PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

Marcos Vinicius Guimarães de Jesus

Modelo Preditivo de Séries Temporais Financeiras utilizando Redes Neurais Artificiais

São Paulo

2018

Marcos Vinicius Guimarães de Jesus

Modelo Preditivo de Séries Temporais Financeiras utilizando Redes Neurais Artificiais

Trabalho de Conclusão de Curso apresentado à Pontifícia Universidade Católica de São Paulo – PUCSP, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Pontifícia Universidade Católica de São Paulo - PUCSP

Departamento de Computação

Programa de Graduação

Orientador(a): Edith Ranzini

Coorientador: Fernando Giorno e Wladimir Esposito

São Paulo

2018

Marcos Vinicius Guimarães de Jesus
Modelo Preditivo de Séries Temporais Financeiras utilizando Redes Neurais Artificiais/ Marcos Vinicius Guimarães de Jesus. – São Paulo, 2018
52 p. : il.

Orientador(a): Edith Ranzini

TCC (Graduação) – Pontifícia Universidade Católica de São Paulo - PUCSP
Departamento de Computação
Programa de Graduação, 2018.

1. Redes Neurais . 2. Ciência de dados. 2. Bitcoin. I. Edith Ranzini. II. Pontifícia Universidade Católica de São Paulo - PUCSP. III. Ciência da Computação. IV. Modelo Preditivo de Séries Temporais Financeiras utilizando Redes Neurais

Marcos Vinicius Guimarães de Jesus

Modelo Preditivo de Séries Temporais Financeiras utilizando Redes Neurais Artificiais

Trabalho de Conclusão de Curso apresentado à Pontifícia Universidade Católica de São Paulo – PUCSP, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Edith Ranzini
Orientadora

Professor
Convidado 1

Professor
Convidado 2

São Paulo
2018

Resumo

Devido a grande popularidade em 2017, a criptomoeda bitcoin tornou-se uma *commodity* bastante procurada por diversos investidores com o objetivo de lucrar diante da oscilação da criptomoeda. Entretanto, por ser um tipo de *commodity* cuja a precificação é altamente volátil, identificar os melhores momentos de compra e venda da moeda torna o ato de investir bastante complexo. Visando ser um parâmetro plausível para tomada de decisão, um modelo capaz de identificar tendências sobre o preço da criptomoeda é essencial para a saúde econômica do investidor. Este trabalho tem como ação principal analisar o desempemnho de duas classificações de redes neurais artificiais e identificar quais características a rede conter para realizar a tarefa proposta que se resume a previsão de séries temporais financeiras. Dentre as arquiteturas abordadas, a arquitetura de redes neurais recorrentes apresentou um resultado considerável na previsão da precificação da criptomoeda. O erro quadrático médio entre os valores coletados para teste e a generalização gerada pela Rede Neural foi de ≈ 0.219, com os valores normalizados em um intervalo de [-1,1]. De um modo geral, o desafio é identificar o conjunto de parâmetros e arquitetura de Redes Neurais que produza uma solução satisfatória para o problema proposto.

Palavras-chave: Previsão de Series Temporais. Redes Neurais. Bitcoin.

Abstract

One of the most popular investments in 2017, the bitcoin has been seeked by many investors that wants to gain a lot of money with the criptocurrency oscilation. However, this criptocurrency has a high volatile behavior, that's make this kind of investment be so hard to invest. Aiming be a plausible parameter for decision making, a model able to identify trends is essencial to maintain the financial healths of investor. This paper has as main job analyze two types of Neural Network and identify which parameters and which kind of archicteture and topology a Neural Network needs to realize that job summarized by time series prediction. The mean squared error between the test values and the respective generalization getted by the best topology builded was 0.219, with the bitcoin price normalized between an interval of [-1,1]. In general, the bigest challenge of this work is find the set of values that produces a good solution to the proposed problem.

Keywords:Bitcoin. Neural Network. Time Series Prediction.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
EQM	Erro Quadrático Médio
MLP	Multilayer Perceptron
RNR	Rede Neural Recorrente
RNRS	Rede Neural Recorrente Simples
LSTM	*Long Short Term Memory*
PCA	*Principal Component Analysis*
RNA	Redes Neurais Artificiais

Sumário

1 Introdução

Neste capítulo serão apresentados os elementos básicos deste trabalho: motivação, objetivo, o método de trabalho e a organização do texto.

1.1 Motivação

Commodity[1] criada há quase uma década, o bitcoin é uma moeda totalmente virtual que possui grande destaque no sistema econômico atual. Mas somente em 2017 o interesse pela aquisição de bitcoins passou a ser um dos investimentos mais comentados do planeta, valorizando cerca de 1.400% em um ano (EXAME, 2017). Devido ao alto índice de valorização, investidores, economistas e pesquisadores de diversas áreas procuram modelos e informações que possam ajudá-los a entender e possivelmente prever o comportamento da criptomoeda, assim como ocorre em outros componentes do mercado financeiro.

Entretanto, a modelagem/previsão do domínio de problema não é trivial, partindo do pressuposto de que existe um número imenso de variáveis, fontes e tipos de informação que possam influenciar em seu comportamento.

De um modo geral, a modelagem ou previsão da precificação de *commodities* é um domínio de problema clássico e interdisciplinar, que envolve ramos da Economia, Estatística, Física e Ciência da Computação. O intuito deste trabalho é ser capaz de prever o preço futuro da criptomoeda analisada, tendo como maior desafio o mapeamento das informações que melhor representam o seu comportamento. A possibilidade de previsão contribui para a definição de estratégias de investimento que otimizem o retorno sobre as negociações realizadas.

> Se no primeiro dia de 1999, o leitor deste artigo houvesse sido acometido pelo impulso em adquirir 15.000 reais (supondo-se, é claro, que ele dispusesse desse capital) em ações preferenciais da Vale do Rio Doce, ao último dia de abril de 2008, esse leitor teria se tornado praticamente um milionário, uma vez que suas ações estariam valendo cerca de 935 mil reais. Caso o leitor tivesse preferido adquirir e vender, no mesmo intervalo de tempo, ações das Lojas Americanas sua felicidade teria sido ainda maior: ele amealharia a fabulosa soma de um milhão e 800 mil reais com a operação (BONALDI, 2010, p. 2).

Dentre os métodos e artifícios para analisar o comportamento de uma *commodity*, a análise técnica tem como base a utilização de métodos matemáticos aplicados a dados históricos de preços e ao volume negociado, visando a previsão da precificação e observação

[1] Mercadoria em estado bruto ou produto básico de grande importância no comércio internacional, como café, cereais, algodão etc., cujo preço é controlado por bolsas internacionais.

de tendências futuras. Outro método é a análise fundamentalista que tem como base o estudo dos dados econômicos de empresas vinculadas ao ativo estudado.

Existem outros fatores externos que podem interferir no comportamento do bitcoin, como fatores macroeconômicos e sóciopolíticos. Notícias que envolvem principalmente a regulamentação desta moeda podem impactar em sua precificação ou até mesmo o mercado como um todo.

Por haver alta disponibilidade dos dados históricos, este trabalho utiliza conceitos da análise técnica para construir um modelo capaz de prever as observações futuras da *commodity* estudada. A proposta é utilizar o histórico, o respectivo volume negociado e os indicadores derivados dos dados coletados, para que através de técnicas de aprendizado de máquina seja possível determinar o comportamento da série temporal financeira.

Dentre as técnicas de aprendizado de máquina, o uso de redes neurais artificiais (RNA) é uma técnica computacional atraente devido à característica de modelagem não paramétrica e não linear, em que não há grande necessidade de conhecer o domínio do problema, ao contrário de técnicas básicas utilizadas na análise estatística de séries temporais (BRAGA *et al.*, 2007).

Devido a alta recorrência do termo "Redes Neurais Artificiais" e como não existe a possibilidadede confundir este termo com o utilizado na neurociência, a expressão *redes neurais artificiais* será simplificada por *redes neurais* no decorrer deste trabalho.

1.2 Objetivo

Utilizando a premissa de que redes neurais apresentam eficiência considerável quando aplicadas a ativos do mercado financeiro, este trabalho utiliza esta técnica, visando analisar empiricamente a sua aplicabilidade no mercado de criptomoedas.

O objetivo deste trabalho é avaliar o desempenho de duas classificações de redes neurais: redes neurais *feedforward* multicamada e redes neurais recorrentes, em particular redes *Long Short Term Memory* (LSTM), para a visualização dos preços futuros da criptomoeda e quais benefícios que esta técnica pode trazer para este domínio de problema. Ao final deste trabalho, tem se como objetivo identificar o conjunto de parâmetros e arquitetura de Redes Neurais que produza uma solução para o problema proposto, de tal forma que a previsão obtida seja equivalente ou próxima à precificação futura da moeda.

1.3 Método de Trabalho

Este trabalho dividiu-se em duas partes fundamentais: o estudo dos conceitos básicos que o fundamenta e o desenvolvimento da rede neural, utilizando os conhecimentos

adquiridos ao decorrer do estudo. Nesta seção são descritas as atividades que definem o método de trabalho:

- **Levantamento Bibliográfico dos temas**: Pesquisa bibliográfica referente aos temas que são abordados neste trabalho: Aprendizado de Máquina e Redes Neurais Artificiais (RNA).
- **Coleta de dados**: Esta etapa está destinada à extração da base de dados históricos da cotação do bitcoin em tempos regulares, utilizados para treinar e avaliar a rede neural.
- **Aplicação dos Indicadores Técnicos**: Esta etapa está destinada à aplicação dos indicadores da análise técnica aos dados coletados no passo anterior.
- **Pré-processamento**: Os dados submetidos à análise podem apresentar algumas inconsistências ou podem ser organizados de forma inadequada, além de problemas relacionados a grandezas desproporcionais. Esta etapa inclui a definição de estratégias para suprir as inconsistências e outros problemas no conjunto de dados.
- **Modelagem da Rede Neural**: Esta etapa está destinada à junção dos conceitos adquiridos no levantamento bibliográfico, afim de identificar uma topologia de rede neural que possa realizar a tarefa proposta.
- **Aplicação dos conceitos base abordados na fundamentação**: Aplicação de algoritmos e suas combinações que contribuam na construção e melhoria do desempenho de uma rede neural que seja capaz de realizar a tarefa proposta.
- **Interpretação dos resultados**: Etapa é destinada à análise dos resultados obtidos em relação ao resultado esperado.

1.4 Organização do Texto

No capítulo 1, Introdução, foi apresentada uma breve contextualização do tema abordado, juntamente com a definição da técnica utilizada para concluir o trabalho proposto. Este capítulo foi segmentado entre as seguintes seções; a motivação, o objetivo e o método utilizado.

No capítulo 2, Revisão Bibliográfica, são abordados os conceitos gerais para realização do modelo computacional, tais como fundamentos sobre redes neurais artificiais, modelagem, treinamento e otimização de uma rede neural. Junto a revisão bibliográfica,

destina-se o conteúdo referente aos trabalhos relacionados e como eles poderão ser utilizados para o desenvolvimento deste estudo.

No capítulo 3, Desenvolvimento da Rede Neural, são apresentados todos os passos para a construção da rede neural e os resultados obtidos através do modelo escolhido utilizando os conceitos citados nos capítulos anteriores.

Por fim, no capítulo 4, Conclusão, há uma comparação entre as arquiteturas de redes neurais utilizadas. Segue também sugestões que possam complementar o trabalho posteriormente.

2 Revisão Bibliográfica

Este capítulo está destinado a proporcionar uma visão geral sobre redes neurais artificiais e suas classificações. Além disso, uma visão do treinamento de redes neurais.

2.1 Redes Neurais Artificiais

Iniciada na década de 40 do século passado, a pesquisa sobre redes neurais foi inspirada pelo modo de funcionamento do cérebro humano. Segundo Haykin (2009, p.23), o cérebro humano é altamente complexo, não linear e realiza computações paralelas. O cérebro possui a capacidade de se organizar por meio de componentes estruturais conhecidos como neurônios, que efetuam através de computações, reconhecimento de padrões e percepção de forma muito eficaz quando comparados a muito dos computadores atuais.

Diante da fácil adaptabilidade, o cérebro humano é considerado *plástico*, ou seja, possui a possibilidade de realizar novas conexões entre os neurônios para cumprir uma nova tarefa a ser realizada. A *plasticidade* do cérebro humano é essencial para o funcionamento dos neurônios como unidade de processamento de informações.

De um modo geral, redes neurais artificiais constituem um tipo de aprendizado de máquina, inspirado no modelo de como o cérebro humano executa uma tarefa em particular ou uma função desejada. Uma rede neural artificial é composta por várias unidades de processamento denominadas neurônios artificiais. Essas unidades de processamento são representadas como nós que se interconectam através de pesos.

2.1.1 Neurônio Artificial

O neurônio artificial é a unidade de processamento de informação que é fundamental para o modelo de redes neurais. A figura 1 a seguir ilustra a arquitetura de um neurônio artificial que é composto por três elementos básicos:

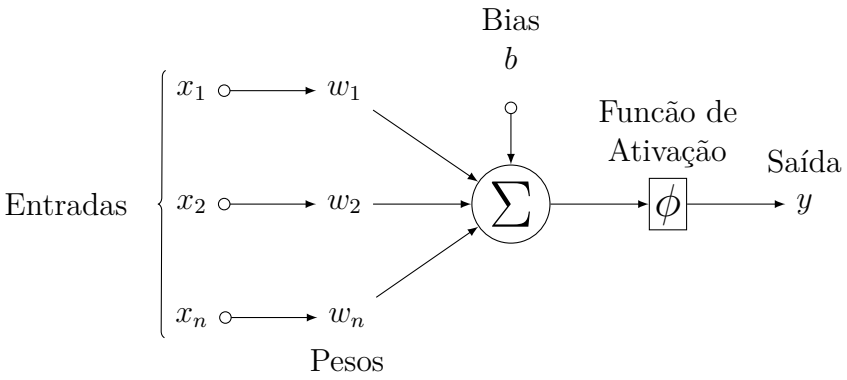

Figura 1 – Modelo de um neurônio artificial não-linear
Fonte: Autor

1. Conjunto de sinapses: Conjunto de sinapses ou conexões caracterizadas por um peso. Uma entrada é associada a um peso de forma ponderada, que significa a relevância da entrada para o neurônio.

2. Somador: Um componente que realiza a soma dos sinais de entrada, ponderados pelos respectivos pesos que constituem um *combinador linear*.

3. Função de ativação: Uma função de ativação para limitar a amplitude da saída do neurônio artificial. Segundo Haykin (2009, p. 34), a função de ativação é também referida como função restritiva, já que restringe o intervalo permissível de amplitude do sinal de saída a um valor finito.

Em termos matemáticos, Haykin (2009, p. 33) descreve um neurônio k com m entradas escrevendo o seguinte par de equações:

$$u_k = \sum_{j=1}^{m} \omega_{kj} \times x_j \tag{2.1}$$

$$u'_k = u_k + b_k \tag{2.2}$$

$$y_k = \phi(u'_k) \tag{2.3}$$

onde,
x_j ⟵ entrada j
ω_{kj} ⟵ peso j referente ao neurônio k
b_k ⟵ bias (polarização)
u_k ⟵ combinador linear
u'_k ⟵ $u_k + b_k$
ϕ ⟵ função de ativação
y_k ⟵ saída do neurônio k

Dentre os componentes de um neurônio artificial, o bias b tem o efeito de aplicar uma transformação à saída do combinador linear conforme a equação 2.2. Segundo Haykin (2009, p. 33), dependendo do bias b_k ser positivo ou negativo, a relação entre o potencial de ativação representado por u'_k do neurônio k e a saída do combinador linear u_k é modificada. A transformação resultante da aplicação do bias é exemplificada na figura 2.

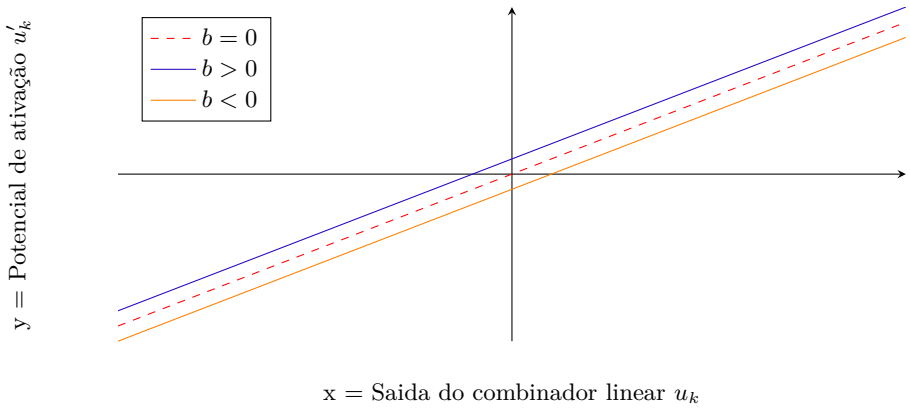

Figura 2 – Transformação produzida pela presença de um bias
Fonte: Autor

2.1.2 Funções de Ativação

As funções de ativação são representadas por $\phi(u'_k)$ e basicamente definem a saída do neurônio. Nesta seção são definidos algumas funções de ativação mais comuns:

1. **Função Limiar**: Segundo Haykin (2009, p.34), este tipo de função de ativação é definida pela equação 2.4:

$$\phi(u'_k) = \begin{cases} 1, se\ u'_k \geq 0 \\ 0, se\ u'_k < 0 \end{cases} \tag{2.4}$$

Neste modelo, a saída de um neurônio assume o valor de 1 se o o potencial de ativação desse neurônio não assumir um valor negativo e 0 para outros casos. A função

limiar é utilizada em redes neurais *Perceptron* que será abordada na secção 2.2.1.1 à seguir.

2. **Função Sigmoide**: Segundo Haykin (2009, p.36), a função de ativação sigmoide é uma das mais utilizadas na modelagem de redes neurais. É definida como uma função estritamente crescente, demonstrando um equilíbrio entre um comportamento linear e não linear. A função pode ser representada por diversos valores de a que equivale a tangente da inclinação da reta. A função é representada matematicamente por:

$$\phi(u'_k) = \frac{1}{1 + \exp(-a \times u'_k)} \tag{2.5}$$

A figura 3 a seguir representa a função sigmoide com valores diferentes para a tangente da inclinação da reta denotada por a:

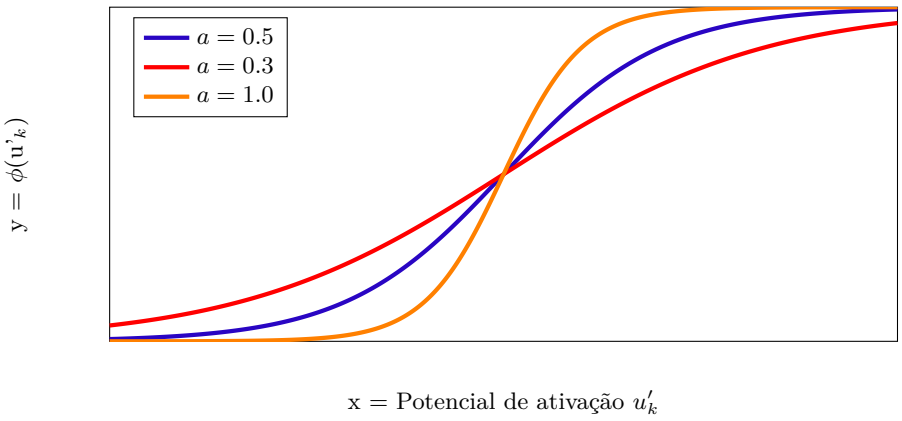

Figura 3 – Função de ativação Sigmoide
Fonte: Autor

2.1.3 Aprendizado

Segundo Braga *et al.* (2007, p. 12), uma das características mais importantes da RNA é a capacidade de aprender através de exemplos. O aprendizado de uma rede neural ocorre por meio de um processo iterativo de ajustes dos pesos. Os parâmetros da rede são ajustados visando a minimização do erro que pode ser calculado por uma função de custo. Entende-se que a partir deste ponto, o treinamento da rede neural é um problema de otimização.

Os paradigmas de aprendizado diferem basicamente na forma como o ajuste dos pesos são realizados. Neste estudo, o paradigma de aprendizagem adotado é o aprendizado supervisionado, aplicável a tarefas em que se deseja obter um mapeamento entre os padrões de entrada e saída da rede.

2.1.4 Aprendizado Supervisionado

O paradigma do aprendizado supervisionado requer a existência de um supervisor responsável para estimular as entradas da rede por meio de padrões de entrada e coletar o resultado obtido, comparando-o com a saída desejada.

Como a resposta da rede esta altamente relacionada aos valores do vetor de pesos, eles são ajustados a cada iteração para diminuir o erro entre os valores obtidos na saída da rede e a saída desejada. A minimização do erro é incremental, já que os ajustes são feitos nos pesos a cada etapa (época) de treinamento, para que a rede alcance uma solução se houver (BRAGA *et al.*, 2007, p.13).

Um dos exemplos mais conhecidos de algoritmos para aprendizado supervisionado é o *back-propagation*. O funcionamento deste algoritmo será abordado no decorrer deste capítulo.

2.2 Arquiteturas de Redes Neurais Artificiais

O modo como os neurônios são estruturados está diretamente ligado ao algoritmo de aprendizado utilizado no treinamento da *RNA*. Na obra de Haykin (2009), foram identificados três tipos de arquiteturas de redes neurais artificiais, que serão abordadas nesta seção:

2.2.1 Redes Neurais feedforward com uma única camada

Segundo Haykin (2009, p. 43), esta arquitetura é composta por uma camada de entrada que direcionam os dados à uma camada de saída. Essa rede é chamada de uma rede de camada única. A camada de entrada não é contabilizada devida a inatividade de computação nos nós de entrada, o que leva a definição da rede como *feedforward* com uma única camada.

2.2.1.1 Perceptron

O modelo *Perceptron* foi o primeiro modelo de Redes Neurais introduzido (ROSENBLATT, 1957). Segundo Estrada (2015), este trabalho foi o ponto de partida na área de aprendizagem supervisionada. O perceptron é construído em torno de um neurônio não-linear, ilustrado na figura 1.

Este modelo consiste em um combinador linear seguido por uma função de ativação ϕ, executando a função *limiar*. O neurônio calcula a soma ponderada dos sinais de entrada da rede com seus respectivos pesos e incorpora o bias. O campo local induzido formado

por meio da soma ponderada é fornecido à função de ativação e então o neurônio produz uma saída igual a 1 se o resultado obtido for positivo e -1 se for negativo.

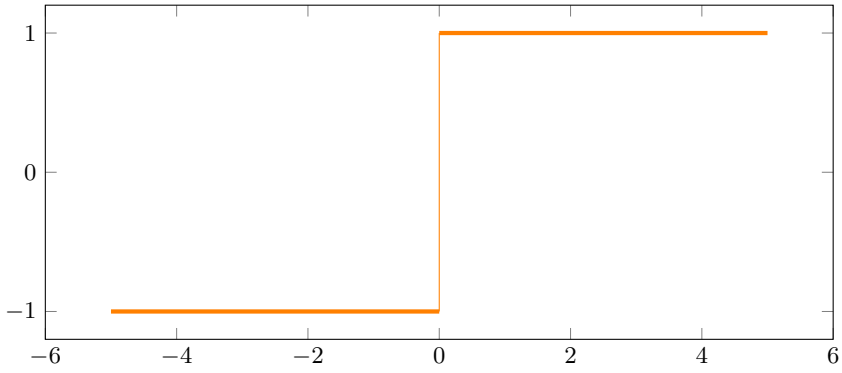

Figura 4 – Função de ativação *limiar* utilizada pelo Perceptron
Fonte: Autor

Diante deste comportamento, Estrada (2015) enfatiza que o *Perceptron* pode classificar corretamente os sinais de entrada entre classes representadas por C_1 e C_2. A classificação ocorre de acordo com a saída da rede y_n, em que, os resultados os quais $y = 1$ pertence a classe C_1 e $y = -1$ pertence a classe C_2.

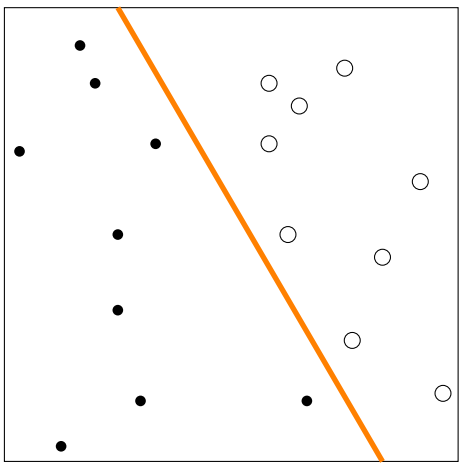

Figura 5 – Conjunto de dados linearmente separados
Fonte: Autor

Segundo Braga *et al.* (2007), para que o ajuste dos pesos de um *Perceptron* leve a uma solução satisfatória, é necessário que as classes em questão sejam linearmente separáveis, conforme ilustrado na figura 5, o que leva modelo *Perceptron* a ser restrito à tarefas de classificação.

2.2.2 Redes Neurais feedforward multicamadas

A segunda classe de uma rede neural feedforward difere principalmente pela presença de uma ou mais camadas de processamento ocultas no qual os neurônios são denominados neurônios ocultos ou unidades escondidas. Haykin (2009) refere ao termo "escondido" devido ao fato de que esta parte da rede neural não é vista diretamente da entrada ou saída da rede.

Ao adicionar uma ou mais camadas ocultas, a rede é capaz de realizar tarefas mais complexas quando comparadas a uma rede feedforward simples, enquadrando na classificação de algoritmos de aprendizado profundo (*Deep Learning*) mencionados por Goodfellow *et al.* (2016). A figura 6 a seguir representa a arquitetura de uma rede neural artificial com uma camada oculta.

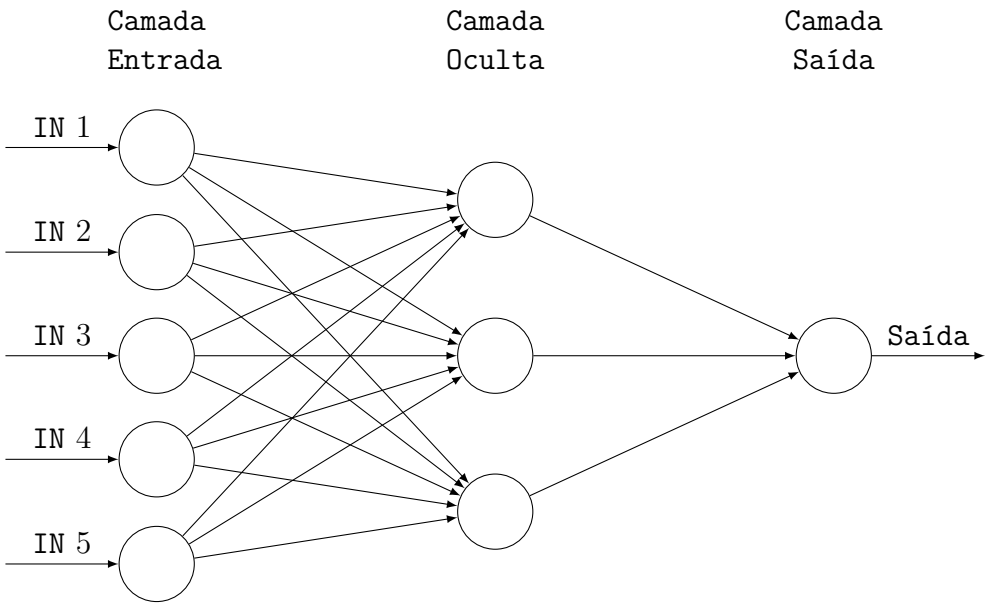

Figura 6 – Exemplo de Rede neural feedforward com uma camada oculta e uma camada de saída
Fonte: Autor

A RNA na figura 6 está totalmente conectada no sentido de que cada nó em cada camada da rede está conectado a qualquer outro nó na camada a frente. Se algumas das ligações (conexões sinápticas) estiver faltando na rede, diz-se que a rede neural está parcialmente conectada.

2.2.2.1 Perceptron Multicamadas (MLP)

Segundo Braga *et al.* (2007, p. 67), o papel das múltiplas camadas de procesamento em uma rede *feedforward* como a rede Perceptron de Múltiplas Camadas (MLP) é trans-

formar, sucessivamente, o problema descrito pelo conjunto de dados no espaço de entrada em uma representação tratável para a camada de saída, o que possibilita a representação de problemas não-linearmente separáveis.

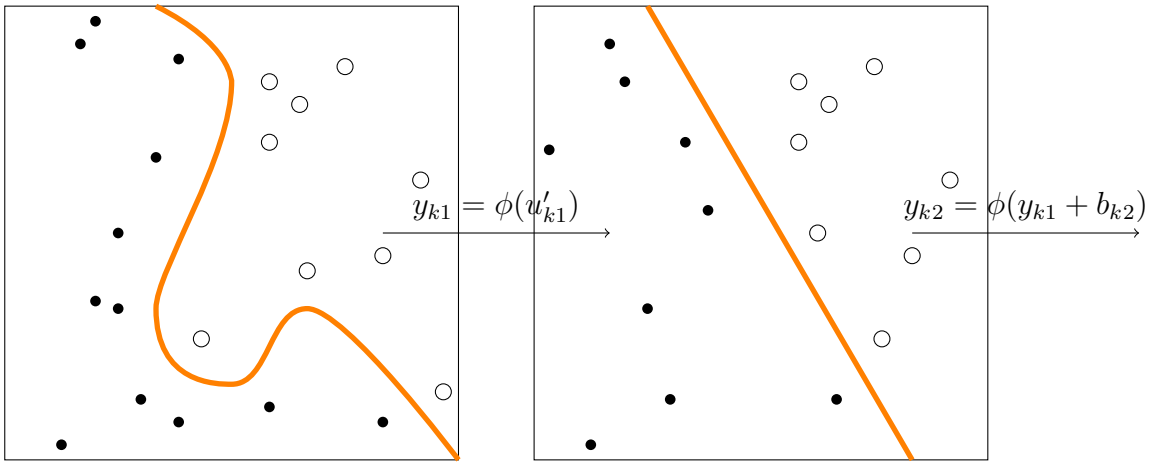

Figura 7 – Esquema do mapeamento sucessivo em camadas realizado por uma rede MLP
Fonte: Autor

Segundo Braga *et al.* (2007, p. 68), redes com duas camadas intermediárias podem implementar qualquer função, seja ela linearmente separável ou não. O comportamento de uma rede MLP pode ser representado na figura 7 por meio de transformações sucessivas.

Apesar de demonstrar a capacidade de tratar problemas mais complexos, este tipo de rede é mais adequada para tarefas de classificação do que tarefas que envolvem valores sequenciais, pois a saída de uma rede *feedforward* multicamada depende apenas da entrada corrente e não de quaisquer registros passados ou futuros (GRAVES, 2012, p.13). Esta tarefa pode ser trivial quando modelada utilizando redes recorrentes, tópico que será abordado a seguir.

2.2.3 Redes Neurais Recorrentes

A terceira classificação citada por Haykin (2009) são as Redes Neurais Recorrentes que diferem da rede MLP por existir ao menos um mecanismo de realimentação (*feedback*). A realimentação existe em um sistema dinâmico sempre que a saída de um elemento no sistema influencia na proxíma entrada aplicada à rede. Segundo Haykin (2009) uma rede recorrente pode consistir em uma ou mais camadas de neurônios, em que existe pelo menos uma malha de realimentação(*loops de feedback*) dos neurônios da camada de saída para as entradas. A saída da rede recorrente é determinada pela entrada corrente e a saída anterior conforme ilustrado na figura 8.

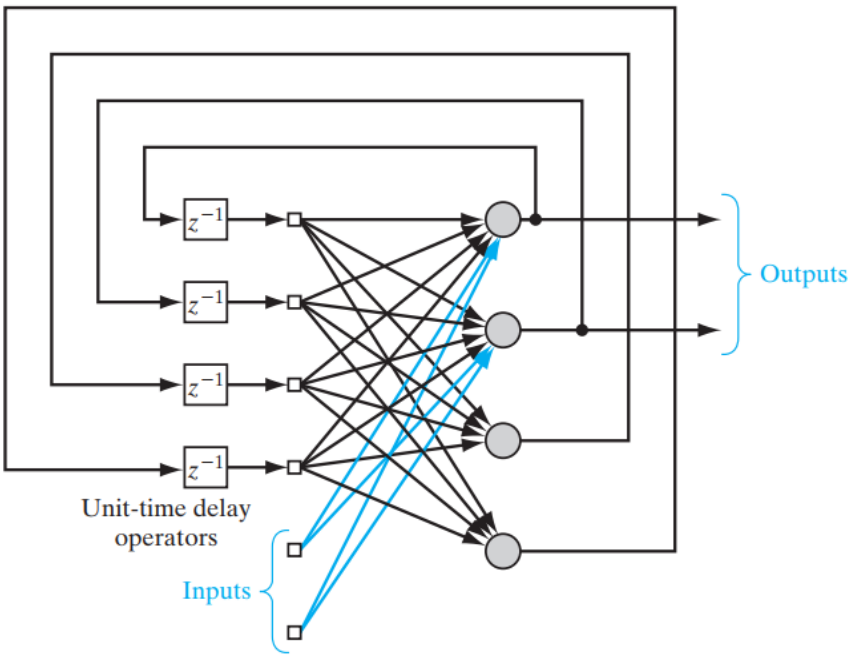

Figura 8 – Rede Recorrente com uma camada oculta
Fonte: Haykin (2009)

Para Haykin (2009), a presença de mecanismos de realimentação tem um profundo impacto na capacidade de aprendizagem da rede e em seu desempenho. Os mecanismos envolvem o uso de tipos de neurônios que contêm o resultado anterior obtido (representado por z^{-1}) conhecidos como neurônios de contexto. Esse tipo de neurônio permite que a rede neural mantenha o seu estado anterior. Como resultado, uma determinada entrada nem sempre produz a mesma saída.

Segundo Pascanu *et al.* (2013), embora a rede recorrente seja simples e modelo poderoso, na prática, é difícil treinar. Entre os principais motivos pelos quais esse modelo é complexo são problemas como *vanishing gradient* e *exploding gradient*. O problema nas redes recorrentes convencionais é que a influência causada por uma determinada entrada nas camadas ocultas decai exponencialmente a medida em que passa através da rede. Este efeito é referido na literatura como *vanishing gradient*, ilustrado na figura 9.

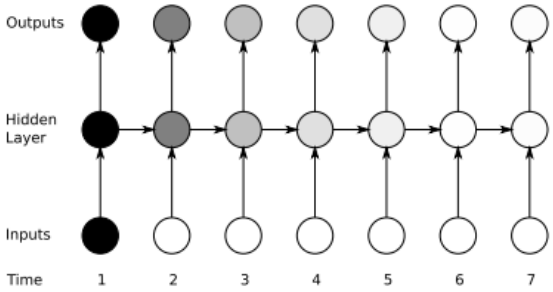

Figura 9 – Vanishing Gradient em RNNs.
Fonte: Graves (2012)

O sombreamento nos nós representa a sensibilidade da rede em relação aos dados de entrada. Quanto mais escuro for a coloração, maior a sensibilidade. A figura 9 ilustra como a sensibilidade da rede decai ao longo do tempo quando as novas entradas sobrescrevem as ativações da camada oculta,"esquecendo" as primeiras entradas.

Por outro lado, se os pesos desta matriz forem grandes, isso pode levar a uma situação em que o gradiente é tão grande que pode causar o problema referido na literatura como como *exploding gradient*.

2.3 Redes Long Short-Term Memory

As redes de memória de curto prazo, conhecida na literatura como *Long Short-Term Memory* introduzidas por Hochreiter e Schmidhuber (1997) é uma variante de RNR capaz de aprender dependências de longo prazo. A arquitetura LSTM contribui principalmente para mitigar os problemas relacionados ao aprendizado de dependências de longo prazo mencionados anteriormente (*vanishing gradient* e *exploding gradient*).

A arquitetura LSTM consiste em um conjunto de sub-redes conectadas na estrutura de uma RNR, conhecidas como blocos de memória. Cada bloco contém uma ou mais células de mémoria e três unidades multiplicativas: os portões de entrada (*input gate*), os portões de saída (*output gate*) e os portões de "esquecimento" (*forget gate*) ilustrados na figura 10.

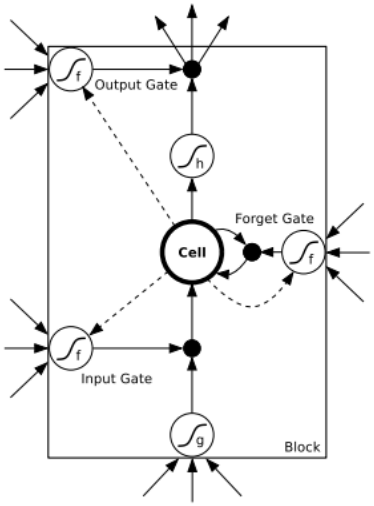

Figura 10 – Bloco de memoria LSTM com uma célula
Fonte: Graves (2012)

Uma rede LSTM é equivalente a rede recorrente convencional, exceto que a soma as unidades de entrada na camada oculta são substituídas por blocos de memória, como ilustrado na figura 11.

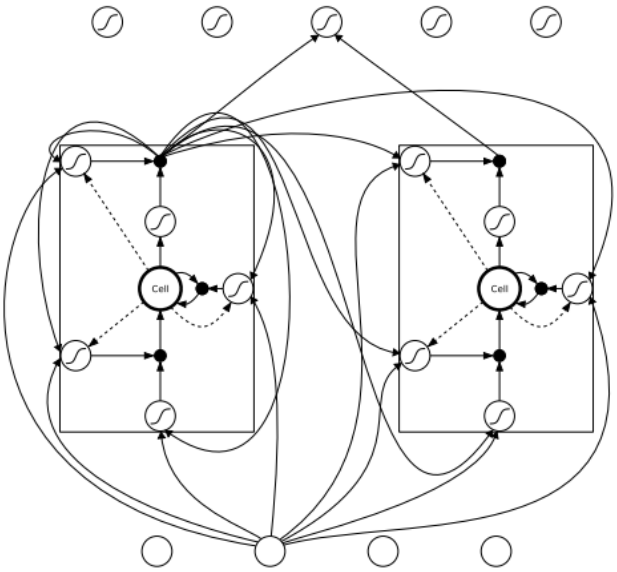

Figura 11 – Rede LSTM
Fonte: Graves (2012)

As unidades multiplicativas permitem que células de memória LSTM armazenem e acessem informações durante longos períodos de tempo, mitigando a ocorrência dos problemas citados. Graves (2012) explica que, enquanto o portão de entrada permanecer fechado (ativação perto de 0), a ativação da célula não será substituída pelo novas entradas que chegam à rede e, portanto, podem ser disponibilizadas para a rede muito mais tarde em sequência, abrindo o portão de saída.

Na figura 12, é ilustrada a utilização das unidades multiplicativas da célula LSTM, utilizando o mesmo cenário da figura 9.

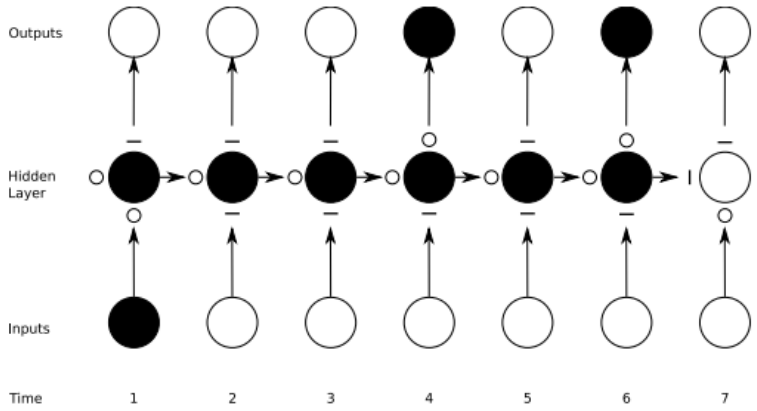

Figura 12 – Preservação do gradiente pelo LSTM
Fonte: Graves (2012)

Segundo Graves (2012, p. 33), os portões de entrada e saída multiplicam a respectiva entrada e saída da célula, enquanto o *Forget gate* multiplica o estado anterior da célula. Nenhuma função de ativação é aplicada dentro da célula. A função de ativação do portão representada na figura 10 por f é geralmente o sigmoide logístico, de modo que as ativações do portão estejam entre {0,1} (porta fechada é representada por 0 e a porta aberta por 1). A forma como as células de memórias da arquitetura LSTM são utilizadas contribuem para a preservação do gradiente na rede recorrente, ilustrada na figura 12.

2.4 Treinamento de Redes Neurais

Esta seção está destinada a proporcionar uma visão do treinamento de redes neurais. O primeiro algoritmo que será apresentado é o *back-propagation* que, segundo Haykin (2009), é o algoritmo mais popular para treinamento de redes neurais e, geralmente, é utilizado no treinamento de redes *feedforward*. Nas seções a seguir, serão definidos os recursos necessários para o funcionamento do algoritmo back-propagation.

2.4.1 Correção de erros

Segundo Braga *et al.* (2007), a abordagem mais típica aplicada ao aprendizado supervisionado é a correção de erros, em que se procura minimizar o erro da saída atual da rede em relação à saída desejada. Em termos matemáticos, a expressão para o erro $e(t)$ no instante de tempo t pode0 ser representada por:

$$e(t) = y_d(t) - y(t) \tag{2.6}$$

onde $y_d(t)$ é a saída desejada da rede e $y(t)$ representa a saída atual calculada pela rede. Os algoritmos de aprendizado que atualizam os pesos por correção de erros utilizam a seguinte expressão:

$$w_i(t+1) = w_i(t) + \eta e(t) x_i(t) \tag{2.7}$$

onde $w_i(t)$ corresponde ao vetor de pesos no instante t, η representa a taxa de aprendizado, $e(t)$ representa a medida de erro e x_i a entrada para o neurônio i no instante t.

2.4.2 Backpropagation

Segundo Graves (2012) as redes feedforward multicamada são por construção operadores diferenciais, o que possibilita o treinamento da rede visando minimizar a função custo (erro) através do método de descida do gradiente (*gradient descent*). Nesta seção, será abordado o conceito do método descida do gradiente quando relacionado ao algoritmo back-propagation. O treinamento pelo algoritmo *back-propagation* ocorre em duas fases; fase *forward* e *backward*. A fase *forward* é utilizada para definir a saída da rede dado um padrão de entrada. Os pesos sinápticos da rede são fixos e os dados de entrada da rede neural são propagados pela rede, camada por camada, até chegar a camada de saída.

Segundo Braga *et al.* (2007), a fase *forward* envolve os seguintes passos:

1. O vetor de entrada x é apresentado as entradas da rede, e as saídas do neurônio da primeira camada escondida C_1 são calculadas.

2. As saídas da camada escondida C_1 proverão as entradas da camada seguinte C_2. As saídas da camada C_2 são então calculadas. O processo se repete até que se chegue à camada de saída C_k.

3. As saídas produzidas pelos neurônios da camada de saída são então comparadas às saídas desejadas y_d para aquele vetor de entrada x, e o erro correspondente $y_d - y$ é calculado.

Na fase *backward*, um sinal de erro produzido é propagado através da rede, camada por camada, na direção oposta da fase *forward*. O objetivo da segunda fase é atualizar os pesos das conexões da rede. As etapas da fase *backward* serão detalhadas a seguir:

1. O erro da camada de saída C_k é utilizado para ajustar diretamente os seus pesos, utilizando-se para isso o gradiente descendente do erro, que, basicamente tem como objetivo de atualizar os pesos na direção oposta do vetor gradiente $\nabla\varepsilon(w)$. O vetor

gradiente identifica o sentido e a direção em que pode ser obtido o maior incremento possível para o vetor de pesos w.

2. Os erros dos neurônios da camada de saída C_k são propagados para a camada anterior C_{k-1}, utilizando-se para isso os pesos das conexões entre as camadas, que erão multiplicados pelos erros correspondentes. Assim, tem-se um valor de erro estimado para cada neurônio da camada escondida que representa uma medida de influencia de cada neuronio da camada C_{k-1} no erro de saída da camada C_k.

3. Os erros calculados para os neurônios da camada C_{k-1} são então utilizados para ajustar os pesos pelo gradiente descendente, analogamente ao procedimento utilizado para a camada C_k.

4. O processo se repete até que os pesos da camada C_1 sejam ajustados, concluindo-se assim o ajuste dos pesos de toda a rede para o vetor de entrada x e sua saida desejada y_D.

O numero de iterações do algoritmo pode ser controlado através do número de épocas fornecido.

2.4.3 Backpropagation Recorrente

Segundo Braga *et al.* (2007), existem dois meios para realizar o treinamento da rede neural, que são basicamente generalizações do algoritmo back-propagation citado anteriormente denominadas de *back-propagation through time* e redes recorrentes de tempo. Braga *et al.* (2007, p. 182) explicam que o algoritmo *back-propagation through time* é uma expansão do algoritmo convencional do back-propagation. Neste algoritmo, ao final da fase *forward pass*, os valores esperados são apresentados e o sinal de gradiente é calculado, retropropagando o sinal do erro no tempo. A desvantagem do treinamento por este método é que o nenhum aprendizado é realizado até que se alcance o fim da sequência.

2.4.4 Heurísticas para melhorar a performance do treinamento

Haykin (2009) lista um grupo de ações que empiricamente melhora o desempenho do algoritmo back-propagation. Nos tópicos a seguir, serão abordadas as heurísticas consideradas neste trabalho, além de outras ações observadas por Silva (1998):

- Maximizar a informação disponível: Haykin (2009) explica que os dados apresentados ao algoritmo de back-propagation devem ser escolhidos com base no fato de que o conteúdo selecionado seja o maior possível para a tarefa em questão. Esta heurística é motivada pelo desejo de obter um maior espaço de busca para atualização dos pesos da rede (HAYKIN, 2009, p. 145).

- Funções de ativação: Utilizar a função de ativação *tanh* contribui na aceleração do processo de aprendizagem do algoritmo. A função de ativação que segundo Haykin (2009) tem um processo de convergência eficiente é dada pela equação 2.8 a seguir, sendo que os valores de α e β são 1.7159 e $\frac{2}{3}$ respectivamente:

$$\phi(v) = \alpha tanh(\beta v) \tag{2.8}$$

- Valor esperado: O valor esperado deve estar em um intervalo aceito pela função de ativação *sigmoid* que equivale à [-1,1]. Haykin (2009, p.145) explica que caso os valor desejado esteja fora do intervalo acima, os parâmetros da rede tendem ao infinito, o que torna o processo de aprendizado lento.

- Normalização do conjunto de dados: Segundo Haykin (2009, p.146), os dados devem ser normalizados para que a média seja próxima de zero. A correlação entre os dados deve ser retirada.

- Inicialização: Haykin (2009, p.148) sugere que os pesos iniciais da rede devem seguir uma uma distribuição uniforme com média e variância igual ao número de conexões de um neurônio.

2.4.5 Parâmetros da Rede Neural

Além das modificações propostas por Haykin (2009), o uso de diversos algoritmos de aprendizagem envolvem diferentes conjuntos de parâmetros. Bengio (2012) contribui para a seleção dos valores iniciais destes parâmetros e demonstra formas de aprimoramento.

Para escolha de valores para os parâmetros da rede, existem diversos métodos de busca. Neste trabalho será abordado o conceito de *Grid Search* para a busca de valores para esses parâmetros.

Grid Search é um algoritmo de busca de força bruta, que, para cada parâmetro da rede, é escolhido um conjunto finito de valores para se explorar, sendo que o algoritmo treina o modelo especificado para cada combinação dos valores fornecidos (HEATON, 2015).

2.5 Trabalhos Relacionados

Nesta seção, são apresentados os trabalhos científicos relacionados e o impacto causado neste trabalho.

2.5.1 Pascanu *et al.* (2013)

Existem duas questões amplamente conhecidas com o treinamento de Redes Neurais Recorrentes, a ocorrência de problemas como *exploding gradients* e *vanishing gradients*. O autor Pascanu *et al.* (2013) demonstra para melhor compreensão os problemas citados acima, explorando-os a partir de uma análise empírica. Pascanu *et al.* (2013) mencionou soluções empíricas para mitigar o acontecimento do problema destacado e reforçou soluções previas apropriadas para evita-los, como por exemplo, o uso de aplicações de redes recorrentes LSTM. A partir deste, foi possível definir qual tipo de arquitetura de redes neurais que pode realizar a tarefa proposta.

2.5.2 Bengio (2012)

Os algoritmos de aprendizagem relacionados às redes neurais artificiais possuem diversos parâmetros. Bengio (2012) disponibiliza um guia prático que auxilia na escolha de valores para os parâmetros, particularmente no contexto de algoritmos que utilizam o método gradiente na correção dos pesos.

Em geral, este artigo descreve os elementos da prática utilizada para treinar com êxito redes neurais de grande escala. Bengio (2012) contribui principalmente na inicialização dos parâmetros da rede neural construída.

Entretanto, é cabível métodos de aprimoramento dos parâmetros. O autor Goodfellow *et al.* (2016) contribui para a escolha do método que auxilia na escolha de valores para os parâmetros apresentados.

3 Desenvolvimento da Rede Neural

Neste capítulo serão abordados todos os passos para percorridos para o desenvolvimento de um modelo de redes neurais que possa realizar a tarefa proposta, por meio dos conceitos adquiridos no decorrer do capítulo 2.

3.1 Domínio do Problema

O desafio deste trabalho é utilizar os conceitos da análise técnica para prever a precificação futura da criptomoeda por meio de duas arquiteturas de redes neurais: redes *feedforward* multicamada e redes recorrentes LSTM.

Por ser um domínio de problema que pode ser solucionado através de técnicas de aprendizado de máquina, este trabalho tem como desafio: coletar e pré-processar os dados visando obter um reconhecimento de padrão ou característica. A partir dos dados históricos, o preço da criptomoeda (X), o modelo construído será treinado de modo que possa representar os possíveis valores da precificação futura da moeda (Y).

Os experimentos previstos tem como objetivo avaliar a saída obtida pelo modelos construídos e compará-los à saída desejada. A avaliação deste resultado será realizada através da métrica `EQM`, que tem como objetivo principal analisar a diferença entre dois conjuntos de dados, no caso a saída obtida e a saída desejada.

Nas seções a seguir, serão abordados todos os passos percorridos para o desenvolvimento de um modelo de redes neurais que possa realizar a tarefa proposta.

3.2 Coleta dos dados

Nesta seção, serão abordados quais dados e como foram coletados para realizar a tarefa proposta. No trecho a seguir são definidas as principais premissas utilizadas na análise técnica mencionadas pelo autor Chaves (2004).

1. Os preços de mercado descontam tudo (fundamentos econômicos, políticos, psicológicos entre outros)
2. Os preços movem-se em tendências
3. A história se repete, ou seja, que o comportamento dos preços no passado se repetem no futuro

Utilizando o preceito de que as três premissas citadas sustentam a análise técnica, foram coletadas as cotações históricas da criptomoeda negociadas pela *exchange*[1] Mercado Bit Coin com um intervalo de tempo diário. Os dados utilizados neste estudo foram coletados através de API's de dados disponibilizadas e apresentam um histórico de 5 anos de cotações.

Os dados utilizados que compõem as cotações coletadas são informações de preço (abertura, fechamento, máximo, mínimo) consolidadas, denominadas *candles*. Em outras palavras, um *candle* é uma forma gráfica que consolida o preço no início do período, o preço final do período, o máximo e mínimo que o preço atingiu dentro de um intervalo em uma única representação, exemplificada na figura 13.

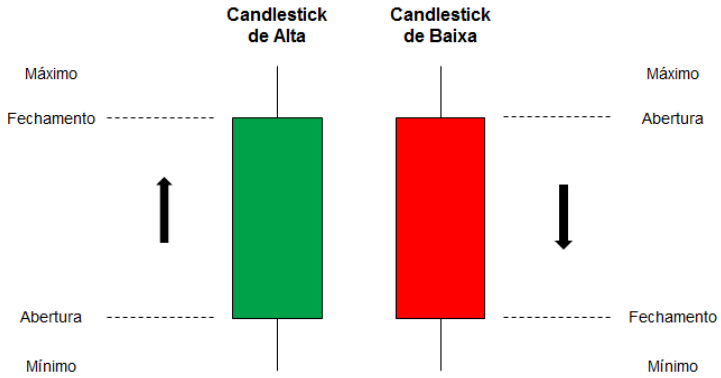

Figura 13 – *Candlestick*
Fonte: Bussola do Investidor (2013)

3.3 Pré Processamento de dados

A partir dos dados coletados, é necessário a realização de um conjunto de ações, para que os dados sejam postos de maneira plausível a um modelo de redes neurais. Nesta seção, serão abordados quais foram os pontos críticos durante o pré-processamento dos dados coletados.

3.3.1 Indicadores Técnicos

A partir dos dados de coletados, são gerados um conjunto de indicadores técnicos através da ferramenta `TA-Lib`.

Essa ferramenta é capaz de gerar mais de 150 diferentes indicadores de análise técnica, listados na Tabela. Esses indicadores expressam diversas características da série de preços, como força e direção de tendências, medidas de risco e possíveis reversões. Estas informações serão utilizadas para ajudar a prever se haverá ou não alta em n dias no futuro.

[1] Exchange é uma plataforma de negociação para compra e venda de um ativo ou moeda

Embora existam diversos indicadores que contribuam para a análise do comportamento dos preços, foram utilizados somente alguns destes escolhidos ao acaso, que, serão utilizados como entrada para o algoritmo de aprendizado de máquina responsável por prever as variações de preço.

Lista dos indicadores técnicos utilizados	
Sigla	**Indicador**
SMA	*Simple Moving Average*
ADX	*Average Directional Movement Index*
RSI	*Relative Strength Index*
BBAND	*Bollinger Bands*
DX	*Directional Movement Index*

Tabela 1 – Lista dos indicadores técnicos escolhidos
Fonte: Autor

3.3.2 Normalização dos dados

A influência dos dados de entrada no treinamento de uma `RNA` pode ser vista da seguinte maneira: se duas entradas, conectadas à mesma unidade, possuem intervalos de variação muito diferentes, os pesos ligados a estas unidades devem ser atualizados de maneira equalizada (SILVA, 1998, p. 80). Este autor ressalta que o aprendizado da rede pode ser muito demorado do ponto de vista computacional quando há uma variação grande nos intervalos de valores das diferentes entradas. O problema citado pode ser mitigado utilizando-se uma simples transformação linear nos padrões de entrada, resultando em uma normalização, com objetivo de facilitar a convergência no treinamento e generalização na previsão.

A equação 3.1 a seguir representa o cálculo realizado para a normalização de cada observação i do conjunto de dados coletado, de modo que os dados de entrada assumam valores em um intervalo [-1,1].

$$z_i = \frac{x_i min(x)}{max(x) min(x)} \tag{3.1}$$

3.3.3 Janelas Deslizantes (*Sliding Window*)

Tratando se de uma tarefa que envolve aprendizado supervisionado, é necessário dispor de um conjunto de entradas e relacioná-los a outro conjunto denominado conjunto de saída. O método janelas deslizantes converte um problema de aprendizado supervisionado de sequencias em problemas clássicos de aprendizado supervisionado. Para melhor entendimento, Heaton (2015) exemplificou o método da seguinte forma:

Considerando uma sequência dada por uma série de números:

$$1, 2, 3, 4, 3, 2, 1, 2, 3, 4, 3, 2, 1 \tag{3.2}$$

Uma rede neural que prevê números dessa sequência pode utilizar três neurônios de entrada e um único neurônio de saída. O conjunto de treinamento a seguir tem uma janela de previsão de tamanho 1 e um tamanho de janela anterior de tamanho 3 conforme a tabela abaixo.

Entrada	**Saída**
[1,2,3]	[4]
[2,3,4]	[3]
[3,4,3]	[2]
[4,3,2]	[1]

Tabela 2 – Disposição dos dados quando utilizada a técnica Janelas Deslizantes

De um modo geral, a transformação realizada pelo método de janelas deslizante tem como objetivo o uso de dados de treinamento, validação e teste que movem conforme o tempo. Utilizando desta técnica para a segmentação dos dados, o conjunto de observações foi disposto conforme a descrição abaixo para cada observação coletada:

- X(Entrada): 5 dias de informações sobre a cotação como dados de entrada
- Y(Saída): 10 valores sequencias que representam a cotação dos 10 próximos dias (dados de saída)

Além disso, os dados coletados foram separados em conjuntos de treino e validação, no qual tem-se:

- 1571 *candles* que compõe o conjunto de treinamento
- 175 *candles* que compõe o conjunto para validação
- Total de 1756 *candles* coletados, que representam um histórico cujo a data está entre 12/06/2013 à 02/04/2018

Logo, para cada observação são utilizados as cinco *candles* anteriores e seus respectivos indicadores técnicos para prever as 10 observações seguintes que são os preço no fechamento de mercado.

3.4 Construção da Rede Neural

O algoritmo utilizado para realizar a previsão será de redes neurais multicamada seguindo a premissa de que uma única camada intermediária é suficiente para aproximar qualquer função contínua e duas camadas intermediárias são suficientes para aproximar qualquer função (BRAGA *et al.*, 2007).

A rede construída tem como tarefa prever a precificação da criptomoeda bitcoin nas 10 próximas observações dada 5 observações anteriores.

Diante deste cenário, foi necessário avaliar o desempenho de redes neurais multicamada convencionais (*multilayer perceptron* no caso) comparado a arquitetura de redes neurais capaz de preservar dependências entre os dados a longo prazo, denominadas redes LSTM.

Os modelos construídos para realizar este comparativo utilizam 85 variaveis de entradas listadas abaixo:

Quantidade	**Descrição da variavel**
20	5 candles com informações do bitcoin
5	Volume transacionado para cada candle coletado
5	Quantidade negociada de bitcoins no dia para cada candle coletado
5	Número de negociações realizadas no dia para cada candle coletado
5	Preço unitário médio das negociações no dia para cada candle coletado
45	Conjunto de indicadores para cada candle

Tabela 3 – Variáveis utilizadas na construção do modelo de RNA para cada observação analisada
Fonte: Autor

Inicialmente, foi utilizada uma rede neural com uma camada intermediária, sendo que cada neurônio da camada intermediária utiliza células de memória LSTM, implementando a função de ativação *tanh* conforme a heurística abordada na seção 2.4.4. Já a camada de saída é composta por uma função de ativação linear dada pela equação 3.3 abaixo:

$$\phi(x) = x \tag{3.3}$$

Entre as camadas, foram utilizados conceitos de Regularização, que é abordado por Heaton (2015). Regularização é uma técnica que reduz o sobreajuste[2] (*overfitting*). No contexto de redes neurais, a regularização é implementada através de Camadas de *Dropout* (SRIVASTAVA *et al.*, 2014), que escolhe randomicamente neurônios e conexões da camada

[2] Produção de um modelo que possui um bom desempenho com os dados de treino, entretanto não produz um desempenho regular utilizando os dados de teste.

anterior para serem descartados temporariamente durante o treinamento da rede dada uma probabilidade de este caso acontecer (HEATON, 2015, pos. 2681).

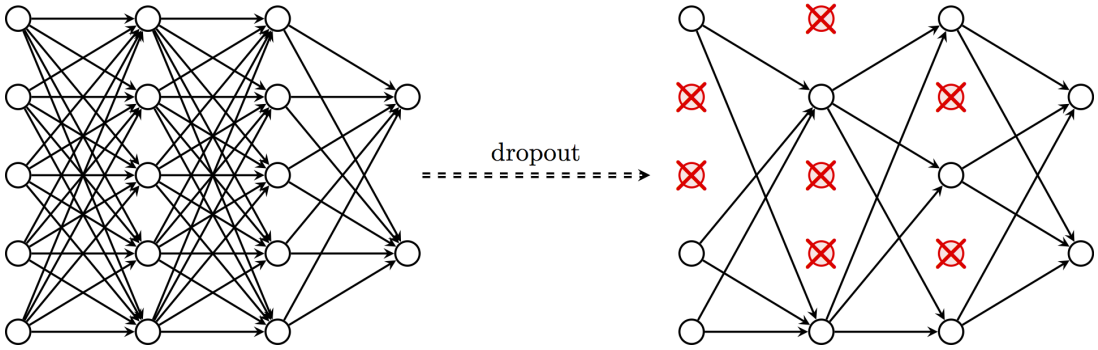

Figura 14 – Demonstração do efeito da aplicação de camadas *Dropout* em uma rede neural
Fonte: Heaton (2015)

A rede foi treinada utilizando o método de otimização *RMSprop*, que é uma generalização do método de descida do gradiente, utilizado para a correção dos pesos da rede. Este método foi escolhido com base nas conclusões de Ruder (2016). De acordo este autor, o uso do método *RMSprop* é indicado a problemas que possuem escassez de dados. Este método é baseado em gradiente com taxa de aprendizado adaptativa, que possibilita uma rápida convergência no treinamento da rede. O desenvolvimento da rede neural foi feita através da linguagem de programação Python utilizando o framework *Keras*, que possui abstrações de alto nível de diversas ferramentas de aprendizado de máquina.

3.5 Análise dos experimentos

Os experimentos previstos para este trabalho consistem na execução de *backtests* para o modelo construído usando os dados históricos da criptomoeda. *Backtesting* é o processo de validar uma estratégia de investimento usando dados passados. Neste caso, parte dos dados coletados serão separados com a finalidade de avaliar a performance da rede neural por meio da métrica erro médio quadrático (EQM). Essa métrica tem como resultado a média das diferenças entre a saída obtida (y_o) e a saída esperada (y_e) ao quadrado em conjunto de n observações.

$$EQM = \frac{1}{n}\sum_{i=1}^{n}(y_o - y_e)^2 \tag{3.4}$$

3.5.1 Conjunto dos parâmetros e o respectivo desempenho para cada arquitetura de rede neural construída

Neste trabalho foi abordado o conceito de *Grid Search* para a busca de valores para os parâmetros mencionados. Com base na obra de Goodfellow *et al.* (2016, p.428), a

definição do conjunto de valores para o número de neurônios é dada por uma escala logarítmica de base 2. O número de épocas foi definido por meio de uma escala aritmética dada por $50x$. A taxa de regularização foi definida entre dois valores {0.4,0.5} respectivamente de uma forma empírica. O espaço de busca definido possibilita a geração de 24 modelos possíveis para cada arquitetura e topologia de rede construída. Os valores utilizados estão representados na tabela 4 à seguir:

Parâmetros	**Conjunto de valores utilizados**
Neurônios	{32,64,128,256}
Épocas	{50,100,150}
Taxa de Regularização (Dropout)	{0.4,0.5}

Tabela 4 – Espaço de busca utilizado

O uso de um espaço de busca limitado se dá pelo fato de que a execução de cada modelo computacional gerado exige um esforço computacional relativamente alto.

3.6 Desempenho dos modelos construídos

Nesta seção, serão apresentados os resultados obtidos para cada arquitetura de rede neural construída. Para os dois tipos de arquiteturas abordadas neste trabalho, foram utilizadas três tipos topologias:

- RNA com uma camada de regularização após a entrada de dados
- RNA com uma camada de regularização após a camada oculta
- RNA com camadas de regularização após a entrada dos dados e após a camada oculta

3.6.1 Redes LSTM

A topologia ilustrada na figura 15 representa a aplicação do modelo de redes LSTM com uma camada de regularização após a camada de entrada dos dados, utilizando a combinação de parâmetros que minimiza o EQM obtido. Os valores obtidos para a topologia descrita está presente na tabela 6 do anexo A.

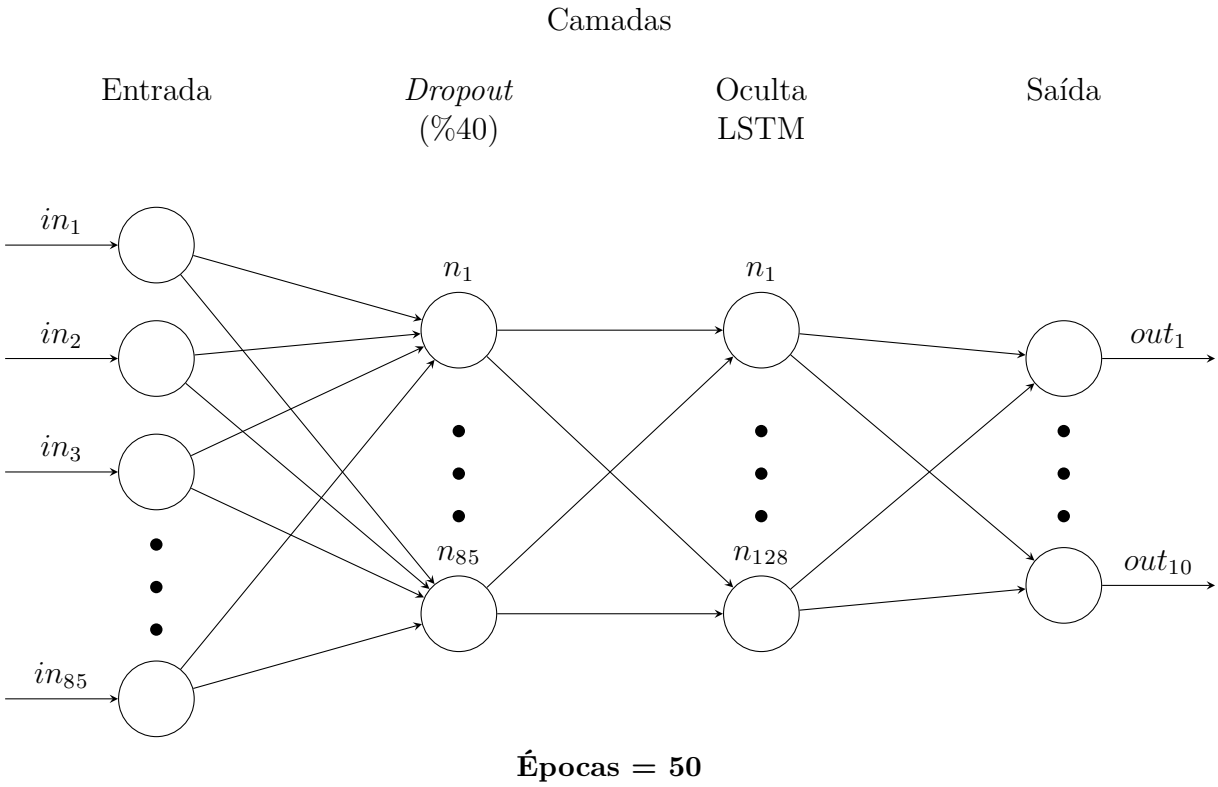

Figura 15 – Topologia da rede construída com uma camada de regularização após a camada de entrada de dados

Fonte: Autor

Quando aplicado ao conjunto de treinamento, o erro médio quadrático (EQM) entre a valor real e o valor gerado pela rede é ≈ 0.236. Esta mesma métrica quando aplicada ao conjunto de teste e os valores gerados é de ≈ 0.219. Na figura 16, são apresentados os resultados da previsão no conjunto de treino e teste.

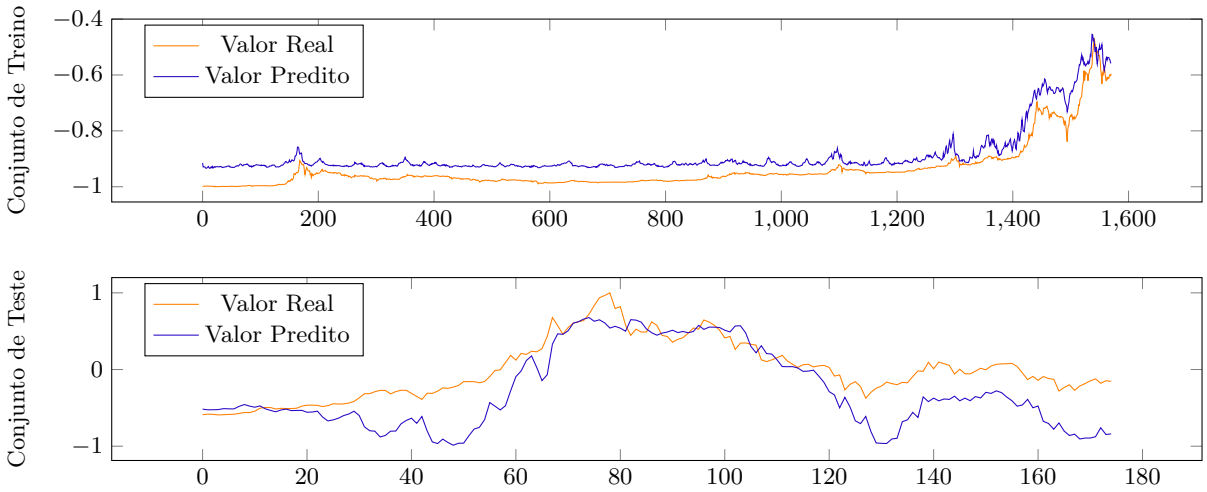

Figura 16 – Aplicação do modelo de redes LSTM no conjunto de treino e teste
Fonte: Autor

A topologia ilustrada na figura 17 representa a aplicação do modelo de redes LSTM com camadas de regularização antes e depois da camada oculta , utilizando a combinação de parâmetros que minimiza o EQM obtido. Os valores obtidos para a topologia descrita está presente na tabela 7 do anexo A.

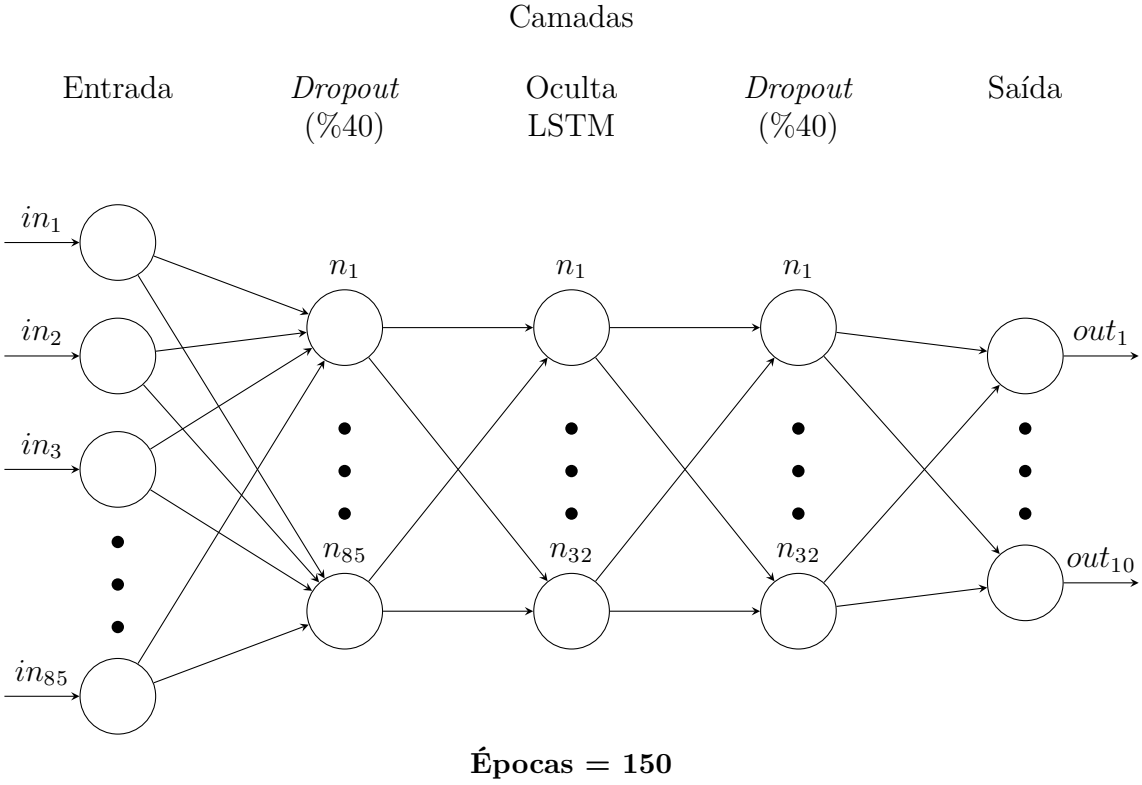

Figura 17 – Topologia da rede construída com camadas de regularização após a camada de entrada de dados e após a camada oculta

Fonte: Autor

Quando aplicado ao conjunto de treinamento, o erro médio quadrático (EQM) entre a valor real e o valor gerado pela rede é ≈ 0.202. Esta mesma métrica quando aplicada ao conjunto de teste e os valores gerados é de ≈ 0.480. Na figura 18, são apresentados os resultados da previsão no conjunto de treino e teste.

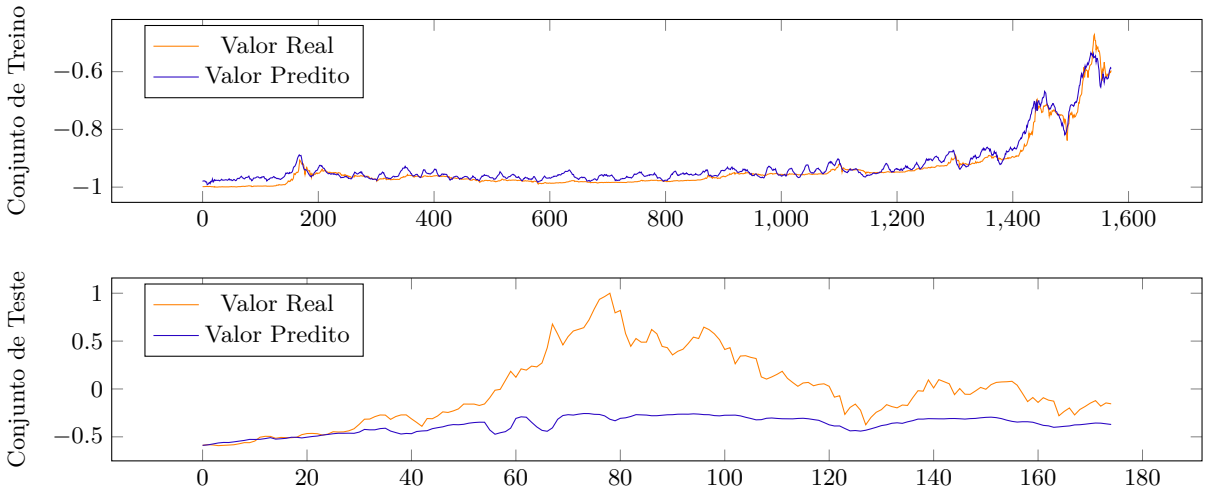

Figura 18 – Aplicação do modelo de redes LSTM no conjunto de treino e teste
Fonte: Autor

A topologia ilustrada na figura 19 representa a aplicação do modelo de redes LSTM com uma camada de regularização após a camada oculta, utilizando a combinação de parâmetros que minimiza o EQM obtido. Os valores obtidos para a topologia descrita está presente na tabela 8 do anexo A.

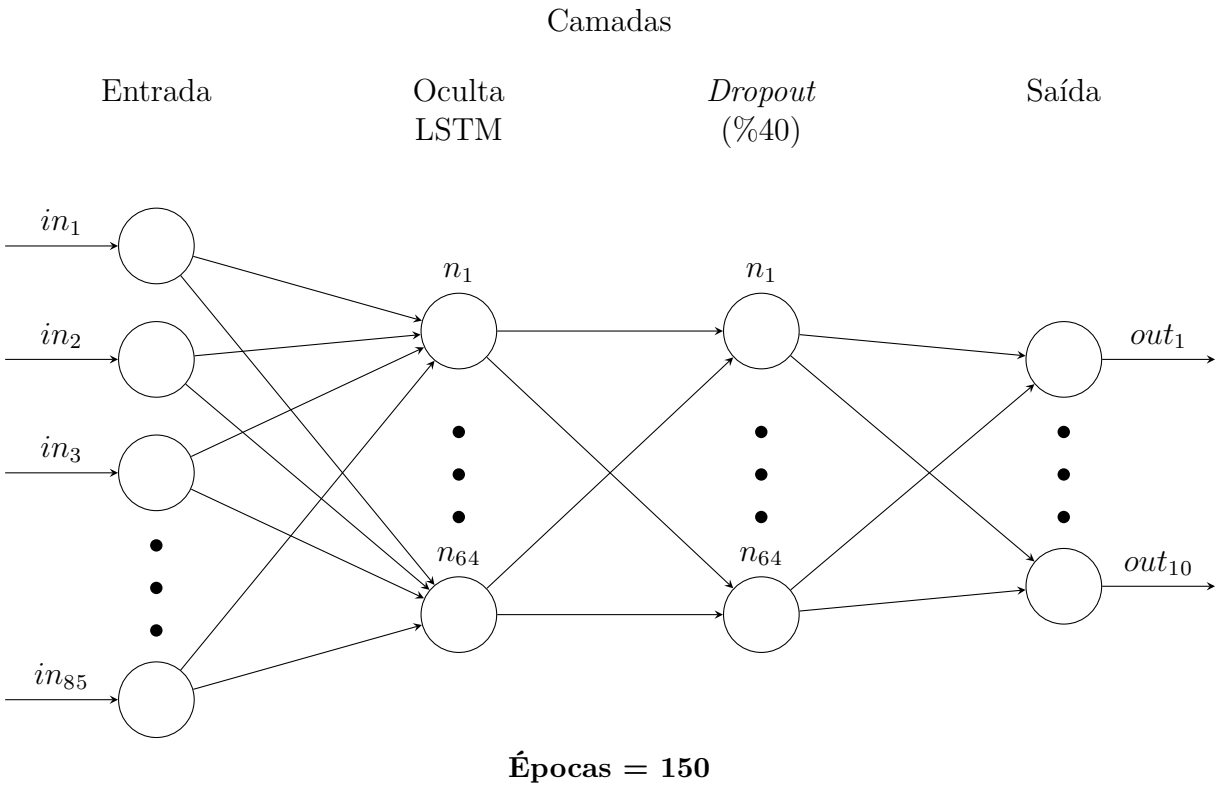

Figura 19 – Topologia da rede construída com uma camada de regularização após a camada oculta

Fonte: Autor

Quando aplicado ao conjunto de treinamento, o erro médio quadrático (EQM) entre a valor real e o valor gerado pela rede é ≈ 0.205. Esta mesma métrica quando aplicada ao conjunto de teste e os valores gerados é de ≈ 0.387. Na figura 20, são apresentados os resultados da previsão no conjunto de treino e teste.

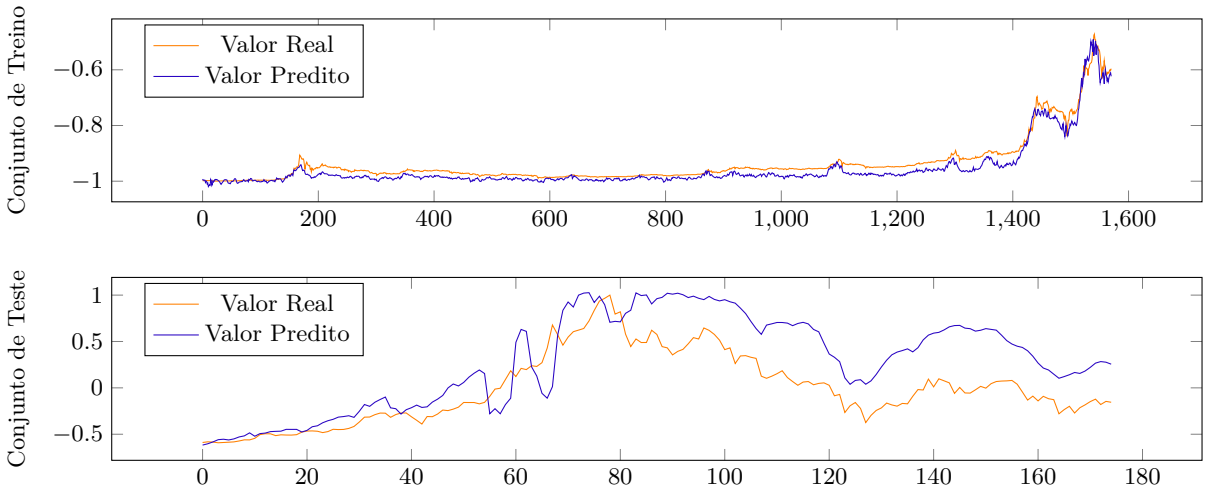

Figura 20 – Aplicação do modelo de redes LSTM no conjunto de treino e teste
Fonte: Autor

3.6.2 Redes *feedforward* MLP

Nesta seção serão apresentados os resultados para cada topologia descrita para arquitetura *feedforward* MLP.

A topologia ilustrada na figura 21 representa a aplicação do modelo de redes *feedforward* MLP com uma camada de regularização após a camada de entrada dos dados, utilizando a combinação de parâmetros que minimiza o EQM obtido. Os valores obtidos para a topologia descrita está presente na tabela 9 do anexo A.

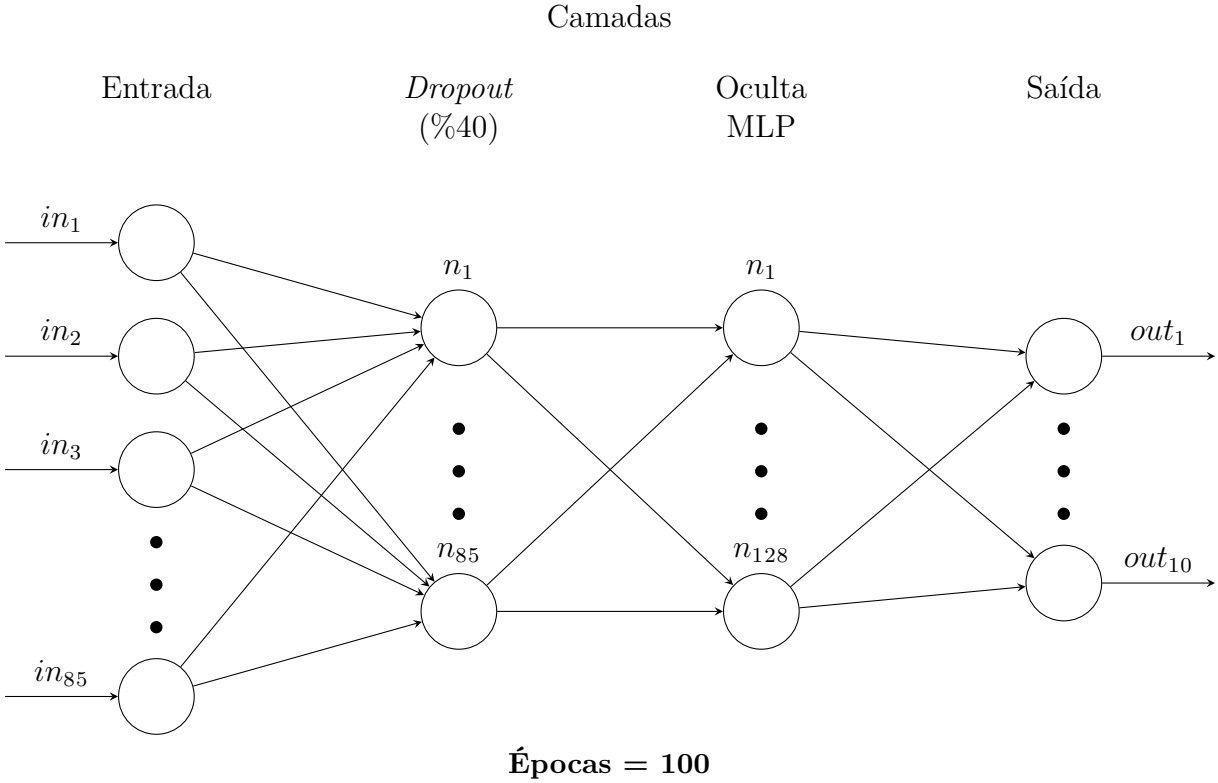

Figura 21 – Topologia da rede construída com uma camada de regularização após a camada de entrada de dados

Fonte: Autor

Quando aplicado ao conjunto de treinamento, o erro médio quadrático (EQM) entre a valor real e o valor gerado pela rede é ≈ 0.205. Esta mesma métrica quando aplicada ao conjunto de teste e os valores gerados é de ≈ 0.630, o que caracteriza uma situação de *overfitting*, ou seja, a combinação entre a respectiva arquitetura e topologia generaliza apenas os dados de treino, porém não é utilizável quando aplicado a novos dados. Na figura 22, são apresentados os resultados da previsão no conjunto de treino e teste.

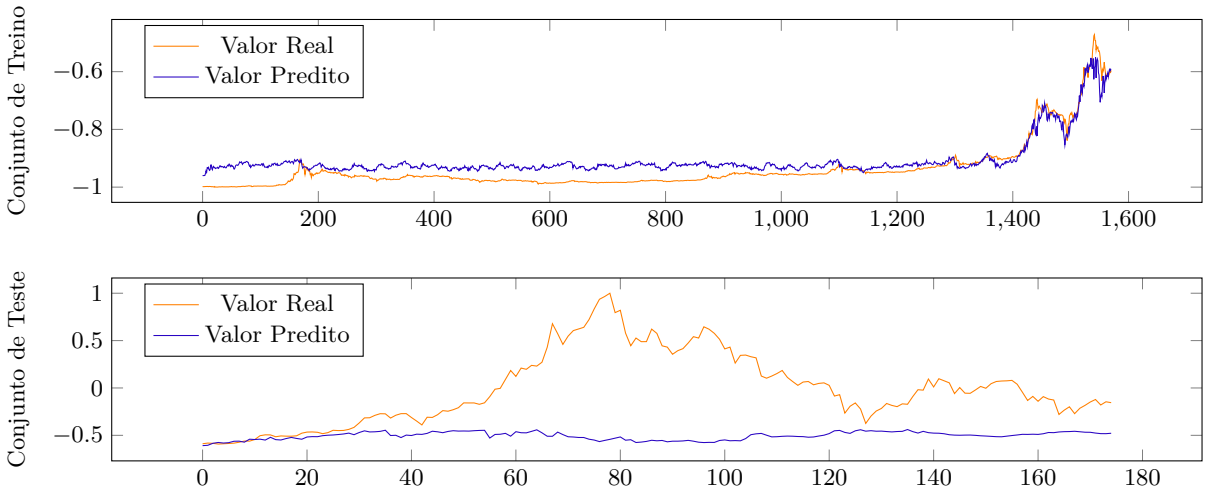

Figura 22 – Aplicação da topologia apresentada na figura 21 no conjunto de treino e teste
Fonte: Autor

A topologia ilustrada na figura 23 representa a aplicação do modelo de redes *feedforward* MLP com uma camada de regularização após a camada de entrada dos dados e após a camada oculta, utilizando a combinação de parâmetros que minimiza o EQM obtido. Os valores obtidos para a topologia descrita está presente na tabela 10 do anexo A.

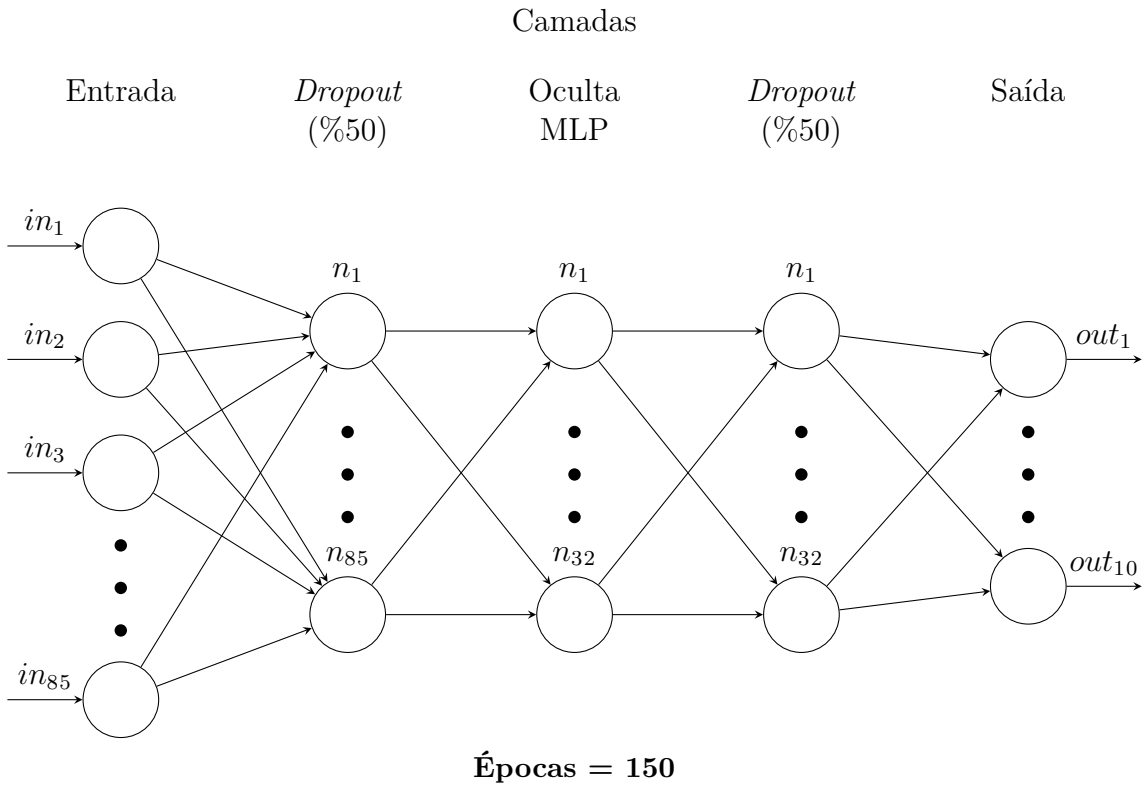

Figura 23 – Topologia da rede construída com camadas de Regularização após a camada de entrada de dados e após a camada oculta

Fonte: Autor

Quando aplicado ao conjunto de treinamento, o erro médio quadrático (EQM) entre a valor real e o valor gerado pela rede é ≈ 0.198. Esta mesma métrica quando aplicada ao conjunto de teste e os valores gerados é de ≈ 0.695, o que caracteriza também um quadro de *overfitting*. Na figura 24, são apresentados os resultados da previsão no conjunto de treino e teste.

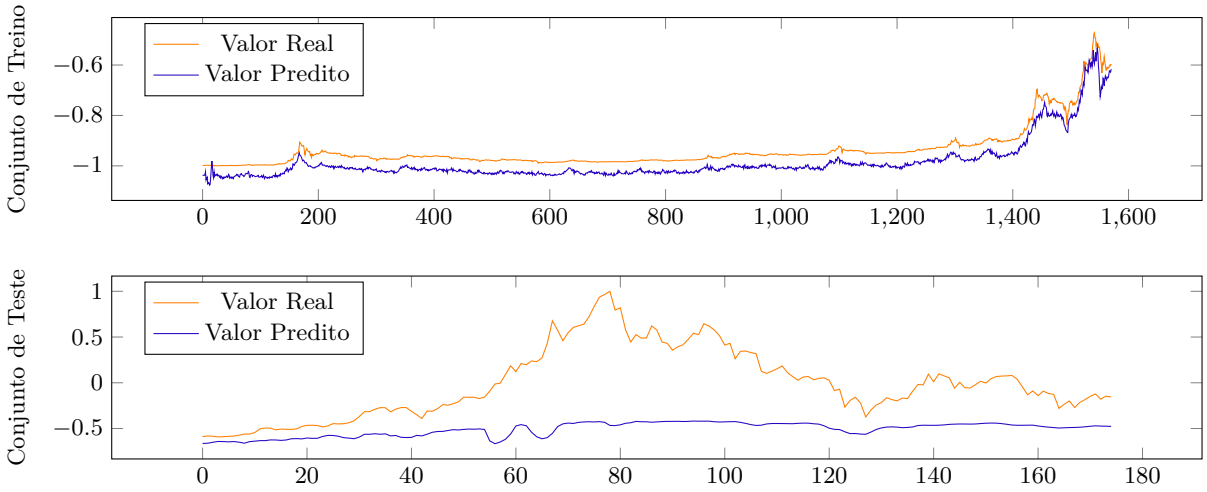

Figura 24 – Aplicação do modelo de redes *feedforward* no conjunto de treino e teste
Fonte: Autor

A topologia ilustrada na figura 25 representa a aplicação do modelo de redes *feedforward* MLP com uma camada de regularização após a camada oculta dos dados, utilizando a combinação de parâmetros que minimiza o EQM obtido. Os valores obtidos para a topologia descrita está presente na tabela 11 do anexo A.

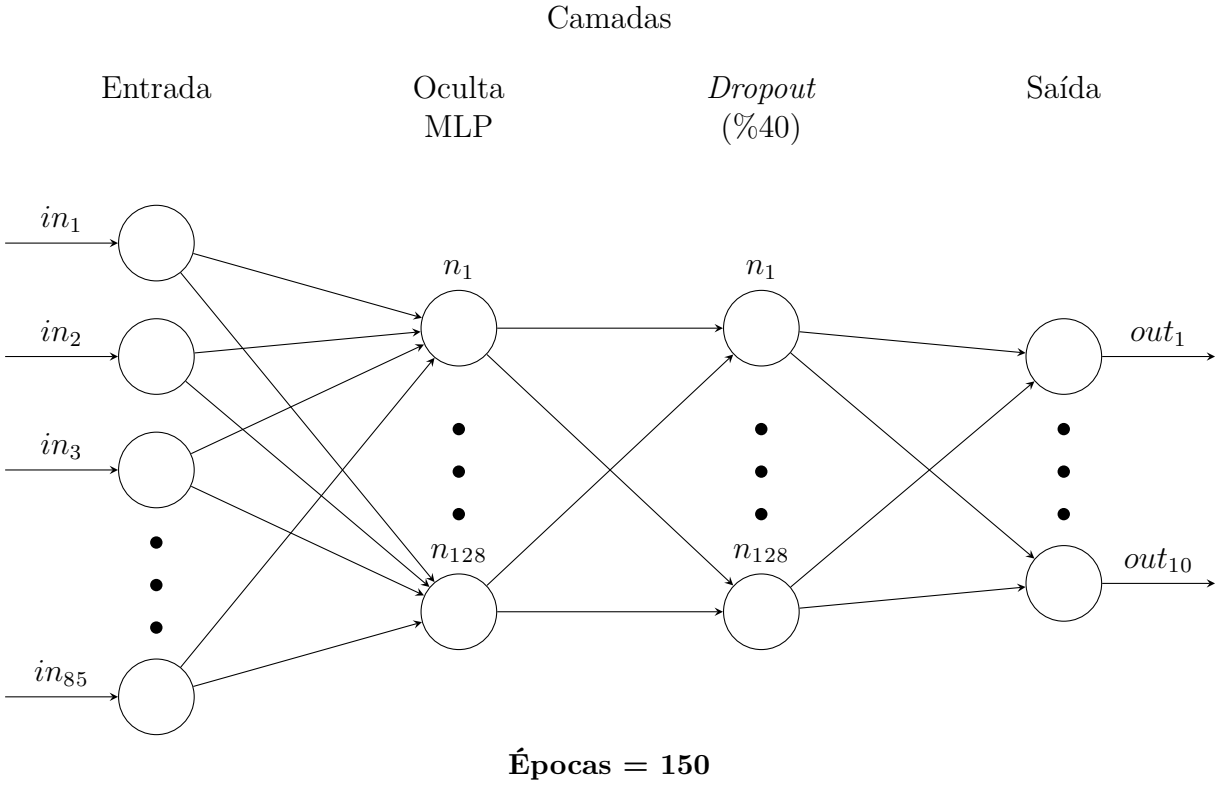

Figura 25 – Topologia da rede construída com uma camada de regularização após a camada oculta

Fonte: Autor

Quando aplicado ao conjunto de treinamento, o erro médio quadrático (EQM) entre a valor real e o valor gerado pela rede é ≈ 0.2. Esta mesma métrica quando aplicada ao conjunto de teste e os valores gerados é de ≈ 0.250. A performance da rede quando aplicada a novos dados é similar a aplicação da rede neural aos dados de treinamento, o que caracteriza uma generalização aceitável. Entretanto, a rede construída de acordo com essa arquitetura tende a não prever os picos de volatilidade característicos da série analisada conforme a figura 26 abaixo.

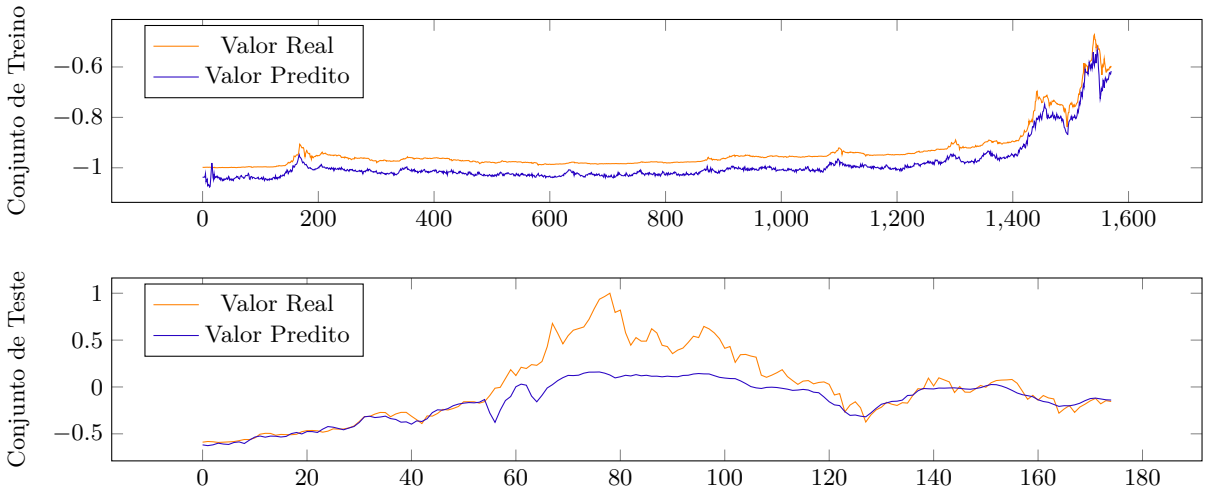

Figura 26 – Aplicação do modelo de redes *feedforward* no conjunto de treino e teste
Fonte: Autor

4 Conclusão

A precificação da moeda virtual bitcoin apresentou uma grande volatilidade em todo o seu histórico principalmente após o segundos semestre de 2017. Devido a este comportamento altamente volátil, traçar uma estimativa futura da criptomoeda é uma tarefa complexa. Portanto, para a visualização futura deste *commodity*, o desenvolvimento de soluções que consigam identificar de forma assertiva a tendência da precificação é de grande relevância para o publico que acompanha este mercado.

Neste trabalho, foram apresentados um dos meios para o tratamento de cotações do mercado financeiro em geral. Procurou-se aplicar os conceitos utilizados no mercado financeiro para estimativa futura de *commodities* e ativos em geral, a análise técnica.

Para a análise, foi coletada uma base de dados com 1756 dias de cotação da criptomoeda sem necessidade de limpeza dos dados. O próximo passo foi gerar para cada observação coletada, o conjunto de indicadores técnicos que auxiliam na obtenção de uma tendência futura, segundo a análise técnica. Depois, como o seguinte trabalho utiliza redes neurais para resolver a tarefa proposta, as observações coletadas foram reorganizadas de acordo com a técnica de janelas deslizantes (*Sliding Window*), transformando os dados sequenciais em dados passíveis de aprendizado supervisionado. Além disso, os dados foram segmentados em dois conjuntos; treino e teste.

O próximo passo foi identificar quais arquiteturas de redes neurais, no caso redes *feedforward* MLP e redes recorrentes LSTM. Conforme o autor (BRAGA *et al.*, 2007), somente uma camada oculta foi utilizada para a modelagem, pois segundo o autor, apenas uma camada é o suficiente para adquirir funções de aproximação. Na fase de modelagem, topologias com camadas *Dropout* foram utilizadas para melhor o desempenho da rede.

Dentre os testes realizados, a utilização de redes LSTM com uma camada de *Dropout* após a camada de entrada mostrou-se eficaz na generalização e tratamento de domínios que envolvam dados sequenciais. O melhor resultado para cada arquitetura de rede está demonstrado na tabela 5:

Tipo	**EQM (Treino)**	**EQM (Teste)**
LSTM	0.236	0.219
MLP	0.2	0.250

Tabela 5 – Benchmark entre Redes *feedforward* MLP e LSTM
Fonte: Autor

Topologias que envolvem arquiteturas *feedforward* MLP não obtiveram êxito na previsão onde existiram picos relativamente grandes de alta e baixa na cotação da cripto-

moeda, conforme a figura 26. A partir deste resultado, um conjunto de ações podem ser tomadas para aprimorar o resultado obtido.

De acordo com Haykin (2009) maximizar a informação disponível pode trazer melhores resultados do que os atingidos neste trabalho. De todos os indicadores técnicos existentes na ferramenta `Ta-lib` citada na seção 3.3.1, foram utilizados somente 5 indicadores escolhidos ao acaso.

O uso de todos os indicadores para um estudo de métodos de redução de dimensionalidade como *Principal Component Analysis* (PCA) é essencial para identificar quais indicadores tem um impacto relativamente significativo para a realização da tarefa em questão.

Outra sugestão para trabalhos futuros seria expandir o espaço de busca utilizado para identificar os melhores parâmetros da rede que minimizem o erro obtido. A maior dificuldade encontrada foi identificar o conjunto de parâmetros que possibilita um baixo índice de erro na previsão. Por ser um método de busca por força bruta, o esforço para se identificar o melhor conjunto no método *Grid Search* é de $O(n^m)$, onde m são os parâmetros da rede e n os respectivos valores. Infelizmente, por possuir um custo computacional exponencial, o espaço de busca à ser utilizado tende a ser pequeno.

Todavia, um ponto importante deste trabalho é a definição de um conjunto reduzido de opções para o parâmetro *Dropout*, possibilitando explorar mais valores para outros parâmetros da rede. A utilização do conjunto {0.4,0.5} para o parâmetro *Dropout* mostrou-se eficiente para o problema proposto.

APÊNDICE A – Tabelas de simulação

Neste apêndice são demonstrados os valores da métrica *Erro Médio Quadrático* para cada combinação de variáveis, arquitetura e topologia construída. As linhas coloridas simbolizam o conjunto de parâmetros que minimiza o valor da métrica EQM.

Regularização	**Épocas**	**Neurônios**	**EQM**	σ(**EQM**)
0.4	50	128	0.235646	0.294379
0.4	50	256	0.289809	0.359506
0.4	100	128	0.245939	0.306824
0.4	100	256	0.250580	0.298394
0.4	150	128	0.286270	0.348354
0.4	150	256	0.237935	0.281292
0.5	50	128	0.255631	0.310580
0.5	50	256	0.301991	0.399380
0.5	100	128	0.351464	0.468924
0.5	100	256	0.324461	0.422721
0.5	150	128	0.314329	0.413147
0.5	150	256	0.243614	0.312911

Tabela 6 – Resultado de cada combinação de parâmetros para o modelo de redes LSTM com uma camada de regularização após a camada de entrada dos dados
Fonte: Autor

Regularização	**Épocas**	**Neurônios**	**EQM**	σ(**EQM**)
0.4	50	32	0.226309	0.221907
0.4	50	64	0.236531	0.262295
0.4	50	128	0.224942	0.263560
0.4	50	256	0.231489	0.251632
0.4	100	32	0.214796	0.256333
0.4	100	64	0.221080	0.261605
0.4	100	128	0.219007	0.236909
0.4	100	256	0.213006	0.265049
0.4	150	32	0.201650	0.247637
0.4	150	64	0.213788	0.258193
0.4	150	128	0.213676	0.261817
0.4	150	256	0.222543	0.257960
0.5	50	32	0.213814	0.255307
0.5	50	64	0.215613	0.255920
0.5	50	128	0.208402	0.247995
0.5	50	256	0.211363	0.258754
0.5	100	32	0.201861	0.256903
0.5	100	64	0.224384	0.249115
0.5	100	128	0.222143	0.259771
0.5	100	256	0.213746	0.254377
0.5	150	32	0.203543	0.252129
0.5	150	64	0.203313	0.250095
0.5	150	128	0.210727	0.259231
0.5	150	256	0.212168	0.242396

Tabela 7 – Resultado de cada combinação de parâmetros para o modelo de redes LSTM com camadas de regularização entre a camada oculta
Fonte: Autor

Regularização	**Épocas**	**Neurônios**	**EQM**	σ(**EQM**)
0.4	50	32	0.206239	0.254340
0.4	50	64	0.216618	0.256217
0.4	50	128	0.249924	0.252555
0.4	50	256	0.267185	0.285493
0.4	100	32	0.214011	0.251788
0.4	100	64	0.211605	0.246984
0.4	100	128	0.214021	0.262130
0.4	100	256	0.248163	0.264834
0.4	150	32	0.233125	0.274259
0.4	150	64	0.205147	0.257493
0.4	150	128	0.206114	0.251091
0.4	150	256	0.230799	0.281724
0.5	50	32	0.239418	0.236273
0.5	50	64	0.209353	0.254184
0.5	50	128	0.207855	0.260909
0.5	50	256	0.244288	0.291682
0.5	100	32	0.222295	0.278760
0.5	100	64	0.220432	0.240192
0.5	100	128	0.205409	0.253743
0.5	100	256	0.222586	0.249900
0.5	150	32	0.289922	0.365547
0.5	150	64	0.226089	0.283386
0.5	150	128	0.205838	0.250097
0.5	150	256	0.224233	0.250707

Tabela 8 – Resultado de cada combinação de parâmetros para o modelo de redes LSTM com uma camada de regularização após a camada oculta
Fonte: Autor

Regularização	**Épocas**	**Neurônios**	**EQM**	σ(**EQM**)
0.4	50	32	0.273954	0.282598
0.4	50	64	0.230415	0.264650
0.4	50	128	0.213689	0.246958
0.4	50	256	0.242480	0.261591
0.4	100	32	0.249555	0.316623
0.4	100	64	0.242141	0.306910
0.4	100	128	0.207813	0.259227
0.4	100	256	0.211184	0.265754
0.4	150	32	0.248624	0.314100
0.4	150	64	0.214176	0.270538
0.4	150	128	0.207865	0.262178
0.4	150	256	0.205243	0.252310
0.5	50	32	0.249911	0.316209
0.5	50	64	0.233159	0.277933
0.5	50	128	0.238737	0.256901
0.5	50	256	0.229248	0.255431
0.5	100	32	0.278170	0.342645
0.5	100	64	0.212105	0.241674
0.5	100	128	0.208771	0.253871
0.5	100	256	0.213256	0.261469
0.5	150	32	0.269957	0.322032
0.5	150	64	0.246853	0.309155
0.5	150	128	0.211516	0.259442
0.5	150	256	0.214719	0.267311

Tabela 9 – Resultado de cada combinação de parâmetros para o modelo de rede *feedforward* com uma camada de regularização após a camada de entrada
Fonte: Autor

Regularização	**Épocas**	**Neurônios**	**EQM**	σ(**EQM**)
0.4	50	32	0.211993	0.240487
0.4	50	64	0.231069	0.244032
0.4	50	128	0.242154	0.238427
0.4	50	256	0.216087	0.227234
0.4	100	32	0.213477	0.260901
0.4	100	64	0.228786	0.226499
0.4	100	128	0.204028	0.239046
0.4	100	256	0.214902	0.260181
0.4	150	32	0.203090	0.248672
0.4	150	64	0.211349	0.257255
0.4	150	128	0.200125	0.241188
0.4	150	256	0.214060	0.229140
0.5	50	32	0.216350	0.237637
0.5	50	64	0.232096	0.248191
0.5	50	128	0.231626	0.269164
0.5	50	256	0.236142	0.272164
0.5	100	32	0.211804	0.260965
0.5	100	64	0.224519	0.241043
0.5	100	128	0.239449	0.207641
0.5	100	256	0.222375	0.237226
0.5	150	32	0.214430	0.243177
0.5	150	64	0.210846	0.247704
0.5	150	128	0.220065	0.235349
0.5	150	256	0.221878	0.224723

Tabela 10 – Resultado de cada combinação de parâmetros para o modelo de rede *feedforward* com uma camada de regularização após a camada oculta
Fonte: Autor

Regularização	**Épocas**	**Neurônios**	**EQM**	σ(**EQM**)
0.4	50	32	0.219331	0.224047
0.4	50	64	0.226998	0.221661
0.4	50	128	0.245089	0.251182
0.4	50	256	0.223233	0.260578
0.4	100	32	0.214129	0.255354
0.4	100	64	0.233805	0.240257
0.4	100	128	0.223639	0.253470
0.4	100	256	0.221574	0.230559
0.4	150	32	0.221635	0.237805
0.4	150	64	0.222480	0.251421
0.4	150	128	0.214476	0.241509
0.4	150	256	0.215144	0.245318
0.5	50	32	0.243352	0.220219
0.5	50	64	0.237990	0.212858
0.5	50	128	0.214478	0.239494
0.5	50	256	0.225537	0.226149
0.5	100	32	0.207878	0.257686
0.5	100	64	0.207799	0.252380
0.5	100	128	0.202596	0.246857
0.5	100	256	0.209998	0.245517
0.5	150	32	0.198108	0.247470
0.5	150	64	0.216219	0.247764
0.5	150	128	0.217956	0.243017
0.5	150	256	0.225679	0.235761

Tabela 11 – Resultado de cada combinação de parâmetros para o modelo de rede *feedforward* com uma camada de regularização antes e depois da camada oculta

Fonte: Autor

APÊNDICE B – Codigo para coleta dos dados

```
from datetime import date
from datetime import timedelta
import json
from threading import Thread
import requests
import abc
from DatabaseConnection import MongoDB
from pymongo import errors


class RequestHistorico(Thread):
    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def request_data(self):
        "Assinatura da fun   o que realiza a requisicao de dados"

class MercadoBitcoin(RequestHistorico):
    market = "Mercado Bit Coin — Historical Data"

    def __init__(self, coin, date_request):
        super().__init__()

        self.coin = coin
        self.date_request = date_request
        self.source = "https://www.mercadobitcoin.net/api/" + self.coin + \
                      "/day-summary/" + self.date_request
        self.data = {}

    def request_data(self):

        try:
            requisition = requests.get(self.source)
```

```
            self.data.update(json.loads(requisition.text))
            self.data.update({"coin": self.coin})

        except requests.exceptions.ConnectionError as error:

            print("<Erro na requisicao dos dados> : %s" % error)

        except requests.exceptions.Timeout as error:

            print("<Erro na requisicao dos dados> : %s " % error)


class ColetorHistorico:
    cryptocurrencies = ['BTC', 'LTC', 'BCH']

    def __init__(self, request_type):
        self.data = {}
        self.utc_offset = 2
        self.request_type = request_type

    def get_historico(self, dinicio=None):
        if dinicio is None:
            dinicio = date(2014, 1, 1)

        dfim = date.today() - timedelta(days=1)
        periodo = dfim - dinicio

        for i in range(periodo.days + 1):
            date_parsing = str(dinicio + timedelta(days=i))
            date_request = date_parsing.replace("-", '/')

            iterator = [self.request_type(item, date_request) for item in self.cryptocurrencies]
            for item in iterator:
                item.request_data()

                print(item.data)


class Scraping:
```

```
def __init__(self, db_conn):
    "Coleta dos dados"
    print("Starting Scraping job ...")
    self.db_conn = db_conn
    self.historical_data = []

def get_historico_batch(self):

    coin_first_date = [{'BTC': date(2013, 6, 12)},\
                       {'LTC': date(2013, 8, 23)},\
                       {'BCH': date(2017, 8, 21)}]

    for item in coin_first_date:
        dinicio = list(item.values())[0]
        dfim = date.today() - timedelta(days=1)
        periodo = dfim - dinicio

        for i in range(periodo.days + 1):
            date_parsing = str(dinicio + timedelta(days=i))
            date_request = date_parsing.replace("-", '/')

            item_request = MercadoBitcoin(coin=list(item.keys())[0], date_request=date_request)
            item_request.request_data()
            self.historical_data.append(item_request.data)
            print(json.dumps(item_request.data, indent=4, sort_keys=True))

def armazenar_dados(self):
    db = db_conn.session['smartbot_database_tcc'].get_collection(MercadoBitcoin.market)

    for item in self.historical_data:
        try:
            db.insert_one(item)
            print("Data was successfully inserted in the follow \
            Colletion: %s " % MercadoBitcoin.market)
        except errors.OperationFailure as error:
            print("Could not apply the \
            operation %s" % error)
```

```
if __name__ == '__main__':
    db_conn = MongoDB()
    db_conn.iniciar_sessao()

    coletor_historico = Scraping(db_conn)
    coletor_historico.get_historico_batch()
    coletor_historico.armazenar_dados()
```

APÊNDICE C – Modelo de Redes Neurais Construido

"""Construção de uma arquitetura de redes neurais que utiliza os 8 parametros dados"""

```
from sklearn.metrics import mean_squared_error
from math import sqrt

def mean_squared_error(y_true, y_pred):
return K.mean(K.square(y_pred - y_true), axis=-1)

def root_mean_squared_error(y_true, y_pred):
return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1))

def rmse_metric(y_true,y_pred):
return np.sqrt(np.square(np.subtract(y_true, y_pred)).mean())

def visible_build_model_v3(train_X,train_y,test_X,test_y):

model = Sequential()

model.add(Dense(85, input_shape=(1, 85), activation = custom_tanh))

model.add(Dense(128, input_shape=(1, 85), activation = custom_tanh , kernel_initializer = 'random_un

model.add(Dropout(0.4))

model.add(Dense(10, activation = 'linear'))

model.compile(loss=root_mean_squared_error,
optimizer='rmsprop',
metrics =[root_mean_squared_error])


history = model.fit(train_X,
train_y,
```

```
batch_size=32,
epochs = 50,
verbose=2,
validation_data = (test_X,test_y))

return model,history
```

Referências

BENGIO, Y. Practical recommendations for gradient-based training of deep architectures. **CoRR**, abs/1206.5533, 2012. Disponível em: <http://arxiv.org/abs/1206.5533>.

BONALDI, E. V. **O pequeno investidor na bolsa de valores: Uma análise da ação e da cogitação econômica**. Dissertação (Artigo) — Universidade de São Paulo, 2010. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/317723/modresource/content/1/Bonaldi-constr_peq_invest.pdf>.

BRAGA, A. de P. *et al.* **Redes Neurais Artificiais: Teoria e Aplicações**. [S.l.]: LTC, 2007.

Bussola do Investidor. **Como Interpretar o Grafico de Candlestick**. 2013. Acesso em: 08/04/2018. Disponível em: <https://blog.bussoladoinvestidor.com.br/grafico-de-candlestick/>.

CHAVES, D. A. T. **Analise Tecnica e Fundamentalista: Divergencias, Similaridades e Complementariedades**. 2004. Acesso em: 02/03/2018. Disponível em: <goo.gl/PRjVfz>.

ESTRADA, G. B. **Deep Learning for Multivariate Financial Time Series**. Dissertação (Artigo) — KTH Royal Institute of Technology, 2015. Disponível em: <https://www.math.kth.se/matstat/seminarier/reports/M-exjobb15/150612a.pdf>.

EXAME, R. **Entenda o que e bitcoin**. 2017. Acesso em: 26/02/2017. Disponível em: <https://exame.abril.com.br/mercados/entenda-o-que-e-bitcoin/>.

GOODFELLOW, I. *et al.* **Deep Learning**. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>.

GRAVES, A. **Supervised Sequence Labelling with Recurrent Neural Networks**. Springer, 2012. v. 385. (Studies in Computational Intelligence, v. 385). ISBN 978-3-642-24796-5. Disponível em: <https://doi.org/10.1007/978-3-642-24797-2>.

HAYKIN, S. O. **Neural Networks and Learning Machines**. 3. ed. [S.l.]: Pearson, 2009. ISBN 0-13-147139-2.

HEATON, J. **Artificial Intelligence for Humans: Deep Learning and Neural Network**. [S.l.]: Jeff Heaton, 2015. (Volume 3).

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.

PASCANU, R. *et al.* On the difficulty of training recurrent neural networks. In: **Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013**. [s.n.], 2013. p. 1310–1318. Disponível em: <http://jmlr.org/proceedings/papers/v28/pascanu13.html>.

PREDIVI, G. de S. Descentralizacao monetaria: Um estudo sobre o bitcoin. 2014. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/116267/000953619.pdf?sequence=1>.

ROSENBLATT, F. The perceptron, a perceiving and recognising automaton. **Cornell Aeronautical Laboratory**, v. 85-460-1, 1957. Disponível em: <https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf>.

RUDER, S. An overview of gradient descent optimization algorithms. **CoRR**, abs/1609.04747, 2016. Disponível em: <http://arxiv.org/abs/1609.04747>.

SILVA, L. N. de C. Analise e sintese de estrategias de aprendizado para redes neurais artificiais. 1998. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/theses/lnunes_mest/>.

SRIVASTAVA, N. *et al.* Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>.