University of Brasilia

Institute of Exact Sciences
Department of Computer Science

Using AI for Forecasting and Trading in Markets: a Study of Deep Learning and Deep Reinforcement Learning in Finance

João Paulo Costa e Souza

Dissertation presented as a partial requirement for
conclusion of the Master's Degree in Informatics

Advisor
Prof. Dr. Geraldo Pereira Rocha Filho

Brasilia
2025

University of Brasilia

Institute of Exact Sciences
Department of Computer Science

Using AI for Forecasting and Trading in Markets: a Study of Deep Learning and Deep Reinforcement Learning in Finance

João Paulo Costa e Souza

Dissertation presented as a partial requirement for the conclusion of the Master's Degree in Informatics

Prof. Dr. Geraldo Pereira Rocha Filho (Advisor)
DCET/UESB - CiC/UnB

Prof. Dr. Francisco Airton Silva
Federal University of Piauí

Prof. Dr. Edna Dias Canedo
University of Brasília

Prof. Dr. Rodrigo Bonifácio Almeida
Coordinator of the Postgraduate Program in Informatics

Brasilia, February 28, 2025

Dedication

I dedicate this work primarily to God who never let giving up settle in my heart. I dedicate it to my family, whose unconditional support was the structuring factor in my achievements. I also dedicate it to my friends, who have always supported me, and believed in my potential.

Acknowledgements

I express my deep gratitude to the Department of Computer Science (CIC/UnB) for the continuous support and opportunities that significantly contributed to my academic training. Furthermore, I am immensely grateful to my advisor, Prof. Dr. Geraldo Pereira Rocha Filho, for his dedication, guidance, and invaluable support throughout the master's degree.

This work was carried out with the support of *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) - Brazil, through Access to the Periodicals Portal.

Resumo

No contexto de negociação financeira, a identificação de tendências e estratégias de tomada de decisão são componentes cruciais para atingir lucratividade consistente e minimizar riscos. Dada a importância desses dois desafios, este projeto propõe uma solução que combina várias técnicas avançadas para aprimorar a precisão preditiva e o desempenho de negociação. A abordagem proposta incorpora Autoencoders para engenharia de recursos, permitindo redução efetiva de dimensionalidade e extração de padrões significativos de indicadores técnicos. Para identificação de tendências, o modelo alavanca uma combinação de Redes Neurais Convolucionais (CNNs) e Redes de Memória de Longo e Curto Prazo Bidirecionais (BiLSTMs), juntamente com rotulagem baseada em regressão linear para definir tendências de mercado com maior precisão. Na fase de tomada de decisão, o sistema emprega o algoritmo Rainbow DQN, aprimorado com um buffer de memória, para otimizar estratégias de negociação e maximizar a lucratividade no mercado de criptomoedas. Para avaliar rigorosamente a eficácia da estrutura proposta, o desempenho de identificação de tendências é avaliado usando as métricas precision, recall e acurácia, enquanto o desempenho de negociação é medido por meio do retorno, sua média e desvio padrão. Os resultados experimentais demonstram que o modelo supera supera a perspectiva aleatória na previsão de tendências, validando a eficácia dos componentes de engenharia de características e aprendizado profundo. Além disso, apesar de experimentar flutuações significativas nos retornos de negociação, o agente obteve um lucro médio positivo, destacando o potencial da abordagem proposta no desenvolvimento de uma estratégia de negociação adaptável e orientada a dados.

Palavras-chave: Rainbow DQN, Aprendizagem por Reforço, Aprendizagem por reforço profundo, Predição de preços, Criptomoedas, Finanças

Resumo Expandido

Usando IA para Previsão e Negociação em Mercados: um Estudo de Aprendizagem Profunda e Aprendizagem por Reforço Profundo em Finanças.

Introdução

A teoria dos mercados eficientes, proposta por Eugene Fama em 1970, defende que todas as informações disponíveis já estão refletidas nos preços dos ativos, impossibilitando ganhos consistentes por meio da especulação. No entanto, diversos estudos questionam essa hipótese, apresentando evidências de ineficiências de mercado e comportamento irracional dos investidores, como o efeito disposição, que os leva a vender ganhos cedo demais e manter perdas por muito tempo.

Diante da possibilidade de prever tendências de mercado e lucrar com elas, o projeto propõe o uso de técnicas de aprendizado de máquina para identificar tendências de alta e baixa. Especificamente, são utilizadas redes neurais profundas como LSTM e CNN, que lidam bem com dados sequenciais e séries temporais ruidosas. A segunda parte do problema abordado é como agir com essa informação de tendência. Para isso, é proposto o uso do algoritmo Rainbow DQN, uma versão avançada do Deep Q-Learning, adaptada com camadas recorrentes para melhor desempenho em séries temporais.

Fundamentação Teórica

Redes Neurais Artificiais (ANNs) são modelos computacionais inspirados no funcionamento do cérebro humano. Elas são compostas por camadas de neurônios interconectados que processam informações e aprendem padrões a partir de dados. São amplamente utilizadas em tarefas como classificação, regressão e previsão.

Redes Neurais Convolucionais (CNNs) são especializadas no processamento de dados com estrutura espacial, como imagens e séries temporais. Utilizam camadas convolucionais para extrair automaticamente características relevantes dos dados, sendo eficazes na remoção de ruídos e na identificação de padrões locais.

Redes Neurais Recorrentes (RNNs) são voltadas para dados sequenciais, onde a ordem das informações é relevante. Elas possuem conexões que permitem "memória" ao longo do tempo. Uma variação popular, o LSTM (Long Short-Term Memory), resolve limitações das RNNs padrão, como o problema do gradiente, sendo eficaz para capturar dependências de curto e longo prazo.

Autoencoders são redes neurais usadas para compressão e reconstrução de dados. Eles aprendem a codificar informações de entrada em uma representação compacta (codificação) e depois decodificá-la para tentar recuperar a entrada original. São úteis para redução de dimensionalidade e extração de características.

Rainbow DQN é uma versão aprimorada do algoritmo Deep Q-Network (DQN), que combina várias técnicas de reforço profundo, como Dueling Networks, Prioritized Experience Replay e Double Q-Learning. Ele é utilizado para tomada de decisão em ambientes sequenciais e é especialmente eficaz em tarefas de negociação automática, onde o agente aprende estratégias de compra e venda visando o maior retorno possível.

Embora muitas dessas variáveis tenham sido propostas na literatura, apenas uma pequena parte é realmente informativa, e o excesso pode introduzir ruído e redundância. Métodos de seleção de variáveis ajudam a lidar com esse problema e se dividem em três categorias principais: filters, wrappers e embedded. Os métodos filter são estatísticos e eficientes, mas ignoram relações entre variáveis; wrappers avaliam o desempenho das variáveis em modelos de aprendizado, mas são mais custosos; e embedded integram a seleção ao treinamento do modelo, equilibrando custo e performance. Diversos trabalhos aplicaram essas técnicas no mercado financeiro com diferentes conjuntos de dados e indicadores técnicos, como fechamento, volume e volatilidade, usando desde métodos estatísticos e algoritmos bioinspirados até redes neurais e autoencoders com LSTM. Os resultados mostram que, embora métodos automáticos e híbridos otimizem a escolha de variáveis, o conjunto final de features ainda pode ser grande, exigindo estratégias adicionais para lidar com a complexidade computacional.

Trabalhos Relacionados

Trabalhos relacionados à engenharia de características: Embora muitas dessas variáveis tenham sido propostas na literatura, apenas uma pequena parte é realmente informativa, e o excesso pode introduzir ruído e redundância. Métodos de seleção de variáveis ajudam a lidar com esse problema e se dividem em três categorias principais: filters, wrappers e embedded. Os métodos filter são estatísticos e eficientes, mas ignoram relações entre variáveis; wrappers avaliam o desempenho das variáveis em modelos de aprendizado, mas são mais custosos; e embedded integram a seleção ao treinamento do modelo, equilibrando custo e performance. Diversos trabalhos aplicaram essas técnicas no mercado financeiro

com diferentes conjuntos de dados e indicadores técnicos, como fechamento, volume e volatilidade, usando desde métodos estatísticos e algoritmos bioinspirados até redes neurais e autoencoders com LSTM. Os resultados mostram que, embora métodos automáticos e híbridos otimizem a escolha de variáveis, o conjunto final de features ainda pode ser grande, exigindo estratégias adicionais para lidar com a complexidade computacional.

A tarefa de previsão de preços é desafiadora, exigindo seleção criteriosa de variáveis e escolha adequada de arquiteturas de aprendizado de máquina. Modelos podem se dividir entre previsão de tendência (classificação) e previsão nominal de preços (regressão). Diversos trabalhos abordam essa tarefa com diferentes abordagens: Patel et al. [1] utilizam uma abordagem em duas etapas com SVR, ANN e Random Forest, enquanto Lin et al. [2] combinam padrões gráficos, trigramas taoístas e indicadores técnicos com um sistema ensemble. Selvin et al. [3] mostram que CNN supera RNN e LSTM na previsão de curto prazo, devido à sua habilidade em detectar mudanças súbitas. He et al. [4] e Livieris et al. [5] combinam CNN com LSTM e mecanismos de atenção, enquanto Liang et al. [6] melhoram a previsão decompondo os dados com ICEEMDAN antes de aplicar redes neurais. Siami-Namini et al. [7] demonstram que BiLSTM supera LSTM em acurácia, apesar de sua complexidade. Zhang et al. [8] propõem uma arquitetura híbrida com GRU, autoatenção e módulos de convolução para capturar dependências entre criptomoedas, obtendo desempenho superior. Por fim, Samarasekara et al. [9] inovam ao utilizar previsões de preço e tendência para otimizar a colocação de stop-loss, reduzindo significativamente as perdas em comparação com métodos tradicionais.

A aplicação de aprendizado por reforço no mercado financeiro envolve desafios significativos de modelagem, como a definição clara do problema, a escolha da arquitetura do agente, o design das recompensas e punições, a engenharia das ações e o equilíbrio entre exploração e exploração. Diversos trabalhos propõem soluções inovadoras, como o uso de redes GRU para previsão de preços aliadas ao agente de reforço [10], estratégias baseadas em DQN e DDPG com extração de informações financeiras [11], e integração com análise de sentimentos por meio de CNNs e LSTM [12]. Outras abordagens envolvem a transformação de dados numéricos em imagens para aproveitamento por CNNs [13], além de técnicas para reduzir ruídos e expandir o espaço de ações [14]. Em geral, os resultados apontam ganhos expressivos em rentabilidade e precisão, reforçando o potencial do aprendizado por reforço profundo no contexto de operações financeiras automatizadas.

Projeto de Pesquisa

O módulo de Feature Engineering é responsável por processar os dados brutos do mercado e transformá-los em informações relevantes para treinar redes neurais nos módulos de previsão e negociação. Para isso, utiliza indicadores técnicos clássicos, como médias

móveis, osciladores e medidores de volume, volatilidade e ciclos, totalizando 21 entradas ao modelo. Os dados são pré-processados por meio de padronização (Z-score) e remoção de outliers, visando facilitar o aprendizado e melhorar a convergência das redes neurais. Para lidar com a alta dimensionalidade e reduzir o ruído, aplica-se um autoencoder baseado em células BiLSTM, que comprime os dados mantendo suas principais informações, tornando o treinamento mais eficiente.

O módulo de predição é responsável por processar os dados de entrada e gerar estimativas sobre os preços futuros, sendo essencial para decisões de compra e venda. Com base em estudos prévios, como os de Selvin et al. e Zhanhong He et al., foram propostas duas arquiteturas: uma para classificar a tendência do mercado (alta ou baixa) e outra para prever a inclinação dessa tendência. O Market Trend Predictor utiliza uma combinação de camadas convolucionais e BiLSTM para extrair características relevantes e capturar dependências temporais, finalizando com uma camada densa com dropout. A rotulagem é feita com base na inclinação da regressão linear aplicada a um conjunto de candles. Já o Trend Slope Regressor tem como objetivo estimar numericamente essa inclinação, utilizando primeiro camadas BiLSTM e, posteriormente, camadas convolucionais e de pooling para refinar a informação. Ambos os modelos são treinados com aprendizado supervisionado, utilizando funções de custo adequadas a seus respectivos problemas (classificação e regressão).

O módulo de negociação tem como objetivo utilizar informações de indicadores técnicos e preditores para tomar decisões otimizadas de compra e venda de ativos. Para isso, é adotado o aprendizado por reforço com redes neurais, utilizando especificamente o algoritmo Rainbow DQN. O agente de negociação é treinado para analisar o mercado com base em preços atuais, tendências previstas e indicadores comprimidos, tomando decisões em um espaço de ações composto por sete possibilidades — que variam entre diferentes intensidades de compra, venda e inação. As recompensas são definidas em função do lucro obtido com as operações. Inicialmente, a rede neural do agente é composta por três camadas densamente conectadas, cada uma com 64 neurônios. Posteriormente, são introduzidas camadas BiLSTM, representando uma "memória" que visa melhorar o desempenho do agente ao permitir o aprendizado de padrões temporais mais profundos.

A avaliação dos modelos propostos é realizada por meio de métricas específicas para cada tipo de problema. O Market Trend Predictor, sendo um classificador, é avaliado com as métricas de acurácia, precisão e recall, que medem, respectivamente, a taxa geral de acertos, a qualidade das previsões positivas e a capacidade de identificar corretamente tendências positivas. Já o Trend Slope Regressor, por tratar-se de um modelo de regressão, é avaliado pelas métricas RMSE, MAE e MAPE, que quantificam o erro das previsões em diferentes perspectivas. O desempenho do módulo de negociação é medido pelo lucro

obtido, refletindo diretamente a efetividade da estratégia. Para validar os modelos, seus resultados são comparados com arquiteturas mais simples, como LSTM ou CNN isoladas. Além disso, os modelos são testados em diferentes cenários envolvendo criptomoedas populares (Bitcoin, Ethereum, Ripple e Cardano), incluindo períodos marcados por eventos geopolíticos críticos, como a proibição das criptomoedas na China, de modo a avaliar a robustez e a adaptabilidade das soluções em situações reais e adversas.

Experimentos e Resultados

O módulo de engenharia de caracteristicas começou com a seleção teórica de indicadores técnicos que fornecessem informações relevantes e não redundantes ao modelo. Após essa etapa, foi analisada a aplicação de diferenciação para tornar a série temporal estacionária, mas essa abordagem foi descartada devido à perda significativa de informação. Em vez disso, optou-se pela padronização dos dados, que não compromete a estrutura da série. Em seguida, utilizou-se um autoencoder com células LSTM e BiLSTM para redução de dimensionalidade e remoção de ruído. Foram avaliadas diferentes arquiteturas de autoencoder variando o tamanho do vetor de contexto, o número de camadas e a janela temporal. Os resultados mostraram que modelos BiLSTM superaram os LSTM em todas as métricas e que um vetor de contexto de 10 dimensões, quatro a oito camadas e janelas temporais curtas (especialmente 6 time steps) produzem melhores resultados, indicando uma curta dependência temporal nos dados.

Os experimentos sobre o módulo de predição apresentaram resultados interessantes. O experimento que variou a amplitude da Classe de Mercado Lateral revelou uma queda em todas as métricas de avaliação. Especificamente, não apenas a acurácia diminuiu, mas também a precisão e o recall. Isso sugere que o modelo apresentou seu melhor desempenho quando a Classe de Mercado Lateral foi completamente desconsiderada. O experimento seguinte, que consistiu no aumento do limiar do Argmax, trouxe melhorias notáveis nas métricas de avaliação. No entanto, esse ajuste resultou em uma redução significativa no número de amostras classificadas. Por fim, o experimento que variou o ativo teve impacto mínimo nos resultados, indicando que o modelo é robusto em relação a diferentes ativos.

Os experimentos realizados mostraram que o uso exclusivo de indicadores técnicos como entrada para o Rainbow DQN não gerou desempenho significativamente superior ao de uma abordagem aleatória, indicando que tais indicadores, sozinhos, não fornecem informações estruturadas suficientes para decisões eficazes. No entanto, ao combinar esses dados com previsões geradas por um módulo preditivo baseado em deep learning, o desempenho do modelo melhorou substancialmente, permitindo identificar padrões de mercado com mais precisão e convertê-los em operações lucrativas. Além disso, a introdução de um mecanismo de memória nas camadas iniciais da rede neural, considerando o ambiente

como um POMDP, mostrou-se promissora ao permitir o uso de informações históricas e decisões mais informadas. Apesar de ainda apresentar instabilidades na curva de retorno, o modelo foi capaz de, ao menos parcialmente, formular uma estratégia de negociação lucrativa baseada na identificação de tendências de mercado.

Conclusão

Este trabalho abordou os desafios de operar no mercado de criptomoedas por meio de uma arquitetura composta por três módulos: engenharia de atributos, previsão de tendências e execução de operações com aprendizado por reforço. Inicialmente, foram identificados dois principais problemas: extrair informações relevantes para prever tendências de mercado e converter essas previsões em estratégias de negociação lucrativas. Após a revisão da literatura relacionada, foram apresentadas as bases teóricas que sustentam o projeto, incluindo redes neurais densas, convolucionais, recorrentes, autoencoders, indicadores técnicos e o algoritmo Rainbow DQN. A arquitetura proposta foi detalhada no quarto capítulo, demonstrando a integração entre os módulos e suas respectivas metodologias. Nos experimentos, o módulo de engenharia de atributos obteve sucesso na redução de dimensionalidade sem perda significativa de informação, enquanto o módulo de previsão apresentou desempenho significativamente superior ao acaso na identificação de tendências. Por fim, o módulo de negociação mostrou que o modelo, apesar da volatilidade dos retornos, conseguiu obter lucros médios positivos, especialmente quando utilizou previsões e memória, demonstrando o potencial da abordagem proposta para tomada de decisões no mercado financeiro.

Palavras-chave: Rainbow DQN, Aprendizagem por Reforço, Aprendizagem por reforço profundo, Predição de preços, Criptomoedas, Finanças

Abstract

In the context of financial trading, trend identification and decision-making strategies are crucial components for achieving consistent profitability and minimizing risk. Given the importance of these two challenges, this project proposes a solution that combines multiple advanced techniques to enhance both predictive accuracy and trading performance. The proposed approach incorporates Autoencoders for feature engineering, enabling effective dimensionality reduction and the extraction of meaningful patterns from technical indicators. For trend identification, the model leverages a combination of Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory Networks (BiLSTMs), alongside linear regression-based labeling to define market trends with greater precision. In the decision-making phase, the system employs the Rainbow DQN algorithm, enhanced with a memory buffer, to optimize trading strategies and maximize profitability in the cryptocurrency market. To rigorously evaluate the effectiveness of the proposed framework, trend identification performance is assessed using accuracy, precision, and recall, while trading performance is measured through return, its mean, and standard deviation. The experimental results demonstrate that the model outperforms a random baseline in trend prediction, validating the effectiveness of the feature engineering and deep learning components. Additionally, despite experiencing significant fluctuations in trading returns, the agent achieved a positive average profit, highlighting the potential of the proposed approach in developing an adaptive and data-driven trading strategy.

Keywords: Rainbow DQN, Reinforcement Learning, Deep reinforcement learning, Price Forecasting, Cryptocurrencies, Finance

Contents

List of Figures

List of Tables

Acronyms

A3C Asynchronous Advantage Actor-Critic.

ADX Average Directional Movement.

ANN Artificial Neural Network.

ATR Average True Range.

CCI Commodity Channel Index.

CNN Convolutional Neural Network.

DPO Detrended Price Oscillator.

DQN Deep Q-Network.

GAN Generative Adversárial Networks.

GRU Gated Recurrent Unit.

LSTM Long Short-Term Memory.

MACD Moving Average Convergence Divergence.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MOM Momentum.

MSE Mean Squared Error.

NATR Normalized Average True Range.

OBV On Balance Volume.

PPO Percentage Price Oscillator.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

ROC Rate of Change.

RSI Relative Strength Index.

Stoch Stochastic.

SVR Support Vector Regression.

TR True Range.

VAR Variance.

WMA Weighted Moving Averages.

Chapter 1

Introduction

1.1 Contextualization and Problem

In 1970, Eugene Fama enunciated the theory of efficient markets [21], according to which all available information about a given asset is already included in the price of that asset. The inevitable conclusion that leads to this theory lies in the impossibility of the investor obtaining significant gains since he will never have access to more information than that already contained in the asset's current price. Any speculation about the future price is completely irrational and even useless. In its mathematical formulation, the theory of efficient markets assumes that market prices have the form of a random walk [22], a stochastic process that has an unpredictable nature and is independent of the sequence of states it assumed in the past; thus, even observing its entire history, it is not possible to say anything about the possibilities that the process will take in the future. The efficient markets hypothesis has had such cultural traction in finance circles that Michael Jensen, a professor at the University of Chicago, was able to write in 1978 that "the efficient markets hypothesis is the best established empirically supported fact in all of economics" [23]. He says: "It has evolved from a mere curiosity, taken seriously by only a few scientists, to the dominant paradigm in finance."

However, there is a wealth of studies in the financial literature aimed at analyzing potential or even common market inefficiencies and statistically refuting the hypothesis of efficient markets. For example, in the studies by Andrew Lo and A. Craig MacKinlay [24], using variance estimators, evidence is shown that the random walk is not a consistent model with stock prices, since when taking certain time intervals and evaluating the price variance in these intervals, there will be a discrepancy about what is expected from a random walk. The work of Shleifer and Summers [25] proposes an alternative to the theory of efficient markets that says that investors are not entirely rational and that their demands for risky assets are affected by their beliefs or feelings that are not fully justified

by the news fundamental. These studies highlight the theoretical plausibility of financial speculation and the possibility of consistent prediction of asset prices, together with the theoretical feasibility of building a predictor since the market is not always efficient. Even so, the prediction task is still acutely difficult, and several predictor models or simple strategies have been tested and retested under this task, mainly to find valuable market inefficiencies. In particular, this is still a widely studied and debated topic.

A common mistake in efficient market theory is to neglect that markets are made up of people, and people are not always completely rational, fully understanding the information contained in asset prices, or never assuming incorrect information. On the contrary, sometimes people follow emotions brought about by the essential market volatility much more than they follow previously established rational assumptions and strategies. One of the best-known phenomena in financial psychology is the disposition effect [26]. Named after Hersh Shefrin and Meir Statman, it states that investors tend to sell an asset that is making a profit too quickly, failing to take advantage of its full potential profit, and, in addition, they also tend to hold on to an asset that is making a loss too long, in the hope that it will reverse its price and become profitable, leading to losses increasingly and continuously. Thus, it can be said that investors hate losses much more than they love profits, which undermines the theory of efficient markets because if a good portion of investors who embody the market have such a pattern of behavior, it shows a blatant inefficiency of the market. The causes of this type of psychological phenomenon can be many and were well studied by Shefrin and Statman [26]. They proposed that losses are much more emotionally charged than profits. This is because they can affect deep feelings that are even rooted in the investor's life history, while profits affect more superficial feelings. The depression of losing money can hurt a search for personal self-affirmation in the markets, lead to an intolerance to regret, and cause a sometimes inescapable lack of self-control, while profit always leads to a feeling of euphoria, which is much more superficial and does not touch on so many critical points of the investor's personality, psyche, and personal history.

Assuming that markets are not entirely efficient, it becomes possible to generate profits through trading. This possibility arises because inefficiencies create opportunities for traders to exploit price discrepancies, capitalize on trends, or identify undervalued or overvalued assets. Researchers have proposed various market rules to guide investors in their decisions, using historical price movements and trading volume in the form of technical analysis. This analysis helps forecast the continuation or reversion of market trends by identifying patterns in past price data. Nevertheless, manual trading remains difficult due to market uncertainty, emotional influences on traders [26], and the infinitude of indicators and other financial data [27]. Algorithmic trading, incorporating machine learning tech-

niques, offers a solution. By automating the decision-making process, machine learning algorithms can analyze vast amounts of historical and real-time data to identify actionable patterns and trends with greater accuracy and speed than human traders. Techniques such as classification, regression, and clustering allow models to predict market directions, classify market states, and detect anomalies.

This is the first problem addressed in this work: based on the theoretical possibility of market inefficiency, accurately identify their trends, which can be bullish if it is an uptrend or bearish if it is a downtrend. This is done via Machine Learning, with techniques such as classification and regression, which allow models to predict market directions, and classify market trends. Deep learning techniques, particularly Convolutional Neural Networks (CNN) and long short-term memory (LSTM) layers, have proven effective for financial forecasting in the cryptocurrency market. LSTM layers efficiently capture sequential patterns in both long- and short-term dependencies, whereas convolutional layers serve to eliminate noise from intricate time-series datasets and extract valuable patterns [5].

The second problem addressed in this work is about what to do with the information given by the identification of trends, specifically: How to trade and make a profit with this information. To solve this problem the Rainbow DQN algorithm is suggested, as it is one of the best deep reinforcement learning algorithms available, even more: a little change is implemented to make the algorithm better suited to time series, which uses recurrent layers at the beginning of the neural network.

The literature presents a variety of models aimed at predicting the dynamics of financial markets, often achieving impressive results in price prediction. For instance, Selvin et al. compare classical econometric methods with cutting-edge deep learning techniques, concluding that the latter are superior [3]. Similarly, Zhanhong He et al. study different architectures for predicting gold prices, identifying a combination of CNN, LSTM, and attention mechanisms as the most effective [4]. Zhuorui Zhang et al. develop a robust price prediction regressor using memory and convolutional layers that consider the interrelations between cryptocurrencies [8]. In another study, Ioannis E. Livieris et al. create a CNN-LSTM-based neural network to predict gold prices, which shows promising results in regression but falls short in trend classification, with accuracy close to randomness [5]. Livieris et al. also propose a non-sequential CNN-LSTM neural network for price regression and trend classification, again finding trend classification results near randomness [28]. Yanhui Liang et al. take a different approach by decomposing gold prices into various frequencies before feeding them into a CNN-LSTM structure, though they do not test the model in trend classification [6]. Sima Siami-Namini et al. compare Bidirectional Long Short Term Memory (BiLSTM) and LSTM for price prediction, showing that BiLSTM achieves lower errors, but they do not test the model in trend classification [7].

Iromie K. Samarasekara et al. focus on risk management rather than trading, developing a dynamic Stop-Loss tool that uses a CNN-LSTM trend classifier to feed a price regressor, the study showed good results compared to other tools of risk management [9]. Faraz et al. enhance an LSTM price regressor with an LSTM autoencoder, achieving good results in price prediction but not testing it in trend classification [29]. Despite these successes in price prediction, trend prediction often yields poor results. To address this gap, the current study proposes a novel trend classification method using a robust BiLSTM-CNN model and suggests linear regression for better sample labeling in trend slope regression and classification tasks.

Yasmeen Ansari et al. [10] propose leveraging deep reinforcement learning for financial market analysis. Their approach utilizes a neural network that processes market data, indicators, and future price estimates. To generate these estimates, a secondary neural network predicts the next price values. The predictive model demonstrated strong performance in terms of root mean square error and mean absolute error, while the reinforcement learning agent achieved significant profitability. Xing Wu et al. [11] explore two machine learning-based trading strategies: the *Gated Deep Q-Learning Trading Strategy* and the *Gated Deterministic Policy Gradient Trading Strategy*. Their results indicate that these strategies generate more buy and sell signals compared to traditional approaches and outperform other state-of-the-art reinforcement learning strategies in profitability. Similarly, Akhil Raj Azhikodan et al. [12] integrate a trend prediction module based on sentiment analysis with reinforcement learning. The trend predictor employs a recurrent neural network to analyze financial news, while the reinforcement learning module uses the *Deep Deterministic Policy Gradient* algorithm. The results showed a test accuracy of approximately 96.88%, confirming the feasibility of predicting trends based on financial news. Salvatore Carta et al. [13] take a novel approach by converting numerical market data into image representations, which are then processed by a deep reinforcement learning agent for trading decisions. Their results demonstrate that this method significantly outperforms the traditional *Buy and Hold* strategy.

This project integrates the ideas of Ansari and He with others, proposing a system based on three modules: the feature engineering module, the price and trend prediction module, and the trading module. The first will be responsible for selecting the input variables, preprocessing them, and compressing them using an autoencoder; the second will be responsible for estimating market trends; and the third will use the information on market trends and perform purchases and sales using reinforcement learning, specifically using the Rainbow DQN algorithm.

1.2 Objetives

This work is founded and developed on the potential for prediction, the challenge of recognizing market trends, and the buying and selling of assets based on knowledge of these trends. Thus, a solution is proposed for identifying trends through deep learning linear regression and the purchase and sale of assets based on deep reinforcement learning, with an improvement in memory in its neural network.

1.3 Document structure

The remainder of this document is organized as follows: Chapter 2 presents a summary of related work, aiming to present the most recent technologies proposed related to the topic and which in some way influenced this work. It also highlights gaps in some key works and how this work intends to fill these gaps. Chapter 3 presents a theoretical review, aiming to review the neural network architectures and machine learning algorithms used in this project. Chapter 4 presents the methodology proposed for this research project, Chapter 5 presents the experiments and results, and Chapter 6 presents a conclusion.

Chapter 2

Related Works

This chapter presents a consistent overview of works related to the problems discussed in the first chapter, aiming to provide an adequate conception of the most recent technologies and discussions on machine learning applied to finance. In this sense, we seek to identify possible gaps that may have been neglected by other researchers, situate the present work in the general context of the subject addressed, and form a solid understanding of the contributions already made.

2.1 Work related to feature selection

One of the most typical problems in financial analysis is the number of features that exist. features are the various data that will serve as input to the system, serving, in the specific context of finance, as sensors that will form the perception of the market, distinguishing and identifying the main signals and patterns relevant to buying and selling. These features may or may not provide useful information for the analysis. Numerous features have been proposed, studied, and tested in the financial literature over the years. The number of factors (or features) indicated in the literature is so large that it led Cochrane [30] to coin the term *factor zoo*. This zoo of factors includes a whole range of variables that provide information about a given asset. If the asset is a company's share, then examples of factors to be analyzed include the company's size, its profitability, the share's momentum, and its volatility, among others.

But despite the large number of factors, only a small subset of them will be truly informative. Taking a very large set of factors will inevitably result in redundant information and unnecessary noise in the study. An example is the study by Kozak, Nagel, and Santosh [31] who applied dimensionality reduction techniques to 130 different financial factors and showed that a very small number of principal components was able to represent almost all the information in the set.

Selecting a subset of the total set of features that is truly informative can be a computationally expensive task. Assuming that the total set of features has cardinality n, performing an exhaustive search to find the best subset implies analyzing 2^n subsets, which makes an exhaustive search completely impractical, even for moderate values of n.

Of the variable selection algorithms, some find the optimal subset and some find a suboptimal subset; there are those that are deterministic and always find the same subset, and there are those that are stochastic and can find a different subset in each round [32]. Of all the divisions and possible taxonomies, variable selection methods are usually divided into three broad groups: those based on filters, those based on *wrappers*, and those *embedded* . [33].

Filter-based algorithms focus on the statistical properties of features. Such algorithms assume that certain statistical properties make the feature more informative to the *machine learning* model. For example, a common metric to evaluate the informativeness of a feature is its variance, assuming that features with higher variance should be included in the selected subset, while those with lower variance should be disregarded. Another common metric is the correlation between the specific feature and the output, where those with a higher correlation would be more informative and would therefore be selected by the filter-based algorithm. The advantage of these algorithms is that their computational cost is relatively low since it is enough to only measure the statistical components and rank these variables according to these components. Furthermore, these methods usually avoid *overfitting* [33]. The disadvantage is that these algorithms disregard intrinsic relationships that may exist between the different inputs, which can be quite informative.

Wrapper-based methods evaluate features according to their performance in a given machine-learning model. A subset of features is initially formed, and at each iteration of the wrapper-based method, this subset of features will be modified, adding or subtracting features. However, for each iteration of the *wrapper based* method, the machine learning algorithm must be retrained. This is why these methods are generally computationally very expensive, and this is their main disadvantage. However, since these methods evaluate features together, wrapper-based methods can capture intrinsic relationships between features, which is an advantage over filter-based methods. [33]

Embedded methods make feature selection an integral part of the training process. Therefore, the selection of the optimal training set occurs concurrently with model training. [33] This approach makes embedded methods less computationally expensive than wrapper-based methods, and they can also evaluate the intrinsic relationships present in the features. Thus, such methods have significant advantages over filter-based and wrapper-based methods.

The work of Tong Niu et. al. [34] performed a two-stage feature selection; in the

first, a filter-based selection was performed, and in the second a wrapper-based selection was performed. For the feature to pass to the second stage, it was first necessary to pass through the first. Thus, the first stage already significantly reduces the number of features and the second reduces it even further. The technique used in the first stage is called *RReliefF-based feature selection*. The technique used in the second stage is known as *multiobjective binary grey wolf optimizer with Cukoo search*. To use the feature selection techniques, 16 features and three datasets were used: the Shangzheng Composite Index, the Shangzheng Fung Index, and the Dow Jones Industrial Average. The results showed that the most informative indicator was the closing price, followed by the indicator change (which is the difference between the closing price of the day and the closing price of the previous day), high (which is the maximum price of the day), and opening (which is the opening price of the day); it is noted that the volume indicator only ranked well when the Shangzheng Fund Index was used, and the Circulation Market Value indicator only appeared well in the rankings when the Shangzheng Composite Index was used.

Yaohao Peng et. al. [35] analyze 120 technical market indicators using two *wrapper*-based methods and one *embedded* method. Namely: Sequential Forward Floating Selection Algorithm (SFFS), Tournament (TS) Screening Algorithm, and Least Absolute Shrinkage and Selection Operator (LASSO). The data were collected from firms that composed financial indices of seven markets, namely: the United States (S&P 100 Index), the United Kingdom (FTSE 100 Index), France (CAC 40 Index), Germany (DAX-30 Index), Japan (Top 50 assets from the NIKKEY 225 Index), China (Top 50 assets from SSE 180 Index), and Brazil (Bovespa Index). The 24 indicators most selected by these methods are shown in Table 2.1. Since there are seven markets and three *feature selection* methods, each of the indicators can be selected a maximum of 21 times.

Furthermore, Peng's work seeks to select from the 51 most widely used and discussed technical indicators in the literature. These same indicators are used in the same seven data sets indicated above. Below, in table 2.2, are the results for these best-known indicators:

The work of Anwar Ul Haq et. al. [36] performs feature selection using three feature selection algorithms: *L1-LR-feature ranking*, *SVM-based feature ranking*, *Random Forest feature ranking, and multi-filter feature selection*. The total feature set consists of 44 technical market indicators calculated from the data of 88 stocks. After selecting the best subset of features, all these variables are fed into a deep generative model. The result was that the model obtained an accuracy of 59.44%; the twenty most selected features are shown in Table 2.3.

The work of Gang Ji et. al. [37] performs a pre-processing of the technical indicators using the *wavelet* transform, thus reducing the noise present in these indicators, which,

Table 2.1: Distribution of the number of times each technical indicator was selected by the feature selection methods across seven markets; work by Peng

Indicator's name	**Times selected**
DPO – Detrended price Oscilator	21
HULL – HULL Moving Average	16
MFM – Money Flow Multiplier	16
ADO – Acumulation Distribution Oscilator	15
APO –Absolute Price Oscilator	15
BIAS –Bias	15
DEMA – Double Exponencial Moving Average,	15
VOLAT – Volatility	15
MOM – Momentum	14
NVI – Negative Volume index	14
RVI – Relative Vigor Index	14
VOLR – Volume Ratio	14
ADX – Average Directional Index	13
BB BW – Bollinger Bands Width	13
DMI – Directional Movement Indicator	13
DSS – Double Smoothed Stochastic	13
STRSI – Stocastic RSI	13
FORCE – Force Index	13
ADL – Accumulation Distribution Line	12
ATRP – Average True Range Percent	12
DIU – Directional Indicator Up	12
MASS – Mass index	12
NATR – Normalized Average True Range	12
ULTOSC – Ultimate Oscilator	12

after undergoing a two-stage feature selection procedure, used to feed a random forest model. The feature selection method was based on *wrapper*, and the optimal subsets were formed by elimination. The result was that the *wavelet* transform improved the accuracy, precision, recall, and F1 by about 30% each. The indicators most selected by the algorithms are shown in table 2.4

The work of Fagner A. de Oliveira et. al. [38] differs in that it selects variables manually without resorting to selection algorithms; instead, it uses the opinions of financial analysts and investors with knowledge and experience in the area. In the work, a multiple-choice questionnaire was made available to 50 investors, and they were asked to choose which indicators, between fundamentalist and technical, were most efficient. The profile of the investors was as follows: 50% have more than 3 years of experience in the market, and 80% operate in the short and medium term; in addition, 100% of the investors claim to use technical indicators. These indicators, once selected, will feed a simple artificial neural network model, composed of an input layer, a hidden layer, and

Table 2.2: Number of times each technical indicator commonly used in the literature appears in the feature selection; work by Peng.

Indicator's Name	**Times selected**
ADO – Acumulation distribution oscilator	16
BIAS – Bias	14
DIU – Directional Indicator UP	14
MACD – Moving Average Conv/Div	14
VOLR – Volume Ratio	14
VOLUME – Volume	14
CCI – Commodity Channel Index	13
DID- Directional Indicator Down	13
LPR – Lowest Price Ratio	13
NVI –Negative Volume Index	13
STOCH K - Stochastic K%	13
WILL R - William's R%	12
REX – Rex Oscilator	12
VARR – Variance Ratio	12
ARATIO – A ratio	11
DISP – Disparity	11
DMI – Directional Movement Indicator	11
STOCH D - Stochastic D%	11
HPR – Highest Price Ratio	10
MPP –Moving Price level Percentage	10
PPO – Percent Price Oscilator	10
PSY – Psychological Line	10
ROC – Rate of Change	10
WMA – Weighted Moving Average	10
MOM - Momentum	09
VOLAT – Volatility	09

an output layer. From a group of 46 variables, only 15 were selected, 11 of which were composed of technical indicators. The indicators used are shown in Table 2.5. The results presented showed an average absolute percentage error of approximately 5.5% for the predictor system composed of the neural network.

However, even with variable selection carried out methodically and scientifically, the number of input variables is still reasonably large, which results in longer training times, which, depending on the machine being used, may take an unfeasible amount of time.

The work of Faraz et. al. [29] uses an autoencoder to perform the work of compressing the information from technical indicators. To do this, both the encoder and the decoder use LSTM cells. After the information has been encoded and the dimensionality of the input set has been reduced, the data is fed into a neural network composed of LSTM cells, which in turn predicts prices. The results were compared with the Generative Adversárial

Table 2.3: The most selected indicators by the methods proposed by Anwar Ul Haq

Indicator's name	**Times Selected**
PLUS-DI - Plus directional indicator	4
PPO - Percentage Price Oscillator	4
TRANGE - True Range	4
AD - Chaikin A/D Line	4
MOM - Momentum	3
CP - Close Price	3
MACD - Moving Average Conv/Div	2
MACDH - MACD Histogram	2
MACDS - MACD signal	2
ROC - Rate of change	2
ADX - Average DX	2
ADXR - ADX Rating	2
DX - Directional Movement Index	2
WMA - weighted MA	2
NATR - Normalized ATR	2
OP - Opening Price	2
SD - Slow Stochastic D%	2

Networks (GAN) model and were shown to have a lower error.

2.2 Work related to price prediction

The task of price prediction is usually difficult, as it requires an appropriate selection of variables that will feed the model, in addition to a very appropriate choice of machine learning architecture that will be necessary to understand price patterns. Generally, the prediction task can be of two types: simpler: the model simply predicts whether the value of future prices will be higher or lower than the current one, using a classification model, which is commonly called trend prediction; or more complex: if the model predicts the nominal value of future prices, and uses a regression model, which is better known as price prediction itself.

The work of Jigar Patel et. al. [1] performs a price prediction based on two stages. The first stage uses Support Vector Regression (SVR) to predict the future behavior of ten technical indicators, and the second stage takes as input the estimates from the first stage and applies them to a second SVR Artificial Neural Network (ANN) and a *random Forest*. The prediction result was better for the model using SVR+ANN. In this work, the chosen features are determined experimentally, and the author suggests that a feature selection technique could improve the model's performance. The author also suggests that

Table 2.4: Technical indicators most selected by the two-stage method proposed by Gang Ji

Indicator's name
WR – Willians %R
ULTISC – Ultimate Oscilator
CCI – Commodity Chanel Index
ROC – Rate of Change
FASTK – Stochastic fast K%
VAR – Variance
PRICE C – Price Change
RSI – Relative Strength Index

the model be fed with an attribute that informs the model of news values, according to the labels: good, great, bad, and very bad.

Yaohu Lin et. al. [2] implement a prediction model using as inputs thirteen candlestick patterns and eight patterns based on the Chinese Taoist trigram that reflect the day's fluctuations, in addition to twenty-one technical indicators. All these inputs go through an *ensemble machine learning* framework that includes logistic regression, *support vector machine*, *K-nearest neighbor*, *gradient-boosting decision tree,* and *long-short-term memory*. Thus, the machine learning model that best predicts the direction of future candlesticks is selected. The accuracy of the proposed model is over 60% and far surpasses the *buy and hold* technique, thus highlighting the relevance and originality of using trigrams as input variables.

Sreelekshmy Selvin et. al. [3] compare the performance of three types of machine learning: Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Convolutional Neural Network (CNN). The model is based on a sliding window. The window size is 100 minutes, and 90 minutes of information is used to predict 10 minutes ahead. The results show that CNN performs better than RNN and LSTM. The author argues that due to the dynamic nature of the market, it CNN performs better than RNN and LSTM since it relies only on current information, not giving as much importance to patterns learned in the past that may not always be realized. Furthermore, the author notes that it CNN can identify sudden changes in trends.

In the work of Zhanhong He et. al. [4] a deep learning model based on the combination of LSTM and convolutional neural networks with an attention mechanism to predict gold prices is proposed and developed. The results showed that using the LSTM layer first and then the CNN layer yields better results than using the CNN layer first and then LSTM. Furthermore, using an attention mechanism after the LSTM layer and before the CNN layer further improves the results. The author concludes that using CNN first causes a loss of information in the features—information that would be useful to the LSTM layer.

Table 2.5: Technical indicators selected by qualified personnel, according to the work of Fagner A. de Oliveira

Indicator's name
Opening Price
Closing Price
Maximum price
Minimum Price
Volume
Bovespa Index Percent Variation
NYSE dow Jones index
Ibovespa Volume
US ending stocks
MACD
Relative Strength Index
Stochastic index
OBV index
Moving Averages
Bollinger Bands
Momentum
Williams Percent Range (%R)

The author also suggests that the use of BiLSTM could provide even better results.

Ioannis E. Livieris et al. [5] also work under the aegis of CNN and LSTM neural networks for gold price and movement prediction. The proposed model exploits the ability of CNN networks to extract important information and learn the internal representation of time series, as well as the ability of LSTM networks to learn long-term and short-term dependencies. In this work, although the results for price prediction were very good, with errors very close to zero, the results for trend prediction were average.

Yanhui Liang et. al. [6] also focus their efforts on the gold price prediction. To do so, gold prices are first decomposed into components of different frequencies according to the *improved complete ensemble empirical mode decomposition with adaptive noise (ICEEMDAN)* algorithm, then each of these components passes through layers of neural networks such as CNN and LSTM, and finally, the price is estimated. The authors found that decomposing the price into multiple frequencies can increase the prediction accuracy, and the prediction effect is better than other decomposition methods.

Sima Siami-Namini et. al. compare the performance of a LSTM and BiLSTM in the price prediction task [7]. The author argues that BiLSTM is expected to perform better than simple LSTM in tasks such as guessing the next word in a sentence, however, it was not clear whether BiLSTM could perform better for the task of predicting time series. The results showed that the accuracy when using BiLSTM in this type of task was 37.78%

higher than when using LSTM. However, training BiLSTM is much more laborious than simple LSTM. The advantage of using BiLSTM on financial data is that the model will be trained in both directions (from input to output and from output to input).

The work of Zhuorui Zhang et. al. [8] innovates by focusing its price prediction work on volatility and strong correlations between cryptocurrencies. A prediction system based on three modules is proposed: the attentive memory module, based on a Gated Recurrent Unit (GRU) layer and a self-attention mechanism; the channel-weighting module; and the convolution and *pooling* module. The attentive memory module captures short- and long-term dependencies through Gated Recurrent Unit (GRU), however, this same component has difficulty in measuring the real importance of instances delayed in time. The self-attention mechanism, in turn, makes it possible to select important information in several time intervals. The channel-weighting module explicitly models the interdependencies between channels, which is important in the case of cryptocurrencies since the price of some is generally linked to the price of others. The convolutional module identifies the form that the features can take. The mean absolute percentage error, accuracy, and other validation measures were calculated, and it was found that the model proposed by these three modules outperformed all other known models, such as CNN, LSTM, or GRU, in all metrics.

The work of Iromie K. Samarasekara et. al. [9] focuses on the dynamics of stop-loss. While most research focuses on building a price prediction system to assist in purchases and sales, this work focuses on its predictor system to assist in placing Stop-Loss. To this end, the author uses CNN and LSTM networks to estimate the future trend, using this result as input for the future price prediction system. After that, both the future trend predictor and the price predictor will serve as input to a regression neural network, also based on CNN and LSTM, responsible for placing the Stop-Loss. This neural network was trained according to an algorithm. The results showed that the losses using the proposed strategy were much smaller than the losses of the usual strategies with the fixed stop-loss and the *trailing stop*.

2.3 Work related to the use of reinforcement learning

The approach of applying reinforcement learning to robots that operate in financial markets is particularly laborious since it involves several complicated design aspects. The first aspect is clearly defining the problem to be solved and what objectives should be achieved. Next, it is necessary to choose the architecture that will make up the agent. If it is composed of an artificial neural network, it is necessary to identify which types of neural networks are most suitable for solving the problem in question. It is also imperative

to define what rewards and punishments will be provided by the environment and that will teach the agent what to do. Another aspect is the engineering of the actions to be performed: in the financial environment, they can be just buying, selling, and doing nothing. However, the actions defined in this way do not take into account the probabilities of success of each action based on the conditions displayed by the market; for example, there are times when it is better to buy a lot since the probability of success is high, and there are times when it is better to buy a little if the probability of success is more uncertain. Another aspect is the engineering of *exploration vs exploitation*, which will define whether the agent should explore more unknown possibilities in the environment or whether it should act more in the field of already known possibilities; acting in the known field will make the agent an expert in that specific part of the environment, but staying only in what is known denies possibilities that could be very promising for the agent's objectives. Finally, there is the computational aspect, since training the agent is a time-consuming process that involves high computational processing costs.

The work of Yasmeen Ansari et. al. [10] proposes the approach of using deep reinforcement learning. To do so, a neural network is fed with both past market data and indicators, as well as future data and indicators. To estimate future data and feed the deep neural network responsible for reinforcement learning, another neural network is used to predict prices. More specifically, for each time step, the predictor module estimates the future candles related to that state. The neural network used to build the predictive module was a GRU with hyperbolic tangent as the activation function because it is less complex and faster than Long Short-Term Memory (LSTM). The architecture of the reinforcement learning module consists of a densely connected neural network with three hidden layers, each with 24, 12, and 8 neurons, followed by a *dropout* layer, which in turn is followed by a last layer with three neurons, representing the action space: buy, sell, and hold. The agent's reward is the total profit that the agent can extract from the market itself. The predictive model achieved good results for the root mean square error and mean absolute error. In addition, the agent based on reinforcement learning also achieved good profit results.

The work of Xing Wu et. al. [11] uses two machine learning methods as market trading strategies: *gated deep Q-learning trading strategy* and *gated deterministic policy gradient trading Strategy*. To this end, both strategies use *Gated Recurrent Unit (GRU)* a module for extracting financial information. The first consists of only one critic module, using two neural networks, one for Q and the other for Q^{target}. The second uses an actor module and a critic module, this time using four neural networks, two for each module. The results showed that the two proposed strategies generate more buy and sell signals than common strategies, in addition to being more profitable than even other state-of-the-art

reinforcement learning strategies. It is also important to say that the strategies capture more opportunities in the market to achieve more stable returns under more acceptable risk.

The work of Akhil Raj Azhikodan et. al. [12] builds a trend prediction module based on sentiment analysis to work in conjunction with the reinforcement learning module. This prediction module is based on a recurrent neural network that uses financial news data. The reinforcement learning module uses the *Deep Deterministic Policy Gradient* algorithm, which is an algorithm that uses stochastic behavior for good exploration of the environment, but seeks a deterministic policy; it is an algorithm that fits into the actor-critic category. The neural network responsible for sentiment analysis uses convolutional neural networks LSTM because a simple RNN would fail to identify discriminative phrases for sentiment analysis. With a convolutional layer, this type of phrase is more easily identified. In addition, the convolutional layer can better capture the semantics of the text when compared to a RNN. The results showed a test accuracy of around 96.88%, which proves that trends can be predicted based on the news.

The work by Salvatore Carta et. al. [13] proposes transforming numeric data into image data and then feeding it to an agent based on deep reinforcement learning to perform financial purchase and sale operations. The transformation of numerical market data into two-dimensional data occurs through CNNs since CNNs can clarify details about these images that would be much less noticeable in numerical format. To this end, about a hundred CNNs are generated with different initialization parameters. Then, each of these CNN will provide a different image as output; these outputs, in turn, will undergo a selection process based on reinforcement learning; the best outputs will then be used to feed a set of agents based on deep reinforcement learning and these, in turn, will perform the buying and selling actions. The results showed that the proposal far outperforms the *Buy and Hold* strategy. The agent was able to increase an initial investment of USD 200,000.00 in the S&P market by USD 45,050.00, representing a return of almost 50% over seven years.

Yang Li et. al. [14] suggest in their work some techniques to filter some types of noise that notably affect the performance of automated trading based on agents trained by reinforcement learning. First: market trends are influenced by news released in the media, and an agent based on reinforcement learning usually has raw market data and technical indicators as input variables; the author suggests that the agent only starts trading at a time when the immediate effect of the news has already passed. Second, the market with low volatility is being traded by individual and non-institutional investors, which generates noise that is disadvantageous for the agents. The work also innovates by proposing a more complete state of actions: instead of assigning the agent only three

actions (buy, sell, and hold), a set of actions is assigned. Going from $-n$ to n, thus varying the volume of assets that are being bought and sold. Two architectures are proposed for the agent: an extended version of DQN, which uses an LSTM layer as the output layer; and an A3C architecture that also uses LSTM layers as output layers. The results showed that both the DQN-extended and A3C-extended cases obtained better performance than the original agent models. The results also show that the A3C-extended model performs better than DQN-extended, the author explains that this is because the models are too complex to learn the Q-value function, and the advantage of A3C is that it is based on both the policy and the Q-value function.

2.4 Innovations and relationships of this project with other works

This project has three distinct modules: the feature engineering module, the trend prediction module, and the trading module. The first module is related to work related to variable selection and noise and dimensionality reduction. The second module is related to work related to trend prediction, and the third module is related to work related to reinforcement learning.

The input variables for this project were selected based on the best-selected variables chosen from the works of Peng et. al. [35], Haq et. al. [36], Gang Ji et. al. [37] and De Oliveira et. al. [38]. These variables were chosen to cover information from different types of technical indicators, namely: trend followers, oscillators, volume indicators, and cycle indicators. In this way, all families of indicators are covered.

Once the set of variables has been selected and useful information has been ensured in this set, the information passes through a process of filtering the market noisy and dimensionality reduction, for this aim, an autoendcoder is proposed and differently of the work of Faraz et al. [29], The proposed architecture is made with BiLSTM; furthermore, his work dos not present a trading module and a sophisticated architecture for trend prediction like this one. So the project is responsible for structuring the architecture of the price and trend prediction tools. To this end, the works of Samarasekara et. al. [9], Siami-Namini et. al. [7], Zhanhong He et. al. [4] served as inspiration, but unlike their work, this one applies market noise reduction and it has a trading prediction module. Samarasekara's work builds a trend predictor and a price predictor, where the output of the first serves as input to the second. The trend predictor is built with a convolutional neural network followed by an LSTM network, while the price predictor is built with the LSTM network before the convolutional network. The work of Samarasekara et. al. is inspired by the work of Zhanhong He et. al, where the author builds a price predictor

in the same way as Samarasekara to predict the price of Gold. However, one of the differences between the other is that the first uses technical indicators as inputs for the neural network, which is not included in Zhanhong He's work. However, an important consideration appears at the end of his work: the use of BiLSTM networks can improve the performance of the predictor. Siami-Namini's work builds a price predictor using the BiLSTM neural network and compares it with the LSTM network; the results show that the BiLSTM network achieved an error about 37% lower than the LSTM network, which shows that the approach of using this type of neural network proved to be effective, despite the increase in training time. The proposal of this project consists of using a predictor composed of 2 types of layers: BiLSTM layers and convolutional layers, and the use of linear regression to label the data.

Once properly trained, the predictor modules will then provide their estimates as useful information for the last module, the negotiator module. Here the probability of success of the trading instruments should be significantly increased with the use of the future price estimates given by the predictor module. This perspective is inspired by the work of Yasmeen Ansari et. al. [10], where the predictor modules were built using GRU technology. However, this work brings two novelties: the use of Rainbow-DQN technology, which currently consists of one of the best reinforcement learning algorithms, in addition to a specific improvement in the algorithm that is an adaptation for time series, which is the memory buffer.

Thus, the total set of proposed structures, formed by the trading and risk management instruments fed by the future price estimates of the predictors and coded technical indicators, is believed to be innovative and has a good chance of generating good results.

Table 2.6 provides a comparative summary between some related works cited in this chapter and the proposals of this work.

Table 2.6: Comparison of this work with related works

Works	Dimensionality Reduction	CNN Predictor	RNN Predictor	BiLSTM Predictor	Convolutional LSTM Predictor	RL-Based Trading	Rainbow DQN
Ansari [10]			√			√	
Iromie [9]		√		√			
Faraz [29]	√		√				
Namini [7]			√	√			
Zhanhong He [4]		√	√				
Selvin [3]		√	√				
Liang [6]		√	√				
Livieris [5]		√	√				
Zhang [8]		√	√				
This work	√	√	√	√	√	√	√

Chapter 3

Theoretical Foundation

This chapter presents a theoretical review of the basic concepts related to the main components used in this work. It addresses the essence of the theory about neural networks and reinforcement learning.

3.1 From biological to artificial neuron

In the animal kingdom, neurons act as small information-processing units. Information is captured from the environment by the senses of perception and transformed by these nerve cells into useful information for survival. For example, a sensation of heat generated by fire in contact with the skin is converted into reflex movement of our body through them.

Nerve cells have a communication system between them called synapses. These synapses are gaps at the ends of neurons, precisely between the axon of one and the dendrite of another. Synapses establish communication between two neurons in sequence using neurotransmitters. When the action potential of the presynaptic neuron reaches the axon terminal, excess calcium in the region causes vesicles loaded with neurotransmitters to release these neurotransmitters into the synaptic cleft, thus altering the electrical configuration of that region and activating the sodium and potassium pumps of the next neuron, generating small electrical signals in the membrane of the postsynaptic neuron. Synapses can be excitatory or inhibitory. When excitatory, they facilitate the formation of an action potential in the postsynaptic neuron. When inhibitory, they hinder the formation of this action potential.

The various synapses that reach the neuron form a set of electrical signals. These signals usually add up or subtract on the cell surface of the dendrites and the cell body of the neuron, depending on the excitatory or inhibitory nature of these synapses.

Finally, it is in the implantation cone that the action potential will form or not. The implantation cone is the initial region of the axon. The action potential is what the neural signal itself is called. The implantation cone therefore establishes the decision as to whether or not to continue the propagation of the neural signal. Usually, if the set of signals that reach the neuron, added together, manage to reach the threshold of -55 mV, then the signal will propagate through the axon; if not, it will attenuate until it completely disappears.

Neurons adjust synapses through a process called synaptic plasticity. Synaptic plasticity refers to the ability of synapses to modify themselves in response to neural activity and is one of the biological bases for learning. There are two main mechanisms of synaptic adjustment: synaptic strengthening (potentiation) and synaptic weakening (depotentiation). These processes are related to the electrical and chemical activity that occurs in synapses.

Synaptic potentiation occurs when a presynaptic (sending) neuron is persistently activated, thus strengthening the corresponding synapse. This occurs due to biochemical changes that lead to an increase in the efficiency of neurotransmitter transmission. In this process, for example, the biochemical receptors on the cell surface of dendrites increase. In synaptic depotentiation, on the other hand, presynaptic activity is reduced or ceased, and the corresponding synapse may suffer a weakening, leading to a decrease in the number of receptors in the dendrites or even to the phagocytosis of the dendrites by glial cells.

In 1943, neurophysiologist Warren McCulloch and logician Walter Pitts, realizing the complex dynamics of neurons, proposed one of the first models of an artificial neuron [39], based on binary logic. To this end, some hypotheses have been established. The first concerns the all-or-nothing logic of the neuron, which states that it either fires the action potential with full force or does not fire at all. The second states that it is a sum of synapses that enables the firing of the action potential. Other hypotheses from the work of McCulloch and Pitts concern the inhibitory nature of certain synapses.

In 1957, Rosenblatt described the perceptron as a single-layer neural network model capable of learning and recognizing patterns [15]. The perceptron is a type of artificial neuron that receives multiple weighted inputs and produces a binary output, based on an activation function. Rosenblatt demonstrated that when the synaptic weights of the perceptron are adjusted correctly, it can learn to correctly classify linearly separable patterns. This made the perceptron one of the first machine learning algorithms capable of performing pattern recognition tasks.

In Rosenblatt perceptron, synapses, both excitatory and inhibitory, are modeled according to a weight w, the cell body, responsible for integrating the various signals coming

from the synapses, is modeled by a summation, while the all-or-nothing law, which says that a neuron always fires with full force or does not fire at all, is modeled by an activation function $\varphi(\cdot)$, which in the specific case of Rosemblatt's perceptron is a step function, that is: the function returns 1 if its argument is greater than 0, and 0 if its argument is less.

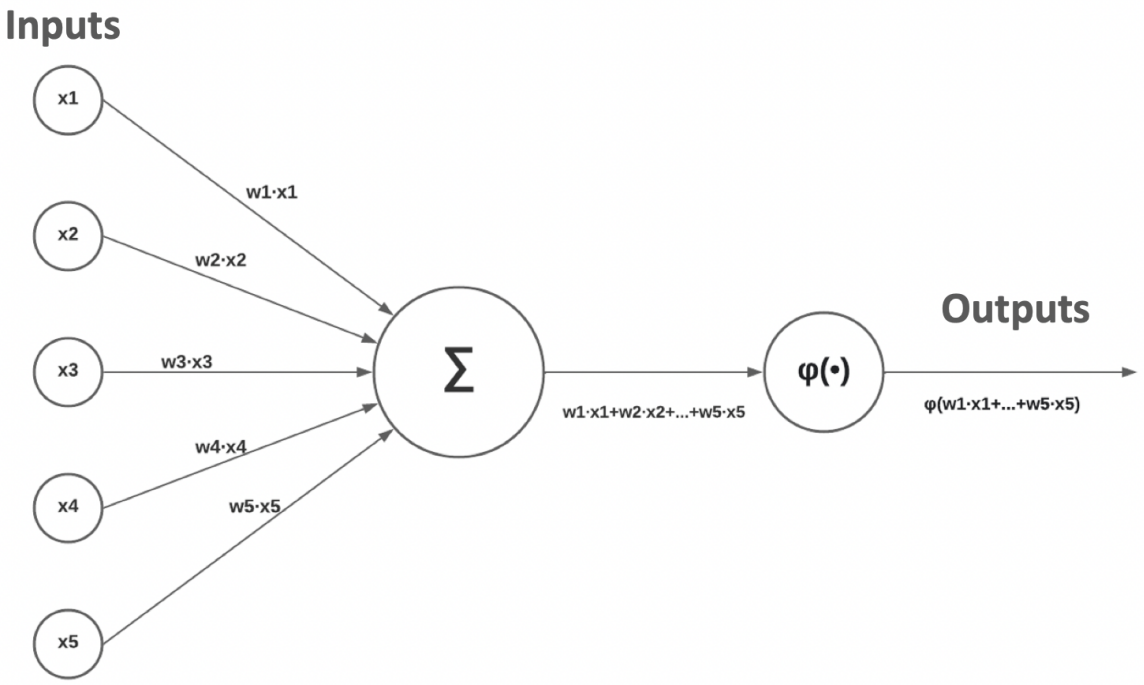


Figure 3.1: Rosenblatt's perceptron for binary classification, a schematic idea conceived from the article *The perceptron: a probabilistic model for information storage and organization in the brain.* by Rosenblatt [15],1958.

However, due to its inability to solve non-linearly separable problems, such as the famous XOR problem, Rosemblatt's perceptron was severely criticized by Misnky and Papert with the publication of their book: *Perceptrons: introduction to computational geometry* [40]. Even so, Rosenblatt's perceptron represents a crucial milestone in the history of artificial intelligence, as it introduced a neural network model capable of learning and recognizing patterns. Its contribution was fundamental to the later development of machine learning algorithms. The perceptron demonstrated that a single-layer neural network, with appropriate adjustment of synaptic weights, could achieve linear pattern separation. Although the perceptron is a simple model, its importance lies in the fact that it was the basis for the development of more complex and sophisticated neural networks. Through Rosenblatt's work, the field of artificial intelligence gained a powerful concept that allowed the field to expand and paved the way for research and application of machine

learning algorithms in a wide variety of areas, from computer vision to natural language processing.

3.2 Artificial Neural Networks

Artificial neural networks (ANNs) are computational models inspired by and developed from the Rosemblatt Perceptron and are often referred to as multilayer perceptrons. They are composed of artificial neurons, which are their basic processing units; they are interconnected by weighted synapses. These synapses are responsible for transmitting signals between neurons and have numerical values associated with them, called synaptic weights.

An ANN is organized into layers, with the input layer being the first layer, followed by one or more hidden layers, and, finally, the output layer. Each layer is composed of several interconnected neurons. The input layer receives the input data, which is propagated through the network to the output layer, where a response or prediction is generated. Figure 3.2 shows an example of a neural network with three neurons in the input layer, a hidden layer of four neurons, and two outputs.

The activation of an artificial neuron is calculated by an activation function. A commonly used function is the sigmoid function, represented by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

This function compresses the input value into a range between 0 and 1, allowing the representation of probabilities. In addition to the sigmoid function, other activation functions such as the ReLU (Rectified Linear Unit) function and TANH (Hyperbolic Tangent) are widely used in modern neural networks. These functions are represented by the equations 3.2 and 3.3.

$$ReLU(x) = \begin{cases} x, \text{se } x \geq 0 \\ 0, \text{se } x < 0 \end{cases} \tag{3.2}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.3}$$

3.2.1 The Dense Layer

Signal propagation in an ANN occurs from the input layer to the output layer. Each neuron in the input layer is connected to a neuron in the hidden layer through weighted synapses; these connect to neurons in the next hidden layer, until reaching the output layer. The activation value of each neuron in the hidden layer is calculated as the weighted

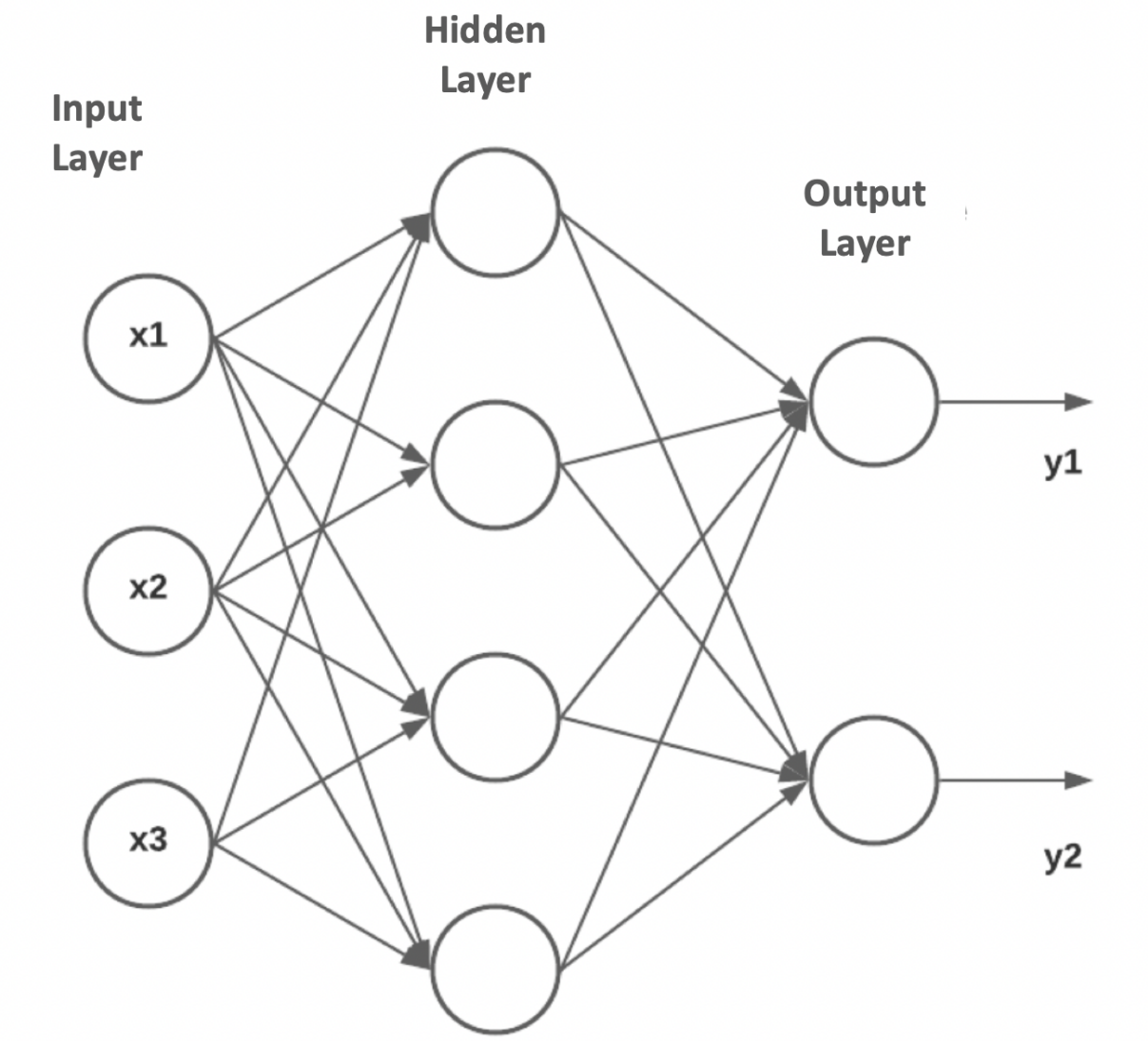


Figure 3.2: Multilayer perceptron, a schematic idea conceived from the article *Learning representations by back-propagating errors*, by Rumelhart et. al. [16], 1986, with an input layer, a hidden layer, and an output layer.

sum of the inputs, followed by the application of the activation function, as illustrated by the equation 3.5.

$$z_j = \varphi\left(\sum_{i=1}^{n} w_{ji} \cdot x_i\right) \tag{3.4}$$

Where z_j is the activation of the neuron in the hidden layer, w_{ji} is the synaptic weight between the input neuron i and the hidden neuron j, and x_i is the activation of the input neuron i.

It is common to add a term known as a bias to the inputs adjusted by the synaptic

weights. Bias is an additional parameter in each neuron that allows the neuron's activation point to be adjusted. While synaptic weights control the contribution of inputs to a neuron's output, bias adjusts the neuron's activation threshold, allowing the neural network to make fine adjustments to the neuron's response, regardless of the inputs. With the addition of the bias term, the equation 3.4 becomes:

$$z_j = \varphi\left(\sum_{i=1}^{n} w_{ji} \cdot x_i + b_j\right) \tag{3.5}$$

In matrix terms:

$$\mathbf{z} = \varphi\left(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}\right) \tag{3.6}$$

Learning in an ANN occurs through the adjustment of synaptic weights and bias. The goal is to find the weights that minimize a cost function, which measures the error between the outputs predicted by the network and the correct values.

A commonly used algorithm for adjusting weights is Gradient Descent. It uses the derivative of the cost function with respect to the weights to update the values of these weights, moving in the direction of the greatest decrease. The process of adjusting the weights is repeated iteratively until the neural network reaches a satisfactory level of accuracy.

3.2.2 The stochastic gradient descent method

The learning of a neural network consists of finding the synaptic weights w that most closely approximate its output to our objectives. To do this, the error between the expected output y and the output that the neural network produces $\hat{y}$ is generally used. A cost function E is generally used to calculate and mathematically define the error. The neural network is then said to learn as soon as the weights $\mathbf{w}$ that minimize the cost function $E(\mathbf{w})$ are found.

The most common method for minimizing the function $E(w)$ is the stochastic gradient descent method.

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \tag{3.7}$$

The equation 3.7 guarantees us that for a sufficiently small η and a sufficiently large number n of iterations, the cost function E at point w will be minimal. The main idea of the gradient descent method is to update the parameters w in the direction of the negative partial derivative of the loss function. The partial derivative of $E(w)$ indicates the direction of greatest growth of the function, therefore, by following the negative gradient,

one moves in the direction of greatest decrease and, therefore, gets closer and closer to the minimum.

3.2.3 The *Backpropagation* algorithm

The backpropagation algorithm or *backpropagation* is the most widely used algorithm in training artificial neural networks (ANNs) to adjust synaptic weights. It was introduced in the paper by Rumelhart, et. al., Learning Representations by Backpropagating Errors [16], from 1986. This algorithm uses the calculation of the gradient of the cost function with respect to the weights to update the values of the weights iteratively.

The first step of the algorithm consists of assigning random values, usually between minus one and one, to all of the parameters w of the neural network, and then calculating the output of the neural network based on these values and later comparing this calculated output with the expected output using a cost function E. Some common cost functions are listed below:

The Cross-Entropy:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} y_i ln(\hat{y}_i(\mathbf{w})) \tag{3.8}$$

Mean Squared Error:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i(\mathbf{w}))^2 \tag{3.9}$$

Where N is the number of samples, y is the expected response, and $\hat{y}$ is the response calculated by the neural network. The next step consists of solving the derivative of the equation 3.7:

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

For this, the chain rule is used:

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial E(\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}(\mathbf{w})}{\partial \mathbf{w}}$$

For the **cross-entropy** function we have:

$$\begin{aligned}\frac{\partial E(\hat{y})}{\partial \hat{y}_k} &= -\frac{\partial}{\partial \hat{y}_k}\sum_{i=1}^{N} y_i log(\hat{y}_i) \\ &= \frac{\partial}{\partial \hat{y}_k}(-y_1 log(\hat{y}_1) - y_2 log(\hat{y}_2) - \cdots - y_k log(\hat{y}_k) - \cdots - y_n log(\hat{y}_n) \\ &= -\frac{\partial}{\partial \hat{y}_k} y_k log(\hat{y}_k) \\ &= -\frac{y_k}{\hat{y}_k}\end{aligned}$$

It can then be said that:

$$\frac{\partial E(\hat{y})}{\partial \hat{\mathbf{y}}} = -(\mathbf{y} \oslash \hat{\mathbf{y}}) \tag{3.10}$$

Where ($\oslash$) is the Hadamard division. For the **Squared error** function we have:

$$\begin{aligned}\frac{\partial E(\hat{y})}{\partial \hat{y}_k} &= \frac{\partial}{\partial \hat{y}_k}\frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_{ik})^2 \\ &= \frac{\partial}{\partial \hat{y}_k}\frac{1}{2}((y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2 + \cdots + (y_k - \hat{y_k})^2 + \cdots + (y_n - \hat{y_n})^2) \\ &= \frac{\partial}{\partial \hat{y}_k}\frac{1}{2}(y_k - \hat{y_k})^2 \\ &= (y_k - \hat{y_k})\end{aligned}$$

In matrix terms:

$$\frac{\partial E(\hat{y})}{\partial \hat{\mathbf{y}}} = \mathbf{y} - \hat{\mathbf{y}} \tag{3.11}$$

3.2.4 The Softmax function

The *softmax* function is commonly used in neural networks as an activation function in the output layer, especially in multi-class classification problems. It converts a vector of real numbers into a probability distribution, assigning probability values to each class.

Given an input $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the softmax function calculates the probabilities $p = (p_1, p_2, \ldots, p_n)$ using the formula:

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad \text{para } i = 1, 2, \ldots, n \tag{3.12}$$

Here, e is the base of the natural logarithm (Euler's number), and the sum in the denominator is performed over all elements of the vector x. The softmax function ensures that the resulting probabilities are non-negative and sum to 1. In vector form, the function

becomes:

$$\sigma_s(\mathbf{x}) = softmax(\mathbf{x}) = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} \tag{3.13}$$

The softmax function is called σ_s because it is a generalization of the sigmoid function. The interpretation of the softmax function is that it provides a measure of confidence or probability for each class, based on the input values. Larger values of x result in higher probabilities after applying the softmax function, increasing the confidence in that particular class.

The softmax function is widely used in multiclass classification problems, along with the cross-entropy loss function (*cross-entropy loss*), to train neural networks. It allows the network to assign probabilities to each class, making it easier to select the likely class or classify based on a threshold value.

3.2.5 The Dropout Layer

The *dropout* layer is a technique commonly used in deep neural networks to mitigate the problem of overfitting. Overfitting occurs when a neural network becomes too specialized on the training data and consequently performs poorly on previously unseen data.

Dropout is implemented as an additional layer in a neural network, usually between densely connected layers. The main idea is to randomly turn off a set of neurons during training. This means that these units do not contribute to the computation of the gradients during the backpropagation process.

Formally, during training, each output unit of a *dropout* layer is multiplied by a random binary variable r, which is equal to 1 with a probability p and 0 with a probability $1 - p$. This probability p is a hyperparameter that determines the *dropout* rate, that is, the proportion of units that are turned off.

The effect of *dropout* is similar to training a set of smaller, independent neural networks in parallel. This technique prevents units from becoming too dependent on each other and helps reduce overfitting since the neural network cannot rely too heavily on a specific subset of units to make predictions.

The *dropout* layer is a valuable tool for training deep neural networks. It helps improve the generalization ability of the model, making it more robust and less sensitive to variations in the input data. In addition, *dropout* can also reduce training time since it forces the neural network to learn more efficient representations that are shared across units.

3.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of deep neural networks that are widely used for data that have spatial patterns, such as images. However, in addition to images, they can be very useful in finding patterns in time series.

The main idea behind CNNs is to apply convolution filters to different regions of an image or signal. These filters are small matrices that are slid over the input to calculate local dot products. This convolution operation allows the network to identify important features, such as edges, textures, and patterns, at different scales and positions in the image.

3.3.1 Convolution

The convolution operation is the main building block of CNNs. It involves applying convolutional filters to an input to extract relevant features. Convolution is defined as superimposing a sliding window (filter or kernel) over the input, multiplying the window values by the corresponding values in the input, and summing the results.

For a two-dimensional input (such as a grayscale image), convolution can be represented mathematically as follows:

Given two functions $f(x)$ and $g(x)$, the one-dimensional convolution between them is defined as:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau \tag{3.14}$$

Where $*$ denotes the convolution operation and τ is the integration variable. This equation expresses that for each value of x, the convolution is obtained by integrating the product of the two shifted functions.

A more common way to express convolution is in terms of discrete summation, where the integral is replaced by a summation:

$$(f * g)(x) = \sum_{k=-\infty}^{\infty} f(k)g(x - k) \tag{3.15}$$

Here, the variable k assumes integer values, and the convolution is calculated by summing the products of the samples of the shifted functions f and g.

3.3.2 Convolutional Layers

A convolutional layer in a CNN is composed of several convolutional filters. Each filter is responsible for extracting a specific feature from the input. Each convolutional filter is

applied to the input, resulting in a feature map.

As in densely connected neural networks, the first step in learning the network in CNNs is the propagation step. During the propagation step, the activations in each layer of the network are calculated.

Let $\mathbf{y}$ be the output of the convolutional layer, $\mathbf{k}$ be the vector with the kernels (or weights) to be learned, $\mathbf{x}$ be the input vector and $\mathbf{b}$ be the bias. We will have the following equation for forward propagation.

$$\mathbf{y} = \mathbf{k} * \mathbf{x} + \mathbf{b} \tag{3.16}$$

It is important to note that if $\mathbf{x}$ has N elements and $\mathbf{k}$ has M elements, $\mathbf{y}$ will have a number of elements Q such that $Q = N - M + 1$

Abrindo a equação 3.16:

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_q \end{pmatrix} = \begin{pmatrix} x_0 k_\tau + x_1 k_{\tau-1} + \cdots + x_\tau k_0 \\ x_1 k_\tau + x_2 k_{\tau-1} + \cdots + x_{\tau+1} k_0 \\ \vdots \\ x_{n-\tau} k_\tau + x_{n-\tau+1} k_{\tau-1} + \cdots + x_n k_0 \end{pmatrix} + \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-\tau} \end{pmatrix} \tag{3.17}$$

In the equation 3.17 n is the index of the last element of $\mathbf{x}$, τ is the index of the last element of $\mathbf{k}$ and q is the index of the last element of $\mathbf{y}$, where it can be inferred that: $q = n - \tau$.

3.3.3 Max Pooling Layers

A camada *Max Pooling* é uma técnica comumente associada às redes neurais convolucionais (CNNs) para reduzir a dimensionalidade dos mapas de características gerados por essas camadas. Ela ajuda a extrair características importantes e a tornar a representação mais robusta e invariante a pequenas variações de posição.

The *Max Pooling* operation is applied to each feature map individually. For each *pooling* region, the maximum value is selected and used as the representation of that region. This is done by moving a window (usually of size 2x2) with a stride defined by the CNN architecture. The window slides across the feature map and the maximum value in each region is retained.

The main advantage of *Max Pooling* is the reduction of the dimensionality of the data, which helps to reduce the number of parameters in the network and avoid overfitting. In addition, the *pooling* operation helps to capture features that are invariant to small translations. For example, if an object appears in different positions in different images,

Max Pooling will select the same representation, as long as the relative position of the object is preserved.

The *Max Pooling* layer also helps to make the representation more robust to noise and distortion. Because the maximum value in each region is retained, small variations in the original feature values have less impact. This helps smooth out noise and reduce sensitivity to small disturbances.

3.4 Recurrent neural networks

Recurrent neural networks (RNNs) are a type of neural network architecture widely used in a variety of fields, such as natural language processing, speech recognition, machine translation, and other tasks involving sequential data. Unlike traditional neural networks, RNNs have feedback connections, i.e., layers at the output or closest to the output connected to layers at the input or closest to the input. This feature allows them to process sequential information.

An important feature of RNNs is their ability to handle inputs of varying lengths. This means that the networks can receive sequences of different sizes and process them efficiently. Each element of the sequence is processed in order, and the output generated in one time step is used as input for the next time step. In this way, RNNs can capture temporal and contextual dependencies in sequences.

The main characteristic of RNNs is that they process data in several time steps in the same way and using the same parameters. This means that the data undergoes the same processing a certain number of times. The figure below exemplifies the unfolding of a recurrent neural network in time.

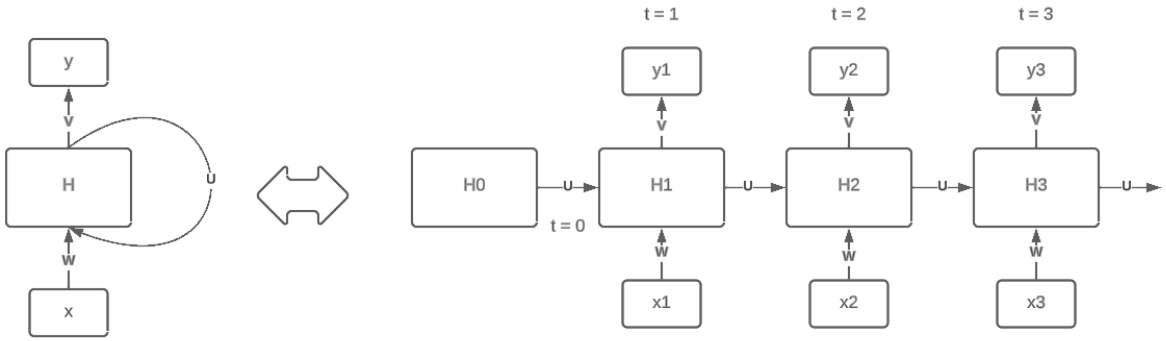


Figure 3.3: Standard RNN, on the left, and time-unfolded RNN, on the right, a schematic idea conceived from the article *Backpropagation through time: what it does and how to do it* by Werbos et. al. [17],1990.

The diagram, both on the left and the right, exemplifies the following mathematical relationship:

$$y_t = g(\mathbf{V} \cdot \mathbf{H_t})$$

$$H_t = f(\mathbf{W} \cdot \mathbf{x_t} + \mathbf{U} \cdot \mathbf{H_{t-1}})$$

W is a weight associated with the inputs of the network, U is a weight associated with the previous hidden state of the network, and V is a weight associated with the current hidden state of the network. The problem is that to obtain the output of the neural network it is necessary to compute the values for each desired time step. So:

$$\begin{aligned} y_1 &= g(\mathbf{V} \cdot f(\mathbf{W} \cdot \mathbf{x_1} + \mathbf{U} \cdot \mathbf{H_0})) \\ y_2 &= g(\mathbf{V} \cdot f(\mathbf{W} \cdot \mathbf{x_2} + \mathbf{U} \cdot \mathbf{H_1})) \\ y_3 &= g(\mathbf{V} \cdot f(\mathbf{W} \cdot \mathbf{x_3} + \mathbf{U} \cdot \mathbf{H_2})) \end{aligned}$$

It is important to realize that the weights W, U, and V do not change as the time step progresses.

3.4.1 The *Vanishing Gradient* Problem

The vanishing gradient problem is a common problem in recurrent neural networks (RNNs) and is an important limitation to consider in both the propagation and backpropagation phases of these architectures. In RNNs, the vanishing gradient occurs when the weights computed during backpropagation multiply, causing the gradients to become close to zero as they propagate through the network's time steps, making learning long-term dependencies difficult.

This problem in RNNs often arises due to the feedback nature of these networks. In RNNs, the output at a given time step is used as input to the next time step, allowing information to propagate down the sequence.

This problem can be particularly damaging when data sequences have very long-term dependencies, where information from much earlier time steps is meaningful for future time steps. For example, in text generation, dependencies between distant words can be crucial to maintaining coherence and cohesion. If the gradients tend to zero, the RNN will have difficulty learning these long-term temporal relationships.

To mitigate the problem of vanishing gradients in RNNs, several techniques have been developed. One of the most important advances is gated memory cells, such as Long

Short Term Memories (LSTMs) and Gated Recurrent Units (GRUs). These architectures incorporate information flow control mechanisms through gates, which help regulate the flow of information across time steps and reduce the vanishing gradient.

3.4.2 Long Short Term Memory

Long Short-Term Memory (LSTM) networks are a special type of recurrent neural network designed to handle long-term dependency in data sequences. They were developed to overcome the vanishing gradient problem in traditional recurrent neural networks. Figure 3.4 shows an example of an LSTM Cell.

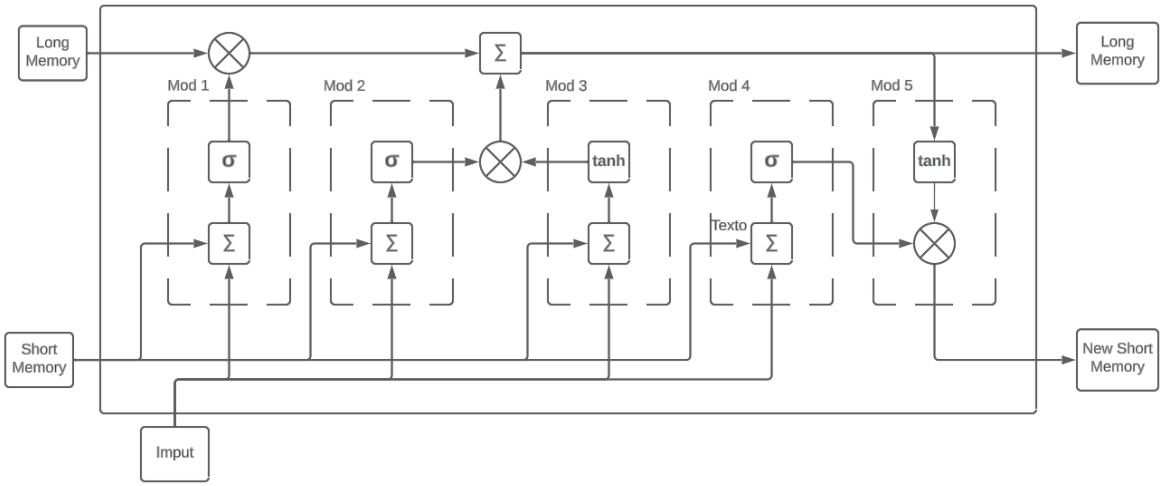


Figure 3.4: LSTM cell, schematic idea conceived from the article *Long short-term memory. Neural computation* by Hochreiter and Schmidhuber [18],1997.

The most important part of an LSTM cell consists of the long memory channel, represented at the top of the cell. In this channel, information can only be forgotten or added. It is important to note that the channel does not have any weights to be learned, and this is precisely what makes the LSTM cell resistant to the vanishing gradient problem. In Figure 3.4, the long memory channel receives, right at the cell input, the long memory of the previous time step, and at the end of the cell the long memory is modified according to the current time step.

The first process to change the cell's long memory is forgetting. The forgetting mechanism is represented by module 1 (Mod 1) in Figure 3.4. The short memory of the previous time step is modified by a weight and added to the input of the current time step, also modified by a weight. The result is passed to a sigmoid activation function, which will have a value between 0 and 1. This value, between 0 and 1, multiplied by the current

value of the long memory will be responsible for erasing it, partially erasing it, or leaving it intact. This is why module 1, represented in figure 3.4 is known as the forget gate.

$$f_t = \sigma(\mathbf{W_f} \cdot \mathbf{x_t} + \mathbf{U_f} \cdot \mathbf{h_{t-1}} + \mathbf{b_f}) \tag{3.18}$$

The equation 3.18 represents the operations performed by the forget gate. Where f_t represents the output of the forget gate, x_t represents the input of the LSTM network for the current time step, h_{t-1} represents the short-term memory of the previous time step, W_f represents the forget weights related to the input, U_f represents the forget weights related to the short-term memory of the previous time step, and b_f represents the forget bias. It is precisely f_t that will be multiplied by the values of the long-term memory and will make those values forgotten, kept, or kept in part.

The second process to alter the long-term memory is represented by module 2 and module 3 in Figure 3.4. Module 3 linearly combines the current input and short-term memory to create a potential memory, while module 2 determines what percentage of this potential memory should be stored, also using a linear combination between the input and short-term memory. Module 2, which controls the percentage of new information to be stored, is known as the input gate, while module 3 is the new information itself.

$$i_t = \sigma(\mathbf{W_i} \cdot \mathbf{x_t} + \mathbf{U_i} \cdot \mathbf{h_{t-1}} + \mathbf{b_i}) \tag{3.19}$$

$$\bar{c_t} = tanh(\mathbf{W_c} \cdot \mathbf{x_t} + \mathbf{U_c} \cdot \mathbf{h_{t-1}} + \mathbf{b_c}) \tag{3.20}$$

The equations 3.20 and 3.19 represent respectively the new information to be added and the input gate, responsible for determining what percentage of the new information should be remembered; i_t represents the output of the input gate and $\bar{c_t}$ is the new information. The terms with index $_i$ represent the input weights and the terms with index $_c$ represent the weights related to the new information to be added to the cell.

Finally, the new long-term memory of the cell is formed. With forgetting having been governed by the forgetting gate, and the new information added, by the input gate. The new long-term memory of the cell is then:

$$c_t = \mathbf{c_{t-1}} \odot \mathbf{f_t} + \mathbf{i_t} \odot \mathbf{\bar{c}_t} \tag{3.21}$$

Where c_{t-1} is the old long-term memory, also known as the previous state of the cell; and c_t is the current long-term memory, or the current state of the cell.

The last process that occurs in the LSTM cell is the formation of the current short-term memory, which is the output of the cell. Modules 4 and 5 govern this process. Module 5

consists of the formation of the short-term memory itself, where the values from the long-term memory are passed, without multiplication by weights, to the short-term memory. Module 4 governs the percentage of this long-term memory that should be passed, this percentage is controlled by the current input and the previous short-term memory. The output of module 4 is known as the output gate of the LSTM network.

$$o_t = \sigma(\mathbf{W_o} \cdot \mathbf{x_t} + \mathbf{U_o} \cdot \mathbf{h_{t-1}} + \mathbf{b_o}) \tag{3.22}$$

$$h_t = \mathbf{o_t} \odot tanh(\mathbf{c_t}) \tag{3.23}$$

3.5 Bidirectional Neural Networks

Bidirectional neural networks were introduced by Schuster and Paliwal in 1997, in the article *Bidirectional Recurrent Neural Networks* [19]. In this article, the authors suggest a recurrent neural network for the direction that goes from the past to the future, combined with one that goes from the future to the past. The outputs are then influenced by both layers that memorize information from the past and by layers that memorize information from the future.

The bidirectional neural network proposes to take into account that intrinsic relationships between sequential data in the direction from the future to the past may contain relevant information. There may be long-term dependencies between the current state of the system and a future state.

Taking into account both types of paths, both the one that goes from the past to the future and the one that goes from the future to the past establishes two independent processing units that can then be combined to obtain a final representation.

In the context of financial time series, for example, future context can be particularly important in determining important information in predicting future prices.

3.5.1 Forward Propagation

In a simple recurrent neural network, the output of the network at the current time step is always determined by the hidden state of the previous time step and the current input. The output of the next time step will be determined by the future input, the current hidden state, and so on. This relationship can be described as follows:

$$y_t = g(h_t, V) \tag{3.24}$$

$$h_t = f(x_t, h_{t-1}, W) \tag{3.25}$$

Where V is a weight that operates on h_t and W is a weight that operates on x_t and h_{t-1}.

For a bidirectional neural network, the current output is determined by two different hidden states. The first hidden state goes from the past to the future and the second goes from the future to the past. It is important to note that, by the very definition of a bidirectional neural network, the two hidden states are independent of each other. These relationships can be represented as follows:

$$y_t = g(\overrightarrow{h_t}, \overleftarrow{h_t}, V) \tag{3.26}$$

$$\overrightarrow{h_t} = f_p(x_t, \overrightarrow{h_{t-1}}, \overrightarrow{W}) \tag{3.27}$$

$$\overleftarrow{h} = f_n(x_t, \overleftarrow{h_{t+1}}, \overleftarrow{W}) \tag{3.28}$$

Where $\overrightarrow{h_t}$ is the hidden state of the recurrent network that goes from the past to the future, which will be called the positive-sense network; $\overleftarrow{h_t}$ is the hidden state of the network that goes from the future to the past, which will be called the negative-sense network. f_p represents the set of operations that the positive-sense network performs and f_n represents the operations of the negative-sense network.

Then the propagation step consists of three steps. The first consists of computing the hidden state of the last time step in the positive-sense network. The second consists of computing the hidden state of the first time step in the negative-sense network. The third consists of computing the outputs for each time step.

3.5.2 BiLSTM Networks

BiLSTM networks are similar to LSTM networks. What sets them apart is that they can use information contained in the direction that goes from the future to the past. This type of information can be particularly relevant when analyzing temporal data, since there may be long-term and short-term dependencies in both directions.

The architecture of a BiLSTM network consists of two LSTM architectures that are completely independent of each other. Figure 3.5 shows an example of a BiLSTM architecture implementation. If there is a need to analyze data in three time steps, t_1, t_2 and t_3, our BiLSTM network will consist of one normal cell and one inverted cell for each of the three steps, in the figure, indicated by LSTM-p, in the positive direction, and LSTM-n, in the negative direction, totaling six cells; and these cells will connect their hidden state, indicated in the figure by H to three outputs y_1, y_2, and y_3.

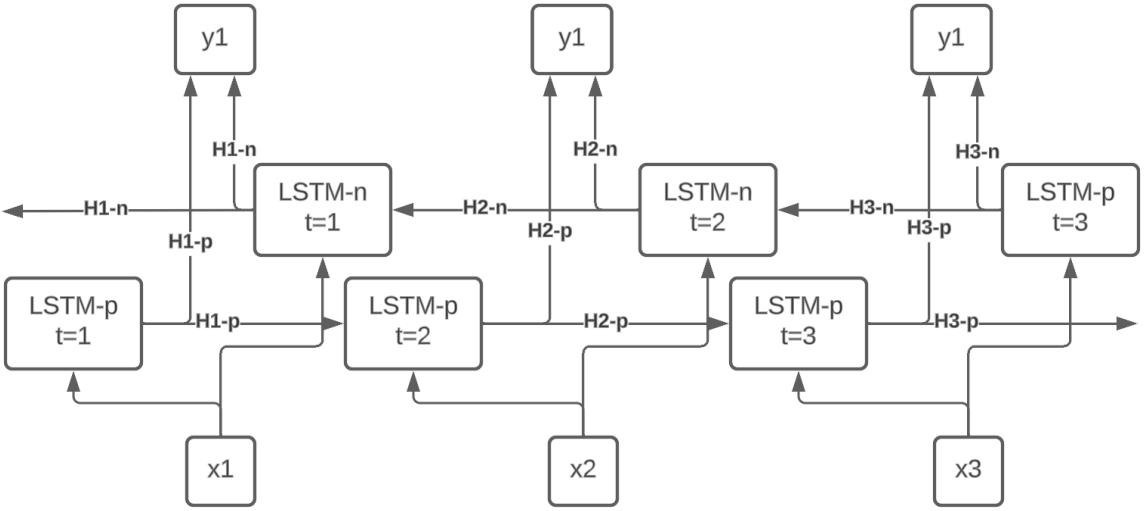


Figure 3.5: Architecture of a BiLSTM network, a schematic idea conceived from the article *Bidirectional recurrent neural networks* by Schuster et. al. [19], 1997.

Another advantage of using BiLSTM networks is that they are resistant to the gradient fading problem, so these networks can store long-term information without compromising their training.

3.6 Autoencoders

Autoencoders are a class of neural networks designed to learn efficient, compact representations of data, typically in an unsupervised manner. They accomplish this by reconstructing the input data from a lower-dimensional, compressed representation known as the bottleneck or latent space. The goal is not just to copy the input to the output but to learn meaningful patterns and features during the compression process.

3.6.1 Encoder

The encoder is the first component of an autoencoder, responsible for compressing high-dimensional input data into a lower-dimensional representation. The goal of the encoder is to extract the most relevant and meaningful features of the input while discarding redundant or irrelevant information.

Mathematically, the encoder can be described as a function:

$$z = f_{encoder}(x; \theta_e) \tag{3.29}$$

$x \in \mathbf{R}$ is the input data (e.g., an image, a time series, or any other type of high-dimensional data).$z \in \mathbf{R}^n$ is the compressed latent representation, with $m < n$, and $f_{encoder}$ is the transformation applied by the encoder, parameterized by θ_e (the weights and biases of the network).

3.6.2 Decoder

The decoder is the second component of an autoencoder, responsible for reconstructing the original input data from the compressed representation produced by the encoder. Its primary goal is to map the low-dimensional latent space representation back to the high-dimensional input space, ensuring the output closely matches the original input. The quality of the decoder's reconstruction determines how well the autoencoder has learned meaningful representations.

Mathematically, the decoder can be expressed as:

$$\hat{x} = f_{decoder}(z; \theta_d) \tag{3.30}$$

Where $\hat{x} \in \mathbf{R}^n$ is the reconstructed input, n is the dimensionality of the original input x, $f_{decoder}$ is the function implemented by the decoder, parameterized by θ_d.

During training, the encoder-decoder pair learns to map the data x to a compressed latent space z and back to a reconstruction $\hat{x}$. The latent space z captures the most salient features of the input data, discarding redundancies and noise.

The objective is to minimize the loss function:

$$L(x, \hat{x}) = \|x - \hat{x}\| \tag{3.31}$$

Where $\|x - \hat{x}\|$ measures the difference between the input and its reconstruction.

3.7 Reinforcement Learning

The field of reinforcement learning dates back to theories related to the learning process in animals through trial and error developed by the behavioral psychologist Burrhus Frederic Skinner, and to the problem of optimal control using the resolution of the Bellman equation and dynamic programming [41]. In the case of dynamic control, the agent needs to make sequential decisions to improve its performance, doing so through an optimal policy that maximizes the rewards accumulated along a trajectory. Skinner made his contributions to reinforcement learning with his method of operant conditioning, according

to which the individual is influenced by the environment and by the consequences that occur after their actions, whether reinforcing or punishing.

The main objective of reinforcement learning is to solve problems where decisions are made in sequence. Generally, when solving these problems there is always a goal, and the decisions made are evaluated based on how close this goal was reached. Sometimes, to get there it is necessary to take several actions sequentially and each of these actions will modify the environment in some way; It is necessary to observe small changes in the environment and note whether the actions taken contributed to moving closer to or further away from the goal. The information, especially errors, about the environment are analyzed and will be used to correct the actions taken. Over time, strategies are developed that maximize rewards, which allows the completion of increasingly complex tasks that optimize performance. Reinforcement learning has proven to be a very efficient machine learning paradigm, especially when there are not enough cases that can drive a supervised learning system.

3.7.1 Marcov Decision Process

In the standard reinforcement learning model, the agent is connected to the environment by a system of actions, states, and rewards. The agent receives from the environment a set of data that contains information about the environment, such as a set of perceptions; the environment, in turn, receives an action from the agent, which will modify the environment, which in turn will modify the set of perceptions and may generate some kind of reward or punishment for the agent.

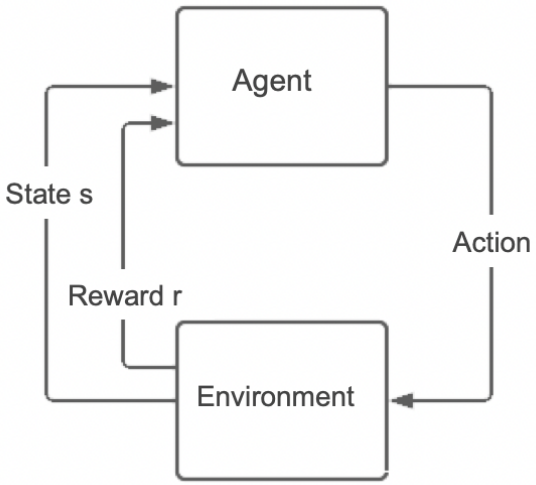


Figure 3.6: *Reinforcement learning control loop*, a schematic idea conceived from the article *Reinforcement Learning: A survey*, by Leslie Pack et. al.[20]; from 1996 .

Figure 3.6 shows the control scheme that governs reinforcement learning. The agent perceives the environment as state s_t, so it takes an action a_t. The environment is slightly modified, causing the agent to receive a reward r_t and now perceive the environment as a new state s_{t+1}. Formally:

$s_t \in S$ é um estado, onde S é o espaço de estados

$a_t \in A$é uma ação, onde A é o espaço de Ações

$r_t = R(s_t, a_t, s_{t+1})$é uma recompensa, onde R é a função de recompensas

The state space is the set of all possible states in the environment. The action space is the set of all possible actions that the agent can take. The function $R(s_t, a_t, s_{t+1})$ assigns a scalar value to each transition (s_t, a_t, s_{t+1}) that can be positive, negative or zero.

To formally model the transition from state s_t to state s_{t+1}, the concept of a transition function is necessary. Assuming that the future state depends solely on the present state and action, we incur the elementary property typical of Markovian processes, where past and future states are completely independent of each other. Assuming the Marcov property implies greater simplicity for the model. The transition function is then given by:

$$P(s_{t+1}|s_t, s_{t-1}, \cdots, s_0) = P(s_{t+1}|s_t) \tag{3.32}$$

Where P is the transition function that assigns a value, $0 \leq p \leq 1$, to the probability of the environment leaving state s_t and going to state s_{t+1}. Finally, the concept of objective must be defined, which the agent must maximize as it learns about the environment. To do this, the concept of trajectory must first be defined:

$$\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), \cdots, (s_T, a_T, r_T)\}$$

A trajectory is a set of experiences (s_t, a_t, r_t) in a time interval that goes from 0 to T. When the last state of the trajectory is terminal, we call this trajectory an episode.

Let $G(\tau)$ be the sum of the discounted rewards in a trajectory τ, where the discount rate is given by γ:

$$G(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t \tag{3.33}$$

G is known as the return on a given trajectory. The objective J is said to be the expectation of $G(\tau)$ over N number of trajectories.

$$J = E[G(\tau_i)]_{i=0}^{N}$$

The discount rate $\gamma \in [0, 1]$ is an important variable that determines how future rewards should be evaluated. The lower the value of γ, the less importance the agent will give to future rewards, and the higher the value, the more these rewards should be observed by the agent.

With the above definitions, it becomes possible to define the Markov decision process as a 5-tuple: (S, A, P, R, γ), defined by the state space S, the action space A, the transition function $P(\cdot)$, the reward function $R(\cdot)$ and the discount rate γ.

3.7.2 The Bellman Equation

A large part of reinforcement learning algorithms work based on value functions. These functions express how good or advantageous it is for the agent to be in a certain state, or how good or advantageous it is to perform a certain action while in a certain state. These value functions are defined for a policy. The policy consists of a function that defines the general strategy of what action to take from a given state. This is the function that tells the agent what it should do in any given state. The policy function is defined as follows:

$$\pi : S \to A$$
$$\pi(s) = a$$

For the stochastic case, where the action to take in a given state is uncertain, the policy should be defined as:

$$\pi : A \times S \to [0, 1]$$
$$\pi(a|s) = p$$

Where p expresses the probability of action a being executed given a state s. That said, the definition of the value functions for a given policy π follows as:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right] \tag{3.34}$$

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{3.35}$$

Where E_{π} expresses the expected value if the agent follows policy π. V is known as the value function and assigns a scalar value to evaluate how good a given state is, and Q is known as the quality value function (from *Quality value*) that evaluates how good a given action a taken in a given state s is.

An important property of the equations 3.34 and 3.35 is that they obey a recursive relationship. To show this relationship, we will proceed to develop the equation 3.34:

$$\begin{aligned} V^{\pi}(s_t) &= E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right] \\ &= E_{\pi}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t = s\right] \end{aligned}$$

Let $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$, then it is possible to write $V(s)$ as:

$$\begin{aligned} V^{\pi}(s_t) &= E_{\pi}\left[r_{t+1} + \gamma G_{t+1} \mid s_t = s\right] \\ &= E_{\pi}\left[r_{t+1} \mid s_t = s\right] + \gamma E_{\pi}\left[G_{t+1} \mid s_t = s\right] \end{aligned}$$

The law of total expectation states that $E[Y|X = x] = E[E[Y|X = x, Z = z]|X = x]$, so it is possible to write $V^{\pi}(s)$ as:

$$V^{\pi}(s_t) = E_{\pi}\left[r_{t+1} \mid s_t = s\right] + \gamma E_{\pi}[E_{\pi}\left[G_{t+1} \mid s_{t+1} = s'\right]|s_t = s]$$

Like $V^{\pi}(s_{t+1}) = E_{\pi}[G_{t+1}|s_{t+1} = s']$. Assures us that:

$$V^{\pi}(s_t) = E_{\pi}\left[r_{t+1} \mid s_t = s\right] + \gamma E_{\pi}[V^{\pi}(s_{t+1})]|s_t = s]$$

Using the linearity property of expectation, we arrive at the Bellman equation:

$$V^{\pi}(s_t) = E_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s]$$

This equation has many forms. One of the best known, for the function Q, is:

$$Q^{\pi}(s,a) = r(s,a) + \gamma \max_{a}(Q^{\pi}(s',a')) \tag{3.36}$$

In this equation, $Q(s,a)$ represents the Q-value of taking action a in state s and this value is determined by the reward of the current state $r(s,a)$ plus the highest Q-value of the next state s' discounted by the rate γ. This equation shows that the action in a state

is always determined by the next state and its proximity to future rewards. Thus, the Q-function helps to decide which action to take from a given state. Thus, the action that has the highest Q-value for that state should be taken.

Thus, it is also possible to define an optimal policy in terms of the Q-function, where the agent's performance is maximized:

$$\pi^*(s) = \arg\max_a(Q(s,a)) \tag{3.37}$$

3.7.3 *Deep Q-Networks*

Deep Q-Networks (DQN) is a reinforcement learning method based on the *Q-learning* algorithm. The difference between the two is that the first uses a neural network to estimate the Q function, which, as already seen, assigns a value to each action in a given state. The *Q Learning* method is generally useful when the number of states is small, and therefore the agent can explore all states; in this way, it is possible to build a table that maps all Q values for each action in each state. However, when the number of states is very large, the most appropriate methodology is to use a neural network to estimate the Q function, since building the table becomes infeasible.

Both the *Q-Learning* and *Deep Q-Networks* methods learn the greedy policy, in which the agent always selects the action that seems to be the best at the current moment, disregarding actions that better explore the environment. In other words, in this type of policy, the agent is always guided to follow the action that results in an immediate reward, neglecting that there may be actions that maximize the reward in the long term. This type of policy has the disadvantage of being able to lead to suboptimal decisions, tending to trap the agent in local optima. However, due to its simplicity and easy implementation, it has good computational efficiency, since it does not take into account a more in-depth analysis of the exploration of all possible states or actions.

Below is the algorithm for the *Q Learning* method:

First, several important definitions are performed for the algorithm to work. All the values of $Q(s,a)$ are initialized, the states and actions are defined, then the final state, the probability of taking an arbitrary action, and the discount rate are defined. Then the policy is defined as a function that receives a state s and returns an action a; which with probability ϵ will return an arbitrary action, and with probability $1-\epsilon$ will return the action whose value of the Q function is maximum.

A seguir é mostrado o algoritmo para *Deep Q-Networks*

Algorithm 1 Q Learning

Initialize the table Q(s,a) with arbitrary values
Set E as the number of episodes
Set $S = \{s_0, \cdots, s_N\}$
Set s_{ter} as the terminal state and $Q(s_{ter}, a) = 0$
Set $A = \{a_0, \cdots, a_k\}$
Set ϵ as the probability of arbitrary action
Set γ as the discount rate
function POLICY(s)
    $p \leftarrow$ arbitrary value between 0 and 1
    if $p < \epsilon$ **then**
        return $a \leftarrow$ arbitrary element of A
    else
        return $a \leftarrow \arg\max_a(Q(s, a))$
    end if
end function
for i $\leftarrow$ 1 **até** E **do**
    $s \leftarrow s_0$
    while $s \neq s_{ter}$ **do**
        $a \leftarrow$ POLICY(s)
        Execute a and observe reward r and successor state s'
        $Q(s, a) \leftarrow r + \gamma \max_a(Q(s', a'))$
        $s \leftarrow s'$
    end while
end for

Algorithm 2 Deep Q-Network

Initialize a regression neural network architecture where the output is $Q(s, a)$
Initialize the network parameters θ
Set E as the number of episodes
Set the states s as the network input
Set $A = \{a_0, \cdots, a_k\}$
Set ϵ as the arbitrary action probability
Set γ as the discount rate
function POLICY(s)
    $p \leftarrow$ arbitrary value between 0 and 1
    if $p < \epsilon$ **then**
        return $a \leftarrow$ arbitrary element of A
    else
        return $a \leftarrow \arg\max_a(Q(s, a))$
    end if
end function
for i $\leftarrow$ 1 **até** E **do**
    $s \leftarrow s_0$
    for t $\leftarrow$ 0 **até** T **do**
        $a_t \leftarrow$ POLICY(s_t)
        Execute a_t and observe the reward r_t and the successor state $s' = s_{t+1}$
        Calculate $Q(s', a)$,
        $y \leftarrow r + \gamma \max_a(Q(s', a'))$
        Calculate $\nabla_\theta L$ where $L = (y - Q(s, a))^2$
        Update network parameters $\theta \leftarrow \theta - \eta \nabla_\theta L$
        $s \leftarrow s'$
    end for
end for

Experience Replay

It is possible that due to the temporal correlations between several consecutive experiences in time, there may be instability and convergence problems in the training of a DQN-type network, and for this reason, the experience replay technique was created. In this technique, several experiences (tuples of the type (s, a, r, s')) are stored in a memory. Thus, instead of training the network with consecutive experiences, experiences are chosen randomly from the memory, so that they do not have a temporal correlation. In this algorithm, the agent first produces a series of experiences due to their relationships with the environment. A fixed number of these experiences are then collected. After that, a batch of experiences is randomly selected and the agent is trained for each of the experiences in this batch. This technique has some advantages, among them: greater stability in learning, since the temporal dependencies between the experiences have been reduced and the oscillations are attenuated; there is greater efficiency in the use of data since the stored experiences are reused several times.

Target networks

Target networks are a fundamental technique for stabilizing training and improving agent convergence. They help mitigate the problem of instability and oscillation that occurs when training a neural network to approximate the function $Q(s, a)$. Due to updates to the parameters θ, the function $Q(s, a)$ is constantly changing, so the network "overestimates" the values $Q(s, a)$, since the same network is used to calculate both the current value and the target value $\max_a(Q(s, a))$. To solve this problem, DQN introduces a target network $Q(s, a; \theta')$ that is a copy of the main network $Q(s, a; \theta)$. This network is used to calculate the target values during training and is kept fixed for a period of time before being updated. This reduces oscillations because the target value changes in a more stable and predictable way.

The Algorithm 3 shows the implementation of DQN with *replay experience* memory and target networks.

3.7.4 Rainbow DQN

The Rainbow DQN algorithm is an advanced reinforcement learning method that integrates several enhancements to the original Deep Q-Network (DQN) to improve its performance and stability. It combines six key techniques: **Double Q-learning**, to mitigate overestimation bias in Q-value predictions by decoupling action selection and Q-value; **Prioritized Experience Replay**, which samples transitions more effectively by focusing on those with higher learning potential by giving priority to experiences with

Algorithm 3 DQN with Experience Replay and Target Network

Initialize the policy neural network $Q(s, a, \theta)$
Initialize the target neural network $Q(s, a; \theta')$
Initialize the policy network parameters θ
Initialize the target network parameters θ'
Set the states s as the states of environment
Set $A = \{a_0, \cdots, a_k\}$
Set E as the number of episodes
Set ϵ as the arbitrary action probability
Set γ as the discount rate
Set μ as the number of steps required to update the target network
Set R as the Experience Replay
function POLICY(s)
    $p \leftarrow$ arbitrary value between 0 and 1
    if $p < \epsilon$ **then**
        return $a \leftarrow$ arbitrary element of A
    else
        return $a \leftarrow \arg\max_a(Q(s, a))$
    end if
end function
for i $\leftarrow$ 1 **até** E **do**
    $s \leftarrow s_0$
    for t $\leftarrow$ 0 **até** T **do**
        $a_t \leftarrow$ POLICY(s_t)
        Execute a_t and get the reward r_t and the successor state $s'_t = s_{t+1}$
        Store the experience (s_t, a_t, r_t, s'_t) in R
        Sample a random batch B of experiences from R
        Calculate $Q(s', a)$ for the batch B
        $y \leftarrow r + \gamma \max_a(Q(s', a'))$
        Calculate $\nabla_\theta L$ where $L = (y - Q(s, a))^2$
        Update policy network parameters $\theta \leftarrow \theta - \eta \nabla_\theta L$
        if $t \mod \mu = 0$ **then**
            Update target network parameters $\theta' \leftarrow \theta$
        end if
        $s_t \leftarrow s'_t$
    end for
end for

higher temporal-difference errors; **Dueling Network Architectures**, which separate the state values and action advantage functions to better generalize state values; **Multi-step Learning**, which incorporates multiple future rewards rather than just single-step rewards for faster learning and capture longer-term dependencies in the environment.; **Noisy Networks**, which replace e-greedy exploration with parameterized noise linear layers in the neural network for better exploration of the state space; and **Distributional RL**, which models the full distribution of returns instead of just the expected value, providing the agent with richer information about the uncertainty and variability of outcomes. By unifying these innovations, Rainbow DQN achieves state-of-the-art performance across various benchmark environments, demonstrating significant improvements in sample efficiency and learning stability [42].

Double DQN

designed to address the issue of overestimation bias in Q-value predictions. In the original DQN, the target for the Q-value update is computed as:

$$y = r + \gamma \max_{a'}(Q(s', a'; \theta')) \tag{3.38}$$

Where the same Q-network is used to select the action and to estimate its value. This can lead to overestimation because the action selection and value estimation rely on the same set of parameters, causing super estimation of Q-value updates.

Double DQN solves this by decoupling the action selection from the value estimation. Specifically, it uses the main network to select the action and the target network to evaluate it. The updated target becomes:

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s, a; \theta); \theta') \tag{3.39}$$

where θ represents the parameters of the main network, and θ' represents the parameters of the target network. This separation reduces de bias due to superestimation.

Prioritized Experience Replay

This improvement assigns a priority to each transition based on the temporal difference (TD) error, which measures the discrepancy between the predicted Q-value and the target Q-value:

$$\delta = |r + \gamma \max_{a'}(Q(s', a'; \theta')) - Q(s, a; \theta)| \tag{3.40}$$

Transitions with larger TD errors are considered more surprising or important because they indicate areas where the model is making larger mistakes. These transitions are sampled more frequently during training.

The probability of sampling a transition i is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{3.41}$$

Where $p_i = |\delta_i| + \epsilon$ is the priority of transition i, $\epsilon > 0$ grants that all transitions have a non-zero probability of being sampled, $\alpha \in [0, 1]$ controls the level of prioritization ($\alpha = 0$ is a uniform probability sample)

Prioritized sampling introduces a bias because it over-samples certain transitions. To counteract this, PER uses importance sampling (IS) weights to adjust the updates:

$$w_i = \left(\frac{1}{N \cdot P(i)}\right)^\beta \tag{3.42}$$

Where N is the size of the replay buffer, and $\beta \in [0, 1]$ gradually increases during training to fully correct the bias as the model converges.

Dueling Network Architectures

In a standard DQN, the network outputs Q-values directly for all possible actions. In contrast, a dueling network introduces two separate streams within the neural architecture, which are the value stream that estimates the value of being in a given state, represented as $V(s)$, which captures how good it is to be in the state independent of the actions taken. The advantage stream which estimates the advantage of each action in a given state, is denoted as $A(s, a)$, which reflects how much better or worse an action is compared to the average action in that state.

These streams are then combined to produce the Q-values for all actions using the following formula:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{N} \sum_{a'} A(s, a') \tag{3.43}$$

Where N is the number of actions. The subtraction of the mean advantage ensures that $V(s)$ represents the true state value, disentangled from the action-specific advantages.

Multi-step Learning

Multi-step learning is a reinforcement learning technique that extends the idea of single-step temporal-difference (TD) updates by incorporating rewards and transitions over mul-

tiple steps into the value updates. It provides a balance between the short-term focus of single-step updates and the long-term view of Monte Carlo methods, making it a powerful tool for improving the learning efficiency of agents. In multi-step learning, the target incorporates multiple rewards over n steps, providing a more comprehensive view of the agent's future trajectory. The n-step target is defined as:

$$y_{n-step} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^k \max_{a'}(Q(s', a'; \theta')) \tag{3.44}$$

Noisy Networks

In a standard neural network, the weights W and biases b are deterministic. Noisy Networks introduce stochasticity by adding noise to these parameters during training. Specifically, the weights and biases are modified as follows:

$$W = \mu_W + \sigma_W \odot \epsilon \tag{3.45}$$

$$B = \mu_B + \sigma_B \odot \epsilon \tag{3.46}$$

Where μ is the mean, σ is the standard deviation, and ϵ is a noise variable sampled from a standard distribution, often Gaussian. The final operation is the same as the dense layer:

$$y = W \cdot x + B \tag{3.47}$$

This formulation allows the network to adapt the amount and direction of exploration by learning the parameters μ and σ. During training, the noise ϵ is sampled at each forward pass, injecting randomness into the action-selection process.

Distributional Reinforcement Learning

In traditional RL, the action-value function $Q(s, a)$ represents the expected return:

$$Q(s, a) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{3.48}$$

Where G_t is the return. Distributional RL, however, models the full distribution of returns, denoted as $Z(s, a)$, where:

$$Z(s, a) \sim \text{Distribution of } G_t \mid s_t = s, a_t = a \tag{3.49}$$

Instead of predicting a single scalar value, the agent learns the distribution of possible outcomes, which includes both the expected value and information about the spread (variance), skewness, and other statistical properties of the returns.

3.8 Technical Indicators and Mathematical Formulas

Indicadores técnicos desempenham um papel crucial na análise de mercados financeiros, sendo amplamente utilizados por negociadores profissionais e investidores individuais. Essas ferramentas se baseiam em fórmulas matemáticas aplicadas aos dados de preços e volumes de um ativo financeiro, com o intuito de fornecer informações sobre a direção futura dos preços, identificar tendências, pontos de entrada e saída, e gerar sinais de compra ou venda.

3.8.1 The Candle

Technical analysis data consists of only two pieces of information: price and volume. Even so, since asset prices vary almost every moment, traditionally only the price is analyzed separated by defined time intervals, known as *timeframes*. In this way, the information begins to stratify. Now there is information about the price at which this time interval begins, known as the opening price, the price at which this time interval ends, known as the closing price, and its maximum and minimum in this same interval.

With these four pieces of information about the price in mind, the candlestick chart, commonly known as *candles*, is created. And it works as follows: each candle represents a time interval. If the closing price is higher than the opening price, then the body of the candle is white or green and it is known as a bullish candle, and if the closing price is lower than the opening price, the candlestick chart is red or black and it is known as a bearish candle. The upper shadow or line of the candle indicates the maximum price reached in that time interval and the lower shadow or line indicates the minimum price reached. The base of the bullish candle indicates the opening price and its top, the closing price. The base of the bearish candle indicates the closing price and its top indicates the opening price. Figure 3.7 shows examples of bullish and bearish candles.

Chart patterns and candlestick patterns are contained in the four basic pieces of information about candlesticks. Thus, any information available in these patterns can be correctly represented by price sequences of the candlesticks' closing, high, and low. Technical analysis signals come in many different forms. For example, there are chart patterns that consist of some figures that the price chart ends up forming, and from these figures, it is possible to get an idea of the continuation or reversal of the trend.

With the emergence of candlestick charts, several other types of information related to the shapes of the charts also emerged. In this way, the geometric shape of the chart itself becomes a potential source of information for understanding market fluctuations.

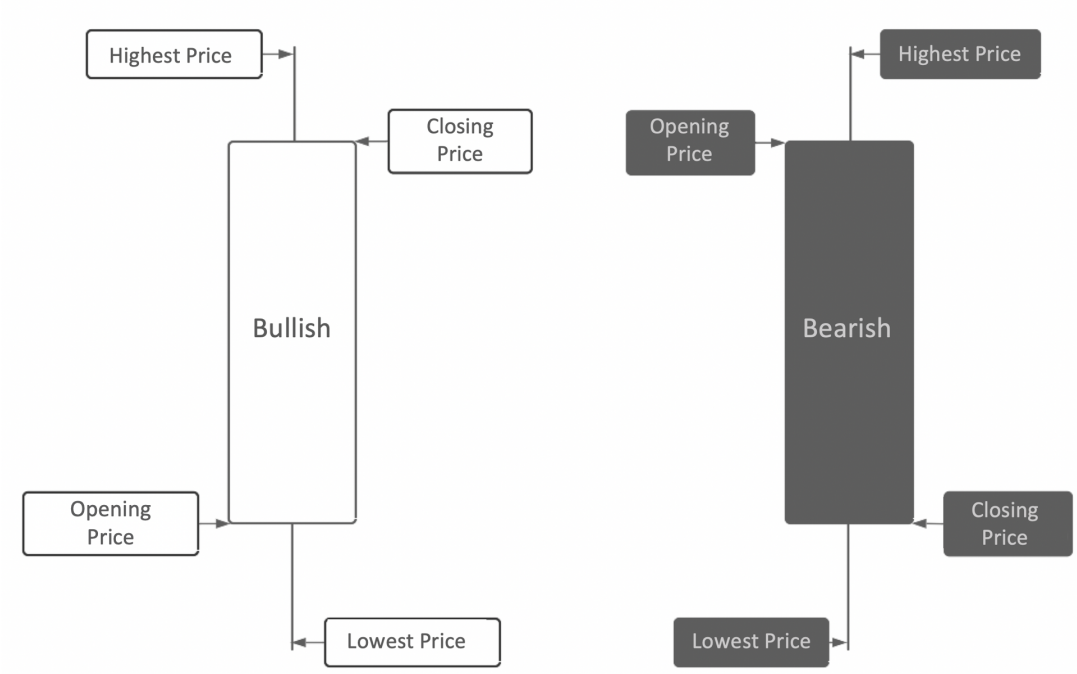


Figure 3.7: Bullish candle, on the left and bearish candle, on the right, invented by Munehisa Honma.

3.8.2 Weighted Moving Average (WMA)

The Weighted Moving Average (WMA) indicator is a technical indicator used in financial market analysis to smooth out price volatility and identify trends more accurately. It assigns different weights to recent prices, making them more influential in calculating the average.

Calculating the WMA indicator involves assigning weights to prices in a given time window and calculating the weighted average. The formula for calculating the WMA is as follows:

$$WMA = \frac{w_1 \cdot P_1 + w_2 \cdot P_2 + \ldots + w_n \cdot P_n}{w_1 + w_2 + \ldots + w_n} \tag{3.50}$$

Where WMA is the value of the indicator at the current time, P_i is the price in period i, and w_i is the weight assigned to the price in period i.

The weights assigned to the prices are usually calculated in decreasing order, assigning more weight to the most recent prices. There are several ways to assign weights, such as Arithmetic Progression, Geometric Progression, or using a specific formula. A commonly used formula for assigning weights is:

$$w_i = n - i + 1 \tag{3.51}$$

Where n is the total number of periods in the time window and i is the current period number. Alternatively, one can write:

$$\text{WMA}_t = \frac{1}{\sum_{i=1}^{n} i} \sum_{i=0}^{n-1} (n-i) P_{t-i} \tag{3.52}$$

The WMA indicator is commonly used for trend following and confirmation. It is often used in conjunction with other technical indicators to confirm or refute trading signals. When the price is above the WMA, it can be interpreted as an uptrend, while when the price is below the WMA, it can be interpreted as a downtrend.

3.8.3 Moving Average Convergence / Divergence (MACD)

The Moving Average Convergence Divergence (MACD) indicator is a technical indicator widely used in financial market analysis to identify possible entry and exit points in an asset. It consists of two lines, the MACD line, and the signal line, and is based on the difference between short-term and long-term exponential moving averages.

The calculation of the MACD indicator involves three main components: the exponential moving average (EMA), the MACD line, and the signal line. The formula for calculating the MACD is as follows:

$$MACD = EMA_{\text{short-term}} - EMA_{\text{long-term}} \tag{3.53}$$

Where $EMA_{\text{short-term}}$ is the short-term exponential moving average and $EMA_{\text{long-term}}$ is the long-term exponential moving average.

The signal line, often represented by a 9-period exponential moving average of the MACD, is calculated as follows:

$$\text{Signal Line} = EMA_{\text{9-period MACD}} \tag{3.54}$$

The MACD indicator is primarily used to identify the strength and direction of a trend in an asset. The MACD line crosses above the signal line when there is an uptrend, indicating a buy signal. Similarly, when the MACD line crosses below the signal line, it indicates a downtrend, suggesting a sell signal.

In addition, MACD crossovers of the zero line are also considered important. When the MACD crosses above zero, it indicates a trend change from bearish to bullish, while when the MACD crosses below zero, it indicates a trend change from bullish to bearish.

3.8.4 Percentage Price Oscillator (PPO)

The Percentage Price Oscillator (PPO) indicator is a technical indicator used in financial market analysis to identify possible trend reversals and generate buy or sell signals. It is a variation of the Moving Average Convergence Divergence (MACD) indicator and is calculated based on the percentage difference between two exponential moving averages (EMA).

The calculation of the PPO indicator involves two main steps: the PPO line, which consists of calculating the percentage difference between two EMAs, and the signal line. The formula for calculating the PPO line is as follows:

$$PPO = \left(\frac{EMA_{\text{short-term}} - EMA_{\text{long-term}}}{EMA_{\text{long-term}}} \right) \times 100 \tag{3.55}$$

Where $EMA_{\text{short-term}}$ is the short-term exponential moving average with 12 periods, and $EMA_{\text{long-term}}$ is the long-term exponential moving average with 26 periods. The signal line is calculated as an exponential moving average of the PPO line as follows:

$$\text{Signal Line} = EMA_{\text{9-period PPO}} \tag{3.56}$$

A very common signal generated by the Percentage Price Oscillator (PPO) indicator is the Signal Line Crossover. It is important to note that the signal line is calculated as a moving average of the PPO line, making it a secondary indicator. Therefore, the signal line always lags behind the PPO line. However, when the PPO line crosses above or below the signal line, it can indicate a potentially strong move.

The strength of this move is what determines the duration of the Signal Line Crossover. Analyzing and understanding the strength of the move, as well as recognizing false signals, is a skill that is acquired with experience.

Two types of Signal Line Crossovers should be looked for. The first is the Bullish Signal Line Crossover, which occurs when the PPO line crosses above the signal line. The second is the Bearish Signal Line Crossover, which occurs when the PPO line crosses below the signal line.

Another signal generated by the PPO is divergence. Simply put, divergence occurs when the PPO and the actual price are not aligned.

For example, Bullish Divergence occurs when the price makes a lower low, but the PPO makes a higher low. Price action can provide information about the current trend, but changes in momentum, as indicated by the PPO, can sometimes foreshadow a significant reversal.

Bearish Divergence is the opposite. It occurs when the price makes a higher high while the PPO makes a lower high.

The Percentage Price Oscillator (PPO) indicator is a very useful tool. Like the Moving Average Convergence Divergence (MACD), it uses two lagging indicators and incorporates the momentum aspect, making it more predictive. It is important to highlight the similarities between MACD and PPO, as they are virtually identical. The only difference is that PPO is presented as a percentage.

3.8.5 Rate of Change (ROC)

The Rate of Change (ROC) indicator is a technical indicator used to measure the percentage change in the price of an asset over a given period. It is often used in trend analysis and in identifying reversal points.

ROC is calculated as the percentage change in the price of an asset relative to its price in a previous period, as described by the formula below:

$$ROC_t = \left(\frac{P_t - P_{t-n}}{P_{t-n}} \right) \times 100 \tag{3.57}$$

The number of periods is determined by the user, however, a commonly used value is 9 periods.

ROC is closely linked to price. When prices are rising or advancing, ROC values remain above the Zero Line (positive) and when prices are falling or decreasing, they remain below the Zero Line (negative).

Additionally, while ROC is an oscillator, it is not limited to a set range. The reason for this is that there is no limit to how far a security can move, but there is of course a limit to how far it can fall. If the price drops to \$0, it obviously won't go any lower. Because of this, ROC can sometimes appear unbalanced.

The Rate of Change (ROC) indicator can be a good tool for identifying the overall long-term trend of a financial instrument. It may not lead to a signal on its own, but it can help confirm other signal-generating conditions.

As mentioned earlier, ROC is not range-bound like many other oscillators. Because of this, identifying overbought and oversold conditions can be a little less straightforward. Knowing where to place the overbought and oversold boundaries can be tricky. The best way to do this is by using historical research and analysis.

3.8.6 Momentum (MOM)

The Momentum (MOM) indicator is a technical indicator used to measure the speed of price change of a financial asset. It is often used to identify moments of strength or weakness in a trend.

The MOM is calculated as the difference between the current price (P_t) and the price in a previous period (P_{t-n}), where n is the number of periods considered. The steps to describe the calculation of the MOM are described below:

The MOM is calculated as the value of the difference between the current price and the previous price:

$$MOM_t = P_t - P_{t-n} \tag{3.58}$$

Where MOM_t is the value of the MOM at time t.

The MOM is an indicator that measures the speed of price change. Positive values indicate that the current price is higher than the previous price, indicating a bullish momentum and a buy signal. Negative values indicate that the current price is lower than the previous price, indicating bearish momentum and a sell signal.

3.8.7 True Range (TR)

The True Range (TR) indicator is a technical indicator used to measure the price volatility of a financial asset. It is often used in technical analysis to assess the risk of a position or to set stop-loss and take-profit levels.

TR is calculated as the difference between the largest absolute value of the following three options: the difference between the maximum price and the minimum price of a period ($H_t - L_t$), the difference between the maximum price and the previous closing price ($H_t - C_{t-1}$), and the difference between the minimum price and the previous closing price ($L_t - C_{t-1}$). The steps for calculating the TR indicator are described below:

First, the three possible differences are calculated:

$$\Delta_1 = H_t - L_t \tag{3.59}$$

$$\Delta_2 = H_t - C_{t-1} \tag{3.60}$$

$$\Delta_3 = L_t - C_{t-1} \tag{3.61}$$

Then, the TR is calculated as the largest absolute value among the three differences:

$$TR_t = \max(|\Delta_1|, |\Delta_2|, |\Delta_3|) \tag{3.62}$$

Where TR_t is the TR value at time t.

TR is an indicator that measures the volatility of an asset's prices. The higher the TR value, the higher the volatility. TR is used to define stop-loss and take-profit levels, where a higher TR level may justify a more distant stop-loss order to accommodate the asset's volatility.

3.8.8 Average Directional Index (ADX)

The ADX (Average Directional Index) indicator is a technical indicator used to measure the strength of a trend in a financial market. It provides an estimate of the strength of the trend, regardless of whether it is bullish or bearish.

ADX is calculated in three main steps: calculation of the Positive Directional (+DI) and Negative Directional (-DI) indicators, calculation of the Average Directional Index (DMI), and calculation of the ADX itself. The steps for the calculation are described below:

First, the DI indicators are calculated for each period. The +DI measures the strength of the bullish trend and the -DI measures the strength of the bearish trend. The formulas for the calculation are as follows:

$$\mathrm{DM}_{t+} = \begin{cases} H_t - H_{t-1}, & \text{se } (H_t - H_{t-1}) > (L_{t-1} - L_t) \\ 0, & \text{otherwise} \end{cases} \tag{3.63}$$

$$\mathrm{DM}_{t-} = \begin{cases} L_{t-1} - L_t, & \text{se } (L_{t-1} - L_t) > (H_t - H_{t-1}) \\ 0, & \text{otherwise} \end{cases} \tag{3.64}$$

$$+\mathrm{DI}_t = \frac{\mathrm{SMA}(\mathrm{DM}_{t+}, 14)}{\mathrm{ATR}_t} \times 100 \tag{3.65}$$

$$\text{-}\mathrm{DI}_t = \frac{\mathrm{SMA}(\mathrm{DM}_{t-}, 14)}{\mathrm{ATR}_t} \times 100 \tag{3.66}$$

Where H_t and L_t are the highest and lowest prices in period t, DM_{t+} and DM_{t-} are the positive and negative directional components of period t, $\mathrm{SMA}(\cdot, 14)$ is the simple moving average with period 14, and ATR_t is the average true range of period t.

Then, DX is calculated as the average of the absolute values of the differences between the DI indicators:

$$\mathrm{DX}_t = \frac{|+\mathrm{DI}_t - \text{-}\mathrm{DI}_t|}{|+\mathrm{DI}_t + \text{-}\mathrm{DI}_t|} \times 100 \tag{3.67}$$

where DX_t is the value of DX at time t.

Finally, the ADX is calculated as the exponential moving average of the DMI:

$$\text{ADX}_t = \text{SMA}(\text{DX}_t, 14) \tag{3.68}$$

Where ADX_t is the ADX value at time t and m is the period for the exponential moving average.

The ADX is an indicator that ranges from 0 to 100. Low ADX values, usually below 20, indicate a weak trend or no trend. High ADX values, usually above 40, indicate a strong trend. The ADX can also be used to identify the strength of the current trend and the possibility of a reversal.

The ADX indicator is a useful tool for measuring the strength of a trend in a financial market. The mathematics behind the ADX involves calculating the DI indicators, the DMI, and the ADX itself. With this information, investors and technical analysts can assess the strength of a trend and make more informed trading decisions.

3.8.9 Stochastic d% and k%

The Stochastic %D and K% indicator is a technical indicator used to identify overbought and oversold conditions in a financial market. It compares the current closing price with a range of previous prices to determine buying or selling pressure.

The Stochastic %D is calculated in three main steps: calculating %K, smoothing %K, and calculating %D. The steps for calculating the Stochastic %D indicator are outlined below:

First, %K is calculated, which represents the current position of the closing price relative to a range of previous prices. The formula for the calculation is as follows:

$$\%K = \frac{C - L_n}{H_n - L_n} \times 100 \tag{3.69}$$

Where C is the current closing price, H_n is the highest price in the last n periods, and L_n is the lowest price in the last n periods.

Then, %K is smoothed to obtain %D. Smoothing is usually done using a simple moving average (SMA) of the last m periods of %K. The formula for the calculation is as follows:

$$\%D = \text{SMA}(\%K, m) \tag{3.70}$$

Where $\%D$ is the value of Stochastic %D at the current time and m is the number of periods for the moving average.

The Stochastic %D ranges from 0 to 100. Values above 80 generally indicate an overbought condition, which may suggest that the asset is approaching a reversal point. Values below 20 generally indicate an oversold condition, which may suggest that the asset

is approaching a bullish reversal point. Traders often use %D crossovers with reference values, such as 80 and 20, to make buy or sell decisions.

3.8.10 Detrended Price Oscillator (DPO)

Detrended Price Oscillator (DPO) is a technical market indicator. It is designed to filter out long-term price trends and highlight short-term cycles in price data. DPO seeks to isolate cyclical price fluctuations, allowing investors and analysts to identify short-term patterns and potential trend reversal points.

The mathematics behind DPO is the difference between the current closing price and the simple moving average shifted in time by half the chosen period. The formula for DPO is:

$$DPO(n) = C - SMA\left(\frac{n+1}{2}\right) \tag{3.71}$$

Where C is the closing price, $SMA(k)$ is the moving average calculated over k periods, and n is the chosen period.

The core idea behind DPO is that the time-shifted moving average acts as a baseline, representing an estimate of the long-term trend. By subtracting the moving average value from the current closing price, DPO reveals the swings that occur above and below this baseline. This allows traders to identify short-term patterns that can be difficult to spot with traditional moving averages.

A distinguishing feature of DPO is that it does not take the long-term trend into account, focusing exclusively on short-term fluctuations. This makes it particularly useful for identifying cycles and reversals in markets with more complex trends.

3.8.11 Commodity Channel Index (CCI)

The Commodity Channel Index (CCI) is a technical indicator widely used in financial market analysis to identify overbought and oversold conditions, as well as possible trend reversals. It was developed by Donald Lambert and applies to a wide range of financial assets.

The calculation of the CCI indicator involves three main components: the average of typical prices, the simple moving average (SMA), and the mean absolute deviation (MAD). The formula for calculating the CCI is as follows:

$$\text{Typical Price}_t = \frac{H_t + L_t + C_t}{3} \tag{3.72}$$

$$CCI = \frac{\text{Typical Price} - \text{Typical Price SMA}}{0.015 \times \text{Typical Price MAD}} \tag{3.73}$$

Where Typical Price MAD is the mean absolute deviation of the Typical Price from its SMA (Simple Moving Average)

The CCI indicator fluctuates around a center line of 0. Values above +100 indicate that the price is above the average and can be interpreted as an overbought condition. Values below -100 indicate that the price is below the average and can be interpreted as an oversold condition. Traders often use CCI crossovers with the centerline and reference levels, such as +100 and -100, to make buy or sell decisions.

3.8.12 Normalized Average True Range (NATR)

The NATR (Normalized Average True Range) indicator is a widely used tool in technical analysis to measure market volatility. It provides information about the Average True Range of an asset relative to its current price.

Calculating the NATR indicator involves two main steps. First, it is necessary to calculate the average true range (ATR) of the asset. The formula for calculating the ATR is as follows:

$$\text{ATR} = \frac{1}{n}\sum_{i=1}^{n} \max(H_t - L_t, |H_t - C_{t-1}|, |L_t - C_{t-1}|) \tag{3.74}$$

Where n is the number of periods considered and H_t, L_t, and C_t are the high, low, and close prices for period t, respectively.

After calculating the ATR, the next step is to normalize it with respect to the current price. The formula for calculating NATR is as follows:

$$\text{NATR} = \frac{\text{ATR}}{C_t} \times 100 \tag{3.75}$$

Where Close is the current closing price.

The NATR indicator is expressed as a percentage and provides a measure of normalized volatility relative to the current price. The higher the NATR value, the higher the expected volatility.

Investors and analysts use NATR to identify periods of high volatility, which can indicate the possibility of large price movements. Conversely, a low NATR value suggests lower volatility and a more stable market.

3.8.13 On Balance Volume (OBV)

The On-Balance Volume (OBV) indicator is a widely used tool in technical analysis to assess the relationship between trading volume and price movements in a financial asset. It was developed by Joseph Granville in the 1960s and aims to identify divergences between trading volume and price movements, which can help traders predict trend reversals.

The OBV is based on the premise that trading volume can be an early indicator of price movements. The central idea is that when volume increases in tandem with a price movement, it suggests a confirmation of the trend. Conversely, when volume increases significantly without the price following suit, it can indicate a potential trend reversal.

The calculation of the OBV is simple and is based on adding or subtracting the day's trading volume from the previous day's OBV value, depending on whether the price rose or fell. So, if the current day's closing price is higher than the previous day's closing price, the current day's trading volume is added to the previous day's OBV value. If the current day's closing price is lower than the previous day's closing price, the current day's trading volume is subtracted from the previous day's OBV value. If the current day's closing price is equal to the previous day's closing price, the current day's trading volume does not affect the previous day's OBV value.

$$OBV(n) = \begin{cases} OBV(n-1) + V_n & \text{if } C_n > C_{n-1} \\ OBV(n-1) - V_n & \text{if } C_n < C_{n-1} \\ OBV(n-1) & \text{if } C_n = C_{n-1} \end{cases} \tag{3.76}$$

Where $OBV(n)$ is the current OBV, $OBV(n-1)$ is the previous period's OBV, V_n is the current trading volume, and C_n is the current closing price.

The OBV is then plotted on a chart, usually below the price chart, as a solid line. The OBV is watched for divergences: when the price is rising but the OBV is falling, this could indicate a possible bullish reversal, and vice versa.

Chapter 4

Research Project

This chapter details the main aspects and components of the research project. It discusses each part, its purpose, the choice of structure, and its function in the overall body of the project. The chapter is divided into five sections. The first section introduces a general outline of the project, and the following three sections present each of the modules that comprise it. The fifth section proposes metrics to evaluate the performance of each of the three modules.

4.1 The Total Model

The model consists of three modules. The first is the feature engineering module, responsible for inputs and information compression that will feed the other two modules with information from the chosen technical indicators. The second is the price and trend prediction module, which is responsible for making predictions about future prices and trends and the strength of such trends. The third is the trading module, which is responsible for buying and selling a given asset. Figure 4.1 shows the proposed system's structure, highlighting the information flow between the three modules.

4.2 Feature Engineering Module

This is the module responsible for feeding information to the two subsequent modules. The main challenges that this module has to solve are the following: processing the raw market data according to the form of technical indicators, presenting the model with a set of information that is truly informative and capable of efficiently training the neural networks of the other two modules; and once this set of characteristics is given, verifying the possibilities of dimensionality reduction, to make the training more efficient, removing

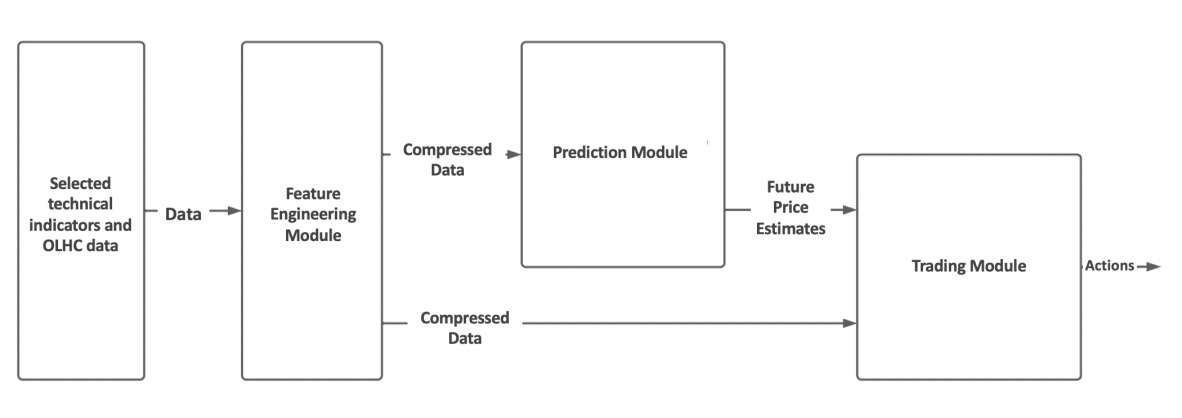


Figure 4.1: Structure and information flow of the three modules that make up the proposed system: the feature engineering module, the price and trend prediction module, and the trading module.

as much noise as possible since this is the main factor that worsens the overall performance of the system.

4.2.1 Feature Selecion

The set of information related to the training of neural networks that predict prices or market trends can have several different natures. One could opt for a set of information related to fundamental analysis, for example. In this case, the assets to be analyzed would almost necessarily be company shares, and the nature of the data would be related to the information disclosed by these companies, such as balance sheets, profits, debt, and growth prospects. Another perspective would be to use sentiment analysis, extracting data from social networks and news to feed the neural networks. This perspective would be quite useful since investors are subject to the disposition effect discussed in Chapter 1. Yet another perspective would be data related to volume flow, an approach known as *Tape Reading*; this would provide information about the largest open and closed operations in the market. These operations allow us to know the nature of the largest investors who are operating.

However, the set of information used will be traditional technical analysis data. It was chosen to use it because it is easy to obtain and simple to process. This data uses only information already contained in the asset's price history and the volume of purchases and sales made. However, this information is slightly different from the data from *Tape reading*, since in that case the information is more individualized and analyzes investor

orders on a case-by-case basis, while the volume used in technical analysis is always analyzed in a general and non-personalized way.

The first four technical analysis signals to be used in the feature engineering module are those related to candles: the opening price, the minimum price, the maximum price, and the closing price.

However, the technical analysis signals known to be the most informative are the technical indicators, and it is also on these signals that this work is based on building the model inputs. There are several types of technical indicators. There are **trend following indicators**, which are those that provide confirmation information of a trend, usually information that usually comes late, providing good signals of when to enter and exit the market. However, these indicators are often not effective in identifying sideways markets and thus provide many false signals. There are **momentum indicators**, which are those that help measure the strength of the trend, in this way they show the "speed" at which prices are moving in a direction. However, they can also provide false signals in the face of a sideways trend. There are **volume indicators**, which are also confirmation indicators, based on one of the principles of Dow theory, according to which, volume follows the trend, that is, if the trend is up, then there should be an increase in the volume of purchases and if the trend is down, there should be an increase in the volume of sales. In this way, these indicators also serve to indicate the strength of trends, and the lack of confirmation can warn of a reversal. There are **volatility indicators**, which are designed to measure the strength of price fluctuations. These can provide information that is particularly useful for identifying sideways trends. There are also **cycle indicators**, which, instead of identifying trend information, identify short-term cycles, which makes them useful for verifying repetitive patterns in price movements.

In this work, representatives of these classes of indicators were chosen. Many were chosen based on the attribute selection work presented in Chapter 2. However, the indicators were chosen to represent the maximum amount of useful information for neural networks. It is important to emphasize that all the indicators were chosen theoretically, with the work related to variable selection presented in Chapter 2 assisting the process.

Of the **trend followers**, three representatives were chosen: Weighted Moving Averages (WMA), Average Directional Movement (ADX), and True Range (TR). The WMA indicator was one of the most selected in the work of Peng et. al.[35], and in the work of Haq et. al. [36], it consists of weighted moving averages where the most recent prices have a greater weight than the past prices. The ADX indicator was selected in the work of Peng et. al., and the work of Haq et. al., It is very good at providing information about the strength of the trend, and can help to identify whether the market is in a sideways trend; it has three signal lines, the ADX itself, the +DI and the -DI. The TR indicator

was selected in the work of Haq et. al. It measures the amplitude of price variations over the analyzed period and is especially useful for identifying daily or intraday movements of the asset. Table 4.1 shows a summary of the chosen trend indicators.

Table 4.1: Selected Trend Following Technical Indicators

Name	**selected from:**	**Importance**
WMA	[35], [36]	* Reports trend continuation * Gives importance to recent prices
ADX +DI, -DI	[35], [36]	* Trend strength * Identifies sideways movement
TR	[36]	* Price range * Asset volatility

Of the **momentum indicators**, six representatives were chosen: Average True Range (ATR), Rate of Change (ROC), Moving Average Convergence Divergence (MACD), Stochastic (Stoch), Commodity Channel Index (CCI), and Relative Strength Index (RSI). The ATR indicator or its normalized form Normalized Average True Range (NATR) was one of the most selected in the work of Peng et. al. and the work of Haq et. al. It is an indicator that measures the volatility of the asset about the last closing price, so it is very sensitive to changes in current volatility, that is, it consists of a smoothing of the TR indicator. The ROC indicator was selected in the work of Peng et. al., in the work of Haq et. al., and the work of Ji et. al. [37]. It measures the rate of change is useful in identifying the strength and direction of the current trend, and is indicated for capturing overbought and oversold signals. The MACD indicator is one of the most popular technical indicators, and it was selected in the work of Peng et. al., in the work of Haq et. al., and the work of De Oliveira et. al., and consists of subtracting a twelve-period moving average from a twenty-six-period moving average. It is useful for signaling market entries and exits and identifying divergences. The Stoch indicator was selected in the work of Peng et. al. and De Oliveira et. al. in its two versions, %K and %D, while in the works of Haq et. al. and Ji et. al., it appears in modified versions. This indicator provides good overbought and oversold signals, but its importance lies in its efficiency in sideways markets. The William's Percent Range (%R) indicator consists of an unsmoothed version of the stochastic, and with an inverted sign, it was selected in the work of Peng et. al. and the work of De Oliveira et. al. The CCI indicator was selected in the work of Peng et. al. and the work of Ji et. al. It measures the variation between the current price of an asset and its average price over a given period. It is useful for identifying overbought and oversold signals and identifying divergences. Finally, the RSI indicator measures the

magnitude of recent price changes to identify overbought and oversold conditions; this indicator was selected in the work of Gang Ji et. al. and de Oliveira et. al. Table 4.2 shows a summary of the chosen oscillator indicators.

Table 4.2: Selected oscillator indicators

Name	**selected from:**	**Importance**
ATR	[35], [36]	* volatility sensitivity
MOM	[35], [38]	* overbought and oversold * trend reversal
ROC	[35], [37]	* Trend strength and direction * Overbought and oversold
RSI	[37], [38]	* Trend strength and direction * Overbought and oversold
MACD	[35], [36],[38]	* Market entry and exit * Divergence identification
Stoch	[35], [38] modified in [36], [37]	* Sideways market efficiency
Will R	[35], [38]	* Sideways market efficiency
CCI	[35], [37]	* Overbought and oversold * divergence identification

Among the **volume indicators**, **volatility indicators**, and **cycle indicators**, three representatives were chosen. The On Balance Volume (OBV), the Detrended Price Oscillator (DPO), and the Variance (VAR). The OBV was selected in the work of De Oliveira et. al. It is designed to measure the flow of positive and negative volume on a given asset. Therefore, measuring the health of a trend. A price increase followed by an increase in OBV means a healthy uptrend, while a price increase followed by a decrease in OBV would be an uptrend with a strong possibility of reversal. The DPO was the best-selected indicator in the work of Peng et. al. It consists of an indicator designed to neutralize any information about the trend and highlight information about short-term cycles that occur in prices. It was chosen, among other things, because it is the best-selected indicator in Peng's work, in addition to the nature of its underlying information, which differs from that of the other indicators. Table 4.3 shows a summary of the indicators chosen

among those of volume, volatility, and cycle. VAR consists of a moving variance, where the variance is calculated over a sliding window of candles, and is an indicator that gives an idea of market volatility. It was selected in the work of Peng et. al. E and Gang Ji et. al.

Table 4.3: Indicators chosen from volume, volatility and cycle

Name	**selected from:**	**Importance**
OBV	[38]	* Variation in volume flow
VAR	[35], [37]	* Market volatility
DPO	[35]	* Small short-term cycles

The set of all inputs presented to the model is shown in the table 4.4. It can be seen that there are a total of twenty-one inputs, although only seventeen indicators were selected. This is because some indicators have more than one signal, as is the case with the ADX indicator, which has three signals, or the stochastic indicator, which has two.

4.2.2 Data pre-processing

Financial data has some characteristics that make it difficult for artificial neural networks to process. To make it more suitable for this processing, it is necessary to pre-process it, attenuate certain characteristics, and highlight others.

The transformation to be performed on the data is standardization. This transformation is particularly important in artificial neural networks since it greatly improves training since data with very different scales can lead to much slower convergence. In addition, due to the activation functions, which are often sigmoid or hyperbolic tangents, normalizing the data places it on the same scale as the neural network. If the data is not normalized within the scale of the activation functions, it becomes more difficult for the neural network to produce good results. To normalize the data, the transformation Z is performed:

$$Z_t = \frac{X_t - \mu}{\sigma} \tag{4.1}$$

Where Z is the transformed data, μ is the mean of the data, and σ is the variance of the data.

The last treatment consists of removing the *outliers* or atypical values. To do this, all points Z_t where $Z_t < -3\sigma$ or $Z_t > 3\sigma$ are removed.

4.2.3 Dimensionality Reduction

Once the set of inputs has been selected and it is guaranteed by technical analysis that this set contains useful information, it is necessary to reduce its dimensionality to preserve the information and, at the same time, reduce the computational costs of training the neural networks related to this set.

A technique that introduces aspects of non-linear modeling required by the market is the *autoencoder*. The *autoencoder* consists of a neural network trained from its inputs, which is why it is said that the autoencoder is a self-supervised technique. The tapering of the autoencoder is responsible for reducing the dimensionality of a given input. Thus, from an input with several features, the encoder produces an output with the same information and with few features, but it is as if this information were compressed into a code. The decoder, on the other hand, is responsible for translating this code into familiar values, decompressing the code, and having an output with the same dimension as the encoder input. Of course, the loss of some degree of information is inevitable, as is typical of data compression.

The BiLSTM cell was chosen to build the *autoencoder* architecture. The choice of this architecture is justified by the existence of an abundance of noise and non-linearity in the data, which can be correctly handled by the BiLSTM cell. In addition, this cell has the property of memory, which correlates previous data with subsequent data.

The idea of using the Autoencoder is to reduce the number of input features to facilitate the training of the neural networks responsible for the Prediction Module and the Trading Module. Figure 4.2 shows the structure of the encoder and decoder, each using a total of three BiLSTM cells; the encoder significantly reduces the number of inputs and the decoder reestablishes the number of inputs.

4.3 Prediction Module

The structure of the prediction instruments is something very important to consider since these are the instruments that will process the chosen input data and provide useful estimates about future prices that will be used to make purchase and sale decisions. The engineering of this structure is the main factor responsible for the type of processing to be used. It is at this stage that it should be chosen whether the processing will take into account internal correlations, spatial characteristics, temporal dependencies, or other types of information about the given data set.

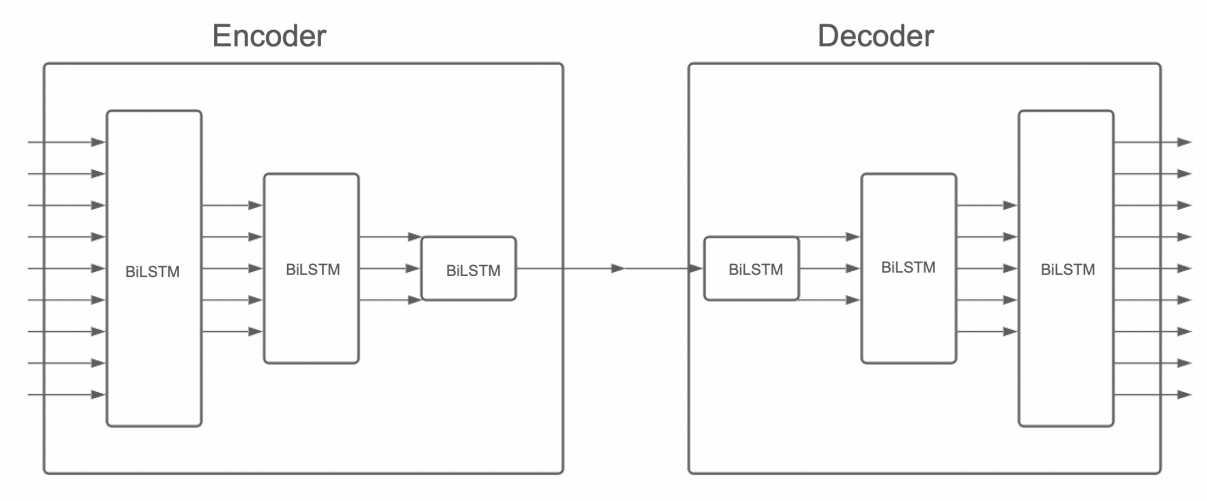


Figure 4.2: Structure of the encoder, on the left, and the decoder, on the right, both using BiLSTM cells .

The work of Selvin et. al. [3] compares some neural network architectures for price prediction. The performance of the RNN network, the LSTM network and the CNN network in predicting the behavior of the shares of the companies Infosys, TCS, and Cipla are compared. The results showed that there were some chunks that the RNN and LSTM networks were unable to predict. In addition, the results showed that the CNN networks were the ones that obtained the best results. The author attributes the superiority of CNN's performance to the fact that it does not use prior information for prediction and that CNN is also capable of capturing patterns and dynamic changes that occur in the price.

However, the work of Zhanhong He et. al. [4] compares the performance of CNN networks with hybrid networks composed of an LSTM layer followed by a CNN layer, and the results showed that the latter has a better performance for price prediction. The performance of both types of networks was evaluated in predicting the behavior of the gold price.

It is based on these results that the architectures of Market Trend Prediction and Trend slope Regressor are proposed.

4.3.1 Market Trend Prediction

Market trend prediction is done by trying to determine whether the market is bullish or bearish. In this problem, since what needs to be determined is not a continuous numerical value but rather a binary value that oscillates between these two possibilities,

it is a classification problem, where there will be just two classes. The function of the trend predictor is to determine which of these two classes the market is placed in.

The proposed structure for the trend predictor is composed of three layers and a final dense layer. The first layer will be composed of convolutional and *max pooling* networks. The first layer was chosen to be convolutional to be able to extract, from the beginning, the most important characteristics of the price. By selecting the maximum values, the *max pooling* layer will extract the most salient features of the data, which will be most representative for determining the trend.

The second and third layers of the trend predictor will consist of a BiLSTM neural network to capture temporal dependencies between time steps already selected by the previous convolutional layers. It is important to remember that due to the convolution operations performed by these layers, the number of time steps sent to the third layer becomes significantly reduced; it follows that only the most informative time steps will be taken into account for estimating the trend.

The last layer of the trend predictor instrument consists of a densely connected layer with a dropout factor. The use of dropout is justified to mitigate the possibility of overfitting since this layer will randomly "turn off" some neurons, forcing the neural network not to rely on a select group of neurons for good prediction.

The methodology for training the trend predictor neural network is supervised learning. Where the future values of the price series themselves will serve as labels to train a set of past data. The cost function to be used is the binary cross-correlation since it is the usual cost function used in two-class classification problems. Figure Figura 4.3 shows the structure of the Market Trend Classifier.

Labeling

To label the market and train the neural network, a specific process involving linear regression is used. A set of candles is selected from the market data, and from this set of candles, the closing prices are extracted and used to compute the linear regression of the entire set. Linear regression is applied to determine the overall trend of the market within the selected window of candles. The slope of the linear regression line serves as the key indicator of the market's trend direction. If the slope of the linear regression line is positive, this indicates an upward trend, and the market is classified as bullish. Conversely, if the slope of the linear regression line is negative, this signifies a downward trend, and the market is classified as bearish.

Once the market has been labeled, this data is used to train the neural network. The labeled data provides the neural network with clear examples of market behavior, enabling it to learn the patterns and relationships necessary for accurate market predictions in

future scenarios. Through this methodology, the neural network is trained to recognize the trends.

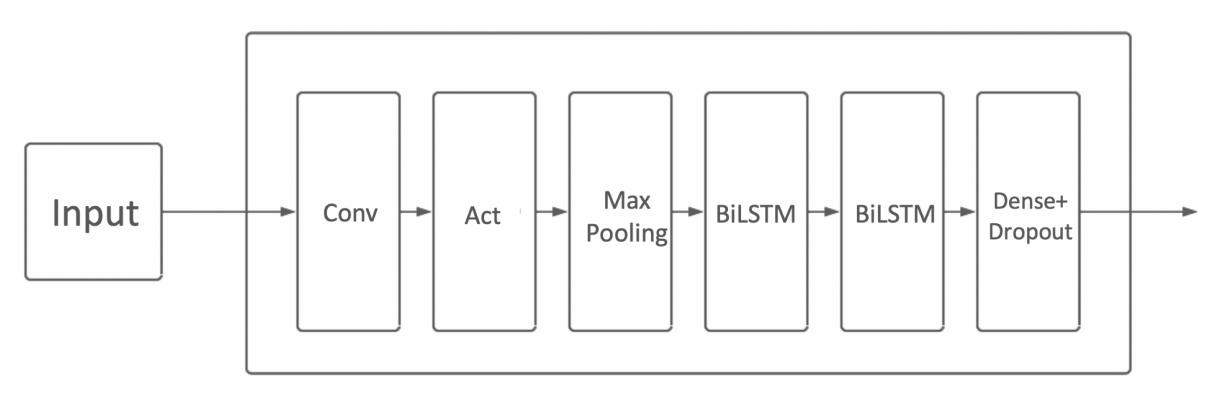


Figure 4.3: Structure of the Market Trend Classifier, starting from the input layer, convolutional layer, activation layer, Max Pooling layer, two BiLSTM layers, and dense layer with dropout factor.

4.3.2 Trend Slope Regressor

Price prediction is done by trying to estimate the slope of the linear regression since its direction is already determined by the trend predictor. In this problem, what needs to be determined is precisely a numerical value, which characterizes a regression problem.

The proposed structure for the price predictor consists of three layers of neural networks plus a dense layer. The first and second layers will be composed of a BiLSTM neural network. This network's purpose is to map long-term dependencies between prices. The choice of the BiLSTM network is justified by the fact that this type of network captures both dependencies between past and current prices and between future and current prices. In a typical LSTM network, the network's current values are influenced only by the past, while in BiLSTM networks, the current values are influenced by both the past and the future. Here, the structure of the price predictor differs from that of the trend predictor, since in the latter the first layers are convolutional to capture more elementary information, while in the trend slope regressor, the first layers are intended to capture complex temporal dependencies.

The last layer will be composed of a simple convolutional network and *max Pooling*. The choice of this type of layer to be the last lies in the fact that placing the convolutional layer or the *max pooling* layer in the first place would result in a loss of information that could be valuable for determining the slope. Unlike what happens in the trend predictor, it is not interesting to refine the information too much right at the beginning; on the

contrary, this information needs to be worked on and processed by the BiLSTM layers at the beginning, to be refined only at the end by the convolutional and *max pooling* layers.

The training methodology for the trend slope regressor is the same as that used for the market trend classifier and it uses supervised learning. The cost function used will be the mean squared error, as is classic in regression problems. Figure Figura 4.4 shows the structure of the trend slope regressor. Highlighting its information flow that passes through the two BiLSTM, Convolutional, Activation, Max Pooling, and dense layer with dropout factor.

The same way the occurs in the market trend classifier, a set of candles is selected, half of this set is made of past candles, and half of then is made of future candles, then the linear regression slope is calculated, and the value of this slope is calculated. This predictor is useful to determine the strength of the trend.

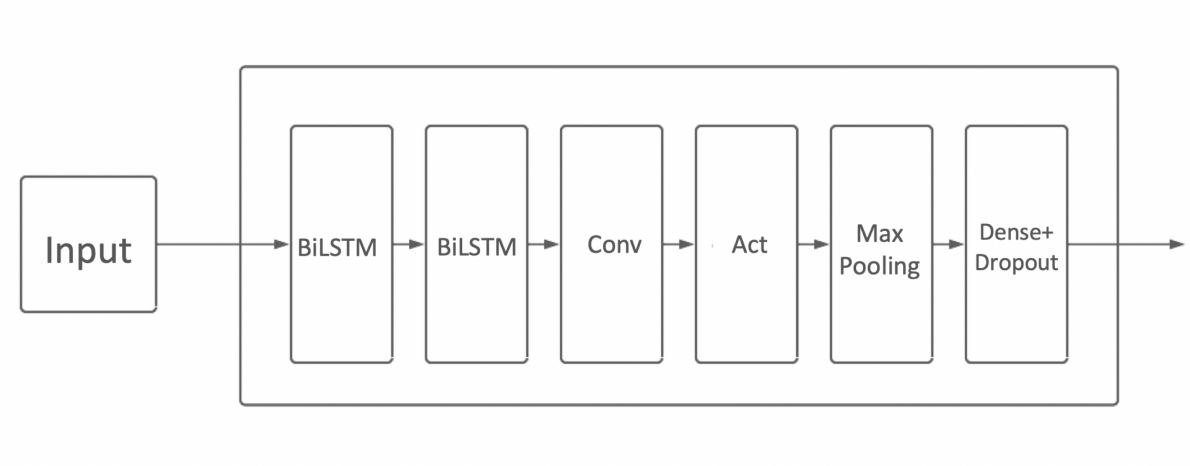


Figure 4.4: Trend Slope Regressor structure, starting from the input layer, two BiLSTM layer, convolutional layer, activation layer, Max Pooling layer, and dense layer with dropout factor.

4.4 Trading Module

The objective of this module is to use information from technical indicators and predictors to make the best decisions to buy and sell the asset. There can be many decision-making methods. The method chosen for this project is reinforcement learning using neural networks, more specifically **Rainbow DQN**

4.4.1 Trading by Reinforcement Learning

The trading module consists of a neural network capable of making buying and selling decisions in a given financial market. This module must be able to analyze the best times to buy and sell and perform operations accordingly.

To model the trading instrument in an agent guided by reinforcement learning, it is necessary to systematically and efficiently plan its states, actions, and rewards, as these aspects constitute the whole responsible for the agent's performance.

The agent's states are modeled according to the agent's perception of the market. To form this perception, the agent must have access to current prices, trends estimated by trend predictors, as well as information from a set of technical indicators, compressed as much as possible, to improve the agent's performance in its training. Since what matters to the agent is information regarding the best times to buy and sell, information about the estimate of future prices should be the most considered. To this end, the predictor module must produce good trend estimates, whether upward or downward.

The agent's actions are, in principle, very simple: buy, sell, or hold the asset. When there is a moment of overbuying in the market, the agent, noticing this overbought, starts to sell the asset, and when there is a moment of oversold, the agent starts to buy. If the agent's actions are limited to buying, selling, and holding, this limits the agent's probabilistic perceptions, so that if he perceives that one operation is very likely to be profitable and another is only slightly profitable, the agent invests in both equally. A more interesting approach consists of investing more money in situations with a higher probability of profit and investing less money in situations with a lower probability of profit. Therefore, the engineering of the agent's actions must consider a space of actions with more elements. The probability of profit of the operation should be considered in three levels: probable profit, more or less probable profit, and unlikely profit. Thus, the buying action is subdivided into three, and the selling action into three. This guarantees a seven-element action space, which will be modeled as follows:

$$A = \{-3, -2, -1, 0, 1, 2, 3\} \tag{4.2}$$

Where negative numbers represent sales and positive numbers represent purchases, and the variation from 1 to 3 represents how much the agent should invest in each operation; whether it should invest a lot, more or less, or a little, the number is proportional to the probability of success of the operation.

Finally, the engineering of the agent's rewards must be established, and it is only necessary that this reward be proportional to the profit that the agent can extract from a market operation.

4.4.2 Rainbow DQN with Memory Buffer

The algorithm established in the reinforcement learning will be Rainbow DQN. The structure of the neural network responsible for this trading instrument will initially consist of three densely connected layers, each with approximately 64 neurons. Later, BiLSTM layers will be used instead of the densely connected layers, since it is considered that the agent's performance should improve as memory is added to the Rainbow DQN algorithm. This BiLSTM layers are the Memory Buffer. A densely connected neural network was initially chosen for its simplicity and low computational resources for its training. After experiments will be performed with the memory buffer.

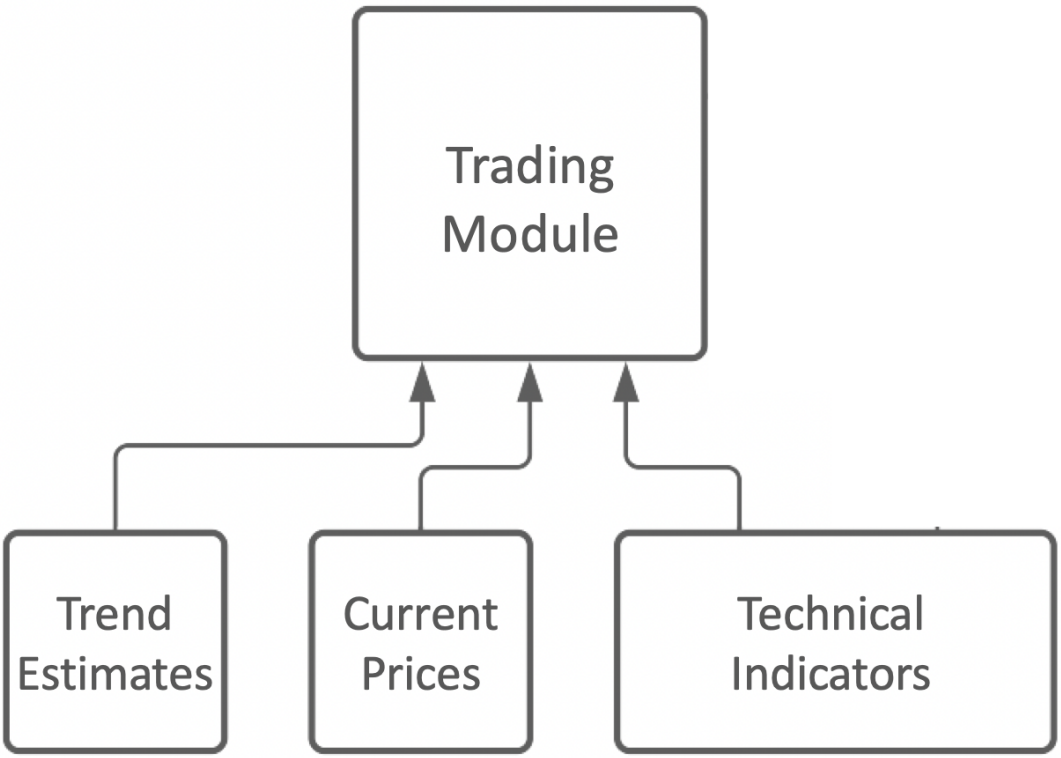


Figure 4.5: Structure of the trading instrument and risk management instrument and the flow of information received, representing the states.

4.5 Evaluation

4.5.1 Evaluation metrics

Evaluation metrics are needed to measure the performance of machine learning models so that it is possible to understand how well the model is making predictions concerning real data. Metrics provide an objective way to quantify the quality of predictions and help to make informed decisions about model selection and tuning.

Since the Market Trend Predictor engages in classification problems, its evaluation metric must be consistent. Thus, its performance is evaluated according to the following metrics: precision, recall, and accuracy.

Precision

$$P = \frac{p_v}{p_v + p_f} \tag{4.3}$$

Recall

$$R = \frac{p_v}{p_v + n_f} \tag{4.4}$$

Accuracy

$$A = \frac{p_v + n_v}{p_v + p_f + n_v + n_f} \tag{4.5}$$

Where p_v are true positives, p_f are false positives, n_v are true negatives, and n_f are false negatives. Precision measures the proportion of correctly predicted positive instances concerning the total number of instances predicted as positive. Recall measures the proportion of correctly predicted positive instances in relation to the total number of true positive instances. Accuracy measures the proportion of correct predictions about the total number of predictions; it is a general metric that provides an initial idea of the quality of the predictions.

The Trend Slope Regressor, on the other hand, is inserted in regression problems, so its evaluation metric must be different. Its performance should be evaluated according to three metrics:

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.6}$$

Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \times 100 \tag{4.7}$$

Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{4.8}$$

The Root Mean Squared Error (RMSE) calculates the standard deviation of the errors. In the formula above, y_i represents the real values and $\hat{y}_i$ represents the predicted values, n is the total number of samples. The MAE calculates the absolute value of the errors. Then Mean Absolute Percentage Error (MAPE) calculates the average error as a percentage of the real values. It is useful when it is necessary to evaluate the relative error about the size of the real values. In the formula above.

Finally, the performance of trading is given by the engineering of their reward functions. Since the reward function for both instruments is the percentage profit obtained, their performance will also be evaluated as a function of the profit.

4.5.2 Evaluation comparisons

The idea is that to validate the proposed model, its results, through evaluation metrics, are compared with other simpler models. Trend predictors should be evaluated based on classification problem metrics, that is, accuracy, precision, and recall, where the proposed predictors should have these metrics compared with simpler predictors, for example, those composed only of an LSTM layer and a CNN layer. Just as price predictors should be evaluated based on regression problem metrics: RMSE and MAPE), they should be evaluated according to their comparison with less complex price predictors, such as those built only from LSTM and CNN.

In addition, trading instruments should also be evaluated. The metric to evaluate them will be simple profit. That is: the performance of these instruments will be measured based on the profit they can make.

4.5.3 Evaluation Scenarios

Evaluating a system in different scenarios is essential to gain a full understanding of its performance, adaptability, and limitations. Different scenarios represent different situations that the system might face in reality, allowing for a more robust and realistic evaluation. Evaluating a system in a variety of scenarios helps ensure that it can generalize well to different situations; this is especially important in the case of predictive systems. Evaluation in different scenarios tests the robustness of the system in the face of variation in the data. A system that only works under ideal conditions may not be very useful. Testing in different scenarios helps ensure that the system is resilient to change.

Evaluating in different scenarios allows the system to be continuously adapted to ensure that it remains relevant and effective.

The scenarios proposed for this work consist of different types of assets at specific time intervals. The assets chosen are cryptocurrencies: Bitcoin, Ethereum, Ripple, and Cardano. All of these assets were chosen based on their popularity, with Bitcoin being the most popular cryptocurrency.

Cryptocurrency prediction is generally considered one of the most difficult prediction problems in terms of time series due to the high number of unpredictable factors involved and high volatility. Cryptocurrencies are time series close to random walks, from which it can be deduced that the prediction problem for this type of series is usually too difficult [28].

One point to be made is regarding the time interval for the three assets in which the model will be tested. As for Bitcoin and cryptos in general, the period in which China, until then one of the world's leading cryptocurrency markets, banned mining and declared cryptocurrency transactions illegal in the country should be included. These periods are being highlighted to test the model's performance in the face of major geopolitical events, thus providing a better idea of the model's robustness.

Table 4.4: All indicators presented at the model input

	Acronym	**Name**
1	WMA	Weighted Moving Averages (WMA)
2	TR	True Range (TR)
3	ADX	Average Directional Movement (ADX)
4	DI+	Directional Index Positive
5	DI-	Directional Index Negative
6	ATR	Average True Range (ATR)
7	MACD	Moving Average Convergence Divergence (MACD)
8	MACD - signal	Moving Average Convergence Divergence (MACD) - signal line
9	RSI	Relative Strength Index (RSI)
10	ROC	Rate of Change (ROC)
11	Stoch % K	Stochastic (Stoch) %K
12	Stoch % D	Stochastic (Stoch) %D
13	Will % R	William's Percent Range
14	CCI	Commodity Channel Index (CCI)
15	OBV	On Balance Volume (OBV)
16	DPO	Detrended Price Oscillator (DPO)
17	VAR	Variance (VAR)
18	Close	Close price
19	High	High price
20	Low	Low price
21	Volume	Volume

Chapter 5

Experiments and Results

This chapter will present the experiments related to the three modules that make up the model, revealing the main studies concerning the aspects and details that make up the model. The first section will investigate the best architectures for efficient compression of financial data using autoencoders and some of the main parameters that make up these architectures. In the second section, the best ways to extract useful estimates about the financial market using price and trend predictors will be investigated. The performance of the proposed architectures will also be investigated, as well as some of their main parameters. The third section will investigate the general performance of the trading module, about purchases and sales.

5.1 Feature Engineering Module

The first stage of the construction of the feature engineering module consisted of selecting the inputs that would feed the model. The selection of these inputs was done theoretically, with the variable selection work being used as an aid for this choice. All selection was made according to the requirement that each indicator provide useful information to the model. Some technical indicators, because they have waveforms that are completely identical to those of indicators already selected, were overlooked during the selection process, such as the Momentum (MOM) indicator, which has the same waveform as the Rate of Change (ROC) indicator, or the Percentage Price Oscillator (PPO) indicator, which also has the same waveform as the Moving Average Convergence Divergence (MACD) indicator.

The premise that the chosen technical indicators could provide the model with conflicting information was taken into consideration, as well as the amount of noise that would be added. This is one of the problems to be solved, or at least mitigated, by the feature engineering module.

5.1.1 Preprocessing

An experiment was conducted on the need for data differentiation as a pre-processing step. Differentiation is characterized by trying to transform a non-stationary time series into a stationary series. This transformation ensures that the statistical aspects of the series, such as the mean and variance, are more constant throughout the series. Figure 5.1 shows the data without this pre-processing and the data to which differentiation was applied. We can see, through this figure, the loss of original information inherent to differentiation; the original waveform was completely compromised. For this reason, it was decided not to apply differentiation to the data.

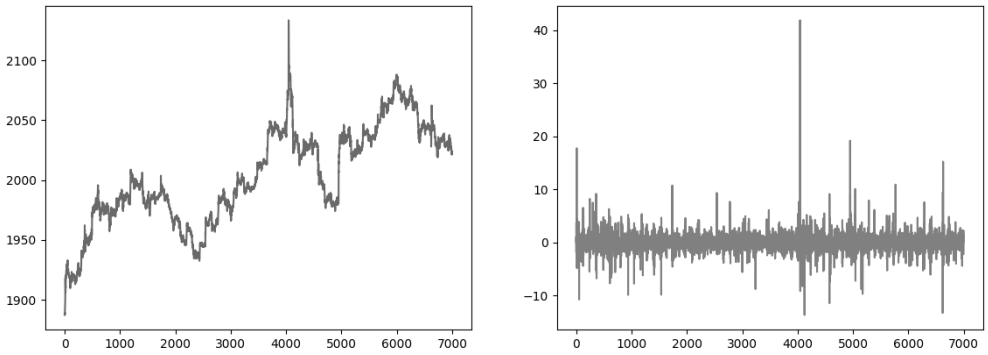


Figure 5.1: Closing price of gold given in dollars per ounce with 7000 samples, on the left. The closing price of gold applied to differentiation, on the right, also with 7000 samples.

However, for the correct introduction of data into artificial neural networks, the data must have a zero mean and a unitary standard deviation. This transformation is known as standardization. Standardization is important because it places the data on the same scale as the neural network activation functions. Figure Figura 5.2 shows the non-normalized and normalized data side by side. It can be seen that this transformation does not result in a severe loss or alteration of information. Therefore, normalization was applied to the data.

The next step consists of performing a dimensionality reduction itself to compress the information that will feed the rest of the model, filtering out some of the noise. For this purpose, the autoencoder was chosen.

5.1.2 Choosing the Autoencoder architecture

To choose the best architecture for the autoencoder, several models built from LSTM and BiLSTM cells were proposed. The models were proposed to analyze the performance

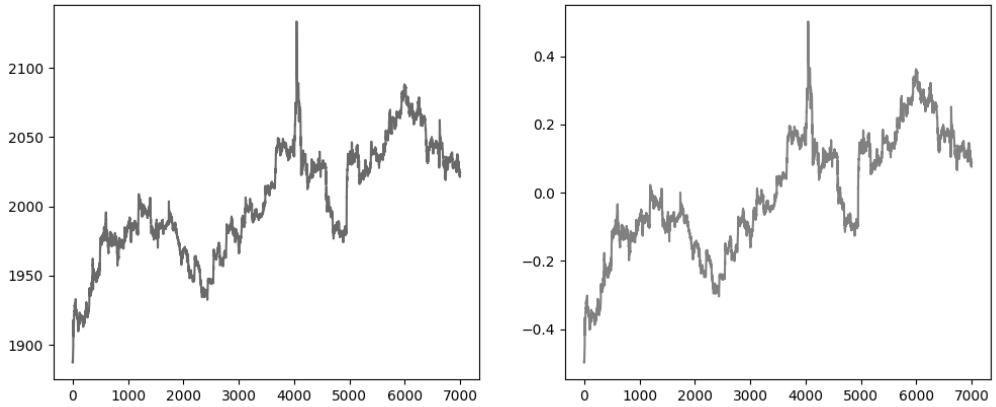


Figure 5.2: Closing price of gold given in dollars per ounce with 7000 samples, on the left. The closing price of gold minus the mean, and divided by the standard deviation, on the right, also with 7000 samples.

of the autoencoder according to the size of the information compression, given by the dimension of the context vector; according to the time window that feeds the LSTM and BiLSTM cells; and according to the number of layers that make up both the encoder and the decoder.

Performance according to the dimension of the context vector

To evaluate the performance of the autoencoder according to the compression of the information, or more specifically, according to the dimension of its context vector, a three-layer model was proposed, both for the encoder and for the decoder. In this model, each layer is composed of either LSTM cells or BiLSTM cells. The number of epochs chosen to train the models was 700, the minibatch size was 100 samples, and the learning rate was 10^{-3}. The time window size in all models was 60 time steps. The loss function chosen to train the model was the mean squared error. The dimensionality of the model input corresponds to twenty-one attributes. Furthermore, the optimization method chosen was ADAM.

Figure 5.4 shows the performance of the autoencoders for a context vector with dimensionality equal to 1, 5, 10, and 15. It is noteworthy that in both models, the performance worsens as more information compression is required, with the case with the highest loss being the one whose context vector has dimensionality equal to 1, and the one with the lowest loss being the one with a context vector with dimensionality equal to 15. However, we can see that the loss does not increase in the same proportion as the dimension of the context vector increases. The loss decreases slightly between the case where the context

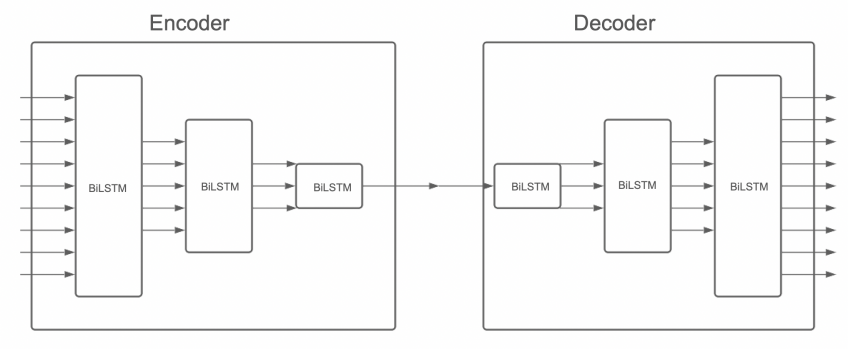


Figure 5.3: Proposed scheme for the autoencoder with three layers for dimensionality reduction in both the encoder and decoder.

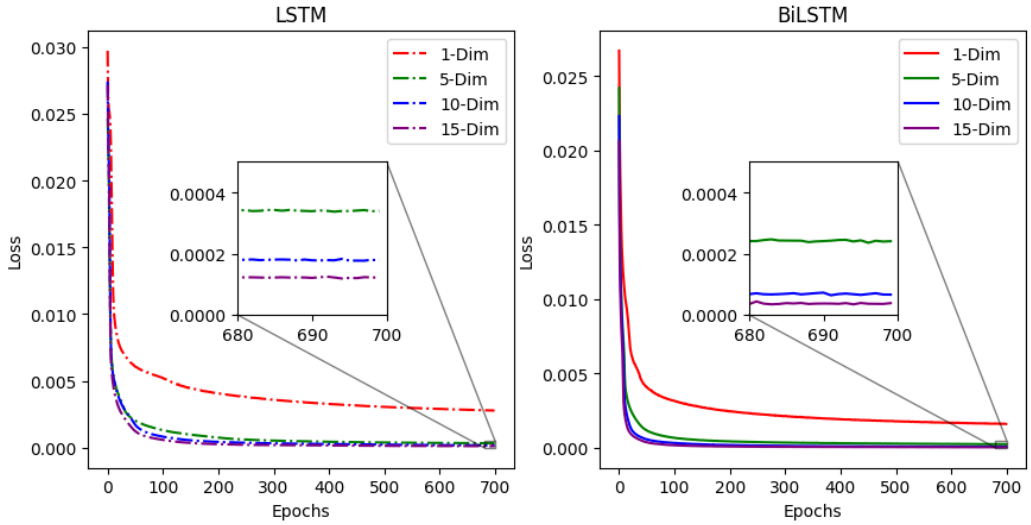


Figure 5.4: Comparison of the evolution of the losses of the LSTM and BiLSTM models calculated over the training epochs. The figure on the right shows the evolution of the losses for four possible dimensions of the context vector for the autoencoder-BiLSTM. The figure on the left shows the evolution of the losses for the Autoencoder-LSTM.

vector has dimensionality equal to 10 and the case where it has dimensionality equal to 15, but the same difference cannot be noted between the losses of the cases with dimensionality equal to 1 and 5. Furthermore, the difference in performance is even smaller in the model that uses BiLSTM cells. Figure Figura 5.5 further highlights the performance superiority of BiLSTM autoencoder over LSTM, and it is also possible to note that the BiLSTM model converges faster.

The total sample set was divided into training data and test data in a proportion of 80 and 20 percent respectively. To evaluate the performance of the autoencoder against the training data, three metrics were considered: Mean Squared Error (MSE), Mean Absolute

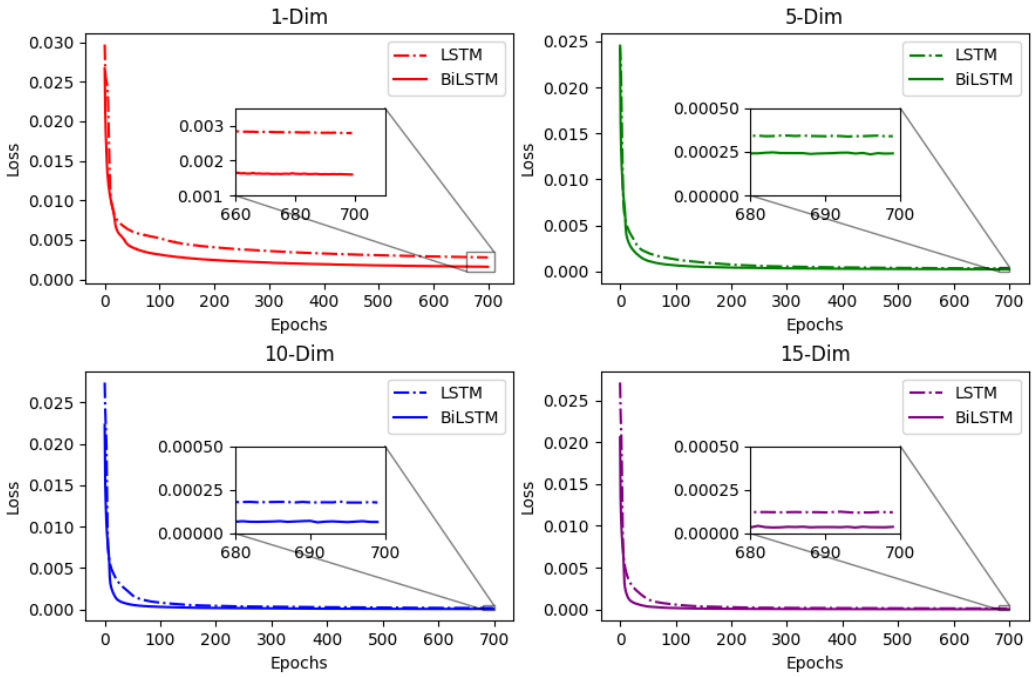


Figure 5.5: Comparison of the evolution of the model loss with context vector of dimensionality equal to 1, 5, 10, and 15, computed over the training epochs for LSTM and BiLSTM Autoencoders. Top left image: context vector of dimensionality equal to 1. Top right image: context vector of dimensionality equal to 5. Bottom left image: context vector of dimensionality equal to 10. Bottom right image: context vector of dimensionality equal to 15..

Error (MAE), and Mean Absolute Percentage Error (MAPE). Table 5.1 shows the result of each of these metrics for the four dimensionality values of the context vector. It is possible to observe the decrease in errors as the dimensionality is increased.

Performance according to the number of layers

To evaluate the performance of the autoencoder according to the number of layers, four architectures were proposed: the first with an encoder and a decoder with two layers; the second with four layers each; the third with six layers and the fourth with eight layers. These layers were tested for both the LSTM cell and the BiLSTM cell. The number of epochs used was 700, the learning rate was 10^{-3}, and the time window size was 60 time steps. The loss function chosen for the error calculation was the mean squared error. The dimensionality chosen for the context vector was 3 and the optimization method was ADAM.

Table 5.1: Performance of *LSTM and BiLSTM autoencoders* according to different dimensions for the context vector and according to different metrics

Context Vector Dimension	LSTM			BiLSTM		
	MSE	MAE	MAPE	MSE	MAE	MAPE
1	0.0278	0.1254	186.1556	0.0214	0.1017	223.9342
5	0.0089	0.0659	153.5034	0.0052	0.0539	135.4673
10	0.0055	0.0518	130.4784	0.0024	0.0348	96.4087
15	0.0039	0.0428	108.2602	0.0019	0.0306	77.4144

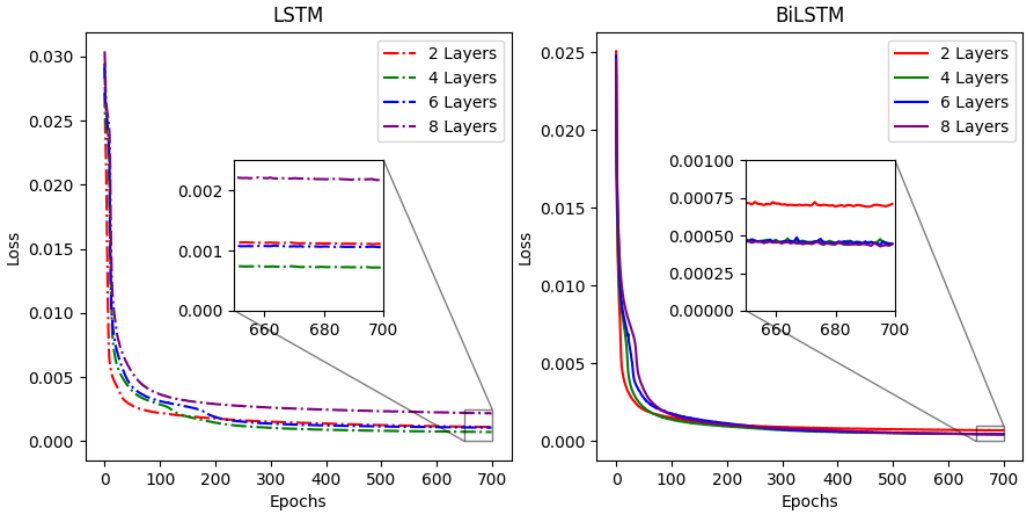


Figure 5.6: Comparison of the loss evolution as the number of layers in the encoder and decoder increases; for 2, 4, 6, and 8 layers. The figure on the right shows the model where the layers used are BiLSTM cells. The figure on the left shows the model where the layers used are LSTM cells..

Figure 5.6 shows that for the LSTM case, the case with the lowest error was four layers, and the worst case was two layers; the cases with six and eight layers were intermediate. The case where BiLSTM cells were used to compose the layers showed that the cases with four, six, and eight layers all had very similar errors, and the worst case was where only two layers were used. Figure 5.7 shows the superior performance of the BiLSTM autoencoder over the LSTM since in all cases we can see that the error of the former was lower.

Table 5.2 shows the performance of the LSTM and BiLSTM autoencoders for testing data. The data were divided between training and testing in a ratio of 20 and 80 respectively. It can be seen that the autoencoder with four layers has the lowest MSE and MAE values when using the LSTM cell. When using the BiLSTM cell, we notice that the lowest MSE and MAE values occur when using eight layers. The lowest MAPE values always

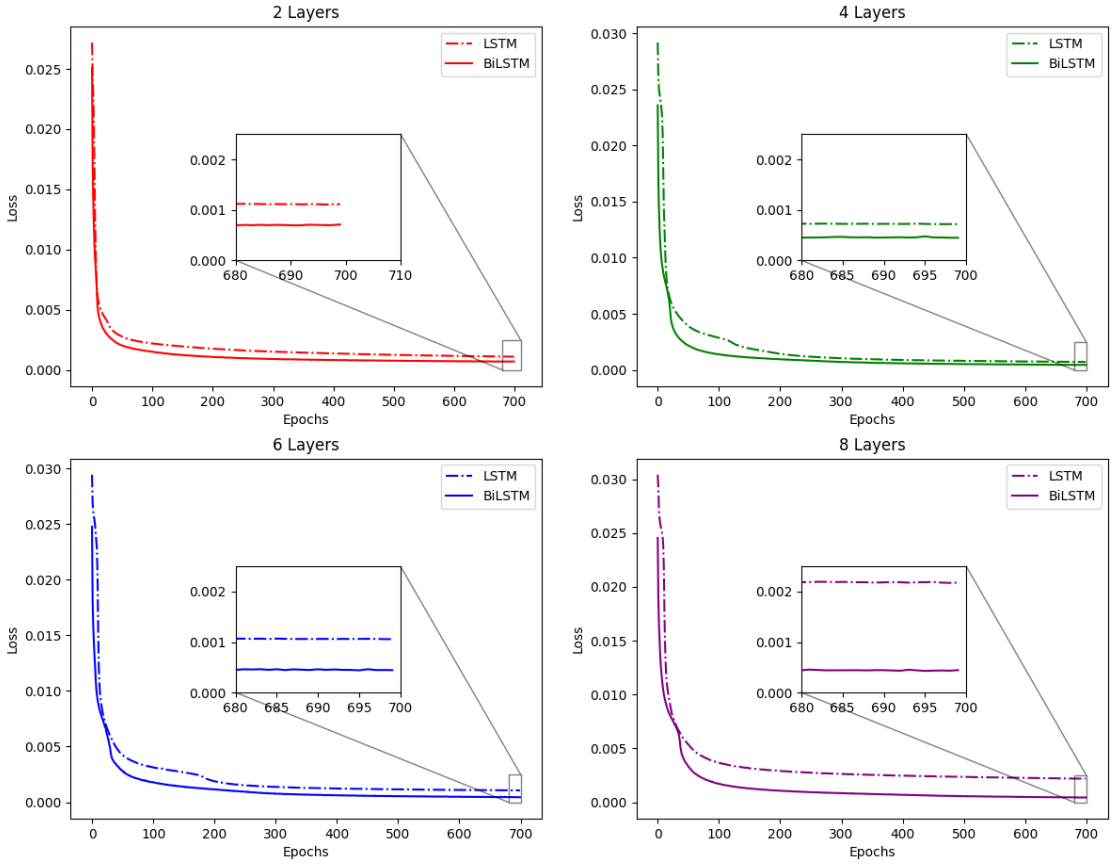


Figure 5.7: Comparison of loss evolution between LSTM and BiLSTM autoencoders. The top left figure shows the models using 2 layers. The top right figure shows 4 layers. The bottom left figure shows 6 layers, and the bottom right figure shows 8 layers..

occur when using eight layers, whether LSTM or BiLSTM. Furthermore, we notice that the architecture that uses BiLSTM has a lower error than LSTM in almost all cases.

Performance according to different time windows

Since the autoencoder is built using layers of recurrent neural networks, it is necessary to evaluate the performance of the autoencoders according to the size of the time window used. Thus, the performance of the autoencoder was evaluated in four time window sizes: 6, 60, 350, and 1050. The model used three layers and the dimension of the context vector was equal to 10. The number of epochs used was equal to 700; the learning rate was 10ˆ−3 and the size of the minibatch was 100 samples. In addition, the mean square error was chosen to calculate the error and the optimization method was ADAM.

Figures 5.8 and 5.9 show that there is little difference between the evolution of losses for the various time windows considered. Still, it is clear that *autoencoders* built according

Table 5.2: Performance of *LSTM and BiLSTM autoencoders* according to different numbers of layers used in the encoder and decoder

Number of Layers	LSTM			BiLSTM		
	MSE	**MAE**	**MAPE**	**MSE**	**MAE**	**MAPE**
2	0.0187	0.0986	195.809	0.0086	0.0706	165.9187
4	0.0184	0.0924	173.4737	0.0079	0.068	167.0501
6	0.0188	0.0924	183.7128	0.0083	0.0656	172.7861
8	0.0317	0.1303	135.9925	0.0077	0.0642	150.6123

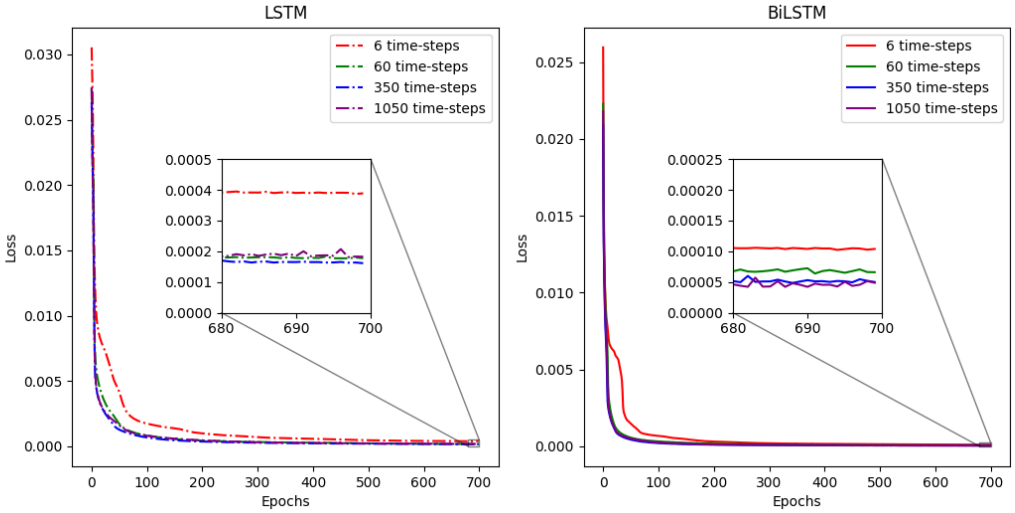


Figure 5.8: Comparison of the loss evolution for different time windows, for time windows of 6, 60, 350, and 1050 time steps. The left figure shows the model where the layers are LSTM cells, and the right figure shows the model where the layers are BiLSTM cells..

to BiLSTM cells always guarantee smaller errors than those built according to LSTM cells. However, it should be taken into account that this difference in performance is only valid when considering only the training data.

Table 5.3 shows that, for the test data, the smaller the time window considered, the smaller the errors. When the time window considered was 6 time-steps, the smallest error values were obtained, regardless of the metric considered. These errors say more about the nature of the data used than about the architecture of the autoencoder; they say that the temporal dependence of the data is short.

Discussion

First, the performance of the autoencoders about the compression size was tested; this evaluation was carried out based on the dimensionality of the context vector. Regarding

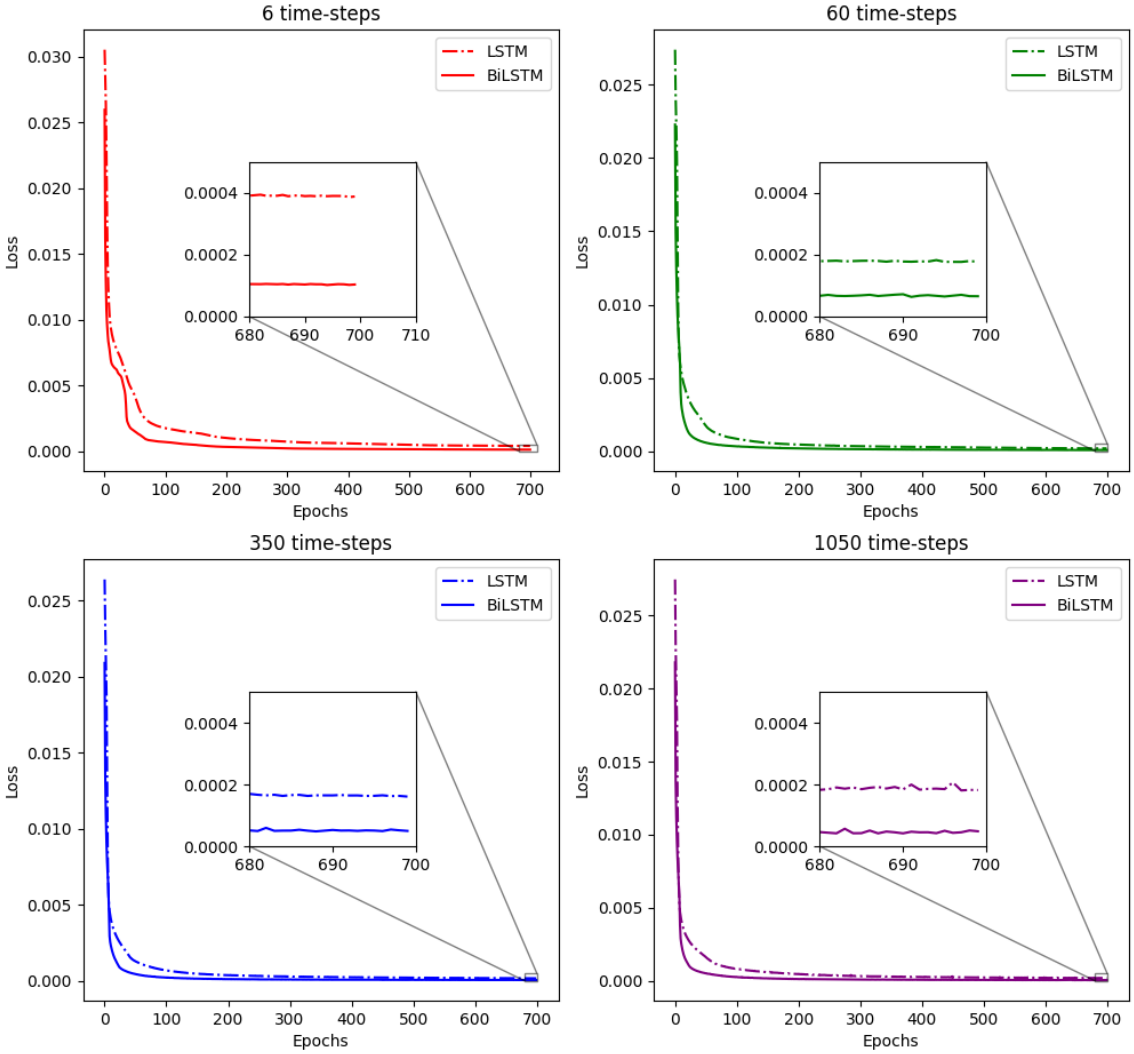


Figure 5.9: Loss evolution comparison between LSTM and BiLSTM autoencoders. The top left figure shows the models using 6 time steps. The top right figure shows 60 time steps. The bottom left figure shows 350 time steps, and the bottom right figure shows 1050 time steps..

the training data, it was observed that the performance of the autoencoder worsens as more compression is required; however, the relationship between error and compression is not linear. Therefore, it is necessary to choose an optimal amount of compression so as not to compromise the quality of the data too much. It was observed that there was little difference in error between the compression with a context vector of 10 and the compression with a context vector of 15.

Then, the performance of the autoencoders about the number of layers present in their architecture was tested. The results for the test and training data were different.

Table 5.3: Performance of LSTM and BiLSTM autoencoders according to different numbers of layers used in encoder and decoder

Time	LSTM			BiLSTM		
Steps	MSE	MAE	MAPE	MSE	MAE	MAPE
6	0.0040	0.0440	107.6606	0.0016	0.0286	96.4087
60	0.0055	0.0518	130.4784	0.0024	0.0348	96.4087
350	0.0089	0.0666	137.4917	0.0034	0.0394	111.2179
1050	0.0037	0.0439	146.695	0.0053	0.0539	143.6422

For the training data, it was found that the autoencoders of 4, 6, and 8 layers have similar performance.

As for the test data, the best case was 8 layers composed of BiLSTM cells. However, a chaotic relationship between the number of layers and the error is observed in LSTM data, so it is not possible to say whether the error increases or decreases as the number of layers increases.

Finally, the performance of the autoencoders was tested about the size of the time window. The training data showed little difference in error between the chosen time windows. However, when we analyzed the test data, we noticed that as we increased the size of the time window, we also increased the error. Thus, we realized that the best values for the time window are of 6 time steps.

5.2 Prediction Module

As discussed in the previous chapter, the prediction module is the tool responsible for identifying and predicting the strength of the current market trend. It has two tools, the Market Trend Classifier to identify the current trend, and the Trend Slope Regressor to estimate the strength of that trend. This tool is extremely important because it provides useful information to the Trading Module.

5.2.1 Market Trend Classifier

The Market Trend Classifier was defined with a two-layer BiLSTM architecture followed by a CNN layer. A time window of 6 time steps was defined, with a total of 150 epochs for training.

Determining a Sideways Market Class

This instrument was initially developed to classify the market into three categories: bullish, bearish, and sideways. The classification is based on the slope of the calcu-

lated linear regression. If the slope value is within a certain limit, the market is classified as sideways; if it exceeds this limit upwards, it is classified as bullish; and if it exceeds it downwards, it is classified as bearish. The size of this limit is defined as a percentage of the standard deviation of the samples, ranging from 0% to 25% of the standard deviation, which controls the amplitude of the sideways market class. Furthermore, the experiments for this variation were only carried out using the Bitcoin data. Tables 5.4 to 5.9 show the results of the variation of the amplitude of the Sideways Market class. What was observed was that, as this amplitude increased, there was a decrease in accuracy, average precision, and average recall. We can see that the best values for this metric occur when there is simply no Sideways Market class (or the value of the standard deviation range of the samples is 0%). Therefore, in the experiments that followed, the Sideways Market class was disregarded.

Table 5.4: Metrics, considering a sideways market class with a length of 0% of the standard deviation. Variation from 2 to 10 candles for the linear regression set. (Test Samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
Accuracy	0.5903	0.6953	0.6833	0.7317	0.7057
Precision Bull	0.6124	0.7095	0.7677	0.7430	0.7693
Precision Sideways	0	0	0	0	0
Precision Bear	0.5509	0.6757	0.6173	0.7150	0.6452
Precision Mean	0.58165	0.6926	0.6925	0.7290	0.70725
Recall Bull	0.7077	0.7518	0.6109	0.7932	0.6731
Recall Sideways	0	0	0	0	0
Recall Bear	0.4447	0.6268	0.7725	0.6541	0.7466
Recall Mean	0.5762	0.4596	0.4611	0.4824	0.4732

Table 5.5: Evaluation metrics, considering a sideways market class with a length of 5% of the standard deviation. Variation from 2 to 10 candles for the linear regression set. (Test Samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
Accuracy	0.5452	0.6687	0.667	0.6807	0.6833
Precision Bull	0.5732	0.673	0.6693	0.7257	0.7512
Precision Sideways	0	0	0	0	0
Precision Bear	0.4841	0.6618	0.663	0.6282	0.6177
Precision Mean	0.3525	0.4449	0.4441	0.4513	0.4563
Recall Bull	0.7518	0.7909	0.8044	0.7385	0.6964
Recall Sideways	0	0	0	0	0
Recall Bear	0.3745	0.6003	0.5798	0.6988	0.7494
Recall Mean	0.3754	0.4637	0.4614	0.4791	0.4819

Table 5.6: Evaluation metrics, considering a sideways market class with a length of 10% of the standard deviation. Variation from 2 to 10 candles for the linear regression set. (Test Samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
Accuracy	0.5348	0.64	0.653	0.6617	0.6667
Precision Bull	0.5524	0.6387	0.6452	0.6806	0.7
Precision Sideways	0	0	1	0	0.2048
Precision Bear	0.4925	0.6424	0.667	0.6343	0.6516
Precision Mean	0.3483	0.427	0.7707	0.4383	0.5188
Recall Bull	0.79	0.8291	0.8385	0.807	0.7896
Recall Sideways	0	0	0.0031	0	0.0563
Recall Bear	0.373	0.5729	0.5951	0.6647	0.6658
Recall Mean	0.3907	0.4673	0.4789	0.4918	0.5039

Table 5.7: Evalutaion metrics, considering a sideways market class with a length of 15% of the standard deviation. Variation from 2 to 10 candles for the linear regression set. (Test Samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
Accuracy	0.5022	0.5993	0.6147	0.6143	0.594
Precision Bull	0.5395	0.6263	0.6514	0.6555	0.6351
Precision Sideways	0.6923	0.3478	0.4211	0.304	0.3919
Precision Bear	0.4399	0.5763	0.5712	0.6558	0.5556
Precision Mean	0.5572	0.5168	0.5479	0.5384	0.5275
Recall Bull	0.7194	0.7638	0.7723	0.7964	0.7184
Recall Sideways	0.0324	0.0488	0.0159	0.214	0.0643
Recall Bear	0.4706	0.6368	0.688	0.5656	0.6493
Recall Mean	0.4075	0.4831	0.4921	0.5253	0.4773

Varying the Argmax Threshold

The Softmax function produces values between 0 and 1, which represent the probability that a sample belongs to a given class. In a binary classification problem, the final decision is usually made based on a default threshold of 0.5: if the Softmax value for a class is greater than 0.5, the sample is assigned to that class; otherwise, it does not belong to the class. This threshold is known as the Argmax threshold. To improve the accuracy, precision, and recall metrics, this threshold was adjusted, allowing for more stringent or flexible classification as needed. The experiments were conducted using Bitcoin data, and the Argmax threshold was varied from 0.5 to 0.75, and the impact of these changes on the model performance was analyzed. Tables 5.10 through 5.15 show the results of the evaluation metrics.

Table 5.8: Evaluation metrics, considering a sideways market class with a length of 20% of the standard deviation. Variation from 2 to 10 candles for the linear regression set. (Test Samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
Accuracy	0.5025	0.5473	0.5607	0.586	0.5557
Precision Bull	0.5446	0.5339	0.55	0.6648	0.7317
Precision Sideways	0.4023	0.3627	0.4624	0.4029	0.4101
Precision Bear	0.4749	0.6135	0.6121	0.5272	0.4928
Precision Mean	0.474	0.5034	0.5415	0.5316	0.5448
Recall Bull	0.7508	0.8895	0.8707	0.7091	0.503
Recall Sideways	0.283	0.0551	0.1207	0.0866	0.2362
Recall Bear	0.355	0.4351	0.4505	0.7415	0.8093
Recall Mean	0.4629	0.4599	0.4806	0.5124	0.5162

Table 5.9: Evaluation metrics, considering a sideways market class with a length of 25% of the standard deviation. Variation from 2 to 10 candles for the linear regression set. (Test Samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
Accuracy	0.4415	0.5513	0.5763	0.5513	0.5480
Precision Bull	0.4291	0.6451	0.6633	0.6982	0.7214
Precision Sideways	0.5021	0.4324	0.4449	0.4441	0.3945
Precision Bear	0.4398	0.5362	0.5350	0.4978	0.5708
Precision Mean	0.4570	0.5379	0.5477	0.5467	0.5622
Recall Bull	0.8705	0.6088	0.6953	0.5433	0.5492
Recall Sideways	0.2708	0.4093	0.2662	0.3639	0.6031
Recall Bear	0.0775	0.5998	0.6828	0.7103	0.5026
Recall Mean	0.4063	0.5393	0.5481	0.5392	0.5516

Varying the Asset

Finally, the assets on which the Market Trend Classifiers were trained were varied. The assets used were Bitcoin, Ethereum, Ripple, and Cardano. The assets were trained with only two classes: bull and bear market, and with an Argmax function threshold of 0.5. Tables 5.16 to 5.19 show the evaluation metrics for the assets chosen.

Discussion

The experiments yielded intriguing results. The experiment involving the variation of the amplitude of the Sideways Market Class revealed a decrease across all evaluation metrics. Specifically, not only did accuracy drop, but precision and recall also diminished. This suggests that the model performed best when the Sideways Market Class was entirely disregarded. The next experiment, which involved increasing the Argmax threshold, produced notable improvements in the evaluation metrics. However, this adjustment came

Table 5.10: Evaluation Metrics, considering a threshold of 0.5 in Argmax, from 2 to 14 candles taken for linear regression. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.512	0.642	0.655	0.677	0.670	0.677	0.684
Precision Bull	0.537	0.671	0.680	0.695	0.757	0.763	0.700
Precision Bear	0.503	0.616	0.632	0.658	0.617	0.622	0.666
Precision Mean	0.520	0.644	0.656	0.676	0.687	0.692	0.683
Recall Bull	0.298	0.594	0.632	0.678	0.550	0.564	0.716
Recall Bear	0.734	0.692	0.680	0.675	0.803	0.803	0.647
Recall Mean	0.516	0.643	0.656	0.677	0.677	0.684	0.682
Samples ratio	1.0	1.0	1.0	1.0	1.0	1.0	1.0

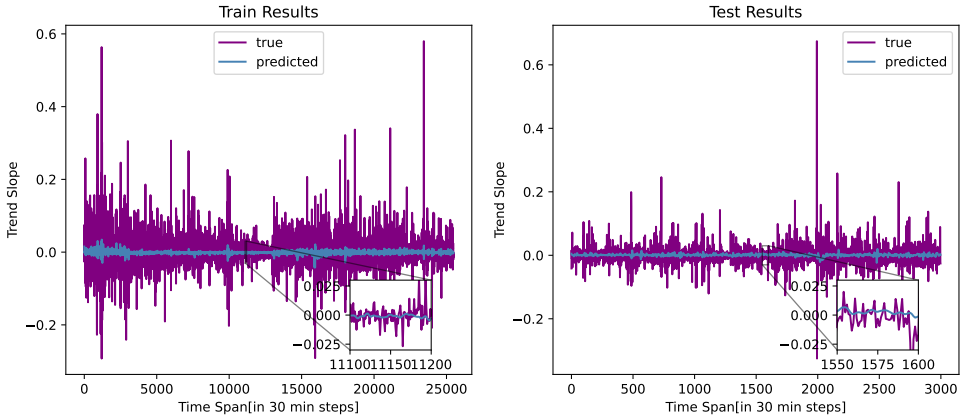
Table 5.11: Evaluation Metrics, considering a threshold of 0.55 in Argmax, from 2 to 14 candles taken for linear regression. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.538	0.664	0.682	0.708	0.696	0.697	0.710
Precision Bull	0.566	0.691	0.705	0.725	0.790	0.788	0.720
Precision Bear	0.532	0.642	0.661	0.689	0.642	0.642	0.697
Precision Mean	0.549	0.667	0.683	0.707	0.716	0.715	0.709
Recall Bull	0.084	0.481	0.548	0.588	0.466	0.505	0.632
Recall Bear	0.377	0.591	0.604	0.591	0.732	0.757	0.568
Recall Mean	0.231	0.536	0.576	0.590	0.599	0.631	0.600
Samples ratio	0.424	0.805	0.843	0.833	0.851	0.895	0.848

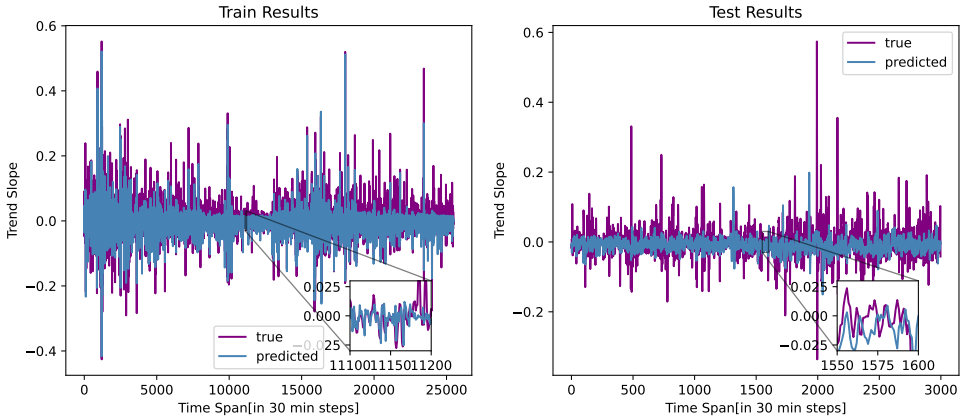
at the cost of a significant reduction in the number of classified samples. Finally, the experiment that varied the asset had minimal impact on the results, indicating that the model is robust across different assets.

5.2.2 Trend Slope Regressor

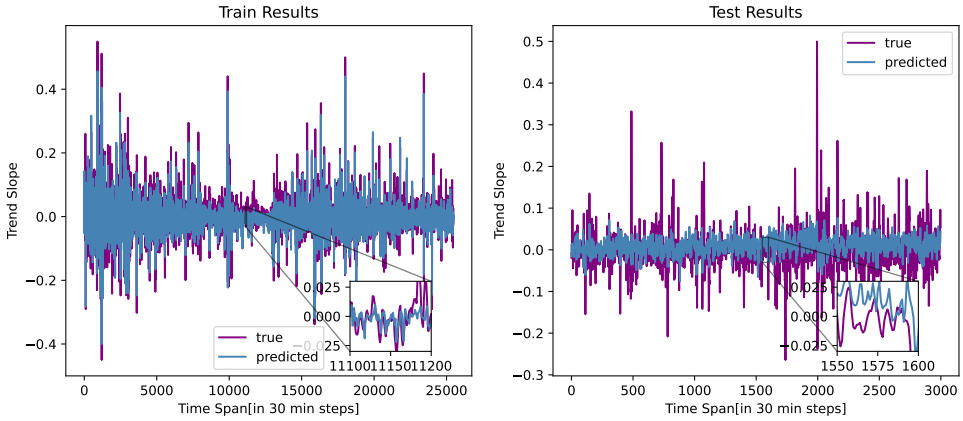
The training and testing results for the trend slope regressor are presented in Table 5.20 and Figure 5.10. Visually, the graphs demonstrate a similar pattern to that of the market trend classifier: as the number of candles used in linear regression increases, the true and predicted value curves tend to align more closely. This can be attributed to the increased volatility of linear regression when calculated with fewer candles. However, Table 5.20 indicates that error values are larger as the number of candles rises. The lowest error values occur when only 2 candles are used for linear regression, which is the worst scenario in the graphical representations. This happens because the model may become trapped in local minima during the gradient descent phase of training, resulting in small error values, but failing to match the overall trend of the actual curve.



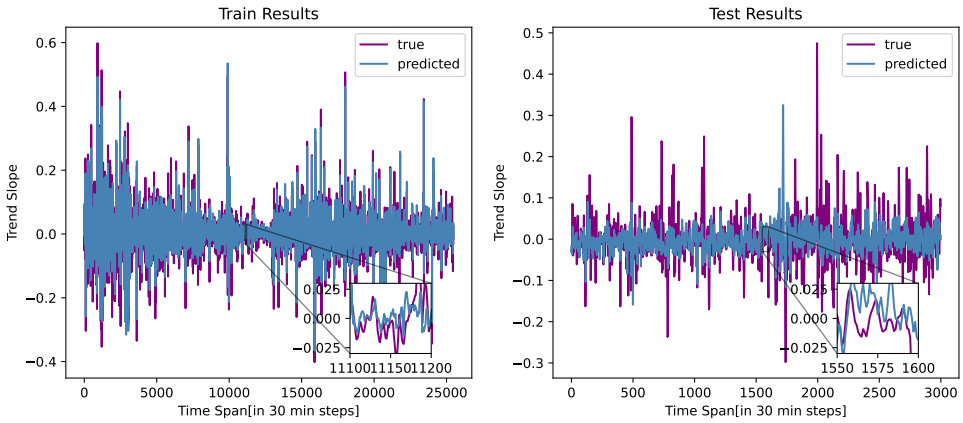
(a) 2 candles



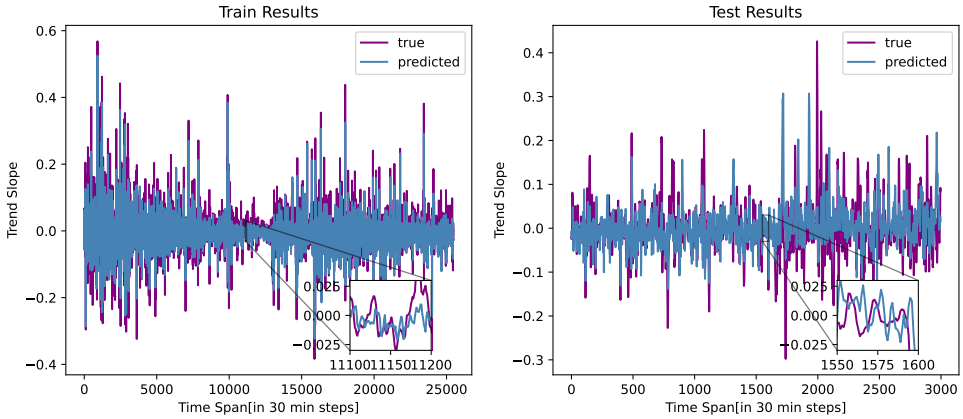
(b) 4 candles



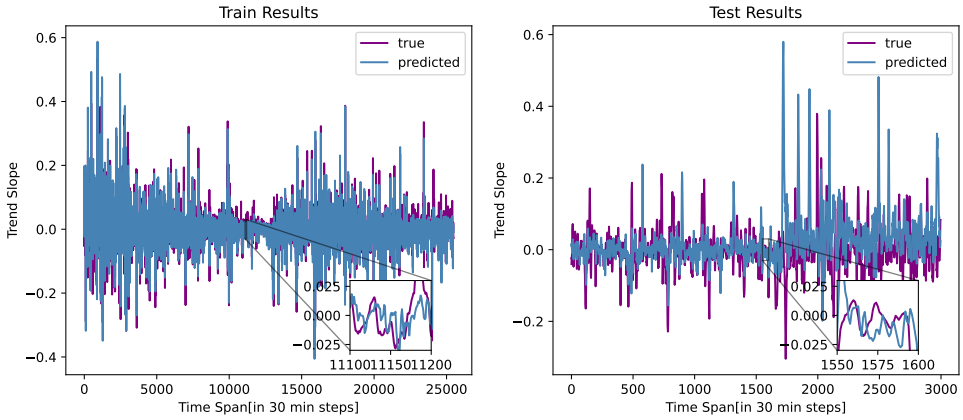
(c) 6 candles



(d) 8 candles



(e) 10 candles



(f) 12 candles

Figure 5.10: Trend slope over time, for the trend slope predictor built with (a) 2 candles, (b) 4 candles, (c) 6 candles, (d) 8 candles, (e) 10 candles, and (f) 12 candles. The curves in purple show the true values of the slope, whereas the curves in blue show the predicted values. The slopes are shown for train and test samples

Table 5.12: Evaluation Metrics, considering a threshold of 0.60 in Argmax, from 2 to 14 candles taken for linear regression. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.532	0.692	0.709	0.738	0.728	0.720	0.733
Precision Bull	0.609	0.718	0.734	0.755	0.823	0.809	0.739
Precision Bear	0.522	0.673	0.687	0.719	0.676	0.668	0.725
Precision Mean	0.566	0.695	0.710	0.737	0.750	0.738	0.732
Recall Bull	0.009	0.378	0.468	0.503	0.388	0.442	0.538
Recall Bear	0.064	0.485	0.507	0.493	0.650	0.697	0.488
Recall Mean	0.036	0.431	0.487	0.498	0.519	0.569	0.513
Samples ratio	0.068	0.621	0.686	0.675	0.703	0.780	0.703

Table 5.13: Evaluation Metrics, considering a threshold of 0.65 in Argmax, from 2 to 14 candles taken for linear regression. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.000	0.726	0.742	0.769	0.762	0.743	0.761
Precision Bull	0.000	0.756	0.766	0.786	0.849	0.829	0.761
Precision Bear	0.000	0.703	0.719	0.751	0.716	0.694	0.762
Precision Mean	0.000	0.730	0.743	0.768	0.782	0.761	0.761
Recall Bull	0.000	0.278	0.382	0.414	0.311	0.386	0.443
Recall Bear	0.000	0.376	0.411	0.387	0.549	0.623	0.417
Recall Mean	0.000	0.327	0.396	0.400	0.430	0.505	0.430
Samples ratio	0.000	0.449	0.533	0.521	0.556	0.670	0.566

5.3 Trading Module

The trading module executes buy and sell orders to maximize the agent's profit. In each training episode, the agent starts at a random point within the historical price series and must make decisions based on the characteristics extracted from the data (features). Its possible actions include buying, selling, or maintaining the position in the asset. The episode has a predefined duration, which in this specific case was set at 700 candles. The agent has a set of seven possible actions: it can buy 2%, 4%, or 6% of the available balance; maintain its position without making transactions; or sell 2%, 4%, or 6% of the current position. The reward function is directly proportional to the percentage of profit obtained with the action. For example, if the agent starts with a capital of $10,000 and, at the end of the episode, its balance totals $12,000, the profit is 20%. Therefore, the reward attributed to the agent will be proportional to this percentage of gain. The Rainbow DQN algorithm used in the experiment has several parameters that influence its performance and behavior. The parameter responsible for controlling the agent's exploration, σ (sigma), was set to 0.5, which regulates the intensity of noise addition in the network layers to encourage the exploration of new strategies.

Table 5.14: Evaluation Metrics, considering a threshold of 0.70 in Argmax, from 2 to 14 candles taken for linear regression. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.000	0.764	0.769	0.799	0.793	0.773	0.797
Precision Bull	0.000	0.798	0.792	0.819	0.875	0.851	0.789
Precision Bear	0.000	0.740	0.748	0.776	0.752	0.727	0.806
Precision Mean	0.000	0.769	0.770	0.798	0.813	0.789	0.797
Recall Bull	0.000	0.194	0.282	0.321	0.234	0.326	0.345
Recall Bear	0.000	0.270	0.312	0.293	0.440	0.535	0.344
Recall Mean	0.000	0.232	0.297	0.307	0.337	0.430	0.344
Samples ratio	0.000	0.302	0.386	0.385	0.418	0.549	0.433

Table 5.15: Evaluation Metrics, considering a threshold of 0.75 in Argmax, from 2 to 14 candles taken for linear regression. (Test samples)v

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.000	0.795	0.806	0.837	0.830	0.804	0.822
Precision Bull	0.000	0.834	0.813	0.855	0.896	0.879	0.804
Precision Bear	0.000	0.770	0.800	0.815	0.797	0.761	0.842
Precision Mean	0.000	0.802	0.806	0.835	0.847	0.820	0.823
Recall Bull	0.000	0.123	0.192	0.231	0.168	0.266	0.251
Recall Bear	0.000	0.182	0.229	0.207	0.329	0.446	0.267
Recall Mean	0.000	0.152	0.210	0.219	0.249	0.356	0.259
Samples ratio	0.000	0.190	0.260	0.263	0.295	0.436	0.315

The modeling of the distribution of returns follows the C51 approach, where the minimum and maximum values of the expected return, represented by V_{min} and V_{max}, were set to -30 and +30, respectively. These limits define the interval within which the discrete distribution of the Q function will be constructed.

To improve training efficiency, the network is updated every 4 time steps, which ensures a balance between stability and learning speed.

The representation of the probability distribution of the Q function is made using 71 atoms, allowing for greater granularity in the prediction of returns and better use of the categorical distribution method.

Training episodes are started randomly in one of the four assets considered in the study: Bitcoin, Ethereum, Ripple, and Cardano, ensuring diversity in the market conditions faced by the agent.

5.3.1 Simple Rainbow DQN

The first experiment was made with a dense neural network as the "brain" of the agente in Rainbow DQN. The agent was trained in 10,000 episodes. The training took a total of

Table 5.16: Evaluation Metrics, of **Bitcoin**, from 2 to 14 candles. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.512	0.642	0.655	0.677	0.670	0.677	0.684
Precision Bull	0.537	0.671	0.680	0.695	0.757	0.763	0.700
Precision Bear	0.503	0.616	0.632	0.658	0.617	0.622	0.666
Precision Mean	0.520	0.644	0.656	0.676	0.687	0.692	0.683
Recall Bull	0.298	0.594	0.632	0.678	0.550	0.564	0.716
Recall Bear	0.734	0.692	0.680	0.675	0.803	0.803	0.647
Recall Mean	0.516	0.643	0.656	0.677	0.677	0.684	0.682

Table 5.17: Evaluation Metrics, of **Ethereum**, from 2 to 14 candles. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	**12C**	**14C**
Accuracy	0.525	0.651	0.668	0.688	0.692	0.700	0.702
Precision Bull	0.540	0.645	0.670	0.711	0.735	0.687	0.745
Precision Bear	0.513	0.660	0.665	0.666	0.656	0.719	0.667
Precision Mean	0.527	0.652	0.668	0.689	0.696	0.703	0.706
Recall Bull	0.462	0.729	0.699	0.664	0.636	0.777	0.649
Recall Bear	0.590	0.567	0.635	0.713	0.752	0.617	0.760
Recall Mean	0.526	0.648	0.667	0.689	0.694	0.697	0.704

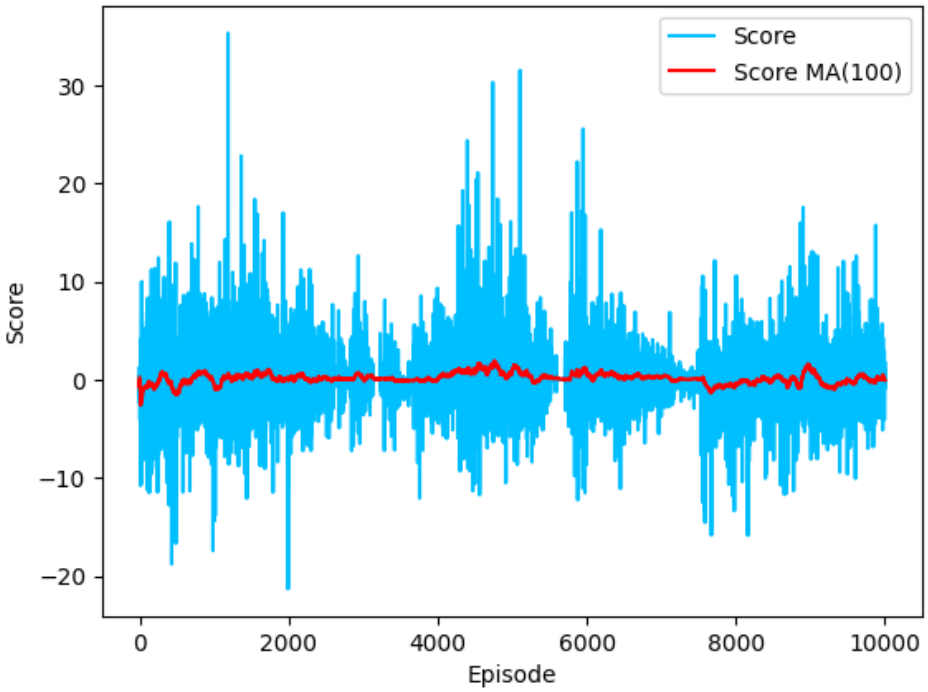
10 hours to train.

Figure 5.11 illustrates the balance and score curves for the Rainbow DQN agent trained without incorporating forecast results from the prediction module and without utilizing a memory buffer. The plot reveals a high standard deviation, as reflected by pronounced fluctuations around the moving average curve. These oscillations indicate unstable learning dynamics, where the agent struggles to maintain consistent performance over time.

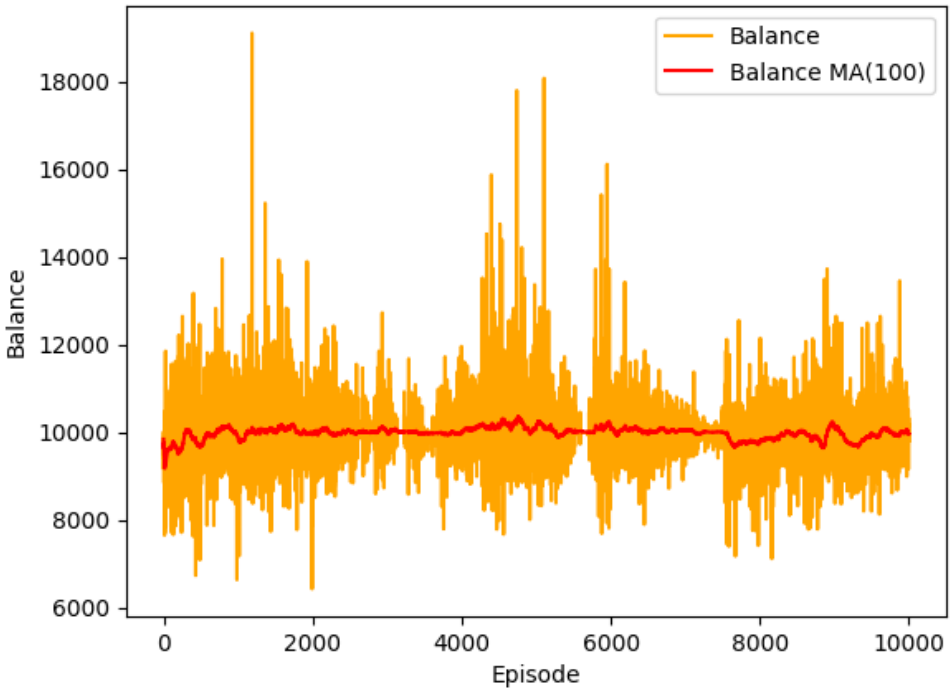
In contrast, Figure 5.12 displays the balance and score curves for the Rainbow DQN agent trained with the forecasted results from the prediction module. The observed standard deviation is significantly lower, with reduced oscillations around the moving average curve. This suggests that integrating predictive estimations enhances stability and improves the agent's ability to generalize across episodes. Furthermore, Tables 5.21 and 5.22 present a comparative analysis of the mean performance over the last 100 and 1000 episodes. The results demonstrate a substantial increase in both balance and score metrics, confirming that leveraging the prediction module leads to a more robust and efficient learning process.

5.3.2 Rainbow DQN with memory

The second experiment was made with a recurrent neural network as the first layer of the neural network of the agent in Rainbow DQN. The agent was also trained in 10,000

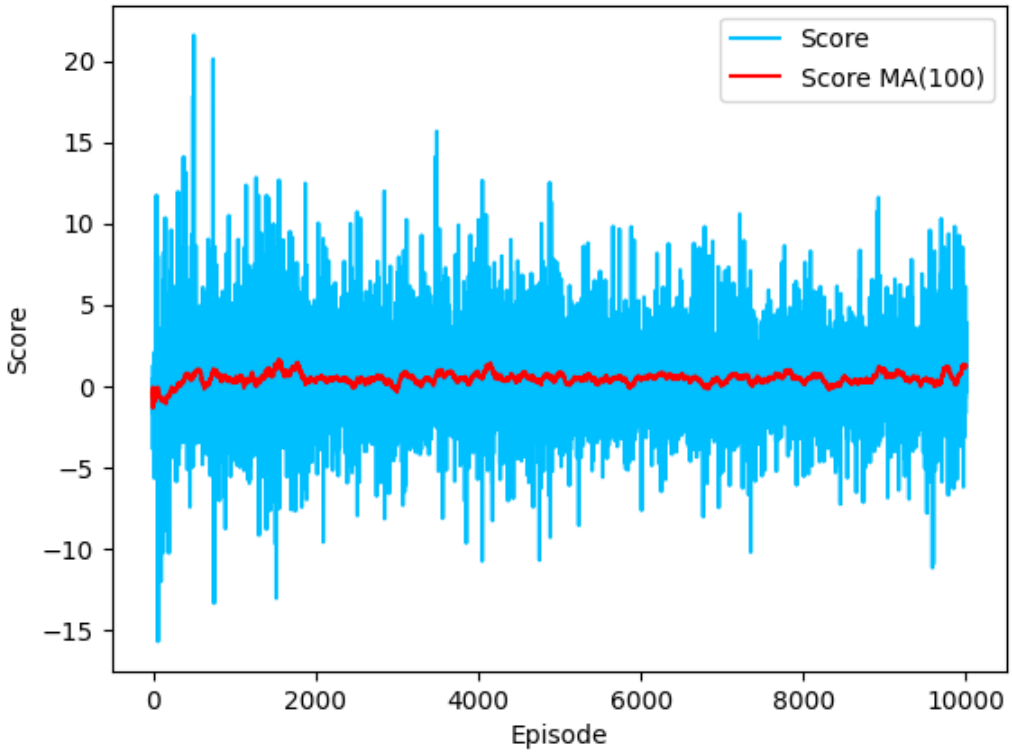


(a) Score

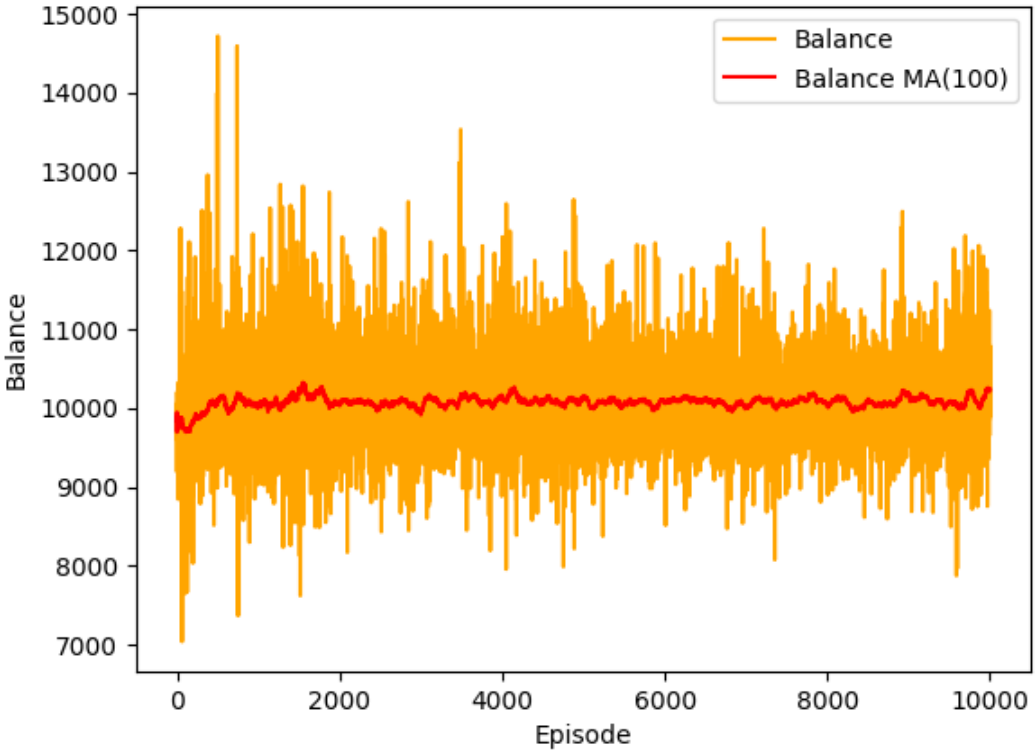


(b) balance

Figure 5.11: The curves of Score and Balance for an agent with no use of the prediction module. The red curve shows the moving average of 100 periods of both score and balance



(a) Score



(b) balance

Figure 5.12: The curves of Score and Balance for an agent with the use of the prediction module. The red curve shows the moving average of 100 periods of both score and balance

Table 5.18: Evaluation Metrics, of **Ripple**, from 2 to 14 candles. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	1**2C**	**14C**
Accuracy	0.519	0.642	0.667	0.681	0.693	0.699	0.684
Precision Bull	0.514	0.622	0.657	0.648	0.670	0.736	0.747
Precision Bear	0.523	0.677	0.680	0.743	0.726	0.671	0.643
Precision Mean	0.519	0.649	0.669	0.696	0.698	0.703	0.695
Recall Bull	0.548	0.770	0.737	0.828	0.782	0.633	0.575
Recall Bear	0.489	0.507	0.592	0.526	0.601	0.767	0.797
Recall Mean	0.519	0.638	0.665	0.677	0.691	0.700	0.686

Table 5.19: Evaluation Metrics, of **Cardano**, from 2 to 14 candles. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**	1**2C**	**14C**
Accuracy	0.525	0.649	0.675	0.687	0.701	0.704	0.695
Position Bull	0.520	0.645	0.670	0.663	0.692	0.692	0.661
Position Bear	0.528	0.653	0.679	0.719	0.710	0.716	0.739
Precision Mean	0.524	0.649	0.675	0.691	0.701	0.704	0.700
Recall Bull	0.381	0.653	0.683	0.756	0.710	0.717	0.767
Recall Bear	0.663	0.645	0.667	0.619	0.692	0.691	0.626
Recall Mean	0.522	0.649	0.675	0.687	0.701	0.704	0.697

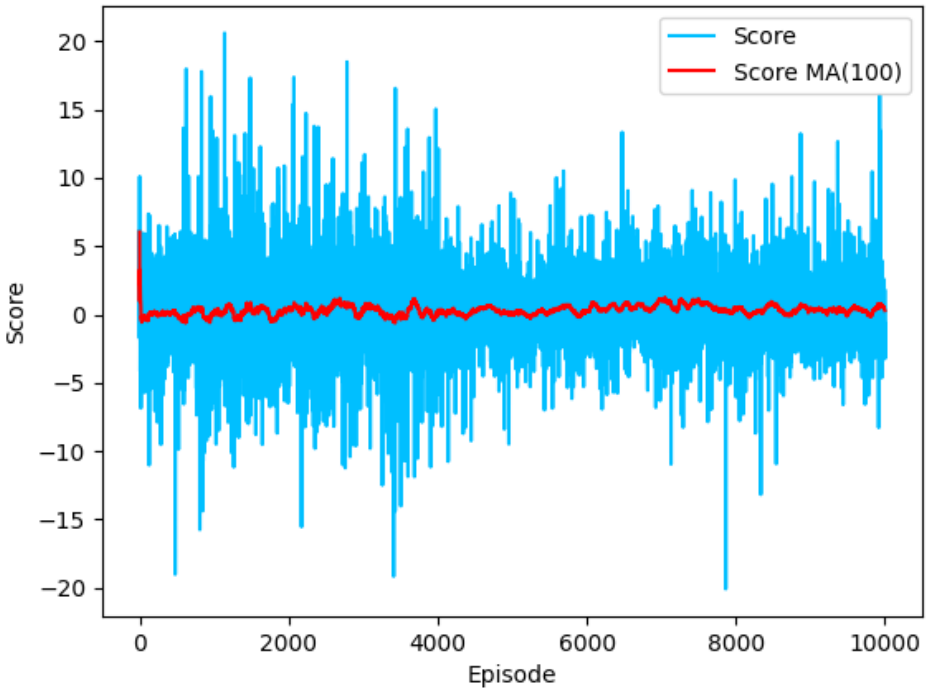
episodes. The training took a total of 3 days to train, and it used a "memory" of 6 time steps.

Figure 5.13 presents the score and balance curves for the Rainbow DQN algorithm trained with a memory buffer but without incorporating forecast data. The plot reveals increased oscillations in both metrics, resulting in a higher standard deviation compared to the model without memory. However, despite the greater variability, the use of memory improves the overall mean performance of the model. This is further supported by Tables 5.22 and 5.21, which show a notable increase in the mean balance and score over the last 100 and 1000 episodes, demonstrating that memory enhances performance.

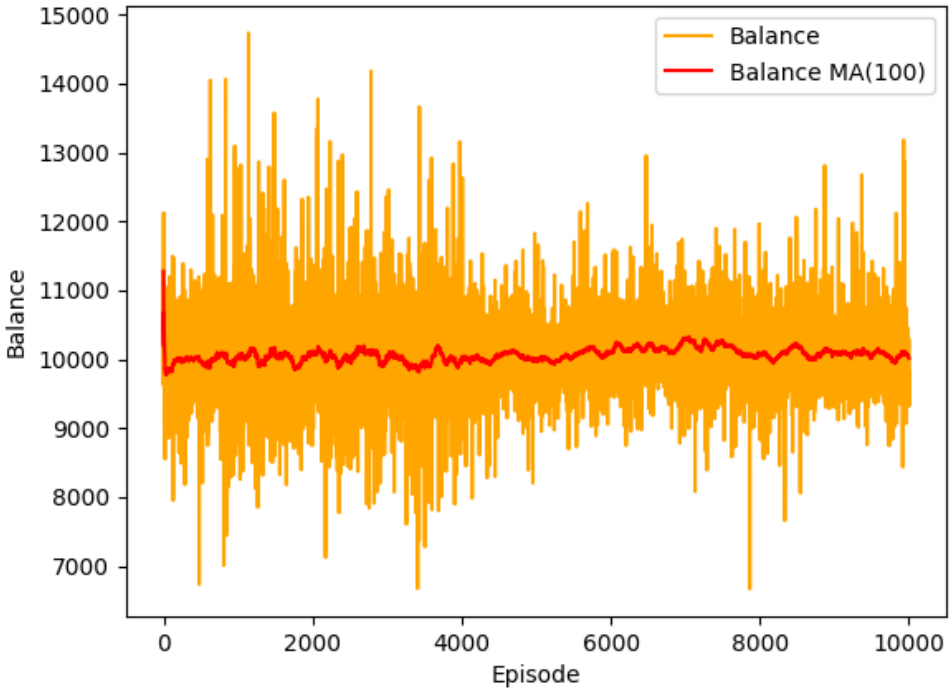
Figure 5.14 displays the score and balance curves for the Rainbow DQN agent trained with both the memory buffer and the forecast data from the prediction module. While the inclusion of memory continues to introduce more oscillations, the overall mean performance further improves. The results indicate that the combination of memory and predictive estimations leads to the best performance among the tested configurations. This is evidenced by the highest mean values observed in Tables 5.22 and 5.21.

5.3.3 Discussion

The experiments performed demonstrated that the exclusive use of technical indicators as input to the Rainbow DQN algorithm did not result in significantly better performance

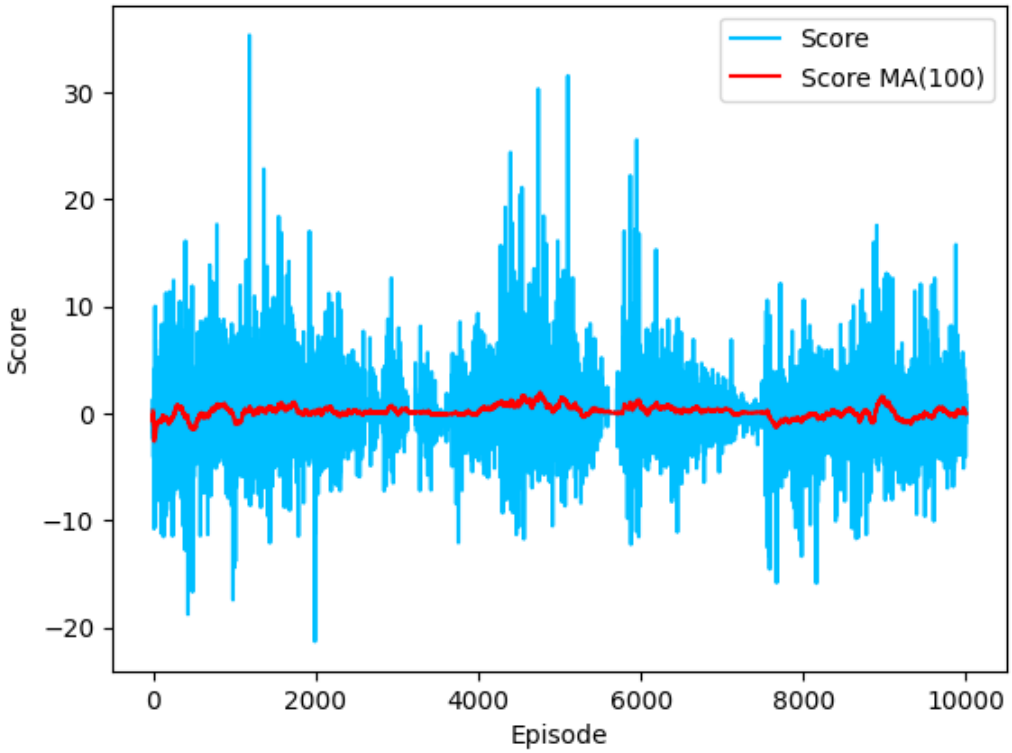


(a) Score

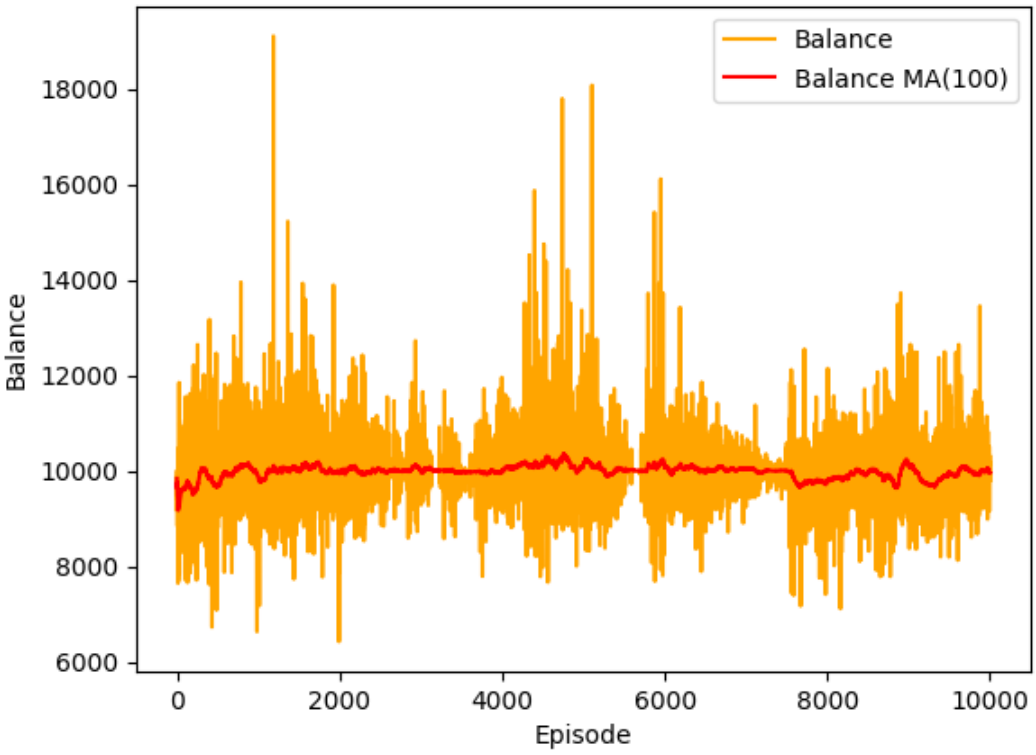


(b) balance

Figure 5.13: The curves of Score and Balance for an agent with memory but no use of the prediction module. The red curve shows the moving average of 100 periods of both score and balance



(a) Score



(b) balance

Figure 5.14: The curves of Score and Balance for an agent with memory and use of the prediction module. The red curve shows the moving average of 100 periods of both score and balance

Table 5.20: Evaluation Metrics, of the Trend Slope Regressor, from 2 to 10 candles. (Test samples)

Metrics	**2C**	**4C**	**6C**	**8C**	**10C**
MSE	0.0021	0.0021	0.0026	0.0033	0.0045
MSE	0.0189	0.0311	0.0360	0.0406	0.0453
MAPE	390	344	628	591	454

Table 5.21: Evaluation of score, for different Rainbow configurations: mean of the last 100 periods, mean of the last 1000 periods, standard deviation of the last 100 periods, and standard deviation of the last 1000 periods

Score	**mean(100)**	**mean(1000)**	**std(100)**	**std(1000)**
Rainbow	-0.0163	-0.1248	1.8390	2.9068
Rainbow w/ Forecast	1.2680	0.6024	2.5056	2.4488
Rainbow w/ Memory	0.3012	0.2987	3.1303	2.2185
Rainbow w/ Memory and Forecast	1.1342	1.9519	4.6745	3.8715

than a purely random approach. This result suggests that technical indicators alone are not sufficient to provide structured information that allows the model to learn effective decision-making policies in the financial market.

However, when the data extracted from technical indicators was combined with the estimates generated by the prediction module, the model performed substantially better than random. This integration allowed the agent not only to identify market patterns with greater accuracy but also to convert this information into profitable trades over time. The results indicate that incorporating deep learning-based predictions improves the Rainbow DQN's ability to make strategic decisions, increasing its effectiveness in identifying buying and selling opportunities.

In addition, considering that financial market behavior can be modeled as a Partially Observable Markovian Decision Process (POMDP), the introduction of a memory mechanism in the initial layers of the Rainbow DQN neural network proved to be a promising approach. The use of this structure allowed the model to store and process historical information, reducing its exclusive dependence on instantaneous observations and enabling more informed decisions. The results indicate that this modification contributed to improving the agent's performance, despite the high variability in financial returns.

Finally, these experiments demonstrated that, although the model still presents instabilities in its return curve, it was able to solve, at least partially, the second problem outlined in the first chapter of this project, that is, the formulation of a profitable trading strategy based on the identification of market trends.

Table 5.22: Evaluation of balance, for different Rainbow configurations: mean of the last 100 periods, mean of the last 1000 periods, standard deviation of the last 100 periods, and standard deviation of the last 1000 periods

Balance	**mean(100)**	**mean(1000)**	**std(100)**	**std(1000)**
Rainbow	9969.4	9918.1	362.77	584.49
Rainbow w/ Forecast	10236	10101	513.95	494.30
Rainbow w/ Memory	10015	10064	630.06	443.36
Rainbow w/ Memory and Forecast	10442	10639	839.16	762.52

Chapter 6

Conclusion

The first chapter introduced the problems that this work aims to solve, outlining the challenges faced in analyzing and operating the cryptocurrency market. The first challenge consists of effectively extracting relevant information from market data, to correctly identify bullish and bearish trends in asset prices. The second problem addressed in this work concerns the application of this trend identification in the execution of profitable operations in the financial market. In other words, in addition to correctly recognizing market movements, it is essential to establish operational strategies that maximize profits and minimize risks, considering the volatility characteristic of cryptocurrencies.

The second chapter presented a review of the main works related to this project, with an emphasis on those that offer theoretical and methodological bases that complement the approach developed here. Among the studies analyzed, those focused on price prediction and the application of reinforcement learning in operations in the financial market stand out. Specifically, this project complements the research of Zhanhong He [4] and Samarazekara [9] on price modeling and forecasting in the financial market, expanding their approaches with improved methodologies adapted to the context of cryptocurrencies. Furthermore, regarding decision-making for automated financial operations, this study builds on and expands on the work of Yasmeen Ankari [10], who uses reinforcement learning to optimize trading strategies. In this way, this project positions itself as an evolution of these previous contributions, unifying and improving forecasting and decision techniques to improve performance in the cryptocurrency market.

The third chapter presented a comprehensive overview of the main neural network architectures used in this project, detailing their mathematical formulations and applicability in the context of time series analysis and reinforcement learning. Initially, dense, convolutional, and recurrent neural networks were introduced, highlighting their respective structures, activation functions, and learning mechanisms. For each of these architectures, the main equations that govern their operation were discussed. In addition, an

introduction to autoencoders was provided, addressing their role in extracting relevant features and reducing dimensionality, essential aspects for optimizing data processing in model training. The chapter also included an explanation of the fundamentals of reinforcement learning, with an emphasis on Rainbow DQN and its six main improvements, which include Double DQN, Dueling Networks, Prioritized Experience Replay, Noisy Networks, Multi-Step Learning, and Distributional RL. Finally, the theoretical basis of the technical indicators that will be used throughout the project was presented, explaining the mathematical formulation behind each of them and their relevance for identifying market patterns. This theoretical basis is essential for building a solid and well-founded model, ensuring that investment decisions are based on quantitative and statistically relevant metrics.

The fourth chapter presented a detailed description of the architecture of each of the three modules that make up the project, highlighting their functionalities, methodologies, and how they are integrated to build a robust system for analysis and decision-making in the financial market. Initially, the feature engineering module was introduced, responsible for extracting and selecting the main technical indicators that will serve as input for subsequent models. This module plays an essential role in building an optimized dataset, filtering relevant variables, and removing redundant or uninformative information. In addition, a solution for dimensionality reduction was presented, ensuring that the model can operate with greater computational efficiency and a lower risk of overfitting, preserving the data representativeness as much as possible. Next, the prediction module was detailed, whose central function is the accurate identification of upward and downward market trends. For this purpose, advanced neural network architectures were employed, combining convolutional networks (CNNs) for automatic extraction of spatial patterns in financial data and BiLSTM recurrent networks to capture temporal and contextual dependencies in historical series. Finally, the trading module was presented, responsible for transforming trend predictions into profitable operational strategies. This module uses the Rainbow DQN algorithm to optimize decision-making in dynamic and highly volatile environments, such as the financial market. Through reinforcement learning, the model learns trading policies that maximize financial returns, identifying the most opportune moments to execute buy and sell operations based on the predictions generated by the previous module. In this way, the chapter provided a comprehensive and structured view of the internal functioning of the project, detailing the interaction between the modules and the theoretical and practical foundations that support its implementation.

Finally, the fifth chapter presented a detailed analysis of the results obtained in the experiments carried out throughout this project, evaluating the performance of each of the three modules that make up the proposed architecture. Initially, the results of the

experiments related to the feature engineering module were discussed, the objective of which was to identify the most efficient configuration for dimensionality reduction. The tests performed with autoencoders allowed us to determine the ideal hyperparameters for information compression without significant loss of relevance, ensuring that the data provided to subsequent modules preserved the essential patterns for market analysis. This step was essential to optimize data representation and reduce the computational complexity of the model. Next, the experiments conducted in the prediction module demonstrated significantly superior performance compared to the random framework in the task of identifying uptrends and downtrends in the financial market. The convolutional and BiLSTM architectures employed allowed us to capture both local patterns in historical data and broader temporal relationships, resulting in considerably higher accuracy and performance metrics. These results confirm the effectiveness of the proposed model in extracting reliable insights from market data. Finally, the experiments performed in the trading module analyzed the model's ability to convert the predictions generated by the previous module into profitable trades in the financial market. The results indicated that, although there was a significant oscillation between periods of profit and loss, the model, on average, was able to obtain positive returns. This behavior was especially evident when predictions from the prediction module were incorporated and the memory buffer was utilized, allowing the Rainbow DQN algorithm to make more informed and efficient decisions over time.

References

[1] Patel, Jigar, Sahil Shah, Priyank Thakkar, and Ketan Kotecha: *Predicting stock market index using fusion of machine learning techniques.* Expert Systems with Applications, 42(4):2162–2172, 2015. viii, 11

[2] Lin, Yaohu, Shancun Liu, Haijun Yang, and Harris Wu: *Stock trend prediction using candlestick charting and ensemble machine learning techniques with a novelty feature engineering scheme.* IEEE Access, 9:101433–101446, 2021. viii, 12

[3] Selvin, Sreelekshmy, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman: *Stock price prediction using lstm, rnn and cnn-sliding window model.* In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017. viii, 3, 12, 19, 69

[4] He, Zhanhong, Junhao Zhou, Hong Ning Dai, and Hao Wang: *Gold price forecast based on lstm-cnn model.* In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1046–1053. IEEE, 2019. viii, 3, 12, 17, 19, 69, 105

[5] Livieris, Ioannis E, Emmanuel Pintelas, and Panagiotis Pintelas: *A cnn–lstm model for gold price time-series forecasting.* Neural computing and applications, 32:17351–17360, 2020. viii, 3, 13, 19

[6] Liang, Yanhui, Yu Lin, and Qin Lu: *Forecasting gold price using a novel hybrid model with iceemdan and lstm-cnn-cbam.* Expert Systems with Applications, 206:117847, 2022. viii, 3, 13, 19

[7] Siami-Namini, Sima, Neda Tavakoli, and Akbar Siami Namin: *The performance of lstm and bilstm in forecasting time series.* In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3285–3292. IEEE, 2019. viii, 3, 13, 17, 19

[8] Zhang, Zhuorui, Hong Ning Dai, Junhao Zhou, Subrota Kumar Mondal, Miguel Martínez García, and Hao Wang: *Forecasting cryptocurrency price using convolutional neural networks with weighted and attentive memory channels.* Expert Systems with Applications, 183:115378, 2021. viii, 3, 14, 19

[9] Samarasekara, Iromie K, Oshan K Mendis, Sapumal Ahangama, and Ajantha S Atukorale: *Dynamic stop-loss approach for short term trades using deep learning.*

In *2022 IEEE International Conference on Big Data (Big Data)*, pages 3537–3546. IEEE, 2022. viii, 4, 14, 17, 19, 105

[10] Ansari, Yasmeen, Sadaf Yasmin, Sheneela Naz, Hira Zaffar, Zeeshan Ali, Jihoon Moon, and Seungmin Rho: *A deep reinforcement learning-based decision support system for automated stock market trading.* IEEE Access, 10:127469–127501, 2022. viii, 4, 15, 18, 19, 105

[11] Wu, Xing, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita: *Adaptive stock trading strategies with deep reinforcement learning methods.* Information Sciences, 538:142–158, 2020. viii, 4, 15

[12] Azhikodan, Akhil Raj, Anvitha GK Bhat, and Mamatha V Jadhav: *Stock trading bot using deep reinforcement learning.* In *Innovations in Computer Science and Engineering: Proceedings of the Fifth ICICSE 2017*, pages 41–49. Springer, 2019. viii, 4, 16

[13] Carta, Salvatore, Andrea Corriga, Anselmo Ferreira, Alessandro Sebastian Podda, and Diego Reforgiato Recupero: *A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning.* Applied Intelligence, 51:889–905, 2021. viii, 4, 16

[14] Li, Yang, Wanshan Zheng, and Zibin Zheng: *Deep robust reinforcement learning for practical algorithmic trading.* IEEE Access, 7:108014–108022, 2019. viii, 16

[15] Rosenblatt, Frank: *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review, 65(6):386, 1958. xvi, 21, 22

[16] Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams: *Learning representations by back-propagating errors.* nature, 323(6088):533–536, 1986. xvi, 24, 26

[17] Werbos, Paul J: *Backpropagation through time: what it does and how to do it.* Proceedings of the IEEE, 78(10):1550–1560, 1990. xvi, 31

[18] Hochreiter, Sepp and Jürgen Schmidhuber: *Long short-term memory.* Neural computation, 9(8):1735–1780, 1997. xvi, 33

[19] Schuster, Mike and Kuldip K Paliwal: *Bidirectional recurrent neural networks.* IEEE transactions on Signal Processing, 45(11):2673–2681, 1997. xvi, 35, 37

[20] Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore: *Reinforcement learning: A survey.* Journal of artificial intelligence research, 4:237–285, 1996. xvi, 39

[21] Fama, Eugene F: *Efficient capital markets: A review of theory and empirical work.* The journal of Finance, 25(2):383–417, 1970. 1

[22] Kendall, Maurice George and A Bradford Hill: *The analysis of economic time-series-part i: Prices.* Journal of the Royal Statistical Society. Series A (General), 116(1):11–34, 1953. 1

[23] Jensen, Michael C: *Some anomalous evidence regarding market efficiency.* Journal of financial economics, 6(2/3):95–101, 1978. 1

[24] Lo, Andrew W and A Craig MacKinlay: *Stock market prices do not follow random walks: Evidence from a simple specification test.* The review of financial studies, 1(1):41–66, 1988. 1

[25] Shleifer, Andrei and Lawrence H Summers: *The noise trader approach to finance.* Journal of Economic perspectives, 4(2):19–33, 1990. 1

[26] Shefrin, Hersh and Meir Statman: *The disposition to sell winners too early and ride losers too long: Theory and evidence.* The Journal of finance, 40(3):777–790, 1985. 2

[27] Derbentsev, Vasily, Andriy Matviychuk, and Vladimir N Soloviev: *Forecasting of cryptocurrency prices using machine learning.* Advanced studies of financial technologies and cryptocurrency markets, pages 211–231, 2020. 2

[28] Livieris, Ioannis E, Niki Kiriakidou, Stavros Stavroyiannis, and Panagiotis Pintelas: *An advanced cnn-lstm model for cryptocurrency forecasting.* Electronics, 10(3):287, 2021. 3, 77

[29] Faraz, Mehrnaz, Hamid Khaloozadeh, and Milad Abbasi: *Stock market prediction-by-prediction based on autoencoder long short-term memory networks.* In *2020 28th Iranian conference on electrical engineering (ICEE)*, pages 1–5. IEEE, 2020. 4, 10, 17, 19

[30] Cochrane, John H: *Presidential address: Discount rates.* The Journal of finance, 66(4):1047–1108, 2011. 6

[31] Kozak, Serhiy, Stefan Nagel, and Shrihari Santosh: *Shrinking the cross-section.* Journal of Financial Economics, 135(2):271–292, 2020. 6

[32] Jain, Anil and Douglas Zongker: *Feature selection: Evaluation, application, and small sample performance.* IEEE transactions on pattern analysis and machine intelligence, 19(2):153–158, 1997. 7

[33] Chandrashekar, Girish and Ferat Sahin: *A survey on feature selection methods.* Computers & Electrical Engineering, 40(1):16–28, 2014. 7

[34] Niu, Tong, Jianzhou Wang, Haiyan Lu, Wendong Yang, and Pei Du: *Developing a deep learning framework with two-stage feature selection for multivariate financial time series forecasting.* Expert Systems with Applications, 148:113237, 2020. 7

[35] Peng, Yaohao, Pedro Henrique Melo Albuquerque, Herbert Kimura, and Cayan Atreio Portela Bárcena Saavedra: *Feature selection and deep neural networks for stock price direction forecasting using technical analysis indicators.* Machine Learning with Applications, 5:100060, 2021. 8, 17, 64, 65, 66, 67

[36] Haq, Anwar Ul, Adnan Zeb, Zhenfeng Lei, and Defu Zhang: *Forecasting daily stock trend using multi-filter feature selection and deep learning.* Expert Systems with Applications, 168:114444, 2021. 8, 17, 64, 65, 66

[37] Ji, Gang, Jingmin Yu, Kai Hu, Jie Xie, and Xunsheng Ji: *An adaptive feature selection schema using improved technical indicators for predicting stock price movements.* Expert Systems with Applications, 200:116941, 2022. 8, 17, 65, 66, 67

[38] De Oliveira, Fagner A, Cristiane N Nobre, and Luis E Zárate: *Applying artificial neural networks to prediction of stock price and improvement of the directional prediction index–case study of petr4, petrobras, brazil.* Expert systems with applications, 40(18):7596–7606, 2013. 9, 17, 66, 67

[39] McCulloch, Warren S and Walter Pitts: *A logical calculus of the ideas immanent in nervous activity.* The bulletin of mathematical biophysics, 5:115–133, 1943. 21

[40] Minsky, Marvin and Seymour Papert: *An introduction to computational geometry.* Cambridge tiass., HIT, 479:480, 1969. 22

[41] Sutton, Richard S and Andrew G Barto: *Reinforcement learning: An introduction.* MIT press, 2018. 38

[42] Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver: *Rainbow: Combining improvements in deep reinforcement learning.* In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 48