Summary Report

Machine Learning for Stock Market Predictions
-
Advanced Analysis of Financial Time Series

Louis SIMON

Nossa IYAMU

Table of Contents

1. Introduction

1.1. Motivations

The rapid rise of Artificial Intelligence (AI) in recent years has opened up new perspectives and opportunities, particularly in the field of finance. Cutting-edge technologies such as Deep Learning and Deep Reinforcement Learning offer the potential to analyse stock market signals from a novel angle. The development of these systems aligns perfectly with the scope of our studies and represents an ideal project opportunity for Cassiopée, as well as for our professional careers.

1.2. State of the art

The rapid advancement of Artificial Intelligence (AI) has significantly impacted the financial industry, introducing new methods for analysing and predicting market behaviour. Current state-of-the-art AI techniques in finance include Deep Learning (DL) and Deep Reinforcement Learning (DRL), which are utilized for tasks ranging from stock market predictions to portfolio management. DL, with its ability to process and learn from vast amounts of data, has been effectively employed in predicting stock price movements and enhancing trading strategies. Models such as Deep Neural Networks (DNNs) leverage historical price data and technical indicators to improve the accuracy of predictions. On the other hand, DRL techniques, including Double Deep Q-Networks (DDQNs), focus on decision-making processes by optimizing trading actions to maximize returns. These methods use complex algorithms to analyse patterns and trends, adapting to changing market conditions. Despite the promising results, the application of AI in finance also faces challenges, including overfitting, data quality issues, and the need for substantial computational resources.

1.3. Objectives of the report

For our project, we have implemented a DNN (Deep Learning Neural Network) and a DDQN (Deep Reinforcement Learning) model. This report serves to demonstrate our development methodology, training, and validation processes, as well as to present our results and conduct an introspective analysis of our work. Furthermore, we aim to explore potential perspectives for improvement.

2. First Approach: Deep Neural Network (DNN)

2.1. Research article presentation

The article "Intraday Stock Prediction Based on Deep Neural Network" by Nagaraj Naik and Biju R. Mohan focuses on predicting stock price movements using a Deep Neural Network (DNN). Here is a summary of the main points:

Objective

The study aims to predict intraday stock price movements to enhance trading profitability. The authors propose a method that combines candlestick chart data with technical indicator values to feed a DNN, which then classifies the up and down movements of stock prices.

Methodology

1. Data Used: National Stock Exchange (NSE) of India data from 2008 to 2018.

2. Technical Indicators: Ten technical indicators are used, including SMA, EMA, MOM, STCK, RSI, MACD, and CCI.

3. Data Combination: Candlestick chart data and technical indicators are combined to identify trends.

4. Prediction Model: A five-layer DNN is employed to predict price movements. The performance of the proposed model is compared with an existing three-layer ANN model.

Results

Experimental results indicate that the proposed DNN model outperforms state-of-the-art methods by 8-11% in predicting the up and down movements of stock prices. The stocks tested include Reliance, Infosys, HDFC, and HDFC Bank.

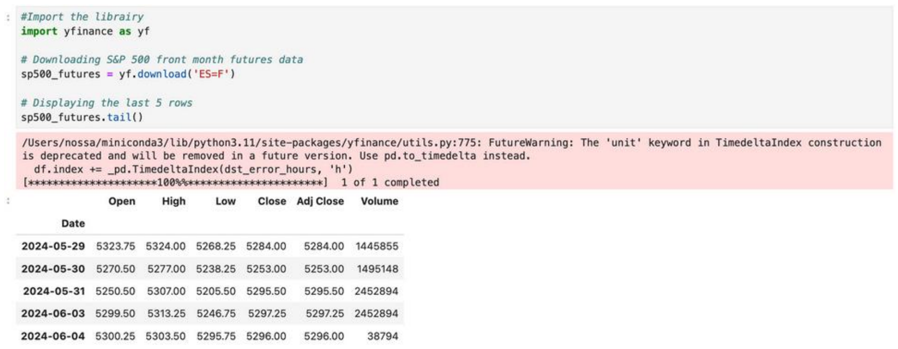
Conclusion

The authors conclude that combining candlestick chart data with technical indicators improves the accuracy of stock price movement predictions. The five-layer DNN model demonstrates superior performance compared to existing three-layer ANN models. They suggest that future work should examine short-term and long-term predictions along with fundamental data analysis, such as price-earnings ratio, book value, and quarterly profits.

In summary, this article presents an innovative approach for predicting stock prices using a deep neural network that integrates technical analysis and candlestick chart data, showing a significant performance improvement over traditional methods.

2.2. Retrieving the data

When delving into stock market analysis, one of the primary tools at our disposal is Yahoo Finance's API. This API provides access to a vast repository of historical stock market data spanning several decades. It enables researchers and traders to retrieve key metrics and characteristics of various stocks, indices, and other financial instruments. This rich source of data serves as the foundation for developing predictive models aimed at forecasting market trends and making informed investment decisions.



```python
#Import the librairy
import yfinance as yf

# Downloading S&P 500 front month futures data
sp500_futures = yf.download('ES=F')

# Displaying the last 5 rows
sp500_futures.tail()
```

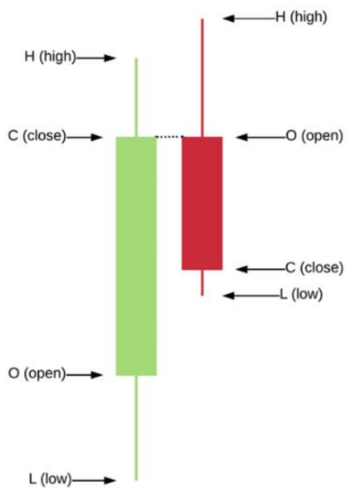
```
/Users/nossa/miniconda3/lib/python3.11/site-packages/yfinance/utils.py:775: FutureWarning: The 'unit' keyword in TimedeltaIndex construction
is deprecated and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
[*********************100%%**********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2024-05-29	5323.75	5324.00	5268.25	5284.00	5284.00	1445855
2024-05-30	5270.50	5277.00	5238.25	5253.00	5253.00	1495148
2024-05-31	5250.50	5307.00	5205.50	5295.50	5295.50	2452894
2024-06-03	5299.50	5313.25	5246.75	5297.25	5297.25	2452894
2024-06-04	5300.25	5303.50	5295.75	5296.00	5296.00	38794

Figure 1: Importation of Yahoo Finance data

These characteristics include:

- Open: The price of the stock at the beginning of the trading day.
- Close: The price of the stock at the end of the trading day.
- High: The highest price reached by the stock during the trading day.
- Low: The lowest price reached by the stock during the trading day.
- Adj Close: The adjusted closing price of the stock, which considers dividends, stock splits, and other corporate actions to reflect the stock's true value over time.
- Volume: The total number of shares traded during the trading day.



These features provide crucial insights into the performance and behaviour of a stock over time.

They are often visualized using candlestick charts, which display the open, high, low, and close prices for a given period. Candlestick charts offer a concise yet comprehensive way to analyse stock price movements and identify patterns that may inform trading decisions.

Figure 2: Candlestick Chart Representation

2.3 Processing the data

It seems natural to use the candlestick feature as input to train our Machine Learning model, indeed, they provide all available information about a stock. However, meddling with these values to create new indicators can be event more efficient.

2.3.1. Indicators analysis

The metrics derived from manipulating stock data provide invaluable insights into market dynamics. These indicators include:

- Simple Moving Average (SMA)
    - Interpretation: SMA smooths out price data to identify trend directions and potential support/resistance levels.
    - Practical Use: It generates buy or sell signals based on price crossing above or below the SMA, and identifies trend reversals through crossovers.
- Exponential Moving Average (EMA)
    - Interpretation: EMA gives more weight to recent prices, making it more responsive to price changes than SMA.
    - Practical Use: Similar to SMA, it helps identify trend directions, support/resistance levels, and potential buy or sell signals.
- Momentum (MOM)
    - Interpretation: MOM measures the speed at which prices change, indicating the strength of a trend.
    - Practical Use: It signals potential buy opportunities when crossing above the zero line and sell opportunities when crossing below, and detects divergence between price and momentum.
- Stochastic Oscillator
    - Interpretation: It identifies overbought and oversold conditions based on %K values.
    - Practical Use: Buy signals occur when %K crosses above 20, and sell signals when %K crosses below 80, with additional signals provided by crossovers.
- Relative Strength Index (RSI)
    - Interpretation: RSI indicates overbought or oversold conditions and potential trend reversals.
    - Practical Use: It generates buy signals when moving above 30 and sell signals when moving below 70, and detects divergence between RSI and price.
- Moving Average Convergence Divergence (MACD)
    - Interpretation: MACD identifies changes in trend strength, direction, and momentum.

    - Practical Use: It generates buy signals when the MACD line crosses above the Signal line and sell signals when it crosses below, while also detecting divergence between MACD and price.

- Williams %R
    - Interpretation: It identifies overbought and oversold conditions based on %R values.
    - Practical Use: Buy signals occur when %R moves from below -80 to above, and sell signals when it moves from above -20 to below, with divergence signalling potential reversals.
- Accumulation/Distribution Line (A/D)
    - Interpretation: A/D line indicates the flow of volume, confirming trends and potential reversals.
    - Practical Use: Rising A/D lines confirm uptrends, falling lines confirm downtrends, and divergence signals potential reversals.
- Commodity Channel Index (CCI)
    - Interpretation: CCI identifies overbought and oversold conditions and potential trend reversals.
    - Practical Use: Buy signals occur when CCI moves from below -100 to above, and sell signals when it moves from above 100 to below, with divergence indicating potential reversals.

These indicators collectively provide a comprehensive analysis of market conditions, including trend direction, momentum, volatility, and volume dynamics. By integrating these metrics, traders can make more informed decisions, reducing the likelihood of false signals and enhancing their ability to predict market trends effectively.

Here is how the metrics are computed:

Figure 3: Mathematic formulas of the metrics

1. Simple Moving Average (SMA)

$$SMA_n = \frac{1}{n}\sum_{i=0}^{n-1} P_{t-i}$$

2. Exponential Moving Average (EMA)

$$EMA_t = P_t \times \left(\frac{2}{n+1}\right) + EMA_{t-1} \times \left(1 - \frac{2}{n+1}\right)$$

3. Momentum (MOM)

$$MOM_n = P_t - P_{t-n}$$

4. Stochastic Oscillator (STCK)

$$K_t = \frac{P_t - L_n}{H_n - L_n} \times 100$$

5. Relative Strength Index (RSI)

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

6. Moving Average Convergence Divergence (MACD)

$$MACD = EMA_{fast} - EMA_{slow}$$

7. Williams %R (R)

$$\%R = \frac{H_n - P_t}{H_n - L_n} \times -100$$

8. Accumulation/Distribution Line (A/D)

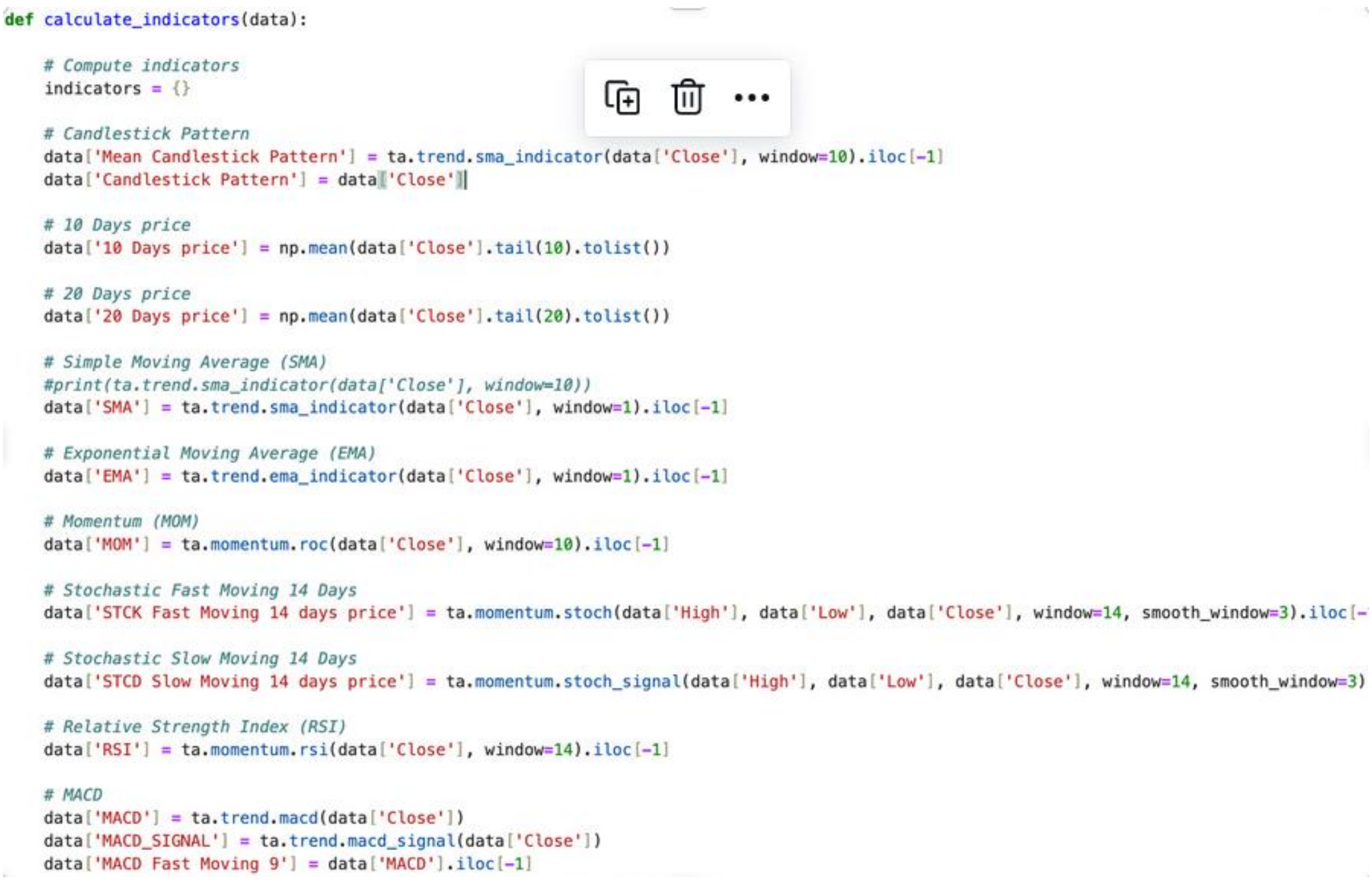
$$A/D_t = A/D_{t-1} + \left(\frac{(P_t - L_t) - (H_t - P_t)}{H_t - L_t} \times V_t\right)$$

9. Commodity Channel Index (CCI)

$$CCI_t = \frac{P_t - SMA_t}{0.015 \times D_t}$$

2.3.2. Processing the data

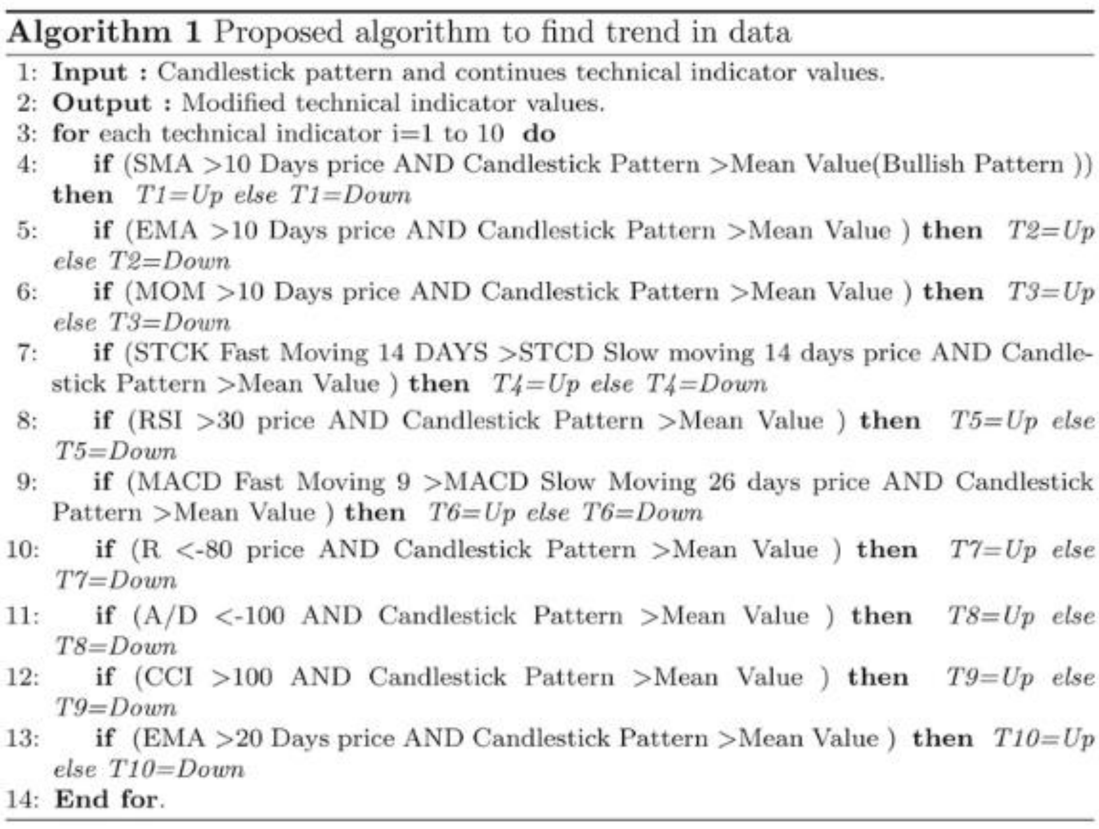
Therefore, to generate input features for our model, we perform transformations on the candlestick values to derive the aforementioned indicators. These transformations allow us to extract additional insights from the raw data, providing a more comprehensive set of features for training our predictive model. By incorporating these indicators, we aim to enhance the model's ability to capture various aspects of market dynamics, including trend direction, momentum, and potential reversal signals.



```
def calculate_indicators(data):

    # Compute indicators
    indicators = {}

    # Candlestick Pattern
    data['Mean Candlestick Pattern'] = ta.trend.sma_indicator(data['Close'], window=10).iloc[-1]
    data['Candlestick Pattern'] = data['Close']

    # 10 Days price
    data['10 Days price'] = np.mean(data['Close'].tail(10).tolist())

    # 20 Days price
    data['20 Days price'] = np.mean(data['Close'].tail(20).tolist())

    # Simple Moving Average (SMA)
    #print(ta.trend.sma_indicator(data['Close'], window=10))
    data['SMA'] = ta.trend.sma_indicator(data['Close'], window=1).iloc[-1]

    # Exponential Moving Average (EMA)
    data['EMA'] = ta.trend.ema_indicator(data['Close'], window=1).iloc[-1]

    # Momentum (MOM)
    data['MOM'] = ta.momentum.roc(data['Close'], window=10).iloc[-1]

    # Stochastic Fast Moving 14 Days
    data['STCK Fast Moving 14 days price'] = ta.momentum.stoch(data['High'], data['Low'], data['Close'], window=14, smooth_window=3).iloc[-

    # Stochastic Slow Moving 14 Days
    data['STCD Slow Moving 14 days price'] = ta.momentum.stoch_signal(data['High'], data['Low'], data['Close'], window=14, smooth_window=3)

    # Relative Strength Index (RSI)
    data['RSI'] = ta.momentum.rsi(data['Close'], window=14).iloc[-1]

    # MACD
    data['MACD'] = ta.trend.macd(data['Close'])
    data['MACD_SIGNAL'] = ta.trend.macd_signal(data['Close'])
    data['MACD Fast Moving 9'] = data['MACD'].iloc[-1]
```

Figure 4: Program to compute the indicators

In the subsequent phase of data processing, we aim to translate the behaviour of these indicators into binary values, where '1' denotes an upward trend and '0' indicates a downward trend. To achieve this, we have programmed the following algorithm:



Algorithm 1 Proposed algorithm to find trend in data

```
1: Input : Candlestick pattern and continues technical indicator values.
2: Output : Modified technical indicator values.
3: for each technical indicator i=1 to 10 do
4:     if (SMA >10 Days price AND Candlestick Pattern >Mean Value(Bullish Pattern )) then T1=Up else T1=Down
5:     if (EMA >10 Days price AND Candlestick Pattern >Mean Value ) then T2=Up else T2=Down
6:     if (MOM >10 Days price AND Candlestick Pattern >Mean Value ) then T3=Up else T3=Down
7:     if (STCK Fast Moving 14 DAYS >STCD Slow moving 14 days price AND Candlestick Pattern >Mean Value ) then T4=Up else T4=Down
8:     if (RSI >30 price AND Candlestick Pattern >Mean Value ) then T5=Up else T5=Down
9:     if (MACD Fast Moving 9 >MACD Slow Moving 26 days price AND Candlestick Pattern >Mean Value ) then T6=Up else T6=Down
10:    if (R <-80 price AND Candlestick Pattern >Mean Value ) then T7=Up else T7=Down
11:    if (A/D <-100 AND Candlestick Pattern >Mean Value ) then T8=Up else T8=Down
12:    if (CCI >100 AND Candlestick Pattern >Mean Value ) then T9=Up else T9=Down
13:    if (EMA >20 Days price AND Candlestick Pattern >Mean Value ) then T10=Up else T10=Down
14: End for.
```

Figure 5: Algorithm to find trend within metrics

With the successful implementation of these algorithms, we are equipped to process all the necessary data. In our study, we have considered every financial day spanning the past 20 years, resulting in a dataset comprising 4980 samples.

2.4. Model development

A primary objective of this study was to gain proficiency in deep learning methodologies. To accomplish this, we developed a neural network from scratch, which is publicly accessible in our repository: [Click here to access the repository].

In the construction of our neural network, we adhered to the following steps:

- Initialization: At the outset, we initialize the parameters of the neural network, including the weights and biases. Proper initialization is crucial for ensuring convergence and preventing issues such as vanishing or exploding gradients.
- Forward Propagation: Once initialized, we proceed with the forward propagation step. This involves passing the input data through the network, layer by layer, while applying activation functions and computing the output of each neuron.

- Cost Computation: With the forward propagation completed, we compute the cost or loss function. This metric quantifies the disparity between the predicted output of the network and the actual target values. The choice of an appropriate cost function depends on the nature of the problem being addressed.
- Backward Propagation: Following the computation of the cost, we employ backward propagation to calculate the gradients of the cost function with respect to the network parameters. This step involves traversing the network in reverse order, propagating the error backward, and computing the gradients using techniques such as the chain rule.
- Weight Update: Armed with the gradients computed during backward propagation, we proceed to update the weights and biases of the network using optimization algorithms such as gradient descent. The goal is to minimize the cost function iteratively by adjusting the parameters in the direction that reduces the error.

By diligently following these steps, we are able to train our neural network from scratch and leverage its capabilities for predictive analysis in the domain of quantitative trading.

Below, a representative diagram of our model:

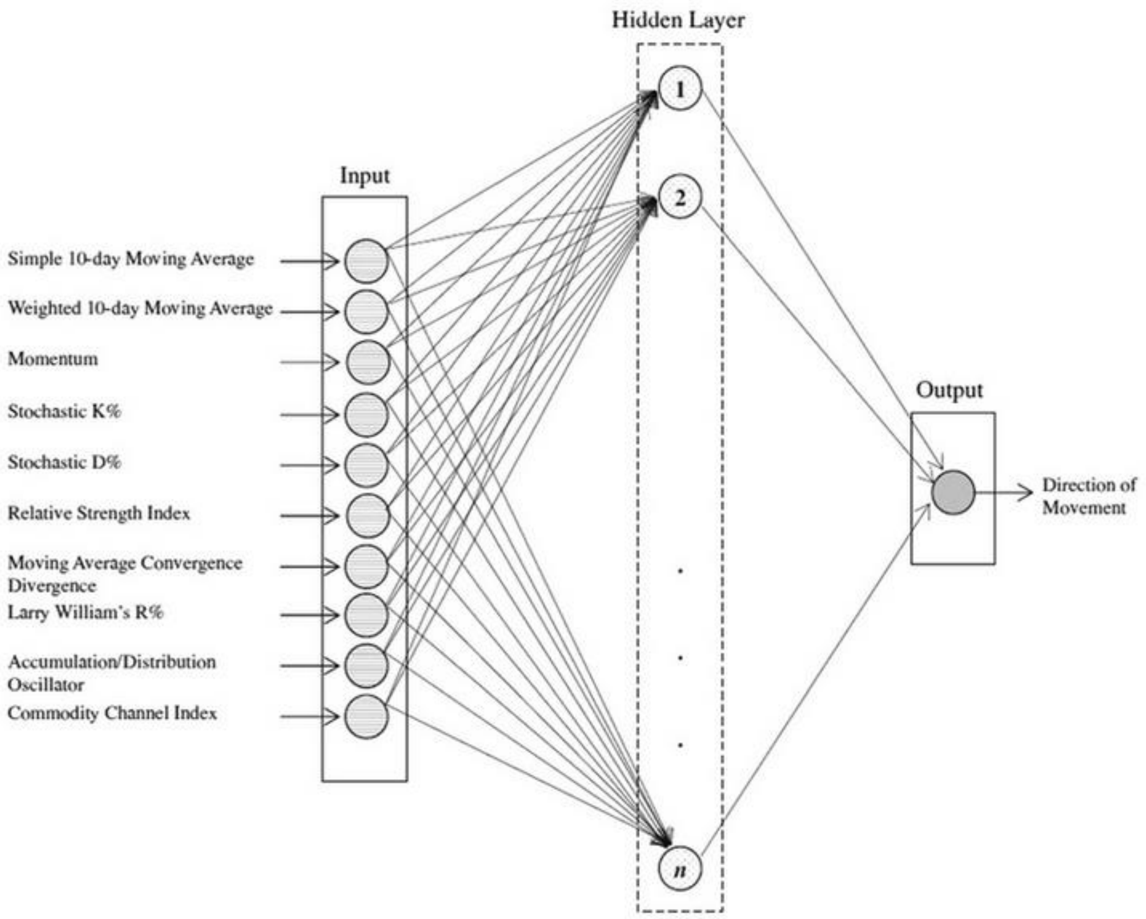


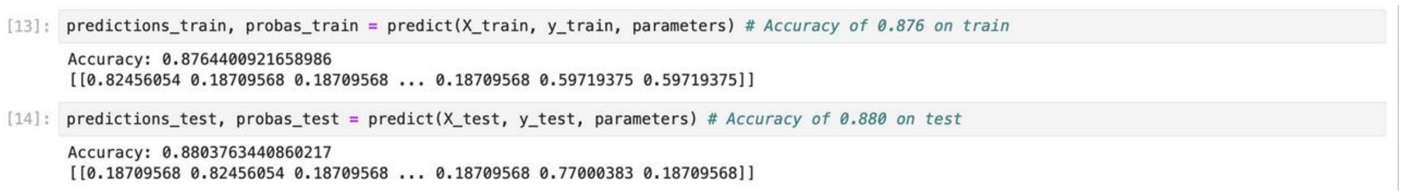
Figure 6: Representative diagram of the Deep Learning model

2.5. Train the model

During the training phase, our neural network underwent extensive learning on a meticulously curated dataset comprising 4980 samples. Leveraging sophisticated algorithms and optimization techniques, the model iteratively adjusted its parameters to minimize prediction errors and enhance performance. Through this process, the network acquired the ability to discern intricate patterns and correlations within the data, thereby gaining proficiency in predicting trends in stock market behaviour. The successful training of our model underscores the effectiveness of deep learning methodologies in quantitative trading, laying a solid foundation for subsequent analysis and evaluation.

2.6. Predictions

2.6.1. Results



```
[13]: predictions_train, probas_train = predict(X_train, y_train, parameters) # Accuracy of 0.876 on train

      Accuracy: 0.8764400921658986
      [[0.82456054 0.18709568 0.18709568 ... 0.18709568 0.59719375 0.59719375]]

[14]: predictions_test, probas_test = predict(X_test, y_test, parameters) # Accuracy of 0.880 on test

      Accuracy: 0.8803763440860217
      [[0.18709568 0.82456054 0.18709568 ... 0.18709568 0.77000383 0.18709568]]
```

Figure 7: Output of our predictive model

Following the training phase, we conducted a comprehensive analysis to evaluate the performance of our predictive model. Our assessment revealed remarkable results indicative of the model's predictive prowess. Notably, the model achieved an impressive accuracy rate of 88%, attesting to its ability to accurately capture and forecast market trends.

2.6.2 Results analysis

Further scrutiny of the model's performance yielded insightful findings. The Receiver Operating Characteristic (ROC) curve exhibited an area under the curve (AUC) value of 0.90, underscoring the model's robust predictive power. Moreover, the confusion matrix provided detailed insights into the model's classification accuracy, with 567 true positives, 178 false positives, 0 false negatives, and 743 true negatives. These findings affirm the reliability and effectiveness of our neural network in quantitative trading, demonstrating its potential to inform and guide investment decisions in financial markets.

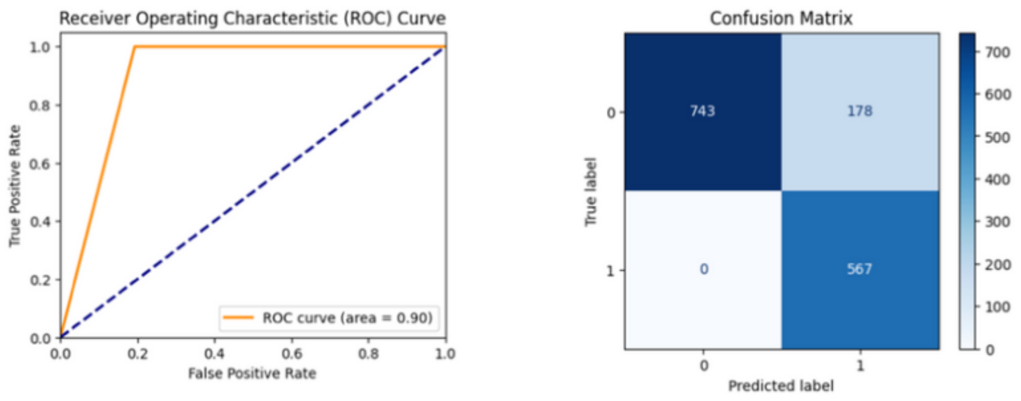


Figure 8: ROC curve and confusion matrix reporting performance of the model on test set

3. Second Approach: Double Deep Q-Network (DDQN)

3.1. Research article presentation

Introduction and Scientific Context

The article under study is titled "Deep reinforcement learning stock market trading, utilizing a CNN with candlestick images" by Andrew Brim and Nicholas S. Flann of Utah State University. It explores the application of Deep Reinforcement Learning (DRL) for stock trading. The authors use a Convolutional Neural Network (CNN) powered by candlestick images and integrated into a Double Deep Q-Network (DDQN). The article not only aims to outperform the returns of the S&P 500 index but also to understand how the model makes its trading decisions using feature map visualizations, thus addressing the "black box" often associated with neural networks.

For our project, we will focus on reproducing the model and the results, and not on the explainability of the model, which is ultimately a "fundamental research" topic that deviates from the purely practical and applied domain.

Scientific Context and Existing Literature

The existing literature mentions the use of CNNs for financial trading via images, notably candlestick charts (Tsai and Chen, 2019; Selvin and Menon, 2017). Additionally, RL techniques with deep neural networks for stock market predictions have already been explored.

Reinforcement Learning and Q-Learning

Reinforcement Learning (RL) is an artificial intelligence technique where an agent interacts with an environment by taking actions to maximize a cumulative reward. In this context, the agent is a DDQN that receives candlestick images as states and chooses an action (buy, sell, or do nothing) based on that state to maximize the return. It is important to note that, unlike other RL use cases, the action does not influence the next state (buying/selling does not affect the price evolution of a stock).

Q-learning is a specific type of learning used to estimate a function (Q-function) that associates each possible action in a given state with a numerical value: the Q-value. The idea is that the action with the highest Q-value is the best action to take. By combining Q-learning with a CNN, the article proposes to approximate this Q-function for the states represented by candlestick images.

Double Deep Q-Network (DDQN)

The Double Deep Q-Network (DDQN) is an extension of the Deep Q-Network (DQN) that uses two networks (one stable and one constantly updated) to reduce the overestimation of Q-values. This technique has shown better performance compared to standard DQNs. The DDQN used receives candlestick images as input and returns a trading action.

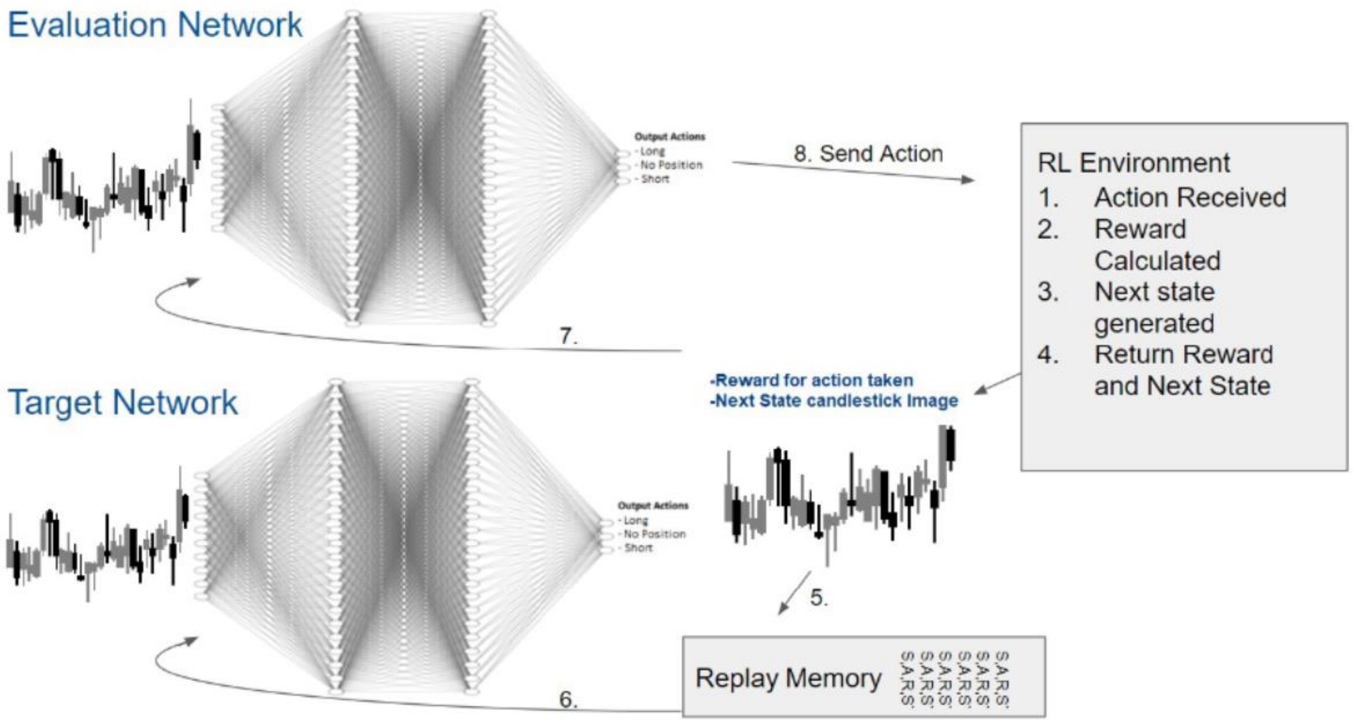


Figure 9: Learning algorithm workflow for the DDQN (will be explained in detail below)

Model performance

The article evaluates the performance of the DDQN by comparing it to the returns of the S&P 500 index during the testing period (from January 2, 2020, to June 30, 2020). The model is trained on the data of the 30 largest S&P 500 stocks from 2013 to 2019. The results show that the DDQN outperforms the S&P 500 index, even during the stock market crash related to the COVID-19 crisis. This performance is measured in terms of geometric returns.

Limitations and technical choices

Certain crucial technical details are not mentioned in the article, such as the use of batches, the optimizer, the learning rate and many more. These details must be determined during the model implementation, requiring decisions to be made in an attempt to reproduce the results obtained in the study.

3.2. Understanding the DDQN algorithm

Initialization:

- The environment is initialized (it already knows all the future states: it is assumed that the actions taken by the agent have no effects on the stock market evolution). The environment sets the current_state variable to the first state in the data set.
- The replay buffer is initialized, which will contain the 1000 most recent experience tuples. An *experience tuple* is a tuple (state, action, reward, next state).

Two processes take place simultaneously when a state is received:

Current state

The main network emits a prediction using its current parameters:

- The main network finds the best action according to its current weights
- The environment computes the corresponding reward (it knows the next state)
- The experience tuple (state, action, reward, next state) is pushed to the replay buffer

A batch is sampled from the replay buffer to update the main network's weights (the loss is calculated using the target network):

If the number of experience tuples inside the replay buffer is greater than the batch_size:

- We update the main network's weights as follows:
    - We sample a batch of experience tuples from the replay buffer
    - For each tuple in the batch, we compute the best action to be taken at the next_state *according to the main network*
    - For each tuple in the batch, we compute the Q-value associated to the action given previously (by the main network) *using the target network*
    - This whole process amounts to:
        - $Q_{target} = r + \gamma^* Q(s',a',\theta^-)$ where a' is the best action for the next state according to the main network ($a'=\mathrm{argmax}_{a'}\, Q(s',a',\theta)$).
    - We finally run a backpropagation on the main_network with the current batch using:
        - Loss = MSE [$Q(s,a,\theta)$; Qtarget]

If, moreover, the step_id (also state_id) is divisible by 100 :

- The main network's weights are copied to the target network.

Current state $\leftarrow$ Next state

Note: The fundamental difference between a DDQN and a DQN (deep Q-network) lies in the loss calculation, specifically in the calculation of Qtarget. Instead of using Q-values (based on the parameters of the target network) in the Bellman-Ford equation for an action given by the target network, we let the choice of action be made by the main network itself. This allows us to decouple the selection of actions and the evaluation of actions, which reduces the overestimation of Q-values.

In our DDQN architecture, we have: $\mathbf{Q_{target} = r + \gamma * Q(s',a',\theta^-)}$ **where** $\mathbf{a'=argmax_{a'}\ Q(s',a',\theta)}$

In a DQN architecture, we would have: $\mathbf{Q_{target} = max_{a'} [\ r + \gamma * Q(s',a',\theta^-)\ ]}$

3.3. Methodology

3.3.1. Data preparation

For data preparation, we utilized the Yahoo Finance API to import the daily high, low, open, and close prices for each stock. A Python code was developed to convert these numerical values into candlestick images, where each image represents 28 consecutive trading days. The candlestick images were generated in grayscale, with grey candles indicating an upward price movement and black candles indicating a downward price movement. This grayscale approach reduced the amount of data being processed compared to using RGB images. As described in the article, the body of each candle was made three pixels wide to represent the open and close prices, while the stick representing the high and low prices was one pixel wide. This specific representation simplified the images and reduced erroneous information, resulting in candlestick images of size 84 x 84 pixels. Importantly, all the candlestick images were created beforehand for both the training and testing datasets, as the actions taken by the model during training or testing did not affect the next states.

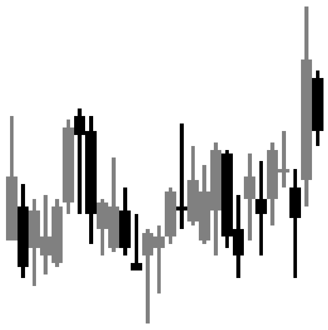


Figure 10: Example state provided by the environment to the DDQN

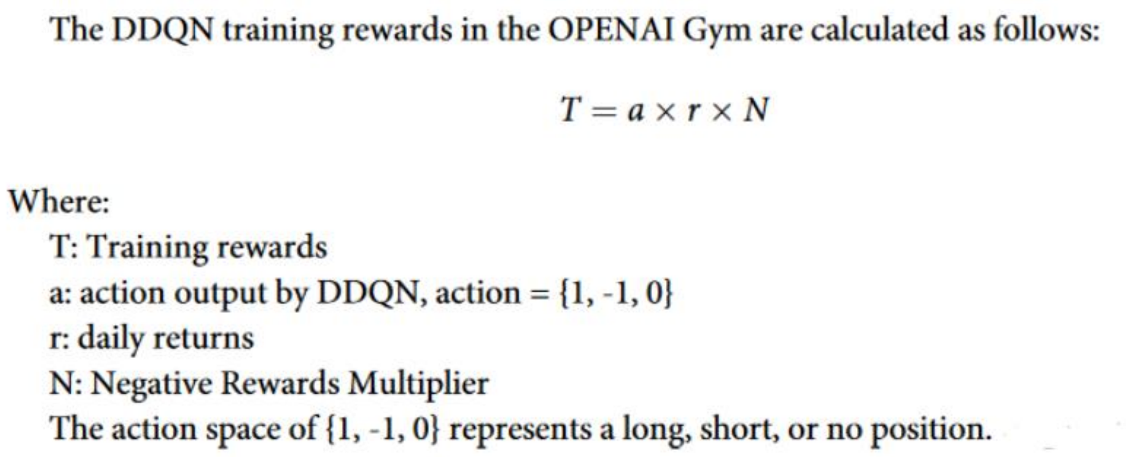
3.3.2. Model definition and architecture

For the model architecture, we followed the approach described in the article and implemented a Double Deep Q-Network (DDQN) using convolutional neural networks (CNNs) in TensorFlow. The DDQN consisted of two neural networks: a Target network for training on the candlestick images and an Evaluation network for producing actions to be sent to the reinforcement learning environment. The CNN architecture comprised an input layer that accepted the 84 x 84 pixel candlestick images, followed by three convolutional layers with 128, 256, and 512 neurons, respectively. The output layer had three neurons corresponding to the three possible actions: long, short, or no position.

3.4. Training

Regarding the training process, the article does not provide specific technical details such as the learning rate, optimizer, batch size, or discount rate used. However, it outlines the general approach to training the DDQN on the candlestick image data.

The training rewards were calculated based on the action output by the DDQN (long, short, or no position), the daily returns of the stock, and a negative rewards multiplier:



The DDQN training rewards in the OPENAI Gym are calculated as follows:

$$T = a \times r \times N$$

Where:
T: Training rewards
a: action output by DDQN, action = {1, -1, 0}
r: daily returns
N: Negative Rewards Multiplier
The action space of {1, -1, 0} represents a long, short, or no position.

Figure 11: Training rewards with NRM

The negative rewards multiplier was introduced to assist in training the DDQN to take a "no position" action when appropriate.

The article mentions that 1000 training episodes were selected as a sufficient number to train the DDQN without overfitting to the training data.

3.5 Inference and evaluation

During the inference and evaluation phase of our project, we utilized the trained main network to predict actions based on new input data, represented as candlestick images. The evaluation phase involved comparing the performance of the DDQN against a baseline, such as the S&P 500 Index, by assessing returns over a specific period. Notably, the original article did not specify the method for calculating episode rewards, which are crucial for performance evaluation. Therefore, we devised our own metric by summing the daily returns to calculate the total episode reward. This approach ensured a consistent and transparent evaluation of the model's performance, providing a clear measure of its trading effectiveness .

3.6 Discussion on results

3.6.1. Comparing results to article results

Our model took approximately 50 hours to complete 1000 epochs of training. Unfortunately, it did not exhibit the capacity to adapt to new, unseen test data, resulting in returns gravitating around 0%, indicating neither a loss nor a gain. However, during training, the model demonstrated signs of learning, as the episode returns rapidly increased, reaching a peak return of approximately 8 within just 50 episodes. This suggests that the DDQN was able to learn from the training data, though it ultimately overfitted. In comparison, the results in the referenced article show a more gradual increase in training returns, also peaking at around 8, but over the course of 1000 episodes. Moreover, the article's model successfully adapted to unseen data, achieving notable results: "the results of Test 1 show the DDQN tested on the largest 30 stocks of the S&P 500, yield an average of 13.2% geometric returns in 124 trading days".

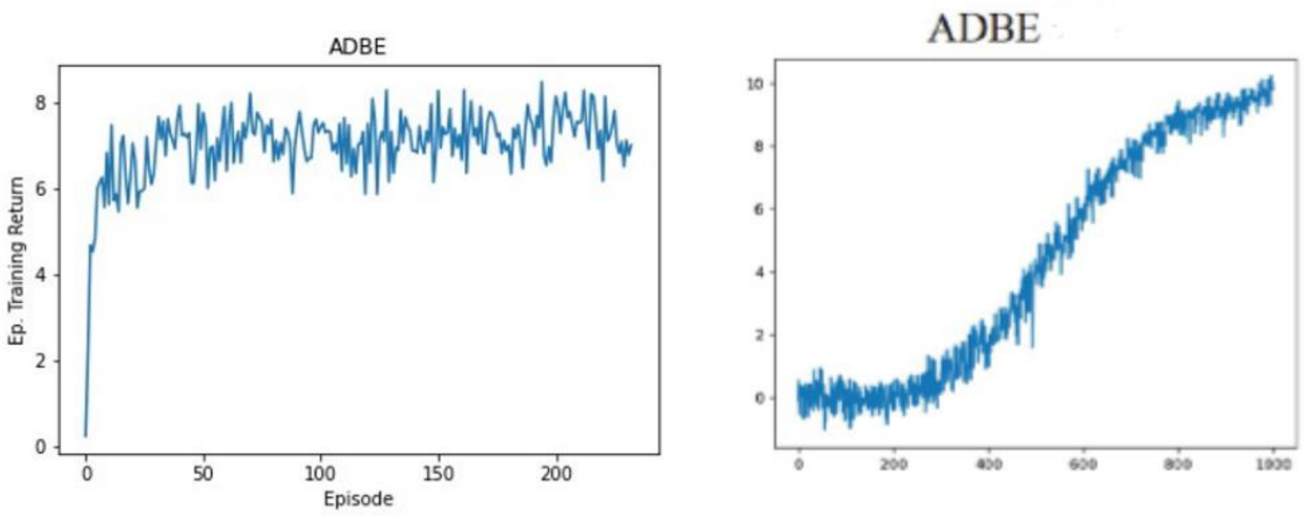


Figure 12: Episode training returns for the ADBE stock for our model (left) and for the article’s model (right)

3.6.2. Critique paper

The critique thesis paper titled "A Critique of: Deep reinforcement learning stock market trading, utilizing a CNN with candlestick images" challenges the claims made by Brim et al. regarding the effectiveness of their DDQN model with a CNN for stock market trading. The authors of the critique were unable to replicate the impressive results reported by Brim et al., raising significant concerns about the validity of their findings. Specifically, the critique article states, "We were unable to obtain evidence that supports the claim of Brim et al. that a DDQN with an incorporated CNN can outperform the market. [...] This raises an important question: Are the findings reported by Brim et al. the result of systematic factors or mere chance?" This highlights a fundamental issue in the original study: the lack of transparency and detail in the experimental setup, particularly regarding parameter settings and the absence of a sensitivity analysis.

The critique emphasizes that without a sensitivity analysis, it is unclear whether the reported results are due to the model's actual learning capability or simply favourable initial weight configurations. The authors attempted to contact Dr. Brim for more details on the parameters used but were unsuccessful, which further complicates the replication and validation of the original study's results.

Thus, the critique article corroborates our own findings that the results in the initial article are not attainable without additional technical details that were not provided, and underscores the importance of thorough documentation and sensitivity analysis in experimental research.

3.6.3. Conclusion

In conclusion, the initial article in question claims to achieve impressive returns, yet it fails to provide sufficient technical details, which significantly hinders the reproducibility of the results. One could hypothesize that by adjusting the model's parameters, the DDQN (Double Deep Q-Network) framework could develop the capability to discern and incorporate the relationship between the features and the target. Additionally, increasing the size of the dataset could potentially improve the performance of the DDQN model, as it is plausible that the current dataset might be inadequate for establishing robust relationships. However, these assumptions remain uncertain due to the authors' reluctance to share the technical details of their work.

4. Comparing the Two Approaches

4.1. Performance analysis

The performance analysis of the two approaches reveals significant differences in both training and testing outcomes. The Double Deep Q-Network (DDQN) took approximately 50 hours to complete 1000 epochs of training. Despite this extensive training period, the DDQN did not adapt well to new, unseen test data, resulting in returns that hovered around 0%, indicating no significant gain or loss. During training, however, the DDQN showed signs of learning, with episode returns quickly increasing and reaching a peak return of about 8 within just 50 episodes, suggesting initial learning from the training data but ultimately overfitting it. In stark contrast, the Deep Neural Network (DNN) required only a few seconds for training and achieved an impressive train/test accuracy rate of 88%, meaning 88% of its predictions were correct guesses. It is important to note that the evaluation metrics for the two models were different: for the DNN, performance was measured simply by the accuracy of good versus bad guesses, whereas for the DDQN, it was based on geometric returns. This difference in evaluation metrics highlights the distinct approaches and challenges in assessing the performance of each model.

4.2. Conclusion

The Cassiopée project has provided us with an exceptional opportunity to delve into the theory and practical applications of deep learning. Our journey has been marked by significant growth in our understanding of both the financial sector and the intricacies of deep learning, including reinforcement learning. A particularly valuable insight we have gained is that escalating the complexity of a model does not necessarily lead to superior outcomes. In essence, the responsibility of an AI engineer lies in discerning when to employ AI, and when to opt for complex models versus more straightforward ones.

To conclude, the Critique Article brings to light an important point regarding the use of purely numerical representations, such as historical price data, for making investment decisions, a practice commonly referred to as technical analysis. According to prevailing financial theory, as outlined by

Hull (2003), technical analysis is not considered capable of consistently generating above-average returns.

Lastly, it is crucial to remember that there is a substantial gap between model results on paper and in real-life trading. Numerous other factors, such as geopolitical events, trading fees, and portfolio management, can significantly impact the profitability of a strategy. Therefore, discovering a model with excellent results on paper does not guarantee success in the dynamic and complex world of real-life trading.

We extend our heartfelt gratitude to Mrs. Tetiana Lemishko and Mr. Alexandre Landi for their invaluable support and guidance throughout the course of our project.

5. References

Naik N, Biju R. Mohan (2019). Intraday Stock Prediction Based on Deep Neural Network. https://doi.org/10.1007/s40009-019-00859-1

Brim A, Flann NS (2022) Deep reinforcement learning stock market trading, utilizing a CNN with candlestick images. PLoS ONE 17(2): e0263181. https://doi.org/10.1371/journal.pone.0263181

Frits S. Tuininga (2023). Uncovering the Potential of Deep Learning in Algorithmic Trading. A Critique of: Deep reinforcement learning stock market trading, utilizing a CNN with candlestick images. http://essay.utwente.nl/95077/1/AM_Thesis%20%2818%29.pdf