Handschrifterkennung mittels neuronaler Netze mit Backpropagation

Nadja Kurz

Studiengang Angewandte Informatik, Hochschule RheinMain
nadja.kurz@student.hs-rm.de

Abstract

In der folgenden Arbeit wird das Konzept hinter neuronalen Netzen und deren Bestandteile mit besonderem Schwerpunkt auf Backpropagation erläutert. Darauf aufbauend wird das Forschungspaper "Handwritten Digit Recognition with a Back-Propagation Network" von Y. LeCun et al. vorgestellt und die Ergebnisse der Autoren zu Ihrer Arbeit präsentiert. Die Autoren befassen sich mit der Erkennung von handgeschriebenen Ziffern mittels eines neuronalen Netzes, ohne die Eingabedaten vorher aufwändig aufzubereiten. Dafür wurde eine spezielle Architektur entwickelt, die im Verlauf genauer erläutert wird.

I. Machine Lerning-Konzept: Neural Networks with Backpropagation

I.1 Neuronale Netzwerke

Die Idee hinter neuronalen Netzen ist die mathematische Modellierung der menschlichen Gehirnfunktionalität, um über diese ein besseres Verständnis zu erlangen und auf technische Systeme abzubilden. In der Informatik ist ein neuronales Netz ein gewichteter, gerichteter Graph mit mindestens zwei verschiedenen Schichten, der Ein- und der Ausgabeschicht. Diese werden dazu genutzt, um Systeme darauf zu trainieren, mit einer Eingabe eine bestimmte Ausgabe zu erzielen. [1]

I.1.1 Bestandteile eines Netzes

Ein neuronales Netz besteht aus sogenannten Neuronen (= units), den Verarbeitungseinheiten, die untereinander verbunden sind und somit ein komplexes Netz bilden. Ferner sind die Verbindungen zwischen den einzelnen Neuronen unterschiedlich gewichtet und modifizierbar, wodurch das antrainierte Verhalten repräsentiert wird. Die Struktur eines neuronalen Netzes, also die Netzwerk-Topologie, lässt sich dabei am besten als ein gewichteter, gerichteter Graph beschreiben, in dem die einzelnen Neuronen Knoten darstellen und deren Verbindungen untereinander die gewichteten Kanten zwischen den Knoten.

Eine weitere Komponente eines neuronalen Netzes ist eine sogenannte Lernregel. Diese beschreibt ein Verfahren, welches bestimmt, wie die Neuronen "lernen" bzw. auf bestimmte Ergebnisse trainiert werden. Dafür verändert diese in der Regel die Gewichte der Verbindungen zwischen den Neuronen. [1, 2, 3, 4]

I.1.2 Neuron

Die künstlichen Neuronen, die in neuronalen Netzen Anwendung finden, sind dem biologischen Vorbild, welches sich im menschlichen Gehirn wiederfindet, nachempfunden. Die Neuronen des menschlichen Gehirns bestehen aus Dendriten, Axonen und Zellkernen. Die Dendriten empfangen dabei die Reize, welche an den Zellkern weitergeleitet werden, wo die eigentliche Verarbeitung erfolgt und das Ergeb-

nis schließlich über ein Axon an das nächste Neuron weitergeleitet wird. Der Zustand von Neuronen, also deren "Aktivität", wird somit durch die Eingangssignale anderer Neuronen verändert. Dabei wird die Aktivität entweder gesteigert oder reduziert. [5]

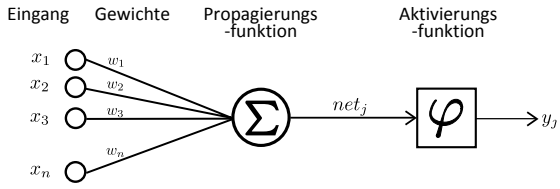

Figure 1: Schematische Darstellung eines künstlichen Neurons auf Basis von Quelle [4]

Die künstlichen Neuronen basieren auf einem ähnlichen Prinzip, welches schematisch in Abbildung 1 dargestellt ist.
Ein künstliches Neuron besitzt, wie sein biologisches Vorbild, einen Aktivierungszustand $y_j \in \mathbb{R}$ und hat in der Regel mehrere gewichtete Verbindungen zu anderen Neuronen. Von diesen erhält das jeweilige Neuron seine Eingangssignale $x_1, ..., x_n \in \mathbb{R}$. Diese werden zusammen mit den Gewichten $w_1, ..., w_n \in \mathbb{R}$ von einer sogenannten Propagierungsfunktion (= Netzwerkeingabefunktion) zu einer Netzeingabe $net_j \in \mathbb{R}$ weiterverarbeitet. Die Netzeingabe net_j wird dann schließlich mittels einer Aktivierungsfunktion $\varphi(net_j)$ auf einen neuen Aktivierungszustand y_j abgebildet. Der neue Aktivierungszustand y_j bildet ein neues Eingangssignal für alle weiteren Neuronen, zu denen eine direkte Verbindung besteht. [2]

I.2 Backpropagation

Die ersten neuronalen Netze hatten in der Regel nur eine Ein- und eine Ausgabeschicht, bei der die Eingabeschicht vollständig mit der Ausgabeschicht verbunden war. Allerdings gibt es verschiedene Anwendungsfälle, in denen zwei Schichten nicht ausreichen, wie z.B. die XOR-Funktion (genauer nachzulesen unter [3]). In solchen Fällen sind mehrere Schichten notwendig. Diese zusätzlichen Schichten, die sich zwischen der Ein- und Ausgabeschicht befinden, werden "hidden layer" genannt, entsprechend die Neuronen dieser Schichten "hidden units". Die Idee hinter Backpropagation ist die Anpassung der Gewichte, sodass eine vorgegebene Fehlerfunktion minimal wird. Dies wird mittels eines Gradientenabstiegsverfahrens erreicht, welches in I.2.5 genauer erläutert wird. Da im Gegensatz zu den Ausgabeneuronen für die hidden units allerdings kein gewünschtes Ergebnis bekannt ist, muss die Veränderung der Gewichte der hidden units anders erfolgen. Hier kommt das Prinzip der Backpropagation (= Rück-Übertragung) ins Spiel: Die Fehlerberechnung, die für die Modifikation der Gewichte notwendig ist, erfolgt von hinten nach vorne, also von den Ausgabeneuronen zu den Eingabeneuronen. [6]

I.2.1 Lernalgorithmus

Im Folgenden wird der Backpropagation-Lernalgorithmus erläutert. Dieser läuft auf einem Eingabe-Datensatz, der für die Trainingsphase zusammengestellt wird und dementsprechend Trainingsset genannt wird. Im Anschluss an die Trainingsphase gibt es noch eine Testphase, in der geprüft wird, ob das Netz etwas gelernt hat, ohne die Gewichte des Netzes zu modifizieren. Die Trainingsphase läuft auf einem von dem Trainingsset unabhängigen Datensatz, dem sogenannten Testset.

Der Lernalgorithmus selbst wird in drei getrennte Phasen unterteilt.

In der erste Phase, dem Forward-Pass, wird aus den Eingaben, ohne Modifikationen der Gewichte, die Ausgabe des neuronalen Netzes bestimmt. In der zweiten Phase, der Fehlerbestimmung, wird für alle Neuronen der Ausgabeschicht ein Fehler mittels einer Fehlerfunktion E berechnet. Dafür werden die im Forward-Pass zuvor bestimmten Ausgaben mit den tatsächlichen Ausgabe-Werten, die von Beginn an bekannt sind, verglichen.

In der dritten Phase, dem Backward-Pass werden die Fehlerterme, von der Ausgabe-Schicht ausgehend hin zur Eingabe-Schicht

propagiert. Dabei werden die Gewichte entsprechend modifiziert. [7, 8]

In den folgenden Abschnitten werden die drei Phasen mathematisch erläutert.

I.2.2 Forward-Pass

Um die Eingabedaten zu verarbeiten, bildet die Propagierungsfunktion (vgl. Grafik 1) in der Regel eine gewichtete Summe der Eingaben für ein Neuron j:

$$net_j := \sum_{i=1} w_{ij} x_i \tag{1}$$

Wobei w_{ij} das Gewicht der Verbindung zwischen Neuron i und Neuron j und x_i die Ausgabe des Neurons i und damit eine Eingabe für Neuron j darstellt.

Die Aktivierungsfunktion entspricht in der Regel einer sigmoid Funktion, welche eine Approximation der Treppenfunktion darstellt, wie auch in Grafik 2 zu erkennen ist. Wichtig ist vor allem, dass die Aktivierungsfunktion eine stetige, differenzierbare Funktion ist, da die Ableitung dieser später noch bei der Gewichtsmodifikation im Backward-Pass notwendig wird. Eine sigmoid Funktion bietet sich dabei an, da ihre Ableitung relativ simpel ist (eine Herleitung der Ableitung findet sich unter der Quelle [4]).

$$\varphi(x) := \frac{1}{1+e^{-x}}$$

$$\varphi'(x) := \varphi(x)(1-\varphi(x))$$

Die Ausgabe y_j ist wie folgt definiert:

$$y_j := \varphi(net_j)$$

Woraus nach Einsetzen von 1 in die Aktivierungsfunktion folgt:

$$y_j = \varphi(net_j) = \frac{1}{1+e^{-net_j}}$$

[3]

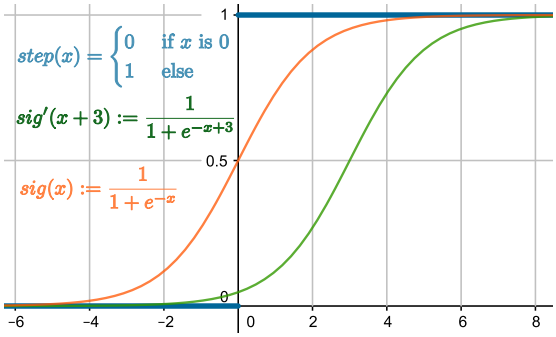

Figure 2: Vergleich einer sigmoid Funktion (mit und ohne Bias) und einer Treppenfunktion.

I.2.3 Bias

Je nach Lernproblem kann es sein, dass eine bestimmte Aktivierungsfunktion nicht optimal ist. Dafür gibt es den sogenannten "Bias". Dieser ist ein zusätzliches Gewicht w_{0j} für jeden layer mit einer konstanten Ausgabe $x_0 = 1$ und ist mit allen anderen Neuronen im Netz verbunden. Durch den Bias lässt sich die Aktivierungsfunktion auf der x-Achse verschieben (vgl Grafik 2). Dies kann für einige Lernprobleme von großem Vorteil sein. In Formel 1 des vorigen Abschnittes wurde der Bias der Einfachheit wegen außen vor gelassen. Wenn man diesen aber nun mit einbeziehen möchte gilt:

$$net_j := w_{0j} x_0 + \sum_{i=1} w_{ij} x_i$$

$$net_j = \sum_{i=0} w_{ij} x_i$$

Der Bias kann somit wie ein weiteres Gewicht behandelt werden, wodurch die Formeln in den folgenden Abschnitten unabhängig davon sind, ob ein Bias genutzt wird oder nicht. [9]

I.2.4 Fehlerbestimmung

Damit ist der Forward-Pass beendet und der Fehler der Output-Schicht wird bestimmt. Dies erfolgt über die Fehlerfunktion E:

$$E := \frac{1}{2} \sum_j (t_j - y_j)^2$$

wobei t_j der erwartete und y_j der tatsächliche Output ist.

I.2.5 Gradientenabstiegsverfahren

Grundlage für den Backward-Pass ist das Gradientenabstiegsverfahren, dessen Ziel die Minimierung einer stetig differenzierbaren Funktion ist.

$$\min_{x \in \mathbb{R}^n} f(x)$$

Dafür wird in einem beliebig gewählten Startpunkt $x_{t,k}$ gestartet und iterativ der nächste Punkt $x_{t+1,k}$ folgendermaßen bestimmt

$$x_{t+1,k} = x_{t,k} - \underbrace{\eta \frac{\partial f}{\partial x_k}}_{\Delta x_k}$$

Wobei t die Iteration angibt und k, um welches x es sich handelt. Weiterhin stellt η die Schrittweite bzw. im Falle der Backpropagation die sog. Lernrate dar.
Sobald $||x_k - x_{k-1}|| < \epsilon$ gilt, wird das Verfahren abgebrochen und man kann davon ausgehen, dass ein lokales Minimum gefunden wurde. [10]

I.2.6 Backward-Pass

Das Gradientenabstiegsverfahren bildet nun die Grundlage für den letzten Schritt: den Backward-Pass, also die Übertragung des Fehlers in die inneren Schichten und die entsprechende Modifikation der Gewichte.

Für die Herleitung der Formeln wird $y_j = \varphi(net_j)$ als Ausgabe des Neurons j einer inneren Schicht und $y_k = \varphi(net_k)$ als Ausgabe des Neurons k der Ausgabeschicht definiert.
Dementsprechend gilt für die Netzwerkeingabe nach Gleichung 1:

$$net_j = \sum_i w_{ij} x_i \tag{2}$$

$$net_k = \sum_j w_{jk} y_j \tag{3}$$

mit x_i als Eingabe für die inneren Schichten und y_j als Eingabe der Ausgabeschicht, sowie w_{ij} als Gewicht zwischen zwei hidden units oder einer hidden unit und einem Neuron der Eingabeschicht und w_{jk} als Gewicht zwischen einer hidden unit und einem Neuron der Ausgabeschicht.

Für die Gewichtsmodifikation gilt nach dem in I.2.5 erläuterten Gradientenabstiegsverfahren

$$\Delta w_{jk} := -\eta \frac{\partial E}{\partial w_{jk}}$$

wobei η die Lernrate ist.

Mittels Anwendung der Kettenregel und Verschiebungs des Minuszeichens ergibt sich

$$\Delta w_{jk} = \eta \, -\frac{\partial E}{\partial net_k} \, \frac{\partial net_k}{\partial w_{jk}} \tag{4}$$

Für den zweiten Teil von 4 wird net_k durch 3 ersetzt und die Ableitungsregeln entsprechend angewendet:

$$\begin{aligned} \frac{\partial net_k}{\partial w_{jk}} &= \frac{\partial}{\partial w_{jk}} \sum_j w_{jk} y_j \\ &= \frac{\partial}{\partial w_{jk}} w_{jk} y_j \\ &= y_j \end{aligned} \tag{5}$$

Für den vorderen Teil von 4 ergibt sich entsprechend wieder durch Anwendung der Kettenregel:

$$-\frac{\partial E}{\partial net_k} = -\frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial net_k} \tag{6}$$

Durch Anwendung der Ableitungsregeln ergibt sich für den ersten Teil von 6

$$\begin{aligned} -\frac{\partial E}{\partial y_k} &= -\frac{\partial}{\partial y_k} \frac{1}{2} \sum_m (t_m - y_m)^2 \\ &= (t_k - y_k) \end{aligned} \tag{7}$$

sowie für den zweiten Teil

$$\begin{aligned} \frac{\partial y_k}{\partial net_k} &= \varphi'(net_k) \\ &= \varphi(net_k)(1 - \varphi(net_k)) \end{aligned} \tag{8}$$

Setzt man nun 7 und 8 in 6 ein, ergibt sich

$$-\frac{\partial E}{\partial net_k} = (t_k - y_k)\varphi(net_k)(1 - \varphi(net_k)) \quad (9)$$

Somit gilt nach Einsetzen von 9 und 5 in 4

$$\Delta w_{jk} = \eta \delta_k y_j \quad (10)$$

mit

$$\delta_k = \varphi(net_k)(1 - \varphi(net_k))(t_k - y_k) \quad (11)$$

Allerdings gilt δ_k nur für den Fall, dass es sich bei dem Neuron k um ein Neuron der Ausgabeschicht handelt. Falls das Neuron Teil eines hidden layers ist, gilt für δ eine etwas andere Formel (eine ausführliche Herleitung findet sich unter der Quelle [4]).
Zusammenfassend gilt für δ:

$$\delta_k = \begin{cases} \varphi'(net_k)(t_k - y_k) & \text{, falls k Neuron der} \\ & \text{Ausgabeschicht ist.} \end{cases}$$

$$\delta_j = \begin{cases} \varphi'(net_j) \sum_k \delta_k w_{jk} & \text{, falls j Neuron eines} \\ & \text{hidden layers ist.} \end{cases}$$

Das neue Gewicht ergibt sich dann entsprechend:

$$w_{jk}^{neu} = w_{jk}^{alt} + \Delta w_{jk}$$

I.2.7 Probleme von Backpropagation und Lernrate

Das Gradientenabstiegsverfahren birgt einige Probleme, die stark mit der Schrittweite λ (hier Lernrate genannt) zusammenhängen [10]. Das ist auch einer der Gründe, warum die Lernrate einen besonders wichtigen Parameter des Backpropagation-Algorithmus darstellt.
Das erste Problem sind lokale Minima. Dadurch, dass der Startpunkt zufällig gewählt wird, ist es möglich, dass das Gradientenabstiegsverfahren in einem lokalen, nicht aber in einem globalen Minimum endet. In dem globalen Minimum landet man unter Umständen, wenn man den Startpunkt in der Nähe von diesem wählt, was selbstverständlich nicht

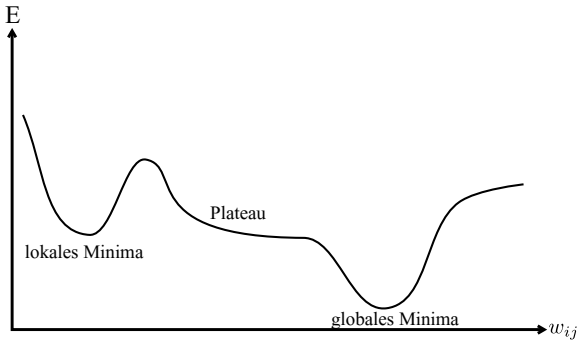

Figure 3: Verlauf der Fehlerfunktion E frei nach [11]

einfach ist. Dies kann man sich auch relativ gut an der Grafik 3 deutlich machen.
Das nächste Problem sind flache Plateaus. In diesen wird der Gradient sehr klein und es kann passieren, dass ein Plateau fälschlicherweise als Minimum interpretiert wird.
Bei einer zu großen Lernrate kann es außerdem dazu kommen, dass das Verfahren entsprechend große Sprünge macht und im Falle eines Tals ständig hin und her springt (=Oszillation) und gar keine Konvergenz erreicht oder sogar über das globale Minimum drüberspringt.
Wählt man dagegen eine zu kleine Lernrate, dauert es sehr viel länger, bis ein Minimum erreicht ist, wodurch auch die benötigte Trainingszeit für das System länger wird. Zudem ist die Wahrscheinlichkeit, aus einem lokalen Minimum herauszugelangen, ebenfalls geringer.[7, 12]
Die Wahl der Lernrate spielt also eine wichtige Rolle, um das Training eines Systems effizient zu gestalten. Da diese allerdings immer stark vom jeweiligen Lernproblem abhängt, ist es nicht möglich, einen allgemeingültigen Ansatz zu definieren.

I.2.8 Lern-Abbruch

Um den Punkt zu bestimmen, ab dem man das Lernen eines neuronalen Netzes beenden sollte, muss in den Trainings- und Testphasen ein mittlerer quadratischer Fehler (=MSE) ermittelt

werden, der Trainings- und der Testfehler.

Für den Trainingsfehler wird für jedes Trainingsset der Fehler E bestimmt. Diese werden dann aufsummiert und durch die Anzahl der Trainingssets geteilt. Das gleiche Prinzip gilt auch für den Testfehler mit den Testsets.

Sobald der Testfehler steigt, sollte man die Trainingsphase abbrechen, da das Netz sonst zu stark auf die Trainingsdaten trainiert wird und an Generalisierungsfähigkeit verliert. [8]

II. Forschungspaper: Handwritten Digit Recognition with a Back-Propagation Network

II.1 Vorstellung des Themas und Bestandteile

Das wissenschaftliche Paper von Y. LeCun, et al. [13] befasst sich mit der Erkennung von handgeschriebenen Ziffern mittels eines Backpropagation-Netzwerkes. Mit ihrer Arbeit wollen die Autoren zeigen, dass es möglich ist, Backpropagation-Netzwerke auf reale Bilderkennungsprobleme abzubilden, ohne die Eingaben vorher aufwändig aufzubereiten. Ein wichtiger Hauptbestandteil dieser Arbeit ist die für diesen Anwendungsfall speziell entwickelte Netzwerkarchitektur, die in II.3 genauer erläutert wird.

II.1.1 Anwendung und Eingabedaten

Die Autoren entschieden sich als Anwendungsfall für das Erkennen von handgeschriebenen Ziffern.
Als Eingabedaten wurden Zipcodes genutzt, also die Postleitzahlen der USA. Der Vorteil von Zipcodes liegt darin, dass diese in der Regel nur aus schwarzen und weißen Pixel bestehen und sich die Ziffern relativ gut vom Hintergrund abheben. Außerdem gibt es nur zehn Ausgabekategorien, die den Zahlen 0-9 entsprechen. Die handgeschriebenen Eingabedaten können aufgrund von Variation in Größe, Schreibstil, Schriftart, sowie Sorgfalt des Schriftbildes als komplex aufgefasst werden.

II.1.2 Trainings- und Testset

Die Datenbank der Eingaben bestand sowohl aus handgeschriebenen, als auch maschinellen Ziffern. Für die 9298 handgeschriebenen Ziffern wurden die Zipcodes vorher segmentiert und digitalisiert. Die 3349 maschinellen Ziffern wurden aus 35 verschiedenen Schriftarten zusammengestellt. Das Trainingsset, mit dem das Backpropagation-Netzwerk trainiert wurde, enthielt 7291 handgeschriebene und 2549 maschinelle Ziffern. Das Testset bestand aus den verbliebenen 2007 handgeschriebenen und 700 maschinellen Ziffern. Ein Teil der Daten wurde dabei aufgrund des Schriftbildes fehlerhaft segmentiert oder falsch klassifiziert (vgl. [13], Zipcode recognition).

Einen Beispielausschnitt sieht man auch in Grafik 4.

Figure 4: Beispielausschnitt der Eingaben des Testsets [13]

II.2 Feature Maps

Bevor die spezielle Netzwerkarchitektur erläutert wird, wird hier das Prinzip der Feature Maps grob erklärt, welche in der Netzwerkarchitektur dieses Anwendungsfalles eine große Rolle spielen.

Feature Maps sind das Ergebnis von "Convolution", also der "Faltung" eines Eingabebildes zu einem kleineren Ausgabebild, welches bestimmte Informationen stärker repräsentiert. Ein Beispiel hierfür wäre z.B. Kantenerkennung, wie in Grafik 5 zu sehen ist.

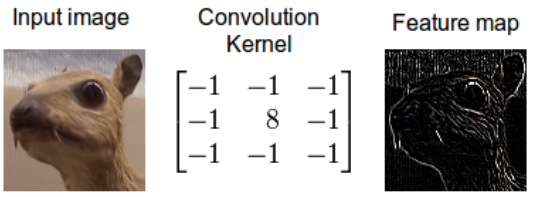

Figure 5: Beispiel einer Faltung mit einem Kantenerkennungs-Kernel [14]

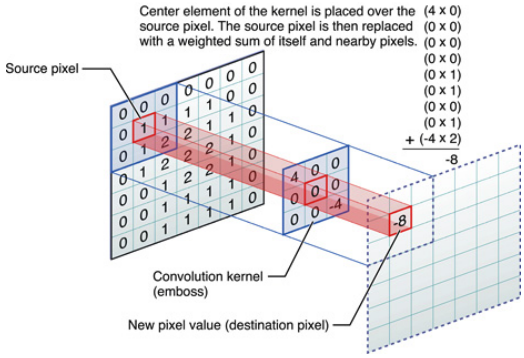

Figure 6: Beispielrechnung einer Matrix mit einem Kernel zu einer Feature Map [15]

Vom Prinzip her wird ein Eingabebild als eine Matrix dargestellt. Jedes Pixel entspricht einem Eintrag in der Matrix.

Da der Anwendungsfall in diesem Paper lediglich mit schwarz-weiß Bildern arbeitet, reicht es, mit einer Matrix für das Eingabebild zu arbeiten.
Der nächste wichtige Bestandteil der Faltung ist ein sogenannter "Kernel". Dieser entspricht einer Gewichtsmatrix, mit der das Eingabebild zu der Feature Map verarbeitet wird.

Die Idee ist, dass der Kernel z.B. beginnend an der linken oberen Ecke des Eingabebildes positioniert und alle Pixel in diesem Bereich mit dem jeweiligen Gewicht des Kernels multipliziert werden. Die Summe der einzelnen Multiplikationen ergibt den Wert für genau ein Pixel der Feature Map, wie auch in Grafik 6 gut zu erkennen ist.

Im nächsten Schritt wird der Kernel um einen Pixel in eine andere Richtung verschoben und ein neuer Pixel der Feature Map berechnet, bis der Kernel einmal über das komplette Eingabebild gelaufen ist.
Je nach Größe des Kernels fällt die Feature Map entsprechend kleiner aus, da ein *Bereich* von Pixeln des Eingabebildes auf *genau ein* Pixel der Feature Map abbgebildet wird. Zum Beispiel führt ein 3x3 Kernel zu einer um 1 Pixel in alle Richtungen kleineren Feature Map als das Eingabebild. Gibt es zusätzlich einen Bias, kann dieser als Sensitivität der Merkmalerkennung aufgefasst werden. [14, 16]

II.3 Netzwerk Architektur

Das Netzwerk selbst besteht neben der Ein- und Ausgabeschicht aus insgesamt vier weiteren hidden layern H1 bis H4, wie auch in Grafik 7 zu sehen ist. Als Lernalgorithmus wird Backpropagation genutzt. Die Eingabe stellt ein auf 28x28 Pixel normalisiertes Bild dar, wobei die Ziffer selbst nur 16x16 Pixel ausmacht. Grund hierfür ist das Vermeiden von Problemen, wenn der Kernel beim Erstellen der Feature Maps die Randpixel des Eingabebildes bearbeitet.

Im Folgenden werden die einzelnen hidden layer der Architektur und deren Funktionsweise genauer erläutert.

II.3.1 H1

In H1 wird die Eingabe in insgesamt vier 24x24 Pixel Feature Maps (H1.1, H1.2, H1.3, H1.4) umgewandelt, wobei jeder Feature Map ein eigener Satz aus 26 Gewichten zugeordnet wird. Aus den 24x24 Pixeln der Feature Maps ergeben sich somit jeweils 576 notwendige Neuronen pro Feature Map. Der Kernel entspricht hier einer 5x5 Gewichtsmatrix, woraus sich auch die 5x5 (+1 Bias) = 26 Gewichte pro Set ergeben.

II.3.2 H2

In H2 findet ein Subsampling der Ausgaben von H1 auf insgesamt vier 12x12 Pixel Bilder (H2.1, H2.2, H2.3, H2.4) statt, um Spielraum für Verschiebungen und Verzerrungen der

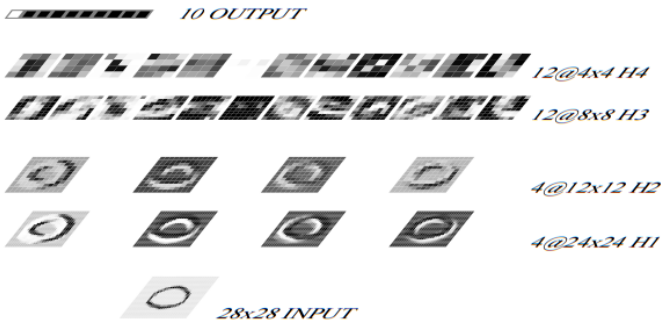

Figure 7: Übersicht über die Netzwerkarchitektur [13]

Eingabeziffer zu ermöglichen. Jedes Neuron von H2 nimmt dabei jeweils vier Neuronen von der entsprechenden Feature Map von H1 als Eingabe. Sprich jedes Neuron von H2.1 nimmt vier Neuronen von H1.1 als Eingabe usw.

II.3.3 H3

In H3 findet wieder eine Umwandlung in Feature Maps statt. Das Ergebnis sind insgesamt 12 8x8 Pixel Feature Maps (H3.1, ..., H3.12), bestehend aus jeweils 64 Neuronen. Der Kernel entspricht hier wieder wie bei H1 einer 5x5 Gewichtsmatrix. Die Verbindungen zwischen den Neuronen von H2 und H3 entsprechen dabei den in Tabelle 1 beschriebenen Zusammenhängen.

H2: \ H3:	1	2	3	4	5	6	7	8	9	10	11	12
1	X	X	X		X	X						
2		X	X	X	X	X						
3							X	X	X		X	X
4								X	X	X	X	X

Table 1: Verbindungen zwischen H2 und H3 [13]

II.3.4 H4

Der letzte hidden layer H4 erfüllt wieder die gleiche Aufgabe, wie layer H2. Das heißt, es findet ein weiteres Subsampling der Ausgaben von H3 auf insgesamt 12 4x4 Pixel Bilder statt. Jede der 12 Gruppen von H3 besteht also aus 12 Neuronen.

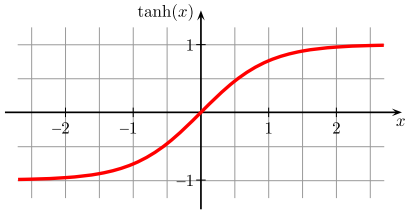

Figure 8: Tangenshyperbolikus-Funktion [17]

II.3.5 Ausgabe

Die Ausgabe besteht dagegen aus 10 Neuronen, je eines pro Ziffer 0-9. Die Neuronen sind vollständig mit H4 verbunden. Sollte die Eingabe als eine Ziffer erkannt werden, ist die Ausgabe des entsprechenden Neurons +1, für die restlichen Neuronen -1. Aufgrund des Ausgabebereiches von -1 bis +1 kann man davon ausgehen, dass keine Sigmoid-Funktion verwendet wurde, sondern z.B. eine Tangenshyperbolikus-Funktion (vgl. Grafik 8). Im Paper ist dies nicht genau spezifiziert.

II.3.6 Ergebnis

Nach 30 Trainingsdurchläufen betrug die Fehlerrate des Trainingssets 1.1% mit einem mittleren quadratischen Fehler von 0.017. Die Fehlerrate für das Testset betrug dagegen 3.4% mit einem MSE von 0.024 (vgl. [13], Results).

Für ihre Anwendung versuchten die Autoren eine möglichst hohe Genauigkeit für ihr Netz zu erreichen, indem sie den Prozentsatz an

Testeingaben bestimmten, der abgelehnt werden musste, um eine Fehlerrate von 1% zu erreichen.

Die Ablehnungskriterien waren:

1. $y_j > t_1$, wobei j ein Neuron der Ausgabeschicht mit dem höchsten Aktivitätszustand y_j darstellt.

2. $y_k < t_2$, wobei k ein Neuron der Ausgabeschicht mit dem zweithöchsten Aktivitätszustand y_k darstellt.

3. $||y_j - y_k|| > t_d$

t_1, t_2, t_d stellen dabei jeweils einen gegebenen Schwellenwert dar.

Das beste Ergebnis für das gesamte Testset stellte eine Ablehnungsrate von 5.7% dar, um eine Fehlerrate von 1% zu erzielen. Um die gewünschte Genauigkeit zu erzielen, ergab sich nur für die handgeschriebenen Daten (ohne die maschinellen Daten) des Testsets eine Ablehnungsrate von 9% (vgl. [13], Results).

Die meisten Fehler des Netzes ergaben sich durch fehlerhafte Segmentierung, Unleserlichkeit oder manuelles, fehlerhaftes Zuweisen der Daten zu einer Kategorie.

In einer Simulation des Trainings benötigte das Netzwerk auf einer SUN SPARCstation 1 ungefähr 3 Tage um 30 Trainingsdurchläufe durch das Trainings- und Testset zu absolvieren, was im Vergleich zu der Größe der beiden Sets einen relativ guten Wert darstellt.

Abschließend konnte das Netzwerk erfolgreich auf einem DSP implementiert und kommerziell genutzt werden.

II.3.7 Abschließende Meinung

Wie bereits zu Beginn in II.1 erwähnt, wollten die Autoren zeigen, dass man reale Bilderkennungsprobleme mithilfe von Backpropagation-Netzwerken lösen kann, ohne die Eingabedaten aufwändig aufzubereiten. Ich stimme dieser Behauptung zu, da die besondere Netzwerkarchitektur für dieses Problem tatsächlich nur eine relativ simple Aufbereitung der Eingabedaten beinhaltet und trotzdem relativ zuverlässige Ergebnisse liefert. Allerdings empfinde ich die Zusammenstellung der Eingabedaten als aufwändig und daher für reale Anwendungen, je nach Problem, als nicht optimal. Die Daten für das Trainings- und Testset mussten vorher manuell segmentiert, eingescannt und den entsprechenden Kategorien zugeordnet werden.

Zudem waren einige Punkte in dem Paper nicht genau definiert oder nur oberflächlich erläutert, wie z.B. die Erklärung zu den Feature Maps oder die fehlende Angabe der im Paper verwendeten Aktivierungsfunktion.

Interessant sind sicher auch die Web-Demos von Yann LeCun zu diesem Thema, die sich auf seiner Webseite [18] befinden und unter anderem auf den Erkenntnissen des hier besprochenen Papers basieren. Leider gibt es keine Demo für die in diesem Paper entwickelte Anwendung, dennoch kann man sich unter Betrachtung der LeNet-5-Demo (ebenfalls unter [18] zu finden) zumindest einen Eindruck machen, wie die Erkennung der Eingabedaten in Korrespondenz mit den entsprechenden Schichten des neuronalen Netzes abläuft.

References

[1] Prof. Dr.-Ing. Hans-Jürgen Scheibl. Grundlagen neuronaler Netze. available from `http://home.f1.htw-berlin.de/scheibl/Algor/index.htm?./Neuronal/Grundlagen.htm` (retrieved: November 2015), August 2015.

[2] Roman Kohut. Neuronale Netze als Modell Boolescher Funktionen, 2007. Freiberg (Sachsen), Techn. Univ., Diss., 2007.

[3] Jürgen Paetz. Soft Computing in der Bioinformatik : Eine grundlegende Einführung und Übersicht, 2006.

[4] Gerald Reif. Grundlagen neuronale Netze. available from `http://www.iicm.tugraz.at/greif/node10.html` (retrieved: November 2015), Februar 2000.

[5] Michael Koops. Erklärung zum biologischen Neuron. available from http://www.biologie-lexikon.de/lexikon/neuron.php (retrieved: November 2015), Januar 2015.

[6] Lars Larsson. *Zur Validierung der Spezifikationen von videobildverarbeitenden Komponenten unter realen Anwendungsbedingungen*. PhD thesis, 2000. Hamburg, Univ., Diss., 2000.

[7] Karl F. Wender Günter Daniel Rey. Grundlagen neuronale Netze. available from http://www.neuronalesnetz.de/ (retrieved: November 2015), November 2015.

[8] Stephan Joerrens. Anwendung eines neuronalen Netzes zur Teilchenklassifizierung in einem Bleiglaskalorimeter. available from https://www.uni-muenster.de/imperia/md/content/physik_kp/agwessels/thesis_db/ag_santo/joerrens_1998_diplom.pdf (retrieved: Dezember 2015), März 1998.

[9] Jeremy Kun. Neural Networks and the Backpropagation Algorithm. available from http://jeremykun.com/2012/12/09/neural-networks-and-backpropagation/ (retrieved: November 2015), Dezember 2012.

[10] Adrian Ulges. Numerische Verfahren und Analysisgrundlagen WS14/15 Skript . available from http://www.cs.hs-rm.de/~ulges/teaching/14NUMANA/index.html (retrieved: November 2015), November 2013.

[11] Stephan Wiesebach Sebastian Seifert, Christian Weidner. Grundlagen Backpropagation Netzwerke.

[12] Prof. Dr. Wolfram-M. Lippe. Einführung in neuronale Netze - Backpropagation Learning (Probleme bei Backpropagation). available from http://cs.uni-muenster.de/Professoren/Lippe/lehre/skripte/wwwnnscript/probleme.html (retrieved: November 2015), Januar 2007.

[13] Y. Le et al. Cun. Advances in Neural Information Processing Systems 2. chapter Handwritten Digit Recognition with a Back-propagation Network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[14] Tim Dettmers. Understanding Convolution in Deep Learning. available from http://timdettmers.com/2015/03/26/convolution-deep-learning/ (retrieved: November 2015), März 2015.

[15] Apple Inc. Performing Convolution Operations. available from https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html (retrieved: November 2015), Oktober 2011.

[16] Theano Development Team. Convolutional neural Networks (LeNet). available from http://deeplearning.net/tutorial/lenet.html (retrieved: November 2015), November 2015.

[17] Tangenshyperbolikus-Funktion Graph. available from https://upload.wikimedia.org/wikipedia/commons/8/87/Hyperbolic_Tangent.svg (retrieved: Dezember 2015), November 2015.

[18] Yann LeCun. Convolutional neural Networks (LeNet-5). available from http://yann.lecun.com/exdb/lenet/ (retrieved: November 2015), November 2015.