DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Aufbauen und Trainieren eines neuronalen Netzes

STUDIENARBEIT

im Studiengang Informationstechnik

an der DHBW Ravensburg
Campus Friedrichshafen

von

Patrick Schlipf
Aglika Mawrodinowa

14.07.2017

Bearbeitungszeitraum	5-6 Theoriephase	
Matrikelnummer, Kurs	TIT14	
	P. Schlipf	5486114
	A. Mawrodinowa	5451317
Betreuer der DHBW	Dipl. Ing. (BA) Claudia Zinser	

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2015.
Wir versichern hiermit, dass wir unsere Projektarbeit mit dem Thema:

Aufbauen und Trainieren eines neuronalen Netzes

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, den 14.07.2017

Patrick Schlipf

Aglika Mawrodinowa

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

INHALTSVERZEICHNIS

1. Aufgabenstellung

Als Anregung für diese Studienarbeit dient ein Artikel aus der Zeitschrift „c't Magazin für Computertechnik“ über neuronale Netze. Dort kann ein neuronales Netz heruntergeladen und angelernt werden. Dabei besteht die Schwierigkeit in der Parametrisierung des Netzes.

In dieser Studienarbeit soll ein einfaches Feed Forward Netz zur Erkennung von handschriftlichen Ziffern aufgebaut werden. Weiterhin ist der Einfluss der Parametrisierung auf das Lernverhalten zu untersuchen. Das aufgebaute neuronale Netz soll zu einem Convolutional Neural Network (Gefaltetes neuronales Netzwerk) erweitert werden. Zum Trainieren des neuronalen Netzes kann grundsätzlich der MNIST-Datensatz verwendet werden. Anschließend soll das neuronale Netz unter Verwendung selbstdefinierter Datensätze trainiert werden.

2. Grundlagen

In diesem Kapitel wird die grundlegende Funktionalität eines künstlichen neuronales Netzes erklärt, sowie ein solches Netz aufgebaut ist.

2.1. Künstliche Intelligenz

Künstliche Intelligenz (kurz KI) ist der Oberbegriff für folgende Themen:

- Natural Language Processing
- Machine Learning
- Deep Learning

„Natural Language Processing“ beschäftigt sich mit der Erkennung und Verarbeitung von schriftlichen und gesprochenen Texten. „Machine Learning“ ist wiederum ein Oberbegriff. Es beschreibt alle Verfahren, die es Maschinen ermöglichen, aus bestehenden Daten Wissen zu extrahieren, also zu lernen. Eine besonders effiziente Methode für das permanente maschinellen Lernens ist das „Deep Learning“ mit künstlichen neuronale Netzen. Die künstlichen neuronalen Netze basieren auf einer statistischen Analyse großer Datenmengen und sind die bedeutendste Zukunftstechnologie innerhalb der KI. [1]

2.2. Lernverfahren

Es gibt drei Lernverfahren, mit denen „Machine Learning“-Algorithmen lernen können. Diese Verfahren werden nachfolgend kurz erklärt.

2.2.1. Überwachtes Lernen

Die Verfahren der Klassifikationsanalyse fallen in die Kategorie des überwachten Lernens. Die Überwachung im Lernprozess besteht dadurch, dass den Trainingsdaten das Ergebnis beigefügt ist.

Ein Beispiel hierfür ist, wie in diesem Projekt, eine handschriftliche Zahlenerkennung, wobei das Bild mit der geschriebenen Zahl und die, für einen Computer verstehende, selbe Zahl dem Algorithmus übergeben werden.

2.2.2. Unüberwachtes Lernen

Das unüberwachte Lernen ist im Wesentlichen ein Synonym für das Clustering. Da man für das Clustering die zu findenden Klassen nicht kennen muss, gilt dieser Lernprozess als nicht überwacht.

Verwendet man hierfür dasselbe Beispiel wie für das überwachte Lernen, so besitzt man anfangs nur die handschriftlichen Zahlen. Der Algorithmus sollte für dieses Problem im besten Falle zehn Cluster finden, die alle Zahlen von Null bis Neun darstellen. Das Clustering gibt dennoch keine Rückschlüsse darauf, welches Cluster zu welcher Klasse gehört. Experten müssen dafür zuerst die Cluster analysieren.

2.2.3. Semi-überwachtes Lernen

Das semi-überwachte Lernen ist eine Mischung der beiden vorherigen Verfahren. Bei diesem Lernprozess besitzt man einen kleinen Teil an klassifizierten Daten, aber bei dem Großteil der Daten ist die Klasse nicht bekannt. Dem Algorithmus werden zuerst die klassifizierten Daten gezeigt, sodass der Algorithmus die Klassen kennt. Anschließend werden die nicht klassifizierten Daten einem Cluster mit nun bekannter Klasse hinzugefügt. [2]

2.3. Künstliche neuronale Netze

Computer sind gut darin, mathematische oder algorithmische Probleme lösen zu können. Doch oftmals gibt es keinen einfachen mathematischen Algorithmus, um ein Problem zu beschreiben. Gesichtserkennung und die Verarbeitung der menschlichen Sprache sind einige

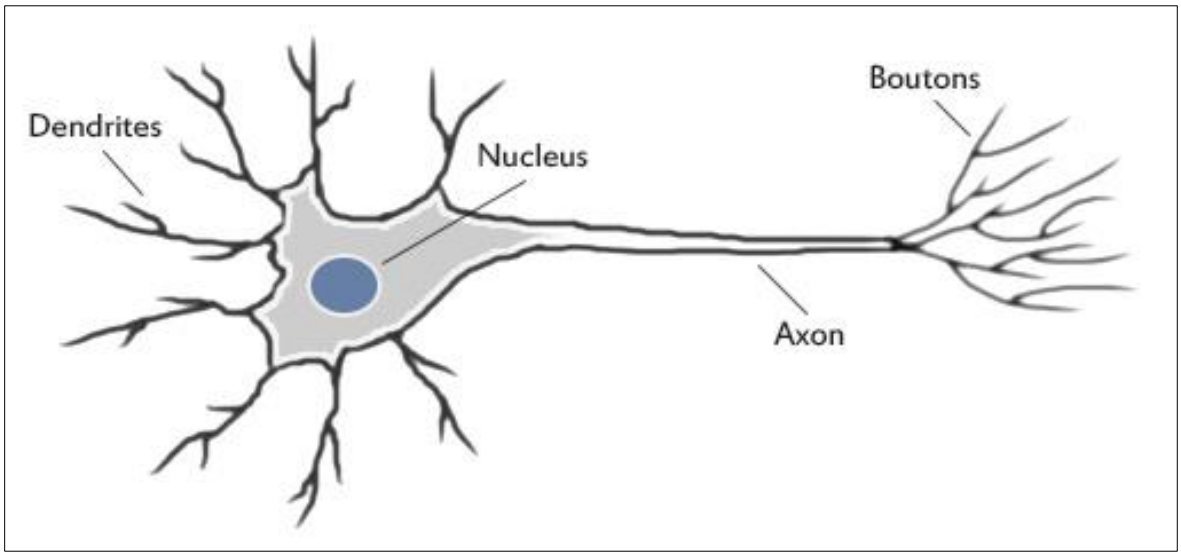


Abbildung 1: Aufbau einer biologischen Nervenzelle [3]

Beispiele, die für einen Computer eine Herausforderung darstellen. Für Menschen hingegen ist das nur eine triviale und alltägliche Aufgabe.

Um diese Probleme zu lösen, muss ein Computer in einer ähnlichen Art und Weise Informationen verarbeiten, wie ein menschliches Gehirn. Es muss ein „Artificial Neural Network“ (ANN), ein „Künstliches Neuronales Netz“, erstellt werden. Der Aufbau des ANN soll von einem biologischen Nervensystem, wie dem Gehirn, inspiriert sein. [3]

2.3.1. Aufbau eines Neurons

Zuerst wird die Funktionsweise eines einzelnen Neurons ermittelt und dies in ein künstliches Neuron umgewandelt. Abbildung 1 zeigt den Aufbau eines biologischen Neurons. Dendriten nehmen die Eingangsinformationen über ein weit verzweigtes Astsystem auf und leiten sie zum Zellkern weiter. Das Neuron summiert alle Eingangsinformationen und leitet diese weiter, falls ein bestimmtes elektrisches Potential überschritten wird. Die Weiterleitung der Informationen übernimmt das Axon, ein langer Nervenzellfortsatz, der elektrische Reize an die synaptischen Endköpfchen weitergibt. Die synaptischen Endköpfchen sind mit weiteren Nervenzellen über Synapsen an den Dendriten verbunden. [4]

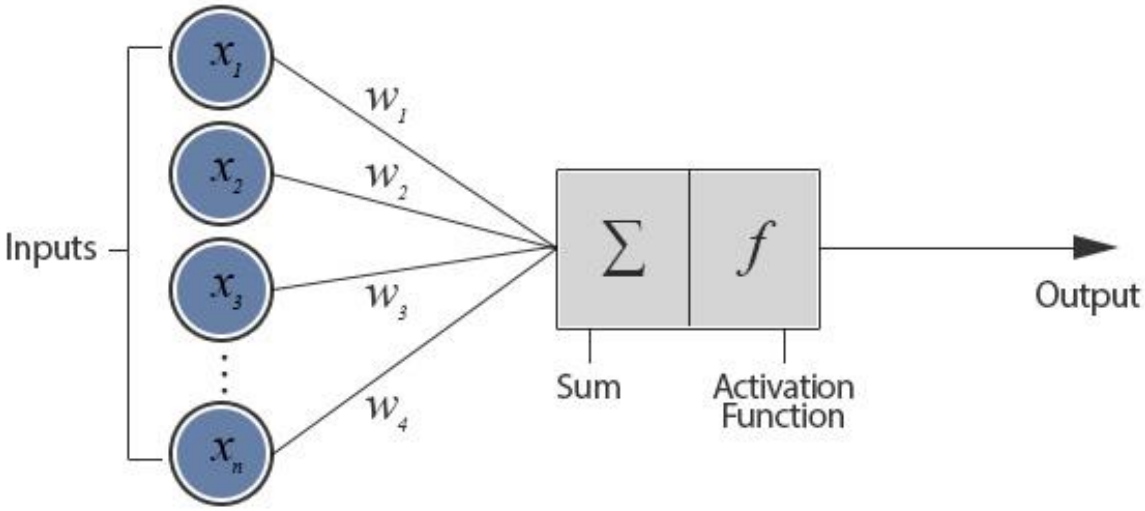


Abbildung 2: Aufbau einer künstlichen Nervenzelle [3]

In gleicher Art und Weise sind künstliche Neuronen aufgebaut. Abbildung 2 lässt einen einfachen Vergleich des Aufbaus biologischer und künstlicher Nervenzellen zu. So sind die Dendriten in Form von $x_1 - x_n$ vorhanden, um die Eingangsinformationen anzunehmen. Diese Werte werden anschließend mit den zugehörigen Gewichten $w_1 - w_n$ verrechnet. Anschließend werden alle Werte summiert und es wird in der Aktivierungsfunktion überprüft, ob ein bestimmter Schwellenwert überschritten ist. Ist der Schwellenwert überschritten, so wird eine Eins an das nachfolgende Neuron übergeben, andernfalls eine Null. Diese Art der künstlichen Nervenzelle wird „Perceptron“ (dt. Perzeptron) genannt.

2.3.2. Aufbau eines künstlichen neuronalen Netzes

Um ein Feedforward Network, wie in Abbildung 3, zu erstellen, müssen mehrere Neuronen mit einander verbunden werden. Feedforward Networks geben Informationen immer an Neuronen der nächsten Schicht weiter. Jede künstliche Nervenzelle einer Schicht ist mit jeder Nervenzelle der vorherigen und nächsten Schicht verbunden. Die Anzahl der Neuronen pro Schicht, sowie die Anzahl der versteckten Schichten, ist nicht limitiert. Trotzdem ist es wichtig, die richtige Anzahl an Neuronen und Schichten auszuwählen. Dies spielt im weiteren Verlauf bei der Optimierung eine große Rolle. [3]

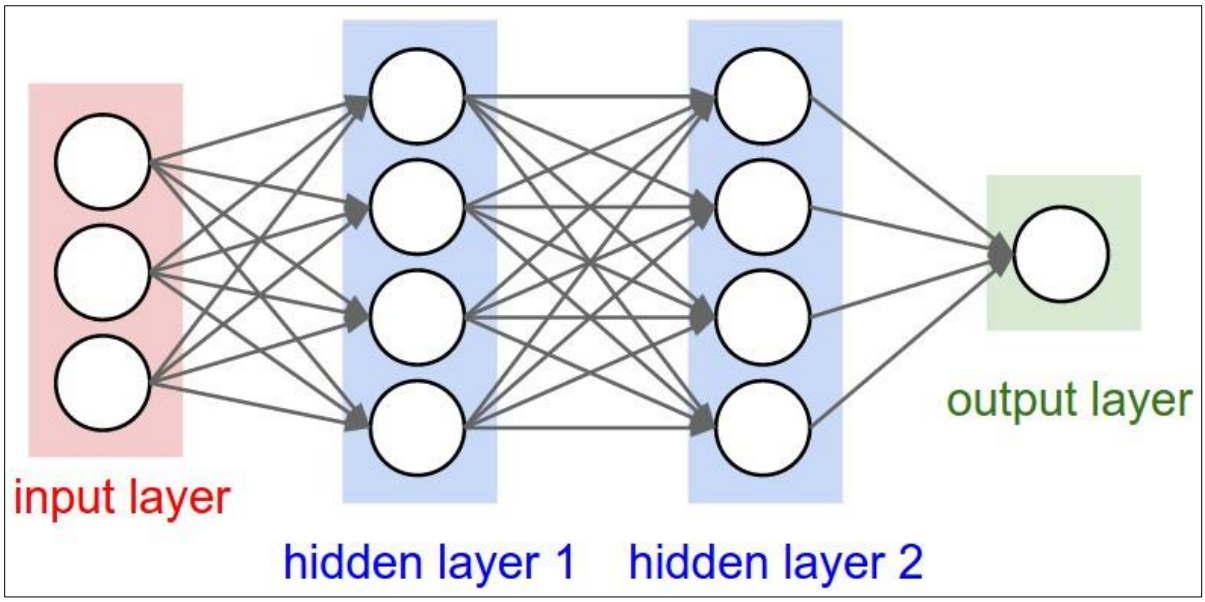


Abbildung 3: Aufbau eines Künstlichen Neuronalen Netzes [14]

2.3.3. Vorteile und Nachteile eines neuronalen Netzes

Ein Vorteil der ANNs ist die Qualität des Ergebnisses bei komplexen Problemen, d. h. für nicht lineare Probleme oder Probleme mit hochdimensionalen Eingabedaten. Da die einzelnen Neuronen die Werte ohne weitere Abhängigkeiten von außen berechnen, sind ANNs vom Ansatz her parallel und können auf Mehrkernrechnern verwendet werden.

Ein weiterer Vorteil der ANNs ist zugleich auch ein Nachteil. Denn ANNs sind lernfähig, d. h. sie werden nicht programmiert, sondern trainiert. Dies bedeutet gleichzeitig, dass diese Modelle ein Basiswissen besitzen, sondern alles selbst lernen müssen. Das führt, im Vergleich zu anderen „Machine Learning“-Algorithmen, zu einem relativ zeitintensiven Trainingsverfahren. Zudem verhalten sich ANNs wie eine Blackbox. Es ist nicht nachzuprüfen, aus welchem Grund ein Netzwerk eine bestimmte Entscheidung getroffen hat. [5]

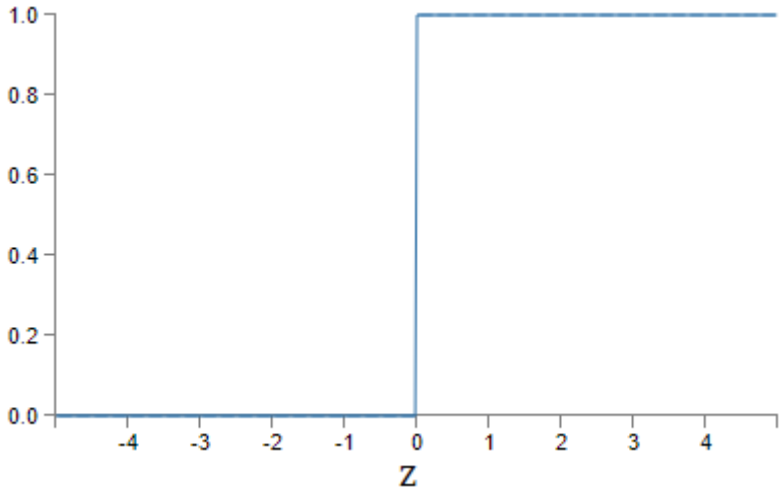


Abbildung 4: Treppenfunktion eines Perzeptrons [7]

2.3.4. Output eines Neurons

Um den Output eines einzigen Perzeptrons zu berechnen, hat Frank Rosenblatt, der Erfinder des Perzeptrons, eine einfache Regel vorgestellt. Rosenblatt hat die Gewichte eingeführt. Diese Gewichte sind reelle Zahlen, die die Bedeutung der jeweiligen Eingabe im Verhältnis zu ihrer Ausgabe ausdrücken. Die Ausgabe des Neurons, Null oder Eins, wird durch Formel 1 dargestellt:

$$output = \begin{cases} 0\ wenn \sum_i w_i x_i \ \leq Schwellwert \\ 1\ wenn \sum_i w_i x_i \ > Schwellwert \end{cases}$$

Formel 1: Berechnung des Outputs eines Perzeptrons [6]

Übersteigt die gewichtete Summe aller Eingaben den Schwellenwert wird eine Eins ausgegeben, andernfalls eine Null. Die Bedingungen dieser Formel sind jedoch etwas schwerfällig ausgedrückt. Um dies zu vereinfachen wird das Skalarprodukt verwendet, $w * x \equiv \sum_i w_i x_i$, wobei w und x Vektoren der Gewichte und Eingaben sind. Die zweite Änderung ist die Verschiebung des Schwellenwerts auf die andere Seite der Gleichung. Hierbei ersetzt man den Schwellenwert durch den Bias, $b \equiv -Schwellenwert$. Formel 2 beschreibt die vereinfachte Formel zur Berechnung des Outputs eines Perzeptrons.

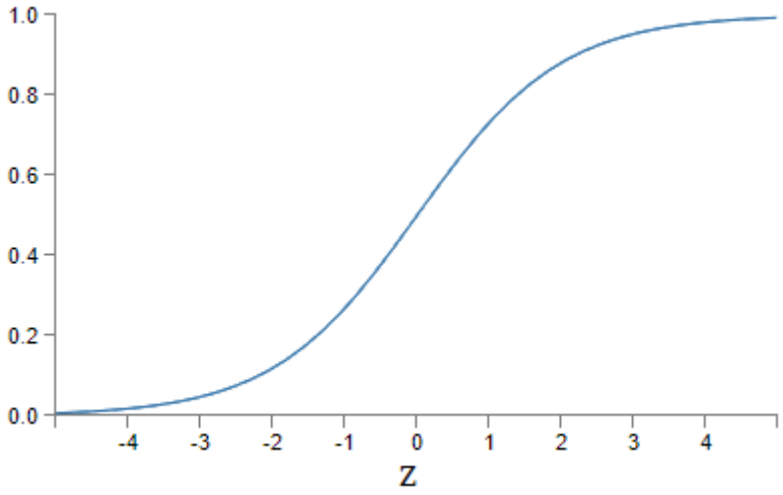


Abbildung 5: Geglättete Treppenfunktion eines Sigmoid Neurons

$$output = \begin{cases} 0 \; wenn \; w * x + b \; \leq 0 \\ 1 \; wenn \; w * x + b \; > 0 \end{cases}$$

Formel 2: Vereinfachte Berechnung des Outputs eines Perzeptrons [7]

2.3.5. Sigmoid Neuron

Das Problem mit Perzeptrons ist, wie in Abbildung 4 dargestellt, die Treppenfunktion. Versucht man ein neuronales Netz anzulernen, so führt man kleine Änderungen an den Bias und Gewichten durch. Gewünscht wäre eine zusammenhängende kleine Änderung des Outputs. Wegen der Treppenfunktion ist dies nicht möglich, da eine kleine Änderung der Gewichte und Bias den Output eines Perzeptrons vollständig umdrehen kann, zum Beispiel von Null auf Eins. Diese Änderung könnte das ganze Netzwerk beeinflussen, sodass sich das Verhalten des ganzen Netzwerks verändert.

Um dieses Problem zu bewältigen benötigt es eine neue Art der künstlichen Neuronen mit dem Namen „Sigmoid Neuron“. Vom Aufbau ist das Sigmoid Neuron genau gleich wie das Perzeptron. Der Unterschied liegt bei den Eingabe- und Ausgabewerten. Denn ein Sigmoid Neuron akzeptiert nicht nur Null und Eins, sondern es akzeptiert jeden beliebigen Wert

zwischen Null und Eins. Ebenso kann die Ausgabe jeglichen Wert zwischen Null und Eins annehmen. Der Ausgabewert $\sigma(w * x + b)$ kann mit Formel 3 ermittelt werden.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Formel 3: Berechnung der Sigmoid Funktion [7]

Es scheint nun so, als haben das Sigmoid Neuron und das Perzeptron nicht viel miteinander zu tun. Vergleicht man aber das Schaubild der Treppenfunktion eines Perzeptrons aus Abbildung 4 mit dem Schaubild der Sigmoid Funktion in Abbildung 5, so erkennt man schnell, dass es sich hierbei um eine geglättete Treppenfunktion handelt.

Mit dem Sigmoid Neuron ist es nun möglich, mittels kleiner Änderungen an den Bias und Gewichten, kleine Änderungen des Ausgabewerts zu erzielen. [7]

2.4. Kostenfunktionen

Es wird ein Algorithmus benötigt, um die passenden Gewichte und Bias zu finden, sodass die Ausgabe des Netzwerks für alle Trainingseingaben x ungefähr dem erwünschten Ergebnis $y(x)$ entspricht. Um zu bestimmen, wie gut dieses Ziel erreicht wird, benötigt man eine Kostenfunktion.

2.4.1. Quadratic Cost Function

Formel 4 zeigt eine solche Funktion. Hierbei handelt es sich um die mittlere quadratische Abweichung (kurz MSE).

$$C(w, b) = \frac{1}{2n} \sum_{x} \|y(x) - a\|^2$$

Formel 4: Mittlere quadratische Abweichung

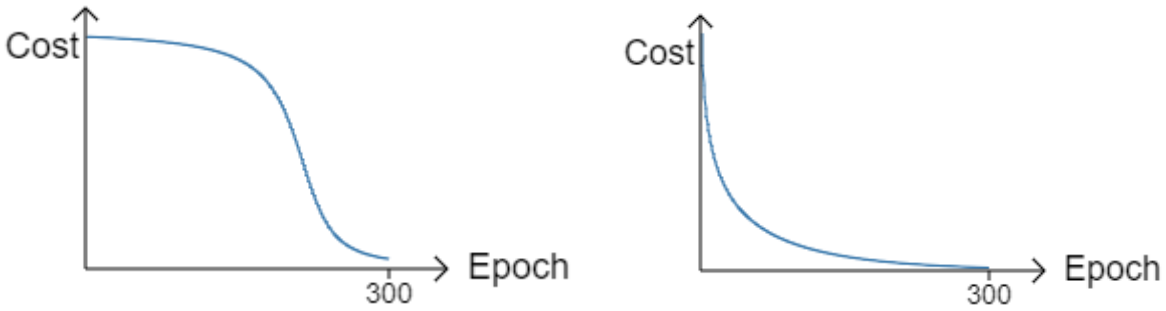


Abbildung 6: Ergebnis der Kostenfunktionen: MSE (l.), Cross-Entropy (r.) [7]

In dieser Formel ist w die Sammlung aller Gewichte im Netzwerk, b alle Bias, n ist die Anzahl aller Trainingseingaben und a ist ein Vektor mit dem Ergebnis des Netzwerks bei der Eingabe x.

Die Kosten $C(w, b)$ werden kleiner, je ähnlicher sich die Ausgabe des Netzwerks und die gewünschte Ausgabe werden. Das bedeutet, der Algorithmus hat eine gute Arbeit geleistet, wenn er Gewichte und Bias findet, sodass $C(w, b) \approx 0$ entspricht. Deshalb ist das Ziel unseres Trainings Gewichte und Bias zu finden, die die Kosten so stark wie möglich verringern.

Man könnte sich jetzt fragen, wieso benötigt man die MSE oder warum interessiert man sich nicht für die Anzahl der korrekt klassifizierten Bilder. Das Problem hierbei ist, dass die Anzahl der korrekt klassifizierten Bilder keine geglättete Funktion der Gewichte und Bias im Netzwerk ist. Eine kleine Änderung der Gewichte und Bias wird für die meiste Zeit keine Auswirkungen auf die Anzahl der richtig klassifizierten Bilder haben. Dies erschwert es deutlich, herauszufinden, wie man die Gewichte und Bias ändern muss, um eine Verbesserung zu erfahren. Es stellt sich heraus, dass mit Hilfe einer geglätteten Kostenfunktion wie die MSE, es jedoch relativ einfach ist.

2.4.2. Cross-Entropy Cost Function

Formel 5 beschreibt die Cross-Entropy Kostenfunktion:

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)]$$

Formel 5: Cross-Entropy Cost Function

Der Output des Neurons ist $a = \sigma(z)$, wobei $z = \sum_i w_i x_i + b$ die gewichtete Summe aller Eingaben ist. Die Anzahl aller Werte der Trainingsdaten wird als n dargestellt. Die Summe aller Trainingseingaben ist x and y ist das dazugehörige gewünschte Ergebnis.

Die Cross-Entropy Funktion besitzt zwei Eigenschaften, die sie zu einer Kostenfunktion machen. Sie wird nicht negativ, d.h. $C > 0$. Des Weiteren nähert sich die Funktion Null, je mehr das Ergebnis des neuronalen Netzes mit dem gewünschten Output übereinstimmt.

Einen Vorteil gegenüber der MSE besitzt die Cross-Entropy Funktion. Sie umgeht das Problem des langsamen Lernens. Um dieses Problem darzustellen kann Abbildung 6 verwendet werden. Diese zeigt den Verlauf der MSE (links) und Cross-Entropy Kostenfunktion. Die Werte wurden mit einem Neuron berechnet, dass von den Gewichten und dem Bias vollkommen falsch initialisiert war. Die Kurven zeigen den Lernfortschritt, beziehungsweise die Kosten der jeweiligen Funktion.

Es ist deutlich zu sehen, dass die Cross-Entropy Kostenfunktion kaum Probleme hat, die Gewichte und den Bias anfangs stark zu verändern. Die MSE hingegen zeigte in den ersten 150 Durchläufen kaum eine Änderung.

Um zu erläutern, wieso die Cross-Entropy Funktion eine solche Reaktion ausübt, muss man die Funktion partiell nach den Gewichten ableiten und vereinfachen.

$$\frac{\partial C}{\partial w_i} = \frac{1}{n}\sum_{x} x_i(\sigma(z) - y)$$

Formel 6: Ableitung der Cross-Entropy Cost Function

Die Gleichung aus Formel 6 sagt, dass die Geschwindigkeit der Lernrate der Gewichte von $\sigma(z) - y$ gesteuert wird. Das bedeutet, umso größer der Fehlerunterschied ist, desto schneller lernt das Neuron. Das Gleiche gilt für das Bias.

Anzumerken ist noch, dass diese Funktion auch dem menschlichen Vorbild folgt. Passiert einer Person ein großer Fehler, so lernt diese Person deutlich schneller daraus als bei kleinen Fehlern. [7]

2.5. Wichtige Algorithmen

2.5.1. Gradient Descent

Die Kostenfunktion muss minimal werden, d.h. es soll das globale Minimum der Funktion gefunden werden. Eine Art dieses Problem zu lösen ist die Verwendung von Differential- und Integralrechnung. Man könnte die Ableitungen berechnen und damit versuchen, Extremwerte zu finden. Dieser Ansatz funktioniert aber nur bei einer geringen Anzahl an Variablen. Aber heutzutage besitzen die größten neuronalen Netzwerke mehrere Milliarden Gewichte und Bias. Die Verwendung von Differential- und Integralrechnungen funktioniert dafür schlichtweg nicht.

Hier kommt der Algorithmus „Gradient Descent“ ins Spiel. Die Funktionsweise des Algorithmus kann man sich folgendermaßen vorstellen. Angenommen man rollt einen Ball ein Tal hinunter. Aus eigener Erfahrung kann man behaupten, dass der Ball auch wirklich in die tiefste Stelle des Tals rollt. Auf dieser Annahme basiert „Gradient Descent“.

Was passiert, wenn man den Ball einen kleines Stück Δv_1 in die Richtung v_1 und einen kleines Stück Δv_2 in die Richtung v_2 bewegt? Unter der Annahme, dass C die Funktion ist, für die das Minimum gefunden werden soll, so ergibt sich die Gleichung aus Formel 7.

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

Formel 7: Gleichung für eine Bewegung in zwei Richtungen [7]

Es sollen Werte für Δv_1 und Δv_2 gefunden werden, damit ΔC negativ wird, also dass der Ball ins Tal rollt. Um herauszufinden, wie die Werteauswahl getroffen wird, ist es hilfreich Δv als einen Vektor aller Änderungen in v zu definieren, $\Delta v = (\Delta v_1, \dots, \Delta v_m)$. Die Steigung von C wird als Vektor der partiellen Ableitung definiert. Somit entspricht der Steigungsvektor $\nabla C = (\frac{\partial C}{\partial v_1}, \dots, \frac{\partial C}{\partial v_m})$. Mit diesen Definitionen kann die Formel 7 zu Formel 8 umformuliert werden.

$$\Delta C \approx \nabla C * \Delta v$$

Formel 8: Gleichung für eine Bewegung in zwei Richtungen umformuliert [7]

Das Interessante an dieser Gleichung ist, dass sie uns zeigt, wie man Δv wählen muss, damit ΔC negativ wird. Angenommen man wählt Formel 9 um Δv zu beschreiben.

$$\Delta v = -\eta \nabla C$$

Formel 9: Ermittlung der Änderungen [7]

Bei η handelt es sich um die Lernrate, die eine kleine, positive Ganzzahl ist. Setzt man Formel 9 in Formel 8 ein, so formt sich $\Delta C = -\eta \nabla C * \nabla C = -\eta \|\nabla C\|^2$.

Da $\|\nabla C\|^2 \geq 0$ ist, wird garantiert, dass $\Delta C \leq 0$ ist. D.h. C wird immer abnehmen und nie zunehmen, wenn man Δv, wie in Formel 9 beschrieben, ändert.

Die Position v verändert sich somit um Δv. Dies ist in Formel 10 dargestellt.

$$v \rightarrow v' = v - \eta \nabla C$$

Formel 10: Berechnung der absoluten Änderung [7]

Zusammengefasst wird bei dem „Gradient Descent“ Algorithmus immer wieder die Steigung ∇C berechnet, um sich dann in die entgegengesetzte Richtung zu bewegen. [7]

2.5.2. Stochastic Gradient Descent

Nun geht es darum zu wissen, wie man „Gradient Descent“ anwenden muss, damit ein neuronales Netz damit lernen kann. Die Idee dahinter ist, dass der „Gradient Descent“ Algorithmus dazu verwendet wird, um die Gewichte w_k und Bias b_l zu finden, damit die MSE minimal wird. Um zu verstehen, wie dies funktioniert, formuliert man Formel 10 nun mit den Gewichten und Bias. Diese Werte ersetzen dabei die Variablen v_i. Die „Position“ bekommt die Bestandteile w_k und b_l und der Steigungsvektor ∇C bekommt die dazugehörigen Bestandteile $\partial C/\partial w_k$ und $\partial C/\partial b_l$. Setzt man dies nun in Formel 10 ein, entsteht:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

Formel 11: Berechnung der absoluten Änderung für Gewichte und Bias

Doch es gibt ein Problem. Die Kostenfunktion hat die Form $C = \frac{1}{n}\sum_x C_x$, d.h. es ist der Durchschnitt der Kosten für individuelle Trainingsbeispiele. Um die Steigung ∇C zu berechnen, müssen zuerst alle Steigungen ∇C_x für jeden Trainingsdatensatz einzeln berechnet werden und anschließend wird der Durchschnitt berechnet. Problematisch wird es, wenn die Anzahl der Trainingsdaten sehr groß ist. Wenn dies der Fall ist, kann die Berechnung lange benötigen und das Lernen könnte langsam erscheinen.

Um die Berechnungen und somit das Lernen zu beschleunigen, gibt es den „Stochastic Gradient Descent“ Algorithmus. Die Idee ist es, die Steigung ∇C abzuschätzen, indem man ∇C_x für eine kleine Gruppe zufällig ausgewählter Trainingsdaten berechnet. Durch die Mittelung über die ausgewählten Daten kann sehr schnell eine gute Schätzung der wahren Steigung ∇C gemacht werden. Die Gruppe der zufällig ausgewählten Daten wird „Mini-batch“ genannt.

Hat man nun Trainingsdaten mit der Anzahl $n = 60.000$, wie MNIST, und setzt die Anzahl der Daten pro Mini-batch auf $m = 10$, so beschleunigt man die Abschätzung der Steigung

um den Faktor 6.000. Die Schätzung wird zwar nicht perfekt sein, aber das muss sie auch nicht. Man will nur wissen, ob man in die richtige Richtung geht, damit C kleiner wird. [7]

2.5.3. Backpropagation

Dieser Algorithmus ist deutlich komplexer als die zuvor beschriebenen Algorithmen. Aus diesem Grund wieder der Backpropagation Algorithmus nur oberflächlich beschrieben.

Der Backpropagation Algorithmus wurde ursprünglich in den 1970ern vorgestellt, aber der Algorithmus bekam erst mit dem berühmten Paper aus 1986 von David Rumelhart, Geoffrey Hinton und Ronald Williams die Aufmerksamkeit, die ihm zustand. Das Paper beschreibt mehrere neuronale Netzwerke, bei denen Backpropagation deutlich schneller arbeitet als frühere Ansätze für das Lernen. Dies ermöglichte es neuronale Netze für Probleme zu verwenden, die bis zum damaligen Zeitpunkt unlösbar waren. [8]

The Backpropagation Gleichungen ermöglichen es, die Steigung der Kostenfunktion zu berechnen. Die einzelnen Schritte des Algorithmus sind:

- **Eingabe x:** Berechne den Output der Eingabeschicht.
- **Feedforward:** Berechne den Output für jede nachfolgende Schicht: $l = 2, 3, \dots, L$
- **Output Fehler:** Berechne den Vektor des Outputfehlers δ^L
- **Propagiere den Fehler zurück:** Berechne den Fehler δ^l in Bezug auf die Fehler folgenden Schicht δ^{l+1}
- **Output:** Berechne die Änderungsrate der Kosten bezüglich der Gewichte und Bias

Schaut man sich den Algorithmus an, bemerkt man, warum er Backpropagation heißt. Man berechnet die Fehlervektoren δ^l rückwärts, angefangen an der letzten Schicht. Die Gewichte und Bias werden dann abhängig von ihrem Einfluss auf den Fehler geändert. Dies garantiert bei einem erneuten Anlegen der Eingabe eine Annäherung an das gewünschte Ergebnis.

2.5.4. Optimierung der Hyperparameter

Ein neuronales Netz zu debuggen ist herausfordernd. Es gibt mehrere Hyperparameter die richtig eingestellt sein müssen, z.B. die Lernrate η, die Mini-batch Größe, die Anzahl der Schichten und Neuronen und weitere Parameter. Wenn die anfängliche Auswahl der Hyperparameter keine besseren Ergebnisse erzielt als wenn man den Zufall entscheiden lässt, so sind einer oder mehrere der Parameter falsch eingestellt. Selbst wenn nur ein Wert von den Hyperparametern komplett falsch initialisiert ist, führt dies im schlechtesten Fall schon zu dem Problem, dass die Ergebnisse willkürlich erscheinen. Trifft man zum ersten Mal auf diese Problem, so weiß man nicht, welche Hyperparameter richtig eingestellt sind und welche nicht. An diesem Punkt könnte jeder Hyperparameter falsch sein.

Das erste Herausforderung ist es, nicht triviale Lernerfolge zu erzielen, z.B. das Netzwerk erreicht bessere Ergebnisse als die Wahrscheinlichkeit beim Raten. Dies ist gar nicht so einfach wie es sich anhört.

Datenreduktion

Zuerst sollten die Trainingsdaten deutlich reduziert werden. Es ist möglich mehrere Klassen wegfallen zu lassen und auf beispielsweise 80% der Daten anfangs zu verzichten. Dies ermöglicht viel schnelleres Experimentieren.

Eine weitere Möglichkeit ist die Anzahl der Trainings- und Testdaten pro Durchlauf zu verringern. Es ist wichtig, dass man so schnell wie möglich Rückmeldung über die Ergebnisse bekommt, um notfalls sofort eingreifen zu können und neue Parameter testen.

Early stopping

Am Ende jedes Durchlaufs wird automatisch die Klassifizierungsgenauigkeit für die Validierungsdaten berechnet. Falls die Genauigkeit nicht mehr weiter steigt, wird das Lernen gestoppt. Durch diese Technik muss man sich keine Gedanken mehr über die Anzahl der Durchläufe machen.

Ein zusätzlicher Vorteil des „Early stopping“ ist, dass zudem automatisch das Overfitting unterbunden wird.

Learning rate schedule

Statt eine konstanten Lernrate zu besitzen, ist es möglich mehrere Lernraten zu verwenden. Am Anfang des Trainings sind die Gewichte noch sehr falsch eingestellt. Deshalb ist es am besten, zu diesem Zeitpunkt eine große Lernrate zu verwenden, um eine möglichst schnelle Änderung der Gewichte hervorzurufen. Später kann die Lernrate für das Finetuning angepasst werden.

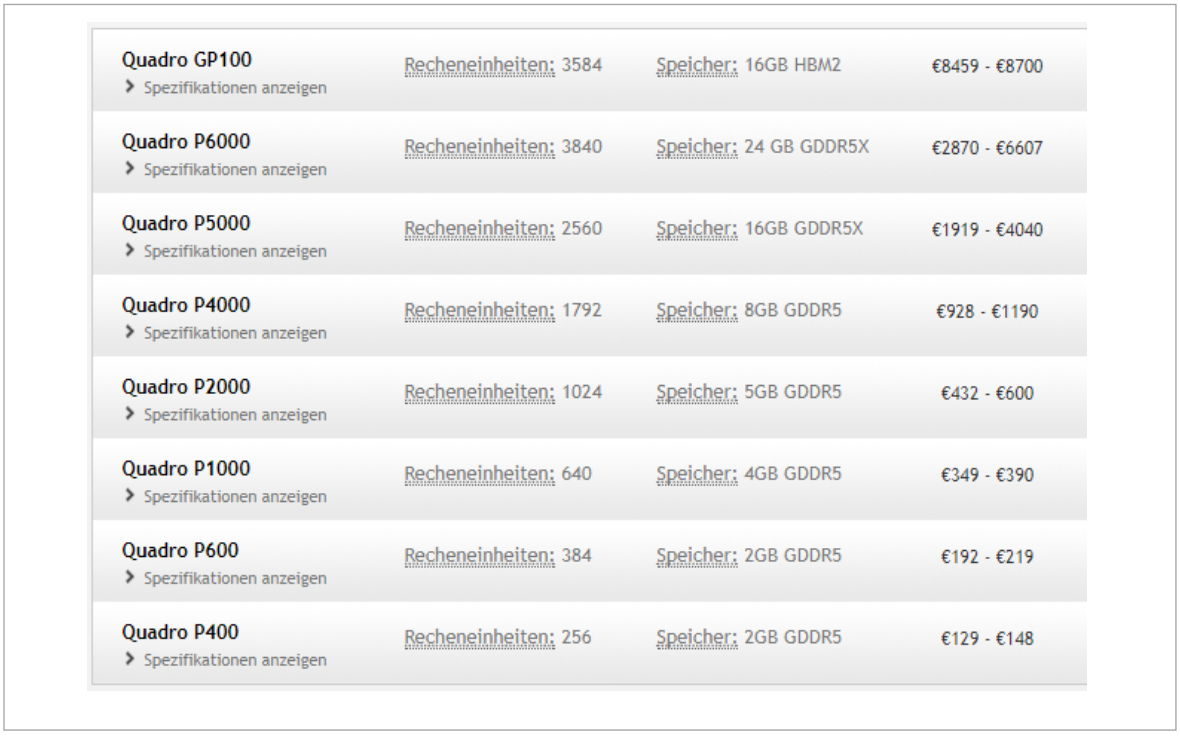
Automatisierte Techniken

Eine bekannte Technik für das Finden von Hyperparametern ist „Grid search“. Dieses Verfahren sucht automatisch aus einer Reihe angegebener Werte die besten Hyperparameterwerte. [7]

3. Aufbau unseres neuronalen Netzes

3.1. Vorgehensweise

Im Artikel der Zeitschrift c't Magazin wird der Aufbau eines neuronalen Netzes mit Python 2.7 beschrieben. Dabei sind Bibliotheken wie brainstorm, pycuda und pillow im Einsatz. Brainstorm stellt unter Python ein fertiges neuronales Netz dar. Sie beruht auf Numpy und setzt die Installation von OpenBLAS voraus. PyCUDA ist eine Python- Programmierumgebung für CUDA. Sie bietet eine Schnittstelle für Python zu der Grafikkarte, die die CUDA-Technologie unterstützen. Mit PyCUDA kann auf Nvidias CUDA-Parallel-Berechnungs-API von Python aus zugegriffen werden. Die CUDA Technologie von NVIDIA wird von den Produkten GeForce, ION, Quadro und Tesla Grafikprozessoren unterstützt. Der Preis solcher leistungsfähigen Grafikkarten kann bei bis zu mehreren tausend Euro liegen (Abbildung 7). Aus diesem Grund wird in dieser Studienarbeit auf den Einsatz einer Grafikkarte mit leistungsfähiger GPU verzichtet. Der Aufbau eines neuronalen Netzes mit brainstorm wird aufgrund der Verwendung von CUDA nicht umgesetzt.



Quadro GP100 › Spezifikationen anzeigen	Recheneinheiten: 3584	Speicher: 16GB HBM2	€8459 - €8700
Quadro P6000 › Spezifikationen anzeigen	Recheneinheiten: 3840	Speicher: 24 GB GDDR5X	€2870 - €6607
Quadro P5000 › Spezifikationen anzeigen	Recheneinheiten: 2560	Speicher: 16GB GDDR5X	€1919 - €4040
Quadro P4000 › Spezifikationen anzeigen	Recheneinheiten: 1792	Speicher: 8GB GDDR5	€928 - €1190
Quadro P2000 › Spezifikationen anzeigen	Recheneinheiten: 1024	Speicher: 5GB GDDR5	€432 - €600
Quadro P1000 › Spezifikationen anzeigen	Recheneinheiten: 640	Speicher: 4GB GDDR5	€349 - €390
Quadro P600 › Spezifikationen anzeigen	Recheneinheiten: 384	Speicher: 2GB GDDR5	€192 - €219
Quadro P400 › Spezifikationen anzeigen	Recheneinheiten: 256	Speicher: 2GB GDDR5	€129 - €148

Abbildung 7: Preise von NVIDIA Quadro Desktop Grafikkarten (Stand 07/2017) [9]

Eine weitere Möglichkeit, die Funktionsweise von Neuronalen Netzen kennen zu lernen, bietet das von Microsoft angebotene Produkt „Azure Machine Learning Studio“ (AMLS).

Microsoft Azure Machine Learning Studio ist ein Tool mit dem man Lösungen für Vorhersageanalysen erstellen, testen und bereitstellen kann, die mit den zur Verfügung stehenden Daten arbeiten. Machine Learning Studio veröffentlicht Modelle als Webdienste, die von benutzerdefinierten Apps oder BI-Tools wie Excel problemlos genutzt werden können. [10]

Da das Tool sehr stark an den weiteren Microsoft Produkten, wie Cortana, orientiert ist, erst nach einer Anmeldung im Netz zur Verfügung steht und keine ausreichende Flexibilität beim Einsatz bietet, wurde dieses in der Studienarbeit nicht eingesetzt.

Der Aufbau des neuronalen Netzes ist in Python unter Verwendung von Bibliotheken, wie numpy, pillow, matplotlib und pickle entwickelt

3.2. Entwicklungsumgebung und Einschränkungen

Die neuronalen Netze sind in Python 3.5 programmiert. Da Python plattformunabhängig ist, sollte das neuronale Netz ohne weitere Probleme auf Windows und Linux laufen. Es ist anzumerken, dass das Programm nicht auf Linux getestet wurde.

Es werden folgende Bibliotheken verwendet:

- numpy (Matrixoperationen)
- pillow (Image Library)
- matplotlib (mathematische Darstellungen)
- pickle (Objektseriallisierung)
- gzip (gz-Dateiformat)
- os
- theano

Für die Verwendung von Convolutional Layers wurde die Pythonbibliothek Theano verwendet. Theano stellt verschiedene Arten von neuronalen Netzen zur Verfügung. Zusätzlich ist es damit möglich, ein neuronales Netz mit der Grafikkarte zu trainieren.

Da Theano spezielle Abhängigkeiten besitzt, die sich nicht installieren ließen, wurde Anaconda installiert. Dabei handelt es sich um ein eigenes kleines Software Ökosystem. Anaconda wird mit Python 3.x installiert. Außerdem sind über 100 Python Packages vorinstalliert und es können einfach neue Packages installiert werden. Theano ist in Anaconda standardmäßig installiert.

Als Datenquelle wird einerseits der öffentlich verfügbare Datensatz von MNIST verwendet. Zudem wurde ein weiterer Datensatz erstellt. Dieser Datensatz beinhaltet maschinelle Großbuchstaben.

3.3. Ablauf des neuronalen Netzes

Die Funktionsweise und der Ablauf eines neuronalen Netzes sind im Grunde relativ einfach. Es werden zuerst den Gewichten und Bias zufällige Zahlen mit dem Mittelwert Null und einer Standardabweichung von Eins zugewiesen. Anschließend liegt ein untrainiertes neuronales Netz bereit.

Wird das neuronale Netz mit den Trainingsdaten trainiert, so liegt an jedem Eingabeneuron ein Wert an. Jedes Neuron berechnet seinen Ausgabewert und gibt diesen an die Neuronen in der nächsten Schicht weiter. Dieser Prozess wird wiederholt, bis an den Ausgabeneuronen ein Signal anliegt, das das Ergebnis bestimmt. Anschließend wird das neuronale Netz rückwärts durchpropagiert um die Fehlerrate zu bestimmen. Zudem werden die Gewichte und Bias mit Hilfe des „Stochastic Gradient Descent“ angepasst, sodass die Kosten der MSE Funktion verringert werden.

Dieser beschriebene Ablauf wird bei jedem Trainingsdatensatz wiederholt. Durch die ständigen Anpassungen der Gewichte und Bias lernt das neuronale Netz.

3.3.1. Verbesserungen des zweiten neuronalen Netzes

Bei dem zweiten neuronalen Netz sind einige Verbesserungen vorgenommen worden. Es wurde eine neue Kostenfunktion verwendet. Statt der MSE wird in diesem neuronalen Netz die „Cross-Entropy“ (dt. Kreuzentropie) verwendet.

Zudem wurde die Initialisierung der Gewichte und Bias verändert. Angenommen wir haben ein Neuron mit n_{in} Gewichten. Diese Gewichte werden als Gaußsche Zufallszahl mit der Mittelwert Null und einer Standardabweichung von $1/\sqrt{n_{in}}$ initialisiert. Dies hat den Vorteil, dass die Neuronen nicht von Beginn an gesättigt sind und nur noch minimalste Änderungen an den Gewichten durchführen.

Die letzte Verbesserung ist das Hinzufügen von Regularisierungen. Sogenannte Regularisierungen sind dazu da, um „Overfitting“ zu vermeiden. Eine der meistbekannten Techniken ist die L2 Regularisierung. Die Idee hinter dieser Regularisierung ist es, einen extra Ausdruck an die Kostenfunktion zu hängen. Dieser Ausdruck heißt Regularisierungsterm und ist in Formel 12 zu sehen.

$$\frac{\lambda}{2n}\sum_{w} w^2$$

Formel 12: L2 Regularisierungsterm [7]

Um diese Regularisierung anwenden zu können, muss der „Stochastic Gradient Descent“ Algorithmus angepasst werden. Dazu wird die Kostenfunktion mit angehängtem Regularisierungsterm partiell nach den Gewichten und Bias abgeleitet. Bei den Bias ändert sich nichts. Die Formel für die Gewichte (siehe Formel 11) erfährt aber eine Änderung.

$$w \to \left(1 - \frac{\eta\lambda}{n}\right) w - \frac{\eta}{m}\sum_{x} \frac{\partial C_x}{\partial w}$$

Formel 13: Berechnung der absoluten Änderung mit L2 Regularisierung [7]

Formel 13 zeigt die Formel zur Berechnung des „Stochastic Gradient Descent“ mit der L2 Regularisierung. Es ist exakt die Formel 11, bis auf den Regularisierungsfaktor $1 - \frac{\eta\lambda}{n}$.

3.3.2. Aufbau und Verwendung von Convolutional Layers

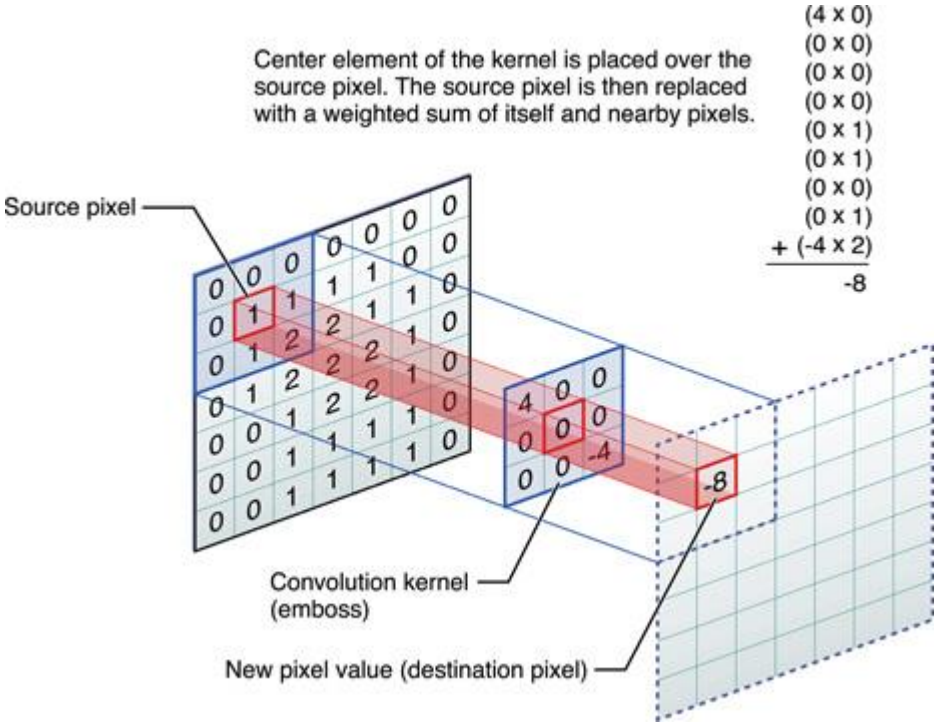


Abbildung 8: Berechnung der Faltung mit einem Filterkern [15]

Ein Convolutional Neural Network besteht aus einer Anzahl von Convolutional und Subsampling Schichten. Optional können daraufhin noch „Vollständig verbundene Schichten“ kommen. Die Eingabedaten einer Convolutional Schicht sind $m * m * r$, wobei m die Höhe und Breite des Bildes sind und r ist die Anzahl der Kanäle. Ein RGB Bild besitzt $r = 3$ Kanäle. Die Convolutional Schicht besitzt k Filterkerne, oder auch Kernel genannt, mit der Größe $n * n * q$, dabei ist n kleiner als die Dimension des Bilds und q ist entweder gleich der Anzahl an Kanälen r oder kleiner. Die Anzahl der Filterkerne kann sich zwischen Convolutional Schichten unterscheiden. Aus dem Eingangswerten des Bilds und den Filterkernen lassen sich k Feature Maps berechnen. Zum einfacheren Verständnis stellt Abbildung 8 diesen Prozess dar. Es muss jedoch beachtet werden, dass in der Abbildung nur ein Filterkern verwendet wird.

In der Subsampling Schicht werden überflüssige Informationen verworfen. Den für die Objekterkennung in Bildern ist es nicht wichtig die exakte Position einer Kante zu kennen. Es reicht die ungefähre Position eines Features. Eines der bekanntesten Verfahren dazu ist das

„Max Pooling“ mit einer Größe von $p * p$, dabei nimmt p Werte zwischen zwei und fünf an. In Abbildung 9 sieht man ein Beispiel des „Max Pooling“ mit einem 2x2 Filter. Es wird immer ein 2x2 Feld ausgewählt und nur die höchste Zahl wird übernommen.

Convolutional Neural Networks sind heutzutage „State-of-the-Art“ wenn es ums Thema „Künstliche Intelligenz“ geht. Mit ihnen werden in Gebieten, wie der Bild- und Spracherkennung, Bestleistungen erzielt. [11]

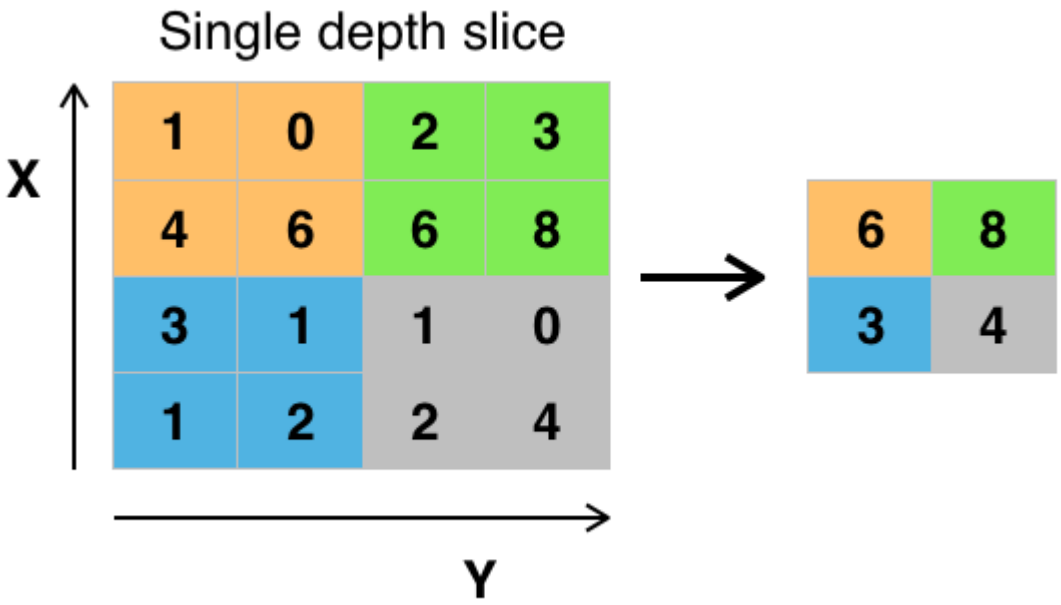


Abbildung 9: Max Pooling mit 2x2 Filter [16]

3.4. Training mit handgeschriebenen Ziffern aus dem MNIST-Datensatz

3.4.1. Der MNIST-Datensatz

Das neuronale Netz wird zuerst mit handgeschriebenen Ziffern aus dem MNIST-Datensatz angelernt.

Für das Problem handgeschriebene Ziffern zu erkennen, gibt es den MNIST-Datensatz (Mixed National Institute of Standards and Technology database). Er besteht aus 60 000 Graustufenbildern von 28 x 28 Pixeln Größe, die aus größeren Schwarzweißbildern berechnet wurden. Die Ziffern wurden auf eine einheitliche Größe gebracht und zentriert.

Die Menge der Eingabeneuronen legt fest, wie viele Zahlen das Netz als Eingabe akzeptiert. Der MNIST-Datensatz kann vom Internet heruntergeladen werden. Die Daten werden geteilt. Ein Teil von den Bildern wird zum Trainieren des neuronalen Netzes benutzt. Ein anderer Teil wird zum Prüfen der Trainingsergebnisse eingesetzt. [12]

3.4.2. Ergebnisse des ersten neuronalen Netzes

Nachdem das erste neuronale Netz fertig war, lief zugleich der erste Testlauf. Das erste Mal ein eigenes neuronales Netz zu trainieren ist schon etwas Besonderes. Zumindest bis zu einem von zwei Punkten.

- **Falsche Hyperparameter:** Sind die Hyperparameter nicht richtig eingestellt, ratet das neuronale Netz. Bis man die passenden Parameter gefunden hat, können schnell mal mehrere Tage vergehen.
- **Blackbox:** Da ein neuronales Netz wie eine Blackbox ist, bringt ein trainiertes neuronales Netz nicht viel. Man kann Objekte damit klassifizieren und man kann sich die Zahlen der Variablen anschauen. Mehr gibt es nicht zu sehen.

In diesem Projekt waren akzeptable Hyperparameter schon gegeben und diese konnten verwendet werden. Deshalb wurde selbst im ersten Testlauf eine außerordentlich hohe Genauigkeit erreicht. Abbildung 10 zeigt die Genauigkeit des neuronalen Netzes während des Trainings.

Nach Abschluss des Trainings erzielte das neuronale Netz eine Genauigkeit von ca. 93.3%.

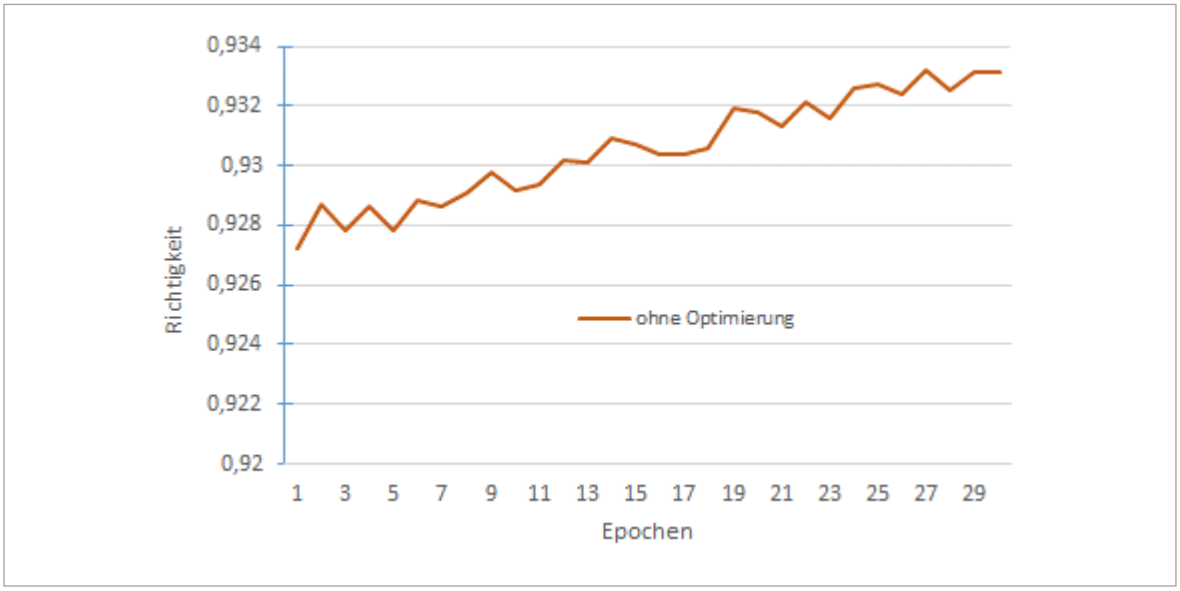


Abbildung 10: Trainingsverlauf des neuronales Netzes mit MNIST-Datensatz ohne Optimierung

3.4.3. Optimiertes neuronales Netz

Das optimierte neuronale Netz erzielte durch die Neuerungen, wie z.B. eine bessere Initialisierung der Gewichte und Bias oder eine neue Kostenfunktion, weitaus beeindruckendere Ergebnisse.

Hiermit waren Genauigkeiten von 97% zu erreichen. Dieses neuronale Netz lässt seinen kleinen, nicht optimierten Bruder im Schatten stehen.

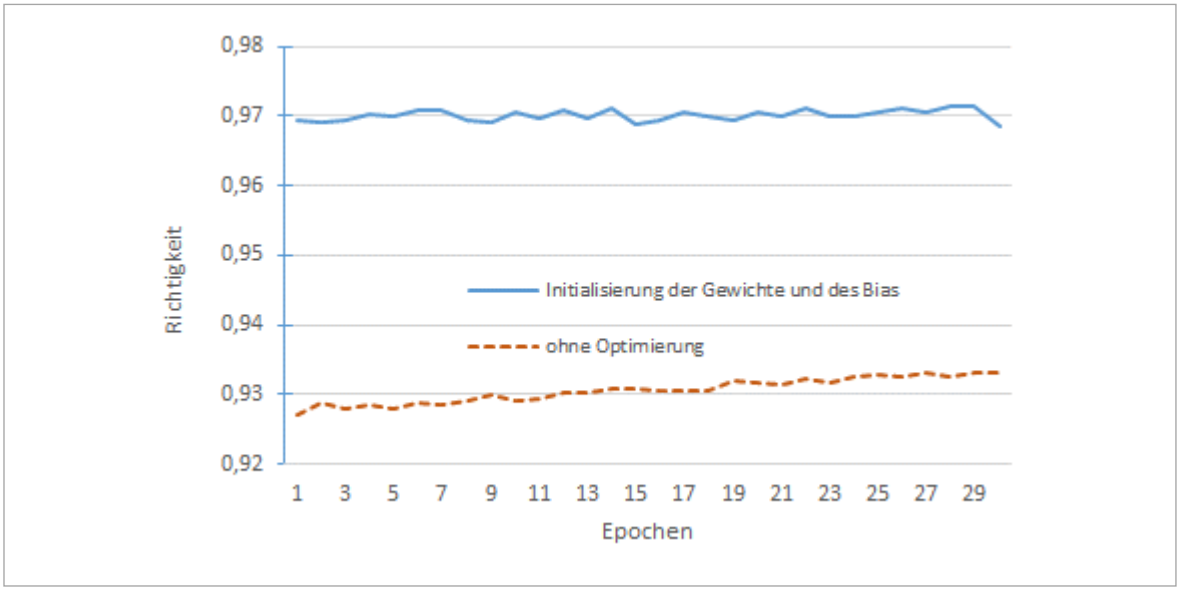


Abbildung 11: Trainingsverlauf des optimierten neuronalen Netzes mit MNIST-Datensatz

3.4.4. Convolutional Neural Network

Zuletzt gab es noch das Convolutional Neural Network. Dieses neuronale Netz ist deutlich komplexer als die zwei Vorgängernetze und erzielte somit bei den Tests auch die höchste Genauigkeit. Dieses Netzwerk hat mit einer Genauigkeit von knapp über 99% handgeschriebene Zahlen der richtigen Klasse zugeteilt. Das bedeutet, von den 10.000 Testzahlen sind lediglich 100 Zahlen nicht der richtigen Klasse zugeordnet worden. Die Unterschiede der einzelnen Netze werden in Abbildung 12 nochmals aufgezeigt.

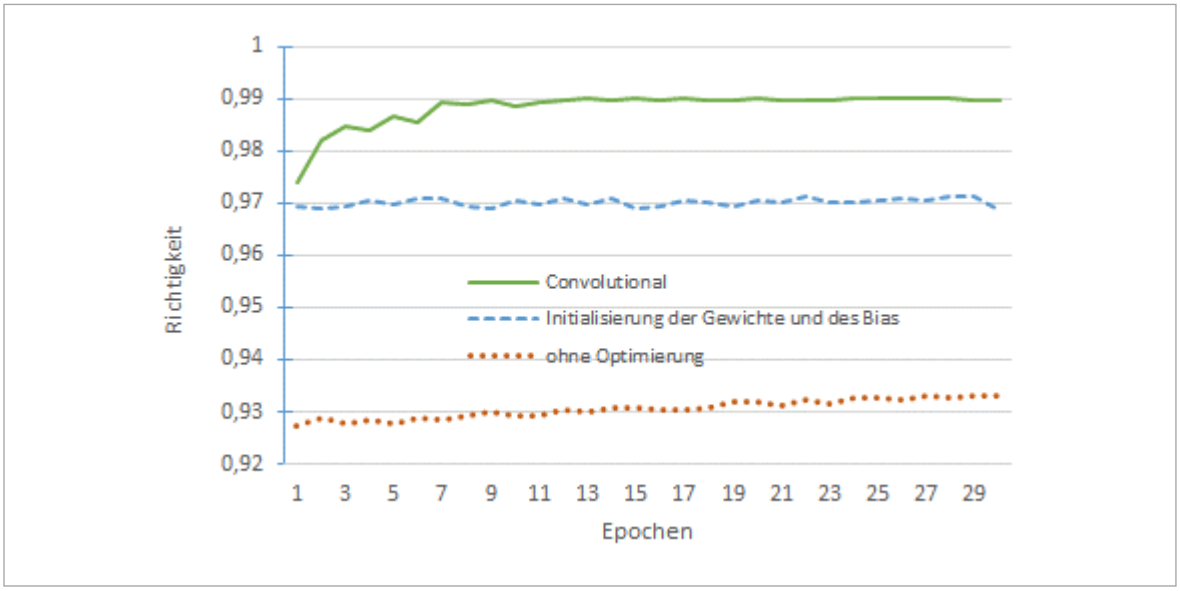


Abbildung 12: Trainingsverlauf aller erstellten neuronalen Netzen mit MNIST-Datensatz

3.5. Training mit vordefinierten Zeichensatz

3.5.1. Wahl des Zeichensatzes

Um das Netz zu trainieren ist eine große Zahl unterschiedlicher Bilder eines Zeichens notwendig.

Die besten Variationen bekommt man, wenn man vordefinierte Symbole von Hand nachschreibt. Im Rahmen dieser Arbeit ist das Erstellen eines Datensatzes mit handgeschriebenen Zeichen nicht möglich.

Um verschiedene Bilder von Zeichen automatisch erzeugen zu können, wurde nach Symbolen gesucht, die über einen Algorithmus angesprochen werden können.

Demzufolge enthält der Trainingsdatensatz die 26 Großbuchstaben A bis Z. Diese werden anhand ihres ASCII-Code 0x41 bis 0x5A ausgewählt.

Um die Trainingsbilder nutzen zu können, müssen diese als einen pkl.gz Datensatz abgespeichert sein.

In diesem Datensatz sind alle Bilder als Graustufenbilder ohne Transparenz in der Größe 28 × 28 Pixel abgelegt. Jedes Bild ist mit einem Label markiert. Das Label enthält Informationen über die echte Klasse des Bildes. Beim Lernprozess wird das Label mit dem Ergebnis des neuronalen Netzes verglichen, um mögliche Fehler zu entdecken. Beim Erstellen des Labels bekommt die Buchstabe A die Kennzeichnung 0 und die Buchstabe Z hat dementsprechend 25 als Bezeichner.

Die Vielfalt der Formen der Buchstaben wird durch Variationen

- der Schriftart
- der Schriftgröße
- des Winkels
- und der Neigung

beim Erstellen des Zeichens erreicht.

Wie beim Training mit dem MNIST-Datensatz werden die Daten hier auch in Trainings-, Validierungs- und Testdaten aufgeteilt.

3.5.2. Ergebnisse des ersten neuronalen Netzes

Abbildung 13 zeigt den Verlauf der Klassifikation von Großbuchstaben während des Trainings. Es fällt sofort auf, dass dieses neuronale Netz nicht mit den Netzen für die Zahlenklassifizierung mithalten kann. Es stellte sich jedoch die Frage woran es liegt.

Wären die Hyperparameter falsch, so würde die Linie bei 4% bleiben. Dennoch lernt dieses neuronale Netzwerk sehr langsam. Dies lässt darauf schließen, dass die Hyperparameter nicht falsch sind, aber sie müssen weiter optimiert werden.

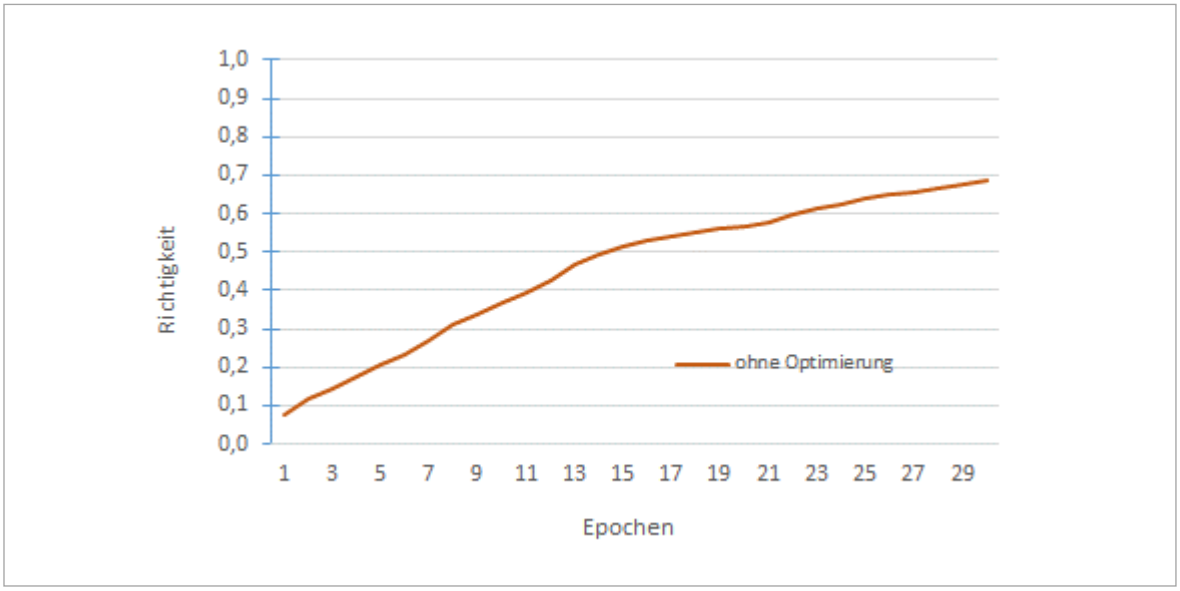


Abbildung 13: Trainingsverlauf des neuronales Netzes mit Großbuchstaben ohne Optimierung

3.5.3. Optimiertes neuronales Netz

Für dies optimierte Netzwerk wurden dieselben Neuerungen hinzugefügt, die auch das neuronale Netz zur Erkennung handschriftlicher Zahlen erweiterten. Des Weiteren wurden die Hyperparameter angepasst und schon liefert dieses Netz hervorragende Ergebnisse, siehe Abbildung 14. Es wurde eine Genauigkeit von über 99% ermittelt.

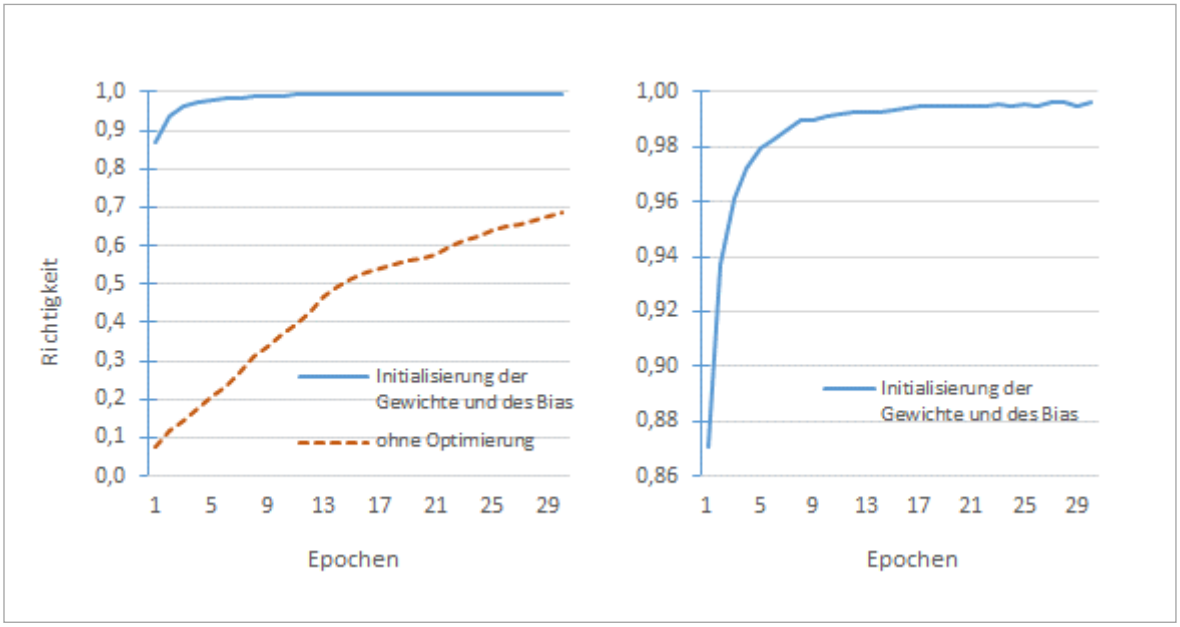


Abbildung 14: Trainingsverlauf des optimierten neuronales Netzes mit Großbuchstaben

3.5.4. Convolutional Neural Network

Für die Klassifizierung von Großbuchstaben wurde kein Convolutional Neural Network erstellt, da wir uns nicht besonders mit dieser Art von Netzwerk auseinandergesetzt haben. Aus diesem Grund war es nicht möglich, die nötigen Änderungen am Quellcode vorzunehmen.

Dennoch wird davon ausgegangen, dass man mit diesem Netzwerk fast alle Buchstaben richtig klassifizieren könnte. Somit hätte das Netzwerk wahrscheinlich eine Genauigkeit von $\sim 100\%$.

4. Erstellen von Test- und Trainingsbildern

Die Bilder werden mit Hilfe vom Microsofts Framework Windows PowerShell ISE und das Softwarepaket ImageMagick erstellt.

4.1. Windows PowerShell

Windows PowerShell ist ein plattformübergreifendes Framework von Microsoft zur Automatisierung, Konfiguration und Verwaltung von Systemen, bestehend aus einem Kommandozeileninterpreter, sowie einer Skriptsprache. [13]

PowerShell ist unter Windows standardmäßig installiert. Die verwendeten ImageMagick Kommandos werden in PowerShell ab der Version 5 unterstützt. Diese Version ist erst in Windows 10 integriert. Die PowerShell Version kann mit *get-host* ausgelesen werden.

Um Scripts unter *PowerShell* starten zu können, müssen die Sicherheitseinstellungen zur Skriptausführung auf dem System geregelt werden. Die Standardkonfiguration lässt keine Ausführung zu.

Die Richtlinie muss mit *Set-ExecutionPolicy RemoteSigned -force* nach dem Start von PowerShell mit Administratorrechten geändert werden.

Alle ImageMagick Kommandos werden in PowerShell nach dem Aufruf der Applikation mit *magick* ausgeführt.

4.2. ImageMagick

ImageMagick ist ein freies Softwarepaket zur Erstellung und Bearbeitung von Rastergrafiken. Es läuft unter die ImageMagick License und wird weltweit aktiv weiterentwickelt.

Mit den integrierten Funktionen können sowohl vorhandene Bilder bearbeitet, als auch neue einfache Formen und Schriften erzeugt werden. Filterfunktionen wie Unschärfe, Solarisation, Kontrastanpassung oder Invertierung werden ebenso unterstützt. Das Paket liefert auch eine Reihe von Kommandozeilenwerkzeugen.

Zur Bilderstellung und Bearbeitung wird die aktuelle Version ImageMagick 7.0.5 verwendet.

4.3. Auswahl der Schriftarten

Als Quelle für die Buchstaben werden die unter Windows installierten Schriften benutzt.

Diese können mittels des Kommandos *magick convert -list font* in der PowerShell Konsole ausgegeben werden. Der Befehl liefert die Schrifteigenschaften font, family, style, stretch, weight und glyphs (Abbildung 15).

```
Font: Franklin-Gothic-Heavy-Italic
   family: Franklin Gothic
   style: Italic
   stretch: Normal
   weight: 900
   glyphs: c:\windows\fonts\frahvit.ttf
```

Abbildung 15: Ausgabe der Schrifteigenschaften mit magick convert –list font

Über diese Informationen kann ein Filter gelegt werden. Zum Trainieren des neuronalen Netzes sind extreme Abweichungen unerwünscht. So wird auf fette und kursive Schriften verzichtet. Die Namen der gefilterten Schriften werden in einer Textdatei gespeichert (Abbildung 16).

```
magick convert -list font |
   Select-String -pattern "font:", "weight: 400", "stretch: Normal" |
   Where-Object { $_ -NotMatch "Wing*"}|
   Where-Object { $_ -NotMatch "Web*"} |
   Where-Object { $_ -NotMatch "Bold*"} |
   Select-String -pattern "font: " |
   foreach{$_.line.substring(8)}
   > fonts.txt
```

Abbildung 16: Filtern der Schriftarten anhand ihrer Eigenschaften

Nicht alle fetten, dekorativen und symbolischen Schriften können über die ausgelesenen Eigenschaften von der Liste entfernt werden. Bekannte Kennungen werden auch im Schriftnamen gesucht.

Da das Schrifteigenschaft Symbol von ImageMagick nicht gelesen werden kann, muss die Liste mit den Schriftarten noch manuell überprüft werden. Die begrenzte Anzahl

symbolischer Schriftarten kann mithilfe MainType Free Edition Version 7 ermittelt werden. Hier können die Schriftarten nach der Eigenschaft Symbol sortiert werden. Die symbolischen Schriftarten werden aus der Liste mit den Schriften entfernt.

4.4. Erstellen von Bildern mit ImageMagick

ImageMagick bietet die Möglichkeit ein Bild zu erstellen und zu speichern. Dabei kann die Größe des Bildes *-size* in Pixel, die Hintergrundfarbe *xc* und der Speicherort angegeben werden (Abbildung 17). Der Farbraum wird je nach Bildinhalt automatisch bestimmt. Dieser kann mit einem externen Tool geprüft werden. Das Farbmodell und die Farbtiefe in Bits werden zum Beispiel in XnView bei den Bildeigenschaften dargestellt. Bei XnView handelt es sich um eine Freeware, die Grafiken anzeigen und konvertieren kann.

```
magick convert -size 28x28 xc:black -colorspace Gray ".\bild.png"
```

Abbildung 17: Erstellen und Speichern eines Bilds

Das Farbmodell des Bildes kann mit dem Parameter *-colorspace* festgelegt werden. Die für die Graustufenbilder typische Farbtiefe von 8 Bit wird mit *-colorspace Gray* definiert.

4.5. Bild mit einem Zeichen erstellen

Auf einem vordefinierten Bild können zusätzlich Labels mit beliebigem Text hinzugefügt werden. Dafür sind folgende Angaben erforderlich (Abbildung 18):

- Schrift *-font*
- die Schriftgröße *-pointsize*
- die Schriftfarbe *-fill*
- Textausrichtung *-gravity*
- Textinhalt *-label*

Das Nutzen der Option *label* setzt das Erstellen eines Bildes voraus.

```
magick convert -size 28x28 -background black -font Arial -fill white `
   -pointsize 22 -gravity center label:M -colorspace Gray ".\label.png"
```

Abbildung 18: Erstellen vom Bild mit Label

4.6. Drehen des Symbols in einem Bild

In einem erstellten Bild kann nachträglich ein Label mit Text hinzugefügt werden. Beim Einfügen des Zeichens besteht die Möglichkeit einen Drehwinkel über die Option *-annotate* zu definieren (Abbildung 19). Die Option kann nur benutzt werden, wenn ein vorhandenes Bild zum Bearbeiten geladen wird.

```
magick convert bild.png -font Arial -fill white -pointsize 22 `
   -gravity center -annotate 45 M -colorspace Gray ".\ labelTranslate.png"
```

Abbildung 19: Drehen des Labels

4.7. Verzerren des Symbols in einem Bild

Die *-annotate* Option dient zur besseren Positionierung des Textes und besitzt mehrere Freiheitsgrade. Über einen x- und y-Winkel wird die Neigung des Zeichens genau vorgegeben. Dies kann mit einer kursiven Formatierung eines Textes verglichen werden. Die genaue Richtung und Stärke wird dabei vom Benutzer definiert. Mit *-annotate 45x45x-2+2 M* wird des Buchstabe M mit einem x- und y-Winkel von 45° abgebildet. Das Zeichen ist, vom Mittelpunkt *-gravity center* gesehen, um 2 Punkt in y- und um -2 Punkte in x-Richtung verschoben (Abbildung 20).

```
magick convert bild.png -font Arial -fill white -pointsize 22 `
   -gravity center -annotate 45x45x-2+2 M -colorspace Gray `
   ".\labelDeform.png"
```

Abbildung 20: Verzerren des Zeichens in einem Label

4.8. Ändern der Schriftstärke

Die Schriftstärke kann mit ImageMagick nicht festgelegt werden. Diese lässt sich jedoch indirekt über die Option *-stroke* beeinflussen. Die Ränder vom Zeichen können mit einer vordefinierten Farbe und Stärke *-strokewidth* überzeichnet werden. Wenn die Zeichenränder mit der Schriftfarbe überzeichnet werden, wirkt die Schrift fetter. Wird zum Überzeichnen die Hintergrundfarbe benutzt, entsteht ein Schatten des Buchstaben (Abbildung 21). Es ist drauf zu achten, dass bei kleinen Bildern, wie 28 × 28 Pixel, die Änderungen in der Schriftstärke, bedingt durch die kleine Pixelzahl, entweder keine oder sehr große Wirkung haben.

```
magick convert bild.png -font Arial -fill white -pointsize 22 `
   -gravity center -stroke white -strokewidth 1 -annotate 0 M `
   -colorspace Gray ".\labelBold.png"
```

Abbildung 21: Schriftweite durch Überzeichnen der Ränder

4.9. Erzeugen von Unschärfe beim Symbolen

Um eine Unschärfe in den Zeichen zu erzeugen, gibt es die Option *-blur*. Sie definiert die Unschärfe in x- und y-Richtung (Abbildung 22). Bei kleinen Bildern gilt auch hier die Regel: kleine Änderungen haben bei einer geringen Pixelzahl eine große Auswirkung.

```
magick convert -size 28x28 -background black  -font Arial -fill white `
   -pointsize 22 -gravity center label:M -blur 1x1 -colorspace Gray `
   ".\labelSoft.png"
```

Abbildung 22: Unschärfe bei Symbolen

4.10. Trainingsbildern

Zum Erzeugen der Trainingsbilder als Graustufenbilder mit der Größe 28 × 28 Pixel werden 147 Schriftarten benutzt. Jeder Buchstabe mit ASCII-Code zwischen 0x41 und 0x5A (A bis Z) wird für zwei gewählte Schriftgrößen in mehreren Schritten gedreht und verzerrt, sodass 22 verschiedene Bilder pro Buchstabe und Schriftart entstehen.

Der Datensatz besteht dementsprechend aus 84 084 Bilder. Auf Änderungen der Schriftstärke und Unschärfe wird verzichtet, da das Trainieren des neuronalen Netzes mit dem Datensatz, je nach Rechnerleistung, mehrere Stunden in Anspruch nehmen kann. Die Trainingszeit bei gleicher Optimierungsstufe des neuronalen Netzes ist auch von der Anzahl der Klassen im Netz abhängig.

Die Dateinamen bestehen aus dem Buchstaben, sowie einer Zahl, um ein Überschreiben bereits gespeicherter Buchstaben zu vermeiden. Durch den Buchstaben im Dateinamen wird das Label des Bildes festgelegt. Ein Label hat ein Wert zwischen 0 und 25. Diese Werte repräsentieren die Buchstaben A bis Z.

4.11. Testbilder

Um das trainierte neuronale Netz zu testen, werden zwei Arten von Testdaten eingesetzt. Als Erstes werden zum Erkennen Bilder mit Buchstaben aus verschiedenen nicht verwendeten Schriftarten benutzt. Die Bilder werden wie die Trainingsbilder erzeugt.

Weitere Testdaten sind gescannte handgeschriebene farbige Großbuchstaben auf weißem Hintergrund.

Nach dem Einscannen werden die Buchstaben mithilfe von GIMP (GNU Image Manipulation Program) zentriert und als einzelne Bilder gespeichert. Die Bilder haben eine 32 Bit Farbtiefe – RGB-Kanäle und einen Alphakanal für die Transparenz.

Um ein Graustufenbild mit 28 × 28 Pixel für das Testen zu erstellen, werden die Bilder mit ImageMagick bearbeitet.

Folgende Schritte sind notwendig (Abbildung 23):

- Konvertierung in 24 Bit durch Entfernen des Alphakanals *-alpha off*
- Invertieren des Bildes *-negate*
- Anpassung der Helligkeit und Kontrast *-brightness-contrast*
- Ändern des Farbmodells in 8 Bit Graustufen *-colorspace Gray*
- Skalieren des Bildes auf 28 Pixel *-resize*

```
magick convert r32.png -alpha off -negate `
   -brightness-contrast 30x100 -colorspace Gray -resize 28 `
   ".\r28x28.png"
```

Abbildung 23: Testbild aus einem gescannten handgeschriebenen Buchstabe

5. Test des neuronalen Netzes

Die Testbilder werden mithilfe von *minst_creator.py* als pkl.gz-Datensatzes gespeichert. Zum Testen werden mit *visualize.py* die Testdaten dem gewählten neuronalen Netz zum Evaluieren weitergegeben.

Das Ergebnis der Erkennung wird visualisiert. Dabei wird das geladene Testzeichen mit seinem Label angezeigt. Zusätzlich ist das Label der geschätzten Klasse aus dem neuronalen Netz als *Guess* ausgegeben (Abbildung 24).

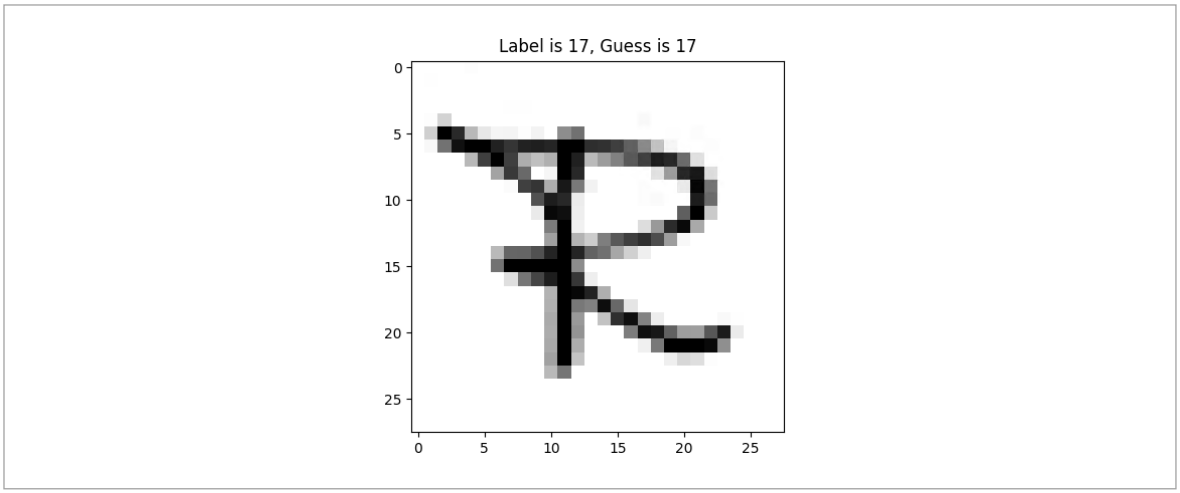


Abbildung 24: Visualisierung der Erkennung eines handgeschriebenen Zeichens

Beim Testen werden über die Konfigurationsdatei *config.py* ein angelerntes neuronales Netz und die pkl.gz-Daten ausgewählt.

Bei acht handgeschriebenen Buchstaben werden vom neuronalen Netz ohne Optimierung vier richtig erkannt. In den optimierten neuronalen Netzen werden dagegen alle Testdaten richtig erkannt. Ein erneutes Trainieren der neuronalen Netze kann zu einer anderen Erkennungsquote führen. Sollte das Erkennen der Testdaten unabhängig von dem Anlernprozess sein, kann dies ein Zeichen für Overfitting sein. Dabei lernt das Netz, statt allgemeiner Konzepte lediglich die Trainingsdaten auswendig.

6. Fazit und Ausblick

Das Projekt wurde erfolgreich bearbeitet. Es wurde ein einfaches, künstliches neuronales Netz in Python entwickelt. Dadurch erlernte man die Grundkenntnisse, wie ein neuronales Netz funktioniert, lernt und klassifiziert.

Durch die gewonnenen Erkenntnisse was es möglich, das neuronale Netz mit Neuerungen, wie z.B. einer besseren Initialisierung der Gewichte und einer neuen Kostenfunktion, das bestehende System zu verbessern.

Es gab zudem einen kurzen Einblick in die Welt der Convolutional Neural Networks. Diese Art der neuronalen Netze sind „State-of-the-Art“ und werden von vielen großen IT Firmen verwendet und weiterentwickelt.

Zum Schluss wurde ein Datensatz mit eigens erzeugten maschinellen Buchstaben generiert. Die neuronalen Netze wurden mit diesen Buchstaben trainiert. Überraschend war hierbei, dass das neuronale Netz nicht nur die Testbilder erkannte, sondern auch handgeschriebene Buchstaben konnte es ohne Probleme erkennen. Letzteres ist nur eine Annahme, da die Anzahl der handgeschriebenen Buchstaben sehr klein war.

Dennoch war es schade, dass keine kompatible Grafikkarte zur Verfügung stand, da man ansonsten deutlich komplexere neuronale Netze erstellen hätte können.

Heutzutage spezialisieren sich neuronale Netze noch auf spezielle Probleme. Aber vor 50 Jahren konnte man sich noch nicht mal vorstellen, dass ein Computer zu so etwas Komplexem in der Lage sei. Also wer weiß, wie es in 50 Jahren aussieht. Bei der rasanten Entwicklung der Technologie würde es uns nicht wundern, wenn dann neuronale Netze existieren, die nicht mehr nur auf ein spezielles Problem trainiert wurden.

Abkürzungsverzeichnis

A

AMLS *Azure Machine Learning Studio*

M

MSE *Mittlere Quadratische Abweichung (en. Mean Squared Error)*

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Abbildungsverzeichnis

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Formelverzeichnis

Literaturverzeichnis

[1] D. Petereit, „T3N Digital Pioneers," 17 12 2016. [Online]. Available: http://t3n.de/news/ai-machine-learning-nlp-deep-learning-776907/. [Zugriff am 10 7 2017].

[2] J. Han, J. Pei und M. Kamber, Data Mining: Concepts and Techniques, 3rd Hrsg., Elsevier, 2011, pp. 1-2.

[3] L. Jacobson, „Introduction to Artificial Neural Networks - Part 1," 5 12 2013. [Online]. Available: http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7. [Zugriff am 26 8 2016].

[4] V. Failli, V. May und R. Lederer, „Wie funktioniert ein Neuron?," 13 11 2013. [Online]. Available: www.wingsforlife.com/de/aktuelles/wie-funktioniert-ein-neuron-561/. [Zugriff am 23 8 2016].

[5] N. Hezel, „KNN: Vorteile und Nachteile," 12 9 2015. [Online]. Available: https://www.data-science-blog.com/blog/2015/09/12/knn-vorteile-und-nacheile/. [Zugriff am 23 8 2016].

[6] F. Rosenblatt, „The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," Psychological Review, 1958, pp. 386-408.

[7] M. Nielsen, „Neural Networks and Deep Learning," 5 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html. [Zugriff am 10 7 2017].

[8] D. Rumelhart, G. Hinton und R. Williams, „Learning representations by back-propagation errors," Nature, 1986, pp. 533-536.

[9] NVIDIA, „Quadro Grafikkarten vergleichen und kaufen | NVIDIA,“ NVIDIA, [Online]. Available: http://www.nvidia.de/object/buy-quadro-graphics-cards-de.html. [Zugriff am 09 07 2017].

[10] Microsoft, „Was ist Azure Machine Learning Studio?,“ Microsoft, [Online]. Available: https://docs.microsoft.com/de-de/azure/machine-learning/machine-learning-what-is-ml-studio. [Zugriff am 09 07 2017].

[11] „Unsupervised Feature Learning and Deep Learning Tutorial,“ Stanford University, [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. [Zugriff am 10 7 2017].

[12] c’t Magazin für Computertechnik, „Ziffernlerner,“ *c’t Magazin für Computertechnik,* p. 143, 06 2016.

[13] PowerShell, „PowerShell/README.md at master · PowerShell/PowerShell · GitHub,“ 06 07 2017. [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/README.md.

[14] A. Karpathy, „CS231n Convolutional Neural Networks for Visual Recognition,“ 2016. [Online]. Available: http://cs231n.github.io/neural-networks-1/#nn. [Zugriff am 23 8 2016].

[15] *Kernel convolution.* [Art]. 2016.

[16] Aphex34, Artist, *Max Pooling.* [Art]. 2015.