**Universidade Federal Fluminense**

**DOUGLAS DE OLIVEIRA**

# Desenvolvimento de modelo de inteligência artificial aplicado a *day trading* com criptoativos baseado em *Double Deep Q-Learning*

VOLTA REDONDA

2023

Ficha catalográfica automática - SDC/BEM
Gerada com informações fornecidas pelo autor

O48d Oliveira, Douglas de
Desenvolvimento de modelo de inteligência artificial aplicado a day trading com criptoativos baseado em Double Deep Q-Learning / Douglas de Oliveira. - 2023.
93 f.: il.

Orientador: Weslley Luiz da Silva Assis.
Coorientador: Vanessa Silva Garcia.
Dissertação (mestrado)-Universidade Federal Fluminense, Escola de Engenharia Industrial e Metalúrgica de Volta Redonda, Volta Redonda, 2023.

1. Aprendizado de máquina. 2. Algotrading. 3. Aprendizado por reforço. 4. Double Deep Q-Learning. 5. Produção intelectual. I. Assis, Weslley Luiz da Silva, orientador. II. Garcia, Vanessa Silva, coorientadora. III. Universidade Federal Fluminense. Escola de Engenharia Industrial e Metalúrgica de Volta Redonda. IV. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

Desenvolvimento de modelo de inteligência artificial aplicado a *day trading* com criptoativos baseado em *Double Deep Q-Learning*

Douglas de Oliveira

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Métodos matemáticos e computacionais aplicados à engenharia e ciência.

Aprovada por:

Prof. Weslley Luiz da Silva Assis, Ph.D. / MCCT-UFF
(Presidente)

Prof. Vanessa Silva Garcia, Ph.D. / MCCT-UFF

Digitally signed by Alberto Paiva
Date: 2023.12.13 16:20:41 -03'00'

Prof. Alberto Paiva, Ph.D. / UFF

Prof Amaury Alvarez Cruz, Ph.D. /UFRJ

Volta Redonda, 23 de Novembro de 2023.

*Dedico este trabalho ao meu Senhor e Salvador, Jesus Cristo, reconhecendo Sua graça imerecida que permeia todos os aspectos da minha vida. Que este esforço acadêmico seja uma expressão humilde de gratidão por Sua orientação constante e pela graça abundante que Ele generosamente concede.*

# Agradecimentos

Agradeço, em primeiro lugar, a Deus por Sua graça abundante em minha vida.

Expresso minha sincera gratidão ao meu professor orientador, Weslley Luiz da Silva Assis, por dedicar seu tempo e compartilhar seu vasto conhecimento durante os anos de orientação e colaboração neste projeto.

Quero estender meus agradecimentos à minha co-orientadora, Vanessa Silva Garcia, por seu valioso apoio e contribuições para o desenvolvimento deste trabalho.

Agradeço aos estimados professores do MCCT, que me forneceram orientação e apoio inestimáveis ao longo desta jornada acadêmica. Além disso, agradeço ao Professor Tiago Neves do VCE/EEIMVR/UFF pela gentileza de disponibilizar o template.

Não posso deixar de mencionar meu profundo agradecimento a meu pai e meu irmão, cujo apoio constante tem sido fundamental em toda a minha jornada.

Minha mãe, Mara Ruth Pereira de Oliveira, merece um agradecimento especial por seu exemplo de vida inspirador, suas orações constantes e suas palavras de encorajamento, que foram uma fonte de força durante os momentos desafiadores.

Por fim, quero expressar minha gratidão a todos os professores, amigos e colegas de estudo que estiveram ao meu lado durante todo o processo até este ponto. Suas contribuições e apoio foram inestimáveis para o sucesso deste trabalho.

Agradeço a todos vocês por fazerem parte desta jornada e por tornarem este projeto possível.

# Resumo

A análise das flutuações nos preços dos ativos financeiros desperta considerável interesse entre os participantes do mercado financeiro, pois oferece informações cruciais para orientar decisões de negociação. No cenário financeiro contemporâneo, o fenômeno *algo-trading* domina, caracterizado pela execução automatizada de operações, sem intervenção direta de agentes humanos. O presente trabalho apresenta desenvolvimento de um modelo computacional inovador projetado para tomar decisões de negociação diárias em relação a ativos financeiros. Baseando-se em análises das variações diárias nos preços e volumes de negociação, incorpora indicadores técnicos fundamentais, como média móvel, Bandas de Bollinger, Índice de Força Relativa (RSI) e *Stop and Reverse* (SAR) Parabólico. Esses parâmetros foram escolhidos por sua capacidade de caracterizar o comportamento dos ativos e oferecer previsões em intervalos curtos. Devido à natureza intrinsecamente imprevisível dos mercados, o modelo foi concebido para ser adaptativo, ajustando-se às flutuações temporais com base no desempenho observado. Além disso, o modelo incorpora o capital disponível e o patrimônio líquido do investidor de forma cautelosa para otimizar as decisões de negociação, reconhecendo a inutilidade da capacidade de compra quando o capital não está acessível. A abordagem metodológica escolhida para enfrentar esse desafio é o *Double Deep Q-Learning* (DDQL), que se destaca pelo uso de dois estimadores baseados em redes neurais artificiais (RNA). Esses estimadores otimizam a função Q, que orienta as decisões financeiras tomadas pela RNA. O DDQL representa uma estratégia de aprendizado por reforço, conferindo ao modelo a flexibilidade necessária para se adaptar e ajustar às flutuações dos dados de entrada. O modelo alcançou um aumento no patrimônio líquido médio em transações de curto prazo, frequentemente realizadas em intervalos horários, especialmente no âmbito do trading de moedas digitais. Essa conquista é atribuída à habilidade das RNAs no modelo de se adaptarem aos dados, juntamente com o sistema de aprendizado por reforço, que identifica e corrige desvios automaticamente, buscando maximizar recompensas futuras durante oscilações de mercado. O modelo demonstrou ser viável para essa aplicação, alcançando seus objetivos de ganhos. Embora exija alguns ajustes, sua aplicação pode ser implementada em cenários reais.

# Abstract

The analysis of fluctuations in the prices of financial assets arouses considerable interest among participants in the financial market, as it offers crucial insights to guide trading decisions. In the contemporary financial landscape, the phenomenon of algotrading predominates, characterized by the automated execution of operations without direct human intervention. This work introduces the development of an innovative computational model designed to make daily trading decisions regarding financial assets. Building on analyses of daily variations in prices and trading volumes, it incorporates fundamental technical indicators such as moving averages, Bollinger Bands, Relative Strength Index (RSI), and Parabolic SAR. These parameters were chosen for their ability to characterize asset behavior and provide short-term interval forecasts. Due to the inherently unpredictable nature of markets, the model was designed to be adaptive, adjusting to temporal fluctuations based on observed performance. Additionally, the model cautiously incorporates available capital and investor equity to optimize trading decisions, recognizing the futility of purchasing power when capital is not accessible. The chosen methodological approach to tackle this challenge is Double Deep Q-Learning (DDQL), distinguished by the use of two estimators based on artificial neural networks (ANNs). These estimators optimize the Q-function, which guides the financial decisions made by the ANN. DDQL represents a reinforcement learning strategy, providing the model with the necessary flexibility to adapt and adjust to input data fluctuations. The model succeeded in increasing the average net worth in short-term transactions, often conducted on an hourly basis, especially in the context of digital currency trading. This achievement is attributed to the ability of ANNs in the model to adapt to data, coupled with the reinforcement learning system, which automatically identifies and corrects deviations, seeking to maximize future rewards during market fluctuations. The model has proven to be feasible for this application, achieving its profit objectives. Although some adjustments are needed, its application can be implemented in real-world scenarios.

# Palavras-chave

1. Aprendizado de máquina
2. Previsão da Cotação de Criptoativos
3. Algotrading
4. Double Deep Q-Learning
5. Aprendizado por reforço

# Keywords

1. Machine Learning
2. Cryptocurrency Price Prediction
3. Algorithmic Trading
4. Double Deep Q-Learning
5. Reinforcement Learning

# Sumário

**Lista de Figuras** **xii**

**Lista de Símbolos, Siglas e Abreviaturas** **xiii**

**1 Introdução** **15**

1.1 Contextualização . . . . . 15

1.2 Objetivos . . . . . 16

1.2.1 Objetivo Geral . . . . . 16

1.2.2 Objetivo Específicos . . . . . 17

1.3 Apresentação dos capítulos . . . . . 17

**2 Referencial Teórico** **18**

2.1 Aprendizado de máquina . . . . . 18

2.2 Aprendizado por Reforço . . . . . 20

2.3 Processo de Decisão de Markov . . . . . 21

2.3.1 Estado, agente, recompensa e desconto . . . . . 21

2.4 A equação de Bellman . . . . . 23

2.5 Política . . . . . 24

2.5.1 Iteração de Política . . . . . 26

2.5.2 Iteração de Valor . . . . . 27

2.6 Aprendizado por diferenças temporais . . . . . 28

2.7 Model-based and model-free learning . . . . . 28

2.8 On-policy learning v.s. off-policy learning . . . . . 29

2.9 Q-Learning . . . . . 29
2.9.1 Dilema *Exploration* e *Exploitation* . . . . . 31
2.9.2 A estratégia $\epsilon$-*greedy* . . . . . 32
2.10 Deep Q-Learning . . . . . 33
2.10.1 Experiência de Repetição . . . . . 33
2.11 Double Deep Q-Learning . . . . . 34
2.11.1 Redes *Online* e *Alvo* . . . . . 36
2.12 Redes Neurais Artificiais . . . . . 36
2.12.1 Redes *Feedforward* . . . . . 38
2.12.2 Backpropagation . . . . . 38
2.13 Funções de ativação . . . . . 39
2.13.1 Função Relu e suas variantes . . . . . 39
2.14 Otimizadores . . . . . 41
2.14.1 Gradiente Descendente . . . . . 41
2.14.2 Taxa de Aprendizagem e Momento . . . . . 42
2.15 Sobre-ajuste e Sub-ajuste . . . . . 42
2.16 Regularização . . . . . 43
2.16.1 *Dropout* . . . . . 43
2.17 Algotrading e o Mercado de ações . . . . . 44
2.17.1 A Teoria de Dow . . . . . 45
2.17.2 Indicadores técnicos . . . . . 46

**3 Estado da Arte 47**

3.1 Reinforcement Learning for Trading . . . . . 48
3.2 Deep Reinforcement Learning for Trading . . . . . 48
3.3 A Q-learning agent for automated trading in equity stock markets . . . . . 49

3.4 Improving Financial Trading Decisions Using Deep Q-learning: Predicting the Number of Shares, Action Strategies, and Transfer Learning . . . . . 50

3.5 Stock trading rule discovery with double deep Q-network . . . . . 51

**4 Metodologia 53**

4.1 Seleção de dados . . . . . 53

4.2 Pré-processamento de Dados . . . . . 55

4.3 Indicadores técnicos . . . . . 56

4.3.1 Média Móvel Simples . . . . . 56

4.3.2 Bandas de Bollinger . . . . . 57

4.3.3 SAR Parabólico . . . . . 58

4.3.4 Índice de Força Relativa . . . . . 58

4.4 Modelo de Aprendizado de Máquina . . . . . 59

4.4.1 Espaço dos estados . . . . . 60

4.4.2 Agente . . . . . 60

4.4.3 Espaço das ações . . . . . 61

4.4.4 Ambiente . . . . . 61

4.4.5 Recompensa . . . . . 61

4.4.6 Parâmetros iniciais . . . . . 62

4.5 Estratégia de negociação . . . . . 62

4.6 Configuração do Ambiente de Experimentação . . . . . 63

4.7 Implementação do Projeto . . . . . 64

**5 Resultados e Discussões 68**

5.1 Resolução do Desafio de Memory Overflow . . . . . 68

5.2 O modelo . . . . . 69

5.2.1 Dinâmica do modelo . . . . . 69

5.3 Fluxogramas . . . . . 70

5.3.1 Preparando o ambiente . . . . . 70
5.3.2 Obtenção dos dados . . . . . 71
5.3.3 Incialização do Modelo . . . . . 71
5.3.4 Visão Geral do Modelo . . . . . 73
5.3.5 Ambiente de Negociação . . . . . 74
5.3.6 Agente e processo de treinamento . . . . . 75
5.4 Geração de Resultados Gráficos . . . . . 77
5.5 Apresentação e discussão dos resultados obtidos . . . . . 78
5.5.1 Comparação CPU vs GPU . . . . . 79
5.5.2 Avaliação do Modelo com Variação no Número de Episódios . . . . 82

**6 Conclusões e Sugestões para Trabalhos Futuros 86**

6.1 Conclusões . . . . . 86
6.2 Sugestões para Trabalhos Futuros . . . . . 87

**Referências 88**

# Lista de Figuras


1 Aprendizado por Reforço . . . . . 22

2 Double Deep Q-Learning . . . . . 36

3 Rede Neural . . . . . 37

4 Ativo financeiro Bitcoin (cotação em dólar) . . . . . 54

5 Ativo financeiro Ethereum (cotação em dólar) . . . . . 55

6 DDQL: Etapa de experimentação com o ambiente . . . . . 64

7 DDQL: Etapa de treinamento . . . . . 65

8 Preparação do ambiente . . . . . 70

9 Obtenção dos dados . . . . . 71

10 Execução do modelo . . . . . 72

11 Visão geral do processo de treinamento . . . . . 73

12 Ambiente de Treinamento . . . . . 75

13 O agente e o processo de treinamento . . . . . 76

14 Resultados Gráficos . . . . . 77

15 Comparativo do Tempo de processamento por Rede e Unidade de Processamento . . . . . 81

16 Comparativo do Tempo de processamento por Rede e Unidade de Processamento . . . . . 83

17 Comparativo da Taxa de sucesso por Rede e Unidade de Processamento . . 84

# Lista de Símbolos, Siglas e Abreviaturas

| | | |
|---|---|---|
| ANN | : | Artificial Neural Network |
| API | : | Application Programming Interface |
| DDQL | : | *Double-Deep Q-Learning* |
| DL | : | *Deep Learning* |
| DNN | : | *Deep Neural Network* |
| DQL | : | *Deep Q-Learning* |
| DRL | : | *Deep Reinforcement Learning* |
| MDP | : | *Markov Decision Process* |
| ML | : | *Machine Learning* |
| MMS | : | *Média Móvel Simples* |
| NASDAQ | : | *National Association of Securities Dealers Automatic Quotation System* |
| NN | : | *Neural Network* |
| OCHL | : | Open, High, Low and Closing prices |
| RL | : | *Reinforcement Learning* |
| RRL | : | *Recurrent Reinforcement Learning* |
| RSI | : | *Relative Strength Index* |
| SMA | : | *Simple Moving Average* |
| TWAP | : | *Time-Weighted Average Price* |

# Lista de códigos

Código 5.1 Execução com Download de Dados . . . . . 71

Código 5.2 Variáveis de ambiente no .env . . . . . 72

Código 5.3 Formas de realizar o início da execução do modelo . . . . . 73

# Capítulo 1

# Introdução

## 1.1 Contextualização

A era das máquinas de inteligência artificial geral, aquelas que possuirão a capacidade de entender e aprender qualquer tarefa sem especificidade e com liberdade de interesses, será um marco na evolução da sociedade. Apesar de ser dito que o marco foi alcançado pelo 'Gato' da *DeepMind* e que seria apenas necessário resolver os problemas de escala do modelo, este aparenta tratar-se apenas de um sistema de inteligencia artificial multi-modal avançado [1].

A área de aprendizado de máquina, que é uma subárea da inteligência artificial, tem desempenhado um importante papel em nossa sociedade há décadas e está cada vez mais presente em nosso cotidiano. Suas aplicações incluem robôs de investimento, recomendação de filmes em serviços de *streaming*, filtragem de spam em e-mails e modelos de gerenciamento de corridas em aplicativos de caronas.

Este trabalho concentra-se no desenvolvimento de um agente autônomo capaz de realizar negociações diárias de ativos financeiros, conhecidas como *trading*. O *trading* é uma importante prática para empresas que investem em títulos e envolve operações de curto prazo realizadas ao longo de horas, dias ou semanas, com o objetivo de maximizar o capital do investidor que alocou recursos em ativos financeiros [2].

Os agentes autônomos são uma categoria de negociadores algorítmicos, conhecidos como *algotrading*, que consistem em um conjunto de instruções computacionais que permitem a execução automatizada de negociações. Estima-se que o *algotrading* represente mais de 80% do volume de negociações no mercado de ações dos Estados Unidos [3, 4].

Devido à importância desse tema, as negociações financeiras têm sido objeto de nu-

merosos estudos, uma vez que a capacidade de identificar tendências bem estabelecidas permite fazer previsões de médio e longo prazo [5, 6].

No entanto, a imprevisibilidade inerente aos processos estocásticos que descrevem o comportamento dos ativos financeiros cria a oportunidade de desenvolver modelos que possam se ajustar continuamente durante a previsão autônoma. Essa complexidade é influenciada por fatores como políticas públicas, relatórios trimestrais de empresas, notícias de negócios, influências internacionais e conflitos de interesses comerciais, que afetam diretamente o comportamento dos ativos [7].

Além disso, o fator humano desempenha um papel significativo, uma vez que as emoções podem afetar negativamente as análises e as decisões tomadas devido ao medo de perdas futuras ou ao otimismo excessivo em relação aos ganhos consecutivos [8].

Portanto, como solução para esses desafios, buscou-se desenvolver um modelo de aprendizado por reforço capaz de extrair características desses dados de processos estocásticos, permitindo a realização de previsões sobre os preços dos ativos e a capacidade de se adaptar às oscilações futuras, avaliando o impacto das ações tomadas e ajustando-se às novas tendências do mercado enquanto busca maximizar as recompensas futuras [9].

O aprendizado por reforço continua sendo um campo de estudo relevante e promissor na gestão de portfólio, com o uso de Q-learning, embora apresente desafios de escalabilidade [2]. Além disso, o uso de *double and dueling double deep q-learning* tem se mostrado altamente lucrativo na negociação de criptomoedas [10], e o *double deep q-learning* (DDQL) tem superado os modelos supervisionados na previsão de tendências, fornecendo ferramentas aos investidores para tomar decisões mais rentáveis [11].

## 1.2 Objetivos

Esta sessão visa apresentar os objetivos deste trabalho.

### 1.2.1 Objetivo Geral

O objetivo geral deste trabalho é implementar um agente de aprendizado por reforço baseado na técnica Double Deep Q-Learning (DDQL) para realizar operações horárias de compra e venda de ativos financeiros com o objetivo de aumentar o patrimônio líquido futuro.

### 1.2.2 Objetivo Específicos

Para alcançar o objetivo geral, são definidos os seguintes objetivos específicos:

- Avaliar e selecionar o mercado financeiro apropriado para a extração de dados.
- Escolher o agente autônomo mais adequado entre as diversas opções disponíveis, considerando a solução do problema em estudo.
- Propor e implementar um agente de aprendizado por reforço que seja capaz de executar operações e obter lucro (ou seja, aumentar o patrimônio líquido).
- Capacitar o agente para aprender e adaptar suas estratégias durante sua atuação no mercado financeiro.
- Tornar o agente capaz de aprender enquanto opera no mercado.

A definição desses objetivos orienta a implementação do agente autônomo e a avaliação de seu desempenho ao longo do tempo.

## 1.3 Apresentação dos capítulos

No segundo capítulo, apresenta-se o referencial teórico, que estabelece os fundamentos necessários para as discussões subsequentes. Inicialmente, aborda-se o conceito de aprendizado de máquina, que serve como base para as discussões posteriores. Em seguida, são explorados aspectos do aprendizado por reforço, com ênfase em *Q-Learning* e suas variantes. Por fim, encerra-se o capítulo com uma análise das negociações financeiras conduzidas por meio do algoritmo de negociação algorítmica, conhecido como *algotrading*.

O terceiro capítulo apresenta o estado da arte, destacando os avanços alcançados no campo até o momento.

No quarto capítulo, é detalhada a metodologia empregada neste estudo. Nele, são descritos o processo de desenvolvimento do modelo, a fonte de dados utilizada e as ferramentas empregadas.

No quinto capítulo, são apresentados os resultados obtidos pelo modelo implementado. O sexto e último capítulo aborda as conclusões deste estudo e propõe direções para trabalhos futuros.

# Capítulo 2

# Referencial Teórico

Com o objetivo de emular a capacidade intelectual humana de forma artificial, foram conduzidos inúmeros estudos para replicar o funcionamento do cérebro humano e seu processo de aprendizado em programas de computador. Entre as várias abordagens disponíveis, o aprendizado por reforço se destaca, uma vez que seu processo de aprendizado assemelha-se ao método de tentativa e erro, semelhante à forma como uma criança humana aprende.

Este capítulo tem como propósito apresentar o embasamento teórico fundamental para uma compreensão aprofundada do aprendizado por reforço, com foco especial na abordagem conhecida como DDQL. Inicialmente, foi abordada a definição do aprendizado de máquina.

## 2.1 Aprendizado de máquina

Os algoritmos de aprendizado de máquina têm como objetivo, de certa forma, emular a capacidade cognitiva humana. Para alcançar esse feito em um contexto de aprendizado supervisionado, um cientista de dados seleciona um algoritmo que seja adequado à natureza dos dados de treinamento. Durante o processo de treinamento, esse algoritmo é ajustado para se adaptar às peculiaridades dos dados, resultando em um modelo matemático capaz de, ao receber dados de entrada que não foram usados durante o treinamento, produzir respostas com um erro e uma precisão aceitáveis. Esse modelo pode, assim, generalizar seu conhecimento e fazer previsões ou classificações precisas em novos dados [12].

Devido à ampla variedade de técnicas desenvolvidas para abordar uma diversidade de

problemas, o campo do Aprendizado de Máquina (ML) pode ser categorizado com base na natureza do problema a ser resolvido e na abordagem utilizada. Dentre as técnicas mais notáveis, destacam-se:

- **Aprendizado Supervisionado**: Nesse cenário, o algoritmo é treinado utilizando dados de entrada que estão previamente rotulados com a saída desejada. O modelo é, então, ajustado para aprender os padrões e as relações nos dados de entrada que levam à saída rotulada.
- **Aprendizado Não-Supervisionado**: Essa abordagem envolve algoritmos que analisam e agrupam conjuntos de dados que não possuem rótulos. O modelo busca identificar estruturas e padrões ocultos nos dados de entrada, realizando aprendizado de características sem a necessidade de rótulos.
- **Aprendizado Semi-supervisionado**: Nesse caso, uma pequena quantidade de dados é rotulada, enquanto a maioria dos dados permanece não rotulada durante o treinamento. Essa técnica combina elementos do aprendizado supervisionado e não supervisionado para aproveitar ao máximo os dados disponíveis.
- **Aprendizado por Reforço**: Essa abordagem é fundamentada no treinamento de modelos de ML para tomar sequências de decisões. O modelo aprende por meio de experiências anteriores, interagindo com um ambiente complexo e incerto. O objetivo é maximizar a obtenção de recompensas ao longo do tempo.
- **Aprendizado em Lote**: Nesse cenário, o sistema não é capaz de aprender de forma incremental. O modelo é treinado utilizando todos os dados disponíveis de uma só vez, o que pode ser demorado. Geralmente, esse tipo de treinamento é realizado de forma offline.
- **Aprendizado Online**: Ao contrário do aprendizado em lote, o aprendizado online permite que o sistema seja treinado de forma incremental. O sistema recebe um fluxo contínuo de dados e precisa se adaptar rapidamente às mudanças ou atualizações, muitas vezes de forma autônoma.

Essas abordagens distintas dentro do campo do Aprendizado de Máquina permitem a aplicação de técnicas específicas de acordo com a natureza e os requisitos de cada problema, tornando o ML uma ferramenta versátil e poderosa em uma ampla gama de aplicações.

## 2.2 Aprendizado por Reforço

O Aprendizado por Reforço é uma área da Inteligência Artificial que desperta grande interesse, principalmente devido à sua semelhança com o processo de aprendizagem humano, baseado em tentativa e erro. Ele se destaca em relação ao Aprendizado Supervisionado e ao Aprendizado Não-Supervisionado por imitar essa abordagem de aprendizado autônomo, tornando-se um dos tópicos mais estudados no campo da Inteligência Artificial nos últimos anos, devido à sua aplicabilidade em uma ampla gama de soluções que requerem tomada de decisões automatizada [13, 14].

Nesse método de aprendizagem, um agente opera em um ambiente e, ao realizar uma ação, recebe uma recompensa. Essa ação não apenas modifica o estado do agente, mas também pode afetar o ambiente em si. Esse processo é repetido várias vezes, até que uma ação leve o agente a um estado terminal, encerrando o episódio [15, 16, 14]. Para uma descrição mais detalhada da estrutura desse processo, consulte a seção 2.3.1.

Para ilustrar esse conceito, pode-se fazer uso do exemplo clássico do jogo dos anos 90, *Super Mario World*. No jogo, o agente, Mario, interage com o ambiente, conhecido como "Mundo do Mario"(*Mario World*), executando uma série de ações com o objetivo de alcançar a faixa de chegada (*goal tape*) no final da fase. Quando Mario conclui uma fase com sucesso, obtém uma recompensa positiva. Durante sua jornada, Mario também realiza outras ações, como derrotar inimigos ou coletar moedas, que podem gerar recompensas positivas. Por outro lado, morrer ou tomar ações que não o levem ao objetivo podem resultar em recompensas negativas. Essas recompensas guiam o aprendizado de Mario ao longo do jogo.

Para tarefas de alta complexidade, como aprender com imagens, sons ou vídeos, pode-se contar com o *Deep Reinforcement Learning* (DRL). O DRL combina o Aprendizado por Reforço (RL) com a Aprendizagem Profunda (DL), resultando em avanços significativos nas técnicas de Aprendizado Profundo.

O Aprendizado por Reforço tem raízes no Processo de Decisão de Markov, conforme discutido na Seção 2.3, bem como na equação de Bellman, uma ferramenta de otimização matemática, detalhada na Seção 2.4. Essas bases teóricas fornecem o alicerce para a compreensão e aplicação do Aprendizado por Reforço em uma variedade de contextos.

## 2.3 Processo de Decisão de Markov

O Processo de Decisão de Markov (MDP) é um modelo amplamente empregado na resolução de problemas de aprendizagem em contextos estocásticos. Neste modelo, o ambiente é representado como um conjunto de estados nos quais as ações são realizadas. As ações são tomadas como uma forma de controlar o estado do sistema, e o objetivo do agente é maximizar a recompensa total, que é obtida pela soma das recompensas associadas a cada ação tomada ou pelo alcance de um objetivo final [17, 18].

O MDP é composto por quatro elementos essenciais: estados ($S$), ações ($A$), transições entre estados ($T$), e uma função de recompensa ($R$). Esses elementos são definidos da seguinte forma:

- $S$ é o conjunto de estados possíveis que o processo pode assumir.
- $A$ é o conjunto de ações que podem ser executadas, dependendo do estado atual do ambiente em cada tomada de decisão.
- $T$: $S \times A \times S \rightarrow [0,1]$ é a função probabilística que determina a probabilidade de um estado $s \in S$ fazer a transição para um estado $s' \in S$ após o agente executar uma ação $a \in A$ (denotada como $T(s'/s,a)$).
- $R$: $S \times A \rightarrow \mathbb{R}$ é a função que fornece a recompensa associada à realização de uma ação $a \in A$ quando o processo está no estado $s \in S$.

Dessa forma, o conjunto de todas as ações possíveis em um estado específico ($A_s$) para um estado $s \in S$ pode ser definido como $A = U_{s \in S} A_s$ [19, 20].

O MDP encontra aplicações em diversas áreas, incluindo agricultura, gestão de recursos hídricos, promoção de vendas, mecanismos de busca, finanças, investimentos, estudos epidemiológicos, inspeção, manutenção e reparos [21]. A seção 2.3.1 detalha a estrutura do MDP no contexto do aprendizado por reforço.

### 2.3.1 Estado, agente, recompensa e desconto

Com intuito de facilitar o entendimento da estrutura de um algoritmo de aprendizado por reforço, apresenta-se a figura 1

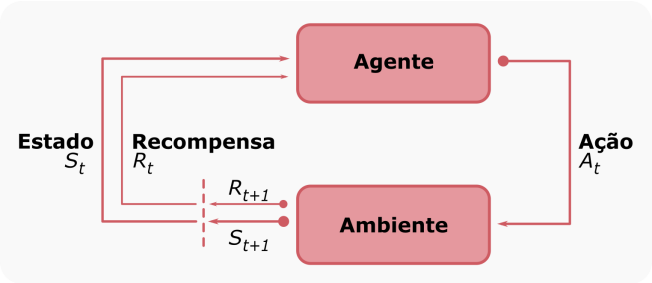


Figura 1: Aprendizado por Reforço

Fonte: Adaptado de [22]

No contexto do aprendizado por reforço, o agente é concebido como um programa de computador que se empenha em adotar decisões inteligentes em um ambiente complexo e repleto de incertezas. Este processo de tomada de decisão é conduzido por meio de um ciclo de tentativa e erro, em que o agente executa ações e, em resposta a cada ação, é recompensado ou penalizado, com o objetivo primordial de maximizar sua recompensa global [23, 14].

O ambiente é o cenário em que o agente opera e adquire experiência. Por sua vez, o estado representa a descrição da posição e da situação atual do agente, sendo considerado um reflexo da situação do agente no contexto do ambiente. A ação, por sua vez, corresponde à interação do agente com o ambiente, resultando em mudanças no estado e/ou no ambiente circundante [14].

A fim de capacitar o agente a discernir se suas ações são apropriadas, um sistema de recompensas é empregado, fornecendo *feedback* na forma de recompensas positivas ou negativas (penalidades). Sempre que o agente realiza ações desejadas ou conclui um episódio com sucesso, ele é agraciado com uma recompensa, que é registrada para fins de aprendizado. O objetivo subjacente à inteligência artificial nesse contexto é maximizar a recompensa total, representada como um valor escalar que quantifica a qualidade do desempenho do agente [15, 23, 14].

Um episódio é uma sequência ordenada de ações, estados e recompensas que culmina no estado final, com o propósito de permitir que o agente aprenda uma política que otimize a recompensa esperada após cada ação. Essa abordagem iterativa de aprendizado por reforço capacita o agente a melhorar gradualmente sua tomada de decisão e aprimorar seu desempenho no ambiente complexo [15, 23, 14].

## 2.4 A equação de Bellman

A equação de Bellman, uma formulação recursiva fundamental, desempenha um papel central na teoria da programação dinâmica. Ela tem a capacidade de calcular as recompensas associadas a ações específicas tomadas em um Processo de Decisão de Markov (MDP). Através dessa formulação, as recompensas em cada etapa da tomada de decisão são calculadas e posteriormente somadas para determinar a recompensa final [24].

A base teórica da equação de Bellman foi estabelecida por Richard Bellman em suas investigações sobre programação dinâmica na década de 1950. Inicialmente, seu foco estava na otimização e resolução de sistemas complexos de tomada de decisão [25, 26, 24]. A aplicação inicial dessa abordagem estava relacionada a problemas de controle em engenharia, mas ao longo do tempo, ela encontrou uma ampla gama de aplicações em vários setores, incluindo o campo econômico.

A equação de Bellman é empregada em problemas que exigem otimização após a modelagem inicial. A solução desses problemas envolve a aplicação de métodos iterativos, uma abordagem conhecida como programação dinâmica.

Em resumo, a resolução de um MDP implica, inicialmente, a determinação da política ideal que orienta o agente através do MDP. As ações tomadas pelo agente são definidas por essa política. O objetivo subjacente é identificar a política que maximiza a recompensa obtida em cada estado, permitindo a determinação do caminho ótimo entre todos os estados possíveis, de modo a alcançar a recompensa máxima [17]. Nesse contexto, a equação de Bellman, combinada com outros modelos, desempenha um importante papel na formulação da equação da política ideal, conforme apresentado na equação 2.1.

$$\pi(s) = \max_a \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] \tag{2.1}$$

A função $\max_a \mathbb{E}$ desempenha um papel crucial na representação da maximização da recompensa em relação a um conjunto específico de estados e ações. Ela permite determinar a ação $a$ que maximiza a expectativa da recompensa, representada pela expressão $\mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a]$. Vamos analisar os componentes desta expressão em detalhes.

- $R_{t+1}$ representa a recompensa imediata obtida ao realizar a ação $a$ no estado $s$ no momento $t$.

- $\gamma$ é o fator de desconto, que introduz uma consideração temporal na otimização. Este parâmetro pondera a importância das recompensas futuras mais distantes em relação à recompensa futura imediata. Um $\gamma$ próximo de 1 valoriza recompensas futuras mais distantes, enquanto um valor próximo de 0 prioriza recompensas imediatas.

- $V_\pi(S_{t+1})$ representa o valor do estado $S_{t+1}$ sob a política $\pi$. Em outras palavras, é a expectativa da recompensa acumulada futura a partir do próximo estado, dado que seguimos a política $\pi$.

A equação 2.1, uma extensão da equação de Bellman, é comumente empregada para denotar a qualidade, representada como $q$ ou $Q$ [17]. Essa equação descreve de maneira abrangente como as recompensas futuras são avaliadas e ponderadas, considerando a política atual $\pi$ e o fator de desconto $\gamma$, a fim de determinar a qualidade das ações em um dado estado. Essa equação é aplicada no contexto do MDP e é frequentemente utilizada em algoritmos de aprendizado por reforço para aprimorar a tomada de decisão do agente.

## 2.5 Política

A política, no contexto do aprendizado por reforço, determina como o agente deve agir em seu ambiente. Em termos simples, a política mapeia cada estado possível, denotado como $s_t$, as ações correspondentes, representadas como $a_t$, que o agente deve executar. Para avaliar a qualidade da política em relação a um estado específico, utiliza-se uma função de valor, denotada como $V_\pi$, que é atualizada iterativamente para os estados subsequentes, como demonstrado na Equação 2.2 [20, 27].

$$V_\pi(s_t) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...] \tag{2.2}$$

Assim, empregando a equação de Bellman, apresentada na Equação 2.1, é possível derivar uma função de valor para um estado, conforme mostrado na Equação 2.3 [28, 27].

$$V_\pi(s_t) = \mathbb{E}_\pi[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s] \tag{2.3}$$

A expectativa ($\mathbb{E}$) nessa formulação implica um processo iterativo que atualiza o valor do estado ($V$) com base na política atual ($\pi$) e nas transições de valor de estado ($P$) do modelo. Isso cria uma sequência de atualizações de valor representada por $V_0..V_k$. Além disso, é importante notar que a expectativa leva em consideração a recompensa imediata,

somada ao valor do próximo estado multiplicado pelo fator de desconto ($\gamma$), levando em consideração a ação escolhida pelo agente [16].

A função de atualização da política, representada na Equação 2.4, é aplicável quando tanto as transições de estado ($P$) quanto as recompensas ($R$) são conhecidas para todos os estados ($s$) e ações ($a$). Em outras palavras, essa abordagem se aplica apenas quando o modelo do ambiente é completamente conhecido. A primeira soma na equação reflete a probabilidade ponderada de cada ação tomada pelo agente, considerando uma política estocástica, embora, por simplicidade, este trabalho considera como políticas determinísticas [20, 27]. Essa sequência de atualizações converge à medida que $k$ se aproxima do infinito ($k \to \infty$).

$$V(s_t) = \mathbb{E}_\pi[R_{t+1} + \gamma V_k(S_{t+1})|S_t = s] = \sum_a \pi(s_t, a_t) \sum_{s_{t+1}, r_t} P(s_{t+1}|s_t, a_t)[R_t + \gamma V_k(s_{t+1})] \tag{2.4}$$

O próximo passo é o Aprimoramento de Política, cujo objetivo é encontrar a melhor política ($\pi^*$), como indicado na Equação 2.5 [29, 17].

$$\pi^*(s_t) = \underset{a_t}{max} Q_\pi(s_t, a_t) = \underset{a_t}{max} \sum_{s_{t+1}, r_t} P(s_{t+1}|s_t, a_t)[r_t + \gamma V_\pi(s_{t+1})] \tag{2.5}$$

A nova política ($\pi^*$) é derivada selecionando a ação que resulta no maior retorno, com base na política atual ($\pi$), e é uma representação da qualidade ($Q_\pi$). A função $Q_\pi$ expressa a recompensa futura esperada dado uma ação ($a$) tomada em um estado ($s$). Como demonstrado, a política $\pi^*$, será superior à política $\pi$ se e somente se a função de valor ($V$) for ótima [29, 17].

A combinação de avaliação de política e melhoria de política resulta em duas abordagens para calcular a política ideal: iteração de política e iteração de valor. Ambas as abordagens envolvem avaliação de política para aprimorar a função de valor e melhorar a política, realizando essas etapas iterativamente [22]. No entanto, a iteração de política executa essas duas fases ciclicamente, enquanto a iteração de valor combina as duas em uma única atualização, conforme abordado nas subseções 2.5.1 e 2.5.2.

### 2.5.1 Iteração de Política

A iteração de política, apresentada no algoritmo 1, é uma abordagem que se baseia na atualização da função de valor $V_\pi$ com base na política atual $\pi$, fazendo uso tanto da avaliação de política (conforme indicado na Equação 2.4) quanto do aprimoramento de política (conforme apresentado na Equação 2.5). Após um número definido de ciclos, o algoritmo converge para a política ótima $\pi^*$.

No início da iteração, os valores de $V_\pi$ e $\pi$ são inicializados para o estado $s$. O processo de iteração continua enquanto a política $\pi$ não atinge o estado ótimo. O procedimento compreende a avaliação de política, que visa calcular a função de valor da política a partir da avaliação realizada na iteração anterior, e o aprimoramento de política, que busca otimizar a política atual, selecionando a ação que proporciona a maior recompensa futura [30, 31, 22].

**Algoritmo 1:** Iteração de política

```
1  início
2      Inicializa V_π(s) e π(s) para todo estado s
3      while π ótimo (estável) do
4          Início Avaliação de Política
5          while V_π não for estável do
6              for cada estado s do
7                  V_π(s_t) = Σ_{s_{t+1},r_t} P(s_{t+1}, r_t|s_t, π(s_t))[r_t + γV_π(s_{t+1})]
8              end
9          end
10         Fim Avaliação de Política.
11         Início Aprimoração de Política.
12         for cada estado s do
13             π = max_{a_t} Σ_{s_{t+1},r_t} P(s_{t+1}, r_t|s_t, a_t)[r_t + γV_π(s_{t+1})]
14         end
15         Fim Aprimoração de Política.
16     end
17 fim
```

### 2.5.2 Iteração de Valor

Um outro algoritmo de programação dinâmica utilizado para otimizar um MDP é a iteração de valor que atualiza a função valor para o estado atual dada a ação que máxima o retorno futuro. Diferentemente da iteração de política que realiza a avaliação e iteração simultaneamente na repetição. A iteração de valor é dada pela Equação (2.6)

$$V_\pi(s_t) = \max_{a_t} \sum_{s_{t+1}, r_t} P(s_{t+1}, r_t | s_t, a_t)[r_t + \gamma V_\pi(s_{t+1})] \tag{2.6}$$

Posteriormente, a política ideal é calculada com base na equação (2.7). A diferença está na atualização da estimativa de valor e na ausência de a iteração de política no laço de repetição principal [31].

$$\pi(s_t) = \max_{a_t} \sum_{s_{t+1}, r_t} P(s_{t+1}, r_t | s_t, a_t)[r_t + \gamma V_\pi(s_{t+1})] \tag{2.7}$$

O algoritmo de iteração de valor é notavelmente mais simples e pode ser resumido conforme apresentado no Algoritmo 2.

**Algoritmo 2:** Iteração de Valor

```
1  início
2      Inicializa V_π(s) para todo estado s
3      while π* (ótimo e estável) do
4          Início Iteração de Valor.
5          for cada estado s do
6              V_π(s_t) = max_{a_t} Σ_{s_{t+1},r_t} P(s_{t+1}|s_t, a_t)[r_t + γV_π(s_{t+1})]
7          end
8          Fim Iteração de Valor.
9      end
10     Início Cálculo da política ideal
11     for cada estado s do
12         π(s_t) = max_{a_t} Σ_{s_{t+1},r_t} P(s_{t+1}, r_t|s_t, a_t)[r_t + γV_π(s_{t+1})]
13     end
14     Fim Cálculo da política ideal
15 fim
```

## 2.6 Aprendizado por diferenças temporais

O aprendizado por diferenças temporais se refere a algoritmos livre de modelo que aprendem incrementando a função valor. Esse processo começa com um estado desconhecido e ajusta as previsões visando aumentar a precisão da ação futura [32]. Em essência, concentra-se na análise das diferenças entre os aprendizados ao longo da experiência.

A Equação 2.8 representa uma forma simplificada do método TD-learning (*Temporal Difference Learning*) denominado TD(0). Nessa, a função de valor $V$ é atualizada com base na política $\pi$, levando em consideração a recompensa $r$ obtida pelo agente ao fazer a transição do estado $s$ para o estado seguinte $s'$ [16].

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.8}$$

A expressão $[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ representa o erro associado à diferença temporal e é empregada em vários métodos de aprendizado por reforço. Neste contexto, o erro é calculado como a diferença entre as estimativas $V_t$ e $V_{t+1}$ e a recompensa obtida na transição do estado $s_t$ para o estado $s_{t+1}$. Portanto, essa equação permite a atualização lenta da função de valor $V_t$, sendo controlada pela taxa de aprendizado $\alpha$. Essa abordagem é valiosa para aprender e aprimorar o desempenho do agente em ambientes desconhecidos.

## 2.7 Model-based and model-free learning

Um sistema de aprendizagem baseada em modelo (*model-based*) se apoia em modelos que buscam identificar as ações a serem tomadas através da realização de inferências sobre o ambiente. Esses modelos permitem que o sistema tome decisões considerando as possíveis situações futuras, em contraste com os métodos livres de modelo (*model-free*), que não dependem de um modelo explícito do ambiente e aprendem por meio de tentativa e erro [33].

O aprendizado livre de modelo (*model-free*) apresenta uma vantagem significativa em relação ao aprendizado baseado em modelo quando o ambiente se torna excessivamente complexo para ser representado com precisão por um modelo. Nessas situações, quando ocorrem mudanças no ambiente que afetam a forma como o agente interage com ele, o agente pode adquirir novas experiências no ambiente e, com base nessa experiência, atualizar tanto a política quanto a função de valor para se adaptar à nova realidade

[34, 33]. Essa capacidade de adaptação torna os métodos livres de modelo mais flexíveis e adequados para ambientes dinâmicos e complexos.

## 2.8 On-policy learning v.s. off-policy learning

Os algoritmos *on-policy* são caracterizados pelo uso de uma política para otimizar a função de valor, com o objetivo de melhorar a qualidade das ações tomadas. Em outras palavras, esses algoritmos se concentram na avaliação e melhoria da política atualmente em uso. No início do treinamento, quando a política ainda não está otimizada, é comum usar uma política estocástica para permitir alguma forma de exploração [28].

Por outro lado, os algoritmos *off-policy* seguem uma abordagem diferente, pois otimizam a política usada para selecionar as ações a serem tomadas com base em uma política alvo que é independente das ações anteriores do agente. Essa independência permite que esses algoritmos encontrem a política ótima independentemente da trajetória ou motivação do agente [15, 28]. Essa distinção entre abordagens *on-policy* e *off-policy* tem implicações importantes no design e no desempenho de algoritmos de aprendizado por reforço.

## 2.9 Q-Learning

O Q-Learning é um algoritmo de aprendizado por reforço livre de modelo (*model-free*) que atualiza a função de valor com base em uma equação, geralmente a equação de Bellman (2.1). Este algoritmo, por ser *off-policy*, pode estimar, por meio de ações aleatórias, qual ação resultará na maior recompensa para cada estado [35, 29].

Embora seja eficaz em domínios discretos, com estados e ações limitados, o Q-Learning apresenta limitações em ambientes complexos. Isso ocorre porque o algoritmo produz uma tabela de valores $Q$ que contém a ação a ser tomada para cada estado. Em estados altamente complexos, isso levaria a um número quase infinito de possibilidades ($10^{120}$), tornando-o inviável para resolver problemas mais complexos, como o xadrez, por meio de tabelas Q [36].

No entanto, avanços significativos foram alcançados em ambientes mais complexos por meio de algoritmos de aprendizado por reforço, como o AlphaZero, que atingiu o estado da arte ao usar uma combinação de métodos de aprendizado por reforço e busca em árvore para jogar xadrez [37]. Além disso, o uso de redes neurais com funções de ativação de

base radial em algoritmos Q-Learning também demonstrou resultados promissores, como no caso do xadrez chinês [38].

A função $Q$ de uma política $\pi$ mede o retorno futuro esperado, somando as recompensas obtidas com base nas ações tomadas. Sua forma ideal é representada pela Equação (2.9) [39, 22].

$$Q^*(s_t, a_t) = \mathbb{E}[R_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})] \tag{2.9}$$

Os valores da função $Q$ são inicializados como zero e a função é atualizada usando a equação de diferenças temporais da Equação (2.10).

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.10}$$

Nesta equação, a recompensa futura para um estado $s_t$ dado uma ação $a_t$ é calculada como a recompensa futura $R_{t+1}$ somada ao valor $Q_{t+1}$ máximo entre as possíveis ações $a_t$, descontado pelo fator de desconto $\gamma$, e subtraída do valor $Q$ atual. O processo de atualização ocorre com uma taxa de aprendizado $\alpha$ [15, 23, 38].

Essa abordagem realiza uma iteração sob $Q$, atualizando-a iterativamente de forma que $Q_t$ convirja para $Q^*$ à medida que $t$ aumenta infinitamente. Basicamente $Q_{t+1} \leftarrow \mathbb{E}[R_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1})]$ convergirá quando $Q_t \rightarrow Q^*$ com $t \rightarrow \infty$. Essencialmente, a função $Q$ reflete a qualidade de uma ação dado um estado, representando a eficácia dessa ação na maximização da recompensa futura, mesmo em abordagens exploratórias [15, 23, 40]. Desta forma, o valor de $Q$ depende dos valores futuros, da forma:

$$Q(s_t, a_t) = \gamma Q(s_{t+1}, a_{t+1}) + \gamma^2 Q(s_{t+2}, a_{t+2}) + ... + \gamma^n Q(s_{t+n}, a_{t+n}) \tag{2.11}$$

A implementação do algoritmo Q-Learning pode ser resumida conforme apresentado no Algoritmo 3.

**Algoritmo 3:** *Q-Learning*

1 **início**
2 Inicializa $\boldsymbol{Q(s,a)}$ aleatoriamente
3 **while** *s não for o estado terminal* **do**
4 Escolhe uma ação, $\pi(a)$, para um estado, $s$, usando uma política derivada de $Q$.
5 Toma uma ação, $a$, e recebe uma recompensa, $R_{t+1}$, e o novo estado $Q$
6 $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
7 **end**
8 **fim**

Este algoritmo resulta em uma função Q para cada par de estado e ação, em que cada posição indica a ação que maximiza a recompensa futura para o estado correspondente [41]. No entanto, para evitar problemas de convergência em mínimos locais, em alguns casos, é introduzido o algoritmo *$\epsilon$-greedy*, que visa promover a exploração (consulte a subseção 2.9.2).

### 2.9.1 Dilema *Exploration* e *Exploitation*

Durante o treinamento por reforço, o agente pode se encontrar em um dilema em que ele explora as recompensas disponíveis sem nunca atingir a melhor recompensa possível. Para abordar esse problema, é comum adotar um equilíbrio entre dois fatores conhecidos em inglês como *exploration* e *exploitation* [42, 43]. Por ambas as palavras poderem ser traduzidas ao português por exploração, utiliza-se os nomes em inglês para explicar os conceitos.

Quando um agente utiliza sua política para escolher a ação que maximiza a recompensa em um estado dado, diz-se que ele está realizando um *exploit*, pois está escolhendo a melhor ação disponível [44, 43, 45].

Por outro lado, a abordagem *explore* envolve a escolha de uma ação de forma aleatória, sem considerar a recompensa futura dessa ação. A importância dessa abordagem reside no fato de que o agente pode continuar a descobrir novos estados que ainda não foram explorados. A recompensa obtida por meio da ação tomada também é considerada durante a melhoria da política ou o preenchimento da tabela $Q$ [42, 43].

Dessa forma, o treinamento começa com escolhas aleatórias e, gradualmente, evolui para a adoção de uma política que maximize a recompensa. A cada iteração, a proba-

bilidade de o agente escolher uma ação aleatória diminui, e ele tem mais probabilidade de selecionar a ação que maximiza a recompensa, culminando com a escolha da ação de maior retorno conforme o treinamento decorre. Isso é alcançado por meio de uma estratégia conhecida como $\epsilon$-*greedy* [46].

### 2.9.2 A estratégia $\epsilon$-*greedy*

A estratégia $\epsilon$-*greedy* apresentada no Algoritmo 4 é uma abordagem que equilibra *exploitation* e *exploration*. Essa estratégia envolve a definição do valor da variável $\epsilon$, conhecida como a taxa de exploração, geralmente configurada para 0,1. Isso significa que o agente tem 10% de probabilidade de escolher uma ação aleatória no ambiente, realizando uma *explore*, em vez de optar pela ação que maximiza a recompensa, realizando um *exploit* [46, 47, 41].

Dessa forma, à medida que o agente realiza um *explore*, ele aprende mais sobre o ambiente e, a cada episódio, a taxa $\epsilon$ é reduzida gradualmente, seguindo uma taxa de decaimento. Isso torna cada vez mais provável que o agente escolha a ação de maior recompensa, realizando um *exploit*. Essa estratégia é chamada de *greedy* (gananciosa) em inglês, pois à medida que o agente aprende, ele tende a realizar cada vez mais a *exploit* em vez de apenas a *explore*, visando maximizar as recompensas obtidas por suas ações [48, 42, 45, 47].

**Algoritmo 4:** Algoritmo $\epsilon$-*greedy* para escolha da ação a ser tomada

```
1 início
2     n ← Valor aleatório entre 0 e 1
3     if n < ε then
4         a ← Ação aleatória do espaço das ações A
5     end
6     else
7         a ← max_a Q^π(s, a)
8     end
9 fim
```

## 2.10 Deep Q-Learning

Como apresentado na seção anterior (seção 2.9), o algoritmo Q-learning é limitado a domínios discretos com espaços de ação e estados finitos. No entanto, em ambientes mais complexos, a implementação básica da função Q torna-se inviável, pois não é possível criar tabelas de estado-ação de tamanho suficiente para representar todas as possibilidades. Portanto, foi proposta uma nova abordagem que utiliza uma função de aproximação capaz de lidar com essa complexidade [49].

A solução para o problema de lidar com ambientes que não possuem estados ou espaços de ação bem definidos envolve o uso de uma função que, dado um estado, seja capaz de produzir a ação ideal a ser tomada, eliminando a necessidade de armazenar uma tabela completa de estado-ação [22].

Uma escolha lógica para essa função de aproximação são as redes neurais, que são poderosas em tarefas de aproximação de funções. No entanto, incorporar uma rede neural ao algoritmo Q-learning não é tão simples quanto parece. É necessário fazer ajustes significativos para adaptar o algoritmo original, que utiliza tabelas, para funcionar com uma rede neural [22].

Portanto, o *deep q-learning* não se resume apenas a adicionar uma rede neural ao algoritmo Q-learning. Envolve uma série de ajustes e algoritmos adicionais para que a abordagem funcione corretamente. Alguns desses elementos-chave incluem o uso de experiencia de repetição (*experience replay*) e o aprimoramento do algoritmo para lidar com redes neurais [50].

### 2.10.1 Experiência de Repetição

Conforme apresentado na seção (seção 2.12), uma rede neural é uma função complexa não linear cujo treinamento ocorre por meio de aprendizado supervisionado. Nesse processo, os pesos da rede são ajustados com base nas diferenças entre os valores alvo e os valores previstos, com o objetivo de reduzir o erro por meio da técnica de retropropagação [39].

No contexto do deep Q-learning, uma das técnicas essenciais é a experiência de repetição (*experience replay*). Essa técnica envolve o armazenamento da "experiência"do agente no ambiente em uma memória de transição, geralmente implementada como uma tabela conhecida como (*replay buffer*). Essa tabela armazena tuplas compostas por infor-

mações como estado atual, ação tomada, recompensa obtida e próximo estado resultante $((s_t, a_t, R_t, s_{t+1}))$. Para reduzir os efeitos negativos da correlação entre as experiências durante o treinamento, amostras desses valores são selecionadas aleatoriamente da replay buffer para atualizar a rede neural Q, também conhecida como *Q-network* [51, 52, 40].

## 2.11 Double Deep Q-Learning

O algoritmo de *Deep Q-learning* possui uma tendência a superestimar os valores de Q para as combinações estado-ação, o que pode prejudicar o desempenho da política aprendida em certas condições. Isso ocorre porque a probabilidade de escolher uma nova ação é reduzida em relação a outras ações, o que pode impedir uma otimização eficaz da política [51, 50].

Uma solução proposta para esse problema é o algoritmo conhecido como *Double Deep Q-learning*, que utiliza dois estimadores distintos para o valor de Q e aprende a partir de uma memória de repetição (*replay buffer*) [52, 53]. Esses estimadores são atualizados de forma conjunta, conforme apresentado no Algoritmo 5.

**Algoritmo 5:** Baseado em Deep Reinforcement Learning with Double Q-learning [50]

1 **início**
2 Inicializa as redes neurais $\sigma_{online}$, $\sigma_{alvo}$, o *replay buffer* **for** *cada iteração* **do**
3 **for** *cada passo no ambiente* **do**
4 Observa o estado $s_t$ e seleciona a ação $a_t$ com base na política $\pi(s_t, a_t)$
5 Realiza a ação $a_t$ e observa o próximo estado $s_{t+1}$ e a recompensa $r_t = R(s_t, a_t)$.
6 Armazena $(s_t, a_t, r_t, s_{t+1})$ no replay buffer
7 **end**
8 **for** *cada atualização de passo* **do**
9 utiliza a amostra $e_t = (s_t, a_t, r_t, s_{t+1}) \sim$ replay buffer
10 Calcula o valor $Q_{online}$ ideal:
11 $Q^*_{online}(s_t, a_t) \approx r_{t+1} + \gamma Q_{alvo}(s_{t+1}, \underset{a_{t+1}}{max} Q_{online}(s_{t+1}, a_{t+1}))$
12 Treina a rede online $\sigma_{online}$ com base no estado $s_t$ e nos valores do Q ideal, $Q^*_{online}(s_t, a_t)$
13 Atualiza a rede alvo $\sigma_{alvo}$ com os pesos da rede online $\sigma_{online}$
14 $\sigma_{alvo} \leftarrow \sigma_{online}$
15 **end**
16 **end**
17 **fim**

Dessa forma, o algoritmo utiliza dois modelos independentes, denotados como $\sigma_{online}$ e $\sigma_{alvo}$. O modelo $\sigma_{online}$ é empregada para a seleção da ação a ser executada, enquanto a rede $\sigma_{alvo}$ é utilizado para avaliação da política, como ilustrado na Figura 2.

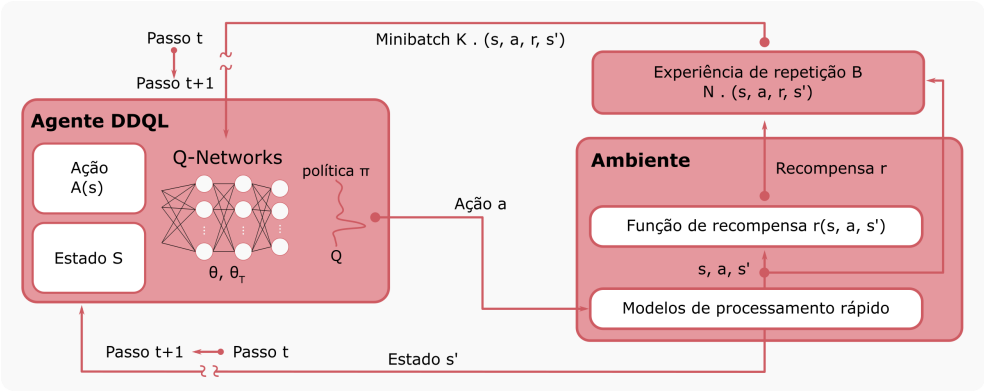


Figura 2: Double Deep Q-Learning

Fonte: Elaborada pelo autor com base em [54]

### 2.11.1 Redes *Online* e *Alvo*

Para solucionar essa questão, foi introduzida a ideia de uma rede alvo (*target network*). Assim, durante o treinamento, a rede *online* é utilizada para otimizar o valor de $Q(s_t, a_t)$, enquanto a rede alvo, que é uma cópia da rede *online* no tempo $t$, é empregada para estimar o valor de $Q(s_{t+1}, a_{t+1})$ uma vez que representa a rede no estado passado. Vale ressaltar que a rede alvo não passa por treinamento, mas seus parâmetros são sincronizados periodicamente com os parâmetros da *Q-network* principal. Essa abordagem tem como objetivo aumentar a estabilidade do treinamento, minimizando as instabilidades decorrentes das atualizações simultâneas dos valores de $Q(s_t, a_t)$ e $Q(s_{t+1}, a_{t+1})$ na mesma rede [55].

## 2.12 Redes Neurais Artificiais

As redes neurais artificiais consistem em uma organização de neurônios dispostos em camadas. Tipicamente, uma rede neural possui uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Os neurônios, também denominados nós, dentro dessa rede estão interconectados entre camadas, como ilustrado na figura 3. A estrutura de entrada de cada neurônio inclui conexões com os nós da camada anterior, juntamente com um parâmetro chamado de viés.

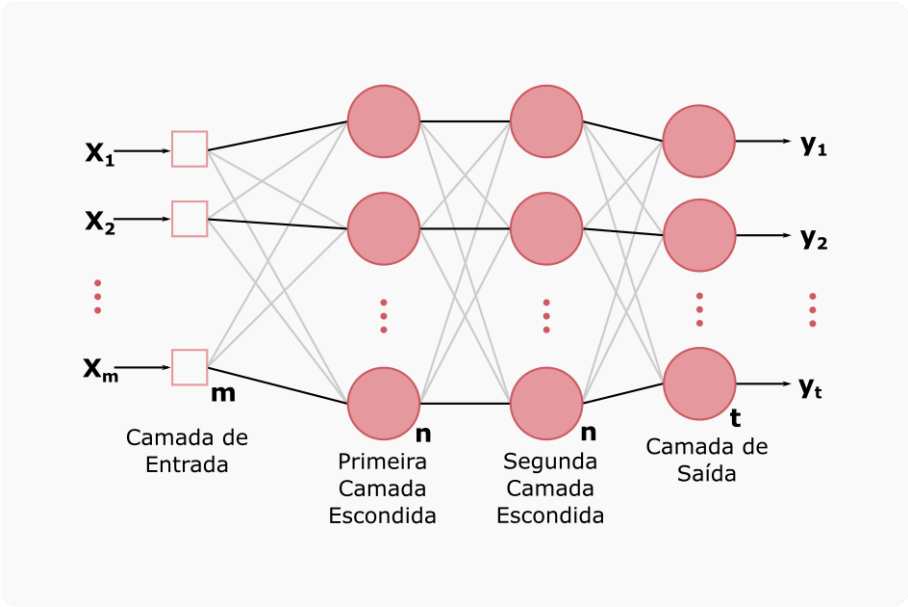


Figura 3: Rede Neural

Fonte: Elaborada pelo autor com base em [56]

Além disso, após o cálculo do somatório das ponderações multiplicadas pelos valores de entrada, uma função de ativação é aplicada. Essa função de ativação é ativada quando o valor do somatório supera um certo limiar. Quando essa condição é satisfeita, os valores são propagados para os nós da próxima camada da rede; caso contrário, os dados não são transmitidos para a camada subsequente. Esse processo ocorre em todas as camadas da rede, da entrada à saída [56].

Na camada final da rede neural, ocorre a geração de uma saída que pode representar, por exemplo, uma probabilidade ou uma classe, dependendo da função de ativação utilizada. O treinamento da rede envolve a correção dos valores dos pesos por meio do algoritmo de retropropagação do erro (*backpropagation*). O objetivo desse processo é minimizar uma função de custo, e isso é alcançado ajustando os pesos da rede com base no gradiente da função de custo em relação aos pesos [56].

Em resumo, as redes neurais artificiais realizam operações de propagação direta (*feedforward*) para obter previsões ou classificações e, em seguida, usam o algoritmo de retropropagação para ajustar os pesos e minimizar o erro. Esse ciclo de treinamento é necessário para o aprendizado e a capacidade da rede neural em realizar tarefas complexas, como reconhecimento de padrões, previsões e muito mais.

### 2.12.1 Redes *Feedforward*

As redes neurais artificiais, especificamente as *feedforward*, possuem uma arquitetura em que a informação flui de uma camada para outra sem formar ciclos, sendo amplamente empregadas em diversas aplicações, como reconhecimento de padrões, classificação, clusterização, otimização, controle e previsões [57, 58, 59].

Uma limitação inicial das redes *feedforward* foi evidenciada por Minsky e Papert, que demonstraram que um perceptron de uma única camada não conseguia resolver problemas não lineares [60]. No entanto, a adição de camadas ocultas nas redes neurais, criando o modelo de perceptron multi-camadas, possibilitou a resolução de problemas não linearmente separáveis [61].

A introdução de camadas ocultas trouxe novos desafios, como o tempo de treinamento demorado. Naquela época, o poder computacional disponível não era suficiente para lidar com a ineficiencia da abordagem disponível durante o treinamento de redes neurais com várias camadas. A situação começou a melhorar com o desenvolvimento do algoritmo de *backpropagation*, proposto por Rumelhart, que se mostrou muito mais eficiente do que os métodos anteriores para ajustar os pesos durante o treinamento [61, 62]. Esses avanços na eficiência do treinamento contribuíram significativamente para a viabilidade e a popularização das redes neurais multi-camadas em diversas aplicações.

### 2.12.2 Backpropagation

Por muitos anos, os pesquisadores enfrentaram desafios ao treinar redes neurais multi-camadas devido à ineficiência das técnicas disponíveis para ajustar os pesos. O algoritmo de *backpropagation* resolveu esse problema ao utilizar uma técnica que calcula o gradiente descendente de forma eficiente e automática, otimizando o processo de treinamento e reduzindo significativamente o tempo necessário.

O *backpropagation* opera em duas etapas: a *forward pass* e a *backward pass*. Na primeira etapa, ocorre o somatório do produto entre os pesos e os valores de entrada em cada nó da rede, somado a um viés. Posteriormente, esse somatório passa por uma função de ativação que, com base em um limiar, decide se o nó será ativado ou não. Esse valor ativado é então transmitido como entrada para os próximos nós da rede, e esse processo é repetido até a camada de saída, que fornece o resultado da rede.

Na segunda etapa, esse resultado é comparado com os rótulos dos dados e calcula-se

o erro. O algoritmo de *backpropagation* utiliza o gradiente do erro em relação a cada parâmetro, ou seja, ele identifica como cada peso e viés devem ser ajustados para reduzir o erro. Esse processo é repetido até que o erro esteja dentro de um limite aceitável [63, 64].

Em resumo, o *backpropagation* emprega a técnica do gradiente descendente, que envolve mover-se no sentido contrário ao gradiente da função no ponto para encontrar um mínimo local. Isso visa minimizar a função de custo em relação ao conjunto de treinamento, calculando a derivada parcial do custo em relação aos pesos $\frac{\partial C}{\partial \omega}$. A função de custo depende dos pesos e vieses de todos os neurônios em todas as camadas, e a derivada parcial indica como a função de custo muda à medida que os pesos e vieses são ajustados [62, 63]. Esse avanço na técnica de treinamento permitiu o uso eficaz de redes neurais multicamadas em diversas aplicações.

## 2.13 Funções de ativação

Em redes neurais as funções de ativação determinam se um neurônio será ativado ou não. Essas funções podem ser classificadas em dois tipos: lineares e não-lineares [65].

As funções de ativação não-lineares recebem os dados de entrada e realizam uma transformação nesses dados, introduzindo não-linearidade na rede. Essa não-linearidade é essencial para que a rede neural seja capaz de resolver problemas mais complexos que não podem ser separados linearmente. É importante ressaltar que uma rede neural sem funções de ativação seria essencialmente equivalente a um modelo de regressão linear [66, 65].

No contexto deste trabalho, uma das funções de ativação mais proeminentes é a função ReLU, que foi discutida na subseção 2.13.1. A função ReLU é amplamente utilizada devido à sua eficácia em lidar com problemas de não linearidade em redes neurais. Ela ajuda a rede a aprender representações mais complexas dos dados, contribuindo para o desempenho geral da rede neural.

### 2.13.1 Função Relu e suas variantes

A função de ativação mais amplamente utilizada em redes neurais é a ReLU, que significa "Unidade Linear Retificada"(*Rectified Linear Unit*, em inglês). Ela é uma função não-linear definida pela equação 2.12.

$$f(x) = \max(0, x) \tag{2.12}$$

Uma das principais vantagens da ReLU em relação a outras funções de ativação é que ela não ativa todos os neurônios ao mesmo tempo. Se a entrada de um neurônio for negativa, a saída será zero, desativando efetivamente o neurônio. Isso torna a rede esparsa e eficiente. No entanto, a ReLU pode apresentar um problema conhecido como "dissipação do gradiente,"no qual os gradientes podem se tornar zero durante o treinamento, afetando a convergência da rede [53, 66].

Para superar o problema de neurônios que podem "morrer"devido à ativação zero em valores negativos, foi proposta a função *Leaky* ReLU. Nessa função, quando x $<$ 0, o gradiente não é zero, mas sim um valor negativo, que é uma fração de x. Isso permite que informações fluam mesmo para entradas negativas e ajuda a mitigar os problemas relacionados à dissipação do gradiente [67]. A função *Leaky* ReLU é definida como na equação 2.13

$$\begin{cases} \mathrm{f(x) = ax} & \mathrm{se\ x < 0,} \\ \mathrm{f(x) = x} & \mathrm{se\ x \geq 0,} \end{cases} \tag{2.13}$$

Outra função de ativação é a ELU, que significa "Unidade Linear Exponencial". A ELU é definida como na equação 2.14.

$$\begin{cases} \alpha(e^x - 1) & \mathrm{se\ x \leq 0,} \\ \mathrm{f(x) = x} & \mathrm{se\ x > 0,} \end{cases} \tag{2.14}$$

A ELU é eficaz em relação à ReLU, pois também evita o problema de neurônios "mortos"com gradientes zero em entradas negativas. No entanto, devido à necessidade de calcular a função exponencial, a ELU pode ser mais lenta durante o treinamento, tornando-o demorado. Portanto, muitos sugerem treinar inicialmente com ReLU e, após ajustar os hiperparâmetros, mudar a função de ativação das camadas internas para ELU para obter os benefícios de ambas as funções [68].

## 2.14 Otimizadores

Os otimizadores são algoritmos responsáveis por acelerar a otimização dos valores dos parâmetros durante as etapas de treinamento das redes neurais. Um dos otimizadores amplamente conhecidos é o algoritmo de *backpropagation*. Esse algoritmo faz uso da técnica que calcula o gradiente descendente de forma eficiente e automática, o que otimiza o processo e reduz o tempo gasto durante o treinamento.

Além do *backpropagation*, outros otimizadores são amplamente utilizados em redes neurais, incluindo o Gradiente Descendente e a Taxa de Aprendizagem, que foram apresentados nas seções 2.14.1 e 2.14.2, respectivamente. Cada otimizador tem suas características específicas e pode ser mais adequado para diferentes tipos de problemas ou arquiteturas de rede. A escolha do otimizador adequado desempenha um papel fundamental no sucesso do treinamento de uma rede neural.

### 2.14.1 Gradiente Descendente

Um dos otimizadores mais comuns é o gradiente descendente, que é amplamente utilizado para minimizar a função de custo iterativamente em direção ao mínimo local. Essa técnica é fundamental em machine learning, pois permite a atualização dos parâmetros dos modelos de forma eficaz [69].

Em termos matemáticos, o objetivo do gradiente descendente é encontrar o mínimo local de uma função objetivo $f(\sigma)$, $\sigma \in \theta$. O objetivo é encontrar o minimizador $\sigma^* = \underset{\sigma \in \theta}{min} f(\theta)$. O gradiente descendente é utilizado como método padrão para alcançar esse objetivo, conforme mostrado na sequência apresentada na equação 2.15 [69].

$$\sigma_{t+1} = \sigma_t - \alpha_t \bigtriangledown f(\sigma_t) \tag{2.15}$$

Nessa equação, $f(\sigma_t)$ representa os parâmetros do modelo na iteração t, $\alpha$ é a taxa de aprendizado que controla o tamanho dos passos de atualização, e $\bigtriangledown f(\sigma_t)$ é o gradiente da função objetivo em relação aos parâmetros $\sigma_t$. O gradiente aponta na direção de maior crescimento da função, e o sinal negativo é usado para garantir que a solução está indo no sentido da descida, em direção ao mínimo local.

Portanto, o gradiente descendente é uma técnica poderosa para otimizar os parâmetros de modelos de machine learning e é a base de muitos outros algoritmos de otimização utilizados na área.

### 2.14.2 Taxa de Aprendizagem e Momento

A taxa de aprendizado ($\eta$) é um parâmetro crítico no treinamento de redes neurais, pois influencia a velocidade e a estabilidade do aprendizado. Taxas de aprendizado elevadas significam que os pesos da rede são atualizados em incrementos maiores a cada iteração, o que pode levar a oscilações e instabilidades durante o treinamento. Por outro lado, taxas de aprendizado muito baixas tornam o aprendizado mais lento e podem fazer com que o treinamento fique preso em mínimos locais [56, 70].

A alteração dos pesos da rede ocorre através da retro-propagação do erro, e a magnitude dessa alteração é proporcional ao inverso do gradiente do erro. Para melhorar a estabilidade e a eficácia do treinamento, o parâmetro de momento é frequentemente introduzido no modelo. O momento, nesse contexto, serve para aumentar a taxa de aprendizado à medida que o modelo aprende. Isso ajuda a evitar oscilações e mínimos locais, contribuindo para um treinamento mais rápido e eficiente da rede neural [56, 70].

Portanto, a combinação adequada da taxa de aprendizado e do parâmetro de momento auxilia na otimização do processo de treinamento de redes neurais, permitindo que o modelo aprenda de forma estável e eficaz.

## 2.15 Sobre-ajuste e Sub-ajuste

O *overfitting*, também conhecido como sobre-ajuste, é um fenômeno que ocorre quando um modelo se ajusta em excesso ao conjunto de treinamento. Nesse caso, o modelo treinado apresenta um desempenho excelente durante o treinamento, mas falha em obter bons resultados quando aplicado a um conjunto de teste ou validação. O *overfitting* é caracterizado por alta variança e baixo viés. Para lidar com esse problema, existem várias abordagens, tais como a utilização de técnicas de regularização, aumento da base de dados de treinamento, implementação de critérios de parada antecipada ou a redução da complexidade do modelo [71].

Por outro lado, o *underfitting*, ou subajuste, ocorre quando o modelo falha em aprender adequadamente com o conjunto de treinamento, não conseguindo capturar as características importantes presentes nos dados e não sendo capaz de representar a variabilidade dos dados. Nessa situação, o viés do modelo é geralmente alto, e a variança é baixa. Para resolver o *underfitting*, pode-se aumentar a complexidade do modelo. No contexto de redes neurais, isso pode ser feito aumentando o número de camadas ocultas e/ou o

número de nós nessas camadas [56, 71].

Portanto, o *overfitting* e o *underfitting* são desafios comuns no treinamento de modelos de machine learning, e encontrar o equilíbrio certo é essencial para obter um modelo que generalize bem para novos dados.

## 2.16 Regularização

A regularização é uma técnica amplamente utilizada para combater o *overfitting* em tarefas de aprendizado de máquina. Em muitas aplicações de visão computacional, como classificação de imagens e detecção de objetos, a regularização $L_2$ é comumente empregada como técnica padrão. Já em processamento de linguagem natural, modelos como o *transformer* fazem uso de *dropout*, enquanto o renomado modelo BERT utiliza a regularização $L_2$ [72].

No contexto do aprendizado profundo por reforço, conhecido como *deep reinforcement learning*, a situação é diferente devido à ausência do problema de generalização, que ocorre devido à diferença entre o ambiente de treinamento e o ambiente de testes. Até o momento, muitos pesquisadores têm se concentrado principalmente no desenvolvimento de algoritmos e tendem a negligenciar os desafios relacionados ao treinamento de redes neurais, incluindo a aplicação de técnicas de regularização [72].

No entanto, é importante destacar que, mesmo no contexto de *deep reinforcement learning*, a regularização pode desempenhar um papel significativo, especialmente em cenários onde o *overfitting* pode ocorrer, como em ambientes complexos ou com dados ruidosos. Portanto, a escolha e aplicação adequada de técnicas de regularização ainda podem ser relevantes para melhorar o desempenho de modelos de aprendizado profundo por reforço.

### 2.16.1 *Dropout*

A função Dropout funciona de maneira distinta das regularizações $L_1$ e $L_2$, pois não envolve modificações diretas na função de custo. Seu funcionamento consiste em temporariamente desativar (ou "eliminar") alguns dos neurônios nas camadas ocultas da rede durante o treinamento, especificamente durante o processamento de mini-lotes de dados. Dessa forma, os valores de entrada são encaminhados pela rede neural sem considerar os neurônios desativados. O erro é então calculado e as correções nos pesos da rede

são aplicadas normalmente apenas nos neurônios que estão ativos durante esse mini-lote. Esse processo de "forward pass"e "backward pass"é repetido com diferentes neurônios desativados em cada mini-lote [73].

À medida que esse processo é repetido várias vezes durante o treinamento, a rede neural aprende os pesos e vieses sob essas condições de desativação intermitente de neurônios. Quando a rede é usada para inferência completa, ou seja, para fazer previsões ou classificações em todo o conjunto de dados, costuma-se reduzir os valores dos pesos pela metade, compensando o efeito da desativação temporária durante o treinamento. Além disso, a técnica de Dropout também tem o benefício de reduzir a probabilidade de coadaptações entre neurônios, uma vez que esses neurônios não podem depender da presença de neurônios específicos durante o processo de aprendizado [74, 73].

## 2.17 Algotrading e o Mercado de ações

O mercado de ações apresenta uma ampla gama de técnicas disponíveis para tomar decisões que visam maximizar os retornos futuros dos ativos financeiros. Entre essas técnicas, destacam-se a análise fundamentalista, a análise técnica e a análise algorítmica, também conhecida como *algotrading*. A análise fundamentalista se baseia em fatores econômicos para estimar o valor intrínseco dos títulos financeiros, enquanto a análise técnica utiliza dados históricos das ações como base para suas decisões.

Em geral, os sistemas de *trading* buscam fazer previsões por meio de modelos preditivos treinados em séries históricas de preços de ativos. Esses modelos avaliam as mudanças de valor com base em variáveis exploratórias, que compõem o conjunto de características do sistema. Essas previsões são obtidas por meio de um módulo de treinamento, alimentado com dados históricos, com o objetivo de tomar ações. A eficácia dessas ações é posteriormente comparada aos resultados reais, visando a minimização do erro quadrático médio nas amostras de treinamento, permitindo assim ajustar o modelo durante o processo de treinamento [9, 75].

No entanto, devido ao fato da minimização do erro é um passado intermediário e não o objetivo durante o treinamento do modelo têm-se perda de informações importantes e, consequentemente, um desempenho insatisfatório [9]. Além disso, muitas vezes esses modelos não consideram fatores externos ao mercado, como a falta de liquidez e os custos de transação, que são elementos cruciais na prática financeira e raramente são incorporados aos componentes de negociação. O aprendizado por reforço surge como uma abordagem

promissora para superar essas limitações, uma vez que os agentes de aprendizado, interagindo diretamente com o ambiente, acabam incorporando esses fatores exógenos ao seu processo de tomada de decisão [75].

Dessa forma, em vez de simplesmente adotar estratégias baseadas em dados históricos, o aprendizado por reforço permite que o agente se adapte e aprenda com as ações tomadas em negociações cotidianas [2]. Essa flexibilidade é viável porque, nos mercados financeiros, as negociações podem ser simplificadas de acordo com os princípios da Teoria de Dow 2.17.1 e tratadas como um MDP simplificado, o que é compatível com a abordagem de aprendizado por reforço.

### 2.17.1 A Teoria de Dow

A Teoria de Dow, que fornece uma base fundamental para a análise técnica no mercado de ações, pode ser sintetizada em cinco princípios essenciais:

- **As Médias Descontam Tudo:** Este princípio implica que as médias, que representam o comportamento coletivo de milhares de investidores, as médias incorporam todas as informações disponíveis no mercado. Isso inclui eventos previsíveis, bem como fatores que afetam a oferta e demanda, como desastres naturais.

- **O Mercado se Move em Tendências:** A Teoria de Dow postula que os preços das ações não se movem de forma aleatória, mas sim em tendências. Essas tendências podem ser de alta ou baixa e incluem tendências primárias, secundárias e terciárias. Dentro de cada tendência primária, existem movimentos oscilatórios menores conhecidos como correções. Além disso, as tendências secundárias contêm movimentos oscilatórios menores em relação a elas, chamados de tendências terciárias.

- **Tendência Permanece até que Haja Evidência de Reversão:** A Este princípio enfatiza que uma tendência persiste até que haja sinais claros e convincentes de reversão. Os investidores técnicos buscam identificar esses sinais para tomar decisões informadas.

- **Os Padrões se Repetem:** A Teoria de Dow argumenta que o mercado passa por ciclos e que esses ciclos tendem a se repetir. Os analistas técnicos procuram identificar padrões de comportamento passados que podem se repetir no futuro.

- **O Volume Acompanha a Tendência:** O volume de negociações é considerado um indicador importante que acompanha as tendências. O aumento do volume pode

fornecer evidências adicionais para confirmar ou refutar uma tendência.

Esses princípios fundamentais da Teoria de Dow permitem que os comportamentos do mercado sejam modelados e expressos matematicamente, contribuindo assim para a análise técnica e a tomada de decisões no mercado financeiro [76].

Portanto, é amplamente aceito que o uso de indicadores técnicos, que se baseiam em conceitos da Teoria de Dow e outros princípios da análise técnica, podem ser valiosos na previsão e tomada de decisões no mercado de ações.

### 2.17.2 Indicadores técnicos

Os indicadores técnicos desempenham um papel fundamental na análise técnica dos mercados financeiros. São ferramentas projetadas para avaliar a probabilidade de ocorrência de eventos futuros com base em padrões de preços e volumes observados no passado. A análise técnica utiliza esses indicadores para identificar tendências de preços, volatilidade,momento, forças do mercado, ciclos e outros aspectos relevantes [77].

Esses indicadores são calculados com base em dados como preços de abertura, fechamento, máximos e mínimos horários, bem como volumes de negociação. Eles fornecem informações valiosas aos investidores e *traders*, ajudando-os a tomar decisões informadas sobre compra e venda de ativos financeiros. Dessa forma, os indicadores técnicos são amplamente utilizados por especialistas no campo das finanças para a análise de tendências e a tomada de decisões no mercado financeiro [77].

# Capítulo 3

# Estado da Arte

Este capítulo tem como objetivo apresentar os avanços e estudos realizados na modelagem de séries temporais que descrevem o comportamento dos preços dos ativos financeiros. Dado que as informações relacionadas aos ativos podem ser entendidas como uma coleção de observações sequenciais registradas ao longo de um período específico, esses dados são tratados como séries temporais. A análise e previsão dessas séries temporais, juntamente com a capacidade de modelar padrões matematicamente, têm despertado grande interesse entre pesquisadores do mercado financeiro.

As várias abordagens de pesquisa nesta área podem ser divididas em algumas categorias principais:

- **Abordagens Estatísticas:** Algumas pesquisas utilizam ferramentas estatísticas tradicionais, como modelos ARIMA (*Autoregressive Integrated Moving Average*), ARMA (*Autoregressive Moving Average*), GARCH (*Generalized Autoregressive Conditional Heteroskedasticity*) e ARCH (*Autoregressive Conditional Heteroskedasticity*), para modelar os movimentos dos preços dos ativos [78, 79].

- **Sistemas Híbridos:** Outra linha de pesquisa envolve o uso de sistemas híbridos, que combinam diferentes técnicas para obter resultados mais precisos. Um exemplo disso é a combinação de redes neurais fuzzy com algoritmos genéticos para a negociação de ações [80].

- **Aprendizado Supervisionado:** Algoritmos de aprendizado supervisionado, como *Support Vector Machines* (SVM) e Árvores de Decisão, também são utilizados para modelagem e previsão de preços de ativos financeiros [81].

- **Aprendizado por Reforço:** Por fim, sistemas de aprendizado por reforço, nos

quais agentes autônomos aprendem estratégias para negociação de ativos, têm sido amplamente explorados. Um exemplo é o uso de agentes autônomos para aprender estratégias de contratos futuros contínuos [6].

É evidente que existem várias abordagens que podem ser consideradas nos desafios relacionados à modelagem e previsão de preços de ativos financeiros, e essas abordagens podem auxiliar significativamente no processo de tomada de decisão ou mesmo automatizá-lo. Este trabalho está focado no aprendizado por reforço, e as pesquisas analisadas a seguir se concentram em estratégias de compra e venda de ativos financeiros.

## 3.1 Reinforcement Learning for Trading

O trabalho apresentado por [82] é baseado em um sistema de aprendizado por reforço recorrente, conhecido como *Recurrent Reinforcement Learning* (RRL).

Neste estudo, os autores treinaram sistemas de negociação (*trading*) por meio do aprendizado por reforço com o objetivo de otimizar funções objetivas, utilizando o índice de Sharp como métrica de desempenho para aprendizado online. Os resultados obtidos sugerem que o aprendizado por reforço é mais eficaz do que abordagens baseadas em aprendizado supervisionado, especialmente quando se consideram os custos de transação envolvidos nas operações financeiras.

Além disso, o método RRL utilizado neste estudo superou as aplicações baseadas em Q-Learning, uma vez que estas podem ser afetadas pela "maldição da dimensionalidade". Esse fenômeno ocorre quando o aumento na quantidade de informações analisadas leva a uma redução na precisão do modelo, tornando o método RRL uma escolha mais adequada para lidar com problemas complexos de aprendizado em finanças.

## 3.2 Deep Reinforcement Learning for Trading

No artigo apresentado por [6], foi utilizado um modelo de *Deep Reinforcement Learning* (DRL) no desenvolvimento de estratégias de negociação para contratos futuros contínuos. Esta abordagem considerou tanto espaços de ação contínuos quanto discretos, incorporando funções de recompensa que levam em conta a volatilidade de mercado.

O estudo foi conduzido em um conjunto de 50 contratos futuros de alta liquidez, abrangendo diversos ativos, como commodities, índices de mercado, renda fixa e mercado

de câmbio, durante o período de 2011 a 2019. Os algoritmos desenvolvidos foram comparados com estratégias de momento em séries temporais, demonstrando que o método tem melhores resultados que as linhas de base comparativa.

Para espaços de ações discretos, foi adotado um conjunto de ações simples ($A \in [-1, 0, 1]$). A ação -1 representou a posição *short*, na qual o investidor vende ativos sem tê-los em mãos, comprometendo-se a adquiri-los posteriormente para cumprir a venda. A posição *short* representada por -1 (menos um), ocorre onde o investidor assume a posição de venda sem ter o ativo em mãos e quando firmado o acordo de venda o ator adquire os ativos prometidos e cumpre a venda. Também tem-se a ação 0 (zero) que consistem e segurar o dinheiro. Por fim, têm-se a ação 1 (um) que representa a posição *long*, ou posição comprada, onde o investidor compra ativos de uma empresa. Essa abordagem de espaço de ações é conhecida como *target orders*, a qual a posição na negociação é retornada em vez da ação a ser realizada.

No caso de espaços de ações contínuos, as posições possíveis eram semelhantes às ações discretas, mas permitiam valores contínuos entre -1 e 1 ($a_t \in [-1, 1]$).

Por fim, os resultados mostraram que o uso de características do momento de séries temporais e indicadores técnicos para formar representações de estado, em conjunto com escala de volatilidade para aprimoramento da função de recompensa obtiveram resultados superiores aos modelos que servem de referência para avaliação de performance dos modelos desenvolvidos mesmo considerando custos de transação altos.

## 3.3 A Q-learning agent for automated trading in equity stock markets

O estudo realizado por [15] aborda a questão crítica de selecionar a estratégia de negociação adequada para ações em diferentes períodos no mercado financeiro. Eles destacam que uma estratégia de negociação não é universalmente lucrativa para todas as ações o tempo todo, tornando a escolha da estratégia apropriada um desafio significativo no mercado de ações.

Para abordar esse problema, os autores propõem o uso de um modelo de aprendizado por reforço, que é capaz de encontrar a melhor estratégia de negociação dinâmica interagindo com o mercado por meio de um ambiente. Eles treinaram um agente usando o algoritmo Q-learning para encontrar a política que descreve a melhor estratégia de negociação dinâmica, e enfatizam que a representação do estado do ambiente é crucial para o

desempenho do modelo. No trabalho, foram propostas duas abordagens para a representação do estado: uma baseada em agrupamento (*cluster*) e outra usando *candlestick*.

Os resultados apresentados pelos autores são encorajadores, destacando que o desempenho do agente Q-learning é fortemente dependente da representação do estado. Na primeira abordagem, que utiliza o agrupamento, eles propõem uma maneira inovadora de formar os estados do ambiente, representando o comportamento do mercado de ações por meio de um número finito de estados (clusters) obtidos por um algoritmo de aprendizado não supervisionado, o k-means. Essa abordagem mostra que o aumento no número de clusters é benéfico para prever tendências de alta, embora não tenha efeito significativo nas ações de tendência lateral.

A segunda abordagem emprega *candlestick* para representar os estados do ambiente. Os resultados experimentais demonstram que o desempenho do modelo não depende das tendências das oscilações dos ativos. Além disso, ambos os modelos propostos superam modelos de referência, como árvores de decisão e modelos baseados em estratégias de *buy-and-hold* (comprar e segurar), em termos de retorno de investimento. Isso sugere que as abordagens de aprendizado por reforço são promissoras para a seleção dinâmica de estratégias de negociação no mercado de ações.

## 3.4 Improving Financial Trading Decisions Using Deep Q-learning: Predicting the Number of Shares, Action Strategies, and Transfer Learning

No artigo de [83], são propostos três métodos de aprendizado por reforço com o objetivo de maximizar os lucros totais em situações reais do mercado financeiro, buscando as limitações dos dados financeiros. O estudo avaliou ações do S&P500, KOSPI, HSI e EuroStoxx50.

O primeiro método apresentado utiliza um sistema autônomo de negociação que emprega um regressor de rede neural profunda (DNN) para prever o número de ações a serem negociadas e a ação a ser tomada pelo agente. O treinamento da rede consistiu em reduzir o espaço de busca a cada etapa do treinamento, passo, e avaliar a eficiência das previsões.

O segundo método investigou várias estratégias de ação que, com base em valores Q, analisam o desempenho de diferentes estratégias de ação em um mercado incerto, com o objetivo de maximizar os lucros. Uma estratégia notável foi a ação de segurar o ativo

com um limiar de 0.2, o que permitiu adiar a decisão de compra ou venda em situações de incerteza.

A terceira abordagem apresentada envolve estratégias de transferência de aprendizado, com o objetivo de evitar o sobre-ajuste quando há uma quantidade insuficiente de dados disponíveis durante o treinamento.

Os resultados obtidos foram promissores, mesmo considerando a escassez e alta volatilidade dos dados financeiros, que tornam a tomada de decisão desafiadora. No primeiro método, o sistema autônomo de negociação com regressor DNN para ajustar o número de ativos negociados resultou em aumentos significativos nos lucros totais das negociações em quatro vezes para a S&P500, cinco vezes para KOSPI, doze vezes para a HSI e seis vezes EuroStoxx50 quando comparado ao sistema com um número fixo de ativos negociados.

Na segunda abordagem, a estratégia de adiar a tomada de decisão em mercados confusos resultou em aumentos substanciais nos lucros totais em 18% na S&P500, 24% para a KOSPI e 49% para a EuroStoxx50. Ao comparar o resultado obtido com o original todos os modelos aumentaram seus lucros sendo que obteve-se 13 vezes mais lucros em S&P500, 24 vezes em KOSPI, 30 vezes em HSI e 18 vezes em EuroStoxx50.

Por fim, o uso de transferência de aprendizado em cenários com menos dados financeiros e a disponibilidade de modelos pré-treinados para situações semelhantes também se mostrou eficaz, aumentando os lucros totais em todas as ações avaliadas. Nesta abordagem o eventual sobre-ajuste aos dados é tratado com a escolha de componentes altamente relacionados ao índice em estudo. Assim, este método aumentou o lucro total obtido duas vezes para S&P500 e HSI, três vezes para KOSPI e duas vezes e meia para EuroStoxx50.

Esses resultados sugerem que os métodos de aprendizado por reforço têm o potencial de melhorar significativamente o desempenho na negociação de ativos financeiros em ambientes complexos e voláteis.

## 3.5 Stock trading rule discovery with double deep Q-network

O estudo conduzido por [11] se baseia em um sistema de aprendizado por reforço que faz uso do *Double Deep Q-Learning* (DDQL). Este trabalho propõe a utilização de um novo modelo de aprendizado, no qual a política é definida por meio de um conjunto de múltiplas camadas convolucionais, com o intuito de extrair os relacionamentos latentes

presentes nos dados das negociações de ações ao longo de diferentes datas.

As camadas convolucionais em uma rede neural constituem-se em um conjunto de filtros aplicados a uma imagem ou a um mapa de características. Esses mapas de características são gerados por meio de filtros ou detectores de características aplicados aos dados de entrada.

O modelo desenvolvido visa prever a tendência de preço das ações e, posteriormente, realizar operações de negociação. A análise demonstra que o uso de políticas baseadas em camadas convolucionais possibilita a exploração e extração das dependências latentes nos dados das ações.

Os resultados obtidos pelos autores indicam que o DDQL pode ser uma abordagem eficaz para reduzir a superestimação e aumentar a estabilidade do modelo. O agente, ao empregar a estratégia de transação gerada no ambiente controlado proposto no estudo, obteve êxito ao gerar altos retornos financeiros. No entanto, ressalta-se que é necessária a realização de estudos adicionais em diferentes ativos financeiros para validar a aplicabilidade dessa abordagem de forma mais abrangente.

# Capítulo 4

# Metodologia

A proposta central deste trabalho é a implementação de um agente autônomo baseado na técnica *Double Deep Q-Learning* (DDQL), com o objetivo primordial de maximizar o retorno nas operações de *trading*. Para alcançar esse objetivo, o DDQL utiliza uma rede neural para prever a ação mais adequada com base no estado observado do mercado. Nesse contexto, o estado é definido pelas métricas que alimentam a rede, como o valor monetário e o volume negociado do ativo no período analisado.

É importante ressaltar que, considerando que os indicadores relacionados ao preço representam de forma eficaz o comportamento histórico do ativo ao longo do tempo, eles fornecem um histórico que pode ser utilizado tanto por um analista humano na tomada de decisões quanto por um agente autônomo. Portanto, os indicadores de preço são suficientes para representar o ambiente como um Processo de Decisão de Markov (MDP) simplificado. Esse MDP simplificado é essencial para a construção e treinamento do agente de aprendizado por reforço.

## 4.1 Seleção de dados

Os dados necessários para alimentar o modelo foram adquiridos por meio da API Bitfinex[1]. Esta API oferece a possibilidade de obtenção das métricas OCHL (Open, Close, High, Low) e do volume de transações para os ativos escolhidos, que neste caso são o *Bitcoin* e o *Ethereum*. A escolha deste ativo se deve a possibilidade da obtenção dos dados históricos no intervalo desejado. Além disso, foram calculados diversos indicadores comumente utilizados por traders, conforme detalhado na Seção 4.3, com base nos valores

[1] A API Bitfinex qproporciona acesso aos recursos da plataforma Bitfinex e estabelece as regras de interação entre aplicativos, seguindo um conjunto predefinido de diretrizes [84].

de fechamento dos ativos.

As métricas OCHL representam os preços de abertura, fechamento, máxima e mínima do ativo durante o período de tempo considerado. É importante observar que os preços do *Bitcoin* e do *Ethereum* estão cotados em dólares americanos. As métricas OCHL, o volume de transações e os indicadores calculados estão disponíveis em intervalos de 1 hora.

Os dados relacionados ao *Bitcoin* foram coletados no período compreendido entre 14 de março de 2016 e 08 de setembro de 2023, e podem ser visualizados na Figura 4.

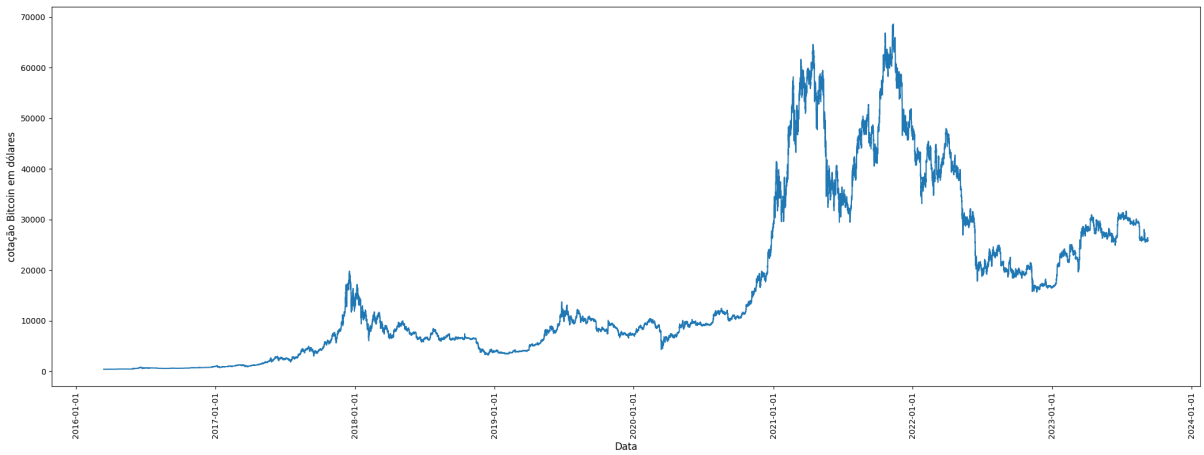


Figura 4: Ativo financeiro Bitcoin (cotação em dólar)

Fonte: Autor (2022)

Os dados do *Ethereum*, também foram obtidos no período compreendido entre 14 de março de 2016 e 08 de setembro de 2023, como demonstrado na Figura 5.

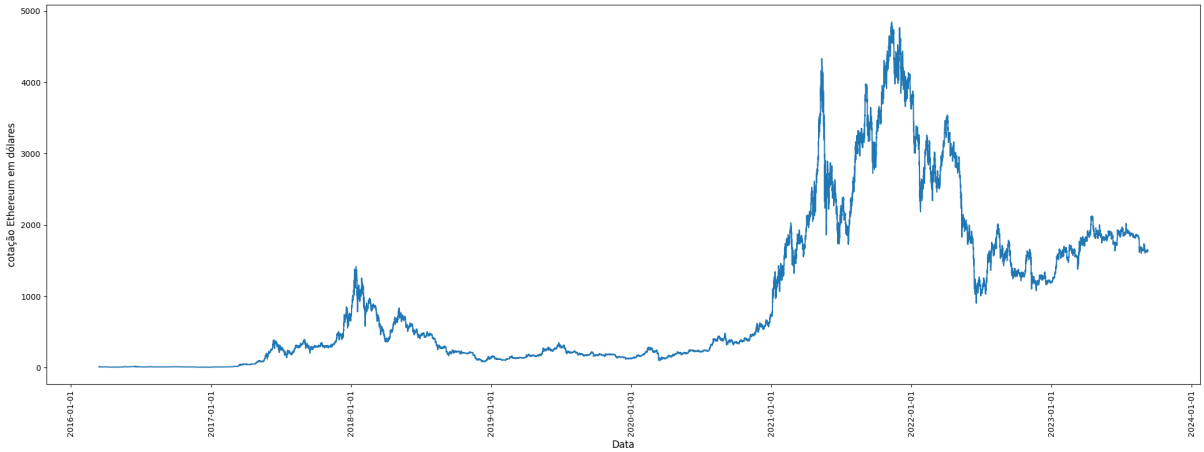


Figura 5: Ativo financeiro Ethereum (cotação em dólar)

Fonte: Autor (2022)

A coleta de dados em intervalos de 1 hora possibilita ao algoritmo de aprendizado identificar os padrões sazonais de acordo com a periodicidade. Além disso, durante o processo de preparação dos dados, foram adicionados os indicadores mencionados na Seção 4.2.

## 4.2 Pré-processamento de Dados

Os dados utilizados neste estudo passaram por um processo de pré-processamento rigoroso para garantir a qualidade e a adequação ao treinamento do modelo de DDQL.

Primeiramente, no processo de remoção de valores faltantes, os dados foram minuciosamente examinados, e qualquer entrada que contivesse valores faltantes ou informações incompletas foi eliminada. Esse procedimento ajuda à evitar distorções e imprecisões no treinamento do modelo, garantindo a integridade dos dados utilizados.

A normalização permite dimensionar os valores de entrada para que estejam dentro de um intervalo padrão para, desta forma, evitar que diferenças significativas entre os valores possam prejudicar o aprendizado do modelo. Portanto, a normalização contribui para melhorar a estabilidade e a convergência do modelo durante o treinamento.

Essas etapas de pré-processamento são essenciais para assegurar que os dados estejam em perfeitas condições para o treinamento do modelo de Double Deep Q-Learning. Isso, por sua vez, possibilita que o modelo faça previsões precisas e confiáveis com base em informações de alta qualidade e devidamente normalizadas.

## 4.3 Indicadores técnicos

No contexto da análise de mercado e na identificação de tendências e padrões de preço, os indicadores técnicos desempenham um papel crucial. Eles servem como ferramentas auxiliares valiosas na determinação do momento ideal para tomar decisões de compra ou venda de ativos financeiros.

Nesta sessão, foram apresentados alguns indicadores técnicos relevantes e a metodologia empregada para o cálculo deles. Esses indicadores são amplamente empregados na análise de dados financeiros e desempenham um papel importante no desenvolvimento e treinamento do modelo de Double Deep Q-Learning para negociações dos ativos. Portanto, é essencial compreender como esses indicadores são calculados e como podem influenciar o processo de tomada de decisão do agente autônomo.

### 4.3.1 Média Móvel Simples

A média móvel é um indicador fundamental na análise técnica de ativos financeiros, desempenhando um papel crucial na identificação de tendências nos preços, sejam elas de alta, neutra ou baixa. Esse indicador recebe o nome de "média móvel"devido ao cálculo da média ser baseado em um intervalo de $n$ períodos, conhecido como janela de observações, que se desloca à medida que novos dados são incorporados, permitindo calcular a média para os $n$ períodos mais recentes.

A média móvel simples (MMS) é calculada conforme apresentado na equação (4.1):

$$MMS = \frac{PF_1 + PF_2 + ... + PF_n}{n} \tag{4.1}$$

Onde $PF$ representa o preço de fechamento no período, e $n$ é o tamanho da janela de observações.

Neste contexto, foram adicionados três indicadores calculados com base na média móvel simples de 7, 25 e 99 períodos do preço de fechamento. Esses indicadores são empregados na filtragem de ruídos de mercado e na identificação da direção das tendências de preços, bem como na detecção de reversões e movimentos significativos de compra ou venda no mercado. Sendo relevantes para o processo de tomada de decisão do agente autônomo durante as negociações diárias.

### 4.3.2 Bandas de Bollinger

Outro indicador adicionado a este estudo foram as Bandas de Bollinger, que constituem um indicador de volatilidade amplamente empregado para prever se um ativo financeiro está em um estado de sobrecompra, estabilidade ou sobrevenda.

Um ativo é considerado sobrecomprado quando seu preço registra um aumento constante e, como resultado, passa a operar acima de seu valor intrínseco. Por outro lado, um ativo é classificado como sobrevendido quando seu preço cai devido a um intenso movimento de venda. Em contrapartida, ele é considerado estável quando se encontra entre esses dois estados.

As Bandas de Bollinger consistem em três curvas distintas: a linha central, a banda superior e a banda inferior. Essas curvas são empregadas para antecipar os pontos de máximo e mínimo nos preços, indicando se o preço de um ativo está em um estado de subvalorização ou sobrevalorização. Essas bandas, derivadas de médias móveis, sinalizam que os preços tendem a retornar ao equilíbrio.

O cálculo das Bandas de Bollinger baseia-se em uma média móvel simples, geralmente de 20 períodos, como segue:

- *Linha central:* Média móvel simples (20 períodos)
- *Banda superior:* Média móvel simples (20 períodos) + (2 vezes o desvio padrão de 20 períodos)
- *Banda inferior:* Média móvel simples (20 períodos) - (2 vezes o desvio padrão de 20 períodos)

É importante observar que as Bandas de Bollinger tendem a funcionar eficazmente em mercados que apresentam tendências claras. Quando um ativo ultrapassa a banda superior, isso indica uma forte tendência de alta, enquanto quando ele cai abaixo da banda inferior, sugere uma tendência de baixa. No entanto, é crucial estar atento a possíveis reversões de tendência, nas quais os ativos podem romper as bandas superior e inferior, sinalizando uma possível mudança na direção dos preços. Esses indicadores desempenharão um papel significativo na análise de tendências e na tomada de decisões do agente autônomo durante as negociações diárias de ações.

### 4.3.3 SAR Parabólico

Outro indicador adicionado à entrada de dados deste estudo é o SAR Parabólico, que atua como um rastreador de tendências e um gerador de sinais de entrada e saída no contexto das negociações diárias de ações. A sigla SAR deriva do termo em inglês *Stop And Reverse*, que se traduz como "pare e reverta". O termo "parabólico"refere-se à forma da curva formada pelos conjuntos de pontos gerados por este indicador, que assemelham-se a uma parábola. Portanto, quando o indicador sinaliza uma possível mudança de tendência, é momento de considerar uma reversão na posição do ativo.

A fórmula que define o SAR Parabólico é apresentada na Equação (4.2).

$$SAR_{i+1} = SAR_i \pm \alpha_i(PE_i - SAR_i) \tag{4.2}$$

Nessa equação, o SAR futuro ($SAR_{i+1}$) é calculado com base no $SAR$ atual ($SAR_i$), no Ponto Extremo ($PE$) e no Fator de Aceleração ($\alpha$). O Ponto Extremo ($PE_i$) representa o valor mais extremo de uma tendência de alta ou baixa, geralmente correspondente aos preços *High* ou *Low* do histórico OCHL. O Fator de Aceleração ($\alpha$) inicia-se em 0.02 e é incrementado em 0.02 sempre que ocorre um novo valor para o Ponto Extremo ($PE$).

Em termos gerais, quando os pontos do SAR Parabólico se movem abaixo das barras de preços, isso pode indicar uma tendência de alta. Por outro lado, quando esses pontos se deslocam acima das barras de preços, sugere-se uma tendência de baixa e pode ser um momento apropriado para considerar a venda do ativo. Este indicador desempenha um papel significativo na identificação de tendências e na tomada de decisões do agente autônomo durante as operações de compra e venda de ações.

### 4.3.4 Índice de Força Relativa

Por fim, último indicador incorporado ao conjunto de dados de treinamento do modelo é o Índice de Força Relativa, conhecido como *Relative Strength Index* (RSI). Trata-se de um oscilador de *momentum* amplamente utilizado para avaliar a velocidade e a magnitude das mudanças nos preços dos ativos financeiros.

O cálculo do RSI é realizado com base na Equação (4.3):

$$RSI = 100 - (1 + RS) \tag{4.3}$$

Onde $RS$ representa a razão entre o ganho médio ($AG$) e a perda média ($AL$). Os ganhos e perdas médias necessários para calcular a Força Relativa ($RS$) são inicializados de acordo com as Equações (4.4) e (4.5), respectivamente. O período utilizado para o cálculo foi estabelecido em 14.

$$1AG = \frac{(G_{i-13} + G_{i-12} + ... + G_{i-1} + G_i)}{14} \tag{4.4}$$

$$1AL = \frac{(L_{i-13} + L_{i-12} + ... + L_{i-1} + L_i)}{14} \tag{4.5}$$

Onde $1AG$ representa o primeiro ganho médio e $1AL$ é a primeira perda média. Posteriormente, a Força Relativa ($RS$) é calculada de acordo com a Equação (4.6):

$$RS = \frac{AG}{AL} = \frac{\frac{AG_{i-1}*13+G_i}{14}}{\frac{AL_{i-1}*13+L_i}{14}} = \frac{AG_{i-1} * 13 + G_i}{AL_{i-1} * 13 + L_i} \tag{4.6}$$

Os valores resultantes do Índice de Força Relativa variam de 0 a 100. Em resumo, valores acima de 70 indicam que o ativo está sobrecomprado, sugerindo a possibilidade de uma correção de preço ou reversão de tendência iminente. Por outro lado, valores abaixo de 30 indicam que o ativo está sobrevendido, o que pode ser interpretado como um sinal de possível valorização. Esse indicador é particularmente útil em períodos de incerteza de mercado, quando os preços oscilam entre movimentos de alta e baixa.

## 4.4 Modelo de Aprendizado de Máquina

O algoritmo escolhido para ser empregado neste estudo é o *Double Deep Q-Learning* (DDQL), uma evolução do *Deep Q-Learning* que se destaca como uma técnica eficaz de aprendizado por reforço. O DDQL é um algoritmo livre de modelo (*model-free*) que utiliza duas redes neurais artificiais (RNAs) para otimizar a função Q. Essa escolha foi embasada em pesquisas prévias, como o trabalho conduzido por [10], que evidenciou a eficácia do DDQL em comparação com outras variantes do Q-Learning.

Ademais, destaca-se o estudo realizado por [52], no qual o DDQL superou o *Time-Weighted Average Price* (TWAP) em 7 das 9 ações financeiras analisadas. O TWAP é uma medida de desempenho comumente utilizada no mercado financeiro para indicar o preço médio de um ativo financeiro ao longo de um período de tempo específico, e uma estratégia de negociação deve superar esse preço para ser considerada eficaz.

A escolha do DDQL para este estudo baseou-se na sua capacidade comprovada de evitar a superestimação (*overestimation*) das estimativas de valor Q, o que resulta em decisões de negociação mais precisas e confiáveis durante o treinamento do agente.

### 4.4.1 Espaço dos estados

O espaço de estados analisados neste contexto refere-se à representação dos dados que compõem o índice de preços do *Bitcoin* e *Ethereum*, bem como os indicadores derivados desses dados. Os estados do ambiente, que são fundamentais para a tomada de decisões do agente de aprendizado por reforço, consistem nos elementos previamente descritos nas seções 4.1 e 4.3. Esses elementos compõem o conjunto de informações que o agente utiliza para avaliar a situação atual do mercado e determinar as melhores ações a serem executadas em suas transações de compra e venda de ativos. Portanto, o espaço de estados analisados abrange os dados históricos de preços, bem como os indicadores técnicos calculados a partir desses preços, fornecendo ao agente uma visão abrangente do ambiente de negociação sempre considerando o patrimônio líquido disponível e a quantidade de dinheiro para compra de novos ativos.

### 4.4.2 Agente

A abordagem adotada neste estudo envolve a utilização de um agente baseado em Double Deep Q-Learning (DDQL). Esse agente é responsável por tomar decisões, que são referidas como ações, com base em estimativas de qualidade obtidas por dois estimadores. As ações realizadas pelo agente resultam em recompensas, que podem ser positivas ou negativas, indicando lucro ou prejuízo nas transações.

O aprendizado do agente ocorre por meio de experiências repetidas nas quais as informações relacionadas ao estado observado pelo agente são armazenadas. Essas informações abrangem o estado atual do agente, a ação tomada, a recompensa obtida e o estado subsequente. Todas essas informações são registradas na memória de transição por meio de um mecanismo denominado *replay buffer*.

Após a execução de ações pelo agente e o preenchimento do *buffer*, inicia-se a fase de aprendizado. Durante esse processo, a rede neural do agente é atualizada com base nos dados armazenados na memória de repetição. Duas redes neurais são empregadas nesse processo: a rede alvo e a rede online. Notavelmente, a rede alvo não passa por um processo de aprendizado, sendo utilizada exclusivamente para avaliar o aprimoramento da

rede online.

No decorrer do processo de aprendizado, a rede online é treinada com os dados e realiza previsões, que são subsequentemente comparadas com as saídas da rede alvo. Após essa etapa, os pesos da rede online são copiados para a rede alvo, a qual será utilizada na avaliação do próximo ciclo de treinamento. Esse procedimento se repete ao longo do processo de treinamento do agente.

### 4.4.3 Espaço das ações

O espaço de ações disponíveis para o agente é composto por três interações distintas: compra, venda ou manutenção da posição (segurar). Cada uma dessas ações é representada por valores discretos variando de 0 a 2. O valor 0 representa a ação de manutenção da posição, o valor 1 corresponde à ação de compra e o valor 2 está associado à ação de venda.

Essas ações são representadas por meio de listas, onde cada elemento possui um valor atribuído, de modo que todas as ações tenham o mesmo peso durante o treinamento. Portanto, a ação 0 é representada como [1, 0, 0], a ação 1 como [0, 1, 0], e a ação 2 como [0, 0, 1]. Esse método de representação garante que todas as ações sejam tratadas de maneira equitativa durante o processo de aprendizado do agente.

### 4.4.4 Ambiente

O ambiente é o contexto no qual os estados são observados e onde ocorre o processo de aprendizado do agente. No âmbito deste estudo, o ambiente engloba informações relacionadas aos valores dos ativos nas horas avaliadas e registra as ações realizadas pelo agente durante as interações. Além disso, o ambiente engloba as rotinas necessárias para o armazenamento das métricas utilizadas na avaliação do desempenho do modelo.

### 4.4.5 Recompensa

A recompensa é empregada no objetivo global do agente, que é maximizar o retorno em cada interação com o ambiente ao longo do tempo. O cálculo da recompensa varia conforme a ação tomada pelo agente, que pode ser compra, venda ou manutenção da posição. A recompensa reflete variação percentual do patrimônio líquido entre as iterações descontado o risco, volatilidade e custo de oportunidade, e é um componente crucial do

processo de treinamento do agente.

### 4.4.6 Parâmetros iniciais

No início da simulação, diversos parâmetros são definidos para o agente, permitindo que ele interaja com o ambiente de forma eficaz. Abaixo estão listados alguns desses parâmetros:

- **Arquitetura da rede neural:** A rede neural utilizada no processo de tomada de decisão do agente é composta por duas camadas ocultas, cada uma com 64 neurônios. A camada de entrada possui dimensões que correspondem ao estado observado, e a camada de saída tem um número de unidades igual ao número de ações possíveis.
- **Taxa de aprendizagem:** Este parâmetro controla a taxa de aprendizado aplicada durante o processo de aprendizado da função Q.
- **Fator de desconto:** O fator de desconto, representado por $\gamma$, é um valor entre 0 e 1 que determina o peso dado às recompensas futuras em relação à recompensa imediata. Ele influencia a importância atribuída às recompensas de longo prazo.
- **Fator de exploração:** Inicialmente, o fator de exploração $\epsilon$ é definido como 0.1. Ao longo do treinamento, esse valor decairá, fazendo com que o agente realize menos *exploration* e mais *exploitation* à medida que ganha experiência.
- **Capacidade de repetição:** A capacidade de repetição é configurada para criar a memória de repetição usada durante o treinamento. Essa memória armazena experiências passadas que são usadas para melhorar as decisões futuras.

Esses parâmetros auxiliam no desempenho e na capacidade de aprendizado do agente, garantindo que ele possa aprender e tomar decisões de forma eficaz ao longo do treinamento.

## 4.5 Estratégia de negociação

A estratégia de negociação adotada neste estudo se baseia no algoritmo de aprendizado por reforço conhecido como *Double Deep Q-Learning* (DDQL). Essa abordagem, caracterizada por ser livre de modelo (model-free), tem como propósito a tomada automatizada e eficaz de decisões de compra e venda de ativos financeiros, como ações.

Para iniciar a estratégia, a primeira etapa envolveu a escolha dos ativos financeiros a serem negociados. Neste estudo, foram selecionadas as cotações de duas moedas digitais.

Para auxiliar na tomada de decisões, a estratégia incorpora indicadores técnicos amplamente utilizados para a análise de mercado. Esses indicadores incluem os mencionados na seção anterior.

Com base nos indicadores técnicos mencionados em 4.3, o algoritmo DDQL toma decisões de compra ou venda. Essas decisões são orientadas pelo aprendizado da função Q, que avalia a recompensa esperada de cada ação em um determinado estado de mercado.

O DDQL é um algoritmo adaptativo que permite ajustes dinâmicos com base no desempenho observado. Ele é capaz de aprender com os erros e otimizar suas decisões ao longo do tempo.

A escolha dessa estratégia se deve à sua capacidade de lidar com a imprevisibilidade inerente aos mercados financeiros, bem como à sua flexibilidade para se adaptar a diferentes ativos e condições de mercado. A implementação do DDQL na estratégia visa maximizar o patrimônio líquido do investidor ao longo do tempo, mitigando os riscos associados à negociação de ativos financeiros.

## 4.6 Configuração do Ambiente de Experimentação

Os experimentos e treinamentos conduzidos neste estudo foram realizados em um computador com a seguinte configuração de hardware:

- **Processador (CPU):** AMD Ryzen 5 5600, com 6 núcleos e 12 threads a uma frequência base de 3,5 GHz e uma frequência máxima de *boost* de 4,4 GHz.
- **Placa de Vídeo (GPU):** NVIDIA GeForce RTX 3070 Ti, escolhida por sua capacidade de aceleração em tarefas de aprendizado de máquina e processamento gráfico, com seus 6144 processadores Cuda sendo utilizados para processamento via Tensorflow.
- **Memória RAM:** 64 GB de RAM DDR4, operando a uma frequência de 3000 MHz.
- **Armazenamento (SSD):** Um SSD de 512 GB foi utilizado para garantir acesso rápido a dados durante os experimentos.

## 4.7 Implementação do Projeto

A implementação do projeto foi conduzida com o auxílio da IDE Visual Studio Code, da Microsoft, e a linguagem de programação Python na versão 3.10. A infraestrutura tecnológica do projeto foi sustentada pelo TensorFlow, um framework de código aberto amplamente reconhecido por sua robustez e eficácia na implementação de estruturas relacionadas a aprendizado de máquina. Em particular, a API Keras, integrada ao TensorFlow na versão 2.10, desempenhou um papel central no desenvolvimento da rede neural utilizada pelo agente durante o treinamento [85].

O ambiente de treinamento foi configurado de maneira cuidadosa, considerando a inicialização das variáveis ambientais necessárias para o processo de aprendizado bem como a configuração necessária para possibilitar o processamento via GPU por meio do Tensorflow e processadores Cuda.

O ciclo de aprendizado em sistemas DDQL consiste em duas etapas principais: a fase de experimentação, na qual o agente interage com o ambiente, e a etapa de treinamento. Durante a fase de experimentação, o agente realiza ações no ambiente e armazena experiências relevantes no chamado *replay buffer*, conforme ilustrado na Figura 6. Esse *buffer* serve como um repositório de dados que serão posteriormente usados no treinamento do agente.

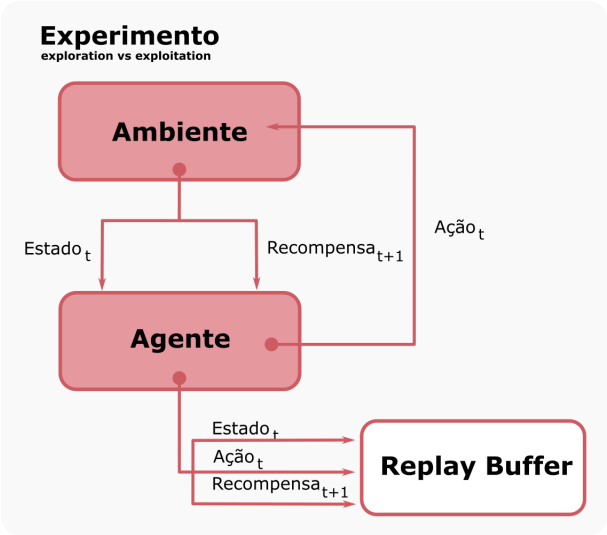


Figura 6: DDQL: Etapa de experimentação com o ambiente

Fonte: Autor (2023)

A partir das amostras armazenadas no *replay buffer*, têm-se o processo de treinamento.

Neste, os pesos da rede neural online são atualizados com base na função de perda, conforme ilustrada na figura 7. Além disso, é importante destacar que, em função do parâmetro tau, a função alvo é atualizada de forma mais suave, contribuindo para um processo de treinamento mais estável e eficiente.

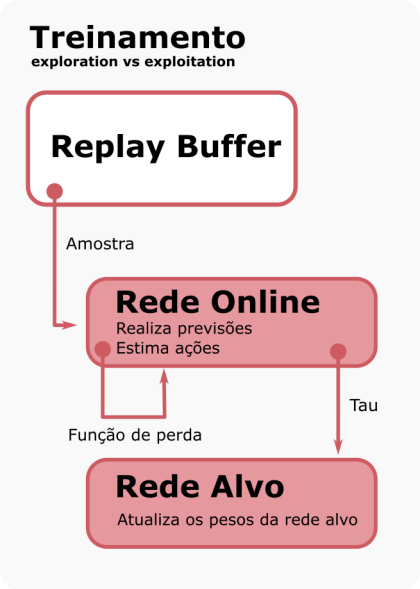


Figura 7: DDQL: Etapa de treinamento

Fonte: Autor (2023)

A biblioteca Python *collections* desempenhou um papel relevante na implementação, fornecendo o método Deque, empregado na criação dos *replay buffer* utilizados no treinamento, por meio da técnica conhecida como experiência de repetição, ver Sessão 2.10.1.

Outro aspecto relevante diz respeito à função de ativação empregada na rede neural, que foi a ReLU. Tal escolha se justifica pelo fato de a ReLU ser mais eficiente na propagação do gradiente em comparação com a função Sigmoid. Além disso, ela ajuda a mitigar o problema de dissipação do gradiente, no qual os valores dos pesos tendem a se aproximar de zero durante o processo de retropropagação (*backpropagation*), levando a uma estimativa de Q mais objetiva.

Durante o treinamento, os dados resultantes das experiências da fase de experimentação são armazenados e utilizados para diversas finalidades, incluindo o cálculo da recomensa, o registro das ações executadas pelo agente, a análise do retorno por etapa, a coleta de informações sobre os preços OCHL (Open, Close, High, Low), o volume negociado, os indicadores técnicos calculados, bem como o acompanhamento do balanço e patrimônio líquido do agente. Esses dados desempenham um papel crucial na avaliação

do desempenho do agente e no refinamento contínuo do modelo durante o processo de treinamento.

O Algoritmo 6 apresenta o processo de treinamento do modelo com suas entradas, processamentos e saídas. Este algoritmo descreve o ciclo de treinamento do agente Double Deep Q-Learning, que é de vital importância para o desenvolvimento de uma política eficaz na tomada de decisões no mercado financeiro. Os resultados desse treinamento são subsequentemente aplicados para orientar as ações do agente no ambiente de negociação.

**Algoritmo 6:** Processo de Aprendizado do Double Q-Learning.

**Entrada:** OCHL, Volume, Balanço inicial e Patrimônio Líquido

**Saída:** Ação $a_t$ tomada pelo agente

1 **início**
2 Cálculo de indicadores com base nos valores de fechamento do histórico OCHL.
3 Inicialização do ambiente de treinamento.
4 Inicialização do Agente e construção das Redes Neurais.
5 **início**
6 Inicialização do processo de aprendizado.
7 Acrescenta-se Balanço inicial e Patrimônio Líquido aos dados de entrada.
8 Inicialização das redes neurais $\sigma_{online}$, $\sigma_{alvo}$ e o *replay buffer*
9 **for** *cada iteração* **do**
10 **for** *cada passo no ambiente* **do**
11 O agente observa o estado atual $s_t$ e seleciona a ação $a_t$ com base na política $\pi(s_t, a_t)$.
12 O agente executa a ação $a_t$ e observa o próximo estado $s_{t+1}$ e a recompensa instantânea $r_t = R(s_t, a_t)$ associada à ação.
13 Os pares estado-ação-recompensa $((s_t, a_t, r_t, s_{t+1}))$ são armazenados no replay buffer.
14 **end**
15 **for** *cada atualização de passo* **do**
16 Atualiza a rede alvo com os pesos da rede online
17 $\sigma_{alvo} \leftarrow \sigma_{online}$
18 O agente seleciona uma amostra $e_t = (s_t, a_t, r_t, s_{t+1})$ do replay buffer.
19 É calculado o valor $Q_{online}$ ideal usando a fórmula do DDQL:
20 $Q^*_{online}(s_t, a_t) \approx r_{t+1} + \gamma Q_{alvo}(s_{t+1}, \underset{a_{t+1}}{max} Q_{online}(s_{t+1}, a_{t+1}))$.
21 A rede online $\sigma_{online}$ é treinada com base no estado $s_t$ e no valor Q ideal, ou seja, $Q^*_{online}(s_t, a_t)$.
22 Os pesos da rede alvo $\sigma_{alvo}$ são atualizados com os valores dos pesos da rede online $\sigma_{online}$:
23 $\sigma_{alvo} \leftarrow \sigma_{online}$.
24 **end**
25 **end**
26 **fim**
27 **fim**

# Capítulo 5

# Resultados e Discussões

Nesta seção, estão os resultados da pesquisa, destacando o agente *Double Deep Q-Learning* desenvolvido para a realização das negociações de ativos, bem como a saída visual desenvolvida a visualização das negociações em tempo real.

Este capítulo trata dos desafios enfrentados e resultados obtidos durante o desenvolvimento do modelo. Inicia-se tratando do desafio do *memory overflow* que impedia que o modelo fosse treinado durante um período de tempo maior. Posteriormente apresenta-se o modelo e as saídas visuais. E finaliza-se com as discussões dos resultados.

## 5.1 Resolução do Desafio de Memory Overflow

Durante a fase de implementação do agente, um desafio de natureza significativa que estava diretamente relacionado ao fenômeno conhecido como *memory overflow* (sobrecarga de memória). Esse inconveniente se manifestou em virtude do extenso volume de dados que o agente precisava manipular em sua operação.

Para superar esse desafio, foi adotada uma estratégia de gerenciamento de memória, a qual compreendeu tanto a otimização da forma como os dados eram armazenados quanto a introdução de um algoritmo de descarte de informações baseado na utilização de uma estrutura de dados conhecida como "deque"(abreviação de *double-ended queue* ou fila de dupla extremidade). A utilização desse recurso permitiu ao agente manter um desempenho estável e eficiente, mesmo quando lidando com enormes conjuntos de dados, uma vez que ele mantinha consigo somente as informações mais recentes e relevantes.

Esta abordagem específica consistia em armazenar as observações mais recentes, descartando aquelas mais antigas à medida que novos valores eram registrados.

Neste contexto, o termo experiência refere-se ao conjunto de informações acumuladas pelo agente durante sua interação com o ambiente, compreendendo estados, ações, recompensas, estados subsequentes e um indicador de continuidade que sinaliza se a interação está em andamento ou foi finalizada. Essa estratégia proporcionou ao agente a capacidade de gerenciar sua memória de maneira eficiente, permitindo-lhe manter um histórico recente e relevante de informações para a tomada de decisões.

## 5.2 O modelo

O modelo desenvolvido consiste em um agente que interage com o ambiente executando ações e recebendo uma recompensa pela ação, a qual é então utilizada no processo de aprendizagem autônoma. O código-fonte completo deste projeto está disponível no GitHub, facilitando o acesso e a revisão detalhada. Para mais informações e para acessar o código, consulte o repositório do projeto disponível na referência [86]

### 5.2.1 Dinâmica do modelo

A implementação do ambiente de negociação utilizado para o treinamento compreende três componentes distintas. Na primeira parte, a obtenção de dados é realizada por meio de uma API conhecida como Bitfinex, com a finalidade de capturar as cotações do *Bitcoin* e *Ethereum* em dólares, conforme detalhado na seção 4.1.

A segunda parte desta implementação se concentra na simulação das negociações. Nesta etapa, são calculadas a recompensa associada às ações executadas e os retornos resultantes.

Por fim, na terceira parte da implementação do ambiente, todas as informações pertinentes ao estado atual e à recompensa obtida são agregadas. O agente implementado interage com este ambiente, recebendo em contrapartida uma recompensa. Essa recompensa é armazenada em uma memória de repetição denominada *replay buffer*. Após várias iterações, as informações acumuladas nesse *buffer* são empregadas no processo de treinamento do agente.

## 5.3 Fluxogramas

Nesta seção, estão apresentados os fluxogramas que elucidam o funcionamento abrangente da aplicação.

A representação visual dos fluxogramas desempenha um papel crucial na compreensão detalhada do nosso modelo. Esses diagramas proporcionam uma visão clara e concisa dos processos envolvidos na aplicação, oferecendo aos leitores uma compreensão aprofundada de seu funcionamento.

### 5.3.1 Preparando o ambiente

A automação da configuração do ambiente de desenvolvimento foi implementada para garantir a consistência e a correta configuração do ambiente de desenvolvimento, para isto foi desenvolvido um script em Bash. Este script abrange todas as bibliotecas e dependências necessárias, sendo compatível com várias distribuições Linux, mais precisamente aquelas baseadas em Debian, como o Ubuntu, e distribuições baseadas em Arch, como o Manjaro.

A Figura 8 ilustra o processo de execução do script de preparação do ambiente. Os usuários podem simplesmente executar este script no terminal do Linux, e todas as dependências necessárias serão instaladas automaticamente, garantindo um ambiente de desenvolvimento coeso e pronto para uso.

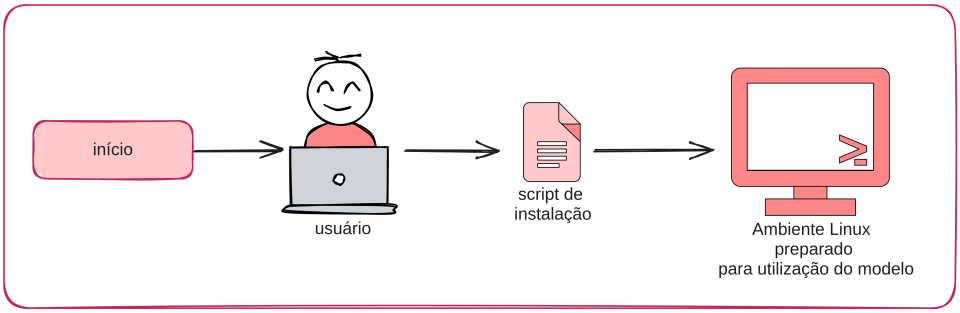


Figura 8: Preparação do ambiente

Fonte: Autor (2023)

### 5.3.2 Obtenção dos dados

Um passo crítico em nossa metodologia envolve a aquisição de dados. Para isso, utilizou-se um comando específico durante a compilação, conforme exemplificado no trecho de código 5.1. Esse comando desencadeia uma chamada à API da Bitfinex, permitindo a recuperação de dados para o intervalo de tempo definido. Posteriormente, esses dados são armazenados em arquivos CSV para facilitar o acesso e a análise futura.

```
# Obtendo dados do Bitcoin
python job.py --dd=BTCUSD
# Obtendo dados do Ethereum
python job.py --dd=ETHUSD
```

Código 5.1: Execução com Download de Dados

A Figura 9 oferece uma visualização detalhada dessa etapa, demonstrando como nosso programa interage com a API, extrai os dados e os armazena em formato CSV para uso posterior.

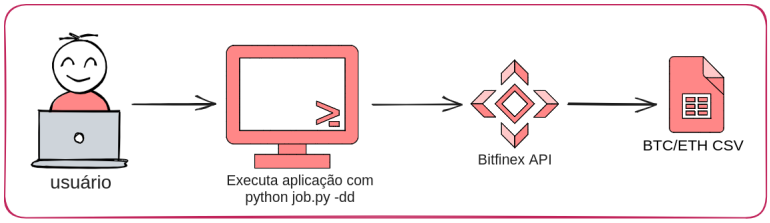


Figura 9: Obtenção dos dados

Fonte: Autor (2023)

### 5.3.3 Incialização do Modelo

Esta seção apresenta uma explicação detalhada da etapa inicial da execução do modelo desenvolvido neste trabalho. A Figura 10 ilustra o processo .

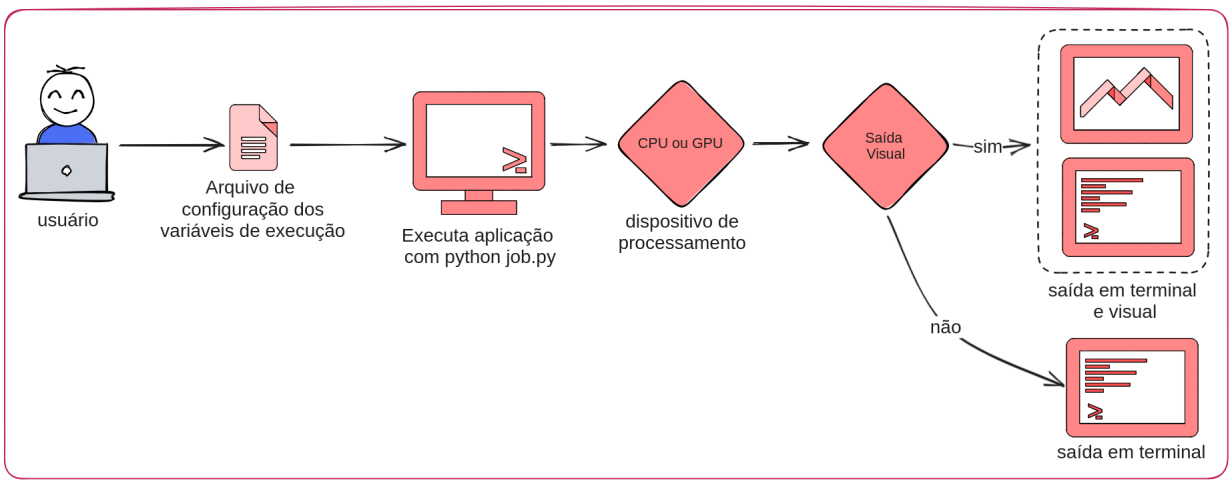


Figura 10: Execução do modelo

Fonte: Autor (2023)

A execução da aplicação é realizada em um ambiente de terminal Linux no diretório do repositório correspondente. Durante a execução, o usuário pode especificar o dispositivo de processamento e o tipo de saída desejados. Por padrão, a execução é configurada para ocorrer em um dispositivo GPU e sem saída visual, otimizando assim o aprendizado mais rápido do modelo. É importante observar que o programa inicializa variáveis de ambiente a partir do arquivo `.env`.

Para maior clareza, aqui estão as variáveis de ambiente passadas em uma das execuções conforme apresentado no Código 5.2.

```
STOCK_NAME=BTCUSD_1h
REPLAY_CAPACITY=1e6
GAMMA=0.99
BATCH_SIZE=4096
NN_ARCHITECTURE=(64,64)
NN_ACTIVATION=relu
NN_LEARNING_RATE=0.0001
NN_L2_REG=1e-6
NN_OPTIMIZER=Adam
NN_TAU=100
MODEL=Double Deep Q-Learning Network
COMMENT=An agent to learn how to negotiate stocks
TRAIN_EPISODES=100
EPISODE_STEPS=240
INITIAL_BALANCE=1000
```

Código 5.2: Variáveis de ambiente no .env

As variáveis de ambiente, definidas no arquivo `.env`, desempenham um papel crucial na configuração inicial do modelo. Essas variáveis são essenciais para ajustar os parâmetros iniciais que auxiliam na criação e configuração do modelo, e são conforme apresentados na Subseção 4.4.6.

Para permitir flexibilidade na configuração do modelo, foram desenvolvidas algumas variações. Além da configuração padrão que utiliza GPU e não possui saída visual, os usuários podem optar por processamento via CPU com saída visual. Isso é realizado através dos comandos exemplificados no Código 5.3.

Deste modo, em caso de necessidade de configuração adiciona quanto ao processamento padrão e o processamento via CPU, têm-se as seguintes variações descritas no código 5.3

```
# Processamento realizado com GPU e sem sa da visual (padr o)
python job.py
# Processamento realizado via CPU (--processing_device=CPU) com
    sa da visual (-v)
python job.py --processing_device=CPU -v
```

Código 5.3: Formas de realizar o início da execução do modelo

### 5.3.4 Visão Geral do Modelo

Esta seção oferece uma visão panorâmica do modelo, ampliando o resultado discutido na Subseção 5.3.3. A Figura 11 fornece uma representação visual das etapas críticas envolvidas no funcionamento do modelo.

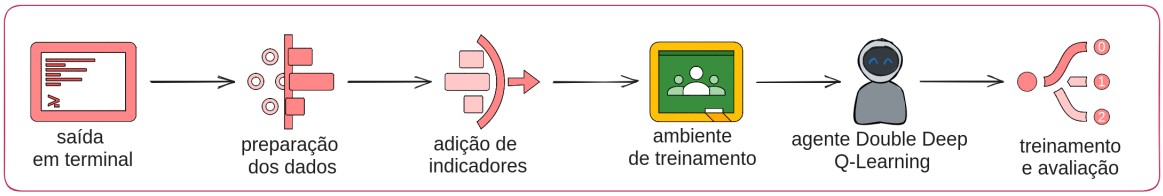


Figura 11: Visão geral do processo de treinamento

Fonte: Autor (2023)

A primeira etapa crucial nesse processo é a leitura dos dados previamente armazenados em formato CSV, seguida de um rigoroso processo de tratamento. Durante o tratamento, valores nulos e outras inconsistências são minuciosamente eliminados, preparando assim os dados para a próxima etapa do fluxo.

Em seguida, o modelo passa por um estágio de adição de indicadores e construção dos dados de treinamento. Nessa etapa os dados são enriquecidos com informações relevantes que alimentam o processo de treinamento. A combinação de indicadores aprimora a capacidade do modelo de aprender e tomar decisões informadas durante as simulações de negociação.

O ambiente de negociação, detalhadamente explorado na Subseção 5.3.5, é responsável pela simulação das negociações e interações entre o agente e o mercado. Ele atua como um cenário virtual onde as ações realizadas pelo agente no ambiente possibilitam o calculo das recompensas e métricas relevantes em cada passo. Essa simulação permite que o agente aprenda com a experiência durante o treinamento do modelo.

O agente, cujo treinamento é minuciosamente explicado na Subseção 5.3.6, é o componente central do modelo. Ele passa por um processo de aprendizado, executando ações no ambiente de negociação e recebendo feedback contínuo. Esse processo de aprendizado é vital para que o agente desenvolva estratégias de negociação eficazes e tome decisões informadas à medida que interage com o mercado.

### 5.3.5 Ambiente de Negociação

O ambiente de treinamento concebido para o modelo é representado na Figura 12. Este ambiente é inicializado com os dados adquiridos através da cotação dos ativos e indicadores calculados. Além disso, são definidos valores iniciais para o balanço que o agente usará para efetuar as transações, bem como para o patrimônio líquido e outras heurísticas empregadas durante a escolha do intervalo de dados a serem utilizados no treinamento.

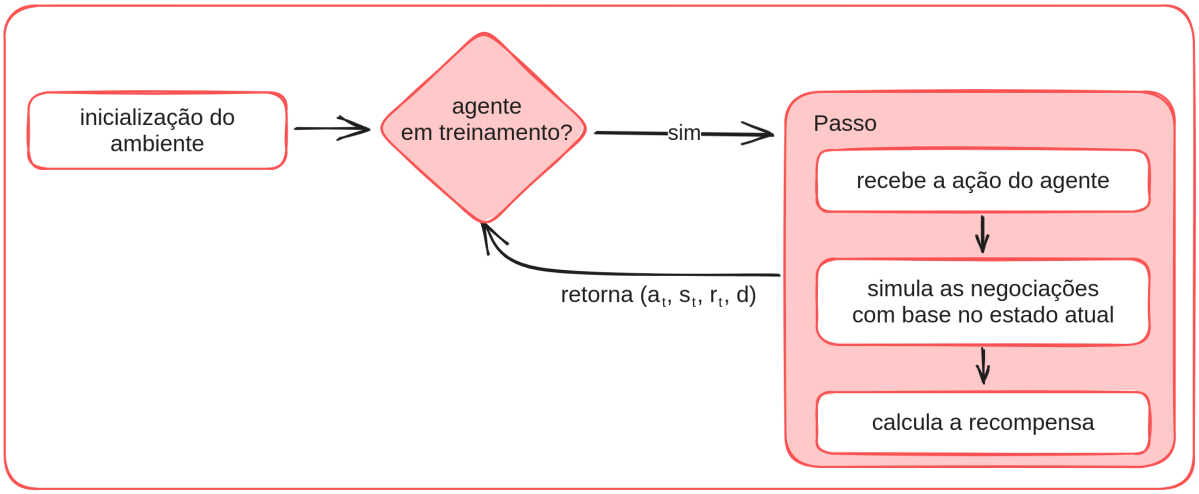


Figura 12: Ambiente de Treinamento

Fonte: Autor (2023)

Durante o período de treinamento, o ambiente é periodicamente invocado pelo agente para simular as negociações. Nesse sentido, o ambiente recebe a ação tomada em um estado específico e, com base nessa ação, calcula a quantidade de ativos que podem ser negociados, o saldo disponível e o patrimônio líquido. É importante destacar que, em conformidade com as regras de negociação estabelecidas, o agente não pode comprar ou vender ativos sem possuir os recursos necessários para honrar a ação. Adicionalmente, a venda a descoberto não é permitida, sendo, nesse caso, a ação recomendada a retenção do ativo. Por fim, a recompensa pela ação tomada é calculada e retornada juntamente com outras métricas.

### 5.3.6 Agente e processo de treinamento

O funcionamento do agente implementado no modelo engloba uma fase de treinamento, a qual faz uso da implementação do ambiente de treinamento detalhado na Subseção 5.3.5.

O agente, ilustrado na Figura 13, emprega o aprendizado por meio da técnica de *Double Deep Q-Learning*.

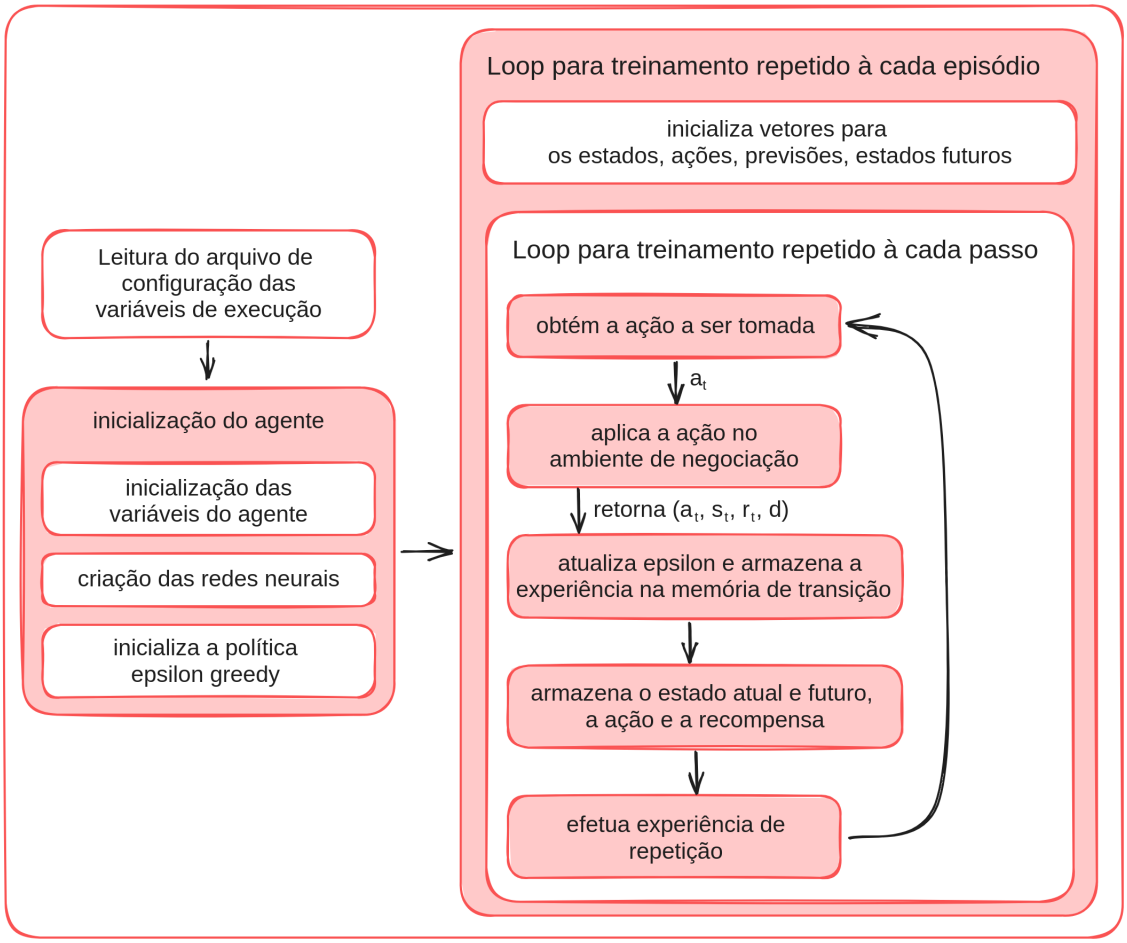


Figura 13: O agente e o processo de treinamento

Fonte: Autor (2023)

O processo do nosso agente inicia com a leitura de um arquivo contendo parâmetros iniciais, conforme apresentado no Código 5.2. A partir disso, são inicializadas as variáveis necessárias para o funcionamento do agente.

A rede neural utilizada pelo agente é configurada com base nos parâmetros de entrada, camadas ocultas, saída desejada para o espaço de ações, taxa de aprendizado, técnicas de regularização e otimizadores. A segunda rede, conhecida como rede alvo, é uma cópia da rede da primeira e é conhecida por rede online.

A etapa final na inicialização do agente envolve a configuração do método $\epsilon$-*greedy*, responsável por equilibrar a exploração e exploração, para mais detalhes em 2.9.2

Outro componente crucial no processo de aprendizado do agente são os laços de repetição utilizados pelo processo de aprendizado, sendo eles: por episódio e por passo dentro de cada episódio. Nesse contexto, o agente escolhe uma ação inicialmente de forma ale-

atória, mas à medida que o tempo avança, torna-se cada vez mais dependente da rede neural. A ação selecionada para o estado atual é enviada para o ambiente de negociação, que retorna a ação validada, o estado atual, a recompensa e indica se o treinamento deve continuar.

A cada passo, a experiência é armazenada em uma memória de transição e o valor de $\epsilon$ é atualizado. Por fim, o agente prossegue com o processo de aprendizado, fazendo uso das duas redes, conforme detalhado na Seção 4.7.

## 5.4 Geração de Resultados Gráficos

A implementação do modelo incorpora uma componente visual que permite a observação das variações nas cotações dos ativos durante o período analisado. Nesta seção, apresentamos uma descrição concisa da saída visual, seguida de uma análise das principais características visuais e de como essas contribuem para uma compreensão mais profunda do desempenho do modelo.

A Figura 14 ilustra a saída visual, que compreende as oscilações nos preços do ativo em análise, o comportamento do patrimônio líquido do agente, as interações executadas e a representação gráfica dos indicadores-chave.

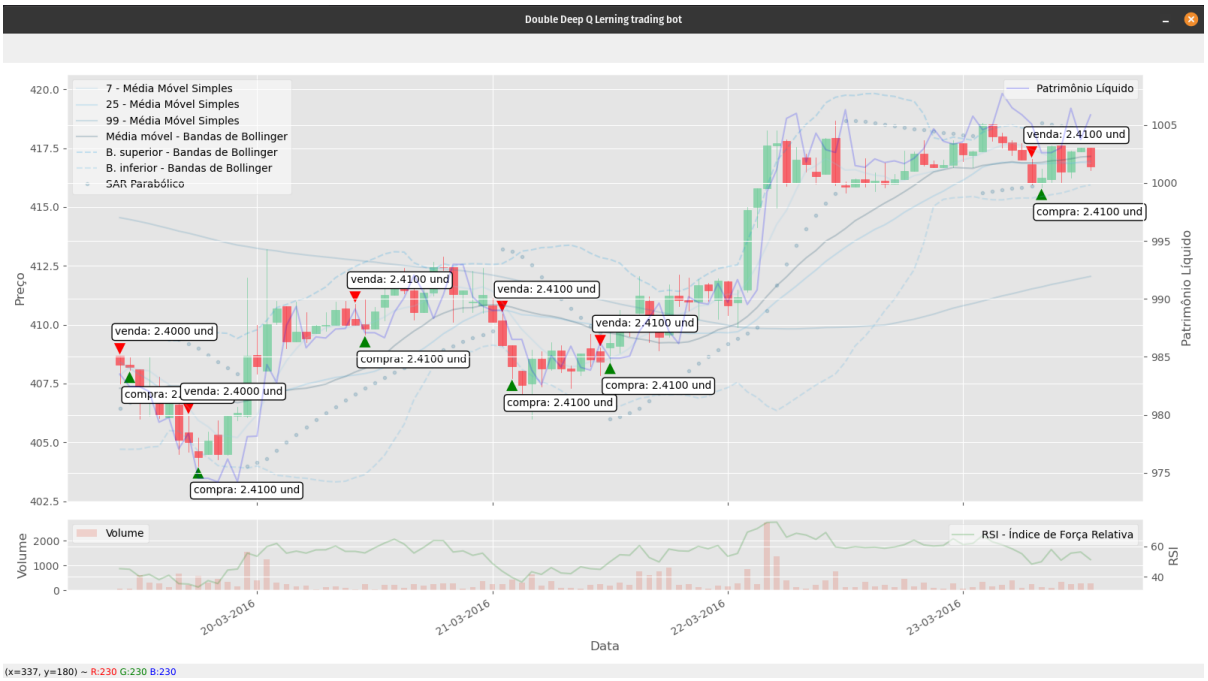


Figura 14: Resultados Gráficos

Fonte: Autor (2023)

Adicionalmente, nesta visualização, são disponibilizados os indicadores média móvel (com janelas temporais de 7, 25 e 99 períodos), as bandas de Bollinger, o SAR Parabólico e o Índice de Força Relativa (IFR), conforme anteriormente discutidos na seção 4.3.

A finalidade da geração visual reside na facilitação da compreensão do modelo e no suporte ao processo de identificação e correção de erros durante o desenvolvimento de melhorias nos processos de tomada de decisão do agente.

Para implementar a componente visual, foi adotada a biblioteca de visão computacional amplamente reconhecida, denominada OpenCV, a qual é disponibilizada para a linguagem de programação Python. O OpenCV foi instrumental na criação do ambiente visual, o quadro onde os gráficos foram são colocados sequencialmente para gerar a animação, enquanto os gráficos foram gerados com o auxílio da biblioteca Matplotlib, também em Python. Inicialmente, uma janela é gerada exibindo os pontos iniciais da série temporal. Subsequentemente, essa janela é atualizada em tempo real, a cada interação entre o agente e o ambiente, proporcionando, dessa forma, um acompanhamento em tempo real das interações realizadas pelo agente.

Essa abordagem resulta na criação de uma animação que possibilita o acompanhamento das interações executadas pelo agente bem como as oscilações do mercado e seus impactos nos indicadores, enriquecendo significativamente a experiência de análise com base nas considerações apresentadas no decorrer da seção 4.3, permitindo um feedback em tempo hábil.

## 5.5 Apresentação e discussão dos resultados obtidos

Esta seção conclui a parte de resultados deste trabalho e tem como objetivo apresentar os resultados obtidos durante a execução do modelo desenvolvido. Foram realizadas várias simulações com diferentes combinações de camadas ocultas na rede neural. Na Tabela 5.1, a coluna intitulada "Configuração da Rede"refere-se à arquitetura da rede neural utilizadas nas simulações. Conforme detalhado na coluna "Detalhamento", a configuração inclui 16 neurônios na camada de entrada, a quantidade indicada nas camadas ocultas e 3 neurônios na camada de saída (o mesmo padrão é seguido nas demais tabelas que apresentam a coluna "Configuração da Rede").

Durante o treinamento do modelo, uma série de parâmetros críticos foi cuidadosamente selecionados para otimizar o desempenho do agente. A Tabela 5.2 apresenta esses parâmetros, oferecendo uma visão abrangente das configurações específicas utilizadas.

Tabela 5.1: Diferentes combinações de número de camadas ocultas na rede neural

| **Configuração da Rede** | **Detalhamento** |
|---|---|
| 64 e 64 | Rede neural com 16 neurônios na camada de entrada, duas camadas ocultas de 64 neurônios cada e 3 neurônios na camada de saída |
| 128 e 128 | Rede neural com 16 neurônios na camada de entrada, com duas camadas ocultas de 128 neurônios cada e 3 neurônios na camada de saída |
| 64, 128 e 64 | Rede neural com 16 neurônios na camada de entrada, com três camadas ocultas de 64, 128 e 64 neurônios e 3 neurônios na camada de saída |
| 128, 256 e 128 | Rede neural com 16 neurônios na camada de entrada, com três camadas ocultas de 128, 256 e 128 neurônios e 3 neurônios na camada de saída |

Dentre os aspectos destacados, incluem-se a capacidade da memória de repetição do treinamento, o fator de desconto aplicado às recompensas futuras, o tamanho do batch de treinamento e a arquitetura das camadas ocultas da rede neural, cujos detalhes podem ser encontrados na Tabela 5.1. A função de ativação ReLU, a taxa de aprendizagem, a taxa de regularização L2, o otimizador Adam, a frequência de atualização da rede alvo e outros parâmetros foram escolhidos para proporcionar um ambiente de treinamento eficaz. Essas escolhas são fundamentais para moldar o comportamento do agente durante as 100 iterações de treinamento, cada uma com 240 passos, em busca de maximizar o patrimônio líquido, partindo de um capital inicial de 1000 dólares.

Através de extensas análises e simulações, este estudo procurou identificar as configurações mais eficazes da rede neural e avaliar seu impacto nas decisões de negociação. Os resultados obtidos fornecem insights significativos sobre o desempenho do modelo, contribuindo para o avanço do conhecimento na área de algoritmos de negociação.

### 5.5.1 Comparação CPU vs GPU

Na busca por otimização do processo de treinamento do modelo, a escolha entre processamento realizado via CPU (Unidade Central de Processamento) ou GPU (Unidade de Processamento Gráfico) desempenha um papel crucial. Este estudo realizou uma série de testes para avaliar o tempo de processamento e o desempenho do patrimônio líquido com base em diferentes unidades de processamento. A decisão de realizar esses testes se justifica pela necessidade de identificar a configuração mais eficaz da rede neural adotada pelo modelo, uma vez que o tempo de processamento é diretamente afetado pela complexidade da arquitetura da rede.

Tabela 5.2: Parâmetros do Treinamento

| **Parâmetro** | **Valor** |
|---|---|
| Nome do criptoativo | Bitcoin |
| Capacidade da memória de repetição do treinamento | $1 \times 10^6$ |
| Fator de Desconto ($\gamma$) | 0.99, uma variação do apresentado em [2] |
| Tamanho do Batch de Treinamento | 4096 |
| Configuração da Rede | Varia conforme os parâmetros apresentados na tabela 5.1 |
| Função de Ativação da Rede | ReLU, conforme apresentado em [29] |
| Taxa de Aprendizagem da Rede | $1 \times 10^{-4}$, conforme apresentado em [29] |
| Taxa de Regularização L2 da Rede | $1 \times 10^{-6}$ uma variação do apresentado em [29] |
| Otimizador da Rede | Adam, , conforme apresentado em [6] |
| Frequência de Atualização da Rede Alvo ($\tau$) | 100 |
| Nome do Modelo | Double Deep Q-Learning Network |
| Comentário | Agente treinado para aprender a negociar ações |
| Quantidade de Episódios durante o Treinamento | 100 |
| Iterações por episódio (Passos) | 240 |
| Capital Inicial | 1000 |

Ao comparar CPU e GPU, é fundamental entender suas características distintas. A CPU, comumente utilizada para tarefas gerais, destaca-se pela capacidade de lidar com uma variedade de operações, sendo eficaz em situações que demandam flexibilidade. Por outro lado, a GPU, projetada para processamento paralelo e tarefas intensivas, é especialmente eficiente em lidar com grandes volumes de dados simultaneamente, tornando-a ideal para cálculos exigentes e, preferencialmente não dependentes temporalmente, como os realizados em treinamentos de redes neurais.

Portanto, a escolha entre CPU e GPU para o treinamento do modelo não apenas influencia o tempo de processamento, mas também impacta diretamente no desempenho e na eficácia do processo. A análise das características específicas de cada unidade de processamento é necessária para tomar decisões informadas sobre a configuração mais adequada às demandas do modelo e do problema em questão.

Os testes iniciais foram conduzidos ao longo dos 100 primeiros episódios, com a aleatoriedade controlada por meio de um seed para garantir a consistência das análises. Conforme ilustrado na Figura 15, pode-se observar que o processamento usando CPU é substancialmente mais rápido. Isso é condizente com a simplicidade da rede neural e a

dependência sequencial dos dados, uma vez que cada observação posterior depende das anteriores para análise.

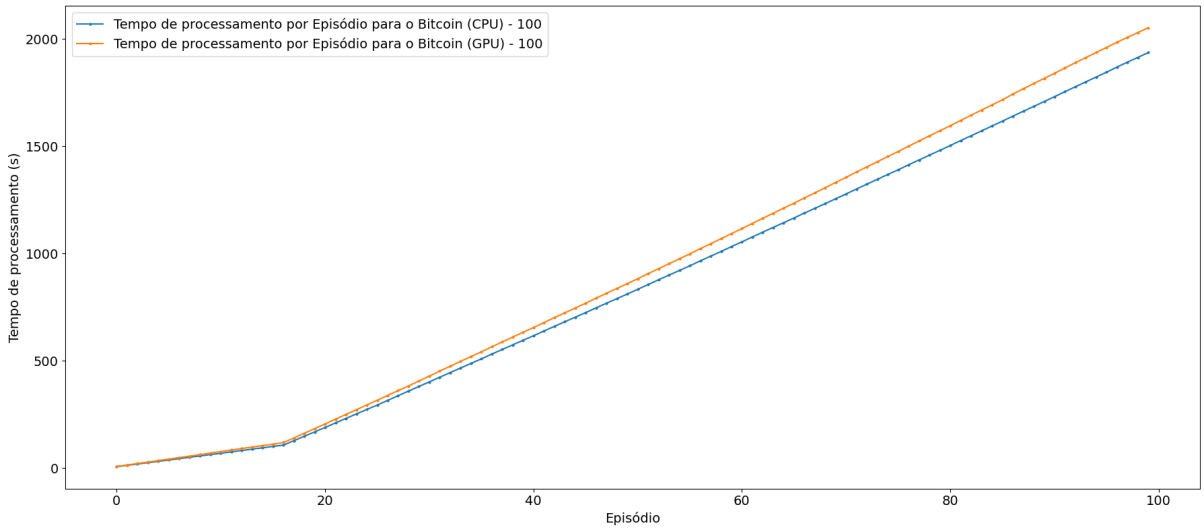


Figura 15: Comparativo do Tempo de processamento por Rede e Unidade de Processamento

Fonte: Autor (2023)

Além disso, é importante notar que a diferença de tempo entre os processamentos não é significativa no escopo da análise, como demonstrado na Tabela 5.3. Pode-se considerar que, nas tabelas aos valores de entrada e saída seguem imutáveis e a coluna Configuração da Rede indica as camadas ocultas da rede.

Tabela 5.3: Tempo de Processamento para o Bitcoin (100 episódios)

| **Configuração da Rede** | **CPU (s)** | **CPU (min)** | **GPU (s)** | **GPU (min)** |
|---|---|---|---|---|
| 64 e 64 | 1804.78 | 30.08 | 1927.41 | 32.12 |
| 128 e 128 | 1976.44 | 32.94 | 2071.07 | 34.52 |
| 64, 128 e 64 | 1819.67 | 30.33 | 2061.60 | 34.36 |
| 128, 256 e 128 | 2148.55 | 35.81 | 2153.16 | 35.89 |

Esses resultados sugerem que a escolha entre CPU e GPU pode depender de outros fatores, além do tempo de processamento, como a complexidade da rede neural e o requisito de otimização dos recursos computacionais. Durante a análise dos resultados, observou-se que agente implementado no modelo apresentou um melhor desempenho quando treinado usando a GPU. Isso pode ser atribuído à otimização da representação dos números de ponto flutuante, que são codificados em 16 bits, em contraste com os 32 bits da CPU, reduzindo o viés durante o treinamento.

O impacto dessa escolha também se reflete no desempenho do patrimônio líquido médio por episódio. A Tabela 5.4 mostra os resultados do patrimônio líquido médio para as configurações de rede, distinguindo entre CPU e GPU. Observa-se que, embora as diferenças não sejam substanciais, a utilização da GPU tende a resultar em um desempenho ligeiramente superior

Tabela 5.4: Patrimônio Líquido Médio para o Bitcoin (100 episódios)

| **Configuração da Rede** | **CPU (US$)** | **GPU (US$)** |
|---|---|---|
| 64 e 64 | 1023.95 | 1025.36 |
| 128 e 128 | 1012.12 | 1019.28 |
| 64, 128 e 64 | 982.03 | 990.67 |
| 128, 256 e 128 | 1009.71 | 1027.04 |

### 5.5.2 Avaliação do Modelo com Variação no Número de Episódios

Para a etapa de treinamento de longo prazo, foram selecionadas as duas redes que melhor se comportaram na análise exploratória realizada durante 100 episódios de treinamento para o criptoativo Bitcoin apresentada na seção 5.5.1. Essas redes demonstraram um desempenho promissor durante a fase inicial de treinamento. As configurações de rede escolhidas para o treinamento final, que compreendeu 1000 episódios, foram as seguintes:

- **Rede 1:** Composta por duas camadas ocultas, cada uma contendo 64 neurônios.
- **Rede 2:** Composta por três camadas ocultas, contendo 128, 256 e 128 neurônios.

O experimento final dessas redes também foi replicado para o ativo Ethereum, permitindo uma avaliação mais abrangente do desempenho do modelo em diferentes ativos.

Essa abordagem de treinamento estendido com mais episódios visa fornecer uma visão mais sólida do potencial do modelo em situações de mercado de longo prazo e seu desempenho em diferentes ativos. Os resultados desses treinamentos mais extensos serão posteriormente analisados e discutidos, contribuindo para uma avaliação completa do modelo proposto.

A avaliação do modelo com variação no número de episódios revelou insights valiosos sobre seu comportamento em cenários de treinamento de curto a longo prazo. Assim, dois aspectos foram avaliados: o patrimônio líquido médio até o ponto avaliado e a taxa de sucesso, que representa o percentual de episódios que foram finalizados com um patrimônio maior que o investimento inicial.

A Figura 16 representa a variação do patrimonio líquido médio ao término de cada episódio.

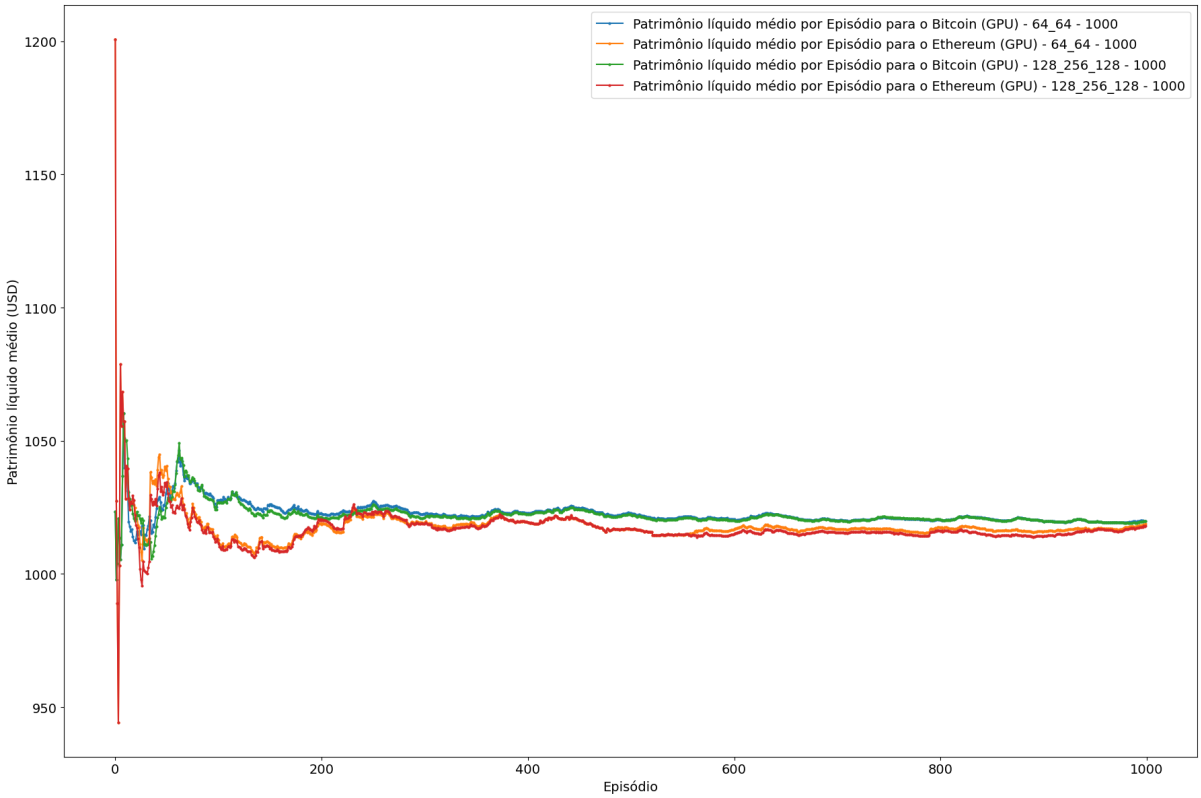


Figura 16: Comparativo do Tempo de processamento por Rede e Unidade de Processamento

Fonte: Autor (2023)

Para uma melhor compreensão têm-se duas tabelas, apresenta-se duas tabelas: Tabela 5.5 representa a Rede 1 e a Tabela 5.6 representa a Rede 2. As tabelas exibem a média do patrimônio líquido durante o treinamento até o episódio. Isso permite observar uma variação inicial enquanto o modelo se ajusta aos dados. Posteriormente, nota-se que ambos variam menos com o tempo, mantendo um aumento de 2% no patrimônio.

Tabela 5.5: Patrimônio Líquido Médio para Bitcoin e Ethereum (Rede 1)

| **Episódios** | **Bitcoin (US$)** | **Ethereum (US$)** |
|---|---|---|
| 50 | 1020.33 | 1031.40 |
| 100 | 1026.96 | 1027.49 |
| 200 | 1026.24 | 1019.83 |
| 400 | 1024.78 | 1019.74 |
| 800 | 1023.20 | 1018.43 |
| 1000 | 1022.60 | 1018.12 |

Tabela 5.6: Patrimônio Líquido Médio para Bitcoin e Ethereum (Rede 2)

| **Episódios** | **Bitcoin (US$)** | **Ethereum (US$)** |
|---|---|---|
| 50 | 1021.05 | 1026.55 |
| 100 | 1027.05 | 1023.78 |
| 200 | 1025.44 | 1017.53 |
| 400 | 1023.90 | 1018.63 |
| 800 | 1022.60 | 1017.45 |
| 1000 | 1022.11 | 1017.06 |

Do ponto de vista da taxa de sucesso, representado pela Figura 17 é possível observar um comportamento similar, onde inicialmente nota-se um ajuste aos dados e posteriormente pode-se notar que o modelo permanece próximo de 55% de sucesso.

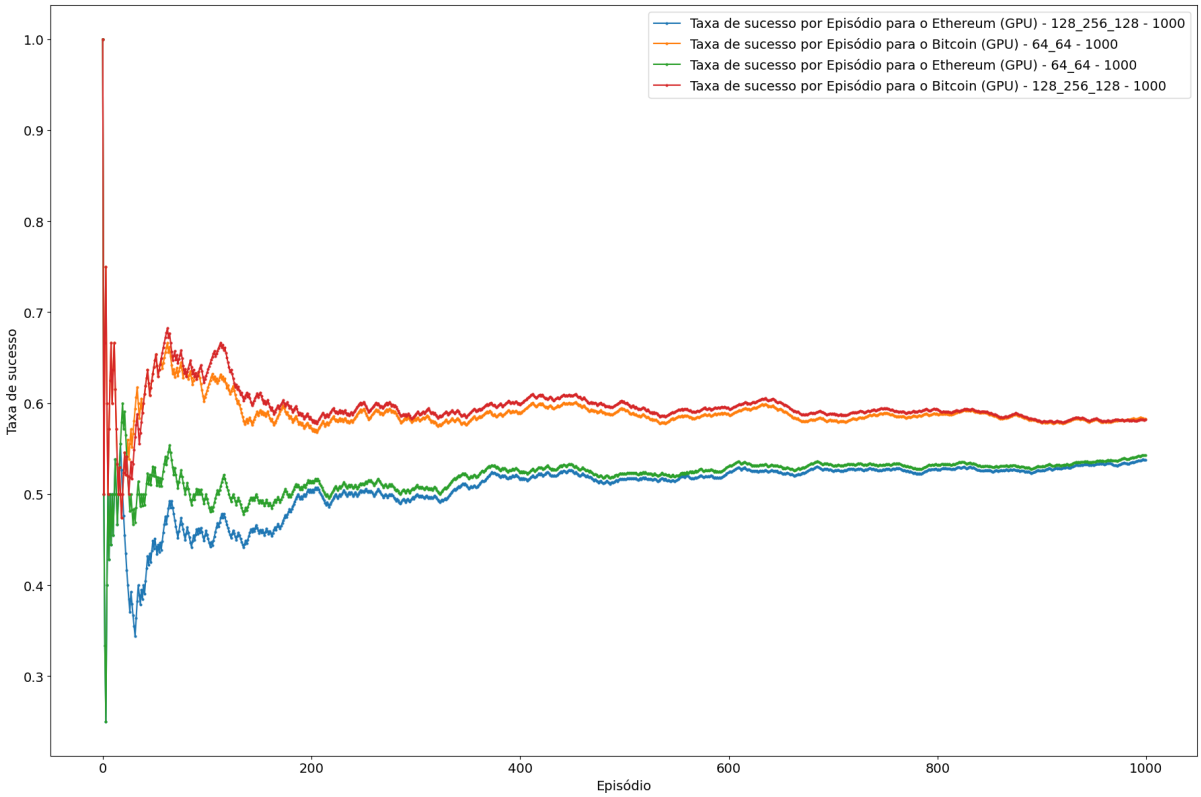


Figura 17: Comparativo da Taxa de sucesso por Rede e Unidade de Processamento

Fonte: Autor (2023)

Portanto, é possivel afirmar que:

- **Desempenho Inicial (50 a 200 Episódios):** Durante os estágios iniciais do treinamento (episódios 50 a 200), ambas as redes apresentaram um comportamento típico de ajuste do modelo, onde pode-se notar aumento do patrimônio líquido médio

Tabela 5.7: Taxa de Sucesso (Rede 1)

| **Episódios** | **Bitcoin (%)** | **Ethereum (%)** |
|---|---|---|
| 50 | 64 | 52 |
| 100 | 61 | 50 |
| 200 | 57.5 | 51.5 |
| 400 | 59 | 52.5 |
| 800 | 58.88 | 53.25 |
| 1000 | 58.30 | 54.30 |

Tabela 5.8: Taxa de Sucesso (Rede 2)

| **Episódios** | **Bitcoin (%)** | **Ethereum (%)** |
|---|---|---|
| 50 | 64 | 44 |
| 100 | 63 | 46 |
| 200 | 58.5 | 50.5 |
| 400 | 60 | 51.75 |
| 800 | 59.38 | 52.75 |
| 1000 | 58.2 | 53.8 |

a cada episódio e o ajuste aos dados. Esse aumento pode ser atribuído à capacidade do modelo de aprender e se adaptar a padrões de mercado iniciais.

- **Estabilidade e Consistência (200 a 500 Episódios):** Nos episódios 200 a 500, observou-se uma fase de estabilização do desempenho. O patrimônio líquido médio por episódio manteve-se relativamente consistente, indicando que o modelo estava se adaptando aos movimentos de mercado de forma mais estável. Isso sugere que, após um período inicial de aprendizado, o modelo conseguiu manter um desempenho sólido.

- **Desempenho de Longo Prazo (1000 Episódios):** Ao atingir os 1000 episódios, ambas as redes mostraram seu desempenho de longo prazo. O patrimônio líquido médio por episódio atingiu estabilidade, destacando a capacidade do modelo de manter um bom desempenho ao longo do tempo, apesar de carecer de ajustes para que o patamar de estabilidade ofereça um maior retorno.

# Capítulo 6

# Conclusões e Sugestões para Trabalhos Futuros

## 6.1 Conclusões

A consecução dos objetivos específicos delineados neste trabalho revela uma abordagem cuidadosa no desenvolvimento de operações financeiras autônomas, embasadas em escolhas estratégicas e implementação eficaz.

Em relação à avaliação e seleção do mercado financeiro para a extração de dados, optou-se por criptoativos. A escolha foi realizada devido a acessibilidade e riqueza de dados, o que estabeleceu um cenário propício para análises e treinamento do agente autônomo.

A escolha do Double Deep Q-Learning (DDQL) como agente autônomo, foi fundamentada na necessidade de um modelo capaz de se adaptar às oscilações do mercado, e se demonstrou congruente com os requisitos estabelecidos. Uma vez que devido a robustez e a flexibilidade do DDQL tornam o agente capaz de ajustar dinamicamente as estratégias aprendidas.

A proposta e implementação do agente de aprendizado por reforço, focado no aumento do patrimônio líquido, adotou o DDQL como protagonista. Entretanto, é relevante observar que o incremento no patrimônio líquido foi, de fato, relativamente modesto. Essa constatação aponta para uma área de oportunidade e destaca a necessidade de aprimoramentos significativos em trabalhos futuros.

A capacitação do agente para aprender e adaptar suas estratégias durante sua atuação no mercado financeiro foi evidenciada na prática, corroborando a eficácia e viabilidade do

modelo proposto. No entanto, a observação do aumento moderado no patrimônio líquido indica claramente a demanda por melhorias substanciais para maximizar os ganhos.

A habilidade intrínseca do agente DDQL de aprender enquanto opera no mercado é um aspecto positivo, mas o tamanho do incremento no patrimônio líquido sugere que o modelo pode se beneficiar consideravelmente de ajustes e otimizações adicionais.

A análise abrangente dos resultados, ao considerar o aumento do patrimônio líquido, evidencia de forma clara a necessidade de aprimoramentos em futuras pesquisas.

## 6.2 Sugestões para Trabalhos Futuros

Neste trabalho, na parte de modelagem, realizamos a construção de um agente DDQL de negociação com base nos conceitos discutidos no capítulo 2, mostrando ser possível a realização de negociações lucrativas por meio do agente proposto.

Para a parte visual, uma animação onde é possível realizar o acompanhamento e até mesmo avaliar o comportamento do agente também foi implementada.

Com relação aos trabalhos futuros, dentre as oportunidades de melhoria têm-se a implementação de refinamentos nas estratégias do agente, conforme sugerido por [10], que propõe a incorporação do Sharpe Ratio no sistema de recompensas do agente. Além disso, existe a possibilidade de incorporar um segundo modelo para determinar a quantidade a ser investida em cada ativo, visando reduzir o risco assumido pelo agente e melhorar o desempenho do modelo durante as negociações de maior risco [83]. Este estudo, ao destacar as realizações alcançadas, proporciona uma base sólida para futuras investigações, as quais visam elevar significativamente o patamar de eficácia do modelo na área de operações financeiras autônomas. Por fim, pode-se ainda implementar melhorias na parte visual para permitir a configuração do modelo sem que seja preciso codificar os parâmetros.

# Referências


[1] REED, S. et al. A generalist agent. 5 2022. Disponível em: <http://arxiv.org/abs/2205.06175>.

[2] PARK, H.; SIM, M. K.; CHOI, D. G. An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, Elsevier Ltd, v. 158, 11 2020. ISSN 09574174.

[3] ECONOMIST, T. *The stockmarket is now run by computers, algorithms and passive managers | The Economist*. 2019. Disponível em: <https://www.economist.com/briefing/2019/10/05/the-stockmarket-is-now-run-by-computers-algorithms-and-passive-managers>.

[4] CHAN, E. P. *Quantitative Trading*. 2021.

[5] RUNDO, F. Deep lstm with reinforcement learning layer for financial trend prediction in fx high frequency trading systems. *Applied Sciences (Switzerland)*, v. 9, n. 20, 2019. ISSN 20763417.

[6] ZHANG, Z.; ZOHREN, S.; ROBERTS, S. Deep reinforcement learning for trading. *arXiv*, i, 2019. ISSN 23318422.

[7] NAZÁRIO, R. T. F. et al. *A literature review of technical analysis on stock markets*. [S.l.]: Elsevier B.V., 2017. 115-126 p.

[8] BAKER, M.; WURGLER, J. Investor sentiment in the stock market. *Journal of Economic Perspectives*, v. 21, n. 2, p. 129–152, June 2007. Disponível em: <https://www.aeaweb.org/articles?id=10.1257/jep.21.2.129>.

[9] MOODY, J. et al. *Performance Functions and Reinforcement Learning for Trading Systems and Portfolios*. [S.l.]: John Wiley  Sons, 1998. 441-470 p.

[10] LUCARELLI, G.; BORROTTI, M. A deep reinforcement learning approach for automated cryptocurrency trading. In: . [S.l.]: Springer New York LLC, 2019. v. 559, p. 247–258. ISBN 9783030198220. ISSN 18684238.

[11] SHI, Y. et al. Stock trading rule discovery with double deep q-network. *Applied Soft Computing*, Elsevier Ltd, v. 107, 8 2021. ISSN 15684946.

[12] NAQA, I. E.; MURPHY, M. J. Machine learning in radiation oncology. *Machine Learning in Radiation Oncology*, p. 3–11, 2015.

[13] MATSUO, Y. et al. Deep learning, reinforcement learning, and world models. *Neural Networks*, v. 152, p. 267–275, 8 2022. ISSN 08936080. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0893608022001150>.

[14] SILVER, D. et al. *Reward is enough*. [S.l.]: Elsevier B.V., 2021.

[15] CHAKOLE, J. B. et al. A q-learning agent for automated trading in equity stock markets. *Expert Systems with Applications*, Elsevier Ltd, v. 163, 1 2021. ISSN 09574174.

[16] CARTA, S. et al. Multi-dqn: An ensemble of deep q-learning agents for stock market forecasting. *Expert Systems with Applications*, Elsevier Ltd, v. 164, 2 2021. ISSN 09574174.

[17] OTTERLO, M. V. *Markov Decision Processes: Concepts and Algorithms*. 2009. 1–23 p.

[18] MARKOV, A. A. Classical text in translation: An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains. *Science in Context*, v. 19, n. 4, p. 591–600, 2006. ISSN 02698897.

[19] PELLEGRINI, J.; WAINER, J. Processos de Decisão de Markov: um tutorial. *Revista de Informática Teórica e Aplicada*, v. 14, n. 2, p. 133–179, 2007. ISSN 01034308.

[20] SUTTON, R. S. et al. *Policy Gradient Methods for Reinforcement Learning with Function Approximation **. [S.l.]: Bertsekas and Tsitsiklis, 1995.

[21] WHITE, D. J. A survey of applications of markov decision processes. *Journal of the Operational Research Society*, v. 44, n. 11, p. 1073–1096, 1993. ISSN 14769360.

[22] ARULKUMARAN, K. et al. A brief survey of deep reinforcement learning. 8 2017. Disponível em: <http://arxiv.org/abs/1708.05866 http://dx.doi.org/10.1109/MSP.2017.2743240>.

[23] LEE, J. W. et al. A multiagent approach to q-learning for daily stock trading. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, v. 37, p. 864–877, 11 2007. ISSN 10834427.

[24] BELLMAN, R. Dynamic programming and stochastic control processes. *Information and Control*, v. 1, n. 3, p. 228–239, 1958. ISSN 00199958.

[25] BELLMAN, R. *Some Problems in the Theory of Dynamic Programming*. 1954. 37-48 p.

[26] BELLMAN, R.; KALABA, R. On the Role of Dynamic Programming in Statistical Communication Theory. *IRE Transactions on Information Theory*, v. 3, n. 3, p. 197–203, 1957. ISSN 21682712.

[27] BARRETO, A. da M. S.; ANDERSON, C. W. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, v. 172, p. 454–482, 3 2008. ISSN 00043702.

[28] SINGH, S.; JAAKKOLA, T.; LITTMAN, M. L. *Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms*. 2000. 287-308 p.

[29] FUJIMOTO, S.; MEGER, D.; PRECUP, D. *Off-Policy Deep Reinforcement Learning without Exploration*. Disponível em: <https://github.com/sfujim/BCQ>.

[30] BHASIN, S.; JOHNSON, M.; DIXON, W. E. B. *A model-free robust policy iteration algorithm for optimal control of nonlinear systems.* IEEE, 2010. v. 2. 3060-3065 p. ISBN 9781424477463. Disponível em: <https://ieeexplore.ieee.org/document/5717295>.

[31] SUTTON, R. S.; BARTO, A. G. *Reinforcement learning : an introduction.* [S.l.: s.n.], 2018. 526 p. ISBN 9780262039246.

[32] SUTTON, R. S. *Learning to Predict by the Methods of Temporal Differences.*

[33] GLäSCHER, J. et al. States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, v. 66, p. 585–595, 5 2010. ISSN 08966273.

[34] DOYA, K. et al. Multiple model-based reinforcement learning. v. 1369, p. 1347–1369, 2002.

[35] WATKINS, C. J. C. H.; DAYAN, P. *Q-Learning.* 1992. 279-292 p.

[36] MAHARAJ, S.; POLSON, N.; TURK, A. Chess ai: Competing paradigms for machine intelligence. 9 2021. Disponível em: <http://arxiv.org/abs/2109.11602>.

[37] SILVER, D. et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 12 2017. Disponível em: <http://arxiv.org/abs/1712.01815>.

[38] HE, W.; ZHAO, W.; JIANG, Y. Application of q-learning and rbf network in chinese chess game system. In: . [S.l.]: IOP Publishing Ltd, 2019. v. 677. ISSN 1757899X.

[39] MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, v. 518, p. 529–533, 2 2015. ISSN 14764687.

[40] WU, X. et al. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, Elsevier Inc., v. 538, p. 142–158, 10 2020. ISSN 00200255.

[41] GUO, M.; LIU, Y.; MALEC, J. A new q-learning algorithm based on the metropolis criterion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, v. 34, p. 2140–2143, 10 2004. ISSN 10834419.

[42] YOGESWARAN, M.; PONNAMBALAM, S. G. Reinforcement learning: Exploration-exploitation dilemma in multi-agent foraging task. *OPSEARCH*, v. 49, p. 223–236, 9 2012. ISSN 00303887.

[43] KROEMER, O.; PETERS, J. *Active Exploration for Robot Parameter Selection in Episodic Reinforcement Learning.*

[44] REJEB, L.; GUESSOUM, Z.; HALLAH, R. M. . *LNAI 3690 - An Adaptive Approach for the Exploration-Exploitation Dilemma for Learning Agents.*

[45] WIEWIORA, E. *Efficient Exploration for Reinforcement Learning.* 2004.

[46] TOKIC, M. Adaptive $\epsilon$-greedy exploration in reinforcement learning based on value differences. In: DILLMANN, R. et al. (Ed.). *KI 2010: Advances in Artificial Intelligence.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 203–210. ISBN 978-3-642-16111-7.

[47] MIGNON, A. D. S.; ROCHA, R. L. D. A. D. An adaptive implementation of -greedy in reinforcement learning. In: . [S.l.]: Elsevier B.V., 2017. v. 109, p. 1146–1151. ISSN 18770509.

[48] WUNDER, M.; LITTMAN, M.; BABES, M. Classes of multiagent q-learning dynamics with -greedy exploration. *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, p. 1167–1174, 2010.

[49] TAN, F.; YAN, P.; GUAN, X. Deep reinforcement learning: From q-learning to deep q-learning. In: . [S.l.]: Springer Verlag, 2017. v. 10637 LNCS, p. 475–483. ISBN 9783319700922. ISSN 16113349.

[50] HASSELT, H. V.; GUEZ, A.; SILVER, D. *Deep Reinforcement Learning with Double Q-Learning*. 2016. Disponível em: <www.aaai.org>.

[51] SCHAUL, T. et al. Prioritized experience replay. 11 2015. Disponível em: <http://arxiv.org/abs/1511.05952>.

[52] NING, B.; LIN, F. H. T.; JAIMUNGAL, S. Double deep q-learning for optimal execution. 12 2018. Disponível em: <http://arxiv.org/abs/1812.06600>.

[53] ZHANG, Q. et al. A double deep q-learning model for energy-efficient edge scheduling. *IEEE Transactions on Services Computing*, Institute of Electrical and Electronics Engineers, v. 12, p. 739–749, 9 2019. ISSN 19391374.

[54] REINISCH, N. et al. Successful pass schedule design in open-die forging using double deep Q-learning. *Processes*, v. 9, n. 7, 2021. ISSN 22279717.

[55] FAN, J. et al. A theoretical analysis of deep q-learning. 1 2019. Disponível em: <http://arxiv.org/abs/1901.00137>.

[56] LEK, S.; PARK, Y. S. Multilayer perceptron. *Encyclopedia of Ecology, Five-Volume Set*, Academic Press, p. 2455–2462, 1 2008.

[57] JAIN A.K.; MAO, J. M. K. Artificial neural networks: a tutorial. p. 31–44, 1996. Disponível em: <https://ieeexplore.ieee.org/document/485891>.

[58] HOPFIELD, J. J. *Learning algorithms and probability distributions in feed-forward and feed-back networks*. 1987. 8429-8433 p.

[59] HAGAN, M. T.; MENHAJ, M. B. *Brief Papers Training Feedforward Networks with the Marquardt Algorithm*. 1994.

[60] MINSKY, M. L.; PAPERT, S. A. *Perceptrons: An Introduction to Computational Geometry*. 1969.

[61] BAUM, E. B. *On the Capabilities of Multilayer Perceptrons*. 1988. 193-215 p.

[62] RUMELHART DAVID E.; HINTON, G. E. W. R. J. Learning representations by back-propagating errors. 1986.

[63] CUN, Y. L. A theoretical framework for back-propagation. *Connectionist Models Summer School*, p. 21–28, 1988.

[64] HECHT-NIELSEN, R. *Theory of the Backpropagation Neural Network*. 1989.

[65] KARLIK, B.; OLGAC, A. V. *Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks*.

[66] AGOSTINELLI, F. et al. Learning activation functions to improve deep neural networks. 12 2014. Disponível em: <http://arxiv.org/abs/1412.6830>.

[67] DUBEY, A. K.; JAIN, V. Comparative study of convolution neural network's relu and leaky-relu activation functions. In: . [S.l.]: Springer Verlag, 2019. v. 553, p. 873–880. ISBN 9789811367717. ISSN 18761119.

[68] RASAMOELINA, A. D.; ADJAILIA, F.; SINčáK, P. *A Review of Activation Function for Artificial Neural Network*. [S.l.: s.n.], 2020. 23-25 p. ISBN 9781728131498.

[69] ANDRYCHOWICZ, M. et al. Learning to learn by gradient descent by gradient descent. In: LEE, D. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016. v. 29. Disponível em: <https://proceedings.neurips.cc/paper/2016/file/fb87582825f9d28a8d42c5e5e5e8b23d-Paper.pdf>.

[70] MITTAL, G. S. *Artificial neural network based process modeling*. [S.l.]: Elsevier, 2019. 525-531 p.

[71] JABBAR, H. K.; KHAN, R. Z. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). p. 163–172, 2015.

[72] LIU, Z. et al. Regularization matters in policy optimization - an empirical study on continuous control. In: *International Conference on Learning Representations*. [s.n.], 2021. Disponível em: <https://openreview.net/forum?id=yr1mzrH3IC>.

[73] HAHN, S.; CHOI, H. Understanding dropout as an optimization trick. *Neurocomputing*, Elsevier B.V., v. 398, p. 64–70, 7 2020. ISSN 18728286.

[74] HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. 7 2012. Disponível em: <http://arxiv.org/abs/1207.0580>.

[75] FISCHER, T. G.; FISCHER, T. Reinforcement learning in financial markets-a survey. v. 12, 2018. ISSN 1867-6707. Disponível em: <https://www.iwf.rw.fau.de/research/iwf-discussion-paper-series/>.

[76] EDWARDS, R. D.; MAGEE, J.; BASSETTI, W. H. C. *The technical approach to trading and investing*. [S.l.: s.n.], 2019. 13–24 p. ISBN 1574442929.

[77] NEELY, C. J. et al. Forecasting the equity risk premium: The role of technical indicators. *Management Science*, v. 60, n. 7, p. 1772–1791, 2014. ISSN 15265501.

[78] MONDAL, P.; SHIT, L.; GOSWAMI, S. Study of effectiveness of time series modeling (arima) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*, Academy and Industry Research Collaboration Center (AIRCC), v. 4, p. 13–29, 4 2014. ISSN 22310088.

[79] XIAO, L.; AYDEMIR, A. Volatility modelling and forecasting in finance. *Forecasting Volatility in the Financial Markets*, Butterworth-Heinemann, p. 1–45, 1 2007.

[80] CHAVARNAKUL, T.; ENKE, D. A hybrid stock trading system for intelligent technical analysis-based equivolume charting. *Neurocomputing*, Elsevier, v. 72, p. 3517–3528, 10 2009. ISSN 0925-2312.

[81] POWELL, N.; FOO, S. Y.; WEATHERSPOON, M. Supervised and unsupervised methods for stock trend forecasting. *Proceedings of the Annual Southeastern Symposium on System Theory*, p. 203–205, 2008.

[82] SAFFELL, J. M. e M. Reinforcement learning for trading. 1997.

[83] JEONG, G.; KIM, H. Y. Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, Elsevier Ltd, v. 117, p. 125–138, 3 2019. ISSN 09574174.

[84] BITFINEX. *The stockmarket is now run by computers, algorithms and passive managers | The Economist*. 2022. Disponível em: <https://docs.bitfinex.com/docs>.

[85] ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

[86] OLIVEIRA, D. de. *Desenvolvimento de modelo de inteligência artificial aplicado a day trading com criptoativos baseado em Double Deep Q-Learning*. [S.l.]: GitHub, 2023. https://github.com/douglasnacl/master.