Technische Universität Dortmund

Fakultät Informatik
Design Automation for Embedded Systems Group

TODO Titel

Fachprojekt

Jack Diep, Florian Köhler, Yannick Naumann

Betreuung:
M.Sc. Mikail Yayla

31. August 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Neuronale Netzwerke bilden eine Unterkategorie des Machine-Learning und erlauben Auswertungen von Eingaben auf Basis von zuvor angelernten, empirischen Ergebnissen. Ein neuronales Netzwerk bildet ein System aus Neuronen ab, welche Schichtweise verbunden sind einen unidirektionalen Datenfluss erzeugen. Dieses System besteht üblicherweise aus einer Eingabeschicht, einer Ausgabeschicht und dazwischen beliebig viele *versteckte* Schichten, welche die eigentliche Arbeit des Netzwerks verrichten. Die Anzahl der Neuronen in den Eingabe- und Ausgabeschichten ist intuitiv wählbar. Die Größe der Eingabeschicht wird häufig durch die Anzahl der möglichen Eingaben bestimmt, die Größe der Ausgabeschicht durch die Anzahl der möglichen Ergebnisse. Die Größe und Anzahl der dazwischen liegenden Schichten hingegen muss je nach Anforderung und Gegebenheiten individuell ermittelt werden. Je größer das Netzwerk desto höher sind die Anforderungen an die benötigte Hardware um dieses zu betreiben. Bei schwächerer Hardware oder Einschränkungen bezüglich der Energieversorgung können kleinere Netzwerke eingesetzt werden, wenn auch häufig mit geringerer Genauigkeit verglichen mit einem größeren Netzwerk.

1.1 Motivation

Die Größe eines Netzwerks kann beim Entwurf dessen direkt beeinflusst werden. In Anwendungsgebieten, bei denen der Fokus auf geringen Hardwareanforderungen liegt, stoßen schnell das Problem der schwindenden Genauigkeit. Auf IoT-Geräten oder mobilen Platformen finden klassische neuronale Netzwerke daher nur eingeschränkt Nutzen.

2016 veröffentlichten M. Courbariaux und Y. Bengio [1] eine wissenschaftliche Arbeit und stellen dort das *binarisierte neuronale Netzwerk* (kurz BNN) vor. Dieses könne im Vergleich zu einem klassischen neuronalen Netzwerk eine theoretische Geschwindigkeitssteigerung auf das 32-fache erreichen. Die Genauigkeit des BNN liege jedoch nur knapp unter derer klassischer Netzwerke. Das BNN ermöglicht dank dieser Eigenschaften den Einsatz von neuronalen Netzen auf vergleichsweise schwacher Hardware und verspricht zugleich nur geringe Genauigkeitseinbüße.

In dieser Ausarbeitung wird die allgemeine Funktionsweise von neuronalen Netzwerken erläutert und anschließend der Entwurf eines binarisiertes Netzwerk mit Fokus auf Handschriftenerkennung auf Basis des MNIST Datensatzes dokumentiert. Dabei werden grundlegende Überlegungen, das Vorgehen, Herausforderungen sowie dazu erarbeitete Lösungen vorgestellt.

Kapitel 2

Neuronale Netzwerke

Unter einem neuronalen Netzwerk versteht man ein System aus Neuronen. Diese sind schichtweise organisiert wobei jedes Neuron einer Schicht jeweils zu allen Neuronen der direkt anliegenden Schichten verbunden ist. Abbildung 2.1 zeigt beispielhaft ein neuronales Netzwerk aus 19 Neuronen mit insgesamt 5 Schichten.

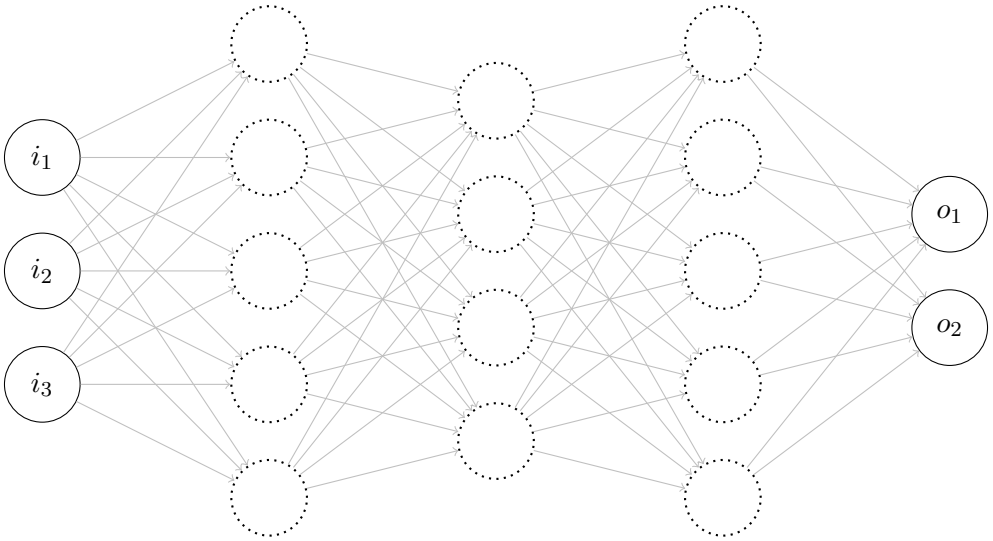


Abbildung 2.1: Struktur eines neuronalen Netzwerks

2.1 Neuronen

Unter einem Neuron versteht sich bei neuronalen Netzen lediglich ein Knoten, in dem üblicherweise ein 32-bit großer Wert hinterlegt ist. Häufig kommen hier Gleitkommazahlen zwischen -1.0 und 1.0 zum Einsatz, da sich dieser Wertebereich besonders gut zum Rechnen eignet und Eigenschaften bezüglich der Multiplikation besitzt, die eine Wertexplosion verhindern. Die Werte aller Neuronen, mit Ausnahme derer in der Eingabeschicht, setzen sich jeweils aus den Werten aller Neuronen der direkt davor liegenden Schicht zusammen. Das genaue Vorgehen bei der Wertermittlung hängt jeweils vom Netzwerk und den dort verwendeten Aktivierungsfunktionen ab.

2.2 Schichten und Kanten

In einem neuronalen Netzwerk ist jedes Neuron einer Schicht mit allen Neuronen der jeweiligen davor liegenden und danach liegenden Schicht über Kanten verbunden. Neuronale Netzwerke sind unidirektionale Graphen, demnach fließen Informationen über Schichten (und folglich Kanten) nur in eine Richtung.

Allen Kanten wird initial eine Gewichtung zugewiesen, welche erneut netzwerkabgängig generiert werden oder durch zuvor angelernte Daten bestimmt werden. Beim Trainieren des Netzwerks werden diese bei jeder Lerniteration (auch *Epoch* genannt) justiert, während sie beim Betrieb für gewöhnlich keine Änderungen mehr erfahren. Die Genauigkeit eines Netzwerks wird überwiegend durch diese Gewichte bestimmt, daher ist das Ziel beim Trainieren eines Netzes die Optimierung jener.

Kantengewichte wirken sich maßgeblich auf die Wertberechnung von Neuronen aus. Diese wird in zwei Schritten ausgeführt. Im ersten Schritt wird aus den Kantengewichten aller eingehenden Kanten und den Werten der darüber verbundenen Neuronen die Produktsumme gebildet. Für den zweiten Schritt sind jeweils Aktivierungsfunktionen notwendig, welche im nachfolgenden Kapitel erläutert werden.

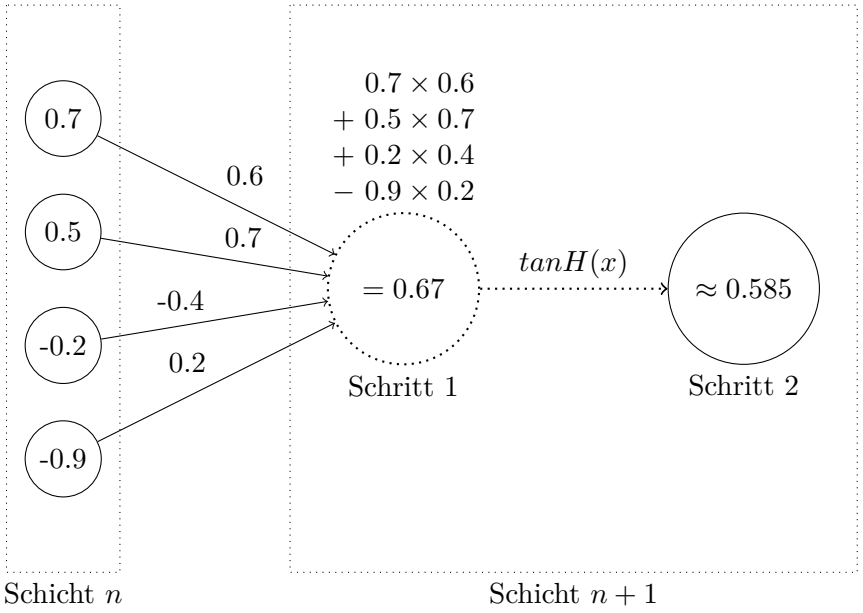


Abbildung 2.2: Berechnung eines Neuronenwertes
mit der $tanH$ Aktivierungsfunktion

2.3 Aktivierungsfunktionen

Im zweiten Schritt wird der zuvor errechnete Wert durch eine weitere Funktion modifiziert. Sogenannte *Aktivierungsfunktionen* können beliebig gewählt werden, müssen jedoch offensichtlich alle möglichen Eingaben auf einen Wert abbilden können. Die verwendete Aktivierungsfunktion kann je nach Schicht variieren. Einmal gewählt, ist diese jedoch für die jeweilige Schicht im Netzwerk für die Laufzeit fest.

Die beiden erläuterten Berechnungsschritte werden schichtweise in Richtung des Datenflusses des Netzwerks für alle Neuronen durchgeführt. Im folgenden Beispiel wird bildhaft dargestellt, wie solch eine Neuronenwertberechnung für ein Netzwerk aussieht, welche in der betrachteten Schicht die $tanH$-Aktivierungsfunktion verwendet.

Die Wahl der Aktivierungsfunktion beeinflusst die möglichen Werte, die Neuronen innerhalb einer Schicht annehmen können. Um eine Wertexplosion zu vermeiden (z.B. bei Verwendung von $ReLU(x) : \max(0, x)$ als Aktivierungsfunktion), können zusätzliche Normalisierungsschichten verwendet werden, welche die Neuronenwerte einer Schicht auf einen erwünschten Zielbereich einschränken. Es existieren jedoch auch Aktivierungs-

funktionen, die diese *Squashing*-Eigenschaft direkt besitzen. Darunter zählt auch die im vorherigen Beispiel verwendetete Funktion $tanH$, bei der alle Eingaben auf den Zielbereich [-1,1] abgebildet werden.

Im Folgenden sind sind einige, für neuronale Netzwerke übliche Aktivierungsfunktionen abgebildet.

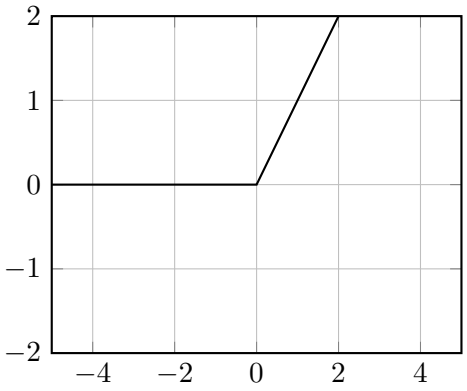


Abbildung 2.3: ReLU

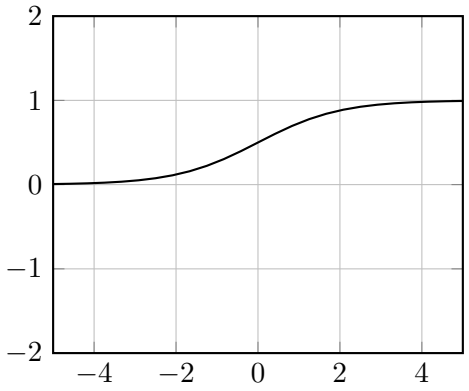


Abbildung 2.4: Sigmoid

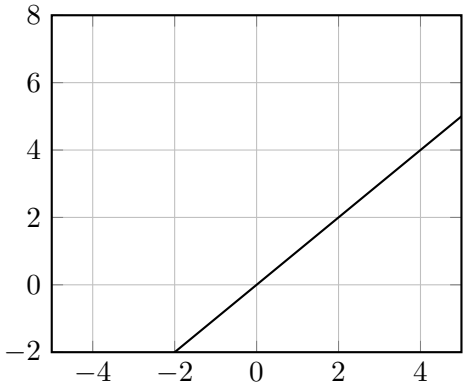


Abbildung 2.5: Softmax

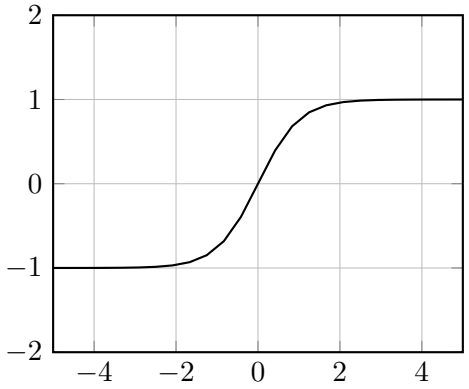


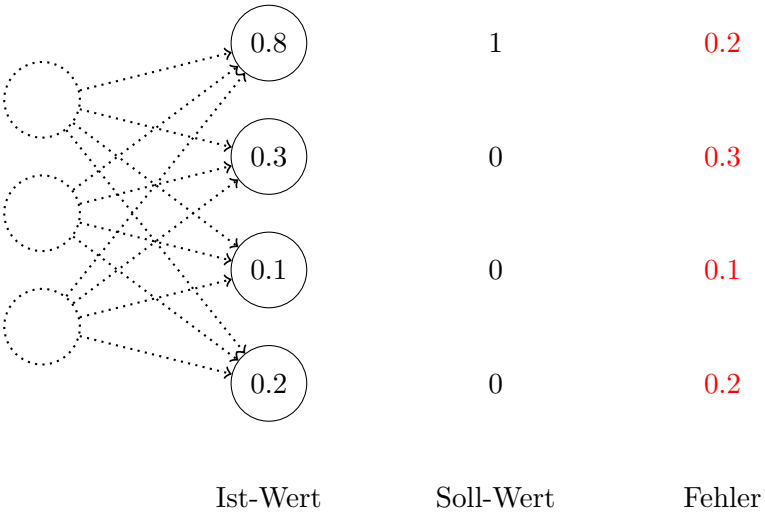
Abbildung 2.6: tanH

2.4 Training

Die Stärken und Schwächen eines neuronalen Netzwerks liegen neben der Struktur der Neuronen, der Schichten und den gewählten Aktivierungsfunktionen vor allem in den verwendeten Kantengewichten. Diese zu optimieren kann ein langwieriger und rechenintensiver Prozess sein. Kleine Änderungen im Aufbau des Netzwerks können große Teile von zuvor erlernten Gewichten unbrauchbar machen, daher werden Netzwerke häufig an sich selbst trainiert. Das bedeutet, dass beim Trainieren eines neuen Netzwerks die initialen Kantengewichte keine emprischen Daten aus anderen (ähnlichen) Netzwerk verwenden, sondern diese zu Beginn pseudozufällig gewählt werden.

2.4.1 Ablauf

Beim Training werden wiederholt verschiedene Eingaben in das Netzwerk getätigt, zu denen das korrekte Ergebnis bekannt ist. In der letzten Schicht des Netzwerks, der Ausgabeschicht, werden die vom Netzwerk bestimmten Werte der jeweiligen Neuronen mit den erwarteten Werten verglichen. Die Differenz dieser wird als Fehler bezeichnet. Das Ziel des Trainings ist es, den Fehler durch Anpassung der Kantengewichte zu verringern oder im Optimalfall ganz zu eliminieren (Fehler = 0). In folgender Abbildung ist ein Fehlerbeispiel dargestellt.



2.4.2 Backpropagation

Es stellt sich nun die Frage, ob und wie errechnet werden kann, welches Kantengewicht wie angepasst werden muss, um den Fehler zu verringern. Der Begriff der *Backpropagation* beschreibt das Durchlaufen des Netzwerks, jedoch entgegen der eigentlichen Laufrichtung und das Rückschließen der notwendigen Gewichtsanpassung, um die Ausgabe in die gewünschte Richtung zu verändern.

Bevor der genaue Wert für eine einzige Anpassung ermittelt werden kann, muss die Entwicklungsrichtung des Fehlers bei Änderung des Kantengewichts errechnet werden.

Kapitel 3

Das BNN

3.1 Aktivierungsfunktion

3.2 Binärer Linear-Layer

3.3 BatchNorm

Die Rolle des *BatchNorm-Layers* in nicht-binären Neuronalen-Netzwerken ist eine leicht andere als in unserem BNN. Durch *BatchNorm* werden die Gewichte einer Schicht normalisiert. Hier wird dafür gesorgt, dass *Batches* einen Mittelwert von Null und eine Standartabweichung von Eins haben[2] .
Durch diese reduzierte Streuung der Werte, können in Neuronalen Netzen höhere Lernraten verwendet werden.
Bei binären Netzwerken hat dies, aufgrund der diskreten Kantengewichte, kaum einen Effekt und hat somit kaum Auswirkungen auf die verwendbare Lernrate.In BNNs erfüllt die *BatchNorm* Schicht eine zentrale Rolle für das Lernen des Netzwerkes. Primär dient die Normalisierung der Schicht dazu, das *expolding-gradient* Problem zu verhindern. Hier wird beim Training des Netzwerkes, welches die Error-Werte reduzieren soll, ein sehr hoher Error-Wert akkumuliert. Dies führt zu einer zu starken Anpassung der Gewichte, welche folgend zu einer niedrigeren Genauigkeit führt. Da durch wiederholtes auftreten die Genauigkeit stark abgesenkt wird, kann dieses Netzwerk sich, ab einer gewissen Schwelle, nicht mehr verbessern.

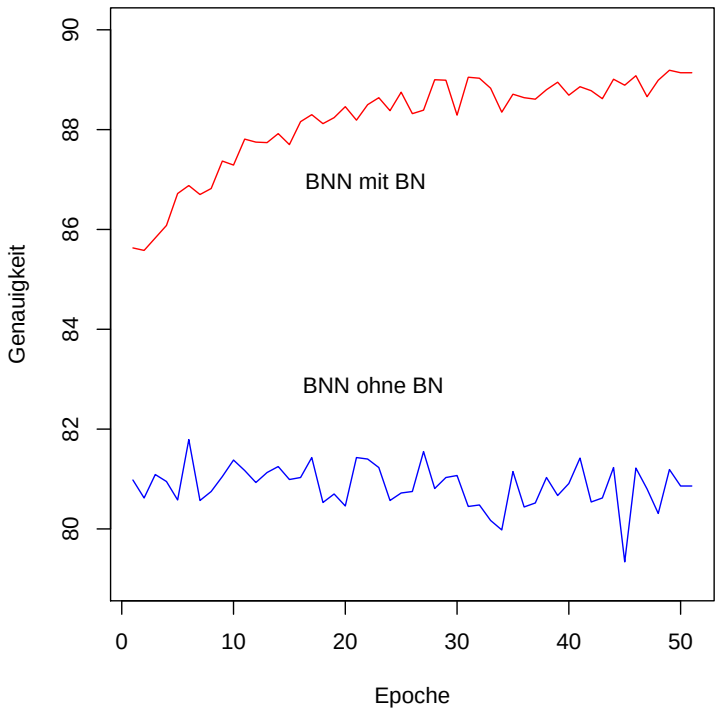


Abbildung 3.1: Training mit und ohne BatchNorm

Der Versuch in Abbildung 3.1 zeigt einen Vergleich des Netzwerkes mit und ohne *BatchNorm*. Hierfür wurde ein Netzwerk für 50 Epochen trainiert. nach jeder Epoche wurde die Genauigkeit getestet. Anschließend wurde weiter Trainiert. Um eine bessere Vergleichbarkeit zu erzielen, wurden die Bilder über die Schwellwert-Methode binarisiert.

Wie in Abbildung 3.1 zu sehen ist, performt das Netzwerk mit *BatchNorm* Schicht zu jeder Trainingsdauer besser als ohne *BatchNorm* Schicht. Während in den ersten 20 Epochen mit *BatchNorm* ein klarer Aufwärtstrend zu erkennen ist, bleibt die Genauigkeit ohne *BatchNorm* immer zwischen 79% und 82%. Hier sind besonders gut die starken Einbrüche nach Unten zu erkennen, die mit *BN* deutlich schwächer sind. Diese zeigen besonders deutlich den starken Genauigkeitsverlust, der durch *exploding gradients* verursacht wird. Die Konsequenz ist, dass eine längere Trainingsdauer keine Verbesserung des Netzwerkes mehr bedeutet, was bedeutet, dass die Konvergenz-Schwelle bei < 82 erreicht ist. Ab dieser Schwelle verbessert sich das Netzwerk ohne *BN* nicht mehr.

3.4 Auswertung

Um am Ende das Ergebnis des Netzwerkes auszuwerten, müssen die Ergebnisse des letzten *Linear-Layers* normalisiert werden. Diese Normalisierung entscheidet, ob ein Neuron feuert oder nicht. Für die Normalisierung der Daten wird hier die *logsoftmax(x)* Funktion verwendet. Durch *softmax(x)* werden die Aktivierungen der letzten Schicht so normalisiert, dass ihre Summe eins ergibt. Dies ist mit Wahrscheinlichkeiten zu vergleichen, mit der das Bild die jeweilig zugeordnete Zahl widerspiegelt. Um bessere Ergebnisse in Kombination mit der Verlustfunktion, *negative log likelihood loss*, zu erzielen, wird die *logsoftmax*-Funktion verwendet.
Da bei der Anwendung des MNIST-Datensatzes immer nur das Neuron ausgewählt werden sollte, da immer nur eine Zahl auf einem Bild abgebildet ist, wird am bei der Auswertung über die *argmax*-Funktion das aktivste Neuron ausgewählt. Dieses entspricht dann der Zahl, die am wahrscheinlichsten abgebildet ist.

3.4.1 logsoftmax vs. softmax

3.5 Verluste durch binäre Linear-Layer

Durch die binarisierung der *Linear-Layer* ist zu vermuten, dass diese, im Vergleich zu normalen *Linear-Layer*, etwas schlechter performen. Dies ist der Fall, da die Anzahl der möglichen Kantengewichte stark, auf Null und Eins, eingeschränkt ist.

Durchgang	binarär	normal
1	88.29	97.43
2	87.32	96.98
3	87.19	97.2

Abbildung 3.2: Verlustmessung für den *Linear-Layer*

Trainiert wurde hier das gleiche Netzwerk, ein mal mit binären *Linear-Layern*, das andere mal mit normalen *Linear-Layer*. Jedes Netzwerk wurde für 50 Epochen trainiert, bevor die Genauigkeit ausgewertet wurde. Um sicher zu gehen, ob die Genauigkeit gegen diesen Wert konvergiert, wurde jede Messung drei mal wiederholt.

Wie in Abbildung 3.2 zu sehen, leidet die Genauigkeit des Netzwerkes beachtlich unter der Binarisierung der *Linear-Layer*. Der Mittelwert für das Training mit normalem *Linear-Layer* ist hierbei $97,2\%$, während bei binären Schichten ein Durchschnitt von $87,6\%$ erreicht wird. Es ist klar zu sehen, dass die Einschränkung der Gewichte auf Null und Eins und der damit einhergehende Granularitätsverlust, sich stark auf die Genauigkeit des Netzwerkes, bei gleicher Größe, auswirken.

Kapitel 4

Training des BNNs

Wie bereits in Kapitel 2.4 beschrieben, werden Netzwerke über eine rechenintensive Annäherung der Kantengewichte an das optimale Ergebnis trainiert. Diese Konvergenz in Richtung eines besseren Ergebnisses kann hierbei, selbst bei schlechtem Training, meist erreicht werden, da selbst marginale Verbessrungen eine Auswirkung haben. Da *BNNs* jedoch die stärkste Form von quantisierten Netzwerken sind, können hier keine stetigen Änderungen an den Kantengewichten vorgenommen werden.

4.1 title

Kapitel 5

Binarisierung

Kapitel 6

Export

Nachdem das Netzwerk nun trainiert wurde, müssen die Ergebnisse, die Kantengewichte und Schwellwerte der Neuronen, nun exportiert werden. Im Folgenden sollen diese dann in den BNN-Beschleuniger Baustein importiert und verwendet werden. Da der Import in VHDL stattfindet, eignen sich hier simple Formate, sprich eine einfache Textdatei. Diese kann dann, Zeichen nach Zeichen, von dem Import-Buffer eingelesen und in einer Matrix gespeichert werden.

6.1 Export der Kantengewichte

Da es sich bei unserem Netzwerk um ein *FullyConnected Neural Network* handelt, ist insbesondere jedes Neuron mit jedem Neuron der Folgenden Schicht verbunden. Bei unserem BNN ergibt sich also folgende Kantenanzahl

$$784 \cdot 500 + 500 \cdot 1024 + 1024 \cdot 1024 = 1.952.576$$

Diese Gewichte müssen alle, mit möglichst wenig Mehrkosten, in die Datei geschrieben werden. Da es sich bei den Gewichten lediglich um binäre Werte, Einsen und Nullen, handelt, ist kein Trennzeichen zwischen den Gewichten notwendig. Die Gewichte sind außerdem, trivialer Weise, präfixfrei und können fortlaufend in die Datei geschrieben werden.
Um die Gewichte zu extrahieren, wird zuerst über jeden *Layer* iteriert. In jedem *Layer* wird nun jedes Neuron abgelaufen. Jedes dieser Neuronen hat nun jeweils eine Kante

zu jedem Neuron in der nachfolgenden Schicht. Hier wird ebenfalls über alle Kanten iteriert und das jeweilige Gewicht wird hinten an eine Variabel an gehangen. Ist nun ein *Layer* fertig, wird der Inhalt der Variable, welche als Zwischenspeicher dient, in die Datei *weights.txt* geschrieben. So wird für jede Schicht ein IO-Zugriff gemacht.

6.2 Export der Schwellwerte

Literatur

[1] Matthieu Courbariaux und Yoshua Bengio. “BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *CoRR* abs/1602.02830 (2016). arXiv: 1602.02830. URL: http://arxiv.org/abs/1602.02830.

[2] Sergey Ioffe und Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Hrsg. von Francis Bach und David Blei. Bd. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, S. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html.