Modellierungswoche

Mathematik-Informatik

5. – 11. Februar 2012

JUFA Pöllau

Clemens Andritsch

Philipp Gaulhofer

Fabian Peter Hammerle

Sebastian Kollegger

Michael Pucher

Matthias Stocker

betreut von

Mag. Stefan Fürtinger

Projekt:

Bildverarbeitung mit neuronalen Netzen

Inhaltsverzeichnis

Bildverarbeitung mit neuronalen Netzen

Einführung

Für einen Menschen ist es einfach, eine Katze von einem Hund zu unterscheiden, doch mathematische Aufgaben zu lösen erweist sich als deutlich schwieriger. Das exakte Gegenteil dazu ist der Computer:
Er kann problemlos schwierige mathematische Probleme lösen, doch wenn es um das Thema Bilderkennung geht, tut sich ein Computer genauso schwer wie ein Mensch, der mathematische Probleme lösen muss. Jedoch ist es für uns Menschen möglich, mathematische Problemstellungen und ihren Lösungsweg zu erlernen und zu trainieren. Warum sollte das nicht auch ein Computer können?
Das Ziel unseres Projektes war es also, auch einen Computer dazu zu bringen zu erlernen wie man Bilder unterscheiden kann. Dazu beschäftigten wir uns mit neuronalen Netzen, die es auch in unserem Gehirn gibt. Dabei stießen wir auf die sogenannte McCulloch-Pitts Zelle.

Die McCulloch-Pitts Zelle:

Die McCulloch-Pitts Zelle, auch Schwellwertgatter genannt, ist das erste Neuronenmodell für künstliche Intelligenz und wurde im Jahr 1943 von Warren McCulloch und Walter Pitts entwickelt. Das Modell soll reale Vorgänge in einem einfachen System modellieren und vereinfachen.

Das Netzwerk in dem die McCulloch-Pitts Zellen angewandt werden hat drei Ebenen: Die Inputebene (mit Inputwerten von x_1 bis x_D), den Hiddenlayer (mit Aktivitätslevel z_1 bis z_M) und die Outputebene (mit Aktivitätslevel y_1 bis y_K). Die Informationen erhält das System über die „Inputebene“. Dabei hat jedes Neuron dieser Ebene einen bestimmten Inputwert. Die Neuronen der Inputebene senden an die Neuronen des Hiddenlayers einen Wert, der abhängig von ihrer Wichtigkeit ist. Dafür hat man für Verbindung zwischen einem Neuron der Inputebene und einem Neuron im Hiddenlayer einen eigenen Faktor, der „Gewicht“ genannt wird, und die Wichtigkeit des Neurons der Inputebene auf das des Hiddenlayers steuert ($w_{1\ 1}^{(1)}$ bis $w_{MD}^{(1)}$, dabei steht die erste Zahl für das Ziel im Hiddenlayer, die zweite Zahl für das Ausgangsneuron in der Inputebene und die Zahl in der Klammer für die Schicht, in der sich das abspielt).

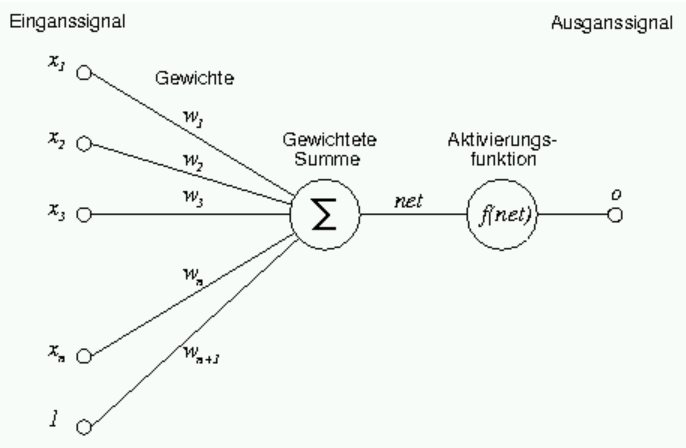


Abbildung 1

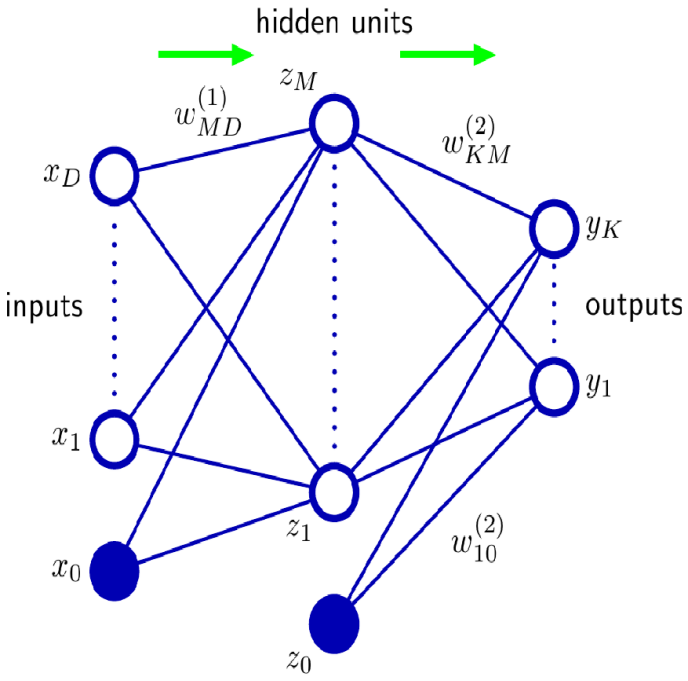


Abbildung 2

Ab einem gewissen Wert, der bei jedem Neuron des Hiddenlayers unterschiedlich ist, schickt das Neuron des Hiddenlayers Impulse an die Outputebene. Der Wert, der dabei überschritten werden muss, nennt man den Schwellwert (meist als Theta bezeichnet). Um diesen Schwellwert zu beeinflussen gibt es Bias-Units(x_0, z_0), die je Schicht einen bestimmten Wert für jedes Neuron annehmen können.
Von den Neuronen des Hiddenlayers gehen jeweils wieder Verbindungen zu den Neuronen der Outputebene aus. Diese haben wieder Gewichte, die dieses Mal von $w_{11}^{(2)}$ bis $w_{KM}^{(2)}$ gehen. Das Neuron, das in der Outputebene am stärksten angesprochen wurde, bestimmt den Output des Netzwerks.

PyBrain

Um neuronale Netzwerke bauen zu können, verwendeten wir eine auf künstliche Intelligenz spezialisierte Python-Bibliothek namens PyBrain.
Dabei fokussierten wir uns hauptsächlich auf Supervised Learning, darunter versteht man, dass dem Programm vorgegeben wird welches Ergebnis es errechnen soll. Hierfür benötigt man ein Dataset mit den dazugehörenden Targets.

- Datasets

Um ein Dataset zu generieren wandelten wir unsere Bilder in Arrays um, in welche wir die Graustufenwerte jedes Pixels schrieben. Für die Targets wird ein weiteres Array definiert, in dem festgelegt wird, welches Neuron der Outputebene am stärksten angesprochen werden soll. Das Dataset wird in weiterer Folge in ein Trainingsset und ein Testset aufgeteilt.

- Netzwerk

In PyBrain ist es möglich mit wenigen Befehlen die Struktur des Netzwerks zu bestimmen, wobei die Inputebene (Anzahl der Pixel) und die Outputebene (Anzahl der möglichen Targets), bereits vorgegeben sind. Einzig die Anzahl der Hiddenlayer und die Anzahl der Hiddenunits bleiben uns überlassen.
Es wird auch noch zwischen Feedforward Netzwerken und Recurrent Netzwerken

unterschieden. Der Unterschied zwischen den beiden liegt darin, dass bei einem Feedforward Netzwerk alle Informationen nur von einer Schicht zur nächsten gesendet werden. Bei einem Recurrent Netzwerk schicken die einzelnen Neuronen auch ein Feedback an sich oder an ein Neuron aus den vorigen Schichten zurück, und beeinflussen sich damit zusätzlich.

- Trainer

Als nächstes benötigt man einen Trainer, der das Netzwerk trainiert. Dazu verwendeten wir den sogenannten Backpropagation-Trainer. Wichtige Einstellungen, die am Trainer getätigt werden können sind zum Beispiel die Anzahl der Epochen (Anzahl der Trainings), die Learningrate und ob Bias-Units verwendet werden sollen. Je mehr Epochen man benützt, desto bessere Netzwerke werden erstellt (dies gilt aber nicht immer, da sich das Netzwerk manchmal innerhalb eines Trainingslaufs auch verschlechtern kann). Der Trainer verwendet die Ableitung der Übertragungsfunktionen. Damit sucht er nach lokalen Minima. Oft passiert es aber, dass das Netzwerk diese überspringt. Dafür gibt es die Learningrate, die genau das verhindern soll. Da man für jedes Neuron verschiedene Schwellwerte braucht, gibt es die Bias-Units.

- Trainingsvorgang

Das Netzwerk wird nun auf das Trainingsset trainiert. Um es zu trainieren werden die Gewichte im Netzwerk verändert. Zuerst wählt der Trainer für jedes Gewicht einen zufälligen Wert, den er nach jedem Trainingsdurchlauf zu verbessern versucht. Ein wichtiger Begriff hierbei ist das Overfitting. Dies tritt auf, wenn sich das Netzwerk zu sehr auf das Trainingset einstellt und somit das Testset nicht mehr gut erkennt.

- Auswertung

Nachdem die angegebenen Epochen abgehandelt worden sind, wird das Netzwerk gespeichert und mit dem Testset getestet. Mit diesem es das Ziel ist, eine ähnlich hohe Hitrate wie beim Trainingsset zu erreichen. Zur besseren Übersicht, wird der Verlauf der Trainingsdaten in einem Diagramm angezeigt. So kann man sehr leicht und schnell erkennen, ob es sinnvoll ist, mit dem Netzwerk weiter zu arbeiten, oder nicht.

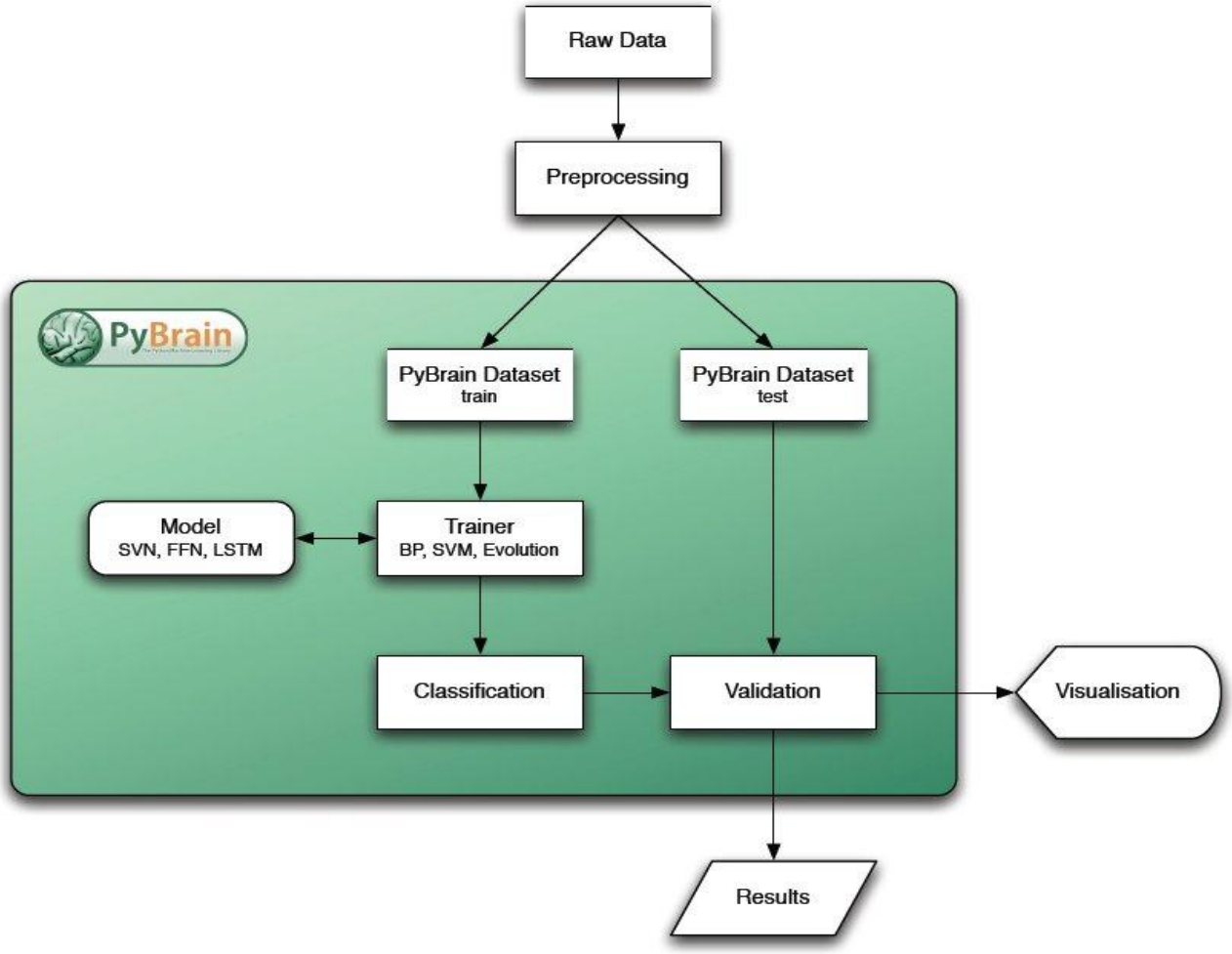


Abbildung 3

Übertragungsfunktionen

Die McCulloch-Pitts Zelle arbeitet mit einer "Heavy-Side"-Funktion. Das bedeutet, wenn das Argument x einen Schwellenwert überschreitet, nimmt H(x) den Wert eins an. Für komplexere Aufgaben braucht man jedoch komplexere Funktionen, um bessere Ergebnisse zu erzielen.

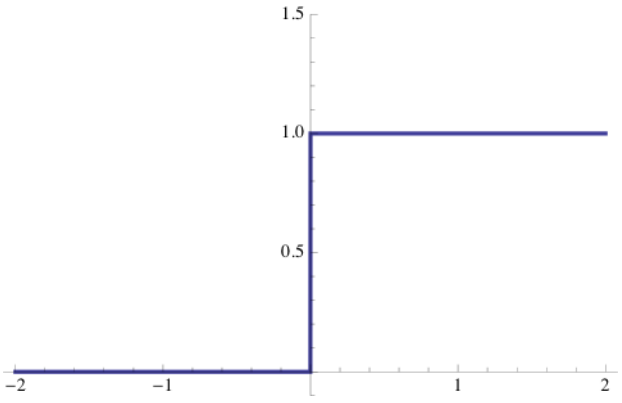


Abbildung 4

Der Trainer von Pybrain verwendet automatisch eine lineare Funktion. Daher gilt: Je größer die Werte, die das Neuron erhält, desto mehr sendet das Neuron an das nächste Neuron. Dies ist für einfache Problemstellungen sinnvoll, doch bei komplizierteren braucht man eine der folgenden Funktionen:

Tangenshyperbolicus-Funktion:

$$\tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

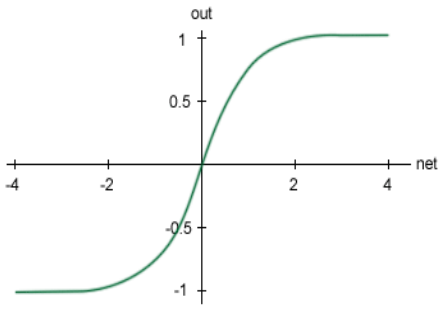


Abbildung 5

Sigmoid-Funktion:

$$\mathrm{sig}(x) = \frac{1}{(1 + e^{-x})}$$

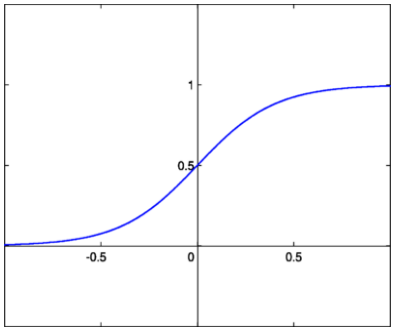


Abbildung 6

Softmax-Funktion:

$$\omega(x) = \frac{e^{a_1}}{\sum_{i=2}^{2} e^{a_i}}$$

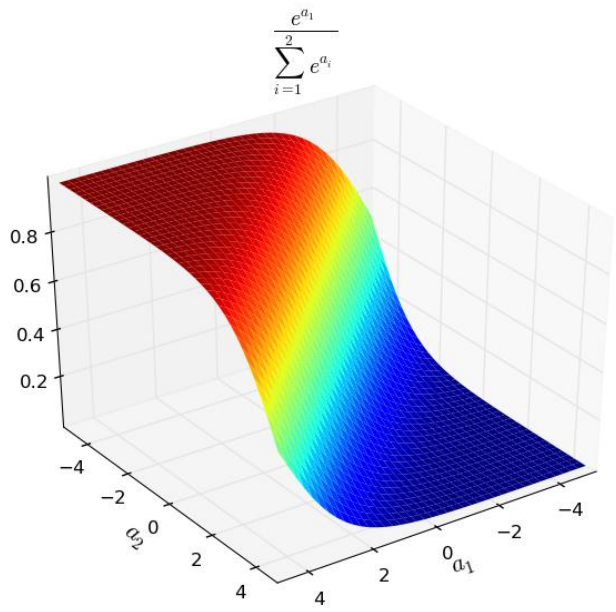


Abbildung 7

Funktionale Darstellung eines Netzwerks:

N_i … i-tes Neuron

x_i … Inputwert von N_i

z_i … Aktivitätslevel von N_i im Hiddenlayer

y_i … Aktivitätslevel von N_i im Outputlayer

a_i … Netinput von N_i im Hiddenlayer

b_i … Netinput von N_i im Outputlayer

h(x) … Übertragungsfunktion des Hiddenlayers

g(x) … Übertragungsfunktion des Outputlayers

$w_{ab}^{(c)}$ … Gewicht, wobei: a … Zielneuron

b … Ausgangsneuron

c … Schicht in der es wirkt

Die Werte der Outputneuronen in Abhängigkeit von den Gewichten und der Inputwerte errechnen sich wie folgt:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \right)$$

$$b_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j$$

$$y_k = g(b_k) = g\left(\sum_{j=1}^{M} w_{kj}^{(2)} z_j \right) = g\left(\sum_{j=1}^{M} w_{kj}^{(2)} * h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \right) \right)$$

Recurrent Netzwerke

Im Gegensatz zu Feedforward Netzwerken, wo die Übertragung immer nur in eine Richtung geschieht, benutzen Recurrent Netze auch Rückkopplungen. Deshalb können auch Verbindungen zur gleichen Neuronenschicht (direkte Rückkopplung) oder zu einer früheren Schicht (indirekte Rückkopplung) des Netzwerks existieren. Die Anwendungsgebiete von Recurrent Netzwerken liegen bei dem Treffen von Prognosen über zukünftige Ereignisse und der Simulation von menschlichen Verhaltensweisen.

Für die Anwendung im Bereich der Bilderkennung sind Recurrent Netze im Gegensatz zu Feedforward Netzen weniger gut geeignet. In Bezug auf Leistung und Ergebnisse haben Feedforward Netze einen entscheidenden Vorteil. Ein Recurrent Netz mit zwei Layern und drei Hidden Units beispielsweise, versagte bei der Erkennung von Ziffern: Die Hitrate lag bei unter 40%. Nach einigen Versuchen konnte allerdings mit einem Recurrent Netz eine akzeptable Leistung von 87% bei der Erkennung von handschriftlichen Ziffern erzielt werden, wenn auch mit gewaltigem Rechenaufwand.

Da mit einfachen Recurrent Netzen keine brauchbaren Ergebnisse produziert werden konnten, wurde ein wesentlich komplexeres Netzwerk mit fünf Layern entwickelt:

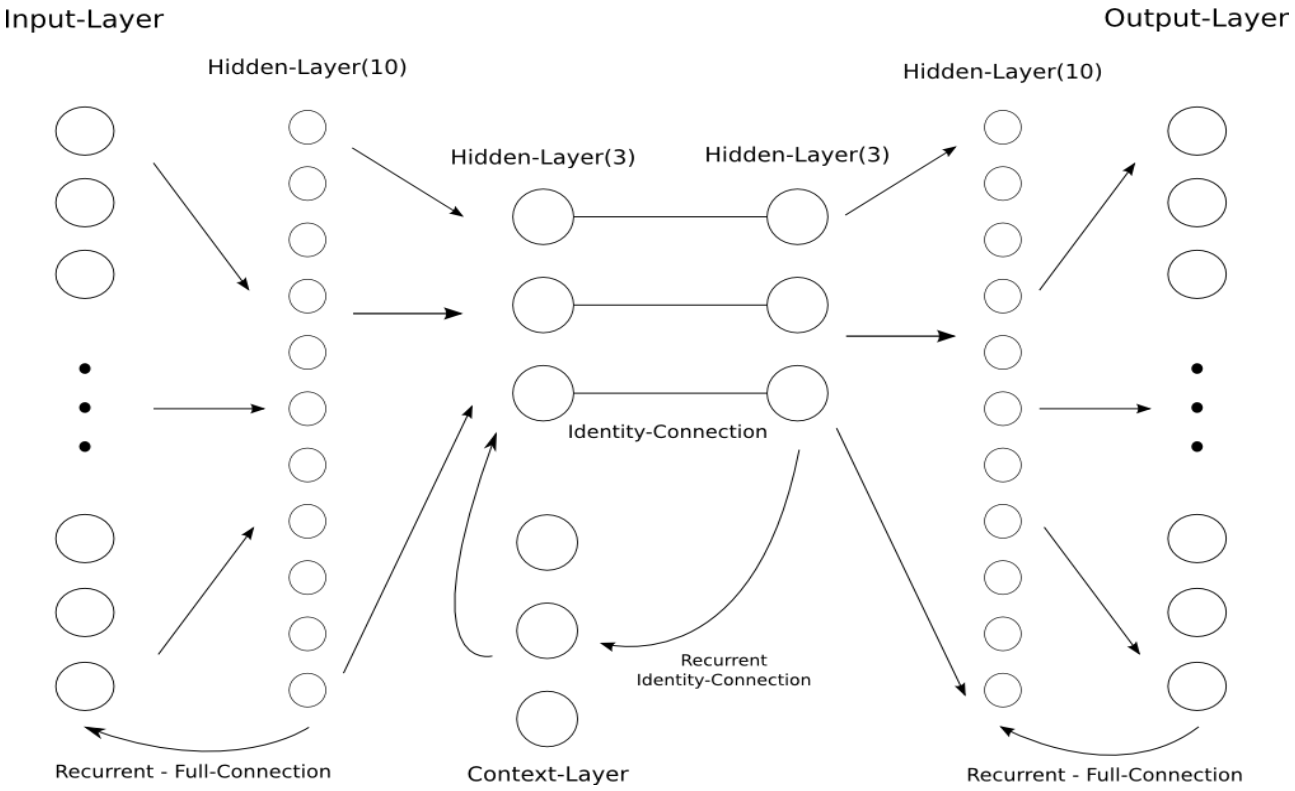


Abbildung 8

Projekt 1: Ziffernerkennung

Unsere erste Problemstellung bestand darin Bilder im Format von 16 x 16 Pixel von handgeschriebenen Ziffern zu erkennen und die Ziffern identifizieren können. Die Bilder wurden in Graustufen umgewandelt um die Erkennung zu erleichtern.

Das Netzwerk, das wir erstellten, bestand aus 256 Inputneuronen, eines pro Pixel und zehn Outputneuronen, eines für jede Ziffer. Die Anzahl der Neuronen im Hiddenlayer sollte sich erst nach und nach ergeben.

Die weiteren Parameter für das Netzwerk wurden erst mit der Zeit nach vielen Testläufen bestimmt. Wir erkannten, dass die Wahl der Übertragungsfunktionen der verschiedenen Schichten essentiell für das Ergebnis war.

Nach vielen Tests kristallisierten sich einige gute Netzwerke heraus und es wurde beschlossen, dass die besten sechs Netze über Nacht austrainiert werden sollten und im Anschluss daran das Beste genommen wird.

Es stellte sich heraus, dass ein FeedForward Netzwerk mit sigmoiden Funktionen, einer Learningrate von 0,1 und neun Neuronen im Hiddenlayer am besten die Bilder zuweisen konnte.

Dieses Netzwerk erreichte eine Hitrate von 92% für das Trainingsset und von 90% für das Testset. Dabei soll jedoch erwähnt werden, dass die Bilder auch für uns nicht immer eindeutig zu erkennen sind. (siehe Abb.: 10)

Zur besseren Veranschaulichung, programmierten wir eine Oberfläche, die das Bild der Ziffer, den Tipp des Computers und das tatsächliche Ergebnis ausgibt. Das Programm zeigt dabei die Ziffer in grün oder rot an, je nachdem ob das Bild richtig oder falsch erkannt wurde.

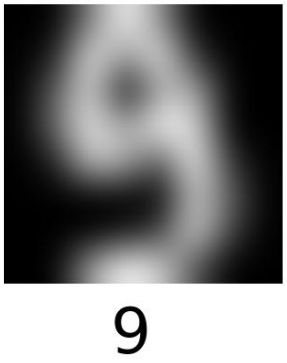


Abbildung 9

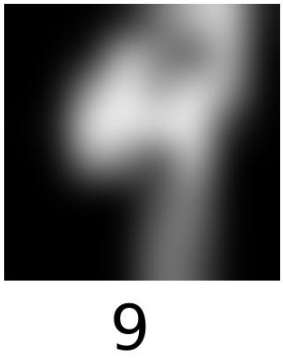


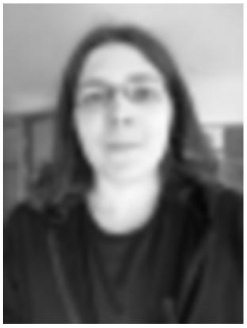
Abbildung 10

Projekt 2: Gesichtserkennung

Nach dem Erfolg beim Erkennen der Ziffern widmeten wir uns einer größeren Herausforderung, der Gesichtserkennung. Zunächst benötigten wir ausreichend Fotos (20.000 Stück), um ein Trainingsset und ein Testset erstellen zu können. Dazu haben wir uns vor einem einheitlichen Hintergrund selbst fotografiert und die Bilder in Graustufen umgewandelt. Für das neuronale Netz haben wir ein Feedforward-Netz mit fünf Hiddenunits, sigmoiden Funktionen, Biasunits und einer Learningrate von 0,1 verwendet. Unsere Fotos hatten eine Größe von 48 x 64 Pixel, daher betrug die Anzahl der Neuronen in der Inputebene 3072. Wir benötigten sechs Neuronen in der Outputebene, da wir sechs Teammitglieder waren (sechs mögliche Antworten). Das Ergebnis sprach für sich selbst: Die Fotos mit einem einheitlichen Hintergrund wurden von unserem Netz zu 96% richtig erkannt.

Danach gingen wir noch einen Schritt weiter: Wir fotografierten uns vor unterschiedlichen Hintergründen mit teils schwierigen Lichtverhältnissen (nochmals mehr als 20.000 Bilder). Es wurden abermals ein Trainingsset und ein Testset erstellt und wir verwendeten dieselben Netzwerkeinstellungen, wie bei den normierten Gesichtern mit dem Ergebnis, dass immer noch circa 90% der Bilder von unserem Netz richtig erkannt wurden.

Nachdem unsere Netzwerke gut funktionierten, interessierte uns noch, welche Hitrate bei vertauschten Datensätzen erreicht werden kann. Wir ließen also das Netzwerk, das auf den einfacheren Datensatz trainiert war, die Bilder mit den ständig wechselnden Hintergründen auswerten und umgekehrt. Dabei bemerkten wir, dass sich das Netzwerk, das den schwierigeren Datensatz zum Trainieren hatte viel besser abschnitt, als das, das mit dem monotonen Hintergrund arbeitete. Ersteres erreichte eine Trefferquote von über 50%, Letzteres eine von circa 35%.



Philipp

Abbildung 11

Matthias

Abbildung 12

Quellen

Abbildung 1:
http://www.iicm.tugraz.at/greif/node10.html

Abbildung 2, 5, 6:
Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Cambridge: Springer

Abbildung 3:
Schaul, T.; Bayer, J.; Wierstra, D.; Sun, Y.; Felder, M.; Sehnke, F.; Rückstieß, T.; Schmidhuber, J. (2010). PyBrain. Journal of Machine Learning Research(S. 743-746)

Abbildung 4:
http://www.cmg.org/measureit/issues/mit62/m_62_15.html