Hochschule Neu-Ulm
University of Applied Sciences

Technische Hochschule Ulm

Bachelorarbeit
in Wirtschaftsinformatik
an der Hochschule für angewandte Wissenschaften Neu-Ulm und Technische Hochschule Ulm

Verbessung der Prognosegenauigkeit von Kursentwicklungen bei Finanzinstrumente durch die Integration von Multi-Temporal-Daten und technischen Indikatoren auf Basis von Recurrent Neural Networks (RNNs)

Erstkorrektor: Prof. Dr. Grieble

Zweitkorrektor:

Verfasser/-in: Valentino Ciavarrella (3140736)

Thema erhalten: 05.03.2025
Arbeit abgegeben: 04.07.2025

Abstract

Diese Arbeit untersucht, ob multitemporale technische Indikatoren die Prognosegenauigkeit eines LSTM-RNN bei der Vorhersage von Schlusskursen verbessern. Ziel ist es, den Nutzen zusätzlicher zeitlich aggregierter Informationen im Vergleich zu rein tagesbasierten Daten zu bewerten. Der Untersuchungsrahmen basiert auf dem CRISP-DM-Modell. Analysiert wurden acht Finanzinstrumente, darunter BTC-USD, ETH-USD und AAPL. Es wurden drei Modelle entwickelt: ein Basismodell mit Tagesdaten, ein Modell mit zusätzlichen Indikatoren (RSI, ADX) sowie ein Modell mit Multi-Temporal-Daten. Die Auswertung anhand von MSE, RMSE und MAE zeigt, dass keinem der Modelle eine klare Überlegenheit nachgewiesen werden konnte. Multitemporale Daten zeigten nur in Einzelfällen Vorteile, insbesondere bei stark trendbasierten Kursverläufen. Alle Modelle reagierten träge auf Trendwechsel. Für künftige Forschung sind hybride Modellansätze sinnvoll.

Key Words: LSTM, Kursvorhersage, Multitemporal, Deep Learning, RNN

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

ADX ..... *Average Directional Index*
BPTT ..... *Backpropagation Through Time*
CRISP-DM ..... *Cross Industry Standard Process for Data Mining*
DSR ..... *Design Science Research*
KNN ..... *Künstliches Neuronales Netzwerk*
LSTM ..... *Long Short Term Memory*
MAE ..... *Mean Squared Error*
ML ..... *Maschinelles Lernen*
MSE ..... *Mean Squared Error*
RMSE ..... *Root Mean Squared Error*
RNNs ..... *Recurrent Neural Networks*
RSI ..... *Relative Strength Index*

1 Einleitung

1.1 Beschreibung der Anwendungsdomäne

Um in der heutigen Welt sein Kapital langfristig vor Wertverlust zu schützen und der Inflation entgegenzuwirken, ist es notwendig, sein Kapital zu investieren. Eine Möglichkeit besteht darin, das Kapital in den Aktienmarkt oder den Kryptomarkt zu investieren. Beide Märkte sind über Broker im Internet zugänglich.

Investitionen in diese Märkte sind jedoch mit Unsicherheiten behaftet, da Kursverläufe starken Schwankungen unterliegen und von vielfältigen, oft schwer vorhersehbaren Faktoren beeinflusst werden. Ein zentrales Problem besteht daher in der möglichst präzisen Vorhersage zukünftiger Kursentwicklungen, um fundierte Kauf- und Verkaufsentscheidungen zu treffen. Hier setzt der Einsatz maschineller Lernverfahren an, insbesondere Recurrent Neural Networks (RNNs), die aufgrund ihrer Fähigkeit zur Verarbeitung sequenzieller Daten als geeignetes Werkzeug für Zeitreihenanalysen gelten.

1.2 Chance

Die Fähigkeit, Kursentwicklungen zuverlässig vorherzusagen, bietet Anlegern einen entscheidenden Wettbewerbsvorteil. Präzise Prognosen ermöglichen es, potenziell profitable Zeitpunkte für Ein- und Ausstiege zu identifizieren, wodurch sich das Risiko von Verlusten verringert und die Chance auf nachhaltige Renditen erhöht. Insbesondere in volatilen Märkten kann eine verbesserte Vorhersagegenauigkeit dazu beitragen, Fehlinvestitionen zu minimieren und die Kapitalallokation effizienter zu gestalten.

1.3 Forschungsdesgin

Das Forschungsdesign folgt dem Design-Science-Research-Ansatz (DSR) nach Hevner [1], da dieser die Entwicklung, Implementierung und Evaluation eines Artefakts systematisch ermöglicht. Ziel ist die iterative Konstruktion und Bewertung von drei Varianten eines LSTM-RNN zur Vorhersage von Kursentwicklungen.

Die Modelle werden mit TensorFlow/Keras implementiert und mithilfe des Keras Tuners optimiert. Zur Bewertung der Vorhersagequalität werden die Metriken Mean Squared Error (MSE), Root Mean Squared Error (RMSE) und Mean Absolute Error (MAE) herangezogen. Zusätzlich erfolgt ein Vergleich der Modelle gemäß der Zielsetzung des CRISP-DM.

Ziel der Arbeit ist es, den Einfluss unterschiedlicher Input-Daten, insbesondere technischer Indikatoren und Multi-Temporal-Daten, auf die Vorhersagegenauigkeit von LSTM-RNN basierten Kursprognosen zu analysieren. Damit leistet die Arbeit einen Beitrag zur Identifikation effektiver Modellierungsstrategien im Finanzbereich.

1.3.1 Forschungsmethode

Diese Arbeit verfolgt das Ziel, drei LSTM-RNN Modelle zu entwickeln und dessen Wirksamkeit anhand realer Finanzmarktdaten zu evaluieren. Methodisch handelt es sich daher um die konstruktive Entwicklung eines Artefakts in Form eines LSTM-RNN zur Vorhersage von Schlusskursen auf Basis multitemporaler Daten. Der Ansatz ist überwiegend quantitativ, da numerische Metriken zur Bewertung der Modellleistung herangezogen werden.

Zur systematischen Umsetzung wird das CRISP-DM-Modell (Cross Industry Standard Process for Data Mining) verwendet. Es bietet einen etablierten Rahmen für datengetriebene Projekte und eignet sich besonders für die Entwicklung und Bewertung praxistauglicher Artefakte. CRISP-DM berücksichtigt nicht nur die technische Korrektheit eines Modells, sondern auch dessen Zweckmäßigkeit im Anwendungskontext.[2, S.13]

Der Prozess besteht aus sechs Phasen:

1. **Business Understanding:** Definition des Projektziels. In diesem Fall die Verbesserung der Prognosegüte durch Integration multitemporaler Daten.
2. **Data Understanding:** Analyse und Exploration der verfügbaren Kursdaten, um Struktur, Qualität und Relevanz der Variablen (z. B. Schlusskurs, Handelsvolumen) zu verstehen.
3. **Data Preparation:** Aufbereitung der Rohdaten, u. a. durch Aggregation und Normalisierung, um eine konsistente Eingabebasis für die Modelle zu schaffen.
4. **Modelling:** Entwicklung und Training verschiedener LSTM-RNN-Modelle mit unterschiedlichen Feature-Kombinationen.
5. **Evaluation:** Bewertung der Modellleistung anhand von Metriken wie MSE, MAE und RMSE sowie Vergleich der Modelle untereinander.
6. **Deployment:** Die Phase der produktiven Integration wird in dieser Arbeit nicht vollständig umgesetzt, da der Fokus auf der Entwicklung und Evaluation des Modells liegt. Eine potenzielle Anwendung wird im Ausblick diskutiert.

2 Grundlagen des Maschinellen Lernens

Maschinelles Lernen ist ein Teilgebiet der Künstlichen Intelligenz. Künstliche Intelligenz lässt sich definieren als „die Studie, wie Computer Aufgaben ausführen, in denen der Mensch derzeit überlegen ist“ [3, S.138] (nach Elaine Rich).

Die grundlegende Idee des Maschinellen Lernens (ML) besteht in der automatischen Mustererkennung innerhalb von Daten. Obwohl es verschiedene Arten von ML-Systemen gibt, folgt der Lernprozess bei den meisten Verfahren einem ähnlichen Ablauf: Daten werden analysiert, statistische Zusammenhänge erkannt und darauf basierend ein Modell zur Lösung einer spezifischen Aufgabe erstellt.

Beim Maschinellen Lernen wird grundsätzlich zwischen überwachtes Lernen (Supervised Learning), unüberwachtes Lernen (Unsupervised Learning) und bestärkendes Lernen (Reinforcement Learning) unterschieden. Der wesentliche Unterschied zwischen Supervised und Unsupervised Learning liegt darin, ob die Daten gelabelt sind oder nicht.

Beim Unsupervised Learning werden ungelabelte Daten verwendet. Das bedeutet, dass zu den vorhandenen Eingabedaten keine Zielwerte (Labels) vorliegen, entweder weil sie nicht definiert sind oder nicht definiert werden können. Ziel dieses Lernverfahrens ist es, Strukturen, Muster oder Zusammenhänge innerhalb der Daten eigenständig zu identifizieren. Typische Methoden sind Clusteranalyse und Dimensionsreduktion. Ein häufig genanntes Anwendungsbeispiel ist die Kundensegmentierung, bei der Kunden anhand ihres Verhaltens gruppiert werden, ohne dass die Gruppen im Vorfeld festgelegt sind.

Im Gegensatz dazu verwendet Supervised Learning gelabelte Daten, also Datensätze, bei denen zu jedem Eingabewert ein zugehöriger Zielwert (Label) vorliegt. Das Modell lernt während des Trainings, die Abweichung zwischen Vorhersage und tatsächlichem Zielwert zu minimieren. Anschließend kann es Zielwerte für bisher unbekannte Eingabedaten vorhersagen. Supervised Learning wird weiter unterteilt in Klassifikation (kategoriale Zielwerte) und Regression (kontinuierliche Zielwerte).

In dieser Arbeit werden RNNs eingesetzt, das auf dem Prinzip des Supervised Learning basiert. Die zugrunde liegenden Rohdaten (historische Kursverläufe) dienen als Eingabe, während bekannte Zielwerte (zukünftige Kurswerte) im Training als Referenz verwendet werden. Da es sich bei den vorherzusagenden Werten um kontinuierliche Zahlen handelt, wird das Problem als Regressionsaufgabe behandelt. [4, Kapitel 5 Machine Learning Basics]

Reinforcement Learning stellt eine dritte Kategorie dar, bei der ein Agent durch Interaktion mit einer Umgebung lernt, optimale Handlungen auszuführen. Dabei erhält er Belohnungen oder Bestrafungen als Feedback (eine numerische Zahl) für seine Entscheidungen.[5, S.2]

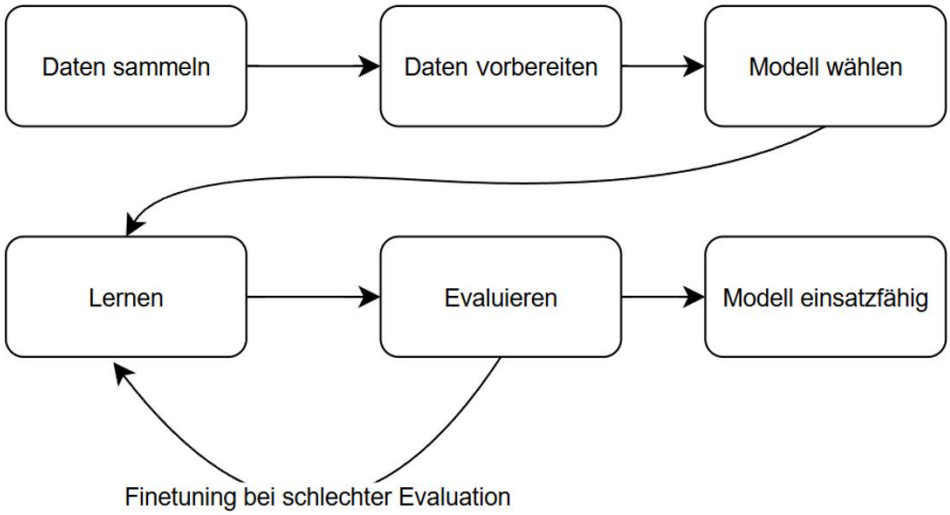

Abbildung 1 Ablauf Erstellen eines Machine Learning Modells

Abbildung 1 zeigt den grundlegenden Ablauf beim Aufbau eines ML-Modells sowie die Vorbereitung der benötigten Daten.

Zunächst erfolgt die Datensammlung. Diese kann auf unterschiedliche Weise geschehen: Daten können entweder selbst erhoben, aus öffentlich zugänglichen Quellen bezogen oder kostenpflichtig erworben werden.

Anschließend werden die Daten vorbereitet. Dazu gehört unter anderem der Umgang mit fehlenden Werten, die entweder entfernt oder durch geeignete Ersatzwerte ersetzt werden. Es wird geprüft, ob die Daten konsistent und die Bezeichnungen einheitlich sind (zum Beispiel einheitliche Geschlechtsangaben wie „m“ und „w“ statt gemischter Schreibweisen wie „männlich“, „m“ oder „Mann“). Einige Modelle, wie das RNN, erfordern zusätzlich eine Skalierung der Daten, etwa auf einen Wertebereich von 0 bis 1.

Nach der Datenaufbereitung wird ein Modelltyp ausgewählt. Grundsätzlich wird dabei zwischen zwei Hauptkategorien unterschieden: Klassifikation, bei der Daten in Kategorien eingeteilt werden (zum Beispiel „gesund“ oder „ungesund“) und Regression, bei der ein kontinuierlicher Zielwert vorhergesagt wird (zum Beispiel der zukünftige Kurs eines Finanzinstruments). Je nach Problemstellung wird anschließend ein konkretes Modell ausgewählt.

Im Schritt der Evaluation wird überprüft, wie gut das Modell trainiert wurde. Hierbei werden auch sogenannte Hyperparameter angepasst, die das Verhalten des Modells wesentlich beeinflussen. Gerade bei künstlichen neuronalen Netzen ist eine Feinjustierung (Hyperparameter-Tuning) entscheidend für die Modellqualität. Nach jeder Anpassung wird das Modell erneut trainiert. Dieser Prozess ist iterativ und wird so lange wiederholt, bis die festgelegten Qualitätskriterien erfüllt sind. Erst dann gilt das Modell als einsatzbereit.

2.1 Künstliche Neuronen

2.1.1 Grundaufbau

Um ein künstliches neuronales Netzwerk (KNN) aufzubauen, benötigt man künstliche Neuronen. Ein künstliches Neuron ist der grundlegende Baustein jedes neuronalen Netzwerks. Obwohl es mittlerweile viele Varianten künstlicher Neuronen gibt, beruhen sie alle auf demselben Grundprinzip.

Das einfachste und zugleich eines der ersten künstlichen Neuronen, das in der Lage ist zu lernen, ist das Perzeptron, das von Frank Rosenblatt beschrieben wurde. [3, S.211]

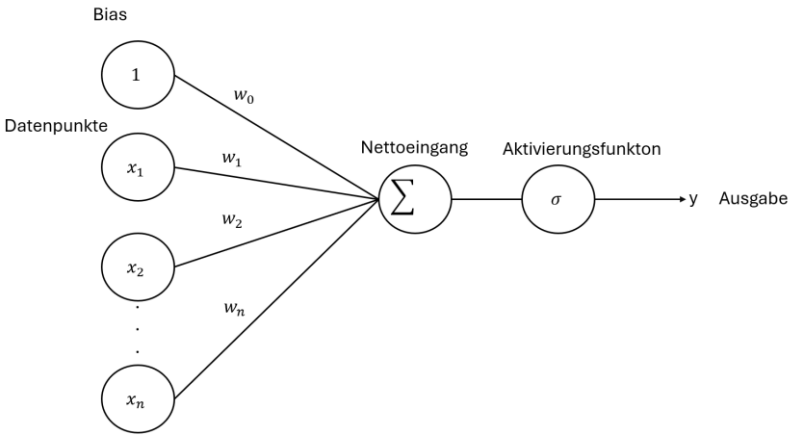

Abbildung 2 Rosenblatt Perzeptron

Quelle: In Anlehnung an Matteo Sangiorgo, 2025, https://www.researchgate.net/figure/Structure-of-Rosenblatts-perceptron-The-neuron-first-computes-a-weighted-sum-of-input_fig4_338428101

Der Aufbau eines künstlichen Neurons ist wie folgt: Gegeben ist ein Eingabevektor x der Länge m, welcher die Eingangsdaten repräsentiert. Dieser Vektor wird mit einem Gewichtsvektor w der gleichen Dimension multipliziert. Anschließend werden die gewichteten Eingaben summiert, was den sogenannten Nettoeingang ergibt.

Zusätzlich wird ein Bias-Term eingeführt. In der Abbildung 2 ist dies als Konstante 1 dargestellt, die mit einem eigenen Gewicht w_0 multipliziert wird. Der Bias ermöglicht es dem

Modell, die Entscheidungsgrenze unabhängig vom Ursprung zu verschieben, und wird aus mathematischen Gründen oft direkt zum Eingabevektor hinzugefügt.

Anschließend wird der Nettoeingang an eine Aktivierungsfunktion übergeben. Diese Funktion berechnet den Ausgabewert y des Neurons. Beim klassischen Perzeptron nach Rosenblatt handelt es sich um eine binäre Ausgabe. Das Ergebnis ist entweder 0 oder 1. Dadurch eignet sich dieses Neuron für einfache binäre Klassifikationsaufgaben, sofern die Daten linear trennbar sind.

Im Lernprozess werden die Gewichte iterativ angepasst, um die Klassifikationsgenauigkeit zu verbessern. Ziel ist es, dass das Modell bei wiederholter Präsentation der Trainingsdaten eine möglichst korrekte Trennung der Klassen vornehmen kann.

2.1.2 Mathematische Betrachtung

Gegeben sei ein Eingabevektor x und ein Gewichtungsvektor w. Der Nettoeingang z ergibt sich durch das Skalarprodukt der beiden Vektoren. Der erste Vektor (Eingabevektor) ist der Datenpunkt x. Der zweite Vektor sind die Gewichte.

$$Z = x^T * w$$

Formel 1 Nettoeingang

Dabei ist zu beachten, dass der Bias bereits im Eingabevektor enthalten ist. Typischerweise wird hierfür eine 1 als zusätzliches Element in den Eingabevektor aufgenommen. Der zugehörige Gewichtswert w_0 fungiert dann als Bias-Term (bzw. als negativer Schwellenwert θ). Dies hat den Vorteil, dass die Aktivierungsfunktion keinen expliziten Vergleich mit einem Schwellenwert benötigt, stattdessen wird mit 0 verglichen.

Die Aktivierungsfunktion $\sigma(Z)$ im klassischen Perzeptron ist binär definiert:

$$\sigma(Z) = \begin{cases} 0, z \leq 0 \\ 1, z > 0 \end{cases}$$

Formel 2 Binäre Aktivierungsfunktion

Beim Rosenblatt-Neuron ist die Aktivierungsfunktion gleichzeitig auch die Ausgabe y. Die Ausgabe y des Neurons entspricht direkt dem Ergebnis der Aktivierungsfunktion.

Das Lernen erfolgt iterativ über mehrere Epochen. Eine Epoche entspricht einem vollständigen Durchlauf durch den Trainingsdatensatz. Die Gewichte werden dabei nur angepasst, wenn das Modell falsch klassifiziert:

- Wenn der tatsächliche Zielwert 1 ist, aber der Output 0, wird der Eingabevektor x zum Gewichtungsvektor w addiert.
- Wenn der tatsächliche Zielwert 0 ist, aber der Output 1, wird der Eingabevektor x vom Gewichtungsvektor w subtrahiert.

Diese einfache Regel sorgt dafür, dass das Modell Schritt für Schritt lernt, die Daten korrekt zu klassifizieren.

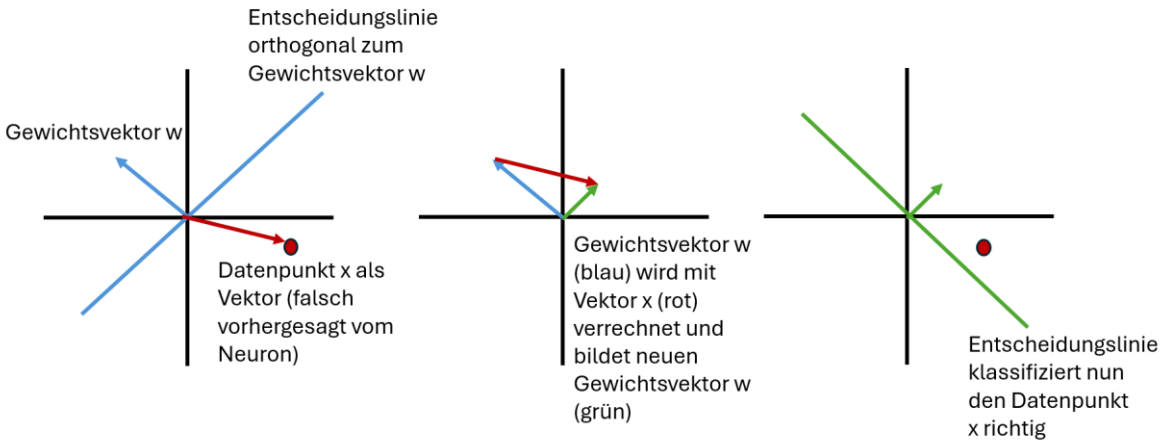

Abbildung 3 Visuelle Interpretation Entscheidungsline und Neuron Lernen

Quelle: In Anlehnung an [3, S.216]

In Abbildung 3 ist das Lernverhalten des Rosenblatt-Perzeptrons visuell dargestellt. Dabei wird gezeigt, wie das Perzeptron durch Anpassung seiner Parameter eine Entscheidungsgrenze lernt. Der Input-Vektor x, der Gewichtungsvektor w sowie der Bias b lassen sich geometrisch als eine Hyperplane interpretieren, die zwei Klassen im Merkmalsraum voneinander trennt.

Der Bias fungiert als y-Achsenabschnitt und verleiht der Hyperplane zusätzliche Flexibilität, da die Trennlinie dadurch nicht mehr durch den Ursprung verlaufen muss. Die resultierende Entscheidungslinie (in zwei Dimensionen eine Gerade) trennt die beiden Klassen voneinander. Der Gewichtungsvektor w steht dabei senkrecht auf der Entscheidungsgrenze und definiert deren Orientierung.

Wenn ein Eingabevektor x falsch klassifiziert wird, wird der Gewichtungsvektor durch das Lernverfahren angepasst. Dies führt zu einer Verschiebung der Entscheidungsgrenze in Richtung des betreffenden Datenpunkts. Auf diese Weise wird die Hyperplane iterativ so verändert, dass sie die Klassen zunehmend besser voneinander trennt.

Moderne künstliche Neuronen verwenden verbesserte Aktivierungsfunktionen im Vergleich zum ursprünglichen Rosenblatt-Perzeptron. Beim Rosenblatt-Perzeptron führt jede Fehlklassifikation zu einer vollständigen Addition oder Subtraktion des Input-Vektors. Dadurch verändert sich die Entscheidungsgrenze oft abrupt und nicht immer effizient. In heutigen neuronalen Netzen werden kontinuierliche Aktivierungsfunktionen wie z. B. die Sigmoid-, Tanh- oder ReLU-Funktion verwendet.

Darüber hinaus nutzen moderne Neuronen eine Loss-Funktion zur quantitativen Bewertung des Fehlers sowie das Verfahren der Backpropagation, um den Fehler gezielt auf die Gewichte zurückzuführen und diese entsprechend anzupassen (siehe 2.2 Künstliche Neuronale Netzwerke). Dies führt zu einem deutlich stabileren Lernprozess als beim Rosenblatt Perzeptron.

2.2 Künstliche Neuronale Netzwerke

2.2.1 Grundaufbau

Das Grundkonzept eines künstlichen neuronalen Netzwerks besteht darin, mehrere künstliche Neuronen miteinander zu verbinden. Dadurch erhöht sich die Modellkomplexität, sodass das Netzwerk in der Lage ist, auch nichtlineare Probleme zu lösen. Im Rahmen dieser Arbeit wird ausschließlich ein künstliches neuronales Netzwerk betrachtet, das supervised learning im Bereich der Regression unterstützt.

Ein künstliches Neuronales Netzwerk ist wie aus mindestens 3 Schichten aufgebaut, siehe Abbildung 4.

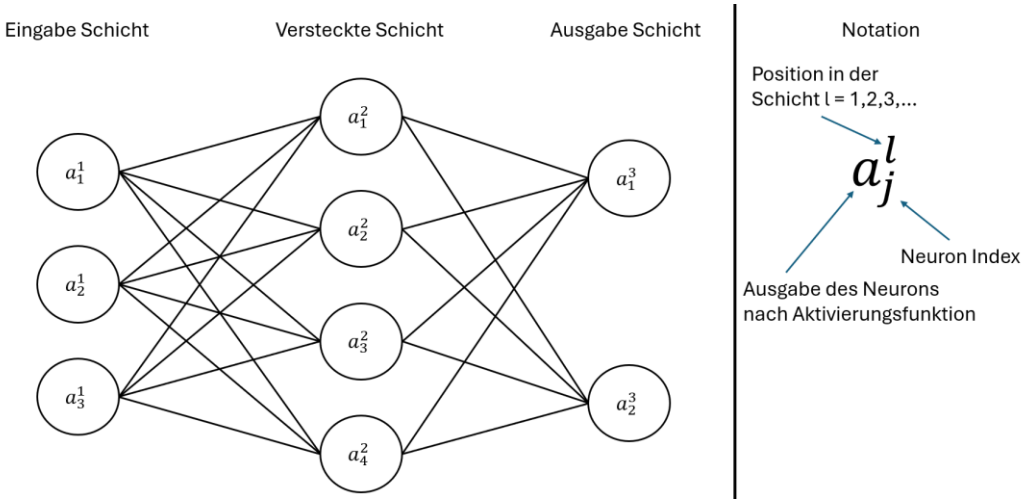

Abbildung 4 Neuronales Netzwerk Beispiel mit Notation

Die Eingabeschicht (Input Layer) stellt die erste Ebene eines künstlichen neuronalen Netzwerks dar und ist für die Entgegennahme der Eingabedaten verantwortlich. Ihre Größe ist nicht frei wählbar, sondern richtet sich strikt nach der Anzahl der Merkmale (Features) der Eingangsdaten.

Darauf folgt mindestens eine versteckte Schicht (Hidden Layer), welche die Ausgabe der Eingabeschicht weiterverarbeitet. Sowohl die Anzahl der versteckten Schichten als auch die Anzahl der Neuronen innerhalb dieser Schichten sind modellabhängig frei wählbar. Durch die Verwendung versteckter Schichten kann die Modellkapazität erhöht werden, wodurch auch komplexere nichtlineare Zusammenhänge in den Daten erfasst werden können.

Die Ausgabeschicht (Output Layer) bildet die letzte Schicht des Netzwerks. Ihre Neuronen liefern das finale Ergebnis des Vorhersageprozesses. Die Anzahl der Neuronen in dieser Schicht hängt von der jeweiligen Aufgabenstellung ab, z. B. der Anzahl der Zielklassen bei einer Klassifikation oder der Anzahl der Regressionsausgaben.

Ein vollständiger Durchlauf aller Trainingsdaten durch das Netzwerk wird als Epoche bezeichnet. Innerhalb jeder Epoche erfolgt zunächst der sogenannte Vorwärtsdurchlauf (Forward Pass), bei dem die Eingabedaten unter Verwendung der aktuellen Gewichtungen und Biases durch das Netzwerk propagiert werden. Dies erfolgt von der Eingabeschicht über die versteckten Schichten bis hin zur Ausgabeschicht. Die dabei entstehende Ausgabe (Output) wird mit den tatsächlichen Zielwerten (Labels) verglichen, woraus ein Fehlerwert berechnet wird.

Anschließend erfolgt die Rückwärtsausbreitung des Fehlers (Backpropagation). Dabei wird der Fehlerwert durch das Netzwerk zurückgeführt, um den Beitrag jedes einzelnen Neurons und jeder Gewichtung zum Gesamtergebnis zu bestimmen. Auf Basis dieser Informationen werden die Gewichte angepasst, typischerweise unter Verwendung eines Optimierungsverfahrens wie dem Gradientenabstieg. Dieser iterative Anpassungsprozess wird allgemein als Lernprozess bezeichnet.[4, S 164-166]

2.2.2 Mathematische Betrachtung

Jedes neuronale Netzwerk besitzt eine Eingabe Schicht, Versteckte Schicht und eine Ausgabe Schicht. Die Ausgabe eines Neurons einer Schicht wird wie folgt berechnet:

$$a_j^{(l)} = \sigma\left(\sum_{k=1}^{n^{(l-1)}} w_{jk}^{(l)} * a_k^{(l-1)} + b_j^{(l)}\right)$$

Formel 3 Berechnung der Aktivierung des Neurons j in Schicht l [6, Kapitel Warm up Formel (23)]

- $a_j^{(l)}$ Aktuelle Ausgabe des Neuron j in Schicht l
- $n^{(l-1)}$ Anzahl der Neuronen der vorherigen Schicht
- $w_{jk}^{(l)}$ Gewicht der Verbindung vom Neuron i in der vorherigen Schicht l-1 zum Neuron j in der aktuellen Schicht l. Dabei gilt:
  - l ist die aktuelle Schicht (z. B. 0, 1, 2, …),
  - j ist der Index des Neurons der aktuellen Schicht,
  - k ist der Index des Neurons in der vorherigen Schicht.
- $a_k^{(l-1)}$ Ausgabe (Output), der Verbindung aus der vorherigen Schicht l-1
- $b_j^{(l)}$ Bias des Neuron j
- σ Aktivierungsfunktion

Konkret bedeutet das beispielsweise, dass die Aktivierung des Neurons $a_1^{(2)}$ von Abbildung 4 wie folgt berechnet wird:

$$a_1^{(2)} = \sigma\left(w_{1,1}^{(2)} * a_1^{(1)} + w_{1,2}^{(2)} * a_{12}^{(1)} + w_{1,3}^{(2)} * a_{13}^{(1)} + b_1^{(2)}\right)$$

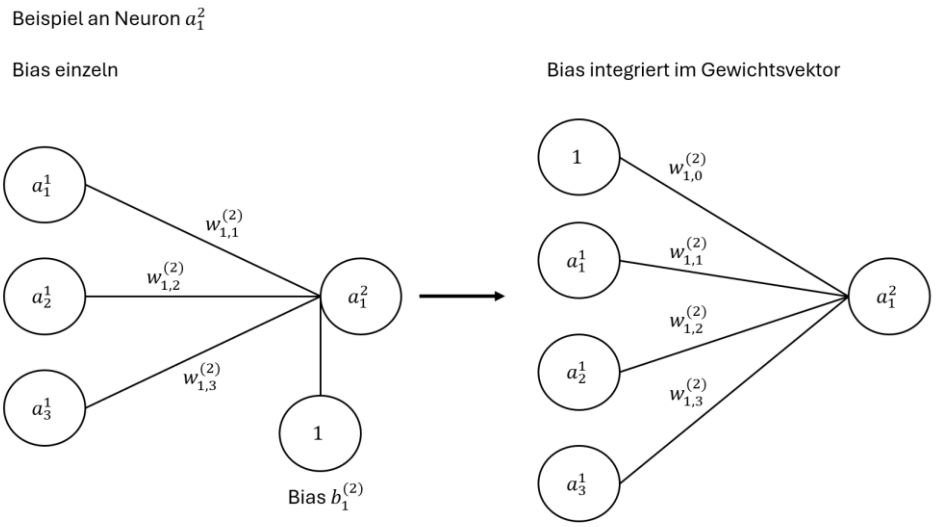

Abbildung 5 Bias Integration eines Neurons

Vereinfacht kann der Bias als zusätzliches Gewicht integriert werden. Der Vorteil dieser Methode ist, dass der Bias direkt in den Eingabevektor aufgenommen wird, wodurch keine

separate Behandlung mehr nötig ist. Die Berechnung wird dadurch kompakter und benötigt weniger einzelne Rechenschritte. Für das Neuron $a_1^{(2)}$ ergibt sich dann z. B. die vereinfachte Aktivierungsformel:

$$a_1^{(2)} = \sigma\left(w_{1,0}^{(2)} * 1 + w_{1,1}^{(2)} * a_1^{(1)} + w_{1,2}^{(2)} * a_{12}^{(1)} + w_{1,3}^{(2)} * a_{13}^{(1)}\right)$$

Es gibt grundsätzlich zwei Möglichkeiten, die Ausgabe eines Neurons zu berechnen. In der Praxis hat sich die Berechnung für ganze Schichten durchgesetzt. Dabei werden die Gewichte und die Eingabewerte (also die Ausgaben der vorherigen Schicht) mit einer Matrixmultiplikation verrechnet. Danach wird der Bias-Vektor hinzugefügt. Mathematisch sieht das so aus:

$$a^{(l)} = \sigma\left(W^{(l)} * a^{(l-1)} + b^{(l)}\right)$$

Formel 4 Berechnung der Aktivierung der Schicht l

- $W^{(l)} \in R^{m \times n}$ Gewichtsmartix, Dabei gilt:
- $a^{(l-1)} \in R^n$ Ausgang (Output) der vorheriger Neuronenschicht
- $b^{(l)} \in R^m$ Bias Vektor der Schicht
    - m = Anzahl Neuronen in der aktuellen Schicht (Zeilen der Gewichtsmatrix)
    - n = Anzahl Neuronen in der vorherigen Schicht (Spalten der Gewichtsmatrix)

Für eine einfache Notation später in der Backpropagation wird z zusammengefasst als:

$$z^{(l)} = W^{(l)} * a^{(l-1)} + b^{(l)}$$

Daraus leitet sich folgenden Term ab für a:

$$a^{(l)} = \sigma(z^{(l)})$$

Aktivierungsfunktionen sollten bestimmte Eigenschaften erfüllen, um in neuronalen Netzen effektiv eingesetzt werden zu können. Sie sind idealerweise monoton, um stetige Ausgaben ohne Unterbrechungen zu gewährleisten. Zudem müssen sie nicht-linear sein. Die Wertebereiche solcher Funktionen liegen typischerweise in einem definierten Intervall, das vom gesamten Definitionsbereich $(-\infty,+\infty)$ der Eingaben auf einen festen Wertebereich abbildet. Schließlich ist Differenzierbarkeit eine zentrale Anforderung, da sie die Berechnung von Gradienten im Backpropagation-Algorithmus ermöglicht und somit das Training der Netzwerke erst praktikabel macht.

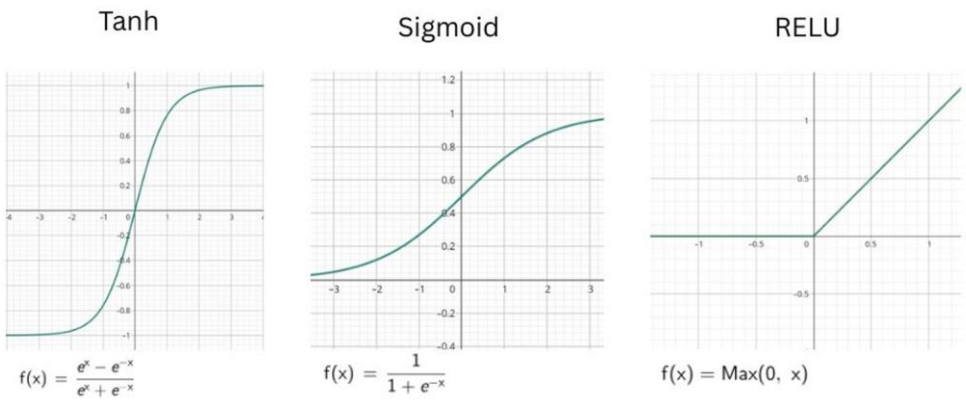

Abbildung 6 Aktivierungsfunktionen

$$\sigma = Aktivierungsfunktion$$

Das heißt das Ergebnis von $a^{(l)}$ geht durch die Aktivierungsfunkt. In Abbildung 6 sind drei Aktivierungsfunktionen abgebildet. Die Wertebereiche unterscheiden sich. Welcher Aktivierungsfunktion dann am Ende im Netzwerk verwendet wird hängt vom Finetuning (Hyperparameter) ab.

Am Ende enthält das Netzwerk eine Ausgabe von a^L (die Ausgabe-Schicht). Nun kann mithilfe einer Kostenfunktion, in diesem Fall dem MSE, der Fehler berechnet werden.

$$C = \frac{1}{2}\sum_{j=1}^{n^{(L)}}\left(a_j^{(L)} - y_j\right)^2$$

Formel 5 Kostenfunktion

- $n^{(L)}$ Anzahl der Ausgang Neuronen
- $a_j^{(L)}$ Output von Neuron mit Index j
- y_j Zielwert

Mithilfe des Fehlers wird nun die Backpropagation durchgeführt. Ziel der Backpropagation ist es, die Kostenfunktion zu minimieren. Die Backpropagation realisiert dabei das Lernen durch Anpassung der Gewichte im Netzwerk. Um zu bestimmen, wie stark die Kostenfunktion von den Gewichten beeinflusst wird, wird ein Fehlerterm berechnet:

$$\delta^{(out)} = \left(a^{(out)} - y\right) \odot \sigma'\left(z^{(out)}\right)$$

Formel 6 Berechnung des Fehlerterms der Ausgabe Schicht

- $\delta^{(out)}$ Fehlervektor der Ausgabe Schicht
- $a^{(out)}$ Vektor der Ausgabe Schicht

- y Zielwert
- $\sigma'(z^{(out)})$ beschreibt wie empfindlich die Aktivierung $a^{(L)}$ auf Änderungen im gewichteten Input $z^{(L)}$

Mithilfe des Fehlerterms wird berechnet, um wie viel der vorherigen Schicht zum Fehler beigetragen hat:

$$\delta^{(l)} = \left(W^{(l+1)}\right)^T * \delta^{(l+1)} \odot \sigma'(z^{(l)})$$

Formel 7 Berechnung des Fehlerterms pro Schicht

- $\delta^{(l)}$ Fehlerterm der aktuellen Schicht l
- $\left(W^{(l+1)}\right)^T$ Gewichtsmatrix der nächsten Schicht
- $\delta^{(l+1)}$ Fehlerterm der nächsten Schicht
- $\sigma'(z^{(l)})$ Ableitung der Aktivierungsfunktion in der aktuellen Schicht (dafür wird der Input von der vorherigen Schicht benötigt)

Wenn alle Fehlerterme berechnet sind, werden die Gradienten der Fehlerfunktion bestimmt, um die Gewichtsmatrizen anzupassen.

$$\frac{\partial C}{\partial W^{(l)}} = \delta^{(l)} * \left(a^{(l-1)}\right)^T$$

Formel 8 Gradient der Gewichtsmatrix der Schicht l

- $\frac{\partial C}{\partial W^{(l)}}$ Gradient des Fehlers
- $\delta^{(l)}$ Fehlerterm der aktuellen Schicht l
- $\left(a^{(l-1)}\right)^T$ Aktivierung der vorherigen Schicht

$$\frac{\partial C}{\partial b^{(l)}} = \delta^{(l)}$$

Formel 9 Gradient der Bias der Schicht l

- $\frac{\partial C}{\partial b^{(l)}}$ Gradient des Fehlers für den Bias

Die Gradienten geben die Richtung an, in welche Richtung der Fehler korrigiert wird. Zuletzt werden nun die Gewichte angepasst.

$$W^{(l)} = W^{(l)} - \eta * \frac{\partial C}{\partial W^{(l)}}$$

Formel 10 Aktualisierung der Gewichte

$$b^{(l)} = b^{(l)} - \eta * \frac{\partial C}{\partial b^{(l)}}$$

Formel 11 Aktualisierung der Biases

- $\frac{\partial C}{\partial W^{(l)}}$ Gradient des Fehlers
- η Lernrate (Learning rate)

Durch die Backpropagation wird der Fehler vom Ausgabe Layer bis hin zur Eingabe Layer propagiert und pro Schicht wird der beigetragene Fehler bestimmt und die Gewichte der Layer werden dementsprechend angepasst.[6, Kapitel The four Fundamental Equations behind Backpropagation]

2.3 Recurrent Neural Network

2.3.1 Grundaufbau

Ein RNN (rekurrentes neuronales Netzwerk) ist ein neuronales Netz, dessen Neuronen Rückkopplungen (rekurrente Verbindungen) besitzen. Das bedeutet, dass die Ausgabe eines Neurons als Eingabe für sich selbst oder andere Neuronen in späteren Zeitschritten genutzt werden kann. Der große Vorteil liegt darin, dass ein RNN, Sequenzen als Input verarbeiten kann und nicht nur einzelne Datenpunkte. Ein RNN kann zeitliche Abhängigkeiten erlernen. Dadurch lassen sich zeitliche Datenreihen mit Mustern und Abhängigkeiten effektiv modellieren.

Bei RNNs unterscheidet man zwischen verschiedenen Strukturtypen. Von einer Many-to-One-Architektur spricht man, wenn aus einer Eingabesequenz ein einzelner Output erzeugt wird. Ein Beispiel: Der Input besteht aus einer Sequenz von 15 Datenpunkten, und das Modell gibt den 16. Datenpunkt als Output aus. Im Rahmen dieser Arbeit werden ausschließlich Modelle mit einer Many-to-One-Architektur entwickelt.

Von einer Many-to-Many-Architektur spricht man, wenn sowohl der Input als auch der Output aus Sequenzen bestehen. Zum Beispiel könnten 15 Eingabedaten zu 10 darauffolgenden Ausgabedaten führen.

Dies ist möglich, weil ein RNN nicht aus einzelnen Neuronen im klassischen Sinne aufgebaut ist, sondern aus sogenannten versteckten Schichten (Hidden Layers). Eine

Hidden Layer enthält eine bestimmte Anzahl an Neuronen. Diese Neuronen sind über versteckte Status (Hidden States) zeitlich miteinander verknüpft und können dadurch zeitliche Abhängigkeiten innerhalb der Daten erfassen und lernen. [7, Kapitel 2.1.3]

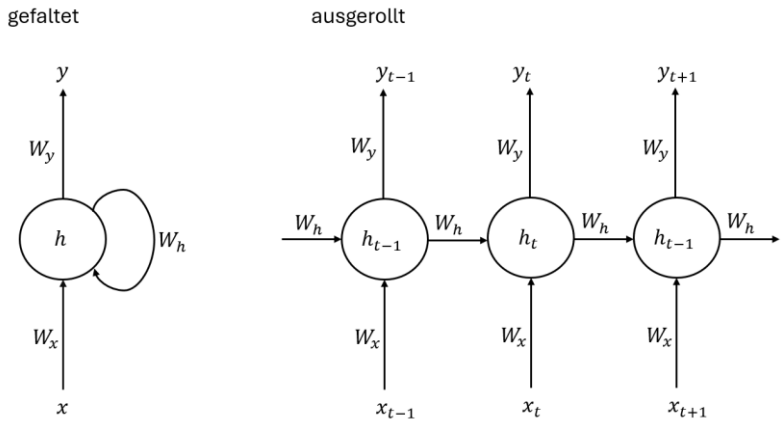

Abbildung 7 Aufbau eines Hidden State

Quelle: In Anlehnung [8, Seite 1 Figure 1]

Wie in Abbildung 7 zu sehen ist, enthält eine Hidden Layer mehrere Neuronen. Im Unterschied zu einem klassischen neuronalen Netzwerk werden bei einem RNN drei verschiedene Gewichtsmatrizen verwendet:

- Eine Gewichtsmatrix W_x verbindet den Input mit der Hidden State.
- Eine weitere Gewichtsmatrix W_h modelliert die rekurrenten Verbindungen, also die Verbindung des vorherigen Hidden State mit dem aktuellen Hidden State. Diese rekurrenten Verbindungen ermöglichen es dem Netzwerk, zeitliche Abhängigkeiten zu speichern.
- Eine dritte Gewichtsmatrix W_y verbindet die Hidden State mit der Ausgabeschicht.

Durch diese drei Matrizen kann das RNN nicht nur die aktuellen Eingabedaten, sondern auch die Informationen aus früheren Zeitschritten berücksichtigen, was für die Verarbeitung von Sequenzen und zeitabhängigen Daten entscheidend ist.

2.3.2 Mathematische Betrachtung

„p“ sei die Dimension der hidden states, also die Anzahl der Neuronen in der Hidden Layer. „d“ sei die Dimension des Inputs, also die Anzahl der Merkmale (Features) des Eingabevektors. „q“ sind die Anzahl der Ausgabe Neuronen.

Der Hidden State, Input und Output haben folgende Dimensionen:

$$h_t \in R^p, \qquad x_t \in R^p, \qquad y_t \in R^p$$

Die Gewichtsmatrizen haben folgende Dimensionen:

$$W_h \in R^{p \times p}, \qquad W_x \in R^{p \times d}, \qquad W_y \in R^{q \times p}$$

Die Bias-Vektoren haben folgende Dimensionen:

$$b_h \in R^p, \qquad b_y \in R^q$$

$$h_t^{(l)} = \sigma(W_x^{(l)} x_t^{(l)} + W_h^{(l)} h_{t-1}^{(l)} + b_h^{(l)})$$

Formel 12 Berechnung hidden State der Schicht l

- $h_t^{(l)}$ Hidden State zu Zeit t in Schicht l
- $W_x^{(l)}$ Gewichtsmatrix zu $x_t^{(l)}$
- $W_h^{(l)}$ Gewichtsmatrix zu $h_{t-1}^{(l)}$
- $h_{t-1}^{(l)}$ Hidden State aus der gleichen Schicht l, aber vom vorherigen Zeitpunkt t−1
- $x_t^{(l)}$ Input
- $b_h^{(l)}$ Bias des hidden State

Der Hidden State verarbeitet sowohl den aktuellen Input als auch den vorherigen hidden State (t-1) und ermöglicht so die Berücksichtigung vergangener Zustände im zeitlichen Kontext.

$$y = W_y * h_t^{(L)} + b_y$$

Formel 13 Ausgabe des RNN

- y Output des Netzwerks
- W_yGewichtsmatrix zu $h_t^{(L)}$
- $h_t^{(L)}$ Hidden State des Output-Layer
- b_y Bias des Outputs

Der Fehler des Outputs wird anschließend mithilfe einer Loss-Funktion bestimmt. Da beim RNN pro Zeiteinheit (Sequenzschritt) ein Output erzeugt wird, werden die Fehler über alle Zeitschritte aufsummiert.

Diese Verlustfunktion wird typischerweise bei RNNs für Regression und in Many-to-One-Architekturen verwendet.

$$L = \frac{1}{T}\sum_{t=1}^{T} \|y_t^{out} - y_t^{true}\|^2$$

Formel 14 Verlustfunktion RNN Many-to-One

- L Kostenfunktion des RNNs
- T Sequenzlänge
- y_t^{out} Output
- y_t^{true} Zielwert

Anschließend werden mithilfe des Fehlers die Fehlerterme bestimmt.t

$$\delta_t^{(out)} = y_t^{pred} - y_t^{true} \odot \sigma'\left(z_t^{(out)}\right)$$

Formel 15 Fehlerterm Output RNN

- $\delta_t^{(out)}$ Fehlerterm des Outputs in Zeitschritt t
- y_t^{pred} Ausgabe des RNN (Prediction, bzw. Vorhersage)
- y_t^{true} Target Value
- $\sigma'\left(z_t^{(out)}\right)$ Ableitung der Aktivierungsfunktion der Ausgabenschicht

$$\delta_t^{(h)} = \left(W_{hy}^T \delta_t^{(out)} + W_{hh}^T \delta_{t+1}^{(h)}\right) \circ \sigma'(a_t)$$

Formel 16 Fehlerterm pro Schicht RNN

- $\delta_t^{(h)}$ Hidden State Fehlerterm des Zeitschritt t
- W_{hy}^T Transponiertes Gewicht von Hidden State zu Output
- $\delta_t^{(out)}$ Fehlertem des Outputs zu Zeit t
- W_{hh}^T Transponiertes Gewicht von Hidden State zu Hidden State
- $\delta_{t+1}^{(h)}$ Fehlerterm von Zeitpunkt t+1
- $\sigma'(a_t)$ wobei gilt: $a_t = W_x x_t + W_h h_{t-1} + b$

Diese Fehlerterme repräsentieren, wie stark ein Hidden State am Fehler beteiligt war. Mithilfe der Fehlerterme kann man nun die Gradienten berechnen.

$$\frac{\partial L}{\partial W_{hy}} = \sum_{t=1}^{T} \delta_t^{(out)} * h_t^T$$

Formel 17 Gradient des Hidden State Zeitpunkt t

- $\delta_t^{(out)}$ Fehlertem des Outputs zu Zeit t
- h_t^T Transponierte Hidden State zu Zeitpunkt t

Der Gradient $\frac{\partial L}{\partial W_{hy}}$ misst, wie stark Änderungen im Hidden State den Loss über den Output beeinflussen.

$$\frac{\partial L}{\partial W_x} = \sum_{t=1}^{T} \delta_t^{(h)} * x_t^T$$

Formel 18 Gradient Gewichtsmatrix

- $\delta_t^{(h)}$ Fehlertem des hidden state zu Zeit t
- x_t^T Transponierte Input zu Zeitpunkt t

Der Gradient $\frac{\partial L}{\partial W_x}$ misst, wie stark sich Änderungen in der Gewichtsmatrix auf den Loss auswirken.

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^{T} \delta_t^{(h)} * x_{t-1}^T$$

Formel 19 Gradient hidden state Zeitpunkt t-

- $\delta_t^{(h)}$ Fehlertem des hidden state zu Zeit t
- x_{t-1}^T Transponierte Input zu Zeitpunkt t-1

Der Gradient $\frac{\partial L}{\partial W_h}$ zeigt, wie stark sich Änderungen im Einfluss des vorherigen Hidden States auf den aktuellen Zustand auf den Loss auswirken. Nun können die Gewichte angepasst werden.

$$W_{hy} = W_{hy} - \eta * \frac{\partial L}{\partial W_{hy}}$$

Formel 20 Aktualisierung Gewichtsmatrix y

$$W_h = W_h - \eta * \frac{\partial L}{\partial W_x}$$

Formel 21 Aktualisierung Gewichtsmatrix h

$$W_x = W_x - \eta * \frac{\partial L}{\partial W_h}$$

Formel 22 Aktualisierung Gewichtsmatrix x

Neben den Gewichtsmatrizen werden auch die Bias-Vektoren des Netzwerks während des Trainings mittels Backpropagation durch die Zeit angepasst, um den Fehler zu minimieren.

Ein Problem bei klassischen RNNs ist, dass die Gradienten beim Training dazu neigen, entweder sehr groß (Gradient Exploding) oder nahezu null (Gradient Vanishing) zu werden. Dadurch ist das Netzwerk entweder nicht mehr in der Lage zu lernen oder es vergisst frühere Informationen, was zu schlechter Generalisierung (gelerntes Wissen auf unbekannte Daten anwenden) führt.

Um dieses Problem zu lösen, wurde das LSTM (Long Short-Term Memory) entwickelt. LSTM-Zellen ersetzen die Neuronen des Hidden Layer durch eine komplexere Struktur mit mehreren Gates. Diese Gates steuern, welche Informationen gespeichert, vergessen oder weitergegeben werden. So kann das Netzwerk sowohl Kurzzeit- als auch Langzeitinformationen effektiver speichern und verarbeiten.[8, Seite 4-5]

2.4 Long Short Term Memory

2.4.1 Grundaufbau

Um das Gradientenproblem zu lösen, stellten Hochreiter und Schmidhuber 1997 eine neue Neuronenarchitektur vor: das Long Short-Term Memory (LSTM). Durch den Einsatz spezieller Gates ermöglicht das LSTM die gezielte Steuerung von Kurzzeit- und Langzeitgedächtnis.

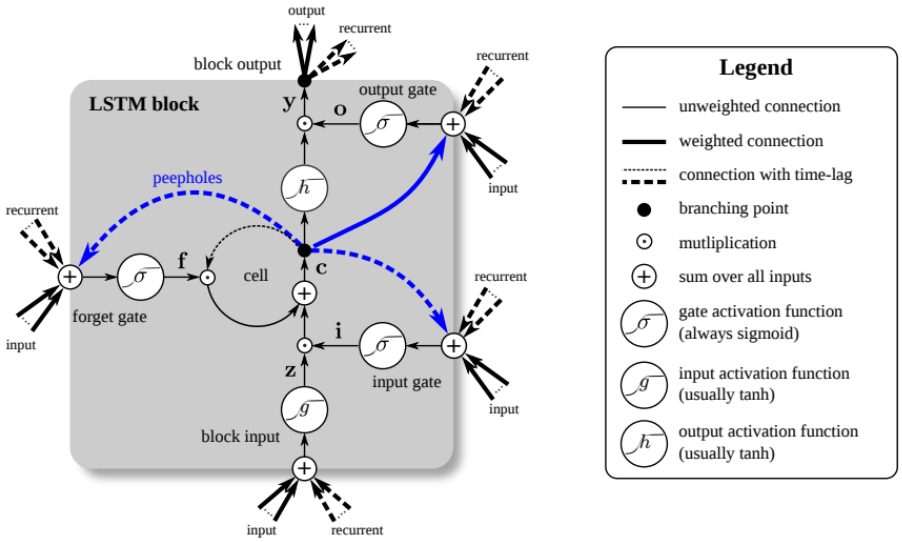

Abbildung 8 Aufbau eines LSTM

Quelle: [9, S.2]

Der Zellzustand (Cell state) C_t im LSTM stellt das Langzeitgedächtnis dar, da er den Gradientenfluss über viele Zeitschritte weitgehend unverändert weitergibt und somit

langfristige Abhängigkeiten ermöglicht. Im Gegensatz zu einfachen RNNs steuert das LSTM die Informationsverarbeitung über drei Gates: das Forget Gate, das Input Gate und das Output Gate. Alle drei Gates erhalten denselben Input, erfüllen jedoch unterschiedliche Funktionen. Das Forget Gate bestimmt, welche Informationen aus dem vorherigen Zellzustand gelöscht werden, während das Input Gate entscheidet, welche neuen Informationen in den Zellzustand aufgenommen werden. Schließlich regelt das Output Gate, welcher Teil des Zellzustands als Hidden State ausgegeben wird.

2.4.2 Mathematische Betrachtung

Block input

$$z^t = g\left(W_z x^t + R_z h^{t-1} + b_z\right) \text{ mit } g(x) = \tanh(x)$$

Formel 23 Block Input LSTM

Input Gate:

$$i^t = \sigma\left(W_i x^t + R_i h^{t-1} + p_i \circ c^{t-1} + b_i\right) \text{ mit } \sigma(x) = \frac{1}{1+e^{-x}}$$

Formel 24 Input Gate LSTM

Forget Gate

$$f^t = \sigma\left(W_f x^t + R_f h^{t-1} + p_f \circ c^{t-1} + b_f\right) \text{ mit } \sigma(x) = \frac{1}{1+e^{-x}}$$

Formel 25 Forget Gate LSTM

Output Gate:

$$o^t = \sigma\left(W_f x^t + R_f h^{t-1} + p_f \circ c^{t-1} + b_f\right) \text{ mit } \sigma(x) = \frac{1}{1+e^{-x}}$$

Formel 26 Output Gate LSTM

Cell state

$$c^t = z^t \circ i^t + c^{t-1} \circ f^t$$

Formel 27 Cell state LSTM

Block output

$$y^t = h^t = \tanh(c^t) \circ o^t \text{ in Abb.7 y benannt}$$

Formel 28 Block output LSTM

W bezeichnet die Gewichtsmatrizen, die auf den aktuellen Input x^t wirken. R sind die sogenannten *recurrent weights*, die den Einfluss des vorherigen Hidden States h^{t-1} modellieren. b steht für die Bias-Vektoren, die pro Gate individuell trainiert werden.

Die Terme $p \circ c^t$ oder auch $p \circ c^{t-1}$ bezeichnen die sogenannten Peephole-Verbindungen, bei denen der vorherige Zellzustand c^{t-1} direkt in die Berechnung der Gates einfließt. Diese Peephole-Gewichte p ermöglichen es den Gates, auf den internen Speicherzustand zuzugreifen, bevor dieser vom Output beeinflusst wird. Dadurch kann die Zellzustandsinformation direkter berücksichtigt werden.

Der Zellzustand c^t selbst speichert die Langzeitinformation der Sequenz. Er wird über die Zeit hinweg additiv aktualisiert und durch das Forget- und Input-Gate kontrolliert. Der Zellzustand ist entscheidend für die Fähigkeit des LSTMs, relevante Informationen über lange Zeiträume hinweg beizubehalten oder gezielt zu vergessen. [9, S. 2-3]

Wie bei klassischen RNNs wird auch bei LSTMs das Training mittels Backpropagation Through Time (BPTT) durchgeführt. Aufgrund der Architektur mit separatem Zellzustand und Gate-Mechanismen fließen die Gradienten in LSTMs jedoch stabiler über viele Zeitschritte. Besonders der additive Pfad im Zellzustand verhindert das Vanishing Gradient Problem weitgehend. [7, Kapitel 2.3.1]

2.5 Hyperparameter

Beim Arbeiten mit RNN ist die Wahl und Anpassung der Hyperparameter entscheidend. Ziel ist es, den Verlust (Loss) des Modells zu minimieren und somit die Modellleistung zu optimieren. Wie in [10, S. 393], beschrieben, kann eine gezielte Hyperparametrisierung die Performance eines neuronalen Netzes signifikant verbessern.

Folgende Hyperparameter sind bei RNNs von zentraler Bedeutung:

- **Lernrate (Learning rate):** Die Lernrate bestimmt, wie stark die Modellgewichte pro Trainingsschritt angepasst werden. Ist sie zu hoch, konvergiert das Modell möglicherweise nicht, ist sie zu niedrig, verläuft das Training langsam. Typische Werte liegen zwischen 0.01 und 0.0001.
- **Anzahl der Hidden Layer und Neuronen:** Die Tiefe des Netzwerks sowie die Anzahl der Neuronen pro Layer beeinflussen direkt die Modellkomplexität. Mehr Neuronen bedeuten mehr Parameter, was nicht zwangsläufig zu besseren Ergebnissen führt. Ein zu komplexes Modell neigt zudem zu Overfitting, was im späteren Verlauf noch genauer erläutert wird.

- **Typ der Neuronen:** Neben klassischen RNN-Zellen existieren erweiterte Zelltypen wie LSTM, GRU oder bidirektionale LSTM (BiLSTM). In diesem Kontext wird ausschließlich LSTM verwendet, da diese Architektur bereits das Vanishing-Gradient-Problem adressiert.
- **Aktivierungsfunktionen:** LSTM-Zellen nutzen verschiedene Aktivierungsfunktionen. Gängige Varianten sind ReLU (Ausgabe zwischen 0 und ∞), tanh (Ausgabe zwischen -1 und 1) sowie Sigmoid (Ausgabe zwischen 0 und 1). Die Wahl der Funktion beeinflusst das Ergebnis des RNN maßgeblich.
- **Dropout Layer:** Dropout-Layer dienen der Reduktion von Overfitting, indem zufällig Neuronen während des Trainings deaktiviert werden. Wie in [11, S.1937] gezeigt, profitieren KNNs signifikant von dieser Methode. Die Dropoutrate legt fest, welcher Anteil der Neuronen pro Schritt deaktiviert wird.
- **Batch Size:** Sie gibt an, wie viele Trainingsbeispiele pro Iteration verarbeitet werden. Größere Batch Sizes führen zu schnelleren Trainingsprozessen, erfordern aber mehr Speicher und können die Modellgeneralität negativ beeinflussen.
- **Sequenzlänge (Sequence Length):** Die Sequenzlänge legt fest, wie viele aufeinanderfolgende Zeitpunkte als Eingabe verwendet werden. Bei einer Sequenzlänge von 30 verarbeitet das RNN 30 Datenpunkte, um eine Vorhersage für den 31. Datenpunkt zu treffen (Many-to-One RNN Architektur). Längere Sequenzen liefern potenziell mehr Kontext, reduzieren jedoch die Anzahl der verfügbaren Trainingssequenzen.

Hyperparameter werden hauptsächlich angepasst, um den Loss zu minimieren. Dabei treten jedoch häufig die Probleme Overfitting und Underfitting auf. Ein niedriger Loss bedeutet nicht zwangsläufig, dass das Modell gut ist. Im Gegenteil: Ein zu niedriger Loss deutet oft auf Overfitting hin. Overfitting bedeutet, dass das Modell die Trainingsdaten „auswendig“ gelernt hat und nicht generalisiert. Das führt dazu, dass der Loss auf den Trainingsdaten sehr niedrig ist, auf unbekannten Daten jedoch sehr hoch, da das Modell mit neuen Daten nicht umgehen kann. Andererseits weist ein hoher Loss nach dem Training darauf hin, dass das Modell noch nicht ausreichend generalisiert und wahrscheinlich mehr Trainingszeit benötigt.

3 Grundlagen Finanzanalyse

3.1 Grundlage Börse

Eine Börse ist ein Ort, an dem sich Marktteilnehmer treffen, um Wertpapiere zu handeln. Dort werden Gegenstände wie Aktien, Anleihen oder Fonds gehandelt. Es gibt aber auch spezielle Börsen, auf denen ausschließlich Kryptowerte gehandelt werden. Ein Beispiel für eine klassische Börse, auf der Wertpapiere gehandelt werden, ist Xetra. Ein Beispiel für eine Kryptobörse, auf der Kryptowerte gehandelt werden, ist Coinbase.

An elektronischen Börsen erfolgt die Preisbildung eines Wertpapiers oder Kryptowerts durch das Zusammenwirken von Angebot und Nachfrage, wobei sämtliche Kauf- und Verkaufsabsichten im sogenannten Limit Order Book (LOB) erfasst werden. Dabei stellt jede Order die Absicht eines Marktteilnehmers dar, eine bestimmte Menge eines Vermögenswertes zu einem definierten Preis zu kaufen oder zu verkaufen. Das LOB fungiert als zentrale Datenstruktur, in der alle aktiven Limit Orders nach Preis und Zeit priorisiert gespeichert sind. Der Marktpreis entsteht durch den Abgleich von Orders mit komplementären Bedingungen, das heißt, wenn sich ein Kauf- und ein Verkaufsauftrag in Preis und Menge decken, kommt es zu einer Transaktion. Der zuletzt ausgeführte Handelspreis gilt als aktueller Kurs. Die besten sichtbaren Kauf- (Bid) und Verkaufsangebote (Ask) spiegeln die derzeitige Marktliquidität wider und bilden die Grundlage für die Preisfindung. Somit ist die Preisbildung ein selbstorganisierter Prozess, der durch das Einstellen, Ausführen und Stornieren von Orders bestimmt wird [12, S. 3-4]

Als Privatperson kann man nicht direkt an der Börse handeln. Dafür braucht man einen Broker, der ein Depotkonto bereitstellt. Über dieses Konto kann man Wertpapiere kaufen und verkaufen. Dabei sollte man beachten, dass bei jedem Handel, also beim Kauf und Verkauf, meist Gebühren oder Provisionen an den Broker anfallen.

Außerdem muss man beim Kauf oder Verkauf eines Wertpapiers angeben, welchen Ordertyp man verwenden möchte.

- **Market Order:** Diese Order wird sofort zum besten verfügbaren Marktpreis ausgeführt. Sie garantiert die Ausführung, aber nicht den genauen Preis.
- **Limit Order:** Hier legt man einen Höchstpreis (beim Kauf) oder Mindestpreis (beim Verkauf) fest, zu dem die Order ausgeführt werden darf. Die Ausführung erfolgt nur, wenn der Marktpreis diesen Limitpreis erreicht oder besser ist.

- **Stop Order:** Diese Order wird erst aktiviert, wenn ein bestimmter Kurs (Stop-Preis) erreicht oder überschritten wird. Danach wird sie wie eine Market Order ausgeführt. Sie wird oft genutzt, um Verluste zu begrenzen oder Gewinne abzusichern.
- **Stop-Limit Order**: Diese Order wird ebenfalls bei Erreichen eines Stop-Preises aktiviert, aber anstatt als Market Order ausgeführt zu werden, wird eine Limit Order mit einem vorgegebenen Limitpreis ausgelöst. Die Ausführung erfolgt nur, wenn der Kurs das Limit erreicht oder besser ist. [13, S.76-80]

3.1.1 Aktien

Aktien werden von Unternehmen ausgegeben und stellen Wertpapiere dar, die den Inhaber zum Miteigentümer des Unternehmens machen. Mit dem Besitz von Aktien erwirbt der Aktionär zwei wesentliche Rechte: Erstens ein Mitbestimmungsrecht auf der Hauptversammlung der Aktionäre, bei der wichtige Unternehmensentscheidungen getroffen werden. Zweitens einen Anspruch auf eine Dividende, also auf einen Anteil am Unternehmensgewinn, sofern das Unternehmen beschließt, eine Dividende auszuschütten.

Ein Unternehmen stellt in erste Linie Aktien aus, um Finanzielle Mittel zu erlangen. In Online-Börsen findet man am häufigsten zwei Arten von Aktien, die Stammaktie und die Vorzugsaktie. Eine Stammaktie bietet dem Halter ein Mitbestimmungsrecht und ein Teil des Unternehmensgewinns. Bei einer Vorzugsaktie verzichtet man auf das Mitbestimmungsrecht, bekommt aber dafür eine höhere Dividende.[14, S.49]

Eine Aktie, die zu ersten Mal von einem Unternehmen ausgegeben wird, hat einen Nennwert. Das ist der minimaler Wert der ein zukünftiger Aktionär ausgegeben muss um eine Aktie (vom Unternehmen) zu kaufen. Wie sich der Kurs der Aktie danach entwickelt, hängt von Angebot und Nachfrage der Aktie ab.

3.1.2 Kryptowährungen

Eine neue Form digitalen Geldes sind Kryptowährungen. Diese wurden erstmals von Satoshi Nakamoto in dem Whitepaper ‚Bitcoin: A Peer-to-Peer Electronic Cash System' beschrieben. Dabei werden Transaktionen in einer sogenannten Blockchain gespeichert. Der Vorteil dieses Systems besteht darin, dass keine dritte Partei benötigt wird, um eine Transaktion abzuschließen, anders als im Online-Handel, wo üblicherweise Drittanbieter wie Zahlungsdienstleister zwischengeschaltet sind.[15, S.1]

Kryptowährung haben sich mit der Zeit weiterentwickelt. Aber im Grunde funktionieren alle Kryptowährungen nach dem gleichen Prinzip wie im Whitepaper von Satoshi beschrieben wurde.

Transaktionen zwischen Nutzern werden in einer Blockchain gespeichert. Eine Blockchain ist eine chronologisch verkettete Folge von Blöcken. Am Beispiel von Bitcoin lässt sich dies wie folgt erläutern. Jeder Block enthält folgende Informationen:

- Version
- Vorherige Block Hash
- Merkle Root Hash
- Zeitstempel (Timesteps)
- Schwierigkeitsziel (Difficulty)
- Einmalige Zufallszahl (Nonce)

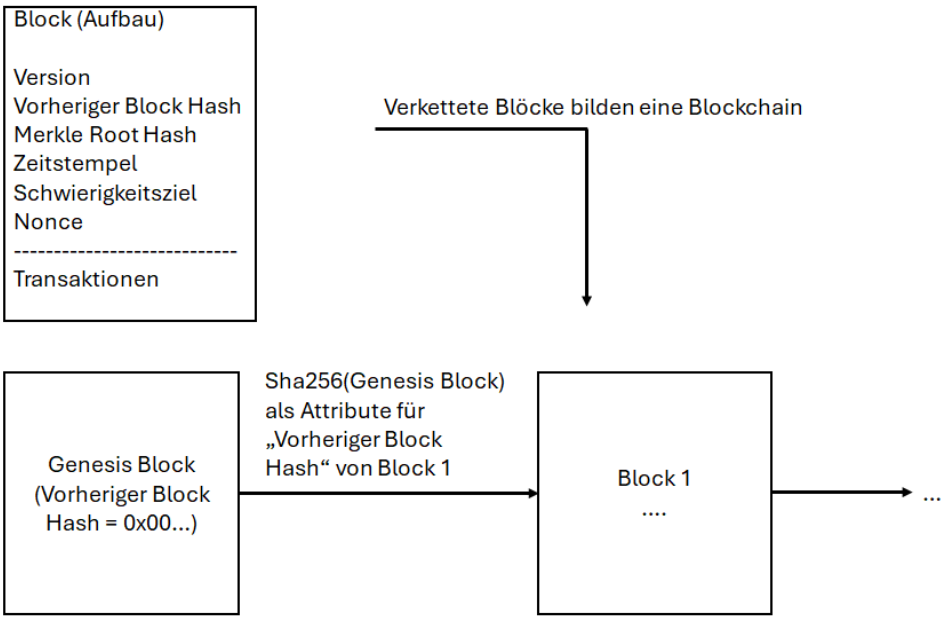

Abbildung 9 Blockchain Aufbau am Beispiel Bitcoin

Der Hash des vorherigen Blocks stellt die Verbindung zur vorherigen Blockgeneration her. Diese Referenz sorgt für die Integrität der Blockchain: Da jeder Block den Hash des vorherigen Blocks enthält, entsteht eine Kette von Blöcken. Wenn ein böswilliger Angreifer versuchen würde, eine alte Transaktion zu verändern, würde sich der Hash dieses alten Blocks ändern und dadurch auch alle nachfolgenden Block-Hashes ungültig werden. So wäre die Manipulation sofort erkennbar.

Die Difficulty gibt an, wie schwer es ist, einen gültigen Block zu finden. Genauer gesagt: Der Hash des neuen Blocks muss unter einem bestimmten Zielwert (Target) liegen, was meist

bedeutet, dass der Hash mit einer bestimmten Anzahl anführenden Nullen beginnt. [16, Kapitel Block Chain]

Um diesen gültigen Hash zu finden, verändern Miner eine spezielle Variable namens Nonce. Die Nonce wird so lange verändert und der Block neu gehasht, bis der resultierende Hash die Difficulty erfüllt (Proof of Work). Das erfordert enorme Rechenleistung und macht Manipulationen praktisch unmöglich (51% Attack ist möglich, wenn ein Teilnehmer 51% Rechenpower im Netzwerk besitzt und somit immer den nächsten Block zuerst findet), da ein Angreifer alle betroffenen Blöcke neu berechnen müsste.

Die Blockchain-Technologie ist in der Regel dezentral organisiert. Das bedeutet, dass das Netzwerk nicht von einer einzelnen Instanz gesteuert wird, sondern viele Teilnehmer gemeinsam über einen Konsensmechanismus das Netzwerk kontrollieren. Die einzelnen Knoten (Nodes) sind weltweit verteilt und jeder Knoten besitzt eine vollständige Kopie der Blockchain. Neue Blöcke, die Informationen über Transaktionen enthalten, werden nur mit Zustimmung der Mehrheit der Knoten dem Netzwerk hinzugefügt. Dadurch können Transaktionen bestätigt werden, ohne dass eine zentrale Vertrauensperson oder Drittpartei erforderlich ist. [15, Kapitel Berechnung]

Um Transaktionen durchzuführen, benötigt ein Marktteilnehmer lediglich eine Adresse. Diese basiert auf einem sogenannten Private Key, der meist durch eine zufällige Schlüsselphrase (beispielsweise 24 zufällige Wörter) erzeugt wird. Aus dem Private Key wird ein Public Key abgeleitet, der nur in eine Richtung berechnet werden kann. Der Private Key ist entscheidend, da nur mit ihm Transaktionen autorisiert werden können. Das bedeutet, nur der Inhaber des Private Keys kann Kryptowährungen an andere Teilnehmer senden.[15, Kapitel Transaktion]

3.2 Technische Analyse

In der Technischen Analyse wird versucht, die zukünftige Preisentwicklung eines Finanzinstruments anhand historischer Marktdaten vorherzusagen. Dabei kommen technische Indikatoren, Chartmuster sowie weitere quantitative Werkzeuge zum Einsatz. John J. Murphy definiert Technische Analyse als „the study of market action [...] to forecast future price trends“. [20, S.1]

Die Technische Analyse basiert auf drei zentralen Grundannahmen [20, S.2]:

1. Der Markt diskontiert alles (market discounts everything). Alle verfügbaren Informationen sind bereits im Preis enthalten.

2. Preise bewegen sich in Trends (price moves in trends). Kursbewegungen folgen bestimmten Mustern, die sich fortsetzen können.
3. Die Geschichte wiederholt sich (history repeats itself). Menschliches Verhalten führt dazu, dass sich Kursmuster über die Zeit ähneln.

Über die Jahre hat sich die technische Analyse weiterentwickelt. Von Chart- und Trendanalysen bis hin zur künstliche Neuronale Netzwerke die interne Zusammenhänge sehen sollen, um die zukünftige Marktentwicklung zu vorherzusagen. [21, Kapitel 1 Introduction]

Die Random-Walk-Theorie und die Effizienzmarkthypothese gehen jedoch davon aus, dass Kursentwicklungen zufällig verlaufen beziehungsweise alle verfügbaren Informationen bereits im Preis enthalten sind, sodass es nicht möglich sein sollte, den Markt systematisch zu schlagen. Kaufman weist darauf hin, dass die Existenz erfolgreicher Handelssysteme und empirische Befunde zeigen, dass insbesondere die Random-Walk-Theorie in ihrer starken Form umstritten ist.[21, Kapitel 1 Random Walk]

3.2.1 Technische Indikatoren

Technische Indikatoren sind mathematische Berechnungen, die auf historischen Kursdaten basieren und den Zweck haben, Trends, Umkehrpunkte oder die Marktstärke zu identifizieren. Sie dienen als Werkzeuge zur Unterstützung von Handelsentscheidungen, indem sie Signale für Ein- und Ausstiege liefern.

Es gibt drei Indikatorentypen:

- Trendfolgende Indikatoren
- Oszillatoren
- Mischindikatoren

Trendfolgende Indikatoren bestätigen Trends und oder erkennen Trends. Sie funktioniert am besten, wenn ein Trend vorherrscht. In Phasen ohne klaren Trend, wie etwa in einer Trading Range, neigen Trendfolgeindikatoren dazu, falsche Trendsignale zu erzeugen. Außerdem sind diese Indikatoren meist verzögert. Bekannte Trendfolgende Indikatoren sind EMA, MACD, ADX, On-Balance-Volume und weitere.

Um eine Trendwende/Wendepunkte zu erkennen werden Oszillatoren eingesetzt. Oszillatoren unterscheiden sich von trendfolgenden Indikatoren, da sie vorauseilende Indikatoren sind. Im Optimalfall zeigen diese Indikatoren bereits die Trendwende an, bevor sie eingetreten ist. Bekannte Oszillatoren sind RSI, Momentum, Elder-Ray und weitere.

Mischindikatoren zeigen wie stark der bullischen (aufwärtstrend) oder der bärische (abwerttrend) ist. Zu denen zählen New-High-New-Low-Index oder auch Put-Call-Ratio.[21, S. 158-159]

3.2.1.1 ADX (Average Directional Index)

ADX (Average Directional Index) misst die Stärke eines Trends, nicht dessen Richtung. Der ADX wird mit einem geglätteten Durchschnitt (EMA) berechnet. Die EMA-Formel lautet:

$$EMA = x_t * K + EMA_{yest} * (1 - K)$$

Wobei gilt: $K = \frac{2}{N}$

x_t ist der zu glättender Wert (z.B. True Range, +DM, −DM oder DX) und N die Anzahl der Perioden (meist wird 14 gewählt). Zur Berechnung des Average Directional Index (ADX) wird zunächst der True Range (TR) ermittelt:

$$TR_t = \max\left(H_t - L_t, |H_t - C_{t-1}|, |L_t - C_{t-1}|\right)$$

Formel 29 True Range

Anschließend werden die positiven und negativen Directional Movements berechnet:

$$+DM = \begin{cases} H_t - H_{t-1}, wenn\ \ H_t - H_{t-1} > L_{t-1} - L_t\ und\ H_t - H_{t-1} > 0 \\ 0\ , sonst \end{cases}$$

Formel 30 Positive Directional Movement

$$-DM = \begin{cases} L_{t-1} - L_t, wenn\ \ L_{t-1} - L_t > H_t - H_{t-1}\ und\ L_{t-1} - L_t > 0 \\ 0\ , sonst \end{cases}$$

Formel 31 Negative Directional Movement

Basierend darauf ergeben sich die Directional Indicators:

$$+DI = \left(\frac{EMA(+DM_t)}{EMA(TR_t)}\right) * 100$$

Formel 32 Positiver Directional Identicator

$$-DI = \left(\frac{EMA(-DM_t)}{EMA(TR_t)}\right) * 100$$

Formel 33 Negativer Directional Identicator

Der Directional Movement Index (DX) und ADX ergibt sich als:

$$DX = \left(\frac{+DI - -DI}{+DI + -DI}\right) * 100$$

Formel 34 Directional Movement Index

$$ADX = EMA(DX)$$

Formel 35 Average Directional Index

Der ADX kann Werte von 0 bis 100 annehmen, wobei Werte nahe 0 auf keinen oder einen sehr schwachen Trend hinweisen, während Werte ab etwa 25 bis 30 einen starken Trend signalisieren, Werte nahe 100 sind sehr selten und stehen für einen extrem starken Trend.[22 S.42-47]

3.2.1.2 RSI (Relative Strength Index)

Der Relative Strength Index (RSI) misst die Dynamik einer Preisbewegung (Momentum) und dient zur Identifikation potenzieller Trendwenden. Er bewegt sich auf einer Skala von 0 bis 100 und hilft dabei, überkaufte oder überverkaufte Marktphasen zu erkennen. Ein RSI-Wert über 70 deutet auf eine überkaufte Marktsituation hin. Wird dieser Wert im Verlauf nach unten durchbrochen, kann dies als Verkaufssignal interpretiert werden. Liegt der RSI hingegen unter 30, gilt der Markt als überverkauft. Ein anschließender Anstieg des RSI über die 30er-Marke wird häufig als Kaufsignal gewertet.

$$RSI = 100 - \frac{100}{1 + RS}$$

Formel 36 Relative Strength Index

$$RS = \frac{Durchschnitt\ der\ netto\ Aufwärts - Veränderung\ in\ einem\ best. Zeitraum}{Durchschnitt\ der\ netto\ Abwärts - Veränderung\ im\ gleichem\ Zeitraum}$$

Formel 37 Relative Strength

In der Regel wird für die Berechnung eine Periodenlänge von 14 gewählt.[21, S.207-215]

3.2.2 Technische Handelssysteme

Die bloße Kenntnis darüber, ob ein Finanzinstrument tendenziell steigen oder fallen wird, reicht für erfolgreiche Handelsentscheidungen nicht aus. Insbesondere beim Handel mit Finanzinstrumenten mit Hebelwirkung (Leverage) kommt dem Zeitpunkt des Einstiegs und des Ausstiegs eine entscheidende Bedeutung zu.

Ein verspäteter Einstieg in eine Bewegung oder ein zu früher Ausstieg kann erhebliche Auswirkungen auf die Rendite und das Risiko eines Trades haben.

Aus diesem Grund wurden sogenannte technische Handelssysteme entwickelt. Diese Systeme zielen darauf ab, auf Basis historischer und aktueller Marktdaten systematisch Handelsentscheidungen zu treffen. Dabei kommen unterschiedliche Methoden der technischen Analyse zum Einsatz, unter anderem:

- Chartmuster-Erkennung
- Mehrzeitfenster-Analysen (Multi-Timeframe-Analyse)
- Technische Indikatoren (z. B. RSI, ADX, gleitende Durchschnitte)
- Volumen- und Preisbewegungen

Technische Handelssysteme verfolgen das Ziel, Einstiegs- und Ausstiegspunkte klar zu definieren und somit subjektive Entscheidungen und emotionale Fehlreaktionen zu vermeiden. [23, Kapitel 22 Selective Trading] Je nach Strategie und Handelsstil können sie entweder diskretionär (manuell durch den Trader interpretiert) oder automatisiert (algorithmisch umgesetzt) ausgeführt werden. Aufgrund ihrer strukturierten Vorgehensweise spielen sie eine bedeutende Rolle im modernen Finanzhandel, insbesondere in volatilen Märkten oder beim Einsatz von Derivaten.

3.2.2.1 Elder Ray Triple Trading System

Das von Alexander Elder entwickelte Triple Screen Trading System analysiert nicht nur eine einzelne Zeitebene, sondern nutzt drei verschiedene Zeitzonen gleichzeitig. Es basiert auf den Grundprinzipien der Dow-Theorie. Diese besagt, dass übergeordnete Zeiteinheiten die übergeordnete Trendrichtung bestimmen, während kürzere Zeiteinheiten kurzfristige Schwankungen abbilden.

Die drei genutzten Zeiteinheiten stehen in einem Faktor-5-Verhältnis zueinander. Wenn beispielsweise der Tageschart als mittlere Zeiteinheit verwendet wird, sollte der erste Screen (langfristige Perspektive) ein Wochenchart sein (also 5-mal länger), und der dritte Screen (kurzfristige Perspektive) ein Stunden- oder 4-Stunden-Chart (also 5-mal kürzer).

Screen 1 Trendbestimmung:

Auf der höchsten Zeiteinheit wird der übergeordnete Trend analysiert. Dafür kommt ein trendfolgender Indikator zum Einsatz, beispielsweise ein gleitender Durchschnitt oder Moving Average Convergence Divergence. Sobald der Trend festgestellt wurde, werden ausschließlich Handelsentscheidungen getroffen, die diesem Trend entsprechen. Das heißt: Bei einem Aufwärtstrend werden nur Kaufpositionen in Betracht gezogen, bei einem Abwärtstrend nur Verkaufspositionen.

Screen 2 Einstiegssignale:

Auf der mittleren Zeiteinheit wird mit Hilfe eines Oszillators (z. B. Stochastic oder RSI) nach konkreten Ein- oder Ausstiegsmöglichkeiten gesucht. Dabei dürfen jedoch nur Signale beachtet werden, die mit der Trendrichtung aus dem ersten Screen übereinstimmen. Gegen den übergeordneten Trend wird nicht gehandelt.

Screen 3 Feintuning des Einstiegs:

Die unterste Zeiteinheit dient nicht der Signalgenerierung im klassischen Sinne. Sie wird zur Feinabstimmung des Einstiegs oder zur Skalierung bestehender Positionen verwendet. Hier kann zum Beispiel mit Buy-Stop- oder Sell-Stop-Orders gearbeitet werden. Zudem lässt sich an dieser Stelle das Risikomanagement konkret umsetzen, etwa durch das Setzen von Stop-Loss-Marken oder durch das Schließen von Teilpositionen.[19, S.305-312]

4 Vorhandene Erkenntnisse

In “Stock Market Prediction via Deep Learning Techniques: A Survey” [21, S.22-25] wird eine Vielzahl bestehender Modelle zur Aktienkursvorhersage aufgeführt. Obwohl RNNs bereits seit den 1990er Jahren bekannt sind, werden sie weiterhin intensiv erforscht und weiterentwickelt.

Hybride Modellarchitekturen zeigen in aktuellen Forschungsarbeiten häufig verbesserte Vorhersageergebnisse. In [22, S. i] wird beispielsweise ein LSTM-Netzwerk mit einem ARIMA-Modell kombiniert: ARIMA modelliert lineare Trends, während LSTM nichtlineare Muster abbildet. Dieser Ansatz verbessert die Prognosegüte im Vergleich zu den Einzelmodellen. Ähnlich zeigt die Kombination von Convolutional Neural Networks (CNNs) mit RNNs Vorteile, da CNNs zur Extraktion lokaler Merkmale genutzt werden, die dann durch RNNs zeitlich interpretiert werden. [23, Kapitel Conclusion]

Ein weiterer Ansatz in [24, Kapitel Conclusion] verwendet ein Dual-Stage Attention-basiertes RNN, das mittels Aufmerksamkeitsmechanismen relevante zeitliche Muster gezielt gewichtet. Dies führt zu einer höheren Modellgenauigkeit bei komplexen Zeitreihen.

Besonders relevant für diese Arbeit ist das Konzept der Multitemporalität. In [25, Kapitel Conclusion] wird ein Verfahren vorgestellt, bei dem Variational Mode Decomposition (VMD) zur Zerlegung von Zeitreihen in verschiedene Frequenzkomponenten eingesetzt wird. Diese werden als multiskalare Features an ein LSTM-Modell übergeben, was sowohl kurzfristige als auch langfristige Muster erfasst. Zusätzlich berücksichtigt ein Temporal Multi-Feature Graph (TMFG) die strukturellen Abhängigkeiten zwischen den Merkmalen. Auch dieses hybride Verfahren zeigt eine verbesserte Prognoseleistung gegenüber klassischen Einzelmodellen.

Trotz vielversprechender numerischer Ergebnisse (z. B. niedriger MSE oder RMSE) ist die praktische Relevanz dieser Ansätze im realen Handel nach wie vor nicht abschließend geklärt. Insbesondere bleibt offen, wie effektiv eine explizite Einbindung multitemporaler Eingabedaten in RNN-Modelle zur Verbesserung der Prognosegüte beiträgt.

5 Forschungslücke und Forschungsfrage

Es gibt zwei zentrale Gründe, warum die Verwendung von RNNs mit multitemporalen Daten zur Vorhersage von Finanzinstrumenten sinnvoll und notwendig ist. Erstens wurde dieses Vorgehen bislang kaum untersucht und sollte daher systematisch getestet werden. Zweitens nutzen Handelssysteme oft mehrere Zeitebenen, um Preise von Finanzinstrumenten zu analysieren und Fehlsignale zu minimieren. Diese Mehrzeitebenen geben einen umfassenderen Überblick über den Markt und sind entscheidend für fundierte Handelsentscheidungen. Daher ist es naheliegend, dass RNNs mit multitemporalen Daten trainiert werden sollten, um deren Prognosegenauigkeit zu verbessern.

In dieser Arbeit wird die Umsetzung der Einbindung der Multi Temporal Daten in das LSTM-RNN am Triple Screen Trading System orientiert sein. Dieses System ist dafür bekannt, durch die Kombination mehrerer Zeitebenen eine zuverlässige Handelssignalgenerierung zu ermöglichen, was sich in der Praxis durch Gewinne bestätigt hat. Daraus lässt sich ableiten, dass eine systematische Struktur hinter diesem Ansatz steht. Ziel dieser Arbeit ist es, diese Struktur dem LSTM-RNN über multitemporale Daten zugänglich zu machen, damit das Modell auf dieser Basis präzisere Vorhersagen treffen kann.

Daraus leitet sich die Forschungsfrage ab:

„Verbessert die Integration multitemporaler technischer Indikatoren in ein LSTM-RNN die Prognosegenauigkeit von Schlusskursen im Vergleich zu Modellen mit ausschließlich tagesbasierten Eingabedaten?“

6 Implementierung

Zur Umsetzung wurde die CRISP-DM-Methode gewählt. Die Datenaufbereitung erfolgte gemäß den darin definierten Schritten, insbesondere in Bezug auf Datenverständnis, Datenvorbereitung und Modellierung. Als Business-Ziel wurde definiert, die Prognosegenauigkeit des LSTM RNN mithilfe von Multi Temporal Daten zu verbessern. Die Modelle werden anhand der Kennzahlen MSE, MAE und RMSE evaluiert und miteinander verglichen.

Die Implementierung und das Training der Modelle wurden unter Ubuntu 24.04 mit Python 3.8 und TensorFlow (GPU-Version) in Jupyter Notebook (Version 2.13.0) durchgeführt. Als Hardware kam ein lokales System mit einer NVIDIA GeForce RTX 5060 Ti 16 GB (CUDA 12.9) sowie 64 GB RAM zum Einsatz.

6.1 Datenerfassung

Die Daten wurden über Yahoo Finance bezogen und im CSV-Format gespeichert, um die verschiedenen Modelle mit identischen Datensätzen zu versorgen. Verwendet wurden tägliche Kursdaten der letzten fünf Jahre, welche bei Bedarf auf Wochenbasis aggregiert wurden. Die verfügbaren Variablen umfassen Schlusskurs (Close), Tageshoch (High), Tagestief (Low), Eröffnungskurs (Open) und Handelsvolumen (Volume). Yahoo Finance gilt als zuverlässige, kostenlose und in der Finanz-Community anerkannte Datenquelle.

Bei einem Tageschart (Daychart) beziehen sich die Kursdaten auf einen Handelstag. Der Schlusskurs (Close) ist der letzte gehandelte Preis am Ende des Börsentages. Der Eröffnungskurs (Open) ist der erste Preis zu Beginn des Handelstages. High und Low bezeichnen den höchsten bzw. niedrigsten Kurs, der im Verlauf des Tages erreicht wurde. Volume gibt die Anzahl der gehandelten Einheiten während des Tages an und liefert Hinweise auf die Marktaktivität.

Die ausgewählten Ticker sind BTC-USD, ETH-USD, DOGE-USD, AAPL, MSTR, PFE, ^GSPC und ^GDAXI.

Die ersten drei Ticker sind Kryptowährungen. Bitcoin (BTC) ist die erste und wertvollste Kryptowährung. Ethereum (ETH) ist die erste Kryptowährung, die Smart Contracts implementiert hat, entwickelt von Vitalik Buterin. Dogecoin (DOGE) ist ein sogenannter Meme Coin, der keinen fundamentalen Nutzen besitzt und dessen Wert hauptsächlich durch spekulative Aktivitäten und Einfluss der Community bestimmt wird.

Die nächsten drei Ticker sind Aktien. Apple Inc. ist ein führendes Technologieunternehmen und Hersteller der iPhone-Produktreihe. MicroStrategy Inc (MSTR) ist ein Softwarehersteller, dessen Aktien mittlerweile unter anderem genutzt werden, um spekulativ auf den Bitcoin-Kurs zu setzen. Pfizer (PFE) ist ein Pharmaunternehmen, dessen Aktienverlauf stabil ist (im Vergleich zu den genannten Kryptowährungen und Aktien).

^GSPC steht für den S&P 500 Index, der die 500 größten börsennotierten US-amerikanischen Unternehmen umfasst. ^GDAXI bezeichnet den DAX, welcher die 40 größten deutschen Aktiengesellschaften beinhaltet.

6.2 Datenvorbereitung

Zunächst wurden die Datenpunkte in den Datensätzen in das Format *float* konvertiert. Da drei verschiedene Modelle untersucht werden, erfolgte die Datenvorbereitung jeweils unterschiedlich:

- **Modell A (Basismodell):** Eingabedaten sind alle verfügbaren Tagesdaten des betrachteten Zeitraums.
- **Modell B (ebenfalls Basismodell):** Verwendet die Tagesdaten und die Indikatoren ADX und RSI, basierend auf Tagesdaten. Diese Indikatoren wurden gewählt, da sie sich gut für die Skalierung mittels Min-Max eignen.
- **Modell C:** Die Tagesdaten wurden zunächst auf Wochenbasis aggregiert. Die Aggregation erfolgte über eine 7-Tage-Sample-Methode, wobei für alle Variablen außer *Volume* der Mittelwert berechnet wurde, das *Volume* wurde aufsummiert. Auf Wochenbasis wurde der ADX als trendfolgender Indikator berechnet, während für die Tagesdaten der RSI-Oszillator berechnet wurde. Anschließend wurde der wöchentliche ADX auf die Tagesdaten übertragen (Broadcast) wie in Abbildung 10 dargestellt

Date	Open	High	Low	Close	Volume	rsi	adx_w	plus_di_w	minus_di_w
2021-01-14	37325.109375	39966.406250	36868.562500	39187.328125	6.361599e+10	66.338566	68.179769	72.804560	3.520613
2021-01-15	39156.707031	39577.710938	34659.589844	36825.367188	6.776076e+10	57.834579	68.179769	72.804560	3.520613
2021-01-16	36821.648438	37864.367188	35633.554688	36178.140625	5.770619e+10	55.726502	68.179769	72.804560	3.520613
2021-01-17	36163.648438	36722.351562	34069.320312	35791.277344	5.235985e+10	54.448962	68.179769	72.804560	3.520613
2021-01-18	35792.238281	37299.285156	34883.843750	36630.074219	4.951170e+10	56.763413	68.391612	62.851266	10.596462

Abbildung 10 Modell C Features am Beispiel Ticker BTC-USD

Der Zielwert (Target) entspricht dem Schlusskurs (Close) der jeweils folgenden Zeiteinheit. Das bedeutet, dass das Modell für Eingabedaten zum Zeitpunkt t den Schlusskurs zum Zeitpunkt t+1 vorhersagen soll. Die letzte Zeile wurde entfernt, da für sie kein Zielwert existiert.

Die Daten wurden zunächst im Verhältnis 80 % zu 20 % in Trainings- und Testdaten aufgeteilt. Anschließend wurden 20 % des Trainingsdatensatzes für die Validierung verwendet.

Zur Vermeidung von Data Leakage wird die Min-Max-Skalierung ausschließlich anhand der Trainingsdaten berechnet und anschließend auf die Testdaten angewandt. Dabei wird der Wertebereich auf [0, 1] normiert, wodurch sich folgende Formel ergibt:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Formel 38 Min-Max Scaler [26, Kapitel 2 Related Work]

Die Umsetzung der Datenvorbereitung erfolgt mit den Bibliotheken pandas, numpy, scikit-learn und ta.

6.3 Modellierung

Durch den Einsatz des Keras Tuners wird für jeden Ticker ein spezifisches RNN-Modell erstellt. Das bedeutet, dass jede Modellvariante (A, B, C) jeweils acht individuelle LSTM-RNN-Modelle umfasst. Das allgemeine Schema eines solchen Modells ist in Abbildung 11 dargestellt, die verwendeten Hyperparameter sind in Tabelle 2 aufgeführt.

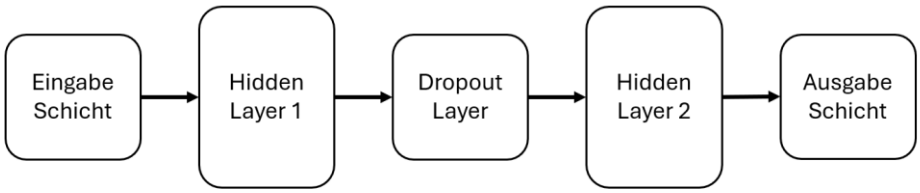

Abbildung 11 Schematischer Aufbau der RNNs

Hyperparameter	Modell A	Modell B	Modell C
Input Layer (Number Neurons)	5	9	9
Hidden Layer 1	32-640	32-640	32-640
Dropout Layer 1	0.2-0.5	0.2-0.5	0.2-0.5
Hidden Layer 2	32-128	32-128	32-128
Learning rate	0.01-0.0001	0.01-0.0001	0.01-0.0001
Batch size	16; 32; 64	16; 32; 64	16; 32; 64
Sequence length	3	3	3

Tabelle 1 Modell Aufbau und Hyperparameter

Für das Training der Modelle wurde Early-Stopping mit einer Patience von 2 verwendet, um Overfitting zu vermeiden. Zur Auswahl der besten Modellarchitektur pro Ticker kam der Keras Tuner zum Einsatz. Pro Ticker wurden maximal 15 Trials durchgeführt, das Modell mit dem besten Loss wurde übernommen.

Die Sequenzlänge (Sequence Length) beschreibt, wie viele vergangene Datenpunkte als Eingabesequenz genutzt werden, um den nächsten Wert vorherzusagen. Anfangs wurden Modelle mit einer Sequenzlänge von 30 trainiert. Bereits nach kurzer Zeit zeigte sich jedoch, dass kürzere Sequenzen deutlich bessere Ergebnisse im Hinblick auf den Loss lieferten. Daher wurde die Sequenzlänge im weiteren Verlauf dieser Arbeit auf 3 festgelegt.

Der vollständige Code dieser Arbeit ist beigefügt und dokumentiert alle durchgeführten Schritte reproduzierbar.

6.4 Evaluationsmetriken

Zur Bewertung der Modellgüte werden drei standardisierte Fehlermaße verwendet:

- Mean Squared Error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Formel 39 Mean Squared Error

Der MSE misst die durchschnittliche quadratische Abweichung zwischen tatsächlichen Werten y_i und Vorhersagen $\hat{y}_i$. Große Fehler werden dabei stärker gewichtet.

- Root Mean Squared Error:

$$\mathrm{R}MSE = \sqrt{MSE}$$

Formel 40 Root Mean Squared Error

Der RMSE ist die Quadratwurzel des MSE und besitzt dadurch die gleiche Einheit wie die Zielvariable, was die Interpretation erleichtert.

- Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\|y_i - \hat{y}_i\|^2$$

Formel 41 Mean Absolute Error

Der MAE gibt die durchschnittliche absolute Abweichung an und ist weniger empfindlich gegenüber Ausreißern als der MSE.

6.5 Evaluation

Die Tabellen 2 bis 4 zeigen die Ergebnisse der Fehlermaße RMSE, MSE und MAE für die drei Modelle A, B und C über verschiedene Tickersymbole. Die Fehlerwerte ermöglichen einen quantitativen Vergleich der Modellleistungen im Hinblick auf ihre Prognosegüte.

Modell C erreichte bei BTC-USD und ^GDAXI die niedrigsten Fehlerwerte über mehrere Metriken hinweg und zeigte damit in diesen Fällen die beste Prognoseleistung. Modell B schnitt bei ETH-USD, PFE und ^GSPC am besten ab, während Modell A bei AAPL und MSTR überzeugte. Für DOGE-USD lieferten alle drei Modelle identische Fehlerwerte, was auf eine geringe Komplexität oder Vorhersagbarkeit der zugrunde liegenden Kursbewegung hindeuten könnte.

Insgesamt lässt sich keine eindeutige Überlegenheit eines einzelnen Modells feststellen. Modell C lieferte in einigen Fällen deutliche Verbesserungen, Modell A zeigte in mehreren Szenarien robuste Grundperformance, und Modell B konnte bei einzelnen Tickersymbolen mit Zusatzfeatures leichte Vorteile erzielen.

RMSE:

Ticker	Modell A	Modell B	Modell C	Bestes Modell
BTC-USD	6071.67	8134.83	**3323.10**	C
ETH-USD	132.88	**124.68**	153.06	B
DOGE-USD	0.02	0.02	0.02	ABC
AAPL	**9.99**	14.34	14.92	A
MSTR	**37.42**	66.74	47.16	A
PFE	0.50	**0.49**	0.71	B
^GSPC	102.11	**96.99**	104.23	B
^GDAXI	1103.69	691.73	**674.99**	C

Tabelle 2 RMSE der Modelle A, B und C pro Ticker

MSE:

Ticker	Modell A	Modell B	Modell C	Bestes Modell
BTC-USD	366865204.29	66175426.22	**11043016.51**	C
ETH-USD	17656.99	**15546.32**	23427.80	B
DOGE-USD	0.00	0.00	0.00	ABC
AAPL	**99.89**	205.66	222.54	A
MSTR	**1400.41**	4453.97	2223.69	A
PFE	0.25	**0.24**	0.50	B
^GSPC	10425.89	**9407.16**	10864.74	B
^GDAXI	1218122.97	478490.37	**455617.41**	C

Tabelle 3 MSE der Modelle A, B und C pro Ticker

MAE:

Ticker	Modell A	Modell B	Modell C	Bestes Modell
BTC-USD	4906.93	6609.36	**2599.42**	C
ETH-USD	96.43	**92.58**	115.74	B
DOGE-USD	0.01	0.01	0.01	ABC
AAPL	**8.50**	13.10	13.65	A
MSTR	**29.09**	54.21	38.22	A
PFE	**0.37**	0.38	0.58	A
^GSPC	78.20	**72.20**	81.51	B
^GDAXI	926.91	**550.41**	553.60	B

Tabelle 4 MAE der Modelle A, B und C pro Ticker

Die grafischen Abbildung 12-14 zeigen den Vergleich zwischen den tatsächlichen (True) und den prognostizierten (Predicted) Werten der Modelle für ausgewählte Tickersymbole. Zur besseren Lesbarkeit werden pro Modell jeweils zwei Grafiken in dieser Arbeit angezeigt: Ein Ticker mit niedrigen Loss und ein mit vergleichsweise höheren Loss.

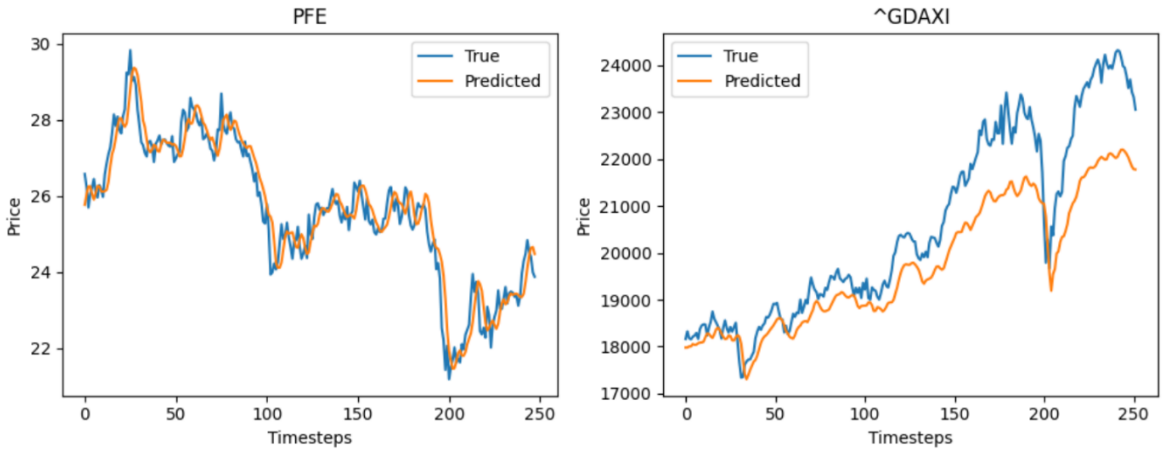

Abbildung 12 Modell A Ticker PFE und GDAXI (DAX)

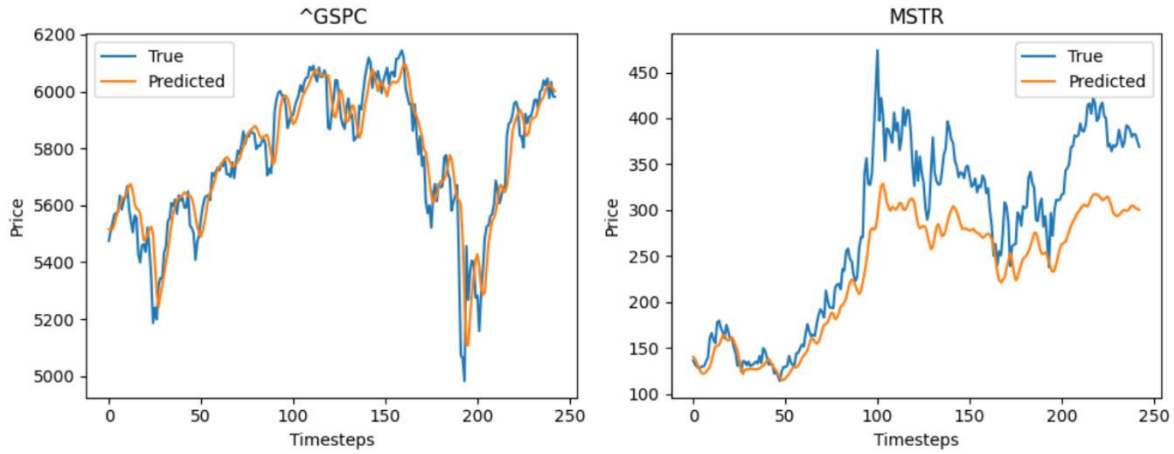

Abbildung 13 Modell B Ticker GSPC (S und P 500) und MSTR

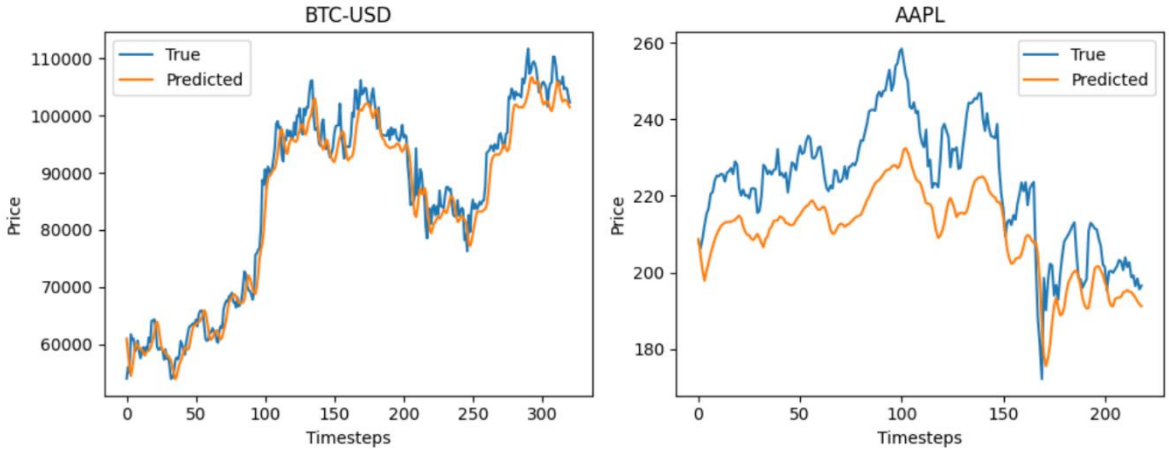

Abbildung 14 Modell C Ticker BTC-USD und AAPL

6.6 Diskussion

Nach der Evaluation zeigte sich, dass Modell C, welches zusätzlich wöchentliche ADX-Werte (ein Trenderkennungsindikator) einbezog, keine signifikant bessere Prognoseleistung erzielte als die beiden Basismodelle. Modell A, das ausschließlich Rohdaten (Close, High, Low, Open, Volume) auf Tagesbasis verwendete, erreichte auf mehreren Tickersets sogar die beste MSE und RMSE. Auch Modell B, das zusätzlich zwei Indikatoren integrierte, schnitt vereinzelt besser ab.

Auffällig ist, dass bei DOGE-USD alle Modelle Fehlerwerte von 0 ausweisen. Dies bedeutet nicht, dass der Fehler tatsächlich exakt null ist, sondern dass er so gering ist, dass er bei einer Darstellung auf zwei Dezimalstellen als Null erscheint. Da der DOGE-Kurs historisch nie über einen US-Dollar gestiegen ist, fallen die absoluten Fehler im Vergleich zu anderen Tickersymbolen entsprechend niedrig aus.

Diese Beobachtung legen nahe, dass die reinen Tagesdaten bereits ausreichend Information enthalten, um den nächsten Schlusskurs t+1 zuverlässig vorherzusagen. Es spricht auch dafür, dass zusätzliche Indikatoren, insbesondere der auf Wochenbasis berechnete ADX, nicht zur Modellverbesserung beitragen konnten.

Ein zentrales Problem zeigt sich allerdings unabhängig vom Modell: Alle Modelle neigen dazu, dem realen Kursverlauf zeitlich hinterherzuhinken. Insbesondere bei Trendwechseln reagieren sie verspätet, was in der Praxis fatale Folgen hätte, zum Beispiel etwa das Verpassen von Kauf- oder Verkaufssignalen. Obwohl der Loss-Wert dabei formal niedrig bleiben kann, ist der praktische Nutzen im aktiven Handel dadurch stark eingeschränkt.

Ein möglicher Erklärungsansatz ist, dass das RNN primär Muster lernt, die eine nächste Vorhersage möglichst nah am aktuellen Kurs ermöglichen, ohne dabei tiefere Trends oder strukturelle Wechsel zu erkennen. Das Modell extrapoliert kurzfristige Muster, statt tatsächlich zukünftige Entwicklungen abzuleiten.

Zur Einbindung zusätzlicher Daten: Es besteht die Möglichkeit, dass zusätzliche Features wie der ADX von Modell C gar nicht stark gewichtet oder gar ignoriert werden. Eine Studie von Rajkomar et al.[27] zeigte, dass RNNs in der Lage sind, irrelevante oder redundante Informationen größtenteils zu ignorieren, ohne dass die Performance signifikant leidet. Im Umkehrschluss kann dies auch bedeuten, dass relevante, aber schlecht eingebundene Features ebenfalls keinen Nutzen bringen.

Ein technischer Schwachpunkt von Modell C ist die Implementierung der Multi-Temporalität: Die wöchentlichen ADX-Werte wurden über Broadcasting auf tägliche Daten repliziert, was faktisch dazu führt, dass derselbe ADX-Wert siebenmal wiederholt wird. Da RNNs Zeitabstände nicht explizit modellieren, sondern alle Schritte als gleichwertig betrachten, kann dies zu einer fehlerhaften Interpretation der ADX-Wochen-Daten führen. Das Modell behandelt Wochenwerte wie Tageswerte und verliert damit deren eigentlichen zeitlichen Kontext.

Eine Limitation dieser Arbeit besteht darin, dass die Bewertung ausschließlich anhand der Loss-Funktionen (MSE) erfolgt. Diese Kennzahlen spiegeln nicht zwangsläufig das insgesamt beste Modell wider, insbesondere im Hinblick auf praktische Anwendbarkeit. Zudem basiert das Hyperparameter-Tuning mit Keras Tuner darauf, für jeden Ticker ein optimales Modell zu finden. Allerdings hängt der Erfolg dabei stark von der Anzahl der Trials und einem gewissen Zufall ab, da bereits kleine Änderungen der Hyperparameter das Modellverhalten beeinflussen.

7 Zusammenfassung

Ziel dieser Arbeit war es, zu untersuchen, ob ein LSTM-RNN durch den Einsatz von Multi-Temporal-Daten eine verbesserte Prognoseleistung bei der Vorhersage von Schlusskursen verschiedener Finanzinstrumente erzielen kann. Hierfür wurden drei Modellvarianten entwickelt:

- Modell A als Basismodell mit reinen Tagesdaten
- Modell B mit Tagesdaten und mit zusätzlichen technischen Indikatoren (RSI und ADX)
- Modell C mit Multi-Temporal-Daten (Tagesdaten + täglicher RSI + wöchentlicher ADX)

Die Evaluation anhand der Metriken MSE, RMSE und MAE zeigte, dass kein Modell durchgehend überlegen war. Modell A erreichte in den meisten Fällen die besten Fehlerwerte. Modell B schnitt vereinzelt besser ab, insbesondere bei stabilen Kursverläufen. Modell C zeigte bei stark trendbasierten Kursen (z. B. BTC-USD) leichte Vorteile, jedoch keine konsistente Überlegenheit.

Damit kann die Forschungsfrage beantwortet werden. Die Integration von Multi-Temporal-Daten in Form eines wöchentlichen ADX bringt keine signifikante Verbesserung der Vorhersagegenauigkeit. Die Ergebnisse deuten vielmehr darauf hin, dass die Qualität und Einbindung zusätzlicher Features entscheidend ist, nicht deren bloße Existenz.

Ein zentrales Problem aller Modelle ist das zeitverzögerte Reagieren auf Trendwechsel, was im praktischen Handel kritisch ist. Das weist auf eine Limitierung reiner LSTM-RNN-Modelle hin, die vorrangig kurzfristige Muster erkennen, aber strukturelle Marktveränderungen nicht adäquat antizipieren können.

8 Ausblick

In dieser Arbeit wurde ein einzelnes LSTM-Modell pro Ticker zur Kursvorhersage eingesetzt. Dabei zeigte sich, dass die Prognosefähigkeit reiner RNN-Architekturen begrenzt ist, insbesondere bei plötzlichen Trendwechseln. Wie aktuelle Studien belegen (z. B. Choi [22] mit einem ARIMA-LSTM-Hybrid oder Mehtab & Sen [23] mit CNN-LSTM), können hybride Modelle durch die Kombination komplementärer Methoden deutlich bessere Ergebnisse erzielen.

Ein naheliegender nächster Schritt besteht darin, ein hybrides Modell zu entwickeln, das nicht nur unterschiedliche Modelltypen integriert, sondern explizit auf multi-temporalen Daten basiert. Während Qin et al. [24] mit Attention-Mechanismen und Zhang et al. [25] mit spektraler Feature-Zerlegung bereits zeitliche Strukturen differenziert gewichten, bleibt die explizite Kombination mehrerer Zeitskalen (z. B. Minuten, Stunde, Tag) innerhalb eines kohärenten Modells bislang unterrepräsentiert.

Ein solches erweitertes Modell könnte etwa aus mehreren Modulen bestehen, die jeweils auf unterschiedlichen Zeitebenen operieren: CNNs zur Erkennung lokaler Muster und Trendkanäle, Random-Forest-Modelle zur Identifikation von Marktphasen (z. B. basierend auf Elliott-Wellen), sowie klassische LSTM- oder Attention-basierte RNNs zur Modellierung zeitlicher Abhängigkeiten.

Die Teilergebnisse dieser Komponenten würden in einer Fusionsschicht zusammengeführt, um eine Vorhersage zu erzeugen. Dadurch könnten Signale aus verschiedenen Zeitebenen konsistent und dynamisch verarbeitet werden. Dadurch lässt sich das Entscheidungsverhalten auf Finanzmärkten realitätsnäher modellieren.

Im Vergleich zum hier eingesetzten LSTM-RNN-Einzelmodell stellt dieser Ansatz eine deutlich vielversprechendere Richtung dar, sowohl in Bezug auf die Prognosequalität als auch auf die praktische Anwendbarkeit im aktiven Handel.

9 Literaturverzeichnis

[1] Hevner, March, Park, und Ram, „Design Science in Information Systems Research“, *MIS Q.*, Bd. 28, Nr. 1, S. 75, 2004, doi: 10.2307/25148625.

[2] P. Chapman, „CRISP-DM 1.0: Step-by-step data mining guide“, 2000. Zugegriffen: 28. Juni 2025. [Online]. Verfügbar unter: https://www.semanticscholar.org/paper/CRISP-DM-1.0%3A-Step-by-step-data-mining-guide-Chapman/54bad20bbc7938991bf34f86dde0babfbd2d5a72

[3] W. Ertel, *Grundkurs Künstliche Intelligenz*, 6. Springer Vieweg.

[4] Ian Goodfellow, Yoshua Bengio, und Aaron Courvilla, *Deep Learning*. MIT Press, 2016. [Online]. Verfügbar unter: https://www.deeplearningbook.org/

[5] R. S. Sutton und A. G. Barto, „Reinforcement Learning: An Introduction“.

[6] M. A. Nielsen, „Neural Networks and Deep Learning“, 2015, Zugegriffen: 1. Juli 2025. [Online]. Verfügbar unter: http://neuralnetworksanddeeplearning.com/chap2.html

[7] S. Das, A. Tariq, T. Santos, S. S. Kantareddy, und I. Banerjee, „Recurrent Neural Networks (RNNs): Architectures, Training Tricks, and Introduction to Influential Research“, in *Machine Learning for Brain Disorders*, Bd. 197, O. Colliot, Hrsg., in Neuromethods, vol. 197. , New York, NY: Springer US, 2023, S. 117–138. doi: 10.1007/978-1-0716-3195-9_4.

[8] B. Ghojogh und A. Ghodsi, „Recurrent Neural Networks and Long Short-Term Memory Networks: Tutorial and Survey“, 22. April 2023, *arXiv*: arXiv:2304.11461. doi: 10.48550/arXiv.2304.11461.

[9] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, und J. Schmidhuber, „LSTM: A Search Space Odyssey“, *IEEE Trans. Neural Netw. Learn. Syst.*, Bd. 28, Nr. 10, S. 2222–2232, Okt. 2017, doi: 10.1109/TNNLS.2016.2582924.

[10] J. A Ilemobayo *u. a.*, „Hyperparameter Tuning in Machine Learning: A Comprehensive Review“, *J. Eng. Res. Rep.*, Bd. 26, Nr. 6, S. 388–395, Juni 2024, doi: 10.9734/jerr/2024/v26i61188.

[11]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, und R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“, *J. Mach. Learn. Res.*, Bd. 15, Nr. 56, S. 1929–1958, 2014.

[12] A. Briola, S. Bartolucci, und T. Aste, „Deep Limit Order Book Forecasting“, 4. Juni 2024, *arXiv*: arXiv:2403.09267. doi: 10.48550/arXiv.2403.09267.

[13] R. Schittler und M. Michalky, *Das große Buch der Börse: investieren für jedermann, fundamentale Analyse für jedermann, technische Analyse für jedermann, Derivate für jedermann, Trading für jedermann*, 1. Aufl. München: FinanzBuch-Verl, 2008.

[14] H. Sperber, *Finanzmärkte: Eine praxisorientierte Einführung*, 2. Auflage 2020. Freiburg: Schäffer-Poeschel Verlag für Wirtschaft Steuern Recht GmbH, 2020.

[15] S. Nakamoto, „Bitcoin: Ein elektronisches Peer-to-Peer-Cash-System“.

[16] „Block Chain — Bitcoin“. Zugegriffen: 22. Mai 2025. [Online]. Verfügbar unter: https://developer.bitcoin.org/reference/block_chain.html

[17] J. J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*, 2nd ed. East Rutherford: Penguin Publishing Group, 1999.

[18] P. J. Kaufman, *Trading systems and methods*, 5th ed. in Wiley trading series. Hoboken, N.J: Wiley, 2013.

[19] A. Elder und A. Elder, *Die Formel für Ihren Börsenerfolg: Strategien, Money-Management, Psychologie. workbook*, 2. Aufl. München: Finanzbuch-Verl, 1999.

[20] J. W. Wilder, *New concepts in technical trading systems*. Greensboro, N.C: Trend Research, 1978.

[21] J. Zou *u. a.*, „Stock Market Prediction via Deep Learning Techniques: A Survey“, 2022, *arXiv*. doi: 10.48550/ARXIV.2212.12717.
[22] H. K. Choi, „Stock Price Correlation Coefficient Prediction with ARIMA-LSTM Hybrid Model“, 2018, *arXiv*. doi: 10.48550/ARXIV.1808.01560.
[23] S. Mehtab und J. Sen, „Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models“, 2020, doi: 10.48550/ARXIV.2010.13891.
[24] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, und G. Cottrell, „A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction“, 2017, *arXiv*. doi: 10.48550/ARXIV.1704.02971.
[25] Z. Zhang, Q. Liu, Y. Hu, und H. Liu, „Multi-feature stock price prediction by LSTM networks based on VMD and TMFG“, *J. Big Data*, Bd. 12, Nr. 1, S. 74, März 2025, doi: 10.1186/s40537-025-01127-4.
[26] S. G. K. Patro und K. K. Sahu, „Normalization: A Preprocessing Stage“, 2015, *arXiv*. doi: 10.48550/ARXIV.1503.06462.
[27] E. Laksana, M. Aczon, L. Ho, C. Carlin, D. Ledbetter, und R. Wetzel, „The impact of extraneous features on the performance of recurrent neural network models in clinical tasks“, *J. Biomed. Inform.*, Bd. 102, S. 103351, Feb. 2020, doi: 10.1016/j.jbi.2019.103351.

10 Anhang

- Ähnlichkeitsbericht „turnitin“
- Ticker-Datensätze als CSV-Datei
- Python Code in Jupyter-Notebook

11 Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel

„Verbessung der Prognosegenauigkeit von Kursentwicklungen bei Finanzinstrumente durch die Integration von Multi-Temporal-Daten und technischen Indikatoren auf Basis von Recurrent Neural Networks (RNNs)“

selbstständig und ohne fremde Hilfe angefertigt habe. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Alle wörtlich oder sinngemäß übernommenen Stellen aus veröffentlichten oder unveröffentlichten Quellen sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mir ist bekannt, dass ein Verstoß gegen diese Erklärung schwerwiegende Konsequenzen nach sich ziehen kann.

04.07.2025, Munderkingen

Valentino Ciavarella