Anwendung von Machine Learning und Deep Learning im automatisierten Finanzhandel

Ansätze zur Entwicklung und Anwendung intelligenter Handelssysteme

Dogus Tansel (M. Sc.)

Technische Hochschule Mittelhessen

Fachbereich MND
Wilhelm-Leuschner-Straße 13
61169 Friedberg
E-Mail: tansel.dogus@gmail.com

Prof. Dr. Harald Ritz

Technische Hochschule Mittelhessen

Fachbereich MNI
Wiesentraße14
35390 Gießen
E-Mail: harald.ritz@mni.thm.de

Prof. Dr. Oliver Hein

Technische Hochschule Mittelhessen

Fachbereich MND
Wilhelm-Leuschner-Straße 13
61169 Friedberg
E-Mail: oliver.hein@mnd.thm.de

Abstract

Im Zuge der steigenden Verfügbarkeit von Daten (bis zum Jahr 2025 weltweit über 180 Zettabyte (Redgate 2021)) und durch die rasante Entwicklung im Rahmen künstlicher Intelligenz kristallisieren sich zunehmend zukunftsorientierte Ansätze zur Automatisierung des finanziellen Handels mit Blick auf Machine Learning (ML)- und Deep Learning (DL)-Methoden heraus. Insbesondere DL-Modelle haben in den letzten Jahren im Finanzsektor an Bedeutung gewonnen, da sie in der Lage sind, komplexe Zusammenhänge und Muster innerhalb der Datenmengen zu erkennen, die für menschliche Händler schwer zugänglich sind (Magwa 2023).

Der vorliegende Artikel stellt ganzheitliche Ansätze zur Entwicklung intelligenter Handelssysteme und die damit einhergehende Nutzbarkeit vor. Hierbei ist das Ziel, zu untersuchen, wie der Einsatz von DL-Modellen innerhalb eines Handelssystems realisiert werden kann. Im Rahmen dieser Realisierung steht dabei die Entwicklung und Anwendung eines algorithmischen Handelssystems für das Treffen automatisierter Handelsentscheidungen im Fokus. Als Ergebnis wurde deutlich, dass der Einsatz von DL-Modellen eine Unterstützung für das aktive Handeln mit Preis- oder Trendvorhersagen sein kann. Teilnehmer auf den Finanzmärkten, die algorithmische Handelssysteme einsetzen, stehen vor der spannenden Aufgabe, verschiedene Ansätze zu berücksichtigen. Ob sie sich für den Einsatz von klassischen ML- oder erweiterten DL-Modellen entscheiden, um Muster in umfangreichen Datenmengen zu erkennen, oder ob Techniken zur Preis- und Trendvorhersage eingesetzt werden, hängt von den individuellen Zielen und Präferenzen ab.

Schlüsselwörter

Algorithmic Trading, Deep Learning, Machine Learning, Tradingbot, Stable Baselines3

I. AUSGANGSSITUATION

Die dynamische Welt der Finanzmärkte hat sich in den letzten Jahrzehnten radikal verändert, wobei technologische Fortschritte und die Verfügbarkeit großer Datenmengen neue Möglichkeiten für den Handel eröffnet haben. In diesem Zeitalter des digitalen Wandels hat sich der algorithmische Handel zu einem Schlüsselfaktor entwickelt, der den Finanzsektor sowie die Art und Weise, wie Handelsentscheidungen getroffen werden, revolutioniert. So erzielte laut SelectUSA allein der automatisierte Handel im Jahr 2018 bereits um die 60 – 73% des gesamten Handelsvolumens auf dem US-Aktienmarkt (Shah 2019). Dabei wird die erwartete weltweite Steigerung der Marktgröße des algorithmischen Handels mit derzeit 15,77 Mrd. US$ im Jahr 2023 wird auf 23,74 Mrd. US$ für das Jahr 2028 geschätzt (Mordor Intelligence o. D.).

Angesichts dieser auffallenden Zahlen und des anhaltenden Wachstumspotenzials der aufkommenden Technologieneuerungen ist es fundamental, den Markt für algorithmischen Handel weiter zu erforschen und zu verstehen. Aufgrund der divergierenden Menge an Daten und der damit einhergehenden Analyse auf dem Finanzmarkt wird es für konventionelle Händler, auch als Trader bezeichnet, immer herausfordernder die richtige Handelsentscheidung auf der Grundlage der Marktsituation zu treffen.

Die einst konventionelle Analyse des Marktes und das Eröffnen von Handelspositionen profitiert durch den Einsatz intelligenter Handelssysteme und optimiert die Entscheidungsfindung mit Hilfe von großen Datenmengen in Echtzeit. ML, insbesondere DL, hat sich als leistungsstarke Methode erwiesen, um Muster in riesigen Datensätzen zu erkennen und präzise Handelsentscheidungen zu

treffen, die für Trader schwer zugänglich sind. Dieser Ansatz verspricht, die herkömmlichen Methoden des algorithmischen Handels zu erweitern und in Echtzeit auf sich ändernde Marktbedingungen zu reagieren.

II. VERWANDTE ARBEITEN

In der Forschung gibt es inzwischen vielfältige Arbeiten, die sich mit der Entwicklung von Handelssystemen für die Teilbereiche des Finanzmarktes beschäftigen. Einige dieser Forschungsarbeiten haben ihren Schwerpunkt auf den Einsatz von LSTM-Modellen zur Prognose des Schlusspreises von Finanzanlagen gelegt (Huang, Huang und Chen 2022), (Singh, Thulasiram und Thavaneswaran 2022). Andere Forschungsarbeiten fokussieren sich auf die Anwendung und Kombination von Long Short-Term Memory (LSTM) und Deep Reinforcement Learning (DRL)-Modellen, um zeitliche Muster in Finanzdaten präziser zu analysieren und darauf aufbauend Handelsstrategien zu entwickeln (Ansari, et al. 2022). Insgesamt haben diese Arbeiten gezeigt, dass DL-Modelle komplexe Zusammenhänge in den Marktbewegungen erkennen sowie die Entscheidungsfindung in Echtzeit verbessern.

III. HERAUSFORDERUNGEN

Das Ziel des algorithmischen Handelssystems ist es, einen ML-basierten Handelsbot für die Generierung von Kauf- und Verkaufsaufträgen (Ordererzeugung), Platzierung von Kauf- und Verkaufsaufträgen (Orderausführung), das Managen von bereits bestehenden Positionen (Schließen, Halten) sowie die Einhaltung von Risikomanagementmaßnahmen (z. B. Stop Loss, Take Profit) inkl. entsprechender Zuverlässig- und Geschwindigkeit zu entwickeln. Um dieses Ziel zu erreichen, sind insbesondere folgende Herausforderungen mit der Verwendung verbunden und sicherzustellen (Quantified Strategies 2023):

Die **Datenqualität** spielt eine entscheidende Rolle für die Genauigkeit und Effektivität der Handelsstrategien, die durch ML-Algorithmen entwickelt werden. Schlechte Daten können zu ungenauen Vorhersagen und fehlerhaften Handelsentscheidungen führen, weswegen eine Sicherstellung der Qualität essenziell für den Erfolg eines Handelssystems ist. Die Qualität und Vollständigkeit der Trainingsdaten sind entscheidend für die Leistung der Modelle bzw. des Systems.

Verzerrte oder unvollständige Daten können zu fehlerhaften Handelsstrategien führen, weshalb auf eine Reduzierung der **Datenverzerrung (Bias)** Rücksicht genommen werden muss.

Overfitting (zu dt. Überanpassung) beschreibt den Zustand eines Modells, sobald dieses zu stark auf historische Daten trainiert worden ist und dadurch den Datensatz memorisiert. Infolgedessen entstehen weniger effektive Ergebnisse im Einsatz bei der Anwendung auf neue und unbekannte Daten.

Underfitting (zu dt. Unteranpassung) tritt auf, wenn ein Modell während des Trainings zu wenig auf die Merkmale des Trainingsdatensatzes eingeht und dadurch nicht in der Lage ist, selbst die vorhandenen Muster angemessen zu erfassen. Im Gegensatz zum Overfitting deutet Underfitting darauf hin, dass das Modell zu einfach bzw. zu allgemein gehalten ist, um die zugrunde liegenden Strukturen innerhalb der Daten zu verstehen.

Die **Interpretierbarkeit** der Modelle durch den Anwender muss bei jeder Handelsentscheidung nachvollziehbar sein. **Menschliches Eingreifen** ist trotz der fortschrittlichen Technologie weiterhin notwendig und sollte mit der Zeit immer mehr reduziert werden. Sichterstellung bei der **Einhaltung gesetzlicher Vorschriften** für einen risikofreien Handel.

Sobald bei der Implementierung eines ML-basierten algorithmischen Handelssystems die aufgeführten Herausforderungen berücksichtigt werden, können Trader von den zahlreichen Vorteilen profitieren, die ML in Bezug auf eine präzisere Marktanalyse, automatisierte Handelsentscheidungen und verbesserte Risikomanagementstrategien mit sich bringt. Durch die effektive Nutzung von ML kann der algorithmische Handel effizienter und profitabler gestaltet werden, während menschliches Fachwissen und Überwachung gewährleisten, dass die Strategien den regulatorischen Standards entsprechen. Somit eröffnet der Einsatz von ML im algorithmischen Handel neue Perspektiven für Trader und trägt zur kontinuierlichen Weiterentwicklung und Innovation auf den Finanzmärkten bei. Die Einhaltung wird dazu beitragen, dass das Handelssystem effektiv und nützlich für den operativen Einsatz durch Trader ist sowie die Effizienz und Genauigkeit der Handelsprognosen zu verbessern.

IV. THEORETISCHER RAHMEN

In diesem Abschnitt wird zunächst auf den theoretischen Hintergrund und dann im darauffolgenden Kapitel auf die Implementierung eingegangen.

Candlesticks

Candlesticks, auch als Kerzenhalter bezeichnet, stellen eine nicht-konventionelle, aber bedeutende Darstellungsmethode für die Analyse bzw. den Handel auf Märkten dar. Bei der Analyse einzelner Candlestick-Diagramme liefern diese Informationen über die Eröffnungs-, Höchst-, Tiefst- und Schlusskurse des Marktes. In der englischen Sprache wird dies oft mit der Abkürzung OHLC (Open, High, Low, Close) abgekürzt (Udagawa 2018). Mit diesen Kursdaten bzw. dieser Datenstruktur wird im weiteren Verlauf das Training und die Anwendung des Handelssystems auf dem Markt durchgeführt.

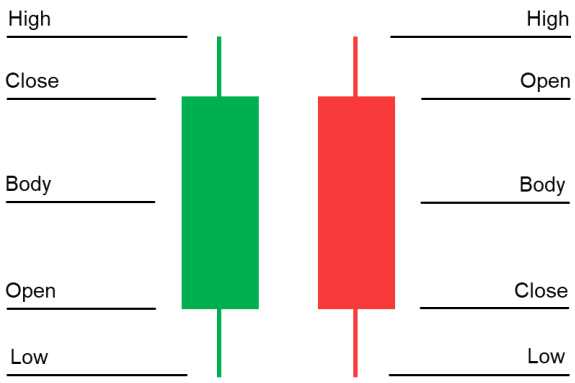

Abbildung 1 Candlesticks (Bullish links, Bearish rechts)

Algorithmische Handelssysteme

Konventionelle algorithmische Handelssysteme bauen auf der Grundlage markttechnischer Bedingungen auf. Dabei werden bewährte technische Indikatoren für das Implementieren von Handelssignalen verwendet. Aufgrund dessen existieren unterschiedliche Arten von markttechnischen Handelssystemen:

Trendfolgesysteme versuchen, von bestehenden Trends in den Finanzmärkten zu profitieren. Sie identifizieren Aufwärts- oder Abwärtstrends und setzen auf steigende oder fallende Kurse (Hofmann 2014). Im Gegensatz zu trendfolgenden Systemen versuchen *Kontra-Trend-Systeme* Trendumkehrungen zu erkennen und davon zu profitieren. Sie setzen darauf, dass der Markt nach einem starken Trend in die entgegengesetzte Richtung korrigiert. (Economy-Pedia o. D.) *Channel-Breakout-Systeme* signalisieren mögliche Handelssignale beim Ausbruch von Kursen aus einem etablierten, durch Trendlinien hergeleiteten Preiskanal (Kuepper, Stapleton und Schmitt 2021).

Machine Learning

ML bildet einen Teilbereich der KI ab, bei dem Algorithmen entwickelt werden, die auf der Grundlage von aufbereiteten Daten lernen können. Hierbei ist das automatische Erkennen von Patterns innerhalb der Datensätze essenziell, um Vorhersagen mit einer hohen Genauigkeit treffen zu können. ML greift dabei auf verschiedene Instrumente der statistischen Analyseverfahren wie Regression, Klassifikation und Clustering zurück (Alpaydın 2004, 1-14).

ML-Standardprozess

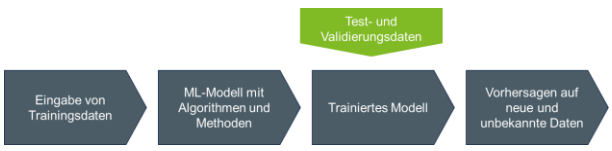

Abbildung 2 ML-Prozess (vgl. (Wuttke o. D.))

Ganz gleich, um welche Art von ML-Modell (Supervised, Unsupervised und Reinforcement Learning) es sich handelt, das grundlegende Prinzip der Ausführung bleibt unabhängig vom Anwendungsbereich dasselbe. Im ersten Schritt wird der zu bearbeitende (Trainings-)Datensatz definiert und in das gewünschte System eingespeist. Anschließend erfolgt der Aufbau des Modells unter Verwendung von Algorithmen und Methoden sowie notwendigen Datenaufbereitungsverfahren. Dabei werden Zusammenhänge, Patterns, Korrelationen und Strukturen aus den Daten abgeleitet und dem System als Ausgabe zur Verfügung gestellt (trainiertes Modell). Zum Abschluss können auf Grundlage des trainierten Modells neue und unbekannte Test- sowie Validierungsdaten eingespielt werden, um bspw. Vorhersagen für einen Anwendungsfall zu erstellen (Wuttke o. D.).

ML im algorithmischen Handel

ML bietet aufgrund seiner Fähigkeit, Patterns in großen Datenmengen zu erkennen, vielfältige Möglichkeiten für den Handel. Diese fortschrittliche Technologie kann auf abwechslungsreiche Weise Einfluss nehmen und den Handelsprozess optimieren. Nachfolgend sind einige der potenziellen Einsatzbereiche und Anwendungsmöglichkeiten von ML im Handel aufgeführt (Chopra 2022).

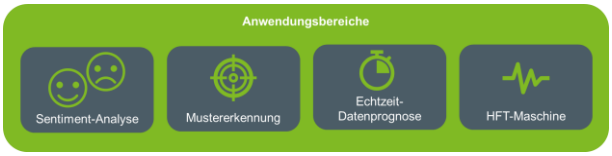

Abbildung 3 Anwendung ML im algorithmischen. Handel (vgl. (Chopra 2022))

Sentiment-Analyse: ML kann mithilfe von Daten aus verschiedenen Quellen wie Nachrichten, sozialen Medien und anderen unstrukturierten Daten Tradern ermöglichen die Marktsituation oder das Anlegerverhalten zu erkennen. Mit der Hilfe von Natural Language Processing (NLP) wird der Kontext der Daten analysiert, ausgewertet sowie verstanden, um die sogenannte Marktstimmung zu bestimmen.

Mustererkennung: ML entlastet die Trader von zeitaufwändigen Aufgaben der Datensammlung sowie -verarbeitung und trägt zur Reduzierung von manuellen Arbeiten durch Automatisierung der Analyse bei.

Echtzeit-Datenprognose: Maschinelle Lernalgorithmen können kontinuierlich Echtzeitdaten verarbeiten und lernen, um die Genauigkeit ihrer Vorhersagen zu verbessern. Sie können wichtige Aspekte wie globale Wetterbedingungen, politische Unruhen und den Klimawandel berücksichtigen, die direkte Auswirkungen auf die Handelsbranche haben.

HFT-Maschine: Mit der Hilfe einer Hochfrequenzhandelsmaschine wird Tradern das Eröffnen und Schließen

von tausenden Transaktionen pro Tag unter der Zuhilfenahme maschineller Algorithmen ermöglicht.

Mit einem Blick auf die Anwendungsbereiche kristallisiert der Einsatz von ML in der Erweiterung algorithmischer Handelssysteme neben der vereinfachten Handelsentscheidung deutliche bzw. bemerkbare Vorteile für einzelnen Teilnehmer des Finanzmarktes heraus (Quantified Strategies 2023):

- Verbesserte Genauigkeit von Handelssignalen
- Erhöhte Geschwindigkeit in der Auftragsausführung
- Verbessertes Risikomanagement durch die Reduzierung menschlicher Fehler und Einflüsse
- Zeitersparnis durch Automatisierung von wiederkehrenden Eingaben
- Situationsbedingte Anpassungsfähigkeit von Handelsstrategien

Reinforcement Learning

Abbildung 4 Reinforcement Learning (vgl. (Bhatt 2018))

Reinforcement Learning (verstärkendes Lernen) unterscheidet sich von anderen Formen des maschinellen Lernens, da es darauf abzielt, eine optimale Strategie zu entwickeln, um Probleme zu lösen, ohne vorab bereitgestellte Trainingsdaten. Ein Agent folgt vordefinierten Regeln und einem Belohnungssignal, um durch Trial-and-Error eine erfolgreiche Strategie zu erlernen. Der Fokus liegt auf der Gesamttaktik oder Abfolge von Aktionen, die zu einem gewünschten Zustand führen, anstatt auf einzelnen Aktionen. Der Algorithmus bewertet und lernt aus positiven Aktionen, um eine effektive Strategie zu entwickeln. Ein Beispiel hierfür ist ein Brettspiel, bei dem die Reihenfolge der richtigen Züge entscheidend für den Erfolg ist (Mockenhaupt 2021, 141-142), (Alpaydın 2019, 14).

Deep Learning

DL bzw. tiefes Lernen bildet wiederum einen spezifischen Teilbereich des MLs ab, der sich auf den Einsatz künstlicher NN konzentriert. Diese Netze bestehen aus mehreren Schichten von Neuronen, die hintereinandergeschaltet sind, um komplexe Funktionen zu approximieren. Durch das Lernen aus Daten sind NN in der Lage, anspruchsvolle Aufgaben wie Bild- oder Spracherkennung durchzuführen (Bhattacharyya, et al. 2020, 8-9).

Deep Reinforcement Learning

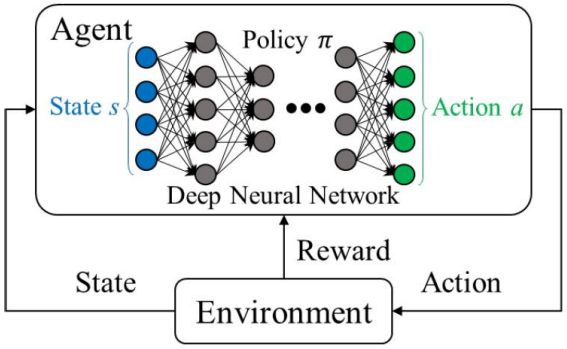

Abbildung 5 DRL (Huang, et al. 2019, 82195)

DRL kombiniert Reinforcement Learning mit neuronalen Netzwerken (NN), um komplexe Entscheidungsprobleme zu bewältigen. DRL-Agenten können Wissen aus Rohdaten, wie visuellen Informationen, ohne vordefinierte Merkmale erlernen. DRL baut auf früheren Fortschritten im Reinforcement Learning auf, indem es NN verwendet, um hochdimensionale Probleme zu bewältigen. CNNs sind Teil dieser DL-Algorithmen und ermöglichen das direkte Lernen aus rohen und hochdimensionalen Daten. Die Spezialisierung des DRL konzentriert sich darauf, optimale Wertfunktionen Q zu finden, um die beste Entscheidung anzunähern (Vijayan PV 2020).

Deep Q-Learning

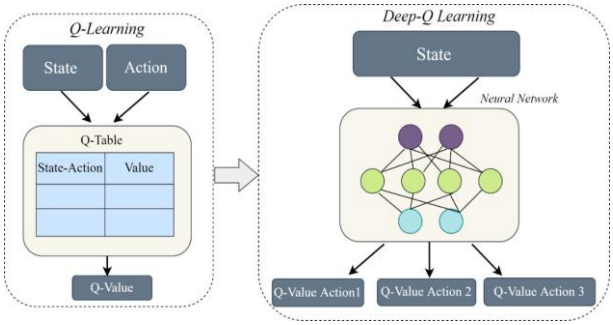

Abbildung 6 Deep Q-Learning (Ansari, et al. 2022, 127473)

Das Q-Learning basiert auf der Q-Funktion, die die erwarteten Belohnungen für Aktionen in einem Zustand berechnet. Die optimale Q-Funktion, Q*, repräsentiert die maximale Belohnung von einem Zustand ausgehend, wenn die beste Aktion gemäß einer Richtlinie π gewählt wird. Mit diesen Erkenntnissen ordnet das Q-Learning jedem Zustands-Aktionspaar einen entsprechenden Q-Wert innerhalb einer Tabelle zu, wobei der höchste Wert in dieser Tabelle die optimale Q-Funktion darstellt, also die maximale Belohnung. Das Deep Q-Learning (DQL) kombiniert diese Idee mit DRL, indem es ein NN verwendet, um Eingabezustände auf Paare (Q-Wert, Aktion) abzubilden und dadurch die bestmögliche Entscheidung mit der höchsten erwarteten Belohnung zu erhalten (TensorFlow o. D.), (Wang 2020).

Long Short-Term Memory

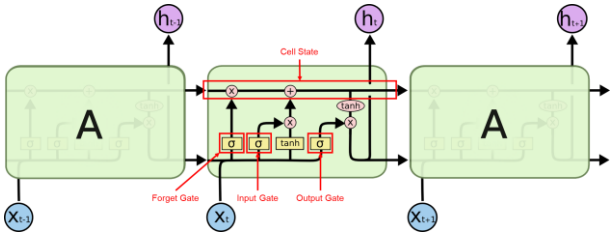

Abbildung 7 LSTM (Olah 2015) (Anpassung, vgl. (Lang 2022))

Recurrent Neural Networks (RNN) sind spezialisierte neuronale Netzwerke zur Verarbeitung von Sequenzen, da sie Informationen aus früheren Schritten speichern und für zukünftige Schritte verwenden können. Dies ist besonders nützlich bei aufeinanderfolgenden Daten, z. B. in der Sprachverarbeitung oder Zeitreihenanalysen.

RNNs haben jedoch das Problem des „Vergessens“, wenn die Trainingsdauer zunimmt und viele Rückkopplungsschleifen in den Hidden-Layern auftreten. Um dieses Problem zu lösen, wurden Long Short-Term Memory (LSTM) entwickelt. LSTMs basieren auf RNNs, verfügen jedoch über Mechanismen, die es ihnen ermöglichen, langfristige Abhängigkeiten besser zu erfassen, indem sie entscheiden, welche Informationen behalten und welche vergessen werden sollen. Dies geschieht durch die Verwendung von Toren wie dem Forget Gate, dem Input Gate und dem Output Gate, um das Langzeitgedächtnis effizient zu aktualisieren und relevante Informationen zu erhalten. Dies ermöglicht LSTMs, langfristige Abhängigkeiten in den Daten effektiver zu berücksichtigen und ist besonders in der Verarbeitung sequenzieller Informationen von Vorteil (Kempkes o. D.), (Luber und Litzel 2018 II), (Lang 2022).

V. BASELINE-HANDELSSYSTEM

Nachdem der theoretische Rahmen mit seinen Konzepten rund um das algorithmische Handeln sowie einem Einblick in die Themen des MLs vermittelt wurde, wird im folgenden Kapitel der Aufbau eines Baseline-Handelssystems präsentiert. Dieses Handelssystem dient als Ausgangspunkt und Basis für die weiteren Entwicklungen, Experimente sowie der Integration von DL-Modellen.

Programmiersprache und Frameworks

Das Handelssystem wird mit Hilfe der Programmiersprache Python aufgebaut, da Python eine weit verbreitete und beliebte Sprache für ML (Gülen 2022) sowie und algorithmisches Handeln (Quantified Strategies 2023) ist. Es bietet eine breite Palette von Bibliotheken und Frameworks, die sich besonders gut für den Aufbau eines Handelssystems eignen.

Einige sinnvolle Frameworks für den Aufbau des Handelssystem mit Python sind (Thakar 2023):

- NumPy: Bibliothek für numerische Berechnungen, dass eine effiziente Handhabung von großen Datenmengen ermöglicht.
- Pandas: Framework für Datenmanipulation und Analyse, dass besonders gut geeignet ist, um Finanzdaten zu verarbeiten.
- TensorFlow & Keras: Framework für ML und der Implementierung von DL-Modellen bzw. dem Aufbau von NN.
- Scikit-learn: Bibliothek für ML mit vorgefertigten Algorithmen wie Regression und Klassifikation für die Modellvalidierung

Architektur und Design

Abbildung 8 Wechselseitige Interaktion (Freepik Company S.L. o. D.), (One Financial Markets o. D.), (Python Software Foundation o. D.)

Die drei Hauptkomponenten der Implementierung bestehend aus dem Finanzmarkt, der Handelsplattform und der Python-Applikation bilden die grundlegenden Bausteine der geplanten Umsetzung des Handelssystems. Der Finanzmarkt stellt die Daten und Informationen bereit, auf deren Grundlage Handelsentscheidungen getroffen werden. Die Python-Applikation übernimmt die Analyse der Daten, die Implementierung von Handelsstrategien und die Ausführung bzw. Signalisierung von Handelsaufträgen. Die Handelsplattform MetaTrader 5 (MT5) fungiert als Schnittstellte zwischen der Python-Applikation und dem Finanzmarkt. Sie ermöglicht die Interaktion mit dem Finanzmarkt, indem sie Positionen platziert und verwaltet sowie notwendige Kontoinformationen abruft. Eine effiziente Zusammenarbeit dieser drei Hauptkomponenten gewährleistet den reibungslosen Betrieb des algorithmischen Handelssystems.

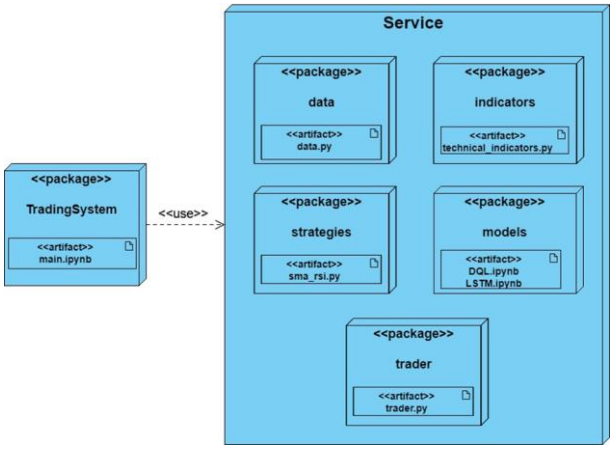

Abbildung 9 Verteilungsdiagramm

Zu Beginn der Implementierung eines Handelssystems müssen geeignete Komponenten festgelegt werden.

Diese Komponenten sind im Verteilungsdiagramm des Baseline-Handelssystems dargestellt (siehe Abbildung 9). Die Abbildung zeigt die geplante Ordnerstruktur bzw. den modularen Aufbau des Systems mit dem Namen „AlgoBot“. Die Pakete in der Abbildung repräsentieren die verschiedenen Module und Funktionen, die im Handelssystem enthalten sein werden.

- data-Modul: Diese Komponente ist verantwortlich für das Herunterladen und Vorverarbeiten von Marktdaten aus verschiedenen Quellen, z. B. Finanzmärkten, APIs oder anderen Datenanbietern.
- indicators-Modul: Zuständig für die Berechnung technischer Indikatoren auf der Basis von Bibliotheken und Frameworks.
- strategies-Modul: Diese Komponente enthält die Logik, um Handelsentscheidungen basierend auf den vorverarbeiteten Daten zu treffen. Dies kann eine Vielzahl von Strategien umfassen, z. B. einen Indikator-Crossover-Ansatz. Ein Indikator-Crossover-Ansatz bezieht sich darauf, Handelsentscheidungen auf Basis von Überkreuzungen (Crossovers) von technischen Indikatoren zu treffen. Hierbei werden typischerweise zwei oder mehrere Indikatoren verwendet. Es wird ein Handelssignal generiert, wenn diese beiden Indikatoren sich über- oder untereinander kreuzen.
- models-Modul: Diese Komponente beinhaltet die implementierten DL-Ansätze für den Handel.
- trader-Modul: Diese Komponente gewährleistet die Verbindung zu den Finanzmärkten und der damit einhergehenden Hauptfunktionalität für das Öffnen und Schließen von Handelspositionen. Ihre Aufgabe besteht darin, die Handelsstrategien und -entscheidungen an die MT5-Plattform zu senden
- TradingSystem-Modul: Dieses Modul fungiert als der zentrale Einstiegspunkt oder das Hauptprogramm des Handelssystems. In den unterschiedlichen Funktionen werden verschiedene Komponenten des Handelssystems aufgerufen und miteinander verknüpft, um den Handelsprozess zu steuern als auch zu koordinieren.

Funktionalitäten

```
class Data:
    def __init__(self):...

    def fetch_data(self, symbol, timeframe, start_pos, num_bars):...

    def download(self, symbol):...
```

Abbildung 10 Data-Modul

Durch das Data-Modul wird dem Handelssystem ermöglicht, historische Marktdaten im OHLC-Format (siehe Abschnitt Candlesticks) zu sammeln, zu verarbeiten und für die technische Analyse zu nutzen. Es ermöglicht dem Handelssystem, Kursdaten von verschiedenen Finanzmärkten über die Handelsplattform abzurufen, wie z.B. Forex-Währungspaare oder Aktienkurse und diese Daten für die Generierung von Handelssignalen und die Ausführung von Handelsstrategien zu verwenden. Die „fetch_data“-Methode empfängt dabei ein gewünschtes Symbol über einen ausgewählten Zeitraum aus der Handelsplattform MT5. Dem Nutzer steht es dabei frei, den Zeitstempel der Daten zu wählen und die Anzahl der zurückgegebenen Candlesticks anzugeben. Wohingegen die „download“-Methode für das Laden der Daten aus der YahooFinance-API zuständig ist.

Date	Open	High	Low	Close	Adj Close	Volume
2003-12-01	189.330002	189.660004	187.740005	187.630005	187.630005	0
2003-12-02	187.669998	188.809998	187.559998	188.009995	188.009995	0
2003-12-03	188.020004	188.240005	186.490005	187.089996	187.089996	0
2003-12-04	187.029999	187.029999	185.899994	186.220001	186.220001	0
2003-12-05	186.190002	186.740005	185.830002	185.880005	185.880005	0
...	...	...	...	...	...	...
2023-09-18	183.307999	183.332001	182.740005	183.307999	183.307999	0
2023-09-19	182.770004	183.468994	182.731995	182.770004	182.770004	0
2023-09-20	183.089005	183.341003	182.492996	183.089005	183.089005	0
2023-09-21	182.834000	182.919006	181.087997	182.834000	182.834000	0
2023-09-22	181.389008	182.304001	181.199005	181.628998	181.628998	0

5157 rows × 6 columns

Abbildung 11 Datenausgabe

Das Trader-Modul enthält neben den Klassenmethoden auch weitere Methoden für den operativen Einsatz. Bei der Erstellung eines Objekts der Klasse Trader wird eine Accountbezeichnung und ein Passwort angegeben, die zur Initialisierung des Kontos und zur Anmeldung bei der Handelsplattform erforderlich sind.

```
class Trader:
    def __init__(self, account, password):...

    def __del__(self):...

    def buy(self, symbol, quantity):...

    def sell(self, symbol, quantity):...

    def close(self, symbol, quantity):...
```

Abbildung 12 Trader-Modul

Die Methoden „buy“ und „sell“ ermöglichen den Kauf und Verkauf von Handelsobjekten mit einer angegebenen Menge zu einem aktuellen Preis auf dem Markt. Um eine offene Position zu schließen, wird die Methode „close“ verwendet.

```
class TechnicalIndicators:
    def __init__(self):...

    def get_SMA(self, data):...

    def get_SMA100(self, data):...

    def get_RSI(self, data):...

    def indicators_stack(self, data):...
```

Abbildung 13 Indicators-Modul

Das Indicators-Modul stellt dem Handelssystem gängige technische Indikatoren für die Analyse der Finanzmärkte auf Basis geladener Kursdaten zur Verfügung. Die Klasse „TechnicalIndicators“ nutzt das Bibliothekspaket „ta-lib“ (Technical Analysis Library) zur Berechnung

und Generierung verschiedener technischer Indikatoren. Diese Indikatoren können zur Identifizierung von Trends, Volatilität, Momentum und anderen Aspekten der Marktdynamik verwendet werden.

```
def sma_rsi(df):
    close = df["close"]
    rsi = talib.RSI(close, 20)
    sma1 = talib.SMA(close, 20)
    sma2 = talib.SMA(close, 50)

    signal = ""

    if crossover(sma2.iloc[-1], sma1.iloc[-1]) or (rsi.iloc[-1] > 70):
        signal = "sell"
    elif crossover(sma1.iloc[-1], sma2.iloc[-1]) or (rsi.iloc[-1] < 30):
        signal = "buy"

    return signal
```

Abbildung 14 Strategies-Modul: SMA-RSI-Crossover

Das Strategies-Modul ist für die zur Verfügungstellung unterschiedlicher Handelsstrategien verantwortlich. Es beinhaltet verschiedene Funktionen, die jeweils eine spezifische Handelsstrategie implementieren. Diese Strategien können je Anforderung auf die Marktsituation ausgewählt und angewendet werden. Die implementierte Handelsstrategie innerhalb des Moduls ist die SMA-RSI-Crossover-Strategie.

Der modulare Aufbau des Baseline-Handelssystems ermöglicht durch die Trennung der einzelnen Funktionalitäten eine bessere Gesamtübersicht. Innerhalb der „main.ipynb“-Datei im TradingSystem-Modul wird das gesamte Handelssystem orchestriert und ausgeführt. Hier werden alle Komponenten des Systems zusammengeführt, einschließlich des Traders, der technischen Indikatoren und Datenverarbeitung. Die „main.ipynb“-Datei steuert den Ablauf des Handelssystems und ruft die entsprechenden Funktionen auf, um Daten abzurufen, Handelssignale zu generieren, Handelspositionen zu platzieren sowie Informationen über das Handelskonto zu erhalten. Außerdem können in dieser Datei auch die verschiedenen Strategien mit den einhergehenden Parametern des Handelssystems konfiguriert werden.

Das folgende Sequenzdiagramm veranschaulicht die Zusammenarbeit der Komponenten unter Berücksichtigung der unterschiedlichen Module im Rahmen des Baseline-Handelssystems während der Initialisierung einer Handelsposition mit der SMA-RSI-Crossover-Strategie.

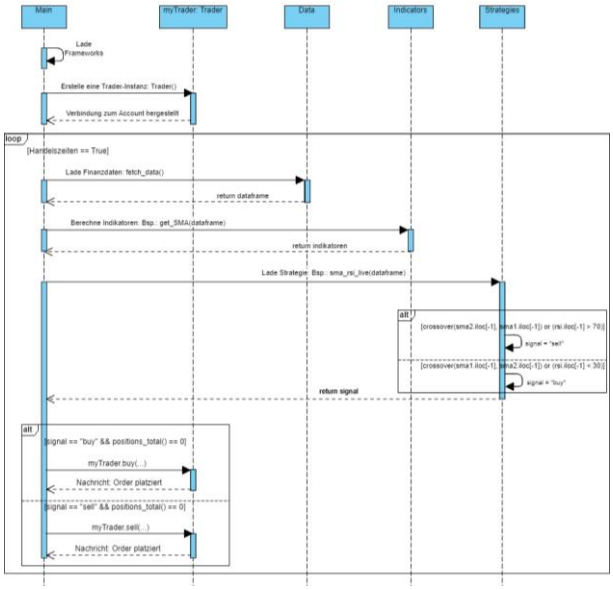

Abbildung 15 Handelssystem-Sequenzdiagramm

VI. DL-HANDELSAGENTEN

In diesem Kapitel werden verschiedene Ansätze der Implementierung von DL-Modellen für den Handel beleuchtet, um diese als Handelsagenten in das bestehende algorithmische Handelssystem zu integrieren.

Feature Engineering

Um eine konsistente und markttechnische Grundlage für die Analyse zu schaffen, ist es von entscheidender Bedeutung einen allgemeinen Datensatz zu erstellen, der für die Anwendung verschiedener Ansätze geeignet ist. Dadurch ist es möglich, unterschiedliche Modelle und Algorithmen auf derselben Datengrundlage zu trainieren. Die Konstruktion eines solchen universellen Datensatzes trägt i. d. R. dazu bei, die Vergleich- und Wiederholbarkeit der Modelle zwischen unterschiedlichen Algorithmen bzw. Handelsansätzen sicherzustellen.

Dafür werden in der Anwendung jedes Handelsansatzes folgende Schritte vorausgesetzt:

- Daten laden,
- Datensatz mit Indikatoren anreichern und
- Daten ggf. normalisieren (MinMaxScalers() → Werte zw. 0 und 1 erzeugen)

LSTM Handelsansatz

Die Entscheidung zur Verwendung eines LSTM-Ansatzes basiert auf der bereits dargelegten Tatsache einer effektiven Leistungsfähigkeit im Umgang mit Zeitreihendaten. Ähnlich zu anderen Ansätzen, wie sie in den Arbeiten von Huang und Singh dargelegt wurden, liegt der Fokus dieses Ansatzes darauf den Schlusspreis zu bestimmen.

```
df = Data().download(symbol="GBPJPY=X")
df = TechnicalIndicators().indicators_stack(df)
df = df.drop(["Volume"], axis=1)
df.dropna(inplace=True)
df
```

Abbildung 16 LSTM: Daten laden

Zu Beginn der Implementierung eines LSTM-Modells muss die zu vorhersagende Zielvariable definiert werden. In diesem Ansatz setzt sich die Zielvariable aus dem Schlusspreis bzw. Wert einer Kerze des nächsten Tages zusammen. Für das DataFrame wird eine neue Spalte mit der Bezeichnung „target“ erzeugt. Das Setzen des Schlusspreises des nächsten Tages wird durch eine Iteration über alle verfügbaren Handelstage abgerufen und in der Spalte gespeichert. Im Anschluss werden ergebnisverzerrende Spalten entfernt und der Datensatz normalisiert.

Die Grundlage für das erfolgreiche Vorhersagen des LSTM-Modells bildet das festgesetzte bzw. vergangene Zeitfenster, dass das Modell für die Berechnungen der Zielvariable miteinbezieht. Die Auswahl des Zeitfensters innerhalb der Funktion „transformLookBack()“ wird mit der folgenden Abbildung näher erläutert:

```
def transformLookBack(df, look_back):
    # Anzahl der Merkmale (Spalten) im DataFrame
    num_features = df.shape[1] - 1

    # Zweidimensionale Liste von Werten, die aus Zeitschritten der Ver-
    # gangenheit bestehen. Die Größe der Liste richtet sich dem über-
    # gebenen Zeitfenster und der Länge des Datensatzes.
    dataX = [
        [df[i - look_back:i, j] for i in range(look_back, df.shape[0])]
        for j in range(num_features)
    ]

    # Transponieren der dataX-Matrix, um die gewünschte Form zu erhalten.
    dataX = np.moveaxis(dataX, 0, 2)

    # Extrahieren der Zielvariable aus der letzten Spalte des DataFrames.
    dataY = df[look_back:, -1].reshape(-1, 1)

    return dataX, dataY

# Fenstergröße
look_back = 25

X, y = transformLookBack(df, look_back)
```

Abbildung 17 LSTM: Lookback

Die Funktion erwartet neben einem DataFrame einen Parameter „look_back“, der die Anzahl der vergangenen Zeitschritte festlegt, die zur Vorhersage eines zukünftigen Werts verwendet werden. Dies entspricht dem blauen Fenster innerhalb der Abbildung 18.

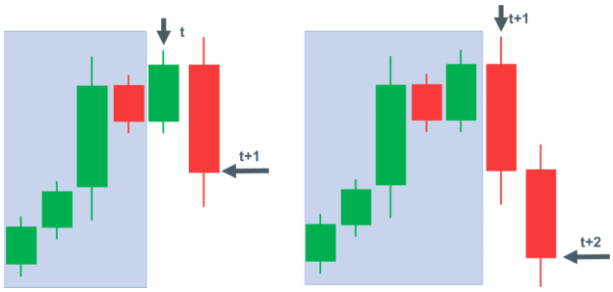

Abbildung 18 LSTM-Lookback-Logik

Anschließend erstellt die Funktion eine Liste von Eingangsdaten „dataX“, indem sie durch die Werte des DataFrames iteriert und ab dem Zeitpunkt t bis t+n-1 Sequenzen von vergangenen Werten jeder Kerze innerhalb des Fensters erstellt werden. Die Zielvariable bzw. der Schlusswert „dataY“ für den Zeitpunkt t+1 wird demnach in Abhängigkeit von der zuletzt betrachteten Kerze zum Zeitpunkt t festgelegt. Dieser Wert wird aus der letzten Spalte des DataFrames zum Zeitpunkt t+1 extrahiert, um die zugehörigen Zielwerte für den betrachteten Zeitraum zu liefern. In der darauffolgenden Iteration wird demnach versucht vom Zeitpunkt t+1 den Schlusswert der nächsten Kerze t+2 zu berechnen.

Nach dem Aufteilen in Trainings- und Testdaten erfolgt im nächsten Schritt der Aufbau sowie das Trainieren des LSTM-Netzwerks, welches wie folgt konstruiert wird:

Layer	Typ	Input Dim.	Output Dim.	Aktivierungsfunktion
1	LSTM	20	150	tanh
2	Dropout	150	150	-
3	LSTM	150	300	tanh
4	Dropout	300	300	-
5	Dense	300	1	linear

Tabelle 1 Aufbau des NN-Layer

Nach dem Training kann auf Basis der Testdaten folgender Graph erzeugt werden.

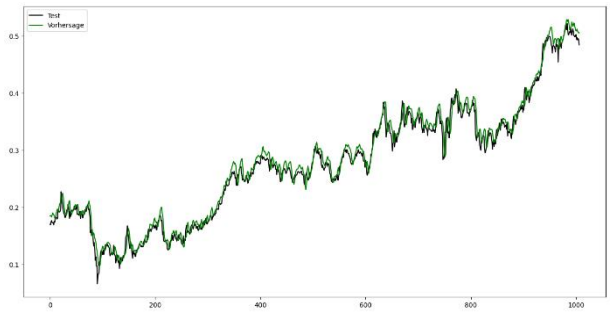

Abbildung 19 LSTM-Vorhersagenplot

Die schwarze Linie bildet den tatsächlichen Werteverlauf (Validierungsdaten) des Graphen ab, während der grüne Graph auf der Grundlage des LSTM-Modells berechnet wurde. Hierbei wird erkenntlich, dass das Modell durchaus in der Lage ist, Zeitreihendaten möglichst nahe an den tatsächlichen Werten zu berechnen. An bestimmten

Momenten erfolgt ein identischer Verlauf, wobei wiederum an anderen Stellen eine Abweichung bzw. eine versetzte Darstellung zu einem späteren Zeitpunkt der tatschlichen Werte auffällt.

Mit diesen Ergebnissen können unter Einhaltung bestimmter Regeln und durch die Erweiterung des trainierten Modells wichtige Einblicke in den automatisierten Handel gewonnen werden. Dadurch können Trader informiert handeln, sobald bspw. erwartet wird, dass der Schlusswert des nächsten Tages gegenüber dem Vortag sinkt.

Deep RL Handelsansatz

In diesem Abschnitt werden die Konzepte angelehnt an die Implementierung eines DRL-Ansatzes für das Spielen von Atari (Mnih, et al. 2013) mit Blick auf die Handelsebene vertieft, indem die Implementierung von DRL mithilfe NN betrachtet wird.

Zum Start der Implementierung werden die benötigten Bibliotheken in das Programm aufgenommen. Neben den bereits bekannten Frameworks wird das OpenAI-Gym-Framework Anytrading (Haghpanah o. D.) sowie für den Einsatz von RL-Techniken Stable Baselines3 (SB3) (Raffin o. D.) geladen.

Anschließend erfolgt das Deklarieren eines DataFrames mit dem erforderlichen Datensatz aus der YahooFinance-Umgebung und dem Anreichern zusätzlicher Features bzw. Spalten aus dem Indicators-Modul

Wie aus dem klassischen RL bekannt, wird eine Umgebung bzw. Environment für das Trainieren eines RL-Modells benötigt. Da in diesem Ansatz Forex-Währungspaare gehandelt werden sollen, bezieht sich die Variable an dieser Stelle auf die „ForexEnv"-Umgebung der Anytrading-Bibliothek. Das Environment bezieht sich auf die Umgebung oder das System, in dem ein RL-Agent agiert und lernt sowie den damit einhergehenden Spielregeln für das Entwickeln der bestmöglichen Aktionen.

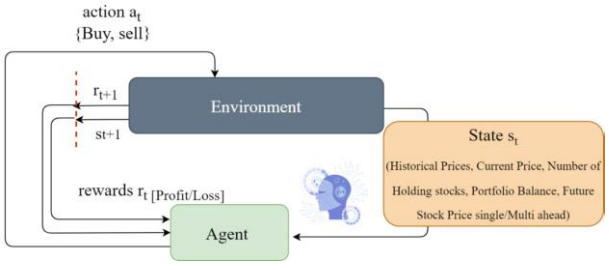

Abbildung 20 RL im Handel (Anpassung, vgl. (Ansari, et al. 2022, 127471))

In diesem Ansatz wird lediglich das Kaufen und Verkaufen von Währungspaaren anvisiert, weswegen der Agent dem Environment „env" zwei durchführbare Aktion $a_i = [0, 1]$ übermittelt. Dabei steht 0 für das Verkaufen und 1 für das Kaufen. Nach jeder durchlaufenen Iteration registriert das Environment die übermittelte Aktion und berechnet daraufhin die Belohnung bzw. den Reward (Profit/Verlust) als auch den nächsten Schritt in Iteration t+1. Um optimale Entscheidungen auf der Basis von Aktionen a_i in einem Zustand s_i zu ermöglichen, wird mithilfe von SB3 ein Deep Q-Network (DQN)-Modell aufgespannt bzw. der Agent initialisiert. Dafür wird eine Instanz durch die Variable „env_maker" erstellt und anschließend in eine vektorisierte Umgebung „DummyVecEnc" gewickelt. Diese ist erforderlich, um RL-Modelle in der SB3-Bibliothek zu verwenden. Die „DummyVecEnv" ermöglicht es, die Umgebung parallel zu betreiben, was für das Training von RL-Modellen von Vorteil ist. Eine Möglichkeit den Agenten zu initialisieren, ist der Aufbau eines bekannten NN aus Dense-Layern sowie Aktivierungsfunktionen.

```
# Erstelle eine Dummy-Umgebung, um das wiederholende Training
# innerhalb des Netzwerks zu ermöglichen
env_maker = lambda: ForexEnv(df=df, frame_bound=(1, len(df)),
window_size=1)
env = DummyVecEnv([env_maker])

# Initialisierung
model = DQN("MlpPolicy", env, verbose=1)

# Training des DQN-Modells
model.learn(total_timesteps=1000000)
```

Abbildung 21 Initiierung einer DQN-Environment

Als Richtlinie π verwendet das Modell die „MlpPolicy", die auf einem Multi-Layer Perceptron (MLP) basiert. Die Umgebung, in der das Modell trainiert wird, wird durch die Variable „env" repräsentiert, die zuvor erstellt wurde. Mit diesem Konstrukt und dem Befolgen der optimalen Richtline wird die bestmögliche Entscheidung aus den Tupeln [Q-Wert, Aktion] mit der höchsten erwarteten Belohnung getroffen.

Mit dem Ausführen der Lernfunktion werden 1 Mio. Lernschritte innerhalb der erzeugten Environments durchgeführt und folgendes Ergebnis ersichtlich:

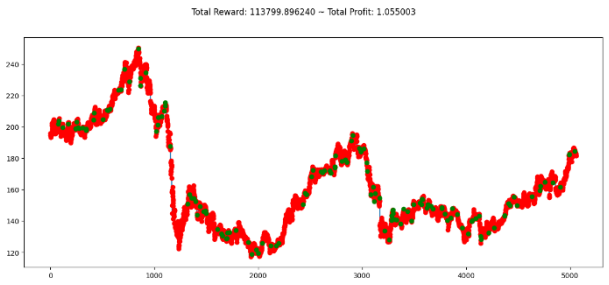

Abbildung 22 Ergebnis DQL

Das DQL-Modell hat über einen bestimmten Zeitraum einen positiven Reward-Wert und eine Erhöhung des Portfolios erzielen können. Das „Total Profit" von 1,055003 bzw. 105,50% vermittelt einen Gewinn von ca. 5,5% des zu Beginn bestehenden Gesamtkapitals.

VII. ANWENDUNG UND ERGEBNISSE

In diesem Abschnitt werden die Implementierungs- bzw. Anwendungsmöglichkeiten auf den realen Markt untersucht. Hierzu werden die zuvor präsentierten Modelle mitsamt ihrer spezifischen Konfiguration in eigenständigen Anwendungsszenarien integriert. Für den praktischen Einsatz des LSTM- und DQN-Modells auf dem echten Markt ist es notwendig, spezielle Hilfsfunktionen zu entwickeln, die die aktuellen Marktdaten verarbeiten. Hierfür wird die Grundstruktur des Modells verwendet und in eine neue, verschachtelte Funktion eingebettet.

Für beide Ansätze wird eine Funktion „get_prediction(model)“ in der „main.ipynb“ des TradingSystem-Moduls implementiert, die je nach übergebenen Modell einen prognostizierten Schlusswert oder ein Kauf- oder Verkaufssignal ausgibt. Innerhalb dieser Funktion werden aktuelle Daten geladen, verarbeitet und normalisiert. Schließlich erfolgt das Übergeben der „Live-Daten“ an das trainierte Modell, welches im Anschluss eine Prognose „prediction“ zur Verfügung stellt. Anhand dieser Prognose wird in den vordefinierten Handelszeiten eine Handelsentscheidung bzw. Handelsposition platziert.

LSTM

```
if prediction[0] > price:
    if mt5.positions_total() == 0:
        print(prediction[0])
        print(mt5.positions_total())
        trader.buy(symbol,0.5)
        print("Handelsposition platziert")

else:
    if mt5.positions_total() == 0:
        print(prediction[0])
        print(mt5.positions_total())
        trader.sell(symbol, 0.5)
        print("Handelsposition platziert")
        time.sleep(5)
```

Abbildung 24 LSTM-Handelsbedingung

Im ersten Schritt wird überprüft, ob die Vorhersage des LSTM-Modells „prediction“ über dem aktuellen Angebotspreis liegt. Wenn dies der Fall ist und keine offenen Positionen vorhanden sind (überprüft mit „mt5.positions_total() == 0“), wird über das Objekt der Klasse Trader ein Kaufauftrag für 0,5 Einheiten des Währungspaares „GBPJPY“ platziert. Wenn die Vorhersage niedriger ausfällt als der aktuelle Angebotspreis und keine offenen Positionen bestehen, erfolgt die Platzierung eines Verkaufsauftrags für 0,5 Einheiten von „GBPJPY“ mithilfe der Funktion „trader.sell(symbol, 0.5)“.

In diesem Fall wurde ein Anstieg des Kurses prognostiziert, weshalb eine Kaufposition eröffnet wurde. Diese Position erwies sich über mehrere Stunden hinweg als profitabel (siehe Abbildung 23).

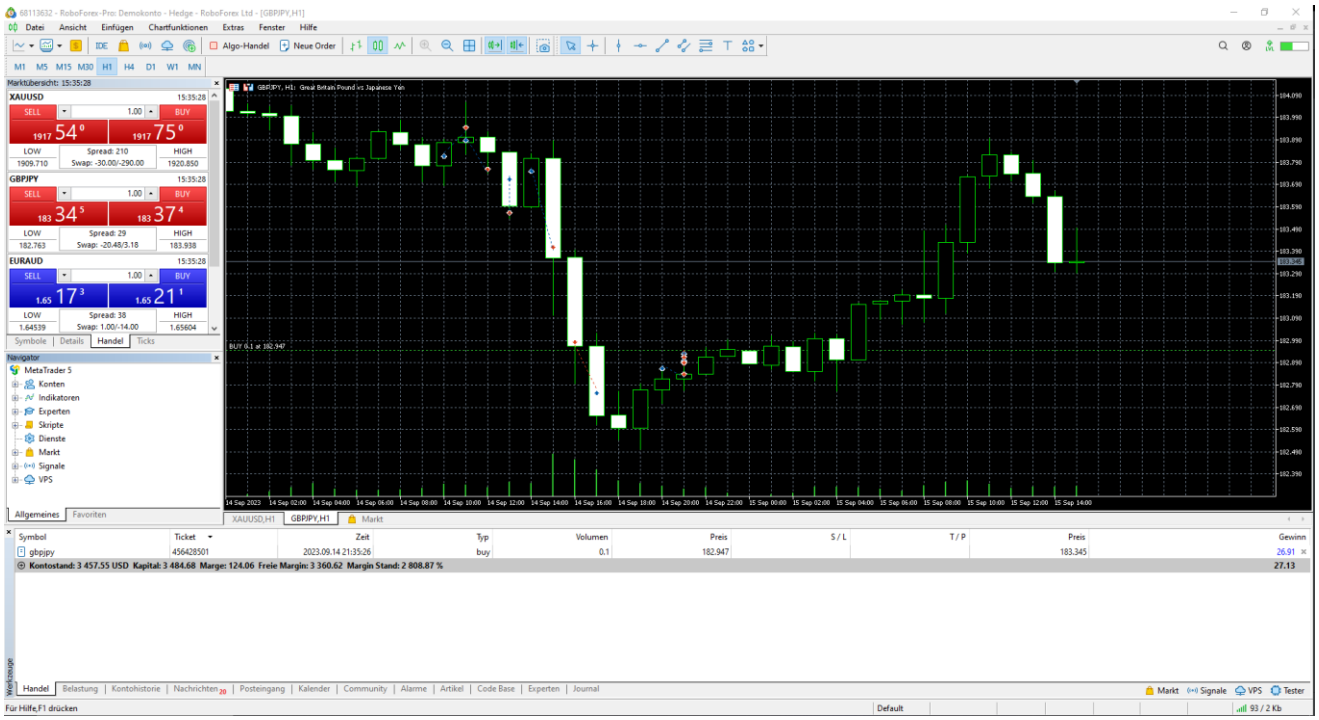

Abbildung 23 LSTM-Einsatz im MT5

DQL

```
if prediction == 1:
    if mt5.positions_total() == 0:
        print(prediction)
        print(mt5.positions_total())
        trader.buy(symbol,0.5)
        print("Handelsposition platziert")

else:
    if mt5.positions_total() == 0:
        print(prediction)
        print(mt5.positions_total())
        trader.sell(symbol, 0.5)
        print("Handelsposition platziert")
        time.sleep(5)
```

Abbildung 25 DQL-Handelsbedingung

Zur Vorhersage eines Kauf- (1) oder Verkaufssignals (0) muss ein korrekter Datensatz das zuvor trainierte Modell übergeben werden.

Abbildung 26 DQL-Einsatz im MT5

Im Beispiel in Abbildung 26 ist ersichtlich, dass die beiden eröffneten Verkaufspositionen in den ersten Stunden mit einem Verlust konfrontiert wurden, während sich der Markt in einer bullischen Bewegung befand, bevor sie schließlich der vorhergesagten Variable „prediction“ mit dem Wert 0 gefolgt sind. Dies führte zu einem Gewinn von über 150 € auf dem Demokonto.

VIII. FAZIT UND AUSBLICK

Im Rahmen dieser Arbeit wurde unter Zuhilfenahme von gängigen ML-Bibliotheken wie Keras, Tensorflow, Gym und Stable Baselines3 eine Python-Applikation zum Treffen automatisierter Handelsentscheidungen konzipiert und entwickelt. Hierbei wurde ein zuvor implementiertes Baseline-Handelssystem, das Handelsentscheidungen auf der Grundlage technischer Indikatoren getroffen hat, mit DL-Methoden erweitert und trainiert. Dabei wurden verschiedene Architekturen, darunter DRL und LSTM-Ansätze zur Vorhersage von Preisen sowie einer Handelsrichtung (Trend) im Devisenmarkt (Forex) eingesetzt. Während sich der DRL-Ansatz unter Zuhilfenahme des DQN-Konzepts aus bekannten Anwendungen wie selbstlernenden Agenten zum Spielen von Atari durch das Abschätzen von Aktionen und Belohnungen entwickelte, konzentriert sich der LSTM-Ansatz auf die Vorhersage zukünftiger Marktentwicklungen basierend auf Zeitreihendaten.

Die entwickelte Python-Applikation bietet nicht nur die Möglichkeit, historische Daten zu analysieren und Handelsentscheidungen zu treffen, sondern ermöglicht auch den Handel in Echtzeit auf verschiedenen Märkten durch die Integration der Handelsplattform MetaTrader 5. Dabei werden die DL-Modelle kontinuierlich aktualisiert, um die jüngsten Marktentwicklungen zu berücksichtigen sowie die Genauigkeit der berechneten Vorhersagen zu optimieren.

In jedem Fall zeigt die Entwicklung im Bereich des algorithmischen Handels, dass fortgeschrittene Technologien und automatisierte Datenanalysen eine immer wichtigere Rolle in der Handelsbranche spielen und den Weg für präzisere, effizientere und profitablere Handelsstrategien ebnen. Ein wichtiger Aspekt ist die Notwendigkeit einer kontinuierlichen Weiterentwicklung und Anpassung von Handelsstrategien. Hierbei bildet der Einsatz von DL eine weitreichende Unterstützung der bereits bestehenden Systeme, weshalb diese weder ignoriert noch ersetzt werden dürfen. Die Finanzmärkte unterliegen ständigen Veränderungen, sei es durch geopolitische Ereignisse, wirtschaftliche Entwicklungen oder technologische Fortschritte. Daher ist es von entscheidender Bedeutung, dass Handelssysteme flexibel und anpassungsfähig sind, um auf neue Gegebenheiten zu reagieren.

Neben den erweiterten Ansätzen des DL innerhalb algorithmischer Handelssysteme gibt es das sogenannte Quantitative Trading (Quant Trading). Im Gegensatz zum konventionellen Algorithmic Trading, das auf festgelegten Algorithmen basiert, zeichnet sich Quantitative Trading durch einen erweiterten Ansatz aus. Hier liegt der Fokus auf der Anwendung fortgeschrittener statistischer und mathematischer Modelle zur Entwicklung von Handelsstrategien. Dabei wird gezielt auf den Einsatz von Wahrscheinlichkeitstheorie und Zeitreihenanalyse gesetzt. Dies ermöglicht eine präzisere und leistungsstärkere Herangehensweise an den Finanzmärkten (Sharma, Anderson und Schmitt 2021).

Die Möglichkeit menschlichen Eingreifens sollte während der Einführung algorithmischer Systeme beibehalten werden, da die Verwendung von DL-basierten Vorhersagen weiterhin als Empfehlung und Unterstützung angesehen werden muss, anstatt sie als Ersatz für menschliche Entscheidungen zu verwenden. Mit den entwickelten und angewandten Ansätze kann festgehalten werden, dass die Integration von ML und DL in den algorithmischen Handel ein vielversprechendes und mächtiges Werkzeug darstellt. Mit dieser Integration ist ein entscheidender Schritt in Richtung einer effizienteren, präziseren und automatisierten Handelspraxis auf den Finanzmärkten untersucht worden.

LITERATUR

Alpaydın, Ethem . 2004. *Introduction to Machine Learning.* Cambridge, Massachusetts: The MIT Press.

Alpaydın, Ethem. 2019. *Maschinelles Lernen.* Berlin, Boston: De Gruyter Oldenbourg.

Ansari, Yasmeen, Sadaf Yasmin, Sheneela Naz, Hira Zaffar, Zeeshan Ali, Jihoon Moon, and Seungmin Rho. 2022. "A Deep Reinforcement Learning-Based Decision Support System for Automated Stock Market Trading." *IEEE Access.*

Bhatt, Shweta. 2018. *Reinforcement Learning 101.* 19 März. Accessed Juli 26, 2023. https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292.

Bhattacharyya, Siddhartha, Vaclav Snasel, Aboul Ella Hassanien, Satadal Saha, and B. K. Tripathy. 2020. *Deep Learning: Research and Applications.* Berlin: De Gruyter.

Chopra, Yashica. 2022. *Machine Learning for Trading – Can It Predict the Trend?* 16 Juni. Accessed Mai 02, 2023. https://www.datatobiz.com/blog/machine-learning-for-trading/.

Economy-Pedia. o. D. *Arten von Handelssystemen.* Accessed Juli 21, 2023. https://de.economy-pedia.com/11029980-types-of-trading-systems.

Freepik Company S.L. o. D. *Stock Icons.* Accessed August 14, 2023. https://www.flaticon.com/free-icon/stock-market_2910311?term=stock&page=1&position=85&origin=tag&related_id=2910311.

Gülen, Kerem. 2022. *How to choose a programming language for your machine learning project?* 17 November. Accessed Juli 31, 2023. https://dataconomy.com/2022/11/17/best-language-for-machine-learning/.

Haghpanah, Mohammad Amin. o. D. *gym-anytrading.* Accessed September 01, 2023. https://github.com/AminHP/gym-anytrading.

Hofmann, Jonathan. 2014. *Markttechnische Handelssysteme für ausgewählte Wechselkurse.* Wiesbaden: Springer Gabler Wiesbaden.

Huang, Peisen, Guanhai Huang, and Hongyu Chen. 2022. "Comprehensive scoring trading model based on LSTM prediction." *International Conference on Electronics and Devices, Computational Science (ICEDCS).* Marseille, Frankreich. 303-307.

Huang, Xuefei, Seung Ho Hong, Mengmeng Yu, Yuemin Ding, and Junhui Jiang. 2019. "Demand Response Management for Industrial Facilities: A Deep Reinforcement Learning Approach." *IEEE Access.*

Kempkes, Christoph. o. D. *Neuronale Netze und der MNIST Datensatz.* 14 April. Accessed Juli 29, 2023. https://medium.com/@wahlschwabe/neuronale-netze-und-der-mnist-datensatz-4cf57bc4b22.

Kuepper, Justin, Chip Stapleton, and Kirsten Rohrs Schmitt. 2021. *Channeling: Charting a Path to Success.* 21 Oktober. Accessed Juli 21, 2023. https://www.investopedia.com/trading/channeling-charting-path-to-success/.

Lang, Niklas. 2022. *Long Short-Term Memory Networks (LSTM) – einfach erklärt!* 4 Juni. Accessed Juli 29, 2023. https://databasecamp.de/ki/lstm.

Luber, Stefan, and Nico Litzel. 2018 II. *Was ist ein Long Short-Term Memory?* 12 November. Accessed Juli 29, 2023. https://www.bigdata-insider.de/was-ist-ein-long-short-term-memory-a-774848/.

Magwa, Lusanele. 2023. *https://www.robeco.com/de-de/einblicke/2023/03/machine-learning-modelle-koennen-interessante-zusammenhaenge-identifizieren.* 01 März. Accessed Mai 02, 2023. https://www.robeco.com/de-de/einblicke/2023/03/machine-learning-modelle-koennen-interessante-zusammenhaenge-identifizieren.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. *Playing Atari with Deep Reinforcement Learning.* DeepMind Technologies: arXiv.

Mockenhaupt, Andreas. 2021. *Digitalisierung und Künstliche Intelligenz in der Produktion.* Wiesbaden: Springer Vieweg .

Mordor Intelligence. o. D. *ALGORITHMIC TRADING MARKET SIZE & SHARE ANALYSIS - GROWTH TRENDS & FORECASTS (2023 - 2028).* Accessed Juli 03, 2023. https://www.mordorintelligence.com/industry-reports/algorithmic-trading-market.

Olah, Christopher. 2015. *Understanding LSTM Networks.* 27 August. Accessed Juli 29, 2023. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

One Financial Markets. o. D. *ONE | MT5.* Accessed August 14, 2023. http://www.vimarkets.me/one-mt5.

Python Software Foundation. o. D. *The Python Logo.* Accessed August 14, 2023. https://www.python.org/community/logos/.

Quantified Strategies. 2023. *Best Programming Language for Algorithmic Trading Strategies and Systems?* 23 Juni. Accessed Juli 31, 2023. https://www.quantifiedstrategies.com/best-programming-language-for-algorithmic-trading-strategies/.

—. 2023. *Trading Strategies with AI and Machine learning.* 13 Juni. Accessed Juli 24, 2023. https://www.quantifiedstrategies.com/trading-strategies-with-ai-and-machine-learning/.

Raffin, Antonin. o. D. *Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations.*

Accessed September 01, 2023. https://stable-baselines3.readthedocs.io/en/master/.

Redgate. 2021. *Volumen der jährlich generierten/replizierten digitalen Datenmenge weltweit in den Jahren 2012 und 2020 und Prognose für 2025 (in Zettabyte).* 08 September. Accessed Juni 05, 2023. https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/.

Shah, Raj. 2019. *Global Algorithmic Trading Market to Surpass US$ 21,685.53 Million by 2026.* 05 Februar. Accessed Juni 03, 2023. https://www.businesswire.com/news/home/20190205005634/en/Global-Algorithmic-Trading-Market-to-Surpass-US-21685.53-Million-by-2026.

Sharma, Rakesh, Somer Anderson, and Kirsten Rohrs Schmitt. 2021. *What Is Quantitative Trading? Definition, Examples, and Profit.* 13 Mai. Accessed September 18, 2023. https://www.investopedia.com/terms/q/quantitative-trading.asp.

Singh, Japjeet, Ruppa Thulasiram, and Aerambamoorthy Thavaneswaran. 2022. "LSTM based Algorithmic Trading model for Bitcoin." *IEEE Symposium Series on Computational Intelligence (SSCI).* Singapur, Singapur. 344-351.

TensorFlow. o. D. *Introduction to RL and Deep Q Networks.* Accessed Juli 30, 2023. https://www.tensorflow.org/agents/tutorials/0_intro_rl.

Thakar, Chainika. 2023. *Popular Python Libraries for Algorithmic Trading.* 08 Februar. Accessed Juli 31, 2023. https://blog.quantinsti.com/python-trading-library/.

Udagawa, Yoshihisa. 2018. "Predicting Stock Price Trend Using Candlestick Chart Blending Technique." *2018 IEEE International Conference on Big Data (Big Data).* doi:10.1109/BigData.2018.8622402.

Vijayan PV, Vishnu. 2020. *Deep Reinforcement Learning: Artificial Intelligence, Machine Learning and Deep Learning — Introduction, Overview and Contrast for Beginners.* 03 August. Accessed Juli 30, 2023. https://medium.com/@vishnuvijayanpv/deep-reinforcement-learning-artificial-intelligence-machine-learning-and-deep-learning-e52cb5974420.

Wang, Mike. 2020. *Deep Q-Learning Tutorial: minDQN.* 18 November. Accessed Juli 30, 2023. https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.

Wuttke, Laurenz. o. D. *Machine Learning: Definition, Algorithmen, Methoden und Beispiele.* Accessed Juli 26, 2023. https://datasolut.com/was-ist-machine-learning/.