

Söderblom-Gymnasium

Sebastian Kenter

Facharbeit

Thema:

Ein neuronales Netz lernt Zahlen –
Mustererkennung am PC

Fach: Informatik

Kurs: GK-Informatik 2

Fachlehrer: Herr Helmich

Beginn: 02.02.2004

Abgabe: 12.03.2004

Inhaltsverzeichnis

<u>1. Einleitung</u>	3
<u>2. Was sind neuronale Netze?</u>	3
<u>2.1. Aufbau neuronaler Netze</u>	3
<u>2.2. Ein einfaches rezeptives Feld</u>	3
<u>2.3. Grundlagen zur Informationsverarbeitung an Neuronen</u>	5
<u>3. Besondere Vorteile neuronaler Netze</u>	6
<u>3.1. Parallele Informationsverarbeitung</u>	7
<u>3.2. Lernfähige neuronale Netze</u>	7
<u>3.2.1. Lern-Phase und Kann-Phase</u>	7
<u>3.2.2. Bahnung von Synapsen</u>	8
<u>3.2.3. Der Lernprozess im Detail</u>	9
<u>3.2.4. Fehlertoleranz und differenzierte Ergebnisausgabe</u>	10
<u>4. Lernfähige neuronale Netze in der Informatik</u>	10
<u>4.1. Aufbau neuronaler Netze in der Informatik</u>	10
<u>4.2. Anwendungsbereiche</u>	11
<u>4.3. Funktionsweise des Mustererkennungs-Programms als Beispiel</u>	11
<u>Anhang A: Programm-Quelltext</u>	13
<u>Datei Unit1.pas</u>	13
<u>Datei Neuronen.pas</u>	19
<u>Datei NeuroDisplay.pas</u>	23
<u>Anhang B: Hinweise zum Programm</u>	27
<u>Bedienungshinweise</u>	27
<u>Hinweise zum Lernprozess</u>	28
<u>Literatur- und Quellenverzeichnis</u>	30
<u>Erklärung</u>	31

1. Einleitung

Bei neuronalen Netzen handelt es sich eigentlich um „Bestandteile“ des Gehirns von Menschen und Tieren. Informatiker haben sich diese Systeme aus der Natur abgeschaut und zu Nutze gemacht. Auf diese Weise ist aus diesem ursprünglich rein biologischen Thema auch ein Teilgebiet der Informatik geworden.

Die vorliegende Facharbeit gibt daher zunächst Einblick in die Thematik, indem sie das Grundprinzip neuronaler Netze im biologischen Sinn unter anderem anhand eines Beispiels erläutert. Anschließend werden die Besonderheiten solcher Strukturen herausgestellt, die sie vor allem für den Einsatz in der Informatik interessant machen. Wichtige Begriffe in diesem Zusammenhang werden stets hervorgehoben. Zum Schluss wird als Schwerpunkt der Arbeit an einem selbst geschriebenen Programm eine sinnvolle Einsatzmöglichkeit eines neuronalen Netzes in der Informatik gezeigt.

2. Was sind neuronale Netze?

2.1. Aufbau neuronaler Netze

Einfach ausgedrückt sind neuronale Netze in der Biologie mehr oder weniger komplexe Verschaltungen mehrerer Nervenzellen. Solche Verschaltungen sind derartig strukturiert, dass sie bestimmte Berechnungs- und Denkprozesse durchführen können, weswegen sich die meisten neuronalen Netze im Gehirn befinden. Je nach genauem Einsatzzweck sieht die Struktur eines neuronalen Netzes von Fall zu Fall anders aus, doch das Prinzip ist überall gleich.

Jede Nervenzelle bekommt im Einzelnen über ihre Dendriten Informationen angeliefert und gibt über das Axon Informationen aus. Genauso bekommt ein neuronales Netz Informationen als Berechnungsgrundlage und gibt als Ergebnis andere Informationen zurück. Die eingehenden Informationen werden von den in der so genannten **Input-Schicht** am Anfang des Netzes befindlichen Neuronen aufgenommen. Die Axone dieser Neuronen sind über Synapsen mit den Dendriten anderer Nervenzellen verbunden, sodass die Informationen von Neuron zu Neuron weitergeleitet und -verarbeitet werden können. Je nach Komplexität des Netzes können viele verschiedene Neuronen auf diese Art hintereinander geschaltet sein. Zum Schluss wird das entstandene Ergebnis über Neuronen in der **Output-Schicht** am Ende des Netzes ausgegeben.

2.2. Ein einfaches rezeptives Feld

Einfache Beispiele dafür, wie solch eine geschickte Verschaltung von Nervenzellen eine konkrete Aufgabe erfüllt, lassen sich in der Netzhaut vieler Lebewesen finden. Da es sich hierbei um mehrere Lichtrezeptoren der Netzhaut handelt, die über das neuronale Netz miteinander verknüpft sind und die Input-Schicht bilden, spricht man in diesem Fall auch von **rezeptiven Feldern**. Wir haben sie bereits im Biologie-Unterricht kennen gelernt. Das folgende rezeptive Feld kann, wenn es auch stark vereinfacht ist, senkrechte und waagerechte Linien erkennen und unterscheiden:

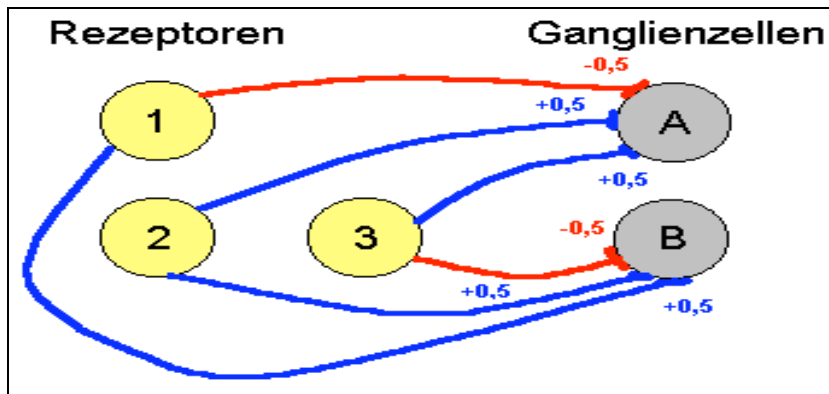


Abbildung 1: Ein rezeptives Feld zur Linienerkennung

Das Schwellenpotenzial von A und B soll bei 1 liegen. Wenn das Bild einer waagerechten Linie auf diesen Teil der Netzhaut fällt, werden nur die nebeneinander liegenden Rezeptoren 2 und 3 belichtet. Beide zusammen können über die erregenden Synapsen das Membranpotenzial von Ganglienzelle A auf den Schwellenwert 1 erhöhen. Ganglienzelle B dagegen wird nur von Rezeptor 2 erregt und von Rezeptor 3 sogar mit der gleichen Stärke gehemmt, sodass das Membranpotenzial hier insgesamt 0 beträgt. Bei einer waagerechten Linie wird folglich nur Ganglienzelle A erregt.

Bei einer senkrechten Linie verhält sich alles genau umgekehrt: Die Rezeptoren 1 und 2 werden belichtet, beide zusammen aktivieren Ganglienzelle B, und Ganglienzelle A wird zwar von Rezeptor 2 erregt, von Rezeptor 1 jedoch gehemmt, sodass nur Ganglienzelle B aktiviert wird.

Die Aktivierung von Zelle A entspricht also der Information, dass das rezeptive Feld eine waagerechte Linie erkannt hat, während es sich bei Aktivierung von Zelle B um eine senkrechte Linie handelt. Die Ganglienzellen bilden daher in diesem Netz die Output-Schicht. Zwischen Input- und Output-Schicht befinden sich keine weiteren Neuronen; bei diesem einfachen Beispiel reichen die Synapsen an den Ganglienzellen bereits zur Verrechnung der eingehenden Informationen aus.

Wenn es keine Linie, sondern eine Fläche ist, deren Licht auf die Netzhaut fällt, werden alle drei Rezeptoren aktiviert. Ganglienzelle A wird von 2 und 3 mit einer Stärke von je 0,5 erregt und von 1 mit der gleichen Stärke gehemmt. Genauso wird Ganglienzelle B von 1 und 2 um je 0,5 erregt und von 3 um 0,5 gehemmt. Keine der beiden Zellen erreicht also ihr Schwellenpotenzial, was auch logisch ist, denn schließlich handelt es sich hierbei um überhaupt keine Linie. Ohne die hemmenden Synapsen, die auf den ersten Blick vielleicht überflüssig erscheinen, würden in dieser Situation beide Ganglienzellen aktiviert werden, das rezeptive Feld hätte also falsch reagiert.

Bei einfachen neuronalen Netzen wie diesem lassen sich die unterschiedlichen Reaktionen auf verschiedene eingehende Informationen gut in einer Tabelle zusammenfassen:

Objekt	Input-Schicht			Output-Schicht	
	Rezeptor 1	Rezeptor 2	Rezeptor 3	Zelle A	Zelle B
<i>nichts</i>	0	0	0	0	0
<i>waagerechte Linie</i>	0	1	1	1	0
<i>senkrechte Linie</i>	1	1	0	0	1
<i>Fläche</i>	1	1	1	0	0

Tabelle 1: Übersicht über das Verhalten (In- und Outputs) des neuronalen Netzes zur Linienerkennung (1 = Neuron ist aktiviert)

Die so gewonnene Information, ob es sich um eine senkrechte, waagerechte oder um gar keine Linie handelt, wird von den Ganglienzellen zum Gehirn weitergeleitet, oder vielmehr zum „eigentlichen“ Gehirn; schließlich ist die Netzhaut gerade wegen der dort stattfindenden Vorverarbeitung von Informationen schon Teil des Gehirns.

Die durch das schon sehr spezialisierte neuronale Netz in diesem Beispiel gewährleistete schnelle und direkte Erkennung unterschiedlicher Linien eignet sich gut dazu, bestimmte Reflexe auszulösen, etwa bei einem Frosch den Beutefangreflex nach Erkennung eines Wurms (waagerechte Linie) und den Fluchreflex, wenn ein Feind (senkrechte Linie) erblickt wird. Dieses Beispiel soll nur das Prinzip neuronaler Verschaltungen verdeutlichen, echte Mustererkennungs-Mechanismen sind weitaus komplizierter. So haben solche rezeptiven Felder in Wirklichkeit natürlich viel mehr Rezeptoren, denn es wäre übertrieben, einen Lichtreiz, der nur zwei benachbarte Rezeptoren aktiviert, schon als Linie zu bezeichnen. Das vorgestellte einfache Prinzip lässt sich allerdings auf ein Raster von beliebig vielen Rezeptoren übertragen. Auf weitere Verbesserungen eines solchen Mustererkennungs-Mechanismus, für die das Verständnis anderer Merkmale neuronaler Netze notwendig ist, wird im weiteren Verlauf der Facharbeit noch näher eingegangen, denn an diesem Beispiel lässt sich noch vieles mehr zeigen.

2.3. Grundlagen zur Informationsverarbeitung an Neuronen

Wie am vorangegangenen Beispiel deutlich wurde, spielen bei der eigentlichen Verrechnung der Informationen innerhalb des Netzes Anzahl, Stärke und Art (erregend oder hemmend) der Synapsen an einem Neuron sowie das Schwellenpotenzial des betreffenden Neurons eine Rolle. Es sollen nun noch grundlegende Verrechnungsvorgänge, die in unterschiedlichsten Anwendungen neuronaler Netze zum Einsatz kommen, erläutert werden.

Logische Verschaltungen von Neuronen werden in Ewert, Prof. Dr. J.-P.: Nerven- und Sinnesphysiologie, S. 137 f., beschrieben. Bei ihnen kommt es zunächst nicht auf die Stärke der Reize an, sondern darauf, ob überhaupt ein Reiz ankommt oder nicht. Angenommen, zwei erregende Synapsen führen zu einem Neuron, dessen Schwellenpotenzial so gering ist, dass bereits die Erregung einer Synapse zur Aktivierung ausreicht. Das Neuron ist in diesem Fall immer dann erregt, gibt also immer dann seinerseits einen Reiz aus, wenn von der einen ODER von der anderen Synapse ein Reiz ankommt (natürlich auch, wenn von beiden gleichzeitig Reize eintreffen). Aus diesem Grund spricht man hier von einer **ODER-Schaltung**. Ist das Schwellenpotenzial jedoch so hoch, dass die Erregung einer Synapse nicht genügt, um das Neuron zu aktivieren, so gibt dieses erst dann einen Reiz aus, wenn von der einen UND von der anderen Synapse ein Reiz eintrifft. Hierbei handelt es sich deshalb um eine **UND-Schaltung**. Der vorgestellte Linienerkennungs-Mechanismus bietet gute Beispiele für UND-Schaltungen: Es wird erst

dann eine Linie erkannt, wenn Reize von zwei benachbarten Rezeptoren gleichzeitig (von einem Rezeptor UND einem benachbarten Rezeptor) eintreffen. Die bei der Linienerkennung zum Einsatz kommende UND-Schaltung ist noch zusätzlich mit einer **NICHT-Schaltung** kombiniert, denn der Rezeptor, der für die jeweilige Linie nicht infrage kommt, darf NICHT aktiviert sein (hemmende Synapse).

Input		Output
Synapse 1	Synapse 2	Neuron
0	0	0
0	1	0
1	0	0
1	1	1

Tabelle 2: Verhalten einer UND-Schaltung

Input		Output
Synapse 1	Synapse 2	Neuron
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 3: Verhalten einer ODER-Schaltung

Eine häufig durchgeführte Berechnung ist die **Addition** von Informationen. In neuronalen Netzen kann sie leicht realisiert werden: Es befinden sich wieder zwei erregende Synapsen an einem Neuron, diesmal wird auch die Stärke der Erregungen betrachtet. Die Erregungsstärken der beiden Synapsen werden im Neuron addiert, weil jede einzelne Synapse das Membranpotenzial um einen bestimmten Wert heraufsetzt.¹ Das Neuron leitet also über sein Axon die Summe der beiden über die Synapsen eingehenden Erregungen weiter, vorausgesetzt, das Schwellenpotenzial ist nicht zu groß. In dem Beispiel mit der Linienerkennung wird ein Additionsvorgang verwendet, um auch die Länge einer Linie ermitteln zu können: Je länger eine Linie ist, desto mehr benachbarte Rezeptoren werden aktiviert, desto mehr Synapsen setzen also das Membranpotenzial der jeweiligen Ganglienzelle herauf: Die Erregungen der einzelnen Rezeptoren werden addiert. Demnach ist die Stärke der Erregung dieser Ganglienzelle ein Maß für die Länge der erkannten Linie. Mit solchen Methoden können viele verschiedene Berechnungen dieser Art durchgeführt werden, zum Beispiel auch **Subtraktion** mithilfe hemmender Synapsen.

3. Besondere Vorteile neuronaler Netze

Logische Verknüpfungen wie UND- und ODER-Schaltungen sowie Berechnungen wie Addition sind nicht nur Grundlagen der Informationsverarbeitung in der Biologie, sondern auch Grundlagen der Informatik. Es besteht eine gewisse Ähnlichkeit zwischen Verschaltungen einzelner Neuronen und Verschaltungen elektronischer Bauteile. Tatsächlich bezeichnet der Begriff „neuronale Netze“ dem heutigen Verständnis nach „elektronische Schaltsysteme“², die „ursprünglich in Anlehnung an die netzartige Struktur der Nervenzellen beispielsweise im Gehirn von Menschen und Tieren entworfen wurden“³. In den vorangegangenen Ausführungen habe ich diesen Ausdruck auch auf die biologischen Vorbilder bezogen, weil das zu Grunde liegende Prinzip genau das gleiche ist. Doch warum werden neuronale Netze überhaupt in der Informatik eingesetzt, wenn sie, wie oben erwähnt, im Einzelnen auf Berechnungen und Schaltungen basieren, wie sie in der Informatik sowieso üblich sind? Als Antwort auf diese Frage sind noch zwei herausragende Eigenschaften komplexerer neuronaler Netze zu untersuchen.

¹ Vgl. Nerven- und Sinnesphysiologie, S. 37 f.

² Miram, W./Scharf, K.-H. (Hg.): Biologie heute SII, S. 322

³ Hasebrook, J.: Neuronale Netzwerke, in: Microsoft® Encarta® 99 Enzyklopädie

3.1. Parallele Informationsverarbeitung

Zum Einen können solche Netze zusätzlich zur Input- und Output-Schicht aus mehreren Zwischenschichten bestehen, in denen die eingegangenen Informationen parallel verarbeitet werden.⁴ Als Beispiel könnte man die Linienerkennung so verbessern, dass auch registriert wird, ob sich das jeweilige Objekt in Längsrichtung oder quer zur Längsrichtung bewegt. Zu diesem Zweck kommen zwei Schaltungen zum Einsatz, wovon je eine für die Erkennung einer Bewegung in Längsrichtung und einer Bewegung quer zur Längsrichtung zuständig ist. Wie diese Schaltungen im Einzelnen funktionieren, ist in diesem Zusammenhang weniger interessant. Wichtig ist, dass beide Mechanismen parallel geschaltet sind, was eine schnellere Informationsverarbeitung ermöglicht. Die Ergebnisse beider Erkennungsvorgänge werden am Schluss miteinander verrechnet, und so wird entschieden, in welche Richtung sich das Objekt bewegt. Es ist beispielsweise für einen Frosch überlebenswichtig, einen Feind (Linie mit Bewegung quer zur Längsrichtung) im Gegensatz zu einer Beute (Bewegung in Längsrichtung) schnell identifizieren zu können, damit in möglichst kurzer Zeit ein Fluchtverhalten ausgelöst werden kann.⁵

In Situationen, bei denen es auf schnelle Berechnungen und Reaktionen ankommt, ist parallele Informationsverarbeitung natürlich auch in der Informatik nützlich. Sie zählt zu den Techniken, die als Eigenschaft neuronaler Netze aus der Biologie übernommen wurden. Auf dieses Thema soll hier jedoch nicht weiter eingegangen werden, nicht zuletzt weil echte Parallelverarbeitungen nur mithilfe von Rechnersystemen mit mehreren Prozessoren, wie sie für den Alltagsgebrauch nicht üblich sind, durchgeführt werden können.

3.2. Lernfähige neuronale Netze

Allerdings gibt es noch eine weitere Besonderheit, durch die sich viele neuronale Netze auszeichnen. Auch dies lässt sich anhand der Mustererkennung verdeutlichen. Natürlich reicht es meist nicht aus, wenn ein Lebewesen nur Linien erkennen kann, es sollte auch auf kompliziertere Objekte reagieren können. So muss es auch Mustererkennungs-Mechanismen geben, welche einem Menschen ermöglichen, Buchstaben und Zahlen zu lesen. Im Endeffekt bestehen auch solche Zeichen nur aus auf bestimmte Weise miteinander verbundenen Linien. Das Interessante hierbei ist, dass Menschen diese Zeichen nicht von Anfang an erkennen können, sondern eine Schrift erst erlernen müssen. Es muss also lernfähige neuronale Netze geben, denen man für jeweils vorgegebene Input-Informationen bestimmte Reaktionen „antrainieren“ kann. Beispielsweise soll ein rezeptives Feld lernen können, dass es, wenn die belichteten Rezeptoren ringförmig angeordnet sind, als Output meldet, dass es sich bei dem erkannten Objekt um den Buchstaben O oder die Ziffer 0 handelt.

3.2.1. Lern-Phase und Kann-Phase

Wenn ein Mensch eine neue Schrift lernen soll, zeigt man ihm nacheinander die Buchstaben und sagt jeweils dazu, wie sie heißen. Zum Beispiel legt man einem Grundschulkind einen schwarzen Kreis auf einem Blatt Papier vor und erklärt ihm, dies sei ein O. Bei einem Lernvorgang führt man einem Menschen also die notwendigen Eingangsinformationen (schwarzer Kreis) zu und gibt das dazu passende Ergebnis (Buchstabe O) vor. Dies tut man so lange, bis er von den Eingangsinformationen von sich aus auf das richtige Ergebnis kommt, in diesem Fall den Buchstaben O selbst erkennt. Der

⁴ Vgl. Meyers Lexikonredaktion (Hg.): Schülerduden Informatik, neuronales Netz, S. 331

⁵ Vgl. Nerven- und Sinnesphysiologie, S. 140

Prozess, bei dem das Ergebnis vorgegeben wird, damit man die selbstständige Erkennung lernt, heißt dementsprechend **Lern-Phase**, und die darauf folgende selbstständige Erkennung an sich, bei der man sich das Gelernte zu Nutze macht, um selbst das richtige Ergebnis zu finden, wird **Kann-Phase** genannt.⁶

Hinter diesem lernfähigen Buchstabenerkennungs-Mechanismus stehen bei einem Menschen komplizierte Vorgänge, für die neuronale Netze benötigt werden. Da in diesem Zusammenhang die Funktionsweise lernfähiger Netze verdeutlicht werden soll, wird jetzt stark vereinfacht als Modellvorstellung angenommen, es würde hier lediglich ein neuronales Netz in Form eines rezeptiven Feldes verwendet werden, das so ähnlich aufgebaut ist wie das zur Erkennung einfacher Linien: Die Input-Schicht wird wie bei jedem rezeptiven Feld wieder von einem Raster aus Rezeptoren gebildet, und so wie es in der Output-Schicht der Linienerkennung je eine Ganglienzelle für senkrechte und waagerechte Linien gab, steht hier jede Ganglienzelle für einen anderen Buchstaben. In der Lern-Phase wird vorgegeben, welche Ganglienzelle jeweils aktiviert werden soll.

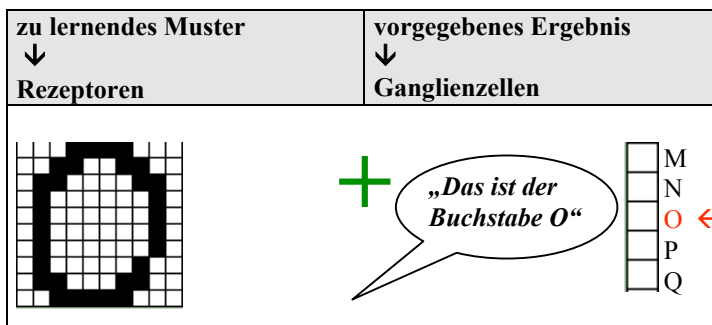


Abbildung 2: grundlegendes Lernprinzip neuronaler Netze

3.2.2. Bahnung von Synapsen

Um lernfähig zu sein, muss ein neuronales Netz irgendetwas an seiner Struktur oder an seinen Komponenten ändern können, denn wie sonst sollte es das gelernte Verhalten speichern können. Diese dauerhafte Speicherung erfolgt meist durch so genannte **Bahnung** der Synapsen. Dabei wird, zumindest bei neuronalen Netzen in Lebewesen, die Menge des von einer Synapse ausgeschütteten Neurotransmitters verändert, was zur Folge hat, dass die Synapse bei Aktivierung das anschließende Neuron mit einer anderen Stärke erregt bzw. hemmt als vorher. Anders ausgedrückt liefert die Synapse nach ihrer Bahnung eine andere Information an das nachfolgende Neuron, sodass sich auch das von der Output-Schicht ausgegebene Ergebnis der Informationsverarbeitung verändern kann, obwohl der Input vielleicht der gleiche ist wie vor der Bahnung. Durch die Bahnung der Synapsen ist es einem neuronalen Netz also möglich, auf die gleichen eingehenden Informationen unterschiedlich zu reagieren, je nachdem, wie es „trainiert“ wurde. Wie die Bahnung in der Natur genau vonstatten geht, ist für den Einsatz in der Informatik natürlich uninteressant. Zu diesem Thema soll lediglich festgehalten werden, dass die Bahnung, je nachdem, ob sie mit positivem oder negativem „Vorzeichen“ erfolgt, eine Synapse verstärken bzw. abschwächen kann.

⁶ Vgl. Nerven- und Sinnesphysiologie, S. 139

3.2.3. Der Lernprozess im Detail

Doch welche Synapsen müssen nun konkret wie gebahnt werden, damit ein neuronales Netz ein bestimmtes Verhalten lernt? Um das herauszufinden, kann man sich zunächst noch einmal das Prinzip des Linienerkennungs-Mechanismus (Abbildung 1) betrachten und versuchen, es auf das Prinzip einer Erkennung beliebiger Muster zu verallgemeinern: Jeder belichtete Rezeptor trägt zur Erregung der für das jeweilige Objekt zuständigen Ganglienzelle bei. Natürlich kann ein Rezeptor auch mehrere Ganglienzellen erregen, aber nur die für das Objekt zuständige Zelle wird von so vielen Rezeptoren erregt, dass sie wegen Überschreitung des Schwellenpotenzials auch aktiviert wird. Dies passiert eben deswegen, weil alle Rezeptoren, die von diesem Objekt belichtet werden, auch die betreffende Ganglienzelle erregen. So werden in der Abbildung sowohl Rezeptor 2 wie auch Rezeptor 3 bei einer waagerechten Linie erregt. Rezeptor 2 erregt zwar auch Ganglienzelle B, doch Ganglienzelle A wird von beiden Rezeptoren erregt, weswegen die Erregung hier zu einer Aktivierung führt. Zusätzlich hemmt jeder Rezeptor, der von einem Objekt eben nicht belichtet werden kann (Rezeptor 1 bei waagerechter Linie), die Ganglienzelle für das betreffende Objekt (in diesem Fall B).

Bei einem Netz, dem beliebige Muster „antrainiert“ werden können, steht zunächst nicht fest, welche Rezeptoren genau zur Erregung welcher Ganglienzelle beitragen müssen. Daher muss ein solches flexibles neuronales Netz auf jeden Fall jeden Rezeptor durch je eine Synapse mit jeder Ganglienzelle verbinden. In der Lern-Phase werden die Synapsen dann so gebahnt, dass diejenigen Synapsen, die sich zwischen einer Ganglienzelle, die für ein bestimmtes Muster zuständig sein soll, und einem Rezeptor befinden, der von genau diesem Muster erregt wird, möglichst stark erregend sind und alle anderen Synapsen an diesem Rezeptor ihre Ganglienzelle entsprechend hemmen. Dies geschieht einfach dadurch, dass in einem Lernschritt alle Synapsen, die von den belichteten Rezeptoren zu derjenigen (vorgegebenen) Ganglienzelle führen, die für das richtige Ergebnis zuständig ist, durch Bahnung verstärkt werden; alle anderen Synapsen an diesen Rezeptoren können zusätzlich abgeschwächt werden.⁷ Wenn also das Grundschulkind den Buchstaben O lernen soll, werden diesem einfachen Modell nach alle Synapsen verstärkt, welche diejenigen Rezeptoren, die den Kreis wahrnehmen, mit derjenigen Ganglienzelle verbinden, die ab sofort für den Buchstaben O stehen soll; alle anderen Synapsen dieser Rezeptoren werden abgeschwächt.

In der Informatik wurden Regeln aufgestellt, wie z. B. die Hebb-Regel⁸, die besagen, in welchem Maße die einzelnen Synapsen bei jedem einzelnen Lerndurchgang gebahnt werden müssen, um einen optimalen Lernerfolg zu erzielen. Für das hier behandelte einfache Modell reicht es jedoch aus, alle betreffenden Synapsen jedes Mal um den gleichen Wert zu verstärken bzw. abzuschwächen.

Das Problem, einem neuronalen Netz alle Buchstaben und Zahlen „beizubringen“, ist natürlich etwas komplexer als die Entwicklung eines rezeptiven Feldes zur Erkennung bloßer Linien. Schließlich handelt es sich hier um mehrere Zeichen, die immer anhand bestimmter Kombinationen aktivierter Rezeptoren möglichst zuverlässig identifiziert und voneinander unterschieden werden sollen. Diese Aufgabe ist mit dem vorgestellten Modell, zusammen mit dem simplen Bahnungs-Mechanismus für die Lern-Phase, tatsächlich zu bewerkstelligen, allerdings sind viele Lerndurchgänge mit jedem Zeichen erforderlich, ehe

⁷ Vgl. Neuronale Netzwerke, in: Microsoft® Encarta® 99 Enzyklopädie

⁸ Vgl. ebenda

das neuronale Netz seine Synapsen der zu lernenden Schrift optimal angepasst hat: Niemand kann das Alphabet lernen, indem er sich jeden Buchstaben einmal anschaut.

3.2.4. Fehlertoleranz und differenzierte Ergebnisausgabe

Hinzu kommt die Tatsache, dass von einem Menschen geschriebene Zeichen nie genau gleich aussehen, obwohl es sich eigentlich um das gleiche Zeichen handelt. Deutlich wird dies besonders bei der unterschiedlichen Handschrift mehrerer Menschen; aber auch dann, wenn ein Mensch zweimal den gleichen Buchstaben aufschreibt, wird man immer einen Unterschied feststellen können. Es müssen bei einem Zeichen also nicht immer genau die gleichen Rezeptoren erregt sein. Auch damit muss ein neuronales Netz fertig werden. Doch gerade hier zeigt sich eine enorme Stärke lernfähiger neuronaler Netze: Wenn sich nämlich auch bei den Lernvorgängen – die, wie bereits erwähnt, pro Zeichen mehrfach stattfinden müssen – die als Input verwendeten Muster geringfügig voneinander unterscheiden, werden diese unterschiedlichen Versionen „mitgelernt“ und in der Kann-Phase berücksichtigt. Aus diesem Grund kann ein solches Netz auch bei mehreren leicht unterschiedlichen Mustern das gleiche Ergebnis ausgeben und sogar neue Muster richtig zuordnen, die selbst in der Lern-Phase nie genau so aufgetreten sind, sondern nur in leicht veränderter Form.⁹ Man spricht auch von einer **Fehlertoleranz** solcher Systeme.

Man kann sich die Vorteile solcher Mechanismen noch weiter zu Nutze machen, indem man nicht mehr davon ausgeht, dass als eindeutiges Ergebnis genau eine Ganglienzelle aktiviert wird, weil sie von den Synapsen über den Schwellenwert erregt wird, sondern die Erregungsstärken aller Ganglienzellen betrachtet und miteinander vergleicht. Jedem Menschen kann es passieren, dass er ein Zeichen nicht lesen kann, obwohl er es gelernt hat. Er fragt sich dann zum Beispiel, ob es sich um ein handschriftliches u oder n handelt. Das neuronale Netz erkennt in diesem Fall Kombinationen von erregten Rezeptoren, die zum Teil auf ein u und zum Teil auf ein n hindeuten. Praktischerweise werden dann automatisch beide jeweiligen Ganglienzellen etwa gleich stark erregt, sodass der Mensch zwischen beiden Buchstaben abwägen kann. Aus der Stärke der Erregung jeder Ganglienzelle im Verhältnis zu den Erregungsstärken anderer Ganglienzellen lässt sich also die Wahrscheinlichkeit dafür berechnen, dass das ihr zugeordnete Muster erkannt wurde. Möchte man zu einem endgültigen und eindeutigen Ergebnis kommen, muss man durch eine Art „Extremwertschaltung“¹⁰ ermitteln, welche Ganglienzelle am meisten erregt ist.

4. Lernfähige neuronale Netze in der Informatik

4.1. Aufbau neuronaler Netze in der Informatik

In der Informatik ist der Aufbau eines solchen Netzes aus (in diesem Fall virtuellen) Neuronen und Synapsen den biologischen Vorbildern nachempfunden, ist doch gerade diese Struktur entscheidend für die besondere Funktionsweise eines derartigen Systems. Die Stärken der virtuellen Synapsen werden hier durch so genannte Gewichte oder Gewichtszahlen ausgedrückt.¹¹ Es handelt sich dabei um im System gespeicherte Zahlenwerte, die für jede Synapse angeben, wie stark sie bei Aktivierung die Erregung des nachfolgenden Neurons beeinflusst. Beim Bahnen einer Synapse wird einfach ihre Gewichtung verändert, wobei die bereits angesprochenen Regeln zum Einsatz kommen.

⁹ Vgl. Miram, W.: Informationsverarbeitung, S. 102, rechte Spalte

¹⁰ Ebenda, S. 102, linke Spalte

¹¹ Vgl. Neuronale Netzwerke, in: Microsoft® Encarta® 99 Enzyklopädie

4.2. Anwendungsbereiche

Lernfähige neuronale Netze werden zur Lösung wichtiger Probleme in der Informatik benötigt, die ohne deren Hilfe nicht ohne Weiteres bewältigt werden können. Allen voran geht dabei die Mustererkennung, die somit auch an Computern genauso funktioniert, wie es in den letzten Abschnitten am Modell erklärt wurde: Man „füttert“ das System in der Lern-Phase zunächst mehrfach mit den zu erkennenden Mustern, im einfachsten Fall Buchstaben und Zahlen, wobei es seine Gewichte anpasst (entspricht dem biologischen Vorgang der Bahnung). Wenn das System ausreichend trainiert ist, erkennt man dies daran, dass „die Gewichte stabil bleiben“.

Ohne einen solchen Mechanismus würde es einem Rechner wesentlich schwerer fallen, von einem Menschen geschriebene Zeichen zu identifizieren. Man könnte einem neuronalen Netz jedoch auch menschliche Gesichtszüge als Muster antrainieren, um eine Verwendung zur Gesichtserkennung etwa bei einem Bankautomaten zu ermöglichen. Es können an dieser Stelle nicht alle Einsatzorte aufgezählt werden. Ganz allgemein formuliert versucht man sich das Prinzip neuronaler Netze bei solchen Problemen zu Nutze zu machen, die sich nicht mit ausreichend schnellen Algorithmen lösen lassen.¹² Anstatt sich also einen konkreten Algorithmus als Mechanismus auszudenken, der zu den möglichen Input-Informationen immer direkt genau das richtige Ergebnis liefert, trainiert man einfach ein neuronales Netz so oft mit Input-Informationen und den vorgegebenen Ergebnissen, bis es fast genauso zuverlässig funktioniert, wie es ein Algorithmus tun würde, der aber vielleicht nur sehr schwer zu finden wäre. Da neuronale Netze auch vom Gelernten leicht abweichende Strukturen von alleine richtig zuordnen können, zählt man sie oft zu den Systemen mit *künstlicher Intelligenz*.¹³

Das im Folgenden vorgestellte Programm zeigt, wie man ein neuronales Netz zur Mustererkennung konkret am Rechner realisieren kann.

4.3. Funktionsweise des Mustererkennungs-Programms als Beispiel

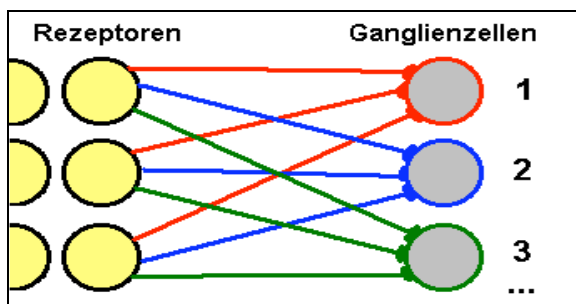


Abbildung 3: Jeder Rezeptor ist mit jeder Ganglienzelle über eine Synapse verbunden – diese Struktur wird auch im vorgestellten Programm eingesetzt

In dem im Rahmen dieser Facharbeit erstellten Delphi-Programm wurde ein neuronales Netz zur Mustererkennung implementiert, das nach dem auf den letzten Seiten vorgestellten Prinzip arbeitet. Die einzelnen Komponenten sollen wie beim biologischen Vorbild mit den Begriffen Rezeptoren, Synapsen und Ganglienzellen bezeichnet werden. Zur Vereinfachung sollen hier keine Buchstaben, sondern nur die Zahlen von 0 bis 9 identifiziert werden können, weswegen 10 Ganglienzellen benötigt werden, welche die Output-Schicht bilden. Die Input-Schicht besteht aus einem Raster von 10*10, also

¹² Vgl. Schülerduden Informatik, neuronales Netz, S. 331

¹³ Vgl. Nerven- und Sinnesphysiologie, S. 142

insgesamt 100 Rezeptoren. An jedem Rezeptor befinden sich 10 Synapsen (insgesamt $100 \cdot 10 = 1000$), die den einzelnen Ganglienzellen zugeordnet sind. Die Tatsache, dass es sich um drei verschiedene Komponenten handelt, die jeweils in großer Zahl vorhanden sind, spricht für eine objektorientierte Programmierung auf Basis der drei Klassen TRezeptor, TSynapse sowie TGanglienzelle. Die Klassen TRezeptor und TGanglienzelle stellen beide unterschiedliche Sorten von Nervenzellen dar, deshalb wurde zusätzlich eine Klasse TNeuron geschrieben, von der die Klassen TRezeptor und TGanglienzelle grundlegende Eigenschaften erben. Ein Rezeptor-Objekt muss allerdings im Unterschied zu einem Ganglienzellen-Objekt zum Beispiel noch die von ihm ausgehenden Synapsen als Unterobjekte enthalten, damit es über sie die Ganglienzellen „erregen“ oder „hemmen“ kann.

Die Gewichtungszahlen der Synapsen werden ebenfalls durch Integer-Werte dargestellt, die während der Lern-Phase von einer Bahnungs-Methode geändert werden. In der Kann-Phase werden die Gewichtungszahlen aller erregten Synapsen zur Berechnung der Erregung der jeweiligen Ganglienzelle aufaddiert. Damit die gelernten Zahlenmuster beim Beenden des Programms nicht verloren gehen, können die Gewichte aller Synapsen in einer Datei auf die Festplatte gespeichert (und umgekehrt wieder eingelesen) werden.

Der kommentierte Quelltext ist zusammen mit ergänzenden Hinweisen im Anhang zu finden.

Anhang A: Programm-Quelltext

Datei Unit1.pas

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Neuronen, NeuroDisplay, Buttons;

type
  TForm1 = class(TForm)
    ImageZeichnung: TImage;
    BitBtnAuswerten: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    LabelPhase: TLabel;
    ButtonPhase: TButton;
    ListBoxLernZahl: TListBox;
    LabelLernZahl: TLabel;
    GroupBox1: TGroupBox;
    BitBtnLaden: TBitBtn;
    BitBtnSpeichern: TBitBtn;
    Shapel: TShape;
    BitBtnZeichnungLoeschen: TBitBtn;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    procedure FormCreate(Sender: TObject);
    procedure ImageZeichnungMouseDown(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure ImageZeichnungMouseMove(Sender: TObject; Shift: TShiftState;
      X, Y: Integer);
    procedure ImageZeichnungMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure BitBtnAuswertenClick(Sender: TObject);
    procedure ButtonPhaseClick(Sender: TObject);
    procedure BitBtnZeichnungLoeschenClick(Sender: TObject);
    procedure BitBtnSpeichernClick(Sender: TObject);
    procedure BitBtnLadenClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);

    procedure InitLernphase;
    //wechselt zur Lern-Phase
    procedure InitKannphase;
    //wechselt zur Kann-Phase

    procedure NeuronenReset;
    //setzt alle Neuronen zurück
    procedure ZeichnungAufRezeptoren;
    //erregt die Rezeptoren anhand der Zeichnung auf der Zeichenfläche,
    //womit auch bereits die Erregung der Ganglienzellen (Kann-Phase)
    //bzw. Bahnung der Synapsen (Lern-Phase) ausgelöst wird
    procedure ShowOutput;
    //zeigt Output des Netzes und die daraus berechneten Wahrscheinlichkeiten
    //mithilfe der Klassen aus der Unit NeuroDisplay auf dem Formular an
    procedure ShowErkannteZahl;
    //ermittelt die Zahl mit der höchsten Ganglienzellen-Erregung und zeigt diese an
    //entspricht der im Theorie-Teil erwähnten "Extremwertschaltung"

    procedure Speichern;
    //zeigt den Speichern unter-Dialog an und ruft dann SpeichereDort auf
    procedure SpeichereDort(Dateiname: string);
    //speichert Lernzähler und Gewichte unter dem übergebenen Dateinamen
    function Laden: boolean;
    //zeigt den Öffnen-Dialog an und ruft dann LadeDas auf
    //gibt false zurück, wenn der Dialog abgebrochen wurde
    procedure LadeDas(Dateiname: string);
```

```

    //lädt Lernzähler und Gewichte aus der angegebenen Datei
private
  { Private declarations }
  Lernphase,           //true=Lernphase, false=Kannphase
  MouseIsDown: boolean; //Maustaste gedrückt?

  LernZahl: integer;   //zu lernende Zahl in der Lern-Phase

  Rezeptoren: array[0..9,0..9] of TRezeptor;
  Ganglienzellen: TGanglienzellen;
  ErregungsLabels: array[0..9] of TErregungsLabel;
  ErgebnisAnzeigen: array[0..9] of TErgebnisAnzeige;
  LernZaehler: array[0..9] of TLernZaehler;
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var i,x,y: integer;
begin
  //Initialisierung
  SaveDialog1.FileName:='Unbenannt.lpf';
  ImageZeichnung.Canvas.Pen.Color:=clBlack;

  //Rezeptorenraster erzeugen
  for x:=0 to 9 do
    for y:=0 to 9 do
      Rezeptoren[x,y]:=TRezeptor.Create(Form1,x,y);

  //für jede zu erkennende Zahl (0 bis 9) die entsprechenden Ganglienzellen
  //und Anzeigen erzeugen
  for i:=0 to 9 do begin
    Ganglienzellen[i]:=TGanglienzelle.Create(Form1,i);
    ErregungsLabels[i]:=TErregungsLabel.Create(Form1,i);
    ErgebnisAnzeigen[i]:=TErgebnisAnzeige.Create(Form1,i);
    LernZaehler[i]:=TLernZaehler.Create(Form1,i);
    ListBoxLernZahl.Items.Add(inttostr(i));
  end;

  //man beginnt in der Lern-Phase
  InitLernphase;
end;

procedure TForm1.ImageZeichnungMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  //beim Runterdrücken der Maustaste auf der Zeichenfläche
  //den Stift an die Zeigerposition bewegen
  MouseIsDown:=true;
  ImageZeichnung.Canvas.MoveTo(X,Y);
end;

procedure TForm1.ImageZeichnungMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  //wenn der Mauszeiger bei gedrückter Maustaste über die Zeichenfläche
  //bewegt wird, an der Zeigerposition mitzeichnen
  if MouseIsDown
    then ImageZeichnung.Canvas.LineTo(X,Y);
end;

procedure TForm1.ImageZeichnungMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  MouseIsDown:=false;
end;

```

```

procedure TForm1.BitBtnAuswertenClick(Sender: TObject);
begin
  NeuronenReset;
  if Lernphase //in der Lern-Phase...
  then begin
    with ListBoxLernZahl do
      if ItemIndex>-1 //wenn eine Zahl in der ListBox
      then begin //ausgewählt wurde, wird sie
global als
        LernZahl:=strtoint(Items[ItemIndex]); //LernZahl gespeichert, dann
werden
        ZeichnungAufRezeptoren //die Rezeptoren aktiviert
        LernZaehler[LernZahl].Hochzaehlen; //und der entsprechende Zähler
wird hochgezählt
        end
      else //Fehlermeldung, wenn keine Zahl
gewählt
        MessageDlg('Bitte geben Sie rechts an, welche Zahl Sie gezeichnet
haben.',mtError,[mbOK],0);
        end
      else begin //in der Kann-Phase...
        ZeichnungAufRezeptoren; //die Rezeptoren werden
aktiviert, //nach dem Durchlauf durch das
Netz //die Ergebnisse angezeigt
        ShowErkannteZahl; //und es wird ein eindeutiges
Ergebnis gesucht
        end;
end;

procedure TForm1.InitLernphase;
var i: integer;
begin
  Lernphase:=true;
  LabelPhase.Caption:='Lern-Phase';
  ButtonPhase.Caption:='-> Kann-Phase';
  ListBoxLernZahl.Visible:=true; //Anzeigen der für die Lern-Phase
  LabelLernZahl.Visible:=true; //benötigten Komponenten im Ergebnis-
Bereich
  for i:=0 to 9 do begin //und Ausblenden der überflüssigen
    ErgebnisAnzeigen[i].Hide; //Komponenten
    LernZaehler[i].Show;
  end;
end;

procedure TForm1.InitKannphase;
var i: integer;
begin
  Lernphase:=false;
  LabelPhase.Caption:='Kann-Phase';
  ButtonPhase.Caption:='-> Lern-Phase';
  ListBoxLernZahl.Visible:=false; //Anzeigen der für die Lern-Phase
  LabelLernZahl.Visible:=false; //benötigten Komponenten im Ergebnis-
Bereich
  for i:=0 to 9 do begin //und Ausblenden der überflüssigen
    ErgebnisAnzeigen[i].Show; //Komponenten
    LernZaehler[i].Hide;
  end;
end;

procedure TForm1.NeuronenReset;
var x,y,i: integer;
begin
  for i:=0 to 9 do begin
    Ganglienzellen[i].Reset; //alle Ganglienzellen zurücksetzen,
    ErregungsLabels[i].Reset; //die Labels, die deren Erregungsstärken
anzeigen, //natürlich auch
  end;

  for x:=0 to 9 do
    for y:=0 to 9 do
      Rezeptoren[x,y].Reset(Ganglienzellen); //alle Rezeptoren zurücksetzen

```

```

end;

procedure TForm1.ZeichnungAufRezeptoren;
var x,y,i,j,px,py: integer;
    SchwarzerPixel: boolean;
begin
    //Hier werden die schwarzen Pixel der Zeichenfläche zu den Aktivierungen der
    Rezeptoren
    //zusammengefasst. Dabei wird die Zeichenfläche in Felder eingeteilt, von dem jedes
    //einem Rezeptor entspricht. Sobald auch nur ein Pixel in einem Feld schwarz ist,
    //wird der zugehörige Rezeptor aktiviert. Dadurch entsteht ein gröberes Abbild der
    //gezeichneten Zahl auf dem Rezeptorenraster.

    for x:=0 to 9 do //durchläuft alle Rezeptoren
        for y:=0 to 9 do begin
            SchwarzerPixel:=false;

            for i:=0 to 9 do begin //durchläuft alle Pixel der Zeichenfläche,
                for j:=0 to 9 do begin //für die dieser Rezeptor zuständig ist
                    px:=x*10+i; //Pixelkoordinaten berechnen
                    py:=y*10+j;
                    if ImageZeichnung.Canvas.Pixels[px,py]=clBlack
                        then SchwarzerPixel:=true;
                    end;
                    if SchwarzerPixel then break; //wurde der erste schwarze Pixel gefunden,
                                                    //wird in diesem Feld gar nicht mehr
weitergesucht
                end;

                if SchwarzerPixel then
                    if Lernphase //Aktivierung des betreffenden Rezeptors,
                                //zum Bahnen in der Lern-Phase
                                //und zum Weiterleiten in der Kann-Phase
                    then Rezeptoren[x,y].ErregenUndBahnen(Ganglienzellen, LernZahl)
                    else Rezeptoren[x,y].ErregenUndWeiterleiten(Ganglienzellen);
                end;
            end;
        end;

    procedure TForm1.ShowOutput;
    var i, Staerke, Gesamtstaerke: integer;
    begin
        //Gesamterregungsstärke aller Ganglienzellen zur Berechnung der Prozentwerte
        ermitteln
        Gesamtstaerke:=0;
        for i:=0 to 9 do begin
            Staerke:=Ganglienzellen[i].GetErregung;
            inc(Gesamtstaerke,Staerke);
        end;

        for i:=0 to 9 do begin
            Staerke:=Ganglienzellen[i].GetErregung;
            ErregungsLabels[i].ShowErregung(Staerke);
            ErgebnisAnzeigen[i].ShowErgebnis(Staerke,Gesamtstaerke); //berechnet Prozentwert
                                                                    //und zeigt ihn an
        end;
    end;

    procedure TForm1.ShowErkannteZahl;
    var MaxWert, i, NaechsterWert, ErkannteZahl: integer;
        Mehrdeutig: boolean;
    begin
        MaxWert:=0;
        ErkannteZahl:=0;
        Mehrdeutig:=true;

        for i:=0 to 9 do begin //durchläuft alle zu erkennenden
            Zahlen
                NaechsterWert:=Ganglienzellen[i].GetErregung;
                if NaechsterWert=MaxWert //wenn die höchste Erregung
                    mehrfach auftritt, //gibt es kein eindeutiges
                    then Mehrdeutig:=true //gibt es kein eindeutiges
            Ergebnis
        end;
    end;

```

```

        else if NaechsterWert>MaxWert //wurde eine höhere Erregung
gefunden,
        then begin //so ist diese jetzt die höchste
            MaxWert:=NaechsterWert;
            ErkannteZahl:=i; //i ist jetzt die Zahl mit der
höchsten Erregung
            Mehrdeutig:=false;
            end;
        end;

        if Mehrdeutig
        then showmessage('Es konnte keine Zahl eindeutig erkannt werden.')
        else showmessage('Es wurde die Zahl '+inttostr(ErkannteZahl)+' erkannt.');
```

```

end;

procedure TForm1.ButtonPhaseClick(Sender: TObject);
begin
    if Lernphase
        then InitKannphase
        else InitLernphase;
end;

procedure TForm1.BitBtnZeichnungLoeschenClick(Sender: TObject);
begin
    ImageZeichnung.Canvas.Rectangle(-1,-1,Width,Height); //Löschen durch weißes
Rechteck
end;

procedure TForm1.BitBtnSpeichernClick(Sender: TObject);
begin
    Speichern;
end;

procedure TForm1.BitBtnLadenClick(Sender: TObject);
begin
    if Laden //wenn der Dialog nicht abgebrochen wurde,
        then InitKannphase; //wird gleich in die Kann-Phase gewechselt
end;

procedure TForm1.Speichern;
var Schleifenabbruch:boolean;
begin
    repeat
        Schleifenabbruch:=True;
        if SaveDialog1.Execute //Speichern unter-Dialog anzeigen
        then
            if FileExists(SaveDialog1.FileName) //wenn die gewählte Datei schon
existiert...
            then
                begin
                    if MessageDlg('Vorhandene Datei ersetzen?',mtWarning,[mbYes,mbNo],0)=mrYes
                    //wenn die Frage mit Ja beantwortet wurde...
                    then SpeichereDort(SaveDialog1.FileName)
                    else Schleifenabbruch:=False;
                end
            else SpeichereDort(SaveDialog1.FileName);
        until Schleifenabbruch=True;
end;

procedure TForm1.SpeichereDort(Dateiname: string);
var f: textfile;
    s: string;
    x,y,i,z: integer;
begin
    assignfile(f,Dateiname);
    rewrite(f);

    //Speichern der Lernzähler-Werte
    for i:=0 to 9 do begin
        z:=LernZaehler[i].GetZaehlerwert;
        s:=inttostr(z);
        writeln(f,s);
    end;
end;

```

```

//Speichern der Gewichte
for x:=0 to 9 do
  for y:=0 to 9 do
    for i:=0 to 9 do begin
      z:=Rezeptoren[x,y].Synapsen[i].GetGewichtung;
      s:=inttostr(z);
      writeln(f,s);
    end;
  end;
end;
closefile(f);
end;

function TForm1.Laden: boolean;
begin
  if OpenDialog1.Execute
  then begin
    LadeDas(OpenDialog1.FileName);
    result:=true;
  end
  else result:=false;
end;

procedure TForm1.LadeDas(Dateiname: string);
var f: textfile;
    s: string;
    x,y,i,z: integer;
begin
  assignfile(f,Dateiname);
  reset(f);

  //Einlesen der Lernzähler-Werte
  for i:=0 to 9 do begin
    readln(f,s);
    z:=strtoint(s);
    LernZaehler[i].SetZaehlerwert(z);
  end;

  //Einlesen der Gewichte
  for x:=0 to 9 do
    for y:=0 to 9 do
      for i:=0 to 9 do begin
        readln(f,s);
        z:=strtoint(s);
        Rezeptoren[x,y].Synapsen[i].SetGewichtung(z);
      end;
    end;
  end;
  closefile(f);
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var x,y,i: integer;
begin
  //alle Speicherbereiche freigeben

  for x:=0 to 9 do
    for y:=0 to 9 do
      Rezeptoren[x,y].Free;

  for i:=0 to 9 do begin
    Ganglienzellen[i].Free;
    ErregungsLabels[i].Free;
    ErgebnisAnzeigen[i].Free;
    LernZaehler[i].Free;
  end;
end;
end.

```

Datei Neuronen.pas

```
unit Neuronen;
//enthält die am biologischen Vorbild orientierten Bestandteile des
//neuronalen Netzes

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls;

type

TNeuron = class
  Shape: TShape;           //jedes Neuron wird auf dem Formular
                          //durch ein quadratisches Shape dargestellt,
                          //das mit seiner Farbe eine Erregung anzeigt
  Formular: TForm;
  Erregung: integer;      //gibt an, wie stark das Neuron erregt ist
end;

TGanglienzelle = class(TNeuron)
  Index: integer;         //Zahl von 0-9, kennzeichnet jede Ganglienzelle
                          //eindeutig, gibt gleichzeitig an, für welche Zahl
                          //sie beim Erkennungsvorgang zuständig ist

  constructor Create(aFormular: TForm; aIndex: integer);
  destructor Destroy; override;

  procedure Erregen(Staerke: integer);
  //erhöht die Erregungsstärke um den übergebenen Wert
  procedure Reset;
  //setzt Erregungsstärke auf Null zurück, setzt Darstellung des Shapes zurück
  function GetErregung: integer;
  //gibt die Erregungsstärke zurück (Observator zur Datenkapselung)
end;

TGanglienzellen = array[0..9] of TGanglienzelle;
//eigener Datentyp, damit den Methoden von TSynapse
//auch ein statisches Array übergeben werden kann

TSynapse = class
  Gewichtung: integer;    //gibt die Stärke der Synapse an
                          //Gewichtung<0 bedeutet hemmende Synapse
  Erregt: boolean;        //Boolean-Wert reicht aus,
                          //Erregungsstärke einer Synapse unwichtig
  Ganglienindex: integer; //Index der zugeordneten Ganglienzelle

  constructor Create(aGanglienindex: integer);
  destructor Destroy; override;

  procedure Erregen(Ganglienzellen: TGanglienzellen);
  //erregt die Synapse
  procedure Reset(Ganglienzellen: TGanglienzellen);
  //hebt die Erregung auf,
  //leitet den Reset-Befehl an die zugehörige Ganglienzelle weiter
  procedure Bahnen(Wert: integer);
  //ändert die Gewichtung durch Addition um den übergebenen Wert
  procedure SetGewichtung(nGewichtung: integer);
  //setzt die Gewichtung auf den übergebenen Wert
  //(Transformator zur Datenkapselung)
  function GetGewichtung: integer;
  //gibt die Gewichtung zurück (Observator zur Datenkapselung)
end;

TRezeptor = class(TNeuron)
  x,y: integer;           //Position im Array und Position des
                          //Shapes auf dem Formular
  Synapsen: array[0..9] of TSynapse; //von diesem Rezeptor ausgehende Synapsen
  //eine für jede Ganglienzelle
```

```

    constructor Create(aFormular: TForm; ax,ay: integer);
    destructor Destroy; override;
    procedure ErregenUndWeiterleiten(Ganglienzellen: TGanglienzellen);
    //erregt den Rezeptor, leitet die Erregung über die Synapsen weiter
    //Aufruf in der Kann-Phase
    procedure ErregenUndBahnen(Ganglienzellen: TGanglienzellen; LernZahl: integer);
    //erregt den Rezeptor, bahnt die Synapsen anhand der übergebenen
    //Ergebnis-Vorgabe (LernZahl)
    //Aufruf in der Lern-Phase
    procedure Reset(Ganglienzellen: TGanglienzellen);
    //setzt Erregung und Darstellung des Shapes zurück,
    //leitet den Reset-Befehl an alle zugehörigen Synapsen weiter
end;

implementation

constructor TRezeptor.Create(aFormular: TForm; ax,ay: integer);
var g,i: integer;
begin
    inherited Create;

    //Belegung der Eigenschaften
    Erregung:=0;
    x:=ax;
    y:=ay;
    Formular:=aFormular;

    //Erzeugen des Shapes zur Darstellung auf dem Formular
    Shape:=TShape.Create(Formular);
    g:=10; //Größe des Shapes
    with Shape do begin
        Parent:= Formular;
        Left:= 200+x*g; //Berechnung der Position aus Größe
        Top := 100+y*g; //und Reihen-/Spaltennummer
        Width:= g+1; //g+1, weil sonst die Ränder nebeneinander doppelt wären
        Height:= g+1;
    end;

    for i:=0 to 9 do
        Synapsen[i]:=TSynapse.Create(i); //Erzeugen der vom Rezeptor ausgehenden Synapsen
    end;

destructor TRezeptor.Destroy;
var i: integer;
begin
    Shape.Free; //Speicherbereiche aller Unterobjekte freigeben
    for i:=0 to 9 do
        Synapsen[i].Free;
    inherited Destroy;
end;

procedure TRezeptor.ErregenUndWeiterleiten(Ganglienzellen: TGanglienzellen);
var i: integer;
begin
    Erregung:=1; //die Erregung wird einfach auf 1 gesetzt,
    //weil die Stärke beim Rezeptor keine Rolle spielt
    Shape.Brush.Color:=clBlack; //Erregung am Shape durch Schwarzfärben anzeigen
    for i:=0 to 9 do
        Synapsen[i].Erregen(Ganglienzellen); //Erregung weiterleiten
    end;

procedure TRezeptor.ErregenUndBahnen(Ganglienzellen: TGanglienzellen; LernZahl:
integer);
var i: integer;
begin
    Erregung:=1; //die Erregung wird einfach auf 1 gesetzt,
    //weil die Stärke beim Rezeptor keine Rolle spielt
    Shape.Brush.Color:=clBlack; //Erregung am Shape durch Schwarzfärben anzeigen
    for i:=0 to 9 do
        if i=LernZahl //wenn die jeweilige Synapse zu der Ganglienzelle
            then Synapsen[i].Bahnen(1) //führt, die dem vorgegebenen Ergebnis (LernZahl)
            else Synapsen[i].Bahnen(-1); //zugeordnet ist, dann Gewichtung erhöhen,

```

```

//sonst verringern
end;

procedure TRezeptor.Reset(Ganglienzellen: TGanglienzellen);
var i: integer;
begin
  Erregung:=0; //Erregung zurücksetzen,
  Shape.Brush.Color:=clWhite; //Darstellung des Shapes entsprechend
  anpassen
  for i:=0 to 9 do
    Synapsen[i].Reset(Ganglienzellen); //Reset-Befehl an Synapsen weiterleiten
  end;

constructor TSynapse.Create(aGanglienindex: integer);
begin
  inherited Create;
  //Belegen der Eigenschaften
  Gewichtung:=0;
  Ganglienindex:=aGanglienindex;
end;

destructor TSynapse.Destroy;
begin
  inherited Destroy;
end;

procedure TSynapse.Erregen(Ganglienzellen: TGanglienzellen);
begin
  Erregt:=true;
  //die zugeordnete Ganglienzelle um den Gewichtungswert erregen
  //wenn Gewichtung<0, wird die Ganglienzelle gehemmt
  Ganglienzellen[Ganglienindex].Erregen(Gewichtung);
end;

procedure TSynapse.Reset(Ganglienzellen: TGanglienzellen);
begin
  Erregt:=false;
  Ganglienzellen[Ganglienindex].Reset; //Reset-Befehl an Ganglienzelle weiterleiten
end;

procedure TSynapse.Bahnen(Wert: integer);
begin
  inc(Gewichtung,Wert); //Gewichtung um den übergebenen Wert erhöhen
end;

procedure TSynapse.SetGewichtung(nGewichtung: integer);
begin
  Gewichtung:=nGewichtung; //Gewichtung auf den übergebenen Wert setzen
end;

function TSynapse.GetGewichtung: integer;
begin
  result:=Gewichtung; //Gewichtungswert zurückgeben
end;

constructor TGanglienzelle.Create(aFormular: TForm; aIndex: integer);
var g: integer;
begin
  inherited Create;

  //Belegung der Eigenschaften
  Erregung:=0;
  Index:=aIndex;
  Formular:=aFormular;

  //Erzeugen des Shapes zur Darstellung auf dem Formular
  Shape:=TShape.Create(Formular);
  g:=15; //Größe des Shapes
  with Shape do begin
    Parent:= Formular;
    Left:= 350;
    Top := 100+Index*g; //Berechnung der Position aus Größe und Indexnummer der Zelle
    Width:= g+1; //g+1, weil sonst die Ränder nebeneinander doppelt wären
  end;
end;

```

```

    Height:= g+1;
end;

end;

destructor TGanglienzelle.Destroy;
begin
    Shape.Free;           //Speicherbereich für Unterobjekt freigeben
    inherited Destroy;
end;

procedure TGanglienzelle.Erregen(Staerke: integer);
begin
    inc(Erregung,Staerke);           //Erregungsstärke um übergebene Stärke
    erhöhen                          //wenn Staerke<0, wird die Erregung
                                      //verringert (Hemmung)

    if Erregung<=0
    then begin                        //wenn nicht erregt,
        Shape.Brush.Color:=clWhite; //Shape weiß färben
        Erregung:=0;                 //sorgt dafür, dass es keine negativen
                                      //Erregungswerte gibt
    end
    else if Erregung>0
    then Shape.Brush.Color:=clRed;   //wenn erregt (Stärke egal),
                                      //Shape rot färben
end;

procedure TGanglienzelle.Reset;
begin
    Erregung:=0;                    //Erregung zurücksetzen,
    Shape.Brush.Color:=clWhite;     //Darstellung des Shapes entsprechend
    anpassen
end;

function TGanglienzelle.GetErregung: integer;
begin
    result:=Erregung;               //Erregungsstärke zurückgeben
end;

end.

```

Datei NeuroDisplay.pas

```
unit NeuroDisplay;
//enthält Hilfsmittel zur Anzeige zusätzlicher Werte auf dem Formular

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls;

type

  TErregungsLabel = class
    //zeigt die Erregungsstärke einer Ganglienzelle mit einem Label an
    Formular: TForm;
    Labelfeld: TLabel;          //Labelfeld genannt, weil Label ein reserviertes Wort
  ist

    constructor Create(aFormular: TForm; i: integer);
    destructor Destroy;

    procedure ShowErregung(Staerke: integer);
    //ändert die angezeigte Zahl
    procedure Reset;
    //setzt die angezeigte Zahl auf Null zurück
  end;

  TErgebnisAnzeige = class
    //berechnet aus den Erregungsstärken die Wahrscheinlichkeit für eine einzelne Zahl
    //und stellt diese als Prozentzahl und grafisch (Balken) dar
    Formular: TForm;
    LabelZahl, LabelProzent: TLabel;
    HGBalken, Balken: TShape;  //HGBalken ist der Balken (weiß),
                                //der immer in voller Breite hinter dem
                                //roten (Ergebnis-)Balken angezeigt wird;
                                //erweckt den Eindruck, als ob er je nach
                                //Ergebnis mit rot "gefüllt" sei

    constructor Create(aFormular: TForm; i: integer);
    destructor Destroy;

    procedure Show;              //ändern die visible-Eigenschaften
    procedure Hide;             //der Unterobjekte
    procedure ShowErgebnis(Staerke, Gesamtstaerke: integer);
    //Berechnet die Wahrscheinlichkeit in Prozent und zeigt sie an
    procedure Reset;
    //setzt alle Anzeigen zurück
  end;

  TLernZaehler = class
    //verwendet um die Lerndurchläufe für eine einzelne Zahl zu zählen
    Formular: TForm;
    LabelZaehler: TLabel;
    Zaehlerwert, LernZahl: integer; //LernZahl ist die Zahl, für die
                                      //der Zähler die Lerndurchläufe zählt

    constructor Create(aFormular: TForm; i: integer);
    destructor Destroy;

    procedure Show;              //ändern die visible-Eigenschaften
    procedure Hide;             //der Unterobjekte
    procedure Hochzaehlen;
    //zählt den Zähler um eins hoch
    procedure SetZaehlerwert(nZaehlerwert: integer);
    //setzt den Zähler auf den übergebenen Wert (Transformator zur Datenkapselung)
    function GetZaehlerwert: integer;
    //liefert den Wert des Zählers zurück (Observator zur Datenkapselung)
    procedure Reset;
    //setzt den Zähler auf Null zurück
  end;
```

```

implementation

constructor TErregungsLabel.Create(aFormular: TForm; i: integer);
var h: integer;
begin
    Formular:=aFormular;

    Labelfeld:=TLabel.Create(Formular);
    h:=15;
    with Labelfeld do begin
        Parent:= Formular;
        Left:= 400;
        Top := 100+i*h;      //Position aus Höhe berechnen
        Height:= h+1;
        Caption:='0';
    end;
end;

destructor TErregungsLabel.Destroy;
begin
    Labelfeld.Free;
end;

procedure TErregungsLabel.ShowErregung(Staerke: integer);
begin
    Labelfeld.Caption:=inttostr(Staerke);    //übergebene Zahl im Label anzeigen
end;

procedure TErregungsLabel.Reset;
begin
    Labelfeld.Caption:='0';
end;

constructor TErgebnisAnzeige.Create(aFormular: TForm; i: integer);
var h: integer;
begin
    Formular:=aFormular;

    //LabelZahl gibt die Zahl an, für die diese Ergebnisanzeige steht
    //sie wird als i übergeben
    LabelZahl:=TLabel.Create(Formular);
    h:=15;
    with LabelZahl do begin
        Parent:= Formular;
        Left:= 500;
        Top := 100+i*h;      //Position aus Höhe berechnen
        Height:= h+1;
        Caption:=inttostr(i);
    end;

    LabelProzent:=TLabel.Create(Formular);
    h:=15;
    with LabelProzent do begin
        Parent:= Formular;
        Left:= 650;
        Top := 100+i*h;
        Height:= h+1;
        Caption:='0%';
        Alignment:=taRightJustify;    //rechtsbündig, damit Prozentzahlen aller
Ergebnisanzeigen //untereinander
    end;

    HGBalken:=TShape.Create(Formular);
    h:=15;
    with HGBalken do begin
        Parent:= Formular;
        Left:= 520;
        Top := 100+i*h;
        Width:= 100;                //immer in voller Breite im Hintergrund angezeigt
(s.o.)
        Height:= h+1;

```

```

end;

Balken:=TShape.Create(Formular);
h:=15;
with Balken do begin
  Parent:= Formular;
  Left:= 520;
  Top := 100+i*h;
  Width:= 0; //Breite auf 0, zunächst nicht angezeigt
  Height:= h+1;
  Brush.Color:=clRed;
end;
end;

destructor TErgebnisAnzeige.Destroy;
begin
  //Seicherbereiche für Unterobjekte freigeben
  LabelZahl.Free;
  LabelProzent.Free;
  HGBalken.Free;
  Balken.Free;
end;

procedure TErgebnisAnzeige.Show;
begin
  //Ergebnisanzeige auf dem Formular anzeigen
  LabelZahl.Visible:=true;
  LabelProzent.Visible:=true;
  HGBalken.Visible:=true;
  Balken.Visible:=true;
end;

procedure TErgebnisAnzeige.Hide;
begin
  //Ergebnisanzeige auf dem Formular verbergen
  LabelZahl.Visible:=false;
  LabelProzent.Visible:=false;
  HGBalken.Visible:=false;
  Balken.Visible:=false;
end;

procedure TErgebnisAnzeige.ShowErgebnis(Staerke, Gesamtstaerke: integer);
var Prozent: integer;
begin
  //berechnet Prozentzahl für die Wahrscheinlichkeit der zugeordneten Zahl
  //als Anteil der Erregungsstärke der zugeordneten Ganglienzelle
  //an der Geamterregungsstärke aller Ganglienzellen

  if Gesamtstaerke=0 //verhindert Division durch Null
  then Prozent:=0
  else Prozent:=round((100/Gesamtstaerke)*Staerke);

  //Prozentzahl anzeigen und Balken entsprechend langziehen
  LabelProzent.Caption:=inttostr(Prozent)+'%';
  Balken.Width:=Prozent;
end;

procedure TErgebnisAnzeige.Reset;
begin
  LabelProzent.Caption:='0%';
  Balken.Width:=0;
end;

constructor TLernZaehler.Create(aFormular: TForm; i: integer);
var h: integer;
begin
  Formular:=aFormular;

  LernZahl:=i;

  LabelZaehler:=TLabel.Create(Formular);
  h:=15;
  with LabelZaehler do begin

```

```

    Parent:= Formular;
    Left:= 640;
    Top := 100+i*h;
    Height:= h+1;
    //zeigt an, für welche Zahl er zählt, und den Zählerstand (zu Anfang 0)
    Caption:=inttostr(LernZahl)+' : 0x';
end;

end;

destructor TLernZaehler.Destroy;
begin
    LabelZaehler.Free;
end;

procedure TLernZaehler.Show;
begin
    LabelZaehler.Visible:=true;
end;

procedure TLernZaehler.Hide;
begin
    LabelZaehler.Visible:=false;
end;

procedure TLernZaehler.Hochzaehlen;
begin
    //Zählerwert hochzählen und neu anzeigen
    inc(Zaehlerwert);
    LabelZaehler.Caption:=inttostr(LernZahl)+' : '+inttostr(Zaehlerwert)+'x';
end;

procedure TLernZaehler.SetZaehlerwert(nZaehlerwert: integer);
begin
    //Zählerwert auf übergebenen Wert setzen und neu anzeigen
    Zaehlerwert:=nZaehlerwert;
    LabelZaehler.Caption:=inttostr(LernZahl)+' : '+inttostr(Zaehlerwert)+'x';
end;

function TLernZaehler.GetZaehlerwert: integer;
begin
    result:=Zaehlerwert;
end;

procedure TLernZaehler.Reset;
begin
    LabelZaehler.Caption:='0';
end;

end.

```

Anhang B: Hinweise zum Programm

Bedienungshinweise

Auch das Programm unterscheidet zwischen einer Lern-Phase und einer Kann-Phase. Mit einem Button oben im Fenster kann zwischen den beiden Phasen umgeschaltet werden. Je nach gewählter Phase erscheinen im Bereich „Ergebnis“ unterschiedliche Komponenten.

Das zu erkennende Muster wird immer direkt mit der Maus auf die Zeichenfläche gezeichnet. Beim Klick auf den Button „Auswerten“ wird die Zeichnung automatisch in ein Aktivierungsmuster der Rezeptoren umgewandelt.

Zunächst müssen Muster in der Lern-Phase antrainiert werden, indem dem System nach dem Zeichnen über die ListBox im Ergebnisbereich zusätzlich die jeweils korrekte Zahl mitgeteilt wird. Durch Klick auf „Auswerten“ werden die Rezeptoren aktiviert und die entsprechenden Synapsen gebahnt. Wurde vergessen, ein Ergebnis anzugeben, so erscheint eine Fehlermeldung. Die in der Lern-Phase sichtbaren Zähler am rechten Fensterrand zählen mit, wie oft jede einzelne Zahl bereits gelernt wurde.

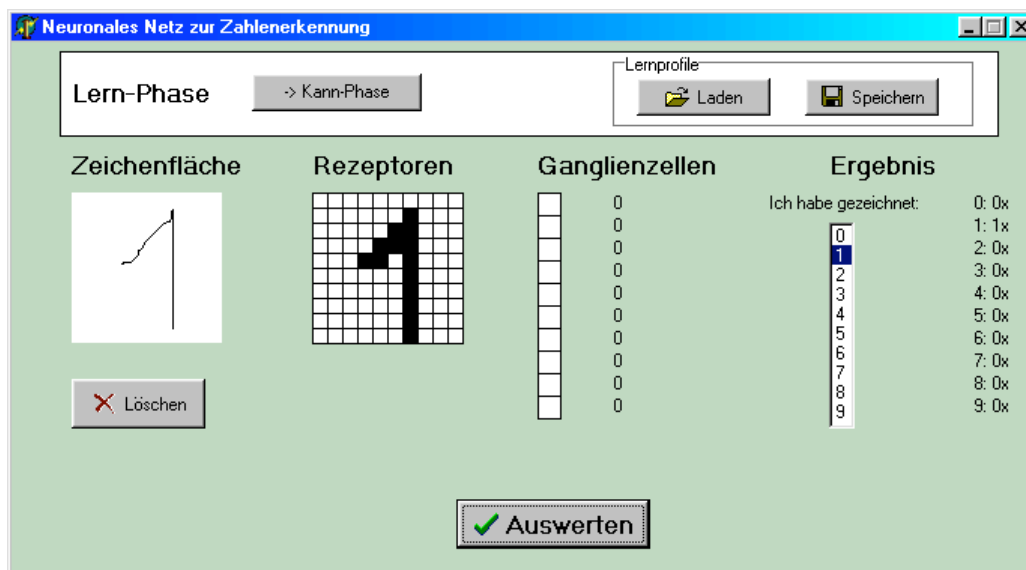


Abbildung 4: Aussehen des Formulars während der Lern-Phase

In der Kann-Phase kann anschließend ein beliebiges gelerntes Muster gezeichnet werden. Wenn es mit „Auswerten“ bestätigt wird, werden wiederum die Rezeptoren aktiviert, welche jetzt über ihre Synapsen die entsprechenden Ganglienzellen erregen. Die Erregung jeder einzelnen Ganglienzelle wird im Bereich „Ganglienzellen“ angezeigt. Dieser direkt ermittelte Wert ist auch stark davon abhängig, welche Größe die Gewichte der Synapsen in der Lern-Phase erreicht haben. Erregte Ganglienzellen sind rot gefärbt, egal, wie stark sie erregt sind. Um brauchbare Auswertungen der Ganglienzellenerregungen zu erhalten, werden für alle Zahlen Wahrscheinlichkeiten ermittelt und unter „Ergebnis“ als Prozentwert und in Balkenform dargestellt. Zusätzlich wird als eindeutiges Ergebnis die mit der größten Wahrscheinlichkeit erkannte Zahl in einem Meldungsfenster ausgegeben. Falls aufgrund gleicher Wahrscheinlichkeitswerte kein eindeutiges Ergebnis gefunden werden kann, erscheint statt dessen ein entsprechender Hinweis.

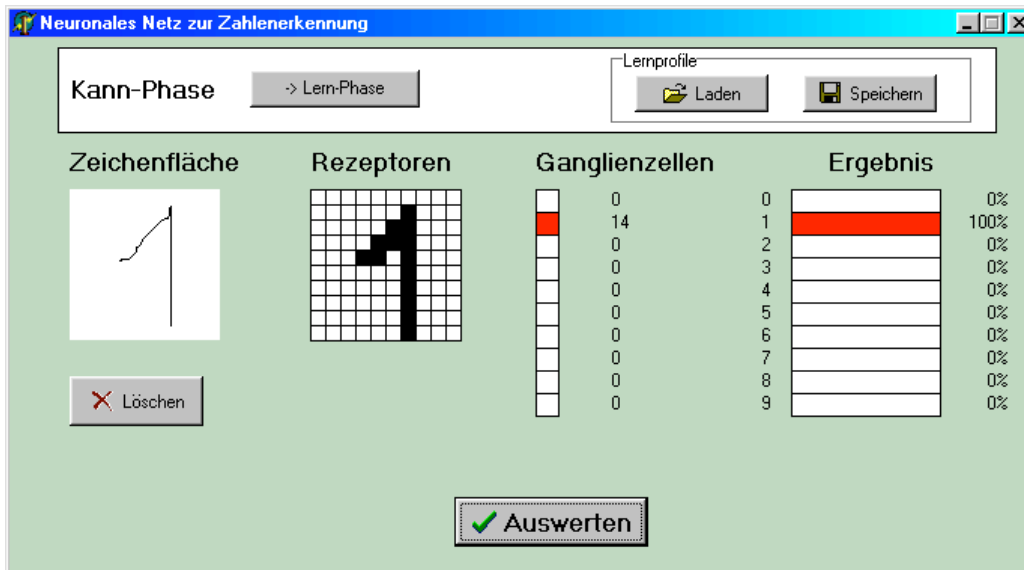


Abbildung 5: Aussehen des Formulars während der Kann-Phase

Alle in der Lern-Phase gesammelten Daten können mit dem Button „Speichern“ zu jedem Zeitpunkt auf die Festplatte gespeichert werden. Es werden dann die Werte der Lerndurchgangs-Zähler und die Gewichtungen sämtlicher Synapsen in eine Textdatei geschrieben. Umgekehrt können diese Werte auch wieder eingelesen werden (Button „Laden“).

Hinweise zum Lernprozess

Der Lernvorgang des implementierten neuronalen Netzes basiert auf dem im theoretischen Teil angesprochenen sehr einfachen Mechanismus, alle Synapsen zwischen aktivierten Rezeptoren und der Ganglienzelle, die dem richtigen Ergebnis zugeordnet ist, um den gleichen Wert (hier 1) zu verstärken, und die Synapsen zwischen aktivierten Rezeptoren und allen anderen Ganglienzellen um diesen Wert abzuschwächen. Für optimale Lernergebnisse sind weitaus bessere Methoden erforderlich. Es folgen zwei Tipps, um auch mit diesem Programm möglichst hohe Erkennungsquoten zu erzielen.

Zunächst sollte darauf geachtet werden, jede Zahl unter Beachtung der Lerndurchgangs-Zähler möglichst gleich oft zu trainieren, ansonsten würden öfter gelernte Zahlen beim Erkennungsvorgang bevorzugt werden. Um dieses Problem zu verhindern, kann man als mögliche Verbesserung des Programms den Bahnungsvorgang optimieren, sodass die Gewichtungen nicht um einen konstanten Wert geändert werden, sondern um einen aus dem bisherigen Gewichtungswert oder auch weiteren Faktoren berechneten Wert.

Weiterhin ist zu beachten, dass die Zuverlässigkeit der Erkennung rapide absinkt, wenn man das gleiche Zeichen an eine andere Stelle auf die Zeichenfläche zeichnet, als man es in der Lern-Phase getan hat, denn dann werden ganz andere Rezeptoren aktiviert als vorher. Abhilfe könnte hier auch eine zusätzliche Prozedur schaffen, die weiße Ränder von dem Bild auf der Zeichenfläche abschneidet, bevor die Rezeptoren aktiviert werden. Hierdurch könnte die Verschiebung von Zeichen, die an einer anderen als der gelernten Position gezeichnet wurden, ausgeglichen werden.

In dem hier verwendeten Modell eines neuronalen Netzes kann das „Vorzeichen“ eines Bahnungsvorgangs bestimmen, ob sich eine Synapse zu einer erregenden oder hemmenden Synapse entwickelt: Eine negative Gewichtung entspricht einem hemmenden Verhalten,

weil eine Synapse in diesem Fall die Erregung der Ganglienzelle vermindert. Ohne hemmende Synapsen würde das Netz weniger zuverlässig funktionieren, was sich auch durch Ausprobieren zeigt. In der Natur jedoch ist der Zusammenhang zwischen erregenden und hemmenden Synapsen in einem neuronalen Netz wahrscheinlich viel komplizierter.

Literatur- und Quellenverzeichnis

Ewert, Prof. Dr. Jörg-Peter: Nerven- und Sinnesphysiologie. Reihe: Biologie Oberstufe. Hg.: Joachim Knoll. Westermann Schulbuchverlag, Braunschweig 1990

Hasebrook, Joachim: Neuronale Netzwerke. In: Microsoft® Encarta® 99 Enzyklopädie. © 1993-1998 Microsoft Corporation.

Meyers Lexikonredaktion (Hg./Bearb.): Schülerduden Informatik. Ein Sachlexikon für die Schule. Wiss. Bearb.: Prof. Dr. Volker Claus und Prof. Dr. Andreas Schwill. Dudenverlag, Mannheim 1997

Miram, Wolfgang: Informationsverarbeitung. Reizphysiologie – Sinnesphysiologie – Neurophysiologie – Kybernetik. Reihe: Materialien für die Sekundarstufe II, Biologie. Schroedel Verlag, Hannover 1978

Miram, Wolfgang / Scharf, Karl-Heinz (Hg.): Biologie heute SII. Ein Lehr- und Arbeitsbuch. Schroedel Verlag, Hannover 1997

Folgende Online-Quelle wurde als Hilfestellung zur Programmierung der erbenden Klassen benutzt:

Vererbung und Polymorphie – Delphi-Source.de:
<http://www.delphi-source.de/grundlagen/sprache/oo3.php>, 11.03.2004

Erklärung

Hiermit versichere ich, dass ich die Facharbeit selbstständig verfasst habe, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen sind, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.
