Künstliche Neuronale Netzwerke und ihr Verhalten beim MNIST-Datensatz

---

Verfasser: Tobias Prisching, 8C 2018/19
Betreuer: Mag. Christoph Hödl

BRG/BORG St. Pölten
Schulring 16, 3100 St. Pölten

Abgabe: Februar 2019

Abstract

Die vorliegende Arbeit aus dem Bereich der Informatik beschäftigt sich mit Künstlichen Neuronalen Netzwerken, genauer den Feedforward Neural Networks, Convolutional Neural Networks und Long Short-Term Memory Neural Networks, und wie sich diese beim MNIST-Datensatz, einem bekannten Vergleichstest bestehend aus insgesamt 70.000 Ziffern, verhalten. Dabei werden in den ersten Kapiteln die Grundlagen für Künstliche Neuronale Netzwerke, die drei in der Arbeit behandelten Typen und der Lernprozess anhand des Feedforward Neural Networks erklärt. Im darauf folgenden Kapitel werden die Programmier-Experimente, welche mit der Sprache Python und der Library Keras erstellt wurden, mit den verschiedenen Netztypen und dem MNIST-Datensatz beschrieben und ihre Ergebnisse ausgewertet. Dabei stellt sich heraus, dass jeder Netztypus brauchbare Netze hervorgebracht hat. Des Weiteren werden die Zusammenhänge zwischen der Trefferquote eines KNN und den verschiedenen sogenannten Hyperparametern festgestellt, welche sich auch teilweise untereinander beeinflussen.

Vorwort

Im Verlauf der letzten Monate wurde ich immer wieder von MitschülerInnen, FreundInnen, Bekannten und Verwandten nach meinem VWA Thema gefragt. Prompt kam jedes Mal die auswendig gelernte Antwort zurück. Bis heute konnte noch keiner auf Anhieb etwas damit anfangen oder sich vorstellen, worum es dabei gehen könnte. Mit der Antwort „Ich versuche einem Computer beizubringen, einzelne Ziffern erkennen zu können“ konnten die meisten schon mehr anfangen, beließen es allerdings dabei.

Ich möchte mich im Folgenden bei allen Personen bedanken, ohne die diese Arbeit in ihrer jetzigen Form nie möglich gewesen wäre. Zuerst möchte ich mich bei meinem Betreuer Mag. Christoph Hödl für seine Unterstützung, Ratschläge und konstruktive Kritik bedanken. Auch möchte ich mich bei Herrn Mag. Scheibenpflug für seinen hilfreichen Unterricht im Wahlpflichtfach VWA und Frau Mag.[a] Gertrud Aumayr für die Beantwortung von Fragen bezüglich mathematischer Ausdrücke bedanken. Des Weiteren möchte ich mich bei meinen Eltern bedanken, welche mich bei meiner Arbeit unterstützt und diese auf mathematische, logische und Rechtschreib- sowie Grammatikfehler gegengelesen haben.

Noch lange werden mir die langen Sommernächte, welche bis 3 Uhr morgens mit der Arbeit an der VWA gefüllt waren, sowie das grausame Gefühl, ein Stück Information zu wissen, von dem man weiß, dass es sich irgendwo im Papierberg vor einem befindet, man es jedoch nicht finden kann, in Erinnerung bleiben.

St. Pölten, am 20. 01. 2019
Tobias Prisching

Inhaltsverzeichnis

1 Einleitung

Das Interesse in das Gebiet der Künstlichen Neuronalen Netzwerke ist in den vergangen Jahren stark gestiegen. Entwicklungen, die auf dieser Technologie beruhen, von automatischer Sprach- und Bilderkennung bis hin zum autonomen Fahren, sind bereits teilweise Realität. Und obwohl das Gebiet der Künstlichen Neuronalen Netzwerke schon über 50 Jahre alt ist, waren diese Entwicklungen vor noch zwei Jahrzehnten unvorstellbar, da viele der notwendigen Erkenntnisse erst um die Jahrtausendwende herum gewonnen wurden und die notwendige Rechenkapazität erst seit kurzer Zeit verfügbar ist.[1]

Diese Arbeit beschäftigt sich mit den Grundlagen der Künstlichen Neuronalen Netzwerke, wie diese aufgebaut sind und funktionieren. Die Kapitel 3, 4 und 5 beschäftigen sich mit drei Arten von Netzwerken und im sechsten Kapitel wird beschrieben, wie diese lernen. Dieser Teil der Arbeit beruht rein auf Literatur, welche sowohl in gedruckter Form als auch digital im Internet zu finden ist. Da die behandelte Thematik erst seit relativ kurzer Zeit relevant und interessant ist, waren die meisten Werke erst seit nur wenige Monaten zur Zeit des Verfassens dieser Arbeit alt. In Kapitel 7 wird untersucht, wie sich verschiedene Netztypen beim MNIST-Datensatz, einem Vergleichstest für Künstliche Neuronale Netzwerke, verhalten und wie ein Netz beschaffen sein muss, um diese Aufgabe weitgehendst zu bewältigen. Um diese Fragen zu beantworten, werden zusätzlich zur Literatur auch Experimente in Form von Programmiertätigkeiten ausgewertet.

[1] vgl. Gibson & Patterson, 2017, S. 1

2 Bausteine und Grundlegendes zu Künstlichen Neuronalen Netzwerken

2.1 Grundlegendes zur Verwendung von Fachbegriffen und mathematischen Ausdrücken in dieser Arbeit

Damit man über die verschiedenen Konzepte in dieser Arbeit schreiben kann, benötigt man Fachbegriffe, Terme und Gleichungen. Wie in anderen wissenschaftlichen Gebieten auch gibt es im Bereich der Künstlichen Neuronalen Netze keine standardisierte Schreibweise.

2.1.1 Fachbegriffe

Da der Großteil der verwendeten Literatur in englischer Sprache verfasst ist, liegen auch sämtliche Fachbegriffe nur in dieser vor. Um mögliche Übersetzungsfehler und Differenzen zu anderen deutschen Werken zu verhindern, werden in dieser Arbeit hauptsächlich die englischen Fachbegriffe eingedeutscht. Das hat die Vorteile, dass einerseits der/die LeserIn sich in weiterführender Literatur besser zurecht findet, und andererseits, dass die Herleitungen der mathematischen Variablenbezeichnungen offensichtlich sind. Falls jedoch auch andere, deutsche Bezeichnungen vorkommen, werden diese bei Erstnennung des Begriffes ebenfalls erwähnt.

2.1.2 Mathematische Ausdrücke

Ebenfalls nicht einheitlich sind mathematische Ausdrücke in der Literatur. Häufig werden unterschiedliche Buchstaben, Nummerierungen und Indexierungen für die Variablen verwendet. In dieser Arbeit wird versucht, eine eigene Schreibweise zu verwenden, welche möglichst einfach zu verstehen ist, jedoch nicht die eleganteste oder kürzeste Ausdrucksweise ist. Sämtliche Parameter sind im Anhang A in einer Notationstabelle aufgelistet.

2.2 Definition eines Künstlichen Neuronalen Netzes

Um den Inhalt in den folgenden Kapiteln zu verstehen, ist eine Definition von Künstlichen Neuronalen Netzwerken, kurz KNN oder auch nur Neuronales Netz(werk), notwendig, da die verschiedenen Netztypen auf dieser Definition aufbauen. Ein KNN ist ein rechnerisches Modell, welches ein Netz bestehend aus miteinander verbundenen Knoten, auch künstliche Neurone genannt, dessen Aufbau lose an dem von biologischen Gehirnen orientiert ist, modelliert. Diese Neuronen können miteinander Signale über Verbindungen, den Weights (auch Gewichte genannt), welche die Strukturen aus den Informationen lernen, austauschen und sind in verschiedene Schichten, auch Layer genannt, eingeteilt. Die Weights werden in einem Lernprozess, auch als Training bezeichnet, so angepasst, dass das Netz in den ihm eingespeisten Informationen Strukturen erkennen kann.[1] Aus mathematischer Sicht lassen sich KNNs als komplexe Funktionen mit einigen wenigen bis zu Milliarden Parametern aufschreiben. Die einzige Grenze für die Komplexität und Größe dieser Funktion ist die verfügbare Rechenkapazität.

2.3 Das künstliche Neuron

Das künstliche Neuron ist der Grundbaustein für alle in dieser Arbeit behandelten Arten von KNNs. Ein Neuron n der Schicht l lässt sich am einfachsten als eine mathematische Funktion beschreiben. Es nimmt die Ausgabewerte $x_{(l-1,1)}$, $x_{(l-1,2)}$, $\ldots$, $x_{(l-1,m)}$ der Neurone der vorherigen Schicht $l-1$, multipliziert diese Werte mit Weights $w_{(l-1,1),(l,n)}$, $w_{(l-1,2),(l,n)}$, $\ldots$, $w_{(l-1,m),(l,n)}$, summiert diese auf und addiert einen weiteren Parameter, genannt Bias, $b_{(l,n)}$. Diese Summe wird einer Aktivierungsfunktion $f_{(l)}$ übergeben, deren Wert der endgültige Ausgabewert dieses Neurons ist und an Neurone der Schicht $l+1$ weitergegeben werden kann. Mathematisch lässt sich dies in Formel 1 ausdrücken.[2]

$$x_{(l,n)} = f_{(l)}\left(\sum_{i=1}^{m}(w_{(l-1,i),(l,n)} \cdot x_{(l-1,i)}) + b_{(l,n)}\right) = f_{(l)}(z_{(l,n)}) \tag{1}$$

Diese Formel lässt sich auch so aufschreiben, dass man gleich einen Vektor mit allen Ausgabewerten aller Neurone der Schicht l erhält (Siehe Formel 2).[3] Dabei ist $W_{(l-1),(l)}$ eine Matrix der Form $n \times m$, wobei m die Anzahl der Neurone der Schicht $l-1$ und n die Anzahl der Neurone in Schicht l ist.[4]

$$\vec{x}_{(l)} = f_{(l)}(W_{(l-1),(l)} \cdot \vec{x}_{(l-1)} + \vec{b}_{(l)}) = f_{(l)}(\vec{z}_{(l)}) \tag{2}$$

[1] vgl. Gurney, 1997, S. 1
[2] vgl. Buduma, 2017, S. 8
[3] vgl. ebd., S. 8
[4] vgl. Rashid, 2017, S. 45-49

Das Gewicht einer Eingabe gibt an, wie viel Aussagekraft bzw. wie wichtig der Wert eines Neurons ist. Der Bias eines Neurons lässt sich mit einem Schwellenwert vergleichen, welchen die gewichteten Eingaben überwinden müssen, damit dass Neuron aktiviert wird.[5] Allerdings fehlt der Bias bei Netzen mancher Quellen, so z. B. bei Rashid.

Neurone wie beim menschlichen Gehirn haben keine lineare Funktion für ihren Ausgabewert. Erst wenn ein bestimmter Schwellenwert erreicht ist, geben sie ein Ausgabesignal aus. Dieses Konzept der Nichtlinearität wird bei künstlichen Neuronen übernommen.[6] Diese Nichtlinearität ist wichtig, da diese es einem KNN erst ermöglicht, komplexere Aufgaben zu lösen.[7] Erreicht wird sie durch eine Aktivierungsfunktion, welche die Aktivierung eines Neurons steuert. Ein Neuron gilt dann als aktiviert, wenn sein Ausgabewert ungleich 0 ist.[8] Es gibt verschiedene Aktivierungsfunktionen, welche je nach Aufgabe des KNNs bzw. des Layers im KNN eingesetzt werden.[9] Innerhalb eines KNNs können die Layer unterschiedliche Aktivierungsfunktionen verwenden.[10] Die Graphen von vier Funktionen sind in den Abbildungen 1 - 4 dargestellt, die Sigmoid-Funktion, die TanH-Funktion, die Rectified Linear-Funktion und die Leaky Rectified Linear-Funktion.[11] Die Sigmoid-Funktion ist in der Literatur eine der am häufigsten anzutreffenden Aktivierungsfunktionen, jedoch werden aufgrund eines Nachteils dieser Aktivierungsfunktion, dem Vanishing Gradient (Siehe 6.4.2), andere Funktionen verwendet. Auch die TanH-Funktion besitzt diesen Nachteil, die Rectified Linear-Funktion hat das Problem des „Dying ReLU“ (Siehe 6.4.3), weshalb die Leaky Rectified Linear-Funktion oft bevorzugt wird[12].[13]

[5] vgl. Nielsen, 2015, Kapitel 1/Perceptrons
[6] vgl. Rashid, 2017, S. 32
[7] vgl. Buduma, 2017, S. 13
[8] vgl. Gibson & Patterson, 2017, S. 53
[9] vgl. ebd., S. 255f
[10] vgl. ebd., S. 50
[11] vgl. Buduma, 2017, S. 13ff
[12] Es gibt genaue, mathematische Begründungen, warum eine Aktivierungsfunktion besser ist als die andere, welche allerdings nicht zielführend zur Beantwortung der Forschungsfragen sind.
[13] vgl. Gibson & Patterson, 2017, S. 254

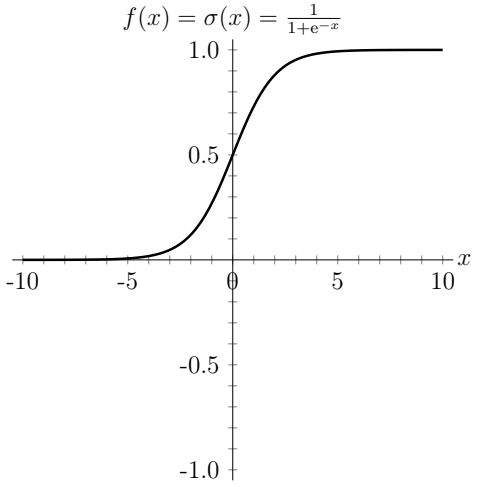


Abb. 1: Sigmoid-Funktion (Quelle: E. D.)

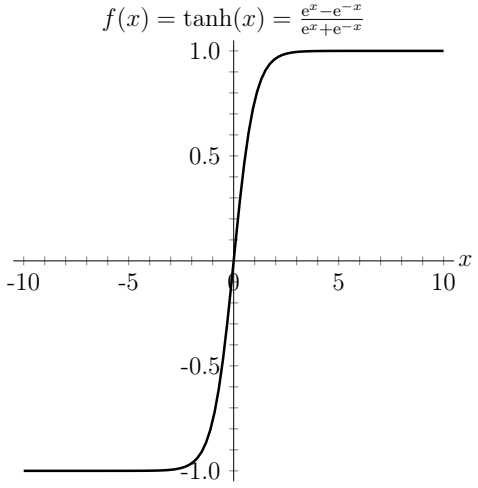


Abb. 2: TanH-Funktion (Quelle: E. D.)

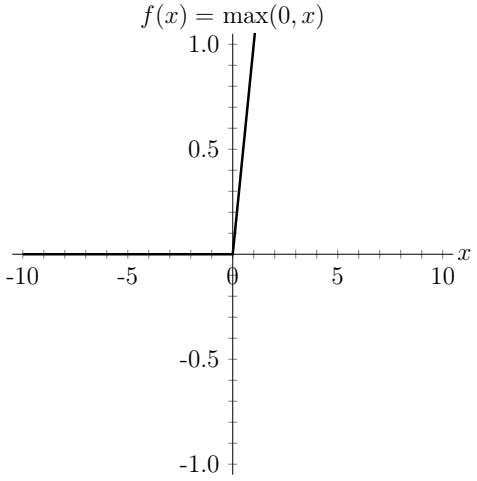


Abb. 3: Rectified Linear-Funktion (Quelle: E. D.)

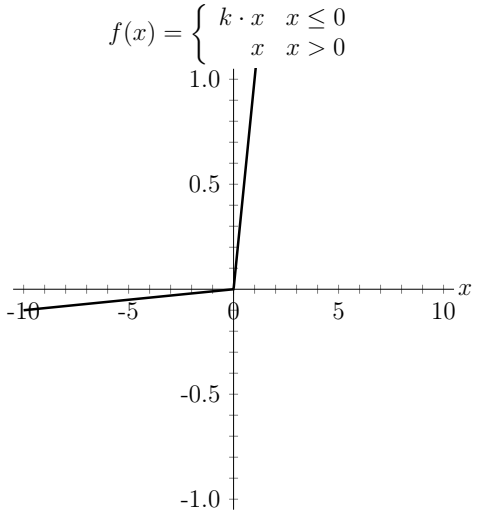


Abb. 4: Leaky Rectified Linear-Funktion, $k = 0,01$ (Quelle: E. D.)

2.4 Schichten

Die Neuronen eines KNNs werden meistens in drei verschiedene Arten von Schichten gruppiert. Dabei sind die Werte der vorherigen Schicht die Eingabe für die nächste Schicht.[14] Es gibt zwar Ausnahmen mit beispielsweise nur einer einzigen Schicht welche aber für diese Arbeit nicht relevant sind.[15] Die in den folgenden Kapiteln behandelten Arten basieren auf folgender Einteilung, welche durch Abb. 5 visualisiert wird.

[14] vgl. Gibson & Patterson, 2017, S. 55f
[15] vgl. ebd., S. 48

2.4.1 Input Layer

Der Input Layer nimmt die dem KNN übergebenen Daten an und leitet diese an den ersten Hidden Layer weiter. Die Anzahl der Neurone in diesem Layer ist oft gleich der Anzahl der Daten einer Eingabe.[16] Werden einem KNN beispielsweise Bilder mit einer Auflösung von 28 mal 28 Pixeln übergeben, besteht der erste Layer aus 784 (28^2) Neuronen. Des Weiteren haben die Neurone des Input Layers keine Parameter und es wird keine Aktivierungsfunktion auf diese angewendet, da sie exakt jene Werte ausgeben sollen, welche dem Netz übergeben wurden. Dies hat keinen genauen Grund und hängt mit der Entwicklungsgeschichte von KNNs zusammen.[17]

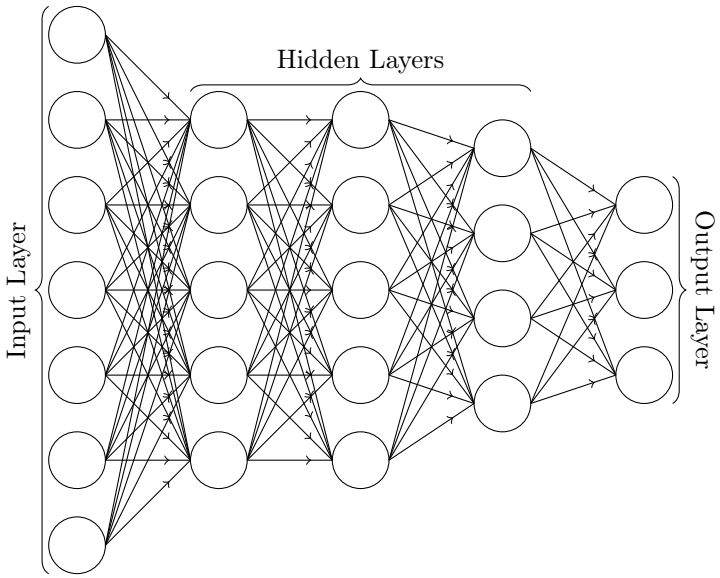


Abb. 5: Ein Beispiel für ein KNN: Der Input Layer hat sieben Neurone, es gibt drei Hidden Layer mit je fünf, fünf und vier Neuronen. Der Output Layer hat drei Neurone. (Quelle: E. D.)

2.4.2 Hidden Layer

Jede in dieser Arbeit behandelte Art von KNNs besitzt mindestens einen oder mehr Hidden Layer. Diese Layer sind verantworlich für den Erfolg von KNNs in den letzten Jahren.[18] Der Name dieser Layer hat keine besondere Bedeutung und bedeutet nur, dass die Ausgabewerte ihrer Neurone nicht die finalen Ausgabewerte des Netzes sind.[19] Der Aufbau der Hidden Layer ist im Gegensatz zu denen der Input und Output Layer nicht so einfach zu entwickeln. Die Anzahl der Neurone in diesen Layern ist meistens durch die Art von Daten gegeben, zudem gibt es auch nur je einen Layer von beiden.[20]

[16] vgl. Gibson & Patterson, 2017, S. 55
[17] vgl. Rashid, 2017, S. 41
[18] vgl. Gibson & Patterson, 2017, S. 55
[19] vgl. Bengio, Courville & Goodfellow, 2016, S. 165
[20] vgl. Nielsen, 2015, Kapitel 1/The architecture of neural networks

2.4.3 Output Layer

Der Output Layer gibt die endgültige Antwort des KNNs aus, welche, je nach Aufgabe des Netzes (Regression[21] oder Classification[22]), eine bestimmte Dimension hat. Abhängig von der in diesem Layer benutzten Aktivierungsfunktion und der Anzahl der Neuronen handelt es sich bei der Ausgabe meistens um entweder einen reellen Wert (Regression) oder einer (Menge von) Wahrscheinlichkeit(en) (Classification).[23] Da der Schwerpunkt dieser Arbeit der MNIST-Datensatz ist (Siehe 2.6) und es sich bei diesem um eine Multiclass Classifications-Aufgabe handelt, wird nur auf diese Kategorie von Aufgaben Rücksicht genommen.

2.5 Arten von Künstlichen Neuronalen Netzwerken

In den folgenden Kapiteln werden drei Netzwerktypen behandelt und erklärt. Ausgewählt wurden dafür das Feedforward Neural Network (FFNN), das Convolutional Neural Network (CNN) und das Long Short-Term Memory Neural Network (LSTM). Alle drei Netzwerktypen gehören zum Supervised Learning (engl. für überwachtes Lernen), d.h. sie lernen mithilfe von Trainingsdaten, bei denen Eingabe und Ausgabe gegeben sind.[24] Das FFNN wurde ausgewählt, da es im Vergleich zu anderen Netztypen sehr einfach aufgebaut ist, das CNN, weil es v. a. bei Bilderkennung sehr erfolgreich ist, und das LSTM, weil es durch die Rückkopplung von Daten interessant ist. [25]

2.6 Der MNIST-Datensatz

Der MNIST-Datensatz ist eine **m**odifizierte Version zweier Datensätze des **N**ational **I**nstitute of **S**tandards and **T**echnology der USA. Das Urheberrecht für den MNIST-Datensatz liegt bei Yann LeCun (Courant Institute, NYU) und Corinna Cortes (Google Labs, New York), welche diesen unter der Creative Commons Attribution-Share Alike 3.0 Lizenz zum freien Gebrauch zur Verfügung stellen. Der Datensatz besteht aus zwei Teilen: Der erste Teil dient zum Trainieren des KNNs und besteht aus 60.000 Ziffern, welche von einer Gruppe, bestehend aus 250 Personen, hand-

[21] Regression modelliert den Zusammenhang zwischen Eingabe und Ausgabe und versucht, für eine gegebene Eingabe die Ausgabe zu ermitteln. (vgl. Gibson & Patterson, 2017, S. 23)

[22] Classification kategorisiert die Eingabe in zwei oder mehr Klassen. Bei zwei Klassen spricht man von Binary Classification. In diesem Fall hat der Output Layer ein Neuron, bei dem der Ausgabewert, welcher oft zwischen 0 und 1 liegt, mit einem Schwellenwert aufgeteilt wird. Für den Fall von x Klassen ($x > 2$), Multiclass Classification genannt, gibt es x Neurone. Die Klasse dessen entsprechendes Neuron den höchsten Wert hat ist die Antwort des KNN. (vgl. ebd., S. 25f)

[23] vgl. Gibson & Patterson, 2017, S. 55 & S. 95

[24] vgl. Wartala, 2018, S. 23ff

[25] vgl. ebd., S. 26 & S. 29

geschrieben wurden. Diese Gruppe setzt sich zusammen aus 125 MitarbeiterInnen des US Census Bureau und 125 High School SchülerInnen. Der zweite Teil besteht aus 10.000 Ziffern, welche von einer zweiten Gruppe (Größe und Zusammensetzung gleich der ersten Gruppe) geschrieben wurden, um das KNN auf Daten zu testen, die es davor noch nie gesehen hat. Die handgeschriebenen Ziffern wurden mit einer Auflösung von 28 mal 28 Pixel in 256 Graustufen digitalisiert und in CSV-Dateien, welche die einzelnen Helligkeitswerte beinhalten, konvertiert.[26] Abb. 6 zeigt Beispiele für die Ziffern Null bis Neun aus dem zweiten Teil von MNIST.

Abb. 6: Beispiele für MNIST-Ziffern. Anmerkung: Alle Darstellungen von Ziffern des MNIST-Datensatzes wurden aus den durch die Library Keras (Siehe 7.1.2) bereitgestellten Tabellen generiert. (Quelle: E. D.)

[26] vgl. Nielsen, 2015, Kapitel 1/Learning with gradient descent

3 Feedforward Neural Networks

Das Feedforward Neural Network (auch Multilayer Perceptron Network[1] oder Multilayer Feed-Forward Network genannt) ist trotz seiner Einfachheit ein schon relativ leistungsstarkes Netzwerk, da mit diesem jede stetige Funktion approximiert werden kann.[2]

3.1 Aufbau

Ein FFNN besteht aus einem Input- und einem Output-Layer und einem oder mehreren Hidden-Layern.[3] Diese Layer sind bei diesem Netztypus fully connected (engl. für komplett verbunden), was bedeutet, dass jedes Neuron einer Schicht mit jedem Neuron der benachbarten Schichten verbunden ist.[4] Jeder Layer besteht aus einem oder mehreren Neuronen, wobei es nicht empfehlenswert ist, aufeinanderfolgende Layer mit gleicher Anzahl an Neuronen zu verwenden.[5]

Des Weiteren ist der Begriff Feedforward in der Bezeichnung wichtig. Er bedeutet, dass Information nur in eine Richtung, nämlich vom Input- zum Output-Layer fließt. Dabei gibt es keine Verbindungen zwischen Neuronen im selben Layer oder in eine vorherige Schicht, da sich sonst Schleifen bilden, sodass der Input in ein Neuron von seinem Output abhängt. Solche Schleifen kommen bei LSTMs in Kapitel 5 vor.[6]

3.2 Funktionsweise

Bisher gab es nur eine Formel für die Berechnung der Werte eines Layers in Abhängigkeit der Ausgabewerte der vorherigen Schicht. Diese lässt sich durch Einsetzen in

[1] Diese Bezeichnung ist irreführend (und wird deshalb in dieser Arbeit auch nicht verwendet), da MLPs meistens nicht aus den im Namen stehenden Perceptrons, welche die Heaviside-Funktion als Aktivierungsfunktion verwenden, bestehen. (vgl. Nielsen, 2015, Kapitel 1/The architecture of neural networks)
[2] vgl. Wartala, 2018, S. 17; Gibson & Patterson, 2017, S. 50
[3] vgl. Gibson & Patterson, 2017, S. 50
[4] vgl. ebd., S. 54
[5] vgl. Gibson & Patterson, 2017, S. 50; Buduma, 2017, S. 11
[6] vgl. Buduma, 2017, S. 11; Nielsen, 2015, Kapitel 1/The architecture of neural networks

sich selbst zu Formel 3 erweitern, sodass der Vektor des Output-Layers in Abhängigkeit des Input-Layers berechnet werden kann.

$$\vec{x}_{(L)} = f_{(L)}\left(W_{(L-1),(L)} \cdot f_{(L-1)}\left(\ldots \cdot f_{(2)}\left(W_{(1),(2)} \cdot \vec{x}_{(1)} + \vec{b}_{(1)}\right) + \ldots\right) + \vec{b}_{(L)}\right) \quad (3)$$

Dies erklärt jedoch nicht, wie die einzelnen Parameter so angepasst werden, dass $\vec{x}_{(L)}$ ein brauchbares Ergebnis liefert. Dieser Vorgang wird Training oder auch Lernen genannt und in Kapitel 6 beschrieben. [7]

[7] vgl. Buduma, 2017, S. 17; ebd., S. 5

4 Convolutional Neural Networks

KNNs, wie das FFNN, sind für Bilderkennung ungeeignet, da sich mit der Verdopplung der Auflösung entlang der Kanten die Anzahl der benötigten Weights vervierfacht. Handelt es sich dabei noch zusätzlich um Farbfotos, verdreifacht sich nochmal die Anzahl der Gewichte[1].[2] Je mehr Parameter ein KNN hat, umso mehr Rechenleistung benötigt es. Deshalb gibt es CNNs, welche auf die Klassifizierung von Bildern spezialisiert sind (und sich daher für MNIST sehr gut eignen).[3]

4.1 Aufbau

Der Aufbau von CNNs basiert auf dem von FFNNs aus Kapitel 3. Die beiden größten Unterschiede zu FFNNs sind, dass erstens die Neurone mancher Layer in drei Dimensionen angeordnet sind, und zweitens nicht jedes Neuron mit allen Neuronen der benachbarten Schichten verbunden ist. Des Weiteren werden für unterschiedliche Verbindungen die gleichen Gewichte genutzt.[4]

4.2 Arten von Schichten

Ein CNN besteht grundsätzlich aus drei verschiedenen Arten von Schichten: Convolutional Layer, Pooling Layer, und Fully-Connected Layer. Außerdem wird eine häufig als Output Layer verwendete Art von Schicht betrachtet, der Softmax Layer[5].[6]

4.2.1 Convolutional Layer

Besonders am Convolutional Layer ist, dass bei diesem die Neurone nicht mit allen Neuronen der vorherigen Schicht verbunden sind. Stattdessen ist jedes Neuron mit Neuronen eines Ausschnitts, welcher eine bestimmte Größe hat, der vorherigen Schicht verbunden, dem Local Receptive Field $R_{(l,d,n)}$, kurz LRF (Siehe Abb. 7).

[1] Aufgrund der geringen Auflösung und fehlender Farbe sind FFNNs trotzdem gut für MNIST und ein Vergleich mit anderen Netzwerktypen geeignet. (vgl. Nash & O'Shea, 2015, S. 3)
[2] vgl. Buduma, 2017, S. 89
[3] vgl. Nielsen, 2018, Kapitel 6/Introducing convolutional networks
[4] vgl. Nash & O'Shea, 2015, S. 4 & 7
[5] Dieser kommt auch in anderen Netztypen vor. (vgl. Gibson & Patterson, 2017, S. 55)
[6] vgl. Nash & O'Shea, 2015, S. 4

Die Gewichte zwischen den Neuronen des LRFs und dem zugehörigen Neuron des Convolutional Layers befinden sich in einer Matrix, dem Kernel $K_{(l,d)}$.[7] Es wird das Skalarprodukt des Kernels und des zum Neuron zugehörige LRF gebildet, zu diesem wird noch ein Bias $b_{(l,d)}$ addiert. Auf die Summe wird wieder eine Aktivierungsfunktion $f_{(l)}$ angewandt.[8] Dieser Ausschnitt wird über den gesamten Input um einen Wert, den Stride, verschoben. Daraus entsteht eine sogenannte Feature Map, eine zweidimensionale Matrix bestehend aus Neuronen. Zu dieser können noch weitere Feature Maps hinzukommen, wodurch der Convolutional Layer dreidimensional wird. Dabei verwenden alle Neurone innerhalb einer Feature Map den gleichen Kernel und den gleichen Bias.[9] Formel 4 berechnet den Ausgabewert $x_{(l,d,n)}$ des Neurons n in Feature Map d von Layer l.[10]

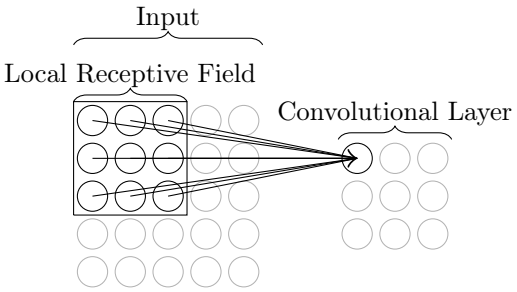


Abb. 7: Visualisierung des Convolutional Layers. Es hilft, sich den Input Layer nicht wie bisher als Vektor, sondern als Matrix vorzustellen. (Quelle: E. D.)

$$x_{(l,d,n)} = f_{(l)} \left(\langle K_{(l,d)}, R_{(l,d,n)} \rangle_F + b_{(l,d)} \right) \tag{4}$$

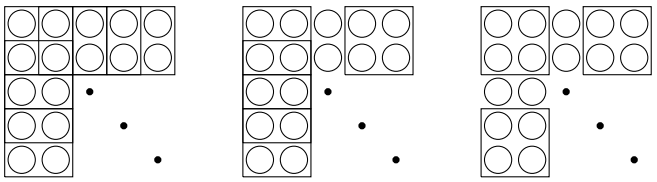


Abb. 8: Visualisierung verschiedener Stride-Werte - Links: $s_{(l)} = (1, 1)$; Mitte: $s_{(l)} = (3, 1)$; Rechts: $s_{(l)} = (3, 3)$ (Quelle: E. D.)

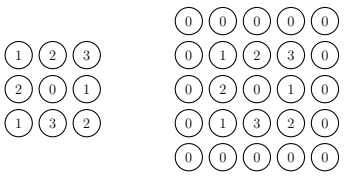


Abb. 9: Beispiel für unterschiedliche Zero Padding Werte - Links: $p_{(l)} = 0$; Rechts: $p_{(l)} = 1$ (Quelle: E. D.)

Der Stride $s_{(l)}$ bestimmt, wie sehr sich die einzelnen LRFs überlappen (Siehe Abb. 8). Stride und Größe der LRFs beeinflussen die Größe der Feature Maps.[11] Diese lässt sich zusätzlich durch den Wert $p_{(l)}$ des Zero Paddings steuern. Indem um die Input Matrix herum Zeilen und Spalten hinzugefügt werden (Siehe Abb. 9), können die Feature Maps vergrößert werden, was v. a. dann angewendet wird, wenn die Feature Maps die gleiche Größe wie die vorherige Schicht haben sollen. Diese Zeilen und Spalten werden meistens mit dem Wert 0 gefüllt.[12]

[7] vgl. Nielsen, 2015, Kapitel 6/Introducing convolutional networks
[8] vgl. Buduma, 2017, S. 93
[9] vgl. Nash & O'Shea, 2015, S. 7
[10] vgl. Nielsen, 2015, Kapitel 6/Introducing convolutional networks
[11] vgl. Nash & O'Shea, 2015, S. 7
[12] vgl. Wu, 2017, S. 12f

Die Dimensionen des Convolutional Layer lassen sich wie folgt berechnen: Hat $X_{(l-1)}$ die Form $\mathrm{H}(X_{(l-1)}) \times \mathrm{W}(X_{(l-1)}) \times \mathrm{D}(X_{(l-1)})$ (der vorangehende Layer kann auch dreidimensional sein, wenn es sich dabei z.B. ebenfalls um einen Convolutional Layer handelt) und $K_{(l)}$ die Form $\mathrm{H}(K_{(l)}) \times \mathrm{W}(K_{(l)}) \times \mathrm{D}(X_{(l-1)}) \times \mathrm{D}(K_{(l)})$ (wobei $K_{(l)}$ die Menge aller Kernel des Layer l und $\mathrm{D}(K_{(l)})$ die Anzahl aller Kernel ist), dann hat $X_{(l)}$ die folgende Form:[13]

$$\frac{\mathrm{H}(X_{(l-1)}) - \mathrm{H}(K_{(l)}) + 2 \cdot p_{(l)}}{\mathrm{H}(s_{(l)}) + 1} \times \frac{\mathrm{W}(X_{(l-1)}) - \mathrm{W}(K_{(l)}) + 2 \cdot p_{(l)}}{\mathrm{W}(s_{(l)}) + 1} \times \mathrm{D}(K_{(l)}) \tag{5}$$

4.2.2 Pooling Layer

Das Ziel eines Pooling Layers ist es, die Größe der Feature Maps zu verkleinern, wodurch gleichzeitig die Komplexität des Models verringert wird. Ähnlich wie die Convolutional Layer haben auch die Pooling Layer drei Dimensionen und ein LRF mit Stride Wert, allerdings keine Kernel oder Biases. Auf das LRF wird eine Pooling-Funktion, wie z. B. Max-Pooling oder Average-Pooling angewandt. Im Fall von Max-Pooling erhält das Neuron des Pooling Layers den höchsten Wert innerhalb des LRF, bei Average-Pooling das arithmetische Mittel der Neurone im LRF.[14]

4.2.3 Fully-Connected Layer

Die Fully-Connected Layer sind gleich den Layern eines FFNN. Jedes ihrer Neurone ist mit jedem Neuron der vorherigen Schicht verbunden und hat seinen eigenen Bias, jede Verbindung hat ihr eigenes Gewicht.[15]

4.2.4 Softmax Layer

Der Softmax Layer ist ein Output Layer, der sich bis auf die Aktivierungsfunktion von bisherigen Output Layern nicht unterscheidet und eine besondere Eigenschaft hat: Die Summe der Ausgabewerte aller Neurone dieses Layers hat den Wert 1. Dies ist insofern sinnvoll, da die Ausgabewerte als eine Wahrscheinlichkeitsverteilung betrachten werden können. Deshalb wird der Softmax Layer v. a. bei Classification-Aufgaben als Output Layer eingesetzt, da der Wert $x_{(L,n)}$ als Wahrscheinlichkeit dafür betrachtet werden kann, dass die korrekte Klasse jene des Neurons n ist. Formel 6 gibt die Berechnung der Aktivierung eines Neurons eines Softmax Layers an.[16]

$$x_{(L,n)} = \frac{\mathrm{e}^{z_{(L,n)}}}{\sum_{n=1}^{\mathrm{len}(L)} \left(\mathrm{e}^{z_{(L,n)}}\right)} \tag{6}$$

[13] vgl. Wu, 2017, S. 12f; Nash & O'Shea, 2015, S.7
[14] vgl. Nash & O'Shea, 2015, S. 8
[15] vgl. ebd., S. 8
[16] vgl. Nielsen, 2015, Kapitel 3/Softmax

4.3 Anordnung der Schichten

Es gibt keine genauen Regeln, wie die verschiedenen Layer kombiniert werden sollen, allerdings kann man nicht einfach ein paar Schichten miteinander kombinieren und brauchbare Ergebnisse erwarten. Es gibt jedoch Reihenfolgen für die Layer, welche sich in der Literatur durchgesetzt haben. So folgt meistens auf einen Convolutional Layer ein Pooling Layer. Diese Abwechslung zwischen den beiden kann mehrmals wiederholt werden, bevor zum Schluss ein oder mehrere Fully-Connected Layer folgen. Des Weiteren hat es sich bewährt, mehrere Convolutional Layer vor einen Pooling Layer zu stapeln, da dadurch die Komplexität des Modells gesteigert werden kann.[17]

[17] vgl. Nash & O'Shea, 2015, S. 8f

5 Long Short-Term Memory Neural Networks

Bisher wurden in der Arbeit nur KNNs betrachtet, welche für viele Aufgaben geeignet sind, jedoch die einzelnen Eingaben voneinander getrennt behandeln. Es gibt allerdings Aufgabenstellungen, bei denen die Eingaben in einem Zusammenhang stehen, wie z. B. bei der Klassifizierung, ob die Aussage eines Satzes positiv oder negativ ist[1], die Beschreibung eines Bildes[2] oder die Übersetzung von Sprachen[3].[4] Dafür gibt es die Recurrent Neural Networks (kurz RNNs), welche die Besonderheit haben, dass sie zusätzlich die zeitliche Dimension, in der die Daten eingegeben werden, modellieren, wodurch Eingabedaten die Ausgaben späterer Eingabedaten beeinflussen können[5]. Dies wird erreicht indem die Ausgabe eines Neurons diesem als zusätzliche Eingabe zu einem späteren Zeitpunkt übergeben wird.[6]

Diese Arbeit befasst sich genauer nur mit dem Long Short-Term Memory Neural Network, kurz LSTM Netz, welches eines der häufigsten Typen von RNNs ist, da es im Gegensatz zum allgemeinen Modell nicht vom Vanishing Gradient (Siehe 6.4.2) betroffen ist.[7]

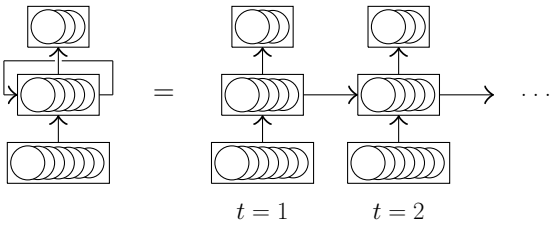


Abb. 10: Visualisierung der Rückkopplung eines RNNs. (Quelle: E. D.)

[1] „Many-To-One“: Eine Sequenz von Eingaben (Wörter) erzeugt eine Ausgabe (positive/negative Aussage).
[2] „One-To-Many“: Eine Eingabe (Bild) erzeugt eine Sequenz von Ausgaben (mehrere Wörter, die das Bild beschreiben).
[3] “Many-To-Many“: Eine Sequenz von Eingaben (Wörter einer Sprache) erzeugt eine Sequenz von Ausgaben (Wörter einer anderen Sprache).
[4] vgl. Gupta, 2017
[5] Da diese Modelle mehrere Eingaben pro Trainingsbeispiel benötigen, werden im Beispiel von MNIST die einzelnen Spalten eines Bildes dem Netz übergeben.
[6] vgl. Gibson & Patterson, 2017, S. 143-146; Gupta, 2017
[7] vgl. Gibson & Patterson, 2017, S. 149-150

5.1 RNNs allgemein

RNNs gehen, wie CNNs, von den FFNNs aus, und wandeln diese ab.[8] Der Unterschied liegt darin, dass die Ausgaben von Neuronen der Hidden Layer $\vec{x}_{(l)}^{(t)}$ im nächsten Zeitschritt $t+1$ als zusätzliche Eingaben, neben der Ausgabe $\vec{x}_{(l-1)}^{(t+1)}$ der Neuronen der vorherigen Schicht, verwendet werden. Berechnen lassen sich die Ausgabewerte des Layer l zum Zeitpunkt t mit Formel 7[9]. Abb. 10 visualisiert, wie die Berechnungen zum Zeitpunkte t von den vorherigen Zeitpunkten abhängig sind.[10]

$$\vec{x}_{(l)}^{(t)} = f_{(l)}(W_{(l-1),(l)}^{(t),(t)} \cdot \vec{x}_{(l-1)}^{(t)} + W_{(l),(l)}^{(t-1),(t)} \cdot \vec{x}_{(l)}^{(t-1)} + \vec{b}_{(l)}) \tag{7}$$

5.2 LSTM Netzwerke

LSTM Netze unterscheiden sich in der Funktionsweise ihrer Neurone, den LSTM Zellen, von anderen Netzen. Diese lösen auch das Problem des Vanishing Gradients.[11]

5.2.1 LSTM Zellen

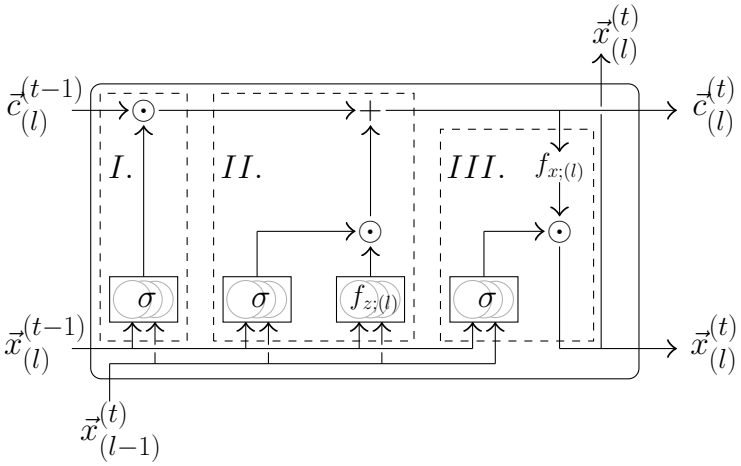


Abb. 11: Visualisierung der Datenströme einer LSTM Zelle. (Quelle: E. D.)

Das Besondere an LSTM Zellen ist der Zell Status, welcher sich über sogenannte Gates (auch Gatter genannt) verändern lässt. Ein LSTM hat normalerweise drei Gates, welche in Abb. 11 mit strichlierten Boxen gekennzeichnet sind: Ein Forget Gate ($I.$), ein Input Gate ($II.$) und ein Output Gate ($III.$). Das Forget Gate bestimmt, welche Informationen vom Zell Status des Zeitpunktes $t-1$ übernommen werden sollen. Dies geschieht in Form des Vektors $\vec{u}_{(l)}^{(t)}$, welcher mit Formel 8 berechnet wird, dessen Elemente einen Wert zwischen 0 und 1 haben, wobei 0 bedeutet, dass diese Information vergessen, und 1, dass sie weiterhin gespeichert werden soll.[12]

[8] vgl. Gibson & Patterson, 2017, S. 149

[9] Hier widersprechen sich verschiedene Autoren. Während Gupta eine 1×1 Matrix als Gewicht für $\vec{x}_{(l)}^{(t-1)}$ verwendet, benutzt Karpathy keinen Bias in seinen Berechnungen. Daher behandelt diese Arbeit diesen Vektor genau gleich wie $\vec{x}_{(l-1)}^{(t)}$ und verwendet einen Bias. (vgl. Gupta, 2017; Karpathy, 2015)

[10] vgl. Gupta, 2017

[11] vgl. ebd.

[12] vgl. Olah, 2015

$$\vec{u}_{(l)}^{(t)} = f_{u;(l)} \left(W_{u;(l-1),(l)}^{(t),(t)} \cdot \vec{x}_{(l-1)}^{(t)} + W_{u;(l),(l)}^{(t-1),(t)} \cdot \vec{x}_{(l)}^{(t-1)} + \vec{b}_{u;(l)} \right) \tag{8}$$

Das Input Gate entscheidet, welche neuen Informationen zum Zell Status hinzugefügt werden sollen. Der Vektor $\vec{i}_{(l)}^{(t)}$ (Formel 9) hat für den Vektor $\vec{z}_{(l)}^{(t)}$ (Formel 10) dabei die gleiche Funktion, wie $\vec{u}_{(l)}^{(t)}$ für $\vec{c}_{(l)}^{(t-1)}$: $\vec{z}_{(l)}^{(t)}$ gibt dabei die neuen Informationen an, $\vec{i}_{(l)}^{(t)}$ welche der neuen Daten übernommen werden sollen. Dabei ist $f_{z;(l)}$ die Aktivierungsfunktion für die neuen Daten, wobei hierfür meistens die TanH-Funktion eingesetzt wird. Mit Formel 11 wird der Zell Status anschließend aktualisiert.

$$\vec{i}_{(l)}^{(t)} = f_{i;(l)} \left(W_{i;(l-1),(l)}^{(t),(t)} \cdot \vec{x}_{(l-1)}^{(t)} + W_{i;(l),(l)}^{(t-1),(t)} \cdot \vec{x}_{(l)}^{(t-1)} + \vec{b}_{i;(l)} \right) \tag{9}$$

$$\vec{z}_{(l)}^{(t)} = f_{z;(l)} \left(W_{z;(l-1),(l)}^{(t),(t)} \cdot \vec{x}_{(l-1)}^{(t)} + W_{z;(l),(l)}^{(t-1),(t)} \cdot \vec{x}_{(l)}^{(t-1)} + \vec{b}_{z;(l)} \right) \tag{10}$$

$$\vec{c}_{(l)}^{(t)} = \vec{u}_{(l)}^{(t)} \odot \vec{c}_{(l)}^{(t-1)} + \vec{i}_{(l)}^{(t)} \odot \vec{z}_{(l)}^{(t)} \tag{11}$$

Schlussendlich wird der Output berechnet, wobei $\vec{o}_{(l)}^{(t)}$ (Formel 12) ebenfalls ein Vektor ist, welcher entscheidet, welche Einträge eines Vektors, in diesem Fall des Vektors $\vec{c}_{(l)}^{(t)}$, verwendet werden. Die Endgültige Ausgabe zum Zeitpunkt t wird mit Formel 13 berechnet, wobei $f_{x;(l)}$ eine Aktivierungsfunktion ist, für die auch meistens die TanH-Funktion verwendet wird.[13]

$$\vec{o}_{(l)}^{(t)} = f_{o;(l)} \left(W_{o;(l-1),(l)}^{(t),(t)} \cdot \vec{x}_{(l-1)}^{(t)} + W_{o;(l),(l)}^{(t-1),(t)} \cdot \vec{x}_{(l)}^{(t-1)} + \vec{b}_{o;(l)} \right) \tag{12}$$

$$\vec{x}_{(l)}^{(t)} = \vec{o}_{(l)}^{(t)} \odot f_{x;(l)}(\vec{c}_{(l)}^{(t)}) \tag{13}$$

Es gibt verschiedene Varianten von LSTM Zellen, welche beispielsweise den vorherigen Zell Status $\vec{c}_{(l)}^{(t-1)}$ in die Berechnung von $\vec{u}_{(l)}^{(t)}$, $\vec{i}_{(l)}^{(t)}$ und $\vec{o}_{(l)}^{(t)}$ miteinbeziehen, allerdings in dieser Arbeit nicht untersucht werden.[14]

5.2.2 Aufbau

Ein LSTM Layer besteht aus einer solchen Zelle, wobei die Anzahl der Dimensionen von $\vec{x}_{(l)}^{(t)}$ (bzw. auch von $\vec{c}_{(l)}^{(t)}$) vergleichbar mit der Anzahl der Neuronen eines Fully-Connected Layers ist. Dabei kann es einen oder mehrere LSTM Layer geben.[15]

[13] vgl. Olah, 2015; Gibson & Patterson, 2017, S. 154f
[14] vgl. Olah, 2015
[15] vgl. Olah, 2015; Gibson & Patterson, 2017, S. 156

6 Lernen eines KNNs am Beispiel des FFNN

Bisher wurden in der Arbeit nur verschiedene Arten von KNNs, ihr Aufbau und ihre Funktionsweise behandelt. Jedoch fehlt noch ein wichtiger Teil für die Entwicklung von KNNs, das Training. Unter Training versteht man das Bestimmen der passenden Werte für die Parameter des KNNs für eine bestimmte Aufgabe mithilfe von Trainingsdaten.[1] Erklärt wird der Vorgang des Trainings in dieser Arbeit nur anhand des FFNN, und zwar aus dem Grund, dass dieser Prozess bei diesem Netztypus am verständlichsten ist.

Es gibt mehrere Optimierungs-Methoden, d. h. Möglichkeiten, wie das Trainieren umgesetzt wird. Diese Arbeit befasst sich mit dem Gradient Descent (GD) und dessen zwei bekanntesten Abwandlungen, dem Stochastic Gradient Descent (SGD) und Mini-Batch Gradient Descent.[2] Das Problem beim Herausfinden der Werte für diese Parameter ist, dass diese nicht über ein Gleichungssystem gelöst werden können, sondern iterativ berechnet werden müssen.[3]

6.1 Gradient Descent

Um ein KNN trainieren zu können, muss zunächst herausgefunden werden, wie gut oder schlecht die bisherigen Weights geeignet sind. Dazu wird die Eingabe $\vec{x}_{(1)}(\mathrm{tr}:i)$ [4] eines Trainingsbeispiels i in das Netz eingespeist, dessen gewünschte Ausgabe $\vec{y}(\mathrm{tr}:i)$ [5] bekannt ist, und liefert einen Output $\vec{x}_{(L)}(\mathrm{tr}:i)$. Wie nahe dieser Output an den Gewünschten herankommt, lässt sich mit einer Cost-Funktion, auch als Loss-Funktion bezeichnet, ermitteln. Es gibt mehrere Cost-Funktionen (Siehe 6.3.2), eine der einfacheren und gebräuchlicheren ist die mittlere quadratische Abweichung in Formel 14, welche den durchschnittlichen Cost-Wert für alle Trainingsbeispiele berechnet.[6]

[1] vgl. Buduma, 2017, S. 17; Gibson & Patterson, 2017, S. 27
[2] vgl. Gibson & Patterson, 2017, S. 98f; ebd., S. 30ff; Buduma, 2017, S. 25
[3] vgl. Rashid, 2017, S. 71; Buduma, 2017, S. 18f
[4] Im Fall von MNIST ein Vektor mit 784 Dimensionen; eine pro Pixelwert
[5] Im Fall von MNIST ein Vektor mit 10 Dimensionen; eine pro möglicher Ziffer
[6] vgl. Nielsen, 2015, Kapitel 1/Learning with gradient descent

$$C = \frac{1}{2 \cdot \mathrm{Tr}} \cdot \left(\sum_{i=1}^{\mathrm{Tr}} |(\vec{y}(\mathrm{tr} : i) - \vec{x}_{(L)}(\mathrm{tr} : i))^2| \right) \tag{14}$$

Der Wert der Cost-Funktion wird kleiner, je kleiner die Differenz zwischen dem gewünschten und dem eigentlichen Output wird, was wiederum eine Folge von an die Trainingsdaten angepassten Parametern ist. Das Ziel des GD Algorithmus ist es, den Wert der Cost-Funktion zu minimieren[7].[8]

Vor dem Training müssen die Weights und Biases initialisiert werden, wofür man Zufallswerte verwendet.[9] Da der Cost-Wert nur von den Parametern abhängig ist (Input und Output sind bei Trainingsbeispielen fest vorgegeben und lassen sich nicht verändern), lassen sich diese mithilfe der partiellen Ableitung der Cost-Funktion nach diesen verändern. Die Formeln zum Aktualisieren der Parameter sind folgende:

$$w_{(l-1,m),(l,n)} = w_{(l-1,m),(l,n)} - \eta \cdot \frac{\partial C}{\partial w_{(l-1,m),(l,n)}} \tag{15}$$

$$b_{(l,n)} = b_{(l,n)} - \eta \cdot \frac{\partial C}{\partial b_{(l,n)}} \tag{16}$$

Der Vektor mit den partiellen Ableitungen der Cost-Funktion nach den einzelnen Weights und Biases wird Gradient genannt.[10] Die partielle Ableitung $\frac{\partial C}{\partial w_{(l-1,m),(l,n)}}$ bzw. $\frac{\partial C}{\partial b_{(l,n)}}$ gibt an, wie sich die Cost-Funktion ändert, wenn man diesen Parameter ändert.[11] Ist die partielle Ableitung positiv bzw. negativ, muss der Parameter verringert bzw. erhöht werden. Je größer die partielle Ableitung, desto größer der Betrag um den der Parameter verändert werden muss.[12] Der Faktor η wird Learning Rate (oder Lernrate) genannt und steuert, wie groß die Veränderungen der Parameter sein sollen. Die Lernrate ist einer der Hyperparameter (Siehe 6.3) eines KNNs.[13]

Durch mehrfaches, iteratives Anwenden der beiden Formeln 15 und 16 lassen sich die Parameter so anpassen, dass die Cost-Funktion zu einem Minimum kommt. Dabei wird zuerst der durchschnittliche Gradient aller Trainingsbeispiele berechnet, mit welchem dann die Weights und Biases aktualisiert werden.[14] Ein solcher Durchgang durch alle Trainingsdaten wird als Epoch (auch Epoche) bezeichnet.[15] Die Anzahl der Epochen beim Trainieren ist ebenfalls ein Hyperparameter.[16]

[7] Das ein kleinerer Cost-Wert nicht unbedingt ein besseres KNN zufolge hat, sieht man in 6.4.1 (vgl. Buduma, 2017, S. 31)
[8] vgl. ebd., Kapitel 1/Learning with gradient descent
[9] vgl. Nielsen, 2015, Kapitel 1/Implementing our network to classify digits
[10] vgl. Nielsen, 2015, Kapitel 1/Learning with gradient descent; ebd., Kapitel 2/The two assumptions we need about the cost function
[11] vgl. Rashid, 2017, S. 81
[12] vgl. ebd., S. 76
[13] vgl. Gibson & Patterson, 2017, S. 31
[14] vgl. Nielsen, 2015, Kapitel 1/Learning with gradient descent
[15] vgl. Rashid, 2017, S. 155
[16] vgl. Gibson & Patterson, 2017, S. 100

Es gibt allerdings zwei größere Probleme mit diesem Algorithmus: Erstens gibt es keine Garantie, dass die Cost-Funktion sich dem globalen und nicht nur einem lokalen Minimum annähert. Zweitens dauert das Berechnen eines durchschnittlichen Gradienten bei einer großen Menge von Trainingsdaten relativ lange.[17] Eine Möglichkeit diese Probleme zu umgehen wäre, die Parameter nach jedem Trainingsbeispiel, d. h. mit den einzelnen Gradienten der Beispiele zu adjustieren, was als Stochastic Gradient Descent (SGD) bekannt ist. Der Nachteil dieser Optimierungs-Methode ist, dass die einzelnen Gradienten möglicherweise keine genügende Annäherung an den durchschnittlichen Gradienten sind, wodurch die Werte der Parameter fluktuieren.[18] Um diese Aufgabe zu lösen, verwendet man den Mini-Batch GD Algorithmus, welcher den Gradienten für eine kleine Teilmenge (Mini-Batch) aller Trainingsbeispiele berechnet und mit diesen die Parameter aktualisiert[19].[20] Zu Beginn eines Epochs werden die Trainingsbeispiele in ihrer Reihenfolge durchgemischt und in die Mini-Batches aufgeteilt, welche dann zum Trainieren verwendet werden.[21]

Nach dem Trainieren wird das KNN getest, d. h. es werden Testbeispiele, welche nicht bei den Trainingsbeispielen dabei waren, dem Netz übergeben um zu überprüfen, wie viele zuvor noch nie gesehene Beispiele richtig klassifiziert[22] werden.[23] Falls man mit dem Testergebnis zufrieden ist, ist das KNN bereit für seinen Einsatz.[24] Neben den Trainings- und Testdatensätzen gibt es noch den Validationsdatensatz (Siehe 6.3.3).[25]

6.2 Backpropagation

Zur Berechnung der partiellen Ableitungen wird der Backpropagation (auch Fehlerrückführung genannt) Algorithmus verwendet, welcher das Problem ihrer Berechnung bei mehrschichtigen KNNs gelöst hat.[26] Um diese zu berechnen, wird zunächst ein Zwischenwert eingeführt, der Fehler $\delta_{(l,n)}$ eines Neurons, welcher zunächst über $\frac{\partial C}{\partial z_{(l,n)}}$ definiert wird. Das kommt daher, dass diese partielle Ableitung den Faktor angibt, wie stark sich eine Änderung $\Delta z_{(l,n)}$ auf den Wert der Cost-Funktion auswirkt (nämlich um eine Differenz von $\frac{\partial C}{\partial z_{(l,n)}} \cdot \Delta z_{(l,n)}$). Mit Backpropagation lässt

[17] vgl. Nielsen, 2015, Kapitel 1/Learning with gradient descent
[18] vgl. Buduma, 2017, S. 33
[19] Es kommt in der Literatur auch vor, dass SGD und Mini-Batch GD zu SGD zusammengefasst werden, wobei SGD wie es hier erklärt ist wie Mini-Batch GD mit einer Teilmengen-Größe von 1 gehandhabt wird.
[20] vgl. Buduma, 2017, S. 27
[21] vgl. Nielsen, 2015, Kapitel 1/Implementing our network to classify digits
[22] Unter der Annahme, dass es sich um eine Classifications-Aufgabe handelt
[23] vgl. Rashid, 2017, S. 148; Buduma, 2017, S. 29
[24] vgl. Buduma, 2017, S. 33
[25] vgl. Nielsen, 2015, Kapitel 1/Implementing our network to classify digits
[26] vgl. Wartala, 2018, S. 17

sich dieser Fehler für jedes Neuron jeder Schicht berechnen, mit welchem man auch die partiellen Ableitungen $\frac{\partial C}{\partial w_{(l-1,m),(l,n)}}$ und $\frac{\partial C}{\partial b_{(l,n)}}$ berechnen kann.[27] Der Algorithmus verwendet die folgenden vier Formeln 17 bis 20 um die Fehler und partiellen Ableitungen zu berechnen:

$$\vec{\delta}_{(L)} = \nabla_{\vec{x}_{(L)}} C \odot f'_{(L)}(\vec{z}_{(L)}) \tag{17}$$

$$\vec{\delta}_{(l)} = (W_{(l),(l+1)}{}^{\intercal} \cdot \vec{\delta}_{(l+1)}) \odot f'_{(l)}(\vec{z}_{(l)}) \tag{18}$$

$$\frac{\partial C}{\partial b_{(l,n)}} = \delta_{(l,n)} \tag{19}$$

$$\frac{\partial C}{\partial w_{(l-1,m),(l,n)}} = \delta_{(l,n)} \cdot x_{(l-1,m)} \tag{20}$$

Die erste Formel berechnet den Fehler in der letzten Schicht des KNNs. Mithilfe der zweiten Formel lässt sich der Fehler in die Schichten $L-1$, $L-2$, ..., 1 rückführen. Die letzten beiden Formeln dienen der Berechnung der partiellen Ableitungen[28].[29]

6.3 Hyperparameter

Unter Hyperparametern versteht man Parameter, welche das Training eines KNN steuern. Diese wären: Anzahl der Neurone pro Layer, Learning Rate, Aktivierungsfunktion, Initialisierung der Gewichte und Biases, Cost-Funktion, Anzahl der Epochs und Größe der Mini-Batches[30].[31]

6.3.1 Learning Rate

Die Learning Rate steuert, wie groß die Schritte relativ zu den partiellen Ableitungen sein sollen, mit denen die Parameter verändert werden. Dabei hat die Lernrate Einfluss darauf, wie gut und wie lange das KNN trainiert wird. Eine große Learning Rate (z.B. $\eta = 1$) führt zwar schneller zu einem Minimum als eine kleine Lernrate (z.B. $\eta = 0,000001$), hat allerdings den Nachteil, dass der Wert der Cost-Funktion nicht so nahe ans Minimum herankommt, sondern sich nur in seiner Umgebung bewegt. Die kleinere Lernrate nähert sich dem Minimum zwar besser an, benötigt

[27] vgl. Nielsen, 2015, Kapitel 2/The four fundamental equations behind backpropagation
[28] Da diese Formeln für manche Leser nicht einfach verständlich sind, befinden sich in Anhang B Beweise für diese.
[29] vgl. ebd., Kapitel 2/The four fundamental equations behind backpropagation
[30] Hyperparameter, die nur für bestimmte Netztypen benötigt werden, werden nicht behandelt.
[31] vgl. Gibson & Patterson, 2017, S. 78; ebd., S. 100

dafür aber mehr Epochen. Es gibt zwar die Möglichkeit, die Lernrate in Abhängigkeit zur Nähe am Minimum anzupassen, allerdings ist das außerhalb des Rahmens dieser Arbeit.[32]

6.3.2 Cost-Funktion

Es gibt unterschiedliche Cost-Funktionen, welche je nach Aufgabe eines Netzes verwendet werden. So gibt es beispielsweise eigene Cost-Funktionen für Classification-Aufgaben.[33] In dieser Arbeit wird allerdings nur die Mean Squared Error Funktion eingesetzt, da andere Cost-Funktionen bestimme Voraussetzungen an den Output Layer haben oder nur bei bestimmten Aufgaben eingesetzt werden können.[34]

6.3.3 Epochs & Mini-Batch

Die Größe der Mini-Batches hat einen Einfluss darauf, wie effizient die Optimierungs-Methode arbeitet. Zu kleine Mini-Batches (z.B. 1) nutzen oft die Fähigkeiten der Hardware nicht komplett aus, wodurch das Trainieren länger dauert. Zu große Mini-Batches sind ebenfalls ineffizient, da eine ausreichende Annäherung an den durchschnittlichen Gradienten auch mit kleineren Mini-Batches erzielt werden kann.
Die Anzahl der Epochs gibt an, wie oft das KNN die Trainingsbeispiele durcharbeitet. Dabei gibt es einen Zusammenhang zwischen Anzahl der Epochs und der Learning Rate. Je mehr Epochen das KNN beim Training durchläuft, desto kleiner sollte die Learningrate sein, da sich die Optimierungs-Methode in vielen Schritten besser ans Minimum annähern kann, wenn diese kleiner sind. Durchläuft das KNN nur wenige Epochs sollte die Learning Rate größer sein, da die wenigen Aktualisierungen größer sein müssen, um sich dem Minimum zu nähern.[35]
Das Training kann allerdings mit Early Stopping schon vor Durchlauf aller Epochs beendet werden. Nach jedem Epoch wird der Validationsdatensatz durchgegangen. Sobald sich die Trefferquote nur noch bei den Trainingsbeispielen, nicht aber bei den Validationsbeispielen verbessert, wird das Training gestoppt, da es sonst zu Overfitting (Siehe 6.4.1) kommt.[36]

6.3.4 Initialisierung der Parameter

Für die Effizienz der Optimierungs-Methode sind gute Startwerte für die Parameter ausschlaggebend.[37] Diese werden mit Zufallswerten initialisiert. Der Grund, warum nicht für alle Weights der gleiche Startwert verwendet wird, ist, dass in diesem Fall

[32] vgl. ebd., S. 100
[33] vgl. ebd., S. 71-78
[34] vgl. Nielsen, 2015, Kapitel 3/Softmax; Rashid, 2017, S. 79
[35] vgl. Rashid, 2017, S. 157
[36] vgl. Nielsen, 2015, Kapitel 3/Overfitting and regularization
[37] vgl. Nielsen, 2015, Kapitel 3/Weight initialization

die Optimierungs-Methode nicht dazu in der Lage wäre, unterschiedliche Gewichte zu erzeugen, d. h. alle Gewichte haben am Ende des Trainings den gleichen Wert (auch wenn dieser nicht gleich dem Startwert sein muss). Wenn alle Gewichte gleich sind, wäre das so, wie wenn alle Hidden Layer nur ein Neuron hätten[38]. Einem solchen Netz würde es an Lernkapazität mangeln.[39] Daher werden häufig Zufallswerte einer Normalverteilung verwendet. Hier kommt es auf die Werte des Erwartungswerts μ und der Standardabweichung σ der Normalverteilung an. Werden diese beispielsweise auf $\mu = 0$ und $\sigma = 1$ gesetzt, kann es zu einer Sättigung (Siehe 6.4.4) kommen, da der Betrag von $z_{(l,n)}$ wahrscheinlich hoch ist, da die Standardabweichung von $z_{(l,n)}$ bei beispielsweise 500 Weights und einem Bias $\sqrt{1 \cdot 500 + 1} \approx 22,38$ ist. Eine Möglichkeit wäre nun, für $\mu = 0$ und $\sigma = \frac{1}{\sqrt{\text{len}(l-1)}}$ einzusetzen, da die Standardabweichung von $z_{(l,n)}$ im genannten Beispiel nur $\sqrt{\frac{1}{500} \cdot 500 + 1} = \sqrt{2}$ beträgt. Diese Regelung gilt nur für die Weights. Für die Biases werden oft die Zufallswerte einer Normalverteilung mit $\mu = 0$ und $\sigma = 1$ verwendet.[40]

6.4 Probleme beim Trainieren

Das Trainieren von KNNs ist oft mit zusätzlichen Aufgaben und Problemen verbunden. Vier der verbreiteteren Probleme werden im folgenden Unterkapitel angesprochen. Für viele Spezialfälle von KNNs gibt es eigene Schwierigkeiten, welche das Ausmaß dieser Arbeit allerdings bei Weitem übersteigen würden.[41]

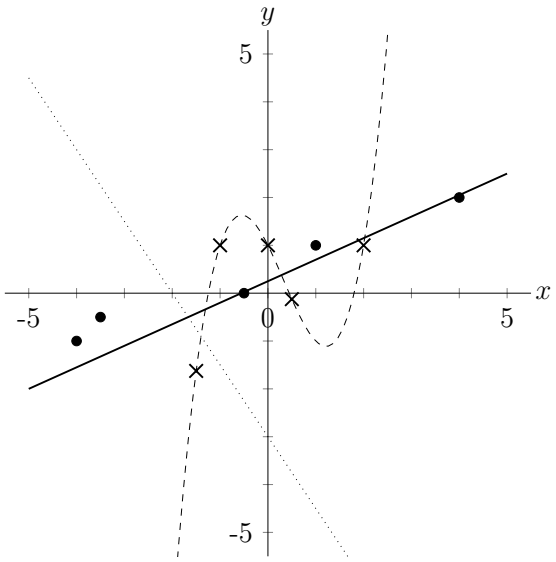


Abb. 12: Die gepunktete Funktion repräsentiert Underfitting, die strichlierte Overfitting und die durchgehende ein für die Aufgabe passendes KNN. Die Kreuze stellen die Trainingsdaten, die Punkte die Testdaten dar. (Quelle: E. D.)

6.4.1 Under- und Overfitting

Beim Trainieren eines KNNs geht es um das Anpassen der Parameter, damit das Netz die Trainingsbeispiele richtig klassifiziert. Je besser allerdings das Netz an diese Beispiele angepasst ist, desto schlechter verhält sich das Netz mit Daten, welche es zuvor nicht gesehen hat.

[38] Angenommen alle Biases hätten den Wert 0
[39] vgl. Rashid, 2017, S. 93; ebd., S. 158
[40] vgl. ebd., Kapitel 3/Weight initialization
[41] vgl. Rashid, 2017, S. 156; Nielsen, 2015, Kapitel 5/Other obstacles to deep learning

Sobald sich die Cost-Funktion einem Minimum nähert, die Trefferquote bei den Testbeispielen aber nicht mehr steigt, spricht man von Overfitting.[42]
Underfitting ist das Gegenteil von Overfitting. Es tritt auf, wenn das Netz zu wenig an die Trainingsbeispiele angepasst und die Eingabe nicht in Verbindung mit der Ausgabe setzen kann.[43] Abb. 12 veranschaulicht den Unterschied anhand einer Regressions-Aufgabe.

6.4.2 Vanishing Gradient

Das Problem des Vanishing Gradient tritt v. a. bei KNNs mit mehreren Hidden Layern auf. Durch das Rückführen des Fehlers mit Formel 18 nähern sich die partiellen Ableitungen für Parameter in den ersten Hidden Layern immer mehr dem Wert 0, wodurch diese Layer langsamer lernen als die Hidden Layer nahe dem Output Layer.[44] Dieses Problem tritt auch bei RNNs auf, da bei diesen mit steigender Länge der Sequenz einer Eingabe die Beträge der partiellen Ableitungen für die Parameter in den ersten Zeitschritten immer kleiner werden.[45]

6.4.3 Dying ReLU

Dieses Problem betrifft normale ReLUs[46]. Es besteht die Möglichkeit, dass $z_{l,n}$ einen Wert kleiner 0 annimmt, wodurch die Ausgabe $x_{(l,n)}$ den Wert 0 annimmt. In diesem Fall schafft es der Backpropagation Algorithmus nicht mehr die Gewichte zu diesem Neuron anzupassen, da Formel 20 ebenfalls immer den Wert 0 erzeugt. Um dieses Problem zu umgehen gibt es die Leaky ReLUs, welche von diesem Problem nicht betroffen sind.[47]

6.4.4 Sättigung

Ein Neuron gilt dann als gesättigt, wenn sich $f'_{(l)}(z_{(l,n)})$ dem Wert 0 annähert. In diesem Fall können die Parameter zu Beginn des Trainings nur in sehr kleinen Schritten verändert werden, da $\delta_{(l,n)}$ durch die geringe Steigung der Aktivierungsfunktion ebenfalls relativ klein ist. Von diesem Problem sind allerdings nur bestimmte Aktivierungsfunktionen, wie die Sigmoid- oder TanH-Funktion, betroffen. Die Rectified Linear-Funktion und ihre Abwandlung haben diesen Nachteil nicht.[48]

[42] vgl. Nielsen, 2015, Kapitel 3/Overfitting and regularization
[43] vgl. Gibson & Patterson, 2017, S. 26-27; Wartala, 2018, S. 195
[44] vgl. Nielsen, 2015, Kapitel 5
[45] vgl. Gupta, 2017
[46] Als Rectified Linear Units (kurz ReLUs) werden Neurone mit der Rectified Linear-Funktion als Aktivierungsfunktion bezeichnet (vgl. Gibson & Patterson, 2017, S. 70)
[47] vgl. Gibson & Patterson, 2017, S. 243
[48] vgl. Nielsen, 2015, Kapitel 2/The four fundamental equations behind backpropagation

7 Aufbau und Durchführung der Experimente

Dieser Teil der Arbeit befasst sich mit praktischen Experimenten zu den in den vorherigen Kapiteln behandelten Typen von KNNs. Ziel der Experimente ist es weniger, neue Rekorde aufzustellen[1], sondern herauszufinden, wie die einzelnen Hyperparameter die Leistung[2] der verschiedenen Typen von KNNs beeinflussen und mit welchen Werten eine besonders hohe Trefferquote[3] erzielt werden kann.

7.1 Aufbau und Durchführung

7.1.1 Untersuchte Hyperparameter

Bei den Versuchen werden die Werte verschiedener, allerdings nicht aller, Hyperparameter verändert. So werden für alle Typen die Lernrate und die Größe der Mini-Batches verändert. Des Weiteren wird auch der Aufbau der KNNs verändert, wobei die verschiedenen Modelle in Tabellen in Anhang C zu finden sind. Nicht verändert wird bei den Experimenten u. a. die Anzahl der Epochs oder die Cost-Funktion.[4]

Speziell bei den FFNNs wird der Einfluss verschiedener Aktivierungsfunktionen und das Benutzen von Biases getestet. Bei den Experimenten zu CNNs haben diese Hyperparameter fixe Werte, der Fokus liegt hier v. a. auf der Größe des Kernels und dem Stride-Wert, da sonst durch die Kombination weiterer Hyperparameter die Experimente zu viel Zeit in Anspruch genommen hätten. Des Weiteren ist auch das Zero Padding so eingestellt, dass die Eingaben so gepaddet werden, dass die Feature Maps gleichgroß wie die Eingaben sind. Bei LSTM Netzen wird der Einfluss der Aktivierungsfunktion und der rekurrenten Aktivierungsfunktion[5] erforscht.

[1] Hierfür hätte man sich mit der Optimierung des Trainings auseinandersetzen müssen

[2] Als Leistung wird hier die Trefferquote eines KNNs bei einem vollständigen Durchlauf aller Beispiele des Testdatensatzes bezeichnet

[3] Die Trefferquote bezieht sich hier nur auf den Testdatensatz, außer es wird explizit von einer anderen Menge gesprochen

[4] Das liegt daran, dass ihre Anzahl in Zusammenhang mit der Lernrate und der Größe der Mini-Batches steht (vgl. Rashid, 2017, S. 157)

[5] $f_{i;(l)}$, $f_{o;(l)}$, $f_{u;(l)}$ werden nachfolgend als rekurrente Aktivierungsfunktionen, $f_{x;(l)}$ und $f_{z;(l)}$ als Aktivierungsfunktionen bezeichnet. Dass die einzelnen Funktionen bei den Experimenten nicht getrennt voneinander behandelt werden liegt an einer Limitierung der Programmierung.

7.1.2 Python und Keras

Umgesetzt werden die Experimente mit der Programmiersprache Python, welche für die Arbeit mit KNNs relativ häufig eingesetzt wird.[6] Außer Python wird die Library Keras verwendet, welche eine kompakte Schreibweise ermöglicht und durch Verwendung einer weiteren Library namens Tensorflow die nötigen Berechnungen beschleunigt.[7]

7.1.3 Aufbau des Programmcodes

Der Programmcode für alle Netztypen ist gleich strukturiert. Zunächst wird, neben einigen nebensächlichen Aufgaben, eine Klasse definiert, welche die Arbeit des Erstellens und Trainierens eines KNNs sowie das Niederschreiben der Ergebnisse in CSV-Dateien (die wichtigsten Tabellen sind in Anhang E zu finden) übernimmt. Anschließend wird mit mehreren Schleifen durch die verschiedenen, festgelegten Werte für die Hyperparameter iteriert. Dieses Verfahren wird als Grid Search bezeichnet.[8]

7.1.4 Durchführung

Durchgeführt wurden die Experimente auf einem Heimcomputer mit etwa 190 GFLOPS[9]. Dabei wurden 378 FFNNs, 576 CNNs und 144 LSTMs in etwa 15 Stunden Rechenzeit getestet.

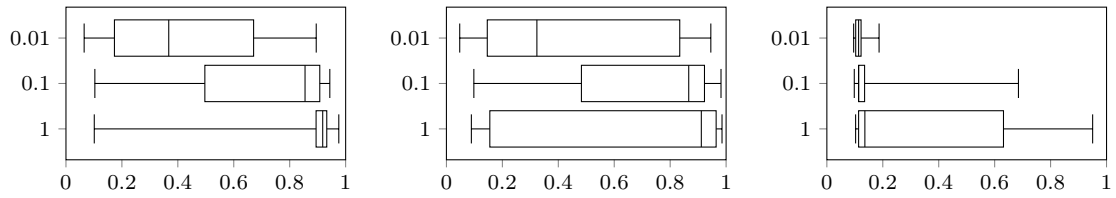
7.2 Auswertung der Ergebnisse

In den folgenden Unterpunkten werden die Auswirkungen der Hyperparameter auf die Trefferquoten der Netze analysiert. Die Boxplots der Abbildungen 13 - 24 visualisieren die Ergebnisse aus den Experimenten.

7.2.1 Allgemeine Erkenntnisse zu Lernrate und Mini-Batch Größe

Die Versuche zu allen drei Netztypen haben bezüglich Lernrate und Größe der Mini-Batches bis auf eine Abweichung die gleichen Ergebnisse geliefert. Die leistungsstärksten KNNs der jeweiligen Netztypen haben alle den Wert 1, den größten der drei möglichen, für die Learning Rate verwendet, wobei mit sinkender Lernrate auch die durchschnittliche Trefferquote sank.

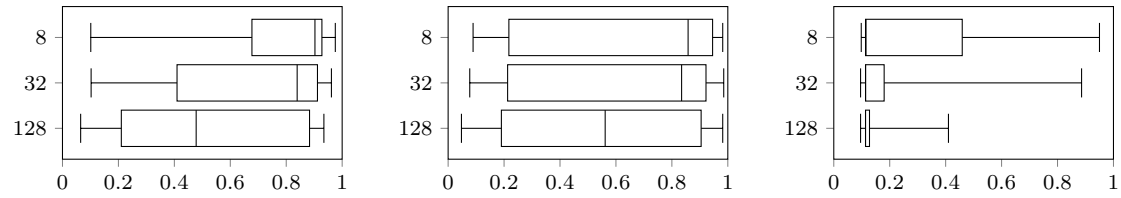
[6] vgl. Rashid, 2017, S. 95
[7] vgl. Wartala, 2018, S. 124
[8] vgl. Buduma, 2017, S. 32
[9] Messung eines Intel Core i7-7700K Prozessors mit dem Intel MKL Linpack Benchmark, verfügbar unter: https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite (Zuletzt besucht am 20. 01. 2019)



(a) FFNNs **(b)** CNNs **(c)** LSTMs

Abb. 13: Boxplots der Trefferquoten der verschiedenen KNN-Arten mit verschiedenen Lernraten (Quelle: Eigene Darstellungen)

Als Größe für die Mini-Batches hat sich bei den FFNNs und LSTMs der Wert 8 durchgesetzt, bei den CNNs verwendete das Netz mit der höchsten Trefferquote eine Mini-Batch Größe von 32, wobei es zum besten CNN mit der Mini-Batch Größe 8 weniger als einen halben Prozentpunkt Unterschied gab.



(a) FFNNs **(b)** CNNs **(c)** LSTMs

Abb. 14: Boxplots der Trefferquoten der verschiedenen KNN-Arten mit verschiedenen Mini-Batch Größen (Quelle: Eigene Darstellungen)

7.2.2 Experimente zu FFNNs

Bezüglich des Aufbaus der FFNNs fällt auf, dass das beste FFNN (Nr. 109, 97,55%) nur einen Hidden Layer mit 1000 Neuronen verwendet, welches jenes Modell mit den meisten Parametern ist.

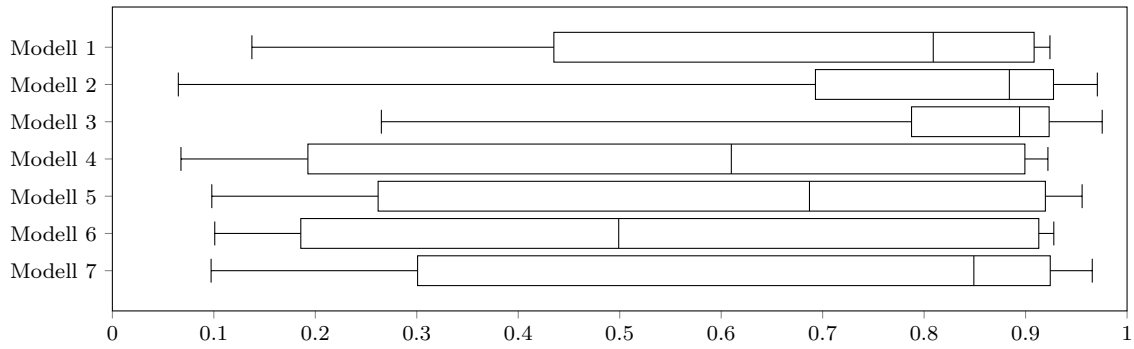


Abb. 15: Boxplots der Trefferquoten der verschiedenen Aufbauten von FFNNs (Quelle: E. D.)

Neben verschiedenen Aufbauten wurden auch drei unterschiedliche Aktivierungsfunktionen getestet, wobei jenes Netz mit der höchsten Trefferquote die Rectified Linear-Funktion verwendet, jedoch schnitt die TanH-Funktion im Durchschnitt mit

einer um zehn Prozentpunkte höheren Trefferquote deutlich besser ab, das beste FFNN mit der TanH-Funktion liegt nur etwa einen Prozentpunkt dahinter. Die Sigmoid-Funktion schnitt am schlechtesten ab.

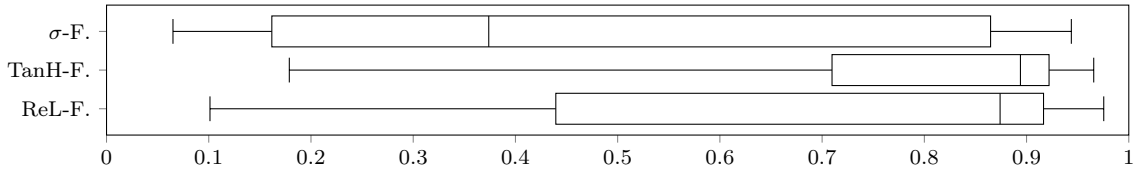


Abb. 16: Boxplots der Trefferquoten von FFNNs mit verschiedenen Aktivierungsfunktionen (Quelle: E. D.)

Des Weiteren wurde auch der Einsatz von Biases getestet, wobei auffiel, dass die durchschnittliche Trefferquote bei FFNNs ohne Biases um ca. 1,6 Prozentpunkte höher ist als bei jenen mit Biases. Bei den beiden besten FFNNs gibt es einen Unterschied von weniger als einem halben Prozentpunkt. Insgesamt gibt es also keinen großen Unterschied, ob man FFNNs mit oder ohne Biases trainiert, was erklären könnte, warum dieser Parameter nicht in jeder Literatur verwendet wird.

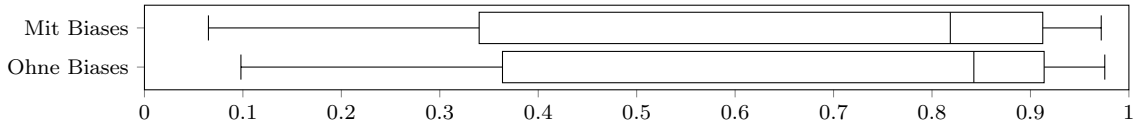


Abb. 17: Boxplots der Trefferquoten von FFNNs mit und ohne Biases (Quelle: E. D.)

7.2.3 Experimente zu CNNs

Interessanterweise hatten CNNs mit einfacheren Aufbauten (Modell 1) eine höhere durchschnittliche Trefferquote als andere, das beste CNN (Nr. 173; 98,54%) verwendet zwei hintereinanderliegende Convolutional Layer, gefolgt von einem Pooling Layer. Jedoch liegen die besten Netze der vier Modelle keinen halben Prozentpunkt auseinander.

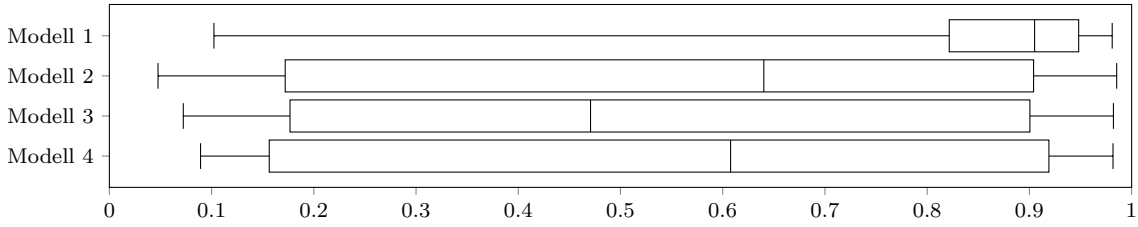


Abb. 18: Boxplots der Trefferquoten der verschiedenen Aufbauten von CNNs (Quelle: E. D.)

Die Größe der Kernels hat kaum einen Einfluss, weder beim Durchschnitt noch bei den besten CNNs gibt es nicht mehr als zwei Prozentpunkte Unterschied, das beste Netz verwendet 3×3 große Kernels.

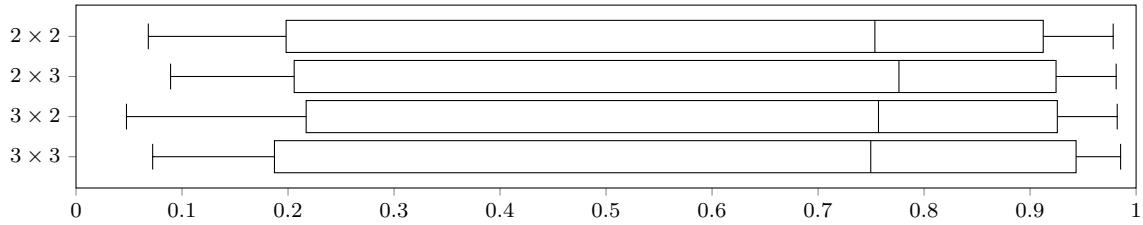


Abb. 19: Boxplots der Trefferquoten von CNNs mit unterschiedlichen Kernels (Quelle: E. D.)

Viel einflussreicher ist der Stride-Wert, wobei mit größeren Stride-Werten die durchschnittliche Trefferquote der CNNs sinkt, weshalb das leistungsstärkste CNN auch den kleinsten Stride-Wert $(1, 1)$ verwendet. Dies lässt sich möglicherweise dadurch erklären, dass Netze mit größerem Stride-Wert kleinere Feature Maps haben und deshalb auch weniger Parameter verwenden, wodurch wiederum die Lernkapazität eingeschränkt ist.

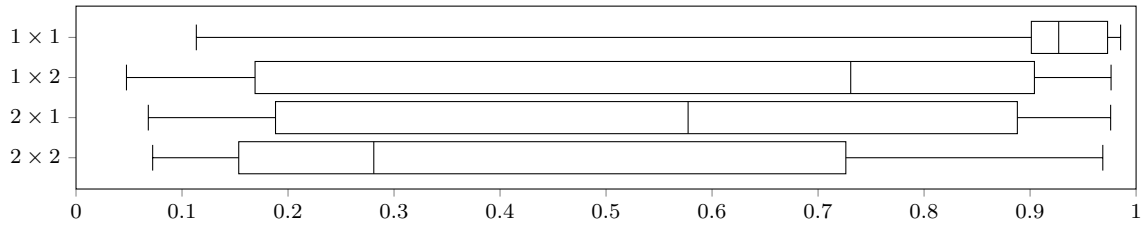


Abb. 20: Boxplots der Trefferquoten von CNNs mit unterschiedlichen Stride-Werten (Quelle: E. D.)

7.2.4 Experimente zu LSTMs

Bei den LSTMs haben zwar Modelle mit zwei LSTM-Schichten durchschnittlich schlechter abgeschnitten als jene mit nur einem, jedoch haben diese höhere maximale Trefferquoten erzielt und das beste Netz hervorgebracht (Nr. 39; 95,03%). Das Verwenden eines Fully-Connected Layers direkt nach dem Input Layer scheint die Leistung eines LSTMs nur zu beeinträchtigen.

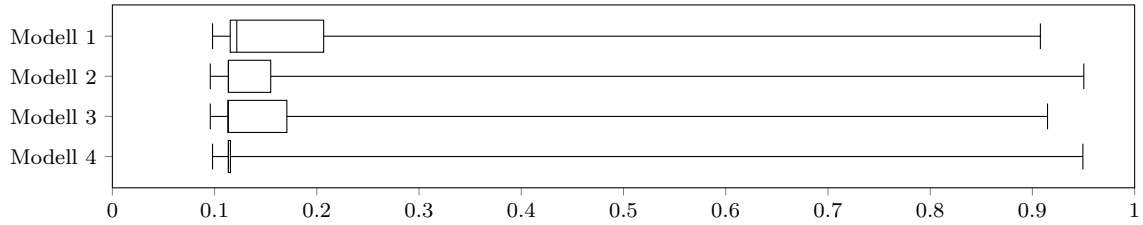


Abb. 21: Boxplots der Trefferquoten der verschiedenen Aufbauten von LSTM Netzen (Quelle: E. D.)

Als Funktion für $f_{x;(l)}$ und $f_{z;(l)}$ ist die TanH-Funktion deutlich besser geeignet als die Sigmoid-Funktion, bei der rekurrenten Aktivierungsfunktion ist es jedoch genau umgekehrt. Für diese ist die Sigmoid-Funktion besser geeignet.

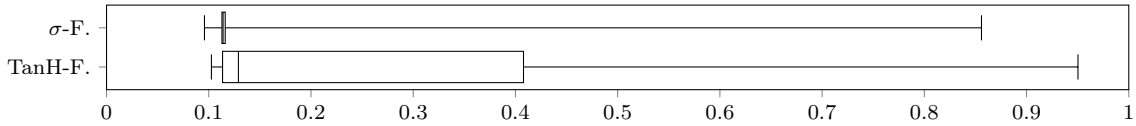


Abb. 22: Boxplots der Trefferquoten von LSTMs mit unterschiedlichen Aktivierungsfunktionen (Quelle: E. D.)

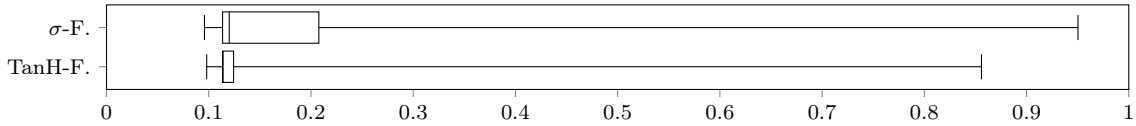


Abb. 23: Boxplots der Trefferquoten von LSTMs mit unterschiedlichen rekurrenten Aktivierungsfunktionen (Quelle: E. D.)

Betrachtet man nun die vier möglichen Kombinationen der Funktionen, so schneiden LSTM Netze, welche für die Aktivierungsfunktion und die rekurrente Aktivierungsfunktion die gleiche Funktion einsetzen, schlechter ab, als LSTMs bei denen sich diese unterscheiden.

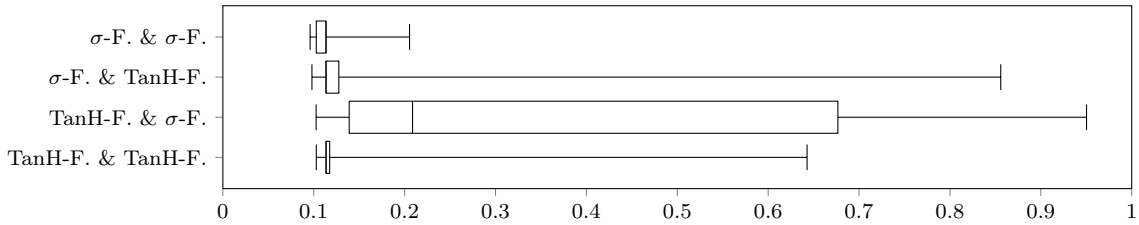


Abb. 24: Boxplots der Trefferquoten der LSTM Netze mit den vier möglichen Kombinationen (Aktivierungsfunktion links, rekurrekte Aktivierungsfunktion rechts) (Quelle: E. D.)

7.3 Erkenntnisse über den MNIST-Datensatz

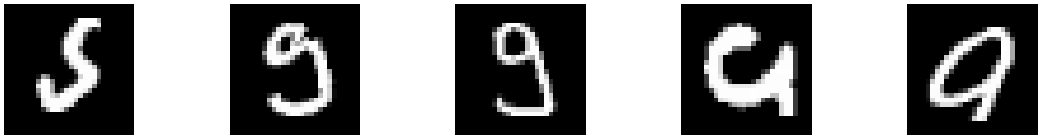
Die Abbildungen 25 - 27 zeigen jeweils jene fünf Ziffern, welche vom jeweiligen Netztypus am häufigsten falsch klassifiziert wurden. In der jeweiligen Unterbildunterschrift steht die Nummer der MNIST-Ziffer, gefolgt von der dargestellten Ziffer und die Anzahl der KNNs des Netztypus, welche diese Ziffer nicht erkannt haben. Dabei fällt auf, dass die Ziffern Fünf und Neun häufig vorkommen und die Ziffer Nr. 9982 in allen drei Listen vorkommt. Betrachtet man die Ziffern genauer, fällt es möglicherweise auch einem Menschen schwer, genau zu sagen, um welche Ziffer es sich handelt. Die Abbildungen 25a und 27d beispielsweise könnten auch ungenau

geschriebene Nullen sein. Es ließe sich bei diesen Beispielen die Frage stellen, wo man genau die Grenze zwischen einer undeutlich geschriebenen Sechs und einer undeutlich geschriebenen Null zieht (Siehe Abb. 28). Ein weiteres Beispiel: Das beste KNN, welches diese Arbeit hervorbrachte, ist ein CNN mit einer Trefferquote von 98,54%, d.h. nur 146 Ziffern der 10.000 Testbeispiele wurde falsch kategorisiert, welche in Abb. 29 dargestellt sind.

Um nun Einzuschätzen, wie gut die besten Netze der drei Netztypen sind, werden diese mit vom Aufbau vergleichbaren Netzen der Ersteller des MNIST-Datensatzes verglichen. Diese erzielen Trefferquoten zwischen 95,3% und bis zu 99,77%, d.h. die besten KNNs dieser Arbeit sind durchaus auf Niveau anderer Forschungsergebnisse.[10] Auch Menschen können keine Trefferquote von 100% erzielen. Die sogenannte Human-Level Performance für MNIST liegt bei etwa 99,75%.[11]

(a) 2118: 6 (377) **(b)** 3853: 6 (376) **(c)** 9982: 5 (375) **(d)** 9729: 5 (375) **(e)** 6166: 9 (375)

Abb. 25: Top 5 der von FFNNs falsch erkannten Ziffern (Quelle: E. D.)



(a) 9982: 5 (568) **(b)** 1247: 9 (564) **(c)** 3597: 9 (562) **(d)** 2293: 9 (560) **(e)** 6505: 9 (559)

Abb. 26: Top 5 der von CNNs falsch erkannten Ziffern (Quelle: E. D.)

(a) 9770: 5 (144) **(b)** 3893: 5 (144) **(c)** 2573: 5 (144) **(d)** 1299: 5 (144) **(e)** 9982: 5 (143)

Abb. 27: Top 5 der von LSTMs falsch erkannten Ziffern (Quelle: E. D.)

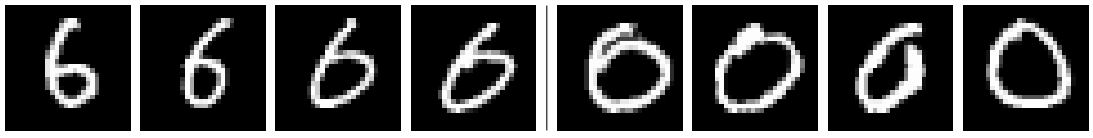


Abb. 28: Verlauf von Sechs zu Null. Die Ziffern links vom Strich sind Sechsen, rechts davon Nullen des Testdatensatzes (Quelle: E. D.)

[10] vgl. Burges, Cortes & LeCun, 1998
[11] vgl. Wartala, 2018, S. 126f

Abb. 29: Alle 146 Ziffern, die das beste KNN (CNN Nr. 173) nicht richtig klassifizieren konnte. Links steht die korrekte Antwort, rechts jene des KNN
(Quelle: E. D.)

8 Resümee

Die Ziele dieser Arbeit sind einerseits, die Funktionsweise von verschiedenen Arten von KNNs zu verstehen, und andererseits herauszufinden, wie sich diese beim MNIST-Datensatz verhalten.

Im zweiten Kapitel werden die Neurone und ihre Gruppierung in verschiedene Schichten als grundlegendes Konzept für die KNNs dieser Arbeit erklärt. Kapitel 3 beschreibt den Aufbau von FFNNs und wie man die Ergebnisse eines solchen Netzes berechnet. Die im vierten Kapitel beschriebenen CNNs bauen auf dem FFNN auf und erweitern dieses durch spezielle Schichten, den Convolutional und Pooling Layern. Die LSTMs bearbeiten die Eingabedaten über mehrere Zeitschritte hinweg und können so durch Eingaben aus der Vergangenheit beeinflusst werden. Kapitel 6 zeigt anhand der FFNNs, wie diese durch iterative Algorithmen lernen, dass der Erfolg dieses Prozesses nicht nur vom Aufbau eines KNNs, sondern auch von den Hyperparametern abhängt und welche Probleme beim Lernen auftreten können.

Im siebten Kapitel werden die Ergebnisse der Experimente der verschiedenen Netztypen analysiert. Dabei fällt auf, dass alle drei Netztypen trotz ihrer Unterschiede erfolgreiche Netze hervorgebracht haben. Des Weiteren zeigt sich auch der Zusammenhang zwischen der Learningrate, der Größe der Mini Batches und der Anzahl der Epochs. Bezüglich der FFNNs fällt auf, dass ihre Leistung v. a. vom Aufbau und der verwendeten Aktivierungsfunktion, nicht jedoch von der Verwendung von Biases abhängt. Die Experimente mit CNNs zeigen, dass die Trefferquote bei der Testmenge neben dem Aufbau hauptsächlich von der Stride-Größe, nicht jedoch der Größe der Kernels, abhängt. Die Leistung von LSTMs wird v. a. durch den Aufbau und die Kombination der verschiedenen Aktivierungsfunktionen beeinflusst.

An dieser Stelle lässt sich anmerken, dass man sich einerseits noch intensiver mit z. B. anderen Optimierungsmöglichkeiten des Lernprozesses hätte auseinandersetzen können und andererseits, dass die Experimente in einem größeren Umfang durchgeführt hätten werden können. So hätte man mehr Kombinationen und den Einfluss von mehr Hyperparametern, wie beispielsweise andere Cost-Funktionen, testen können.

Literaturverzeichnis

[1] BENGIO, Yoshua/COURVILLE, Aaron/GOODFELLOW, Ian: *Deep Learning.* MIT Press, 2016. URL: `http://www.deeplearningbook.org` (Zuletzt besucht am 24. 06. 2018).

[2] BUDUMA, Nikhil/LACASCIO, Nicholas: *Fundamentals of Deep Learning. Designing Nest-Generaation Machine Intelligence Algorithms.* 1. Auflage. Sebastopol: O'Reilly Media, 2017.

[3] BURGES, Christopher J.C./CORTES, Corinna/LECUN, Yann: *The MNIST Database of handwritten digits.* 1998. URL: `http://yann.lecun.com/exdb/mnist/index.html` (Zuletzt besucht am 28. 01. 2018).

[4] GIBSON, Adam/PATTERSON, Josh: *Deep Learning. A Practitioner's Approach.* 1. Auflage. Sebastopol: O'Reilly Media, 2017.

[5] GUPTA, Dishashree: *Fundamentals of Deep Learning – Introduction to Recurrent Neural Networks.* 07. 12. 2017. URL: `https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/` (Zuletzt besucht am 30. 08. 2018).

[6] GURNEY, Kevin: *An introduction to neural networks.* London: UCL Press, 1997.

[7] KARPATHY, Andrej: *The Unreasonable Effectiveness of Recurrent Neural Networks.* 21. 05. 2015. URL: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/` (Zuletzt besucht am 30. 08. 2018).

[8] NASH, Ryan/O'SHEA, Keiron: *An Introduction to Convolutional Neural Networks.* 02. 12. 2015. URL: `https://arxiv.org/pdf/1511.08458.pdf` (Zuletzt besucht am 26. 06. 2018).

[9] NIELSEN, Michael: *Neural Networks And Deep Learning.* Determination Press, 2015. URL: `http://neuralnetworksanddeeplearning.com` (Zuletzt besucht am 02. 07. 2018).

[10] OLAH, Christopher: *Understanding LSTM Networks.* 27. 08. 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (Zuletzt besucht am 30. 08. 2018).

[11] RASHID, Tariq: *Neuronale Netze selbst programmieren. Ein verständlicher Einstieg mit Python*. 1. Auflage. Heidelberg: O'Reilly, 2017.

[12] WARTALA, Ramon: *Praxiseinstieg Deep Learning. Mit Python, Caffe, TensorFlow und Spark eigene Deep-Learning-Anwendungen erstellen*. 1. Auflage. Heidelberg: O'Reilly, 2018.

[13] WU, Jianxin: *Introduction to Convolutional Neural Networks*. 01. 05. 2017. URL: `https://pdfs.semanticscholar.org/450c/a19932fcef1ca6d0442cbf52fec38fb9d1e5.pdf` (Zuletzt besucht am 26. 06. 2018).

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abb. Abbildung

Aktf. Aktivierungsfunktion

CNN Convolutional Neural Network

CSV Comma Separated Values

E. D. Eigene Darstellung

FFNN Feedforward Neural Network

GD Gradient Descent

GFLOPS Giga Floating Point Operations per Second

KNN Künstliches Neuronales Netz(werk)

LRF Local Receptive Field

LSTM Long Short Term Memory (Neural Network)

MNIST Modified National Institute of Standards and Technology (Dataset)

rek. Aktf. rekurrente Aktivierungsfunktion

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SGD Stochastic Gradient Descent

v. a. vor allem

z. B. zum Beispiel

Glossar

A

Aktivierung Zustand eines Neurons; gilt als aktiviert wenn sein Ausgabewert $\neq 0$

Aktivierungsfunktion Funktion, in welche die gewichteten Eingaben samt Bias eingegeben werden und den Ausgabewert des Neurons berechnet

Algorithmus Eine Sequenz von (rechnerischen) Verarbeitungsschritten, die eine Eingabe in eine Ausgabe transformiert

Ausgabewert Wert eines Neurons, welchen es an Neurone der nächsten Schicht weitergibt

Average-Pooling Mögliche Funktion eines Pooling Layer, die jedem seiner Neurone den größten Wert des jeweiligen Local Receptive Fields als Ausgabewert zuordnet

B

Backpropagation Algorithmus zur Berechnung partieller Ableitungen bei mehrschichtigen KNNs

Bias Parameter welcher zu den gewichteten Eingaben eines Neurons addiert wird

Boxplot Graphische Darstellung des Minimums, 1., 2. und 3. Quartils und Maximum einer Liste

C

Classification Aufgabenstellung, bei der die Eingabe in zwei oder mehr Klassen kategorisiert wird

- **Binary Classification** Aufgabenstellungen bei denen entschieden werden muss, ob die Eingabe einer bestimmten Struktur entspricht oder nicht. Oft liegt der Ausgabewert zwischen 0 und 1, weshalb ein Schwellenwert für die Ausgabe festgelegt wird um zwischen den beiden möglichen Kategorien zu unterscheiden
- **Many-To-Many** Eine Sequenz von Eingaben erzeugt eine Sequenz von Ausgaben
- **Many-To-One** Eine Sequenz von Eingaben erzeugt eine Ausgabe
- **Multiclass Classification** Aufgabenstellungen bei denen die Eingabe in eine von mehreren Kategorien eingeordnet werden muss. Die Klasse dessen

entsprechendes Neuron den höchsten Ausgabewert hat ist die Antwort des KNN

One-To-Many Eine Eingabe erzeut eine Sequenz von Ausgaben

Convolutional Neural Network Art von KNNs; besteht grundsätzlich aus drei unterschiedlichen Arten von Layern: Convolutional, Pooling und Fully-connected Layer

CSV-Datei Datei mit einer Tabelle, derer Zellen durch z. B. Kommas getrennt sind

D

Dying ReLU Problem beim Lernen von KNNs mit ReLUs

E

Epoch ein kompletter Durchlauf beim Lernen durch alle Trainingsbeispiele

F

Feature Map (meist) zweidimensionale Matrix von Neuronen eines Convolutional Layer

Feedforward Neural Network KNN mit einem Input, einem Output und einem oder mehreren Hidden Layern, deren Neurone mit allen Neuronen benachbarter Schichten verbunden sind

fully connected zwei (benachbarte) Layer sind fully connected, wenn alle Neurone eines Layers mit allen Neuronen des anderen verbunden sind

G

Gate Steuereinheiten zur Veränderung des Zell Status

Forget Gate Gate zur Bestimmung, welche Informationen des vorherigen Zell Status erhalten bleiben sollen

Input Gate Gate zur Bestimmung, welche neuen Informationen in den aktuellen Zell Status aufgenommen werden sollen

Output Gate Gate zur Bestimmung, welche Informationen an die nächste Schicht und an den nächsten Zeitschritt weitergegeben werden sollen

Gatter Siehe: Gate

Gewicht Siehe: Weight

GFLOPS Milliarden Gleitkommaoperationen pro Sekunde; Einheit zur Messung der Rechengeschwindigkeit von Prozessoren

Gradient Vektor mit partiellen Ableitungen

Gradient Descent Algorithmus zum Trainieren von KNNs

Mini-Batch Gradient Descent Gradient Descent, bei dem nach einer kleinen Teilmenge der Trainingsdaten die Parameter des KNNs verändert werden

Grid Search Suche nach geeigneten Werten für Hyperparameter durch Iterieren durch eine/mehrere Liste/n von verschiedenen Werten

H

Hyperparameter Parameter, die das Trainieren eines KNN steuern

K

Keras Library zur Vereinfachung der Implementierung von KNNs mittels Tensorflow

Kernel Matrix mit den Gewichten zwischen einem Neuron eines Convolutional Layer und seinem dazugehörigen LRF

Künstliches Neuron Mathematische Funktion mit mehreren Parametern

Künstliches Neuronales Netzwerk Rechnerisches Modell, bestehend aus mehreren künstlichen Neuronen

L

Layer Gruppierung von einem oder mehreren Neuronen

- **Convolutional Layer** Typ von Layer, der Grundbestandteil eines CNNs ist; Neurone dieses Layers sind nur mit einem Ausschnitt, dem LRF, des vorherigen Layers verbunden
- **Fully-connected Layer** Layer, bei dem jedes Neuron mit jedem Neuron des vorherigen Layers verbunden ist
- **Hidden Layer** Layer eines KNN, dessen Neurone nicht die finalen Ausgabewerte angeben
- **Input Layer** Erste Layer eines KNN, nimmt die Eingabedaten an und übergibt diese dem ersten Hidden Layer
- **Output Layer** Letzte Layer eines KNN; Neurone geben endültigen Output des KNN an
- **Pooling Layer** Layer eines CNN, dessen Ziel es ist, kleinere Feature Maps als der vorhergehende Layer zu haben
- **Softmax Layer** Output Layer, dessen Ausgabe als Wahrscheinlichkeitsverteilung angesehen werden kann

Leaky Rectified Linear-Funktion Häufige Aktivierungsfunktion, definiert als:

$$f(x) = \begin{cases} k \cdot x & x \leq 0 \\ x & x > 0 \end{cases}$$

Learning Rate Hyperparamter zur Bestimmung der Größe der Veränderungen der Parameter relativ zu dessen partieller Ableitung

Leistung Trefferquote eines KNNs bei einem vollständigen Durchlauf aller Beispiele des Testdatensatzes

Lernen Siehe: Training

Lernrate Siehe: Learning Rate

Library Vorgefertigter Programmcode, auf den ein Programmierer zurückgreifen kann

Local Receptive Field Ausschnitt einer Input Matrix für ein Neuron eines Convolutionsl Layers

Long Short Term Memory (Neural) Network Typ von RNNs, welche LSTM Zellen verwenden

Loss-Funktion Siehe: Cost-Funktion

M

Max-Pooling Mögliche Funktion eines Pooling Layer, die seinen Neuronen die arithemtischen Mittel ihrer jeweiligen Local Receptive Fields als Ausgabewert annehmen lässt

Mini-Batch Kleine Teilmenge von Trainingsdaten

Modified National Institue of Standards and Technology (Datensatz) Datensatz aus insgesamt 70.000 handgeschriebenen, in einer Auflösung von 28 mal 28 Pixel eingescannte Ziffern

N

Neuron Siehe: Künstliches Neuron

Neuronales Netz(werk) Siehe: Künstliches Neuronales Netzwerk

O

Optimierungs-Methode Methode zum Trainieren eines KNN (auch Optimieren der Parameter genannt)

Output Ausgabe eines KNN

Overfitting Problem beim Lernen eines KNN; tritt ein, sobald die Trefferquote bei den Validationsdaten nicht weiter steigt, die Cost-Funktion aber weiterhin sinkt

P

Parameter Weights und Biases eines KNN

Python häufig im Bereich der KNNs eingesetzte Programmiersprache

R

Rectified Linear Unit Neuron, welches die Rectified Linear-Funktion als Aktivierungsfunktion verwendet

Rectified Linear-Funktion Häufige Aktivierungsfunktion, definiert als: $f(x) = \max(0, x)$

Recurrent Neural Network KNNs, deren Ausgabe zu einem Zeitschritt t die Ausgabe zum Zeitschritt $t + 1$ beeinflusst

Regression Aufgabenstellungen bei denen der Zusammenhang zwischen Eingabe und Ausgabe modelliert werden muss um für eine gegebene Eingabe die Ausgabe zu ermitteln

S

Schicht Siehe: Layer

Sigmoid-Funktion Häufige Aktivierungsfunktion, definiert als: $\sigma(x) = \frac{1}{1+e^{-x}}$

Stochastic Gradient Descent Gradient Descent, bei dem die Parameter des KNNs nach jedem Trainingsbeispiel verändert werden

Stride Hyperparameter, welcher angibt, wie eine Input Matrix in die verschiedenen LRFs aufgeteilt werden soll

Sättigung Nähert sich eine Ableitung dem Wert 0, so sind die Veränderungen der Parameter relativ gering und das KNN benötigt zum Lernen mehr Zeit

T

Tensorflow Library von Google für C++/Python zur Implementierung von KNNs

Testdatensatz Menge von Beispielen, deren Eingaben und Ausgaben bekannt sind, welche dazu verwendet werden, um zu überprüfen, wie gut ein KNN mit Daten umgehen kann, die es zuvor noch nie gesehen hat

Training Anpassen der Parameter eines KNN für eine bestimmte Aufgabe mithilfe von Trainingsbeispielen

Trainingsdatensatz Menge von Beispielen, deren Eingaben und Ausgaben bekannt sind, welche zum Trainieren eines KNN verwendet werden

Trefferquote Quotient von Anzahl der erkannten Ziffern einer Menge und Gesamtanzahl der Ziffern einer Menge

U

Underfitting Problem beim Lernen eines KNN; tritt ein, wenn das KNN sich an die Trainingsdaten nicht anpassen kann und eine Eingabe nicht in Verbindung mit der dazugehörigen Ausgabe setzen kann

V

Validationsdatensatz Menge von Daten, deren Eingaben und Ausgaben bekannt sind, welche dazu dienen, um zu überprüfen, ob es beim Trainieren zu Overfitting kommt

Vanishing Gradient Problem beim Lernen von KNNs, bei dem der Gradient für die Parameter von Layern, die näher beim Input Layer liegen, sich dem Wert 0 nähert und das Lernen verlangsamt wird

W

Weight Parameter zur Gewichtung der Eingabe eines künstlichen Neurones

Z

Zell Status Variable eines LSTMs, welche Informationen über mehrere Zeitschritte hinweg speichern kann

Zero Padding Hinzufügen von zusätzlichen, mit dem Wert 0 gefüllten Zeilen und Spalten um eine Input Matrix, um die Größe der Feature Maps des Convolutional Layers zu steuern

Anhang

Anhang A: Notationstabelle

l	Nummer einer Schicht (Indexierung: 1)
L	Letzte Schicht eines KNNs
m	Nummer eines Neurons der Schicht $l-1$ (Indexierung: 1)
n	Nummer eines Neurons der Schicht l (Indexierung: 1)
d	Nummer einer Feature Map in einer Schicht (Indexierung: 1)
t	Variable zur Angabe eines Zeitpunkts (Indexierung: 1)
$w_{(l-1,m),(l,n)}$	Verbindungsgewicht zwischen dem Neuron m der Schicht $l-1$ und dem Neuron n der Schicht l (Skalar)
$\vec{w}_{(l-1),(l,n)}$	Verbindungsgewichte zwischen allen Neuronen der Schicht $l-1$ und dem Neuron n der Schicht l (Vektor)
$W_{(l-1),(l)}$	Verbindungsgewichte zwischen allen Neuronen der Schicht $l-1$ und allen Neuronen der Schicht l; Matrix mit der Form $n \times m$, wobei die m die Anzahl der Neurone in der Schicht $l-1$, n die Anzahl der Neurone der Schicht l angibt
$W^{(t),(t)}_{(l-1),(l)}$	Verbindungsgewichte zwischen allen Neuronen der Schicht $l-1$ zum Zeitpunkt t und allen Neuronen der Schicht l zum Zeitpunkt t; Matrix gleicher Form wie $W_{(l-1),(l)}$
$W^{(t-1),(t)}_{(l),(l)}$	Verbindungsgewichte zwischen allen Neuronen der Schicht l zum Zeitpunkt $t-1$ und allen Neuronen der Schicht l zum Zeitpunkt t; Matrix mit der Form $n \times n$, wobei n die Anzahl der Neurone der Schicht l angibt

$W^{(t),(t)}_{u;(l-1),(l)}$	Verbindungsgewichte zwischen allen Neuronen der Schicht $l-1$ zum Zeitpunkt t und allen Neuronen der Schicht l zum Zeitpunkt t zur Berechnung von $\vec{u}^{(t)}_{(l)}$; Matrix gleicher Form wie $W_{(l-1),(l)}$; gibt auch $W^{(t),(t)}_{i;(l-1),(l)}$, $W^{(t),(t)}_{o;(l-1),(l)}$ und $W^{(t),(t)}_{z;(l-1),(l)}$ zur Berechnung von $\vec{i}^{(t)}_{(l)}$, $\vec{o}^{(t)}_{(l)}$ und $\vec{z}^{(t)}_{(l)}$
$W^{(t-1),(t)}_{u;(l),(l)}$	Verbindungsgewichte zwischen allen Neuronen der Schicht l zum Zeitpunkt $t-1$ und allen Neuronen der Schicht l zum Zeitpunkt t zur Berechnung von $\vec{u}^{(t)}_{(l)}$; Matrix mit der Form $n \times n$, wobei n die Anzahl der Neurone der Schicht l angibt; gibt auch $W^{(t-1),(t)}_{i;(l),(l)}$, $W^{(t-1),(t)}_{o;(l),(l)}$ und $W^{(t-1),(t)}_{z;(l),(l)}$ zur Berechnung von $\vec{i}^{(t)}_{(l)}$, $\vec{o}^{(t)}_{(l)}$ und $\vec{z}^{(t)}_{(l)}$
$W_{(l),(l+1)}{}^{\intercal}$	Transponierte Matrix mit den Verbindungsgewichten zwischen allen Neuronen der Schicht $l-1$ und allen Neuronen der Schicht l; Matrix mit der Form $m \times n$, wobei die m die Anzahl der Neurone in der Schicht $l-1$, n die Anzahl der Neurone der Schicht l angibt
$b_{(l,n)}$	Bias des Neurons n in der Schicht l
$\vec{b}_{(l)}$	Vektor mit allen Bias-Werten der Schicht l
$b_{(l,d)}$	Bias, welcher für alle Neurone der Feature Map d der Schicht l verwendet wird
$z_{(l,n)}$	Wert des Neurons n der Schicht l vor Anwendung einer Aktivierungsfunktion
$\vec{z}_{(l)}$	Vektor mit allen Werten der Neurone der Schicht l vor Anwendung einer Aktivierungsfunktion
$f_{(l)}$	Aktivierungsfunktion der Schicht l
$f'_{(l)}$	Erste Ableitung der Aktivierungsfunktion der Schicht l
$f_{i;(l)}$ $f_{o;(l)}$ $f_{u;(l)}$	Rekurrente Aktivierungsfunktionen der Gates der Schicht l eines LSTM Netzes (meistens Sigmoid-Funktion)
$f_{x;(l)}$ $f_{z;(l)}$	Aktivierungsfunktionen der Schicht l bei LSTM Zellen(meistens TanH-Funktion)
$x_{(l,n)}$	Ausgabewert des Neurons n der Schicht l

$\vec{x}_{(l)}$	Vektor mit allen Ausgabewerten aller Neurone der Schicht l
$x_{(l,d,n)}$	Ausgabewert des Neurons n in der Feature Map d der Schicht l
$x^{(t)}_{(l,n)}$	Ausgabewert des Neurons n der Schicht l zum Zeitpunkt t
$\vec{x}^{(t)}_{(l)}$	Vektor mit allen Ausgabewerten der Neurone der Schicht l zum Zeitpunkt t
$R_{(l,d,n)}$	Local Receptive Field (LRF); Ausschnitt aus der Schicht $l-1$ mit dem das Neuron n der Feature Map d der Schicht l verbunden ist
$K_{(l,d)}$	Verbindungsgewichte, mit denen alle LRFs der Feature Map d der Schicht l gewichtet werden
$s_{(l)}$	Stride-Wert der Schicht l; gibt an, in welchem Abstand sich die LRFs voneinander befinden
$p_{(l)}$	Padding-Wert der Schicht l, gibt an, wie viele zusätzliche Zeilen und Spalten um die Matrix der Schicht $l-1$ mit dem Wert 0 hinzugefügt werden sollen
$\vec{c}^{(t)}_{(l)}$	Zell Status der Schicht l zum Zeitpunkt t
$\vec{i}^{(t)}_{(l)}$	Vektor, welcher angibt, welche Werte von $\vec{z}^{(t)}_{(l)}$ zum Zell Status hinzugefügt werden sollen
$\vec{o}^{(t)}_{(l)}$	Vektor, welcher angibt, welche
$\vec{u}^{(t)}_{(l)}$	Vektor, welcher angibt, welche Werte des vorherigen Zell Status zum Zeitpunkt $t-1$ zur Berechnung des Zell Status zum Zeitpunkt t verwendet werden
$\vec{z}^{(t)}_{(l)}$	Vektor mit den zum Zell Status neu hinzukommenden Werten
C	Cost-Funktion
Tr	Anzahl der Trainingsbeispiele
$\vec{x}_{(1)}(\mathrm{tr}:i)$	Eingabe des i-ten Trainingsbeispiels
$\vec{y}_{(1)}(\mathrm{tr}:i)$	Gewünschte Ausgabe des i-ten Trainingsbeispiels
η	Lernrate; gibt an, wie groß die Veränderung der Parameter relativ zu deren partiellen Ableitungen sein sollen
$\frac{\partial a}{\partial b}$	Partielle Ableitung von a in Abhängigkeit von b
$\delta_{(l,n)}$	Fehler des Neurons n der Schicht l

$\vec{\delta}_{(l)}$	Vektor mit allen Fehlern aller Neurone der Schicht l
$\nabla_{\vec{x}_{(L)}} C$	Gradient; Vektor dessen Elemente die partiellen Ableitungen $\frac{\partial C}{\partial x_{(L,n)}}$ sind; zeigt in die Richtung des steilsten Anstiegs einer Funktion
$\langle A, B \rangle_F$	Frobenius-Skalarprodukt; Zwei Matrizen gleicher Dimensionen werden elementweise miteinander multipliziert und die Produkte aufsummiert und ergeben ein Skalar
$\odot$	Hadamard Produkt; Zwei Vektoren gleicher Dimension werden elementweise miteinander multipliziert und ergeben einen Vektor mit der gleichen Dimension
$\mathrm{H}(X)$ $\mathrm{W}(X)$ $\mathrm{D}(X)$	Gibt die Größe der ersten/zweiten/dritten Dimension einer Matrix X an
$\mathrm{len}(\vec{x})$	Gibt die Anzahl der Elemente des Vektors $\vec{x}$ an
$\max(a, b)$	Funktion, welche zwei Werte a und b vergleicht und den größeren der beiden als Funktionswert annimmt

Tabelle 1: Notationstabelle (Quelle: Eigene Tabelle)

Anhang B: Beweise für die Formeln von Backpropagation

Backpropagation Formel 1

Die erste Backpropagation Formel dient zur Berechnung eines Vektors mit den Fehlern der Neurone der letzten Schicht.

$$\vec{\delta}_{(L)} = \nabla_{\vec{x}_{(L)}} C \odot f'_{(L)}(\vec{z}_{(L)}) \tag{21}$$

Diese lässt sich für einzelne Fehler $\delta_{(L,n)}$ umschreiben:

$$\delta_{(L,n)} = \frac{\partial C}{\partial x_{(L,n)}} \cdot f'_{(L)}(z_{(L,n)}) \tag{22}$$

Ursprünglich wird in Kapitel 6.2 der einzelne Fehler $\delta_{(l,n)}$ folgendermaßen definiert:

$$\delta_{(l,n)} = \frac{\partial C}{\partial z_{(l,n)}} \tag{23}$$

Diese Formel lässt sich umschreiben als:

$$\delta_{(l,n)} = \frac{\partial C}{\partial x_{(l,n)}} \cdot \frac{\partial x_{(l,n)}}{\partial z_{(l,n)}} \tag{24}$$

Da Folgendes gilt...

$$\frac{\partial x_{(l,n)}}{\partial z_{(l,n)}} = \frac{\partial f_{(l)}(z_{(l,n)})}{\partial z_{(l,n)}} = f'_{(l)}(z_{(l,n)}) \tag{25}$$

...lässt sich dies in Formel 24 einsetzen, woraus Formel 22 folgt.

Backpropagation Formel 2

Diese Formel dient zur Berechnung der Fehler einer Schicht l in Abhängigkeit der Fehler der nachfolgenden Schicht $l + 1$.

$$\vec{\delta}_{(l)} = (W_{(l),(l+1)}{}^{\intercal} \cdot \vec{\delta}_{(l+1)}) \odot f'_{(l)}(\vec{z}_{(l)}) \tag{26}$$

Diese kann für einen einzelnen Fehler $\delta_{(l,n)}$ des Neurons n der Schicht l umgeformt werden, wobei der Vektor $\vec{w}_{(l,n),(l+1)}$ alle Gewichte, welche vom Neuron n der Schicht l zu allen Neuronen der Schicht $l+1$ gehen, beinhaltet.

$$\delta_{(l,n)} = \vec{w}_{(l,n),(l+1)} \cdot \vec{\delta}_{(l+1)} \cdot f'_{(l)}(z_{(l,n)}) \tag{27}$$

Ursprünglich wird in Kapitel 6.2 der einzelne Fehler $\delta_{(l,n)}$ folgendermaßen definiert:

$$\delta_{(l,n)} = \frac{\partial C}{\partial z_{(l,n)}} \tag{28}$$

Dieser lässt sich umschreiben als:

$$\delta_{(l,n)} = \frac{\partial C}{\partial \vec{z}_{(l+1)}} \cdot \frac{\partial \vec{z}_{(l+1)}}{\partial z_{(l,n)}} \tag{29}$$

Da $\frac{\partial C}{\partial \vec{z}_{(l+1)}} = \vec{\delta}_{(l+1)}$, lässt sich dies in die vorherige Formel einsetzen:

$$\delta_{(l,n)} = \frac{\partial \vec{z}_{(l+1)}}{\partial z_{(l,n)}} \cdot \vec{\delta}_{(l+1)} \tag{30}$$

Aus der Formel von $z_{(l+1,m)}$

$$\begin{aligned} z_{(l+1,m)} &= \vec{w}_{(l),(l+1,n)} \cdot \vec{x}_{(l)} + b_{(l+1,n)} \\ &= \vec{w}_{(l),(l+1,n)} \cdot f_{(l)}(\vec{z}_{(l)}) + b_{(l+1,n)} \\ &= \sum_{i=1}^{j} (w_{(l,i),(l+1,m)} \cdot f_{(l)}(z_{(l,i)})) + b_{(l+1,m)} \end{aligned} \tag{31}$$

wobei j hier die Anzahl aller Neuronen der Schicht l ist, lässt sich folgende partielle Ableitung bilden:

$$\frac{\partial z_{(l+1,m)}}{\partial z_{(l,n)}} = w_{(l,n),(l+1,m)} \cdot f'_{(l)}(z_{(l,n)}) \tag{32}$$

Diese partielle Ableitung lässt sich auch für den Vektor $\vec{z}_{(l+1)}$ bilden:

$$\frac{\partial \vec{z}_{(l+1)}}{\partial z_{(l,n)}} = \vec{w}_{(l,n),(l+1)} \cdot f'_{(l)}(z_{(l,n)}) \tag{33}$$

Setzt man diese nun in Formel 30 ein, so erhält man Formel 27.

Backpropagation Formel 3

Die dritte Backpropagation Formel gibt an, wie sich der Wert der Cost-Funktion in Abhängigkeit des Bias des Neurons n der Schicht l verändert.

$$\frac{\partial C}{\partial b_{(l,n)}} = \delta_{(l,n)} \tag{34}$$

Dieser Bruch lässt sich erweitern:

$$\frac{\partial C}{\partial x_{(l,n)}} \cdot \frac{\partial x_{(l,n)}}{\partial z_{(l,n)}} \cdot \frac{\partial z_{(l,n)}}{\partial b_{(l,n)}} = \delta_{(l,n)} \tag{35}$$

Die beiden Gleichungen $\frac{\partial x_{(l,n)}}{\partial z_{(l,n)}} = \frac{\partial f_l(z_{(l,n)})}{\partial z_{(l,n)}} = f'_{(l)}(z_{(l,n)})$ und $\frac{\partial z_{(l,n)}}{\partial b_{(l,n)}} = 1$ (da b nur addiert wird, fällt dieser beim Differenzieren weg) lassen sich nun in die vorherige Gleichung einsetzen:

$$\frac{\partial C}{\partial x_{(l,n)}} \cdot f'_{(l)}(z_{(l,n)}) \cdot 1 = \delta_{(l,n)} \tag{36}$$

Da die linke Seite die Definition von $\delta_{(l,n)}$ ist, ist die letzte Gleichung und somit auch die ursprüngliche Formel 34 richtig.

Backpropagation Formel 4

Die letzte Backpropagation Formel gibt an, wie sich der Wert der Cost-Funktion in Abhängigkeit des Gewichts.

$$\frac{\partial C}{\partial w_{(l-1,m),(l,n)}} = \delta_{(l,n)} \cdot x_{(l-1,m)} \tag{37}$$

Dieser Bruch lässt sich erweitern:

$$\frac{\partial C}{\partial x_{(l,n)}} \cdot \frac{\partial x_{(l,n)}}{\partial z_{(l,n)}} \cdot \frac{\partial z_{(l,n)}}{\partial w_{(l-1,m),(l,n)}} = \delta_{(l,n)} \cdot x_{(l-1,m)} \tag{38}$$

Die beiden Gleichungen $\frac{\partial x_{(l,n)}}{\partial z_{(l,n)}} = \frac{\partial f_{(l)}(z_{(l,n)})}{\partial z_{(l,n)}} = f'_{(l)}(z_{(l,n)})$ und $\frac{\partial z_{(l,n)}}{\partial w_{(l-1,m),(l,n)}} = x_{(l-1,m)}$ (da $w_{(l-1,m),(l,n)}$ jener Faktor ist, mit dem $x_{(l-1,m)}$ zur Berechnung von $z_{(l,n)}$ multipliziert wird, ist $x_{(l-1,m)}$ die partielle Ableitung von $z_{(l,n)}$ in Abhängigkeit von $w_{(l-1,m),(l,n)}$) lassen sich nun in die vorherige Gleichung einsetzen:

$$\frac{\partial C}{\partial x_{(l,n)}} \cdot f'_{(l)}(z_{(l,n)}) \cdot x_{(l-1,m)} = \delta_{(l,n)} \cdot x_{(l-1,m)} \tag{39}$$

Da nun der Term $\frac{\partial C}{\partial x_{(l,n)}} \cdot f'_{(l)}(z_{(l,n)})$ die Definition von $\delta_{(l,n)}$ ist, ist die Gleichung und somit auch die ursprüngliche Formel richtig.

Anhang C: In Experimenten verwendete Modelle

C.1 Modelle für Experimente zu FFNNs

Modell-Nr	Aufbau	Bias	Weights und Biases
1	1. Input Layer (784 Neurone) 2. Hidden Layer (10 Neurone) 3. Output Layer (10 Neurone)	False	7.950
		True	7.960
2	1. Input Layer (784 Neurone) 2. Hidden Layer (100 Neurone) 3. Output Layer (10 Neurone)	False	79.410
		True	79.510
3	1. Input Layer (784 Neurone) 2. Hidden Layer (1000 Neurone) 3. Output Layer (10 Neurone)	False	794.010
		True	795.010
4	1. Input Layer (784 Neurone) 2. Hidden Layer (10 Neurone) 3. Hidden Layer (10 Neurone) 4. Output Layer (10 Neurone)	False	8.050
		True	8.070
5	1. Input Layer (784 Neurone) 2. Hidden Layer (100 Neurone) 3. Hidden Layer (10 Neurone) 4. Output Layer (10 Neurone)	False	79.510
		True	79.620
6	1. Input Layer (784 Neurone) 2. Hidden Layer (10 Neurone) 3. Hidden Layer (100 Neurone) 4. Output Layer (10 Neurone)	False	9.850
		True	9.960
7	1. Input Layer (784 Neurone) 2. Hidden Layer (100 Neurone) 3. Hidden Layer (100 Neurone) 4. Output Layer (10 Neurone)	False	89.410
		True	89.610

Tabelle 2: Die verschiedenen, bei den Experimenten verwendeten Modelle von FFNNs und die Anzahl ihrer Weights und Biases (Quelle: Eigene Tabelle)

C.2 Modelle für Experimente zu CNNs

Modell-Nr	Aufbau	Kernel	Stride	Weights und Biases
1	1. Input Layer (784 Neurone) 2. Convolutional Layer (10 FMs) 3. Pooling Layer (10 FMs) 4. Fully-Connected Layer (10 Neurone)	2,2	1,1	78.460
			1,2	19.660
			2,1	19.660
			2,2	4.960
		2,3	1,1	78.480
			1,2	19.680
			2,1	19.680
			2,2	4.980
		3,2	1,1	78.480
			1,2	19.680
			2,1	19.680
			2,2	4.980
		3,3	1,1	78.510
			1,2	19.710
			2,1	19.710
			2,2	5.010
2	1. Input Layer (784 Neurone) 2. Convolutional Layer (10 FMs) 3. Convolutional Layer (10 FMs) 4. Pooling Layer (10 FMs) 5. Fully-Connected Layer (10 Neurone)	2,2	1,1	78.870
			1,2	11.670
			2,1	11.670
			2,2	2.070
		2,3	1,1	79.090
			1,2	11.890
			2,1	11.890
			2,2	2.290
		3,2	1,1	79.090
			1,2	11.890
			2,1	11.890
			2,2	2.290
		3,3	1,1	79.420
			1,2	12.220
			2,1	12.220
			2,2	2.620

3	1. Input Layer (784 Neurone) 2. Convolutional Layer (10 FMs) 3. Pooling Layer (10 FMs) 4. Convolutional Layer (10 FMs) 5. Pooling Layer (10 FMs) 6. Fully-Connected Layer (10 Neurone)	2,2	1,1	78.870
			1,2	6.070
			2,1	6.070
			2,2	870
		2,3	1,1	79.090
			1,2	6.290
			2,1	6.290
			2,2	1.090
		3,2	1,1	79.090
			1,2	6.290
			2,1	6.290
			2,2	1.090
		3,3	1,1	79.420
			1,2	6.620
			2,1	6.620
			2,2	1.420
4	1. Input Layer (784 Neurone) 2. Convolutional Layer (10 FMs) 3. Convolutional Layer (10 FMs) 4. Pooling Layer (10 FMs) 5. Convolutional Layer (10 FMs) 6. Convolutional Layer (10 FMs) 7. Pooling Layer (10 FMs) 8. Fully-Connected Layer (10 Neurone)	2,2	1,1	79.690
			1,2	4.090
			2,1	4.090
			2,2	1.390
		2,3	1,1	80.310
			1,2	4.710
			2,1	4.710
			2,2	2.010
		3,2	1,1	80.310
			1,2	4.710
			2,1	4.710
			2,2	2.010
		3,3	1,1	81.240
			1,2	5.640
			2,1	5.640
			2,2	2.940

Tabelle 3: Die verschiedenen, bei den Experimenten verwendeten Modelle von CNNs und die Anzahl ihrer Weights und Biases (Quelle: Eigene Tabelle)

C.3 Modelle für Experimente zu LSTMs

Modell-Nr	Aufbau	Weights und Biases
1	1. Input Layer (784 Neurone) 2. LSTM Layer ($\text{len}(\vec{x}_{(l)}^{(t)}) = 10$)	1.670
2	1. Input Layer (784 Neurone) 2. LSTM Layer ($\text{len}(\vec{x}_{(l)}^{(t)}) = 10$) 3. LSTM Layer ($\text{len}(\vec{x}_{(l)}^{(t)}) = 10$)	2.510
3	1. Input Layer (784 Neurone) 2. Fully-Connected Layer (10 Neurone) 3. LSTM Layer ($\text{len}(\vec{x}_{(l)}^{(t)}) = 10$)	1.240
4	1. Input Layer (784 Neurone) 2. Fully-Connected Layer (10 Neurone) 3. LSTM Layer ($\text{len}(\vec{x}_{(l)}^{(t)}) = 10$) 4. LSTM Layer ($\text{len}(\vec{x}_{(l)}^{(t)}) = 10$)	2.080

Tabelle 4: Die verschiedenen, bei den Experimenten verwendeten Modelle von LSTMs und die Anzahl ihrer Weights und Biases (Quelle: Eigene Tabelle)

Anhang D: Programmcode

D.1 Programmcode für FFNNs

```
# Programmcode für FFNNs
# VWA: "Künstliche Neuronale Netzwerke und ihr Verhalten beim MNIST Datensatz"
# Autor: Tobias Prisching / 8C / 2018, 2019
# Betreuer: Mag. Christoph Hoedl
# Vorlage: https://elitedatascience.com/keras-tutorial-deep-learning-in-python
#         Zuletzt aufgerufen am 2018-09-09
# Nachschlagwerk: https://keras.io
# Geschrieben für Python 3.6.6, Keras 2.0.8, Tensorflow 1.11.0

# Nötig für Vorbereitung der Daten
import numpy

# Keras Library zum Aufbau der Modelle
import keras

# Nötig für File-Management
import os
import time


# MNIST Daten werden in Trainings- und Testbeispiele eingeteilt
# (Trainingsbeispiele beinhalten Validationsbeispiele)
(xTrainingDaten, yTrainingDaten), (xTestDaten, yTestDaten) = keras.datasets.mnist
            .load_data()

# Anpassung der Dimensionalität der Daten
xTrainingDaten = xTrainingDaten.reshape(xTrainingDaten.shape[0], 28, 28, 1)
xTestDaten = xTestDaten.reshape(xTestDaten.shape[0], 28, 28, 1)

# Umwandlung der Datentypen der Beispiele in Float, um diese anschließend
# durch 255 zu dividieren
xTrainingDaten = xTrainingDaten.astype("float32")
xTestDaten = xTestDaten.astype("float32")

# Dividieren der Daten durch 255 um diese in einen Intervall zwischen 0 und 1
# zu bringen
xTrainingDaten /= 255
xTestDaten /= 255

# Umwandlung der gewünschten Ausgaben in 10-dimensionale Vektoren, eine
# Dimension pro Ziffer
yTrainingDaten = keras.utils.np_utils.to_categorical(yTrainingDaten, 10)
yTestDaten = keras.utils.np_utils.to_categorical(yTestDaten, 10)

```

```
44
45  # Erstellen des Ordners für alle von diesem Programm erstellten Datein
46  ordnerNameZeit = round(time.time())
47  os.makedirs("FFNN"+str(ordnerNameZeit))
48  os.chdir("./FFNN"+str(ordnerNameZeit))
49
50  # Dictonary welches den Index der falsch erkannten Ziffern und von wie vielen
51  # Netzen diese falsch erkannt wurden speichert
52  falschErkannteZiffernVonAllen = {}
53  for i in range(10000):
54     falschErkannteZiffernVonAllen[i]=0
55
56
57  # Kopfzeile für die .csv Datei mit den gesamelten Ergebnissen
58  csvErgebnisseHeader = "Index-Nr;Modell-Nr;Aktf.;Bias;Lernrate;Mini Batch;Epochs;
              Trainingszeit (s);Train-Cost;Val.-Cost;Test-Cost;Train-Quote;Val.-
              Quote;Test-Quote\n"
59
60  # Kopfzeile für die .csv Datei mit jenen Ziffern der Test Menge, welche von allen
61  # Netzen falsch erkannt wurden
62  csvFalschErkanntVonAllenHeader = "Ziffer-Index;Falsch erkannt von\n"
63
64  # Kopfzeile für die .csv Datei mit jenen Ziffern der Test Menge, welche von einem
65  # bestimmten Netz falsch erkannt wurden und was stattdessen seine Antwort hat
66  csvFalschErkanntEinzelnHeader = "Ziffer-Index;Erkannt als;Richtige Antwort:\n"
67
68  # Öffnen dieser .csv Datei und Einfügen des Headers
69  csvMitErgebnissen = open("ergebnisse.csv", "a")
70  csvMitErgebnissen.write(csvErgebnisseHeader)
71
72
73
74
75  class FFNN():
76
77     # Initialisierung eines FFNNs
78     def __init__(self, hiddenLayerAufbau, aktivierungsfunktion, bias, lernrate,
                 miniBatchGroesse, indexNr, hiddenLayerAufbauNr):
79
80        # Speichern der übergebenen Argumente
81        self.aktivierungsfunktion = aktivierungsfunktion
82        self.bias = bias
83        self.lernrate = lernrate
84        self.miniBatchGroesse = miniBatchGroesse
85        self.indexNr = indexNr
86        self.hiddenLayerAufbauNr = hiddenLayerAufbauNr
87
88        # Setzen des Seeds des Zufallsgenerators, um die Ergebnisse der Experimente
89        # reproduzieren zu können
90        # Hierfür wird die Index-Nr. des KNNs verwendet
91        numpy.random.seed(self.indexNr)
92
93        # Es handelt sich um ein sequenzielles Model
94        self.modell = keras.models.Sequential()
95
96        # Dimensionalität der Daten wird angepasst
97        self.modell.add(keras.layers.Flatten(input_shape=(28,28,1)))
```

```
98
99          # Hinzufügen der Hidden Layer zum Modell
100         for i, hiddenLayer in enumerate(hiddenLayerAufbau):
101             self.modell.add(keras.layers.Dense(hiddenLayer, activation=
                    aktivierungsfunktion, use_bias=self.bias))
102
103         # Hinzufügen des Output Layers zum Modell
104         self.modell.add(keras.layers.Dense(10, activation="softmax"))
105
106         # Initialisierung der Optimierungs-Methode
107         stochasticGradientDescent = keras.optimizers.SGD(lr=self.lernrate, decay=0,
                momentum=0, nesterov=False)
108
109         # Konfigurieren des Lernprozesses
110         self.modell.compile(loss="mean_squared_error", optimizer=
                stochasticGradientDescent, metrics=["accuracy"])
111
112
113     # Funktion fürs Trainieren und Testen des FFNNs
114     def trainierenUndTesten(self):
115
116         os.makedirs(str(self.indexNr))
117         os.chdir(str(self.indexNr))
118
119         # Implementierung des Early Stoppings
120         earlyStopping = keras.callbacks.EarlyStopping(monitor="val_acc", min_delta
                =0, patience=5, verbose=1, mode="auto")
121
122         # Speichern der besten Parameter
123         modelCheckpoint = keras.callbacks.ModelCheckpoint("parameters-"+str(self.
                indexNr)+".hdf5", monitor="val_acc", verbose=1, save_best_only=
                True, save_weights_only=False, mode="auto", period=1)
124
125         # Speichern der Loss- und Accuracy-Werte nach jedem Epoch
126         csvLogger = keras.callbacks.CSVLogger("trainingLog-"+str(self.indexNr)+".csv
                ", separator=";", append=False)
127
128         # Liste aller nicht standardmäßigen Callbacks
129         callbacksListe = [modelCheckpoint, earlyStopping, csvLogger]
130
131
132         # Trainieren des KNNs und Messung der benötigten Zeit
133         startZeit = time.time()
134         trainingHistory = self.modell.fit(xTrainingDaten, yTrainingDaten, batch_size
                =self.miniBatchGroesse, epochs=3, verbose=1, callbacks=
                callbacksListe, validation_split=1/6)
135
136         # Berechnung der fürs Trainieren benötigten Zeit
137         # in Sekunden
138         trainingszeit = round(time.time() - startZeit, 1)
139
140
141         # Laden der besten Parameter
142         self.modell.load_weights("parameters-"+str(self.indexNr)+".hdf5", by_name=
                False)
143
144
```

```
145
146         # Testen des KNNs auf die verschiedenen Beispiele
147         trainingQuote = self.modell.evaluate(xTrainingDaten[:50000], yTrainingDaten
                    [:50000], verbose=1)
148         validationQuote = self.modell.evaluate(xTrainingDaten[50000:],
                    yTrainingDaten[50000:], verbose=1)
149         testQuote = self.modell.evaluate(xTestDaten, yTestDaten, verbose=1)
150
151         # Speichern der inkorrekt erkannten Ziffern
152         falschErkannteZiffernEinzeln = []
153         testAntwortKNN = self.modell.predict(xTestDaten, verbose=0)
154         for i in range (10000):
155           if numpy.argmax(testAntwortKNN[i]) != numpy.argmax(yTestDaten[i]):
156             falschErkannteZiffernEinzeln.append([i, numpy.argmax(testAntwortKNN[i])
                          , numpy.argmax(yTestDaten[i])])
157             falschErkannteZiffernVonAllen[i]+=1
158         csvFalschErkanntEinzeln = open("csvFalschErkannt-" + str(self.indexNr) + ".
                    csv", "a")
159         csvFalschErkanntEinzeln.write(csvFalschErkanntEinzelnHeader)
160         for falschErkannteZiffer in falschErkannteZiffernEinzeln:
161           csvFalschErkanntEinzeln.write(str(falschErkannteZiffer[0]) + ";" + str(
                        falschErkannteZiffer[1]) + ";" + str(falschErkannteZiffer[2])
                        + "\n")
162         csvFalschErkanntEinzeln.close()
163
164
165         # Speichern des Modells als .json Datei und der Parameter als .h5 (für
166         # spätere Umwandlung in CoreML Dateien)
167         jsonDatei = open("model-"+str(self.indexNr)+".json", "w")
168         jsonDatei.write(self.modell.to_json())
169         jsonDatei.close()
170         self.modell.save_weights("modelWeights-"+str(self.indexNr)+".h5")
171
172         os.chdir("./..")
173
174         # Zurückgeben der Ergebnisse
175         returnString = str(self.indexNr) + ";" + str(self.hiddenLayerAufbauNr) + ";"
                     + self.aktivierungsfunktion + ";" + str(self.bias) + ";" + str(
                    self.lernrate) + ";" + str(self.miniBatchGroesse) + ";" + str(
                    len(trainingHistory.history["acc"])) + ";" + str(trainingszeit)
                    + ";" + str(round(trainingQuote[0],4)) + ";" + str(round(
                    validationQuote[0],4)) + ";" + str(round(testQuote[0],4)) + ";"
                    + str(round(trainingQuote[1],4)) + ";" + str(round(
                    validationQuote[1],4)) + ";" + str(round(testQuote[1],4)) + "\n"
176         return(returnString)
177
178
179
180 # Index zur Identifizierung der verschiedenen FFNNs
181 index = 1
182
183 # Erstellen der Array mit den verschiedenen Modellen:
184 modell1 = [10]
185
186 modell2 = [100]
187
188 modell3 = [1000]
```

```
189
190 modell4 = [10,10]
191
192 modell5 = [100,10]
193
194 modell6 = [10,100]
195
196 modell7 = [100,100]
197
198 modelle = [modell1,modell2,modell3,modell4, modell5, modell6, modell7]
199
200 # Schleifen zum Testen der verschiedenen Kombinationen
201 for hiddenLayerAufbauNr, hiddenLayerAufbau in enumerate(modelle):
202    for lernrate in [1,0.1,0.01]:
203       for miniBatchGroesse in [8,32,128]:
204          for aktivierungsfunktion in ["relu","sigmoid","tanh"]:
205             for bias in [False,True]:
206                # Erstellung eines FFNNs mit dieser Kombination an Hyperparametern
207                KNN = FFNN(hiddenLayerAufbau, aktivierungsfunktion, bias, lernrate,
                              miniBatchGroesse, index, hiddenLayerAufbauNr+1)
208
209                # Testen des FFNNs und Niederschreiben der Ergebnisse
210                csvMitErgebnissen.write(KNN.trainierenUndTesten())
211                csvMitErgebnissen.close()
212                csvMitErgebnissen = open("ergebnisse.csv", "a")
213
214                # Inkrementierung des Index
215                index+=1
216
217 # Öffnen, speichern und schließen der .csv Datei, welche speichert, welche der
218 # Testbeispiele von wie vielen Netzen falsch erkannt wurden
219 csvFalschErkanntVonAllen = open("falschErkannteZiffernVonAllen.csv", "a")
220 csvFalschErkanntVonAllen.write(csvFalschErkanntVonAllenHeader)
221 for ziffer in falschErkannteZiffernVonAllen:
222    csvFalschErkanntVonAllen.write(str(ziffer)+";"+str(
                falschErkannteZiffernVonAllen[ziffer])+"\n")
223 csvFalschErkanntVonAllen.close()
224
225 # Schließen der Datei mit den Ergebnissen, verlassen des Ordners der FFNNs
226 csvMitErgebnissen.close()
227 os.chdir("./..")
```

D.2 Programmcode für CNNs

```
# Programmcode für CNNs
# VWA: "Künstliche Neuronale Netzwerke und ihr Verhalten beim MNIST Datensatz"
# Autor: Tobias Prisching / 8C / 2018, 2019
# Betreuer: Mag. Christoph Hoedl
# Vorlage: https://elitedatascience.com/keras-tutorial-deep-learning-in-python
#         Zuletzt aufgerufen am 2018-09-09
# Nachschlagwerk: https://keras.io
# Geschrieben für Python 3.6.6, Keras 2.0.8, Tensorflow 1.11.0

# Nötig für Vorbereitung der Daten
import numpy

# Keras Library zum Aufbau der Modelle
import keras

# Nötig für File-Management
import os
import time


# MNIST Daten werden in Trainings- und Testbeispiele eingeteilt
# (Trainingsbeispiele beinhalten Validationsbeispiele)
(xTrainingDaten, yTrainingDaten), (xTestDaten, yTestDaten) = keras.datasets.mnist
            .load_data()

# Anpassung der Dimensionalität der Daten
xTrainingDaten = xTrainingDaten.reshape(xTrainingDaten.shape[0], 28, 28,1)
xTestDaten = xTestDaten.reshape(xTestDaten.shape[0], 28, 28,1)

# Umwandlung der Datentypen der Beispiele in Float, um diese anschließend
# durch 255 zu dividieren
xTrainingDaten = xTrainingDaten.astype("float32")
xTestDaten = xTestDaten.astype("float32")

# Dividieren der Daten durch 255 um diese in einen Intervall zwischen 0 und 1
# zu bringen
xTrainingDaten /= 255
xTestDaten /= 255

# Umwandlung der gewünschten Ausgaben in 10-dimensionale Vektoren, eine
# Dimension pro Ziffer
yTrainingDaten = keras.utils.np_utils.to_categorical(yTrainingDaten, 10)
yTestDaten = keras.utils.np_utils.to_categorical(yTestDaten, 10)


# Erstellen des Ordners für alle von diesem Programm erstellten Datein
ordnerNameZeit = round(time.time())
os.makedirs("CNN"+str(ordnerNameZeit))
os.chdir("./CNN"+str(ordnerNameZeit))

# Dictonary welches den Index der falsch erkannten Ziffern und von wie vielen
# Netzen diese falsch erkannt wurden speichert
falschErkannteZiffernVonAllen = {}
for i in range(10000):
   falschErkannteZiffernVonAllen[i]=0
```

```
55
56
57  # Kopfzeile für die .csv Datei mit den gesamelten Ergebnissen
58  csvErgebnisseHeader = "Index-Nr;Modell-Nr;Kernel;Stride;Aktf.;Pooling;Bias;
                Lernrate;Mini Batch;Epochs;Trainingszeit (s);Train-Cost;Val.-Cost;
                Test-Cost;Train-Quote;Val.-Quote;Test-Quote\n"
59
60  # Kopfzeile für die .csv Datei mit jenen Ziffern der Test Menge, welche von allen
61  # Netzen falsch erkannt wurden
62  csvFalschErkanntVonAllenHeader = "Ziffer-Index;Falsch erkannt von\n"
63
64  # Kopfzeile für die .csv Datei mit jenen Ziffern der Test Menge, welche von einem
65  # bestimmten Netz falsch erkannt wurden und was stattdessen seine Antwort hat
66  csvFalschErkanntEinzelnHeader = "Ziffer-Index;Erkannt als;Richtige Antwort:\n"
67
68  # Öffnen dieser .csv Datei und Einfügen des Headers
69  csvMitErgebnissen = open("ergebnisse.csv", "a")
70  csvMitErgebnissen.write(csvErgebnisseHeader)
71
72
73
74
75  class CNN():
76
77      # Initialisierung eines CNNs
78      def __init__(self, hiddenLayerAufbau, kernelDimensionen, strideDimensionen,
                    aktivierungsfunktion, pooling, bias, lernrate, miniBatchGroesse,
                    indexNr, hiddenLayerAufbauNr):
79
80          # Speichern der übergebenen Argumente
81          self.kernelDimensionen = kernelDimensionen
82          self.strideDimensionen = strideDimensionen
83          self.aktivierungsfunktion = aktivierungsfunktion
84          self.pooling = pooling
85          self.bias = bias
86          self.lernrate = lernrate
87          self.miniBatchGroesse = miniBatchGroesse
88          self.indexNr = indexNr
89          self.hiddenLayerAufbauNr = hiddenLayerAufbauNr
90
91          # Setzen des Seeds des Zufallsgenerators, um die Ergebnisse der Experimente
92          # reproduzieren zu können
93          # Hierfür wird die Index-Nr. des KNNs verwendet
94          numpy.random.seed(self.indexNr)
95
96          # Es handelt sich um ein sequenzielles Model
97          self.modell = keras.models.Sequential()
98
99          # Hinzufügen der Hidden Layer zum Modell
100         for i, hiddenLayer in enumerate(hiddenLayerAufbau):
101             if hiddenLayer[0][0] == "Conv":
102                 if i == 0:
103                     self.modell.add(keras.layers.Conv2D(hiddenLayer[1][0], (
                                kernelDimensionen[0], kernelDimensionen[1]), strides=(
                                strideDimensionen[0], strideDimensionen[1]), activation=
                                aktivierungsfunktion, padding=hiddenLayer[2][0],
                                input_shape=(28,28,1), use_bias=self.bias))
```

```
104             else:
105                 self.modell.add(keras.layers.Conv2D(hiddenLayer[1][0], (
                                kernelDimensionen[0], kernelDimensionen[1]), strides=(
                                strideDimensionen[0], strideDimensionen[1]), activation=
                                aktivierungsfunktion, padding=hiddenLayer[2][0], use_bias
                                =self.bias))
106
107         elif hiddenLayer[0][0] == "Pool":
108             if pooling == "Max":
109                 self.modell.add(keras.layers.MaxPooling2D(pool_size=(
                                kernelDimensionen[0], kernelDimensionen[1]), strides=(
                                strideDimensionen[0], strideDimensionen[1]), padding=
                                hiddenLayer[2][0]))
110         elif pooling == "Av.":
111             self.modell.add(keras.layers.AveragePooling2D(pool_size=(
                            kernelDimensionen[0], kernelDimensionen[1]), strides=(
                            strideDimensionen[0], strideDimensionen[1]), padding=
                            hiddenLayer[2][0]))
112
113         elif hiddenLayer[0][0] == "Flatten":
114             self.modell.add(keras.layers.Flatten())
115
116         else:
117             self.modell.add(keras.layers.Dense(hiddenLayer, activation=
                            aktivierungsfunktion, use_bias=self.bias))
118
119     # Hinzufügen des Output Layers zum Modell
120     self.modell.add(keras.layers.Dense(10, activation="softmax"))
121
122     # Initialisierung der Optimierungs-Methode
123     stochasticGradientDescent = keras.optimizers.SGD(lr=self.lernrate, decay=0,
                momentum=0, nesterov=False)
124
125     # Konfigurieren des Lernprozesses
126     self.modell.compile(loss="mean_squared_error", optimizer=
                stochasticGradientDescent, metrics=["accuracy"])
127
128
129 # Funktion fürs Trainieren und Testen des CNNs
130 def trainierenUndTesten(self):
131
132     os.makedirs(str(self.indexNr))
133     os.chdir(str(self.indexNr))
134
135     # Implementierung des Early Stoppings
136     earlyStopping = keras.callbacks.EarlyStopping(monitor="val_acc", min_delta
                =0, patience=5, verbose=1, mode="auto")
137
138     # Speichern der besten Parameter
139     modelCheckpoint = keras.callbacks.ModelCheckpoint("parameters-"+str(self.
                indexNr)+".hdf5", monitor="val_acc", verbose=1, save_best_only=
                True, save_weights_only=False, mode="auto", period=1)
140
141     # Speichern der Loss- und Accuracy-Werte nach jedem Epoch
142     csvLogger = keras.callbacks.CSVLogger("trainingLog-"+str(self.indexNr)+".csv
                ", separator=";", append=False)
143
```

```
144         # Liste aller nicht standardmäßigen Callbacks
145         callbacksListe = [modelCheckpoint, earlyStopping, csvLogger]
146 
147 
148         # Trainieren des KNNs und Messung der dafür benötigten Zeit
149         startZeit = time.time()
150         trainingHistory = self.modell.fit(xTrainingDaten, yTrainingDaten, batch_size
                    =self.miniBatchGroesse, epochs=3, verbose=1, callbacks=
                    callbacksListe, validation_split=1/6)
151 
152         # Berechnung der fürs Trainieren benötigten Zeit
153         # in Sekunden
154         trainingszeit = round(time.time() - startZeit, 1)
155 
156 
157         # Laden der besten Parameter
158         self.modell.load_weights("parameters-"+str(self.indexNr)+".hdf5", by_name=
                    False)
159 
160         # Testen des KNNs auf die verschiedenen Beispiele
161         trainingQuote = self.modell.evaluate(xTrainingDaten[:50000], yTrainingDaten
                    [:50000], verbose=1)
162         validationQuote = self.modell.evaluate(xTrainingDaten[50000:],
                    yTrainingDaten[50000:], verbose=1)
163         testQuote = self.modell.evaluate(xTestDaten, yTestDaten, verbose=1)
164 
165         # Speichern der inkorrekt erkannten Ziffern
166         falschErkannteZiffernEinzeln = []
167         testAntwortKNN = self.modell.predict(xTestDaten, verbose=0)
168         for i in range (10000):
169           if numpy.argmax(testAntwortKNN[i]) != numpy.argmax(yTestDaten[i]):
170             falschErkannteZiffernEinzeln.append([i, numpy.argmax(testAntwortKNN[i])
                          , numpy.argmax(yTestDaten[i])])
171             falschErkannteZiffernVonAllen[i]+=1
172         csvFalschErkanntEinzeln = open("csvFalschErkannt-" + str(self.indexNr) + ".
                    csv", "a")
173         csvFalschErkanntEinzeln.write(csvFalschErkanntEinzelnHeader)
174         for falschErkannteZiffer in falschErkannteZiffernEinzeln:
175           csvFalschErkanntEinzeln.write(str(falschErkannteZiffer[0]) + ";" + str(
                      falschErkannteZiffer[1]) + ";" + str(falschErkannteZiffer[2])
                      + "\n")
176         csvFalschErkanntEinzeln.close()
177 
178 
179         # Speichern des Modells als .json Datei und der Parameter als .h5 (für
180         # spätere Umwandlung in CoreML Dateien)
181         jsonDatei = open("model-"+str(self.indexNr)+".json", "w")
182         jsonDatei.write(self.modell.to_json())
183         jsonDatei.close()
184         self.modell.save_weights("modelWeights-"+str(self.indexNr)+".h5")
185 
186         os.chdir("./..")
187 
188         # Zurückgeben der Ergebnisse
189         returnString = str(self.indexNr) + ";" + str(self.hiddenLayerAufbauNr) + ";"
                     + str(self.kernelDimensionen[0]) + "x" + str(self.
                    kernelDimensionen[1]) + ";" + str(self.strideDimensionen[0]) + "
```

```
                x" + str(self.strideDimensionen[1]) + ";" + self.
                aktivierungsfunktion + ";" + self.pooling + ";" + str(self.bias)
                 + ";" + str(self.lernrate) + ";" + str(self.miniBatchGroesse) +
                 ";" + str(len(trainingHistory.history["acc"])) + ";" + str(
                trainingszeit) + ";" + str(round(trainingQuote[0],4)) + ";" +
                str(round(validationQuote[0],4)) + ";" + str(round(testQuote
                [0],4)) + ";" + str(round(trainingQuote[1],4)) + ";" + str(round
                (validationQuote[1],4)) + ";" + str(round(testQuote[1],4)) + "\n
                "
    return(returnString)



# Index zur Identifizierung der verschiedenen CNNs
index = 1

# Erstellen der Array mit den verschiedenen Modellen:
modell1 = [["Conv"],[10],["same"]],
[["Pool"],[10],["same"]],
[["Flatten"]]]

modell2 = [["Conv"],[10],["same"]],
[["Conv"],[10],["same"]],
[["Pool"],[10],["same"]],
[["Flatten"]]]

modell3 = [["Conv"],[10],["same"]],
[["Pool"],[10],["same"]],
[["Conv"],[10],["same"]],
[["Pool"],[10],["same"]],
[["Flatten"]]]

modell4 = [["Conv"],[10],["same"]],
[["Conv"],[10],["same"]],
[["Pool"],[10],["same"]],
[["Conv"],[10],["same"]],
[["Conv"],[10],["same"]],
[["Pool"],[10],["same"]],
[["Flatten"]]]

modelle = [modell1,modell2,modell3,modell4]

# Schleifen zum Testen der verschiedenen Kombinationen
for hiddenLayerAufbauNr, hiddenLayerAufbau in enumerate(modelle):
   for lernrate in [1,0.1,0.01]:
      for miniBatchGroesse in [8,32,128]:
         for aktivierungsfunktion in ["relu"]:
            for bias in [True]:
               for kernel in [[2,2],[2,3],[3,2],[3,3]]:
                  for strides in [[1,1],[1,2],[2,1],[2,2]]:
                     for pooling in ["Max"]:
                        # Erstellung eines CNNs mit dieser Kombination an
                        # Hyperparametern
                        KNN = CNN(hiddenLayerAufbau, kernel,strides,
                                    aktivierungsfunktion, pooling, bias, lernrate,
                                    miniBatchGroesse, index, hiddenLayerAufbauNr+1)

```

```
236                     # Testen des CNNs und Niederschreiben der Ergebnisse
237                     csvMitErgebnissen.write(KNN.trainierenUndTesten())
238                     csvMitErgebnissen.close()
239                     csvMitErgebnissen = open("ergebnisse.csv", "a")
240
241                     # Inkrementierung des Index
242                     index+=1
243
244 # Öffnen, speichern und schließen der .csv Datei, welche speichert, welche der
245 # Testbeispiele von wie vielen Netzen falsch erkannt wurden
246 csvFalschErkanntVonAllen = open("falschErkannteZiffernVonAllen.csv", "a")
247 csvFalschErkanntVonAllen.write(csvFalschErkanntVonAllenHeader)
248 for ziffer in falschErkannteZiffernVonAllen:
249     csvFalschErkanntVonAllen.write(str(ziffer)+";"+str(
              falschErkannteZiffernVonAllen[ziffer])+"\n")
250 csvFalschErkanntVonAllen.close()
251
252 # Schließen der Datei mit den Ergebnissen, verlassen des Ordners der CNNs
253 csvMitErgebnissen.close()
254 os.chdir("./..")
```

D.3 Programmcode für LSTMs

```
# Programmcode für LSTMs
# VWA: "Künstliche Neuronale Netzwerke und ihr Verhalten beim MNIST Datensatz"
# Autor: Tobias Prisching / 8C / 2018, 2019
# Betreuer: Mag. Christoph Hoedl
# Vorlage: https://elitedatascience.com/keras-tutorial-deep-learning-in-python
#         Zuletzt aufgerufen am 2018-09-09
# Nachschlagwerk: https://keras.io
# Geschrieben für Python 3.6.6, Keras 2.0.8, Tensorflow 1.11.0

# Nötig für Vorbereitung der Daten
import numpy

# Keras Library zum Aufbau der Modelle
import keras

# Nötig für File-Management
import os
import time


# MNIST Daten werden in Trainings- und Testbeispiele eingeteilt
# (Trainingsbeispiele beinhalten Validationsbeispiele)
(xTrainingDaten, yTrainingDaten), (xTestDaten, yTestDaten) = keras.datasets.mnist
            .load_data()

# Anpassung der Dimensionalität der Daten
xTrainingDaten = xTrainingDaten.reshape(xTrainingDaten.shape[0], 28, 28)
xTestDaten = xTestDaten.reshape(xTestDaten.shape[0], 28, 28)

# Umwandlung der Datentypen der Beispiele in Float, um diese anschließend
# durch 255 zu dividieren
xTrainingDaten = xTrainingDaten.astype("float32")
xTestDaten = xTestDaten.astype("float32")

# Dividieren der Daten durch 255 um diese in einen Intervall zwischen 0 und 1
# zu bringen
xTrainingDaten /= 255
xTestDaten /= 255

# Umwandlung der gewünschten Ausgaben in 10-dimensionale Vektoren, eine
# Dimension pro Ziffer
yTrainingDaten = keras.utils.np_utils.to_categorical(yTrainingDaten, 10)
yTestDaten = keras.utils.np_utils.to_categorical(yTestDaten, 10)


# Erstellen des Ordners für alle von diesem Programm erstellten Datein
ordnerNameZeit = round(time.time())
os.makedirs("LSTM"+str(ordnerNameZeit))
os.chdir("./LSTM"+str(ordnerNameZeit))

# Dictonary welches den Index der falsch erkannten Ziffern und von wie vielen
# Netzen diese falsch erkannt wurden speichert
falschErkannteZiffernVonAllen = {}
for i in range(10000):
   falschErkannteZiffernVonAllen[i]=0
```

```
55
56
57  # Kopfzeile für die .csv Datei mit den gesamelten Ergebnissen
58  csvErgebnisseHeader = "Index-Nr;Modell-Nr;Aktf.;rek. Aktf.;Bias;Lernrate;Mini
                Batch;Epochs;Trainingszeit (s);Train-Cost;Val.-Cost;Test-Cost;Train-
                Quote;Val.-Quote;Test-Quote\n"
59
60  # Kopfzeile für die .csv Datei mit jenen Ziffern der Test Menge, welche von allen
61  # Netzen falsch erkannt wurden
62  csvFalschErkanntVonAllenHeader = "Ziffer-Index;Falsch erkannt von\n"
63
64  # Kopfzeile für die .csv Datei mit jenen Ziffern der Test Menge, welche von einem
65  # bestimmten Netz falsch erkannt wurden und was stattdessen seine Antwort hat
66  csvFalschErkanntEinzelnHeader = "Ziffer-Index;Erkannt als;Richtige Antwort:\n"
67
68  # Öffnen dieser .csv Datei und Einfügen des Headers
69  csvMitErgebnissen = open("ergebnisse.csv", "a")
70  csvMitErgebnissen.write(csvErgebnisseHeader)
71
72
73  class LSTM():
74
75      # Initialisierung eines LSTMs
76      def __init__(self, hiddenLayerAufbau, aktivierungsfunktion,
                    rekurrenteAktivierungsfunktion, bias, lernrate, miniBatchGroesse,
                    indexNr, hiddenLayerAufbauNr):
77
78          # Speichern der übergebenen Argumente
79          self.hiddenLayer = hiddenLayerAufbau
80          self.aktivierungsfunktion = aktivierungsfunktion
81          self.rekurrenteAktivierungsfunktion = rekurrenteAktivierungsfunktion
82          self.bias = bias
83          self.lernrate = lernrate
84          self.miniBatchGroesse = miniBatchGroesse
85          self.indexNr = indexNr
86          self.hiddenLayerAufbauNr = hiddenLayerAufbauNr
87
88          # Setzen des Seeds des Zufallsgenerators, um die Ergebnisse der Experimente
89          # reproduzieren zu können
90          # Hierfür wird die Index-Nr. des KNNs verwendet
91          numpy.random.seed(self.indexNr)
92
93          # Es handelt sich um ein sequenzielles Model
94          self.modell = keras.models.Sequential()
95
96          # Hinzufügen der Hidden Layer zum Modell
97          for i, hiddenLayer in enumerate(hiddenLayerAufbau):
98              if hiddenLayer[0][0] == "LSTM":
99                  if i == 0:
100                 self.modell.add(keras.layers.LSTM(hiddenLayer[1][0], activation=
                            aktivierungsfunktion, recurrent_activation=
                            rekurrenteAktivierungsfunktion, return_sequences=
                            hiddenLayer[2][0], input_shape=(28,28), implementation=2,
                            unroll=True, use_bias=self.bias))
101             else:
102                 self.modell.add(keras.layers.LSTM(hiddenLayer[1][0], activation=
                            aktivierungsfunktion, recurrent_activation=
```

```
                    rekurrenteAktivierungsfunktion, return_sequences=
                    hiddenLayer[2][0], implementation=2, unroll=True, use_bias=
                    self.bias))
103
104         elif hiddenLayer[0][0] == "Flatten":
105             self.modell.add(keras.layers.Flatten())
106
107         else:
108             if i == 0:
109                 self.modell.add(keras.layers.Dense(hiddenLayer[1][0], activation=
                            aktivierungsfunktion, input_shape=(28,28), use_bias=self.
                            bias))
110         else:
111             self.modell.add(keras.layers.Dense(hiddenLayer[1][0], activation=
                        aktivierungsfunktion, use_bias=self.bias))
112
113     # Hinzufügen des Output Layers zum Modell
114     self.modell.add(keras.layers.Dense(10, activation="softmax"))
115
116     # Initialisierung der Optimierungs-Methode
117     stochasticGradientDescent = keras.optimizers.SGD(lr=self.lernrate, decay=0,
                momentum=0, nesterov=False)
118
119     # Konfigurieren des Lernprozesses
120     self.modell.compile(loss="mean_squared_error", optimizer=
                stochasticGradientDescent, metrics=["accuracy"])
121
122
123 # Funktion fürs Trainieren und Testen des LSTMs
124 def trainierenUndTesten(self):
125
126     os.makedirs(str(self.indexNr))
127     os.chdir(str(self.indexNr))
128
129     # Implementierung des Early Stoppings
130     earlyStopping = keras.callbacks.EarlyStopping(monitor="val_acc", min_delta
                =0, patience=5, verbose=1, mode="auto")
131
132     # Speichern der besten Parameter
133     modelCheckpoint = keras.callbacks.ModelCheckpoint("parameters-"+str(self.
                indexNr)+".hdf5", monitor="val_acc", verbose=1, save_best_only=
                True, save_weights_only=False, mode="auto", period=1)
134
135     # Speichern der Loss- und Accuracy-Werte nach jedem Epoch
136     csvLogger = keras.callbacks.CSVLogger("trainingLog-"+str(self.indexNr)+".csv
                ", separator=";", append=False)
137
138     # Liste aller nicht standardmäßigen Callbacks
139     callbacksListe = [modelCheckpoint, earlyStopping, csvLogger]
140
141
142     # Trainieren des KNNs und Messung der dafür benötigten Zeit
143     startZeit = time.time()
144     trainingHistory = self.modell.fit(xTrainingDaten, yTrainingDaten, batch_size
                =self.miniBatchGroesse, epochs=3, verbose=1, callbacks=
                callbacksListe, validation_split=1/6)
145
```

```
146         # Berechnung der fürs Trainieren benötigten Zeit
147         # in Sekunden
148         trainingszeit = round(time.time() - startZeit, 1)
149
150
151         # Laden der besten Parameter
152         self.modell.load_weights("parameters-"+str(self.indexNr)+".hdf5", by_name=
                    False)
153
154         # Testen des KNNs auf die verschiedenen Beispiele
155         trainingQuote = self.modell.evaluate(xTrainingDaten[:50000], yTrainingDaten
                    [:50000], verbose=1)
156         validationQuote = self.modell.evaluate(xTrainingDaten[50000:],
                    yTrainingDaten[50000:], verbose=1)
157         testQuote = self.modell.evaluate(xTestDaten, yTestDaten, verbose=1)
158
159         # Speichern der inkorrekt erkannten Ziffern
160         falschErkannteZiffernEinzeln = []
161         testAntwortKNN = self.modell.predict(xTestDaten, verbose=0)
162         for i in range (10000):
163           if numpy.argmax(testAntwortKNN[i]) != numpy.argmax(yTestDaten[i]):
164             falschErkannteZiffernEinzeln.append([i, numpy.argmax(testAntwortKNN[i])
                          , numpy.argmax(yTestDaten[i])])
165             falschErkannteZiffernVonAllen[i]+=1
166         csvFalschErkanntEinzeln = open("csvFalschErkannt-" + str(self.indexNr) + ".
                    csv", "a")
167         csvFalschErkanntEinzeln.write(csvFalschErkanntEinzelnHeader)
168         for falschErkannteZiffer in falschErkannteZiffernEinzeln:
169           csvFalschErkanntEinzeln.write(str(falschErkannteZiffer[0]) + ";" + str(
                      falschErkannteZiffer[1]) + ";" + str(falschErkannteZiffer[2])
                      + "\n")
170         csvFalschErkanntEinzeln.close()
171
172
173         # Speichern des Modells als .json Datei und der Parameter als .h5 (für spä
                    tere Umwandlung in CoreML Dateien)
174         jsonDatei = open("model-"+str(self.indexNr)+".json", "w")
175         jsonDatei.write(self.modell.to_json())
176         jsonDatei.close()
177         self.modell.save_weights("modelWeights-"+str(self.indexNr)+".h5")
178
179         os.chdir("./..")
180
181         # Zurückgeben der Ergebnisse
182         returnString = str(self.indexNr) + ";" + str(self.hiddenLayerAufbauNr) + ";"
                     + self.aktivierungsfunktion + ";" + self.
                    rekurrenteAktivierungsfunktion + ";" + str(self.bias) + ";" +
                    str(self.lernrate) + ";" + str(self.miniBatchGroesse) + ";" +
                    str(len(trainingHistory.history["acc"])) + ";" + str(
                    trainingszeit) + ";" + str(round(trainingQuote[0],4)) + ";" +
                    str(round(validationQuote[0],4)) + ";" + str(round(testQuote
                    [0],4)) + ";" + str(round(trainingQuote[1],4)) + ";" + str(round
                    (validationQuote[1],4)) + ";" + str(round(testQuote[1],4)) + "\n
                    "
183         return(returnString)
184
185
```

```
186
187 # Index zur Identifizierung der verschiedenen LSTMs
188 index = 1
189
190 # Erstellen der Array mit den verschiedenen Modellen:
191 modell1 = [["LSTM"],[10],[False]]
192
193 modell2 = [["LSTM"],[10],[True]],
194 [["LSTM"],[10],[False]]
195
196 modell3 = [["Dense"],[10]],
197 [["LSTM"],[10],[False]]
198
199 modell4 = [["Dense"],[10]],
200 [["LSTM"],[10],[True]],
201 [["LSTM"],[10],[False]]
202
203 modelle = [modell1,modell2,modell3,modell4]
204
205 # Schleifen zum Testen der verschiedenen Kombinationen
206 for hiddenLayerAufbauNr, hiddenLayerAufbau in enumerate(modelle):
207    for lernrate in [1,0.1,0.01]:
208       for miniBatchGroesse in [8,32,128]:
209          for aktivierungsfunktion in ["sigmoid","tanh"]:
210             for bias in [True]:
211                for rekurrenteAktivierungsfunktion in ["sigmoid","tanh"]:
212                   # Erstellung eines LSTMs mit dieser Kombination an
213                   # Hyperparametern
214                   KNN = LSTM(hiddenLayerAufbau, aktivierungsfunktion,
                                 rekurrenteAktivierungsfunktion, bias, lernrate,
                                 miniBatchGroesse, index, hiddenLayerAufbauNr+1)
215
216                   # Testen des LSTMs und Niederschreiben der Ergebnisse
217                   csvMitErgebnissen.write(KNN.trainierenUndTesten())
218                   csvMitErgebnissen.close()
219                   csvMitErgebnissen = open("ergebnisse.csv", "a")
220
221                   # Inkrementierung des Index
222                   index+=1
223
224 # Öffnen, speichern und schließen der .csv Datei, welche speichert, welche der
225 # Testbeispiele von wie vielen Netzen falsch erkannt wurden
226 csvFalschErkanntVonAllen = open("falschErkannteZiffernVonAllen.csv", "a")
227 csvFalschErkanntVonAllen.write(csvFalschErkanntVonAllenHeader)
228 for ziffer in falschErkannteZiffernVonAllen:
229    csvFalschErkanntVonAllen.write(str(ziffer)+";"+str(
                falschErkannteZiffernVonAllen[ziffer])+"\n")
230 csvFalschErkanntVonAllen.close()
231
232 # Schließen der Datei mit den Ergebnissen, verlassen des Ordners der LSTMs
233 csvMitErgebnissen.close()
234 os.chdir("./..")
```

Anhang E: Ergebnisse der Experimente

E.1 Tabelle der Ergebnisse der Experimente mit FFNNs

Die Spalte mit der Anzahl der Epochs (Wert: 3) wurde weggelassen, da ihr Wert für alle Konfigurationen konstant ist.

Index-Nr	Bauart-Nr	Aktf.	Bias	Lernrate	Mini Batch	Trainingszeit (s)	Train-Cost	Val.-Cost	Test-Cost	Train-Quote	Val.-Quote
1	1	relu	False	1	8	12,8	0,0131	0,0125	0,0139	0,9172	0,9205
2	1	relu	True	1	8	13,1	0,0125	0,0121	0,0127	0,9198	0,9217
3	1	σ	False	1	8	12,9	0,0119	0,0112	0,0118	0,9241	0,9273
4	1	σ	True	1	8	13,4	0,0131	0,0125	0,0136	0,9153	0,9184
5	1	tanh	False	1	8	13,1	0,0121	0,0116	0,0124	0,922	0,9224
6	1	tanh	True	1	8	13,4	0,0119	0,0116	0,0122	0,9221	0,9255
7	1	relu	False	1	32	3,8	0,012	0,0117	0,0121	0,9241	0,9226
8	1	relu	True	1	32	4,0	0,0125	0,0117	0,0124	0,9199	0,9236
9	1	σ	False	1	32	3,9	0,0183	0,0166	0,0175	0,8937	0,9068
10	1	σ	True	1	32	4,0	0,0179	0,0165	0,017	0,8943	0,9026
11	1	tanh	False	1	32	4,0	0,013	0,0123	0,0132	0,9159	0,9204
12	1	tanh	True	1	32	4,1	0,0124	0,0118	0,0124	0,9212	0,9224
13	1	relu	False	1	128	1,6	0,015	0,0137	0,0142	0,9041	0,9111
14	1	relu	True	1	128	1,6	0,0148	0,0137	0,014	0,9072	0,9117
15	1	σ	False	1	128	1,7	0,052	0,0509	0,0515	0,684	0,6988
16	1	σ	True	1	128	1,7	0,0491	0,0481	0,0486	0,6705	0,6853
17	1	tanh	False	1	128	1,7	0,018	0,0163	0,0174	0,891	0,9046
18	1	tanh	True	1	128	1,8	0,0177	0,0161	0,0168	0,8936	0,9031
19	1	relu	False	0,1	8	14,0	0,0134	0,0124	0,0129	0,9154	0,9188
20	1	relu	True	0,1	8	14,3	0,0146	0,0134	0,0141	0,9054	0,9121
21	1	σ	False	0,1	8	14,0	0,034	0,0322	0,0329	0,8262	0,8392
22	1	σ	True	0,1	8	14,4	0,0335	0,0315	0,0327	0,8245	0,8454
23	1	tanh	False	0,1	8	14,1	0,0152	0,0137	0,0144	0,9065	0,9143
24	1	tanh	True	0,1	8	14,5	0,0158	0,0144	0,0153	0,9021	0,9101
25	1	relu	False	0,1	32	4,3	0,0215	0,0197	0,0203	0,866	0,8789
26	1	relu	True	0,1	32	4,5	0,0189	0,017	0,0176	0,8844	0,8965
27	1	σ	False	0,1	32	4,4	0,0759	0,0758	0,0758	0,3981	0,4071
28	1	σ	True	0,1	32	4,5	0,0751	0,0748	0,0747	0,4539	0,4663
29	1	tanh	False	0,1	32	4,5	0,0271	0,0248	0,0261	0,8525	0,87
30	1	tanh	True	0,1	32	4,6	0,0278	0,0257	0,0269	0,8495	0,8679

31	1	relu	False	0,1	128	2,0	0,0598	0,0588	0,0592	0,5996	0,6049
32	1	relu	True	0,1	128	2,1	0,0414	0,0395	0,0402	0,7614	0,7835
33	1	σ	False	0,1	128	2,0	0,0867	0,0867	0,0865	0,3723	0,3758
34	1	σ	True	0,1	128	2,1	0,0868	0,0868	0,0867	0,274	0,279
35	1	tanh	False	0,1	128	2,1	0,0583	0,0574	0,0573	0,6265	0,6424
36	1	tanh	True	0,1	128	2,2	0,0605	0,0592	0,0599	0,6447	0,6713
37	1	relu	False	0,01	8	15,1	0,0358	0,0337	0,0342	0,808	0,8283
38	1	relu	True	0,01	8	15,4	0,0656	0,0643	0,0652	0,5338	0,5458
39	1	σ	False	0,01	8	15,1	0,0837	0,0835	0,0836	0,515	0,5419
40	1	σ	True	0,01	8	15,5	0,0858	0,0856	0,0856	0,4225	0,4373
41	1	tanh	False	0,01	8	15,3	0,0439	0,0424	0,0431	0,7138	0,7315
42	1	tanh	True	0,01	8	15,6	0,0435	0,0413	0,0426	0,7859	0,8108
43	1	relu	False	0,01	32	4,8	0,0801	0,0794	0,0799	0,2967	0,3065
44	1	relu	True	0,01	32	5,0	0,0795	0,0796	0,0788	0,3256	0,321
45	1	σ	False	0,01	32	4,9	0,0892	0,0893	0,0891	0,165	0,1538
46	1	σ	True	0,01	32	5,1	0,0891	0,089	0,089	0,2255	0,2254
47	1	tanh	False	0,01	32	5,0	0,0783	0,0781	0,0779	0,4464	0,454
48	1	tanh	True	0,01	32	5,1	0,0724	0,0721	0,072	0,5129	0,5191
49	1	relu	False	0,01	128	2,4	0,0877	0,0876	0,0876	0,2013	0,203
50	1	relu	True	0,01	128	2,4	0,0885	0,0886	0,0886	0,2064	0,1983
51	1	σ	False	0,01	128	2,4	0,0912	0,0912	0,0912	0,1552	0,1613
52	1	σ	True	0,01	128	2,5	0,0909	0,0908	0,0908	0,1323	0,1428
53	1	tanh	False	0,01	128	2,5	0,087	0,0869	0,0869	0,2395	0,2448
54	1	tanh	True	0,01	128	2,5	0,0875	0,0874	0,0875	0,2451	0,2491
55	2	relu	False	1	8	19,8	0,0033	0,0044	0,0045	0,9797	0,9704
56	2	relu	True	1	8	20,4	0,0044	0,0048	0,0053	0,973	0,9695
57	2	σ	False	1	8	19,4	0,0088	0,0083	0,0089	0,9439	0,9476
58	2	σ	True	1	8	19,8	0,0089	0,0084	0,0089	0,9436	0,9477
59	2	tanh	False	1	8	19,3	0,0052	0,0059	0,0063	0,968	0,9627
60	2	tanh	True	1	8	19,9	0,0045	0,0053	0,0053	0,9722	0,965
61	2	relu	False	1	32	7,0	0,0072	0,007	0,0075	0,9555	0,9565
62	2	relu	True	1	32	7,2	0,0072	0,0073	0,0076	0,9556	0,9549
63	2	σ	False	1	32	7,2	0,0138	0,0125	0,0129	0,9127	0,9213
64	2	σ	True	1	32	7,2	0,0138	0,0126	0,013	0,913	0,9201
65	2	tanh	False	1	32	7,2	0,0083	0,0078	0,0084	0,9492	0,9523
66	2	tanh	True	1	32	7,2	0,0085	0,0082	0,0086	0,946	0,9485
67	2	relu	False	1	128	3,2	0,0121	0,011	0,0116	0,9242	0,9323
68	2	relu	True	1	128	3,3	0,0126	0,0116	0,0121	0,9206	0,9279
69	2	σ	False	1	128	3,3	0,0231	0,021	0,0217	0,8752	0,8874
70	2	σ	True	1	128	3,3	0,0236	0,0213	0,0222	0,8729	0,888
71	2	tanh	False	1	128	3,3	0,0134	0,0122	0,0128	0,9164	0,9218
72	2	tanh	True	1	128	3,5	0,0132	0,0121	0,0125	0,9169	0,9221
73	2	relu	False	0,1	8	20,5	0,0102	0,0095	0,01	0,937	0,9421
74	2	relu	True	0,1	8	20,9	0,0103	0,0097	0,0102	0,936	0,9378
75	2	σ	False	0,1	8	20,9	0,0185	0,0164	0,0173	0,8922	0,9055
76	2	σ	True	0,1	8	21,0	0,0184	0,0165	0,0173	0,8913	0,9049
77	2	tanh	False	0,1	8	20,8	0,0117	0,0108	0,0113	0,9272	0,9312
78	2	tanh	True	0,1	8	21,4	0,0117	0,0108	0,0113	0,9267	0,9302

79	2	relu	False	0,1	32	7,7	0,0166	0,0149	0,0155	0,8974	0,9074
80	2	relu	True	0,1	32	7,7	0,0163	0,0146	0,0153	0,9005	0,9096
81	2	σ	False	0,1	32	7,6	0,0455	0,0439	0,0446	0,7768	0,8005
82	2	σ	True	0,1	32	7,8	0,0451	0,0434	0,0442	0,7581	0,7794
83	2	tanh	False	0,1	32	7,7	0,0168	0,015	0,0158	0,8968	0,9083
84	2	tanh	True	0,1	32	7,7	0,0169	0,0151	0,0158	0,8971	0,9082
85	2	relu	False	0,1	128	3,9	0,0316	0,0292	0,03	0,8378	0,8531
86	2	relu	True	0,1	128	3,7	0,0312	0,0288	0,0295	0,84	0,8576
87	2	σ	False	0,1	128	3,7	0,0815	0,0813	0,0814	0,437	0,4437
88	2	σ	True	0,1	128	3,9	0,0817	0,0817	0,0816	0,3635	0,3637
89	2	tanh	False	0,1	128	3,8	0,0287	0,0264	0,0272	0,8468	0,8654
90	2	tanh	True	0,1	128	3,9	0,0297	0,0273	0,0282	0,8394	0,8579
91	2	relu	False	0,01	8	22,3	0,0243	0,0219	0,0227	0,8635	0,8809
92	2	relu	True	0,01	8	22,4	0,0239	0,0215	0,0223	0,8637	0,8816
93	2	σ	False	0,01	8	22,1	0,0745	0,0742	0,0741	0,477	0,4921
94	2	σ	True	0,01	8	22,5	0,073	0,0728	0,0724	0,5186	0,53
95	2	tanh	False	0,01	8	22,2	0,023	0,0207	0,0216	0,8692	0,8842
96	2	tanh	True	0,01	8	22,7	0,0233	0,021	0,0217	0,8685	0,8815
97	2	relu	False	0,01	32	8,2	0,0629	0,062	0,062	0,6197	0,6374
98	2	relu	True	0,01	32	8,2	0,0609	0,0594	0,0599	0,6719	0,7033
99	2	σ	False	0,01	32	8,2	0,087	0,087	0,087	0,3576	0,3672
100	2	σ	True	0,01	32	8,4	0,087	0,087	0,087	0,3601	0,3649
101	2	tanh	False	0,01	32	8,3	0,0552	0,0533	0,054	0,6863	0,717
102	2	tanh	True	0,01	32	8,3	0,0495	0,0473	0,0485	0,7413	0,7677
103	2	relu	False	0,01	128	3,9	0,0856	0,0856	0,0856	0,328	0,3266
104	2	relu	True	0,01	128	4,1	0,0848	0,0845	0,0848	0,323	0,3322
105	2	σ	False	0,01	128	4,1	0,0912	0,0911	0,0911	0,1174	0,127
106	2	σ	True	0,01	128	4,2	0,0918	0,0918	0,0919	0,0659	0,0623
107	2	tanh	False	0,01	128	4,1	0,0851	0,0852	0,0851	0,3099	0,3044
108	2	tanh	True	0,01	128	4,1	0,0813	0,0811	0,081	0,3746	0,3841
109	3	relu	False	1	8	95,4	0,0027	0,004	0,0039	0,9846	0,9738
110	3	relu	True	1	8	86,6	0,0035	0,0042	0,0044	0,9787	0,9729
111	3	σ	False	1	8	87,3	0,0134	0,0126	0,0133	0,9122	0,9168
112	3	σ	True	1	8	93,4	0,0127	0,0116	0,0123	0,9165	0,9227
113	3	tanh	False	1	8	87,1	0,0086	0,0083	0,009	0,946	0,9475
114	3	tanh	True	1	8	81,8	0,0091	0,0088	0,0096	0,9427	0,9429
115	3	relu	False	1	32	27,7	0,0057	0,0058	0,0062	0,9658	0,9645
116	3	relu	True	1	32	27,9	0,0057	0,0058	0,0061	0,9662	0,9639
117	3	σ	False	1	32	25,3	0,016	0,0144	0,0151	0,8991	0,9082
118	3	σ	True	1	32	26,1	0,0169	0,0155	0,0158	0,8881	0,8949
119	3	tanh	False	1	32	25,8	0,0107	0,0106	0,0107	0,9319	0,9305
120	3	tanh	True	1	32	26,0	0,0106	0,0102	0,0108	0,9319	0,932
121	3	relu	False	1	128	9,9	0,0111	0,0102	0,0108	0,9321	0,9344
122	3	relu	True	1	128	10,0	0,0111	0,0103	0,0108	0,9317	0,9343
123	3	σ	False	1	128	10,0	0,0205	0,0184	0,0191	0,8761	0,8915
124	3	σ	True	1	128	10,3	0,0206	0,0185	0,0192	0,877	0,8915
125	3	tanh	False	1	128	10,0	0,013	0,0119	0,0124	0,918	0,9211
126	3	tanh	True	1	128	10,1	0,013	0,012	0,0124	0,9168	0,9211

127	3	relu	False	0,1	8	93,5	0,0092	0,0088	0,0091	0,9436	0,9457
128	3	relu	True	0,1	8	94,1	0,0094	0,0089	0,0093	0,944	0,9451
129	3	σ	False	0,1	8	93,9	0,0173	0,0155	0,016	0,8935	0,9044
130	3	σ	True	0,1	8	83,1	0,0175	0,0156	0,0163	0,8916	0,9022
131	3	tanh	False	0,1	8	89,5	0,0121	0,0113	0,0118	0,9235	0,9276
132	3	tanh	True	0,1	8	95,4	0,0121	0,0113	0,0118	0,9232	0,9266
133	3	relu	False	0,1	32	28,4	0,0147	0,0131	0,0138	0,9109	0,9217
134	3	relu	True	0,1	32	28,8	0,0148	0,0134	0,0138	0,9112	0,9183
135	3	σ	False	0,1	32	28,8	0,0347	0,0325	0,0334	0,8349	0,8558
136	3	σ	True	0,1	32	29,1	0,0342	0,0319	0,0328	0,838	0,8577
137	3	tanh	False	0,1	32	28,6	0,0155	0,0139	0,0145	0,9037	0,9128
138	3	tanh	True	0,1	32	25,8	0,0155	0,0138	0,0144	0,9038	0,9122
139	3	relu	False	0,1	128	11,0	0,0263	0,024	0,0248	0,8626	0,8787
140	3	relu	True	0,1	128	11,1	0,026	0,0235	0,0244	0,8658	0,8815
141	3	σ	False	0,1	128	11,1	0,0744	0,074	0,0741	0,5191	0,5327
142	3	σ	True	0,1	128	11,3	0,0735	0,0732	0,0731	0,5264	0,5417
143	3	tanh	False	0,1	128	11,1	0,0233	0,0208	0,0218	0,8672	0,8844
144	3	tanh	True	0,1	128	11,3	0,023	0,0205	0,0214	0,8706	0,8871
145	3	relu	False	0,01	8	95,7	0,021	0,0188	0,0196	0,8828	0,8985
146	3	relu	True	0,01	8	97,3	0,0207	0,0186	0,0194	0,8848	0,8985
147	3	σ	False	0,01	8	95,7	0,0619	0,061	0,0613	0,6755	0,6966
148	3	σ	True	0,01	8	89,1	0,063	0,0622	0,0624	0,6413	0,6575
149	3	tanh	False	0,01	8	96,8	0,0197	0,0175	0,0182	0,8833	0,8975
150	3	tanh	True	0,01	8	84,9	0,0195	0,0174	0,0181	0,8846	0,8996
151	3	relu	False	0,01	32	29,3	0,0514	0,05	0,05	0,7573	0,7823
152	3	relu	True	0,01	32	29,4	0,049	0,0475	0,0479	0,7431	0,7641
153	3	σ	False	0,01	32	26,5	0,0848	0,0847	0,0847	0,3655	0,3673
154	3	σ	True	0,01	32	26,9	0,0874	0,0874	0,0874	0,2662	0,2684
155	3	tanh	False	0,01	32	26,3	0,0353	0,033	0,034	0,8306	0,8497
156	3	tanh	True	0,01	32	26,8	0,0352	0,0326	0,0337	0,8213	0,8428
157	3	relu	False	0,01	128	12,1	0,0806	0,0805	0,0805	0,4767	0,4868
158	3	relu	True	0,01	128	10,9	0,0826	0,0824	0,0825	0,4146	0,4334
159	3	σ	False	0,01	128	10,9	0,0889	0,0889	0,0889	0,2834	0,2829
160	3	σ	True	0,01	128	12,1	0,0892	0,0892	0,0891	0,2614	0,2662
161	3	tanh	False	0,01	128	10,9	0,0737	0,0729	0,0734	0,5389	0,5552
162	3	tanh	True	0,01	128	11,0	0,0725	0,0719	0,0722	0,5769	0,5997
163	4	relu	False	1	8	24,4	0,0151	0,0145	0,0147	0,9083	0,9108
164	4	relu	True	1	8	24,8	0,1389	0,1357	0,1378	0,3046	0,321
165	4	σ	False	1	8	24,8	0,0146	0,0137	0,0145	0,9082	0,9138
166	4	σ	True	1	8	25,3	0,0144	0,0133	0,0143	0,9079	0,9134
167	4	tanh	False	1	8	24,9	0,013	0,0125	0,0129	0,9162	0,9171
168	4	tanh	True	1	8	25,4	0,0122	0,0117	0,0125	0,9215	0,9252
169	4	relu	False	1	32	9,2	0,0133	0,0132	0,0133	0,9155	0,9153
170	4	relu	True	1	32	9,4	0,0262	0,0243	0,0262	0,8243	0,8359
171	4	σ	False	1	32	9,2	0,0501	0,0493	0,0494	0,6467	0,6539
172	4	σ	True	1	32	9,4	0,047	0,0464	0,0466	0,6662	0,6706
173	4	tanh	False	1	32	9,3	0,0123	0,0118	0,0124	0,9205	0,9233
174	4	tanh	True	1	32	9,4	0,0135	0,0125	0,0135	0,9129	0,9194

175	4	relu	False	1	128	5,4	0,0152	0,0138	0,0146	0,9019	0,9085
176	4	relu	True	1	128	5,4	0,0174	0,0159	0,0168	0,885	0,8932
177	4	σ	False	1	128	5,3	0,088	0,088	0,088	0,3017	0,2967
178	4	σ	True	1	128	5,4	0,0875	0,0876	0,0875	0,2587	0,2571
179	4	tanh	False	1	128	5,4	0,0164	0,0149	0,0154	0,9005	0,9094
180	4	tanh	True	1	128	5,6	0,0174	0,0165	0,0172	0,8953	0,9005
181	4	relu	False	0,1	8	26,1	0,0139	0,0132	0,0136	0,9103	0,9136
182	4	relu	True	0,1	8	26,6	0,0142	0,0129	0,0143	0,9078	0,9162
183	4	σ	False	0,1	8	26,3	0,0823	0,0825	0,0821	0,278	0,2753
184	4	σ	True	0,1	8	27,2	0,0796	0,0798	0,0794	0,319	0,321
185	4	tanh	False	0,1	8	26,6	0,0151	0,0144	0,0152	0,9055	0,9078
186	4	tanh	True	0,1	8	27,1	0,0147	0,0138	0,0143	0,9079	0,9118
187	4	relu	False	0,1	32	9,8	0,0471	0,0461	0,0466	0,6293	0,6331
188	4	relu	True	0,1	32	10,1	0,0214	0,0195	0,0205	0,861	0,8715
189	4	σ	False	0,1	32	9,9	0,0891	0,0891	0,089	0,1531	0,1461
190	4	σ	True	0,1	32	10,1	0,09	0,0899	0,09	0,1686	0,1751
191	4	tanh	False	0,1	32	10,1	0,0332	0,0314	0,0323	0,7886	0,8017
192	4	tanh	True	0,1	32	10,3	0,0304	0,0281	0,0289	0,832	0,8519
193	4	relu	False	0,1	128	5,7	0,0771	0,0761	0,0766	0,2889	0,3036
194	4	relu	True	0,1	128	5,8	0,0707	0,0701	0,0705	0,4409	0,4452
195	4	σ	False	0,1	128	5,9	0,09	0,0901	0,0901	0,1846	0,1946
196	4	σ	True	0,1	128	6,0	0,0903	0,0903	0,0902	0,1056	0,1105
197	4	tanh	False	0,1	128	6,0	0,0626	0,0617	0,0623	0,6316	0,6499
198	4	tanh	True	0,1	128	6,0	0,0653	0,0646	0,0651	0,5939	0,6032
199	4	relu	False	0,01	8	27,9	0,0497	0,0485	0,049	0,6637	0,6734
200	4	relu	True	0,01	8	28,6	0,0681	0,0673	0,0679	0,4988	0,5092
201	4	σ	False	0,01	8	27,9	0,09	0,09	0,09	0,1795	0,1813
202	4	σ	True	0,01	8	28,9	0,0902	0,0903	0,0902	0,1139	0,1069
203	4	tanh	False	0,01	8	28,3	0,0536	0,0526	0,0533	0,6611	0,6755
204	4	tanh	True	0,01	8	28,9	0,0584	0,0573	0,0577	0,6119	0,6294
205	4	relu	False	0,01	32	10,7	0,0875	0,0874	0,0875	0,2039	0,2089
206	4	relu	True	0,01	32	10,8	0,0863	0,0862	0,0862	0,118	0,1186
207	4	σ	False	0,01	32	10,7	0,091	0,091	0,0911	0,1035	0,109
208	4	σ	True	0,01	32	10,8	0,0924	0,0924	0,0923	0,0994	0,099
209	4	tanh	False	0,01	32	10,8	0,0805	0,0802	0,0802	0,5232	0,536
210	4	tanh	True	0,01	32	11,1	0,0819	0,0817	0,0817	0,4568	0,4764
211	4	relu	False	0,01	128	6,2	0,0892	0,0891	0,0891	0,129	0,1296
212	4	relu	True	0,01	128	6,4	0,0892	0,0892	0,0892	0,1497	0,1416
213	4	σ	False	0,01	128	6,2	0,0918	0,0918	0,0918	0,0986	0,0991
214	4	σ	True	0,01	128	6,6	0,0905	0,0906	0,0906	0,0649	0,0641
215	4	tanh	False	0,01	128	6,2	0,0892	0,089	0,0892	0,1696	0,1755
216	4	tanh	True	0,01	128	6,4	0,0876	0,0874	0,0875	0,2119	0,2126
217	5	relu	False	1	8	33,7	0,0167	0,0151	0,0168	0,9072	0,9148
218	5	relu	True	1	8	34,6	0,1579	0,1581	0,158	0,2103	0,2095
219	5	σ	False	1	8	34,0	0,0099	0,0095	0,0099	0,9371	0,9398
220	5	σ	True	1	8	34,9	0,0103	0,0098	0,0104	0,9363	0,9376
221	5	tanh	False	1	8	34,3	0,0062	0,0068	0,0071	0,9605	0,955
222	5	tanh	True	1	8	34,7	0,0061	0,0067	0,007	0,9611	0,957

223	5	relu	False	1	32	13,3	0,0062	0,0064	0,0068	0,9615	0,9585
224	5	relu	True	1	32	13,2	0,1096	0,1091	0,1103	0,447	0,4486
225	5	σ	False	1	32	13,3	0,0241	0,022	0,0227	0,8629	0,8805
226	5	σ	True	1	32	13,4	0,0243	0,0229	0,0236	0,8648	0,876
227	5	tanh	False	1	32	13,2	0,0072	0,0072	0,0079	0,9546	0,9556
228	5	tanh	True	1	32	13,5	0,0075	0,0074	0,0082	0,9531	0,9522
229	5	relu	False	1	128	7,4	0,0111	0,0105	0,011	0,9308	0,9323
230	5	relu	True	1	128	7,4	0,0127	0,0117	0,0123	0,9181	0,9238
231	5	σ	False	1	128	7,4	0,0807	0,0807	0,0807	0,3362	0,3357
232	5	σ	True	1	128	7,5	0,0833	0,0833	0,0833	0,2676	0,2651
233	5	tanh	False	1	128	7,4	0,014	0,0131	0,0136	0,9179	0,9209
234	5	tanh	True	1	128	7,5	0,0143	0,0133	0,0138	0,9135	0,9199
235	5	relu	False	0,1	8	35,7	0,0092	0,009	0,0093	0,9425	0,9408
236	5	relu	True	0,1	8	36,7	0,0096	0,0091	0,0096	0,9384	0,9409
237	5	σ	False	0,1	8	35,8	0,0749	0,0751	0,0745	0,4263	0,4359
238	5	σ	True	0,1	8	36,7	0,0678	0,0672	0,0675	0,5063	0,5131
239	5	tanh	False	0,1	8	36,1	0,0111	0,0105	0,0109	0,9324	0,9342
240	5	tanh	True	0,1	8	37,0	0,011	0,0103	0,0108	0,9321	0,9381
241	5	relu	False	0,1	32	13,9	0,02	0,0185	0,0193	0,8826	0,894
242	5	relu	True	0,1	32	14,0	0,0193	0,0176	0,018	0,8798	0,8915
243	5	σ	False	0,1	32	14,0	0,0883	0,0883	0,0882	0,2687	0,2692
244	5	σ	True	0,1	32	14,2	0,0887	0,0887	0,0887	0,1594	0,1496
245	5	tanh	False	0,1	32	14,0	0,0223	0,0202	0,021	0,8727	0,8888
246	5	tanh	True	0,1	32	14,3	0,0273	0,0252	0,0262	0,841	0,8609
247	5	relu	False	0,1	128	7,8	0,0568	0,0548	0,0562	0,5911	0,6101
248	5	relu	True	0,1	128	7,9	0,0631	0,0624	0,0625	0,4904	0,5033
249	5	σ	False	0,1	128	7,9	0,0903	0,0903	0,0903	0,1188	0,1109
250	5	σ	True	0,1	128	7,9	0,0897	0,0897	0,0897	0,1912	0,1957
251	5	tanh	False	0,1	128	7,9	0,0509	0,0492	0,0506	0,6687	0,6889
252	5	tanh	True	0,1	128	8,0	0,0469	0,0455	0,0462	0,7774	0,8011
253	5	relu	False	0,01	8	37,2	0,0343	0,0324	0,0335	0,7882	0,8075
254	5	relu	True	0,01	8	38,0	0,0451	0,0431	0,0438	0,6946	0,7133
255	5	σ	False	0,01	8	37,4	0,0896	0,0896	0,0896	0,2361	0,2344
256	5	σ	True	0,01	8	38,3	0,0906	0,0907	0,0906	0,1432	0,1368
257	5	tanh	False	0,01	8	37,8	0,0395	0,0374	0,0385	0,7717	0,7915
258	5	tanh	True	0,01	8	38,4	0,0391	0,0373	0,038	0,8042	0,8279
259	5	relu	False	0,01	32	14,4	0,0875	0,0874	0,0873	0,1645	0,1669
260	5	relu	True	0,01	32	14,9	0,0812	0,0808	0,0809	0,3671	0,3744
261	5	σ	False	0,01	32	14,6	0,0904	0,0904	0,0903	0,1242	0,1207
262	5	σ	True	0,01	32	15,0	0,0907	0,0906	0,0907	0,1134	0,1199
263	5	tanh	False	0,01	32	15,0	0,0729	0,0726	0,0725	0,4859	0,5023
264	5	tanh	True	0,01	32	15,1	0,0704	0,0697	0,0702	0,54	0,559
265	5	relu	False	0,01	128	8,1	0,089	0,0891	0,0891	0,1729	0,1718
266	5	relu	True	0,01	128	8,2	0,0889	0,0888	0,0887	0,1968	0,2041
267	5	σ	False	0,01	128	8,2	0,0914	0,0913	0,0915	0,0988	0,0992
268	5	σ	True	0,01	128	8,3	0,0926	0,0925	0,0927	0,0986	0,0991
269	5	tanh	False	0,01	128	8,3	0,0873	0,0874	0,087	0,2459	0,2434
270	5	tanh	True	0,01	128	8,4	0,0839	0,0838	0,0838	0,358	0,3587

271	6	relu	False	1	8	34,7	0,1796	0,1794	0,1798	0,1021	0,1031
272	6	relu	True	1	8	35,5	0,1573	0,1577	0,1578	0,213	0,2109
273	6	σ	False	1	8	35,0	0,0131	0,0122	0,0127	0,9155	0,9209
274	6	σ	True	1	8	35,9	0,0132	0,0126	0,0131	0,9148	0,9174
275	6	tanh	False	1	8	35,0	0,0112	0,0109	0,0113	0,9282	0,9276
276	6	tanh	True	1	8	36,2	0,0112	0,0112	0,0119	0,9288	0,9285
277	6	relu	False	1	32	13,6	0,0141	0,0133	0,0135	0,9105	0,9143
278	6	relu	True	1	32	14,3	0,1229	0,1225	0,1224	0,3836	0,3855
279	6	σ	False	1	32	13,6	0,0359	0,0346	0,0351	0,7842	0,8021
280	6	σ	True	1	32	13,9	0,0388	0,0372	0,038	0,7712	0,7927
281	6	tanh	False	1	32	13,7	0,0119	0,0109	0,0119	0,9241	0,9291
282	6	tanh	True	1	32	14,0	0,0116	0,0111	0,0119	0,9239	0,9268
283	6	relu	False	1	128	8,0	0,0147	0,0136	0,0137	0,9037	0,9078
284	6	relu	True	1	128	8,2	0,0138	0,0126	0,0133	0,9094	0,9165
285	6	σ	False	1	128	8,1	0,0889	0,0889	0,0888	0,2997	0,2983
286	6	σ	True	1	128	8,3	0,0867	0,0868	0,0867	0,2103	0,2038
287	6	tanh	False	1	128	8,3	0,0143	0,0134	0,0139	0,91	0,9144
288	6	tanh	True	1	128	8,3	0,0142	0,0131	0,0133	0,9077	0,916
289	6	relu	False	0,1	8	36,7	0,0125	0,0118	0,0126	0,9193	0,9221
290	6	relu	True	0,1	8	37,8	0,0131	0,012	0,0129	0,9146	0,9213
291	6	σ	False	0,1	8	37,1	0,0844	0,0843	0,0844	0,372	0,3825
292	6	σ	True	0,1	8	37,9	0,0786	0,0787	0,0784	0,3017	0,3043
293	6	tanh	False	0,1	8	37,2	0,0123	0,0117	0,0123	0,9209	0,9245
294	6	tanh	True	0,1	8	38,2	0,0119	0,0112	0,012	0,9235	0,9282
295	6	relu	False	0,1	32	14,4	0,0185	0,0171	0,0177	0,882	0,8895
296	6	relu	True	0,1	32	14,6	0,0195	0,018	0,0186	0,8781	0,8859
297	6	σ	False	0,1	32	14,3	0,0898	0,0898	0,0898	0,1251	0,1198
298	6	σ	True	0,1	32	14,6	0,0896	0,0896	0,0896	0,1144	0,1071
299	6	tanh	False	0,1	32	14,5	0,0198	0,0181	0,0189	0,8805	0,8941
300	6	tanh	True	0,1	32	14,8	0,0193	0,0179	0,0185	0,8855	0,8954
301	6	relu	False	0,1	128	8,6	0,0827	0,082	0,0826	0,2818	0,2916
302	6	relu	True	0,1	128	8,7	0,0778	0,0772	0,0774	0,3182	0,3278
303	6	σ	False	0,1	128	8,8	0,0899	0,0899	0,0899	0,1804	0,1851
304	6	σ	True	0,1	128	8,8	0,09	0,0901	0,09	0,148	0,1429
305	6	tanh	False	0,1	128	8,7	0,0576	0,057	0,0572	0,6201	0,6299
306	6	tanh	True	0,1	128	8,9	0,0612	0,06	0,0607	0,6826	0,7068
307	6	relu	False	0,01	8	38,6	0,0578	0,057	0,0567	0,6257	0,6371
308	6	relu	True	0,01	8	39,5	0,0691	0,0685	0,0684	0,531	0,5489
309	6	σ	False	0,01	8	39,2	0,0899	0,0899	0,0899	0,1136	0,1064
310	6	σ	True	0,01	8	39,9	0,0902	0,0902	0,0903	0,1136	0,1064
311	6	tanh	False	0,01	8	39,0	0,0383	0,0362	0,0374	0,7673	0,7841
312	6	tanh	True	0,01	8	39,9	0,0423	0,0408	0,0412	0,7139	0,7256
313	6	relu	False	0,01	32	15,1	0,0884	0,0883	0,0883	0,1804	0,1905
314	6	relu	True	0,01	32	15,3	0,0885	0,0884	0,0885	0,2116	0,2119
315	6	σ	False	0,01	32	15,2	0,0904	0,0903	0,0904	0,1015	0,1033
316	6	σ	True	0,01	32	15,3	0,0904	0,0905	0,0904	0,1136	0,1064
317	6	tanh	False	0,01	32	15,2	0,0826	0,0824	0,0823	0,4393	0,4459
318	6	tanh	True	0,01	32	15,5	0,0797	0,0797	0,0795	0,4193	0,4292

319	6	relu	False	0,01	128	9,0	0,0898	0,0899	0,0899	0,1351	0,1331
320	6	relu	True	0,01	128	9,2	0,0898	0,0898	0,0899	0,1303	0,1311
321	6	σ	False	0,01	128	9,4	0,0934	0,0935	0,0934	0,0998	0,0961
322	6	σ	True	0,01	128	9,4	0,0918	0,0916	0,0918	0,1035	0,109
323	6	tanh	False	0,01	128	9,3	0,0887	0,0886	0,0887	0,2041	0,2069
324	6	tanh	True	0,01	128	9,5	0,089	0,0889	0,089	0,1856	0,1906
325	7	relu	False	1	8	46,6	0,1606	0,16	0,1604	0,1969	0,1999
326	7	relu	True	1	8	48,0	0,1515	0,1525	0,1522	0,2419	0,2372
327	7	σ	False	1	8	47,2	0,0095	0,0089	0,0095	0,9383	0,943
328	7	σ	True	1	8	48,3	0,01	0,0096	0,0101	0,9359	0,9368
329	7	tanh	False	1	8	47,7	0,0049	0,0053	0,0057	0,9682	0,9661
330	7	tanh	True	1	8	48,3	0,0042	0,0052	0,0053	0,9732	0,9661
331	7	relu	False	1	32	18,4	0,0079	0,0075	0,0081	0,9496	0,95
332	7	relu	True	1	32	18,6	0,1001	0,0977	0,1001	0,4913	0,5033
333	7	σ	False	1	32	18,5	0,0168	0,0156	0,0161	0,8945	0,9026
334	7	σ	True	1	32	18,9	0,0166	0,0152	0,0158	0,8966	0,9082
335	7	tanh	False	1	32	18,7	0,0066	0,0066	0,0069	0,9587	0,9577
336	7	tanh	True	1	32	19,0	0,0069	0,0069	0,0074	0,9572	0,9548
337	7	relu	False	1	128	10,6	0,0105	0,0098	0,0102	0,9343	0,9385
338	7	relu	True	1	128	10,5	0,012	0,0114	0,0119	0,9229	0,9261
339	7	σ	False	1	128	10,6	0,0727	0,0727	0,0724	0,4768	0,4821
340	7	σ	True	1	128	10,7	0,07	0,0698	0,0697	0,4774	0,4909
341	7	tanh	False	1	128	10,7	0,0117	0,0108	0,0115	0,9263	0,9306
342	7	tanh	True	1	128	10,9	0,0115	0,0106	0,0112	0,9267	0,93
343	7	relu	False	0,1	8	50,1	0,0087	0,0082	0,0089	0,9445	0,9481
344	7	relu	True	0,1	8	49,8	0,0095	0,0091	0,0095	0,9394	0,9402
345	7	σ	False	0,1	8	48,7	0,0439	0,0425	0,0432	0,74	0,7606
346	7	σ	True	0,1	8	50,2	0,0406	0,039	0,0398	0,7645	0,7865
347	7	tanh	False	0,1	8	49,2	0,0096	0,009	0,0096	0,9397	0,9436
348	7	tanh	True	0,1	8	50,5	0,0099	0,0092	0,0099	0,9388	0,941
349	7	relu	False	0,1	32	19,1	0,0144	0,013	0,0135	0,9085	0,9195
350	7	relu	True	0,1	32	19,2	0,014	0,0127	0,0133	0,9117	0,9205
351	7	σ	False	0,1	32	19,0	0,0888	0,0888	0,0888	0,2464	0,2443
352	7	σ	True	0,1	32	19,4	0,0875	0,0875	0,0874	0,2663	0,2613
353	7	tanh	False	0,1	32	19,3	0,0151	0,0135	0,0141	0,9054	0,9145
354	7	tanh	True	0,1	32	19,5	0,0149	0,0134	0,0139	0,9055	0,9133
355	7	relu	False	0,1	128	11,1	0,0351	0,033	0,0335	0,79	0,808
356	7	relu	True	0,1	128	11,2	0,0304	0,0283	0,0291	0,829	0,8453
357	7	σ	False	0,1	128	10,9	0,0901	0,0901	0,0901	0,1694	0,1714
358	7	σ	True	0,1	128	11,1	0,0894	0,0894	0,0894	0,1243	0,1175
359	7	tanh	False	0,1	128	11,1	0,0277	0,0252	0,0263	0,8453	0,8668
360	7	tanh	True	0,1	128	11,2	0,0269	0,0245	0,0255	0,8501	0,868
361	7	relu	False	0,01	8	52,6	0,0212	0,0193	0,02	0,8742	0,8862
362	7	relu	True	0,01	8	52,2	0,024	0,0218	0,0225	0,8629	0,8801
363	7	σ	False	0,01	8	51,1	0,0901	0,0902	0,0901	0,1858	0,1832
364	7	σ	True	0,01	8	51,9	0,0894	0,0894	0,0893	0,1159	0,1086
365	7	tanh	False	0,01	8	53,3	0,0212	0,0189	0,02	0,876	0,8917
366	7	tanh	True	0,01	8	52,6	0,0211	0,0189	0,0198	0,8748	0,8896

367	7	relu	False	0,01	32	20,7	0,0767	0,0762	0,0763	0,5228	0,5365
368	7	relu	True	0,01	32	20,7	0,0709	0,0703	0,0706	0,5761	0,5876
369	7	σ	False	0,01	32	20,2	0,0908	0,0909	0,0907	0,1392	0,1291
370	7	σ	True	0,01	32	20,9	0,0902	0,0902	0,0903	0,1215	0,1113
371	7	tanh	False	0,01	32	20,3	0,0488	0,0469	0,0478	0,7276	0,7479
372	7	tanh	True	0,01	32	20,5	0,049	0,0473	0,0481	0,7238	0,7473
373	7	relu	False	0,01	128	11,5	0,0875	0,0874	0,0874	0,2363	0,2445
374	7	relu	True	0,01	128	11,7	0,0885	0,0884	0,0883	0,2197	0,2336
375	7	σ	False	0,01	128	11,7	0,0908	0,0908	0,0908	0,119	0,1118
376	7	σ	True	0,01	128	11,9	0,091	0,091	0,091	0,0935	0,101
377	7	tanh	False	0,01	128	12,6	0,0803	0,0795	0,0805	0,3822	0,4037
378	7	tanh	True	0,01	128	12,0	0,0811	0,0809	0,0812	0,4397	0,4404

Tabelle 5: Tabelle mit den Ergebnissen der Experimente mit FFNNs (Quelle: Eigene Tabelle)

E.2 Tabelle der Ergebnisse der Experimente mit CNNs

Die Spalten Aktivierungsfunktion (Wert: relu), Pooling (Wert: Max), Bias (Wert: True) und Epoch (Wert: 3) wurden weggelassen, da ihre Werte für alle Konfigurationen konstant sind. Diese Werte lieferten bei Vorversuchen jene CNNs mit den höchsten Trefferquoten.

Index-Nr	Bauart-Nr	Kernel	Stride	Lernrate	Mini Batch	Trainingszeit (s)	Train-Cost	Val.-Cost	Test-Cost	Train-Quote	Val.-Quote	Test-Quote
1	1	2x2	1x1	1	8	35,4	0,0027	0,0039	0,0037	0,9836	0,975	0,975
2	1	2x2	1x2	1	8	23,5	0,0046	0,005	0,0053	0,9707	0,9682	0,965
3	1	2x2	2x1	1	8	23,6	0,005	0,0055	0,0054	0,9683	0,9636	0,9643
4	1	2x2	2x2	1	8	17,0	0,0072	0,0071	0,0073	0,9535	0,9549	0,9504
5	1	2x3	1x1	1	8	44,9	0,0029	0,0038	0,0037	0,9819	0,975	0,976
6	1	2x3	1x2	1	8	26,4	0,0038	0,0039	0,0042	0,9752	0,9736	0,9733
7	1	2x3	2x1	1	8	25,8	0,0047	0,0052	0,0051	0,9713	0,9669	0,9663
8	1	2x3	2x2	1	8	18,6	0,0059	0,0062	0,0059	0,9608	0,9597	0,9611
9	1	3x2	1x1	1	8	45,6	0,0028	0,0037	0,0039	0,9825	0,9775	0,9745
10	1	3x2	1x2	1	8	26,9	0,0036	0,0043	0,0043	0,9775	0,9722	0,9713
11	1	3x2	2x1	1	8	26,5	0,0047	0,005	0,0052	0,9693	0,9677	0,9663
12	1	3x2	2x2	1	8	18,9	0,0054	0,0055	0,0055	0,9658	0,9647	0,9634
13	1	3x3	1x1	1	8	56,3	0,0023	0,0031	0,003	0,9855	0,9798	0,981
14	1	3x3	1x2	1	8	30,0	0,0033	0,0036	0,0035	0,9791	0,9752	0,9764
15	1	3x3	2x1	1	8	29,5	0,0033	0,0037	0,0037	0,9785	0,9766	0,9756
16	1	3x3	2x2	1	8	23,5	0,0049	0,0048	0,0048	0,9677	0,9685	0,9686
17	1	2x2	1x1	1	32	20,1	0,0059	0,0059	0,0061	0,9628	0,9618	0,96
18	1	2x2	1x2	1	32	12,6	0,0069	0,0063	0,0067	0,9576	0,9605	0,9565
19	1	2x2	2x1	1	32	12,3	0,0085	0,0083	0,0086	0,9464	0,946	0,9452
20	1	2x2	2x2	1	32	7,7	0,0137	0,0126	0,0132	0,9112	0,9168	0,9123
21	1	2x3	1x1	1	32	24,3	0,0037	0,0039	0,0041	0,9775	0,9736	0,9732
22	1	2x3	1x2	1	32	15,2	0,0074	0,0068	0,007	0,9531	0,9566	0,9552
23	1	2x3	2x1	1	32	15,0	0,0063	0,0064	0,0063	0,9602	0,9597	0,9594
24	1	2x3	2x2	1	32	9,0	0,0099	0,0093	0,0091	0,9356	0,9397	0,9408
25	1	3x2	1x1	1	32	24,8	0,0045	0,0045	0,0047	0,9714	0,9717	0,9697
26	1	3x2	1x2	1	32	15,8	0,0058	0,0055	0,0058	0,9643	0,9655	0,9611
27	1	3x2	2x1	1	32	15,3	0,0047	0,0048	0,0049	0,9709	0,9701	0,9676
28	1	3x2	2x2	1	32	9,1	0,0082	0,0077	0,0081	0,9481	0,9516	0,9484
29	1	3x3	1x1	1	32	27,0	0,0042	0,0041	0,0044	0,973	0,9738	0,9711
30	1	3x3	1x2	1	32	18,6	0,0042	0,0041	0,0043	0,9733	0,9738	0,973
31	1	3x3	2x1	1	32	17,7	0,0045	0,0045	0,0045	0,9717	0,9719	0,9712
32	1	3x3	2x2	1	32	10,1	0,0094	0,0088	0,0087	0,9385	0,9415	0,9429
33	1	2x2	1x1	1	128	16,3	0,0132	0,0119	0,0124	0,9129	0,9217	0,9181
34	1	2x2	1x2	1	128	7,2	0,0158	0,0141	0,0149	0,8939	0,9069	0,9018
35	1	2x2	2x1	1	128	7,2	0,015	0,014	0,0145	0,9017	0,9084	0,9042
36	1	2x2	2x2	1	128	4,5	0,0211	0,0191	0,0203	0,8597	0,8735	0,8648

37	1	2x3	1x1	1	128	18,8	0,0111	0,0102	0,0106	0,9264	0,9331	0,9317
38	1	2x3	1x2	1	128	7,6	0,0119	0,0106	0,0109	0,9217	0,9296	0,928
39	1	2x3	2x1	1	128	7,6	0,013	0,012	0,0124	0,917	0,9228	0,9206
40	1	2x3	2x2	1	128	5,4	0,0181	0,0163	0,0168	0,8814	0,8929	0,891
41	1	3x2	1x1	1	128	19,4	0,0092	0,0085	0,0087	0,9411	0,9454	0,943
42	1	3x2	1x2	1	128	7,9	0,0115	0,0104	0,0107	0,926	0,9331	0,9313
43	1	3x2	2x1	1	128	7,9	0,0107	0,0098	0,01	0,9332	0,9393	0,9394
44	1	3x2	2x2	1	128	5,4	0,0163	0,0148	0,0153	0,8945	0,904	0,9018
45	1	3x3	1x1	1	128	24,7	0,0055	0,005	0,0053	0,9664	0,9699	0,9674
46	1	3x3	1x2	1	128	9,8	0,0082	0,0074	0,0075	0,9475	0,9531	0,9518
47	1	3x3	2x1	1	128	9,5	0,0087	0,0082	0,0083	0,9462	0,9487	0,948
48	1	3x3	2x2	1	128	5,8	0,0173	0,0156	0,0165	0,8867	0,9007	0,8948
49	1	2x2	1x1	0,1	8	43,4	0,0085	0,0079	0,0085	0,9473	0,9511	0,946
50	1	2x2	1x2	0,1	8	31,0	0,0127	0,0114	0,0121	0,9169	0,9264	0,9209
51	1	2x2	2x1	0,1	8	30,6	0,0154	0,0144	0,0148	0,8983	0,9026	0,9026
52	1	2x2	2x2	0,1	8	23,0	0,018	0,0165	0,0173	0,8807	0,8913	0,8853
53	1	2x3	1x1	0,1	8	50,0	0,0069	0,0065	0,0068	0,9575	0,9601	0,9573
54	1	2x3	1x2	0,1	8	34,2	0,01	0,0091	0,0094	0,9351	0,9407	0,9398
55	1	2x3	2x1	0,1	8	33,8	0,0106	0,01	0,0101	0,9324	0,9355	0,9361
56	1	2x3	2x2	0,1	8	24,6	0,0143	0,0129	0,0133	0,9074	0,918	0,916
57	1	3x2	1x1	0,1	8	49,2	0,0064	0,0058	0,006	0,9618	0,9656	0,9615
58	1	3x2	1x2	0,1	8	32,3	0,0094	0,0084	0,0089	0,9397	0,9473	0,9416
59	1	3x2	2x1	0,1	8	31,8	0,0085	0,0081	0,0081	0,9465	0,949	0,9484
60	1	3x2	2x2	0,1	8	23,2	0,0119	0,0109	0,0114	0,9248	0,9319	0,9254
61	1	3x3	1x1	0,1	8	58,4	0,0046	0,0044	0,0044	0,9725	0,9725	0,9727
62	1	3x3	1x2	0,1	8	33,5	0,0061	0,0056	0,0057	0,9622	0,966	0,9647
63	1	3x3	2x1	0,1	8	32,7	0,0073	0,0066	0,0067	0,9546	0,9587	0,9584
64	1	3x3	2x2	0,1	8	26,0	0,0093	0,0085	0,0088	0,9413	0,9451	0,9449
65	1	2x2	1x1	0,1	32	22,1	0,0123	0,0112	0,0116	0,9207	0,9263	0,9251
66	1	2x2	1x2	0,1	32	14,1	0,0158	0,0138	0,0145	0,8969	0,9112	0,9034
67	1	2x2	2x1	0,1	32	13,8	0,0181	0,0164	0,017	0,8815	0,8924	0,8878
68	1	2x2	2x2	0,1	32	9,0	0,0234	0,0209	0,0222	0,8465	0,8643	0,856
69	1	2x3	1x1	0,1	32	26,2	0,0128	0,0115	0,0118	0,9177	0,9254	0,9226
70	1	2x3	1x2	0,1	32	16,6	0,0143	0,0124	0,0131	0,9091	0,92	0,9143
71	1	2x3	2x1	0,1	32	16,7	0,0166	0,015	0,0155	0,892	0,9009	0,8984
72	1	2x3	2x2	0,1	32	10,3	0,0211	0,0187	0,0196	0,8644	0,878	0,8711
73	1	3x2	1x1	0,1	32	26,7	0,0133	0,0123	0,0125	0,9137	0,9195	0,9191
74	1	3x2	1x2	0,1	32	17,0	0,0157	0,0141	0,0147	0,898	0,9095	0,9058
75	1	3x2	2x1	0,1	32	16,5	0,0161	0,0149	0,0151	0,8975	0,9048	0,9044
76	1	3x2	2x2	0,1	32	10,5	0,0226	0,0203	0,0217	0,8532	0,8692	0,8632
77	1	3x3	1x1	0,1	32	29,5	0,0092	0,0082	0,0084	0,9433	0,9483	0,9481
78	1	3x3	1x2	0,1	32	20,3	0,0123	0,0108	0,0112	0,9208	0,9303	0,9301
79	1	3x3	2x1	0,1	32	19,3	0,015	0,0139	0,0143	0,9042	0,909	0,9071
80	1	3x3	2x2	0,1	32	11,9	0,0193	0,0174	0,018	0,8761	0,8839	0,888
81	1	2x2	1x1	0,1	128	18,0	0,0158	0,014	0,0147	0,9007	0,9111	0,9075
82	1	2x2	1x2	0,1	128	8,5	0,0235	0,0208	0,022	0,8567	0,877	0,8686
83	1	2x2	2x1	0,1	128	8,7	0,0259	0,0233	0,0243	0,8434	0,8601	0,8525
84	1	2x2	2x2	0,1	128	5,4	0,0735	0,0729	0,0732	0,4791	0,4834	0,4849

85	1	2x3	1x1	0,1	128	19,9	0,0156	0,0135	0,0143	0,8988	0,9166	0,9091
86	1	2x3	1x2	0,1	128	9,5	0,0246	0,0219	0,023	0,8589	0,8747	0,8725
87	1	2x3	2x1	0,1	128	9,0	0,0252	0,0226	0,0236	0,8417	0,8554	0,8548
88	1	2x3	2x2	0,1	128	6,5	0,0481	0,0461	0,0474	0,7056	0,7306	0,7137
89	1	3x2	1x1	0,1	128	20,4	0,0163	0,0145	0,015	0,8956	0,908	0,9061
90	1	3x2	1x2	0,1	128	9,5	0,0235	0,021	0,0222	0,856	0,8717	0,8668
91	1	3x2	2x1	0,1	128	9,2	0,0281	0,0259	0,0265	0,8257	0,8396	0,8383
92	1	3x2	2x2	0,1	128	6,6	0,0622	0,0613	0,0618	0,5872	0,604	0,5991
93	1	3x3	1x1	0,1	128	25,8	0,015	0,0132	0,0138	0,9056	0,9174	0,9127
94	1	3x3	1x2	0,1	128	11,3	0,0248	0,0219	0,023	0,8558	0,873	0,87
95	1	3x3	2x1	0,1	128	11,5	0,031	0,0284	0,0296	0,8145	0,834	0,826
96	1	3x3	2x2	0,1	128	7,1	0,0695	0,0689	0,0689	0,4814	0,4925	0,4924
97	1	2x2	1x1	0,01	8	48,1	0,0148	0,0131	0,0137	0,9044	0,915	0,9105
98	1	2x2	1x2	0,01	8	32,4	0,0194	0,017	0,0182	0,8758	0,8903	0,8841
99	1	2x2	2x1	0,01	8	37,9	0,0223	0,0199	0,021	0,8598	0,8728	0,869
100	1	2x2	2x2	0,01	8	30,1	0,0415	0,0391	0,04	0,7435	0,7715	0,7622
101	1	2x3	1x1	0,01	8	53,2	0,014	0,0123	0,0128	0,9093	0,9211	0,916
102	1	2x3	1x2	0,01	8	37,6	0,0183	0,0159	0,017	0,8821	0,8971	0,8926
103	1	2x3	2x1	0,01	8	37,4	0,0198	0,0178	0,0185	0,8752	0,8878	0,884
104	1	2x3	2x2	0,01	8	28,0	0,0416	0,039	0,0403	0,7454	0,7744	0,7586
105	1	3x2	1x1	0,01	8	54,7	0,0144	0,0129	0,0133	0,9074	0,919	0,9132
106	1	3x2	1x2	0,01	8	36,9	0,0189	0,0166	0,0177	0,878	0,8941	0,8868
107	1	3x2	2x1	0,01	8	36,5	0,0224	0,0203	0,0213	0,8588	0,8711	0,8672
108	1	3x2	2x2	0,01	8	27,0	0,0454	0,0435	0,0441	0,709	0,7253	0,7243
109	1	3x3	1x1	0,01	8	62,8	0,0134	0,012	0,0124	0,916	0,9239	0,9217
110	1	3x3	1x2	0,01	8	38,2	0,018	0,0158	0,0165	0,8848	0,8969	0,8973
111	1	3x3	2x1	0,01	8	37,8	0,0194	0,0174	0,0184	0,8815	0,8921	0,8886
112	1	3x3	2x2	0,01	8	31,6	0,034	0,0312	0,0324	0,7917	0,8174	0,8082
113	1	2x2	1x1	0,01	32	25,5	0,0218	0,0193	0,0203	0,8738	0,8886	0,8852
114	1	2x2	1x2	0,01	32	16,1	0,0734	0,0729	0,073	0,5735	0,5958	0,5979
115	1	2x2	2x1	0,01	32	15,9	0,0725	0,0718	0,0723	0,5555	0,5741	0,5691
116	1	2x2	2x2	0,01	32	11,1	0,0883	0,0883	0,0882	0,2193	0,2205	0,2291
117	1	2x3	1x1	0,01	32	29,1	0,0233	0,0206	0,0218	0,8679	0,8858	0,8804
118	1	2x3	1x2	0,01	32	18,5	0,0449	0,0429	0,0438	0,7393	0,7593	0,7546
119	1	2x3	2x1	0,01	32	18,1	0,055	0,0535	0,0544	0,6754	0,6894	0,688
120	1	2x3	2x2	0,01	32	11,9	0,088	0,0879	0,0878	0,2606	0,2566	0,2699
121	1	3x2	1x1	0,01	32	29,2	0,0224	0,0201	0,021	0,867	0,8818	0,8799
122	1	3x2	1x2	0,01	32	20,1	0,0711	0,0703	0,0709	0,5921	0,6066	0,6068
123	1	3x2	2x1	0,01	32	18,5	0,0711	0,0705	0,0707	0,5776	0,5923	0,5857
124	1	3x2	2x2	0,01	32	11,7	0,0882	0,0882	0,0882	0,2879	0,2919	0,299
125	1	3x3	1x1	0,01	32	35,1	0,0225	0,02	0,0212	0,8671	0,8833	0,8774
126	1	3x3	1x2	0,01	32	23,3	0,0826	0,0823	0,0825	0,544	0,5612	0,5547
127	1	3x3	2x1	0,01	32	22,8	0,0659	0,065	0,0651	0,604	0,6257	0,6206
128	1	3x3	2x2	0,01	32	14,7	0,0867	0,0867	0,0866	0,3101	0,3077	0,3157
129	1	2x2	1x1	0,01	128	18,7	0,0697	0,0689	0,0692	0,6025	0,6154	0,6104
130	1	2x2	1x2	0,01	128	10,2	0,0886	0,0886	0,0886	0,2845	0,2804	0,2831
131	1	2x2	2x1	0,01	128	10,0	0,0862	0,0862	0,0861	0,2638	0,2542	0,2713
132	1	2x2	2x2	0,01	128	7,2	0,0896	0,0895	0,0896	0,1575	0,1668	0,159

133	1	2x3	1x1	0,01	128	22,4	0,0837	0,0834	0,0836	0,5562	0,5786	0,5639
134	1	2x3	1x2	0,01	128	10,6	0,0861	0,0859	0,086	0,3942	0,4021	0,4028
135	1	2x3	2x1	0,01	128	10,6	0,0866	0,0865	0,0866	0,386	0,3954	0,3921
136	1	2x3	2x2	0,01	128	8,0	0,09	0,09	0,09	0,1125	0,1172	0,1173
137	1	3x2	1x1	0,01	128	22,3	0,0831	0,0828	0,0829	0,4125	0,421	0,4129
138	1	3x2	1x2	0,01	128	10,7	0,0851	0,085	0,085	0,3661	0,3643	0,3768
139	1	3x2	2x1	0,01	128	10,6	0,089	0,089	0,089	0,2853	0,2872	0,292
140	1	3x2	2x2	0,01	128	8,2	0,0904	0,0904	0,0904	0,0997	0,0945	0,1023
141	1	3x3	1x1	0,01	128	27,3	0,0749	0,0742	0,0745	0,5574	0,578	0,5765
142	1	3x3	1x2	0,01	128	12,2	0,0863	0,0861	0,0861	0,391	0,4051	0,3968
143	1	3x3	2x1	0,01	128	12,2	0,0879	0,0878	0,0878	0,331	0,3464	0,3382
144	1	3x3	2x2	0,01	128	8,4	0,089	0,089	0,0889	0,2086	0,2101	0,2124
145	2	2x2	1x1	1	8	86,8	0,0028	0,0035	0,0041	0,9828	0,9774	0,9741
146	2	2x2	1x2	1	8	51,4	0,1796	0,1794	0,1798	0,102	0,103	0,101
147	2	2x2	2x1	1	8	41,1	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
148	2	2x2	2x2	1	8	31,5	0,0117	0,0108	0,0111	0,9229	0,9278	0,9257
149	2	2x3	1x1	1	8	96,2	0,003	0,0032	0,0037	0,9819	0,9805	0,9767
150	2	2x3	1x2	1	8	48,5	0,1806	0,1803	0,1804	0,0972	0,0983	0,0982
151	2	2x3	2x1	1	8	47,3	0,1803	0,1802	0,1804	0,0986	0,0991	0,098
152	2	2x3	2x2	1	8	33,8	0,1803	0,1802	0,1804	0,0986	0,0991	0,098
153	2	3x2	1x1	1	8	94,7	0,0028	0,0034	0,0034	0,9827	0,9797	0,9789
154	2	3x2	1x2	1	8	48,2	0,1793	0,1782	0,1794	0,1035	0,109	0,1028
155	2	3x2	2x1	1	8	47,7	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
156	2	3x2	2x2	1	8	34,2	0,0091	0,0088	0,009	0,9395	0,9413	0,9408
157	2	3x3	1x1	1	8	118,6	0,0028	0,0033	0,0034	0,9827	0,9794	0,9786
158	2	3x3	1x2	1	8	62,5	0,1802	0,1807	0,1808	0,099	0,0967	0,0958
159	2	3x3	2x1	1	8	62,0	0,1796	0,1794	0,1798	0,102	0,103	0,101
160	2	3x3	2x2	1	8	39,9	0,1803	0,1802	0,1804	0,0986	0,0991	0,098
161	2	2x2	1x1	1	32	45,2	0,0035	0,0039	0,0041	0,9777	0,9747	0,9733
162	2	2x2	1x2	1	32	23,0	0,1644	0,1645	0,1639	0,1779	0,1774	0,1802
163	2	2x2	2x1	1	32	25,3	0,1806	0,1803	0,1804	0,0972	0,0983	0,0982
164	2	2x2	2x2	1	32	14,5	0,1435	0,1425	0,1441	0,2814	0,2867	0,2785
165	2	2x3	1x1	1	32	57,6	0,003	0,0035	0,0035	0,9812	0,9762	0,9774
166	2	2x3	1x2	1	32	29,7	0,1793	0,1782	0,1794	0,1035	0,109	0,1028
167	2	2x3	2x1	1	32	26,9	0,1806	0,1803	0,1804	0,0972	0,0983	0,0982
168	2	2x3	2x2	1	32	16,8	0,182	0,1817	0,1822	0,0901	0,0915	0,0892
169	2	3x2	1x1	1	32	57,0	0,0026	0,0031	0,003	0,9841	0,979	0,9811
170	2	3x2	1x2	1	32	30,2	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
171	2	3x2	2x1	1	32	27,1	0,159	0,1601	0,1591	0,2049	0,1994	0,2046
172	2	3x2	2x2	1	32	15,9	0,1598	0,1612	0,1598	0,2008	0,1937	0,2005
173	2	3x3	1x1	1	32	71,5	0,0021	0,0026	0,0022	0,9874	0,9842	0,9854
174	2	3x3	1x2	1	32	34,9	0,166	0,1674	0,1662	0,17	0,1629	0,1689
175	2	3x3	2x1	1	32	32,0	0,1806	0,1803	0,1804	0,0972	0,0983	0,0982
176	2	3x3	2x2	1	32	19,3	0,1803	0,1802	0,1804	0,0986	0,0991	0,098
177	2	2x2	1x1	1	128	39,1	0,0075	0,0069	0,007	0,9518	0,9564	0,9546
178	2	2x2	1x2	1	128	15,6	0,0113	0,0102	0,0106	0,927	0,9337	0,9316
179	2	2x2	2x1	1	128	15,7	0,0142	0,0131	0,0133	0,9049	0,9148	0,9104
180	2	2x2	2x2	1	128	9,6	0,0595	0,0572	0,06	0,562	0,5798	0,559

181	2	2x3	1x1	1	128	48,7	0,0162	0,0165	0,0162	0,8828	0,8813	0,8816
182	2	2x3	1x2	1	128	17,7	0,0095	0,0086	0,0091	0,9381	0,9428	0,941
183	2	2x3	2x1	1	128	17,7	0,01	0,0091	0,0091	0,9335	0,9423	0,9405
184	2	2x3	2x2	1	128	11,8	0,1107	0,109	0,1083	0,4405	0,4487	0,4531
185	2	3x2	1x1	1	128	48,9	0,0047	0,0045	0,0046	0,9705	0,9722	0,971
186	2	3x2	1x2	1	128	17,9	0,0071	0,0066	0,0068	0,955	0,958	0,9566
187	2	3x2	2x1	1	128	17,9	0,01	0,0089	0,009	0,9343	0,9428	0,9411
188	2	3x2	2x2	1	128	11,9	0,1377	0,1389	0,1378	0,2964	0,2909	0,2966
189	2	3x3	1x1	1	128	64,9	0,0165	0,0168	0,0164	0,882	0,8793	0,8821
190	2	3x3	1x2	1	128	22,2	0,0076	0,007	0,007	0,952	0,9541	0,9544
191	2	3x3	2x1	1	128	22,0	0,0058	0,0056	0,0056	0,9629	0,9647	0,9646
192	2	3x3	2x2	1	128	12,1	0,1382	0,1382	0,138	0,3026	0,303	0,3036
193	2	2x2	1x1	0,1	8	83,8	0,0048	0,0047	0,0049	0,9703	0,9706	0,969
194	2	2x2	1x2	0,1	8	58,7	0,1091	0,1059	0,1072	0,4535	0,4693	0,4625
195	2	2x2	2x1	0,1	8	58,7	0,1328	0,1324	0,1323	0,3359	0,3377	0,3385
196	2	2x2	2x2	0,1	8	48,5	0,0482	0,0438	0,0478	0,7291	0,7547	0,7318
197	2	2x3	1x1	0,1	8	115,1	0,0036	0,0036	0,0037	0,9769	0,9774	0,9763
198	2	2x3	1x2	0,1	8	66,3	0,1636	0,1621	0,163	0,1819	0,1896	0,185
199	2	2x3	2x1	0,1	8	67,1	0,1412	0,142	0,1397	0,2929	0,2884	0,3007
200	2	2x3	2x2	0,1	8	52,2	0,0897	0,0867	0,0883	0,5148	0,5285	0,5229
201	2	3x2	1x1	0,1	8	116,1	0,0038	0,0039	0,004	0,976	0,9754	0,9745
202	2	3x2	1x2	0,1	8	66,8	0,1763	0,1766	0,1755	0,1183	0,1169	0,1225
203	2	3x2	2x1	0,1	8	72,9	0,1624	0,163	0,1626	0,1879	0,1848	0,1869
204	2	3x2	2x2	0,1	8	52,4	0,0286	0,0255	0,0274	0,8018	0,8256	0,8126
205	2	3x3	1x1	0,1	8	127,3	0,0026	0,0029	0,0027	0,9841	0,9813	0,9816
206	2	3x3	1x2	0,1	8	67,4	0,1709	0,1699	0,1707	0,1451	0,15	0,146
207	2	3x3	2x1	0,1	8	70,7	0,161	0,1599	0,1622	0,1946	0,2	0,1886
208	2	3x3	2x2	0,1	8	46,0	0,1269	0,1259	0,1256	0,3562	0,3592	0,3626
209	2	2x2	1x1	0,1	32	47,9	0,0117	0,0108	0,0111	0,9237	0,928	0,9274
210	2	2x2	1x2	0,1	32	29,6	0,0225	0,0202	0,0214	0,8489	0,865	0,8543
211	2	2x2	2x1	0,1	32	29,3	0,0233	0,0211	0,0215	0,843	0,8588	0,8569
212	2	2x2	2x2	0,1	32	20,0	0,0502	0,0479	0,0499	0,6392	0,6653	0,6419
213	2	2x3	1x1	0,1	32	59,6	0,0122	0,0112	0,0114	0,92	0,9258	0,9252
214	2	2x3	1x2	0,1	32	33,9	0,0161	0,0145	0,0157	0,8963	0,9079	0,8983
215	2	2x3	2x1	0,1	32	33,1	0,0267	0,0248	0,0246	0,8439	0,8557	0,8567
216	2	2x3	2x2	0,1	32	21,7	0,0346	0,0315	0,0327	0,76	0,778	0,7749
217	2	3x2	1x1	0,1	32	60,9	0,0089	0,0082	0,0086	0,9437	0,9482	0,9438
218	2	3x2	1x2	0,1	32	33,5	0,0277	0,0253	0,0269	0,808	0,8233	0,8132
219	2	3x2	2x1	0,1	32	34,1	0,034	0,0297	0,0332	0,8184	0,8419	0,8229
220	2	3x2	2x2	0,1	32	21,6	0,034	0,0317	0,0331	0,7593	0,7763	0,7665
221	2	3x3	1x1	0,1	32	75,3	0,0061	0,0056	0,0056	0,9615	0,9653	0,9637
222	2	3x3	1x2	0,1	32	38,8	0,0214	0,0193	0,0195	0,8738	0,8871	0,8848
223	2	3x3	2x1	0,1	32	39,1	0,0196	0,0165	0,0188	0,8673	0,8871	0,8733
224	2	3x3	2x2	0,1	32	24,7	0,0238	0,0216	0,0225	0,8371	0,8505	0,8483
225	2	2x2	1x1	0,1	128	40,1	0,0142	0,0126	0,0132	0,9069	0,9167	0,9128
226	2	2x2	1x2	0,1	128	16,8	0,0486	0,0466	0,0477	0,6889	0,712	0,7066
227	2	2x2	2x1	0,1	128	17,3	0,0852	0,0851	0,0852	0,3067	0,3174	0,31
228	2	2x2	2x2	0,1	128	11,7	0,089	0,0889	0,089	0,2046	0,2104	0,1978

229	2	2x3	1x1	0,1	128	51,2	0,014	0,0123	0,0127	0,9073	0,918	0,9172
230	2	2x3	1x2	0,1	128	18,9	0,0235	0,0213	0,0223	0,8449	0,8587	0,8532
231	2	2x3	2x1	0,1	128	18,9	0,0549	0,0534	0,0539	0,6338	0,661	0,6384
232	2	2x3	2x2	0,1	128	12,6	0,0896	0,0896	0,0896	0,1909	0,1974	0,196
233	2	3x2	1x1	0,1	128	51,0	0,0153	0,0139	0,0141	0,8987	0,9083	0,9057
234	2	3x2	1x2	0,1	128	19,7	0,0342	0,0317	0,0333	0,7889	0,8112	0,7971
235	2	3x2	2x1	0,1	128	19,1	0,0509	0,0489	0,0494	0,6505	0,6819	0,663
236	2	3x2	2x2	0,1	128	13,3	0,0893	0,0894	0,0893	0,248	0,2427	0,255
237	2	3x3	1x1	0,1	128	67,0	0,0154	0,0136	0,0141	0,8976	0,9072	0,9078
238	2	3x3	1x2	0,1	128	24,1	0,0207	0,0188	0,0193	0,8622	0,8756	0,8709
239	2	3x3	2x1	0,1	128	24,2	0,0471	0,0447	0,0457	0,6981	0,7283	0,7073
240	2	3x3	2x2	0,1	128	14,2	0,0891	0,089	0,0891	0,2031	0,2062	0,2101
241	2	2x2	1x1	0,01	8	98,2	0,0139	0,0125	0,013	0,9099	0,9172	0,9142
242	2	2x2	1x2	0,01	8	74,8	0,0289	0,0262	0,0279	0,8109	0,8285	0,8156
243	2	2x2	2x1	0,01	8	71,0	0,0296	0,0274	0,0284	0,7972	0,8104	0,8058
244	2	2x2	2x2	0,01	8	60,1	0,0891	0,0891	0,0891	0,1829	0,1879	0,1801
245	2	2x3	1x1	0,01	8	123,3	0,013	0,0119	0,0121	0,9141	0,9211	0,9202
246	2	2x3	1x2	0,01	8	79,2	0,0164	0,0148	0,0155	0,8942	0,9038	0,8981
247	2	2x3	2x1	0,01	8	74,6	0,0246	0,0224	0,0233	0,8327	0,8488	0,842
248	2	2x3	2x2	0,01	8	59,9	0,0896	0,0896	0,0896	0,2172	0,2136	0,2265
249	2	3x2	1x1	0,01	8	124,2	0,0122	0,011	0,0115	0,9209	0,9284	0,9234
250	2	3x2	1x2	0,01	8	78,8	0,0195	0,0177	0,0185	0,8734	0,885	0,8817
251	2	3x2	2x1	0,01	8	74,5	0,0211	0,0189	0,0206	0,8601	0,8727	0,8612
252	2	3x2	2x2	0,01	8	61,5	0,0888	0,0888	0,0888	0,2479	0,2496	0,2494
253	2	3x3	1x1	0,01	8	148,2	0,0114	0,0104	0,0106	0,9272	0,9326	0,9311
254	2	3x3	1x2	0,01	8	88,3	0,0155	0,0141	0,0147	0,8994	0,9094	0,9035
255	2	3x3	2x1	0,01	8	90,6	0,0155	0,0136	0,015	0,901	0,9125	0,9018
256	2	3x3	2x2	0,01	8	73,7	0,0792	0,079	0,0789	0,398	0,4085	0,411
257	2	2x2	1x1	0,01	32	49,9	0,0198	0,0174	0,0183	0,8769	0,8914	0,8887
258	2	2x2	1x2	0,01	32	31,8	0,0898	0,0898	0,0898	0,1828	0,1856	0,1879
259	2	2x2	2x1	0,01	32	32,5	0,0866	0,0865	0,0865	0,3086	0,3147	0,3162
260	2	2x2	2x2	0,01	32	23,4	0,0898	0,0897	0,0898	0,1152	0,1181	0,1168
261	2	2x3	1x1	0,01	32	63,0	0,0168	0,0147	0,0154	0,8929	0,9058	0,9024
262	2	2x3	1x2	0,01	32	36,9	0,0895	0,0895	0,0895	0,1392	0,1321	0,1283
263	2	2x3	2x1	0,01	32	36,6	0,0886	0,0885	0,0886	0,2761	0,2977	0,2711
264	2	2x3	2x2	0,01	32	26,9	0,0896	0,0896	0,0896	0,1437	0,146	0,1461
265	2	3x2	1x1	0,01	32	63,8	0,0183	0,0162	0,0171	0,8817	0,8962	0,8925
266	2	3x2	1x2	0,01	32	37,3	0,0454	0,0437	0,0446	0,7184	0,7349	0,7298
267	2	3x2	2x1	0,01	32	37,3	0,0891	0,089	0,0891	0,286	0,2859	0,2921
268	2	3x2	2x2	0,01	32	25,4	0,0895	0,0895	0,0895	0,1625	0,1542	0,1656
269	2	3x3	1x1	0,01	32	79,9	0,0178	0,0155	0,0165	0,8854	0,8993	0,8941
270	2	3x3	1x2	0,01	32	43,2	0,089	0,089	0,089	0,1663	0,1677	0,173
271	2	3x3	2x1	0,01	32	43,1	0,0779	0,0779	0,0775	0,3795	0,3789	0,3929
272	2	3x3	2x2	0,01	32	28,1	0,0897	0,0897	0,0897	0,1222	0,1215	0,1209
273	2	2x2	1x1	0,01	128	42,4	0,0859	0,0857	0,0859	0,508	0,5286	0,5179
274	2	2x2	1x2	0,01	128	18,6	0,0897	0,0897	0,0897	0,1324	0,1255	0,1291
275	2	2x2	2x1	0,01	128	18,6	0,0901	0,09	0,0901	0,0672	0,067	0,0682
276	2	2x2	2x2	0,01	128	13,4	0,0898	0,0898	0,0898	0,0974	0,095	0,0946

277	2	2x3	1x1	0,01	128	52,0	0,0754	0,0748	0,0752	0,6153	0,635	0,6162
278	2	2x3	1x2	0,01	128	21,3	0,0898	0,0897	0,0898	0,1355	0,1457	0,1392
279	2	2x3	2x1	0,01	128	21,0	0,0901	0,0901	0,0901	0,099	0,096	0,0983
280	2	2x3	2x2	0,01	128	15,2	0,0902	0,0902	0,0902	0,0895	0,0843	0,0929
281	2	3x2	1x1	0,01	128	52,4	0,0464	0,0449	0,0453	0,7335	0,7517	0,7474
282	2	3x2	1x2	0,01	128	21,5	0,09	0,09	0,09	0,0566	0,0549	0,0476
283	2	3x2	2x1	0,01	128	21,4	0,0896	0,0896	0,0896	0,0899	0,0872	0,0856
284	2	3x2	2x2	0,01	128	15,1	0,0901	0,09	0,0901	0,1206	0,1264	0,1213
285	2	3x3	1x1	0,01	128	68,8	0,0875	0,0874	0,0875	0,3212	0,3244	0,3236
286	2	3x3	1x2	0,01	128	26,5	0,0896	0,0896	0,0896	0,1963	0,1901	0,2013
287	2	3x3	2x1	0,01	128	26,4	0,0898	0,0898	0,0898	0,1443	0,1526	0,1413
288	2	3x3	2x2	0,01	128	16,1	0,0903	0,0903	0,0903	0,095	0,0917	0,0959
289	3	2x2	1x1	1	8	114,2	0,0043	0,0045	0,0043	0,9733	0,9721	0,9721
290	3	2x2	1x2	1	8	84,4	0,1802	0,1807	0,1808	0,099	0,0967	0,0958
291	3	2x2	2x1	1	8	83,3	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
292	3	2x2	2x2	1	8	66,4	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
293	3	2x3	1x1	1	8	136,3	0,0045	0,0042	0,0045	0,9754	0,9767	0,9751
294	3	2x3	1x2	1	8	93,2	0,18	0,1808	0,1798	0,0998	0,0961	0,1009
295	3	2x3	2x1	1	8	87,1	0,1793	0,1782	0,1794	0,1035	0,109	0,1028
296	3	2x3	2x2	1	8	66,5	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
297	3	3x2	1x1	1	8	137,7	0,0036	0,0036	0,0036	0,9777	0,9778	0,9787
298	3	3x2	1x2	1	8	93,9	0,1801	0,1802	0,1794	0,0994	0,099	0,1032
299	3	3x2	2x1	1	8	83,5	0,182	0,1817	0,1822	0,0901	0,0915	0,0892
300	3	3x2	2x2	1	8	68,7	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
301	3	3x3	1x1	1	8	169,9	0,0068	0,0065	0,0069	0,9593	0,9619	0,9589
302	3	3x3	1x2	1	8	97,7	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
303	3	3x3	2x1	1	8	92,3	0,1803	0,1802	0,1804	0,0986	0,0991	0,098
304	3	3x3	2x2	1	8	72,6	0,1806	0,1798	0,1805	0,0968	0,1009	0,0974
305	3	2x2	1x1	1	32	61,3	0,0037	0,0038	0,004	0,976	0,9757	0,9731
306	3	2x2	1x2	1	32	35,9	0,1801	0,1802	0,1794	0,0994	0,099	0,1032
307	3	2x2	2x1	1	32	38,6	0,1489	0,1486	0,15	0,2551	0,2568	0,2499
308	3	2x2	2x2	1	32	27,0	0,158	0,1579	0,1582	0,2078	0,2084	0,2072
309	3	2x3	1x1	1	32	70,4	0,0022	0,0024	0,0026	0,986	0,9839	0,9812
310	3	2x3	1x2	1	32	39,0	0,1796	0,18	0,1799	0,102	0,1	0,1002
311	3	2x3	2x1	1	32	42,1	0,1806	0,1803	0,1804	0,0972	0,0983	0,0982
312	3	2x3	2x2	1	32	28,4	0,1576	0,1591	0,1575	0,2084	0,2009	0,209
313	3	3x2	1x1	1	32	71,6	0,0027	0,003	0,0029	0,9831	0,9804	0,9822
314	3	3x2	1x2	1	32	40,1	0,1792	0,1781	0,1793	0,1042	0,1097	0,1037
315	3	3x2	2x1	1	32	42,2	0,1598	0,161	0,1596	0,2008	0,1952	0,2018
316	3	3x2	2x2	1	32	29,2	0,1485	0,1481	0,1476	0,2559	0,2577	0,2608
317	3	3x3	1x1	1	32	94,6	0,0031	0,0032	0,0032	0,9797	0,9799	0,979
318	3	3x3	1x2	1	32	48,8	0,1803	0,1802	0,1804	0,0986	0,0991	0,098
319	3	3x3	2x1	1	32	48,1	0,18	0,1808	0,1798	0,0998	0,0961	0,1009
320	3	3x3	2x2	1	32	30,2	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
321	3	2x2	1x1	1	128	48,9	0,0098	0,0088	0,0091	0,9378	0,943	0,9407
322	3	2x2	1x2	1	128	20,4	0,0123	0,0112	0,0111	0,9193	0,9279	0,9282
323	3	2x2	2x1	1	128	20,8	0,0147	0,0134	0,0139	0,9033	0,9128	0,9083
324	3	2x2	2x2	1	128	15,5	0,077	0,0759	0,0754	0,5428	0,547	0,5522

325	3	2x3	1x1	1	128	60,3	0,0057	0,0054	0,0055	0,963	0,9641	0,9634
326	3	2x3	1x2	1	128	22,7	0,0086	0,0074	0,0076	0,9444	0,952	0,9504
327	3	2x3	2x1	1	128	22,6	0,0108	0,0097	0,0105	0,93	0,9358	0,9312
328	3	2x3	2x2	1	128	17,0	0,0832	0,0831	0,083	0,2839	0,2869	0,2832
329	3	3x2	1x1	1	128	60,9	0,0041	0,0039	0,004	0,9746	0,9748	0,9731
330	3	3x2	1x2	1	128	23,6	0,0091	0,0082	0,0083	0,9414	0,9465	0,9475
331	3	3x2	2x1	1	128	23,0	0,0133	0,012	0,0128	0,9127	0,9223	0,9172
332	3	3x2	2x2	1	128	18,6	0,0662	0,0656	0,0661	0,4612	0,462	0,4618
333	3	3x3	1x1	1	128	79,7	0,0029	0,0029	0,0029	0,9811	0,9803	0,9804
334	3	3x3	1x2	1	128	28,3	0,0067	0,0062	0,006	0,9574	0,9601	0,9611
335	3	3x3	2x1	1	128	26,4	0,0103	0,0095	0,0095	0,9311	0,9384	0,936
336	3	3x3	2x2	1	128	17,5	0,0758	0,0753	0,0756	0,4493	0,456	0,4544
337	3	2x2	1x1	0,1	8	150,4	0,0044	0,0044	0,0045	0,9724	0,9713	0,9705
338	3	2x2	1x2	0,1	8	102,7	0,1772	0,1772	0,1775	0,1137	0,1139	0,1124
339	3	2x2	2x1	0,1	8	100,5	0,145	0,1444	0,1451	0,2747	0,2776	0,2742
340	3	2x2	2x2	0,1	8	80,7	0,0415	0,0381	0,0402	0,7296	0,7521	0,737
341	3	2x3	1x1	0,1	8	144,7	0,0028	0,0031	0,0031	0,9825	0,9794	0,98
342	3	2x3	1x2	0,1	8	105,0	0,16	0,1594	0,161	0,2	0,2026	0,1947
343	3	2x3	2x1	0,1	8	103,9	0,1558	0,157	0,1554	0,221	0,2148	0,223
344	3	2x3	2x2	0,1	8	105,2	0,0485	0,0462	0,0474	0,6471	0,6702	0,6558
345	3	3x2	1x1	0,1	8	144,4	0,004	0,0038	0,0038	0,9738	0,9764	0,9733
346	3	3x2	1x2	0,1	8	110,6	0,1593	0,1609	0,1599	0,203	0,1951	0,1998
347	3	3x2	2x1	0,1	8	100,1	0,1542	0,1558	0,1547	0,2273	0,2191	0,2246
348	3	3x2	2x2	0,1	8	79,3	0,0444	0,0411	0,0439	0,6872	0,7087	0,6906
349	3	3x3	1x1	0,1	8	172,8	0,003	0,0029	0,0029	0,9815	0,9812	0,9807
350	3	3x3	1x2	0,1	8	109,2	0,1803	0,1802	0,1804	0,0986	0,0991	0,0981
351	3	3x3	2x1	0,1	8	108,8	0,1551	0,1553	0,1541	0,2241	0,2233	0,229
352	3	3x3	2x2	0,1	8	85,3	0,0427	0,0403	0,0419	0,6974	0,719	0,701
353	3	2x2	1x1	0,1	32	67,0	0,0124	0,0115	0,0118	0,9195	0,9233	0,9228
354	3	2x2	1x2	0,1	32	40,6	0,0285	0,0259	0,027	0,8097	0,827	0,8241
355	3	2x2	2x1	0,1	32	40,4	0,0276	0,0248	0,0272	0,8098	0,8314	0,8137
356	3	2x2	2x2	0,1	32	30,1	0,07	0,0684	0,0695	0,4769	0,4949	0,4796
357	3	2x3	1x1	0,1	32	73,7	0,0124	0,0112	0,0115	0,918	0,9249	0,9239
358	3	2x3	1x2	0,1	32	43,5	0,0526	0,0503	0,0513	0,7099	0,7232	0,7174
359	3	2x3	2x1	0,1	32	46,9	0,0312	0,0275	0,0276	0,8352	0,8552	0,8549
360	3	2x3	2x2	0,1	32	32,8	0,0559	0,0546	0,0554	0,5921	0,6037	0,5978
361	3	3x2	1x1	0,1	32	75,2	0,0068	0,0061	0,0063	0,9573	0,9636	0,9589
362	3	3x2	1x2	0,1	32	44,3	0,029	0,0256	0,0273	0,7968	0,8244	0,8093
363	3	3x2	2x1	0,1	32	47,0	0,0234	0,0214	0,0224	0,8387	0,8516	0,8454
364	3	3x2	2x2	0,1	32	32,9	0,0515	0,0489	0,0501	0,6481	0,6805	0,6655
365	3	3x3	1x1	0,1	32	99,9	0,0057	0,0052	0,0051	0,9641	0,9682	0,9674
366	3	3x3	1x2	0,1	32	52,1	0,0368	0,0341	0,036	0,7405	0,7596	0,7416
367	3	3x3	2x1	0,1	32	51,5	0,1215	0,12	0,1217	0,3833	0,3923	0,3824
368	3	3x3	2x2	0,1	32	33,9	0,0496	0,0481	0,0477	0,6461	0,662	0,6625
369	3	2x2	1x1	0,1	128	51,3	0,0146	0,0131	0,0136	0,9032	0,9115	0,9093
370	3	2x2	1x2	0,1	128	22,8	0,0264	0,0235	0,0248	0,8255	0,8472	0,8366
371	3	2x2	2x1	0,1	128	24,4	0,0759	0,0756	0,0756	0,4052	0,4087	0,4053
372	3	2x2	2x2	0,1	128	18,5	0,0894	0,0892	0,0894	0,1766	0,1926	0,1793

373	3	2x3	1x1	0,1	128	63,9	0,016	0,0137	0,0146	0,893	0,9078	0,9019
374	3	2x3	1x2	0,1	128	25,2	0,0832	0,083	0,083	0,3251	0,3239	0,3301
375	3	2x3	2x1	0,1	128	25,2	0,0354	0,0337	0,0344	0,7704	0,7871	0,7777
376	3	2x3	2x2	0,1	128	19,5	0,0896	0,0896	0,0895	0,1454	0,1454	0,1489
377	3	3x2	1x1	0,1	128	62,7	0,0186	0,0167	0,0173	0,8754	0,8876	0,8845
378	3	3x2	1x2	0,1	128	25,3	0,0456	0,0434	0,0441	0,7114	0,7371	0,7317
379	3	3x2	2x1	0,1	128	25,0	0,0496	0,0474	0,0481	0,6456	0,674	0,6569
380	3	3x2	2x2	0,1	128	19,9	0,085	0,0849	0,0848	0,2965	0,297	0,3039
381	3	3x3	1x1	0,1	128	82,2	0,0126	0,0111	0,0118	0,9192	0,9273	0,9252
382	3	3x3	1x2	0,1	128	28,7	0,0379	0,0351	0,0366	0,7425	0,7695	0,7577
383	3	3x3	2x1	0,1	128	28,0	0,0664	0,0656	0,0657	0,4786	0,4886	0,4837
384	3	3x3	2x2	0,1	128	19,8	0,0888	0,0888	0,0887	0,2118	0,2008	0,2193
385	3	2x2	1x1	0,01	8	165,5	0,0136	0,0121	0,0125	0,9099	0,9178	0,9174
386	3	2x2	1x2	0,01	8	118,5	0,0221	0,0188	0,0205	0,8527	0,877	0,8648
387	3	2x2	2x1	0,01	8	118,7	0,0467	0,0449	0,046	0,6685	0,6888	0,6733
388	3	2x2	2x2	0,01	8	95,0	0,0883	0,0883	0,0884	0,202	0,2079	0,1983
389	3	2x3	1x1	0,01	8	184,9	0,0127	0,0114	0,0115	0,9166	0,9224	0,9244
390	3	2x3	1x2	0,01	8	130,6	0,0286	0,0255	0,0273	0,8092	0,8333	0,8147
391	3	2x3	2x1	0,01	8	130,4	0,0253	0,0229	0,0242	0,8266	0,8438	0,8317
392	3	2x3	2x2	0,01	8	102,5	0,0892	0,0892	0,0892	0,2227	0,2133	0,2261
393	3	3x2	1x1	0,01	8	190,4	0,0121	0,011	0,0113	0,9228	0,9313	0,9266
394	3	3x2	1x2	0,01	8	132,3	0,0235	0,0206	0,0219	0,8411	0,8636	0,8548
395	3	3x2	2x1	0,01	8	125,6	0,0803	0,0801	0,0803	0,3275	0,3235	0,3253
396	3	3x2	2x2	0,01	8	98,3	0,089	0,0889	0,089	0,2456	0,2626	0,2564
397	3	3x3	1x1	0,01	8	235,7	0,0163	0,0144	0,0151	0,8914	0,9055	0,9028
398	3	3x3	1x2	0,01	8	136,6	0,0265	0,0229	0,0256	0,8164	0,8458	0,8256
399	3	3x3	2x1	0,01	8	136,9	0,0297	0,0271	0,0286	0,804	0,8274	0,8159
400	3	3x3	2x2	0,01	8	104,7	0,0872	0,0873	0,0871	0,3437	0,3383	0,3488
401	3	2x2	1x1	0,01	32	72,8	0,0174	0,0153	0,0161	0,8873	0,8993	0,8987
402	3	2x2	1x2	0,01	32	46,4	0,0888	0,0887	0,0887	0,2091	0,2214	0,2139
403	3	2x2	2x1	0,01	32	45,4	0,0879	0,0879	0,0879	0,2724	0,2755	0,2721
404	3	2x2	2x2	0,01	32	35,9	0,0902	0,0902	0,0902	0,0741	0,0725	0,0773
405	3	2x3	1x1	0,01	32	84,7	0,0173	0,0151	0,016	0,8877	0,902	0,9001
406	3	2x3	1x2	0,01	32	48,7	0,0858	0,0857	0,0858	0,3943	0,4067	0,3929
407	3	2x3	2x1	0,01	32	48,1	0,088	0,088	0,088	0,2789	0,2805	0,2788
408	3	2x3	2x2	0,01	32	37,0	0,0899	0,0899	0,0899	0,1181	0,1213	0,1215
409	3	3x2	1x1	0,01	32	85,8	0,0177	0,0158	0,0164	0,8912	0,9035	0,9017
410	3	3x2	1x2	0,01	32	56,6	0,0886	0,0885	0,0886	0,2224	0,2304	0,2189
411	3	3x2	2x1	0,01	32	55,0	0,0883	0,0883	0,0883	0,2051	0,2089	0,2059
412	3	3x2	2x2	0,01	32	42,6	0,0899	0,0899	0,09	0,1075	0,1116	0,0947
413	3	3x3	1x1	0,01	32	105,8	0,0193	0,017	0,0181	0,8746	0,8862	0,8874
414	3	3x3	1x2	0,01	32	53,2	0,077	0,0762	0,077	0,4165	0,4274	0,4166
415	3	3x3	2x1	0,01	32	52,9	0,0865	0,0864	0,0864	0,265	0,2699	0,2693
416	3	3x3	2x2	0,01	32	38,5	0,0898	0,0898	0,0898	0,1592	0,1528	0,1636
417	3	2x2	1x1	0,01	128	52,8	0,0886	0,0886	0,0886	0,2511	0,2615	0,2513
418	3	2x2	1x2	0,01	128	24,6	0,0894	0,0894	0,0894	0,1659	0,17	0,1688
419	3	2x2	2x1	0,01	128	25,9	0,0898	0,0898	0,0897	0,0875	0,0927	0,086
420	3	2x2	2x2	0,01	128	20,1	0,0912	0,0911	0,0913	0,1018	0,1077	0,1009

421	3	2x3	1x1	0,01	128	65,0	0,083	0,0826	0,0829	0,4848	0,5118	0,4854
422	3	2x3	1x2	0,01	128	27,8	0,0893	0,0892	0,0893	0,2097	0,2255	0,2111
423	3	2x3	2x1	0,01	128	27,4	0,0899	0,0899	0,0899	0,1327	0,1375	0,135
424	3	2x3	2x2	0,01	128	21,2	0,0903	0,0903	0,0903	0,1317	0,1325	0,1321
425	3	3x2	1x1	0,01	128	66,0	0,0778	0,0773	0,0777	0,4905	0,5122	0,4985
426	3	3x2	1x2	0,01	128	28,6	0,0886	0,0884	0,0885	0,2079	0,2217	0,2116
427	3	3x2	2x1	0,01	128	28,5	0,09	0,09	0,09	0,1495	0,159	0,1504
428	3	3x2	2x2	0,01	128	21,5	0,0902	0,0902	0,0901	0,1052	0,1108	0,1042
429	3	3x3	1x1	0,01	128	83,5	0,0617	0,0603	0,0609	0,6041	0,6264	0,627
430	3	3x3	1x2	0,01	128	30,9	0,09	0,0899	0,0899	0,0915	0,089	0,0934
431	3	3x3	2x1	0,01	128	30,8	0,0897	0,0897	0,0898	0,1372	0,1327	0,1246
432	3	3x3	2x2	0,01	128	21,6	0,0904	0,0905	0,0905	0,0747	0,0695	0,0723
433	4	2x2	1x1	1	8	196,3	0,0037	0,0038	0,0042	0,9771	0,9772	0,9748
434	4	2x2	1x2	1	8	155,0	0,1793	0,1782	0,1794	0,1035	0,109	0,1028
435	4	2x2	2x1	1	8	143,9	0,0899	0,09	0,0899	0,1143	0,1068	0,1141
436	4	2x2	2x2	1	8	109,5	0,0817	0,0798	0,0812	0,5252	0,5389	0,5274
437	4	2x3	1x1	1	8	227,8	0,0082	0,0082	0,0082	0,9524	0,9529	0,9524
438	4	2x3	1x2	1	8	123,1	0,0118	0,0113	0,0104	0,9334	0,9358	0,9412
439	4	2x3	2x1	1	8	113,7	0,1806	0,1798	0,1805	0,0968	0,1009	0,0974
440	4	2x3	2x2	1	8	120,7	0,0493	0,0462	0,0485	0,6496	0,6772	0,6537
441	4	3x2	1x1	1	8	232,5	0,1773	0,1787	0,1773	0,1136	0,1064	0,1135
442	4	3x2	1x2	1	8	114,3	0,0347	0,031	0,0334	0,7654	0,7919	0,7753
443	4	3x2	2x1	1	8	141,7	0,1409	0,1403	0,1384	0,2817	0,2842	0,2947
444	4	3x2	2x2	1	8	122,5	0,09	0,09	0,09	0,1136	0,1064	0,1135
445	4	3x3	1x1	1	8	310,4	0,09	0,09	0,09	0,1136	0,1064	0,1135
446	4	3x3	1x2	1	8	170,0	0,18	0,1808	0,1798	0,0998	0,0961	0,1009
447	4	3x3	2x1	1	8	172,8	0,1793	0,1782	0,1794	0,1035	0,109	0,1028
448	4	3x3	2x2	1	8	130,5	0,1793	0,1782	0,1794	0,1035	0,109	0,1028
449	4	2x2	1x1	1	32	115,7	0,0027	0,003	0,0032	0,9826	0,9807	0,9784
450	4	2x2	1x2	1	32	60,1	0,0057	0,0055	0,0053	0,9633	0,9642	0,9632
451	4	2x2	2x1	1	32	54,6	0,1773	0,1787	0,1773	0,1135	0,1064	0,1135
452	4	2x2	2x2	1	32	43,5	0,0226	0,0208	0,0224	0,8473	0,8582	0,8486
453	4	2x3	1x1	1	32	144,5	0,0032	0,0036	0,0038	0,9794	0,9768	0,9748
454	4	2x3	1x2	1	32	62,9	0,0061	0,0057	0,0055	0,9603	0,9631	0,9629
455	4	2x3	2x1	1	32	58,9	0,0114	0,0104	0,0107	0,9241	0,929	0,9296
456	4	2x3	2x2	1	32	46,7	0,0174	0,0159	0,0165	0,8849	0,8963	0,8943
457	4	3x2	1x1	1	32	145,4	0,0032	0,0034	0,0037	0,9801	0,9786	0,9763
458	4	3x2	1x2	1	32	55,8	0,0054	0,0048	0,0051	0,9652	0,9695	0,9656
459	4	3x2	2x1	1	32	65,3	0,182	0,1817	0,1822	0,0901	0,0915	0,0892
460	4	3x2	2x2	1	32	47,9	0,0177	0,0168	0,0175	0,8868	0,8946	0,8867
461	4	3x3	1x1	1	32	178,6	0,0028	0,0032	0,003	0,9824	0,9803	0,9806
462	4	3x3	1x2	1	32	68,4	0,0542	0,0523	0,0544	0,5994	0,6113	0,5974
463	4	3x3	2x1	1	32	65,2	0,0065	0,0059	0,0061	0,9577	0,9615	0,9608
464	4	3x3	2x2	1	32	50,7	0,0156	0,0143	0,0145	0,9035	0,9117	0,9101
465	4	2x2	1x1	1	128	93,8	0,0058	0,0053	0,0053	0,9628	0,9665	0,9654
466	4	2x2	1x2	1	128	32,9	0,0151	0,0141	0,0139	0,8988	0,9045	0,9067
467	4	2x2	2x1	1	128	31,8	0,0185	0,0169	0,0181	0,8764	0,8875	0,8792
468	4	2x2	2x2	1	128	24,2	0,0504	0,0493	0,0501	0,6325	0,6452	0,6345

469	4	2x3	1x1	1	128	118,1	0,005	0,005	0,0049	0,9672	0,9664	0,968
470	4	2x3	1x2	1	128	35,0	0,1619	0,1601	0,1616	0,1896	0,1984	0,1908
471	4	2x3	2x1	1	128	35,6	0,1546	0,1556	0,1545	0,2104	0,2031	0,2104
472	4	2x3	2x2	1	128	25,3	0,0411	0,039	0,0407	0,7037	0,7156	0,7068
473	4	3x2	1x1	1	128	118,8	0,0046	0,0044	0,0044	0,9713	0,9729	0,9711
474	4	3x2	1x2	1	128	35,2	0,1648	0,1634	0,1659	0,1758	0,1826	0,1703
475	4	3x2	2x1	1	128	35,6	0,0374	0,0368	0,0375	0,7346	0,7363	0,7344
476	4	3x2	2x2	1	128	25,3	0,0408	0,0381	0,0396	0,7037	0,7258	0,7143
477	4	3x3	1x1	1	128	155,0	0,0032	0,0029	0,0029	0,9796	0,9816	0,9818
478	4	3x3	1x2	1	128	41,9	0,1598	0,1592	0,1611	0,1992	0,2023	0,1928
479	4	3x3	2x1	1	128	41,3	0,1806	0,1798	0,1805	0,0968	0,1009	0,0974
480	4	3x3	2x2	1	128	27,2	0,0561	0,0543	0,0557	0,6094	0,6288	0,618
481	4	2x2	1x1	0,1	8	211,4	0,0033	0,0038	0,0038	0,9787	0,9761	0,9743
482	4	2x2	1x2	0,1	8	135,0	0,0096	0,0086	0,0092	0,9373	0,9421	0,9406
483	4	2x2	2x1	0,1	8	151,7	0,0129	0,0115	0,0123	0,9139	0,9245	0,9182
484	4	2x2	2x2	0,1	8	127,4	0,0485	0,0468	0,0474	0,6318	0,6508	0,6468
485	4	2x3	1x1	0,1	8	241,2	0,0032	0,0031	0,0032	0,9797	0,98	0,9784
486	4	2x3	1x2	0,1	8	131,5	0,0093	0,0089	0,0089	0,9392	0,9421	0,9408
487	4	2x3	2x1	0,1	8	127,4	0,0106	0,0093	0,0101	0,9298	0,9375	0,9315
488	4	2x3	2x2	0,1	8	146,4	0,0838	0,0832	0,0837	0,2496	0,2495	0,2568
489	4	3x2	1x1	0,1	8	247,0	0,0035	0,0037	0,0035	0,9778	0,976	0,9771
490	4	3x2	1x2	0,1	8	145,7	0,0068	0,0064	0,0066	0,9558	0,9565	0,9569
491	4	3x2	2x1	0,1	8	150,1	0,0095	0,0086	0,0096	0,9374	0,9433	0,936
492	4	3x2	2x2	0,1	8	142,5	0,0738	0,0737	0,0735	0,347	0,3466	0,3522
493	4	3x3	1x1	0,1	8	288,0	0,003	0,003	0,0027	0,9809	0,9808	0,9818
494	4	3x3	1x2	0,1	8	199,4	0,0045	0,0042	0,0042	0,9709	0,9728	0,9719
495	4	3x3	2x1	0,1	8	153,3	0,0068	0,0063	0,0065	0,9553	0,9588	0,9561
496	4	3x3	2x2	0,1	8	151,4	0,0215	0,0202	0,0202	0,8619	0,8693	0,8701
497	4	2x2	1x1	0,1	32	121,5	0,008	0,0072	0,0072	0,9493	0,9558	0,9534
498	4	2x2	1x2	0,1	32	65,2	0,0192	0,0172	0,0182	0,8742	0,8897	0,8789
499	4	2x2	2x1	0,1	32	65,5	0,0388	0,0363	0,0371	0,7258	0,7489	0,7449
500	4	2x2	2x2	0,1	32	52,3	0,0884	0,0883	0,0884	0,2775	0,2773	0,2775
501	4	2x3	1x1	0,1	32	146,6	0,0055	0,005	0,0053	0,9642	0,9675	0,9652
502	4	2x3	1x2	0,1	32	60,5	0,0181	0,0164	0,0167	0,8787	0,8902	0,889
503	4	2x3	2x1	0,1	32	60,1	0,022	0,0205	0,0206	0,8523	0,8616	0,8615
504	4	2x3	2x2	0,1	32	55,1	0,0876	0,0875	0,0875	0,2715	0,2778	0,2694
505	4	3x2	1x1	0,1	32	149,2	0,0065	0,0059	0,0059	0,9576	0,9616	0,9613
506	4	3x2	1x2	0,1	32	60,2	0,0179	0,0158	0,0165	0,8829	0,8945	0,8917
507	4	3x2	2x1	0,1	32	69,3	0,0293	0,026	0,0284	0,7986	0,8205	0,8055
508	4	3x2	2x2	0,1	32	54,9	0,0891	0,089	0,089	0,2544	0,2514	0,2499
509	4	3x3	1x1	0,1	32	183,3	0,0051	0,0047	0,0048	0,9674	0,9696	0,9675
510	4	3x3	1x2	0,1	32	68,9	0,0134	0,0128	0,0125	0,9121	0,9171	0,9175
511	4	3x3	2x1	0,1	32	66,5	0,0124	0,0112	0,0116	0,9174	0,9253	0,9218
512	4	3x3	2x2	0,1	32	56,4	0,0898	0,0898	0,0898	0,168	0,1627	0,165
513	4	2x2	1x1	0,1	128	97,4	0,017	0,015	0,0157	0,8858	0,8981	0,8951
514	4	2x2	1x2	0,1	128	35,3	0,0897	0,0897	0,0897	0,1878	0,1878	0,1892
515	4	2x2	2x1	0,1	128	35,4	0,0865	0,0864	0,0866	0,2953	0,299	0,2929
516	4	2x2	2x2	0,1	128	26,5	0,0896	0,0896	0,0896	0,159	0,1579	0,164

517	4	2x3	1x1	0,1	128	121,0	0,0189	0,0167	0,0175	0,8755	0,8917	0,8835
518	4	2x3	1x2	0,1	128	37,7	0,0899	0,0899	0,0899	0,1299	0,1257	0,1296
519	4	2x3	2x1	0,1	128	37,9	0,0894	0,0894	0,0894	0,2371	0,2398	0,2362
520	4	2x3	2x2	0,1	128	27,9	0,0899	0,0899	0,0899	0,1747	0,171	0,1776
521	4	3x2	1x1	0,1	128	122,1	0,0154	0,0138	0,0145	0,8993	0,9112	0,904
522	4	3x2	1x2	0,1	128	40,2	0,0896	0,0896	0,0896	0,1877	0,1891	0,1878
523	4	3x2	2x1	0,1	128	37,4	0,089	0,089	0,089	0,2572	0,2668	0,2649
524	4	3x2	2x2	0,1	128	28,1	0,09	0,09	0,09	0,1714	0,1669	0,1747
525	4	3x3	1x1	0,1	128	158,8	0,0066	0,0058	0,0059	0,9576	0,9615	0,9609
526	4	3x3	1x2	0,1	128	45,0	0,0889	0,0889	0,0889	0,2352	0,2403	0,2436
527	4	3x3	2x1	0,1	128	44,3	0,0893	0,0892	0,0892	0,2146	0,224	0,2177
528	4	3x3	2x2	0,1	128	30,7	0,09	0,09	0,09	0,1567	0,1568	0,155
529	4	2x2	1x1	0,01	8	222,3	0,0126	0,0114	0,0116	0,9171	0,9261	0,924
530	4	2x2	1x2	0,01	8	171,2	0,0894	0,0894	0,0894	0,148	0,1482	0,152
531	4	2x2	2x1	0,01	8	131,1	0,0893	0,0893	0,0893	0,247	0,2471	0,2507
532	4	2x2	2x2	0,01	8	149,4	0,0895	0,0895	0,0895	0,134	0,1265	0,132
533	4	2x3	1x1	0,01	8	257,5	0,0127	0,0114	0,0116	0,9179	0,9257	0,9247
534	4	2x3	1x2	0,01	8	172,5	0,09	0,09	0,09	0,1416	0,1398	0,1461
535	4	2x3	2x1	0,01	8	141,9	0,0886	0,0886	0,0886	0,2705	0,2664	0,2784
536	4	2x3	2x2	0,01	8	161,9	0,09	0,09	0,09	0,1307	0,1264	0,1317
537	4	3x2	1x1	0,01	8	260,5	0,0133	0,0123	0,0125	0,9124	0,9194	0,9171
538	4	3x2	1x2	0,01	8	148,8	0,0478	0,0455	0,0467	0,6482	0,6764	0,6582
539	4	3x2	2x1	0,01	8	154,3	0,0412	0,0379	0,0398	0,7038	0,7268	0,7141
540	4	3x2	2x2	0,01	8	163,0	0,0898	0,0898	0,0898	0,1769	0,1763	0,1823
541	4	3x3	1x1	0,01	8	305,3	0,0094	0,0085	0,0085	0,9394	0,9444	0,9454
542	4	3x3	1x2	0,01	8	209,6	0,045	0,0415	0,0436	0,7046	0,7355	0,7148
543	4	3x3	2x1	0,01	8	213,3	0,0352	0,0327	0,0338	0,7567	0,7752	0,7682
544	4	3x3	2x2	0,01	8	170,2	0,0899	0,0899	0,0899	0,1617	0,1534	0,1646
545	4	2x2	1x1	0,01	32	127,4	0,0186	0,0163	0,0172	0,8805	0,8947	0,8929
546	4	2x2	1x2	0,01	32	58,5	0,0899	0,0899	0,0899	0,12	0,1193	0,121
547	4	2x2	2x1	0,01	32	70,1	0,09	0,09	0,09	0,0882	0,0911	0,0912
548	4	2x2	2x2	0,01	32	57,9	0,0898	0,0898	0,0898	0,1137	0,1132	0,1134
549	4	2x3	1x1	0,01	32	153,1	0,0192	0,0165	0,0183	0,8685	0,8912	0,875
550	4	2x3	1x2	0,01	32	74,7	0,0898	0,0898	0,0898	0,1811	0,1759	0,1849
551	4	2x3	2x1	0,01	32	62,2	0,0895	0,0895	0,0895	0,1776	0,182	0,1818
552	4	2x3	2x2	0,01	32	59,5	0,09	0,09	0,0899	0,1359	0,1305	0,1364
553	4	3x2	1x1	0,01	32	153,4	0,0159	0,0146	0,0149	0,8939	0,9011	0,9015
554	4	3x2	1x2	0,01	32	74,3	0,0897	0,0897	0,0897	0,1214	0,1183	0,1192
555	4	3x2	2x1	0,01	32	73,3	0,0898	0,0898	0,0898	0,15	0,1439	0,1589
556	4	3x2	2x2	0,01	32	59,0	0,09	0,09	0,09	0,1901	0,1907	0,1898
557	4	3x3	1x1	0,01	32	191,6	0,0179	0,0156	0,0168	0,8796	0,8971	0,8876
558	4	3x3	1x2	0,01	32	89,5	0,0897	0,0897	0,0897	0,1272	0,1319	0,13
559	4	3x3	2x1	0,01	32	81,0	0,0899	0,0899	0,0899	0,1189	0,1298	0,1238
560	4	3x3	2x2	0,01	32	62,8	0,0899	0,0899	0,0899	0,1768	0,1754	0,1831
561	4	2x2	1x1	0,01	128	99,7	0,0897	0,0897	0,0897	0,1546	0,1623	0,1569
562	4	2x2	1x2	0,01	128	39,7	0,09	0,09	0,09	0,0968	0,1009	0,0974
563	4	2x2	2x1	0,01	128	39,9	0,0899	0,0899	0,0899	0,1066	0,1052	0,1087
564	4	2x2	2x2	0,01	128	30,0	0,09	0,09	0,09	0,1365	0,1418	0,1293

565	4	2x3	1x1	0,01	128	124,1	0,089	0,089	0,089	0,3272	0,339	0,3234
566	4	2x3	1x2	0,01	128	40,5	0,09	0,09	0,09	0,0944	0,093	0,0986
567	4	2x3	2x1	0,01	128	41,5	0,0899	0,0899	0,0899	0,1222	0,1228	0,1166
568	4	2x3	2x2	0,01	128	32,3	0,09	0,09	0,09	0,1144	0,1103	0,1187
569	4	3x2	1x1	0,01	128	124,7	0,0545	0,053	0,0534	0,6682	0,6983	0,6769
570	4	3x2	1x2	0,01	128	40,4	0,09	0,09	0,09	0,1382	0,1466	0,1353
571	4	3x2	2x1	0,01	128	40,7	0,09	0,09	0,09	0,1823	0,1814	0,1813
572	4	3x2	2x2	0,01	128	31,1	0,09	0,09	0,09	0,1043	0,1051	0,1035
573	4	3x3	1x1	0,01	128	162,5	0,0858	0,0855	0,0857	0,3392	0,3579	0,3391
574	4	3x3	1x2	0,01	128	47,4	0,0899	0,0899	0,0899	0,122	0,125	0,1222
575	4	3x3	2x1	0,01	128	46,7	0,0899	0,09	0,09	0,1219	0,1277	0,1173
576	4	3x3	2x2	0,01	128	33,3	0,09	0,09	0,09	0,1097	0,107	0,1077

Tabelle 6: Tabelle mit den Ergebnissen der Experimente mit CNNs (Quelle: Eigene Tabelle)

E.3 Tabelle der Ergebnisse der Experimente mit LSTMs

Die Spalten Bias (Wert: True) und Epoch (Wert: 3) wurden weggelassen, da ihre Werte für alle Konfigurationen konstant sind.

Index-Nr	Bauart-Nr	Aktf.	rek. Aktf.	Lernrate	Mini Batch	Trainingszeit (s)	Train-Cost	Val.-Cost	Test-Cost	Train-Quote	Val.-Quote	Test-Quote
1	1	σ	σ	1	8	68,3	0,0868	0,087	0,0867	0,2055	0,1988	0,2054
2	1	σ	tanh	1	8	68,9	0,0333	0,0309	0,0325	0,7769	0,7997	0,7854
3	1	tanh	σ	1	8	69,0	0,0147	0,0133	0,0139	0,9058	0,9137	0,9078
4	1	tanh	tanh	1	8	69,3	0,0716	0,0699	0,0707	0,4516	0,4656	0,4593
5	1	σ	σ	1	32	20,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
6	1	σ	tanh	1	32	20,8	0,0554	0,0532	0,0556	0,6032	0,6257	0,5995
7	1	tanh	σ	1	32	21,2	0,0359	0,0337	0,0359	0,7543	0,7764	0,7561
8	1	tanh	tanh	1	32	21,4	0,0765	0,0766	0,0772	0,4691	0,4678	0,4638
9	1	σ	σ	1	128	10,0	0,09	0,09	0,09	0,1496	0,1566	0,1544
10	1	σ	tanh	1	128	10,3	0,09	0,09	0,09	0,1156	0,1092	0,1158
11	1	tanh	σ	1	128	10,5	0,0728	0,0722	0,0722	0,3864	0,3911	0,3926
12	1	tanh	tanh	1	128	10,5	0,0897	0,0897	0,0897	0,1179	0,109	0,1173
13	1	σ	σ	0,1	8	72,3	0,09	0,09	0,09	0,1159	0,1092	0,1158
14	1	σ	tanh	0,1	8	72,5	0,0897	0,0896	0,0897	0,1812	0,1835	0,1813
15	1	tanh	σ	0,1	8	72,6	0,0529	0,0521	0,0523	0,6096	0,6216	0,6163
16	1	tanh	tanh	0,1	8	73,5	0,0898	0,0898	0,0898	0,1146	0,1075	0,1146
17	1	σ	σ	0,1	32	23,7	0,09	0,09	0,09	0,1136	0,1065	0,1135
18	1	σ	tanh	0,1	32	23,9	0,09	0,09	0,09	0,1258	0,1242	0,1277
19	1	tanh	σ	0,1	32	24,2	0,0874	0,0875	0,0873	0,1986	0,1954	0,2106
20	1	tanh	tanh	0,1	32	24,2	0,09	0,09	0,09	0,1145	0,1081	0,1144
21	1	σ	σ	0,1	128	12,6	0,0905	0,0906	0,0905	0,1175	0,122	0,1174
22	1	σ	tanh	0,1	128	13,5	0,09	0,09	0,09	0,1257	0,117	0,1247
23	1	tanh	σ	0,1	128	13,4	0,0897	0,0896	0,0897	0,1111	0,1105	0,1078
24	1	tanh	tanh	0,1	128	13,4	0,09	0,09	0,09	0,1159	0,1095	0,1155
25	1	σ	σ	0,01	8	76,1	0,0902	0,0903	0,0903	0,1121	0,1105	0,1052
26	1	σ	tanh	0,01	8	76,9	0,09	0,09	0,09	0,1158	0,1091	0,1166
27	1	tanh	σ	0,01	8	78,6	0,0899	0,0899	0,0899	0,1208	0,1248	0,1217
28	1	tanh	tanh	0,01	8	77,3	0,09	0,09	0,09	0,1154	0,1088	0,1146
29	1	σ	σ	0,01	32	26,5	0,0908	0,0908	0,0908	0,102	0,103	0,101
30	1	σ	tanh	0,01	32	26,9	0,0899	0,0899	0,0899	0,1494	0,1464	0,1522
31	1	tanh	σ	0,01	32	27,2	0,0897	0,0896	0,0897	0,1786	0,1847	0,1783
32	1	tanh	tanh	0,01	32	27,2	0,09	0,09	0,09	0,1196	0,1137	0,1201
33	1	σ	σ	0,01	128	15,4	0,0914	0,0913	0,0914	0,0986	0,0991	0,098
34	1	σ	tanh	0,01	128	16,0	0,09	0,09	0,09	0,1154	0,1111	0,1179
35	1	tanh	σ	0,01	128	15,8	0,0901	0,0901	0,0901	0,1218	0,1247	0,1216
36	1	tanh	tanh	0,01	128	16,1	0,09	0,09	0,09	0,1269	0,1207	0,1282
37	2	σ	σ	1	8	143,1	0,09	0,09	0,09	0,1136	0,1064	0,1135
38	2	σ	tanh	1	8	145,0	0,024	0,0217	0,0231	0,8489	0,8668	0,8559

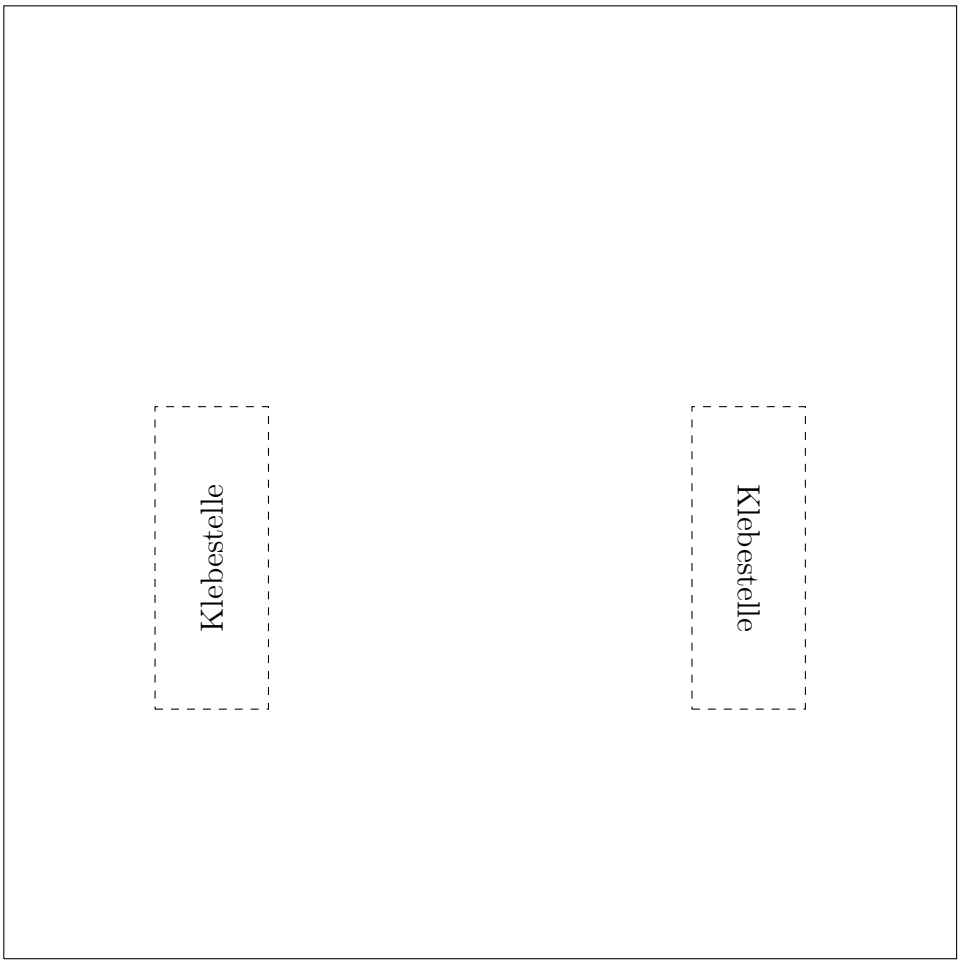
39	2	tanh	σ	1	8	148,4	0,008	0,0076	0,0078	0,9498	0,9521	0,9503
40	2	tanh	tanh	1	8	146,5	0,0502	0,0483	0,0493	0,6336	0,6485	0,6428
41	2	σ	σ	1	32	50,7	0,09	0,09	0,09	0,1136	0,1064	0,1135
42	2	σ	tanh	1	32	50,4	0,0778	0,0769	0,0779	0,2986	0,2993	0,2968
43	2	tanh	σ	1	32	51,4	0,0189	0,0178	0,0177	0,877	0,8848	0,8864
44	2	tanh	tanh	1	32	52,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
45	2	σ	σ	1	128	28,7	0,0901	0,0901	0,0901	0,1136	0,1064	0,1135
46	2	σ	tanh	1	128	29,4	0,09	0,09	0,09	0,1136	0,1064	0,1135
47	2	tanh	σ	1	128	30,1	0,0714	0,0709	0,071	0,3658	0,3719	0,3648
48	2	tanh	tanh	1	128	30,4	0,09	0,09	0,09	0,1136	0,1064	0,1135
49	2	σ	σ	0,1	8	152,5	0,09	0,09	0,09	0,1136	0,1064	0,1135
50	2	σ	tanh	0,1	8	154,7	0,09	0,09	0,09	0,1136	0,1064	0,1135
51	2	tanh	σ	0,1	8	153,6	0,047	0,0464	0,0463	0,6611	0,6715	0,6739
52	2	tanh	tanh	0,1	8	156,5	0,09	0,09	0,09	0,1136	0,1064	0,1135
53	2	σ	σ	0,1	32	57,5	0,09	0,0901	0,09	0,1136	0,1064	0,1135
54	2	σ	tanh	0,1	32	57,6	0,09	0,09	0,09	0,1164	0,1098	0,1159
55	2	tanh	σ	0,1	32	57,9	0,0869	0,087	0,0868	0,2305	0,2247	0,2284
56	2	tanh	tanh	0,1	32	59,5	0,09	0,09	0,09	0,1136	0,1064	0,1135
57	2	σ	σ	0,1	128	35,0	0,0905	0,0905	0,0905	0,1136	0,1064	0,1135
58	2	σ	tanh	0,1	128	36,4	0,09	0,09	0,09	0,1292	0,1202	0,1275
59	2	tanh	σ	0,1	128	36,2	0,0899	0,0899	0,0899	0,1388	0,1388	0,1411
60	2	tanh	tanh	0,1	128	36,6	0,09	0,09	0,09	0,1136	0,1064	0,1135
61	2	σ	σ	0,01	8	162,9	0,0904	0,0905	0,0904	0,0994	0,099	0,1032
62	2	σ	tanh	0,01	8	162,0	0,09	0,09	0,09	0,1243	0,1186	0,1241
63	2	tanh	σ	0,01	8	163,9	0,0888	0,0887	0,0888	0,1656	0,171	0,1668
64	2	tanh	tanh	0,01	8	162,6	0,09	0,09	0,09	0,1136	0,1064	0,1135
65	2	σ	σ	0,01	32	64,2	0,0906	0,0907	0,0907	0,099	0,0967	0,0958
66	2	σ	tanh	0,01	32	64,1	0,09	0,09	0,09	0,121	0,1128	0,1214
67	2	tanh	σ	0,01	32	64,7	0,0899	0,0899	0,0899	0,1499	0,1526	0,1509
68	2	tanh	tanh	0,01	32	65,3	0,09	0,09	0,09	0,1136	0,1064	0,1135
69	2	σ	σ	0,01	128	41,3	0,0903	0,0903	0,0903	0,0998	0,0961	0,1009
70	2	σ	tanh	0,01	128	41,8	0,09	0,09	0,09	0,128	0,1255	0,1283
71	2	tanh	σ	0,01	128	42,5	0,09	0,09	0,0901	0,1312	0,1299	0,1312
72	2	tanh	tanh	0,01	128	42,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
73	3	σ	σ	1	8	109,8	0,09	0,0901	0,09	0,1136	0,1064	0,1135
74	3	σ	tanh	1	8	109,4	0,0859	0,0863	0,0856	0,1788	0,1732	0,1802
75	3	tanh	σ	1	8	109,4	0,0132	0,0122	0,013	0,9151	0,9198	0,9148
76	3	tanh	tanh	1	8	109,7	0,0514	0,0494	0,05	0,6267	0,6377	0,6349
77	3	σ	σ	1	32	51,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
78	3	σ	tanh	1	32	52,2	0,09	0,09	0,09	0,1136	0,1064	0,1135
79	3	tanh	σ	1	32	52,3	0,0256	0,0238	0,0237	0,8325	0,8462	0,8446
80	3	tanh	tanh	1	32	52,5	0,0521	0,0503	0,0521	0,6268	0,6564	0,6304
81	3	σ	σ	1	128	38,3	0,09	0,09	0,09	0,1136	0,1064	0,1135
82	3	σ	tanh	1	128	38,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
83	3	tanh	σ	1	128	39,0	0,0714	0,0712	0,0709	0,401	0,4024	0,4072
84	3	tanh	tanh	1	128	39,2	0,09	0,09	0,09	0,1136	0,1064	0,1135
85	3	σ	σ	0,1	8	114,6	0,09	0,09	0,09	0,1136	0,1064	0,1135
86	3	σ	tanh	0,1	8	114,0	0,0899	0,0899	0,0899	0,1137	0,1065	0,1135

87	3	tanh	σ	0,1	8	115,0	0,0679	0,0671	0,0678	0,4556	0,4531	0,4594
88	3	tanh	tanh	0,1	8	115,0	0,0889	0,0888	0,0889	0,2059	0,21	0,207
89	3	σ	σ	0,1	32	55,3	0,0903	0,0903	0,0903	0,1035	0,109	0,1028
90	3	σ	tanh	0,1	32	56,3	0,09	0,09	0,09	0,1136	0,1064	0,1135
91	3	tanh	σ	0,1	32	57,1	0,0889	0,0889	0,089	0,204	0,2024	0,2067
92	3	tanh	tanh	0,1	32	56,7	0,09	0,09	0,09	0,1145	0,1087	0,114
93	3	σ	σ	0,1	128	42,3	0,0908	0,0908	0,0908	0,102	0,103	0,101
94	3	σ	tanh	0,1	128	42,6	0,09	0,09	0,09	0,1097	0,1142	0,111
95	3	tanh	σ	0,1	128	42,9	0,0898	0,0898	0,0897	0,1612	0,1574	0,1675
96	3	tanh	tanh	0,1	128	42,9	0,09	0,09	0,09	0,114	0,1068	0,1137
97	3	σ	σ	0,01	8	119,0	0,0901	0,0901	0,0901	0,0972	0,0983	0,0982
98	3	σ	tanh	0,01	8	120,3	0,09	0,09	0,09	0,0995	0,0995	0,1033
99	3	tanh	σ	0,01	8	120,2	0,0896	0,0896	0,0896	0,1525	0,1522	0,1598
100	3	tanh	tanh	0,01	8	120,7	0,09	0,09	0,09	0,1146	0,1074	0,1145
101	3	σ	σ	0,01	32	59,8	0,0905	0,0906	0,0904	0,1136	0,1064	0,1135
102	3	σ	tanh	0,01	32	59,4	0,0902	0,0902	0,0902	0,102	0,103	0,101
103	3	tanh	σ	0,01	32	60,1	0,0901	0,0901	0,0901	0,13	0,1301	0,1292
104	3	tanh	tanh	0,01	32	60,9	0,09	0,09	0,09	0,1156	0,1088	0,1152
105	3	σ	σ	0,01	128	45,9	0,0912	0,0913	0,0912	0,099	0,0967	0,0958
106	3	σ	tanh	0,01	128	46,0	0,0903	0,0903	0,0903	0,102	0,103	0,101
107	3	tanh	σ	0,01	128	46,2	0,0901	0,0901	0,0901	0,1048	0,1045	0,1026
108	3	tanh	tanh	0,01	128	46,6	0,09	0,09	0,09	0,1178	0,1102	0,1181
109	4	σ	σ	1	8	195,7	0,09	0,09	0,09	0,1136	0,1064	0,1135
110	4	σ	tanh	1	8	195,9	0,09	0,09	0,09	0,1136	0,1064	0,1135
111	4	tanh	σ	1	8	196,4	0,0076	0,0076	0,0078	0,9517	0,952	0,9494
112	4	tanh	tanh	1	8	197,7	0,09	0,09	0,09	0,1035	0,109	0,1028
113	4	σ	σ	1	32	88,3	0,09	0,09	0,09	0,1136	0,1064	0,1135
114	4	σ	tanh	1	32	88,0	0,09	0,09	0,09	0,1136	0,1064	0,1135
115	4	tanh	σ	1	32	89,6	0,025	0,0233	0,0236	0,8353	0,8476	0,8457
116	4	tanh	tanh	1	32	89,6	0,09	0,09	0,09	0,1136	0,1064	0,1135
117	4	σ	σ	1	128	63,4	0,09	0,09	0,09	0,1136	0,1064	0,1135
118	4	σ	tanh	1	128	63,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
119	4	tanh	σ	1	128	64,2	0,0707	0,071	0,0706	0,4154	0,4161	0,4098
120	4	tanh	tanh	1	128	64,7	0,09	0,09	0,09	0,1136	0,1064	0,1135
121	4	σ	σ	0,1	8	203,4	0,09	0,09	0,09	0,1136	0,1064	0,1135
122	4	σ	tanh	0,1	8	205,9	0,09	0,09	0,09	0,1136	0,1064	0,1135
123	4	tanh	σ	0,1	8	209,2	0,0473	0,0458	0,0466	0,6745	0,688	0,685
124	4	tanh	tanh	0,1	8	208,7	0,09	0,09	0,09	0,1136	0,1064	0,1135
125	4	σ	σ	0,1	32	95,8	0,0904	0,0904	0,0904	0,1136	0,1064	0,1135
126	4	σ	tanh	0,1	32	96,1	0,09	0,09	0,09	0,1136	0,1064	0,1135
127	4	tanh	σ	0,1	32	99,2	0,087	0,0869	0,0869	0,1808	0,1798	0,1854
128	4	tanh	tanh	0,1	32	97,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
129	4	σ	σ	0,1	128	70,5	0,0908	0,0909	0,0908	0,1136	0,1064	0,1135
130	4	σ	tanh	0,1	128	70,5	0,09	0,09	0,09	0,0986	0,0991	0,098
131	4	tanh	σ	0,1	128	71,7	0,09	0,09	0,09	0,1335	0,1315	0,1328
132	4	tanh	tanh	0,1	128	71,8	0,09	0,09	0,09	0,1136	0,1064	0,1135
133	4	σ	σ	0,01	8	214,9	0,0906	0,0907	0,0907	0,1136	0,1064	0,1135
134	4	σ	tanh	0,01	8	216,8	0,09	0,09	0,09	0,1136	0,1064	0,1135

135	4	tanh	σ	0,01	8	217,2	0,0895	0,0895	0,0895	0,1783	0,1814	0,1867
136	4	tanh	tanh	0,01	8	217,0	0,09	0,09	0,09	0,1136	0,1064	0,1135
137	4	σ	σ	0,01	32	103,5	0,0909	0,0909	0,0909	0,102	0,103	0,101
138	4	σ	tanh	0,01	32	103,9	0,09	0,09	0,09	0,1034	0,1089	0,1028
139	4	tanh	σ	0,01	32	104,5	0,09	0,09	0,09	0,1287	0,1252	0,1216
140	4	tanh	tanh	0,01	32	105,0	0,09	0,09	0,09	0,1136	0,1064	0,1135
141	4	σ	σ	0,01	128	77,7	0,0911	0,0909	0,0911	0,1035	0,109	0,1028
142	4	σ	tanh	0,01	128	77,9	0,09	0,09	0,09	0,1136	0,1064	0,1135
143	4	tanh	σ	0,01	128	78,8	0,0899	0,0899	0,0899	0,1276	0,1318	0,1282
144	4	tanh	tanh	0,01	128	79,9	0,09	0,09	0,09	0,1136	0,1064	0,1135

Tabelle 7: Tabelle mit den Ergebnissen der Experimente mit LSTMs (Quelle: Eigene Tabelle)

Anhang F: Daten-DVD



Dieser Datenträger enthält eine digitale Fassung der Arbeit, den originalen Programmcode für alle drei Netztypen sowie die bei den Experimenten entstandenen KNNs und Daten.

Selbstständigkeitserklärung

Name: Tobias Prisching

Ich erkläre, dass ich diese vorwissenschaftliche Arbeit eigenständig angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift