Bachelor Thesis

Analyse und Weiterentwicklung des neuronalen Netzes von Takeuchi & Lee für Momentum Strategien

Steven Hooker

Analyse und Weiterentwicklung des neuronalen Netzes von Takeuchi & Lee für Momentum Strategien

Eine empirische Analyse am Beispiel des deutschen Aktien Marktes

Bachelor Thesis

an der Frankfurt School of Finance and Management

eingereicht bei

Prof. Dr. Olaf Stotz

und

Andreas Albrecht, Dipl. Ing.

von

Steven Hooker

Bachelor of Science

Matrikelnumer: 4920039

Alter Weg 22

35687 Dillenburg

Mobil: +49 1736828892

E-Mail: steven.hooker@fs-students.de

Frankfurt am Main, 15.03.2017

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

ML	Machine Learning
DL	Deep Learning
NN	Neuronale Netze
KI	Künstliche Intelligenz
RBM	Restricted Boltzmann Machine
DBN	Deep Belief Net

1. Einleitung

Folgend werden die Fortschritte im Machine Learning (ML) Bereich und die damit einhergehenden neuen Anwendungsszenarien erläutert. Basierend darauf wird die Positionierung von Deep Learning (DL), einer Unterkategorie von ML, im Asset Management beschrieben und damit die Relevanz des Themas untermauert. Anschließend wird die Zielsetzung, die wissenschaftliche Fragestellung und der Aufbau der Thesis erläutert.

ML und die Unterkategorie DL haben in den letzten Jahren massiv an Relevanz dazugewonnen. Dies ist unter anderem an der Zahl der Publikationen mit ML oder DL im Titel bis 2010, 27.974 Publikationen und ab 2010, 68.710 Publikationen, zu erkennen. Dies liegt vor allem, aufgrund der erhöhten Rechenkapazitäten, an der Realisierbarkeit dessen was noch vor 20 Jahren nicht möglich war. Eine heutige Gamer Grafikkarte, z.B. die Titan x, mit 11 Terraflops ist bereits 23,9 Mal schneller als das Höchstleistungsrechenzentrum in Stuttgart von 1997 mit 461 Gigaflops (O. V.; O. V. 2016). Dieser Anstieg in der Rechenkapazität ermöglicht viele neue Anwendungsszenarien neben den klassischen rechenintensiven Spielen, für die, diese sehr leistungsfähigen Hardware gebaut wurde. Zu den Anwendungsszenarien gehört auch die Entwicklung von neuronalen Netzen (NN) für verschiedenste Aufgaben. Die Theorien, die vor 20 Jahren auf dem Papier bestanden und nur mit extrem teurer Hardware und sehr hohem Aufwand in die Realität umgesetzt werden konnten, können aufgrund heutiger Bedingungen in wenigen Stunden entwickelt und getestet werden.

Die bekanntesten Beispiele von neuronalen Netzen sind im Bereich der Bild-, Spracherkennung oder bei den größeren Suchmaschinen wie Google oder Bing vorzufinden (Schmidhuber 2015). In diesen Bereichen schlagen mittlerweile die Computer, diese neuronalen Netzte, die menschlichen Fähigkeiten (Yu et al. 2016). Anfang des Jahres 2017 gewann Alpha Go, eine entwickelte künstliche Intelligenz (KI) für das chinesische Spiel Go, 60 von 60 Spielen gegen die weltbesten Spieler (Berben 2017). Diese professionellen Spieler lernen von Alpha Gos Spielzügen um selber besser zu werden.

Nun stellt sich die Frage, wenn NN, vom Menschen geschaffen, so komplexe zusammenhalte erkennen können, sollten sie auch in der Lage sein weitere Muster am Kapitalmarkt zu erkennen und bestehende Systeme zu verbessern um die Genauigkeit der Prognosen zu erhöhen.

1.1. Problemstellung und Relevanz des Themas

Ein Kapitalmarkt ist ein organisierter Markt für Börsengeschäfte sowie für die Vermittlung von Angebot und Nachfrage durch die Kreditinstitute. Es ist somit ein System mit vielen Teilnehmern (Wurgler 2000). Prognosen die genauere Ergebnisse erzielen als Zufallsmethoden können zu finanziellen Gewinnen für die Anwender der Prognosen führen. Daher ist eine hohe Genauigkeit von Prognosen gewünscht (Al-Hmouz et al. 2015).

Nun stellt sich die Frage, ob NN in der Lage sind weitere Muster am Kapitalmarkt zu erkennen und bestehende Systeme zu verbessern um die Genauigkeit der Prognosen zu erhöhen. Genau mit dieser Aufgabe haben sich einige Forscher beschäftigt und sind zu unterschiedlichen Ergebnissen gelangt (Jia; Göçken et al. 2016).

Die Herren Takeuchi und Lee haben ein NN bestehend aus modifizierten Restricted Boltzmann Machines (RBMs) genutzt um Muster anhand von historischen Performancedaten zu erkennen und basierend darauf Kaufentscheidungen für die jeweiligen Aktien zu treffen (Lawrence Takeuchi * und Yu-Ying (Albert) Lee; Lawrence Takeuchi * und Yu-Ying (Albert) Lee). Dieses Netz wird in dieser Arbeit repliziert, analysiert und weiterentwickelt. Womit folgend die Zielsetzung der Theis näher betrachtet wird.

1.2. Zielsetzung

Ziel dieser Thesis ist die Analyse und Weiterentwicklung des Modelles von Takeuchi & Lee für Momentum Strategien (Lawrence Takeuchi * und Yu-Ying (Albert) Lee). Diese Arbeit wird folgend als Ursprungsarbeit bezeichnet.

Das dort beschriebene NN wird repliziert und auf den deutschen Aktienmarkt angewendet. In der Ursprungsarbeit wurden die Gewichte im NN während dem Testzeitraum von Januar 1990 bis Dezember 2009 nicht aktualisiert. Dies lässt vermuten, dass das NN nicht immer die aktuellen Marktbedingungen abbilden kann und demnach eine abnehmende Prognosegenauigkeit im späteren Testzeitverlauf aufweist. Diese Hypothese wird untersucht werden und es wird ein NN entwickelt bei dem die Gewichte im Testzeitraum weiter trainiert werden. Dieses entstandene Netz wird analysiert, es soll verstanden werden wie es funktioniert und zu Entschlüssen gelangt. Darüber hinaus wird überprüft ob es höhere und konstantere Prognosegenauigkeiten aufweist als die Urpsrungsarbeit. Um dies zu ermöglichen werden in Kapitel 2 die technischen Grundlagen zum Verständnis der Arbeit vermittelt.

In der Sensitivitätsanalyse werden Fragestellungen zur Robustheit des entstandenen Modelles und weitere Fragestellungen und Verbesserungsmöglichkeiten, die während der Arbeit an dieser Thesis entstanden sind, betrachtet.

1.3. Aufbau der Bachelor Thesis

Die Grundlagen der verwendeten Methoden und Begriffe werden im Kapitel 2 erläutert.

In Kapitel 3 werden das Modell und die nötige Datenvorverarbeitung erläutert.

Folgend werden in Kapitel 4 die Datengrundlage und die Ergebnisse des Backtests betrachtet. Hierfür werden neben den Kennzahlen zur Bestimmung der Güte eines NNs auch Portfolien und Benchmarks mit unterschiedlichen Gewichtungen gebildet um Rendite- und Risikokennzahlen des Asset Managements anzuwenden. Dies ermöglicht es, die gebildeten Portfolios mit potenziellen Konkurrenzprodukten zu vergleichen.

Bei der Analyse des Modelles soll das entstandene Modell an sich verstanden werden. Das entwickelte NN wird ein Modell mit bis zu 120 nichtlinearen Transformationen und 4 Ebenen sein, welches am Ende eine Zahl zwischen 0 und 1 als Wahrscheinlichkeitswert, ob die jeweilige Aktie in der Folgeperiode den Markt übertreffen wird, ausgeben wird. Ziel wird es sein herauszufinden bei welchen Konstellationen eine 1 oder 0 als Ausgabe zu erwarten ist um somit eine Interpretierung des Modelles zu ermöglichen.

Es werden im Zuge der Diskussion in Kapitel 4.4 Fragestellungen zur Robustheit des entstandenen Modelles analysier. In der Sensitivitätsanalyse in Kapitel 5 werden weitere Verbesserungsmöglichkeiten betrachtet.

Der Backtest, die Analyse und Interpretation des Modelles sowie die Sensitivitätsanalyse sollen dem Ziel der Bachelor Thesis dienen, das entstandene NN zu verstehen und das beschriebene Modell in der Ursprungsarbeit weiterzuentwickeln.

Abschließend folgt eine Schlussbetrachtung, inklusive der Zusammenfassung, den Limitationen des Modells und einem Ausblick für weitere Forschungen.

2. Technische Grundlagen

Folgend wird das technische Grundwissen zum Verständnis der Arbeit vermittelt. Hierzu gehört die Unterscheidung von Machine- und Deep Learning, die Architektur von NN und welche Aufgabe Error- und Aktivationsfunktionen haben. Dies ist nötig um zu verstehen wie ein NN lernt. Es werden in dieser Arbeit zwei Lernverfahren angewendet, das allgemein genutzte Backpropagation und ein Verfahren zum Trainieren von gestapelten Autoencodern.

Das folgende technische Wissen wurde Großteils entnommen aus dem bekannten Standardwerk zu DL „Deep Learning Methods and Applications“ (Deng 2013).

2.1. Machine Learning

ML ist eine Unterkategorie im Bereich der KI und definiert den Oberbegriff für die Generierung von Wissen aus Erfahrung. Ein Modell, künstliches System, lernt aus Beispieldaten (Text, Videos, Bilder, Zahlen, ...) und extrahiert während der Lernphase relevante Faktoren um Muster zu erkennen und kann nach Beendigung diese Beispieldaten verallgemeinern. Anders als bei einprogrammierten Regeln, soll das Modell selbst Muster und Gesetzmäßigkeiten in den Lerndaten erkennen. Das weitergehende Ziel vom ML ist es zukünftige Daten zu interpretieren und Prognosen zu erstellen. Dies wird dann als testen oder validieren des Modelles bezeichnet.

Flache ML Architekturen wie Support Vector Machinen oder Self Organized Maps, sind aufgrund der niedrigen Dimensionalität beschränkt in der Erkennung von Mustern, daher wurden seit den 70ern Forschungen bzgl. NN und anderen tieferen Strukturen durchgeführt die diese Limitation nicht haben. Diese NN sind heute unter anderem als Deep Neural Nets, Neural Nets oder Deep learning bekannt. Worauf im Folgenden Kapitel eingegangen wird.

2.2. Neuronale Netze

Folgend wird auf die Architektur, die Bestandteile eines NNs, verschiedene Error- und Aktivationsfunktionen und wie sie im Lernprozess miteinander agieren eingegangen.

2.2.1. Architektur

Grundsätzlich handelt es sich bei NN um eine tiefe Struktur mit mehreren Ebenen. In jeder Ebene werden nicht lineare Transformationen durchgeführt und die Ergebnisse einer Ebene sind die Eingabewerte der nächsten Ebene. Folgende eine Grafik um dies zu verdeutlichen. Im weiteren Verlauf der Thesis wird für Ebene der englische Begriff Layer verwendet.

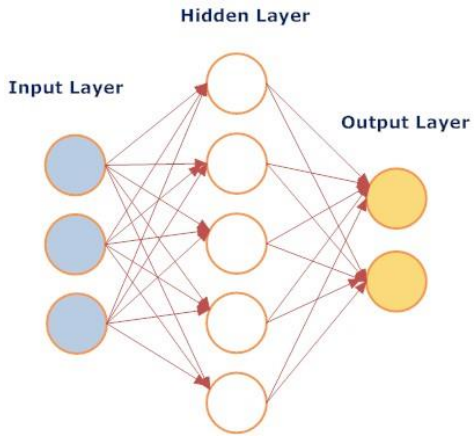

Abbildung 1: NN Abbildung[1]

Der erste Layer, in welchem die Eingabewerte in das NN eingegeben werden, wird Input Layer und der letzten der die gewünschten Ausgabewerte liefert wird Output Layer genannt. Dazwischen werden alle Layer als Hidden Layer bezeichnet, da der Anwender eines NNs nicht weis was in diesen passiert.

Die Elemente in einem Layer werden Neuronen genannt. Diese Neuronen führen die nichtlineare Transformation durch.

Der Aufbau eines Neurons ist in der Abbildung 2 dargestellt.

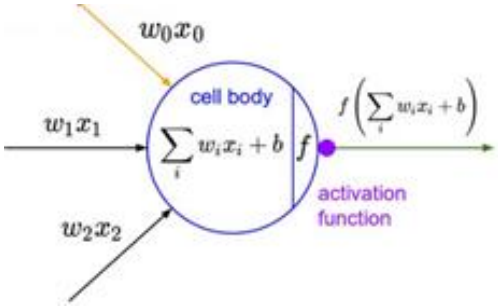

Abbildung 2: Neuron Berechnungen[2]

[1] Vgl. Colin J Holcombe: EBusiness
[2] In enger Anlehnung an Andrej Karpathy: CS231n: Convolutional Neural Networks for Visual Recognition, 2016

Ein Neuron gewichtet und summiert, wie in der Abbildung 2 und in Formel 1 zu erkennen, seine Eingangswerte.

$$Z = \sum_{i=1}^{I} w_i * x_i + b$$

Formel 1: Berechnung im Neuron

Die Gewichte und der Bias werden während dem Lernprozess verändert um die Daten abzubilden. Mehr dazu im Kapitel 2.2.3. Lernprozess. Anschließend wird eine sogenannte Aktivationsfunktion ausgeführt. Der Ausgabewert der Aktivationsfunktion dient als Eingabewert für das nächste Neuron. Folgend werden die in dieser Arbeit genutzten Aktivationsfunktionen beschrieben.

2.2.2. Aktivationsfunktionen

Über die Zeit wurden Aktivationsfunktionen entwickelt, die verschiedene Eigenschaften aufweisen. Die Aktivationsfunktionen werden zum Beispiel nach der Anzahl der Variablen, nach Eigenschaften zur Monotonie und Symmetrie unterschieden. Mit den unterschiedlichen Eigenschaften erzielen die Aktivationsfunktionen unterschiedliche Ergebnisse vor allem bei der Geschwindigkeit des Lernprozesses und auch der anschließenden Klassifikationsgüte.

Im Modell wird eine Suche nach der passendsten Aktivationsfunktion durchgeführt, daher werden die Formeln für die in der Suche verwendeten Aktivationsfunktionen gelistet. Im Modell selber wird anschließend nur eine Aktivationsfunktion verwendet.

$$Tanh(Z) = \frac{\sinh(Z)}{\cosh(Z)} = \frac{e^{2Z} - 1}{e^{2Z} + 1}$$

Formel 2: Aktivationsfunktion Tanh

$$f(Z) = \max(0, Z)$$

Formel 3: Aktivationsfunktion Rectifier

$$Maxout(Z) = \max(w_1 * Z + b_1, w_2 * Z + b_2)$$

Formel 4: Aktivationsfunktion Maxout

In der Abbildung 3 sind die unterschiedlichen Funktionen grafisch dargestellt. Die Variablen der Maxout Funktion besitzen in diesem Fall die Werte (w_1=0,3; w_2=0,5; b_1=0; b_2=-0,15).

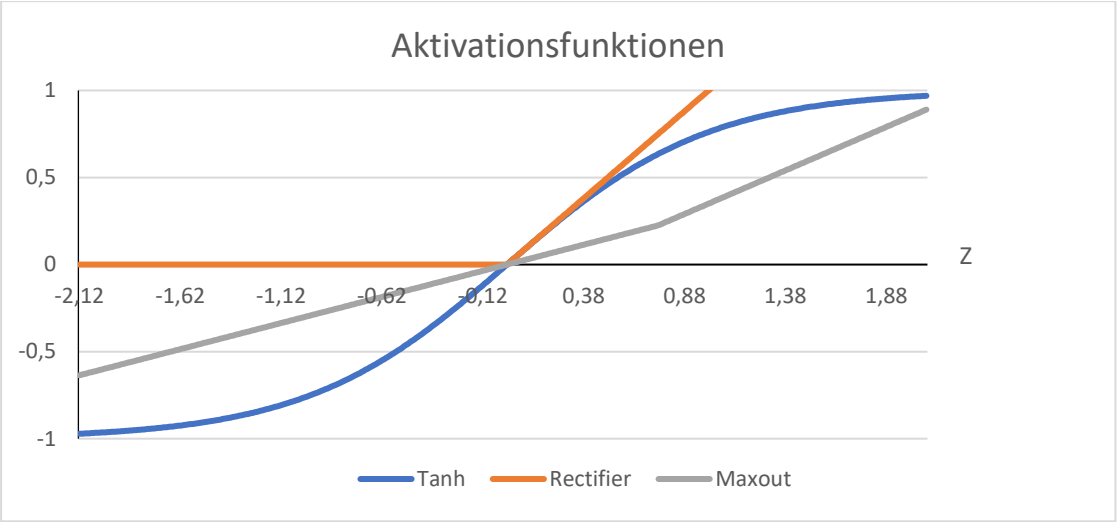

Abbildung 3: Übersicht Aktivationsfunktionen

2.2.3. Lernprozess

Bei den Lernprozessen wird grundsätzlich zwischen überwachten- und unbewachten Lernen unterschieden. Beim überwachten Lernen gibt es Datensätze zu den die gewünschte Prognosewerte bereits vorliegen. Das hier verwendete Lernverfahren zum überwachten Lernen nennt sich Backpropagation.

Bei Backpropagation wird im ersten Schritt ein sogenannter Vorwärtspass durchgeführt. Hierbei werden die Eingabewerte mit den initialen zufällig gesetzten Gewichten durch das neuronale Netz geschleust, die jeweiligen Transformationen mit den Aktivationsfunktionen werden durchgeführt und am Ende, sollte es sich um ein binäres Klassifikationsproblem handeln, wird eine Zahl zwischen 0 und 1 als Wahrscheinlichkeitswert ausgegeben. Da zu diesen Datensätzen der korrekte Wert und auch die Ableitung der verwendeten Aktivationsfunktion bekannt sind ist auch bekannt wie die Gewichte in den Neuronen angepasst werden müssen damit das NN die Realität genauer abbildet. Die Änderung der Gewichte wird im sogenannten Rückwertspass durchgeführt. Wie stark die Gewichte dem aktuellen Datensatz angepasst werden hängt von der sogenannten Learning Rate ab.

Diese Vorwärts- und Rückwertspässe werden sehr oft durchgeführt und somit soll sich das Ergebnis immer näher den Trainingsdaten annähern. Wenn das Modell die Trainingsdaten „zu“ gut abbildet aber keine zukünftigen Prognosen treffen kann wird vom Overfitting gesprochen. Umso mehr Layer und je mehr Neuronen pro Layer desto mehr Zusammenhänge können theoretisch erkannt werden, somit ist es möglich bei fast jedem beliebigen Trainingsset eine komplette Übereinstimmung und somit keine Fehlklassifikation zu haben. Dies ist nicht gewünscht da somit keine oder nur eine geringe Aussagekraft bei zukünftigen Daten und Prognosen existiert.

Um solch ein Overfitting zu vermeiden gibt es mehrere Methoden. Die Reduktion der Layer und Neuronen pro Layer, das setzten eines Grenzwertes nach wie vielen Trainingsrunden gestoppt werden soll oder einen Grenzwert der die minimale Verbesserung pro x Iterationen angibt. Sollte dieser Grenzwert unterschritten werden wird das Training ebenfalls abgebrochen. Eine weitere Methode die auch in dieser Arbeit verwendet wird ist das sogenannte Dropout. Dropout bedeutet, dass beliebige Eingangswerte zufällig auf null gesetzt werden. Somit wird das Modell robuster gestaltet, da es nicht auf die Existenz aller Werte angewiesen ist und somit nicht alle irrelevanten Verhältnisse die im Trainingsset, aber nicht im Testset oder in der Realität, bestehen abbildet. Eine weitere auch hier verwendete Methode ist das Setzten von Limitationen für die Gewichte, welche dafür sorgen können das Gewichte niedrig bleiben oder genullt werden, es sei denn sie tragen maßgeblich zur Prognosegenauigkeit bei.

Als Trainingsset werden die Datensätze bezeichnet welche zum Trainieren, zum Lernen, eines Modelles verwendet werden. Als Testset werden die Datensätze bezeichnet welche zum Testen verwendet werden. Beim Testen soll festgestellt werden, wie genau das Modell vorhersagen bei Daten die das Modell noch nicht gesehen hat, mit denen es noch nicht trainiert hat, treffen kann.

2.2.4. Errorfunktionen

Errorfunktionen dienen der Fehlerberechnung, der Bewertung von NN, damit festgestellt werden kann ob das NN sich von einer Trainingsiteration zur nächsten verbessert oder verschlechtert hat.

In dieser Arbeit wird mit der Logistic Loss Funktion sowie mit der Fehlklassifikationsrate als Errorfunktionen gearbeitet.

$$V(f(\vec{x}), y) = \frac{\ln\left(1 + e^{-y * f(\vec{x})}\right)}{\ln(2)}$$

Formel 5: Logistic Loss Funktion

Die Fehlklassifikationsrate ist die die Wahrscheinlichkeit, dass sich das neuronale Netz bei den vorhersagen irrt.

$$Fehlklassifikationsrate = \frac{Falsch\ Klassifiziert}{Alle\ Klassifikationsfälle}$$

Formel 6: Fehlklassifikationsrate

2.2.5. Restricted Boltzman Machines

RBMs, damals Harmony genannt, wurden bereits im Jahre 1986 von Smolensky Paul konzipiert (Rumelhart und Group 1999, 1999) und im Jahre 2006 durch Geoff Hinton weiter verbreitet (Hinton und Salakhutdinov 2006). RBMs sind NN in welchem es keine Verbindungen zwischen Neuronen im selben Layer gibt. Ein RBM besteht aus einem Input Layer, einem Hidden Layer und einem Output Layer. Die Architektur eines RBMs ist folgender Grafik verdeutlicht.

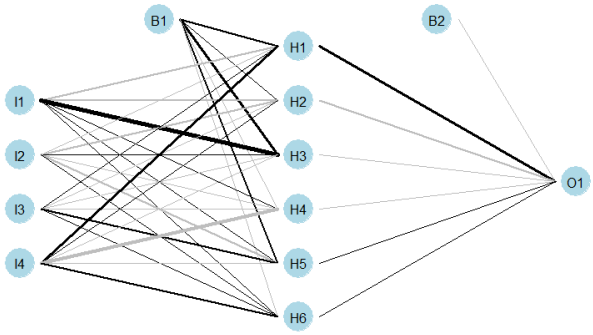

Abbildung 4: 3 Layer RBM

2.2.6. Autoencoder

Ein Autoencoder ist ein NN bestehend aus zwei meistens symmetrischen Deep Belief Nets (DBNs), wobei eines zum kodieren und das andere zum dekodieren genutzt wird. DBNs sind modifizierte RBMs und bestehen aus mehreren Hidden Layern. Ziele von Autoencodern sind unter anderem das Ermöglichen vom unbewachtem Trainieren und der Dimensionalitätsreduktion.

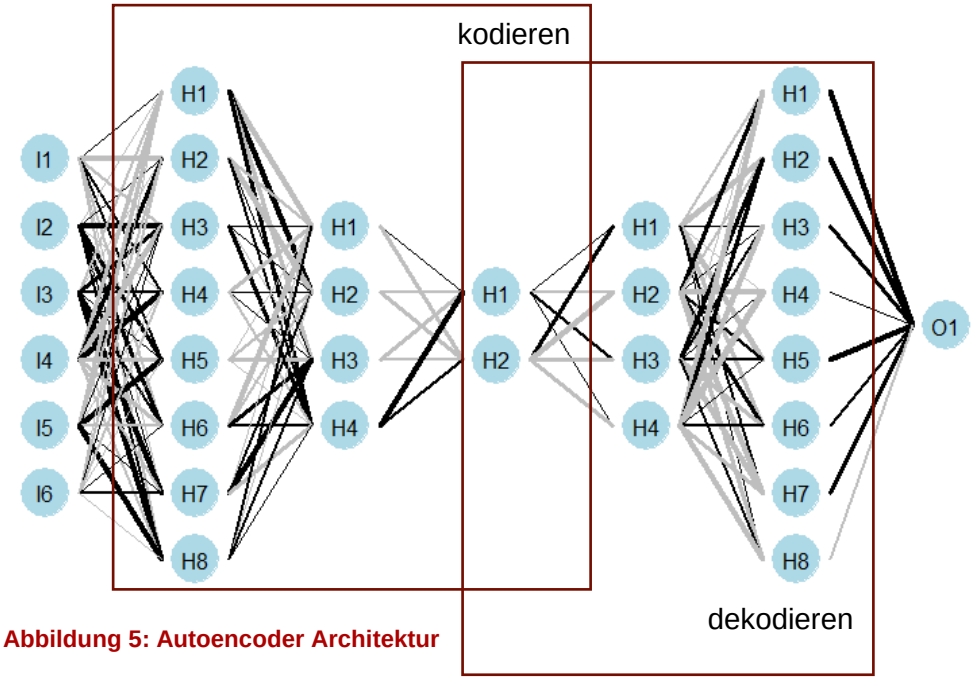

Abbildung 5: Autoencoder Architektur

3. Modell für Kapitalmarktdaten

Folgend werden die Bestandteile des Modelles erläutert. Ziel des Modelles ist es zu bestimmen ob eine gewisse Aktie in der Folgeperiode den Markt übertrifft. Es wird angenommen, dass alleine die Renditekennzahlen der Vergangenheit genügend Aussagekraft besitzen um in einigen Fällen zutreffende Vorhersagen zu treffen. Die theoretische Basis hierfür sind die Arbeiten rund um das Thema des Momentum-Effekts und dem Short- und Long-Term Reversal Effekt (Wu und Mazouz 2016; Geoffrey Booth et al. 2016; Blitz et al. 2013). Der Long-Term Reversal Effekt wird in der Sensitivitätsanalyse berücksichtigt, da die Zeitspanne hierfür erweitert werden muss. In der ursprünglichen Arbeit und im folgenden Modell werden die letzten 20 Tage und die letzten 13 Monate berücksichtigt.

Um den Autoencoder zu entwickeln wird das Packet H2O in R verwendet. Es verfügt über umfangreiche Machine- und Deep Learning Methoden und der Implementierungsaufwand ist im Vergleich zu den verbreiteteren Paketen wie Tensorflow in Python oder Caffe geringer.

3.1. Vorverarbeitung

Die Daten werden bevor sie im NN zum Training verwendet werden vorverarbeitet. Als Eingabewerte zur Vorverarbeitung werden die Tagesendkurse verwendet. Diese Kurse sollten Aktiensplit, Aktienzusammenlegung und Dividendenbereinigt sein. Dividenden werden zum Zeitpunkt der Zahlung reinvestiert. In der Ursprungsarbeit ist nicht ersichtlich ob die Kurse diese Merkmale aufweisen. Die bereinigten Aktienkurse werden als Basis genommen um zum jeweiligen monatlichen Stichtag die kumulierten letzten 20 täglichen und 12 monatlichen Renditen zu ermitteln.

$$Rendite_{i,t} = \frac{Kurs_{i,t} - Kurs_{i,t-1}}{Kurs_{i,t-1}}$$

Formel 7: Rendite

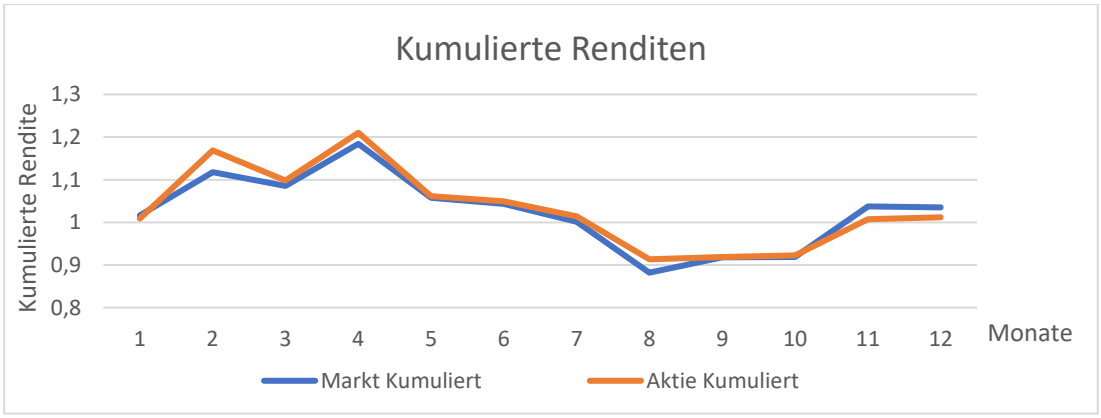

Abbildung 6: Kumulierte Renditen

Für alle Aktien werden jeweils die 20 täglichen und 12 monatlichen kumulierten Renditewerte über den Z Score normalisiert.

$$Z\ Score = \frac{x - \mu}{\sigma}$$

Formel 8: Z Scores

Der Z Score wird anschließend nochmals normalisiert damit alle Eingabewerte zwischen 0 und 1 liegen.

$$Normalisierter\ Z\ Score = \frac{ZScore - \min(ZScores)}{\max(ZScores) - \min(ZScores)}$$

Formel 9: Z Scores Normalisiert

Das NN nimmt als Eingabewerte die normalisierten 12 monatlichen und 20 täglichen Z Scores sowie einen Wert ob es sich bei dem folgenden Monat um den Januar (Wert=1) handelt oder nicht (Wert=0). Um das Modell überwacht zu Trainieren werden Datensätze mit Ausgabewert benötigt. Hierfür werden für den jeweiligen Zeitpunkt die Renditen aller Aktien im nächsten Monat berechnet und alle überdurchschnittlichen Aktien werden mit dem Ausgabewert 1 versehen.

Der Programmcode zur Datenvorverarbeitung befindet sich im Anhang (A Programmcode Datenvorverarbeitung).

3.2. Architektur

Das NN besteht aus einem Input Layer, 3 Hidden Layern und einem Output Layer. Der Input Layer besteht aus 33 Neuronen, eins für jeden Eingabewert. Die Hidden Layer bestehen aus 40, 4 und 50 Neuronen und der Output Layer aus 2 Neuronen, eines für jede Klasse. Die Ursprungsarbeit hat bereits mehrere Variationen getestet und erhielt mit der beschriebenen Architektur die höchste Genauigkeit (Lawrence Takeuchi * und Yu-Ying (Albert) Lee). Es liegen keine genaueren Werte zu den durchgeführten Testläufen vor. Weitere Architekturen, mit variierenden Layer-Größen werden in der späteren Sensitivitätsanalyse betrachtet.

3.3. Hyperparametersuche

Um zu vermeiden unpassende Hyperparameter zu verwendet, wird zunächst eine Hyperparametersuche mit einem initialen Trainingsset durchgeführt. Hyperparameter sind beschreibende Parameter eines NN. Dazu gehören unter anderem die Lernrate, Aktivations- und Errorfunktion, Anzahl der Layer und Neuronen, Input Dropout-Ratios. Eine Auflistung und Beschreibung der verwendeten und modifizierten Hyperparameter befindet sich im Anhang (B Hyperparameter).

Während der Hyperparametersuche werden zufällig verschiedene Kombinationen ausgewählt, mit einem Trainingsset trainiert und mit einem Testset getestet. Die Anzahl der möglichen Kombinationen kann sehr hoch sein. Deswegen werden nicht alle Möglichkeiten getestet, sondern sobald der Ergebniszugewinn nicht höher als eine vorher definierte Grenze ist, wird die Suche gestoppt und es besteht die Möglichkeit die Kombination an Parametern, mit der höchsten Klassifikationsgüte weiterzuverwenden.

Der Code für die durchgeführte Hyperparametersuche befindet sich im Anhang (C Programmcode Hyperparametersuche).

Folgender Optionsspielraum wird bei der Suche für das Modell verwendet:

PARAMETER	OPTIONEN
AKTIVATIONSFUNKTION	Rectifier, Tanh, Maxout, RectifierWithDropout, TanhWithDropout, MaxoutWithDropout
INPUT DROPOUT RATIO	0, 0,02, 0,03, 0,04, 0,05
L1	Sequenz von 0, bis 1e-4, in 1e-6 Schritten
L2	Sequenz von 0, bis 1e-4, in 1e-6 Schritten
ERRORFUNKTION	Logaritmic Loss, Fehlklassifikationsrate

Tabelle 1: Hyperparametersuche

3.4. Iterativer Lernprozess

Das Modell in der Ursprungsarbeit sieht einen längeren zusammenhängenden Trainingszeitraum und einen längeren zusammenhängenden Testzeitraum vor. Dies ermöglicht keine Anpassung der Gewichte im Testzeitraum. Das hier verwendete Modell gleicht diese potentielle Schwachstelle aus. Es sieht einen mittellangen Zeitraum zum initialen Training der Gewichte vor. Anschließend iterative Test- und Trainingsläufe in denen die neuen Erfahrungen und Erkenntnisse immer wieder in das Modell eingebunden werden.

Zunächst wird mit den initialen Trainingsdaten die Hyperparameersuche durchgeführt. Mit den gefundenen Parametern wird anschließend der Autoencoder erstellt und stufenweise unbewacht trainiert. Der Programmcode für die Funktion welche den Autoencoder erstellt und stufenweise trainiert befindet sich im Anhang (D Programmcode Funktion für den gestapelten Autoencoder). Anschließend wird der Autoencoder mittels Backpropagation überwacht trainiert. Anschließend startet der iterative Trainings- und Testvorgang. Somit werden die neuen Erfahrungen immer wieder in das Modell eingearbeitet und neue Rahmenbedingungen können genauer abgebildet werden.

4. Empirischer Backtest des Modells

Folgend wird der Backtest des im Kapitel 3 beschriebenen Modelles für den deutschen Aktienmarkt durchgeführt und das entstandene Modell analysiert. Der Programmcode für die Datenvorverarbeitung, den iterativen Trainings- und Testprozess sowie die Portfoliooptimierung und Ergebnisberechnung ist auf folgendem GitHub Repository zu finden.

https://github.com/Python35/DeepLearningFinance

4.1. Daten

Es werden die Aktien aus Dax und MDax betrachtet. Somit stehen jeweils 90 bis 2002 und 70 Aktien bis November 2016 zur Verfügung. Sollte eine Aktie aus den Indizes herausgenommen oder hinzugefügt werden wird sie in der nächsten Kaufentscheidung nicht mehr betrachtet, weswegen zwangsweise gehaltene Anteile verkauft werden. Zudem müssen mindestens Kursdaten der letzten 13 Monate vorliegen. Die Daten wurden mittels dem Bloomberg Excel Add-In beschafft. Diese exportierten Daten sind bereits Aktiensplit, Aktienzusammenlegung und Dividenden bereinigt da das Feld „TOT_RETURN_INDEX_GROSS_DVDS“ in Bloomberg für den Export ausgewählt wurde und dies beinhaltet. Im Anhang befindet sich die Beschreibung von Bloomberg zu diesem Feld (E Beschreibung Feld Bloomberg). Es wurden alle Kursdaten der Aktien die jemals im Dax oder MDax waren exportiert, somit stehen zum jeweiligen Zeitpunkt mehr als die 90 bzw. 70 Aktien zum Trainieren und Testen zur Verfügung. Dies ist gewünscht um die Anzahl der Trainingsfälle zu erhöhen, denn grundsätzlich wird mit jedem zusätzlichen Trainingsfall das NN genauer. Dies vermeidet Overfitting von zu kleinen Trainingssets. Da 90 bzw. 70 Trainingsfälle pro Monat nicht viel sind konnte die Anzahl somit erhöht werden. Mit diesen Einschränkungen waren 6644 Datensets im Zeitraum vom 01.03.1993 bis 31.12.1997 im initialen Trainingsdatenset und 29334 Datensets im Zeitraum vom 01.01.1998 bis 31.10.2016 im iterativen Trainings- und Testdatenset.

Mit den initialen Trainingsdaten bis Dezember 1997 wurde die Hyperparametersuche durchgeführt. Mit folgenden Parametern wurden die genauesten Vorhersagen nach Logistic Loss erzielt.

PARAMETER	OPTION
AKTIVATIONSFUNKTION	Tanh
INPUT DROPOUT RATIO	0,03
L1	5,9e-5
L2	2,9e-5

Tabelle 2: Hyperparameter Autoencoder

Es wurde ein Logistic Loss von 0,6919 erzielt. Mit diesen Hyperparametern wurde dementsprechend der Autoencoder für das unbewachte Trainieren und das anschließende iterative Trainieren und Testen erstellt. Folgend die Ergebnisse.

4.2. Ergebnisse

Bei den Ergebnissen wird unterschieden zwischen der Klassifikationsgüte des NN, welche sich durch verschiedene Kennzahlen wie z.B. der Treffergenauigkeit ausdrückt und der Ergebnisse der gebildeten Portfolien auf Basis der Vorhersagen des Netzes. Bei den Ergebnissen der Portfolios werden neben der Rendite und der Varianz Kennzahlen zur Erklärung der Über- oder Unterrendite, z.B. Sortino Ratio und Jensen Alpha betrachtet.

4.2.1. Ergebnisse des Netzes

In der Ursprungsarbeit erzielt das NN, welches auf Aktien im NYSE, AMEX und Nasdaq angewendet wird, die in Tabelle 3 dargestellte Wahrheitsmatrix. Eine Wahrheitsmatrix verdeutlicht die Klassifikationsgüte in dem es die Vorhersagewerte mit den realen Werten in einer Matrix darstellt. Es wurde eine Treffergenauigkeit von 53,352% erzielt.

		Vorhergesagt	
		0	1
Real	0	22,38%	27,45%
	1	19.19%	30,97%

Tabelle 3: Wahrheitsmatrix Ursprungsarbeit[3]

Das replizierte neuronale Netz ohne konstantes Updaten der Gewichte im Testzeitraum erzielt für die Daten des deutschen Aktienmarktes die in Tabelle 4 dargestellte Wahrheitsmatrix. Dies entspricht einer Treffergenauigkeit von 50,546%. Diese ist geringer als in der Ursprungsarbeit angegeben. Mögliche Gründe für diese Diskrepanz werden in Kapitel 4.4 Diskussion betrachtet.

		Vorhergesagt	
		0	1
Real	0	25,319%	24,949%
	1	24,505%	25,227%

Tabelle 4: Wahrheitsmatrix Ursprungsarbeit d. Aktienmarkt

[3] In enger Anlehnung an Lawrence Takeuchi * und Yu-Ying (Albert) Lee

Das replizierte neuronale Netz mit konstantem trainieren der Gewichte im Testzeitraum erzielt für die Daten des deutschen Aktienmarktes die in Tabelle 5 dargestellte Wahrheitsmatrix. Dies entspricht einer Treffergenauigkeit von 50,737%. Hier lässt sich eine Verbesserung von 0,378% feststellen.

		Vorhergesagt	
		0	1
Real	0	25,415%	24,409%
	1	24,853%	25,322%

Tabelle 5: Wahrheitsmatrix konstantes Update

Die Hauptfrage dieser Arbeit beschäftigte sich mit dem konstanten Updaten der Gewichte und ob es die Treffergenauigkeit verbessern kann. Es wurde eine Verbesserung von 0,378% erreicht. Eine Begründung hierfür war die Veränderung in den Marktbedingungen. Demzufolge müsste die Treffergenauigkeit des NNs ohne konstantes trainieren im Zeitverlauf niedriger werden. Dies ist jedoch wie in folgendem Diagramm zu sehen nicht der Fall. Die Steigungen zu den beiden dargestellten Verläufen sind mit 0.000002 für das konstante updaten und 0.0000017 für die konstanten Gewichte nicht signifikant.

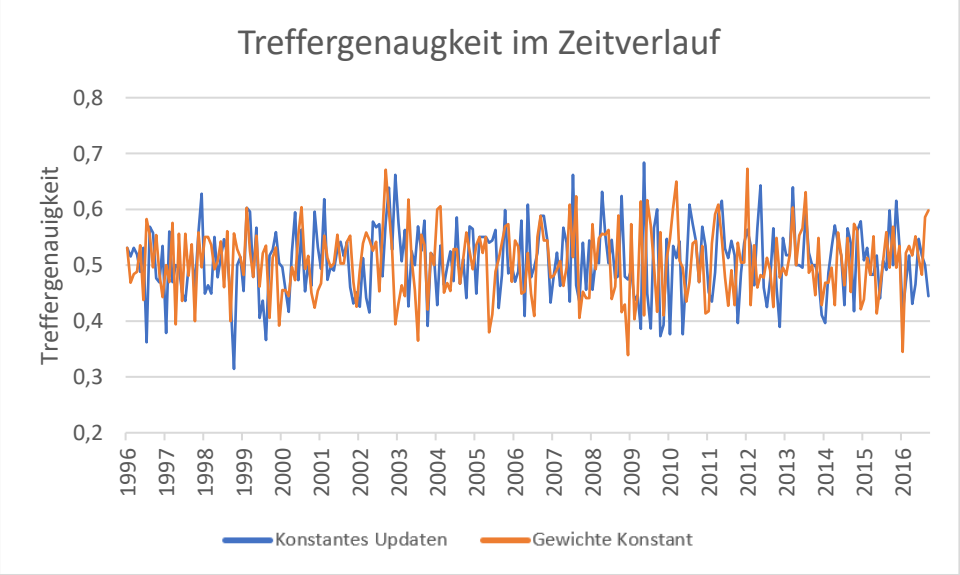

Abbildung 7: Treffergenauigkeit im Zeitverlauf

Für das replizierte NN mit konstanten updaten der Gewichte beträgt die durchschnittliche Treffergenauigkeit 50,737% mit einer Varianz von 0.00377. Folgend wird die Wahrscheinlichkeitsdichtefunktion für den Testzeitraum dargestellt.

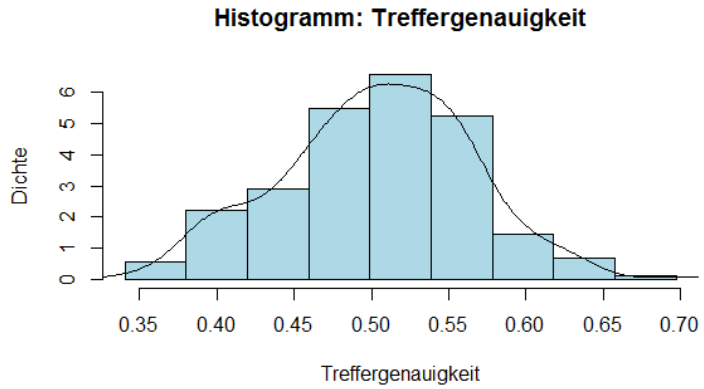

Abbildung 8: Histogramm Treffergenauigkeit

Die Wahrheitsmatrix und Treffergenauigkeit sind Werkzeuge um Ergebnisse eines NNs zu bewerten. Die Aussagekraft und Relevanz der Wahrheitsmatrix ist für das vorgestellte Modell nicht so bedeutend wie die folgend vorgestellte „Top 20 Treffergenauigkeit“ Kennzahl.

Während dem Testen wird den Aktien ein Wahrscheinlichkeitswert zugewiesen mit welcher das NN vorhersagt ob die Aktien den Markt im Folgemonat übertreffen oder nicht. Es gibt viele Aktien denen als Wahrscheinlichkeitswert 45-55% zugewiesen wird. Diese werden in einer Wahrheitsmatrix ebenfalls dargestellt. Jedoch wird in diese nicht investiert, da in diesem Backtest nur in die höchsten 20 Wahrscheinlichkeitswerte investiert wird. Aus diesem Grund wurde die Kennzahl „Top 20 Treffergenauigkeit“ eingeführt. Diese ist wie folgt während dem Testzeitraum verteilt. Der Mittelwert dieser Kennzahl liegt bei 51,01 % mit einer Varianz von 0,01325.

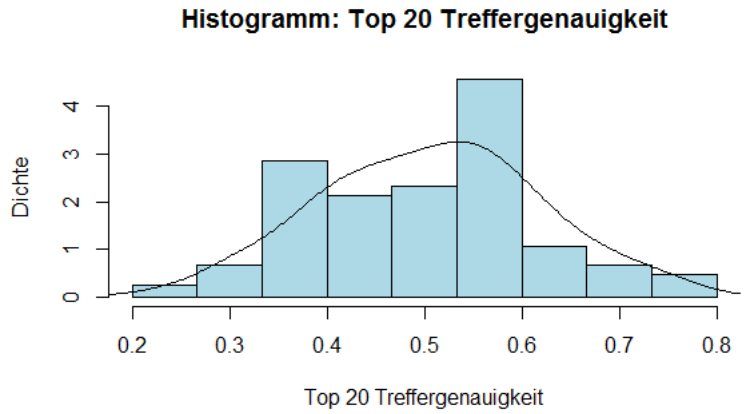

Abbildung 9: Histogramm Top 20 Treffergenauigkeit

Zwischen der Treffergenauigkeit und der „Top 20 Treffergenauigkeit“ besteht ein Zusammenhang da die Top 20 Treffergenauigkeit ein Teil der (Gesamt-) Treffergenauigkeit ist. Zwischen diesen Kenngrößen besteht ein Korrelationskoeffizient von 0,4626.

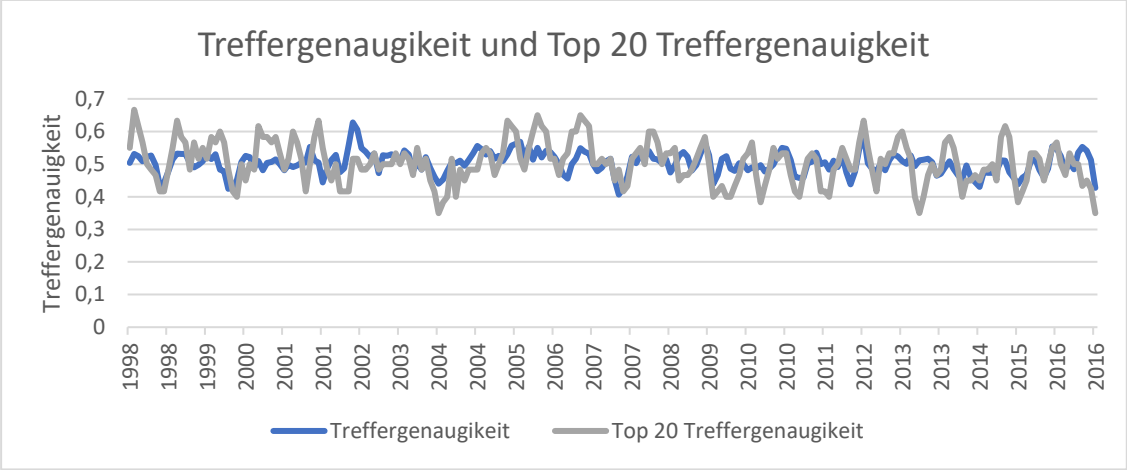

Abbildung 10: Treffergenauigkeit im Zeitverlauf

Das vorgestellte Modell soll derzeit Aktien welche minimal niedrigere Renditen von Aktien welche minimal höhere Renditen im Folgemonat erzielen differenzieren und anderen Klassen zuordnen auch wenn diese in der Kurshistorie kaum oder gar keine Unterschiede aufweisen. Dies führt dazu, dass die Gewichte im NN diesen Sachverhalt abbilden sollen und dementsprechend geändert werden. Ein NN mit 3 Ausgabeklassen wird in der Sensitivitätsanalyse betrachtet, welches dieses Problemlösen soll. Dies könnte die Treffergenauigkeit erhöhen.

4.2.2. Ergebnisse der Portfolios

Zur Beurteilung einer Anlagestrategie, welche nach den Ergebnissen des NNs seine Handelsentscheidungen trifft, ist die Bildung von Portfolien nötig, weswegen für diesen Backtest drei Portfolien und zwei Benchmarks gebildet wurden. Es wurde ein gleichgewichtetes Portfolio, ein Minimum-Varianz-Portfolio und ein Portfolio mit den NN-Wahrscheinlichkeitswerten als Gewichtungen gebildet.

Um eine mögliche Über- oder Unterrendite der Portfolien im Vergleich zu den Benchmarks zu erklären wurde neben der Rendite und Varianz folgende Kennzahlen berechnet.

$$Sharpe\ Ratio = \frac{\bar{R}_{PF} - R_f}{\sigma(R_{PF})}$$

Formel 10: Sharpe Ratio

$$Information\ Ratio = \frac{\overline{R_{PF}} - \overline{R_{Benchmark}}}{\sigma(R_{PF} - R_{Benchmark})}$$

Formel 11: Information Ratio

$$Sortino\ Ratio = \frac{\overline{R_{PF}} - R_{min}}{\sqrt{\frac{1}{T} * \sum_{t=1}^{T} \left(\min\left(R_{PF,t} - R_{min}, 0\right)^2\right)}}$$

Formel 12: Sortino Ratio

$$\alpha_{Jensen} = \overline{R_{PF}} - R_f - \beta * (\overline{R_M} - R_f)$$

Formel 13: Jensen Alpha

$$\alpha_{Fama/French} = \overline{R_{PF}} - R_f - \beta * \left(\overline{R_M} - R_f\right) - s * \overline{SMB} - h * \overline{HML}$$

Formel 14: Fama/French Alpha

$$\alpha_{Carhart} = \overline{R_{PF}} - R_f - \beta * \left(\overline{R_M} - R_f\right) - s * \overline{SMB} - h * \overline{HML} - m * \overline{MOM}$$

Formel 15: Carhart Alpha

Zusätzlich wird der Anteil der Verlustmonate und der MaximumDrawdown über ein Jahr betrachtet.

Formel 16: Maximum Drawdown

$$Drawdown_t = \frac{p_t}{\max\limits_{\forall t}(p_t)} - 1$$

$$Max\ Drawdown = \max\limits_{\forall t}(Drawdown_t)$$

Übergreifend haben die Aktien in den Top 20 jeweils 0,964% Rendite im Folgemonat erzielt und die Aktien außerhalb jeweils 0,811%. In allen gebildeten Portfolien wurden Grenzwerte bei der Gewichtung festgelegt. Eine Aktie darf nicht mit mehr als 15% und nicht mit weniger als 2% im Portfolio gewichtet sein.

Der Programmcode zur Portfoliobildung und Ergebnisberechnung befindet sich ebenfalls im Anhang (F Programmcode Portfoliobildung).

Eine Übersicht mit allen folgenden Kennzahlen in Tabellenform befindet sich im Anhang (G Kennzahlen Übersicht).

4.2.2.1. Ergebnisse des gleichgewichteten Portfolios

Das gleichgewichtete Portfolio erzielt eine jährliche Rendite von 10,25% bei einer Varianz von 0,00320. Während hingegen die Benchmark eine jährliche Rendite von 8,69% bei einer Varianz von 0,00325 erzielt.

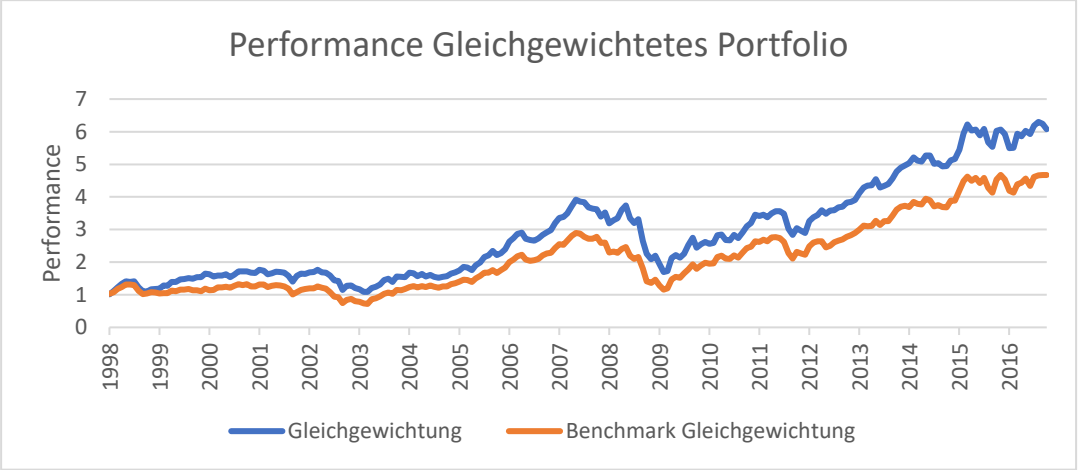

Abbildung 11: Gleichgewichtetes Portfolio

Es ist zu erkennen, dass das gebildete gleichgewichtete Portfolio eine höhere Rendite aufweist als die Benchmark, in der alle im Dax und MDax befindlichen Aktien gleichgewichtet sind.

	Gleichgewichtetes Portfolio	Benchmark Gleichgewichtet
Sharpe Ratio	0,166	0,145
Information Ratio	0,055	
Sortino Ratio	0,737	0,609
Anteil Verlustmonate	38,5%	41,2%
Anteil Verlustjahre	27,8%	22,2%
Max-Drawdown 1 Jahr	54,7%	53,1%
Jensen Alpha	0,005	0,004
Fama French Alpha	0,005	0,003
Carhart Alpha	0,006	0,005

Tabelle 6: Kennzahlen Gleichgewichtes Portfolio

Die Alpha Faktoren sind bei dem Portfolio nur schwach ausgeprägt. Der Anteil der Verlustmonate ist beim gleichgewichteten Portfolio 2,7 Prozentpunkte niedriger und der Anteil der Verlustjahre ist hingegen 5,6 Prozentpunkte höher als bei der Benchmark. Der Maximum Drawdown weist ebenfalls nur geringe Unterschiede auf. Die Information Ratio ist im Vergleich zu professional geführten Portfolien mit 0,055 sehr niedrig. Diese weisen in der Regel über einen längeren Zeitraum eine Information Ratio von 0,2 bis 0,6 auf (O.V.). Der Fonds „Lupus alpha Smaller German Champions“, weist für die Jahre Juni 2008 bis Juni 2011 einen Information Ratio von 0,81 auf (Bennewirtz).

4.2.2.2. Ergebnisse des NN-Ergebnis gewichteten Portfolios

Dieses Portfolio bildet am stärksten die potenziale des NNs ab, da Aktien mit höherem Wahrscheinlichkeitswert in diesem Portfolio stärker berücksichtigt werden. Da die Unterschiede der Wahrscheinlichkeitswerte von den Top 20 Aktien gering ausfallen wird als Benchmark das gleichgewichtete Portfolio angesetzt.

Das NN-Ergebnis gewichtete Portfolio erzielt eine jährliche Rendite von 10,5 % bei einer Varianz von 0,00319. Die Benchmark erzielt eine Rendite von 8,69 % bei einer Varianz von 0,00325.

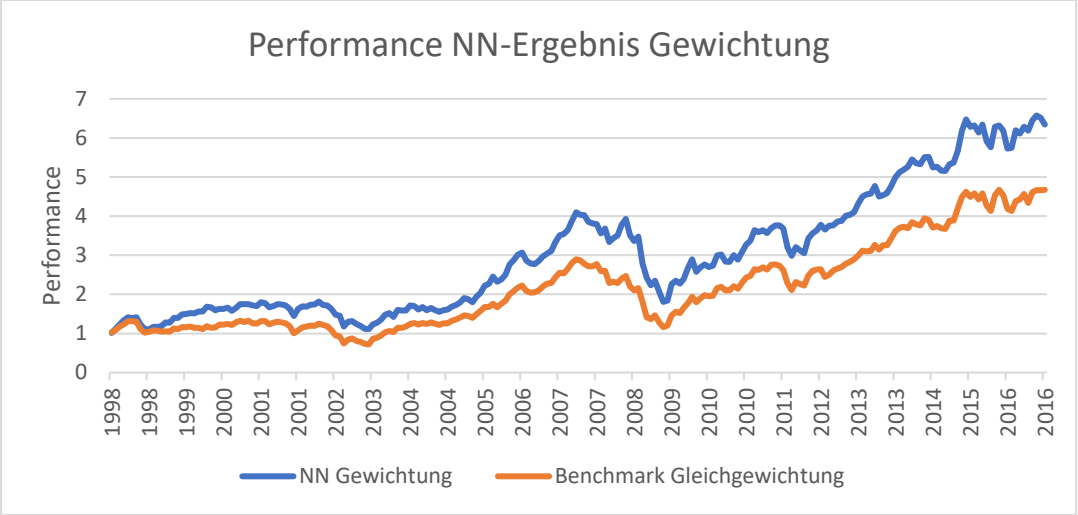

Abbildung 12: NN-Ergebnis Gewichtung

	NN-Ergebnis gewichtetes Portfolio	Benchmark Gleichgewichtet
Sharpe Ratio	0,169	0,145
Information Ratio	0,044	
Sortino Ratio	0,767	0,609
Anteil Verlustmonate	38,5%	41,2%
Anteil Verlustjahre	27,8%	22,2%
Max-Drodown 1 Jahr	54,0%	53,1%
Jensen Alpha	0,005	0,004
Fama French Alpha	0,005	0,003
Carhart Alpha	0,006	0,005

Tabelle 7: Kennzahlen NN-Ergebnis gewichtetes Portfolio

Die Alpha Faktoren sind bei dem Portfolio nur schwach ausgeprägt. Der Anteil der Verlustmonate ist beim gleichgewichteten Portfolio 2,7 Prozentpunkte niedriger und der Anteil der Verlustjahre ist hingegen 5,6 Prozentpunkte höher als bei der Benchmark. Der Maximum Drawdown weist ebenfalls nur geringe Unterschiede auf.

4.2.2.3. Ergebnisse des Minimum-Varianz-Portfolios

Ein Minimum-Varianz-Portfolio wird basierend auf den historischen Korrelationen so Gewichtet, dass es die geringste Varianz aufweisen soll.

Das Minimum-Varianz-Portfolio erzielt eine jährliche Rendite von 10,73% bei einer Varianz von 0,00225 während hingegen die Benchmark eine Rendite von 9,61% bei einer Varianz von 0,0015 erzielt.

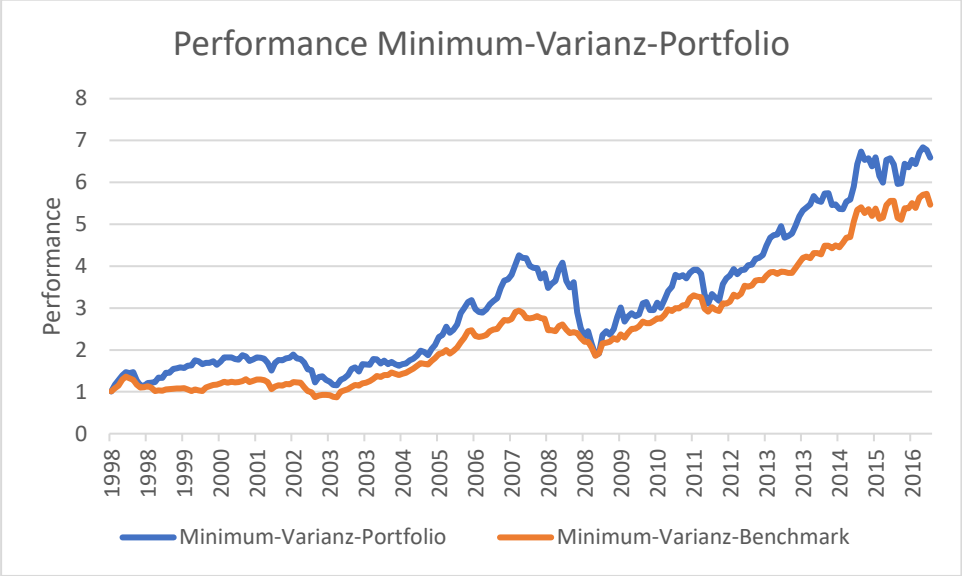

Abbildung 13: Minimum-Varianz Portfolio

	Minimum-Varianz-Portfolio	Benchmark Minimum-Varianz
Sharpe Ratio	0,207	0,208
Information Ratio	0,072	
Sortino Ratio	0,923	1,127
Anteil Verlustmonate	39,3%	36,4%
Anteil Verlustjahre	27,8%	22,2%
Max-Drodown 1 Jahr	53,1%	33,1%
Jensen Alpha	0,004	0,009
Fama French Alpha	0,003	0,011
Carhart Alpha	0,005	0,005

Tabelle 8: Kennzahlen Minimum-Varianz-Portfolio

Die höhere Rendite vom Minimum-Varianz-Portfolio ist mit einer höheren Varianz verbunden. Nach der Sortino Ratio hat die Benchmark eine besseres Rendite-Risiko-Verhältnis. Der Maximum Drawdown weist ebenfalls einen großen Unterschied auf, dieser ist beim gebildeten Portfolio 20% Prozentpunkte höher als bei der Benchmark.

4.3. Analyse des Modelles

Ziel dieses Kapitels ist es, dass entstandene NN zu verstehen und in einem gewissen Rahmen zu interpretieren.

Aufgrund der Architektur von NN und den vielen nichtlinearen Transformationen gibt es wenige Ansätze zur Analyse von NN. Der Autor hat sich demnach zwei Fragen gestellt die, mit diesem Kapitel beantwortet werden. Zum einen wie variabel ist das Modell, wie stark haben sich die Gewichte im Zeitverlauf verändert, und zum anderen welche Konstellation von Eingabewerten sieht das NN als Anzeichen für Überrendite an.

Zur Analyse der Variabilität wurde eine Kennzahl zur Veränderung der Gewichte konzipiert. Diese Kennzahl beinhaltet die durchschnittliche prozentuale Veränderung aller Gewichte pro Iteration. Folgend eine Grafik die diese Kennzahl und den Dax im Zeitverlauf darstellt.

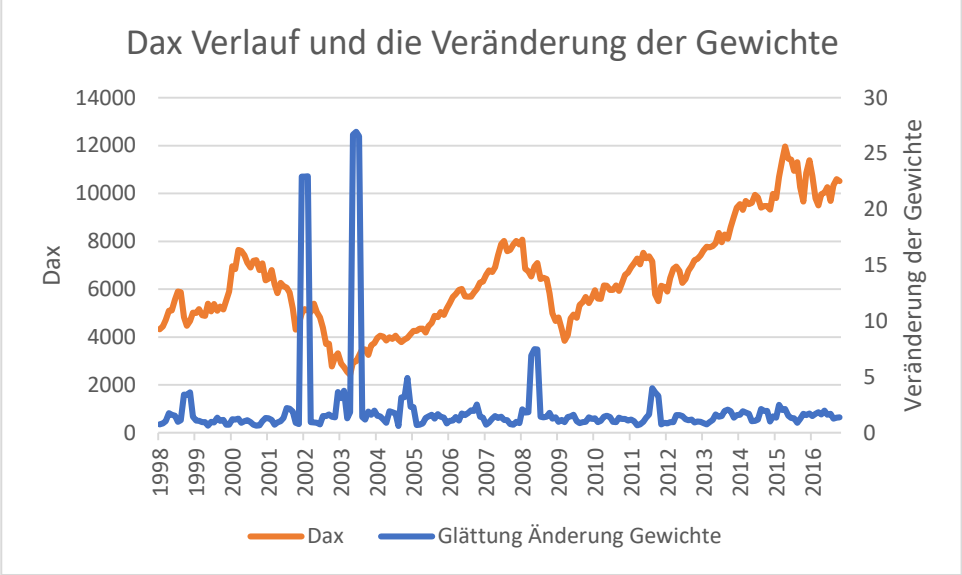

Abbildung 14: Dax Verlauf und Veränderung der Gewichte

Es ist zu erkennen das die Änderungen der Gewichte besonders hoch bei stark volatilen Kursen sind, bzw. bei Trendwechsel. Dies würde dafürsprechen, dass sich das NN mit konstanten updaten der Gewichte sich schneller an neue und kurzzeitige Marktphasen anpassen kann denn genau in diesen Zeiträumen erzielte das NN mit konstanten updaten höhere „Top 20 Genauigkeiten“ von 52,3% zu 48,7% im Vergleich zum Modell ohne konstantes updaten der Gewichte. Aufgrund der geringen Anzahl der kurzzeitigen Marktphasen, 13 Monate mit Veränderungsraten von über 400% pro Gewicht, kann es sich bei diesem Phänomen um Einzelfälle handeln.

Allerdings lässt sich sagen, dass die Gewichte im NN durchgehend stark verändert werden, da durchgehend eine Veränderungsrate von durchschnittlich 183% pro Gewicht pro Iteration festgestellt wurde.

Die zweite Frage dieses Kapitels beschäftigt sich mit den Eingangswerten, dem Kursverlauf der Aktie. Es geht darum zu analysieren welchen Kursverlauf das trainierte NN als positiv im Sinne der Überrendite deutet, hierfür wurden stichprobenartig Datensätze mit sehr hohen und sehr niedrigen Wahrscheinlichkeiten zur Überrendite betrachtet. Folgend ein Datensatz mit einer hohen Wahrscheinlichkeit zur Überrendite.

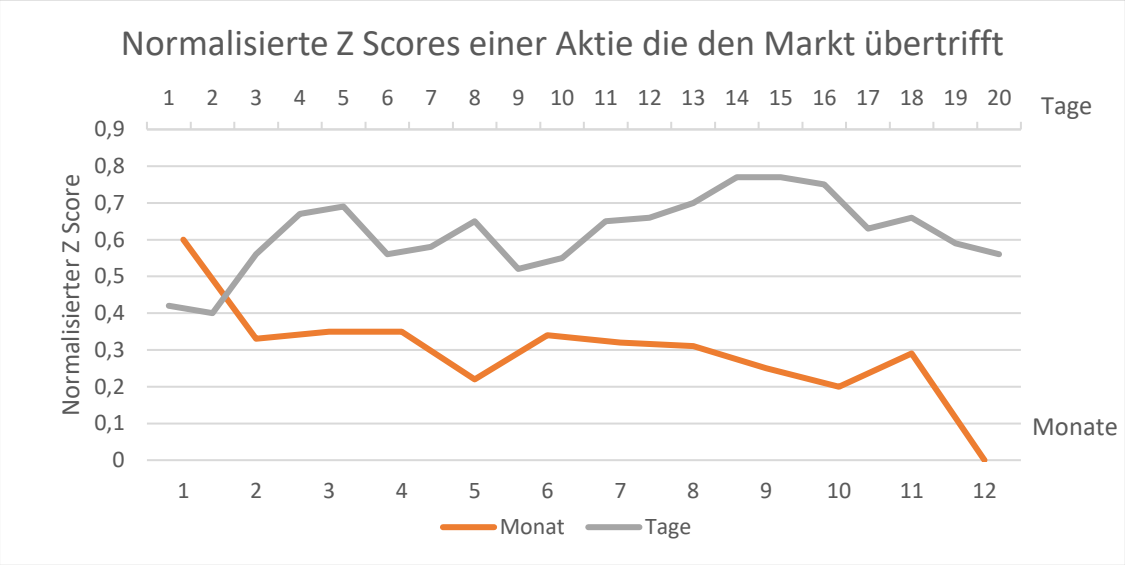

Abbildung 15: Normalisierte Z Scores einer Aktie die den Markt übertrifft

Ein normalisierter Z Score kleiner als 0,5 heißt, dass die Aktie in dieser Periode den Markt nicht übertroffen hat. Demnach handelt es sich bei dieser Aktie um eine die in den letzten 12 bis letzten Monat den Markt nicht übertroffen hat und einen Abwertstrend aufweist, allerdings nun vom NN mit einer ziemlich hohen Wahrscheinlichkeit von 83,7% in Klasse 1 klassifiziert wurde. Der theoretische Ansatz für diese Anomalie ist der Short-Term Reversal Effekt, der besagt, dass eine Aktie den Trend von vor 3-6 Monaten umkehrt (Blitz et al. 2013). Auch wenn bei diesem Effekt von Aktienkursen und nicht von Aktienkursen in Relation zu einem Markt gesprochen wird lässt sich die Hypothese aufstellen, da dieser Effekt bei 7 von 10 Stichproben, welche in der Folgeperiode den Markt übertroffen haben, festgestellt wurde.

Eine weitere Auffälligkeit ist die überdurchschnittliche Entwicklung in den letzten 20 Tagen, welche ebenfalls dazu beiträgt, dass diese Aktie als Klasse 1 klassifiziert wurde. Dies ist mit dem Momentum-Effekt zu erklären, welcher besagt, dass eine Aktie einen kurzfristigen Trend der letzten 2-4 Wochen fortsetzt (McInish et al. 2008; Gong et al. 2015).

4.4. Diskussion

Bei den vorgestellten Kennzahlen der Portfolien wurden keine Transaktionskosten berücksichtigt. Durch die durchschnittlich hohen Turnover Raten von 70% bis 80% pro Monat bei allen drei Portfolien, würden die erzielten Überrenditen bei geringen Transaktionskosten von 0,1% vom Transaktionsvolumen bereits vollständig gedeckt sein und somit würde die Gesamtrendite inkl. Transaktionskosten niedriger als die er Benchmarks ausfallen.

Die unterschiedlichen Ergebnisse des replizierten Modelles im Vergleich zur Ursprungsarbeit lassen sich durch die derzeit vorhandenen Ergebnisse nicht komplett erklären. Mögliche Gründe können die unterschiedliche Datengrundlagen und die nicht ausreichenden Details in der Ursprungsarbeit sein. Es wurden nicht alle nötigen Hyperparameter benannt um das Modell 1:1 zu replizieren.

Auch wenn alle Hyperparameter spezifiziert sind und die Datengrundlage identisch ist werden weitere Details benötigt. Das Aufteilen des Datensets in Trainings- und Testdaten erfolgt i.d.R. zufällig und wenn ein NN mit anderen Daten trainiert wird liefert es andere Ergebnisse. Dies lässt sich in R über den „seed“ steuern. Der seed beeinflusst den internen Zufallsmechanismus um eine Replizierbarkeit zu gewährleisten. Dies reicht jedoch nicht für den Lernprozess in H2O, selbst mit gesetzten seed war die Replizierbarkeit nicht gewährleistet. Eine Replizierbarkeit ist nach H2O Dokumentation mit dem Parameter „reproducible“ möglich, in diesem Modus nutzt H2O jedoch nur einen Kern und würde dementsprechend lange für die Berechnungen benötigen. Dementsprechend wurde dieser Parameter nicht aktiviert (O. V. H2O Documentation).

Während dem Backtest wurden durchgehend hohe Varianzen bei der Treffergenauigkeit und auch der „Top 20 Treffergenauigkeit“ festgestellt. Dies kann unter anderem an den monatlichen Trainings- und Testzyklen und den damit einhergehenden kleinen Datensätzen pro Iteration und der Neigung zum Overfitting von kleinen Datensätzen liegen. Evtl. wäre ein Modell mit Jahreszyklen stabiler.

Aufgrund der erhöhten „Top 20 Treffergenauigkeit“ im Vergleich zur Treffergenauigkeit kann angenommen werden, dass durch das 2 Klassen Modell, welche sich nicht genügend unterscheiden Rauschen im Modell entsteht, da es Aktien unterschiedlich klassifizieren soll, obwohl keine Unterschiede in den Datensätzen vorhanden sind. Diese Annahme wird im Kapitel 5.2 Veränderung der Datenvorverarbeitung näher betrachtet, in dem ein Modell mit 3 Klassen erstellt wird.

Im Kapitel 5.3 Veränderung des Zeitraumes wird ein Modell mit einer erweiterten Zeitspanne erstellt um zu ermitteln ob die Betrachtung der Zeitspanne für den Long-Term Reversal Effekt die Treffergenauigkeit des Modelles erhöhen kann.

5. Sensitivitätsanalyse

Folgend werden Veränderungen an dem Modell vorgenommen um zu ermitteln welche Konsequenzen diese Änderungen mit sich ziehen. Es werden verschiedene Veränderungen betrachten. Es wird unter anderem eine Veränderung der Architektur des NNs betrachten, was passiert, wenn sich die Anzahl der Neuronen je Layer ändern oder die Anzahl der Layer selbst.

Zusätzlich wird eine Veränderung in der Datenvorverarbeitung betrachtet und ein Modell mit 3 Klassen erstellt um die in Kapitel 4 angesprochenen Rauscheffekte zu vermindern.

Abschließend wird eine Veränderung des betrachteten Zeitraumes getestet, in diesem Fall könnten sogar die angesprochenen Long-Term Reversal Effekte erkennbar sein.

Bei den folgenden Modellen wurden die Hyperparameter von vorherigen Modell verwendet.

PARAMETER	OPTION
AKTIVATIONSFUNKTION	Tanh
INPUT DROPOUT RATIO	0,03
L1	5,9e-5
L2	2,9e-5

Tabelle 9: Hyperparameter Autoencoder

5.1. Architekturveränderung

In Tabelle 9 sind die Architekturvarianten die getestet wurden ersichtlich. Die Zahlen in der zweiten Spalte beziehen sich auf die Anzahl der Neuronen je Layer. Beispielsweise hat Variante Nr. 2 3 Hidden Layer mit 20 Neuronen im ersten und dritten Hidden Layer und 4 Neuronen im 2 Layer. Variante Nr. 5 ist die ursprüngliche Architektur.

NR.	ARCHITEKTUR-VARIANTE	TREFFERGENAUIG-KEIT	TOP 20 TREFFERGENAUIGKEIT	LOGISTIC LOSS
1	50, 24, 4, 24, 50	50,81%	51,71 %	0,6937
2	20,4 20	50,64%	50,28 %	0,6959
3	30,30	50,23%	49,85 %	0,7093
4	34,34,34	50,38%	50,76 %	0,7104
5	40,4,50	50,73%	51,01 %	0,6926

Tabelle 10: Kennzahlen Architekturveränderungen

Wie ersichtlich übertrifft der Autoencoder mit 5 Hidden Layern die ursprüngliche Variante. Mit 5 Hidden Layer, insbesondere 3 Layer zum Kodieren und Enkodieren, können die Muster in den Datensätzen genauer extrahiert werden.

5.2. Veränderung der Datenvorverarbeitung

In diesem Kapitel wurde ein Modell mit 3 Klassen getestet. Klasse 2 entspricht den Aktien die im Folgemonat nach der erzielten Rendite zu den besten 20% gehören. Klasse 0 entspricht den schlechtesten 20% und Klasse 1 den 60% in der Mitte.

		Vorhergesagt		
		0	1	2
Real	0	1514	3173	1257
	1	3388	10622	3373
	2	1367	3258	1265

		Vorhergesagt		
		0	1	2
Real	0	5,18%	10,86%	4,30%
	1	11,60%	36,36%	11,54%
	2	4,68%	11,15%	4,33%

Tabelle 11: Wahrheitsmatrix 3 Klassen

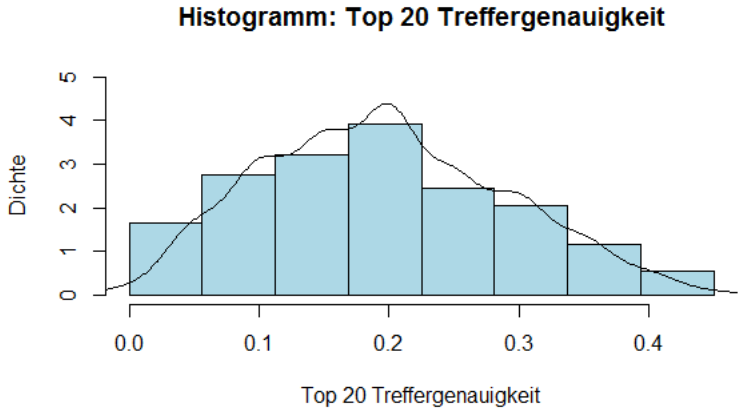

Abbildung 16: Histogramm Top 20 Treffergenaui gkeit 3 Klassen

Der Mittelwert bei der „Top 20 Treffergenauigkeit“ liegt bei 19,58%, dies ist vergleichsweise schlechter als Zufällig zu Raten welche Aktien zu den Top 20% gehören werden. Auch die Performance ist wie in Abbildung 16 zu erkennen niedriger als das 2 Klassen Modell.

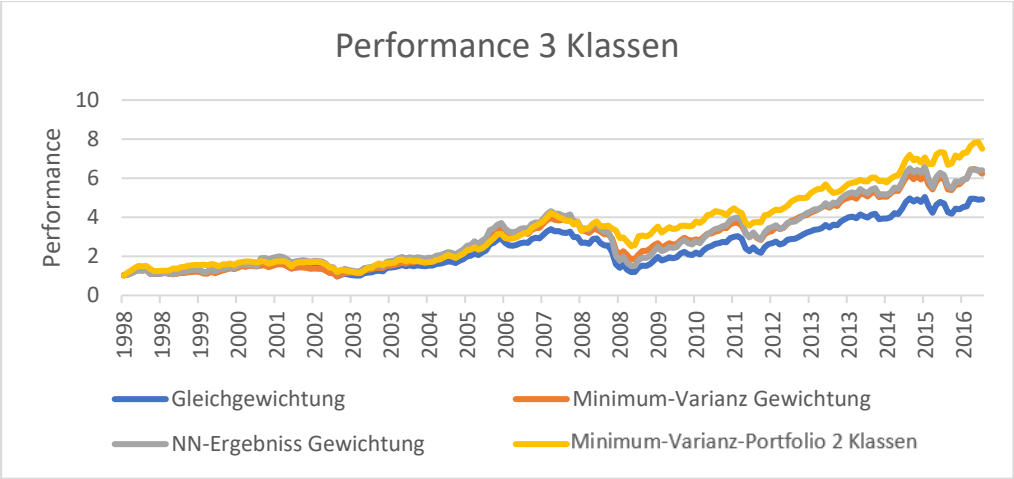

Abbildung 17: Performance 3 Klassen

5.3. Veränderung des Zeitraumes

Weitere Aussagekräftige Daten können die Prognosegenauigkeiten von NNs erhöhen, daher ist die Verlängerung des Betrachtungshorizonts eine Möglichkeit die Treffergenauigkeit zu erhöhen. Die Erweiterung des Zeitraums schränkt die Daten weiter ein, da zu dieser Aktie auch der komplette Zeitraum vorliegen muss wenn keine Substitution der fehlenden Werte durchgeführt wird.

In diesem Modell wird ein weiterer Betrachtungshorizont hinzugefügt, damit Long-Term-Reversal Effekte abgebildet werden können. Hierzu werden die Monate t-60 bis t-37 zusätzlich betrachtet und mit der beschriebenen Methode in Kapitel 3.1 vorverarbeitet. Dementsprechend hat das Modell 24 weitere Eingabewerte. Hierfür wurde eine größere Architektur gewählt mit 4 Hidden Layer, 55 Neuronen im ersten 20 Neuronen im zweiten 5 Neuronen im dritten und 52 Neuronen im vierten Hidden Layer. Diese Architektur wurde wieder mit einer Hyperparametersuche gefunden. Der Programmcode hierfür befindet sich im Anhang (H Programmcode Hyperparametersuche 5 Jahre).

		Vorhergesagt	
		0	1
Real	0	5330	5301
	1	5189	5317

		Vorhergesagt	
		0	1
Real	0	25,22%	25,08%
	1	24,55%	25,15%

Tabelle 12: Wahrheitsmatrix 5 Jahre

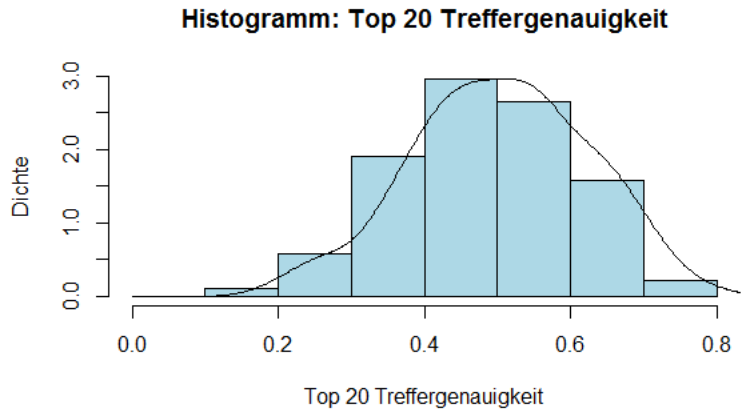

Abbildung 18: Histogramm Top 20 Treffergenauigkeit 5 Jahre

Durch den erweiterten Betrachtungshorizont wurde weder die Treffergenauigkeit noch die „Top 20 Treffergenauigkeit“ erhöht.

6. Schlussbetrachtung

Abschließend eine zusammenfassende Betrachtung der einzelnen Kapitel im Kontext der wissenschaftlichen Arbeit, eine komprimierte Darstellung der Limitationen des Modelles und ein Ausblick mit weiteren Forschungsfragen und Verbesserungsmöglichkeiten.

6.1. Zusammenfassung

Im Bereich der KI wird zu den Methoden im Machine- und Deep Learning weiterhin geforscht werden. Der Anstieg der Publikationen in den letzten Jahren lässt dies Vermuten. Auch der Anstieg der Publikationen im Umfeld der KI in Kombination mit Kapitalmarktthemen lässt vermuten, dass in diesem Bereich noch viel zu erforschen und viele Modelle zu verbessern sind.

In dieser Arbeit wurde ein bestehendes Modell repliziert um dieses anschließend zu analysieren und weiterzentwickeln. Es wurden die relevanten technischen Grundlagen vorgestellt um Modelle, welche auf NN basieren verstehen zu können.

Mit dieser Arbeit wurde gezeigt, dass sich die Prognosegenauigkeit des bestehenden Modelles von den Herren Takeuchi und Lee verbessern lässt. Durch iterative Trainings- und Testzyklen konnten erhöhte Treffergenauigkeiten erzielt werden. Zur Bewertung einer Anlagestrategie, welche die Vorhersagen des NNs als Basis nehmen würde, wurden drei Portfolien gebildet, Kennzahlen berechnet und mit Benchmarks verglichen. Es wurde gezeigt, dass eine Überrendite ohne Berücksichtigung von Transaktionskosten erzielt werden konnte.

Es wurde deutlich, dass die Replizierbarkeit von NN nur mit einem hohen Aufwand gegeben ist und das in dem entstandenen Modell eine hohe Varianz bei den Treffergenauigkeiten vorzufinden ist.

In der Sensitivitätsanalyse wurde festgestellt, dass eine Erweiterung des Zeithorizonts, um den Long-Term Reversal Effekt mit einzuschließen, keine Verbesserung in der Prognosegenauigkeit mit sich zog. Es konnten jedoch Verbesserungen mit einer anderen Architektur erzielt werden.

Zusammenfassend wurde das neuronale Netz weiterentwickelt und durch die Analysen, die konzipierten und berechneten Kennzahlen kann das Modell besser verstanden werden.

6.2. Limitationen des Modelles

NN sind limitiert auf die Daten die sie schon gesehen haben, mit denen sie schon trainiert wurden. Dies ist ihre Erfahrungssammlung. Das weiterentwickelte Modell bezieht sich bisher weiterhin nur auf Kurszahlen von Aktien. Somit kann das Modell auch nur in diesen Daten Mustern erkennen und, wenn diese immer schwächer werden oder gar verschwinden kann das Modell keine Muster mehr erkennen um eine Überrendite zu erzielen.

Jedes NN was trainiert wird ist nahezu einmalig, aufgrund der vielen zufälligen Werte oder bewusst gesetzten Hyperparameter die dazu beitragen wie das NN trainiert wird. Dies trägt dazu bei, dass die Replizierbarkeit von NN nur mit einem hohen Aufwand gegeben ist.

6.3. Ausblick

Die Wahl der Hyperparameter ist sehr komplex und auch mit einer Hyperparametersuche ist es nicht immer möglich die besten Kombinationen von Hyperparametern zu finden. Dementsprechend gibt es in dem Bereich der Hyperparameterfindung noch einige Verbesserungsmöglichkeiten.

Das Paket H2O unterstützt derzeit noch keine Aktivationsfunktionen die erst vor kurzem konzipiert oder an neuronalen Netzen mit Erfolg getestet wurden. Hierzu gehört auch die Aktivationsfunktion „Rectified Linear Unit“ welche bereits bei einigen Anwendungen die Treffergenauigkeiten erhöhen konnte. Eine Anwendung dieser Funktion könnte das Modell ebenfalls verbessern.

Im Artikel „An enhanced artificial neural network for stock price predications“ wurden verschiedene Normalisierungen bei den Eingabewerten getestet und mit einer Normalisierung der Werte zwischen -0,5 und 0,5 wurden niedrigere Logistic Loss Werte festgestellt als bei einer Normalisierung zwischen 0 und 1 (Ma et al.). Dementsprechend könnte eine Normalisierung zwischen -0,5 und 0,5 die Treffergenauigkeit des vorgestellten Modelles erhöhen.

Zusätzlich neben den Kursen könnten auch noch weitere Daten mit einbezogen werden, wie z.B. Bilanzkennzahlen, Smart Beta Faktoren oder makroökonomische Kennzahlen zum jeweiligen Markt, oder Nachrichten Analysen. All diese wurden bereits einzeln mit NNs getestet und haben Überrenditen erzielt (Xiao Ding,Yue Zhang,Ting Liu,Junwen Duan; Wei Cao, Liang Hu, Longbing Cao; Bennewirtz; Ballings et al. 2015; Chourmouziadis und Chatzoglou 2016). Ein NN, welches diese einzelnen NNs vereint, könnte die einzelnen Ergebnisse nochmals verbessern.

Anhangsverzeichnis

A. Programmcode Datenvorverarbeitung

```
#Data Preprocessing

#Working Directory zuhause und Uni
setwd("C:/Users/shook/Google Drive/Bachelor Thesis/Empirisches
Modell")
setwd("H:/Bachelor Thesis/Empirische Umsetzung")

#Restart ACHTUNG löscht alle Daten
rm(list=ls(all=TRUE))

#Libraries

#Schritt 1: Tägliche Kurse zu täglichen & monatl. Performancewerten
kurseTaegl = read.csv("Daten/0_Kurse.csv", sep =",", dec = ".",
stringsAsFactors=FALSE)
komposition = read.csv("Daten/0_Komposition.csv", sep =";", dec = ",",
stringsAsFactors=FALSE)


#Dummy für Perf Array
Perf = array(NA,dim=c(40000,61))

#Stellen der Monatswechsel bestimmen
monthchangepos=c()

for (c in 2:nrow(kurseTaegl)){
  if (as.numeric(format(as.Date(kurseTaegl[c,1],origin="1899-12-30"),
"%m"))!=as.numeric(format(as.Date(kurseTaegl[c-1,1],origin="1899-12-
30"), "%m"))){
    monthchangepos=append(monthchangepos,c)
  }
}
#Für später (neuralnet backtest ergebnisse)
ergebnismatrix = kurseTaegl[monthchangepos,]
ergebnismatrix[,2:228] = NA
write.csv(ergebnismatrix,
file="Daten/0_ergebnismatrix_blank.csv",row.names = F)

#Performance bestimmen

count=1
for (c in 62:(length(monthchangepos)-1)){
#for (c in 14:(length(monthchangepos)-1)){
  for(i in 2:228){
    #if (sum(is.na(kurseTaegl[monthchangepos[c-
13]:monthchangepos[c+1],i]))==0 & kurseTaegl[monthchangepos[c],i]>5){
    if (sum(is.na(kurseTaegl[monthchangepos[c-
61]:monthchangepos[c+1],i]))==0 & kurseTaegl[monthchangepos[c],i]>5){

      Perf[count,1]=kurseTaegl[monthchangepos[c],1]
      Perf[count,2]=colnames(kurseTaegl)[i]

Perf[count,35]=as.numeric(as.numeric(format(as.Date(kurseTaegl[monthch
angepos[c],1],origin="1899-12-30"), "%m"))==1)
      Perf[count,36]=1+(kurseTaegl[monthchangepos[c+1],i]-
kurseTaegl[monthchangepos[c],i])/kurseTaegl[monthchangepos[c],i]


Perf[count,37]=as.numeric(komposition[komposition[,1]==as.numeric(form
```

```
at(as.Date(kurseTaegl[monthchangepos[c],1],origin="1899-12-30"),
"%Y")),i]==1)

      Perf[count, 3]=as.numeric(1+(kurseTaegl[monthchangepos[c-12],i]-
kurseTaegl[monthchangepos[c-13],i])/kurseTaegl[monthchangepos[c-
13],i])

      for(tminus in 2:12){
        Perf[count, 2+tminus]=as.numeric(Perf[count,
1+tminus])*(1+(kurseTaegl[monthchangepos[c-13+tminus],i]-
kurseTaegl[monthchangepos[c-
14+tminus],i])/kurseTaegl[monthchangepos[c-14+tminus],i])
      }
      Perf[count, 15]=1+(kurseTaegl[monthchangepos[c]-19,i]-
kurseTaegl[monthchangepos[c]-20,i])/kurseTaegl[monthchangepos[c]-20,i]
      for(tminus in 1:19){
        Perf[count, 15+tminus]=as.numeric(Perf[count,
14+tminus])*(1+(kurseTaegl[monthchangepos[c]-19+tminus,i]-
kurseTaegl[monthchangepos[c]-
20+tminus,i])/kurseTaegl[monthchangepos[c]-20+tminus,i])
      }

      Perf[count, 38]=as.numeric(1+(kurseTaegl[monthchangepos[c-
60],i]-kurseTaegl[monthchangepos[c-
61],i])/kurseTaegl[monthchangepos[c-61],i])

      for(tminus in 2:24){
        Perf[count, 37+tminus]=as.numeric(Perf[count,
36+tminus])*(1+(kurseTaegl[monthchangepos[c-61+tminus],i]-
kurseTaegl[monthchangepos[c-
62+tminus],i])/kurseTaegl[monthchangepos[c-62+tminus],i])
      }

      count=count+1
    }
  }
}

performanceDF = data.frame(monat=integer(count-1),stringsAsFactors =
F)

#Übertrag in ein Dataframe zur weiterverarbeitung

performanceDF$monat=as.numeric(Perf[1:count-1,1])
performanceDF$titel=Perf[1:count-1,2]
performanceDF$monatl1=as.numeric(Perf[1:count-1,3])
performanceDF$monatl2=as.numeric(Perf[1:count-1,4])
performanceDF$monatl3=as.numeric(Perf[1:count-1,5])
performanceDF$monatl4=as.numeric(Perf[1:count-1,6])
performanceDF$monatl5=as.numeric(Perf[1:count-1,7])
performanceDF$monatl6=as.numeric(Perf[1:count-1,8])
performanceDF$monatl7=as.numeric(Perf[1:count-1,9])
performanceDF$monatl8=as.numeric(Perf[1:count-1,10])
performanceDF$monatl9=as.numeric(Perf[1:count-1,11])
performanceDF$monatl10=as.numeric(Perf[1:count-1,12])
performanceDF$monatl11=as.numeric(Perf[1:count-1,13])
performanceDF$monatl12=as.numeric(Perf[1:count-1,14])
performanceDF$taegl1=as.numeric(Perf[1:count-1,15])
performanceDF$taegl2=as.numeric(Perf[1:count-1,16])
performanceDF$taegl3=as.numeric(Perf[1:count-1,17])
performanceDF$taegl4=as.numeric(Perf[1:count-1,18])
performanceDF$taegl5=as.numeric(Perf[1:count-1,19])
```

```
performanceDF$taegl6=as.numeric(Perf[1:count-1,20])
performanceDF$taegl7=as.numeric(Perf[1:count-1,21])
performanceDF$taegl8=as.numeric(Perf[1:count-1,22])
performanceDF$taegl9=as.numeric(Perf[1:count-1,23])
performanceDF$taegl10=as.numeric(Perf[1:count-1,24])
performanceDF$taegl11=as.numeric(Perf[1:count-1,25])
performanceDF$taegl12=as.numeric(Perf[1:count-1,26])
performanceDF$taegl13=as.numeric(Perf[1:count-1,27])
performanceDF$taegl14=as.numeric(Perf[1:count-1,28])
performanceDF$taegl15=as.numeric(Perf[1:count-1,29])
performanceDF$taegl16=as.numeric(Perf[1:count-1,30])
performanceDF$taegl17=as.numeric(Perf[1:count-1,31])
performanceDF$taegl18=as.numeric(Perf[1:count-1,32])
performanceDF$taegl19=as.numeric(Perf[1:count-1,33])
performanceDF$taegl20=as.numeric(Perf[1:count-1,34])
performanceDF$jan=as.numeric(Perf[1:count-1,35])
performanceDF$outp=as.numeric(Perf[1:count-1,36])
performanceDF$inIndex=as.numeric(Perf[1:count-1,37])
performanceDF$monatl61=as.numeric(Perf[1:count-1,61])
performanceDF$monatl60=as.numeric(Perf[1:count-1,60])
performanceDF$monatl59=as.numeric(Perf[1:count-1,59])
performanceDF$monatl58=as.numeric(Perf[1:count-1,58])
performanceDF$monatl57=as.numeric(Perf[1:count-1,57])
performanceDF$monatl56=as.numeric(Perf[1:count-1,56])
performanceDF$monatl55=as.numeric(Perf[1:count-1,55])
performanceDF$monatl54=as.numeric(Perf[1:count-1,54])
performanceDF$monatl53=as.numeric(Perf[1:count-1,53])
performanceDF$monatl52=as.numeric(Perf[1:count-1,52])
performanceDF$monatl51=as.numeric(Perf[1:count-1,51])
performanceDF$monatl50=as.numeric(Perf[1:count-1,50])
performanceDF$monatl49=as.numeric(Perf[1:count-1,49])
performanceDF$monatl48=as.numeric(Perf[1:count-1,48])
performanceDF$monatl47=as.numeric(Perf[1:count-1,47])
performanceDF$monatl46=as.numeric(Perf[1:count-1,46])
performanceDF$monatl45=as.numeric(Perf[1:count-1,45])
performanceDF$monatl44=as.numeric(Perf[1:count-1,44])
performanceDF$monatl43=as.numeric(Perf[1:count-1,43])
performanceDF$monatl42=as.numeric(Perf[1:count-1,42])
performanceDF$monatl41=as.numeric(Perf[1:count-1,41])
performanceDF$monatl40=as.numeric(Perf[1:count-1,40])
performanceDF$monatl39=as.numeric(Perf[1:count-1,39])
performanceDF$monatl38=as.numeric(Perf[1:count-1,38])

#Überprüfung is dürfen keine NAs auftreten
sum(is.na(performanceDF[1:count-1,]))
write.csv(performanceDF,
file="Daten/1_Performance5Jahre.csv",row.names = F)

#Schritt 2: Z Score berechnung
#kurse = read.csv("Daten/1_Performance.csv", sep =",", dec = ".")
kurse = performanceDF
Zscores_Final = performanceDF
zscores = array(NA,dim=c(nrow(kurse),61))
normalize=function(x){(x-min(x))/(max(x)-min(x))}

for (b in c(3:34,38:61)){
  m1=mean(kurse[which(kurse$monat==kurse[1,1]), b])
  std=sd(kurse[which(kurse$monat==kurse[1,1]), b])
  #meanT1=mean(kurse[which(kurse$monat==kurse[1,1]), 36])
  for (a in 1:nrow(kurse)){
    zscores[a,b]=(kurse[a,b]-m1)/std
    if (b==3){
```

```
      zscores[a,35]=kurse[a,35]

      #zscores[a,36]=as.numeric(kurse[a,36]>median(kurse[which(kurse$m
onat==kurse[a,1]), 36]))

      #if(kurse[a,36]>quantile(kurse[which(kurse$monat==kurse[a,1]),
36],.8)){
      #  zscores[a,36]=1
      #}else{
      #  zscores[a,36]=0
      #}

      if(kurse[a,36]>quantile(kurse[which(kurse$monat==kurse[a,1]),
36],.8)){
        zscores[a,36]=2
      }else
if(kurse[a,36]>quantile(kurse[which(kurse$monat==kurse[a,1]),
36],.2)){
        zscores[a,36]=1
      }else{
        zscores[a,36]=0
      }

    }
    if (a<nrow(kurse)){
      if (kurse[a+1,1]!=kurse[a,1]){
        #meanT1=mean(kurse[which(kurse$monat==kurse[a,1]), 36])
        m1=mean(kurse[which(kurse$monat==kurse[a+1,1]), b])
        std=sd(kurse[which(kurse$monat==kurse[a+1,1]), b])
        zscores[which(kurse$monat==kurse[a,1]),b] =
normalize(zscores[which(kurse$monat==kurse[a,1]),b])
      }
    }
    if (a==nrow(kurse)){
        zscores[which(kurse$monat==kurse[a,1]),b] =
normalize(zscores[which(kurse$monat==kurse[a,1]),b])
    }
  }
  Zscores_Final[,b]=zscores[,b]
}

Zscores_Final[,35]=zscores[,35]
Zscores_Final[,36]=zscores[,36]
Zscores_Final[,37]=kurse$inIndex

#NAs aufgrund von einem "Handelstag" ohne Aktivität (keine
Kursänderung --> Keine Standardabw --> Div durch 0 )
sum(is.na(Zscores_Final[]))

#NAs werden als 0.5 (neutral) weitergeführt
#Zscores_Final[is.na(Zscores_Final[])] = 0.5

write.csv(Zscores_Final, file="Daten/2_ZScoresNorm5Jahre3Klassen.csv",
row.names = F)

#Damit ist die Vorverarbeitung abgeschlossen. Die Zscore dienen als
Input für das neurale Netz
#Speicher bereinigen, nur die KurseTaegl und ZScore werden weiter
benötigt
rm(Perf, performanceDF, zscores, kurse, komposition,)
```

B. Hyperparameter

Eine vollständige Auflistung aller Hyperparameter befindet sich hier. Folgend nur die modifizierten Hyperparameter. Die Beschreibungen wurden 1:1 aus der technischen Dokumentation entnommen:

http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html

- **training_frame**: (Required) Specify the dataset used to build the model. **NOTE**: In Flow, if you click the **Build a model** button from the Parse cell, the training frame is entered automatically.
- **validation_frame**: (Optional) Specify the dataset used to evaluate the accuracy of the model.
- **activation**: Specify the activation function (Tahn, Tahn with dropout, Rectifier, Rectifier with dropout, Maxout, Maxout with dropout).
- **hidden**: Specify the hidden layer sizes (e.g., 100,100). The value must be positive.
- **epochs**: Specify the number of times to iterate (stream) the dataset. The value can be a fraction.
- **input_dropout_ratio**: Specify the input layer dropout ratio to improve generalization. Suggested values are 0.1 or 0.2.
- **hidden_dropout_ratios**: (Applicable only if the activation type is **TanhWithDropout**, **RectifierWithDropout**, or **MaxoutWithDropout**) Specify the hidden layer dropout ratio to improve generalization. Specify one value per hidden layer. The range is >= 0 to <1, and the default is 0.5.
- **l1**: Specify the L1 regularization to add stability and improve generalization; sets the value of many weights to 0.
- **l2**: Specify the L2 regularization to add stability and improve generalization; sets the value of many weights to smaller values.
- **stopping_rounds**: Stops training when the option selected for **stopping_metric** doesn't improve for the specified number of training rounds, based on a simple moving average. To disable this feature, specify 0. The metric is computed on the validation data (if provided); otherwise, training data is used. When used with **overwrite_with_best_model**, the final model is the best model generated for the given **stopping_metric** option.

- **stopping_metric**: Specify the metric to use for early stopping. The available options are:
    - auto: This defaults to logloss for classification, deviance for regression
    - deviance
    - logloss
    - mse
    - rmse
    - mae
    - rmsle
    - auc
    - lift_top_group
    - misclassification
    - mean_per_class_error
- **stopping_tolerance**: Specify the relative tolerance for the metric-based stopping to stop training if the improvement is less than this value.
- **autoencoder**: Specify whether to enable the Deep Learning autoencoder. This option is not enabled by default.
- **max_runtime_secs**: Maximum allowed runtime in seconds for model training. Use 0 to disable.
- **seed**: Specify the random number generator (RNG) seed for algorithm components dependent on randomization. The seed is consistent for each H2O instance so that you can create models with the same starting conditions in alternative configurations.
- **reproducible**: Specify whether to force reproducibility on small data. If this option is enabled, the model takes more time to generate because it uses only one thread.
- **export_weights_and_biases**: Specify whether to export the neural network weights and biases as H2O frames.

C. Programmcode Hyperparametersuche

```
#2. Hyperparametersearch. Um Sicherzustellen das passende
Hyperparameter verwendet werden wird zunächst diese Suche
durchgeführt.
#Auch mit gesetzten Seed ist es nicht gewährleistet immer das selbe
Ergebnis zu erhalten ist.
search_criteria = list(strategy = "RandomDiscrete", max_runtime_secs =
360, max_models = 100, seed=1234567, stopping_rounds=5,
stopping_tolerance=1e-2)

hyper_params <- list(
  #activation=c("Rectifier","Tanh","Maxout","RectifierWithDropout","Ta
nhWithDropout","MaxoutWithDropout"),
  activation="Tanh",
  input_dropout_ratio=c(0,0.02,0.03,0.04,0.05),
  #input_dropout_ratio=0.03,
  l1=seq(0,1e-4,1e-6),
  #l1=5.9E-5,
  #l2=2.9E-5,
  #hidden=list(c(34,34,34),c(20,4,20),c(50,24,4,24,50),c(30,30)),
  l2=seq(0,1e-4,1e-6),
  stopping_metric = c("logloss","AUC","misclassification")
  #stopping metric = "logloss"
)

dl_random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "dl_grid_random",
  training_frame=train_init_h2o,
  validation_frame=test_init_h2o,
  x=colnames(train_init[,3:35]),
  y="outp",
  epochs=3,
  hidden=c(40,4,50),
  stopping_tolerance=1e-3,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  max_w2=10,
  balance_classes=TRUE,
  score_validation_sampling="Stratified",
  hyper_params = hyper_params,
  search_criteria = search_criteria,
  #export_weights_and_biases = T
  seed=12
)
grid <-
h2o.getGrid("dl_grid_random",sort_by="logloss",decreasing=FALSE)
grid

grid@summary_table[1,]
#Hyper-Parameter Search Summary: ordered by increasing logloss
#activation input_dropout_ratio     l1     l2 stopping_metric
#1       Tanh                0.03 5.9E-5 2.9E-5         logloss
#model_ids            logloss
#1 dl_grid_random_model_9 0.6915309208308928

best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest
logloss

#Der folgende Autoencoder wird mit den oben genannten Hyperparams
erstellt.
```

D. Programmcode Funktion für den gestapelten Autoencoder

Die Funktion basiert auf folgendem Code:

https://github.com/h2oai/h2o-3/blob/master/h2o-r/tests/testdir_algos/deeplearning/runit_deeplearning_stacked_autoencoder_large.R

```
Funktion für die Erstellung des Autoencoders
check.deeplearning_stacked_autoencoder <- function() {
  # this function builds a vector of autoencoder models, one per layer
  get_stacked_ae_array <- function(training_data,layers,args){
    vector <- c()
    index = 0
    for(i in 1:length(layers)){
      index = index + 1
      ae_model <- do.call(h2o.deeplearning,
                          modifyList(list(x=names(training_data),

training_frame=training_data,
                                          autoencoder=T,
                                          hidden=layers[i]),
                                     args))
      training_data = h2o.deepfeatures(ae_model,training_data,layer=1)

      names(training_data) <- gsub("DF", paste0("L",index,sep=""),
names(training_data))
      #print(h2o.weights(ae_model,matrix_id=1))
      vector <- c(vector, ae_model)
    }
    vector
  }

  # this function returns final encoded contents
  apply_stacked_ae_array <- function(data,ae){
    index = 0
    for(i in 1:length(ae)){
      index = index + 1
      data = h2o.deepfeatures(ae[[i]],data,layer=1)
      names(data) <- gsub("DF", paste0("L",index,sep=""), names(data))
    }
    data
  }

  TRAIN <- "Daten/2_Train.csv"
  TEST <- "Daten/2_Test.csv"
  response <- 34

  #set to T for RUnit
  #set to F for stand-alone demo
  if (F) {
    train_hex <- h2o.importFile(locate(TRAIN))
    test_hex  <- h2o.importFile(locate(TEST ))
  } else {
    #library(h2o)
    #h2o.init(nthreads=-1)

    #homedir <- "C:/Users/shook/Google Drive/Bachelor
Thesis/Empirisches Modell/" #modify if needed
    #homedir <- "H:/Bachelor Thesis/Empirische Umsetzung/" #modify if
needed
```

```
    train_hex <- h2o.importFile(path = TRAIN, header = T, sep = ',')
    test_hex  <- h2o.importFile(path = TEST, header = T, sep = ',')
  }

  train <- train_hex[,-response]
  test  <- test_hex [,-response]
  train_hex[,response] <- as.factor(train_hex[,response])
  test_hex [,response] <- as.factor(test_hex [,response])

  ## Build reference model on full dataset and evaluate it on the test
set
  #model_ref <- h2o.deeplearning(training_frame=train_hex,
x=1:(ncol(train_hex)-1), y=response, hidden=c(40,4,50), epochs=3)
  #p_ref <- h2o.performance(model_ref, test_hex)
  #print(h2o.logloss(p_ref))

  ## Now build a stacked autoencoder model with three stacked layer AE
models
  ## First AE model will compress the 717 non-const predictors into
200
  ## Second AE model will compress 200 into 100
  ## Third AE model will compress 100 into 50
  layers <- c(40,4)
  #layers <- c(40,4)

  args <- list(activation="Tanh",
               epochs = 2,
               l1=5.9E-5,
               l2=2.9E-5,
               input_dropout_ratio=0.03,
               export_weights_and_biases = T)
  ae <- get_stacked_ae_array(train, layers, args)
  bc=ae
  ## Now compress the training/testing data with this 3-stage set of
AE models
  train_compressed <- apply_stacked_ae_array(train, ae)
  test_compressed <- apply_stacked_ae_array(test, ae)

  ## Build a simple model using these new features (compressed
training data) and evaluate it on the compressed test set.
  train_w_resp <- h2o.cbind(train_compressed, train_hex[,response])
  test_w_resp <- h2o.cbind(test_compressed, test_hex[,response])


  model_on_compressed_data <-
h2o.deeplearning(training_frame=train_w_resp,

x=1:(ncol(train_w_resp)-1),
                                                    y=ncol(train_w_resp),
                                                    hidden=c(50),
                                                    activation="Tanh",
                                                    epochs = 2,
                                                    l1=5.9E-5,
                                                    l2=2.9E-5,

input_dropout_ratio=0.03,

export_weights_and_biases = T)
  p <- h2o.performance(model_on_compressed_data, test_w_resp)
  print(h2o.logloss(p))
  return(bc=c(bc, model_on_compressed_data))
}
```

E. Beschreibung Feld Bloomberg

Beschreibung zum Feld “TOT_RETURN_INDEX_GROSS_DVDS” entnommen aus dem Bloomberg Excel Add-In.

“Available as Historical field

One day total return index as of today. The start date is one day prior to the end date (as of date). Historically, this is the total return index from the provided start date to the provided end date. Applicable periodicity values are daily, weekly, monthly, quarterly, semi-annually and annually. Gross dividends are used.”

F. Programmcode Portfoliobildung

```
# Ergebnisauswertung

#Benötigte Pakete: Installation
install.packages("quadprog")
install.packages("corpcor")

#Benötigte Pakete: Laden
library(quadprog)
library(corpcor)


# Daten einlesen (Ergebnismatrix_buyable)
ergebnismatrix_full =
read.csv("Daten/3_Ergebnismatrix_buyable5Jahre.csv", sep =",", dec =
".")
#ergebnismatrix full = ergebnismatrix buyable

kurseTaegl = read.csv("Daten/0_Kurse.csv", sep =",", dec = ".",
stringsAsFactors=FALSE)
TPRatee=read.csv("Daten/Archiv/5JahreTPRate.csv", sep =",", dec = ".",
stringsAsFactors=FALSE)

perfdaily = kurseTaegl
perfdaily[]=NA
perfdaily[,1]=kurseTaegl[,1]
for ( w in 2:228){
  for (i in 2:6333){
    if(!(is.na(kurseTaegl[i,w]))&!(is.na(kurseTaegl[i-1,w]))){
      perfdaily[i,w]=1+(kurseTaegl[i,w]-kurseTaegl[i-
1,w])/kurseTaegl[i-1,w]
    }
  }
}
write.csv(perfdaily, file="Daten/1_Perfdaily.csv", row.names = F)


perfdaily = read.csv("Daten/1_Perfdaily.csv", sep =",", dec = ".")

#marketcap = read.csv("Daten/0_Marktkapitalisierung.csv", sep =";",
dec = ".",stringsAsFactors=FALSE, header=TRUE)
```

```
#for (i in 1:25){
#  marketcap[i,is.na(marketcap[i,1:228])] =
mean(as.numeric(marketcap[i,2:228]), na.rm=T)
#}

GewichtungGG = ergebnismatrix_full
GewichtungGG[] = NA
GewichtungGG[,1] = ergebnismatrix_full[,1]
#Keine MW Gewichtung mehr sondern nach Wahrscheinlichkeitswert
GewichtungMW = ergebnismatrix_full
GewichtungMW[] = NA
GewichtungMW[,1] = ergebnismatrix_full[,1]
GewichtungMinVar = ergebnismatrix_full
GewichtungMinVar[] = NA
GewichtungMinVar[,1] = ergebnismatrix_full[,1]
BenchGewichtungMinVar = ergebnismatrix_full
BenchGewichtungMinVar[] = NA
BenchGewichtungMinVar[,1] = ergebnismatrix_full[,1]


N = 20

#A = rbind(c(rep(1,20)),diag(20))
#b = c(1,rep(0,20))
A.Equality <- matrix(c(rep(1,20)), ncol=1)
A = cbind(A.Equality, diag(20),-diag(20))
b = c(1, rep(0.02,20), rep(-0.15,20))
#m für 5 Jahre
for (m in 72:nrow(ergebnismatrix_full)) {
  top20=order(ergebnismatrix_full[m,1:228], decreasing = T)[2:(N+1)]
  GewichtungGG[m,top20] = 1/N
  covarianz=
cov.shrink(perfdaily[which(perfdaily[,1]==ergebnismatrix_full[m,1]):(w
hich(perfdaily[,1]==ergebnismatrix_full[m,1])-250),top20])


GewichtungMinVar[m,top20]=solve.QP(covarianz,dvec=rep(0,20),Amat=A,bve
c=b,meq=1)$solution

  #GewichtungMW[m,top20] =
marketcap[as.numeric(format(as.Date(ergebnismatrix_buyable[m,1],origin
="1899-12-30"), "%Y"))-
1==marketcap[,1],top20]/sum(marketcap[as.numeric(format(as.Date(ergebn
ismatrix buyable[m,1],origin="1899-12-30"), "%Y"))-
1==marketcap[,1],top20])
  GewichtungMW[m,top20] =
ergebnismatrix_full[m,top20]/sum(ergebnismatrix_full[m,top20])

  inIndex=227-sum(is.na(ergebnismatrix_full[m,]))
  AT.Equality <- matrix(c(rep(1,inIndex)), ncol=1)
  AT = cbind(AT.Equality, diag(inIndex),-diag(inIndex))
  bT = c(1, rep(0.002,inIndex), rep(-0.1,inIndex))

  covarianz=
cov.shrink(perfdaily[which(perfdaily[,1]==ergebnismatrix_full[m,1]):(w
hich(perfdaily[,1]==ergebnismatrix_full[m,1])-
250),c(F,!is.na(ergebnismatrix_full[m,2:228]))])

BenchGewichtungMinVar[m,c(F,!is.na(ergebnismatrix_full[m,2:228]))]=sol
ve.QP(covarianz,dvec=rep(0,inIndex),Amat=AT,bvec=bT,meq=0)$solution

}
```

```
rowSums(GewichtungGG[,2:228],na.rm=T)
rowSums(GewichtungMinVar[,2:228],na.rm=T)
rowSums(GewichtungMW[,2:228],na.rm=T)
#rowSums(BenchGewichtungMinVar[,2:228],na.rm=T)

GewichtungGG[is.na(GewichtungGG)] = 0
GewichtungMW[is.na(GewichtungMW)] = 0
GewichtungMinVar[is.na(GewichtungMinVar)] = 0
#BenchGewichtungMinVar[is.na(BenchGewichtungMinVar)] = 0

performanceGG=c()
performanceMW=c()
performanceMinVar=c()
Bench_performanceGG=c()
Bench_performanceMinVar =c()

for (i in 72:297){
  #perfdaily[perfdaily[,1]>=GewichtungGG[i,1]&&perfdaily[,1]<Gewichtun
gGG[i+1,1],GewichtungGG[i,1:228]>0]
  portGG =
perfdaily[(perfdaily[,1]>=GewichtungGG[i,1]&perfdaily[,1]<GewichtungGG
[i+1,1]),GewichtungGG[i,1:228]>0]
  performanceGG =
append(performanceGG,sum(((tail(cumprod(portGG[,1:21]), n=1)-
1)*GewichtungGG[i,GewichtungGG[i,1:228]>0])[2:21])+1)

  portMW =
perfdaily[(perfdaily[,1]>=GewichtungMW[i,1]&perfdaily[,1]<GewichtungMW
[i+1,1]),GewichtungMW[i,1:228]>0]
  performanceMW =
append(performanceMW,sum(((tail(cumprod(portMW[,1:ncol(portMW)]),
n=1)-1)*GewichtungMW[i,GewichtungMW[i,1:228]>0])[2:ncol(portMW)])+1)

  portMinVar =
perfdaily[(perfdaily[,1]>=GewichtungMinVar[i,1]&perfdaily[,1]<Gewichtu
ngMinVar[i+1,1]),GewichtungMinVar[i,1:228]>0]
  performanceMinVar =
append(performanceMinVar,sum(((tail(cumprod(portMinVar[,1:ncol(portMin
Var)]), n=1)-
1)*GewichtungMinVar[i,GewichtungMinVar[i,1:228]>0])[2:ncol(portMinVar)
])+1)
  #Benchmark

  Bench_portGG =
perfdaily[(perfdaily[,1]>=GewichtungGG[i,1]&perfdaily[,1]<GewichtungGG
[i+1,1]),!is.na(ergebnismatrix_full[i,])]
  Bench_performanceGG =
append(Bench_performanceGG,mean(as.numeric(tail(cumprod(Bench_portGG[,
2:ncol(Bench_portGG)]), n=1))))

  Bench_portMinVar =
perfdaily[(perfdaily[,1]>=GewichtungGG[i,1]&perfdaily[,1]<GewichtungGG
[i+1,1]),!is.na(ergebnismatrix_full[i,])]
  Bench_performanceMinVar =
append(Bench_performanceMinVar,sum(((tail(cumprod(Bench_portMinVar[,1:
ncol(Bench_portMinVar)]), n=1)-
1)*BenchGewichtungMinVar[i,!is.na(ergebnismatrix_full[i,])])[2:ncol(Be
nch_portMinVar)])+1)

}
```

G. Kennzahlen Übersicht

	Gleichgewichtetes Portfolio	Benchmark Gleichgewichtet
Sharpe Ratio	0,166	0,145
Information Ratio	0,055	
Sortino Ratio	0,737	0,609
Anteil Verlustmonate	38,5%	41,2%
Anteil Verlustjahre	27,8%	22,2%
Max Drawdown 1 Jahr	54,7%	53,1%
Jensen Alpha	0,005	0,004
Fama French Alpha	0,005	0,003
Carhart Alpha	0,006	0,005
	NN-Ergebnis gewichtetes Portfolio	*Benchmark Gleichgewichtet*
Sharpe Ratio	0,169	0,145
Information Ratio	0,044	
Sortino Ratio	0,767	0,609
Anteil Verlustmonate	38,5%	41,2%
Anteil Verlustjahre	27,8%	22,2%
Max Drodown 1 Jahr	54,0%	53,1%
Jensen Alpha	0,005	0,004
Fama French Alpha	0,005	0,003
Carhart Alpha	0,006	0,005
	Minimum-Varianz-Portfolio	*Benchmark Minimum-Varianz*
Sharpe Ratio	0,207	0,208
Information Ratio	0,072	
Sortino Ratio	0,923	1,127
Anteil Verlustmonate	39,3%	36,4%
Anteil Verlustjahre	27,8%	22,2%
Max Drodown 1 Jahr	53,1%	33,1%
Jensen Alpha	0,004	0,009
Fama French Alpha	0,003	0,011
Carhart Alpha	0,005	0,005

H. Programmcode Hyperparametersuche 5 Jahre

```
#2. Hyperparametersearch. UmSicherzustelen das passende Hyperparameter
#verwendet werden wird zunächst diese Suche durchgeführt.
#Auchmit gesetzten Seed ist es nicht gewährleistet immer das selbe
Ergebnis zu erhalten.
search_criteria = list(strategy = "RandomDiscrete", max_runtime_secs =
360, max_models = 100, seed=1234567, stopping_rounds=5,
stopping_tolerance=1e-2)

hyper_params <- list(

activation=c("Rectifier","Tanh","Maxout","RectifierWithDropout","TanhW
ithDropout","MaxoutWithDropout"),
  input_dropout_ratio=c(0,0.02,0.03,0.04,0.05),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6),
  hidden=list(c(60,24,6,54),c(55,20,5,52),c(50,24,4,50)),
  stopping_metric = c("logloss","AUC","misclassification")
)

hyper_params <- list(

activation=c("Rectifier","Tanh","Maxout","RectifierWithDropout","TanhW
ithDropout","MaxoutWithDropout"),
  input_dropout_ratio=c(0,0.02,0.03,0.04,0.05),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6),
  hidden=list(c(60,24,6,54),c(55,20,5,52),c(50,24,4,50)),
  stopping_metric = c("logloss","AUC","misclassification")
)

dl_random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "dl_grid_random",
  training_frame=train_init_h2o,
  validation_frame=test_init_h2o,
  x=colnames(train_init[,3:59]),
  y="outp",
  epochs=3,
  stopping_tolerance=1e-3,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  max_w2=10,
  balance_classes=TRUE,
  score_validation_sampling="Stratified",
  hyper_params = hyper_params,
  search_criteria = search_criteria
  #export_weights_and_biases = T
)
grid <-
h2o.getGrid("dl_grid_random",sort_by="logloss",decreasing=FALSE)
grid

grid@summary_table[1,]
#Hyper-Parameter Search Summary: ordered by increasing logloss
#activation          hidden input_dropout_ratio     l1     l2
stopping_metric
#1       Tanh [55, 20, 5, 52]                0.04 6.9E-5 7.2E-5
AUC
#model_ids            logloss
#1 dl_grid_random_model_0 0.6896824224293736
```

Literaturverzeichnis

Al-Hmouz, Rami; Pedrycz, Witold; Balamash, Abdullah (2015): Description and prediction of time series. A general framework of Granular Computing. In: *Expert Systems with Applications* 42 (10), S. 4830–4839. DOI: 10.1016/j.eswa.2015.01.060.

Ballings, Michel; van den Poel, Dirk; Hespeels, Nathalie; Gryp, Ruben (2015): Evaluating multiple classifiers for stock price direction prediction. In: *Expert Systems with Applications* 42 (20), S. 7046–7056. DOI: 10.1016/j.eswa.2015.05.013.

Bennewirtz, Gerd: Lupus Alpha Smaller German Champions. Online verfügbar unter http://www.stock-world.de/analysen/nc3901365-SJB_FondsEcho_Nebenwerte_Gefragt_Lupus_Alpha_Smaller_German_Champions.html, zuletzt geprüft am 13.03.2017.

Berben, Tobias (2017): Master(P) alias AlphaGo spielt 60:0. Online verfügbar unter http://www.go-baduk-weiqi.de/masterp-alias-mastergo-spielt-60zu0/, zuletzt geprüft am 13.03.2017.

Blitz, David; Huij, Joop; Lansdorp, Simon; Verbeek, Marno (2013): Short-term residual reversal. In: *Journal of Financial Markets* 16 (3), S. 477–504. DOI: 10.1016/j.finmar.2012.10.005.

Chourmouziadis, Konstandinos; Chatzoglou, Prodromos D. (2016): An intelligent short term stock trading fuzzy system for assisting investors in portfolio management. In: *Expert Systems with Applications* 43, S. 298–311. DOI: 10.1016/j.eswa.2015.07.063.

Deng, Li (2013): Deep Learning. Methods and Applications. In: *FNT in Signal Processing* 7 (3-4), S. 197–387. DOI: 10.1561/2000000039.

Geoffrey Booth, G.; Fung, Hung-Gay; Leung, Wai Kin (2016): A risk-return explanation of the momentum-reversal “anomaly”. In: *Journal of Empirical Finance* 35, S. 68–77. DOI: 10.1016/j.jempfin.2015.10.007.

Göçken, Mustafa; Özçalıcı, Mehmet; Boru, Aslı; Dosdoğru, Ayşe Tuğba (2016): Integrating metaheuristics and Artificial Neural Networks for improved stock price prediction. In: *Expert Systems with Applications* 44, S. 320–331. DOI: 10.1016/j.eswa.2015.09.029.

Gong, Qiang; Liu, Ming; Liu, Qianqiu (2015): Momentum is really short-term momentum. In: *Journal of Banking & Finance* 50, S. 169–182. DOI: 10.1016/j.jbankfin.2014.10.002.

Hinton, G. E.; Salakhutdinov, R. R. (2006): Reducing the dimensionality of data with neural networks. In: *Science (New York, N.Y.)* 313 (5786), S. 504–507. DOI: 10.1126/science.1127647.

Jia, Hengjian: Investigation Into The Effectiveness Of Long Short Term Memory Networks For Stock Price Prediction. Online verfügbar unter http://arxiv.org/pdf/1603.07893v3.

Lawrence Takeuchi *; Yu-Ying (Albert) Lee: Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks.

Ma, Jiaxin; Huang, Silin; Kwok, S.H.: An enhanced artificial neural network for stock price predications.

McInish, Thomas H.; Ding, David K.; Pyun, Chong Soo; Wongchoti, Udomsak (2008): Short-horizon contrarian and momentum strategies in Asian markets. An integrated analysis. In: *International Review of Financial Analysis* 17 (2), S. 312–329. DOI: 10.1016/j.irfa.2006.03.001.

O. V.: H2O Documentation. Online verfügbar unter https://www.rdocumentation.org/packages/h2o/versions/3.0.0.22/topics/h2o.deeplearning, zuletzt geprüft am 13.03.2017.

O. V.: Höchstleistungsrechenzentrum Stuttgart. Online verfügbar unter https://de.wikipedia.org/wiki/H%C3%B6chstleistungsrechenzentrum_Stuttgart, zuletzt geprüft am 13.03.2017.

O. V. (2016): GeForce GTX TITAN X Specifications. Online verfügbar unter http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications, zuletzt geprüft am 13.03.2017.

O.V.: Information Ratio. Online verfügbar unter http://www.styleadvisor.com/resources/statfacts/information-ratio, zuletzt geprüft am 13.03.2017.

Rumelhart, David E.; Group, James L. McClelland and the PDP Research (1999): Parallel distributed processing: explorations in the microstructure of cognition. 12. Aufl. Cambridge (Computational Models of Cognition and Perception, ; Volume 1).

Schmidhuber, Jurgen (2015): Deep learning in neural networks: an overview. In: *Neural networks : the official journal of the International Neural Network Society* 61, S. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Wei Cao, Liang Hu, Longbing Cao: Deep Modeling Complex Couplings within Financial Markets.

Wu, Yuliang; Mazouz, Khelifa (2016): Long-term industry reversals. In: *Journal of Banking & Finance* 68, S. 236–250. DOI: 10.1016/j.jbankfin.2016.03.017.

Wurgler, Jeffrey (2000): Financial markets and the allocation of capital. In: *Journal of Financial Economics* 58 (1-2), S. 187–214. DOI: 10.1016/S0304-405X(00)00070-2.

Xiao Ding,Yue Zhang,Ting Liu,Junwen Duan: Deep Learning for Event-Driven Stock Prediction.

Yu, Kun-Hsing; Zhang, Ce; Berry, Gerald J.; Altman, Russ B.; Re, Christopher; Rubin, Daniel L.; Snyder, Michael (2016): Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. In: *Nature communications* 7, S. 12474. DOI: 10.1038/ncomms12474.

Eidesstattliche Versicherung

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Für alle Inhalte, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen gleich welcher Art entnommen sind, habe ich die fremde Urheberschaft kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form weder im Inland noch im Ausland als Prüfungsarbeit eingereicht worden. Sie ist auch noch nicht veröffentlicht.

Frankfurt am Main, 15.03.2017