g	3 richtig
	19:00	alle < 10 sec	O/A	g	

Datum	Uhrzeit	Futter	BS	g/e	Auffälligs Verhalten
20.11.02	7:00	alle < 5 sec	O/A	g	3 richtig
	12:00	alle < 5 sec	C/X	g	2 richtig
	21:00	alle < 5 sec	O/A	g	alle richtig
	23:00	alle < 5 sec	C/X	g	2 richtig
21.11.02	14:00	alle < 5 sec	X/C	e	1 richtig, 3 falsch
	16:00	alle < 5 sec	A/O	e	
	18:00	alle < 5 sec	X/A	e	2 richtig
	20:00	alle < 5 sec	C/O	e	
22.11.02	16:00	alle < 5 sec	A/X	e	keine hat das Futter gefunden!
	18:00	alle < 5 sec	O/C	e	2 richtig
	23:00	alle < 5 sec	A/X	e	2 richtig
	0:00	alle < 5 sec	O/C	e	3 richtig
23.11.02	12:00	alle < 5 sec	X/A	e	keine richtig. Alle Mäuse nach links, anstatt nach rechts. Bisher nur Richtung gemerkt
	15:00	alle < 5 sec	O/C	e	2 richtig
	20:00	alle < 5 sec	X/A	e	2 richtig
	23:00	alle < 5 sec	O/C	e	alle richtig
24.11.02	12:00	alle < 5 sec	X/C	e	3 richtig
	18:00	alle < 5 sec	A/O	e	alle richtig
	21:00	alle < 5 sec	X/C	e	alle richtig
	23:00	alle < 5 sec	A/O	e	alle richtig
25.11.02	12:00	alle < 5 sec	X/C	e	3 richtig
	18:00	alle < 5 sec	A/O	e	alle richtig
	21:00	alle < 5 sec	X/C	e	3 richtig
	22:00	alle < 5 sec	A/O	e	alle richtig
26.11.02	12:00	alle < 5 sec	C/X	e	3 richtig
	18:00	alle < 5 sec	A/X	e	alle richtig
	21:00	alle < 5 sec	C/X	e	alle richtig
	22:00	alle < 5 sec	A/O	e	3 richtig
27.11.02	7:00	alle < 5 sec	X/C	e	3 richtig
	13:00	alle < 5 sec	O/A	e	alle richtig
	18:00	alle < 5 sec	X/C	e	3 richtig
	22:00	alle < 5 sec	O/A	e	alle richtig
28.11.02	7:00	alle < 5 sec	X/C	e	alle richtig
	13:00	alle < 5 sec	O/A	e	alle richtig
	18:00	alle < 5 sec	X/C	e	alle richtig
	22:00	alle < 5 sec	O/A	e	alle richtig

Tab. 1: Trainingsresultate der Mäuse

11.1.2. Nullkontrolle

LN	Datum	Uhrzeit	BS	J/N	MN	LN	Datum	Uhrzeit	BS	J/N	MN
1	29.11.02	15:30	X/C	j	1,2,4	4	30.11.02	13:00	O/A	j	1,3,4
				n	3					n	2
2	29.11.02	11:45	O/A	j	1,2,3	5	1.12.02	13:00	A/X	j	2,3
				n	4					n	1,4
3	30.11.02	10:00	C/X	j	1,2,3						
				n	4						

Tab.2: Nullkontrolle

11.1.3. Testergebnisse

LN	Datum	Uhrzeit	BS	J/N	MN	LN	Datum	Uhrzeit	BS	J/N	MN
1	4.12.02	8:00	A/F	j	M1	21	10.12.02	9:00	A/F	j	M1,M3
				n	M3					n	
2	5.12.02	17:00	F/C	j	M1,M3	22	10.12.02	9:30	C/F	j	M1
				n						n	M3
3	5.12.02	19:00	F/C	j	M1,M3	23	10.12.02	10:00	F/A	j	M1,M3
				n						n	
4	5.12.02	23:00	C/F	j	M3	24	10.12.02	10:30	F/A	j	M3
				n	M1					n	M1
5	6.12.02	18:00	A/F	j	M1,M3	25	10.12.02	18:45	F/C	j	M1,M3
				n						n	
6	6.12.02	18:45	F/A	j	M1,M3	26	10.12.02	19:00	F/A	j	M3
				n						n	M1
7	7.12.02	16:00	F/A	j	M1,M3	27	10.12.02	19:15	F/C	j	M1,M3
				n						n	
8	7.12.02	16:30	A/F	j	M1,M3	28	10.12.02	19:30	A/F	j	M1,M3
				n						n	
9	7.12.02	17:30	C/F	j	M3	29	10.12.02	20:00	A/F	j	M1,M3
				n	M1					n	
10	7.12.02	18:00	F/A	j	M1,M3	30	10.12.02	20:15	F/A	j	M1,M3
				n						n	
11	7.12.02	19:30	F/C	j	M1,M3	31	10.12.02	20:30	C/F	j	M3
				n						n	M1
12	7.12.02	0:00	F/A	j	M1	32	10.12.02	21:30	C/F	j	M3
				n	M3					n	M1
13	8.12.02	15:00	A/F	j	M3	33	11.12.02	07:00	F/C	j	M1,M3
				n	M1					n	
14	8.12.02	21:30	F/C	j	M1,M3	34	11.12.02	07:30	A/F	j	M1
				n						n	M3
15	9.12.02	12:00	F/A	j	M1,M3	35	11.12.02	10:30	F/A	j	M1,M3
				n						n	
16	9.12.02	19:00	C/F	j	M1	36	11.12.02	18:00	F/C	j	M1,M3
				n	M3					n	
17	9.12.02	20:00	C/F	j	M1,M3	37	11.12.02	18:30	F/C	j	M1,M3
				n						n	
18	9.12.02	21:00	F/C	j	M1,M3	38	11.12.02	18:45	C/F	j	M1,M3
				n						n	
19	9.12.02	22:00	A/F	j	M1,M3	39	11.12.02	19:00	C/F	j	M1,M3
				n						n	
20	9.12.02	23:00	F/A	j	M1,M3	40	11.12.02	20:15	A/F	j	M1,M3
				n						n	

LN	Datum	Uhrzeit	BS	J/N	MN	LN	Datum	Uhrzeit	BS	J/N	MN
41	11.12.02	21:00	F/A	j	M1, M3	46	12.12.02	19:00	A/F	j	M1, M3
				n						n	
42	11.12.02	21:30	C/F	j	M1	47	12.12.02	19:30	A/F	j	M1, M3
				n	M3					n	
43	11.12.02	22:30	A/F	j	M1, M3	48	12.12.02	20:00	C/F	j	M1
				n						n	M3
44	12.12.02	18:00	F/C	j	M3	49	12.12.02	21:00	F/C	j	M1, M3
				n	M1					n	
45	12.12.02	18:30	F/A	j	M1, M3	50	12.12.02	22:00	F/C	j	M1, M3
				n						n	

Tab. 3: Testresultate der Mäuse

11.2. Programm

11.2.1. Testresultate

Testlauf	Buchstaben	Ergebnis?	Testlauf	Buchstaben	Ergebnis?
1	C	Fehlerfigur	26	C	Fehlerfigur
2	A	Fehlerfigur	27	C	Fehlerfigur
3	A	Fehlerfigur	28	C	Fehlerfigur
4	A	Fehlerfigur	29	A	Fehlerfigur
5	A	Fehlerfigur	30	A	Fehlerfigur
6	A	Fehlerfigur	31	C	Fehlerfigur
7	A	Fehlerfigur	32	C	Fehlerfigur
8	C	Fehlerfigur	33	C	Fehlerfigur
9	A	Fehlerfigur	34	C	Fehlerfigur
10	A	Fehlerfigur	35	A	Fehlerfigur
11	A	Fehlerfigur	36	C	Fehlerfigur
12	C	Fehlerfigur	37	C	Fehlerfigur
13	A	Fehlerfigur	38	A	Fehlerfigur
14	C	Fehlerfigur	39	A	Fehlerfigur
15	C	Fehlerfigur	40	C	Fehlerfigur
16	A	Fehlerfigur	41	C	Fehlerfigur
17	C	Fehlerfigur	42	C	Fehlerfigur
18	C	Fehlerfigur	43	A	Fehlerfigur
19	C	Fehlerfigur	44	C	Fehlerfigur
20	C	Fehlerfigur	45	A	Fehlerfigur
21	C	Fehlerfigur	46	C	Fehlerfigur
22	C	Fehlerfigur	47	A	Fehlerfigur
23	A	Fehlerfigur	48	A	Fehlerfigur
24	C	Fehlerfigur	49	C	Fehlerfigur
25	A	Fehlerfigur	50	C	Fehlerfigur

Tab. 4: Programmresultate

11.2.2. Mass Resultate

16 Pixel

Muster	1	2	3	1	2	3	1	2	3	1	2	3
qqq	20	20	20	30	30	30	40	40	40	50	50	50
ja	9246	9238	689	23572	23710	1012	37012	37042	1054	46258	46234	1078
Pixel 1	12	11	7	14	15	7	14	14	7	15	16	6
Pixel 2	64	78	22	91	97	13	108	107	13	113	113	20
Pixel 3	275	265	55	375	394	47	473	468	38	511	509	44
Pixel 4	677	642	76	1096	1169	94	1459	1422	113	1602	1616	109
Pixel 5	1258	1257	143	2328	2366	194	3240	3297	171	3729	3770	175
Pixel 6	1744	1826	186	3912	3910	229	5598	5573	269	6692	6666	280
Pixel 7	1982	1931	140	4931	4963	277	7412	7564	263	9237	9307	244
Pixel 8	1607	1611	50	4831	4826	115	7835	7807	142	10010	9949	169
Pixel 9	1025	1015	9	3487	3486	35	6149	6097	38	8075	8039	30
Pixel 10	460	465	0	1849	1821	0	3367	3397	0	4501	4494	0
Pixel 11	127	129	0	576	580	0	1170	1132	0	1538	1528	0
Pixel 12	14	7	0	81	82	0	187	163	0	234	226	0
Pixel 13	0	0	0	0	0	0	0	0	0	0	0	0
Pixel 14	0	0	0	0	0	0	0	0	0	0	0	0
Pixel 15	0	0	0	0	0	0	0	0	0	0	0	0
Pixel 16	0	0	0	0	0	0	0	0	0	0	0	0

Tab. 5: Anzahl erkannter Muster in Abhängigkeit von der Anzahl veränderter Pixel.

100 Pixel

Muster	1	2	3	Muster	1	2	3
Ja	3254	911	2108	Ja	3254	911	2108
qqq	200	200	200	qqq	200	200	200
Pixel 30	0	0	0	Pixel 53	171	33	117
Pixel 31	0	0	0	Pixel 54	138	25	84
Pixel 32	0	0	1	Pixel 55	121	20	79
Pixel 33	0	2	1	Pixel 56	100	8	48
Pixel 34	2	0	2	Pixel 57	54	5	29
Pixel 35	8	2	9	Pixel 58	55	7	25
Pixel 36	10	5	8	Pixel 59	25	2	13
Pixel 37	26	5	12	Pixel 60	25	2	18
Pixel 38	24	16	23	Pixel 61	12	3	3
Pixel 39	39	24	27	Pixel 62	9	1	3
Pixel 40	57	27	55	Pixel 63	10	0	2
Pixel 41	74	33	47	Pixel 64	2	0	0
Pixel 42	118	51	70	Pixel 65	0	0	0
Pixel 43	140	52	111	Pixel 66	2	0	1
Pixel 44	202	74	119	Pixel 67	0	1	0
Pixel 45	199	81	127	Pixel 68	0	0	0
Pixel 46	218	63	159	Pixel 69	0	0	0
Pixel 47	235	74	172	Pixel 70	0	0	0
Pixel 48	279	79	157	Pixel 71	0	0	0
Pixel 49	246	61	162	Pixel 72	0	0	0
Pixel 50	254	69	157	Pixel 73	0	0	0
Pixel 51	216	60	145	Pixel 74	0	0	0
Pixel 52	180	43	126	Pixel 75	0	0	0

Tab. 6: Anzahl erkannter Muster in Abhängigkeit von der Anzahl veränderter Pixel.