UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Rodrigo Silveira da Rosa

REDES NEURAIS ELU FEED-FORWARD APLICADAS A PREVISÃO DE PREÇOS DE ATIVOS EM BOLSAS DE VALORES

Santa Maria, RS
2018

Rodrigo Silveira da Rosa

REDES NEURAIS ELU FEED-FORWARD APLICADAS A PREVISÃO DE PREÇOS DE ATIVOS EM BOLSAS DE VALORES

Trabalho de Conclusão de Curso apresentado ao curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS) como requisito parcial para obtenção do grau de **Graduação em Engenharia de Computação.**

Orientador: Prof. Dr. José Eduardo Baggio

Santa Maria, RS
2018

Rodrigo Silveira da Rosa

REDES NEURAIS ELU FEED-FORWARD APLICADAS A PREVISÃO DE PREÇOS DE ATIVOS EM BOLSAS DE VALORES

Trabalho de Conclusão de Curso apresentado ao curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS) como requisito parcial para obtenção do grau de **Graduação em Engenharia de Computação**

Aprovado em 13 de julho de 2018:

José Eduardo Baggio, Dr. (UFSM)
(Presidente/Orientador)

Rodrigo da Silva Guerra, Ph.D. (UFSM)

Luana da Silva, Eng. (UFSM)

Santa Maria, RS
2018

RESUMO

REDES NEURAIS ELU FEED-FORWARD APLICADAS A PREVISÃO DE PREÇOS DE ATIVOS EM BOLSAS DE VALORES

AUTOR: Rodrigo Silveira da Rosa
ORIENTADOR: José Eduardo Baggio

Uma das formas de investimento financeiro mais em voga na economia atual é a compra e venda de ações. Atualmente a grande maioria das transações com ações já são feitas automaticamente por softwares especialistas, mas ainda há um grande espaço para se desenvolver métodos de tomada de decisão específicos para essa área. O presente trabalho testa uma abordagem de previsão de subidas e descidas de preços de ações utilizando redes neurais que apesar de não ser nova, teve sua viabilidade aumentada nos últimos anos graças aos avanços nos algoritmos de treinamento e paralelização utilizando placas gráficas. Neste trabalho, são comparadas arquiteturas de redes neurais *feed-forward* com diferentes configurações de hiperparâmetros, utilizando especificamente neurônios do tipo *ELU*, aplicadas a previsão de preço mínimo diário do índice *S&P* 500. Para a programação dos algoritmos foi utilizada a linguagem *Python*, em conjunto com o framework para *Deep Learning Tensorflow*, e algumas bibliotecas que simplificam o processo de manipulação de dados (*Pandas*) e de estruturação das redes (*Keras*). Após feita a análise e escolhida a melhor arquitetura, é mostrado um processo de otimização em que se verifica que a performance da rede é melhorada com a adição de novas camadas de neurônios.

Palavras-chave: Redes Neurais. Mercado de Ações. *Deep Learning*.

ABSTRACT

FEED-FORWARD ELU NEURAL NETWORKS APPLIED TO STOCK MARKETS ASSETS PRICES PREDICTION

AUTHOR: Rodrigo Silveira da Rosa
ADVISOR: José Eduardo Baggio

One of the most fashionable forms of financial investment available in nowadays economy is the buying and selling of stocks. Currently the vast majority of stock transactions are already done automatically by specialist software, but there is still a lot of room to develop specific decision-making methods for that area. The present work tests an approach to predict the rise and fall of stock prices using neural networks that although not new, has had its viability increased in the last years thanks to the advances in the algorithms of training and parallelization using graphics cards. In this work, we compare feed-forward neural network architectures with different hyperparameter configurations, specifically using ELU-type neurons, applied to the daily minimum price forecast of the S&P 500 index. For the programming of the algorithms, the Python language was used, along with the framework for Deep Learning Tensorflow, and some libraries that simplify the process of data manipulation (Pandas) and network structuring (Keras). After analyzing and choosing the best architecture, an optimization process is shown in which it is verified that the performance of the network is improved with the addition of new layers of neurons.

KEYWORDS: Neural Networks. Stock Market. Deep Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ELU exponential linear unit

RELU rectified linear unit

NYSE New York Stock Exchange

NASDAQ National Association of Securities Dealers Automated Quotations

S&P 500 índice de 500 principais ações negociadas na NYSE ou NASDAQ

^GSPC código de negociação do índice S&P 500

GPU graphics processing unit

PCA principal componente analysis

TANH tangente hiperbólica

(2D)²PCA 2-directional 2-dimensional principal component analysis

RBFNN radial basis function neural network

LSTM long short-term memory

.CSV comma-sepparated values file

API application programming interface

DATASET conjunto de dados

FEATURES atributos de um dataset

FEED-FORWARD diz-se de rede neural que não forma ciclos entre seus neurônios

HIDDEN LAYERS camadas internas de uma rede neural

FITNESS métrica de treinamento de um modelo de machine learning

EARLY-STOPPING parada do treinamento de uma rede neural quando esta alcança o pico da métrica de validação, antes mesmo de completar totalmente o treinamento programado

BUZZWORD palavra que está em voga em determinado momento ou tópico

SUMÁRIO

1 INTRODUÇÃO

A previsão do preço de ações, de uma forma ou de outra, sempre está presente quando uma operação envolvendo essas é concretizada. Desde o investidor mais passivo até o mais agressivo, o preço futuro de uma ação sempre é questionado, pois ninguém gostaria de possuir uma ação que tenderá a ter seu preço desvalorizado.

A tarefa de prever o preço em si é abordada na prática por variados caminhos, percorrendo desde a análise fundamentalista, que acredita que o preço de uma ação refletirá a situação real de uma empresa, projetada no futuro, até as ferramentas mais avançadas de análise técnica, que acredita que o preço atual em cada momento já reflete todas as informações públicas disponíveis sobre tal empresa.

Há ainda quem diga que o mercado de ações é em sua essência imprevisível, mas a maioria dos sistemas que nos cercam, por mais complexos que sejam, tendem a se mostrar previsíveis se consideradas as variáveis certas, pelo menos em parte.

Até então, não há muito consenso sobre as variáveis que realmente importam para o movimento do preço de uma ação, dado que pesquisas na área são divulgadas somente até certo ponto, onde quem consegue bons resultados, tende a proteger seus métodos. Entretanto, existem fundos de investimentos que baseados em tais técnicas demonstram resultados realmente satisfatórios (pelo motivo de normalmente estes fundos serem obrigados a declarar seus resultados por motivos legais).

1.1 OBJETIVO

Este trabalho tem como objetivo a obtenção de melhores resultados na predição de movimento de preços de ações partindo de uma arquitetura de redes neurais que apesar de já extensivamente estudada, têm agora uma nova luz graças a avanços quanto a funções de ativação e métodos de treinamento alcançados nos últimos anos. Além disso, neste mesmo trabalho são apresentadas novas ferramentas e técnicas que em conjunto com as redes neurais, influem em melhorias de performance nesse e em problemas similares.

1.2 DISTRIBUIÇÃO DOS CAPÍTULOS

Este trabalho se desenvolve em 5 capítulos distintos, sendo: o primeiro uma breve introdução ao leitor acerca do problema a ser tratado; o segundo trazendo os conhecimentos mínimos necessários ao leitor para que seja compreendido o restante do trabalho; o terceiro capítulo mostra trabalhos desenvolvidos anteriormente que seguem a mesma linha ou influenciam a produção deste; o quarto descreve o desenvolvimento completo do trabalho, desde a construção dos softwares, passando pela metodologia seguida, até as análises feitas sobre os resultados obtidos com a mesma; e o quinto e último capítulo textual desse trabalho considera possíveis trabalhos que podem ser desenvolvidos com base no mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentadas as bases de conhecimento sobre as ferramentas e as diferentes técnicas utilizadas para o desenvolvimento desse trabalho. Primeiro uma perspectiva geral sobre o mercado de ações e os dois diferentes vieses de análise do mesmo serão exploradas. Posteriormente, serão abordados os dados disponíveis para análise, estratégias e métricas de investimentos. Num segundo momento, são introduzidas técnicas, ferramentas e algumas particularidades envolvendo inteligência artificial.

2.1 MERCADO DE CAPITAIS

O mercado de capitais sistematiza a negociação de títulos e valores mobiliários, conferindo assim maior liquidez às ações, debêntures e derivativos [10]. Nesse trabalho serão abordadas ações e índices referentes a elas.

Ação é a denominação dada à menor fração de uma empresa quando essa divide seu capital social em montantes iguais [7]. Quando isso é feito, diz-se que a empresa torna seu capital aberto. Abrir o capital é uma forma da empresa crescer, pois assim qualquer indivíduo que acredite na proposta da empresa, ou que acredite que essa tenderá a se valorizar no futuro, pode somar seu capital próprio ao capital social da empresa, simplesmente comprando uma ação, e isso normalmente se reflete para a empresa como uma forma de financiamento próprio mais barata que empréstimos. Já os índices são indicadores calculados com base em múltiplas ações, apresentando assim o desempenho do conjunto como um todo.

A principal motivação para que um indivíduo compre ações, é a crença de que essas tenderão a se valorizar com o passar do tempo, pois assim, no final do período o comprador deterá um valor maior do que o pago no momento da compra. Entretanto, isso não é uma verdade para todas as ações. Além disso, deve ser considerado o período a ser realizado o investimento, pois o preço de um mesmo índice ou ação pode flutuar em relação à sua valorização e desvalorização, como podemos observar na Figura 2.1.

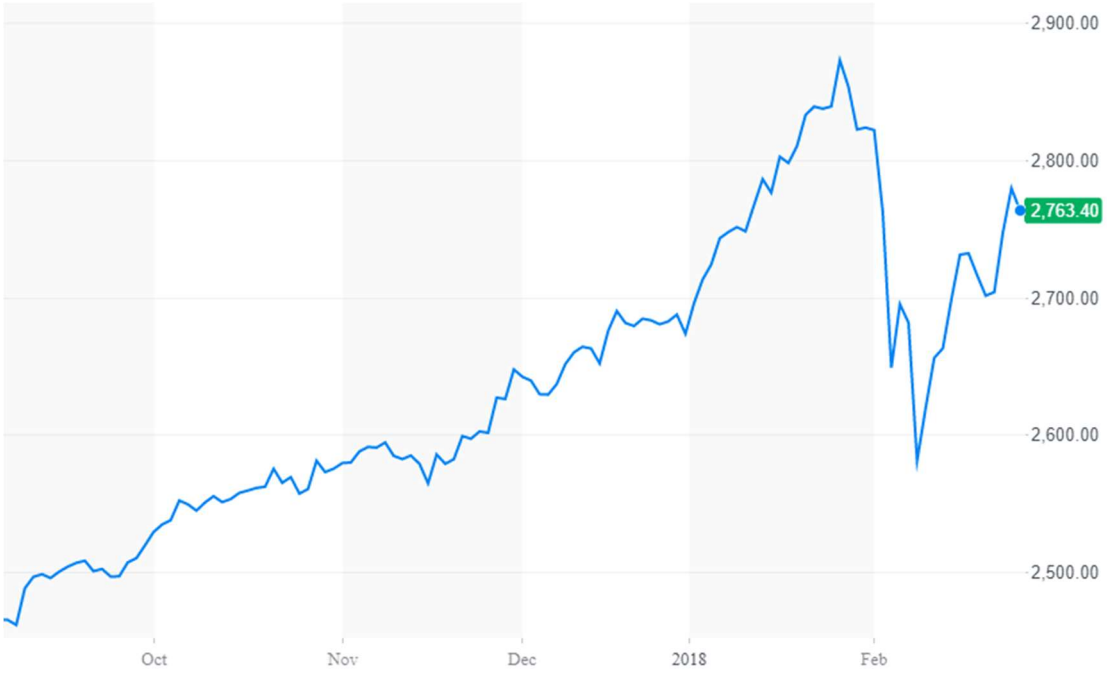

Figura 2.1 – Flutuação do preço índice *S&P* 500. Set. 2017 – Fev. 2018

Fonte: Yahoo! Finance

Foi levando em consideração a flutuação de preço de uma ação, que especialistas começaram a se debruçar sobre o assunto e a criar técnicas para tentar prever tais flutuações. A partir daí surgiram duas linhas de estudo que enfocam a análise de uma ação, a análise fundamentalista, e a análise técnica.

2.1.1 Análise fundamentalista

A análise fundamentalista tem por base o cálculo do valor intrínseco de uma ação, que é descrito como o preço teórico que uma ação deveria ter em dado momento, levando em conta todos os fatores que afetam essa quantidade [7]. A análise fundamentalista se utiliza de dados qualitativos e quantitativos sobre todo o mercado e cadeias que afetam a empresa da ação a ser analisada, e assim calcula um valor numérico do preço da mesma e este é comparado com o preço real. Daí diz-se que uma ação está supervalorizada, ou subvalorizada.

O viés fundamentalista tem por princípio a afirmação de que uma ação sempre tende a corrigir seu valor atual para o valor intrínseco, logo, sabendo-se que uma ação está supervalorizada, por exemplo, espera-se que esta venha a ter uma queda em seu preço.

2.1.2 Análise técnica

A análise técnica, diferentemente da fundamentalista, assume que o preço atual de uma ação já reflete todas as informações disponíveis publicamente sobre dada empresa, e então essa se concentra em somente analisar as movimentações do preço em si, em busca de tendências e padrões que revelem a movimentação futura do mesmo [7].

A análise técnica tem como ferramenta básica a estatística, e usa esta para criar modelos que melhor representem as variações de preços passadas, de modo que se possa utilizar estes para fazer previsões futuras sobre a movimentação de preços dos ativos sendo analisados.

2.1.3 Estratégia "*buy, sell or hold*"

Inúmeras estratégias de investimento podem ser utilizadas quando se negocia ações, uma delas é a "*buy, sell or hold*". Esta consiste em tentar prever em um período definido de tempo (ex. diariamente), se o preço de uma ação aumentará ou diminuirá no período seguinte, sendo que:

- Se o investidor já tem ações em mãos:

Essas são vendidas hoje caso a previsão do preço do período seguinte seja menor que o preço do período atual (*sell*).

As ações são mantidas caso a previsão do preço do período seguinte seja maior que o preço do período atual (*hold*).

- Se o investidor se encontra numa posição em que suas ações estão vendidas:

Compra-se novas ações caso a previsão do preço do período seguinte seja maior que o preço do período atual (*buy*).

2.2 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial constitui um grande campo da computação, sendo definida por (POOLE, MACKWORTH, GOBEL, 1998) como "dispositivos e aplicações que percebem o seu meio-ambiente, e tomam ações que maximizam as chances de ter sucesso em alguma tarefa ou meta específica".

São várias as subdivisões que derivam do extenso campo chamado inteligência artificial, mas atualmente a mais notável delas é o aprendizado de máquina, também chamado de *machine*

learning. O *machine learning* nada mais é que uma abordagem da inteligência artificial em que as generalizações e "entendimentos" que a máquina faz, são formados não por uma programação explícita dos mesmos, mas sim por conclusões próprias retiradas dos dados pelos algoritmos.

2.2.1 Redes neurais

Redes neurais artificiais são estruturas que imitam o comportamento das redes neurais cerebrais, baseando-se no entendimento atual que temos de como funcionam os neurônios humanos e as conexões entre eles. O primeiro registro da ideia data de 1943, quando um neurofisiologista (Warren McCulloch) e um matemático (Walter Pitts) sugeriram um modelo do cérebro utilizando componentes elétricos simples, modelo que posteriormente foi melhorado pela proposição de Donald Hebb de que as conexões entre neurônios mais utilizadas seriam fortalecidas ao longo do tempo [13].

As redes neurais são compostas de múltiplos nós (neurônios) que entre si podem modelar funções complexas com base em funções mais simples não-lineares chamadas de funções de ativação.

Cada nó de uma rede tem entradas, saídas, sua função de ativação específica, e um *bias* (valor somado ao final, independente de pesos). Numa rede neural, cada conexão entre neurônios tem um peso multiplicador, ponderando assim as entradas de um neurônio, como mostra a Figura 2.2.

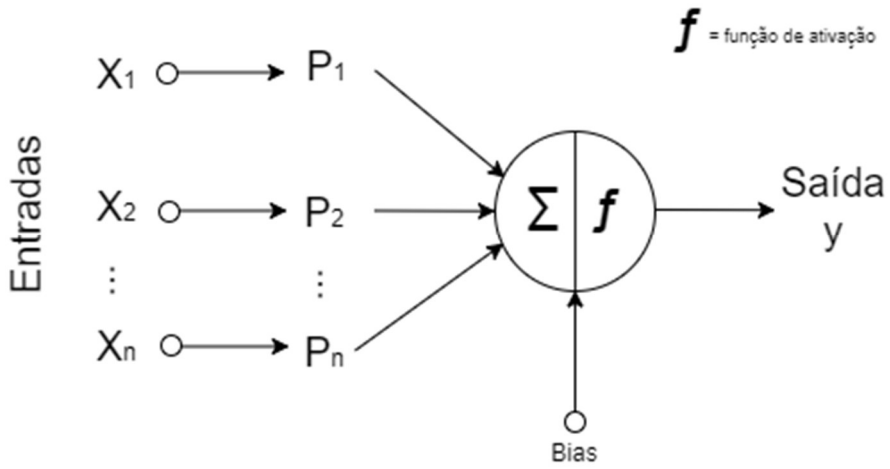

Figura 2.2 – Neurônio artificial e seus componentes

Fonte: Criado pelo autor

Numa rede *feed-forward*, os neurônios são empilhados verticalmente nas chamadas "camadas", em que neurônios da mesma camada não se conectam entre si. São as camadas que se conectam umas com as outras, sendo que cada camada somente se conecta com a seguinte, vide Figura 2.3.

O processo de treinar uma rede neural consiste no ajuste dos pesos e *bias* de cada nó, fazendo com que a função resultante melhor represente os dados apresentados à rede. Para isso, normalmente é utilizada a técnica de *backpropagation*, onde o erro na saída obtido para cada conjunto de entradas, é propagado da saída até as entradas novamente, de acordo com a influência de cada peso e de cada *bias*, sendo esta computada com a utilização de cálculo de gradientes.

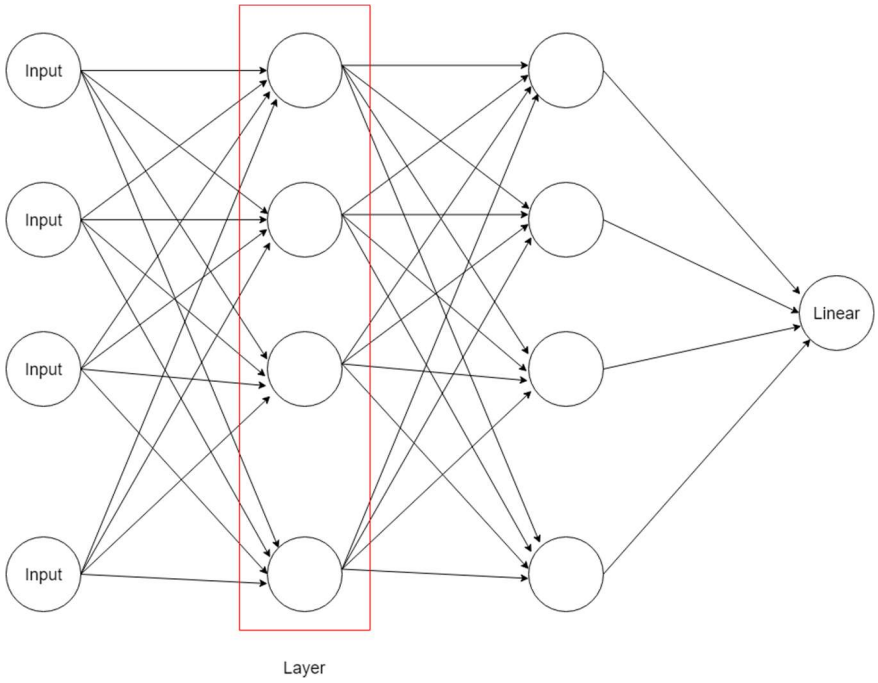

Figura 2.3 – Rede neural *feed-forward* de 2 camadas internas

Fonte: Criado pelo autor

2.2.2 *Deep learning*

Deep learning é uma *buzzword* que, de alguns anos para cá, começou a atrair muita atenção de desenvolvedores, pesquisadores, e da mídia. Isso se deve graças a grandes avanços

nos algoritmos de treinamento e da utilização bem-sucedida de computação paralela (principalmente utilizando *gpu*`s) em conjunto com esses.

Originalmente, *deep learning* foi um termo cunhado no final do século passado para descrever redes neurais que dispunham de muitas camadas de neurônios entre suas entradas e saídas, análogas a "estruturas profundas e hierárquicas relacionadas à fala e sua percepção presentes no cérebro humano" (LIU, WANG, LIU, 2017). Atualmente, esse termo representa, de maneira geral, as redes de múltiplas camadas, e as técnicas e avanços que contribuíram para a melhoria da eficiência das redes neurais atuais, como o treinamento sem supervisão de redes neurais, otimizadores de treinamento como o Adadelta e o Adam, técnicas como *dropout* e regularização de neurônios, plataformas como o *Tensorflow* e a *Theano* e arquiteturas como as redes neurais convolucionais e as LSTM, bem como outros algoritmos hierárquicos que não envolvem redes neurais.

2.2.3 *Tensorflow*

Tensorflow é um *framework* de *machine learning* baseado no paradigma de programação *Dataflow*, em que grafos são construídos para que posteriormente os dados sejam manipulados de acordo com regras de fluxo entre esses grafos. O *Tensorflow* como o nome já diz, opera com tensores (matrizes multidimensionais) nesses grafos, e isto facilita a paralelização dos algoritmos.

O *Tensorflow* foi inicialmente desenvolvido pelo Google para uso próprio, mas hoje dispõe de licença *open-source*, e é provavelmente a biblioteca para *deep learning* mais popular atualmente, sendo utilizado em projetos que abrangem desde trabalhos acadêmicos até grandes produtos comerciais de empresas como Intel e AirBnb.

2.2.4 *Keras*

A Keras é uma biblioteca em *python* que simplifica a criação de arquiteturas de redes neurais, se utilizando de um nível de abstração além do dos *frameworks* usuais. Para isso, o *Keras* se utiliza dos *frameworks* mais completos de *machine learning* e os estende.

No *Keras* as redes são descritas de uma maneira sequencial, e essas são compiladas para o *framework* em uso, facilitando e acelerando a criação de redes com diferentes parâmetros.

Além disso, a biblioteca também dispõe de estruturas mais complexas já prontas, como *layers* e *otimizadores*.

2.2.5 Neurônio *ELU*

Várias funções de ativação são utilizadas nas redes neurais atualmente, e são elas que nomeiam o tipo de neurônio formador da rede. Aqui são utilizados neurônios do tipo *ELU* (unidade linear exponencial).

Neurônios *ELU* (*exponential linear unit*, unidade linear exponencial) são uma evolução dos neurônios *RELU* (*rectified linear unit*, unidade linear retificadora) mas permitindo valores negativos, trazendo os valores médios das ativações mais próximo ao zero, o que resulta em menor tempo necessário para treinamento e em generalizações mais robustas e precisas comparados com seus antecessores de acordo com experimentos [2], vindo a ser utilizados recentemente em substituição a estes.

A função de ativação do neurônio *ELU* pode ser vista na equação 2.1, onde x é a entrada da função, e α um parâmetro que controla a saturação do neurônio para valores negativos.

$$ELU = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

Equação 2.1

2.2.6 *Dropout*

Dropout é uma técnica que diminui o problema conhecido por *overfitting* de redes neurais, situação em que a rede neural artificial memoriza mais do que generaliza os dados que são apresentados a ela no processo de treinamento. Isso se dá em função do número elevado de neurônios capazes de aprender funções de mais complexidade, assim assimilando ruídos mais facilmente quando se tem um *dataset* de treinamento pequeno.

O *dropout* também contribui para evitar o problema chamado de “morte de neurônios”, que consiste em neurônios atingirem uma saída igual a zero e a improbabilidade de saírem desse estado em função do gradiente da função zero também ser igual a zero [2].

O *dropout* consiste em desligar com uma certa probabilidade cada neurônio durante o processo de treinamento, evitando assim a utilização dos mesmos caminhos neurais, e forçando a adaptação dos caminhos que de outra forma seriam menos utilizados, e possivelmente zerados.

2.2.7 *PCA*

A *principal component analysis* se utiliza da álgebra linear e da estatística para determinar quais variáveis menos dependem de outras e posteriormente transformar os dados para uma base de dimensão igual ou menor à original (dependendo do número de componentes principais desejado) que com a menor perda, melhor represente os mesmos dados.

Para fazer isso, a PCA calcula uma matriz de covariância de todas as variáveis (assumindo que quanto maior a covariância entre duas variáveis, mais redundantes essas são entre si), e então gera autovetores ordenados nas direções de menor covariância, resultando em uma matriz de transformação para os dados.

2.2.8 Hiperparâmetros

São os parâmetros que determinam a arquitetura da rede e como esta será treinada, e que devem ser determinados antes do início da etapa de treinamento. Os hipeparâmetros são decisivos quanto à performance da rede.

Normalmente no processo de criação de redes, explora-se uma gama de diferentes hiperparâmetros, fazendo uma busca pelo conjunto que gere a rede de maior performance na tarefa em questão.

3 TRABALHOS RELACIONADOS

A tarefa de prever preços de ações com o uso de técnicas de *machine learning*, em especial, redes neurais, pode tomar diversas formas. O objetivo deste capítulo é analisar abordagens e arquiteturas já implementadas por outros autores, como meio de inspiração e comparação para este trabalho.

Singh e Srivastava (2016) compararam a implementação de *deep neural networks* de 2 camadas internas de neurônios *tanh* em conjunto com *(2D)²PCA* de dimensões 10x10, 15,15, 19x35, utilizando dados de 36 indicadores padrão utilizados em análise técnica aplicados nas ações da empresa *Google*. Eles também compararam diferentes tamanhos de janela deslizante sendo eles 20, 40, 60, 80, 100 dias.

Os resultados mostraram que o tamanho de janela deslizante de 20 dias, e a *(2D)²PCA* de dimensões 10x10 mostraram os melhores resultados, ultrapassando as redes neural recorrente e RBFNN utilizadas para comparação, apresentando precisão de 68.21% para o movimento de preço da ação utilizada.

Em outro trabalho, Bao, Yue e Rao (2017) utilizaram *autoencoders* empilhados e *LSTM*`s em conjunto com decomposição *wavelets* para previsão de índices de diversos mercados, descritos como "em desenvolvimento", "relativamente desenvolvidos" e "desenvolvidos". Primeiramente foi utilizada a transformada *wavelet* de *haar* para remoção de ruído dos dados, depois, as *features* utilizadas foram extraídas por meio do uso dos *autoencoders*, e por fim, foram utilizadas *LSTM*'s para a previsão em si.

Os autores relataram lucro médio anual de 57.05% entre todos os mercados utilizados para teste do modelo fazendo uso de sua solução, contra 17.9% obtidos utilizando somente *LSTM*`s.

Num terceiro trabalho, em busca de melhorias nos métodos de seleção de *features*, Tsai e Hsiao (2010) exploraram diferentes abordagens de utilização conjunta dos principais métodos em uso atualmente. Foram utilizadas união, intersecção e multi-intersecção entre: *PCA*, algoritmos genéticos e árvores de decisão.

Foi constatado pelos autores que a intersecção entre algoritmos genéticos e *PCA*, seguido da multi-intersecção entre as três técnicas utilizadas resultaram nas maiores precisões: 79% e 78.98% respectivamente.

Em um último trabalho, os autores analisando a performance de redes neurais convolucionais na tarefa de previsão de movimento de ações, Gunduz, Yaslan e Catatelpe (2017)

compararam redes com entradas ordenadas aleatoriamente e com sua própria ordenação proposta. Eles utilizaram 100 ações da Bolsa de Valores de Istanbul, e utilizaram um método envolvendo correlação para sua ordenação.

Os resultados baseados na métrica “Medida-F” apontam que o método de ordenação envolvendo correlação sugerido pelos autores em conjunto com a CNN (7 camadas, sendo duas camadas convolucionais paralelas) superou a regressão logística e também a CNN com ordenação aleatória.

4 IMPLEMENTAÇÃO

Seguindo a linha de pensamento da análise técnica, o autor utiliza técnicas de *machine learning* (redes neurais artificias especificamente) para tentar extrair funções que ajudem a prever o movimento dos preços do índice S&P 500 (*Yahoo! Finance*: ^GSPC). O índice em questão, reflete a situação de 500 ações selecionadas negociadas na *NYSE* ou *NASDAQ*, e tem seu valor negociado em Dólares Americanos. Os dados desse índice de ações foram coletados da plataforma *Yahoo! Finance*, e representam valores diários, sendo que o intervalo de tempo das amostras utilizadas para treinamento começa no dia 10 de agosto de 2000 e se estende até o dia 10 de agosto de 2014, e para teste, começa no dia 10 de agosto de 2014 e vai até o dia 20 de fevereiro de 2018.

Em um primeiro momento, o trabalho foi todo desenvolvido utilizando diretamente os conceitos e o *framework Tensorflow*. Os dados eram coletados manualmente do repositório do *Yahoo! Finance*, e lidos de um arquivo *.csv*, utilizando funções programadas pelo autor. Enquanto isso contribuiu para um melhor entendimento acerca do funcionamento em baixo-nível do *framework* utilizado, descobriu-se posteriormente que já existiam bibliotecas prontas e mais eficientes para realizar as mesmas operações, como a *Pandas*. Além disso, foi verificado que com a utilização da biblioteca *Keras*, o processo de busca e otimização das redes poderia ser melhor automatizado, optando-se então pela utilização dessas.

Todos os algoritmos descritos aqui foram executados no mesmo *hardware*, constituído por *Intel I7 4710-HQ*, e *NVIDIA GeForce GTX 860M*.

A coleta dos dados históricos do índice *S&P 500* se dá todo pela biblioteca *pandas_datareader*. Esta biblioteca dispõe de uma *API* que se conecta diretamente ao acervo de dados do *Yahoo! Finance* com o uso de uma única função, necessitando somente ser passado o código do ativo, a data inicial pretendida, e a data final. Para este ativo em específico, se têm 6 diferentes *features* para cada dia, sendo o preço de abertura (*open*), o preço mínimo (*low*), o preço máximo (*high*), o preço de fechamento (*close*), o preço de fechamento ajustado (*adj close*), e o volume de negociações (*volume*).

Com os dados disponíveis ao algoritmo, o passo seguinte seria a limpeza do *dataset*, em que casos como valores omissos ou com erro são lidados. Felizmente, o *dataset* a ser utilizado não necessitou desse processo.

Em seguida, é aplicado aos dados um algoritmo de janela deslizante, em que de acordo com um tamanho de janela pré-definido *n* (50, 40, 30, 20 ou 10), é montado um conjunto de

dados contendo listas *python* de *n* em *n* dias com sobreposição. Cada janela então é etiquetada, seguindo a estratégia "*buy, sell or hold*" com períodos diários, onde o preço a ser previsto é o preço mínimo (*low*). Caso o preço mínimo do índice do dia seguinte ao último dia presente na janela seja menor que o preço mínimo do ativo negociado no último dia presente na janela a ser etiquetada, esta é etiquetada com com "1", e caso o inverso ocorra, a janela é etiquetada com "0". O algoritmo 4.1 descreve o processo:

```
Dados_preparados = [janelas, etiquetas]
Para cada i de 0 até tamanho(lista_de_dados):
    janelas[i] = lista_de_dados[i : i + tamanho_da_janela]
    Se (lista_de_dados[i + tamanho_janela]['preço_mínimo_do_dia']) <
    ( lista_de_dados[i + tamanho_janela - 1]['preço_mínimo_do_dia']):
        etiquetas[i] = 1
    senão:
        etiquetas[i] = 0
```

Algoritmo 4.1 – criação das janelas e etiquetas

Os dados que até então consistiam em cada janela uma lista contendo 6 *features* de *tamanho_janela* dias cada, são então transformados em um grande vetor de 6 x *tamanho_janela* valores, para que possa ser utilizado pela rede neural *feed-forward*.

Antes de estar completamente pronto para uso, os dados passam por dois importantes processos: normalização entre 0.0 e 1.0, e transformação utilizando *Principal Component Analysis*. A normalização além de ser requerida para que a *PCA* funcione corretamente, também é algo padrão quando se trabalha com redes neurais, pois as funções de ativação são normalmente pensadas para comprimir suas saídas entre 0.0 e 1.0.

Neste trabalho, foram selecionadas como entradas para todas as redes, as 60 componentes principais de cada *dataset*, para que houvesse uma regularidade no tamanho do conjunto de entrada independentemente do tamanho da janela sendo utilizada (redução de dimensionalidade). O processo é descrito graficamente na Figura 4.1.

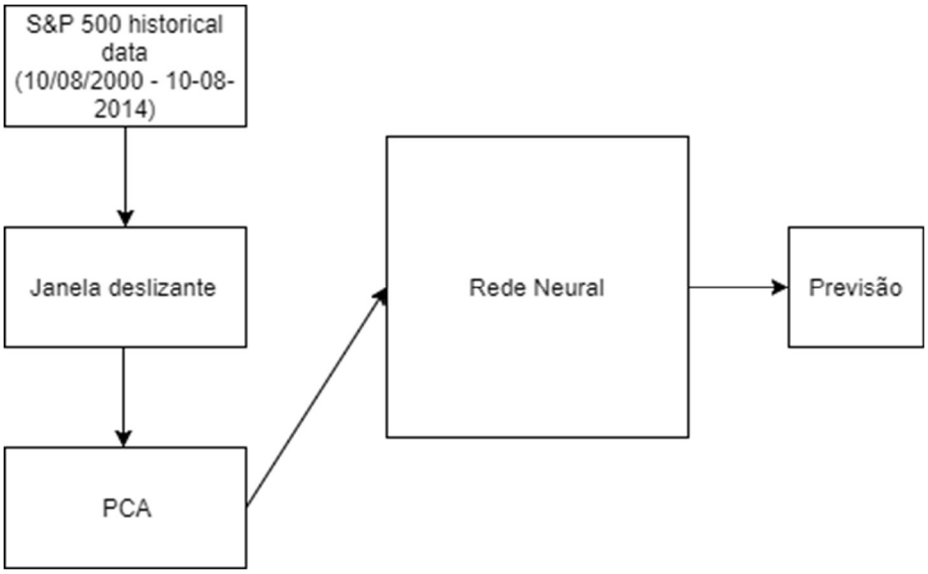

Figura 4.1 – Processo de preparação e utilização dos dados

Fonte: Criado pelo autor

Por fim, os dados estão prontos para treinar a rede neural. A busca da arquitetura que melhor prediz esse *dataset* é o alvo deste trabalho. A seguir, será explorado um universo de valores para três variáveis diferentes de hiperparâmetros: o tamanho da janela a ser utilizado, representando quantos dias anteriores são considerados para prever o dia em questão; o número de neurônios nas camadas da rede, influindo diretamente no tamanho e dificuldade de treinamento das redes; e o *dropout* utilizado no treinamento, que implica diretamente na performance do treinamento.

4.1 BUSCA COM 1 CAMADA

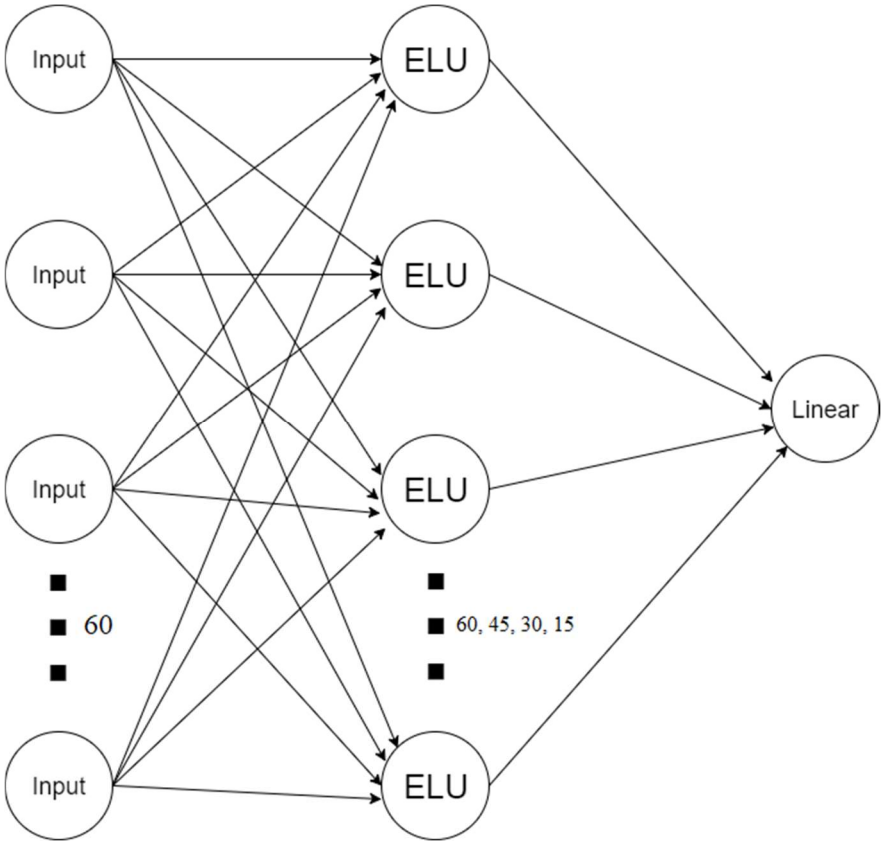

Figura 4.2 – Rede de uma única *hidden layer*

Fonte: Criado pelo autor

Foi programado um algoritmo iterativo de geração de redes neurais de uma única camada de neurônios *ELU*, como a mostrada na Figura 4.2. O algoritmo gerou ao todo 100 redes, que entre si fazem todas as combinações possíveis dos seguintes hiperparâmetros:

- *neurons_layer* = [60, 45, 30, 15]
- *dropout* = [0.4, 0.3, 0.2, 0.1, 0.05]

E dos seguintes tamanhos de janela de entrada da pca:

- *window_size* = [50, 40, 30, 20, 10]

Para treinamento, foi utilizado o otimizador *Adam* com *learning rate* igual a 0.0001, a métrica de *fitness* foi a "entropia cruzada binária", foi utilizado *early-stopping* baseado na métrica "precisão binária", a divisão entre dados de treinamento e validação foi de 75% das

amostras para treinamento e 25% para validação, e cada rede foi treinada por 100.000 épocas (16 minutos em média para cada rede).

4.2 BUSCA DE MÚLTIPLAS CAMADAS

Após a fase inicial de busca das melhores redes com uma só camada, foi feita uma nova busca agora variando-se a profundidade da melhor rede obtida anteriormente, ou seja, aumentando-se o número de camadas de neurônios. Foram geradas e treinadas 9 novas redes, sendo que cada uma continha de 2 até 10 camadas de neurônios, onde cada camada de neurônios é agregada de *dropout* vide Figura 4.3.

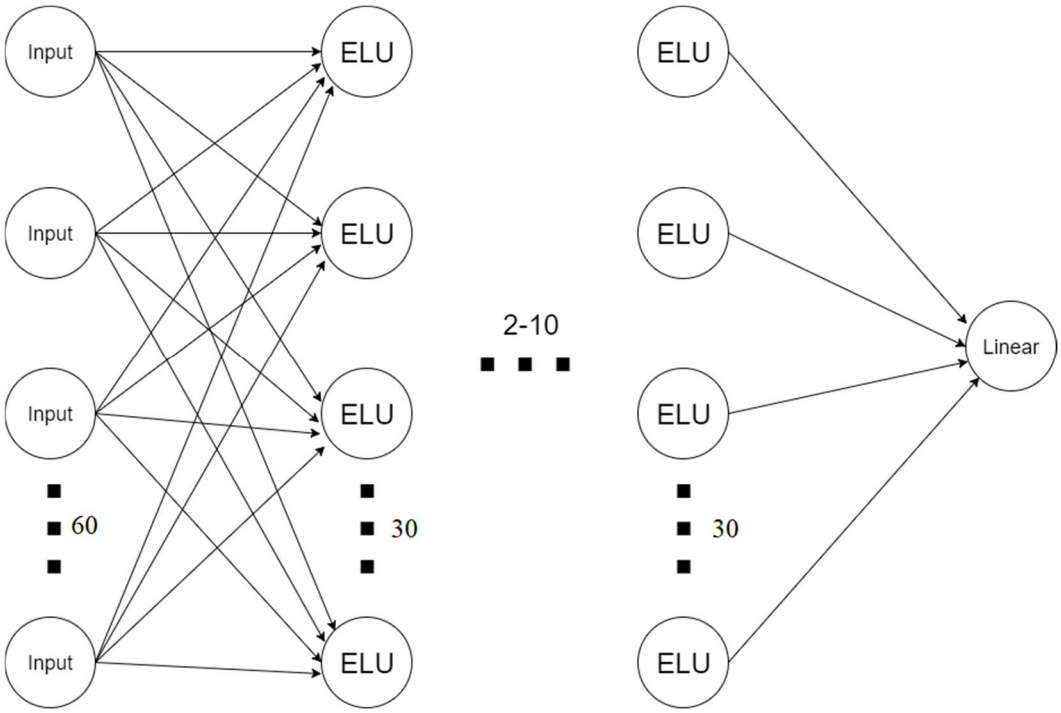

Figura 4.3 – Redes de múltiplas camadas

Fonte: Criado pelo autor

A partir da rede de múltiplas camadas de melhor performance obtida, obteve-se então a rede final.

4.3 RESULTADOS

Na Tabela 4.1 abaixo são apresentados os 10 melhores resultados obtidos a partir do treinamento das 100 diferentes redes de uma única camada:

	window_size	number_neurons	dropout	val_accuracy
1	10	30	0,3	0,675398633
2	10	60	0,3	0,674259681
3	10	45	0,4	0,674259681
4	10	60	0,2	0,673120729
5	10	60	0,05	0,671981777
6	10	15	0,3	0,671981777
7	10	45	0,1	0,671981777
8	10	15	0,4	0,670842824
9	10	60	0,4	0,669703872
10	10	15	0,2	0,669703872

Tabela 4.1 – 10 melhores resultados dentre as 100 redes de uma única camada

Aqui podemos observar que uma janela mais próxima do dia a ser previsto, apresenta melhor performance no geral, e que a configuração com 30 neurônios (metade do número de entradas), utilizando *dropout* 0,3, foi a rede mais precisa de acordo com os dados de validação.

De acordo com as configurações obtidas pela rede de melhor performance, foram geradas novas redes agora com número variável de camadas (2 a 10 camadas). Os resultados representados por *train_loss* (métrica do erro para os dados de treinamento), *train_accuracy* (taxa de acerto de previsões para os dados de treinamento), *val_loss* (métrica do erro para os dados de validação) e *val_accuracy* (taxa de acerto de previsões para os dados de validação) obtidos são apresentados na Tabela 4.2.

	train_loss	train_accuracy	val_loss	val_accuracy
2 layers	0.594379259098	0.682370820669	0.622505297015	0.670842824466
3 layers	0.602922825828	0.68009118541	0.630378377872	0.670842824466
4 layers	0.585493639429	0.685030395137	0.622076371672	0.674259680958
5 layers	0.594036145051	0.685790273556	0.62399768069	0.67767653745
6 layers	0.595195954694	0.677431610942	0.621639291898	0.67198177663
7 layers	0.589390060278	0.686170212766	0.620134732718	0.678815489614
8 layers	0.593590219514	0.678951367781	0.617901456926	0.681093393942
9 layers	0.583871278719	0.689209726444	0.61569393102	0.681093393942
10 layers	0.594129812573	0.677051671733	0.61976321398	0.674259680958

Tabela 4.2 – Resultados obtidos na busca de rede de múltiplas camadas

Notamos que as redes *multilayer* de 8 e 9 *layers* foram as que se saíram melhor, de modo que houve um empate. Escolheu-se então a rede mais simples, no caso a de 8 *layers*, como a rede definitiva, tendo precisão final de acordo com os dados de validação de 68.1% (acerta 68 de cada 100 previsões sobre o movimento do preço mínimo do dia seguinte).

O último passo é o teste final da rede definitiva, utilizando-se os dados de teste (nunca antes apresentados a rede), neste caso os dados do índice *S&P* 500 do dia 10 de agosto de 2014, até o dia 20 de fevereiro de 2018.

A precisão final obtida no teste foi de 0.618885096769, vide Tabela 4.3, o que significa que a rede definitiva acerta cerca de 61 de cada 100 previsões feitas.

test_loss	test_accuracy
0.639546753061	0.618885096769

Tabela 4.3 – Perda e precisão de teste da rede definitiva

Como métrica secundária, foi calculado também o retorno total do período de testes (período de dados nunca antes apresentados a rede), (878 dias úteis, aproximadamente 42 meses no total), assumindo compra e venda sempre pelo valor de fechamento diário, resultando em 765,391%, ou 5,2724% de rendimento aproximado ao mês (8,65391 vezes o valor aplicado inicialmente). O algoritmo e o resultado utilizados podem ser vistos na Figura 4.4.

```
In [10]: import pandas as pd
         import numpy as np

         preds = pd.read_csv("predictions.csv")
         data = pd.read_csv("^GSPC.csv")
         #começando no último dia da primeira janela
         values = data['Close'][9:]
         last_day_price = values.iloc[len(values)-1]

         poup=1000000
         acoes=0
         #terminando no último dia para o qual temos o valor real
         for i in range(len(preds)-1):
             if preds.iloc[i].item()==1:
                 if acoes!=0:
                     poup=acoes*values.iloc[i].item()
                     acoes=0
             else:
                 if poup!=0:
                     acoes=poup/values.iloc[i].item()
                     poup=0

In [11]: print ((acoes*last_day_price)/1000000)

         8.65391406853
```

Figura 4.4 – Demonstração do algoritmo e resultado do cálculo de retorno total obtido com o modelo

Fonte: Criado pelo autor

Para comparação, utilizando previsões aleatórias e também assumindo compra e venda pelo valor de fechamento diário, como vemos na Figura 4.5, obtemos 33,168% de rendimento, ou aproximadamente 1,00684% ao mês (1,33168 vezes o valor aplicado inicialmente).

```
In [18]: import pandas as pd
         import numpy as np

         preds = np.random.choice([0, 1], size=(878,))
         preds = pd.DataFrame(preds)
         data = pd.read_csv("^GSPC.csv")
         #começando no último dia da primeira janela
         values = data['Close'][9:]
         last_day_price = values.iloc[len(values)-1]

         poup=1000000
         acoes=0
         #terminando no último dia para o qual temos o valor real
         for i in range(len(preds)-1):
             if preds.iloc[i].item()==1:
                 if acoes!=0:
                     poup=acoes*values.iloc[i].item()
                     acoes=0
             else:
                 if poup!=0:
                     acoes=poup/values.iloc[i].item()
                     poup=0

In [19]: print ((acoes*last_day_price)/1000000)

         1.33168235833
```

Figura 4.5 – Demonstração do algoritmo e resultado do cálculo de retorno total obtido com previsões aleatórias

Fonte: Criado pelo autor

Se fosse utilizada a técnica "*buy and hold*" somente (compra-se no início do período, e vende-se somente ao final), ao final do período o investimento (preço de fechamento do primeiro dia do período 1988,4, preço mínimo do último dia do período 2732,22) apresentaria rendimento de 37,4%, como visto na equação 4.1.

$$\frac{2732{,}22}{1988{,}4} = 1{,}374$$

$$1{,}374 - 1 = 0.374$$

$$100 * 0.374 = 37.4\%$$

Equação 4.1 – Cálculo de rendimento utilizando "*buy and hold*"

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Com o desenvolver desse trabalho foi possível o domínio das técnicas e tecnologias mais atuais sendo utilizadas em *machine learning*, conhecimento que aqui foi aplicado em uma área muito específica, mas que tem potencial para facilmente ser utilizado para a resolução de problemas em diversos campos.

Pelos resultados, concluímos a exploração do problema aqui em questão como satisfatória, tendo a rede neural profunda final apresentado bons níveis de performance, e também de potencial retorno financeiro, apesar de não considerar nas métricas os preços de corretagem.

A partir desse trabalho, uma série de explorações ainda podem ser realizadas. Um exemplo seria a implementação de indicadores já bem estabelecidos na análise técnica clássica, como os indicadores utilizados por Ritika Singh e Shashi Srivastava [1]. Como este trabalho preocupa-se somente com o preço mínimo de cada dia, investigações levando em conta preço máximo, ou preço de fechamento por exemplo, são possíveis, bem como esses em conjunto. Quanto ao número de componentes principais utilizados pela *Principal Component Analysis*, esses podem ser também explorados num trabalho futuro. Todas as redes de múltiplas camadas geradas nesse trabalho, são redes retangulares, tendo o mesmo número de neurônios em cada *layer*, o que também apresenta potencial para análise. Além de melhorias na rede em si, essa poderia ser utilizada como componente de um trabalho maior, talvez utilizando técnicas de análise de sentimento, fazendo assim, uso de diferentes técnicas de previsão.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] - SINGH, Ritika; SRIVASTAVA, Shashi. Stock prediction using deep learning. **Multimedia Tools And Applications,** [s.l.], v. 76, n. 18, p.18569-18584, 17 dez. 2016. Springer Nature.

[2] - CLEVERT, Djork-Arné; UNTERTHINER, Thomas; HOCHREITER, Sepp. Fast and accurate deep network learning by exponential linear units (ELUs). **arXiv,** [cs.LG], 1511.07289v5, 22 fev. 2016.

[3] - SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER Ilya; SALAKHUTDINOV, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research,** [s.l.], v. 15, p.1929-1958, jun. 2014. Sparc.

[4] - BAO, Wei; YUE, Jun; RAO, Yulei. A deep learning framework for financial time series using stacked autoencoders and long short term memory. **PLoS ONE,** [s.l.], 12(7), jul. 2017. Public Library of Science

[5] - POOLE, David; MACKWORTH, Alan; GOEBEL, Randy. **Computational intelligence a logical approach**. 1998. Oxford University Press.

[6] - LIU, Weibo; WANG, Zidong; LIU, Xiaohui. A survey of deep neural network architechtures and their applications. **Neurocomputing**, [s.l.], v. 234, p.11-26, 2017. Elsevier.

[7] - MURPHY, JJ. **Technical analysis of the financial markets: A comprehensive guide to trading methods and applications**. 1999. Penguin.

[8] - SHLENS, Jonathon. A tutorial on principal component analysis. **arXiv,** [cs.LG], 1404.1100, 03 abr. 2014.

[9] - TENSORFLOW. **Getting Started**. Disponível em: <https://www.tensorflow.org/get_started/>. Acesso em 06 dez. 2017.

[10] - BM&F BOVESPA. **Introdução ao mercado de capitais**. Disponível em: https://web.archive.org/web/20100601204536/http://www.bmfbovespa.com.br/pt-br/a-bmfbovespa/download/merccap.pdf

[11] - TSAI, Chih-Fong; HSIAO, Yu-Chieh. Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches**. Decision Support Systems**, [s.l.], v. 50, n. 1, p.258-269, 21 ago. 2010. Elsevier.

[12] - GUNDUZ, Hakan; YASLAN, Yusuf; CATALTEPE, Zehra. Intraday prediction of Borsa Istanbul using convolutional neural networks and feature selection**. Knowledge-Based Systems**, [s.l.], v. 137, n. 1, p.138-148, 1 dez. 2017. Elsevier.

[13] - JIACONDA. **A Concise History of Neural Networks**. Disponível em: <https://medium.com/@Jaconda/a-concise-history-of-neural-networks-2070655d3fec>. Acesso em 20 jun. 2018. Medium.

APÊNDICE A – CÓDIGOS

```
import pandas_datareader as pdr
import datetime
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Activation
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
import pickle
import keras.callbacks as kcb
import keras.optimizers as koptm

first_day = datetime.datetime(2000,8,10)
last_day = datetime.datetime(2014,8,10)
data = pdr.DataReader("^GSPC", "yahoo", first_day, last_day)

window_size_list = [50, 40, 30, 20, 10]
neurons_first_layer_list = [60, 45, 30, 15]
dropout_list = [0.4, 0.3, 0.2, 0.1, 0.05]

learning_rate = 0.0001
pca_numberpcs = 60

file = open("results.csv","a")
string="pca_numberpcs" +','+ "window_size" +','+ "neurons_first_layer" +','+ "dropout" +','+ "train_loss" +','
+ "train_accuracy" +','+ "val_loss" +','+ "val_accuracy" + '\n'
file.write(string)
file.close()

for window_size in window_size_list:
    for neurons_first_layer in neurons_first_layer_list:
        for dropout in dropout_list:
            pbr=[[],[]]
            for i in range(len(data)-window_size):
                pbr[0].append(data.iloc[i:i+window_size].values)
                if ((data.iloc[i+window_size][2])<(data.iloc[i+window_size-1][2])):
                    pbr[1].append(1)
                else:
                    pbr[1].append(0)

            pbr[0] = np.reshape(pbr[0], [len(pbr[0]), window_size*6])
            pbr[1] = np.reshape(pbr[1], [len(pbr[1]),1])

            scaler = MinMaxScaler()
            scaler.fit(pbr[0])
            pickle.dump(scaler, open("scaler.p", "wb"))
            pbr[0] = scaler.transform(pbr[0])

            pca = PCA(n_components=pca_numberpcs)
            pca.fit(pbr[0])
            pickle.dump(pca, open("pca.p", "wb" ))
            pbr[0] = pca.transform(pbr[0])

            model = Sequential()
            model.add(Dense(neurons_first_layer, input_dim=len(pbr[0][0]), activation = 'elu'))
            model.add(Dropout(dropout))
            model.add(Dense(1))
            optimizer = koptm.Adam(lr=learning_rate)
            model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['binary_accuracy'])
            callbacks=[kcb.ModelCheckpoint(filepath="weights.hdf5", monitor='val_binary_accuracy', verbose=1, save_best_only=True)]
            model.fit(pbr[0], pbr[1], epochs=100000, batch_size=int(len(pbr[0])*0.75), validation_split=0.25, callbacks=callbacks)

            model.load_weights('weights.hdf5')
            model.save('model.h5')
            model = load_model('model.h5')
            file = open("results.csv","a")
            train_evaluation = model.evaluate(pbr[0][0:int(len(pbr[0])*0.75)], pbr[1][0:int(len(pbr[0])*0.75)])
            validation_evaluation = model.evaluate(pbr[0][int(len(pbr[0])*0.75):], pbr[1][int(len(pbr[0])*0.75):])
            string=str(pca_numberpcs) +','+ str(window_size) +','+ str(neurons_first_layer) +','+ str(dropout) +','
            + str(train_evaluation[0]) +','+ str(train_evaluation[1]) +','+ str(validation_evaluation[0]) +','
            + str(validation_evaluation[1]) + '\n'
            file.write(string)
            file.close()
```

Figura A.1 – Código de busca de redes com uma única camada

Fonte: Criado pelo autor

```
import pandas_datareader as pdr
import datetime
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Activation
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
import pickle
import keras.callbacks as kcb
import keras.optimizers as koptm

first_day = datetime.datetime(2000,8,10)
last_day = datetime.datetime(2014,8,10)

data = pdr.DataReader("^GSPC", "yahoo", first_day, last_day)
data.dropna(inplace=True)

window_size = 10
neurons_first_layer = 30
dropout = 0.3

learning_rate = 0.0001
pca_numberpcs = 60

file = open("results.csv","a")
string="pca_numberpcs" +','+ "window_size" +','+ "neurons_first_layer" +','
+ "dropout" +','+ "train_loss" +','+ "train_accuracy" +','+ "val_loss" +','+ "val_accuracy" + '\n'
file.write(string)
file.close()


pbr=[[],[]]
for i in range(len(data)-window_size):
    pbr[0].append(data.iloc[i:i+window_size].values)
    if ((data.iloc[i+window_size][2])<(data.iloc[i+window_size-1][2])):
        pbr[1].append(1)
    else:
        pbr[1].append(0)

pbr[0] = np.reshape(pbr[0], [len(pbr[0]), window_size*6])
pbr[1] = np.reshape(pbr[1], [len(pbr[1])])

scaler = MinMaxScaler()
scaler.fit(pbr[0])
pickle.dump(scaler, open("scaler.p", "wb"))
pbr[0] = scaler.transform(pbr[0])


pca = PCA(n_components=pca_numberpcs)
pca.fit(pbr[0])
pickle.dump(pca, open("pca.p", "wb" ))
pbr[0] = pca.transform(pbr[0])
```

Figura A.2.1 – Código de busca de redes com múltiplas camadas (1)

Fonte: Criado pelo autor

```
model = Sequential()
model.add(Dense(neurons_first_layer, input_dim=len(pbr[0][0]), activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))
model.add(Dense(neurons_first_layer, activation = 'elu'))
model.add(Dropout(dropout))

model.add(Dense(1))
optimizer = koptm.Adam(lr=learning_rate)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['binary_accuracy'])
callbacks=[kcb.ModelCheckpoint(filepath="weights.hdf5", monitor='val_binary_accuracy', verbose=1, save_best_only=True)]
model.fit(pbr[0], pbr[1], epochs=100000, batch_size=int(len(pbr[0])*0.75), validation_split=0.25, callbacks=callbacks)

model.load_weights('weights.hdf5')
model.save('model.h5')
model = load_model('model.h5')
file = open("results.csv","a")
train_evaluation = model.evaluate(pbr[0][0:int(len(pbr[0])*0.75)], pbr[1][0:int(len(pbr[0])*0.75)])
validation_evaluation = model.evaluate(pbr[0][int(len(pbr[0])*0.75):], pbr[1][int(len(pbr[0])*0.75):])
string=str(pca_numberpcs) +','+ str(window_size) +','+ str(neurons_first_layer) +','+ str(dropout) +','
+ str(train_evaluation[0]) +','+ str(train_evaluation[1]) +','+ str(validation_evaluation[0]) +','
+ str(validation_evaluation[1]) + '\n'
file.write(string)
file.close()
```

Figura A.2.2 – Código de busca de redes com múltiplas camadas (2)

Fonte: Criado pelo autor

```
import pandas_datareader as pdr
import datetime
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Activation
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
import pickle
import keras.callbacks as kcb
import keras.optimizers as koptm


first_day = datetime.datetime(2014,8,10)
last_day = datetime.datetime(2018,2,20)

data = pdr.DataReader("^GSPC", "yahoo", first_day, last_day)

window_size = 10

pbr=[[],[]]
for i in range(len(data)-window_size):
    pbr[0].append(data.iloc[i:i+window_size].values)
    if ((data.iloc[i+window_size][2])<(data.iloc[i+window_size-1][2])):
        pbr[1].append(1)
    else:
        pbr[1].append(0)

pbr[0] = np.reshape(pbr[0], [len(pbr[0]), window_size*6])
pbr[1] = np.reshape(pbr[1], [len(pbr[1])])


scaler = pickle.load(open("scaler.p", "rb"))
pbr[0] = scaler.transform(pbr[0])



pca = pickle.load(open("pca.p", "rb" ))
pbr[0] = pca.transform(pbr[0])

model = load_model('model.h5')
file = open("test.csv","a")
string="test_loss" +','+ "test_accuracy" + '\n'
file.write(string)
test_evaluation = model.evaluate(pbr[0], pbr[1])
string=str(test_evaluation[0]) +','+ str(test_evaluation[1]) + '\n'
file.write(string)
file.close()
```

Figura A.3 – Código de teste da rede definitiva

Fonte: Criado pelo autor

APÊNDICE B – TABELAS

	window_size = 50									
number_neurons	60					45				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,6989	0,7055	0,7042	0,8058	0,7613	0,9925	1,5146	5,2825	1,4385	1,6146
train_accuracy	0,5292	0,5196	0,5015	0,5388	0,538	0,54	0,5373	0,5396	0,5346	0,5357
val_loss	0,6973	0,6958	1,1198	0,7506	0,7014	1,176	1,7452	1,619	3,5204	4,1649
val_accuracy	0,5841	0,5783	0,5806	0,5876	0,5864	0,5783	0,5795	0,5749	0,5806	0,5657
	window_size = 50									
number_neurons	30					15				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	4,5446	0,685	0,819	1,9326	1,6187	1,6177	0,6888	0,7686	1,3852	0,8617
train_accuracy	0,4301	0,5538	0,5457	0,5365	0,5342	0,5161	0,5427	0,5396	0,5238	0,5307
val_loss	1,8503	0,6857	0,7695	3,9644	1,429	3,8551	0,6912	0,8476	2,214	0,7893
val_accuracy	0,5899	0,576	0,5818	0,5737	0,5806	0,5737	0,5611	0,5829	0,5818	0,568

	window_size = 40									
number_neurons	60					45				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,6816	0,6807	0,6849	4,9785	0,7162	0,6816	0,6748	2,1669	1,3255	5,6906
train_accuracy	0,5659	0,5613	0,5563	0,5349	0,5406	0,5521	0,5739	0,5345	0,5364	0,536
val_loss	0,6865	0,6835	0,6847	1,6019	0,7013	0,6812	0,685	1,5027	3,9627	1,6913
val_accuracy	0,5839	0,5644	0,5908	0,5793	0,5805	0,5828	0,5598	0,5828	0,5782	0,577
	window_size = 40									
number_neurons	30					15				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,7198	5,6875	0,6773	0,684	0,8536	0,6892	2,0384	0,7322	0,6808	2,2797
train_accuracy	0,531	0,5307	0,5728	0,549	0,5322	0,5375	0,5295	0,5402	0,5651	0,5387
val_loss	2,2008	1,7899	0,6828	0,6839	0,7406	0,6835	5,4679	1,7197	0,6846	0,6945
val_accuracy	0,5759	0,577	0,5782	0,5805	0,5989	0,5816	0,577	0,5793	0,5885	0,5793

	window_size = 30									
number_neurons	60					45				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,6159	0,6145	0,6185	0,6385	0,5995	0,62	0,6162	0,6511	0,6162	0,6018
train_accuracy	0,6462	0,6454	0,6366	0,6347	0,6534	0,6343	0,6442	0,6267	0,653	0,6599
val_loss	0,6449	0,6423	0,6431	0,6499	0,6472	0,6386	0,6592	0,6594	0,6428	0,6468
val_accuracy	0,6346	0,6323	0,622	0,6346	0,6346	0,6277	0,6346	0,6277	0,6266	0,6289
	window_size = 30									
number_neurons	30					15				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,616	0,619	0,6544	0,6025	0,5988	0,6179	0,6139	0,6573	0,6054	0,6272
train_accuracy	0,642	0,6408	0,6251	0,6492	0,6527	0,6488	0,6473	0,6282	0,6465	0,6469
val_loss	0,6411	0,6419	0,6607	0,6476	0,6468	0,6391	0,6574	0,6649	0,6584	0,645
val_accuracy	0,6323	0,6254	0,6243	0,6312	0,6208	0,6312	0,6277	0,6323	0,6323	0,63

Tabela A.1.1 – Hiperparâmetros e respectivos resultados de cada uma das 100 redes de única camada (1)

	window_size = 20									
number_neurons	60					45				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,6023	0,6106	0,6038	0,6054	0,5883	0,6066	0,6058	0,5992	0,6062	0,6041
train_accuracy	0,6579	0,6518	0,6613	0,6564	0,6674	0,6549	0,6632	0,6621	0,6541	0,6537
val_loss	0,6382	0,6408	0,6383	0,6399	0,6543	0,6385	0,6393	0,6357	0,6393	0,6391
val_accuracy	0,6446	0,6411	0,6457	0,6491	0,6434	0,6423	0,6423	0,6411	0,6457	0,6366
	window_size = 20									
number_neurons	30					15				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,6023	0,5961	0,6081	0,593	0,602	0,5894	0,6031	0,6015	0,583	0,6011
train_accuracy	0,6514	0,6526	0,6621	0,6712	0,6651	0,672	0,659	0,661	0,6678	0,6613
val_loss	0,6375	0,6432	0,6383	0,6388	0,6417	0,6561	0,6382	0,6358	0,6706	0,6378
val_accuracy	0,6434	0,6446	0,64	0,6411	0,6331	0,6446	0,648	0,6423	0,6377	0,6446

	window_size = 10									
number_neurons	60					45				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,5824	0,5837	0,578	0,5806	0,5916	0,5872	0,5814	0,5889	0,5807	0,5846
train_accuracy	0,6767	0,6839	0,6774	0,682	0,6847	0,6755	0,6748	0,6786	0,6854	0,6824
val_loss	0,6337	0,6362	0,6363	0,6398	0,6368	0,6354	0,6368	0,6412	0,6417	0,6408
val_accuracy	0,6697	0,6743	0,6731	0,6697	0,672	0,6743	0,6686	0,6697	0,672	0,6651
	window_size = 10									
number_neurons	30					15				
dropout	0,4	0,3	0,2	0,1	0,05	0,4	0,3	0,2	0,1	0,05
train_loss	0,581	0,5799	0,5867	0,5833	0,5847	0,5951	0,5802	0,5927	0,5811	0,5796
train_accuracy	0,6763	0,6786	0,6767	0,674	0,6831	0,671	0,6816	0,6767	0,6733	0,6771
val_loss	0,636	0,638	0,6371	0,6365	0,6353	0,6263	0,6366	0,6361	0,6341	0,6357
val_accuracy	0,6697	0,6754	0,6686	0,6697	0,6674	0,6708	0,672	0,6697	0,6663	0,6674

Tabela A.1.2 – Hiperparâmetros e respectivos resultados de cada uma das 100 redes de única camada (2)