|  |  |
| --- | --- |
|  |  |
|  | Dokumentation zu einem neuronalen Netz zur Erkennung von handschriftlichen Ziffern |
|  | ­ |
|  |  |
|  | Luca Glaser, Technische Universität Darmstadt  MTS23  06.06.2024 |
|  |  |





# Inhaltsverzeichnis

1. Aufgabenstellung 1

2. Allgemeines 2

3. Aktivierungsfunktion 3

4. Fehlerrückführung und Gewichtsanpassung 5

5. Benutzerhandbuch 6

6. Entwicklung des neuronalen Netzes 7

7. Quellenverzeichnis 8

8. Abbildungsverzeichnis 9

# Aufgabenstellung

Die Aufgabe ist es, ein neuronales Netz selbstständig zu programmieren und dabei die zuvor programmierte Matrizenmultiplikation miteinzubinden. Das neuronale Netz soll handgeschriebene Ziffern von Null bis Neun erkennen können. Hierzu wurden Datensätze bereitgestellt, mit denen das neuronale Netz trainiert und getestet wird. Zusätzlich soll implementiert werden, dass selbstgeschriebene Ziffern von dem neuronalen Netz erkannt werden können. Die Datensätze bestehen aus 60.000 Trainingsbildern, die jeweils 28x28 Pixel groß sind, mit den dazugehörigen Labels, die sagen welche Zahl auf dem jeweiligen Bild abgebildet ist. Diese Labels sind essenziell für ein neuronales Netz, um zu erkennen, ob das Netz eine Zahl richtig erkennt oder nicht. Dazu kommen 10.000 Testbilder mit den dazugehörigen Labels, um zu testen, wie genau das neuronale Netz die Zahlen erkennen kann. Zudem wird nur bei den 60.000 Trainingsbildern eine Fehlerrückberechnung durchgeführt und die Gewichte daraufhin angepasst. Die Fehlerrückberechnung wird nicht bei den Testbildern durchgeführt, da sonst die Auswertung der Genauigkeit anhand dieser Bilder verfälscht werden würde.



Abbildung 1: Beispiel wie die handgeschriebenen Ziffern aussehen

Knoten / Neuron

Gewicht

Eingabe-Schicht

Ausgabe-Schicht

Zwischenschicht

Abbildung 2: Beispiel wie das neuronale Netz aussieht

# Allgemeines

Bei einem neuronalen Netz geht es darum, das menschliche Gehirn mithilfe von Schichten, Knoten und Gewichten, so gut es geht nachzuahmen. Hierbei hat ein neuronales Netz immer mindestens zwei Schichten. Diese sind die Eingabe- und die Ausgabe-Schicht. In der Eingabe-Schicht werden sämtliche Informationen in die einzelnen Knoten, die den Neuronen in einem menschlichen Gehirn nachempfunden sind, aufgeteilt.

Bezogen auf die Aufgabe hat die Eingabe-Schicht für jeden Pixel ein Knoten und somit 28x28=784 Knoten. Jeder dieser Pixel hat einen Grauwert zwischen 0 und 255. Um die Berechnungen im Folgenden zu vereinfachen, werden diese Werte normalisiert. Also wird jeder dieser Werte durch 255 geteilt, damit alle Werte in dem neuronalen Netz zwischen Null und Eins bleiben.

Die Ausgabe-Schicht hat so viele Knoten wie es Ergebnisse geben kann. In diesem Fall können Zehn verschiedene Ergebnisse herauskommen, und zwar die Ziffern von Null bis Neun. Der Wert, der in den einzelnen Knoten der Ausgabe-Schicht herauskommt, repräsentiert die Wahrscheinlichkeit, dass die jeweilige Zahl auf dem Bild zu erkennen ist.

Zu diesen zwei Schichten können jedoch noch beliebig viele Schichten dazwischengesetzt werden.

Es gibt verschiedene Arten von Schichten, die zwischen die Eingabe- und Ausgabe- Schicht gesetzt werden, können. Einige davon wären:

* Polling Layer
* Convolutional Layer
* Dropout Layer
* Batch Layer
* Fully Connected Layer

Jede dieser verschiedenen Schichten erfüllen eine andere Aufgabe, um das neuronale Netz entweder zu beschleunigen, Werte zusammen zu fassen oder die Genauigkeit des Netzes zu erhöhen.

Für diese Aufgabe wird nur ein Fully Connected Layer dazwischengesetzt. Dieser Fully Connected Layer wird im Folgenden als Zwischenschicht betitelt und erhält 100 Knoten. Diese Zwischenschicht ist so konzipiert, dass sie eine beliebige Anzahl an Knoten bekommen kann, zwischen der Anzahl an Knoten der vorherigen Schicht und der Anzahl an Knoten der nachfolgenden Schicht. Bei der Anzahl der Knoten für die Zwischenschicht sollte jedoch darauf geachtet werden, wie viele Knoten diese erhält. Ein guter Ansatzpunkt ist, die Quadratwurzel der Anzahl der Knoten der vorherigen Schicht zu nehmen und sich daran zu orientieren. Somit sollen mehr Gewichte eingebracht werden, um die Genauigkeit des neuronalen Netzes zu erhöhen.

Zu den Schichten und Knoten gibt es noch die Gewichte. Diese sind so gewählt, dass jeder Knoten einer Schicht Auswirkungen auf jeden Knoten der kommenden Schicht hat. Jede dieser Verbindungen hat eine eigene Gewichtung. Bei dieser Aufgabe wären es also einmal 784x100 Gewichte für die Verbindungen zwischen der Eingabe-Schicht und der Zwischenschicht, und einmal 100x10 Gewichte für die Verbindungen zwischen der Zwischenschicht und der Ausgabe-Schicht. Jede Gewichtung muss initialisiert werden. Dafür gibt es ebenfalls verschiedene Herangehensweisen.

Eine der Herangehensweise wäre dasselbe Gewicht für alle Gewichtungen zu wählen. Das wäre jedoch unpraktikabel, da hierbei davon ausgegangen werden kann, dass die Gewichte in ein lokales Minimum abrutschen und nichtmehr in das globale Minimum kommen. Somit wird die Genauigkeit des Netzes darunter leiden.

Abbildung 3: Beispiel für lokales und globales Minimum

Anhand der Abbildung 3 kann gesehen werden, dass bei x=1 ein lokales Minimum ist. Das bedeutet, dass an diesem Punkt zwar ein Minimum ist, aber es gibt ein weiteres Minimum, dass tiefer liegt. In diesem Beispiel ist somit an dem Punkt x=3 das globale Minimum.

Eine weitere Möglichkeit ist alle Gewichte zufällig in dem Intervall (-1; 1) zu initialisieren. Hierbei wird eine zufriedenstellende Lösung herauskommen, jedoch gibt es eine noch praktikablere Möglichkeit.

Diese ist die Xavier-Initialisierung. Die Xavier-Initialisierung besagt, dass ein Intervall, aus dem zufällig ausgewählte Werte entnommen werden, weiter eingeschränkt wird. Die Einschränkung beruht auf der Anzahl an Knoten der kommenden Schicht. Bezogen auf das Beispiel ist das Intervall, für die erste Gewichtsmatrix von 784x100, ((); ()) und für die zweite Gewichtsmatrix von 100x10 ist das Intervall ((); ()). Mit dieser Initialisierung sind die Gewichte an geeigneten Startwerten, um mit dem Training des neuronalen Netzes zu starten. Die Gewichte werden für die Aufgabe in eine Textdatei geschrieben, um die Gewichtsanpassungen über die Epochen abzuspeichern.

# Aktivierungsfunktion

Die Aktivierungsfunktion ist darauf aus, dass manche Reize, die von einem Menschen oder einem Lebewesen aufgenommen werden, so gering sind, dass diese gar nicht erst weitergeleitet werden. Dafür müssen diese vorher einen gewissen Schwellenwert überschreiten damit das Neuron diese Eingabe überhaupt weiterleitet. Es ist jedoch sehr ineffizient nach genau dieser Herangehensweise vorzugehen, da es somit nur danach bewertet wird, ob es weitergeleitet wird oder nicht. Es gibt die verschiedensten Aktivierungsfunktionen, die bei einem neuronalen Netz angewendet werden können. Die zuvor erwähnte Funktion ist eine Stufenfunktion, die so aussieht, dass bis zu einem gewissen Wert nichts weitergeleitet wird und alles ab diesem Wert, mit dem Wert 1, weitergeleitet wird.

Eine weiter Aktivierungsfunktion, die weit verbreitet ist und in einigen neuronalen Netzen zum Einsatz kommt ist die ReLU-Funktion (Rectified Linear Unit). Bei dieser Aktivierungsfunktion werden alle negativen Werte mit 0 weiterleitet und alle positiven Werte ohne Veränderung weitergeleitet.

Abbildung 4: ReLU-Funktion

Dazu gibt es noch die Sigmoid-Funktion, die bei diesem neuronalen Netz genutzt wird. Die Sigmoid-Funktion sieht wie folgt aus:

Bei der Sigmoid-Funktion wird selbst der kleinste Reiz weitergeleitet, jedoch mit einem angepassten Wert. Somit überträgt die Sigmoid-Funktion alle Werte aus dem ℝ zwischen 0 und 1.

Abbildung 5: Sigmoid-Funktion

# Fehlerrückführung und Gewichtsanpassung

Dadurch, dass es, trotz Xavier-Initialisierung, unendlich viele Möglichkeiten für jedes einzelne Gewicht gibt, ist die Wahrscheinlichkeit direkt alle genau richtig initialisiert zu haben gleich Null. Um die Genauigkeit des neuronalen Netzes nun zu verbessern, werden die Trainingsbilder, eines nach dem anderen, durch das Netz geführt und in der Ausgabe-Schicht geschaut was hierbei herauskommt. Ein perfektes neuronales Netz sollte bei der Zahl, die auf dem Bild geschrieben ist, eine Eins ausgeben und bei allen anderen eine Null. Aber diese Art von Perfektion zu erreichen ist nahezu unmöglich. Jedoch kann sich dieser Perfektion angenähert werden. Diese Annäherung passiert in verschiedenen Schritten und wird Fehlerrückführung oder Backpropagation genannt. Das Ziel der Annäherung ist es die Gewichte anzupassen.

Um die Gewichte anzupassen, wird eine Matrix benötigt, die angibt, um welchen Wert die einzelnen Gewichte angepasst werden müssen. Um auf diese Änderungen zu kommen wird die folgende Formel benötigt:

Änderung an der Gewichtsmatrix zwischen der vorherigen Schicht j und der nachfolgenden Schicht k

Modifizierung der Änderung (Lernrate)

Fehler der nachfolgenden Schicht k

Werte der nachfolgenden Schicht k

Werte der vorherigen Schicht j transponiert

Die Lernrate muss selbst gewählt werden. Hierbei muss vorsichtig vorgegangen werden. Wenn die Lernrate zu hoch ist kann es passierten, dass die Gewichte über das globale Minimum drüber springen. Ist die Lernrate jedoch zu klein kann es passieren, dass die Gewichte in ein lokales Minimum rutschen. Das Beste ist es also die Lernrate für die ersten Durchgänge etwas höher anzusetzen. Ein Wert um 0,1 ist zu empfehlen. Sobald das neuronale Netz eine hohe Genauigkeit von 80-90% hat, sollte die Lernrate auf 0,05-0,01 reduziert werden. Bei einer Genauigkeit von 90% oder mehr sollte die Lernrate noch geringer gesetzt werden.

Die Fehler der nachfolgenden Schicht k () müssen ermittelt werden. Für die Gewichtsmatrix zwischen der Zwischenschicht und der Ausgabeschicht können die Fehler schnell ermittelt werden. Es werden die Werte aus der Ausgabeschicht genommen und elementweise mit den Werten subtrahiert, die in der Ausgabeschicht zu erwarten waren.

Für die Gewichtsmatrix zwischen der Eingabeschicht und der Zwischenschicht ist das etwas komplexer. Es werden die Fehler einem nach dem anderen, aus der Ausgabeschicht genommen. Beginnend mit dem ersten Fehler wird dieser über die Gewichte auf die vorherige Schicht zurückgeführt.

= 0,8

= 0,6

Abbildung 6.1: Beispiel für die Fehlerrückführung

=

Abbildung 6.2: Gewichte für das Beispiel der Fehlerrückführung

Wird der Fehler genommen, wird dieser mit einen der Gewichte, die auf den dazugehörigen Knoten zeigen multipliziert und durch alle Gewichte, die auf diesen Knoten zeigen, zusammen summiert dividiert.

Abbildung 6.3: Formel, um ein Gewicht aus der Ausgabeschicht auf einen Knoten der Zwischenschicht zurückzuführen

Der Fehler eines Knotens der Zwischenschicht ist der aufsummierte Fehler aller auf diesen Knoten zurückgeführten Fehler der Ausgabeschicht.

Abbildung 6.4: Formel zur Berechnung eines Fehlers der Zwischenschicht mit eingesetzten Werten aus dem Beispiel

Diese Fehlerrückführung kann jedoch mithilfe einer Matrixmultiplikation errechnet werden. Dafür wird die Gewichtsmatrix zwischen der Zwischenschicht und der Ausgabeschicht genommen, transponiert und mit den Fehlern der Ausgabe-Schicht multipliziert.

Somit sind alle Variablen, die für das Ermitteln der Änderung an der Gewichtsmatrix benötigt werden, verfügbar. Die Lernrate wurde ausgewählt, die Fehler der nachfolgenden Schicht wurden ermittelt, die Werte der nachfolgenden Schicht wurde bereits bei der Vorwärtsrechnung errechnet und die Werte der vorherigen Schicht stehen ebenfalls schon zur Verfügung und müssen nur noch transponiert werden.

Die Fehler der Eingabeschicht müssen nicht ermittelt werden, da dies bedeuten würde, dass die Eingabewerte fehlerhaft sein. Es sollte davon auszugehen sein, dass die Eingabewerte fehlerfrei und korrekt sind.

Sobald die Änderung der Gewichtsmatrix ermittelt wurde, wird diese elementweise von der Gewichtsmatrix subtrahiert. Somit wurden die Gewichte angepasst und es kann das nächste Bild betrachtet werden.

# Benutzerhandbuch

Zum Ausführen des Programms werden die Bibliotheken „mnist“ und „PyQt5“ benötigt. Diese können mit den Befehlen „pip install python-mnist“ und „pip install PyQt5“ installiert werden. Mit den vier Konstanten „TRAIN“, „TEST“, „EIGENEZIFFERN“ und „GEWICHTENEUINITIALISIEREN“ können die Aufgaben des Netzes angegeben werden. Ist TRAIN auf True gesetzt werden die 60.000 Trainingsbilder durchgegangen und eine Gewichtsanpassung durchgeführt. Ist TEST auf True gesetzt werden die 10.000 Testbilder durchgegangen und am Ende ausgegeben, wie hoch die Anzahl an korrekt erkannten Ziffern ist, in Prozent. Ist EIGENEZIFFERN auf True gesetzt wird ein Fenster aufgerufen, indem eine Ziffer geschrieben werden soll. Dieses Fenster wird abgespeichert und durch das neuronale Netz gejagt, um zu untersuchen welche Zahl geschrieben wurde. Die Zahl, die erkannt wird, wird im Terminal ausgegeben. In dem Fenster gibt es ein Knopf mit der Beschriftung „save and predict“. Durch das Drücken dieses Knopfes wird das aktuelle Bild abgespeichert und untersucht. Neben diesem Knopf gibt es noch den „clear“-Knopf. Beim Drücken dieses Knopfes wird das Fenster auf das leere Fenster vom Beginn zurückgesetzt. Ist GEWICHTENEUINITIALISIEREN auf True gesetzt werden die Gewichte der beiden Gewichtsmatrizen mit der Xavier-Initialisierung neu initialisiert. Sind Mehrere dieser Konstanten auf True gesetzt werden die Aktionen in der Reihenfolge folgenden Reihenfolge ausgeführt: Gewichte neu initialisieren -> Neuronales Netz trainieren -> Neuronales Netz testen -> Eigene Ziffern untersuchen lassen.

Das neu initialisieren der Gewichte ist in unter einer Sekunde durchgeführt. Das Trainieren mit den 60.000 Bilder dauert im Schnitt 30 bis 35 Minuten. Das Testen mit den 10.000 Bildern dauert zwischen einer und drei Minuten. Die Dauer für das Untersuchen der eigenen Ziffern ist davon abhängig wie viele Ziffern untersucht werden sollen. Das Untersuchen für eine eigene Ziffer ist in unter einer Sekunde durchgeführt.

# Entwicklung des neuronalen Netzes

Die Genauigkeit des neuronalen Netzes wird daraus berechnet, wie viele der 10.000 Testbilder, es korrekt erkennen kann.

# Quellenverzeichnis

Wuttke, Laurenz: Künstliche neuronale Netzwerke: Definition, Einführung, Arten und Funktion, In: datasolut.com. <https://datasolut.com/neuronale-netzwerke-einfuehrung/> 06.06.2024

Unbekannt: Schichten eines neuronalen Netzes, In: alex-ta.gitbooks.io. <https://alex-ta.gitbooks.io/einfuhrung-in-neuronale-netze/content/schichten-eines-neuronalen-netzes.html> 06.06.2024

Unbekannt: Was ist die ReLU-Funktion (Rectified Linear Unit), In: databasecamp.de. [https://databasecamp.de/ki/relu 06.06.2024](https://databasecamp.de/ki/relu%2006.06.2024)

Rashid, Tariq: Neuronale Netze selbst programmieren. Ein verständlicher Einstieg mit Python. O’REILLY-Verlag, 2017

# Abbildungsverzeichnis

Abbildung 1: [digits\_recognition\_cnn.ipynb - Colab (google.com)](https://colab.research.google.com/github/trekhleb/machine-learning-experiments/blob/master/experiments/digits_recognition_cnn/digits_recognition_cnn.ipynb#scrollTo=eRAjNPs3jQFD)

Abbildung 2: Eigene Darstellung

Abbildung 3: Eigene Darstellung

Abbildung 4: Eigene Darstellung

Abbildung 5: Eigene Darstellung

Abbildung 6.1: Eigene Darstellung

Abbildung 6.2: Eigene Darstellung

Abbildung 6.3: Eigene Darstellung

Abbildung 6.4: Eigene Darstellung