UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FELIPE BRUNO MOREIRA NUNES

APRENDIZAGEM POR REFORÇO DIRETO APLICADO EM SÉRIES TEMPORAIS.

SOBRAL

2022

FELIPE BRUNO MOREIRA NUNES

APRENDIZAGEM POR REFORÇO DIRETO APLICADO EM SÉRIES TEMPORAIS.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. José Cláudio do Nascimento

SOBRAL

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

N925a Nunes, Felipe Bruno Moreira.
Aprendizagem por reforço direto aplicado em séries temporais / Felipe Bruno Moreira Nunes. – 2022.
45 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia Elétrica, Sobral, 2022.
Orientação: Prof. Dr. José Cláudio do Nascimento.

1. Aprendizagem por Reforço. 2. Aprendizagem Profunda. 3. Aprendizagem de Máquina . 4. Séries Temporais.

CDD 621.3

FELIPE BRUNO MOREIRA NUNES

APRENDIZAGEM POR REFORÇO DIRETO APLICADO EM SÉRIES TEMPORAIS.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Cláudio do Nascimento (Orientador)
Universidade Federal do Ceará (UFC)

Me. João Paulo Teófilo Rodrigues
Universidade Federal do Ceará (UFC)

Me. Francisco Leonardo Bezerra Martins
Universidade Federal do Ceará (UFC)

À minha família, que esteve ao meu lado em todos os momentos que eu precisei, até quando achava que não seria necessário.

AGRADECIMENTOS

Agradeço a Deus que esteve do meu lado em todos os momentos da minha vida.

Ao meu pai e à minha mãe que sempre me incentivaram a seguir meus sonhos e não desistir por mais complicada que estivessem as condições.

À minha família que sempre acreditaram em mim e fizeram ser quem eu sou hoje.

À Danyela que esteve ao meu lado em todos os momentos, tanto nos bons, quanto nos ruins desse percurso, e que nunca conseguirei retribuir por tudo.

Aos meus colegas, que me ajudaram em diversos momentos: Claudiene, Francisco Matheus, Ianna Kelly, Jadir Dias, Marcelo Estevão, Vinicius Souza, Werley e Wladia.

À Instituição UFC, por todos os ensinamentos aprendidos.

Ao Dr. José Cláudio do Nascimento, pela excelente orientação.

Aos professores participantes da banca examinadora Me. João Paulo Teófilo Rodrigues e Me. Francisco Leonardo Bezerra Martins pelo tempo, pelas valiosas colaborações e sugestões.

“O que sabemos é uma gota, o que ignoramos é um oceano.”

(Isaac Newton, 1697.)

RESUMO

Este trabalho apresenta o desenvolvimento de um modelo de rede neural utilizando aprendizagem por reforço direto, recorrente e profunda, além de realizar a obtenção e pré-tratamento dos dados. Para extrair os parâmetros de entrada da rede a partir dos dados brutos foram empregadas fórmulas, que também são utilizadas por seres humanos, para identificar padrões em séries temporais quando se trata de preço de ações, como *Price Rate of Change*. Os algoritmos para a construção do modelo aplicado neste trabalho são apresentados conforme a sequência de desenvolvimento. O trabalho tem como finalidade demonstrar a eficácia da utilização deste modelo em prever séries temporais complexas, como no caso da série de preços de ações. Para a sua construção foi aplicado o *framework Tensorflow* em conjunto com a linguagem de programação *Python*, estes foram escolhidos pela sua capacidade de serem utilizados em diversas plataformas, podendo assim serem empregados para inúmeras finalidades. Por fim são apresentados os resultados dos testes, onde alguns destes superaram o valor da ação em 300%, além de propor estratégias para otimizações futuras.

Palavras-chave: Aprendizagem por Reforço. Aprendizagem Profunda. Aprendizagem de Máquina. Séries Temporais.

ABSTRACT

This paper presents the development of a neural network model using direct, recurrent, and deep reinforcement learning, as well as obtaining and pre-processing the data. To extract the input parameters of the network from the raw data, formulas were used that are also used by humans to identify patterns in time series when it comes to stock market shares, such as the Rate of Price Change. The algorithms to build the model applied in this paper are presented according to the development sequence. The work aims to demonstrate the effectiveness of using this model to predict complex time series, as in the case of stock price series. For its construction, the Tensorflow framework was applied in conjunction with the Python programming language, which was chosen for its ability to be used on various platforms, and thus can be used for numerous purposes. Finally, the results of the tests are presented, where some of them outperform the stock price by 300%, besides proposing strategies for future optimizations.

Keywords: Reinforcement Learning. Deep Learning. Machine Learning. Time Series.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DNN	*Deep Neural Network*
DRL	*Direct Reinforcement Learning*
RL	*Reinforcement Learning*
RN	rede neural
ROC	*Price Rate of Change*

SUMÁRIO

1 INTRODUÇÃO

Há uma grande importância na previsão de séries temporais, tendo em vista que elas descrevem vários fenômenos encontrados no nosso cotidiano. Os dados de séries temporais e sua respectiva análise assumem uma importância crescente devido à produção volumosa desses mesmos dados, por meio da Internet das Coisas e da digitalização dos sistemas de assistência médica. (NIELSEN, 2021). Assim, o desenvolvimento de uma forma de previsão terá um grande número de finalidades, como controle de torque de motores. (MOREIRA, 2021).

Com isso, este trabalho propõe a utilização de uma rede neural (RN) com aprendizagem por reforço direto, aprendizagem recursiva e profunda. Os dados utilizados para o treinamento e validação escolhidos foram os de preços de ações, pois, estes apresentam um grande número de fatores externos que os influenciam, aumentando assim a complexidade destas previsões.

Antes de tratar sobre a construção da rede é necessário construir uma representação robusta dos dados utilizados, já que, caso isso não seja realizado, irá prejudicar de forma considerável o desempenho do sistema devido ao grande número de incertezas, como descrito por (NEELY *et al.*, 2014). Existem vários indicadores disponíveis que serão utilizados neste trabalho. No campo da negociação de reforço direto, foi introduzido o modelo de codificação esparsa como um extrator de características para análise, alcançando desempenhos mais confiáveis do que a aprendizagem por reforço direto sem o emprego deste recurso. (DENG *et al.*, 2017).

A RN proposta utiliza o conceito de *Direct Reinforcement Learning* (DRL), realimentação (MOODY *et al.*, 1998) e *Deep Learning* (DENG *et al.*, 2017), buscando melhorar a previsão de séries temporais. A aprendizagem por reforço direto, foi desenvolvida para resolver o problema de decisão de Markov (PUTERMAN, 2005), que consiste em métodos baseados em funções de valor e atores que realizam ações.

Algoritmos baseados em críticas estimam diretamente as funções de valor que é uma das estruturas de *Reinforcement Learning* (RL) mais usadas no campo. Esses métodos baseados em função de valor, por exemplo, *Q-learning*, são sempre aplicados para resolver os problemas de otimização definidos em um espaço discreto.

Embora o RL utilizando *TD-learning* apresente um bom desempenho para uma grande quantidade de aplicações, não é um paradigma adequado para o problema de séries temporais, em geral, como, negociações financeiras, pois, há a impossibilidade de obtenção de dados futuros, (MOODY; SAFFELL, 2001). Além do que, o ambiente de negociação é muito

complexo para ser aproximado em um espaço discreto, e que a definição da função valor sempre envolve um termo que codifica os retornos descontados futuros. (TESAURO, 1994).

Assim emprega-se a aprendizagem por reforço, baseada em atores que define um espectro de ações contínuas diretamente de uma família parametrizada de políticas. No método típico baseado em função de valor, a otimização sempre depende de alguma programação dinâmica complicada para derivar ações ideais em cada estado. A otimização do aprendizado baseado em atores é muito mais simples, pois, requer apenas uma função objetivo diferenciável com parâmetros latentes. Além disso, em vez de descrever diversas condições de mercado com alguns estados discretos, como no *Q-learning*, o método baseado em atores aprende a política diretamente dos dados sensoriais contínuos.

Neste trabalho será apresentada a capacidade de previsão de redes de aprendizagem por reforço direto, em parceria com aprendizagem profunda e recorrente. Para esta apresentação, este documento está organizado da seguinte forma: capítulo 2, é realizado o tratamento dos dados, onde eles são tornados estacionários, normalizados, e por fim fuzzyficados. No capítulo 3, é apresentado o modelo de aprendizagem por reforço direto (Seção 3.1), e profundo (Seção 3.2), já o capitulo 4 apresenta o algorítimo construído. No capítulo 5, é demonstrada a prova de conceito realizada. Capítulo 6, são apresentados os resultados obtidos. Por fim, a conclusão é apresentada no capítulo 7.

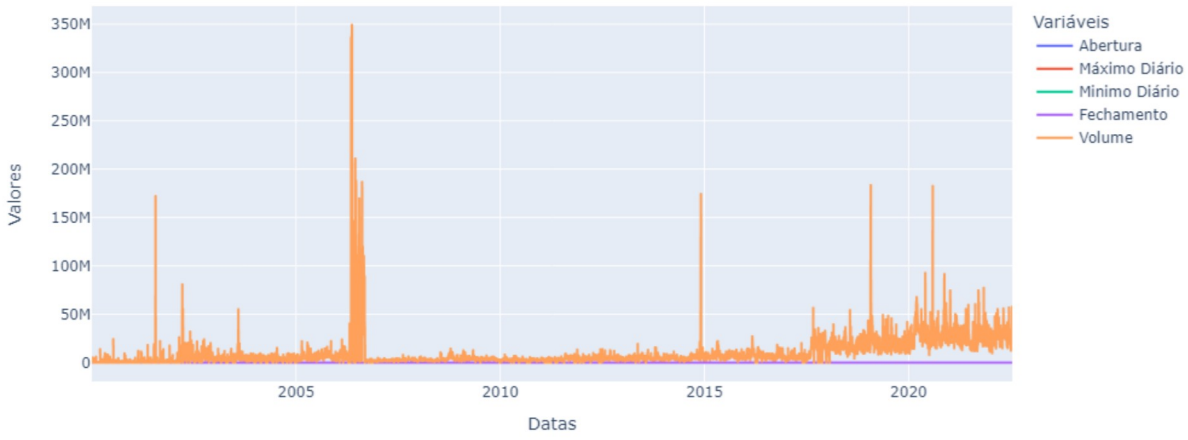
2 DADOS

Em aplicações práticas a DRL apresenta ótimos resultados para diversos problemas, pois, esta se aperfeiçoa a partir das suas tomadas de decisões sem nenhuma informação supervisionada envolvida, facilitando assim a construção da rede, porém, dificultando a aprendizagem. Uma forma de tentar contornar esse problema é tratar os dados buscando extrair o máximo de informações possíveis, torná-los estacionários, ou seja, eliminar a mudança de escala ao longo do tempo, e reduzir a discrepância de escalas entre dados diferentes.

2.1 Aquisição dos dados

Para a obtenção dos dados foi utilizada a biblioteca *yfinance* do *Python* (RAN, 2019), a qual disponibiliza as seguintes informações diárias das ações, preço de abertura, preço máximo, preço mínimo, preço de fechamento e volume (Figura 1). Deve-se observar a diferença de escalas entre os dados de Abertura, Fechamento, Mínimo e Máximo se comparados com os de Volume, evidenciando a necessidade do tratamento. A partir disso foram gerados novos dados buscando obter parâmetros para a RN de modo a facilitar seu aprendizado.

Figura 1 – Gráfico dos dados recebidos diretamente da biblioteca *yfinance*.



Fonte: elaborado pelo autor.

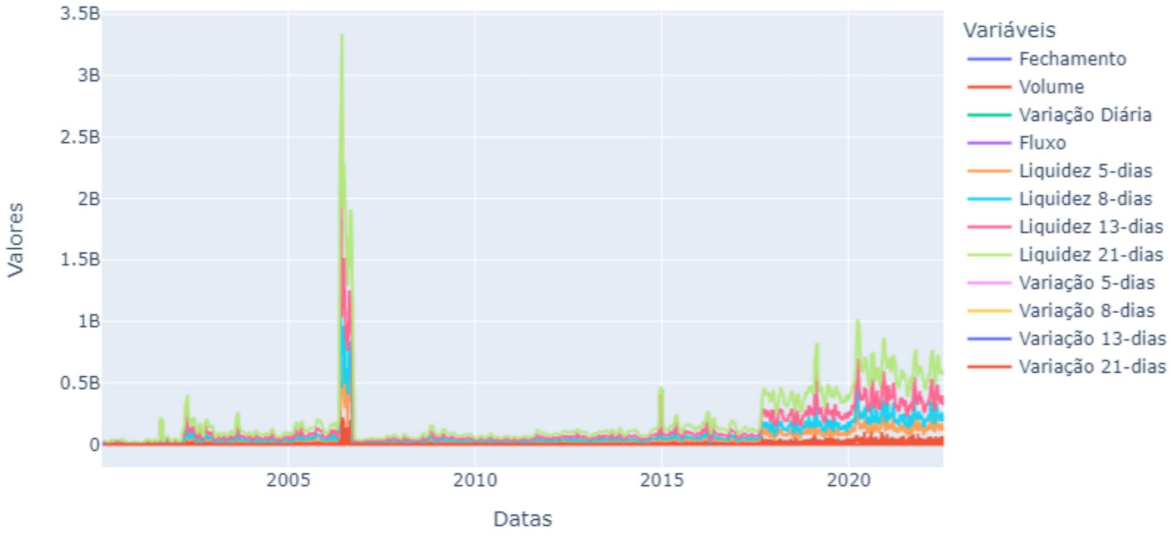
Os indicadores têm como função descrever o ambiente o qual a RN deve tomar a decisão, neste caso, como a ação está reagindo ao mercado. Assim foram calculados os valores de Liquidez observado na janela de tempo 1, 5, 8, 13 e 21 dias, valores de Variação nas janelas

de tempo de 1, 5, 8, 13 e 21 dias, e o valor do Fluxo, equação 2.1. Para realizar o aumento da dimensionalidade dos estados da rede neural, ou seja, fazer com que cada estado possua mais informações referente aos anteriores.

$$Fluxo = (Máximo\ Diário - Fechamento) - (Fechamento - Mínimo\ Diário) \tag{2.1}$$

No Apêndice A pode-se verificar os algoritmos utilizados para a criação desses indicadores. Na Figura 2 pode-se observar que assim como na Figura 1 muitos dados adicionados possuem escalas discrepantes se comparados.

Figura 2 – Gráfico com os indicadores de Fluxo, Liquidez e Variação adicionados.



Fonte: elaborado pelo autor.

2.2 Tratamento dos dados

A descrição do ambiente possui dados não-estacionários, isso significa que eles manifestam uma escala de interpretação diferente ao longo do tempo que pode confundir a rede neural.

Para resolver este problema foi preciso transformar esses dados em estacionários. Onde os dados apenas oscilam em torno de um referencial que está se movendo. Dessa forma, eles terão um referencial dinâmico, onde as mudanças em torno desse referencial são estacionárias

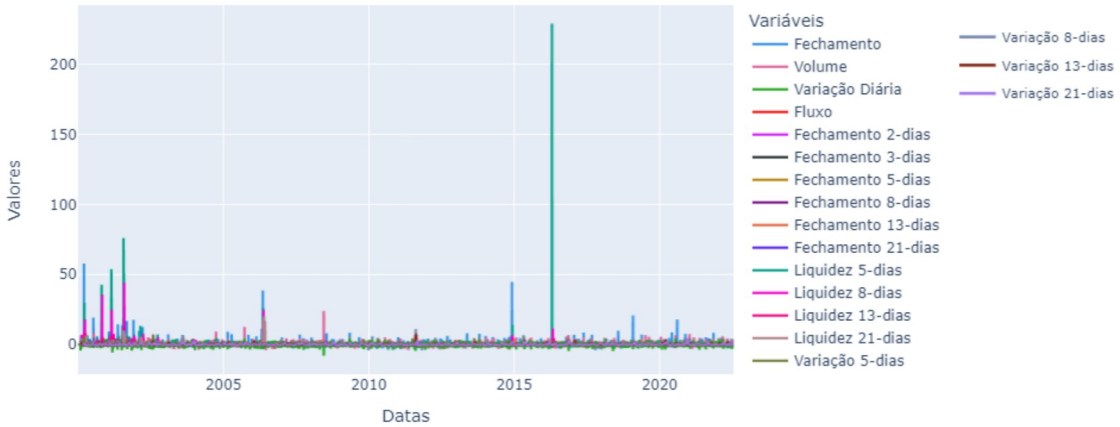
aos olhos da rede neural. Assim, ela não precisará atualizar escalas na sua interpretação, o que favorece a generalização da rede neural. Para tornar os dados estacionários serão usados dois osciladores: *Price Rate of Chang* e *Z-score* Móvel.

2.2.1 Price Rate of Change (ROC)

O *Price Rate of Change* (ROC) é um indicador técnico baseado em momento, que mede a variação percentual no preço entre o valor atual e o valor de um determinado número de períodos anteriores, Equação 2.2. Sendo um indicador plotado contra zero, move-se para cima em território positivo se as mudanças de valor forem positivas, e move-se para território negativo se as mudanças de preço forem negativas. Foram aplicados nos dados de Fechamento em períodos de 2, 3, 5, 8, 13 e 21 dias, essa função também foi aplicada nos valores de Liquidez e Variação em períodos de 5, 8, 13 e 21 dias. Os resultados obtidos podem ser observados na Figura 3.

$$ROC = \left(\frac{Fechamento_t - Fechamento_{t-n}}{Fechamento_{t-n}} \right) \tag{2.2}$$

Figura 3 – Gráfico com os dados de Fechamento, em diferentes janelas de tempo, obtidos a partir da Equação 2.2.



Fonte: elaborado pelo autor.

2.2.2 Z-score e Z-score Móvel

Z-score é o valor que descreve a relação de um número com a média da amostra em desvio padrão, Equação 2.3. Sendo aplicado nas colunas Volume, Variação e Liquidez.

O *Z-score* Móvel tem como diferença a utilização de apenas um conjunto dos dados

por vez para o cálculo do desvio padrão e media, sendo aplicado nas colunas de Volume, Variação e Fluxo com siderando a janela de tempo de 21 dias anteriores.

$$\text{Z-score} = \frac{x - \bar{x}}{\sigma} \tag{2.3}$$

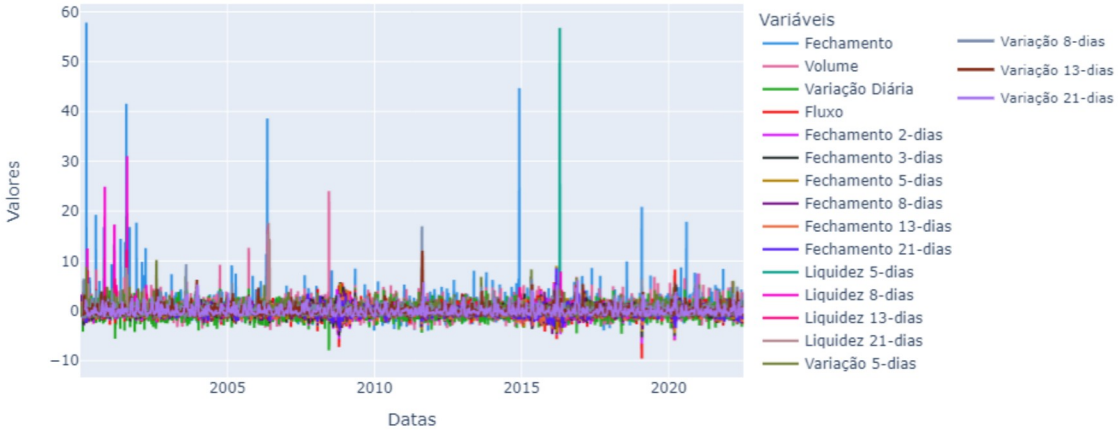
Onde x é o valor do dado o qual se quer calcular o *Z-score*, $\bar{x}$ é a média da amostra e σ é o desvio padrão.

2.2.3 Normalização

Após o desenvolvimento de novos indicadores deve-se atentar aos problemas gerados pelas escalas dos dados, pois, isso fara com que a RN tenda a dar mais importância para maiores valores no processo de tomada de decisão.

Os valores que foram calculados usando o *Z-score* móvel já estão na mesma escala, pois, estão normalizados em relação à média e ao desvio padrão. No entanto, os dados resultantes do ROC não possuem normalização. Desta forma, deve-se aplicar o *Z-score* em todos os resultados obtidos a partir da função ROC. Figura 4, onde já se pode observar uma melhora em relação às escalas dos dados.

Figura 4 – Gráfico com os dados de Fechamento, Liquidez e Variação, em todas as janelas de tempo, normalizados.



Fonte: elaborado pelo autor.

2.2.4 Fuzzificação

Diferente de outros tipos de sinais, como imagens ou fala, as sequências de dados financeiros apresentam uma grande quantidade de ruído devido às incertezas por trás da negociação. Uma quantidade de fontes de informação, desde a atmosfera econômica global até alguns rumores da empresa, pode afetar a direção do sinal financeiro em tempo real. Portanto, reduzir o ruído nos dados brutos é uma abordagem importante para aumentar a robustez da mineração de sinais financeiros.

Assim, adotou-se o processo de fuzzyficação para remover o ruído dos dados. Em vez de adotar descrições precisas de alguns fenômenos, os sistemas *fuzzy* preferem atribuir valores linguísticos *fuzzy* aos dados de entrada. Tais representações fuzzificadas podem ser facilmente obtidas comparando os dados do mundo real com um número de conjuntos aproximados *fuzzy* e então derivando os graus de pertinência *fuzzy* correspondentes. Consequentemente, o sistema de aprendizado só funciona com essas representações difusas para tomar decisões de controle robustas.

O conjunto de valores da função *Z-score* pertence aos reais, mas poderá assumir o valor 0 com maior frequência, e terá a sua frequência reduzida à medida que se afasta do zero. Intervalos no centro da função são mais prováveis, enquanto que fora são menos prováveis. Então ao invés de se utilizar para valores de *Z-score*, serão utilizados *Z-score* móveis, e a fuzzyficação para descrever esses intervalos para a rede neural. Para rede não interessa qual o valor do *Z-score*, mas a qual conjunto ele pertence. Assim, a rede neural interpreta com maior facilidade quais são as chances do próximo evento, o que facilita o aprendizado da rede. Para isso foram utilizados 8 níveis de classificação, e para calcular foi utilizado as equações 2.5 e 2.6.

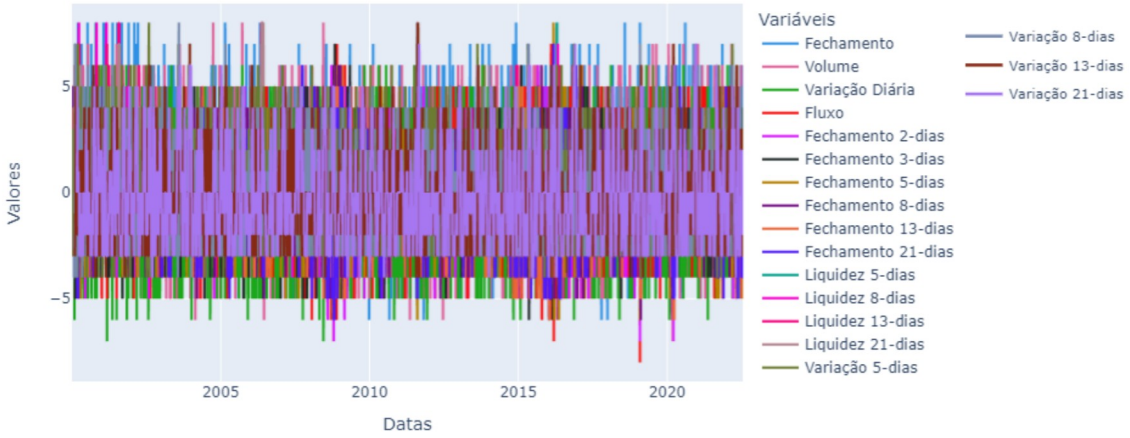
$$sigmoid = \frac{1}{(1+e^{(}-x60000))} \tag{2.4}$$

$$\phi = \frac{(1+\sqrt{5})}{2} \tag{2.5}$$

$$\text{fuzzyficação}(x) = \sum_{n=1}^{8}\left[-sigmoid\left(-x-\frac{\phi^{n-2}}{2}\right)\right] - \sum^{8} n = 1\left[-sigmoid\left(x-\frac{\phi^{n-2}}{2}\right)\right] \tag{2.6}$$

A Figura 5 mostra a representação por níveis após a aplicação da fuzzificação nos dados, se comparado a Figura 4 antes deste processo pode-se observar que o problema de diferença de escala foi solucionado.

Figura 5 – Gráfico dos dados fuzzyficados, onde variação limitada entre 8 e -8.

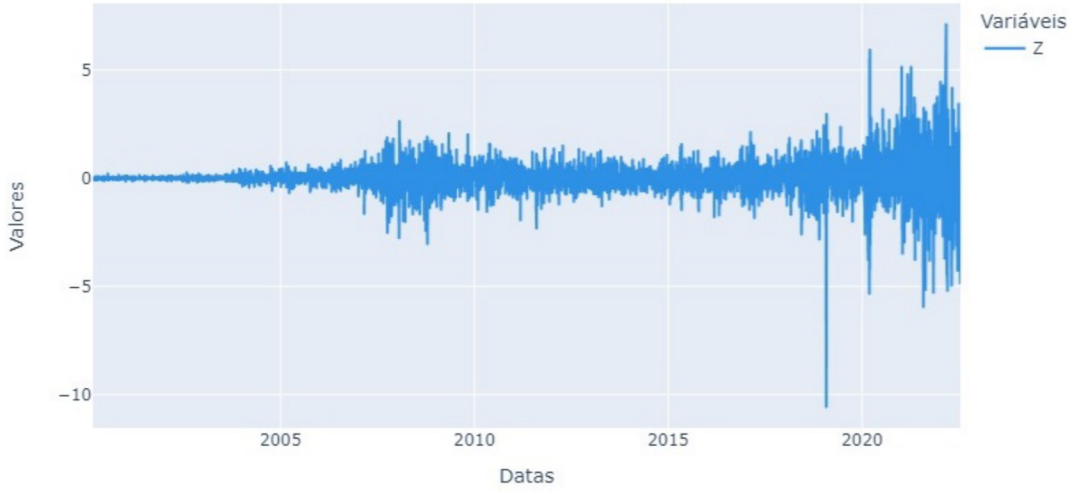


Fonte: elaborado pelo autor.

2.3 Dados de variação

Para finalizar o tratamento dos dados é realizado o cálculo da variação entre o dia anterior e o dia atual, denominado de Z, que serão utilizados posteriormente durante o treinamento da rede e para analisar o resultado do treinamento. Na Figura 6 pode-se observar esse dados.

Figura 6 – Gráfico de variação diária dos preço de Fechamento das ação.



Fonte: elaborado pelo autor.

3 MODELO

3.1 *Direct Reinforcement Learning*

Sejam p_1, p_2, ..., p_t as sequências de preços recebidas do centro de troca através da biblioteca *yfinance*. Então, o retorno no ponto de tempo t é facilmente determinado por $z_t = p_t - p_{t-1}$. Com base nas condições atuais do mercado, a decisão de negociação em tempo real (política) $\delta t \in \{\text{comprado, neutro, vendido}\} = \{1, 0, 1\}$ é feita em cada ponto de tempo t. Com os símbolos definidos acima, o lucro R_t obtido pelo modelo de negociação é mostrado na equação 3.1.

$$Rt = \delta_{t-1} Z_t - c|\delta_t - \delta_{t-1}| \tag{3.1}$$

Em que δ_{t-1} é o lucro, ou perda, obtido com as variações do mercado, $Z_t - c|\delta_t - \delta_{t-1}|$ é o custo de transação (c), ao inverter sua posição, c é a taxa obrigatória paga à corretora somente se $\delta_t \neq \delta_{t-1}$. Quando duas decisões de negociação consecutivas são iguais, ou seja, $\delta_t = \delta_{t-1}$, não ocorre nenhuma transação assim não é gerado cobrança.

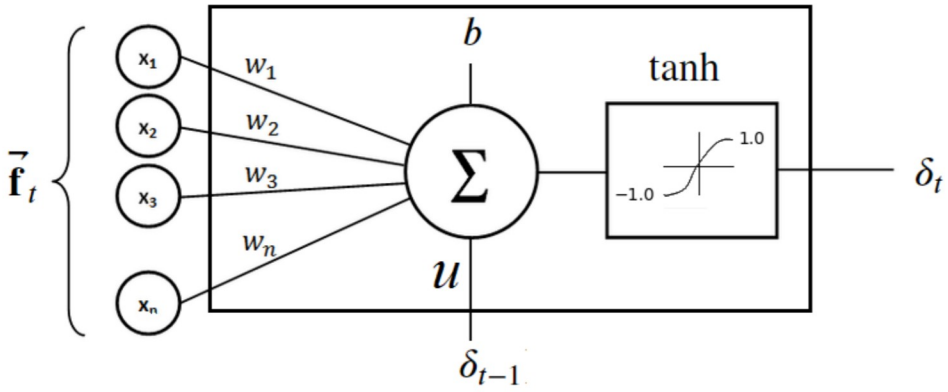
Para a política de aprendizado, o que interessa é o lucro acumulado após uma quantidade significativa de períodos onde operações de compra e venda são realizadas. Dessa forma, é denotado UT como a recompensa acumulada nos períodos de $1, \ldots, T$. Ele é o lucro total (TP - *total profit* no inglês) obtido após T períodos, equação 3.2. Outras funções de recompensa, por exemplo, os retornos ajustados ao risco, também podem ser usadas aqui como objetivo para a DRL. Para facilitar as explicações do modelo, optou-se por usar o TP como função objetivo ao longo do trabalho.

$$UT = \sum_{t=1}^{T} Rt \tag{3.2}$$

Agora resta introduzir uma estratégia para aprender a política de negociação diretamente. Para isso, uma regressão logística é feita com a realimentação de decisões atrasadas, onde um peso u é dado a decisão δ_{t-1} tomada no período anterior. Assim, a ação de negociação (política) em cada ponto de tempo será dada pela equação 3.3, onde $\overrightarrow{w}$ é o vetor de pesos, $\overrightarrow{f_t}$ é o vetor de entrada, b é o valor de bias e u o peso da saída anterior da rede. Veja uma ilustração do neurônio na figura 7.

$$\delta_t = tanh[\langle \overrightarrow{w}, \overrightarrow{f_t} \rangle + b + u\delta_{t-1}] \tag{3.3}$$

Figura 7 – Estrutura da DRL.



Fonte: elaborado pelo autor.

O vetor de características $\overrightarrow{f_t} = [x_1, ..., x_n] \in R_n$ é resultado do processo de implementação da descrição do ambiente. Após isso, a transformação linear $\delta_t = tanh[\langle \overrightarrow{w}, \overrightarrow{f_t} \rangle + b + u\delta_{t-1}$ é mapeada para o intervalo [-1,1] através da função tangente hiperbólica. Assim, o aprendizado visa encontrar a família de parâmetros $\overrightarrow{\theta} = [\overrightarrow{w}, u, b]$ que pode maximizar a função de recompensa global UT, equação 3.4.

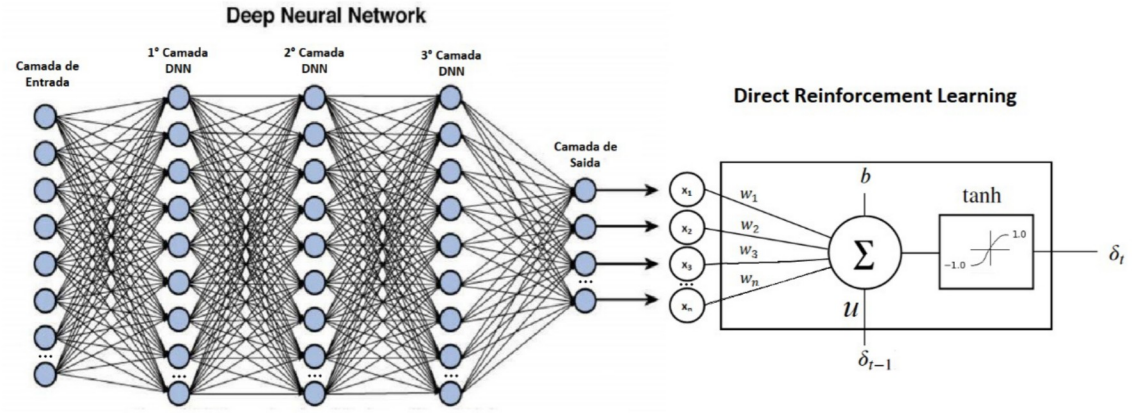
$$\max_{\overrightarrow{\theta}} UT \left[R_1 ... R_t | \overrightarrow{\theta} \right] \tag{3.4}$$

3.2 Deep Learning

Com a camada de saída DRL terminada, foram implementadas as camadas de aprendizagem profunda, buscando melhorar o aprendizado da rede. Estas foram estruturadas da seguinte forma: para o cálculo de cada neurônio de saída $f_{t,1} = \langle \overrightarrow{w_{d,1}}, \overrightarrow{f_t} \rangle$, assim tem-se que a camada de saída apresenta a seguinte representação $F_{t,1} = [f_{t,1}, f_{t,2}, ..., f_{t,18}]$, onde f_t são os dados de entradas após o tratamento, $F_{t,1}$ é a saída da camada intermediaria e F_t é a saída da camada rede de *Deep Neural Network* (DNN). A representação do modelo pode ser vista na figura 8. Redes neurais profundas tem como característica a detecção de padrões sutis em dados, mas se o conjunto de treinamento é muito pequeno ou possui ruído, então o modelo irá detectar padrões no próprio ruído.

Com o aumento do número neurônios da rede a capacidade de aprendizado aumenta, e isso abre espaço para outros problemas como *overfiting*, onde a rede decora os padrões dos dados utilizados, conseguindo um bom resultado no treinamento. No entanto, quando a rede é testada em um novo conjunto de dados a RN obtém um resultado ruim, pois, ela não aprendeu uma regra e sim decorou as entradas do treinamento. Uma forma de contornar este problema é encontrar o ponto onde a rede para de aprender e começa a decorar, para interromper o treinamento, ou realizar um *dropout* de alguns neurônios da rede.

Figura 8 – Estrutura da DRL com DNN.



Fonte: Modificado de: https://en.difesaonline.it/evidenza/cyber/la-nuova-rivoluzione-digitale-il-deep-learning

4 ALGORITMOS

Nesta etapa será mostrado o algoritmo necessário para o desenvolvimento da rede e teste de suas capacidades, conforme a ordem de desenvolvimento, iniciando pela construção da rede de DRL, e seguido pelos algoritmos acrescentados e alterados para adicionar as camadas de *deep learning*.

Tendo início com a especificação das variáveis de configuração da rede:

```
1  # Variáveis de configuração
2  initial_data = 3000 # Seleciona a data inicial para ser
       utilizada na rede
3  percentual = 0.6 # Qual percentual dos dados será separado para
       treinamento
4  c = 0.01 # Taxa de movimentação
5  batch_size = 600 # Número de dados carregados ao mesmo tempo
6  epochs = 1000 # Número de vezes que os dados serão utilizados
       para o treinamento
7  learning_rate = 0.0009 # Taxa de aprendizagem
8  meta_inicial = 20 # Resultado mínimo necessário para os pesos
       iniciais sejam aceitos
9  zerar_pesos = True # Variável para decidir se os pesos serão
       resetados ou não
10 name_pasta = "Funcional/save" # Arquivos onde serão salvos os
       pesos
```

4.1 Algoritmos para construção da DRL

Função para geração dos vetores a partir do *dataframe* de entrada:

```
1  def gerar_vetores(df: pd.DataFrame):
2      """ Gera um array a partir do dataframe de entrada. """
```

```
    array = []
    array = np.array(df.values)
    try:
        array = np.reshape(array, (df.shape[0], df.shape[1]))
    except:
        array = np.reshape(array, (df.shape[0], 1))
    return array
```

Transformando os dados de *dataframe* para *Array*:

```
fuzzy_train = gerar_vetores(df_fuzzy_train)
fuzzy_test = gerar_vetores(df_fuzzy_test)
z_train = gerar_vetores(df_z_train)
z_test = gerar_vetores(df_z_test)
```

Função DRL_DNN, implementa a equação 3.3:

```
def DRL_DNN(entrada, saida_anterior, W_rl, b_rl, U, pesos_dnn,
   pesos_dnn_2, pesos_dnn_3):
    """DRL de saída"""
    # realiza a mudança das dimensões da matriz (m,n) -> (n,m)
    transp_U = tf.transpose(U)
    transp_W2 = tf.transpose(W_rl)

    # tf.matmul realiza a multiplicação entre o tensor de pesos
       e o tensor de entradas (dados fuzzyficados)
    saida_mlp_1 = tf.matmul(transp_W2, entrada) + b_rl
    saida_mlp_2 = tf.matmul(transp_U, saida_anterior)

    # Passa a soma dos resultados pela função de ativação
       (tangente hiperbólica)
    saida = tf.tanh(saida_mlp_1+saida_mlp_2)
```

```
13      return saida
```

Função criada para inicializar os tensores de pesos, com valores aleatórios para o treinamento.

```
1  def init_weights(shape):
2      """Inicializa os pesos com valores aleatórios"""
3      # gera um tensor com dados aleatórios com as dimensões
           passadas como parâmetro
4      rn = tf.random_normal(shape)
5      # Realiza a construção de uma variável que tem tipo e forma
           fixa
6      return tf.Variable(rn)
```

A função *unfolded* calcula o valor Ut, equação 3.2, com base na equação de recompensa 3.1, passando os dados de entrada a função da RN e obtendo seu resultado para ser encaminhado a próxima camada devido à recursividade da rede.

```
1  def unfolded( features, W1, b1, U, dt, dados_z):
2    """Calcula o valor de Rt e Ut na rede"""
3      delta = [dt]
4      r = []
5      rts = []
6      for i in range(1, features.shape[0]):
7          f = features[i]
8          f = tf.reshape(f, [m, 1])
9          delta_atual = DRL_DNN(
10             f, delta[i-1], W1, b1, U, pesos_dnn, pesos_dnn_2,
                   pesos_dnn_3)
11         delta.append(delta_atual)
12         Rt = delta[i-1]*dados_z[i] - c*tf.abs(delta[i] -
               delta[i-1])
```

```
13          r.append(Rt)
14          rts.append(Rt[0][0])
15      Ut = sum(r)
16      return Ut, r, delta, rts
```

O algoritmo a seguir descreve, onde as funções criadas anteriormente serão utilizadas começando com a inicialização dos pesos, seguindo pela inicialização dos tensores para recebimento dos dados. Os tensores são passados para função *unfolded* a qual retorna as variáveis de saídas da rede e por fim é criado a variável de otimização passando os parâmetros da rede e informado para esta minimizar o valor de -Ut.

```
1  # Inicializando pesos.
2  W1 = init_weights([m,1])
3  b1 = init_weights([1,1])
4  U = init_weights([1,1])
5
6  # Inicializando variável que irá receber a saída da rede para
       os pesos anteriores.
7  dt =
       tf.Variable(tf.zeros([1,1]),dtype=tf.float32,trainable=False)
8
9  # Um placeholder é simplesmente uma variável à qual é atribuído
       aos dados posteriormente.
10 features = tf.placeholder(tf.float32,shape = [batch_size,m])
11 z_placeholder = tf.placeholder(tf.float32,shape =
       [batch_size,1])
12 UT,r,delta,rts = unfolded(features,W1,b1,U,dt,z_placeholder)
13
14 # Estimativa de momento adaptável é um algoritmo para técnica
       de otimização para descida de gradiente.
15 optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
16
17 # Definindo o objetivo da Rede
```

```
18 train = optimizer.minimize(-UT)
```

Algoritmo para escolha de pesos aceitáveis para a inicialização da rede, tendo como base um valor mínimo aceitável para iniciar o treinamento, escolhido pelo programador, ao fim deste algoritmo os pesos são salvos em arquivos para poderem ser utilizados no treinamento.

```
1  if zerar_pesos:
2      init = tf.global_variables_initializer() # prepara uma
           variável para inicializar as variáveis do tensorflow
3
4      pesos_validos = False
5      n_tentativas = 0
6      melhor_resultado = 0
7      meta = meta_inicial
8      while(not(pesos_validos)):
9          n_tentativas += 1
10         with tf.Session() as sess:
11             sess.run(init)
12             print(f'Tentativa: {n_tentativas}')
13             ut_temp = 0
14             for i in range(0,
                   math.floor(fuzzy_train.shape[0]/batch_size)):
15                 UT_val, delt_saida, temp_rts = sess.run([UT,
                       delta, rts], feed_dict={
16                     features:
17                     fuzzy_train[i*batch_size
18                     :(i+1)*batch_size],
19                     z_placeholder:
20                     z_train[i*batch_size:
21                     (i+1)*batch_size]})
22                 ut_temp += UT_val[0][0]
23             if ut_temp > meta_inicial:
24                 pesos_validos = True
```

```
25                  save_path = saver.save(sess, "./my_model_final")
26              print(f'Resultado: {ut_temp}')
```

O algoritmo de treinamento abaixo tem início com a criação das variáveis que serão necessárias durante o loop principal. Antes do *loop* ser iniciado é necessário iniciar a sessão no *tensorflow* e carregar os pesos dos arquivos. O *loop* consiste em percorrer todos os valores do *dataframe* de entrada, um certo número de vezes, que se denomina épocas. Os dados de entradas são fragmentados para ficar do tamanho dos *batch size* delimitados anteriormente.

```
1  init = tf.global_variables_initializer() # Prepara uma variável
       para inicializar as variáveis do tensorflow
2
3  obj = []
4  b_val = []
5  ut_temp = 0
6  tempo_restante = "Calculando..."
7  resutlado_parcial = 0
8
9  with tf.Session() as sess:
10     saver.restore(sess, "./my_model_final")
11     for j in range(epochs):
12         delt_train = []
13         saida_train_rts = []
14         print(f'Epoca: {j} / Tempo restante: {tempo_restante} /
               Resultado parcial: {resutlado_parcial}')
15         start = time.time() # Variável de tempo
16         for i in
               range(0,math.floor(fuzzy_train.shape[0]/batch_size)):
17             _, UT_val, delt_saida,temp_rts =
                   sess.run([train,UT,delta,rts],
18             feed_dict = {features:
19             fuzzy_train[i*batch_size:(i+1)*batch_size],
20             z_placeholder:
```

```
                z_train[i*batch_size:(i+1)*batch_size]})
            delt_train = delt_train + delt_saida
            saida_train_rts += temp_rts
            ut_temp += UT_val[0][0]
        obj.append(ut_temp)
        ut_temp = 0
        stop = time.time() # Variável de tempo
        if (stop - start)*(epochs-j) > 60:  # Variável de tempo
            tempo_restante = f'{round((stop -
                start)/60*(epochs-j),0)} minutos'  # Variável de
                tempo
        else:  # Variável de tempo
            tempo_restante = f'{round((stop -
                start)*(epochs-j),0)} segundos' # Variável de
                tempo
        ut_temp = 0
        if obj[-1] < 0:
            resutlado_parcial = colored(f'{obj[-1]}','red')
        else:
            resutlado_parcial = colored(f'{obj[-1]}','green')

    print(f"Acumulado do treinamento: {obj[-1]}")
    save_path = saver.save(sess, "./my_model_final")
```

O algoritmo para realização do teste em dados separados, assim como no treinamento tem que iniciar a sessão no *tensorflow* e em seguida realizar a importação dos pesos calculados no treinamento, e em seguida iniciar o *loop* passando pelos dados de teste, e obter o vetor delta de saída da rede.

```
"""## Import e teste do modelo"""
with tf.Session() as sess:
    saver.restore(sess, "./my_model_final")
    # Trocar por função de teste
```

```
5     delt_test = []
6     saida_test_rts = []
7     for i in
         range(0,math.floor(fuzzy_test.shape[0]/batch_size)):
8         [delt_saida,temp_rts] = sess.run([delta,rts],
9         feed_dict = {features:
10            fuzzy_test[i*batch_size:(i+1)*batch_size],
11            z_placeholder:z_test[i*batch_size:(i+1)*batch_size]})
12        delt_test = delt_test + delt_saida
13        saida_test_rts += temp_rts
```

4.2 Algoritmos para construção das camadas de *deep learning*.

Para construir o algoritmo do modelo com as camadas de *deep learning* foram necessárias algumas alterações iniciando pela criação das seguintes funções, para realizar as multiplicações dos tensores de pesos.

```
1  # Gera o vetor de pesos para a DNN
2  def vet_pesos_dnn(n_paramtros):
3      pesos_dnn = []
4      pesos_dnn.append([init_weights([m, 1]) for i in
          range(n_paramtros)])
5      return pesos_dnn
6
7  # Realiza a multiplicação entre o tensor de pesos e o tensor de
       entradas
8  def mul(entrada, pesos):
9      return tf.matmul(tf.transpose(pesos), entrada)
10
11 # Realiza a multiplicação entre os tensores de pesos e o tensor
       de entradas
12 def DNN(entrada, pesos_dnn):
```

```
13      temp = [mul(entrada, i) for i in pesos_dnn]
14      return temp
```

A função DRL_DNN construída a partir da função DNN, com o acréscimo das linha 7 a 8, e alterações nas linhas 1 e 14, com a finalidade criar os vetores de pesos da DNN, e lidar com eles.

```
1  def DRL_DNN(entrada, saida_anterior, W_rl, b_rl, U, pesos_dnn,
       pesos_dnn_2, pesos_dnn_3):
2      """Neurônio de saída"""
3      # Realiza a mudança das dimensões da matriz (m,n) -> (n,m)
4      transp_U = tf.transpose(U)
5      transp_W2 = tf.transpose(W_rl)
6
7      #Selecione o número de camadas para DNN
8      saida_dnn = DNN(entrada, pesos_dnn)[0][0]
9      saida_dnn_2 = DNN(saida_dnn, pesos_dnn_2)[0][0]
10     #saida_dnn_3 = DNN(saida_dnn_2, pesos_dnn_3)[0][0]
11
12     # tf.matmul realiza a multiplicação entre o tensor de pesos
           e o tensor com
13     # as variáveis de ft.
14     saida_mlp_1 = tf.matmul(transp_W2, saida_dnn_2) + b_rl
15     saida_mlp_2 = tf.matmul(transp_U, saida_anterior)
16
17     # Passa a soma dos resultados pela função de ativação
           tangente hiperbólica
18     saida = tf.tanh(saida_mlp_1+saida_mlp_2)
19     return saida
```

Por fim deve ser alterada a construção da rede, pois, os pesos da DNN devem ser inicializados com os outros pesos, e passados na função *unfolded*, linhas 4 a 8 e 18.

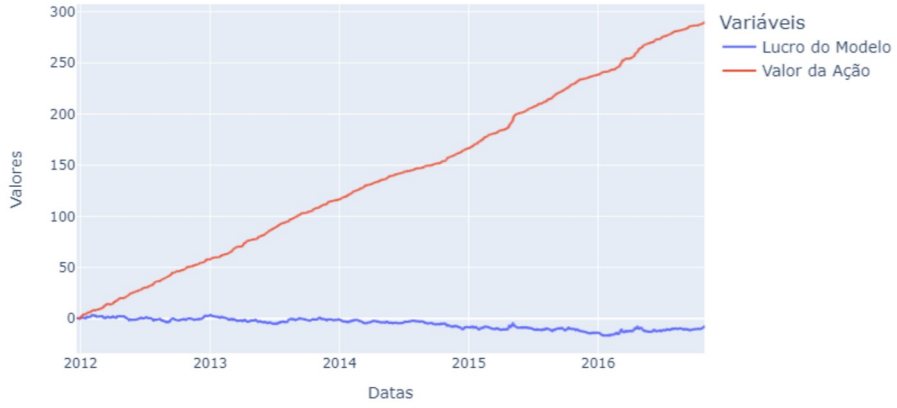
```
# Limpa a pilha de gráficos padrão e redefine o gráfico padrão global.
tf.reset_default_graph()

# Construindo variáveis da rede neural
# Inicializando os pesos do neurônio de saída
pesos_dnn = vet_pesos_dnn(m)
pesos_dnn_2 = vet_pesos_dnn(m)
# pesos_dnn_3 = vet_pesos_dnn(m)

W1 = init_weights([m,1])
b1 = init_weights([1,1])
U = init_weights([1,1])
dt = tf.Variable(tf.zeros([1,1]),dtype=tf.float32,trainable=False)

# Um placeholder é simplesmente uma variável à qual será atribuído dados posteriormente.
features = tf.placeholder(tf.float32,shape = [batch_size,m])
z_placeholder = tf.placeholder(tf.float32,shape = [batch_size,1])
UT,r,delta,rts = unfolded(features,W1,b1,U,dt,pesos_dnn,
            pesos_dnn_2,pesos_dnn_3,z_placeholder)
# Estimativa de momento adaptável é um algoritmo para técnica de otimização para descida de gradiente.
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)

# Definindo o objetivo da Rede
train = optimizer.minimize(-UT)
```

5 PROVA DE CONCEITO

Para submeter a rede a um teste foi substituído o Z_t por Z_{t-1}, visto que com essa modificação a rede necessitará compreender uma regra simples, que consiste em verificar se houve uma variação positiva ou negativa entre o dia anterior e o atual.

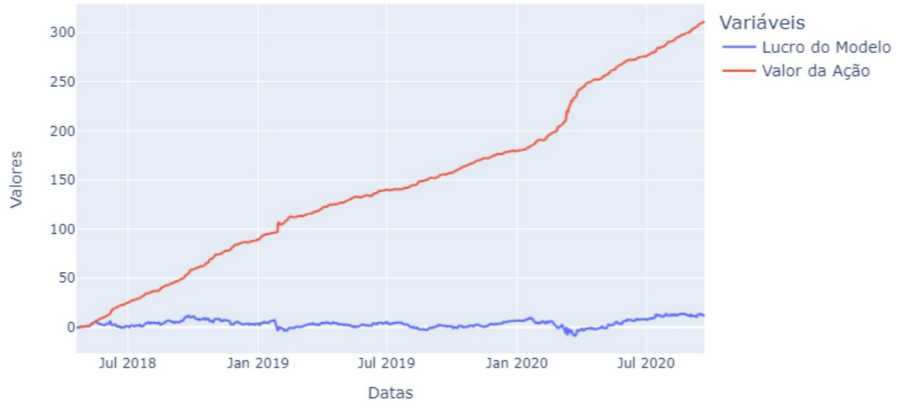
Assim, foram obtidos os resultados apresentados figuras 9 e 10. Resultados semelhantes foram apresentados para a DRL simples, assim como o acréscimo da camada de DNN, porém, houve uma variação no número de épocas necessárias, onde quanto maior o número de camada de DNN utilizadas, menor o número de épocas necessárias para o treinamento.

Figura 9 – Gráfico do resultado do treinamento apresentado pela rede para Z_{t-1}.



Fonte: elaborado pelo autor.

Figura 10 – Gráfico do resultado do teste apresentado pela rede para Z_{t-1}.



Fonte: elaborado pelo autor.

Portanto, se conclui que a rede não apresenta nenhum problema de construção, visto que a tarefa foi realizada com sucesso e de forma simples já que foram necessárias poucas épocas para o aprendizado. Pode-se observar que os períodos em que o preço da ação possui uma maior oscilação é onde se obtém o maior lucro.

6 VERIFICAÇÃO DA APRENDIZAGEM

A rede foi construída de forma gradual, iniciando de uma simples rede de DRL onde seus resultados podem ser observados na tabela 1, e sendo adicionadas novas camadas de *deep learning*, cada uma contendo 18 neurônios.

Para verificar a performance da rede foi utilizado os valores de *Rt* que foram calculados conforme a equação 3.1, a partir da variação dos seguintes parâmetros *batch_size*, épocas e taxa de aprendizagem. Dentre os parâmetros citados o que carreta em maior dificuldade de processamento a partir da sua escolha é o *batch_size*, devido ao fato dele interferir diretamente no consumo de memória RAM do computador que é utilizado para o processamento. Logo, quanto menor o *batch_size*, menor o tempo de execução. Já o número de épocas afeta apenas no tempo necessário para finalizar o treinamento, sem interferir no uso de recursos da máquina.

Os primeiros resultados foram obtidos para a rede DRL utilizando as ações da VALE3.SA, onde os resultados podem ser observados a Tabela 1. A coluna de treinamento e teste representam o quanto o valor de Ut se distanciou do valor da ação após o período observado. Pela tabela pode-se constatar que a rede apresentou resultados positivos em todos os treinamentos, porém, nos testes os resultados não superam os valores de UT. Buscando melhorar os valores nos testes foi acrescentada uma camada inicial de 18 neurônios, com isso foram observados os valores mostrados na Tabela 2, um dos quais apresentou resultado positivo no teste.

Com o acréscimo da segunda camada de 18 neurônios os resultados que apresentaram valores positivos foram consideravelmente maiores que os apresentados por apenas uma camada, esses valores podem ser observados na Tabela 3. Por fim com o acréscimo da terceira camada o modelo já passou a sofrer consideravelmente de *overfiting*, como pode ser constatado na Tabela 4.

Tabela 1 – Resultados obtidos apartir rede com DRL

Épocas	Batch Size	Taxa de Aprendizagem	Treinamento	Teste
1000	500	0,0009	61,56	-73,17
6000	500	0,00009	63,07	-113,42
1000	600	0,0009	59,28	-34,06
6000	600	0,00009	58,70	-10,45

Fonte: elaborado pelo autor.

Tabela 2 – Resultados obtidos apartir rede com DRL e uma camada de DNN

Épocas	Batch Size	Taxa de Aprendizagem	Treinamento	Teste
1000	500	0,0009	25,58	-83,05
6000	600	0,0009	17,14	-11,33
6000	600	0,00009	54,69	-35,90
6000	1000	0,0000009	21,73	6,12

Fonte: elaborado pelo autor.

Tabela 3 – Resultados obtidos apartir rede com DRL e duas camadas de DNN

Épocas	Batch Size	Taxa de Aprendizagem	Treinamento	Teste
1000	500	0,0009	20,04	-0,91
1000	500	0,0009	45,19	-55,23
1000	600	0,00009	51,94	18,32
6000	600	0,00009	25,98	4,04

Fonte: elaborado pelo autor.

Tabela 4 – Resultados obtidos apartir rede com DRL e três camadas de DNN

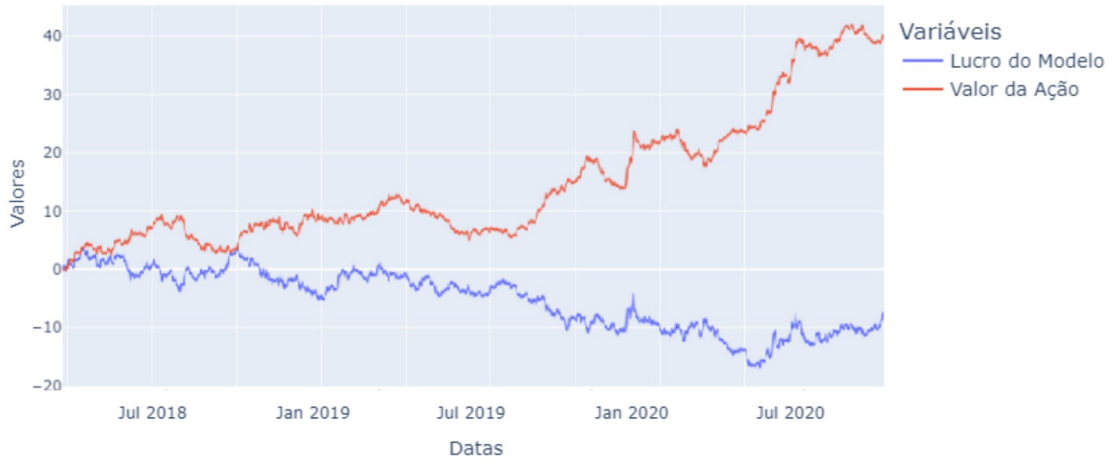
Épocas	Batch Size	Taxa de Aprendizagem	Treinamento	Teste
1000	500	$9{\cdot}10^{-8}$	19,41	-109,90
1000	500	$9{\cdot}10^{-6}$	15,85	-84,58
3000	600	$4{\cdot}10^{-6}$	23,28	64,36
3000	600	$3{\cdot}10^{-6}$	32,31	25,31

Fonte: elaborado pelo autor.

As Figuras mostradas a seguir são os resultados obtidos a partir da aplicação das melhores configurações apresentadas nas tabelas acima para as respectivas construções, onde se manteve a escolha da ação VALE3.SA.

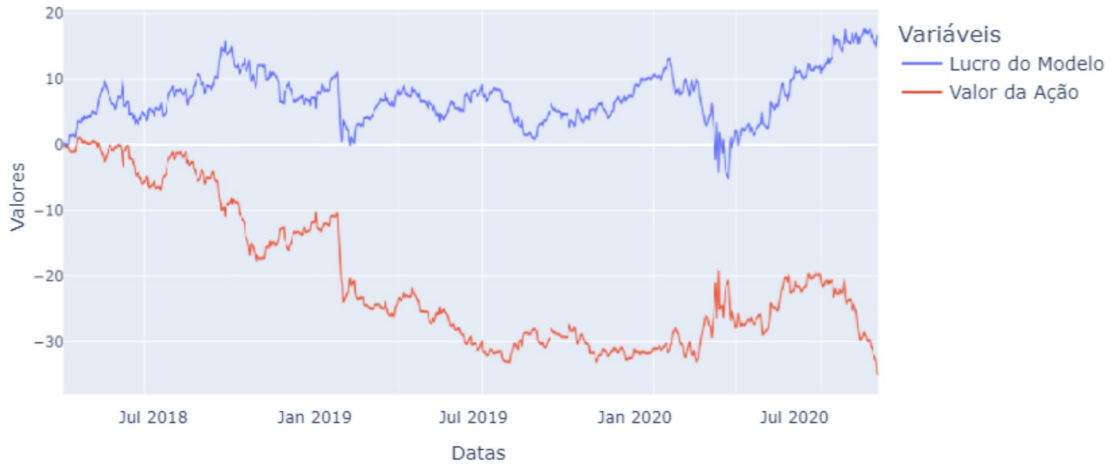
Pode-se observar na Figura 11 o resultado do treinamento para a DRL e na Figura 12 os valores para o teste. Durante o treinamento, o lucro do modelo apresentou uma estrutura de crescimento, que indica que caso o treinamento desse prosseguimento o valor adquirido continuaria a crescer, porém, nos testes, observa-se o decréscimo do valor obtido pela rede, mostrando que este resultado não teria um bom resultado para uma aplicação, pois, quando é dado entrada com novos dados, pode-se perceber que não foi aprendido nenhuma regra que possa ser generalizada para todos os dados.

Figura 11 – Gráfico do resultado para o treinamento com DRL



Fonte: elaborado pelo autor.

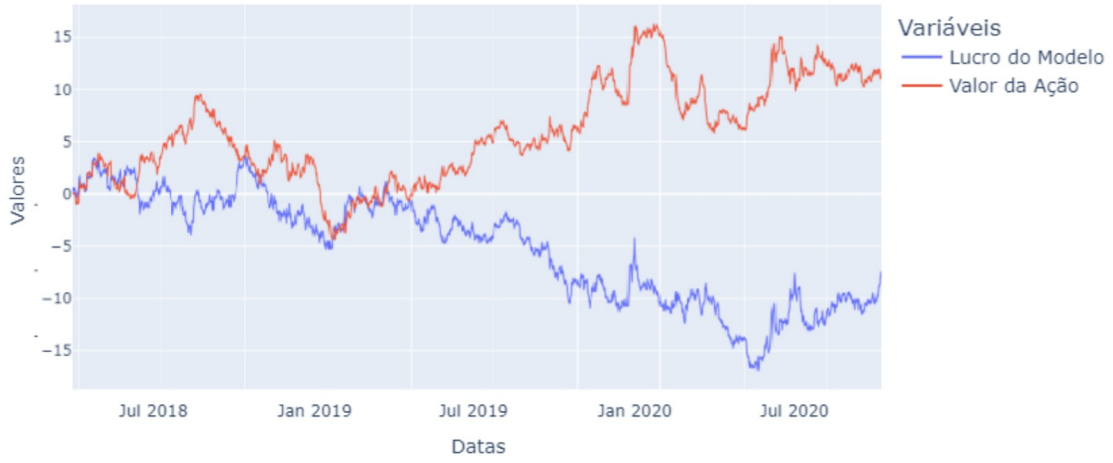
Figura 12 – Gráfico do resultado para o teste com DRL



Fonte: elaborado pelo autor.

As Figuras 13 e 14 exibem os resultados referentes a utilização da RN composta pela DRL e uma camada de DNN. Diferente do resultado anterior, nestes pode-se observar que o lucro do modelo sofreu uma queda se comparado ao gráfico do treinamento anterior, porém, durante teste foi possível superar o crescimento da ação antes de começar a decair, demonstrando uma ligeira melhora.

Figura 13 – Gráfico do resultado para o treinamento com DRL com 1 camada extra.



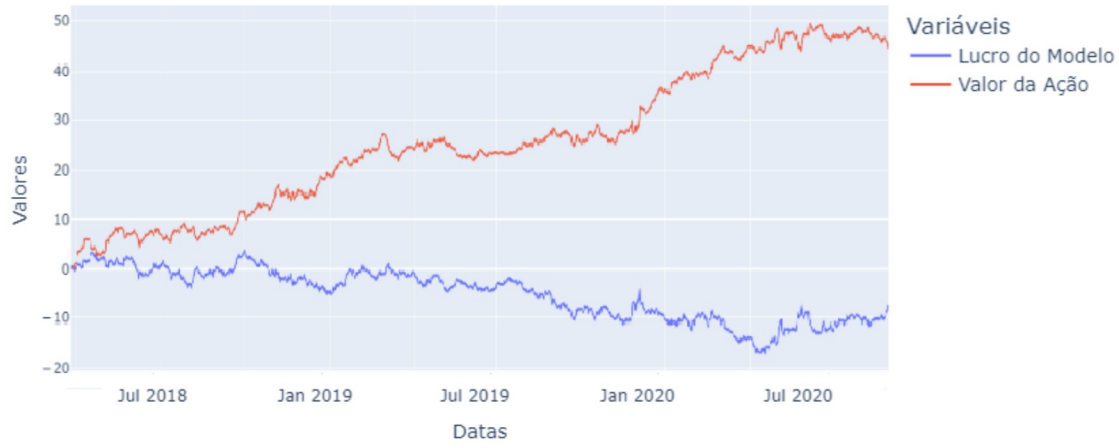
Fonte: elaborado pelo autor.

Figura 14 – Gráfico do resultado para o teste com DRL com 1 camada extra.

Fonte: elaborado pelo autor.

Com 2 camadas obtiveram-se os seguintes resultados: Figuras 15 e 16. Nestes já se pode ver um resultado positivo durante o teste, tornando possível a utilização do modelo, porém, ainda se observa um decréscimo no final da curva.

Figura 15 – Gráfico do resultado para o treinamento com DRL com 2 camadas extras.



Fonte: elaborado pelo autor.

Figura 16 – Gráfico do resultado para o teste com DRL com 2 camadas extras.

Fonte: elaborado pelo autor.

Por fim, com a DRL com 3 camadas de 18 neurônios a mais, obtiveram-se as Figuras 17 e 18, estes foram os melhores resultados obtidos. Tendo em vista que no decorrer do período de teste a rede neural sofreu um baixo número de variações negativas. Diferente do que ocorreu com apenas duas camadas o gráfico só apresentou variações abruptas para valores positivos, sendo algo importante para aplicações reais, visto que perdas grandes desencorajariam usuários, como investidores.

Figura 17 – Gráfico do resultado para o treinamento com DRL com 3 camadas extras.

Fonte: elaborado pelo autor.

Figura 18 – Gráfico do resultado para o teste com DRL com 3 camadas extras.

Fonte: elaborado pelo autor.

Com a DNN de 3 camadas ocorreram resultados positivos, porém a rede também sofreu consideravelmente de overfiting, além do tempo considerável necessário para seu treinamento, sendo praticamente o dobro do tempo necessário para treinar 2 camadas.

7 CONCLUSÕES E TRABALHOS FUTUROS

O modelo de rede proposto neste trabalho atende as expectativas gerando resultados positivos para a previsão de séries temporais, mesmo as consideravelmente complexas. Foi mostrado a capacidade de previsão de redes de aprendizagem por reforço direto, em parceria com aprendizagem profunda e recorrente, e evidenciando esses impactos.

Dessa forma, foi de extrema importância a configuração dos parâmetros da rede, visto que eles influenciaram completamente os resultados finais. Uma questão importante neste caso é a capacidade de processamento da máquina onde será realizado o treinamento da RN e o tempo disponível. Com o treinamento concluído os dados podem ser exportados e utilizados de forma simples, podendo ser usados em dispositivos embarcados e até mesmo em aplicações *web*, sendo executadas no lado do cliente.

A principal dificuldade enfrentada pela rede foi a ocorrência de *overfiting*, visto que a rede não apresentou nenhum resultado negativo no treinamento, porém, não se pode dizer o mesmo em relação aos testes, o que acaba abrindo espaço para futuros trabalhos buscando sanar esse problema. As possíveis técnicas que podem ser utilizadas são o *dropout* de alguns neurônios da rede após o seu treinamento, algoritmos genéticos para escolha dos pesos iniciais da rede e emprego de funções de ativação nos neurônios das camadas de DNN.

Para melhorar a capacidade de previsão da rede ainda podem ser empregadas estruturas de recursividades nos neurônios das camadas de DNN, variações nos números de neurônios nas camadas internas da rede, assim aumentando o número de pesos. Melhorando capacidade da RN em identificar variações sutis nos padrões dos dados.

REFERÊNCIAS

CHEN, J. **Price Rate Of Change Indicator (ROC)**. 2001. Disponível em: <https://www.investopedia.com/terms/r/rateofchange.asp>. Acesso em: 22 mai. 2022.

DENG, Y.; BAO, F.; KONG, Y.; REN, Z.; DAI, Q. **Deep Direct Reinforcement Learning for Financial Signal Representation and Trading**. IEEE Transactions On Neural Networks And Learning Systems, v. 28, 2017.

GERON, A. **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow.** Rio de Janeiro: Alta Books, 2019.

GOOGLE BRAIN TEAM. **TensorFlow Core**. 2015. Disponível em: <https://www.tensorflow.org/overview>. Acesso em: 10 jan. 2022.

HOCHLEITNER, A. **Análise comparativa entre métodos estatísticos tradicionais e baseados em inteligência artificial no problema de previsão de demanda**. Dissertação (Graduação em Engenharia de Produção Mecânica) — Centro Tecnológico da Universidade Federal de Santa Catarina, Florianópolis, 2021.

IGNÁCIO, L. V. R.; RIBEIRO, L. G. A.; VEIGA, C. P.; BITTENCOURT, J. T. **O uso de inteligência artificial para a previsão do preço do petróleo**. **Rev. Espacios.**, v. 38, n. 24, p. 3, 2017.

MOODY, J.; SAFFELL, M. **Learning to trade via direct reinforcement**. IEEE Transactions On Neural Networks And Learning Systems, v. 12, n. 4, p. 875–889, 2001.

MOODY, J.; WU, L.; LIAO, Y.; SAFFELL, M. **Performance functions and reinforcement learning for trading systems and portfolios**. J. Forecasting, v. 17, n. 5-6, p. 441–470, 1998.

MOREIRA, V. R. F. **Controle inteligente de um robô móvel omnidirecional com tomada de decisão utilizando aprendizagem por reforço**. Dissertação (Mestrado em Engenharia Elétrica e de Computação) — Programa de Pós-Graduação em Elétrica e de Computação (PPGEEC) da Universidade Federal do Rio Grande do Norte, Natal, 2021.

NEELY, C. J.; RAPACH, D. E.; TU, J.; ZHOU, G. **Forecasting the equity risk premium: The role of technical indicators**. Management Science, v. 60, 2014.

NIELSEN, A. **Análise prática de séries temporais: Predição com Estatística e Aprendizado de Máquina**. Rio de Janeiro: Alta Books, 2021.

PIRAS, A. **THE NEW DIGITAL REVOLUTION, DEEP LEARNING**. 2022. Disponível em: <https://en.difesaonline.it/evidenza/cyber/la-nuova-rivoluzione-digitale-il-deep-learning>. Acesso em: 16 mar. 2022.

PUTERMAN, M. L. **Markov Decision Processes: Discrete Stochastic Dynamic Programming. Hoboken**. New Jersey: [s.n.], 2005.

RAN, A. **yfinance: Download market data from Yahoo! Finance API. OS Independent. PyPI**. 2019. Disponível em: <https://github.com/ranaroussi/yfinance>. Acesso em: 04 jul. 2022.

TESAURO, G. **TD-Gammon, a self-teaching backgammon program, achieves master-level play.** Neural Comput, v. 6, n. 2, p. 215–219, 1994.

YAHOO. **Python documentation**. 2001. Disponível em: <https://docs.python.org/>. Acesso em: 10 abr. 2022.

YAHOO. **Yahoo!Finance. Vale S.A.** 2022. Disponível em: <https://finance.yahoo.com/quote/VALE3.SA?p=VALE3.SA&.tsrc=fin-srch>. Acesso em: 10 abr. 2022.

APÊNDICE A – ALGORÍTMOS PARA CRIAÇÃO DOS INDICADORES

```
1  def liquidez(df: pd.DataFrame, window: int) :
2      """ Realiza o cálculo da variação em uma janela de tempo"""
3      liquid = df.rolling(window).sum().shift(1)
4      return liquid
5
6  def spread(high, low):
7      """Realiza o cálculo da variação entre o valor máximo e
          mínimo durante os dias"""
8      amplitude = high-low
9      return amplitude
10
11 def flow(high, low, close):
12     """Realiza o cálculo da diferença entre as variações entre
          o preço mínimo e o de fechamento e o preço máximo e o
          fechamento"""
13     fluxo_de_baixa = high-close
14     fluxo_de_alta = close-low
15     fluxo = fluxo_de_alta - fluxo_de_baixa
16     return fluxo
17
18 def amplitude(high, low, window):
19     Donchian = ta.donchian(high, low, lower_length=window,
20                            upper_length=window, offset=1)
21     return Donchian[f'DCU_{window}_{window}'] -
          Donchian[f'DCL_{window}_{window}']
```