Bachelorarbeit

Entwicklung und Evaluation eines Machinelles Lernen Algorithmus zum automatisierten Handeln von Wertpapieren unter Einbeziehung von Nachrichten-Sentiment und weiteren Metriken

An der Fachhochschule Dortmund

im Fachbereich Informatik

erstellte Bachelorarbeit

im Studiengang Informatik Vertiefungsrichtung: Praktische Informatik

zur Erlangung des akademischen Grades *Bachelor of Science in Informatik*

von

Benedikt Willecke

geboren am 15.01.1998

(Matr.-Nr.: 7105861)

Betreuer: Prof. Dr. Sebastian Bab

Zweitprüfer: Prof. Dr. Burkhard Lenze

Bearbeitungszeit: 07.12.2020 - Heute

Zusammenfassung

In der Finanzwelt werden verschiedene Metriken genutzt, um auf eine möglichst optimale Art und Weise Investitionen zu tätigen. Hierbei spielt auch die Marktstimmung eine wichtige Rolle. Natural Language Processing ermöglicht es hierbei Textdaten automatisiert zu verarbeiten und mit Sentiment Analysis diese nach ihrer Stimmung zu klassifizieren.

In dieser Arbeit wird, nach Betrachtung verschiedener Lösungsansätze, das LSTM Modell genutzt, um durch die Nutzung von dem Sentiment von Wirtschaftsnachrichten die Rendite von Aktien vorherzusagen. Es wurden verschiedene Parameter und Datenquellen eingesetzt, um zu untersuchen wie das Machine Learning Modell sich bei diesen verhält. Hierbei wurde sich insbesondere auf die Datenvorverarbeitung sowie das Hyperparamteretuning des Modells fokussiert. Für die Feststellung des Sentiments der Wirtschaftsnachrichten wurde ein darauf abgestimmtes BERT Modell genutzt, welches in dieser Bachelorarbeit zu Grunde liegenden Projektarbeit entwickelt wurde. Bei der Datenvorverarbeitung konnte argumentiert werden, warum die Vorhersage von Renditen statt den Aktienkursen selbst eine geeignetere Problemstellung ist.

Hierbei wurde unter anderem festgestellt, dass sich eine Vergrößerung der Batchgröße die Performance des Modells negativ beeinflussen kann, insbesondere, wenn dadurch die Anzahl an Iterationen verringert wird. Die Batchgröße hat hier im Kontext dieser Bachelorarbeit sogar den größten Einfluss auf die Performance im Vergleich zu anderen Metriken wie die Lernrate. Auch eine Veränderung des Verhaltens seitens des Modells konnte bei auftretenden Ausreißern in Daten beobachtet werden.

Weiterführende Forschungen wären jedoch von Interesse, da zu untersuchen ist, ob mit mehr Ressourcen und einer weitgehenderen Hyperparameteroptimierung, ein Modell entwickelt werden könnte, welches die Renditen genauer vorhersagen kann und robuster auf Ausreißer reagiert. Auch Forschung zu der Ergründung von dem Einfluss der Batchgröße auf die Performance von Machine Learning Modellen wäre wichtig für zukünftige Hyperparameteroptimierungen.

Abstract

In the financial world different metrics are used in order to invest in the best way possible. Market sentiment plays a big part in that. Natural Language Processing enables autoamted processing oft text data where then one can detect the sentiment of such data via sentiment analysis.

In this thesis, after considering several possible solutions an LSTM model is used to predict stock returns by using the sentiment of financial news. Different parameter and data sources were used to observe how the machine learning model would change its behavior. Data pre-processing and hyperparametertuning were especially focused on. Fort he detection of the sentiment of financial news a BERT model was used which was fine tuned especially for this purpose. The model used was developed in the project thesis that is the basis for this thesis. In the data pre-processing part it was shown that using stock returns as a prediction target instead of stock price provides a better and more challenging problem for the model.

It was also shown that increasing the batch size can negatively impact the performance of the model as the number of iterations are reduced. In this thesis the batch size actually had the most impact on performance even when compared to other metrics such as the learning rate. Moreover the change of model behavior due to outliers in the data was able to be observed.

Additional research would be necessary in order to determine if it is possible with more ressources and more extensive hyperparametertuning to develop a model which predicts stock returns more accurately and is more robust towards outliers. Research on the reasons in regards to the performance impact by the batch size is especially important to improve hyperparameter tuning of machine learning models in the future.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1: Einleitung

Algorithmen beeinflussen längst viele Teile der Gesellschaft, Politik und Wirtschaft. Sie treffen Entscheidungen darüber welche Produkte dem Konsumenten vorgeschlagen werden [1], welche Werbespots welches Politikers der Wähler sieht [2] und auch in welche Unternehmen investiert wird und in welche nicht. Letzteres bezeichnet man als „algorithmischen Handel“ oder auch „hochfrequenten algorithmischen Handel“. Wobei ersteres die Bestimmung von Auftragsparametern wie Ausführungszeitpunkt, Preis und Quantität durch einen Algorithmus auf Basis von Eingangsdaten beschreibt.

Dagegen wird bei dem Hochfrequenzhandel zusätzlich die Latenz der Orderübertragungen minimiert und in hoher Frequenz Aufträge autonom ausgeführt [3] [4]. Die Nutzung dieser Technologie gewinnt mehr und mehr an Relevanz und machte bereits 2012 80% des gesamten Handelsvolumens aus [5]. Das liegt daran, dass mittlerweile jede große Investmentbank ihre eigene Software nutzt, um mittels des Hochfrequenzhandels höhere Gewinne zu erzielen [6].

Hierbei werden unter anderem Marktdaten, Firmendaten und auch Stimmungsdaten von Nachrichten genutzt [4], da Neuigkeiten bspw. aus Politik und Wirtschaft die Aktienkurse direkt beeinflussen können [7].

1.1 Motivation

Das Feld der Sentiment Analysis, ein Teilbereich des NLP(Natural Language Processing), dient dazu die Haltung von Personen gegenüber gewissen Dingen zu analysieren [8][9]. Man kann bspw. die negative oder positive Stimmung auf Twitter erfassen, um Schwankungen der amerikanischen Aktienindexe Dow Jones, S&P500 und NASDAQ vorherzusagen [9]. Dies kann geschehen in dem man positive und negative Emotionen erkennt oder durch komplexere Emotionen genauer differenziert zwischen glücklich, beruhigt und sicher etc. Letzteres hat zu einer höheren Genauigkeit der Vorhersage des Aktienmarkts geführt [9]. Die Nutzung von Sentiments von Nachrichten kann erfolgreich eingesetzt werden, um Aktiengewinne vorherzusagen oder in algorithmischen Handelssystemen eingesetzt werden [89].

1.2 Zielsetzung

In dieser Bachelorarbeit werde ich versuchen zu einem tiefen Verständnis des algorithmischen Handels zu gelangen, die verschiedenen Konzepte und Techniken dieses zu verstehen aber auch seine Herausforderungen offenzulegen. Darüber hinaus wird der Kern der Thesis sein zu analysieren, ob das Sentiment von Wirtschaftsnachrichten bei dem algorithmischen Handel unterstützend wirken kann. Hierzu wird der in meiner Projektarbeit entwickelte und auf BERT basierende Algorithmus bei der Sentiment Analysis von Wirtschaftsnachrichten zur Hilfe genommen. Darüber hinaus werden weitere Metriken wie z.B. momentaner Aktienkurs, Datum oder 200-Tage Durchschnitt genommen, um einen Vergleich zwischen einem Sentiment-gestützten und regulären Algorithmus zu ziehen. Auf die Anbindung an eine reelle Handelsplattform, welche die vom Algorithmus berechneten Transaktionen ausführt, wird verzichtet. Somit steht der Algorithmus, welcher entscheidet welches Wertpapier in welchem Maße gekauft, verkauft oder gehalten werden soll, selbst im Fokus. Darüber hinaus sind die Performanceeinflüsse von Sentiments der Wirtschaftsnachrichten auf algorithmische Handelssysteme zu erforschen sowie schlussfolgernd den aus dieser Thesis bestmöglichen Algorithmus zu ermitteln.

1.3 Methodik

Zuerst werde ich Quellen wie Fachbücher und Forschungspapiere nutzen, um einen Überblick über wesentliche Konzepte, Definitionen sowie Probleme des algorithmischen Handels zu schaffen. Hierbei soll besonders auf den Bezug des algorithmischen Handels mit Machine Learning eingegangen werden. Auf die Erläuterung von Themen, die nicht direkt das Thema algorithmische Handelssysteme tangieren (wie Grundlagen des Machine Learning oder der Sentiment Analysis), wird verzichtet. Anschließend werde ich Vor- und Nachteile verschiedener Lösungsansätze ermitteln und dann einen dieser implementieren. Zum Schluss möchte ich die Ergebnisse meines Projektes präsentieren und ein Fazit sowie eine Aussicht auf zukünftige Arbeiten geben.

1.4 Kapitelübersicht

Kapitel 2 Grundlagen

Die für dieses Projekt relevanten Aspekte des algorithmischen Handels werden erklärt.

Kapitel 3 Lösungsansätze

Mögliche Lösungswege werden erkundet.

Kapitel 4 Planung

Die Lösungsansätze werden verglichen und ein Implementierungsansatz entwickelt.

Kapitel 5 Umsetzung

Die Implementierung wird vorgestellt und dokumentiert.

Kapitel 6 Experimente

Untersuchungen zur Implementierung werden erklärt und dargelegt.

Kapitel 7 Fazit

Die Ergebnisse der Bachelorarbeit werden zusammengefasst, es wird ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten dargelegt.

Kapitel 2: Grundlagen

In diesem Kapitel werde ich auf die grundlegenden Konzepte der Thematik dieser Bachelorarbeit eingehen. Hierbei möchte ich insbesondere die Historie des Wertpapierhandels und die Automatisierung dieses darlegen sowie die wichtigsten Konzepte davon erläutern, um das nötige Grundwissen, welches für diese Arbeit notwendig ist, zu ermöglichen.

2.1 Einführung Aktienmarkt & Wertpapierhandel

Die erste Aktiengesellschaft entstand 1602 mit der Gründung der niederländischen Ostindien-Kompanie (kurz: VOC), welche teure Güter, wie Pfeffer aus Indonesien, nach Europa importierte. Da die Reise jedoch lang und hochriskant war, war das Ziel der Gründung einer Aktiengesellschaft dieses Risiko auf mehrere Anteilnehmer zu verteilen. Diese bekamen Aktien, also Anteile an der Gesellschaft, und erhielten so eine Dividende, damals noch in Form von Pfeffersäcken, ausgezahlt. Auf diesem Modell aufbauend entstanden viele solcher Gesellschaften, wobei die Kapitaleigner ihrem Anteil entsprechend an dem Gewinn beteiligt wurden. Im 19. Jahrhundert entstanden im Zuge der Industrialisierung Aktiengesellschaften, um Kapital zu sammeln für Erfindungen wie die Eisenbahn, das Telefon oder das Dampfschiff. Die Anzahl der Aktiengesellschaften stieg so stetig an und es entstand der Aktienmarkt in seinen Grundzügen so wie man ihn heute kennt. Das 20. Jahrhundert war geprägt von neuen Höhen und Tiefen im Aktienmarkt. Insgesamt konnten Wertpapiere als Finanzinstrument den schnellen technologischen Fortschritt ermöglichen. Die Aktien der Unternehmen werden über Börsen ausgegeben und können dort gehandelt werden. Die Börsen selbst haben ihren Ursprung bereits im 15. Jahrhundert in Brügge (im Jahr 1409) und Antwerpen (im Jahr 1531) als organisierte Handelsplätze auf denen Geld gewechselt und Güter (wie Gewürze) aber auch Finanzprodukte (wie Getreide-Futures) gehandelt werden konnten. Andere Städte nahmen sich dieses Konzept zum Vorbild und so eröffneten 1569 in London, 1611 in Amsterdam, 1756 in Berlin und 1792 in New York weitere Börsen. 1971 begann mit dem NASDAQ (National Association of Dealers Automated Quotation System) der vollelektronische bzw. computergestützte Handel in New York. 1997 folgte Frankfurt mit der Einführung von XETRA

(Exchange Electronic Trading). Nun mussten Teilnehmer an der Börse nicht mehr persönlich vor Ort sein. Durch diese elektronischen Handelssysteme wurde zudem algorithmischer Handel möglich gemacht [10] [11] [12] [13] [14] [15].

2.2 Algorithmischer Handel

Als algorithmischen Handel bezeichnet man die Nutzung von prozess- und regelbasierten Algorithmen zur strategischen Ausführung von Wertpapierordern. Die wohl bekannteste Form des algorithmischen Handels ist HFT (High Frequency Trading), zu Deutsch hochfrequentes Handeln. Hierbei werden viele Handelsaufträge in kürzester Zeit ausgeführt. So können auch kleine Preisunterschiede genutzt werden, um Profite zu erlangen. Dazu ist es nötig die Ausführungsgeschwindigkeit von Aufträgen sowie die Latenz zu den Börsen so stark wie möglich zu verbessern, um schneller als andere Marktteilnehmer agieren und reagieren zu können. Die hohe Geschwindigkeit des Handels kann aber auch zu sogenannten Flash-Crashs führen, bei denen in kürzester Zeit ohne menschliche Intervention durch Verkäufe der Preis des jeweiligen Wertpapieres rapide fallen kann. Auch auf Nachrichten reagieren die Märkte so sehr schnell: Als am 23. April 2013 fälschlicherweise die Nachrichtenagentur „The Associated Press“ auf Twitter die Nachricht veröffentlichte, dass durch zwei Explosionen am Weißen Haus der damalige Präsident Barack Obama verletzt wurde verlor der Dow Jones 1% innerhalb von drei Minuten und erlangte seinen vorherigen Wert kurze Zeit später wieder. Firmen wie „Dataminr“ nutzen Quellen wie Twitter, um Ereignisse aus Nachrichten zu erkennen und diese in Handelsaufträge umzuwandeln. Die Idee Informationen schnell zu nutzen, um Profite zu erzielen, ist keine Neuheit: Schon im 17ten Jahrhundert nutzte Nathan Mayer Rothschild das Wissen um den Ausgang der Schlacht bei Waterloo, um Gewinne zu erzielen. Im 19ten Jahrhundert baute Julius Reuter ein System mit Telegraphen und Brieftauben auf, um schneller als andere an Informationen zu gelangen. Solche HFT-Systeme haben jedoch einen hohen Kostenaufwand für die Soft- und Hardware, was die Gewinnmarge schmälert. Die Spitze in der Nutzung von HFT wurde bereits 2009 erreicht. Auch Regulierungen werden strikter, Steuern für den HFT erhöht und durch die steigende Anzahl an Mitbewerbern wird es immer schwieriger konkurrenzfähig zu bleiben. Deshalb suchen Firmen nach kostengünstigeren alternativen Strategien, wie Momentum Trading oder Automated News-

Based Trading, die nicht in Gefahr laufen weiter reguliert zu werden. Auch zu erwähnen sind die verschiedenen Arten von Marktteilnehmern, die algorithmische Handelssysteme nutzen, um ihre Strategien umzusetzen. Dabei handelt es sich zum einen um mittel- bis langfristige Investoren (bzw. Marktteilnehmer auf der Kaufseite), wie Pensionsfonds oder Versicherungen, die algorithmische Handelssysteme nutzen, um diskret große Investments umzusetzen, ohne den Preis des Wertpapiers groß zu beeinflussen. Kurzfristig orientierte Investoren (bzw. Marktteilnehmer auf der Verkaufsseite), wie Spekulanten, können durch algorithmische Handelssysteme strukturiert und automatisiert ihre Handelsaufträge durchführen. Die sogenannten „Systemic Trader“, wie bspw. Hedgefonds, programmieren algorithmische Handelssysteme mit ihren eigenen Regeln und Strategien, um diese automatisiert ausführen zu lassen [16] [17] [18] [19].

2.3 Handelsstrategien

Es gibt zwei grundsätzliche Formen der Finanzanalyse: Die technische Analyse, auch Chartanalyse genannt, und die Fundamentalanalyse. Bei letzterem wird versucht den intrinsischen Wert eines Wertpapieres zu messen, indem Gesichtspunkte wie die allgemeine wirtschaftliche Lage, die wirtschaftliche Lage der Industrie, die finanzielle Situation des Unternehmens und das Management betrachtet werden. Dabei werden Metriken wie Gewinne, Ausgaben, Vermögenswerte und Schulden genutzt. Im Gegensatz dazu steht die technische Analyse, bei der potenzielle Investments basierend auf statistischen Trends, wie Veränderungen des Aktienpreises oder des Handelsvolumens, ermittelt werden. Hierbei spielt der intrinsische Wert des Unternehmens keine Rolle, sondern es sollen lediglich Muster und Trends im Aktienkurs erkannt werden, die bei einem Investment einen Gewinn indizieren. Hierbei wird bei der Analyse nach Kauf- und Verkaufssignalen gesucht. Eine dieser Analysetechniken ist der Simple Moving Average (kurz: SMA). Hierbei wird über eine festgelegte Zeitperiode der durchschnittliche Aktienpreis berechnet. Falls eine SMA Linie eines kürzeren Zeitraumes die eines längeren Zeitraumes überschreitet so wird dies als Kauf- oder Verkaufssignal gewertet. Auch „Unterstützung“ und „Widerstand“ spielen eine wichtige Rolle. Der Widerstand ist ein Bereich im Aktienpreis, bei dem Anleger regelmäßig ihre Anteile wiederverkaufen, und so den Preis gesenkt, haben. Die Unterstützung ist ein Bereich im Aktienpreis, bei dem Anteile regelmäßig von

Anlegern gekauft haben, und so der Preis wieder gesteigert wurde. Die Unterstützung stellt hier ein Kaufsignal und der Widerstand ein Verkaufssignal dar. Diesem Konzept ähnlich sind die sogenannten Trendlinien. Hierbei werden auf dem historischen Aktienpreis basierende Linien in die Zukunft projiziert, die Einstiegs- und Ausstiegsmöglichkeiten definieren. Dies ist nützlich, falls der Aktienkurs immer neue Höhen und Tiefen erreicht [20].

Strategien, die beim algorithmischen Handeln benutzt werden, gehören einen dieser beiden grundlegenden Formen an, wobei häufig Metriken bzw. Strategien aus der technischen Finanzanalyse genutzt werden. Im Folgenden werden einige dieser Strategien vorgestellt. Eine der häufigsten und auch einfachsten Strategien sind sogenannte „Trend-following Stragies“, wie der schon vorgestellte SMA, bei denen versucht wird Trends zu ermitteln und diesen zu folgen. Diese Strategie funktioniert gut, wenn starke und deutliche Trends vorhanden sind. Bei schwankendem Kurs können jedoch häufig falsche Signale durch den SMA erzeugt werden, worunter der erreichte Gewinn durch diese Strategie leidet. Auch zu erwähnen ist, dass der Markt nicht immer die Regeln der Strategie respektiert, weshalb die Ergebnisse vom SMA eher zufällig sind [21] [22].

Als Arbitrage bezeichnet man die Ausnutzung eines zu demselben Zeitpunkt bestehenden Preisunterschiedes eines Wertpapiers zwischen zwei verschiedenen Markplätzen. Dieser Preisunterschied kann durch Ineffizienzen im Markt, wie bspw. durch Wechselkurse von Währungen, entstehen. Angenommen eine Aktie wird auf dem Toronto Stock Exchange (TSX) für $63,50CAD und dem New York Stock Exchange (NYSE) für $47,00 gehandelt werden. Bei einem Wechselkurs USD/CAD von $1USD = $1,37CAD müssten die Aktien in Toronto für $64,39CAD handeln. Dieser Unterschied kann genutzt werden, indem die in Toronto unterbewerteten Wertpapiere dort gekauft und am NYSE gleichzeitig verkauft werden, um so pro Wertpapier $0,89 Gewinn einzubringen. Hierbei sind zum einen die Transaktionskosten zu beachten, welche höher als der erwartete Gewinn sein können und zum anderen ist diese Art der Ausnutzung von Marktineffizienzen nur für Marktteilnehmer anwendbar, welche in kurzer Zeit große Investments tätigen können, da sich dieser Preisunterschied nach kurzer Zeit wieder ändern kann [21] [23].

Eine weitere Strategie ist es die Neugewichtung, die Indexfonds und Pensionsfonds regelmäßig durchführen müssen, um ihre festgelegte Allokation beizubehalten, vorherzusagen und auszunutzen. Angenommen die Allokation des Fonds soll 50% Aktien und 50%

Anleihen betragen. Falls der Wert der Aktien nun schneller steigt als der der Anleihen so kann eine Allokation von 60% Aktien und 40% Anleihen entstehen, welche durch Aktienverkäufe wieder in Balance gebracht werden muss. Durch Handeln vor diesen Aktienverkäufen können algorithmische Handelssysteme Gewinne erwirtschaften [21] [24].

Auch auf mathematischen Modellen basierende Strategien wie Delta-Hedging können zum Einsatz kommen. Optionen auf Wertpapiere haben einen Delta-Wert, welcher angibt um wie viel der Preis der Option sich verändert, wenn der Preis des zu Grunde liegenden Wertpapieres sich verändert. Wenn bei einem Delta-Wert von 0,5 der Preis des zu Grunde liegenden Wertpapieres um $20 steigt so steigt der Preis der Option um $10. Dabei haben Kaufoptionen einen Delta-Wert zwischen 0 und 1 und Verkaufsoptionen einen Delta-Wert von -1 bis 0. Um sich gegen das Risiko von für den Anleger negativen Preisveränderungen (wenn bspw. bei Kaufoptionen der Preis des Wertpapiers sinkt) abzusichern, ist die Strategie ein neutrales Delta zu erreichen. Hierbei wird durch algorithmische Handelssysteme bei Preisveränderungen eine Option mit dem gegenteiligen Delta-Wert zu kaufen. Angenommen die Kaufoption hatte einen Delta-Wert von 0,5, so würde bei dem eben genannten Beispiel dann eine Verkaufsoption mit einem Delta-Wert von -0,5 gekauft werden, um ein insgesamt neutrales Delta zu erreichen. Durch diese Strategie kann sich der Anleger sowohl gegenüber ihm nachteilige Preisveränderungen des zu Grunde liegenden Wertpapieres absichern als auch Profite sichern, ohne die Anlage selbst zu verkaufen. Hierbei ist jedoch zu beachten, dass falls für das Erreichen eines neutralen Delta-Wertes viele Transaktionen notwendig sind, die Transaktionskosten dementsprechend hoch sind. Außerdem können bei weiteren unerwarteten Preisveränderungen der zuvor erreichte Ausgleich den gegenteiligen Effekt haben [21] [25] [26].

Eine weitere Strategie ist die Mean Reversion Strategie, welche auf der Theorie beruht, dass Märkte dazu neigen übertrieben zu steigen oder zu fallen, sie jedoch immer zum langfristigen Durchschnittswert zurückkehren. Dies kann man sich zu Nutze machen, indem man Übertreibungen der Preisänderungen identifiziert, um so günstig zu kaufen und teuer zu verkaufen. Profite sind mit dieser Strategie jedoch nicht garantiert, da Ereignisse in der realen Welt nicht in Betracht gezogen werden. Dies limitiert die Profitabilität dieser Strategie, da bspw. durch neue Produktveröffentlichungen oder Gerichtsverfahren eine große unerwartete Veränderung im Durchschnitt entstehen kann [21] [27].

Der volumengewichtete Durchschnittspreis (Volume Weighted Average Price, kurz: VWAP) ist eine weitere Kennzahl, die als Handelsstrategie genutzt werden kann. Hierbei wird der Preis jeder Transaktion addiert und durch das Gesamtvolumen des Wertpapiers innerhalb eines festgelegten Zeitraumes (meist einen Tag) dividiert. Dies kann durch folgende Formel ausgedrückt werden: $VWAP = \frac{\sum Preis * Volumen}{\sum Volumen}$. Wie beim SMA wird beim VWAP also auch ein Durchschnittspreis berechnet, hierbei wird jedoch das Gesamtvolumen sowie das Volumen jeder Transaktion in Betracht gezogen. Dies ist sinnvoll, da dadurch, dass der Preis über Angebot und Nachfrage geregelt wird, größere Transaktionen auch einen stärkeren Einfluss auf den Preis des jeweiligen Wertpapieres haben. Des Weiteren kann dieser Indikator nützlich sein, da er Einblicke sowohl in den Wert eines Wertpapieres als auch den Trend ermöglicht. Große institutionelle Käufer können dies nutzen, um bei einem Preis unterhalb des VWAPs zu kaufen und oberhalb von diesem zu verkaufen, um diskret große Summen von Wertpapieren zu erwerben, da durch dieses Handeln nach der Transaktion sich der Preis zurück zu dem VWAP anstatt von ihm wegbewegt. Aktienhändler hingegen nutzen diesen Indikator für Trends: Oberhalb des VWAP kann gekauft werden, um bei einem Trend nach oben Gewinne zu erzielen. Bei einem negativen Trend kann unterhalb des VWAP ein Leerverkauf (Short) getätigt werden. Auch dieser Indikator ist in seinen Möglichkeiten jedoch begrenzt. So kann es bei rapide ansteigenden Kursen zu der Situation kommen, dass der Kurs nicht unter den VWAP fällt, wodurch Investmentmöglichkeiten verpasst werden können, falls sich nur auf den VWAP verlassen wird [21] [28].

Time Weighted Average Price (TWAP) oder auch Dollar Cost Averaging genannt (zu Deutsch: Durchschnittskosteneffekt) bezeichnet eine Strategie, bei der eine große Summe an zu investierendem Kapital in kleinere Ordern, welche in regelmäßigen Zeitabständen ausgeführt werden, unterteilt wird. Dadurch wird das Risiko der Volatilität des Wertpapierkurses gesenkt, da so ein Durchschnittspreis in einem festgelegten Zeitraum erreicht wird. Dadurch wird der potenzielle Verlust durch das Volatilitätsrisiko aber auch der potenzielle Gewinn minimiert [21] [29].

Viele der vorgestellten Strategien, wie bspw. Arbitrage, benötigen eine schnelle Ausführung der Order, um zu funktionieren. Dies wird als High-Frequency Trading (kurz: HFT) bezeichnet und benötigt einen großen Kapitalaufwand, um die nötige Infrastruktur aufzubauen. Des Weiteren wird HFT strenger reguliert und die Konkurrenz nimmt zu. Aus

diesen Gründen wurden neue Strategien entwickelt, die im Folgenden vorgestellt werden [19] [21].

Eine mögliche Strategie, die weniger Latenzabhängig ist, ist Momentum Trading. Hierbei werden zukünftige Kursbewegungen identifiziert, woraufhin eine oder gestaffelte Handelsaufträge ausgeführt werden, um auch bei kleinen Kursveränderungen Gewinne erzielen zu können. Es wird versucht früh ein Momentum zu erkennen, um zu einem niedrigen Kurs zu kaufen und dann am Ende des Momentums zum höchst möglichen Kurs wieder zu verkaufen. Auf Grund, der im Vergleich zu HFT längeren Haltedauer der Wertpapiere von wenigen Minuten bis einigen Monaten ist die Latenz weniger wichtig, wodurch der Kapitalaufwand für Infrastruktur des HFT eingespart werden kann [30] [31].

Für das Identifizieren von Momentum können verschiedene Indikatoren genutzt werden, wie z.B. Rate of Change (kurz: RoC). RoC ist ein Indikator, welcher in einer festgelegten Zeit, die die Geschwindigkeit der Preisänderung in Prozent misst. Ein positiver Wert spiegelt einen Preisanstieg und ein negativer Wert einen Preisabfall wider. Der RoC wird mit folgender Formel berechnet:

$RoC = \left(\frac{Kurs(t) - Kurs(t-n)}{Kurs(t-n)}\right) * 100$. Hierbei ist Kurs(t) der aktuelle Kurs und Kurs(t-n) der Kurs vor n Tagen. Es kann ein Wert für n gewählt werden, dem der Strategie des Anlegers entspricht. Langfristige Anleger wählen einen hohen Wert wie bspw. 200 und kurzfristige einen Wert wie 9 oder 12. Bei kleineren Werten für n reagiert der Indikator zwar schneller auf Preisänderungen, aber auch empfindlicher, wodurch alltägliche Preisänderungen große Auswirkungen auf den RoC haben können und so fälschlicherweise ein Momentum erkannt werden kann. Große Werte für n haben genau die gegenteiligen Eigenschaften. Der Indikator kann genutzt werden, indem bspw. in das Wertpapier mit dem größten RoC investiert wird, da hohe historische Gewinne zukünftige indizieren können. Außerdem beginnen Preisanstiege oft mit einem rapiden Anstieg, welcher wiederum von dem RoC identifiziert werden kann [32] [33] [34] [35]. Es existiert noch nicht eine bewiesene Begründung warum die Momentum Investing Strategie funktioniert. Eine Theorie besagt, dass Investoren nicht stark genug auf Informationen reagieren, welche graduell erscheinen, wodurch über Zeit ein Momentum aufgebaut wird, um diese Unterreaktion auszugleichen („Frog in the Pan“ Theorie). Eine andere Theorie ist, dass Investoren korrekt auf Nachrichten reagieren, die ihren Glauben bestätigen, aber nicht auf Nachrichten, welche

ihren bisherigen Glauben widerlegen, weshalb Momentum in optimistischen Perioden (Bullenmärkte) auftritt, da auf schlechte Nachrichten nicht entsprechend reagiert wird („Cognitive Dissonance“ Theorie) [36]. Es lässt sich jedoch beobachten, dass Momentum Investing auf lange Sicht in verschiedenen Asset-Klassen eine Outperformance erreicht [38]. Der MSCI World Momentum konnte in den Jahren 1976 – 2016 einen jährlichen Gewinn von mehr als 13% erreichen, während der MSCI World Index nur etwas über 10% erreichen konnte [37]. Das Risiko bei dieser Strategie sind rapide fallende Kurse, weshalb um die Verluste bei einer Umkehrung im Trend zu minimieren, bspw. ein 12-Monate Momentum Indikator mit einem 6-Monate Momentum Indikator kombiniert werden kann, da so auf neue Trends schneller reagiert werden kann [38]. Des Weiteren ist auf Grund der negativen Korrelation zwischen Momentum und Value (hierbei können Buchwert-Kurs-Verhältnis (kurz: B/P), erwartetes Kurs-Gewinn-Verhältnis (kurz: KGV) und Dividendenrendite betrachtet) eine Kombination der beiden Strategien zur Diversifikation sinnvoll. Auch eine risikoadjustierte Bewertung des Momentums kann helfen, um zu viel Volatilität zu vermeiden [37] [38] [39].

Alternativ zu statistischen und technischen Strategien können auch Strategien, die auf Fundamentaldaten basieren, angewandt werden. Hierbei kann bspw. das B/P Verhältnis in Betracht gezogen werden, indem die obersten 10% der Wertpapiere mit dem höchsten B/P Verhältnis gekauft und die unteren 10% mit dem niedrigsten B/P Verhältnis leerverkauft werden. Wie schon erwähnt kann eine Kombination der Momentum- und der Value-Strategie sinnvoll sein, da in Kombination höhere Gewinne erzielt werden können als bei der alleinigen Nutzung einer der beiden Strategien. Um eine Kombination mehrerer Strategien zu erreichen gibt es verschiedene Möglichkeiten. Man kann alle Faktoren (bspw. Value und Momentum) gleich gewichten und dann mit dem addierten gewichteten Wert beider Faktoren entscheiden in welche Wertpapiere gewinnbringend investiert werden kann, wobei eine 50-50 Gewichtung nicht unbedingt optimal ist. Auch ist es möglich zuerst mit einem Faktor mögliche Investments zu identifizieren und daraufhin aus diesen mit Hilfe des anderen Faktors Wertpapiere auszuwählen [40] [41].

Zum Abschluss dieses Überblicks über verschieden Strategien, ausgenommen Machine Learning Strategien, welche in einem späteren Kapitel behandelt werden, soll erwähnt sein, warum technische Indikatoren, also Indikatoren, welche keine Fundamentaldaten betrachten, wie ein SMA, funktionieren: Der Grund ist, dass das Verhalten des Marktes

durch das Verhalten der Investoren bestimmt wird, welches mit einer gewissen Genauigkeit vorausgesagt werden kann, indem historisches Kursverhalten analysiert wird. Im Falle von der Momentum Strategie existieren, wie erwähnt, verschiedene Theorien, die versuchen den Grund hinter der Voraussagekraft des Indikators zu erklären. Zum einen ist es möglich, dass auf Grund des höheren Risikos von momentumstarken Wertpapieren der Gewinn, also die Entlohnung für das hohe getragene Risiko, höher ist. Zum anderen könnte eine Begründung das Verhalten der Anleger sein, welche eine gewisse Zeit lang nicht stark genug auf Nachrichten reagieren bzw. eine verzögerte Überreaktion aufzeigen. Unabhängig davon welche bzw. ob eine der beiden Theorien korrekt ist, ist zu beobachten, dass die Momentum Strategie dieses Verhalten des Marktes nutzen kann, um eine Outperformance zu generieren, weswegen der Grund für das Auftreten dieser Verhaltensmuster irrelevant ist, solange diese Verhaltensmuster bestehen [41] [42] [43]. Hierbei ist zu erwähnen, dass dadurch, dass systematische Handelsstrategien sich wiederholende Muster im Verhalten des Marktes auszunutzen versuchen, eine systematische Handelsstrategie aufhören zu funktionieren kann, wenn sich das Verhalten des Marktes ändert, während dann andere Handelsstrategien wieder oder erstmals anwendbar wären [45] [46]. Wenn ein algorithmisches Handelssystem Ineffizienzen im Markt auszunutzen versucht, diese Strategie nun aber von anderen Marktteilnehmern auch verfolgt wird, so verschwindet die Ineffizienz und somit der Gewinn, welcher abermals durch Transaktionsgebühren geschmälert werden kann [47]. Auch Veränderungen der Marktbedingungen können dazu führen, dass die Performance eines bisher funktionierenden algorithmischen Handelssystem stark leidet [47]. Solche Veränderungen könnten bspw. durch Ereignisse wie die Covid-19 Pandemie hervorgerufen werden, welche viele Veränderungen in der Wirtschaft ausgelöst hat [48]. Hierbei lässt sich auch feststellen, dass neue Daten, wie Todesfälle auf Grund von Covid-19, auf Grund ihrer Korrelation mit dem Aktienmarkt auch für algorithmische Handelssysteme relevant werden können [49].

2.4 Nachrichten & Sentiment im Algorithmischen Handel

Die Finanzmärkte verarbeiten Informationen, welche oft in Form von Nachrichten auftreten, und adjustieren die Kurse der jeweiligen Wertpapiere dementsprechend. Um Kursbewegungen vorherzusagen kann das sogenannte News Sentiment, auch Nachrichten-Sentiment genannt, also die Stimmung der Nachrichten, automatisiert erfasst und genutzt werden [35]. Es konnte gezeigt werden, dass der (wöchentlich) durchschnittliche Sentiment Wert eine positive Korrelation mit dem Gewinn des S&P 500 Index hat während die Anzahl der Nachrichtenartikel eine negative Korrelation mit dem Gewinn aufweist [44]. Des Weiteren konnte gezeigt werden, dass die unterstützende Nutzung von Nachrichten-Sentiment sowohl den insgesamten Gewinn als auch den risikoadjustierten Gewinn, zumindest in einzelnen Fällen, verbessern konnte. Wobei die Forschungsarbeit, dies zeigen konnte, ohne Nutzung von State-of-the-Art Sentiment Analysis Algorithmen, wie BERT [50]. Tatsächlich lässt sich auch allgemein sagen, dass Nachrichten und Nachrichten-Sentiment sich auf die Finanzmärkte auswirken und das Verhalten dieser vorhersagen können [35] [51] [52] [53]. Das tatsächliche Nutzen dieser Informationen, unterscheidet sich jedoch in den verschiedenen Lösungsansätzen und ist ausschlaggebend für die Performance [53].

Als Quelle der Informationen können unterschiedliche genutzt werden wie Social Media, Quartalsberichte oder gängige Nachrichtenagenturen [35] [51]. In dieser Bachelorarbeit wird sich mit letzterem befasst, um zu untersuchen, ob allein durch die Nutzung der Nachrichten selbst eine Unterstützung für ein algorithmisches Handelssystem gibt. Auch die Identifizierung des Sentiments kann durch verschiedene Methoden geschehen. Durch menschliches annotieren oder durch eine lexikonbasierte Methode [51]. In dieser Bachelorarbeit wird jedoch das von Google entwickelte vortrainierte Modell BERT zur Sentiment Analysis benutzt, welches in meiner Projektarbeit auf Wirtschaftsnachrichten durch Fine-Tuning abgestimmt wurde.

Basierend auf Nachrichten und ihrem Sentiment können Handelssysteme entwickelt werden, welche gewinnbringend im Markt agieren können. Wie diese Handelssysteme aufgebaut werden können wird in Kapitel 3 behandelt. Im nächsten Abschnitt soll zunächst u.a. auf die Handelsumgebung selbst, die Messung der Performance der Handelssysteme und auf die möglichen Inputdaten eingegangen werden.

2.5 Handelsumgebung, Daten & Entwicklungsprozess

In diesem Abschnitt sollen alle Themen, die das Handelssystem umgeben, behandelt werden, sodass im nächsten Kapitel der Fokus auf dem Handelssystem selbst liegen kann. Konkret soll es darum gehen, wie ein Handelssystem entwickelt und getestet werden kann, welche Daten dazu genutzt werden können sowie wie die Handelsumgebung selbst aussieht. Da in dieser Bachelorarbeit das algorithmische Handelssystem mit einem Machine Learning Modell umgesetzt wird, wird auch auf Aufgabe eines ML Modells im Handelssystem eingegangen.

Im Folgenden zunächst ein Überblick über den Workflow in einem Machine Learning gestützten Handelssystem:

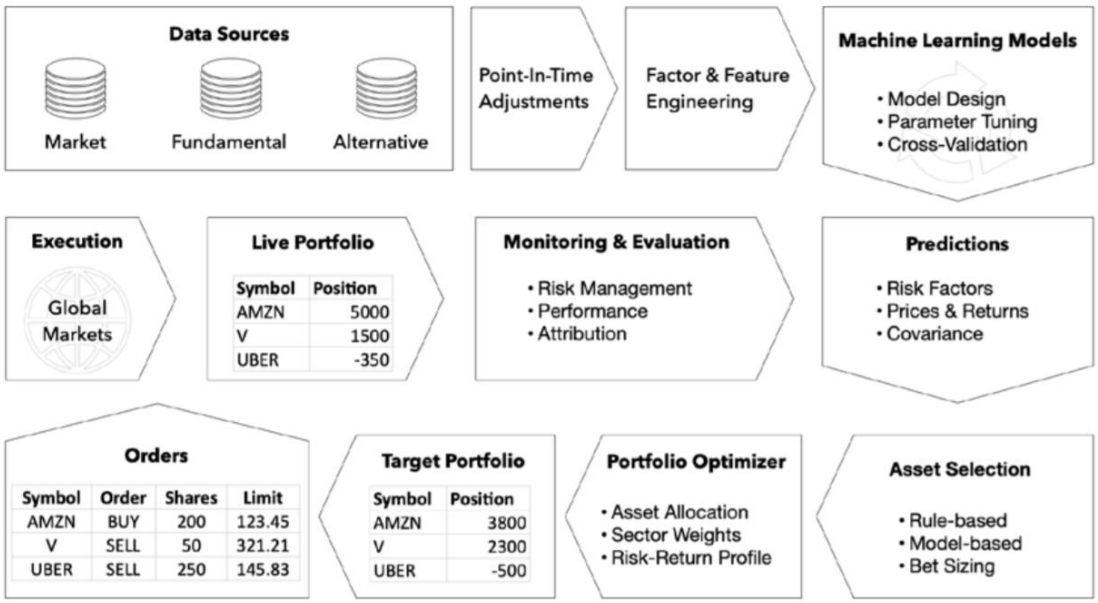

Abbildung 1: ML im Handelssystem [54]

Die Abbildung lässt sich in 4 grundsätzliche Abschnitte in der Entwicklung eines Handelssystems mit Machine Learning unterteilen: Data Sourcing, also das beschaffen der Daten, Feature Engineering, also das auswählen voraussagender Faktoren aus den Daten, Portfolio Optimization, also die Umsetzung der Features in Auswahl und Gewichtung von Wertpapieren, sowie Backtesting also das realistische Testen des Handelssystems, um sicherzustellen, dass das System auch in der Realität funktioniert [54]. Eine detaillierte Ausführung der vier Teile folgt nun.

Data Sourcing

Die Daten selbst sind entscheidend für das Handelssystem, da Machine Learning Modelle Datengetrieben sind, also die Daten effizient auszunutzen versuchen. Die Daten lassen sich in drei Kategorien unterteilen: Markdaten, Fundamentaldaten und Alternativdaten. Die Marktdaten, also die Preise und Handelsvolumina der Wertpapiere zu einem bestimmten Zeitpunkt, stellen die Handelsumgebung des Handelssystems dar und sind entscheidend nicht nur bei der Interpretation der Daten durch das Modell sondern auch bei der Ausführung der Handelsaufträge, konkret bspw. ob innerhalb des Tages gehandelt werden kann oder ob nur Eröffnungs- und Schlusskurse der Wertpapiere zur Verfügung stehen. Für hochfrequentes Handeln ist dabei eine höhere Granularität der Marktdaten nötig im Vergleich zu bspw. Buy-and-Hold Strategien. Um Daten zum Entwickeln aber auch für das Backtesting zu bekommen gibt es verschiedene, oft kostenpflichtige, API-Anbieter, wie z.B. Quandl, welche Schnittstellen für historische aber auch Live-Daten anbieten [54]. Die Fundamentaldaten umfassen alle Daten, die die wirtschaftliche Situation darstellen. Für Aktien sind dies die Finanzen des Unternehmens, der jeweiligen Industrie und auch makroökonomische Daten. Die Finanzen der im Aktienmarkt notierten Unternehmen müssen meist (mögliche länderspezifische Ausnahmen abgesehen) regelmäßig (bspw. jedes Quartal), veröffentlicht werden und sind so für jeden zugänglich. Für in den USA notierten Unternehmen sind dies konkret der 10-K, der Jahresabschlussbericht, sowie der 10-Q, der Quartalsbericht. Diese werden von der United States Securities and Exchange Commission (kurz: SEC) zur Verfügung gestellt [54]. Auf Regelungen in Ländern außerhalb der USA soll nicht eingegangen werden, um den Fokus der Arbeit zu erhalten. Des Weiteren ist zu erwähnen, dass mehrere Daten sinnvoll kombiniert werden können, um weitere wichtige Daten zu erzeugen. So kann der Preis aus den Marktdaten und die Gewinne aus den Fundamentaldaten genutzt werden, um das KGV zu errechnen, eine populäre Kennzahl zur Bewertung von Unternehmen [54]. Wie schon erwähnt können auch alternative Daten genutzt werden. Dies ist eine eher neue Kategorie der Daten im Bereich algorithmischen Handel. Sie bieten zusätzliche Informationen, welche in traditionellen Datenquellen nicht oder langsamer verfügbar sind. Alternativdaten können sehr unterschiedlich sein und beinhalten Telefonkonferenzen zu Quartalsberichten, Ladenbesuche, welche vorrausagend Aufschlüsse auf Verkaufszahlen geben können, und Satellitenbilder, welche bspw. Ernteerträge aufzeigen können. Da Nachrichten weder

Markdaten noch Fundamentaldaten sind, gehören diese auch in die Kategorie Alternativdaten.

Des Weiteren kann Machine Learning dazu eingesetzt werden, um traditionelle Methoden zu ersetzen. Statt also den intrinsischen Wert eines Unternehmens nur durch Finanzberichte zu berechnen, können Machine Learning Modelle eine Vielzahl an Faktoren in ihre Berechnung mit einbeziehen. Auch kann das Sentiment automatisiert und in Echtzeit durch Machine Learning Modelle erfasst werden [54]. Da in dieser Bachelorarbeit jedoch hauptsächlich auf Nachrichten und das Sentiment dieser eingegangen werden soll, wird sich auch darauf beschränkt, obwohl zukünftig Daten wie Suchanfragenaktivität auf Google, Social-Media-Aktivität auf Facebook, Twitter oder LinkedIn sowie Satellitenbilder durchaus interessante Ansätze sind. Das Ziel von Alternativdaten ist es einen Informationsvorteil zu erreichen, welcher bei der Nutzung dieser Informationen beim Handeln zu einer höheren Rendite führt. Um dieses zu erreichen können alternative Daten allein oder in Kombination genutzt werden. Die Wahrscheinlichkeit, dass die Nutzung von einem jeweiligen alternativen Datensatz zu einer höheren Rendite richtet sich in der Regel nach der Exklusivität und Schwierigkeit der Bearbeitung der Daten. Desto exklusiver und desto schwieriger die Verarbeitung, desto wahrscheinlicher ist es, dass die Daten die Rendite erhöhen werden. Bei zunehmender Konkurrenz kann also über Zeit dieser Vorteil schwinden, weshalb ein Fokus auf besondere Exklusivität oder auf die besonders komplexe Verarbeitung von Daten sinnvoll sein kann [54].

Feature Engineering

Beim Feature Engineering im Kontext von algorithmischem Handel wird von Alpha Faktoren gesprochen. Alpha bezeichnet hierbei die Überrendite, die gegenüber einem Vergleichsindex mit diesem Faktor erzielt werden kann. Alpha Faktoren sind die Faktoren, die Handelssignale erzeugen, welche dieses Alpha produzieren. Alpha Faktoren sind Transformationen von Marktdaten, Fundamentaldaten oder Alternativdaten, wobei eine oder mehrere Daten kombiniert werden können. Ein Beispiel von einem Faktor, welcher aus Daten entsteht, ist der bereits erläuterte SMA. Machine Learning Modelle können zwar auch die Daten selbst effektiv nutzen, um Handelssignale zu extrahieren, jedoch bleiben Alpha Faktoren oder eine Kombination dieser hilfreich [54]. Im Folgenden gehe

ich auf einige etablierte Alpha Faktoren ein. Zum einen kann hierfür das Momentum genutzt werden, welches bspw. durch den RoC berechnet werden kann. Auch Sentiment, wie z.B. von Nachrichten oder Social Media kann hilfreich sein. Eine andere Kategorie von Faktoren sind die Value Faktoren, welche unterbewertete Anlagemöglichkeiten identifizieren sollen. Hierbei wird der „faire“ Wert eines Unternehmens bestimmt. Fällt der Preis unter diesen Wert wird gekauft, fällt er darüber wird verkauft. Hierzu kann bspw. die Cashflow-Rendite oder das Kurs-Gewinn-Wachstums-Verhältnis (kurz: PEG), also das Verhältnis vom KGV zum Wachstum. Eine andere Art von Faktoren versucht die Qualität einer möglichen Anlage zu erfassen. Qualität meint hierbei die Profitabilität, Sicherheit und Stabilität einer Anlage. Hierzu kann bspw. die Rentabilität (engl.: Return on Equity, kurz: RoE), welche folgendermaßen berechnet werden kann: $Rentabilität = \frac{Gewinn}{Kapital}$. Auch der Verschuldungsgrad (engl.: debt to equity ratio) kann nützlich sein [54].

Welche Alpha Faktoren oder Kombination von Alpha Faktoren in dem in dieser Bachelorarbeit zu entwickelten algorithmischen Handelssystem zu einem besonders hohen Alpha führen können, gilt es in dieser Bachelorarbeit zu identifizieren. Die genannten Faktoren sind jedoch gut etabliert und somit mindestens eine erste Orientierungshilfe. Welche Faktoren im System verwendet bzw. getestet werden wird im späteren Verlauf der Arbeit diskutiert und Testergebnisse präsentiert. Es lässt sich jedoch sagen, dass allgemein die Nutzung von mehreren Alpha Faktoren, welche dann mit einem Machine Learning Modell optimiert in Investmententscheidungen überführt werden, sinnvoll ist [54].

Portfolio Optimization

Um ein Portfolio zu optimieren muss der Gewinn und das Risiko eines Portfolios gemessen werden, damit evaluiert werden kann, ob und wie gut die jeweilige Strategie funktioniert. Dies beinhaltet das Testen auf historischen Daten, das sogenannte Backtesting, um die Strategie zu optimieren, als auch das Validieren der Performance auf dem Modell noch unbekannten Daten, das sogenannte Forward-Testing. Typischerweise wird die Performance des Portfolios dann mit einem Benchmark verglichen wie bspw. den S&P 500. Im Folgenden werde ich die gängigste Metrik vorstellen, die hierfür verwendet werden kann. In der Notation ist R eine Zeitreihe von Renditen wie folgt $R = (r_1, \dots, r_T)$ von Zeiteinheit 1 bis T. Der risikofreie Zinssatz (engl.: risk free rate) wird mit R_f notiert. Die

Überrendite wird folgendermaßen notiert: $R_e = R - R_f$. Bei dem Sharpe Quotienten (engl.: Sharpe Ratio) wird die erwartete Überrendite mit der Volatilität (Standardabweichung) verglichen. Zur Berechnung der erwarteten Rendite werden historische Daten wie folgt benutzt [54]:

$$\hat{\mu}_{R^e} = \frac{1}{T}\sum_{t=1}^{T} r_t^e$$

$$\hat{\sigma}_{R^e}^2 = \frac{1}{T}\sum_{t=1}^{T} (r_t^e - \hat{\mu}_{R^e})^2$$

$$SR = \frac{\hat{\mu}_{R^e} - R_f}{\hat{\sigma}_{R^e}^2}$$

$\hat{\mu}_{R^e}$ ist also die erwartete Durchschnittsrendite über den Zeitraum T, wobei diese, wie schon erwähnt, sich aus dem Durchschnitt der historischen Renditen r_t^e errechnet. Diese historische Rendite wird als geschätzte zukünftige Rendite gesehen. $\hat{\sigma}_{R^e}^2$ ist die Standardabweichung, wobei die Durchschnittsrendite von der erwarteten Rendite subtrahiert wird, um zu sehen wie stark sich die zu erwartende Rendite von der zu erwartenden Durchschnittsrendite unterscheidet. Das Ergebnis wird dann quadriert, um den Unterschied zwischen beiden Werten als positive Zahl zu erhalten. Dieses Ergebnis wird dann quadriert, um größere Unterschiede besonders hervorzuheben bzw. in Gewicht fallen zu lassen. Zum Schluss wird der Durchschnitt über den Zeitraum T errechnet. Um die Sharpe Ratio zu berechnen, wird also die Durchschnittsrendite, abgezogen der risikofreien Rendite, geteilt durch die Volatilität, gemessen durch die Standardabweichung, berechnet, damit man ein Verhältnis zwischen Überrendite und dem Risiko, hier ja gemessen durch die Standardabweichung, erhält. [54]. Eine Sharpe Ratio >1 bedeutet hierbei, dass risikoadjustiert eine Überrendite zur risikofreien Rendite erwirtschaftet wurde. Eine Sharpe Ratio <1 bedeutet das gegenteilige [55]. Hierbei sei zu erwähnen, dass die Volatilität das Risiko des Portfolios messen soll [54]. Jedoch ist dabei zu beachten, dass dies nur eine Messung des Risikos und nicht das Risiko selbst darstellt. Das Risiko ist als potenzieller Verlust definiert, also wenn z. B. das Wertpapier zu einem niedrigeren Preis im Vergleich zum Einkaufspreis verkauft wird oder durch äußere Umstände das Investment oder den gesamten Sektor stark negativ beeinflussen [56], bspw. durch den Lockdown in einer Pandemie. Auch weitere Metriken wie die Information Ratio, welche ähnlich zu der

Sharpe Ratio ist, jedoch die Rendite eines Benchmarks statt des risikofreien Zinssatzes nimmt [54]. Da jedoch die Sharpe Ratio eine anerkannte Metrik zur Messung der Performance eines Portfolios ist, welche auch in anderen Forschungen zu finden ist, wird auf die Darlegung anderer Metriken an dieser Stelle verzichtet, um den Fokus der Bachelorarbeit zu wahren.

Bei der Portfolio Optimierung ist es wichtig, das Risiko sowie den Gewinn zu managen. Hierbei ist die Selektion einer Position, also eines Wertpapieres, und seine Größe entscheidend, um das Risiko zu minimieren und gleichzeitig den zu erwartenden Gewinn zu maximieren. Des Weiteren bedarf es über Zeit Neugewichtungen, falls sich die vorher festgelegten Gewichtungen verändert haben. Hierbei ist der Ausgangspunkt die modern portfolio theory (MPT) von Howard Markowitz. Markowitz konnte zeigen, dass das Risiko eines Portfolios, gemessen durch die Standardabweichung, von der Kovarianz der Gewinne und Gewichtung der Positionen abhängt [54], also wie die Positionen im Portfolio sich zueinander bei steigenden und fallenden Kursen verhalten. Hierbei ist das Konzept des efficient frontiers entscheidend. Dies beschreibt das Verhältnis des zu erwartenden Gewinnes zu dem Risiko eines Portfolios. Der efficient frontier stellt dann das theoretische Optimum dar. Das Ziel ist es den zu erwartenden Gewinn bei einem gegebenen Risiko zu maximieren. Es existieren verschiedene Ansätze wie man ein Portfolio optimieren kann, basierend auf der modern portfolio theory. Auch hier wird, um den Fokus der Bachelorarbeit zu wahren, eine simple, aber effektive, Strategie vorgestellt. Der 1/N Portfolio Ansatz ist eine naive Lösung zu dem Problem der Positionsgewichtung. Hierbei wird jede Position gleich gewichtet. Victor DeMiguel, Lorenzo Garlappi und Raman Uppal konnten in ihrer 2009 erschienenen Forschungsarbeit zeigen, dass der 1/N Portfolio Ansatz auf verschiedenen Datensätzen eine bessere Sharpe Ratio als viele verschiedene komplexe Lösungsansätze [54] [57]. Eine weitere Forschungsarbeit von Johannes Bock aus dem Jahre 2018 konnte diese Ergebnisse bestätigen [58]. Die erwarteten Gewinne vorherzusagen scheint zu schwierig zu sein [54] [58], woraufhin Fehlinvestitionen zu Verlusten führen, welche nicht ausgeglichen werden können [54]. Deshalb scheint eine Verbesserung der Vorhersagen oder die Entwicklung eine Strategie, die die zu erwarteten Gewinne nicht benötigt, sinnvoll [58]. Wie schon erwähnt geht aus der modern portfolio theory hervor, dass sowohl die Gewichtung als auch die Selektion der Positionen relevant sind. Die genannten Strategien zur Portfoliooptimierung gehen jedoch von einem gegebenen Portfolio aus, welches optimiert werden soll. Die Selektion der Positionen

geschieht durch äußere Entscheidungen sei es durch Menschen, generierte Trading Signale oder Machine Learning Systemen. Auf letzteres wird im Laufe dieser Bachelorarbeit noch eingegangen.

Backtesting

In diesem Abschnitt stelle ich das Backtesting vor, also das Testen des algorithmischen Handelssystems auf historischen Daten. Dabei iteriert die Backtesting Engine über die historischen Daten und gibt die jeweils aktuellen Daten in das algorithmische Handelssystem, verarbeitet seine Handelsaufträge und speichert die daraus resultierenden Positionen und ihre Werte. Hierbei lässt sich zwischen dem vectorized backtesting und dem event-driven backtesting unterscheiden [54]. Bei dem vectorized backtesting werden für jeden Zeitpunkt die Positionen, welche aus den Entscheidungen des algorithmischen Handelssystems entstehen, sowie die Rendite aller möglichen Positionen in Matrizenform gespeichert. Die beiden resultierenden Matrizen können multipliziert werden, wodurch die Rendite des Handelssystems zu jedem Zeitpunkt entsteht. Durch die Matrizenform ist diese Methodik besonders effizient. Die gleiche Berechnung kann auch durch eine For-Schleife umgesetzt werden, was jedoch bei großen Datensätzen besonders rechenaufwändig ist. Durch die Summierung der Rendite jedes Zeitpunktes erlangt man die Rendite über den gesamten Zeitraum [54] [59]. Diese Methodik ist gut dafür geeignet, um schnell und einfach eine ungefähre Renditeerwartung durch die Benutzung des algorithmischen Handelssystems zu erhalten, es hat jedoch auch mehrere Nachteile. Es werden keine Handelskosten oder ähnliches mit einbezogen und es muss manuell sichergestellt werden, dass die Zeitpunkte aufeinander abgestimmt sind [54]. Event-driven backtesting hingegen versucht die Handelsumgebung über die Zeit realistisch zu simulieren, indem z.B. historische Kalender genutzt werden, welche angeben wann gehandelt werden kann oder ab wann welche Wertpapiere zur Verfügung standen. Dies ermöglicht auch den Echtzeithandel. Hierbei produziert das algorithmische Handelssystem Handelsentscheidungen, welche dann von der Backtesting Engine, Handelskosten usw. mit einbezogen, umgesetzt werden [54]. An dieser Stelle soll dies nur als Überblick dienen. Die tatsächlich genutzten Technologien werden im Kapitel zur Umsetzung behandelt.

Allgemein ist es jedoch wichtig beim Backtesting bestimmte Dinge zu beachten, um eine faire Evaluation des algorithmischen Handelssystems zu gewährleisten. Dies ist zum einen die Vermeidung eines look-ahead-bias, also, dass Informationen dem Handelssystem zur Verfügung stehen bevor es in der Realität bekannt war. Dies beinhaltet Informationen über den Gewinn pro Aktie eines Quartals oder den Zeitpunkt eines Aktiensplits. Auch ist die Wahl der Zeitperiode des Backtesting relevant. Es sollte darauf geachtet werden, dass die ausgewählte Zeitperiode dem aktuellen bzw. zukünftigen Markt in Aspekten wie Volatilität und Handelsvolumen ähnelt. Auch sollten nur Daten genutzt werden, die auch voraussichtlich in der Realität zu dem Zeitpunkt zur Verfügung stehen würden [54].

Kapitel 3: Lösungsansätze

In diesem Kapitel möchte ich die möglichen Lösungsmöglichkeiten für das Ziel dieser Bachelorarbeit vorstellen. Auf Grund der Vielzahl an möglichen Algorithmen, die angewandt werden können, kann nicht auf alle im Detail eingegangen werden, jedoch wird versucht einen guten Überblick zu schaffen, auf besonders relevante Ansätze näher einzugehen sowie die verschiedenen Lösungsansätze zu diskutieren. Insbesondere soll auf Lösungen mit der Nutzung von Machine Learning eingegangen werden, wobei der Hauptfokus auf Deep Learning Ansätzen liegen soll. Es folgt also eine allgemeine Einführung von Machine Learning in algorithmischen Handelssystemen und ein tiefer Einblick in Deep Learning Lösungen wie Time Series Forecasting mit LSTMs sowie eine Diskussion welcher Lösungsansatz am passendsten zur Problemstellung ist.

3.1 Machine Learning in Algorithmischen Handelssystemen

Wie schon erwähnt werde ich in diesem Kapitel näher auf die Rolle von Machine Learning in algorithmischen Handelssystemen eingehen. Hierbei werden Grundlagen von Machine Learning vorausgesetzt. In algorithmischen Handelssystemen können verschiedene Machine Learning Methoden für verschiedene Aufgaben einsetzen. Dies bedeutet, dass in dieser Bachelorarbeit eine Entscheidung getroffen werden muss, welchen Machine Learning Lösungsansatz für welche Aufgabe im algorithmischen Handelssystem umgesetzt werden soll. Machine Learning lässt sich nach Tom M. Mitchell wie folgt definieren: „Ein Computerprogramm soll aus Erfahrung E in Bezug auf eine Klasse von Aufgaben T und Leistungsmaß P lernen, wenn sich seine Leistung bei Aufgabe T, gemessen durch P, mit Erfahrung E verbessert.“ [60]. Dies umfasst sowohl Machine Learning Modelle wie lineare Regression, die ohne ein neuronales Netz funktionieren, sowie Deep Learning Modelle, die auf einem neuronalen Netz mit mehreren verdeckten Schichten besteht [61]. Ersteres, was in dieser Arbeit als klassisches Machine Learning bezeichnet wird, kann im Kontext der algorithmischen Handelssysteme, wie schon erwähnt, für verschiedene Aufgaben genutzt werden. Lineare Regression kann für Time Series Forecasting genutzt werden, also im Kontext von algorithmischem Handel von Wertpapieren, um zukünftige Gewinne vorauszusagen, wodurch dann Kauf- oder Verkaufsentscheidungen getroffen

werden können. Eine weitere Möglichkeit sind Auto-Regressive Integrated Moving Average (kurz: ARIMA) Modelle. Dies sind Modelle, bei denen zukünftige Werte basierend auf historischen vorhergesagt werden. Es wurde jedoch gezeigt, dass Deep Learning Modelle wie LSTMs eine bessere Performance besitzen als klassische Machine Learning Modelle wie ARIMA [62] [63]. Es lässt sich aber auch allgemein sagen, dass Deep Learning Modelle eine bessere Performance aufweisen können im Vergleich zu klassischen Machine Learning Modellen bzw. Shallow Neural Networks [64] [65]. Zwar sind die Entscheidungen von Deep Learning Modellen oft schwer nachvollziehbar bzw. erklärbar, jedoch stehen im Kontext dieser Bachelorarbeit viele Daten mit präzisen Labels zur Verfügung, wobei diese Daten oft sehr komplex sein können, weshalb ich hierbei einen Deep Learning Ansatz für sinnvoll halte [66]. Auf die möglichen Deep Learning Lösungsmöglichkeiten gehe ich im folgenden Abschnitt ein.

3.2 Deep Learning Ansätze zum Algorithmischen Handel

Wie schon erwähnt gibt es für Machine Learning verschiedene Einsatzmöglichkeiten in einem algorithmischen Handelssystem. Für die grundlegende Funktion eines solchen Systems ist jedoch das Handeln selbst bzw. das Erzeugen von Handelssignalen notwendig. Eine Machine Learning basierte Portfoliooptimierung bspw. wäre dann darauf aufbauend. Deshalb möchte ich in dieser Bachelorarbeit mich auf die Erzeugung von Handelssignalen bzw. das Handeln selbst mit Machine Learning fokussieren, da dies die grundlegend notwendige Funktion eines algorithmischen Handelssystems darstellt. Hierfür gibt es zwei Lösungsansätze, zum einen Time Series Forecasting, also das Vorhersagen von zukünftigen Gewinnen (im Kontext vom Wertpapierhandel), um darauf basierend Handelsentscheidungen zu treffen, und zum anderen das Handeln mit Reinforcement Learning, wobei hier das System direkt Entscheidungen trifft. Bei Reinforcement Learning tätigt der Agent in einer Umgebung Aktionen, um seine Belohnung zu maximieren. Das Lernen geschieht hierbei während der Agent mit seiner Umgebung interagiert. Damit Reinforcement Learning gut funktioniert, müssen bei der jeweiligen Aufgabe alle Variablen quantifizierbar dem Modell vorliegen, die Funktion zur Berechnung der Belohnung muss klar definiert werden können, Fehler haben keine/marginale Konsequenzen und der Agent hat viel Zeit zu lernen [67]. Im Kontext dieser Bachelorarbeit könnte eine passende

Belohnung, nämlich der Gewinn, definiert werden sowie Konsequenzen aus Fehlern vermieden werden, indem nicht im realen Aktienmarkt gehandelt wird, sondern der Markt nur simuliert wird. Jedoch kann die Umgebung für das Modell nicht vollständig definiert werden, sondern das Modell würde voraussichtlich nur einen Teil der Umgebung als Informationen erhalten, da diverse Finanzdaten existieren, welche nicht alle öffentlich zugänglich sind, besonders weil, wie schon in Kapitel 2 erläutert, Investoren, um ein Alpha zu erlangen, Informationen privat halten. Des Weiteren ist die Menge an verfügbaren Daten begrenzt und somit erschwert dies zumindest das Lernpotenzial des Reinforcement Learning Modell. Da die Anforderungen in der Realität, so wie auch hier, oft nur teilweise erfüllt sind, fand Reinforcement Learning bis jetzt nur in kontrollierten Umgebungen, wie z.B. in dem Spiel Go, große Erfolge [67]. Ich halte eine sinnvolle Anwendung von Reinforcement Learning für algorithmische Handelssysteme zwar nicht für ausgeschlossen, aber zumindest wird ein großer Aufwand nötig sein, um die jetzigen Limitierungen zu überwinden [67]. Aus diesen Gründen halte ich eine Lösung durch Reinforcement Learning basierend auf dem aktuellen Forschungsstand für diese Bachelorarbeit nicht für besonders vielversprechend.

Im Folgenden möchte ich hierzu auf einige möglichen Architekturen eingehen und wie das Time Series Forecasting mit diesen funktioniert. Hierbei handelt es sich hauptsächlich um Sequenzmodelle, konkret also RNNs und LSTMs.

Recurrent Neural Network (RNN) – Zu Deutsch rekurrentes neuronales Netz wurde erstmals 1990 vorgestellt [68]. Anders als Feed-forward Neural Networks, erhält das neuronale Netz als Input eine Sequenz (hier also alle Inputvariablen zu jedem Zeitpunkt), die nach und nach in das RNN übergeben. Dabei entsteht jedes Mal ein Zwischenergebnis. Dieses Zwischenergebnis wird als Zustand des RRNs bezeichnet. Der darauffolgende Zustand berechnet sich durch die Gewichtung des aktuellen Zustands und durch den nächsten Input, wobei diese mit der tanh Aktivierungsfunktion überführt werden. Dies lässt sich folgendermaßen als Gleichung Darstellen:

$h_j = \tanh(W_{hh} h_{j-1} + W_{xh} x_j)$. Wobei h_j der zu berechnende Zustand ist, h_{j-1} der vorherige Zustand und x_j der aktuelle Input sowie W die Gewichtungsmatrizen. Bildlich kann das RNN folgendermaßen dargestellt werden [69]:

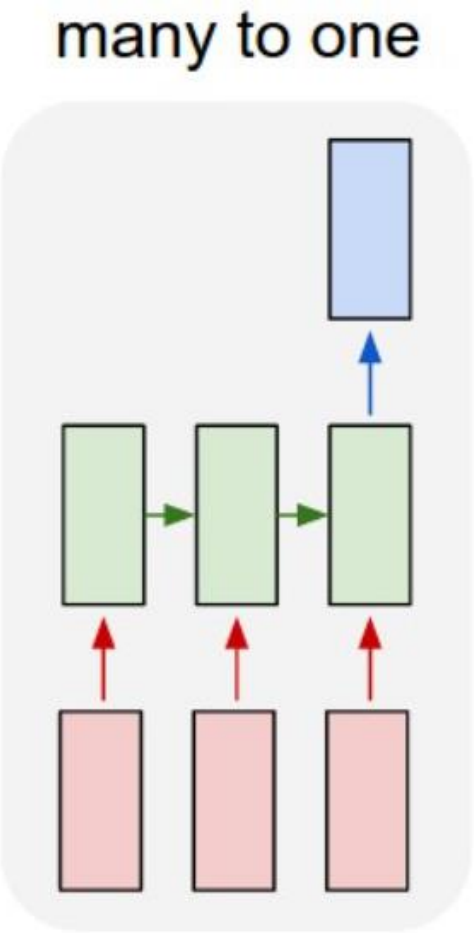

Abbildung 2: Many-to-One RNN [77]

Die verschiedenen Teile nach jedem Schritt werden rekurrente Zellen genannt und besitzen, wie oben erwähnt, jeweils einen Zustand, welcher durch die eben erwähnte Aktivierungsfunktion überführt wird.

Abbildung 2 zeigt ein Many-to-One RNN aber tatsächlich wird nach jedem Schritt ein Output generiert, bei Many-to-One RNNs ist nur der finale Output entscheidend für die Klassifizierung. Die Output Vektoren können auf folgende Weise berechnet werden:

$$\hat{y}_t = W_{hy}^T h_t$$

Hierbei ist W_{hy}^T die Gewichtungsmatrix vom hidden state h_t nach dem Outputvektor $\hat{y}_t$. Zuletzt können wir auch nach jedem Schritt den Verlust bzw. Loss L_t berechnen, wobei

der insgesamte Loss die Summe aller L_t ist. Daraus ergibt sich folgende Struktur:

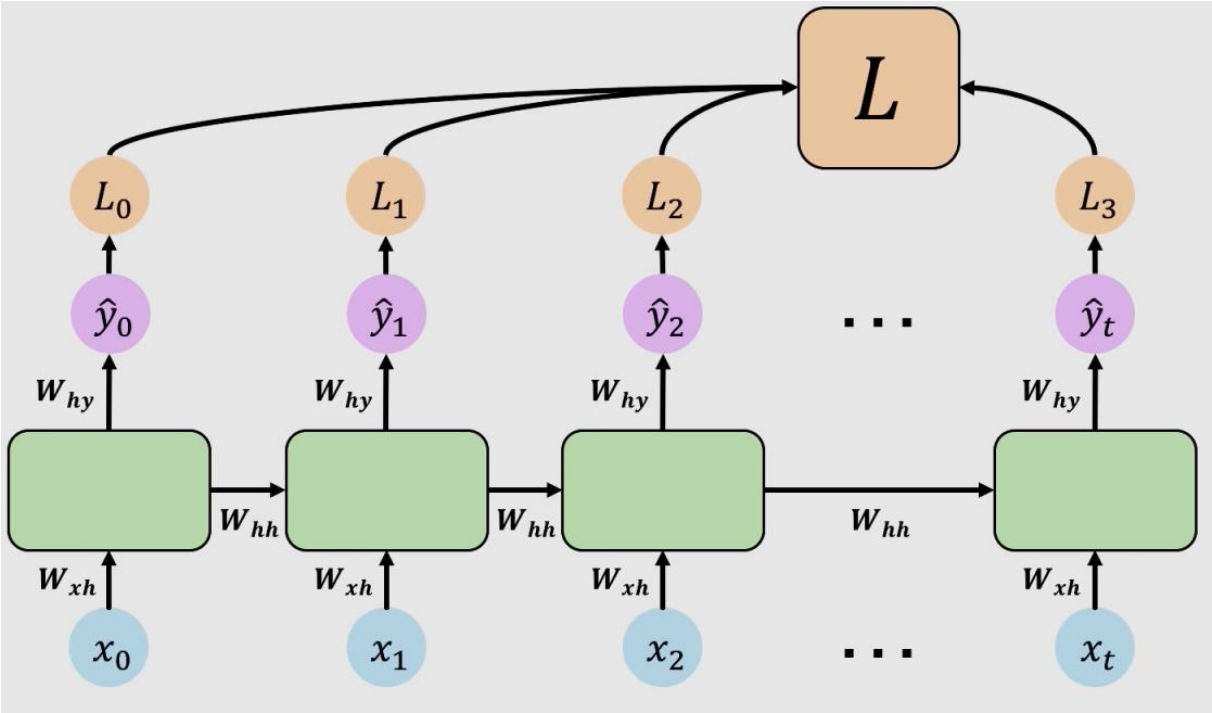

Abbildung 3: RNN mit Loss [69]

Probleme von RNNs:

Vanishing Gradient – Abhängigkeiten von Inputs gehen nach ca. 10-20 Zeitschritten verloren, weswegen Langzeitabhängigkeiten nicht gut durch RNNs dargestellt werden können [77]. Der Grund ist, dass bei der Optimierung des Netzes Backpropagation through time eingesetzt wird. Hierbei werden viele Gradiente, die kleiner als 1 sind, miteinander multipliziert, wodurch insgesamt der Gradient gegen 0 geht und somit verschwindet. Dies kann ein Problem darstellen, falls Informationen gebraucht werden, die weiter hinten im Netzwerk liegen [69]. Die Ursache hierfür ist, dass bei der Backpropagation die Ableitung der Aktivierungsfunktion tanh berechnet werden soll, welche über eine große Wertespanne nahe Null liegt. Soll nun Backpropagation über mehrere Zeitschritte berechnet werden, so müssen die Gradienten der jeweiligen Zeitschritte miteinander multipliziert werden, um per Backpropagation den Loss tiefer im Netz zu berechnen. Durch die Multiplikationen mit sehr kleinen Werten verschwindet schlussendlich der Loss [77].

Exploding Gradient – Bei sehr tiefen neuronalen Netzwerken kann es passieren, dass viele Gradiente, die größer als 1 sind, miteinander multipliziert werden und so sehr groß

werden, sprich explodieren [69]. Dies führt zu einem instabilen Lernprozess [70]. Die Ursache hierfür ist wieder die Aktivierungsfunktion. Statt tanh kann ReLU genutzt werden, welche den Wert 1 bei jeglichem positiven Input hat. In der Praxis kann dies jedoch zu einem explodierenden Gradienten bei der Multiplikation der Gradienten über die Zeitschritte führen [77]. Gemindert werden kann dieses Problem zum Beispiel mit Hilfe von „gradient clipping“, wobei die Gradienten in einem vorher bestimmten Wertebereich gehalten werden, was jedoch eine Parteilichkeit darstellt, da Werte außerhalb des festgesetzten Wertebereichs nicht akzeptiert werden [71].

Long Short Term Memory (LSTM):

Um die Probleme des Vanishing / Exploding Gradients von RNNs zu lösen wurde eine neue Struktur, das Long Short Term Memory kurz LSTM, entwickelt und 1997 von zwei Deutschen erstmals vorgestellt [72]. LSTMs sind ähnlich wie RNNs, besitzen jedoch, anders als diese, sogenannte Gated Cells [68]. Ein Standard RNN hat lediglich eine Aktivierungsfunktion im Inneren jeder Zelle. Mit einer komplexeren Struktur innerhalb der Zelle versucht das LSTM vier Ziele zu erreichen: Irrelevante Informationen des letzten Zustandes zu vergessen, relevante neue Informationen zu speichern, selektiv den Zellenstatus zu aktualisieren und kontrolliert Informationen in die nächste Zelle übergeben [69].

1. Vergessen: Im ersten Schritt wird versucht irrelevante Informationen herauszufiltern. Hierfür wird der interne Zustand (auch history h genannt), der vorherigen Zelle, und der aktuelle Input in eine Sigmoidfunktion gegeben. Der Output ist eine Zahl zwischen 0 und 1, wobei 1 „vollständig behalten“ und 0 „vollständig vergessen“ bedeutet. Die Funktion hat zusätzlich noch eine Gewichtung und ein Bias. Dieser Schritt wird als „forget gate layer“ bezeichnet. Die Gleichung sieht wie folgt aus [69] [73]:

$$f_t = \sigma(W_t[h_{t-1}, x_t] + b_f)$$

Außerdem lässt sich dies folgendermaßen graphisch darstellen:

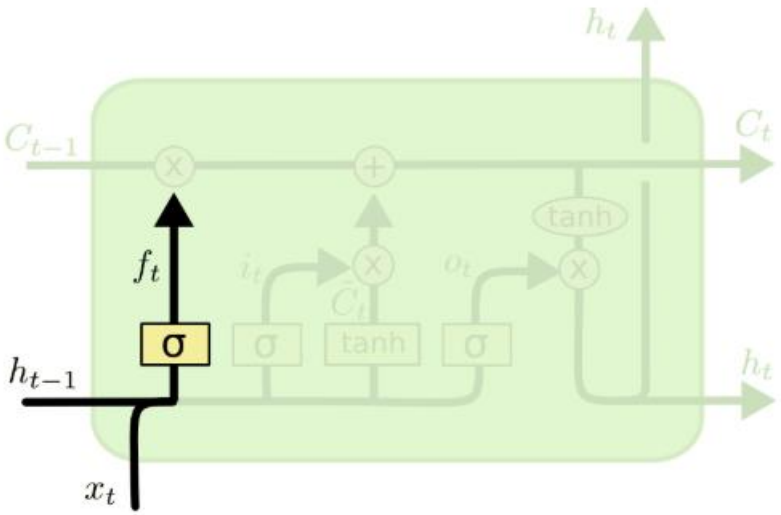

Abbildung 4: LSTM Schritt 1 – Vergessen [73]

2. Speichern: Bei diesem Schritt soll gefiltert werden, welche Informationen gespeichert werden sollen, also welche Informationen in die Aktualisierung des Zustandes beinhaltet sind. Hierbei werden zwei Ebenen durchlaufen: Zum einen wird durch eine Sigmoidfunktion entschieden, welche Werte aktualisiert werden sollen. Dieser Teil wird als „input gate layer“ bezeichnet. Als nächstes wird mit Hilfe einer tanh-Funktion ein Vektor, mit potenziellen Werten, die dem Zustand hinzugefügt werden könnten. Die beiden Gleichungen hierfür und die Grafik können wie folgt dargestellt werden [69] [73]:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

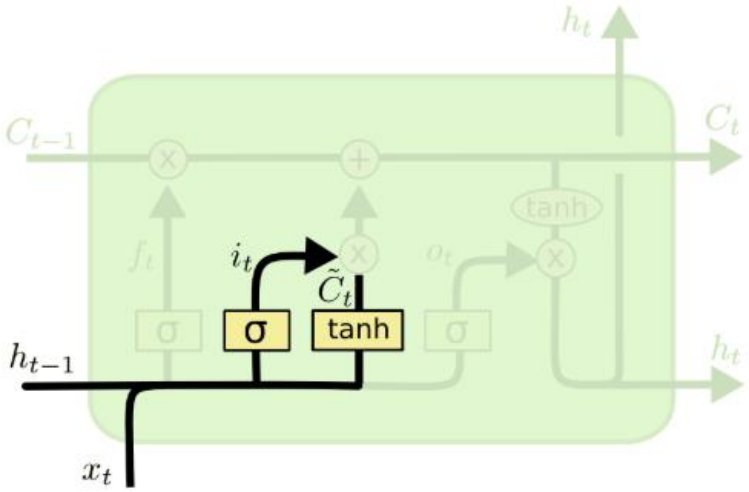

Abbildung 5: LSTM Schritt 2 – Speichern [73]

3. Aktualisieren: Nun muss der alte Zustand C_{t-1} in den aktuellen Zustand C_t überführt werden. Hierzu werden die Informationen aus den vorherigen beiden Schritten genutzt. Die beiden Zwischenergebnisse aus Schritt zwei werden miteinander multipliziert, genauso wie das Ergebnis aus Schritt 1 mit dem Zustand C_{t-1}. Die Addition dieser beiden Multiplikationen ergibt dann den neuen Zustand. Als Gleichung und Grafik sieht dies wie folgt aus [69] [73]:

$$C_t = f_t * C_{t-1} + i_t * \tilde{c}_t$$

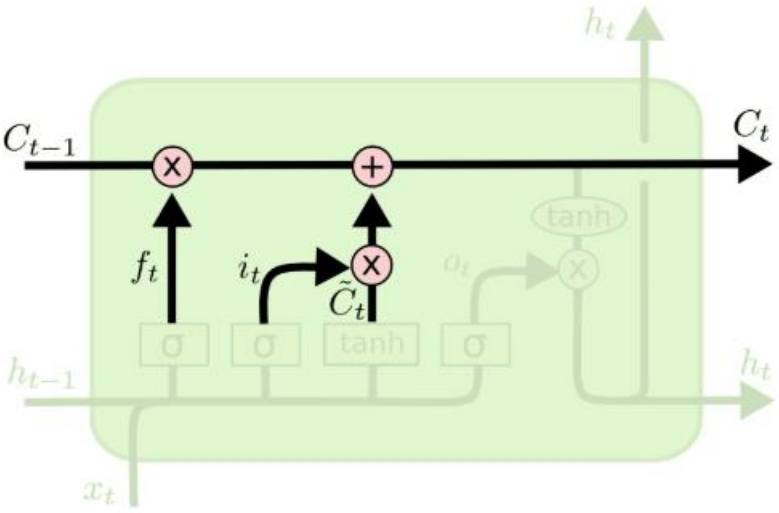

Abbildung 6: LSTM Schritt 3 – Aktualisieren [73]

4. Output: Im letzten Schritt wird entschieden, was vom Zustand als Output gewählt wird. Die Informationen des Zustandes werden also nochmals gefiltert. Hierzu wird die Sigmoidfunktion auf den vorherigen Zustand sowie den aktuellen Input angewendet und mit dem Ergebnis der tanh-Funktion des aktuellen Zustandes multipliziert. Es folgt die Gleichung und Grafik hierzu:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

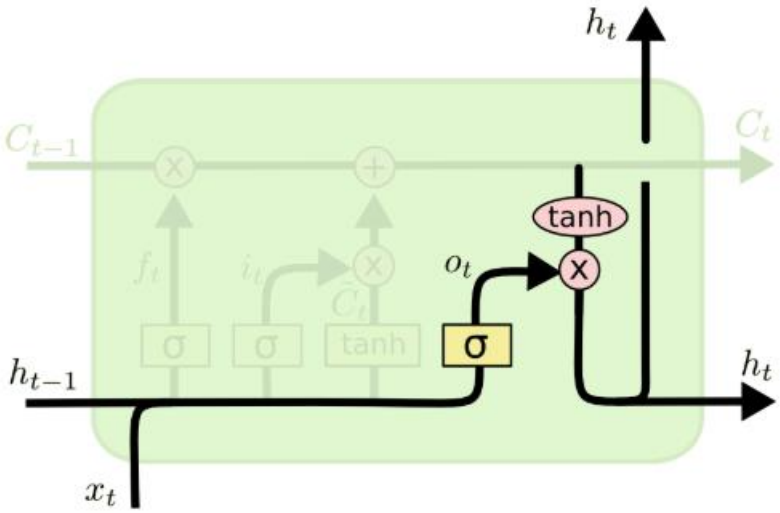

Abbildung 7: LSTM Schritt 4 – Output [73]

Varianten von LSTM – Es gibt sehr viele Varianten von LSTMs. Beispielsweise kann man in die Sigmoidfunktionen zusätzlich den aktuellen Zellenzustand geben. In einer Forschungsarbeit wurden mehr als 10.000 RNN Architekturen untersucht. Hierbei wurde festgestellt, dass einige LSTMs für bestimmte Aufgaben besser geeignet sind [74]. Wobei andere Forschungen ergaben, dass die beliebtesten LSTM Varianten mehr oder weniger gleich sind [75].

Probleme von LSTMs – LSTMs beheben einige Probleme, die RNNs haben. Namentlich Vanishing und Exploding Gradient. Wie bereits erläutert treten diese Probleme durch die sequenzielle Struktur auf. Hierbei verschwindet oder explodiert der Wert des Gradienten, da entweder kleine oder große Werte sehr oft miteinander multipliziert werden. LSTMs

verringern diese Probleme durch ihre Gated Cells, beheben sie jedoch nicht vollständig, da die grundsätzliche sequentielle Struktur bestehen bleibt und so die gleichen Probleme bei sehr langen Sequenzen bzw. sehr tiefen Netzen auftreten können. Zudem sind LSTMs, und auch RNNs, im Training und der Anwendung sehr speicheraufwändig, da man pro Zelle und pro Zeitschritt vier lineare Layer braucht [76].

Transformer:

Eine weitere mögliche Architektur, die zur Lösung von Time Series Forecasting genutzt werden kann, ist die Transformer Architektur. Ohne hier die Architektur im Detail zu erläutern soll gesagt sein, dass es sich hierbei um eine Encoder-Decoder Architektur handelt, bei der ein Attention Mechanismus angibt, welchem Teil der Sequenz besonders viel Aufmerksamkeit geschenkt werden soll. Ursprünglich wurde diese Architektur für den NLP Bereich eingesetzt, also für eine Sequenz aus Wörtern bzw. Buchstaben [78]. Es konnte jedoch gezeigt werden, dass diese Architektur auch für Zeitreihendaten hilfreich ist, wobei viele Forschungen erst seit ca. 2018/2019 betrieben wurden. Die Transformer Architektur bietet bei Time Series Forecasting jedoch eine bessere Performance im Vergleich zu sequenziellen Modellen wie LSTMs [62]. Dies gilt auch für Zeitreihendaten im Finanzsektor [79]. In dieser Bachelorarbeit entscheide ich mich jedoch gegen die Nutzung einer Transformerarchitektur aus zwei Gründen. Zum einen sind auf Grund der Neuheit dieser Architektur im Bereich Time Series Forecasting die zur Verfügung stehenden Quellen recht begrenzt im Vergleich zu den stark verbreiteten LSTMs. Und zum anderen zeigt meine Erfahrung aus meiner Projektarbeit, dass Transformerarchitekturen recht speicheraufwändig im Training sind. Es ist mir in dieser Bachelorarbeit nicht möglich Rechenarchitekturen wie Google Colab zu benutzen, da das Hochladen meiner recht großen Datensätze zu lange dauern würde, um effektiv forschen zu können, weshalb ich meinen lokalen Rechner mit einer GTX 1070 mit 8GB VRAM nutzen werde. Dies werde ich noch detaillierter im Kapitel Planung erläutern, jedoch war an dieser Stelle eine Begründung meiner Wahl einer Machine Learning Architektur relevant.

Kapitel 4: Planung

In diesem Kapitel wird unter der Berücksichtigung der Erkenntnisse der vorherigen Kapitel dargelegt, welche Algorithmen und Techniken am besten im Kontext dieser Projektarbeit geeignet sind und wie diese eingesetzt werden können, um das Ziel der Bachelorarbeit zu erreichen. Hierbei wird auch zusammenfassend auf alle relevanten Aspekte eingegangen, die schon bereits diskutiert wurden.

4.1 Funktionale Eigenschaften

Auswahl einer Lernmethode

Hierbei sei zunächst erwähnt, dass in dieser Bachelorarbeit zunächst Machine Learning eingesetzt wird, um die grundlegende Funktion eines algorithmischen Handelssystems, nämlich das Handeln selbst, zu ermöglichen. Dabei gibt es zwei grundsätzliche Ansätze: Zum einen eine Ende-zu-Ende Möglichkeit mit Reinforcement Learning, wobei das Machine Learning Modell basierend auf den Inputdaten direkt Handelsentscheidungen trifft und der daraus resultierende Gewinn oder Verlust als Belohnung/Bestrafung für das Modell fungiert, um diesem Feedback im Lernprozess zu geben. Anders als bei der erfolgreichen Anwendung von Reinforcement Learning im Beispiel von AlphaGo [80], können jedoch im Falle von algorithmischem Handel von Wertpapieren einige Anforderungen eines Reinforcement Learning Modells, wie eine vollständig definierte Umgebung oder eine große verfügbare Menge von Daten, nicht oder nur teils erfüllt werden, wie in Kapitel 3 detaillierter erläutert [67]. Deshalb ist ein Time Series Forecasting zunächst vielversprechender, wobei eine Lösung mit Reinforcement Learning auch interessant wäre zu untersuchen.

RNN vs LSTM vs Transformer

Zum Time Series Forecasting gibt es viele mögliche Architekturen, wobei die gängisten RNNs, LSTMs und Transformer sind. Hierbei leiden RNNs mehr als LSTMs unter dem Vanishing bzw. Exploding Gradient Problem. LSTMs hingegen stellen eine allgemeine Architektur dar, welche gut für viele Anwendungsgebiete funktioniert [77]. Die Nutzung

von Transformern für Time Series Forecasting ist eine neuartige und vielversprechende Lösungsmöglichkeit. In meiner Projektarbeit konnte ich für die Sentiment Analysis auch Transformer, in dem Sprachmodell BERT, einsetzen, indem ich die von Google kostenlos zur Verfügung gestellten Rechenkapazitäten in Form von Google Colab nutzte. Jedoch ist der Datensatz, den ich hauptsächlich zu nutzen gedenke, relativ groß (380MB), was bei einer Upload Geschwindigkeit von 9,5Mbit/s ca. 5-6 Minuten dauern würde. Dies ist zwar nicht unmöglich schränkt jedoch die Geschwindigkeit des Testens der Parameter, der Feature und der Architektur deutlich ein. Des Weiteren stehen mehr Ressourcen zum Time Series Forecasting mit LSTMs zur Verfügung, weswegen die Nutzung von LSTMs gegenüber einer Transformer-Architektur passender ist, insbesondere, um den Fokus der Bachelorarbeit auf der Erforschung vom Einfluss von Parametern, dem Sentiment von Nachrichten sowie weiteren Metriken beim algorithmischen Handel zu bewahren.

Parameter

Sowohl für das Trainieren als auch das LSTM selbst können verschiedene Parameter genutzt werden, um diese zu adjustieren. Für das Training selbst

Datensatz

In meinen Recherchen zu passenden Datensätze konnte ich vier Datensätze ermitteln, welche potenziell hilfreiche Features enthalten, inklusive Nachrichten zu Wertpapieren. Da nicht ein einzelner Datensatz mit Nachrichten, Kurs, Fundamentaldaten usw. existiert ist die Idee eine Kombination aus Features verschiedener Datensätze zu ermitteln, welche dem Modell beim Time Series Forecasting helfen. Der Hauptdatensatz, den ich nutzen werde, ist auf Kaggle unter dem Namen „Daily Financial News for 6000+ Stocks“ zu finden. Er enthält tägliche Finanznachrichten zu mehr als 6000 Aktien von 2009 bis 2020. Der vollständige Datensatz ist in drei Teile unterteilt. Zum einen Analystenbewertungen von der Webseite benzinga.com, eine gesäuberte/verkleinerte Version dieses sowie Nachrichtenüberschriften zu verschiedenen Aktien. Da sich diese Bachelorarbeit auf das Sentiment von Nachrichten fokussiert und das genutzte Machine Learning Modell zur Sentiment Analysis auf Nachrichten, und nicht Analystenbewertungen, trainiert wurde, ist letzterer Teildatensatz zu wählen. Die Dateigröße des Datensatzes beträgt 382MB. Er umfasst mehr als 1,8 Millionen Einträge, also Nachrichtenüberschriften an einem

bestimmten Tag zu einer gewissen Aktie. Hierbei ist hervorzuheben, dass es sich um tägliche Nachrichten handelt und der Zeitpunkt der Veröffentlichung nicht minutengenau festgehalten ist. Dies wirkt sich auf den Umfang der Bachelorarbeit aus, da somit eine innertägliche Vorhersage der Aktienkurse ausgeschlossen ist. Es wird sich also auf die Vorhersage der täglichen Öffnungs- und/oder Schlusskurse konzentriert. Welche Vorhersage schlussendlich lohnenswert ist wird sich während der Umsetzung zeigen. Des Weiteren sei noch erwähnt, dass dieser Datensatz keine Kursdaten enthält, welche also dann extern gesammelt werden müssen.

Bei dem zweiten Datensatz, welchen man möglicherweise zusätzlich nutzen könnte, handelt es sich um den ebenfalls auf Kaggle zu findenden „Historical Financials Data for 3000+ stocks“. Dieser ist eine Sammlung aus Quartalszahlen aus dem Zeitraum 2006 bis 2020 von mehr als 3000 Aktien, welche auf der Seite der SEC (United States Securities and Exchange Commission) veröffentlicht wurden, dort aber oft in unterschiedlichen Formaten vorliegen, welche nun in diesem Datensatz in einem einheitlichen Format vorliegen. Der Datensatz enthält 45 Variablen darunter die jeweilige Aktie mit vielen Fundamentaldaten wie Gewinn pro Aktie, Kosten sowie die Menge an vorhandenem Kapital. Insgesamt besitzt der Datensatz mehr als 100000 Einträge. Um die Auswirkungen von Fundamentaldaten auf die Performance zu erforschen könnte dieser Datensatz genutzt werden, wobei darauf geachtet werden muss, dass die Schnittmenge an vorhandenen Aktien zu Datensatz 1 nicht vollständig übereinstimmt, da es sich hier um separate Datensätze handelt.

Datensatz 3 enthält ebenfalls die Quartalsberichte (Cash Flow Statement, Balance Sheet, Income Statement) von mehr als 4400 Aktien über den Zeitraum von 2016 bis 2020. Er ist ebenfalls auf Kaggle zu finden unter der Bezeichnung „Financial data of 4400+ public companies“. Dieser Datensatz ist also umfassender bezüglich der Anzahl der Aktien im Vergleich zu Datensatz 2, jedoch begrenzter im Zeitraum. Eine erweiterung des Datensatzes 2 durch Datensatz 3 ist also denkbar, eine alleinige Nutzung des Datensatz 3 ist jedoch auszuschließen, mangels großer zeitlicher Schnittmenge zu Datensatz 1. Die Daten stammen von Yahoo Finance.

Bei Datensatz 4 handelt es sich um „New York Stock Exchange – S&P 500 companies historical prices with fundamental data“, welcher ebenfalls auf Kaggle zu finden ist. Hierbei handelt es sich um die Preise sowie Fundamentaldaten der im S&P 500 enthaltenen

Unternehmen aus dem Zeitraum 2010 bis 2016. Hierbei ist die Zusammenführung von Preis- und Fundamentaldaten vorteilhaft, da weiter Datenquellen diesbezüglich nicht notwendig sind. Da jedoch Datensatz 1 zwingend notwendig ist, um das Sentiment von Nachrichtenüberschriften zu nutzen, schränkt Datensatz 4 hierbei Datensatz 1 bezüglich des Zeitraumes ein. Ob eine sinnvolle Nutzung dieses Datensatzes möglich ist, ist also fraglich.

Des Weiteren ist es nötig eine weitere Quelle für die Preise der im Datensatz 1 vorhandenen Aktien zu finden, da in drei von vier vorgestellten Datensätze keine Preise vorhanden sind. Hierzu existieren Python-APIs wie yfinance, welche es dem Nutzer ermöglichen Kursdaten von Aktien über einem angegebenen Zeitraum herunterzuladen. Mehr dazu im Kapitel Umsetzung. Welcher Datensatz zusätzlich zu Datensatz 1 zu nutzen ist soll ermittelt werden, indem zwar wichtige Daten wie Kurs- oder Fundamentaldaten inkludiert werden, jedoch die Anzahl an Einträgen im Datensatz maximiert werden soll, um die dem Machine Learning Modell zur Verfügung stehende Datenmenge zu maximieren.

Sentiment Analysis

Die im Datensatz enthaltenen Nachrichtenüberschriften müssen per Sentiment Analysis analysiert werden, um das Sentiment zu identifizieren. Hierzu plane ich mein Machine Learning Modell aus meiner Projektarbeit zu benutzen. Hierbei handelt es sich um das pre-trained Modell BERT, mit welchem auf alltäglichen Wirtschaftsnachrichten von verschiedenen Nachrichtenagenturen Fine-Tuning betrieben wurde und eine Genauigkeit von 82,40% zeigen konnte. Somit ist ein für den Finanzbereich passendes Modell gegeben. Zum Vergleich können sich menschliche Analysten bei der Sentiment Analysis 80-85% der Zeit auf eine Klassifizierung einigen [82]. Somit liegt das in dieser Bachelorarbeit zur Sentiment Analysis angewandte Machine Learning Modell im Rahmen der menschlichen Möglichkeiten und stellt somit eine angemessen gute Performance dar. Hierbei gilt jedoch zu überprüfen wie sinnvoll tatsächlich schlussendlich die Klassifizierungen von dem Modell sind. Da die Datenquelle des Trainingsdatensatzes des Modells nicht gleich der des Datensatz 1 ist, ist eine gute Performance nicht unbedingt garantiert, weshalb die Sinnhaftigkeit der Sentiment Analysis geprüft und mit in die Auswertung der Ergebnisse dieser Bachelorarbeit einfließen sollte.

4.2 Nicht-funktionale Eigenschaften

Auswahl der Programmiersprache

Für den Bereich Machine Learning sind mehrere Programmiersprachen geeignet. Die gängigsten sind Python, C++, Java, JavaScript, und R. Wobei 60% der Machine Learning Entwickler Python nutzen. Python gilt als einfach zu lernen, leicht skalierbar und hat viele mächtige Bibliotheken wie „pandas“, „sklearn“ oder „matplotlib“. Andere Sprachen wie C++ haben jedoch andere Vorteile wie bspw. eine höhere Effizienz [81]. Für diese Bachelorarbeit entscheide ich mich dennoch für die Sprache Python, da ich dort bisher am meisten Erfahrung sammeln konnte und Quellen sowie benötigte Bibliotheken leicht zugänglich sind.

Auswahl Entwicklungsumgebung

Bei der Programmierung im Bereich Machine Learning mit Python gibt es zwei grundsätzliche Möglichkeiten: Zum einen kann eine IDE wie PyCharm genutzt werden, um ein Skript zu schreiben. Dies kann auch mit Konzepten der Objektorientierung geschehen. Zum anderen ist es möglich ein sogenanntes Notebook, wie z.B. Jupyter Notebook, zu nutzen. Hierbei können Code und Textteile in Zellen unterteilt werden, welche separat ausgeführt werden können. Dies erleichtert es zu experimentieren und Codeblöcke zu testen. Besonders gut ist Jupyter Notebook auch für die Datenvisualisierung, da die Ausgabe sofort angezeigt wird. Auf der anderen Seite existiert die Gefahr, dass das Notebook schnell zu einem Monolith wird, welcher nur sehr schwer versioniert und gewartet werden kann. Für kleinere Forschungen wie diese Projektarbeit überwiegen jedoch die Vorteile, weshalb die Nutzung von Jupyter Notebook sinnvoll erscheint. Hierzu stellt Google die Plattform „Google Colab“ bereit auf der kostenlos Rechenleistung bezogen werden für die Ausführung von Notebooks. Dies bringt viele Vorteile, da die bereitgestellten High End Grafikkarten für das Trainieren des neuronalen Netzwerks genutzt werden können. Auch wenn ich die Rechenkapazität, wie bereits erwähnt, voraussichtlich nicht nutzen kann, so bin ich mit der Umgebung von Google Colab bereits vertraut, weshalb ich diese mit einer lokalen Grafikkarte, der GTX 1070 mit 8GB VRAM, nutzen werde.

Bibliotheken

Im Folgenden stelle ich die Bibliotheken vor, die in dieser Bachelorarbeit genutzt werden. „pathlib“ bietet Methoden für Dateipfade. So kann z.B. abgefragt werden, ob eine Datei existiert bevor sie gelesen wird. Eine besonders wichtige Bibliothek ist hingegen „pandas“. Diese stellt effiziente DataFrame Objekte zur Verfügung. Zudem ermöglicht die Bibliothek CSV, Textdateien und weitere Dateiformate zu schreiben und zu lesen. Es ist auch möglich DataFrame Objekte zu manipulieren bspw. mit Funktionen, die Duplikate entfernen. Desweiteren wurden „mathplotlib“ für Grafiken sowie „numpy“ für die Datenverarbeitung, konkret das Verändern der Form des Datenobjektes, genutzt. Auch die „logging“ Bibliothek wird genutzt, um Ergebnisse in log Dateien festzuhalten.

Für den Machine Learning Teil dieser Bachelorarbeit sind drei Bibliotheken notwendig. Bei der Wahl eine Machine Learning Frameworks gibt es zwei Möglichkeiten: Das von Facebook entwickelte Pytorch oder Tensorflow von Google. Die allgemeine Meinung ist, dass Tensorflow besser skaliert und besser geeignet für produktionsreife Modelle ist. PyTroch hingegen hat eine intuitivere Syntax, was Vorteile in dem Bau von Prototypen und in der Forschung bringt, weshalb mittlerweile mehr Forschungspapiere mit PyTorch veröffentlicht werden [90] [91]. Essenziell ist auch die Bibliothek „transformers“, welche Architekturen wie BERT für NLP zur Verfügung stellt. Hiermit können diese Pre-Trained Modelle geladen, angewandt und evaluiert werden, was zum Ermitteln der Sentiments notwendig ist. Auch „scikit-learn“ wird in dieser Bachelorarbeit genutzt. Hauptsächlich, um den gelabelten Datensatz in einen Trainingsteil und einen Testteil zu splitten.

Kapitel 5: Umsetzung

In diesem Kapitel werden Art und Weise der Umsetzung der Bachelorarbeit dargestellt sowie dabei aufgekommene Probleme und Erkenntnisse präsentiert.

5.1 Sentiment Analysis

Zuerst nutze ich das Machine Learning Modell aus meiner Projektarbeitarbeit, um dem Datensatz 1 das Sentiment für jede Nachrichtenüberschrift hinzuzufügen. Hierzu lade ich das Modell und den Datensatz und lasse das Modell über den Datensatz laufen, um für jede Überschrift das Sentiment zu ermitteln. Hierbei ist besonders anzumerken, dass wie von BERT erwartet, die Sätze vorher encoded werden müssen, um dem von BERT erwartendem Format gerecht zu werden. Die Methode get_pred nimmt also die übergebene Überschrift entgegen, encoded diese und gibt die Vorhersage des Modells zurück. Diese wird in einer Liste zwischengespeichert, um anschließend als neue Spalte dem Datensatz hinzugefügt werden kann.

```
[ ]  def get_pred(sentence):
       encoded_sentence = tokenizer.encode_plus(
             sentence,                      # input the news headline
             add_special_tokens = True,     # special tokens added to mark beginning & end of sentence
             truncation = True,             # make all sentences a fixed length
             padding = 'max_length',        # pad with zeros to max length
             return_attention_mask = True,  # include attention mask in encoding
             return_tensors = 'pt'          # return as pytorch tensor
         )
       input_ids = encoded_sentence['input_ids'].to(device)
       attention_mask = encoded_sentence['attention_mask'].to(device)
       out = model(input_ids, attention_mask)
       pred = torch.argmax(out[0])
       return pred.item()
```

```
[ ]  sentiments = []
     for index, row in data.iterrows():
       sentiments.append(get_pred(row['headline']))
     data['sentiment'] = sentiments
     print(data)
     data.to_csv("stock_sentiment_data.csv", index=False, header = True, encoding='utf-8-sig')
```

Im Folgenden ein Auszug aus den Ergebnissen:

	headline	sentiment
0	Agilent Technologies Announces Pricing of $5…… Million of Senior Notes	2
1	Agilent (A) Gears Up for Q2 Earnings: What's in the Cards?	2
2	J.P. Morgan Asset Management Announces Liquidation of Six Exchange-Traded Funds	1
3	Pershing Square Capital Management, L.P. Buys Agilent Technologies Inc, The Howard Hughes Corp, ...	0
4	Agilent Awards Trilogy Sciences with a Golden Ticket at LabCentral	0
5	Agilent Technologies Inc (A) CEO and President Michael R. Mcmullen Sold $–.4 million of Shares	2
6	' Stocks Growing Their Earnings Fast	0
7	Cypress Asset Management Inc Buys Verizon Communications Inc, United Parcel Service Inc, ...	0
8	Hendley & Co Inc Buys American Electric Power Co Inc, Agilent Technologies Inc, Paychex ...	0
9	Teacher Retirement System Of Texas Buys Hologic Inc, Vanguard Total Stock Market, Agilent ...	0

Hierbei repräsentiert bei dem Sentiment die Zahl 0 positiv, 1 negativ und 2 ein neutrales Sentiment. Es ist zu erkennen, dass das Sentiment korrekt klassifiziert wurde. Auch das neutrale Sentiment wird korrekt erkannt. Einzig Satz 5 könnte auch negative gewertet werden, da der Verkauf von Aktien durch den CEO als Zeichen zu hohen Preisen gesehen werden kann. Auf der anderen Seite ist dies jedoch kein eindeutiges Zeichen: Oft verkaufen CEOs Aktien regelmäßig oder um bspw. Steuern zu bezahlen, weshalb auch eine neutrale Bewertung dieses Satzes angemessen ist [83]. Im folgenden Bild ist jedoch auch zu sehen, dass das Modell nicht immer richtig liegt:

```
Name: 13170, dtype: object
headline     Philips Remains at Neutral - Analyst Blog
sentiment    0
```

Einen großen Teil scheint das Modell jedoch korrekt klassifizieren zu können, wie auch von den Testergebnissen des Modells auszugehen war, weshalb das Modell weiterhin als angemessen für die Aufgabe in dieser Bachelorarbeit zu sein scheint.

5.2 Vereinigung von Sentiment- und Preisdaten

Als nächsten Schritt ist es nötig dem Nachrichten-Sentiment-Daten Preise hinzuzufügen, um später diese mit dem LSTM vorhersagen zu können. Hierbei ist es zunächst notwendig den vorhandenen Nachrichten-Sentiment-Datensatz genauer zu betrachten. Hierbei fällt auf, dass je nach Aktie über unterschiedliche Zeiträume verschieden viele Nachrichten mit Sentiment zur Verfügung stehen. Dies gilt bei jeder Datenvorverarbeitung sowie beim Training selbst zu beachten. Für die Aktie mit dem Symbol „A“ stehen 936 Einträge über

den Zeitraum 14.05.2012 bis 01.06.2020 zur Verfügung während für die Aktie mit dem Symbol „AGN“ nur 9 Einträge über den Zeitraum 15.05.2020 bis 22.05.2020 zur Verfügung stehen. Hierbei ist weiterhin zu beachten, dass an einem Tag auch mehrere Nachrichten eingetragen sein können. Dies bedarf einer Konfliktlösung für das Training des Modells, da das LSTM eine feste Inputgröße besitzt und somit nicht mal eine Nachricht und mal mehrere übergeben werden können. Dazu gibt es verschiedene Lösungsmöglichkeiten. Man könnte einen Durchschnittswert der Sentiments pro Tag berechnen, die zusätzlichen Sentiments einfach weglassen oder die Inputgröße des Modells vergrößern, aber somit den zusätzlichen Speicheraufwand, der dadurch entsteht, in Kauf nehmen. Falls kein Sentiment zur Verfügung steht wäre der Input an dieser Stelle für das Modell leer. Zunächst entscheide ich mich für die naive Lösung und verwerfe zusätzliche Nachrichten mit Sentiment, schließe jedoch andere Möglichkeiten nicht aus, die sicherlich interessant sind zu probieren.

Um die beiden Datensätze zusammenzuführen lade ich zunächst den Nachrichten-Sentiment-Datensatz. Dabei wird das Format des Datums etwas umgewandelt und irrelevante Informationen entfernt wie die URL oder die Uhrzeit, welche immer 0 Uhr ist, da die Uhrzeit nicht erfasst wurde im Datensatz. Dies wird mit folgendem Code gemacht:

```
[3]  def load_dataset (data_filename):
       data_path = Path(data_filename)
       if data_path.exists():
         print(f"Loading {data_path}...")
         data_csv = pd.read_csv(data_path)
         return data_csv
       else:
         print(f"The dataset in {data_path} does not exist")
```

```
[10] #loading partner headlines dataset
     data = load_dataset("stock_sentiment_data.csv")
     data.rename(columns={'date':'datetime'},inplace = True)
     data[['date','time']] = data.datetime.str.split(expand=True)
     data.drop(['time'], axis=1, inplace=True)
     data.drop(['url'], axis=1, inplace=True)
     data.drop(['datetime'], axis=1, inplace=True)
     data.drop(data.columns[0], axis=1, inplace=True)
     data.head(10)
```

Anschließend sieht die Form des Datensatzes wie folgt aus:

	headline	publisher	stock	sentiment	date
0	Agilent Technologies Announces Pricing of $5..... Million of Senior Notes	GuruFocus	A	2	2020-06-01
1	Agilent (A) Gears Up for Q2 Earnings: What's in the Cards?	Zacks	A	2	2020-05-18
2	J.P. Morgan Asset Management Announces Liquidation of Six Exchange-Traded Funds	GuruFocus	A	1	2020-05-15
3	Pershing Square Capital Management, L.P. Buys Agilent Technologies Inc, The Howard Hughes Corp, ...	GuruFocus	A	0	2020-05-15
4	Agilent Awards Trilogy Sciences with a Golden Ticket at LabCentral	GuruFocus	A	0	2020-05-12
5	Agilent Technologies Inc (A) CEO and President Michael R. Mcmullen Sold $–.4 million of Shares	GuruFocus	A	2	2020-05-11
6	' Stocks Growing Their Earnings Fast	GuruFocus	A	0	2020-05-07
7	Cypress Asset Management Inc Buys Verizon Communications Inc, United Parcel Service Inc, ...	GuruFocus	A	0	2020-05-07
8	Hendley & Co Inc Buys American Electric Power Co Inc, Agilent Technologies Inc, Paychex ...	GuruFocus	A	0	2020-05-05
9	Teacher Retirement System Of Texas Buys Hologic Inc, Vanguard Total Stock Market, Agilent ...	GuruFocus	A	0	2020-05-05

Die Preise für die Aktien im schon bestehenden Datensatz können mit der yfinance API von Yahoo Finance heruntergeladen werden. Da die Zeiträume der Daten von den Aktien abhängen ist es also nötig zu wissen über welchen Zeitraum zu welcher Aktie Daten zur Verfügung stehen.

Dies wird mit folgendem Abschnitt gelöst, indem eine Liste mit den nötigen Informationen erstellt wird:

```
#get overview of how much data and over what time frame is avaliable
prev_row = data.iloc[-1]
# stock_stats_dict = [stock count, end date, start date]
stock_stats_dict = {}
for index, row in data.iterrows():
  if row["stock"] != prev_row["stock"]:
    stock_stats_dict[row["stock"]] = [1,row["date"],row["date"]]
  if row["stock"] == prev_row["stock"]:
    stock_stats_dict[row["stock"]][0] += 1
    stock_stats_dict[row["stock"]][2] = row["date"]
  prev_row = row
print(stock_stats_dict)
```

```
{'A': [936, '2020-06-01', '2012-05-14'], 'AA': [214, '2020-05-29', '2018-10-19'],
```

Diese Liste kann anschließend genutzt werden, um die benötigten Daten herunterzuladen und in einem zunächst separaten Datensatz abzuspeichern:

```
#get stock price data

stock_price_list = []
for key, values in stock_stats_dict.items():
  print(f"Looking for data for {key}")
  try:

    if key == 'MRH':
      pass
    else:
      stock_price = yf.download(key, start=values[2], end=values[1])
      stock_price.reset_index(inplace=True)
      stock_price['Stock'] = key
      stock_price_list.append(stock_price)
  except Exception:
    print(f"There has been an error while looking for data for {key} stock in time period from {values[2]} to {values[1]}")
    pass
stock_price_data = pd.concat(stock_price_list)
print(stock_price_data)
stock_price_data.to_csv("stock_price_data.csv", index=False, header = True, encoding='utf-8-sig')

price_data = load_dataset("stock_price_data.csv")
price_data.head(10)
```

Hierbei kommt es bei einigen Aktien zu Fehlermeldungen, da diese auf Yahoo Finance nicht mehr verfügbar sind. Oft ist der Grund dafür, dass das Unternehmen nicht mehr auf gängigen Börsen gelistet ist. Dies ist im weiteren Verlauf zu beachten, da somit Daten verworfen werden müssen.

Der Datensatz sieht anschließend wie folgt aus:

	Date	Open	High	Low	Close	Adj Close	Volume	Stock
0	2012-05-14	27.804007	28.211731	27.496424	27.825464	25.591705	5526574.0	A
1	2012-05-15	29.263233	29.735336	28.905579	28.977110	26.650898	9783064.0	A
2	2012-05-16	29.091558	29.327612	28.397711	28.440628	26.157486	4976321.0	A
3	2012-05-17	29.971388	29.978540	28.340487	28.347639	26.071957	6900248.0	A
4	2012-05-18	28.612303	28.776825	27.625179	27.703863	25.479870	7216057.0	A
5	2012-05-21	27.775393	28.698139	27.753935	28.605150	26.308800	3890215.0	A
6	2012-05-22	28.655222	29.470673	28.340487	28.962805	26.637741	6180139.0	A
7	2012-05-23	28.633762	29.341917	28.175966	29.163090	26.821951	4817788.0	A
8	2012-05-24	29.213161	29.434908	28.934193	29.341917	26.986423	3591182.0	A
9	2012-05-25	29.341917	29.656652	29.177397	29.334764	26.979847	2633552.0	A

Der Datensatz enthält das Datum, der Eröffnungskurs, Tageshoch, Tagestief, den Schlusskurs, den adjustierten Schlusskurs (dieser berücksichtigt gezahlte Dividende sowie Aktiensplits [84]) sowie das Handelsvolumen und die jeweilige Aktie. Im weiteren Verlauf ist der rohe Schlusskurs zu wählen, da auch Tageshoch, Tagestief und Eröffnungskurs nicht adjustiert sind und sonst diese Informationen verloren gehen würden und

im Datensatz keine Informationen zu Dividenden oder Aktiensplits vorhanden sind, weshalb der rohe Schlusskurs dem Modell ein realeres Bild der Situation verleiht.

Nachdem also Daten zu Sentiment sowie Kurs zur Verfügung stehen ist es notwendig beide Datensätze zu vereinen, um einen ersten Datensatz zu erhalten, welchen man zum Training nutzen kann. Hierzu ist es notwendig die Informationen zu nutzen über welchen Zeitraum zu welcher Aktie Nachrichten mit Sentiment zur Verfügung stehen, dann dies muss bei der Vereinigung beider Datensätze beachtet werden. Dabei ist der naïve Ansatz dies mit einer for-Schleife zu tun, um die jeweiligen Fälle abzufangen, und nach und nach den vereinigten Datensatz aufzubauen.

Ein solcher Ansatz könnte folgendermaßen aussehen:

```
#merging price and sentiment-news dataset

price_data['Headline'] = None
price_data['Publisher'] = None
price_data['Sentiment'] = None
news_added = 0
news_not_added = 0
for index, row in data.iterrows():
  price_index = price_data.loc[(price_data['Stock'] == row['stock']) & (price_data['Date'] == row['date'])].index
  if price_index.empty:
    print(f"Error: Date not avaliable in Price Data")
    news_not_added += 1
  elif row['date'] != price_data.loc[price_index, 'Date'].item():
    print(f"Error: Date Mismatch {row['date']} searched, {price_data[price_index]['Date']} found")
  else:
    price_data.loc[price_index, 'Headline'] = row['headline']
    price_data.loc[price_index, 'Publisher'] = row['publisher']
    price_data.loc[price_index, 'Sentiment'] = row['sentiment']
    news_added += 1
  print(f"{index} / {len(data)} \n{row['stock']}: {row['headline']}")


print(f"Done Merging Datasets! Out of {len(price_data)} Days {news_added} have News. {news_not_added} news not added.")
```

Dabei wird der Nachrichtendatensatz mit über 1,8 Millionen Einträgen durchlaufen, wobei falls Daten zur Verfügung stehen diese dem Preisdatensatz hinzugefügt werden. Diese Form der Berechnung erwies sich jedoch als äußerst ineffizient. Nach ca. 20 Stunden Berechnung waren erst 46714 Einträge hinzugefügt worden. Aus diesem Grund war eine Alternativlösung nötig in Form eines Left-Join wobei der Preisdatensatz die Rolle des linken und der Nachrichtendatensatz die des rechten Datensatzes einnimmt. Der Left-Join ist deswegen nötig, da nicht für jedes Datum, für welches Kursdaten eingetragen sind, Nachrichten mit Sentiment zur Verfügung stehen. An den Stellen, an denen Nachrichten hinzugefügt werden können soll dies also geschehen, ansonsten sollen zumindest die Preisdaten dem Modell beim Training zur Verfügung stehen. Die Implementierung dieses Ansatzes kann in nur drei Zeilen wie folgt ermöglicht werden:

```
# merging price and sentiment datasets using pandas
merged_data = pd.merge(price_data, data, how='left', left_on=['Stock', 'Date'], right_on=['stock', 'date'])
merged_data.drop(['date', 'stock'], axis=1, inplace=True)
merged_data.drop_duplicates(['Date', 'Stock'], keep='last', inplace=True)
```

Da Duplikate entfernt werden, werden 50,04% der Nachrichten, also 923459 Nachrichten, hinzugefügt, sodass 12,42% der Tage, für die ein Preis zur Verfügung steht, auch eine Nachrichtenüberschrift mit Sentiment zur Verfügung steht.

Anschließend sieht der Datensatz wie folgt aus:

	Date	Open	High	Low	Close	Adj Close	Volume	Stock	headline	publisher	sentiment
2	2012-05-14	27.804007	28.211731	27.496424	27.825464	25.591705	5526574.0	A	Groupon, Agilent: After-Hours Headlines	webmaster	2.0
3	2012-05-15	29.263233	29.735336	28.905579	28.977110	26.650898	9783064.0	A	Agilent Sees Improving Trends - Analyst Blog	Zacks	0.0
5	2012-05-16	29.091558	29.327612	28.397711	28.440628	26.157486	4976321.0	A	Agilent Still A Name Worth Owning (A, DHR, WAT...	Investopedia	2.0
11	2012-05-17	29.971388	29.978540	28.340487	28.347639	26.071957	6900248.0	A	Stocks to Watch: Wal-Mart, Sears (Update 1)	TheStreet.Com	2.0
12	2012-05-18	28.612303	28.776825	27.625179	27.703863	25.479870	7216057.0	A	NaN	NaN	NaN
13	2012-05-21	27.775393	28.698139	27.753935	28.605150	26.308800	3890215.0	A	NaN	NaN	NaN
14	2012-05-22	28.655222	29.470673	28.340487	28.962805	26.637741	6180139.0	A	Agilent Pays A Hefty Price For A Fixer-Upper D...	Investopedia	0.0
15	2012-05-23	28.633762	29.341917	28.175966	29.163090	26.821951	4817788.0	A	NaN	NaN	NaN
16	2012-05-24	29.213161	29.434908	28.934193	29.341917	26.986423	3591182.0	A	NaN	NaN	NaN
17	2012-05-25	29.341917	29.656652	29.177397	29.334764	26.979847	2633552.0	A	NaN	NaN	NaN
18	2012-05-29	29.663805	30.221745	29.635193	30.171675	27.749565	5696570.0	A	NaN	NaN	NaN
19	2012-05-30	29.785408	29.785408	29.184549	29.577969	27.203522	4297871.0	A	Agilent Launches New Oscilloscope - Analyst Blog	Zacks	0.0
20	2012-05-31	29.656652	29.771103	28.805435	29.084406	26.749584	5887258.0	A	NaN	NaN	NaN
21	2012-06-01	28.040056	28.211731	27.224607	27.339056	25.144344	7578698.0	A	NaN	NaN	NaN
22	2012-06-04	27.432047	27.632332	26.437769	27.067240	24.894350	7212422.0	A	Agilent Technologies Inc. Reports Operating Re...	GuruFocus	2.0

5.3 Datenvorverarbeitung für das Training

In diesem Abschnitt wird gezeigt wie der Datensatz aus Abschnitt 5.2 vorverarbeitet wird, um anschließend das Modell zu trainieren.

Zunächst ist es wichtig, nachdem der Datensatz geladen ist, die nicht verfügbaren Sentiment-Werte mit dem Wert -1 zu füllen, damit sichergestellt ist, dass mit den Werten gerechnet werden kann.

```
data = load_dataset("stock_price_sentiment_data.csv")
data['sentiment'].fillna(-1, inplace=True)
data.head(10)
```

```
Loading stock_price_sentiment_data.csv...
```

	Date	Open	High	Low	Close	Adj Close	Volume	Stock	headline	publisher	sentiment
0	2012-05-14	27.804007	28.211731	27.496424	27.825464	25.591705	5526574.0	A	Groupon, Agilent: After-Hours Headlines	webmaster	2.0
1	2012-05-15	29.263233	29.735336	28.905579	28.977110	26.650898	9783064.0	A	Agilent Sees Improving Trends - Analyst Blog	Zacks	0.0
2	2012-05-16	29.091558	29.327612	28.397711	28.440628	26.157486	4976321.0	A	Agilent Still A Name Worth Owning (A, DHR, WAT...	Investopedia	2.0
3	2012-05-17	29.971388	29.978540	28.340487	28.347639	26.071957	6900248.0	A	Stocks to Watch: Wal-Mart, Sears (Update 1)	TheStreet.Com	2.0
4	2012-05-18	28.612303	28.776825	27.625179	27.703863	25.479870	7216057.0	A	NaN	NaN	-1.0
5	2012-05-21	27.775393	28.698139	27.753935	28.605150	26.308800	3890215.0	A	NaN	NaN	-1.0
6	2012-05-22	28.655222	29.470673	28.340487	28.962805	26.637741	6180139.0	A	Agilent Pays A Hefty Price For A Fixer-Upper D...	Investopedia	0.0
7	2012-05-23	28.633762	29.341917	28.175966	29.163090	26.821951	4817788.0	A	NaN	NaN	-1.0
8	2012-05-24	29.213161	29.434908	28.934193	29.341917	26.986423	3591182.0	A	NaN	NaN	-1.0
9	2012-05-25	29.341917	29.656652	29.177397	29.334764	26.979847	2633552.0	A	NaN	NaN	-1.0

Im nächsten Schritt ist zu bemerken, dass es sich bei der Aufgabe des Modells um Zeitreihenvorhersage von unterschiedlichen Aktien handelt, weshalb zu beachten ist, dass die Zeitreihe einer Aktie die der anderen nicht tangieren. Das Training muss deshalb Aktie für Aktie vollzogen werden. Dies gilt auch bei der Vorverarbeitung, bei welcher jede Aktie einzeln behandelt werden muss. So kann der Train-Test-Split, also das Aufteilen eines Datensatzes in Training- und Testteil, bei der einen Aktie anders ausfallen als bei der anderen, da die Zeiträume der Aktien unterschiedlich sind. Es ist also zunächst für weitere Schritte eine Liste der verfügbaren Aktien zu haben, was folgendermaßen implementiert wurde:

```
# get the stock list
stock_list = data['Stock'].unique()
print(f"The dataset consists of {len(stock_list)} stocks.\n{stock_list}")
```

```
The dataset consists of 4752 stocks.
['A' 'AA' 'AADR' ... 'ZTR' 'ZTS' 'ZUMZ']
```

Anschließend wird ein Dictionary erstellt, bei der jede Aktie einen Trainings- sowie einen Testdatensatz besitzt. Dies geschieht dann bei der Aufteilung in Trainings- und Testdatensatz wie folgt:

```
# split data
train_test_data = {}
for index, stock in enumerate(stock_list):
    train_test_data[stock] = {}
    try:
      train_test_data[stock]["train"], train_test_data[stock]["test"] =
                              train_test_split(data.loc[(data['Stock'] == stock),
                              ['Date', 'Close', 'sentiment', 'Stock', 'Volume']],
                              test_size = 0.3, shuffle=False)
    except ValueError:
      print(data.loc[(data['Stock'] == stock),
                     ['Date', 'Close', 'sentiment', 'Stock', 'Volume']])
      del train_test_data[stock]
    print(f"{stock}: {index} / {len(stock_list)}")
```

Dabei ist es zum einen wichtig die Reihenfolge der Daten beizubehalten, da die Tageskurse voneinander abhängig sind und zum anderen ist die Festlegung der Größe des Testdatensatzes von Bedeutung. Denn diese beeinflusst wie aussagekräftig das Testen tatsächlich ist und wie genau das Modell ist, da bei einer großen Testdatensatzgröße dem Modell

Trainingsdaten genommen werden. Es existiert hierzu jedoch kein optimales Verhältnis zwischen Trainings- und Testgröße, weshalb in dieser Bachelorarbeit ein übliches Verhältnis von 67% Trainingsdaten und 33% Testdaten benutzt wird [85].

Im nächsten Schritt ist es entscheidend die beiden Datensätze, also Trainings- und Testdatensatz weiter aufzuteilen in einen X (Input) und Y (Label) Teil. Hierbei ist besonders wichtig zu entscheiden, was als Label genommen wird, da diese durch die Loss-Funktion dem Modell vermitteln wie gut oder schlecht die Vorhersage des Modells war. Bei der angezielten Vorhersage von Aktienkursen scheint zunächst der Preis/Kurs der Aktie ein sinnvolles Label zu sein. Und es lassen sich sogar tatsächlich Beispiele finden, bei denen eine nahezu perfekte Vorhersage des Aktienkurses erreicht werden konnte [86] [87]. Falls es tatsächlich durch eine einfache Lösung wie diese möglich wäre Aktienkurse vorherzusagen, so wären extrem hohe Renditen trivial zu erreichen. Der tatsächliche Grund hierfür scheint zu sein, dass der Preis des vorherigen Tages ein guter Prädikator für den Preis des nächsten Tages ist. Dies wird an folgendem Beispiel deutlich: Angenommen der Kurs einer Aktie fällt um 0,5% auf $99 und steigt daraufhin um 1,02% auf ca. $100. Nimmt man hierbei jeweils den vorherigen Wert, um den darauffolgenden vorherzusagen so ist die Vorhersage von +1,02% von einem Vortageswert von -0,5% nicht trivial. Eine Vorhersage von $99 bei einem echten Wert von $100 bietet jedoch schon eine Genauigkeit von 99%, jedoch bietet dies keinen Mehrwert. Zudem behindert dies das Modell bei der Suche nach einer tatsächlichen Vorhersage, da durch das einfach kopieren des vorherigen Wertes bereits eine so hohe Genauigkeit erzielt wird.

Des Weiteren kann der Maßstab der Kurse je Aktie unterschiedlich sein. Während (Stand: April 2021) die Amazon Aktie bei ca. $3000 liegt, steht die Microsoft Aktie bei ca. $240, wobei beide Firmen eine ähnliche Marktkapitalisierung besitzen. Während beide Werte zwar in der Maßeinheit US-Dollar angegeben sind, so liegen hier zwei unterschiedliche Maßstäbe vor, auf Grund der unterschiedlich großen Werte bei ähnlicher Marktkapitalisierung. Da Neuronale Netze lernen den Input zu einem Output zu führen, können diese unterschiedlich großen Werte dem Modell es erschweren ein echtes Modell zum Vorhersagen zu bilden [88]. Eine mögliche Lösung wäre eine Normalisierung der Kurse, um diese auf einen Maßstab (zum Beispiel auf eine Skala von 0 bis 1 oder vom Minimum zum Maximum), jedoch ist dies nur eine temporäre Lösung. Die Kure sich mit der Zeit weiter entwickeln und tendenziell Steigen, würde der festgelegte Maßstab nicht mehr

funktionieren.

Aus diesen Gründen ist es sinnvoll und nötig nicht die Kurse direkt sondern die Rendite als Vorhersageziel zu wählen, da die Rendite auf einen festen Maßstab (in %) für all Aktien festgelegt ist, sowie die Vorhersage nicht trivial ist, was zwar zu einem schwierigeren Problem aber auch zu einer potenziell echten Vorhersage führt.

Aus diesem Grund wird im Folgenden die Rendite pro Tag berechnet und dem Datensatz hinzugefügt:

```
# calculate return, drop stocks with too little data and drop rows with nan values
for index, stock in enumerate(stock_list):
  try:
    print(f"{stock} : {index} / {len(stock_list)}")
    if len(train_test_data[stock]["train"]) < 10:
      del train_test_data[stock]
    else:
      train_test_data[stock]["train"]['PrevClose'] = train_test_data[stock]["train"]['Close'].shift(1)
      train_test_data[stock]["test"]['PrevClose'] = train_test_data[stock]["test"]['Close'].shift(1)
      train_test_data[stock]["train"]['Return'] = (train_test_data[stock]["train"]['Close']
                                                   - train_test_data[stock]["train"]['PrevClose'])
                                                  / train_test_data[stock]["train"]['PrevClose'] *100
      train_test_data[stock]["test"]['Return'] = (train_test_data[stock]["test"]['Close']
                                                   - train_test_data[stock]["test"]['PrevClose'])
                                                  / train_test_data[stock]["test"]['PrevClose'] * 100
      train_test_data[stock]["train"].dropna(how='any', inplace=True)
      train_test_data[stock]["test"].dropna(how='any', inplace=True)
  except KeyError:
    print(f"Some Error at stock {stock}")
```

Da zu manchen Aktien nicht genügend Daten existieren muss eine aktualisierte Liste an Aktien erstellt werden:

```
# get the updated stock list
stock_list = list(train_test_data.keys())
print(f"The dataset consists of {len(stock_list)} stocks.\n{stock_list}")
```

```
The dataset consists of 4638 stocks.
['A', 'AA', 'AADR', 'AAL', 'AAMC', 'AAME', 'AAOI', 'AAON', 'AAP', 'AAU', 'AAWW',
```

Im Folgenden werden beispielhaft ein Auszug der Daten des Unternehmens Agilent Technologies mit dem Tickersymbol „A“ aufgezeigt, sowie alle täglichen Renditen grafisch dargestellt:

	Date	Close	sentiment	Stock	Volume	PrevClose	Return
427674	2017-03-10	2.55	-1.0	AT	577100.0	2.50	1.999998
427675	2017-03-13	2.55	-1.0	AT	457700.0	2.55	0.000000
427676	2017-03-14	2.55	-1.0	AT	442900.0	2.55	0.000000
427677	2017-03-15	2.60	-1.0	AT	494900.0	2.55	1.960782
427678	2017-03-16	2.65	0.0	AT	458500.0	2.60	1.923084
427679	2017-03-17	2.70	-1.0	AT	1024100.0	2.65	1.886791
427680	2017-03-20	2.70	-1.0	AT	285400.0	2.70	0.000000
427681	2017-03-21	2.60	-1.0	AT	421100.0	2.70	-3.703709
427682	2017-03-22	2.60	-1.0	AT	314800.0	2.60	0.000000
427683	2017-03-23	2.60	-1.0	AT	223800.0	2.60	0.000000

```
[<matplotlib.lines.Line2D at 0x144d6711910>]
```

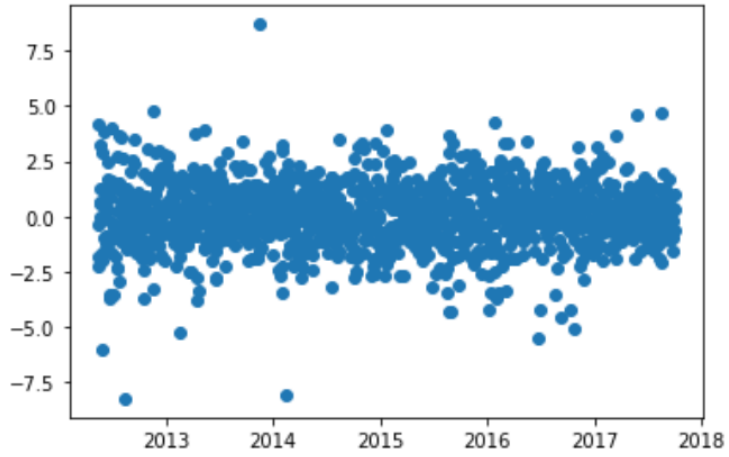

Abbildung 8: Verteilung der Renditen von AT über Zeit

Hierbei ist auffällig, dass die täglichen Renditen über die Jahre sehr ähnlich sind und meist nahe Null liegen. Dies bestätigt zum einen den einheitlichen Maßstab über Zeit, zeigt jedoch das potenzielle Problem aus, dass die Vorhersage äußerst schwierig ist, da die angezielten Werte sehr ähnlich sind. Zum Vergleich im Folgenden der Aktienkurs des selben Unternehmens zum selben Zeitraum, welcher erkennbar über Zeit steigt und sich (Stand: April 2021) seit Ende dieses Zeitraumes fast verdreifacht hat.

Abbildung 9: Aktienkurs von AT

Dies zeigt weiter auf, dass die Rendite über Zeit einen ziemlich gleichen Maßstab behält während der Maßstab des Preises sich über Zeit ändert. Das folgende Histogram zeigt zudem die Verteilung der Rendite, welche ca. der Normalverteilung folgt:

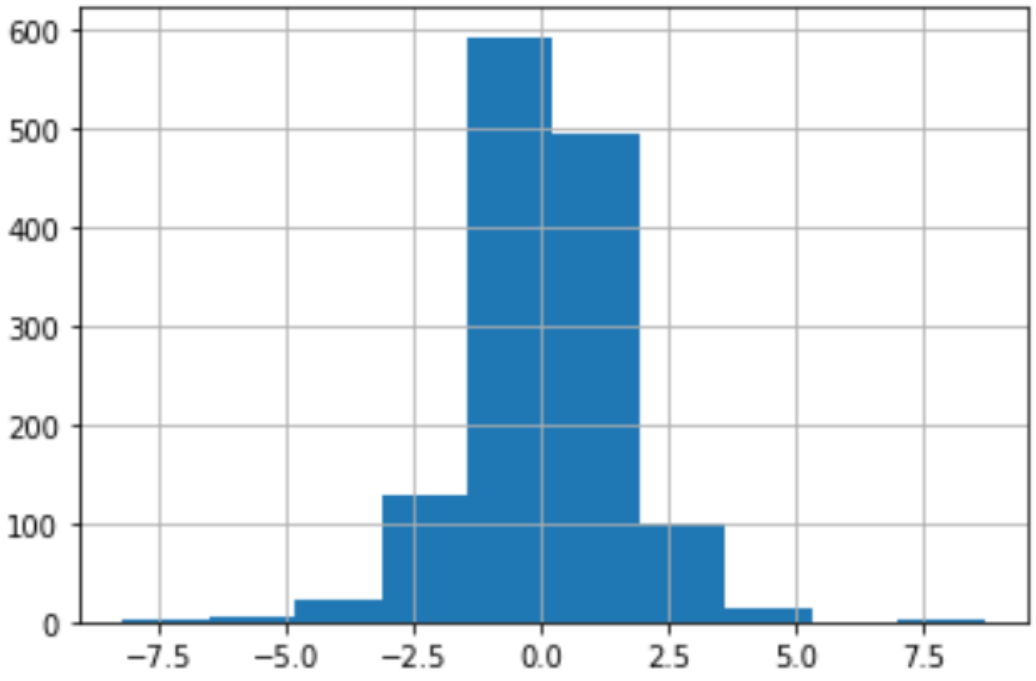

Abbildung 10: Verteilung der Renditen

Dadurch ist zum einen das Problem des sich verändernden Maßstabes gelöst, es könnte jedoch die Vorhersage ein schwieriges Problem sein bzw. bleiben, zum einen durch die Natur der Problematik, und zum anderen durch die Ähnlichkeit der Zielwerte, was die Vorhersage erschwert insbesondere, weil eine genaue Vorhersage erforderliche ist. Eine ungenaue Vorhersage ist hier auch trivial, indem immer der Wert Null vorhergesagt wird. Ob und inwieweit dies ein Problem ist wird sich in dieser Bachelorarbeit zeigen.

Im finalen Datenvorverarbeitungsschritt werden die Trainingsdaten und die Labels erzeugt. Hierbei werden zunächst 10 Tage genutzt, um den 11ten Tag vorherzusagen. Eine Adjustierung dieses Wertes zu Forschungszwecken ist denkbar und sinnvoll im Rahmen dieser Bachelorarbeit. Bei der Vorverarbeitung werden Renditen von 0 ausgelassen. Dies hat den Hintergrund, dass nicht mehr gelistete Aktien oder sehr kleine Aktien mit geringem Handelsvolumen, in dem Datensatz auftauchen, welche zu entfernen sind, um dem Modell ein Bild von gängigen Aktien zu verschaffen.

```
# reformat dataset for using series of length T to predict the return
seq_len = 10
train_data = {}
test_data = {}
for index, stock in enumerate(stock_list):
  train_data[stock] = {}
  test_data[stock] = {}
  X_train = []
  Y_train = []
  X_test = []
  Y_test = []
  for time in range(len(train_test_data[stock]["train"]) - seq_len):
    if train_test_data[stock]["train"]['Return'].iloc[time + seq_len] != 0:
      x_train = train_test_data[stock]["train"].iloc[time:time + seq_len].drop(['Date', 'Stock', 'PrevClose', 'Close'], axis=1)
      X_train.append(x_train)
      y_train = train_test_data[stock]["train"]['Return'].iloc[time + seq_len]
      Y_train.append(y_train)
  for time in range(len(train_test_data[stock]["test"]) - seq_len):
    if train_test_data[stock]["test"]['Return'].iloc[time + seq_len] != 0:
      x_test = train_test_data[stock]["test"].iloc[time:time + seq_len].drop(['Date', 'Stock', 'PrevClose', 'Close'], axis=1)
      X_test.append(x_test)
      y_test = train_test_data[stock]["test"]['Return'].iloc[time + seq_len]
      Y_test.append(y_test)

  X_train = np.array(list(map(lambda x: x.to_numpy(), X_train)))
  X_train = X_train.reshape((-1, seq_len, 3))
  Y_train = np.array(Y_train)

  X_test = np.array(list(map(lambda x: x.to_numpy(), X_test)))
  X_test = X_train.reshape((-1, seq_len, 3))
  Y_test = np.array(Y_test)

  train_data[stock]['input'] = X_train
  train_data[stock]['label'] = Y_train

  test_data[stock]['input'] = X_test
  test_data[stock]['label'] = Y_test

  print(f"{stock}: {index + 1} / {len(stock_list)}")
```

5.4 Training, Testing und Hyperparametertuning

In diesem Abschnitt werde ich die Definition des Modells, seine Hyperparameter, den Trainings- sowie Testvorgang vorstellen. Zusätzlich stelle ich die Umsetzung des Hyperparametertuning vor. Die Ergebnisse und Erkenntnisse davon werden in folgenden Kapiteln behandelt. Die folgenden Ausschnitte aus dem Code sind in einer Methode train enthalten, welcher verschiedene Parameter für das Modell und dem Training übergeben werden können. Dazu später mehr.

Die Definition des Modells geschieht über folgende Klasse:

```
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim, dtype=torch.float32).requires_grad_().cuda()
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim, dtype=torch.float32).requires_grad_().cuda()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out
```

Hierbei wird zwar intern auf die LSTM Implementierung von PyTorch zugegriffen, diese beschreibt jedoch nur das LSTM selbst. Durch die Klasse wird ein Fully Connected Layer hinter dem LSTM gelegt, welcher den Output des LSTM auf einen finalen Output, den vorhergesagten Preis, überführt. Die Variablen h0 und c0 beschreiben jeweils den Hidden State und den Cell State. Diese werden bei jedem Forward Pass auf 0 gesetzt. Die Zustände sind dazu da, um Informationen über Zeitschritte weiterzugeben. Jedoch sollte sichergestellt werden, dass vor einem Durchgang dieser Zeitschritte die Zustände zurückgesetzt werden, um nicht von vorherigen Forward Passes beeinflusst zu werden. Dies ist insbesondere wichtig beim Training, wenn das Training auf Daten der einen Aktie abgeschlossen ist und auf die nächste Aktie gewechselt wird, sodass eine Beeinflussung des Modells durch vorherige Aktien zu verhindern ist. Des Weiteren werden die Anzahl der Features des Hidden State sowie die Anzahl der Layer gesetzt.

Da die Struktur des Modells durch die Klasse somit definiert ist, wird durch diese ein Objekt mit den übergebenen Parametern der Methode erzeugt. Dieses wird auf das festgelegte Gerät zur Berechnung, in diesem Falle die GPU, übergeben. In PyTorch ist es notwendig sowohl die Daten als auch das Modell auf dem selben Gerät zu haben, um mit diesen rechnen zu können. Außerdem wird die Loss Funktion (Mean Square Average,

kurz: MSE) und der Optimizer (Adam) definiert. Beide sind gängiger Standard im Anwendungsfall dieser Bachelorarbeit. Das beschriebene geschieht wie folgt:

```
model = LSTM(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=output_dim, num_layers=num_layers)
model = model.to(device)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Sobald alle Definitionen geschehen sind beginnt das Training und Testing. Hierbei werden alle Aktien nacheinander durchgegangen, woraufhin die definierte Anzahl an Epochen trainiert und getestet wird. Dabei werden die Daten in Batches, also in Teildatensätzen, dem Modell übergeben. Der Loss wird mit der definierten Loss Funktion berechnet und Backpropagation sowie das Aktualisieren der Parameter durch den Optimizer geschieht am Ende jedes Batch-Durchgangs. Beim Training werden zudem wichtige Daten für die Analyse in eine Datei geloggt wie der Loss der Aktie zu jeder Epoche und Batch-Schrittes. Ein Durchschnittswert des Losses beim Training und Testing wird ebenfalls berechnet und nach Abschluss des Trainings in die Log-Datei geschrieben. Für die Berechnung des Durchschnitts werden zunächst noch einige notwendigen Variablen wie folgt definiert:

```
avg_train_loss = 0
avg_test_loss = 0
num_batches_train = 0
num_batches_test = 0
```

Anschließend beginnt das Training selbst:

```
for stock in stock_list:
  print(stock)
  for epoch in range(num_epochs):
    #Training
    input = torch.from_numpy(train_data[stock]['input']).type(torch.cuda.FloatTensor)
    labels = torch.from_numpy(train_data[stock]['label']).type(torch.cuda.FloatTensor)
    i_batch = 0
    for i in range(0, len(input), batch_size):
      if len(labels) > 0:
        i_batch += 1
        model.train()
        model.zero_grad()
        out = model(input[i : i + batch_size])
        train_loss = criterion(out, labels[i : i + batch_size].unsqueeze(-1))
        avg_train_loss += train_loss.item()

        #backpropagation
        train_loss.backward()
        #update parameters
        optimizer.step()
```

Und nach dem Training folgt, wie erwähnt, das Testing:

```
#Testing
input = torch.from_numpy(test_data[stock]['input']).type(torch.cuda.FloatTensor)
labels = torch.from_numpy(test_data[stock]['label']).type(torch.cuda.FloatTensor)
i_batch = 0
for i in range(0, len(input), batch_size):
  if len(labels) > 0:
    i_batch += 1
    model.eval()
    with torch.no_grad():
      out = model(input[i : i + batch_size])
      test_loss = criterion(out, labels[i : i + batch_size].unsqueeze(-1))
      avg_test_loss += test_loss.item()
```

Die Trainingsfunktion kann dann beispielsweise folgendermaßen aufgerufen werden:

```
train(hidden_dim=32, num_layers=2, num_epochs=2, learning_rate=0.01, batch_size=64, log=True)
```

Tatsächlich wurden verschiedene Parameter definiert, welche in For-Schleifen automatisiert in Experimenten ausprobiert und deren Ergebnisse in log-Dateien festgehalten werden. Im Folgenden ist dieser Abschnitt zu sehen. Die daraus hervorgehenden Ergebnisse werden im nächsten Kapitel behandelt.

```
hidden_dims = [2, 4, 6, 8, 16, 32, 64, 128, 256, 512, 1024]
num_layers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20]
num_epochs = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
learning_rates = [0.1, 0.2, 0.4, 0.001, 0.002, 0.004, 0.0001, 0.00001]
batch_sizes = [8, 16, 32, 64, 128, 256, 512]


for dim in hidden_dims:
  for layer in num_layers:
    for epoch in num_epochs:
      for lr in learning_rates:
        for batch in batch_sizes:
          train(hidden_dim=dim, num_layers=layer, num_epochs=epoch, learning_rate=lr, batch_size=batch, log=True)
```

5.5 Herausforderungen bei der Umsetzung

Bei der Umsetzung sind einige Probleme aufgetreten, auf die ich in diesem Abschnitt eingehen werde. Es soll dazu dienen auf die Art der Fehler bei der Entwicklung von Machine Learning Algorithmen aufmerksam zu machen. Das erste Problem war eine Fehlermeldung, welche besagte, dass die Inputdaten und die Hidden Tensor nicht auf dem selben Gerät sind. Konkret waren die Hidden Tensor noch auf der CPU während sich der Input auf der GPU befand. Dies war trotz der Anweisung das Modell auf die GPU zu

überführen. Die Lösung hierzu war die Hidden States und die Cell States explizit auf die GPU zu laden. Die Anweisung model.to(device) scheint also nicht automatisch die Hidden States und Cell States zu beinhalten. Das nächste Problem war, dass der Input kein Float war. Da PyTorch Floats erwartet ist es notwendig, explizit den Typ als Float Tensor anzugeben. Nach Lösung dieser Probleme lief das Training, jedoch war der Loss nan. Die erste Vermutung war, dass durch das Exploding/Vanishing Gradient Problem der Loss nicht berechenbar wird. Tatsächlich lag das Problem jedoch am Datensatz, welcher an einigen Stellen noch nan Werte enthielt, welche, sobald mit diesen gerechnet wird, jedes daraus entstehende Ergebnis auch zu nan werden lässt. Es ist hierbei also wichtig, beim Arbeiten mit Pandas Dataframes, nan Werte zu vermeiden. Nachdem diese Probleme gelöst waren, funktionierte das Training problemlos. Jedoch war das Ergebnis des Durchschnitts-Loss ein extrem großer Wert (über 1 Millionen), obwohl die Berechnung selbst korrekt erschien und meist der Loss zwischen 2 und 20 liegt. Bei der genaueren Betrachtung fällt jedoch auf, dass das Modell bei manchen Aktien extrem hohe Rendite (30% pro Tag) vorhersagt, was natürlich falsch ist, weshalb der Loss auch extrem groß wird, da er zum einen sehr weit von der Realität entfernt ist und dieser Unterschied quadriert, und somit vergrößert wird. Ein weiteres Problem war die Fehlermeldung, dass die Dimension des Inputs sich Unterscheidet zu der Dimension des Targets. Dies ist aus der Perspektive der Loss Funktion. Der Grund hierfür ist das der Input der Loss Funktion, also der Output des Modells, für jeden Zeitschritt eine eigene Zeile für den Input besitzt, während die Targets, also die Labels, einfach hintereinander in einer Liste liegen. Um dieses Problem zu lösen muss also entweder die Form der Labels oder des Outputs des Modells jeweils an die des anderen angeglichen werden. Um den Output des Modells möglichst unberührt zu lassen wird in dieser Bachelorarbeit die Form der Labels angepasst, sodass alle Labels sich in einer Liste befinden und jedes Label sich in einer eigenen Liste/Zeile befindet.

Kapitel 6: Resultate

In diesem Kapitel werde ich die Ergebnisse der Versuche darstellen. Dies soll zeigen wie und ob sich Parameter auf die Performance von LSTMs bei der Vorhersage von Zeitreihendaten auswirken. Hierbei sollten jedoch verschiedene Dinge erwähnt sein. Zum einen wurde aus Zeitgründen eine Optimierung bezüglich der Inputdaten, bspw. hinzufügen oder Entfernen von Features, verzichtet. Es handelt sich also ausschließlich um eine Erforschung von Auswirkungen der Parameter. Andere Forschungen konnten hierbei bspw. zeigen, dass die Vergrößerung des Modells oder der Epochenanzahl an einem gewissen Punkt die Performance verschlechtert. Dies wird darauf zurückgeführt, dass wenn ein Modell, welches gerade so an die Daten angepasst ist, weiter gezwungen wird sich weiter an leicht unsaubere (also Noise enthaltene) Daten anzupassen die globale Struktur des Modells zerstört wird. Eine endgültige Erklärung für das Phänomen bleibt jedoch eine offene Frage in der Forschung [92]. Im NLP Bereich wird insbesondere der Effekt der Batch Size auf die Performance von transformerbasierten Sprachmodellen diskutiert. Hierbei ist zum einen wichtig, dass bei größeren Batchgrößen schneller über die Daten iteriert und somit schneller trainiert werden kann, was Zeit und Kosten sparen kann. Zum anderen konnte jedoch eine Verbesserung der Performance beobachtet werden, da bei größeren Batchgrößen der Optimierungsprozess stabilisiert wird während eine geringe Batchgröße zu Divergenz führen kann [94]. Der Grund hierfür ist, dass normalerweise bei der Optimierung durch Algorithmen wie Adam die Lernrate über Zeit verringert wird, um sicherer zu einer Konvergenz zu gelangen. Es konnte jedoch gezeigt werden, dass, wenn man die Batchgröße erhöht, eine Verringerung der Lernrate nicht unbedingt von Nöten ist [93]. Tatsächlich ist eine erhöhte Lernrate bei größeren Batchgrößen nötig, um die verringerte Anzahl an Iterationen pro Epoche auszugleichen, wobei eine erhöhte Lernrate zu einem instabileren Lernprozess führt. Ohne näher auf die Funktionsweise des Optimierungsalgorithmus LAMB einzugehen konnte mit diesem eine Stabilisierung des Lernprozess unter diesen Umständen geschaffen werden und so gezeigt werden, dass dieser beim Training von BERT besser mit großen Batchgrößen skaliert und somit zu einem schnelleren Training mit einer besseren Performance führt [94]. Ein weiteres Forschungsteam widerspricht diesen Resultaten jedoch und konnte auch mit dem Standardoptimierer Adam mit etwas Parametertuning vergleichbare Ergebnisse wie LAMB bei großen Batchgrößen erhalten. Sie führen die Beobachtungen des LAMB-

Forschungspapier auf Änderungen in der Trainingspipeline (z.B. Regularisierungstricks) anstatt an dem Optimierungsalgorithmus selbst und schließen daraus, dass etablierte Optimierer wie Adam vollkommen ausreichend sind [95]. Für diese Bacheloarbeit schließe ich daraus, dass das Tuning der Parameter sowie Auswahl eines Optimierungsalgorithmus eine offene Frage in der Forschung bleibt, jedoch (zumindest vorerst) Adam weiterhin der Standard-Optimierer bleibt und somit auch für diese Bachelorarbeit auszuwählen ist. Bezüglich der Auswirkungen der Parameter werde ich in diesem Kapitel meine Beobachtungen vorstellen und im folgenden Kapitel meine Schlussfolgerungen aufzeigen sowie ein Fazit ziehen.

Zunächst zu den durchgeführten Experimenten selbst: Auf der im Kapitel Planung dargelegten Hardware wurden die Parameter, welche am Ende des Kapitel Umsetzung gezeigt wurden, über einen Zeitraum von 6 Tagen versucht und die Ergebnisse festgehalten. Auf Grund des Umfangs der angestrebten Anzahl an Experimente konnten diese nicht vollständig durchgeführt werden. Ein großer Teil davon jedoch schon, weshalb dennoch genug Daten zum Analysieren vorhanden sind. Konkret konnte die Performance des Modells mit 98 verschiedenen Parameterkombinationen untersucht werden. Die vollständige Tabelle dieser Ergebnisse ist im Apendix zu finden. Im Folgenden werden nun einige Erkenntnisse dargelegt, welche aus diesen Experimenten erlangt werden können. Dabei sollte zunächst bemerkt werden, dass auf Grund der begrenzten Trainingszeit lediglich unterschiedliche Epochenanzahlen (2 und 4), Lernraten (0.0001, 0.001, 0.01, 0.1, 0.002, 0.02, 0.2, 0.004, 0.04, 0.4, 0.00001) sowie Batchgrößen (8, 16, 32, 64, 128, 256, 512) getestet werden konnten. Bei der Auswahl der Hyperparametern wurde sich an üblichen Werten orientiert, jedoch auch Werte, die etwas über oder unter üblichen Werten liegen, in die Auswahl mit aufgenommen. Andere Hyperparameter können somit in der Analyse nicht in Betracht gezogen werden. Nun zur Analyse selbst: Während des Trainings selbst fällt zum einen auf, dass in fast jeder Batch die Vorhersage des Modells gleich ist. Hier ein beispielhafter Auszug:

```
##################################################
Training === Epoch: 2/2 Loss: 1.6514947414398193 Stock: FITB Batch: 1
##################################################

Output:tensor([0.0740, 0.0740, 0.0740, 0.0740, 0.0740, 0.0740, 0.0740, 0.0740, 0.0740,
        0.0740, 0.0740, 0.0740, 0.0740, 0.0740, 0.0740, 0.0740],
       device='cuda:0', grad_fn=<SqueezeBackward1>)
Labels:tensor([-0.0826,  2.3987,  0.5654,  0.5221,  0.0799, -0.5190,  0.5618,  1.2370,
         0.3547,  3.1422,  0.5712,  2.2340,  2.0741,  0.0363,  0.2902,  0.4702],
       device='cuda:0')
```

Während die Labels, also die tatsächliche Wahrheit, jeden Tag unterschiedlich ist, so sagt das Modell immer denselben Wert vorher. In der nächsten Batch tritt das selbe Phänomen auf nur mit einem anderen Wert. Dadurch lässt sich durchaus beobachten, dass zwischen den Batches das Modell lernt, jedoch scheinbar nicht genug um eine Zeitreihe differenziert vorherzusagen. Eine mögliche Erklärung wäre vielleicht ein Programmierfehler, welcher dazu führt, dass ein Output des Modells für die ganze Batch genutzt wird. Dies lässt sich jedoch widerlegen, da dieses Phänomen nicht immer auftritt, was an folgendem Beispiel deutlich wird:

```
Training === Epoch: 4/4 Loss: 1.5472071170806885 Stock: NEN Batch: 41
##################################################

Output:tensor([ 0.2793,  0.2793,  0.2793,  0.2793,  0.2793, -0.2341,  0.2793,  0.2793,
         0.2793,  0.2793, -0.2341,  0.2793,  0.2793, -0.2341, -0.2341,  0.2793],
       device='cuda:0', grad_fn=<SqueezeBackward1>)
Labels:tensor([ 1.0187,  1.1525, -2.0346, -1.3084,  1.4310, -0.4564, -0.7295,  0.7978,
        -0.0208,  0.4167, -0.4149,  1.6667,  1.3320, -1.8605,  1.2776, -1.9125],
       device='cuda:0')

##################################################
Training === Epoch: 4/4 Loss: 2.202139377593994 Stock: NEN Batch: 42
##################################################

Output:tensor([-0.2598,  0.3119,  0.3119, -0.2598,  0.3119,  0.3119,  0.3119,  0.3119,
         0.3119,  0.3119, -0.2598,  0.3119,  0.3119,  0.3119, -0.2598,  0.3119],
       device='cuda:0', grad_fn=<SqueezeBackward1>)
Labels:tensor([ 0.8090,  0.8436, -0.0204,  0.5918, -1.0144,  0.4304, -0.8776,  0.1441,
         1.4803, -0.0203, -2.5127,  0.3118,  3.6055, -2.0000,  2.0408, -1.0000],
       device='cuda:0')

##################################################
Training === Epoch: 4/4 Loss: 1.4931708574295044 Stock: NEN Batch: 43
##################################################

Output:tensor([ 0.3418, -0.2569,  0.3418, -0.2569,  0.3418,  0.3418,  0.3418,  0.3418,
         0.3418,  0.3418,  0.3418, -0.2569, -0.2569,  0.3418,  0.3418,  0.3418],
       device='cuda:0', grad_fn=<SqueezeBackward1>)
Labels:tensor([ 1.0101, -2.1000,  0.4494,  0.3254,  0.0203,  0.1216, -1.6397,  0.9673,
         0.4280, -1.3396,  0.0823, -0.7194,  2.3602, -0.1820, -1.8845,  0.1652],
```

Ein grundsätzlicher Fehler im Programm scheint also nicht vorzuliegen. Vielmehr handelt es sich um das Verhalten des Modells selbst. Auch lässt sich beobachten, dass selbst bei differenzierten Vorhersagen die Vorhersagen recht ähnlich zueinander sind. Dies könnte mehrere Gründe haben. Zum einen sind die tatsächlichen Renditen meist nah um Null verteilt, weshalb, wie auch schon in vorherigen Abschnitten erläutert, eine Vorhersage besonders herausfordernd ist, gerade durch die Natur des Problems selbst. Zum anderen könnte dies daran liegen, dass das Lernen besonders erschwert ist sei es durch die geringe Anzahl an Inputdaten oder durch die Wahl der Hyperparameter. Letzteres wird nun näher in Betracht gezogen. Denn es fällt auf, dass das genannte Phänomen eher auftritt, wenn

die Epochenanzahl niedrig (konkret bei 2) liegt. Erhöht man diese (auf 4) lassen sich eher differenziert Vorhersagen beobachten, welche durch das längere Training erlernt werden können. Dabei lassen sich große Unterschiede beim Verlust jedoch nicht feststellen. Auf der anderen Seite können zwar kleine Verbesserungen durch Vergrößerung der Lernrate festgestellt werden, starke Veränderungen des Verlustes sind jedoch nicht zu beobachten. Des Weiteren führen kleine Änderungen der Lernrate (wie 0.001 zu 0.002) zu nahezu exakten Ergebnissen. Die größten Unterschiede lassen sich bei der Veränderung der Batchgröße beobachten. Hierbei ist deutlich, dass die Vergrößerung der Batchgröße eine deutliche Verschlechterung der Performance verursacht. Eine Begründunge hierfür könnte sein, dass zwar größere Batchgrößen im Kontext dieser Bachelorarbeit zwar eine genauere Berechnung des Verlustes und somit eine genauere Optimierung erreicht werden könnte. Zu große Batchgrößen führen jedoch zu einer zu geringen Anzahl an Optimierungen, auf Grund der geringeren Menge an Iterationen. Es gilt hierbei also eine optimale Kombination zu finden. Um bei größeren Batchgrößen zu mehr Iterationen zu gelangen könnte eine Erhöhung der Epochenanzahl erwägenswert sein. Grundsätzlich lässt sich jedoch feststellen, dass die Möglichkeiten für das LSTM ein geeignetes Modell für die Vorhersage von Aktienrenditen zu entwickeln sehr begrenzt sind auf Grund der geringen Menge an Input-Features, der geringen Dauer des Trainings sowie ggf. der geringen Größe des Modells selbst, wobei letzteres nicht durch die Resultate der Experimente erfasst werden konnte. Dies Aussage, dass das Modell mehr Zeit bzw. mehr Itertionen zum Lernen benötigt wird weiterhin dadurch gestützt, dass das Modell mit dem geringsten Verlust die höchste Epochenanzahl (4) sowie Lernrate (0.4) und die geringste Batchgröße (8) besitzt, was zu häufigen und starken Optimierungsschritten führt. Dies zeigt, dass das Modell für eine bessere Performance deutlich mehr lernen muss, was durch Erhöhung der Epochenanzahl und ggf. der Lernrate sowie Verringerung der Batchgröße erreicht werden kann. Weiterhin ist zu erwähnen, dass der durchschnittliche Verlust im Vergleich zu dem Verlust der üblicherweise vorkommt, was auch an den Ausschnitten aus dem Training ersichtlich ist, extrem hoch ist. Dies liegt daran, dass an manchen Stellen das die Vorhersage des Modells und die Realität weit auseinander gehen. So werden geringe Renditen (welche normalerweise vorkommen) vorhergesagt, jedoch finden in der Realität zu diesem Zeitpunkt äußerst hohe Renditen statt. Auch das Gegenteil kann der Fall sein. Beides ist in folgenden Ausschnitten ersichtlich:

```
####################################################
Training === Epoch: 2/2 Loss: 19781.037109375 Stock: FNRG Batch: 8
####################################################

Output:tensor([0.3104, 0.2356, 0.2251, 0.1496, 0.2111, 0.1118, 0.1773, 0.1680, 0.1912,
        1.3433, 0.2200, 0.2171, 0.1303, 0.1096, 0.2122, 0.1095, 0.2020, 0.1993,
        0.1991, 0.1980, 0.1107, 0.1104, 0.1932, 0.1109, 0.3483, 0.1095, 0.2035,
        0.1103, 0.1103, 0.1304, 0.9691, 0.1144, 0.1102, 0.1963, 0.1103, 0.1956],
       device='cuda:0', grad_fn=<SqueezeBackward1>)
Labels:tensor([500.0000, -83.3333, 100.0000, 150.0000, -58.0000,  -4.7619, 100.0000,
        -25.0000, -33.3333, 100.0000,  25.0000, -20.0000,  -5.0000,  31.5790,
         60.0000, -37.5000, -60.0000, 150.0000, -60.0000,  25.0000, -59.6000,
        395.0495, -60.0000, -50.0000, 100.0000, -75.0000, 300.0000, -50.0000,
        100.0000, -50.0000, 100.0000, 150.0000, -60.0000, 150.0000, -80.0000,
        100.0000], device='cuda:0')


####################################################
Training === Epoch: 3/4 Loss: 100.61073303222656 Stock: TR Batch: 89
####################################################

Output:tensor([9.7461, 9.7461, 9.7461, 9.7461, 9.7461, 9.7461, 9.7461, 9.7461, 9.7461,
        9.7461, 9.7461, 9.7461, 9.7461, 9.7461, 9.7461, 9.7461],
       device='cuda:0', grad_fn=<SqueezeBackward1>)
Labels:tensor([-0.5338, -1.2626,  1.0230,  0.7278,  0.1885,  0.4704,  0.7179,  1.5184,
        -0.8547, -2.7401, -2.1209, -0.6468,  0.2930, -1.2334, -0.0657,  1.0523],
       device='cuda:0')
```

Da die Renditen ca. der Normalverteilung folgen, sind die meisten Renditen, wie schon erwähnt, nahe Null, was das Modell auch lernt. Solche Ausreißer mit sehr hohen bzw. sehr niedrigen Renditen bereiten dem Modell jedoch Probleme. Das führt dazu, dass das Modell bei den vorher gelernten (niedrigen) Vorhersagen bleibt, was zu hohem Verlust führt, oder das Modell adaptiert zu hohen Renditen und versagt dann, wenn Aktien mit „normalen“ Renditen auftauchen.

Kapitel 7: Fazit

Zum Abschluss dieser Bacheloarbeit fasse ich kurz die erlangten Erkenntnisse zusammen und gebe einen Ausblick auf weitere mögliche Forschungsansätze. Obwohl das entwickelte LSTM Modell mit der umgesetzten Datenvorverarbeitung keine perfekten Vorhersagen treffen kann, können dennoch verschieden Erkenntnisse gewonnen werden. Zum einen konnte gezeigt werden, dass die Batchgröße eine deutliche Auswirkung auf die Perfomance haben kann. Die Begründung hierfür ist noch eine offene Frage in der Forschung, trotzdem kann dies möglicherweise mit der Anzahl an Iterationen und somit der Anzahl der Optimierungsschritte zusammenhängen. Auch konnte gezeigt werden, dass bei der Problemstellung dieser Bachelorarbeit ein intensives Training notwendig ist. Dies kann auch erkannt werden, wenn bei größerer Lernrate bzw. bei mehr Iterationen eine Verbesserung der Performance festgestellt werden kann. Dieses Wissen kann nützlich sein, da bei zukünftigen Optimierungsversuchen durch solche Anzeichen eine bessere Adjustierung der Hyperparameter möglich wird. Konkret kann dies also zeigen, welche Hyperparameter wie adjustiert werden müssen, um dem Modell bspw. mehr Optimierungsschritte zu gewähren. Ein weiteres Anzeichen hierfür kann auch der Output des Modells sein. Falls dieses kaum differenziert ist, kann dies ein Mangel an Lernzeit oder eine zu geringe Größe des Modells bedeuten. Auch die Größe des Datensatz oder die Anzahl der Input-Features können mögliche Fehlerquellen sein. Ähnliches gilt für die Problematik, die Ausreißer beim Training verursachen. Dies zeigt, dass eine reine Vorverarbeitung der Daten nicht ausreichend ist, besonders bei großen unübersichtlichen Datensätze. Stattdessen sollten die Ergebnisse des Modells mit einbezogen werden in die Art und Weise wie die Daten vorverarbeitet werden. So kann diskutiert werden, ob, je nach Anwendungsgebiet, das Entfernen von Ausreißern sinnvoll sein kann. Konkret kann es sinnvoll sein, kleine, stark schwankende Aktien zu entfernen, wenn das Ziel des Systems sein soll, Renditen von großen Unternehmen aus dem S&P 500 vorherzusagen. Auf der anderen Seite ist es wichtig die Daten realitätsnah zu halten, um nicht zu verfälschten Ergebnissen zu gelangen. Je nach Anwendungsgebiet muss dies also abgewogen werden. Insgesamt konnten also verschiedene Problematiken erkannte werden, welche durch die Analyse dieser Bachelorarbeit dazu genutzt werden können die Lösungsfindung bei zukünftige Optimierungsversuche von Machine Learning Modelle zu beschleunigen.

Ausblick

In dieser Bachelorarbeit konnten viele Aspekte der Hyperparameteroptimierung diskutiert werden. Da einige Hyperparameter jedoch nicht getestet werden konnte wäre hier zunächst interessant wie die Hyperparameter sich auf die Performance auswirken. Auch wäre es von Interesse wie sich die verfügbare Datenmenge insbesondere die Anwesenheit von Ausreißern sich auf die Performance auswirkt. Außerdem könnte versucht werden die Anzahl der Features zu erhöhen, um zu erkennen welche Features sich positiv und welche negativ auf die Performance des Modells auswirken. Unabhängig von der Optimierung des Modells wäre es interessant eine Erklärung zu finden für die Veränderung der Performance bei sich verändernder Batchgröße. Auch wenn bisher Erklärungsansätze hierfür existieren, so bleibt dies eine offen Frage. Es ist jedoch eindeutig, auch in dieser Bachelorarbeit, gezeigt worden, dass die Batchgröße durchaus eine große Auswirkung auf die Performance haben kann.

Appendix

Hidden Di	Number o	Number o	Learning R	Batch Size	Train Loss	Test Loss
2	1	2	0.0001	8	13784244	6862975
2	1	2	0.0001	16	13732130	8129618
2	1	2	0.0001	32	14818515	10531333
2	1	2	0.0001	64	16954373	10096392
2	1	2	0.0001	128	21578247	18211320
2	1	2	0.0001	256	22928328	32125147
2	1	2	0.0001	512	40293987	51140648
2	1	2	0.001	8	13784225	6862948
2	1	2	0.001	16	13732116	8129601
2	1	2	0.001	32	14818510	10531323
2	1	2	0.001	64	16954357	10096377
2	1	2	0.001	128	21578228	18211297
2	1	2	0.001	256	22928326	32125158
2	1	2	0.001	512	40293987	51140641
2	1	2	0.01	8	13784140	6862839
2	1	2	0.01	16	13732076	8129540
2	1	2	0.01	32	14818483	10531297
2	1	2	0.01	64	16954364	10096380
2	1	2	0.01	128	21578170	18211236
2	1	2	0.01	256	22928303	32125105
2	1	2	0.01	512	40293855	51140298
2	1	2	0.1	8	13784063	6862777
2	1	2	0.1	16	13732004	8129455
2	1	2	0.1	32	14818337	10531067
2	1	2	0.1	64	16954320	10096312
2	1	2	0.1	128	21577828	18210716
2	1	2	0.1	256	22928226	32124866
2	1	2	0.1	512	51139791	40293708
2	1	2	0.002	8	13784235	6862948
2	1	2	0.002	16	13732104	8129581
2	1	2	0.002	32	14818502	10531305
2	1	2	0.002	64	16954363	10096385
2	1	2	0.002	128	21578185	18211235
2	1	2	0.002	256	22928319	32125116
2	1	2	0.002	512	40293976	51140615
2	1	2	0.02	8	13784166	6862861
2	1	2	0.02	16	13732090	8129580
2	1	2	0.02	32	14818543	10531312
2	1	2	0.02	64	16954356	10096357
2	1	2	0.02	128	21578094	18211136
2	1	2	0.02	256	22928285	32125048
2	1	2	0.02	512	40293873	51140300

2	1	2	0.02	8	13784166	6862861
2	1	2	0.02	16	13732090	8129580
2	1	2	0.02	32	14818543	10531312
2	1	2	0.02	64	16954356	10096357
2	1	2	0.02	128	21578094	18211136
2	1	2	0.02	256	22928285	32125048
2	1	2	0.02	512	40293873	51140300
2	1	2	0.2	8	13783601	6862244
2	1	2	0.2	16	13732432	8129277
2	1	2	0.2	32	14818085	10530841
2	1	2	0.2	64	16953906	10095950
2	1	2	0.2	128	21578092	18211072
2	1	2	0.2	256	22928298	32125042
2	1	2	0.2	512	40293396	51138559
2	1	2	0.004	8	13784231	6862954
2	1	2	0.004	16	13732100	8129552
2	1	2	0.004	32	14818501	10531303
2	1	2	0.004	64	16954368	10096385
2	1	2	0.004	128	21578207	18211270
2	1	2	0.004	256	22928306	32125082
2	1	2	0.004	512	40293971	51140626
2	1	2	0.04	8	13783995	6862693
2	1	2	0.04	16	13732017	8129546
2	1	2	0.04	32	14818378	10531099
2	1	2	0.04	64	16954363	10096361
2	1	2	0.04	128	21578195	18211242
2	1	2	0.04	256	22928262	32124994
2	1	2	0.04	512	40293809	51140268
2	1	2	0.4	8	13783482	6862013
2	1	2	0.4	16	13730979	8128133
2	1	2	0.4	32	14818241	10530703
2	1	2	0.4	64	16954138	10096047
2	1	2	0.4	128	21578143	18210994
2	1	2	0.4	256	22928304	32124979
2	1	2	0.4	512	40293935	51140308
2	1	2	0.00001	8	13784251	6862985
2	1	2	0.00001	16	13732139	8129635
2	1	2	0.00001	32	14818521	10531339
2	1	2	0.00001	64	16954386	10096404
2	1	2	0.00001	128	21578249	18211323
2	1	2	0.00001	256	22928330	32125167
2	1	2	0.00001	512	40294042	51140790
2	1	4	0.1	8	13783850	6862516
2	1	4	0.1	16	13731893	8129217
2	1	4	0.1	32	14818177	10530916
2	1	4	0.1	64	16954155	10096057
2	1	4	0.1	128	21578001	18210683

2	1	4	0.1	256	22928160	32124669
2	1	4	0.1	512	40293834	51140246
2	1	4	0.2	8	13783472	6862065
2	1	4	0.2	16	13731275	8128347
2	1	4	0.2	32	14817583	10530065
2	1	4	0.2	64	16954143	10096072
2	1	4	0.2	128	21578087	18210963
2	1	4	0.2	256	22928093	32124446
2	1	4	0.2	512	40293867	51140195
2	1	4	0.4	8	13782745	6861208
2	1	4	0.4	16	13730846	8127605
2	1	4	0.4	32	14817959	10530230
2	1	4	0.4	64	16953850	10095662
2	1	4	0.4	128	21577866	18210550
2	1	4	0.4	256	22927354	32121883
2	1	4	0.4	512	40293523	51138910

Quellen

[1] „Wie Amazon lernte was wir wollten“ – Artikel über Algorithmen zur Produktempfehlung [Abruf: 16.03.20]

https://www.zeit.de/digital/2019-07/amazon-algorithmus-greg-linden-empfehlungssysteme

[2] “Political Campaigns Know Where You’ve Been. They’re Tracking Your Phone.” – Artikel über die Nutzung von Daten bei Wahlkampagnen [Abruf: 16.03.20]

https://www.wsj.com/articles/political-campaigns-track-cellphones-to-identify-and-target-individual-voters-11570718889

[3] „Algorithmischer Handel/HFT“ – Artikel zur Definition von algorithmischem Handel [Abruf: 15.03.20]

https://www.deutsche-boerse.com/dbg-de/regulierung/regulatory-dossiers/mifid-mifir/mifid-i-to-mifid-ii/market-structure/algo-hft

[4] „Algorithmic Trading Systems“ –Vorstellung von algorithmischen Handelssystemen [Abruf: 15.03.20]

https://seekingalpha.com/instablog/40574325-peter-knight/4926454-algorithmic-trading-systems

[5] Morton Glantz, Robert Kissell. *Multi-Asset Risk Modeling: Techniques for a Global Economy in an Electronic and Algorithmic Trading Era*. Academic Press, 2013

[6] Lin, Tom C. W., The New Investor. 60 UCLA Law Review 678 (2013); 60 UCLA Law Review 678 (2013); Temple University Legal Studies Research Paper No. 2013-45. https://ssrn.com/abstract=2227498

[7] “Trading Halts Don’t Stop Stock Market Carnage” – Aktien fallen stark nach Ankündigung von Trump [Abruf: 15.03.20]

https://www.nasdaq.com/articles/trading-halts-dont-stop-market-carnage-2020-03-12

[8] Bing Liu: Sentiment Analysis and Opinion Mining, Morgan & Claypool Publishers, 2012

https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf

[9] Bing Liu: Sentiment Analysis: Mining Opinions, Sentiments and Emotions, Cambridge University Press, 2015

[10] Geschichte der Wertpapiere [Abruf: 15.12.20]

https://www.kapilendo.de/magazin/geschichte-der-wertpapiere-vom-pfeffer-bis-zum-digitalen-wertpapier-in-2019/

[11] Algorithmischer Handel Definition [Abruf: 25.12.20]

https://www.investopedia.com/terms/a/algorithmictrading.asp

[12] Börse Entstehungsgeschichte [25.12.20]

https://www.godmode-trader.de/know-how/die-boerse-entstehungsgeschichte-und-bedeutung,3751100

[13] Geschichte der Börse [25.12.20]

https://finanzthema.com/die-geschichte-der-boerse.html/

[14] Historie Aktienmärkte [25.12.20]

https://www.investopedia.com/articles/07/stock-exchange-history.asp

[15] Historie Xetra [25.12.20]

https://www.boerse-frankfurt.de/wissen/ueber/geschichte-der-frankfurter-wertpapierboerse/xetra-der-elektronische-handelsplatz-der-deutschen-boerse

[16] Definition algorithmischer Handel [Abruf: 26.12.20]

https://www.investopedia.com/terms/a/algorithmictrading.asp

[17] Geschichte des algorithmischen Handels [Abruf: 26.12.20]

https://medium.com/the-capital/a-brief-history-of-algorithmic-trading-1ac7e90e153f

[18] Historie algorithmischer Handel, HFT und nachrichtenbasierter Handel [Abruf: 26.12.20]

https://blog.quantinsti.com/history-algorithmic-trading-hft/

[19] Alternativen zu HFT [Abruf: 26.12.20]

https://www.investopedia.com/articles/active-trading/081215/new-alternatives-highfrequency-trading.asp

[20] Erklärung von technischer- und Fundamentalanalyse [Abruf: 17.01.21]

https://www.investopedia.com/ask/answers/difference-between-fundamental-and-technical-analysis/

[21] Grundlegende algorithmische Handelsstrategien [Abruf: 17.01.21]

https://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp

[22] Moving Averages Erklärung [Abruf: 17.01.21]

https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp

[23] Arbitrage Erklärung [Abruf: 17.01.21]

https://www.investopedia.com/ask/answers/what-is-arbitrage/

[24] Erklärung einiger Handelsstrategien u.a. Index Fund Rebalancing [Abruf: 17.01.21]

https://www.datadriveninvestor.com/2019/01/31/a-quick-guide-to-investment-algorithms/

[25] Delta Hedging Erklärung [17.01.21]

https://www.investopedia.com/terms/d/deltahedging.asp

[26] Over-Hedging Definition [17.01.21]

https://www.investopedia.com/terms/o/overhedging.asp

[27] Mean-Reversion Erklärung [17.01.21]

https://www.investopedia.com/terms/m/meanreversion.asp

[28] VWAP Erklärung [17.01.21]

https://www.investopedia.com/terms/v/vwap.asp

[29] Dollar-Cost-Averaging Erklärung [17.01.21]

https://www.investopedia.com/terms/d/dollarcostaveraging.asp

[30] Momentum Trading Erklärung [17.01.21]

https://www.investopedia.com/trading/introduction-to-momentum-trading/

[31] Momentum Indikatoren Erklärung [17.01.21]

https://www.investopedia.com/ask/answers/05/measuringmomentum.asp

[32] Rate of Change Erklärung [17.01.21]

https://www.investopedia.com/terms/r/rateofchange.asp

[33] Ausführliche Erklärung von Rate of Change [17.01.21]

https://school.stockcharts.com/doku.php?id=technical_indicators:rate_of_change_roc_and_momentum

[34] Diskussion vom Rate of Change Momentum-Indikator [17.01.21]

https://www.investopedia.com/terms/p/pricerateofchange.asp

[35] Stefan Feuerriegel, Helmut Prendinger: News-based trading strategies, 2018

[36] Mögliche Erklärungen für Momentum Investing [Abruf: 19.01.21]

https://anderson-review.ucla.edu/momentum/

[37] Evaluation von Momentum Investing durch MSCI [Abruf: 19.01.21]

https://www.msci.com/documents/1296102/1339060/Factor+Factsheets+Momentum.pdf/a766ef6b-cd24-4460-8163-900323fc2957

[38] Evaluation von Momentum Investing durch iShares [Abruf: 19.01.21]

https://www.ishares.com/ch/institutional/en/themes/smart-beta/momentum-investing?switchLocale=y&siteEntryPassthrough=true

[39] MSCI World Value Index Beschreibung (unter anderem Methodik des Indexes) [Abruf: 19.01.21]

https://www.msci.com/documents/10199/25465a5a-d52c-4bec-b5ed-a7b56eca8e0d

[40] Zura Kakushadze, Juan Andrés Serur: 151 Trading Strategies, 2018

[41] Clifford S. Asness, Tobias J. Moskowitz, Lasse Heje Pedersen: Value and Momentum Everywhere, 2013

[42] Clifford S. Asness, Andrea Frazzini, Ronen Isreal, Tobias S. Moskowitz: Fact, Fiction and Momentum Investing, 2014

[43] Diskussion warum technische Analyse von Wetpapieren funktioniert [20.01.21]

https://www.thelondoneconomic.com/lifestyle/my-business/why-does-technical-analysis-work/23/05/

[44] Qiang Song, Anqi Liu, Steve Y. Yang: Stock portfolio selection using learning-to-rank algorithms with news sentiment, 2017

[45] Artikel über Handelsstrategien, welche aufhören zu funktionieren [02.02.21]

https://therobusttrader.com/why-do-trading-strategies-stop-working/

[46] Historie Renaissance Technologies [02.02.21]

https://medium.com/ml-everything/how-renaissance-technologies-solved-the-market-part-1-2814eb271dc3

[47] Überblick algorithmischer Handel [02.02.21]

https://catanacapital.com/de/algorithmischer-handel-hochfrequenzhandel-automatisierter-boersenhandel/

[48] Auswirkungen der Covid-19 Pandemie auf die Wirtschaft [03.02.21]

https://www.bbc.com/news/business-51706225

[49] Raéf Bahrini, Assaf Filfilan: Impact of the novel coronavirus on stock market returns: evidence from GCC countries, 2020

[50] Abhishek Nan, Anandh Perumal, Osmar R. Zaiane: Sentiment and Knowledge Based Algorithmic Trading with Deep Reinforcement Learning, 2020

[51] Werner Antweiler, Murray Z. Frank: Is All That Talk Just Noise? The Information Content of Internet Stock Message Boards, 2004

[52] Paul C. Tetlock: Giving Content to Investor Sentiment: The Role of Media in the Stock Market, 2007

[53] Steve Y. Yang, Qiang Song, Sheung Yin Kevin Mo, Kaushik Datta, Anil Deane: The Impact of Abnormal News Sentiment on Financial Markets, 2015

[54] Stefan Jansen: Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python 2nd Edition, Packt Publishing, 2020

[55] Sharpe Ratio Erklärung [16.02.21]

https://boersenlexikon.faz.net/definition/sharpe-ratio/

[56] Volatilität ist kein Risiko [Abruf: 16.02.21]

https://argifinancialgroup.com/volatility-not-risk/#:~:text=Volatility%20can%20be%20considered%20a,the%20entirety%20of%20the%20market.

[57] Victor DeMiguel, Lorenzo Garlappi, Raman Uppal: Optimal Versus Naive Classification: How Inefficient is the 1/N Portfolio Strategy?, 2009

[58] Johannes Bock: An updated review of (sub-)optimal diversification models, 2018

[59] Vorstellung von Backtesting [19.02.21]

https://vinance.vn/posts/for-loop-backtesting-vectorized-backtesting-and-event-driven-backtesting

[60] Machine Learning Definition [23.02.21]

https://www.lernen-wie-maschinen.ai/ki-pedia/was-ist-maschinelles-lernen/#:~:text=Im%20Jahre%201997%20lieferte%20Tom,%E2%80%9C

[61] Vorstellung der Bereiche der KI [23.02.21]

https://www.iks.fraunhofer.de/de/themen/kuenstliche-intelligenz.html

[62] Neo Wu, Bradley Green, Xue Ben, Shawn O'Banion: Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case, 2020

[63] Vergleich zwischen LSTMs und ARIMA [Abruf: 02.03.21]

https://towardsdatascience.com/arima-vs-lstm-forecasting-electricity-consumption-3215b086da77

[64] Vergleich Deep Learning und klassisches Machine Learning [02.03.21]

https://towardsdatascience.com/deep-learning-vs-classical-machine-learning-9a42c6d48aa

[65] Masaya Abe, Hideki Nakayama: Deep Learning for Forecasting Stock Returns in the Cross-Section, 2018

[66] Diskussion von Vor- und Nachteilen von Deep Learning und klassischem Machine Learning [02.03.21]

https://blog.dataiku.com/when-and-when-not-to-use-deep-learning

[67] Diskussion von Reinforcement Learning [02.03.21]

https://www.kdnuggets.com/2017/12/when-reinforcement-learning-not-used.html

[68] Yoav Goldberg: Neural Network Methods in Natural Language Processing, 2017

[69] Zweite Vorlesung über RNNs aus Intro to Deep Learning vom MIT [Abruf: 03.03.21]

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

[70] Erklärung Exploding Gradient [Abruf: 03.03.21]

https://machinelearningmastery.com/exploding-gradients-in-neural-networks/

[71] Definition Gradient Clipping [Abruf: 03.03.21]

https://deepai.org/machine-learning-glossary-and-terms/gradient-clipping

[72] Sepp Hochreiter, Jürgen Schmidhuber: Long short-term memory, 1997

[73] Erklärung von RNNs und LSTMs von Google Mitarbeitern [Abruf: 03.03.21]

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[74] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever: An Empirical Exploration of Recurrent Network Architectures, 2015

[75] Klaus Greff, Ripesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber: LSTM: A Search Space Odyssey, 2015

[76] Beschreibung der Nachteile von RNNs und LSTMs [Abruf: 03.03.21]

https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0

[77] Full Stack Deep Learning RNN & LSTM Vorlesung [Abruf: 03.03.21]

https://fullstackdeeplearning.com/spring2021/lecture-3/

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention Is All You Need, 2017

[79] S.E. Yi, A. Viscardi, T. Hollis: A Comparison of LSTMs and Attention Mechanisms for Forecasting Financial Time Series, 2018

[80] AlphaGo Erklärung [09.03.21]

https://deepmind.com/research/case-studies/alphago-the-story-so-far

[81] Vorstellung der 5 populärsten Programmiersprachen für Machine Learning [Abruf: 09.03.21]

https://www.geeksforgeeks.org/top-5-programming-languages-and-their-libraries-for-machine-learning-in-2020/

[82] Menschliche Genauigkeit bei der Sentiment Analysis [16.03.21]

https://www.lexalytics.com/lexablog/sentiment-accuracy-baseline-testing#:~:text=When%20evaluating%20the%20sentiment%20(positive,%2D85%25%20of%20the%20time.&text=But%20when%20you're%20running,that%20the%20results%20are%20reliable.

[83] Artikel zu Insider Käufen und Verkäufen [26.03.21]

https://edition.cnn.com/2019/05/22/success/wealth-coach-insider-selling/index.html

[84] Adjustierter Schlusskurs Definition [26.03.21]

https://www.investopedia.com/terms/a/adjusted_closing_price.asp

[85] Trainings- und Testdatenverhältnis [29.03.21]

https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

[86] Example #1 of nearly perfect price prediction [04.04.21]

https://www.kaggle.com/humamfauzi/multiple-stock-prediction-using-single-nn#Working-of-gates-in-LSTMs

[87] Example #2 of nearly perfect price prediction [04.04.21]

https://www.kaggle.com/rodsaldanha/stock-prediction-pytorch/notebook

[88] Beschreibung der Auswirkungen sowie Umgang mit Maßstäben von Daten [12.04.21]

https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

[89] Stefan Feuerriegel, Dirk Neumann: Evaluation of News-Based Trading Strategies, 2016

[90] Vergleich von Tensorflow und PyTorch [Abruf: 16.04.21]

https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4

[91] Vergleich der Popularität von Tensorflow und PyTorch [Abruf: 16.04.21]

https://towardsdatascience.com/is-pytorch-catching-tensorflow-ca88f9128304

[92] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, Ilya Sutskever: Deep Double Descent: Where Bigger Models and More Data Hurt, 2019

[93] Samuel L. Smith, Pieter-Jan Kindermans, Christ Ying & Quoc V. Le: Don’t Decay The Learning Rate, Increase The Batch Size, 2018

[94] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh: Large Batch Optimization For Deep Learning: Training BERT in 76 Minutes, 2020

[95] Zachary Nado, Justin M. Gilmer, Christopher J. Shallue, Rohan Anil, George E. Dahl: A Large Batch Optimizer Check: Traditional, Generic Optimizer Suffice Across Batch Sizes, 2021

Eigenständigkeits Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt und mich keiner fremden Hilfe bedient sowie keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten oder nicht veröffentlichten Schriften oder anderen Quellen entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorlegen.

_____ _____

Ort, Datum Unterschrift