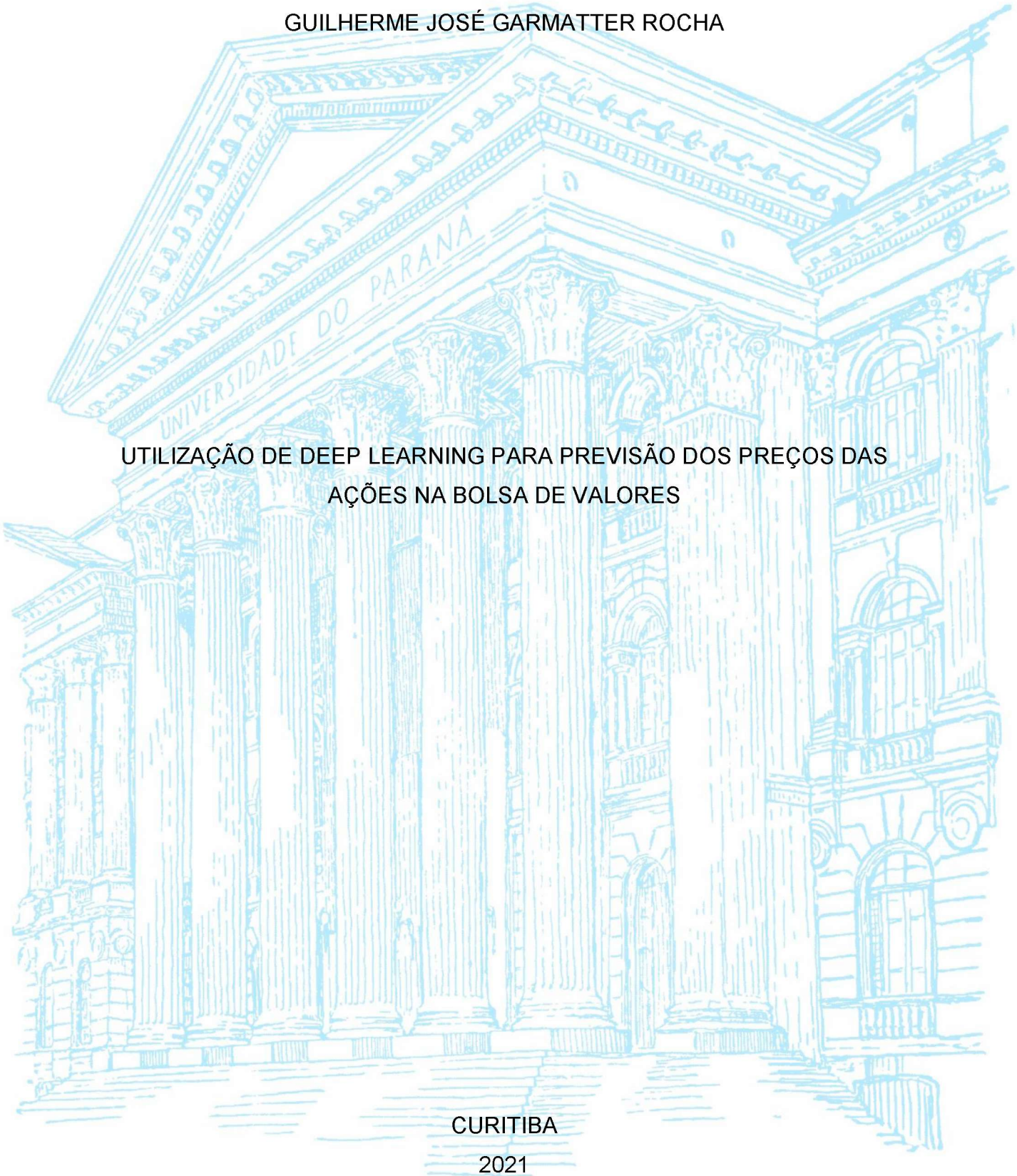
UNIVERSIDADE FEDERAL DO PARANÁ

GUILHERME JOSÉ GARMATTER ROCHA



UTILIZAÇÃO DE DEEP LEARNING PARA PREVISÃO DOS PREÇOS DAS AÇÕES NA BOLSA DE VALORES

CURITIBA

2021

GUILHERME JOSÉ GARMATTER ROCHA

UTILIZAÇÃO DE DEEP LEARNING PARA PREVISÃO DOS PREÇOS DAS AÇÕES NA BOLSA DE VALORES

Monografia apresentada como requisito parcial à obtenção do título de Bacharel em Ciências Econômicas, Curso de Graduação em Ciências Econômicas, Setor de Ciências Sociais Aplicadas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Adalto Acir Althaus Junior

CURITIBA

2021

TERMO DE APROVAÇÃO

GUILHERME JOSÉ GARMATTER ROCHA

UTILIZAÇÃO DE DEEP LEARNING PARA PREVISÃO DOS PREÇOS DAS AÇÕES NA BOLSA DE VALORES

Monografia apresentada como requisito parcial à obtenção do título de Bacharel em Ciências Econômicas, Curso de Graduação em Ciências Econômicas, Setor de Ciências Sociais Aplicadas, Universidade Federal do Paraná, pela seguinte banca examinadora:

Prof. Dr. Adalto Acir Althaus Junior
Setor de Ciências Sociais Aplicadas
Universidade Federal do Paraná

Prof. Drª Dayane Rocha de Pauli
Setor de Ciências Sociais Aplicadas
Universidade Federal do Paraná

Prof. Dr. Ricardo Torres
Setor de Ciências Sociais Aplicadas
Universidade Federal do Paraná

Curitiba, 29 de julho de 2021

AGRADECIMENTOS

É sempre muito bonito agradecer a todas as pessoas que fizeram parte de um processo de entrega de TCC, mesmo as que possam ter ajudado das formas que pareçam insignificantes e/ou indiretas. No entanto, acredito que ao agradecer a todos de forma igualitária estaria menosprezando rudemente os que me ajudaram das formas mais relevantes.

Desta forma, gostaria de agradecer primeiramente ao meu pai, que foi responsável por bancar toda minha educação sozinho. Jamais teria conseguido chegar até a entrega deste TCC se não fosse pelo apoio incondicional dele.

Em segundo lugar gostaria de agradecer à UFPR e ao corpo discente responsável por colaborar com à minha formação de forma relevante. Em especial aos professores Adalto Acir Althaus Junior, José Guilherme Silva Vieira, Rodrigo Leite Kremer, Dayane Rocha de Pauli, Marcelo Luiz Curado e Wellington da Silva Pereira.

Gostaria de agradecer também a Mariana Michels Fontoura, com quem namoro a quase 6 anos, por me incentivar a finalizar o curso de economia e por me apoiar sempre que passei por dificuldades durante minha formação. Por último, gostaria de agradecer também a Deus.

RESUMO

Este trabalho teve como objetivo a criação de um algoritmo capaz de realizar previsões sobre a variação de preços futura de ações negociadas na bolsa de valores brasileira por meio da utilização da técnica de aprendizado de máquina denominada deep learning. Foram utilizados dados históricos de preço e volume para alimentar o modelo em conjunto com uma rede neural híbrida. Ao final do trabalho foi verificado que embora o modelo não tenha obtido resultados significativos para prever o valor da variação de preço no próximo dia útil para as ações testadas, um teste de investimento mostrou que o modelo pode ser útil para prever a tendência na qual os preços se movimentarão com um dia de antecedência, sobretudo quando o modelo é treinado com os dados mais recentes. Para os papéis analisados, o modelo resultou em uma estratégia de investimento com resultados superiores a de *buy and hold* no melhor caso e equivalente a esta mesma estratégia, no pior caso.

Palavras-chave: Deep Learning, Machine Learning, ações, bolsa de valores, CNN, LSTM, CNN-LSTM, trade, rede neural híbrida.

ABSTRACT

The present work had the objective to create an algorithm capable of predicting the price variation of stocks by utilizing a machine learning technique called deep learning. The model was fed with historical price and volume data of stocks traded on the Brazilian stock market in conjunction with a hybrid neural network. In the end, it was verified that even though the model didn't obtain significant results for predicting the price variation for the next business day for the stocks tested, an investment test showed that the model could be capable of predicting the direction in which the prices would move with a day prior information. The results were better when the model was trained using more recent data. For the analyzed stocks the model resulted in an investment strategy with superior results than a buy and hold strategy in the best case scenario and equivalent results in the worst case.

Key-words: Deep learning, machine learning, stocks, stock market, CNN, LSTM, CNN-LSTM, trading, hybrid neural network.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

Este trabalho teve como objetivo a criação de um algoritmo para realizar previsões sobre os preços futuros de ações negociadas na bolsa de valores por meio da utilização da técnica de aprendizado de máquina denominada *deep learning*.

Apesar do comportamento dos preços no mercado de ações parecer caótico, existem aqueles que tentam prever os movimentos de preço no futuro utilizando como base padrões de comportamento nos preços verificados no passado. Este tipo de análise é chamada de análise técnica. A hipótese por trás deste trabalho é de que caso este tipo de análise seja possível, o *deep learning* pode ser capaz de identificar padrões de comportamento ocultos nas séries históricas para se obter a variação de preço futura de determinado ativo.

Isso se dá porque diferentemente dos métodos tradicionais de modelagem e de cálculo de regressão, em que o pesquisador precisa definir manualmente as variáveis que serão utilizadas pelo modelo e seus pesos correspondentes, o método utilizado, chamado *deep learning*, é uma técnica de aprendizado de máquina que ensina os computadores a fazerem algo que ocorre naturalmente com os seres humanos: o aprendizado por meio de exemplos. Muito embora esta técnica possua um potencial disruptivo, sua aplicação para o mercado de capitais ainda é recente, em especial para o brasileiro.

No final deste trabalho, espera-se que o modelo obtido possua grau significativo de assertividade ao prever a variação do preço de determinada ação em "D+1", a partir de dados de preço e volume disponíveis no dia "D".

A respeito da estrutura deste trabalho, o segundo capítulo explicará o que é o *deep learning*, suas aplicações e as variáveis consideradas relevantes para alimentar o modelo. No terceiro capítulo abordamos a metodologia utilizada para construção do modelo, no quarto capítulo analisamos os resultados obtidos e no 5 capítulo trazemos as conclusões.

2 REFERENCIAL TEÓRICO

Neste capítulo abordamos melhor o que é o deep learning, algumas das suas aplicações e vantagens. Também analisaremos outras implementações do método realizadas por outros autores.

2.1 O QUE É O DEEP LEARNING?

Segundo Brownlee (2019), *deep learning* é um subcampo de *machine learning* (aprendizado de máquina) inspirado pela estrutura e funcionamento do cérebro. Essa tecnologia foi idealizada de forma simplificada já em 1943 quando Warren Pitts e Warren McCulloch criaram um modelo de computador baseado nas redes neurais do cérebro humano. Houveram avanços com relação a inteligência artificial (IA) desde esse período e não está no escopo deste trabalho explicar cada um destes avanços e seus modelos individualmente. Todavia serão explicados na seção 2.2 alguns dos métodos de *deep learning* que podem ser utilizados para a previsão de uma série temporal, visto que estes foram utilizados para o desenvolvimento deste trabalho.

Em termos práticos, o *deep learning* é um método de aprendizado de máquina que se utiliza de uma rede neural para aprender por meio de exemplos. Similarmente aos humanos, este algoritmo é capaz de aprender por meio da experiência repetindo o treinamento diversas vezes e melhorando os resultados a cada tentativa.

Muito embora essa tecnologia não seja nova, sua popularização só foi possível após alguns avanços tecnológicos. Um dos fundadores do Google Brain[1], Andrew Ng (2013) na palestra chamada "*Deep Learning, Self-Taught Learning and Unsupervised Feature Learning*" (*Deep Learning,* aprendizado autodidata e aprendizado não supervisionado de características) fez uma analogia interessante. O autor compara a tecnologia de *deep learning* com um foguete, considerando a capacidade computacional como o motor e a quantidade de dados disponíveis como o combustível. A partir desta analogia o crescimento de ambos estes fatores seriam necessários para fazer o foguete (*deep learning*) decolar.

Estes dois fatores – a quantidade de dados e a capacidade computacional – cresceram muito nos últimos 20 anos. As unidades de processamento de vídeo (*Graphical Processing Units*) ou GPU`s, apesar de comumente utilizadas para o processamento de gráficos de jogos para entretenimento, possibilitaram um aumento na capacidade de cálculos em paralelo para fins de pesquisa em inteligência artificial. Da mesma maneira, o

[1] Time de pesquisa da Google que eventualmente teve como resultado a implementação do deep learning em vários de seus serviços

advento da internet, do celular e dos mais variados aparelhos com acesso à internet, apesar de gerarem questionamentos com relação a privacidade, possibilitaram a coleta de dados de usuário de maneiras nunca antes vistas.

Em resumo, o que diferencia o *deep learning* de uma rede neural é o seu tamanho e sua capacidade de compreender conceitos mais abstratos a partir de um conjunto de dados simples. Sua utilização hoje é difundida e pode ser verificada em diversas aplicações como: reconhecimento de fala, reconhecimento de pessoas e animais em fotos, cálculo de rotas de GPS, tradução, direção de veículos autônomos, algoritmos de recomendação de produtos e propagandas, etc.

2.2 VANTAGENS DO DEEP LEARNING PARA PREVISÃO DE SÉRIES TEMPORAIS

De acordo com Mcdonald (2018), uma das maiores vantagens do *deep learning* é o fato de que este método é capaz de construir uma função que mapeia as entradas (*input*s) para as saídas (o*utput*s). Isto é, dada uma função Y = F(x), se tivermos dados suficientes para x e Y, podemos teoricamente descobrir qual a função F(x) para qualquer x com determinada precisão. Embora naquele momento Mcdonald não estivesse se referindo a problemas de séries temporais, podemos aplicar seu entendimento para este tipo de problema.

De forma mais prática, digamos que um pesquisador queira criar um modelo econométrico convencional que defina o preço de uma ação no futuro. Para esta situação o pesquisador precisaria definir quais as variáveis relevantes ao modelo, escolher ou elaborar o próprio modelo econométrico a ser utilizado, comparar o resultado deste modelo com o preço verificado a fim de se obter o erro e avaliar a eficácia do seu modelo.

> Traditional approaches for modeling sequential data include the estimation of parameters from an assumed time-series model, such as autoregressive models (Lütkepohl, 2005) and Linear Dynamical Systems (LDS) (Luenberger, 1979), and the popular Hidden Markov Model (HMM) (Rabiner and Juang, 1986). The estimated parameters can then be used as features in a classifier to perform classification. However, more complex, high-dimensional, and noisy real-world time-series data cannot be described with analytical equations with parameters to solve since the dynamics are either too complex or unknown (Taylor, 2009) and traditional shallow methods, which contain only a small number of non-linear operations, do not have the capacity to accurately model such complex data. (LÄNGKVIST; KARLSSON; LOUTFI, 2014, p. 2)

Já por meio do *deep learning*, ao invés de criar/utilizar este modelo e definir os seus parâmetros manualmente, podemos dar ao computador uma série de dados que considerarmos relevantes como entrada e os resultados que queremos prever futuramente

como saída. Não precisamos nos preocupar tanto com um possível excesso de variáveis de entrada, pois o computador irá identificar quais das variáveis são úteis e em qual medida a fim de se obter o resultado desejado.

> Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features. The objective is to make these higher-level representations more abstract, with their individual features more invariant to most of the variations that are typically present in the training distribution, while collectively preserving as much as possible of the information in the input (BENGIO, 2012, p. 1)

Desta forma, conforme o treinamento do modelo for sendo realizado, o computador definirá quais dados são os melhores para que se chegue aos valores dados como solução. Ficará também à critério do computador se ele utilizará ou não todos os dados fornecidos. Dito isso, a falta de dados relevantes ou sua má preparação poderá afetar negativamente o desempenho do modelo. Para o fim de aumentar a precisão, devemos adicionar variáveis que podem ser importantes para a definição do problema a ser resolvido e realizar seu devido preparo antes de alimentar o modelo.

2.3 MÉTODOS DE DEEP LEARNING PARA PREVISÃO DE SÉRIES TEMPORAIS

Abordaremos 3 dos principais tipos de redes neurais existentes que podem ser utilizadas para resolução de problemas envolvendo séries temporais: *multilayer perceptrons* (MLP), *convolutional neural networks* (CNN) e *recurrent neural networks* (RNN).

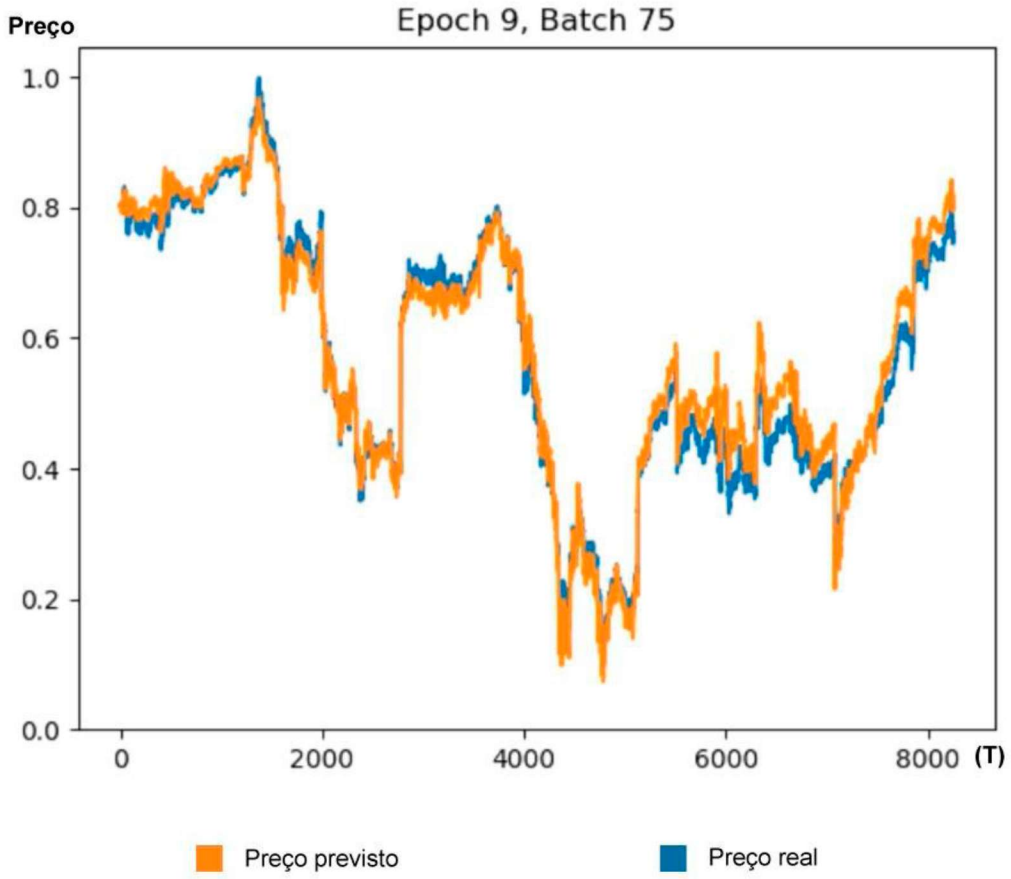
2.3.1 MULTILAYER PERCEPTRON (MLP)

As redes neurais MLP geram uma função aproximada que mapeia as variáveis de entrada para as variáveis de saída. Devido a sua simplicidade podem ser utilizadas em diversas aplicações e em conjunto com outros tipos de rede a fim de se obter melhores resultados.

Segundo Brownlee (2018), este tipo de rede neural é útil para realizar previsões em séries temporais devido a sua capacidade de resistir a interferências e *outliers* nos dados, podendo também realizar previsões mesmo na falta de alguns destes. Outra vantagem deste tipo de rede neural é a possibilidade de mapear tanto funções lineares quanto não lineares.

Este tipo de rede em seu formato puro já foi utilizado por Heinz (2018) para previsão de preços do índice S&P 500. Em seu modelo, Heinz tenta prever o valor do índice para o minuto seguinte utilizando os preços das 500 ações que compõe este índice no minuto anterior como base. Conforme mostrado no Gráfico 1, Heinz consegue obter resultados relativamente precisos (em laranja) versus os valores reais apresentados pelo índice (em azul).

GRÁFICO 1 - PREVISÕES DO MODELO DE HEINZ



FONTE: HEINZ (2018)

Apesar destes resultados parecerem promissores, este modelo possui algumas limitações. Autores como Ganegedara (2020) apontam que a previsão de preços no mercado de ações utilizando como alvo apenas preços no período seguinte aos dados podem gerar resultados que se comportam como médias móveis, possuindo pouca utilidade para realizar previsões.

> Practically speaking, you can't do much with just the stock market value of the next day. Personally what I'd like is not the exact stock market price for the next day, but *would the stock market prices go up or down in the next 30 days*. Try to do this, and you will expose the incapability of the EMA(Exponential Moving Average) method. (GANEGEDARA, 2020, não p.)

Já Flovik (2018) aponta que dados de séries temporais tendem a ser auto correlacionados e que isso pode gerar um problema onde o modelo simplesmente usa a última observação fornecida como previsão para o próximo período.

> Time series data tend to be correlated in time, and exhibit a significant autocorrelation. In this case, that means that the index at time "t+1" is quite likely close to the index at time "t". As illustrated in the above figure to the right, what the model is actually doing is that when predicting the value at time "t+1", it simply uses the value at time "t" as its prediction (often referred to as the persistence model). (FLOVIK, 2018, não p.)

Em seguida, Flovik sugere outra abordagem, onde o objetivo do modelo é prever a diferença entre os valores em cada período como forma de minimizar este tipo de problema:

> Defining the model to predict the difference in values between time steps rather than the value itself, is a much stronger test of the models predictive powers. In that case, it cannot simply use that the data has a strong autocorrelation, and use the value at time "t" as the prediction for "t+1". Due to this, it provides a better test of the model and if it has learnt anything useful from the training phase, and whether analyzing historical data can actually help the model predict future changes. (FLOVIK, 2018, não p.)

A seguir abordamos outros tipos de redes neurais que podem trabalhar em conjunto com uma rede MLP, incrementando sua capacidade para resolução de problemas que envolvem séries temporais.

2.3.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

As redes neurais CNN foram especialmente desenvolvidas para o processamento de imagens e são comumente utilizadas para *computer vision*[2]. Apesar disso, também podem ser aplicadas com sucesso para problemas de séries temporais. Uma das maiores vantagens deste tipo de rede neural é a sua capacidade de trabalhar com dados “crus", ou seja, sem processamento prévio.

Este tipo de rede neural consegue realizar automaticamente a extração de informações nos dados de entrada a fim de resolver o problema proposto, independentemente de como estes dados estão distribuídos. Devido a esta característica,

[2] *Computer vision* é um campo de estudo que tem por objetivo desenvolver tecnologias que ajudem computadores a enxergar e entender o conteúdo de imagens e vídeos. É utilizado em aplicações de direção autônoma, por exemplo.

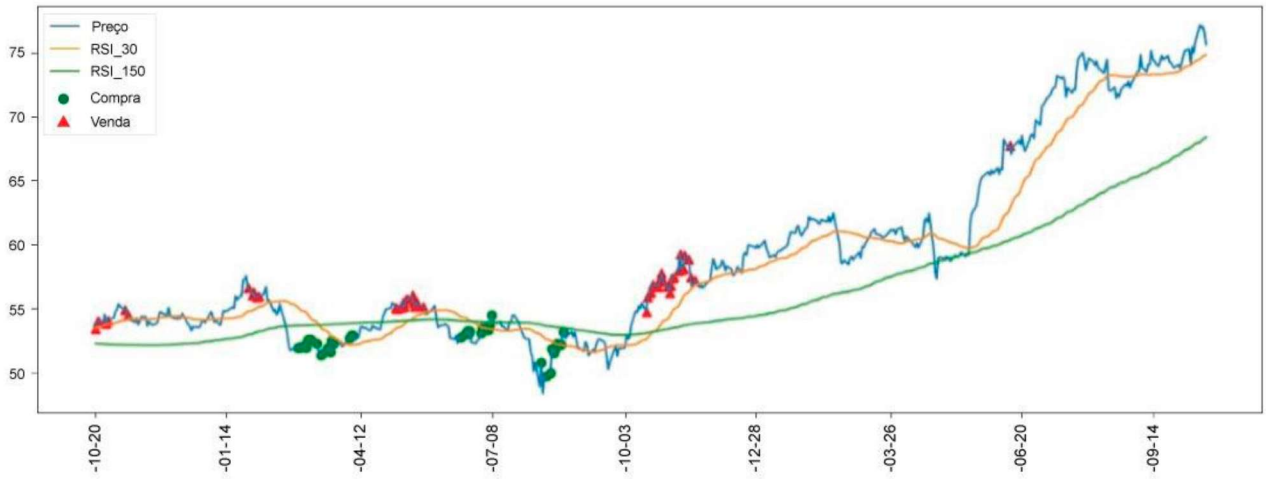
este tipo de rede neural é comumente utilizado para o processamento de imagens. Por exemplo: digamos que o pesquisador queira identificar a existência de um pássaro em uma foto. É fato que o pássaro não estará exatamente nas mesmas posições em todas as fotos. Ele poderá estar em diferentes ângulos, ter diferentes iluminações, estar com as asas abertas ou ainda estar parcialmente atrás de algum objeto. Desta forma, a CNN irá aprender procurar nos dados de entrada características que possam identificar um pássaro a fim de realizar o objetivo proposto. Este aprendizado é chamado de *feature learning*, e pode ser utilizado para interpretação e identificação de características relevantes nos dados a fim de melhorar o desempenho de uma rede neural.

Esta rede, sozinha, não é capaz de gerar um número inteiro – como a variação de preço de uma ação, por exemplo – como saída. Ao invés disso, ela geralmente resulta na classificação dos dados em categorias que o pesquisador define na criação do modelo como: Certo, errado, cachorro, coelho, carro, verdadeiro, falso, etc. Apesar disso, alguns autores conseguiram implementar com relativo sucesso este tipo de rede neural para o uso no mercado de ações. Fornecendo dados históricos e indicadores técnicos de mercado Sezer e Ozbayoglu (2018) fizeram com que sua rede neural classificasse estes dados em 3 categorias: comprar, vender ou segurar. Para isso os autores precisaram classificar os dados históricos manualmente em pontos de compra, venda e de segurar ações a fim de treinar o modelo por meio de exemplos. Utilizando uma janela de 11 dias, eles classificaram o ponto onde o preço era o mais alto como sendo um ponto de venda, o mais baixo como sendo um ponto de compra e o restante como pontos onde o modelo deveria segurar os papéis. Os autores concluem que os resultados obtidos foram superiores aos de uma estratégia de *buy and hold*[3] em longos períodos, porém não disponibilizaram o código fonte do modelo.

Inspirado pelo modelo de Sezer e Ozbayoglu (2018), Nayak (2020) tentou replicar os resultados obtidos pelos autores. Após reconstruir o código com os dados fornecidos pelos autores e realizar algumas adaptações, o autor treinou seu modelo para identificar pontos de compra, venda e de segurar as ações, obtendo os resultados mostrados no Gráfico 2.

[3] Estratégia de investimento que significa “comprar e segurar” em tradução livre. Consiste em comprar um ativo e permanecer com ele por tempo indeterminado.

GRÁFICO 2 - RESULTADOS DO MODELO DE NAYAK



FONTE: NAYAK (2020)

O autor finaliza seu trabalho comentando que o método utilizado para classificar os momentos de compra e venda utilizado os resultados, apesar de se mostrarem relativamente precisos, não resultaram em grandes lucros em seus testes. Em seguida ele sugere a busca por outras formas de classificação para melhorar estes resultados.

2.3.3 RECURRENT NEURAL NETWORKS (RNN)

As redes neurais RNN, em especial as de arquitetura *long short-term memory* (LSTM), são amplamente utilizadas para a resolução de problemas envolvendo séries temporais. Isso se dá devido a sua capacidade de diferenciar a sequência na qual as variáveis de entrada e de saída estão dispostas, o que não é suportado nativamente pelas redes neurais MLP e CNN.

> This capability of LSTMs has been used to great effect in complex natural language processing problems such as neural machine translation where the model must learn the complex inter-relationships between words both within a given language and across languages in translating from one language to another. This capability can be used in time series forecasting. In addition to the general benefits of using neural networks for time series forecasting, recurrent neural networks can also automatically learn the temporal dependence from the data. (BROWNLEE, 2018, p. 7)

Exemplificando melhor, digamos que um pesquisador queira criar uma rede neural que faça previsões da variação dos preços no mercado de ações. Enquanto as redes MLP e CNN veriam as variáveis escolhidas como um todo e desordenadas, uma rede LSTM seria capaz de compreender a relação entre as variáveis fornecidas ao longo do tempo.

Desta forma, o modelo é capaz de mapear uma função que transforma as entradas, ao longo do tempo, em saídas.

Um trabalho que obteve relativo sucesso em exemplificar a previsão de preços futuros no mercado de ações utilizando redes LSTM foi o de Aungiers (2018). Neste trabalho, Aungiers utilizou dados de preço de fechamento e volume diários do índice S&P 500 de 50 dias anteriores para tentar prever os preços futuros. Em seu modelo, Aungiers, ao invés de tentar prever apenas o preço no período imediatamente posterior aos dados de entrada, tenta prever os preços para os próximos 50 dias. Os resultados podem ser visualizados no Gráfico 3.

GRÁFICO 3 - RESULTADOS DO MODELO DE AUNGIERS

FONTE: AUNGIERS (2018)

Apesar dos resultados apresentados se mostrarem capazes de prever algumas tendências, Aungiers conclui que existem limitações no uso de redes LSTM em seu formato clássico como o que foi utilizado para o uso em séries temporais financeiras devido ao caráter não estacionário dos dados. O autor ainda sugere o uso de redes híbridas em combinação com redes LSTM para que possam se obter melhores resultados.

2.3.4 REDES NEURAIS HÍBRIDAS

Em adição aos modelos mais comuns de redes neurais, é possível utilizar ainda mais de um tipo de rede ao mesmo tempo, combinando as vantagens que cada tipo de rede oferece.

> Hybrid neural networks, which combines the strengths of various neural networks, are receiving increasing interest in the computer vision domain, such as image captioning [Mao et al., 2014; Vinyals et al., 2015; Donahue et al., 2015], image classification [Wang et al., 2016a], action recognition [Ballas et al., 2015; Donahue et al., 2015] and so on. But efficient exploitation of such hybrid architectures has not been well studied for time series data, especially the trend forecasting problem. (LIN; GUO; ABERER, 2017, p. 3)

Embora este ainda seja um campo relativamente pouco explorado para previsão de séries temporais, outros autores, como Yitong Ren (2019), obtiveram sucesso em utilizar redes híbridas CNN-LSTM. Em seu trabalho, Ren utiliza uma rede neural híbrida para prever o uso de energia elétrica de uma casa para as próximas 24 horas utilizando os dados das 24 horas anteriores, obtendo resultados satisfatórios. Já autores como LIU, ZHANG e MA (2017) utilizaram uma rede neural híbrida CNN-LSTM para classificar os dados em pontos de compra e venda e concluem terem obtido resultados robustos e lucrativos, porém, não disponibilizaram o código fonte de seu modelo.

2.4 VARIÁVEIS RELEVANTES PARA ALIMENTAR O MODELO

Para que o modelo de *deep learning* seja treinado e possa aprender por meio de exemplos, é necessário fornecer ao modelo uma série de dados históricos do ativo em questão e o resultado/variação de preço final deste ativo ao final de cada uma destas séries históricas. De outra maneira, para cada conjunto de dados X, deve-se fornecer um resultado Y. Assim, a rede poderá buscar por padrões de comportamento nos dados e tentar realizar previsões com base nos dados fornecidos. Portanto, as variáveis fornecidas ao modelo devem ter alguma relação de causalidade com os preços. Dentre as principais formas de análise existentes para previsão dos preços das ações, existem duas que se destacam: A análise fundamentalista e a análise técnica.

> A Escola Fundamentalista é compreendida como o conjunto de instrumentos que auxiliam a decisão de investimento, por meio de dados das demonstrações financeiras da empresa, para realizar a projeção do fluxo de caixa descontado, empregando uma taxa de desconto que gera o valor presente do ativo, e a partir disto, determinar o valor correto do ativo financeiro.
> A Escola Técnica parte do princípio de que padrões de comportamentos anteriores, dos ativos financeiros, tendem a repetir-se no futuro. Os adeptos desta escola não levam em consideração intervenções externas para precificar ativos. Preocupam-se com a tendência de movimento dos preços, não sendo significante a performance

> da empresa emissora dos títulos, mas sim o modo como o mercado percebe o potencial de valorização do ativo. (GUARNIERI, 2006, p. 17)

Dada a forma de funcionamento do *deep learning*, onde se treina o modelo utilizando dados pré-existentes para tentar prever o comportamento de determinada variável no futuro, pode-se observar uma certa compatibilidade entre este método e ambas as análises técnica e fundamentalista. Ambos os métodos buscam informações nos dados históricos a fim de realizar inferências sobre o valor futuro do ativo sendo analisado.

Essa compatibilidade também pode ser verificada em trabalhos como o de Nayak (2020) e Aungiers (2018), já citados anteriormente. Além disso, outros trabalhos, como o de Yibin Ng (2019), obtiveram sucesso ao utilizar dados técnicos como datas, preços de abertura, preços de fechamento, preço máximo/mínimo do dia e volume negociado para realizar previsões de preço de ações no período seguinte aos dados. Já Chong, Han e Park (2017) utilizaram apenas os retornos passados para alimentar uma rede neural MLP e superam os resultados obtidos por um modelo autorregressivo. Neste caso, porém, admitem que o uso de variáveis adicionais como preço de derivativos ligados ao ativo analisado e volume negociado poderiam influenciar positivamente seus resultados.

3 METODOLOGIA

Conforme especificado anteriormente, este trabalho tem por objetivo prever a variação de preço de uma determinada ação em "D+1", a partir de dados de preço e volume disponíveis no dia "D". A fim de se obter os dados dos ativos no dia "D" foi utilizado o site *Yahoo Finance*, que disponibiliza estes de forma gratuita.

O site *Yahoo Finance* disponibiliza alguns dados históricos relevantes para realização de uma análise técnica de preços como: preço de abertura, preço de fechamento, volume negociado, preço mais alto e mais baixo praticados no dia e preço de fechamento ajustado, que desconsidera variações de preço geradas por pagamento de dividendos ou eventos de *split* e *inplit*[4] dos ativos.

Como fonte de dados para esta análise foram escolhidas arbitrariamente algumas ações que são negociadas na bolsa de valores brasileira e possuem bom histórico de negociação e liquidez como ABEV3, ITUB4, PETR4 e VALE3. No entanto, o modelo permite que outros papéis sejam utilizados.

Com estes dados em mãos foi necessário desenvolver um programa que pudesse preparar os dados para utilização no modelo, aplicar o modelo de rede neural desejado e, para fins de avaliação do modelo, demonstrar graficamente as previsões em contraste com os dados reais.

3.1 LINGUAGEM DE PROGRAMAÇÃO E BIBLIOTECAS UTILIZADAS

Para este trabalho foi utilizada a linguagem de programação "python". Esta linguagem foi escolhida pois, segundo Costa (2020), além de ser completamente gratuita, possui uma ampla variedade de bibliotecas[5] para uso em trabalhos que envolvem *deep learning* como *tensorflow*, *keras* ou *pytorch*. Dentre estas 3 bibliotecas mencionadas foi escolhida a biblioteca *keras* para elaboração do modelo devido a sua menor complexidade.

Além disso, foi necessária a utilização de outras bibliotecas para manipulação de matrizes como "pandas" e "numpy". Também foi utilizada a biblioteca "scikit-learn",

---

[4] Eventos de *split* e *inplit*, também chamados de desdobramento e grupamento de ações. Ocorrem quando uma empresa deseja aumentar ou diminuir o número de ações disponíveis no mercado sem alterar seu valor patrimonial. Esses eventos fazem com que os preços das ações dobrem (*inplit*) ou diminuam pela metade (*split*).

[5] Bibliotecas são um conjunto de funções pré-escritas por outros programadores que resolvem determinados problemas. Por exemplo: *tensorflow* é a biblioteca criada pelo Google para que outras pessoas possam utilizar *deep learning* sem que seja necessário escrever o código do zero.

específica para a preparação de dados destinados a *machine learning*, e a biblioteca "matplotlib", destinada a apresentação gráfica dos resultados obtidos.

3.2 VARIÁVEIS UTILIZADAS

Além dos dados iniciais fornecidos pelo site *Yahoo Finance*, foram adicionadas as variáveis "ticker" e "change". Estas variáveis representam o nome do ativo ao qual pertencem os dados e a variação percentual do preço verificada em relação ao dia anterior, respectivamente. A variável "ticker" foi adicionada para que o modelo pudesse distinguir possíveis comportamentos nos preços específicos para cada ativo. Já a variável "change" foi adicionada para servir como alvo do modelo, tendo em vista que a utilização de preços do período seguinte como alvos podem gerar problemas de auto correlação, conforme exposto por Flovik (2018).

Para o treinamento do modelo também foram utilizados os dados históricos de mais de um papel por vez. Este procedimento se fez necessário pois o uso de uma baixa quantidade de dados resulta em um modelo que se adapta para prever apenas os dados fornecidos na fase de treinamento, ou seja, do passado, ao invés de generalizar regras que podem ser aplicadas para o futuro (ex-post). Este fenômeno também é conhecido como *overfitting* ou "sobreajuste". Uma outra analogia possível seria que o modelo estaria "decorando" os dados do passado (de treinamento) para diminuir o erro, o que não gera um modelo capaz de prever dados no futuro.

3.3 PREPARAÇÃO DOS DADOS

Para elaboração do modelo foi primeiramente necessário preparar os dados de preço, volume e data que serão utilizados. Isso foi efetuado utilizando a biblioteca python "pandas", que é própria para a manipulação de matrizes.

FIGURA 1 - EXEMPLO DOS DADOS EM SUA FORMATAÇÃO ORIGINAL

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	05/01/2000	0,520882	0,520882	0,520882	0,520882	0,334178	985
3	06/01/2000	0,494478	0,494478	0,494478	0,494478	0,317238	227
4	07/01/2000	0,494478	0,494478	0,494478	0,494478	0,317238	151
5	10/01/2000	0,494478	0,494478	0,494478	0,494478	0,317238	1516
6	11/01/2000	0,494478	0,494478	0,494478	0,494478	0,317238	3791
7	12/01/2000	0,481293	0,481293	0,481293	0,481293	0,308779	1137

FONTE: AUTOR (2021)

Só é possível utilizar *deep learning* em dados estritamente numéricos. Caso os dados não estejam neste formato, é necessário primeiro convertê-los. Assim, foi necessário separar os dados de data que vieram no formato "dd-mm-aaaa" para que restassem apenas números em células separadas. A fim de evitar que o modelo relacione algum comportamento nos preços com um ano específico removemos a variável "ano" dos dados. Também adicionamos a variável "ticker", que representa o nome do ativo que estamos analisando e a variável "change", que é a variação percentual no preço de fechamento ajustado (*Adj Close*) do ativo sendo analisado.

Segundo Mushailov (2021, não p.) "*A non-stationary series will introduce more error in predictions and force errors to compound faster*". Devido a este fenômeno, foi realizado um processo de diferenciação nos dados para torná-los estacionários e remover quaisquer tendências presentes neles. Isto foi feito calculando a variação percentual diária para todas as colunas com exceção das colunas "ticker", "dia", "mês", que já são estacionárias, e a coluna "change", que é a variação percentual da coluna "Adj Close" e, portanto, já foi diferenciada.

FIGURA 2 - EXEMPLO DOS DADOS APÓS PREPARAÇÃO

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0	1	2	3	4	5	6	7	8	9	10	11	12
2	-0,06137	-0,34143	-0,25537	-0,03852	-0,06137	-0,06137	-0,99994	-0,66667	-1	0,925	0,025	0,025	0,025
3	0,13199	-0,18203	-0,05707	0,151704	0,13199	0,13199	-0,99984	-0,6	-1	0,925	0,025	0,025	0,025
4	0,13199	-0,18203	-0,05707	0,151704	0,13199	0,13199	-0,99759	-0,4	-1	0,925	0,025	0,025	0,025
5	0,13199	-0,18203	-0,05707	0,151704	0,13199	0,13199	-0,9994	-0,33333	-1	0,925	0,025	0,025	0,025
6	0,030278	-0,26587	-0,16138	0,051645	0,030278	0,030278	-0,99993	-0,26667	-1	0,925	0,025	0,025	0,025
7	0,158118	-0,16049	-0,03029	0,177402	0,158113	0,158118	-0,99987	-0,2	-1	0,925	0,025	0,025	0,025
8	0,209828	-0,11786	0,022755	0,228282	0,209833	0,209828	-0,99745	-0,13333	-1	0,925	0,025	0,025	0,025
9	0,38653	0,027806	0,203959	0,402107	0,386529	0,38653	-0,99905	0,066667	-1	0,925	0,025	0,025	0,025
10	0,10815	-0,20168	-0,08152	0,128256	0,108154	0,10815	-0,99998	0,666667	-1	0,925	0,025	0,025	0,025
11	-0,03592	-0,32045	-0,22927	-0,01348	-0,03592	-0,03592	-0,99926	0,733333	-1	0,925	0,025	0,025	0,025
12	-0,1192	-0,38909	-0,31466	-0,09539	-0,11919	-0,1192	-0,99999	-0,8	-0,81818	0,925	0,025	0,025	0,025
13	0,13199	-0,18203	-0,05707	0,151704	0,13199	0,13199	-0,9952	-0,6	-0,81818	0,925	0,025	0,025	0,025
14	0,13199	-0,18203	-0,05707	0,151704	0,13199	0,13199	-0,9999	-0,53333	-0,81818	0,925	0,025	0,025	0,025

FONTE: AUTOR (2021)

Após estas operações os dados de entrada ficaram conforme exemplo na Figura 2. Também foi necessário remover algumas linhas de dados que estavam vazias, pois isso pode afetar negativamente o aprendizado do modelo. Para que o modelo possa aprender com os dados de mais de um ativo, o programa também acrescenta os dados dos outros ativos sendo analisados para que fiquem em uma única matriz.

Para avaliar corretamente os resultados obtidos pelo modelo é necessário separar os dados em uma subseção para treinamento e outra para testes. Os dados de treino são utilizados para ensinar o modelo a fazer previsões com base nos dados passados. Já os dados de teste são utilizados para avaliar se o modelo consegue ou não prever a variação dos preços em dados que ele nunca observou. Para este trabalho os dados de treino foram definidos como todos os dados entre 01/01/2000 e 30/12/2019 e os dados de teste como os dados entre 01/01/2020 a 15/07/2021. Esta data foi selecionada pois o uso de dados de teste de um período menor resulta em dados não representativos o suficiente para avaliação do aprendizado do modelo. Por outro lado, o uso de um conjunto de dados de teste maior pode resultar em previsões piores, já que os dados em que o modelo foi treinado ficam cada vez mais distantes da data em que se deseja prever os preços.

Após a organização inicial dos dados foi necessário transformá-los para que eles pudessem ser utilizados em nosso modelo. Primeiramente escalamos todos dados para valores entre -1 e 1 por meio das funções *"fit_transform"* e *"transform"* disponíveis na biblioteca "sklearn". Em seguida transformamos a variável "ticker" em um valor numérico por meio da função "LabelBinarizer", também disponível na biblioteca "sklearn".

Feito isso separamos os dados em levas para serem alimentados no modelo. O modelo recebe matrizes 60x13(60 dias com 13 variáveis por dia) como entrada e uma variável como saída/resultado. Esta variável corresponde a variação percentual do preço de fechamento ajustado do dia subsequente aos dados históricos fornecidos. Para esta aplicação, uma leva (também chamada de *batch*) foi equivalente a 400 matrizes 60x13 e seus respectivos resultados. Este número de levas foi escolhido pois foi o maior valor suportado pelo equipamento utilizado pelo autor. Para esta separação em levas utilizamos a função "TimeseriesGenerator" pertencente a biblioteca "keras".

3.4 MODELO UTILIZADO E TREINAMENTO

Para este trabalho foi utilizada uma rede híbrida CNN-LSTM pois este tipo de rede neural consegue combinar as vantagens que cada tipo de rede individual oferece (LIN; GUO; ABERER, 2017). Neste caso específico, a rede deve ser capaz de identificar padrões de comportamento nos dados usando uma rede CNN. Após isto, a rede analisa a ordem de ocorrência destes padrões ao longo do tempo com uma rede LSTM. Em seguida este valor é repassado para uma rede MLP que retorna uma previsão da variação percentual no preço do ativo para o próximo período.

Para o treinamento da rede neural, o programa inicialmente preenche os pesos e valores de todas as células que compõe a rede com números aleatórios. Em seguida a rede realiza uma previsão usando estes parâmetros aleatórios e obtém um resultado. O programa então calcula o erro entre este valor previsto e o valor real utilizando uma função de erro (também chamada de *l*oss). Para este caso, foi utilizada a média dos quadrados das diferenças entre o valor que foi previsto e o real (método *mean square error*) como função de erro devido a sua compatibilidade com o problema de predição de séries temporais. Em seguida, a rede neural calcula em qual direção ela deve ajustar os pesos de cada célula que a compõe para que este erro seja diminuído. Isto é feito automaticamente pela rede neural por meio do cálculo de um gradiente da função de erro para cada leva de dados e não está no escopo deste trabalho explicar seu funcionamento mais detalhadamente. Após isso, a rede neural adapta estes pesos de forma a tentar diminuir o erro e analisa outra leva de dados. Este processo ocorre até que acabem as levas de dados disponíveis e corresponde a uma iteração (ou *epoch*). Estas iterações podem ocorrer quantas vezes forem desejadas. Para este trabalho o autor definiu o número de iterações em 500 pois após esta quantidade de iterações foi observada pouca ou nenhuma melhora nos resultados obtidos.

Um ponto que deve ser destacado a respeito deste treinamento é que treinamos o modelo com dados de janeiro de 2000 até dezembro de 2019/2020 e testamos se o modelo consegue ou não realizar previsões em um período de tempo relativamente longo (no ano de 2020/2021). Uma forma de aplicação mais precisa do modelo seria realizar seu treinamento diariamente com os dados mais recentes e comparar as previsões com os dados reais. Assim, eventuais mudanças de comportamento nos preços não afetariam o modelo de forma significativa, pois ele já estaria treinado com os dados do dia imediatamente anterior. Este tipo de análise implicaria, no entanto, em um processo de treinamento customizado onde o modelo é retreinado ao final de cada linha de dados (dia), ao invés de executar apenas um treinamento para todos os dados. Considerando que cada treinamento/execução do modelo no formato atual leva cerca de 8 minutos e que foram utilizados dados de janeiro de 2000 até dezembro de 2019 de 4 empresas diferentes, o autor avaliou que um processo de retreinamento diário tomaria tempo em excesso para execução, ficando, assim, de fora do escopo deste trabalho.

3.5 PREVISÃO DOS RESULTADOS

Após treinamento do modelo é necessário ainda gerar previsões utilizando os dados separados para teste. Assim podemos avaliar corretamente a performance do modelo em dados que ele nunca observou (ex-post). Após a previsão é preciso ainda desescalar os resultados obtidos usando a mesma escala utilizada para transformar os dados de treino e teste. Assim, eles podem ser comparados com os dados reais. Isto foi feito utilizando a função “inverse_transform” da biblioteca “scikit-learn”. A biblioteca “matplotlib” é utilizada em seguida para apresentação dos resultados graficamente.

3.6 ANÁLISE DOS RESULTADOS

Para análise dos resultados foi efetuada, inicialmente, uma análise visual dos resultados obtidos graficamente em comparação com os dados reais. Em seguida foi feita uma análise de regressão. A fim de avaliar melhor a performance do modelo, também foi efetuado um teste de investimento prático. Para este teste, simulamos uma compra do ativo sendo analisado quando o modelo prevê uma alta dos preços, vendendo o ativo ao final do pregão. Da mesma maneira, realizamos uma venda quando o modelo prevê uma queda nos preços e recompramos o ativo ao final do pregão. Para facilitar a visualização dos resultados do teste de investimento, foi utilizado um capital inicial de R$ 10.000,00. O capital total resultante ao final do pregão é sempre reinvestido em sua totalidade utilizando a mesma estratégia. Para fins de comparação também foi adicionada a estratégia de *buy and hold.* Também foi efetuado um teste de investimento com um período de teste menor. Desta forma podemos comparar a performance do modelo quando ele é treinado com dados mais recentes e realizar uma análise de sensibilidade ao período dos dados. Neste teste, o modelo foi treinado com dados de janeiro de 2000 até dezembro de 2020 e testado para janeiro de 2021 até julho de 2021.

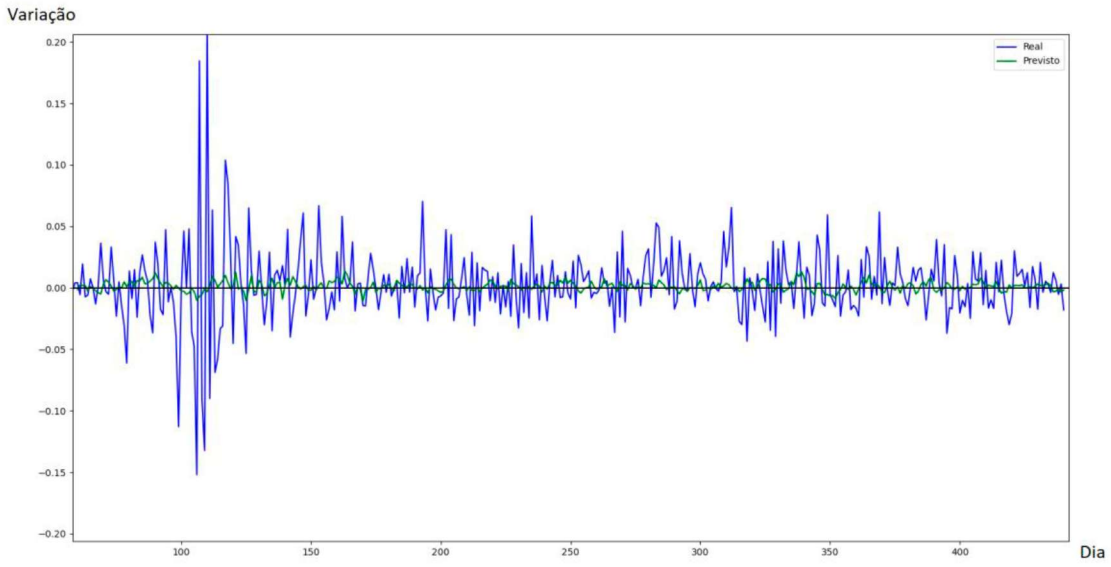
4 ANÁLISE E DISCUSSÃO

Neste capítulo, analisamos os resultados obtidos pelo modelo para os ativos VALE3 e PETR4.

4.1 RESULTADOS PARA VALE3

Após a execução do programa/treinamento utilizando dados históricos dos papéis ABEV3, ITUB4, PETR4 e VALE3 desde janeiro de 2000 até o fim de 2019, foi obtido o seguinte gráfico de previsões para a variação de preços da ação VALE3 em 2020/2021:

GRÁFICO 4 - RESULTADOS PARA VALE3



FONTE: AUTOR (2021)

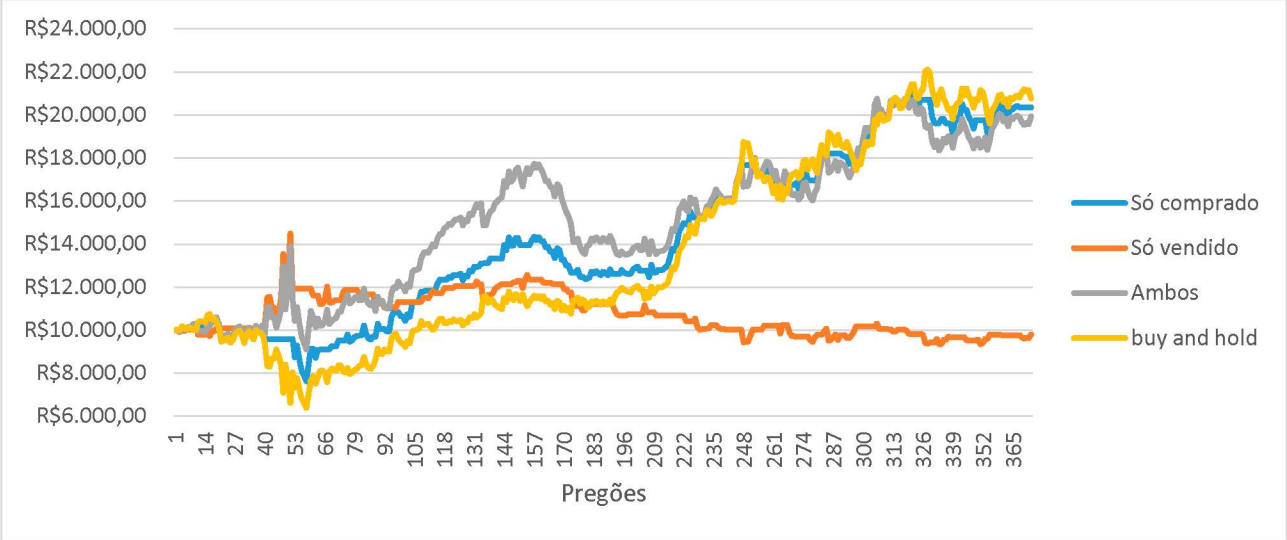
A variação de preço prevista pelo modelo está em verde e a variação de preço real em azul. Uma análise visual preliminar do Gráfico 4 indica que em certos momentos o modelo conseguiu prever a direção em que se dariam as movimentações nos preços com 1 dia de antecedência em dados em que ele não foi treinado.

Uma análise de regressão, no entanto, nos dá uma visão diferente. A correlação entre as previsões e os dados reais é de 0,07, ou seja, a variação dos preços reais está apenas 7% relacionada com a variação das previsões. Já o valor do coeficiente de determinação R^2 é de 0,005, o que significaria que o modelo consegue explicar apenas

0,5% dos valores da variação do preço. Os resultados também não foram significativos. Maiores detalhes destes testes podem ser encontrados no apêndice 2.

A seguir efetuamos o teste de investimento prático, simulando uma compra do ativo quando o modelo prevê uma alta dos preços e simulando uma venda quando o modelo prevê uma queda. Assim, após efetuar os testes, obtivemos os resultados demonstrados no Gráfico 5:

GRÁFICO 5 - RESULTADO DE INVESTIMENTO PARA VALE3

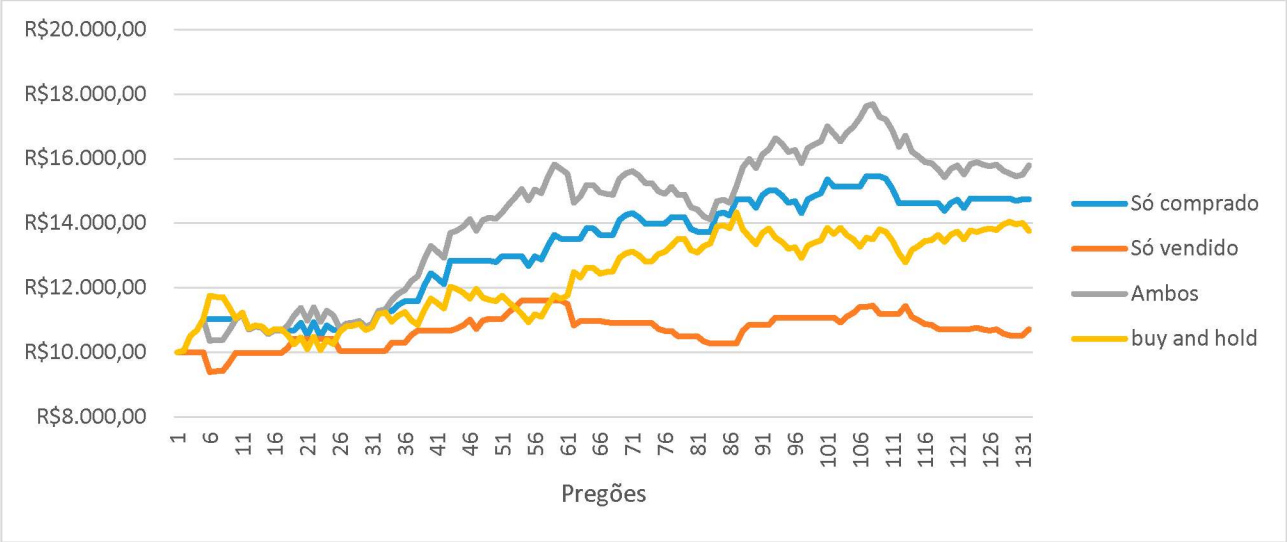


FONTE: AUTOR (2021)

Os resultados foram, inicialmente, positivos. A estratégia proposta gerou no período de janeiro de 2020 até julho de 2021 um retorno de 103% do capital inicial operando somente comprado e -2% operando somente vendido. O resultado utilizando os dois tipos de operação concomitantemente foi de 99% do capital investido, enquanto o para a estratégia de *buy and hold* foi de 108%. Há de se relevar, no entanto, que movimentos de compra e venda poderiam afetar significativamente os preços, além de estarem sujeitos a pagamento de impostos e taxas. Esta simulação serve apenas para verificar se as previsões do modelo estão de acordo com os movimentos reais do mercado.

É necessário pontuar também que o ativo analisado estava em uma forte tendência de alta no período. Isto favoreceu a estratégia comprada e de *buy and hold* e desfavoreceu a estratégia de se operar somente vendido. Nota-se também que no período inicial, onde o modelo havia acabado de ser treinado com os dados mais recentes, os resultados foram mais positivos tanto para operações de compra quanto de venda. De fato, uma análise com um período de teste menor, de janeiro a julho de 2021, e de treino de janeiro de 2000 a dezembro de 2020, revelou resultados melhores, conforme demonstrado no Gráfico 6.

GRÁFICO 6 - RESULTADO DE INVESTIMENTO PARA VALE3 COM PERÍODO DE TESTE MENOR

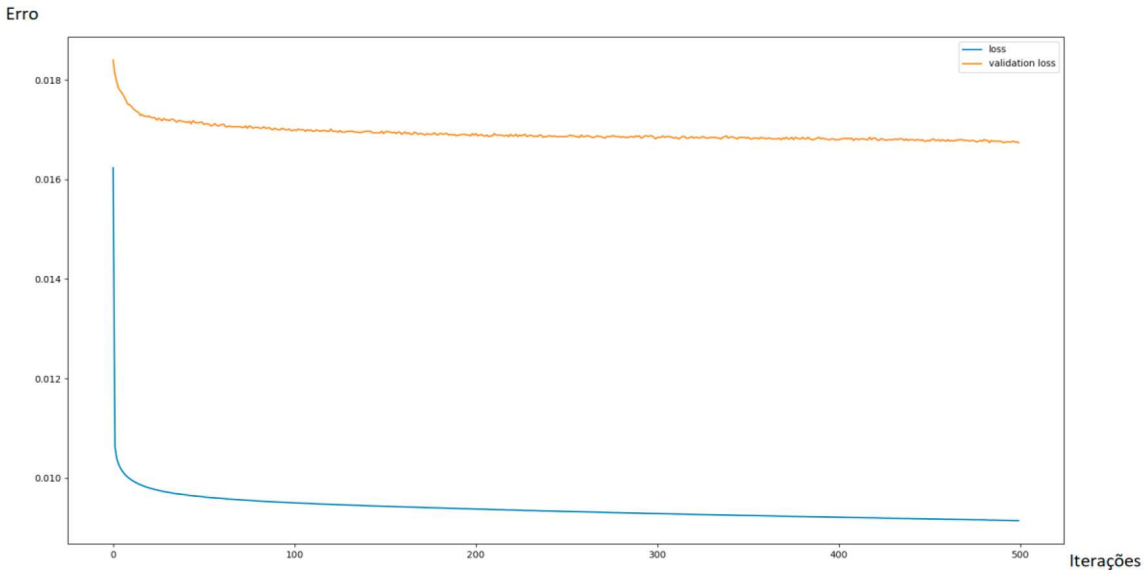


FONTE: AUTOR (2021)

A estratégia proposta gerou no período de 6 meses um retorno de 47% do capital inicial operando somente comprado e 7% operando somente vendido. O resultado utilizando os dois tipos de operação concomitantemente foi de 58%, enquanto o para a estratégia de *buy and hold* foi de 38%. Outro aspecto importante é que mesmo com o ativo em uma tendência de alta, onde os preços tendem a subir, as operações somente vendidas não resultaram em prejuízo no período, o que é um indicador positivo da performance do modelo.

Outra análise interessante a ser feita para trabalhos que utilizam *machine learning* é relativa à curva de aprendizado. Por meio desta curva podemos avaliar a performance do modelo e diagnosticar possíveis problemas de aprendizado.

GRÁFICO 7 - GRÁFICO DE APRENDIZADO DO MODELO PARA VALE3



FONTE: AUTOR (2021)

No Gráfico 7 podemos visualizar a evolução do erro apresentado pelo modelo nos dados de treinamento (azul) e nos dados de teste/validação (laranja) durante o processo de treinamento. O treinamento foi efetuado utilizando dados de 2000 a 2019 e testado no período de janeiro de 2020 até julho de 2021. Podemos verificar que em ambas as curvas há uma diminuição gradual do erro conforme o número de iterações vai aumentando. Isso significa que o modelo está conseguindo, em certa medida, aprender a prever a variação dos preços das ações. Isso vale tanto para os dados nos quais ele treinou (azul) quanto em dados que ele nunca havia verificado antes (laranja). Há de se verificar, no entanto, a disparidade entre os erros para os dois casos.

Segundo Brownlee (2019), uma diminuição dos valores de erro ao longo do tempo para os dados de treino e teste em conjunto com uma grande diferença entre estes valores pode indicar que os dados de treino não fornecem informação suficiente para que o modelo possa aprender a resolver o problema (a variação dos preços das ações no futuro) em sua totalidade. Este problema também é chamado de *unrepresentative train dataset* ou conjunto de dados de treino não representativos, em tradução livre.

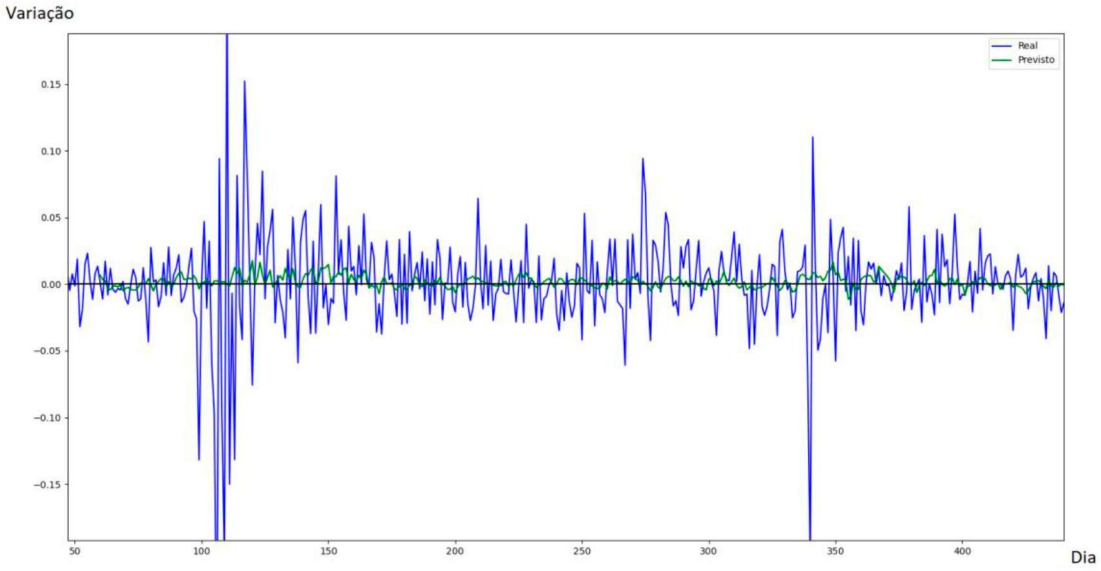
Em um contexto de mercado de ações e análise técnica, esta disparidade faz sentido. Existem diversos fatores que podem afetar o preço de uma ação que não são levados em consideração em uma análise técnica, que utiliza como base dados históricos de preços e volume. Consequentemente, estes fatores não estão sendo levados em consideração pelo modelo. Alguns destes fatores poderiam ser: lucros, receita,

endividamento, política de pagamento de dividendos, governança, taxa de juros, taxa de câmbio, taxa de crescimento do PIB, etc.

4.2 RESULTADOS PARA PETR4

Após o treinamento do programa utilizando dados históricos dos papéis ABEV3, ITUB4, VALE3 e PETR4 desde janeiro de 2000 até o fim de 2019, foi obtido o seguinte gráfico de previsões para a variação de preços da ação PETR4 em 2020/2021:

GRÁFICO 8 - RESULTADOS PARA PETR4

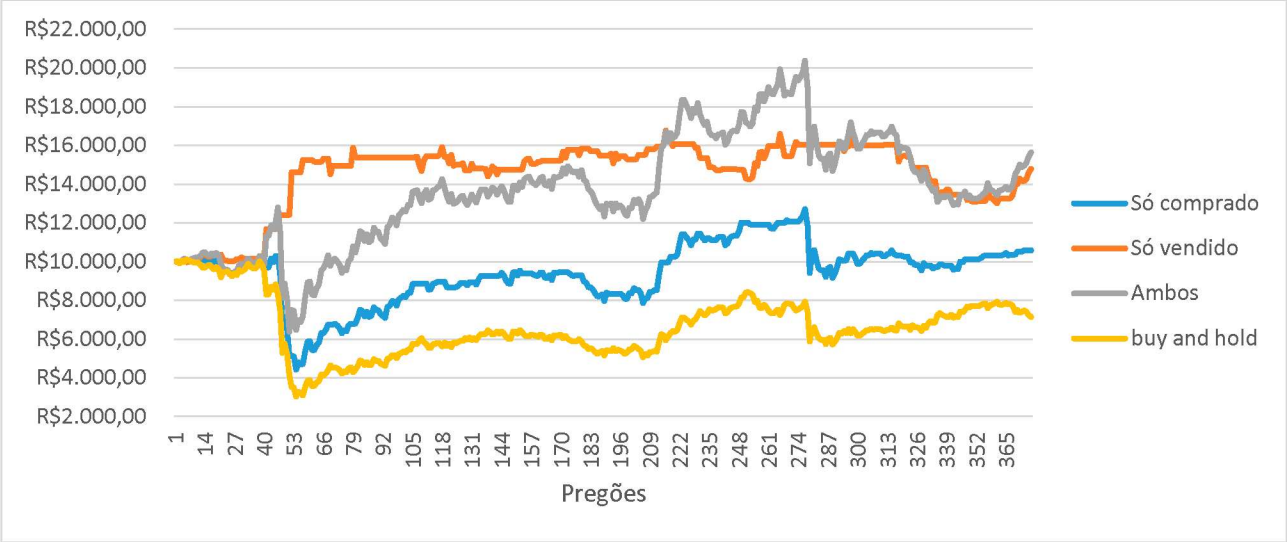


FONTE: AUTOR (2021)

Assim como para VALE3, uma análise visual preliminar dos resultados para PETR4 indica que em alguns momentos o modelo consegue prever a direção em que se dará a variação de preços do ativo com um dia de antecedência e em dados em que o modelo não foi treinado. Já, uma análise de regressão para PETR4, nos deu resultados similares aos da VALE3. A correlação entre as previsões geradas e os dados reais é de 0,05, o que nos indica que a variação dos preços está somente 5% relacionada com a variação das previsões. Este valor foi de 7% para VALE3. Já o valor do coeficiente de determinação R^2 foi de 0,002, o que significa que o modelo consegue explicar apenas 0,2% dos valores da variação do preço. Este valor foi de 0,5% para VALE3. Assim como para VALE3, os resultados da regressão não foram significativos.

Em seguida, realizamos o teste de investimento para PETR4 nas mesmas condições do efetuado para VALE3 a fim de verificar se as previsões do modelo estão de acordo com os movimentos reais do mercado:

GRÁFICO 9 - RESULTADO DE INVESTIMENTO PARA PETR4

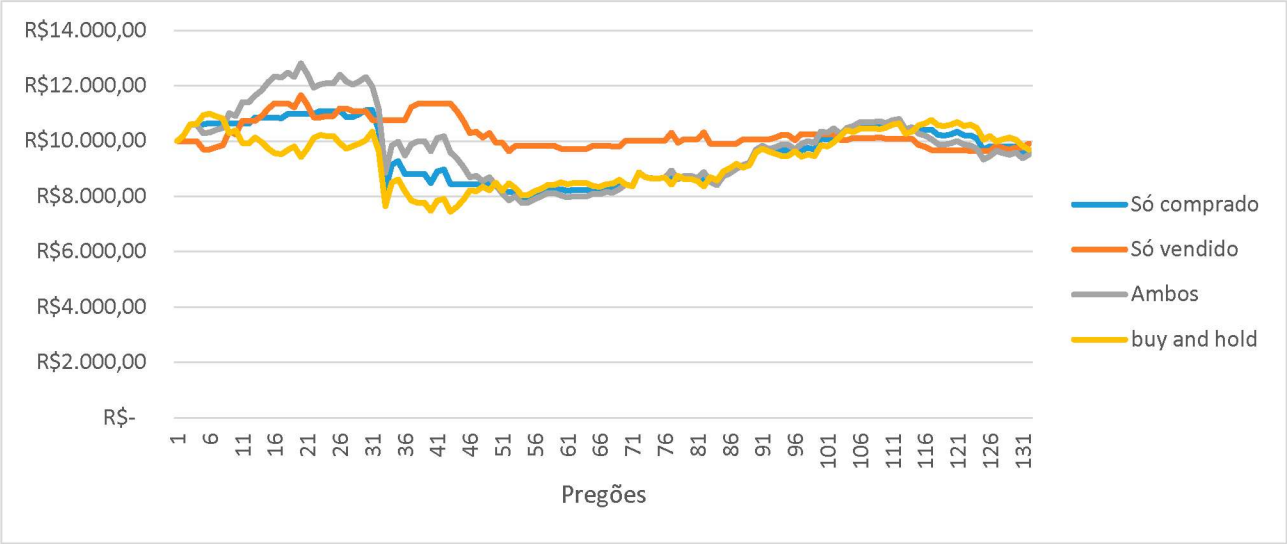


FONTE: AUTOR (2021)

Assim como para VALE3, os resultados dos testes de investimento foram positivos tanto para operações de compra quanto de venda. A estratégia proposta gerou no período de janeiro de 2020 até julho de 2021 um retorno de 6% do capital inicial operando somente comprado e de 48% operando somente vendido. O resultado utilizando os dois tipos de operação concomitantemente foi de 56% do capital investido, enquanto o para a estratégia de *buy and hold* foi de -29%.

Assim como em VALE3, os melhores resultados, tanto para operações de compra quanto de venda, foram obtidos no período inicial, onde o modelo havia acabado de ser treinado com os dados mais recentes. Uma análise com um período de teste menor, de janeiro a julho de 2021, e de treino de janeiro de 2000 a dezembro de 2020 consegue evidenciar melhor este fenômeno que ocorreu, neste caso, para os primeiros 30 pregões após o treinamento.

GRÁFICO 10 - RESULTADO DE INVESTIMENTO PARA PETR4 COM PERÍODO DE TESTE MENOR

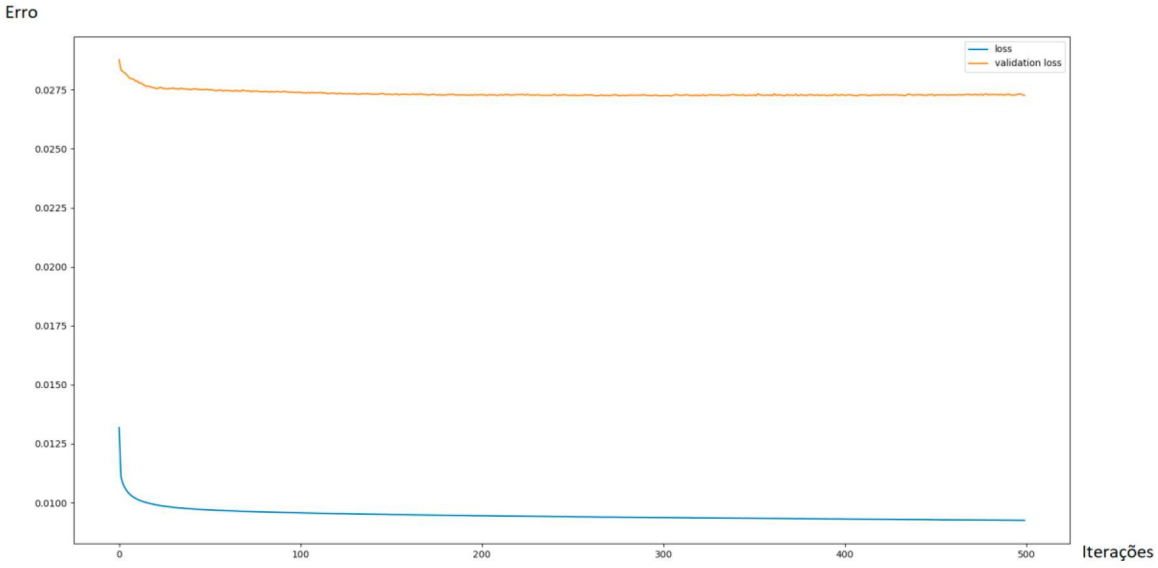


FONTE: AUTOR (2021)

A estratégia com um menor período de testes gerou no período de 6 meses um retorno de -4% do capital inicial operando somente comprado e -1% operando somente vendido. O resultado utilizando os dois tipos de operação concomitantemente foi de -5% do capital inicial, enquanto o para a estratégia de *buy and hold* foi de -3%.

Em seguida, avaliamos o gráfico de aprendizado do modelo para PETR4 treinado com os dados de 2000 a 2019 e testado de janeiro de 2020 a julho de 2021.

GRÁFICO 11 - GRÁFICO DE APRENDIZADO DO MODELO PARA PETR4



FONTE: AUTOR (2021)

Os resultados das curvas de aprendizado para PETR4 foram similares aos da VALE3 e resultam nas mesmas conclusões: Os dados utilizados para treinamento não são representativos o suficiente para que se chegue em uma variação de preços futura nos dados de teste com a mesma precisão que a obtida durante o treinamento.

5 CONCLUSÃO

Este trabalho se propôs a tentar prever a variação dos preços das ações em "D+1" com base em dados de preço e volume disponíveis no dia "D". Com base nos resultados obtidos, podemos concluir que o modelo não obteve sucesso em prever o valor exato das variações dos preços. Isto também pôde ser confirmado por meio de uma análise de regressão.

Uma análise de investimento, no entanto, revelou que o modelo obteve resultados positivos para ambos os papéis analisados (VALE3 e PETR4) ou com resultados equivalentes a uma estratégia de *buy and hold* no pior dos casos. Isto indica que o modelo pode ser útil para identificar a tendência na qual os preços se movimentarão com um dia de antecedência. Esses resultados se mostraram ainda melhores quando o treinamento do modelo é recém efetuado. Assim sendo, o modelo se mostrou útil como possível estratégia de investimento.

Outro resultado relevante foi relativo à utilização de dados de preços e volume passados, comumente utilizados em análise técnica, para tentar prever os movimentos de preços futuros. A curva de aprendizado do modelo indicou que o uso destes dados no período diário, embora tenham se mostrado úteis para prever a tendência de preços, não foram representativos o suficiente para a previsão do valor da variação de preços no dia seguinte.

O código fonte do modelo está incluso no apêndice 1. Para futuros trabalhos, o autor recomenda a realização de mais testes utilizando outros ativos e em períodos de tempo distintos tanto na etapa de treino quanto na de teste. Também pode ser útil a realização de um ajuste fino de parâmetros como tamanho da rede neural, número de iterações e tamanho dos dados utilizados como entrada. O uso de outras variáveis relevantes para alimentar o modelo como: lucros, taxa de câmbio, taxa de juros, taxa de crescimento do PIB e indicadores também deve ser avaliado. Outra adição interessante seria uma metodologia que suporte o treinamento do modelo ao final de cada dia do período de teste. Assim poderíamos avaliar melhor os resultados obtidos em um cenário de uso onde ao final de cada pregão o modelo é treinado novamente com os dados mais recentes. Esta adição, no entanto, pode exigir maior poder computacional e o uso de outras bibliotecas para programação.

REFERÊNCIAS

AUNGIERS, Jakob. **Time Series Prediction Using LSTM Deep Neural Networks**. Disponível em: <https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks>. Acesso em: 1 jul. 2020.

BROWNLEE, Jason. What is deep learning?. *In*: **What is deep learning**. [*S. l.*], 16 ago. 2019. Disponível em: https://machinelearningmastery.com/what-is-deep-learning/. Acesso em: 5 maio 2020.

BENGIO, Yoshua. Deep Learning of Representations for Unsupervised and Transfer Learning. **JMLR Workshop and Conference Proceedings**, [*s. l.*], 2012. Disponível em: http://proceedings.mlr.press/v27/bengio12a/bengio12a.pdf. Acesso em: 6 maio 2020.

BROWNLEE, Jason. **Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python**. [s.l.]: Machine Learning Mastery, 2018. Disponível em: https://books.google.com.br/books?hl=en&lr=&id=o5qnDwAAQBAJ&oi=fnd&pg=PP1&dq=time+series+forecasting+deep+learning&ots=yG7aySqh69&sig=ZycKqiVv3Gc8ZYpmja1EwwOsRBY&redir_esc=y#v=onepage&q=time%20series%20forecasting%20deep%20learning&f=false. Acesso em: 19 maio 2020.

BROWNLEE, Jason. **How to use Learning Curves to Diagnose Machine Learning Model Performance**, 26 fev. 2019. Disponível em: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/. Acesso em: 9 jul. 2021

CHONG, Eunsuk; HAN, Chulwoo; PARK, Frank C. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. **Expert Systems with Applications**, v. 83, p. 187–205, 2017.

COSTA, Claire D. **Best Python Libraries for Machine Learning and Deep Learning**. Medium. Disponível em: <https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c>. Acesso em: 6 maio 2021.

Deep Learning, Self-Taught Learning and Unsupervised Feature Learning. ExtractConf: [s.n.], 2015. Disponível em: <https://www.youtube.com/watch?v=O0VN0pGgBZM>. Acesso em: 12 maio 2020.

FLOVIK, Vegard. **(How (not) to use Machine Learning for time series forecasting: Avoiding the pitfalls | LinkedIn**. Disponível em: <https://www.linkedin.com/pulse/how-use-machine-learning-time-series-forecasting-vegard-flovik-phd/>. Acesso em: 29 jun. 2020.

FOOTE, Keith. A Brief History of Artificial Intelligence. *In*: **A Brief History of Artificial Intelligence**. [*S. l.*], 5 abr. 2016. Disponível em: https://www.dataversity.net/brief-history-artificial-intelligence/. Acesso em: 5 maio 2020.

GUARNIERI, Odir Cantanhede. **Um estudo empírico da eficiência da da análise técnica como instrumento na predição do Comportamento dos preços das ações: o caso Embraer.** 2006. Disponível em:

<http://repositorio.unitau.br:8080/jspui/handle/20.500.11874/1303>. Acesso em: 12 maio 2020.

GDC 2018: John McDonald (Valve) - Using Deep Learning to Combat Cheating in CSGO. [s.l.: s.n., s.d.]. Disponível em: <https://www.youtube.com/watch?v=ObhK8lUfIlc>. Acesso em: 12 maio 2020.

HEINZ, Sebastian. **A simple deep learning model for stock price prediction using TensorFlow**. Medium. Disponível em: <https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877>. Acesso em: 29 jun. 2020.

LÄNGKVIST, Martin; KARLSSON, Lars; LOUTFI, Amy. A review of unsupervised feature learning and deep learning for time-series modeling. **Pattern Recognition Letters**, v. 42, p. 11–24, 2014.

LIN, Tao; GUO, Tian; ABERER, Karl. Hybrid neural networks for learning the trend in time series. *In*: **Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence**. [s.l.: s.n.], 2017, p. 2273–2279. Disponível em: <https://infoscience.epfl.ch/record/262447/files/0316.pdf>. Acesso em: 21 maio 2020.

LIU, S.; ZHANG, C.; MA, J. CNN-LSTM Neural Network Model for Quantitative Strategy Analysis in Stock Markets. In: LIU, D. et al. (Eds.). . **Neural Information Processing**. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017. v. 10635p. 198–206.

MUSHAILOV, Joseph (Iosif). **LSTM Framework For Univariate Time-Series Prediction**. Medium. Disponível em: <https://towardsdatascience.com/lstm-framework-for-univariate-time-series-prediction-d9e7252699e>. Acesso em: 1 jun. 2021.

NAYAK, Asutosh. **Stock Trading with CNNs: Time Series to Image Conversion**. Medium. Disponível em: <https://towardsdatascience.com/stock-market-action-prediction-with-convnet-8689238feae3>. Acesso em: 1 jul. 2020.

NG, Yibin. **Machine Learning Techniques applied to Stock Price Prediction**. Medium. Disponível em: <https://towardsdatascience.com/machine-learning-techniques-applied-to-stock-price-prediction-6c1994da8001>. Acesso em: 25 maio 2020.

REN, Y. **Get Started with Using CNN+LSTM for Forecasting**. Disponível em: <https://towardsdatascience.com/get-started-with-using-cnn-lstm-for-forecasting-6f0f4dde5826>. Acesso em: 3 ago. 2021.

SEZER, O.; OZBAYOGLU, M. Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach. **Applied Soft Computing**, v. 70, 27 abr. 2018.

Yahoo Finance - Stock Market Live, Quotes, Business & Finance News. Disponível em: <https://finance.yahoo.com/>. Acesso em: 15 abr. 2021.

APÊNDICE 1 – CÓDIGO FONTE EM PYTHON

```
#Criado por Guilherme José Garmatter Rocha, estudante de economia pela UFPR.
#Contato: guilhermejgr@outlook.c-o-m
import datetime
import math
import dateutil
from tensorflow.keras.layers.experimental.preprocessing import Normalization
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.core import Activation
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras.layers import Flatten
from keras.layers import Input
from keras.models import Model
from matplotlib import pyplot
from sklearn.metrics import mean_squared_error
import tensorflow.keras.mixed_precision as mixed_precision
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import glob
import tensorflow as tf
import os
from tensorflow.python.keras.callbacks import ModelCheckpoint
from tensorflow.python.keras.models import load_model

# hiper parametros
lookback = 60  # numero de linhas/dias a serem dados para que o modelo tente
prever o futuro
lookfoward = 1  # numero de dias/linhas que quero prever no futuro
epoch = 500  # numero de vezes que a AI irá passar por todos os dados. quanto
maior mais demora
batch_size = 400 # Tamanho das levas. Quanto maior mais rápido. Depende de
quanta memória de vídeo temos disponível no computador para rodar o modelo
nulo = 0  # Valor que vai ser utilizado para os dados que quero ignorar
alvo = "change" # nome da coluna a ser usada como alvo para o modelo
tradingview = 0 # fonte dos dados. se for 0 é yahoo finance, se for 1 é
tradingview

# localização dos dados no computador
if (tradingview == 1):
    inputPath = (r"C:\")
if (tradingview == 0):
    inputPath = (r"C:\")

# Usar a placa de video para processamento. Precisa ser uma placa RTX
policy = mixed_precision.Policy('mixed_float16')
mixed_precision.set_global_policy(policy)
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True) # corrige um erro de
memória que estava dando na minha máquina

import keras.backend as K

def prepara_dados(inputPath, tradingview=tradingview):
```

```
    all_files = glob.glob(os.path.join(inputPath, "*.csv"))
    if (tradingview == 1): #se os dados vierem do tradingview prepara assim
        for f in all_files:
            ticker = f[-14:-8]
            df = pd.read_csv(f, header=0, index_col=False, sep=",", decimal=".")
            if not "ticker" in df.columns:  # adiciona o ticker e indexador em
tudo que nao tiver
                if ticker[0] == "_":
                    ticker = ticker.replace('_', '')
                df.insert(0, "ticker", ticker)
                df.rename(columns={"close": "Close", "high":"High",
"open":"Open", "low":"Low"}, inplace=True)
                df["time"] = pd.to_datetime(df["time"])
                df.insert(len(df.columns), "change", 0)
                df.insert(len(df.columns), "dia", 0)
                df.insert(len(df.columns), "mes", 0)
                #df.insert(len(df.columns), "hora", 0)
                #df.insert(len(df.columns), "minuto", 0)
                df['change'] = df["Close"].pct_change()
                df['dia'] = df['time'].dt.day
                df['mes'] = df['time'].dt.month
                #df['hora'] = df['time'].dt.hour
                #df['minuto'] = df['time'].dt.minute
                df.drop("time", axis=1, inplace=True)
                #colunasadiferenciar = ["Open", "High", "Low", "Close",
"Volume"] #seleciona apenas estas colunas
                colunasadiferenciar = df.drop(columns=['ticker', 'dia', 'mes',
"change"]).columns.values #seleciona todas as colunas menos as descritas
                df[colunasadiferenciar] = df[colunasadiferenciar].pct_change()
# diferencia todas as colunas menos as que retiramos
                df.dropna(inplace=True) #joga fora as linhas vazias (no caso a
primeira linha pois n tem como diferenciar ela)
                df = df[df['Volume'] != 0]
                print("ticker adicionado para " + ticker)
                df.to_csv(f, index=False)
    if (tradingview == 0):
        for f in all_files:
            ticker = f[-13:-7]
            df = pd.read_csv(f, header=0, index_col=False, sep=",", decimal=".")
            if not "ticker" in df.columns:  # se os dados não estiverem
preparados(sem ticker e etc)
                ticker = ticker.lstrip("\\")
                df.insert(0, "ticker", ticker)
                df.insert(len(df.columns), "change", 0)
                df.insert(len(df.columns), "dia", 0)
                df.insert(len(df.columns), "mes", 0)
                df["Date"] = pd.to_datetime(df["Date"])
                df['dia'] = df['Date'].dt.day
                df['mes'] = df['Date'].dt.month
                df['change'] = df["Adj Close"].pct_change()
                df.drop("Date", axis=1, inplace=True)
                df.fillna(0, inplace=True)  # preenche os dados vazios com 0
                df = df[df['Volume'] != 0]  # joga fora os dias em que o volume
no dia é igual a 0. Feriados geralmente
                #colunasadiferenciar = ["Open","High","Low","Close","Adj
Close","Volume"]
                colunasadiferenciar = df.drop(columns=['ticker', 'dia', 'mes',
"change"]).columns.values #seleciona todas as colunas menos as descritas
                df[colunasadiferenciar] = df[colunasadiferenciar].pct_change() #
diferencia todas as colunas menos as que retiramos
                df.dropna(inplace=True) #joga fora as linhas vazias (no caso a
primeira linha pois n tem como diferenciar ela)
                #df.fillna(method="ffill", inplace=True)
                print("arquivo preparado para " + ticker)
```

```
                df.to_csv(f, index=False)
    return 0


def load_stock_data(inputPath): #concatena todos os dados em um único dataframe
    all_files = glob.glob(os.path.join(inputPath, "*.csv"))
    df = pd.concat((pd.read_csv(f, header=0, sep=",", decimal=".") for (f) in
all_files))
    df.fillna(0, inplace=True)
    #df.to_csv("verificar.csv", index=False) # Apenas para debug. Caso ativada
esta linha cria um arquivo .csv com todos os dados concatenados.
    return df


def smooth_labels(labels, factor=0.1): #função pra usar o método smooth labels
na coluna ticker. assim o modelo pode teoricamente generalizar mais
    # smooth the labels
    labels = (labels * (1 - factor))
    labels += (factor / labels.shape[1])
    # returned the smoothed labels
    return labels


from keras.preprocessing.sequence import TimeseriesGenerator


def binarize_stock_data(df, train, test, continuous, nulo=nulo): # escala todos
os dados, binariza as categorias e junta tudo
    # initialize the column names of the continuous data
    # performin min-max scaling each continuous feature column to
    # the range [-1, 1]
    cs = MinMaxScaler(feature_range=(-1, 1))
    #train["change"] = (train["Close"].pct_change())
    #test["change"] = (test["Close"].pct_change())
    trainX = cs.fit_transform(train[continuous].to_numpy())
    testX = cs.transform(test[continuous].to_numpy())
    trainX = np.nan_to_num(trainX, nan=nulo)  # substitui os Nan por 0
    testX = np.nan_to_num(testX, nan=nulo)
    # Define os targets/outputs para alimentar o gerador de timeseries e treinar
o modelo

    trainY = trainX[:, 0]  # aqui é definida a coluna com o resultado do modelo
    testY = testX[:, 0]
    # n_parametros = trainX.shape[1]
    # trainY_novo = np.zeros(shape=(trainY.shape[0], n_parametros))
    # trainY_novo[:, 0] = trainY # essas 5 linhas sao pra poder usar
timedistributed(dense)
    # testY_novo = np.zeros(shape=(testY.shape[0], n_parametros))
    # testY_novo[:, 0] = testY

    # binariza as categorias com o método one-hot encode
    # isso cria uma matriz 80x80 já que estou analizando 80 tickers. no futuro
ver se é possível usar PCA para diminuir isso
    binariza_categoria = LabelBinarizer().fit(df["ticker"])
    trainCategorical = binariza_categoria.transform(train["ticker"])
    testCategorical = binariza_categoria.transform(test["ticker"])
    # metodo label smoothing para que o modelo evite overfit. só se usa nos
dados de treino
    trainCategorical = smooth_labels(trainCategorical)
    # construct our training and testing data points by concatenating
    # the categorical features with the continuous features
    trainX = np.hstack([trainX, trainCategorical])
    testX = np.hstack([testX, testCategorical])
    return trainX, testX, trainY, testY, cs
```

```
# import the necessary packages
from keras.layers import LSTM
from keras.layers import Bidirectional
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.layers import Conv1D
from keras.layers import AlphaDropout
from keras.layers import GlobalAvgPool1D
from tensorflow.keras.activations import elu
from tensorflow.keras.activations import relu
from tensorflow.keras.activations import linear

def create_model(n_input, n_parametros, lookfoward=lookfoward, batches=batches):
    # cria a rede neural
    model = Sequential()
    model.add(Input(shape=(n_input, n_parametros)))
    model.add(Conv1D(filters=64, kernel_size=3, padding="causal",
activation='selu'))
    model.add(Conv1D(filters=64, kernel_size=3, padding="causal",
activation='selu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(RepeatVector(lookfoward))
    model.add(LSTM(100, return_sequences=True))#, recurrent_dropout=0.1))
    model.add(TimeDistributed(Dense(100, activation="relu")))
    model.add(TimeDistributed(Dense(1)))
    model.summary()
    return model

def conversor_multistep(trainX, trainY, lookfoward): # arruma os dados para que
seja possível usar o timeseriesgenerator para prever mais de um momento no
futuro por vez
    saida = np.empty([len(trainY) - lookfoward, lookfoward])
    i = 0
    while i < (len(trainY) - lookfoward):
        saida[i] = trainY[i:(i + lookfoward)]
        i = i + 1
    trainX = trainX[0:(len(trainX) - (lookfoward))]  #
    return trainX, saida


import keras.optimizers
from sklearn.model_selection import train_test_split
import numpy as np
import argparse
import locale
import os

#Programa inicia aqui
batches = batch_size # apenas renomeando para facilitar o entendimento depois
print("[INFO] loading stock attributes...")
prepara_dados(inputPath)
df = load_stock_data(inputPath)
print("[INFO] constructing training/testing split...")
# divide dados em treino e teste
train_size = int(len(df) - (133+lookback)) # tamanho dos dados de teste 381 p
2019 pra cima, 133 p 2020 pra cima. É basicamente quantas linhas de informacao a
mais tem no ativo que eu quero testar em comparaçao com os outros.
train, test = df[0:train_size], df[train_size:len(df)]


#cria uma lista com os nomes das colunas desconsiderando a coluna ticker. a
```

```
variável alvo fica na frente
dfcontinuous = df.drop("ticker", axis=1) #cria um dataframe sem a variavel
ticker
columnscontinuous = dfcontinuous.columns.values #pega os nomes das colunas
colunassemalvo = columnscontinuous[columnscontinuous != alvo] #cria um array com
todos os nomes de colunas mas sem a coluna alvo
colunaalvo = columnscontinuous[columnscontinuous == alvo] #cria um array só com
o nome da coluna alvo
continuousarray = np.concatenate((colunaalvo, colunassemalvo), axis=0)
#concatena as duas colunas de forma que fique na forma [alvo, high, close, etc]
continuous = continuousarray.tolist() #transforma em uma lista para poder ser
usado depois



print("[INFO] binarizing data...")
trainX, testX, trainY, testY, cs = binarize_stock_data(df, train, test,
continuous)
n_parametros = trainX.shape
n_parametros = n_parametros[1]  # retorna o num de colunas de df. precisa pra
usar como argumento depois

import tensorflow as tf

#caso queira prever mais do que t+1 essa função resolve
trainX, trainY = conversor_multistep(trainX, trainY, lookfoward)
#testX, testY = conversor_multistep(testX, testY, lookfoward) # gerador de teste
é usado só para gerar o input para o predict_generator. não precisa dar os
targets

# separa os dados no tamanho especificado. ex: 60 dados(dias) para 1 alvo
geradortreino = TimeseriesGenerator(trainX, trainY, length=lookback,
batch_size=batches)#, stride=lookfoward)
geradorteste = TimeseriesGenerator(testX, testY, length=lookback,
batch_size=batches)#, stride=lookfoward)
# verificar o que está sendo gerado
#i=0
#while i<3:
#    x, y = geradortreino[i]
#    print('%s => %s' % (x, y))
#    i = i + 1

# for i in range(len(geradortreino)):
# x, y = geradortreino[i]
# print('%s => %s' % (x, y))


# cria o modelo
model = create_model(lookback, n_parametros)

# Compila o modelo, define a funcao de erro e define qual o otimizador a ser
utilizado (SGD)
model.compile(loss="mse", optimizer="SGD",
metrics=[tf.keras.metrics.RootMeanSquaredError()])
mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min',
save_best_only=True) # salva apenas o melhor modelo
# train the model
print("[INFO] training model...")
# treina o modelo com base nos dados fornecidos pelo gerador de dados
history = model.fit(geradortreino, epochs=epoch, validation_data=(geradorteste),
validation_freq=10, callbacks=[mc])  # treina o modelo. Testa o modelo a cada 10
epochs nos dados de teste e salva o que deu o melhor resultado
model = load_model('best_model.h5') # carrega o modelo que deu o melhor
resultado
```

```
# make predictions on the testing data
print("[INFO] predicting stock prices...")
results = model.evaluate(geradorteste) #avalia os resultados
print("test loss, test acc:", results)

testpredict = model.predict(geradorteste) #faz as previsões

if len(testpredict.shape) == 3:
    testpredict = testpredict[:, :, 0] #caso o modelo retorne em 3d muda pra 2d


# pega os valores previstos (testpredict) coloca em uma matriz de 1 dimensao de
lookfoward em lookfoward

plotpredict = np.empty(shape=len(testpredict)+lookback+lookfoward-1)
#(len(testpredict) % lookfoward)
plotpredict.fill(np.nan)

k=lookback
i = 0
while i < (len(testpredict)):
    j=0
    while j < testpredict.shape[1]:
        plotpredict[k] = testpredict[i,j]
        k=k+1
        j=j+1
    i = i+lookfoward

# coloca o valor da última previsão caso o tamanho da amostra não seja divisível
por lookfoward

if ((len(testpredict)-1) % lookfoward) != 0:
    plotpredict[(len(test)-1):] = testpredict[-1]


dfpreco = (test["Close"].to_numpy())
dfchange = (test[alvo].to_numpy())

# cria uma tabela vazia no mesmo formato dos dados originais, coloca os dados da
previsão na coluna original, e desescala os dados com a função inverse_transform
testpredicttamcerto = np.zeros(shape=(len(testpredict)+lookback+lookfoward-1,
len(continuous))) #esta matriz precisa ter o mesmo numero de COLUNAS que a
utilizada na hora de trnsformar
# put the predicted values in the right field (na mesma coluna onde está o valor
a ser previsto. como deixei change na coluna zero é zero)
testpredicttamcerto[:, 0] = plotpredict[:]
# inverse transform and then select the right field
plotpredict[:] = cs.inverse_transform(testpredicttamcerto)[:, 0]
pd.DataFrame(dfchange).to_csv("original.csv", index=False) #salva os resultados
em um csv
pd.DataFrame(plotpredict).to_csv("resultado.csv", index=False) #salva os
resultados em um csv

#pyplot.plot(dfpreco, color="blue")
pyplot.plot(dfchange, color="blue", label="Real")
pyplot.plot(plotpredict, color="green", label="Previsto")
pyplot.axhline(y=0.0, color='black', linestyle='-')
pyplot.legend()

pyplot.show()
pyplot.plot(history.history['loss'])
pyplot.plot(history.history['val_loss'])
pyplot.legend(['loss','validation loss'])
pyplot.show()
```

APÊNDICE 2 – OUTROS RESULTADOS

FIGURA 3 - RESULTADOS DA REGRESSÃO PARA VALE3

Estatística de regressão								
R múltiplo	0,075500938							
R-Quadrado	0,005700392							
R-quadrado ajustado	0,003013095							
Erro padrão	0,031032694							
Observações	372							
ANOVA								
	gl	*SQ*	*MQ*	*F*	*F de significação*			
Regressão	1	0,002042811	0,002042811	2,121236791	0,146117241			
Resíduo	370	0,356320397	0,000963028					
Total	371	0,358363208						
	Coeficientes	*Erro padrão*	*Stat t*	*valor-P*	*95% inferiores*	*95% superiores*	*Inferior 95,0%*	*Superior 95,0%*
Interseção	0,001737137	0,001680089	1,033955431	0,30183232	-0,001566583	0,005040857	-0,001566583	0,005040857
Variável X 1	0,603589711	0,414426247	1,456446632	0,146117241	-0,211336484	1,418515907	-0,211336484	1,418515907

FONTE: AUTOR (2021)

FIGURA 4 - RESULTADOS DO TESTE T PARA VALE3

Teste-t: duas amostras presumindo variâncias diferentes		
	Variável 1	*Variável 2*
Média	-0,001274516	0
Variância	0,000962807	0
Observações	372	372
Hipótese da diferença de média	0	
gl	371	
Stat t	-0,792222595	
P(T<=t) uni-caudal	0,214368579	
t crítico uni-caudal	1,648971159	
P(T<=t) bi-caudal	0,428737158	
t crítico bi-caudal	1,966378803	

FONTE: AUTOR (2021)

FIGURA 5 - RESULTADOS DA REGRESSÃO PARA PETR4

Estatística de regressão								
R múltiplo	0,053147788							
R-Quadrado	0,002824687							
R-quadrado ajustado	0,000151295							
Erro padrão	0,037860373							
Observações	375							
ANOVA								
	gl	*SQ*	*MQ*	*F*	*e significação*			
Regressão	1	0,001514529	0,001514529	1,05659295	0,30466			
Resíduo	373	0,534661122	0,001433408					
Total	374	0,53617565						
	Coeficientes	*Erro padrão*	*Stat t*	*valor-P*	*% inferior*	*% superior*	*ferior 95,0*	*perior 95,0*
Interseção	-0,000957605	0,002113906	-0,453002769	0,650810139	-0,00511	0,003199	-0,00511	0,003199
Variável X 1	0,457663619	0,445238321	1,027907072	0,304659644	-0,41783	1,333155	-0,41783	1,333155

FONTE: AUTOR (2021)

FIGURA 6 - RESULTADOS DO TESTE T PARA VALE3

Teste-t: duas amostras presumindo variâncias diferentes		
	Variável 1	*Variável 2*
Média	0,001936764	0
Variância	0,001435262	0
Observações	375	375
Hipótese da diferença de média	0	
gl	374	
Stat t	0,989980646	
P(T<=t) uni-caudal	0,16141181	
t crítico uni-caudal	1,648938048	
P(T<=t) bi-caudal	0,32282362	
t crítico bi-caudal	1,966327183	

FONTE: AUTOR (2021)