UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROJETO DE GRADUAÇÃO

RAFAEL COSTA DE ALMEIDA

USO DE REDES NEURAIS *LONG SHORT-TERM MEMORY* COMO ESTRATÉGIA DE *ALGORITHMIC TRADING*

VITÓRIA

2019

RAFAEL COSTA DE ALMEIDA

USO DE REDES NEURAIS *LONG SHORT-TERM MEMORY* COMO ESTRATÉGIA DE *ALGORITHMIC TRADING*

Parte manuscrita do Projeto de Graduação do aluno **Rafael Costa de Almeida**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Jorge Leonid Aching Samatelo

VITÓRIA
2019

Rafael Costa de Almeida

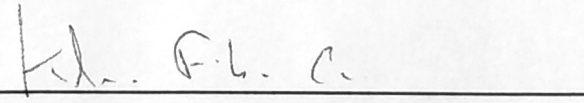
USO DE REDES NEURAIS LONG SHORT-TERM MEMORY COMO ESTRATÉGIA DE ALGORITHMIC TRADING

Parte manuscrita do Projeto de Graduação do aluno **Rafael Costa de Almeida**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

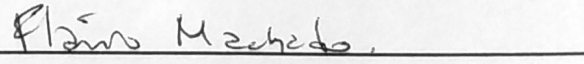
Aprovado em 13, de Dezembro de 2019.

COMISSÃO EXAMINADORA:

Prof. Dr. Jorge Leonid Aching Samatelo
Universidade Federal do Espírito Santo
Orientador



Prof. Dr. Klaus Fabian Coco
Universidade Federal do Espírito Santo
Examinador



Eng. Flávio Machado
Universidade Federal do Espírito Santo
Examinador

AGRADECIMENTOS

Aos meus pais, pelo apoio que me foi dado ao longo dos anos, sem eles nada disso seria possível. Ao professor Jorge Leonid Aching Samatelo pela grande assistência dada ao longo do desenvolvimento do projeto. À banca examinadora pela aceitação do convite e pelo tempo investido para leitura e avaliação desse trabalho. Finalmente, agradeço à Universidade Federal do Espirito Santo pela minha formação.

RESUMO

Com o avanço tecnológico, a forma como investidores atuam no mercado financeiro mudou. Máquinas podem gerar ordens de compra e venda de ativos em uma velocidade muito maior que a de reação humana e com maior acurácia. A bolsa de valores é um ambiente competitivo onde qualquer vantagem é o suficiente para fazer diferença. Prever variações no mercado financeiro é um problema difícil de se modelar. Devido à dinamicidade do mercado financeiro, ordens de compra e venda muitas vezes são feitas sobre emoção humana, decisões precipitadas que não tem embasamento ou até mesmo análises errôneas, levando à perdas em capital investido. Portanto, ter em mãos um sistema capaz de fazer previsões sobre o futuro no mercado financeiro é uma grande vantagem financeira e por isso, indispensável. Indicadores econômicos que visam sintetizar aspectos do mercado, assim como a aplicação de modelos matemáticos são alternativas amplamente utilizadas. Porém, por serem técnicas generalizadas, falham em se adaptar à certos comportamentos do mercado. Redes neurais profundas apresentam uma alternativa para a previsão de comportamentos de forma adaptável e dinâmica, assim como é o comportamento do mercado financeiro. Este trabalho descreve o uso de redes neurais a fim de criar um sistema de *algorithmic trading*. O foco é criar predições de tendência no mercado financeiro visando tomar melhores decisões de compra e venda, aumentando-se, assim, o retorno financeiro em relação a métodos padrões de análise técnica. O método proposto se baseia em utilizar redes neurais recorrentes do tipo *Long Short-Term Memory* (LSTM) sobre aprendizado supervisionado. Utiliza-se um banco de dados composto por séries temporais dos preços de fechamento de várias ações da bolsa de valores.

Palavras-chave: *Deep-learning*. Redes neurais recorrentes. Mercado Financeiro. Séries Temporais. *Long Short-Term Memory*. *Algorithmic Trading*.

ABSTRACT

With the advancement of technology, the way investors act at the financial market has changed. Machines can generate market signals at greater speeds than human reaction and with more accuracy. The stock market is a competitive environment where any advantage is enough to make a significant difference. Predicting variations in the financial market is a difficult problem to model. Due to the dynamics of the financial market, buying and selling orders are often made throught human emotion, hasty decisions that have no basis or even erroneous analyzes, leading to losses on invested capital. Hence, having a system capable of forecasting the future in the financial market is a great financial advantage and therefore indispensable. Economic indicators that aim to synthesize an aspect of the market, as well as the application of mathematical models are widely used alternatives. However, because they are generalized techniques, they fail to adapt to certain market behaviors. Deep neural networks presents an alternative for predicting behavior in an adaptive and dynamic way, as is the behavior of the financial market. This paper describes the use of neural networks in order to create an algorithmic trading system. The focus is to create trend predictions in the financial market in order to make better buying and selling decisions. This increases the financial return when compared to standard methods of technical analysis. The proposed method is to use recurrent neural networks of the Long Short-Term Memory (LSTM) type with supervised learning. A database is used consisting of time series of the closing prices of several shares of the stock exchange.

Keywords: Deep-learning. Recurrent Neural Networks. Financial Market. Time Series. Long Short-Term Memory. Algorithmic trading.

LISTA DE FIGURAS

Figura 1 – Modelo de uma RNN 29
Figura 2 – Tipos arquiteturas RNN 30
Figura 3 – Rede LSTM desdobrada no tempo 31
Figura 4 – Fluxo de informação referente ao *Forget Gate* 33
Figura 5 – Fluxo de informação referente ao *Input Gate* 33
Figura 6 – Atualização do estado da célula 34
Figura 7 – Fluxo de informação referente ao *Output Gate* 34
Figura 8 – Diagrama de redes LSTM totalmente conectadas 35
Figura 9 – Diagrama de redes LSTM totalmente conectadas achatadas 36
Figura 10 – Diagrama de redes LSTM multicamadas 37
Figura 11 – Estrutura de um *autoencoder* LSTM 38
Figura 12 – Diagrama de redes compostas 39
Figura 13 – Diagrama de redes neurais com *Layer Normalization* 48
Figura 14 – Fluxograma de Treinamento de redes neurais 49
Figura 15 – Fluxograma do processo de *Backtrading* 52
Figura 16 – Fluxograma da operação do servidor enviando dados 53
Figura 17 – Fluxograma de filtragem dos dados históricos 61
Figura 18 – Modelo de um Autoencoder 87

LISTA DE GRÁFICOS

Gráfico 1 – Exemplo de uma série temporal 20
Gráfico 2 – Exemplo de velas OHLC 22
Gráfico 3 – Valores de fechamento com indicadores de momento 24
Gráfico 4 – Valores de fechamento com indicador técnico MACD 26
Gráfico 5 – Valores de fechamento com indicador técnico ATR 27
Gráfico 6 – Dados recebidos 45
Gráfico 7 – Dados processados 45
Gráfico 8 – Ação da Petrobras antes da remoção de tendência 46
Gráfico 9 – Ação da Petrobras após da remoção de tendência 46
Gráfico 10 – Treinamento com *overfitting* 63
Gráfico 11 – Erro de validação durante treinamento 65
Gráfico 12 – Erro mínimo de treino e validação para diferentes arquiteturas 66
Gráfico 13 – Comparação de predições entre modelos com o ativo ABEV3 71
Gráfico 14 – Comparação de predições entre modelos com o ativo OIBR3 72
Gráfico 15 – Comparação de predições entre LSTM e regressores clássicos com o ativo ABEV3 74
Gráfico 16 – Comparação de predições entre LSTM e regressores clássicos com o ativo OIBR3 74
Gráfico 17 – Análise de operação do modelo preditor no ativo ABEV3 77
Gráfico 18 – Análise de operação do modelo composto no ativo ABEV3 78
Gráfico 19 – Análise de operação com cruzamento de médias móveis no ativo ABEV3 . 79
Gráfico 20 – Análise de operação com *Buy and Hold* no ativo ABEV3 79
Gráfico 21 – Função de regularização L^1 89
Gráfico 22 – Função de regularização L^2 89

LISTA DE QUADROS

Quadro 1 – Valor-p de séries temporais antes e depois do operador *Detrend* 47
Quadro 2 – Parâmetros dos testes durante o *backtesting* 54
Quadro 3 – Intervalo de parâmetros a serem testados durante *Backtesting* 56
Quadro 4 – Hiper-parâmetros usados no treinamento dos modelos 62
Quadro 5 – Abreviações dos modelos preditores utilizados 64
Quadro 6 – Erro mínimo de treino e validação para diferentes arquiteturas 66
Quadro 7 – Número de amostras de dados OHLC 67
Quadro 8 – Erro mínimo de treino e validação para diferentes janelas 67
Quadro 9 – Hiper-parâmetros do modelo preditor 68
Quadro 10 – Hiper-parâmetros dos modelos utilizados na construção do modelo composto 69
Quadro 11 – Erro mínimo de treino e validação entre modelo preditor e composto 70
Quadro 12 – Abreviações dos tratamentos utilizados 70
Quadro 13 – Erro MdAE para diferentes modelos e diferentes ações 72
Quadro 14 – Erro MSE para diferentes modelos e diferentes ações 73
Quadro 15 – Erro MdAE para diferentes modelos clássicos em ações 75
Quadro 16 – Erro MdAE para diferentes modelos clássicos em ações 75
Quadro 17 – Parâmetros otimizados da estratégia *Trail Stop Loss* 76
Quadro 18 – Diferença em lucro devido à taxa de corretagem 76

LISTA DE TABELAS

Tabela 1 – Lucro e número de operações em diferentes ações 80

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BPTT	*Backpropagation Through Time*
CNN	*Convolutional Neural Network*
DL	*Deep Learning*
GPU	*Graphics Processing Unit*
ML	*Machine Learning*
MCMC	*Markov Chain Monte Carlo*
NLP	*Natural Language Processing*
RNN	*Recurrent Neural Network*
SGD	*Stochastic Gradient Descent*
UFES	Universidade Federal do Espírito Santo

SUMÁRIO

1 INTRODUÇÃO 13

1.1 Apresentação e Objeto de Pesquisa 13

1.2 Trabalhos Relacionados 15

1.3 Objetivos 18

1.4 Estrutura do Texto 18

2 EMBASAMENTO TEÓRICO 20

2.1 Séries Temporais 20

2.2 Análise de Ativos no Mercado Financeiro 21

2.2.1 Dados OHLCV 22

2.3 Indicadores Técnicos no Mercado Financeiro 23

2.3.1 Indicadores de Momento 23

2.3.2 Indicadores de Tendência 24

2.3.3 Indicadores de Volatilidade 26

2.4 Computação Financeira e *Algorithmic Trading* 27

2.5 RNN 29

2.5.1 Tipos de arquiteturas de RNN 30

2.6 LSTM 31

2.6.1 Processamento na célula LSTM 32

2.6.2 Tipos de arquiteturas de LSTM 34

2.7 *Autoencoder* LSTM 37

2.8 Resumo 39

3 SOLUÇÃO PROPOSTA 40

3.1 Solução Proposta 40

3.1.1 Visão Geral 40

3.1.2 Pré-processamento 42

3.1.3 Construção dos Modelos 47

3.1.4 Etapa de treinamento 48

3.1.5 Etapa de inferência 49

3.1.6 *Backtesting* e *Algorithmic Trading* 50

3.2 Resumo 57

4 RESULTADOS 58
4.1 Recursos Computacionais 58
4.2 Detalhes de Implementação 61
4.3 Métricas 62
4.4 Experimentos 62
4.4.1 Hiper-parâmetros utilizados em treinamento 62
4.4.2 Experimentos Efetuados 63
4.4.3 Primeiro Experimento 65
4.4.4 Segundo Experimento 66
4.4.5 Terceiro Experimento 68
4.4.6 Quarto experimento 70
4.4.7 Comparação de resultados via *Backtesting* 75
4.5 Resumo 80
5 CONCLUSÕES 81
REFERÊNCIAS BIBLIOGRÁFICAS 83
APÊNDICE A – AUTOENCODER 87
ANEXO A – CÓDIGO 91

1 INTRODUÇÃO

1.1 Apresentação e Objeto de Pesquisa

A bolsa de valores é o lugar onde é feito a compra e venda de ações, títulos, *commodities*, índices, câmbio entre outros. Podendo assim, se dividir em: Mercado de capitais, de crédito, de câmbio e mercado monetário. O mercado brasileiro é composto pela bolsa de valores oficial chamada de B3, resultado da fusão entre a BM&FBovespa e a Cetip, que tem sua sede em São Paulo. Hoje, em um dia de negociação, o mercado financeiro brasileiro move mais de 10 bilhões de reais (B3, 2019). Com as variações temporais no preço de ativos, seja de valorização ou de desvalorização, investidores conseguem lucrar sobre a volatilidade do mercado.

Há grande interesse por parte de pesquisadores, economistas, investidores e afins de se modelar e prever o comportamento de ativos e adquirir informações de tendências do mercado. O motivo de usar técnicas e sistemas para auxiliar o processo de pôr ordens de compra e venda em ativos na bolsa de valores reside na dinamicidade do mercado financeiro, sendo sujeito a variações causadas por acontecimentos e noticias do país e do mundo, o mercado financeiro pode ser considerado um sistema randômico dependendo da escala de tempo em análise (MAHATA; PRASAD, 2019).

Investidores que operam no mercado financeiro utilizam estratégias para terem vantagens. Sejam estas estratégias: (*i*) modelos matemáticos como forma de análise técnica do comportamento do mercado, tentando adquirir informações sobre tendências ou de forças compradoras e vendedoras, assim como (*ii*) análise fundamentalista, onde o investidor utiliza informações e notícias vindas do cenário político e econômico do país e do mundo para especular o comportamento do mercado. As técnicas e modelos matemáticos podem possibilitar maior facilidade para que desejam se inserir no mercado financeiro uma vez que facilitam a forma de se atuar e analisar o mercado. Porém, um dos problemas em técnicas de análise financeira é o fato de não serem capazes de julgar peculiaridades de certos padrões ou até mesmo de, por serem generalizadas, prover interpretações erradas do mercado. Falta assim, um sistema de compra e venda de ações, denominado sistema de negociação, capaz de dar auxílio correto e eficiente, incapaz de tomar decisões precipitadas e que seja capaz de fazer previsões sobre o futuro do mercado.

Um sistema autônomo atuando como um robô de investimento, isto é, um algoritmo que gera ordens de compra e venda através de computadores, traz a capacidade de simular e visualizar modelos matemáticos e econômicos antes que se atue no mercado financeiro, podendo assim, julgar seu desempenho antes que haja investimento de capital e atuar de forma rápida e precisa quando necessário, tal processo é denominado de *backtesting*. Fora isso existem os benefícios de negociar nos melhores preços possíveis e ter um sistema que aplica gerenciamento de risco mais rápido que um humano devido à velocidade de operação de um computador.

O princípio de funcionamento de uma rede neural artificial se baseia em usar como arquétipo o cérebro humano com seus neurônios e suas sinapses. Assim, imitando uma estrutura biológica, cria-se um modelo matemático que herda características do cérebro humano, como sua habilidade de aprender.

Há três formas em que uma rede neural artificial aprende informações, via aprendizado supervisionado, não-supervisionado e por reforço. Em aprendizado supervisionado, parte-se de um conjunto de dados já conhecido, onde os dados estão correlacionados em pares de entrada e saída. No aprendizado não-supervisionado, há pouco conhecimento, ou nenhum conhecimento, sobre os resultados, desta forma, o objetivo é as relações entre as variáveis através das redes neurais. Já o aprendizado por reforço, busca treinar uma rede neural através de um sistema de recompensas e punições, não há entradas e nem saídas neste treinamento, a rede neural deve aprender a sobreviver em um ambiente através de ações não pré-definidas a fim de obter recompensas e não punições.

Redes neurais artificiais podem ser treinadas, através do aprendizado supervisionado, com dois objetivos distintos: Classificar e regredir. Isto é, redes neurais podem ser treinadas com o intuito de classificar um banco de dados em classes. Um exemplo seria criar um banco de dados com imagens de diferentes letras escritas por serem humanos onde as classes são as letras do alfabeto e o objetivo da rede neural é acertar quais imagens pertencem as suas devidas classes. Assim como podem ser treinadas com o intuito de observar padrões dentro de uma sequência. Um exemplo seria criar um banco de dados com variações de temperatura de uma sala ao longo do tempo e o objetivo da rede neural é prever qual será a próxima temperatura da sala. Neste trabalho será abordado o aprendizado supervisionado com o objetivo de regressão.

Uma rede neural treina acertando e errando predições feitas referentes a um banco de dados e a cada vez que faz isso, passa pelo processo de aprendizado supervisionado usando um processo de otimização onde a rede neural passa a convergir acertando mais que errando. Este processo é feito até que a rede não consiga mais aumentar a taxa de acertos.

Na atualidade, existe uma explosão em estudos em redes neurais e com isso inúmeras novas aplicações estão aparecendo. Detecção de câncer (JOUNI, 2016), detecção de padrões de fala (GRAVES; MOHAMED; HINTON, 2013), reconhecimento de escrita (PERWEJ; CHATURVEDI, 2011), reconhecimento facial (ER, 2002), redes que compõem música (CHEN; MIIKKULAINEN, 2001) e redes que conseguem jogar jogos (SILVER, 2017) são alguns dos exemplos da aplicação de redes neurais.

Entre as arquiteturas de redes neurais propostas ao longo dos anos, as mais comentadas nesse trabalho serão as redes neurais recorrentes (RNN, do inglês *Recurrent Neural Networks*). Uma de suas primeiras aplicações veio como a rede de Hopfield, proposta em Hopfield (1982), esta rede foi aplicada a problemas envolvendo séries temporais. Mais especificamente será comentada a rede recorrente de memória de longo prazo (LSTM, do inglês *Long Short-Term Memory*), proposta por Sepp Hochreiter e Jürgen Schmidhuber (HOCHREITER; SCHMIDHUBER, 1997), aplicada a mesma classe de problemas.

Para problemas onde os dados compõem séries temporais, redes neurais recorrentes são aplicadas com maior sucesso, uma vez que esse tipo de arquitetura possibilita a esse tipo de rede a capacidade de gerir dependências temporais e assim observar padrões.

1.2 Trabalhos Relacionados

Aqui, são descritos os trabalhos estudados relacionados ao problema abordado de gerar predições no mercado financeiro através de redes recorrentes. Todos os trabalhos buscam utilizar redes recorrentes com outras técnicas a fim de suplementar a capacidade da rede em gerar predições. Primeiro estão descritos os trabalhos que analisam o mercado financeiro brasileiro e, seguidamente, os trabalham que analisam outros mercados.

Em um dos trabalhos de redes neurais recorrentes para previsão de séries temporais financeiras o autor estuda a aplicação de redes LSTM ao problema de se gerar previsões em séries temporais (NELSON, 2017). Especificamente, é analisada a bolsa de valores de São Paulo a fim de se prever tendências de preços de ações. O método foi utilizar o histórico de preços de ações em conjunto com indicadores de análise técnica. No artigo foi relatado que o modelo LSTM foi capaz de prever a tendência do preço da ação no futuro imediato com mais de 55% de precisão. Também foi relatado que o modelo foi capaz de detectar variações agudas na tendência, o que gerou maiores retornos a curto prazo. Vale ressaltar que o artigo menciona que mesmo com a dimensionalidade da entrada sendo grande, a rede foi capaz de mostrar aprendizado satisfatório sobre os dados sem que fosse necessário usar técnicas de pré-processamento no banco de dados como a redução de dimensionalidade.

No trabalho *On stock return prediction with LSTM networks* (HANSSON, 2017) o autor faz uso de redes LSTM para gerar predições sobre índices do mercado financeiro. No caso, os índices escolhidos foram o S&P 500 dos Estados Unidos, o Bovespa 50 do Brasil e o OMX 30 da Suécia. Em termos de regressão, a acurácia da rede neural se assemelhou à acurácia de modelos convencionais de análise de séries temporais, no caso, um modelo ARMA GRJ-GARCH (modelo de heteroscedasticidade condicional auto-regressiva generalizada de Glosten–Jagannathan–Runkle). Porém em termos de classificar a mudança em direção de tendência, a rede LSTM superou tais modelos convencionais. Como adendo, foi mencionado que a maior dificuldade em gerar boas predições nos mercados do Estados Unidos e Brasil podem ser indícios de um mercado financeiro que utiliza em maior quantidade dados históricos para se gerar decisões de compra e venda.

No trabalho *Time Series Prediction: Predicting Stock Price* (ELLIOT, 2017) os autores apresentam quatro modelos utilizados para a predição de preços futuros. O banco de dados utilizado foi o índice S&P 500 como série temporal. O artigo relata sobre a necessidade que métodos lineares convencionais de predição em séries temporais sofrem por terem que partir do pressuposto que as séries temporais são estacionárias e demostra que, em termos de testes empíricos, modelos RNN tem maior acurácia em predição, sem ter que partirem de tais pressupostos. O artigo segue em mencionar algumas estratégias que podem ser utilizadas nos algoritmos para se efetuar compra e venda no mercado e como redes neurais podem usar tais estratégias com certo grau de confiabilidade. Vale ressaltar que, assim como citado no artigo, foi utilizado apenas dados

de entrada de uma dimensão, o preço de fechamento do ativo, e que a rede neural LSTM não falhou em criar relações entre os dados, apenas aprendeu a repetir a última informação no tempo. Desta forma, evidencia-se o impacto da caracterização da rede e dos dados relacionados e que possíveis avanços ao estado da arte de modelos LSTM ou de técnicas relacionadas podem levar a melhorias nos resultados.

Por fim, no trabalho *Application of Neural Network for Forecasting of Exchange Rates and Forex Trading* (MAKNICKIENE, 2012) os autores utilizaram diferentes arquiteturas de redes neurais através do método de Delphi onde a média da decisão de todos supera a decisão de apenas um (DALKEY; HELMER, 1963). A fim de se criar um algoritmo que gera predições no mercado financeiro. Foi utilizado como banco de dados, os dados históricos de *Foreign Exchange* (mercado de câmbio) e uma investigação estatística dos resultados mostrou aumento na performance da predição. Os modelos utilizados foram RNN com arquiteturas diferentes e com algoritmos de aprendizagem diferentes. Nos resultados, o estudo demonstra que RNN foram capazes de gerar até 4% de lucro nos períodos de teste no par cambial euro-dólar(EUR/USD), fora ter comprovado que a abordagem via método de Delphi obteve maior eficácia no processo de gerar predições.

Em geral, os trabalhos citados usam diversas técnicas dentro do escopo de redes neurais recorrentes para atacar o problema de predições no mercado financeiro. Diferentemente, a proposta deste trabalho buscou criar uma estrutura de testes que permita rápida prototipagem de modelos através de um sistema que simplifica a criação, testes e validez dos modelos. Além disso, a proposta buscou testar a eficiência de cada modelo contra uma situação real onde o modelo não deve apenas gerar lucro, mas gerar mais lucro que outras estratégias já usadas no mercado sobre parâmetros reais de comissão e *spread*, isto é, quando uma ordem de compra ou venda não consegue ser executada no preço determinado devido à flutuação do mercado. Para tanto, foram utilizadas redes recorrentes LSTM para gerar predições a fim de serem utilizadas por uma estratégia de *algorithmic trading*. Além disto, a estrutura criada foi capaz de expor estas predições para plataformas de negociação para que seja possível obter lucro de forma automatizada.

1.3 Objetivos

O objetivo geral deste trabalho foi o desenvolvimento de um sistema atuando como um robô de investimento que, através de um modelo de rede neural recorrente do tipo LSTM, obtém previsões dos valores futuros de ativos para se encontrar pontos de compra e venda de forma automatizada. O sistema é capaz de expor estas predições à plataformas de operação na bolsa de valores e testar sua rentabilidade sem que haja capital investido.

Como objetivos específicos tem-se as etapas de: *(i)* revisão bibliográfica de trabalhos que estudam o problema de *Algorithmic Trading* com redes neurais recorrentes e escolha de banco de dados com várias séries temporais de diferentes ações da bolsa de valores usadas durante treinamento, *(ii)* análise de diferentes modelos de redes neurais recorrentes LSTM aplicadas no problema de regressão em séries temporais e avaliação do desempenho dos modelos analisados, *(iii)* desenvolvimento do sistema de robô de investimento que utiliza o modelo de rede neural proposto para gerar sinais de compra e venda e teste do sistema nos bancos de dados escolhidos, *(iv)* avaliação do desempenho do sistema proposto, comparando com outras estratégias e por fim, a documentação de todo o estudo feito assim como a solução proposta e seu resultado.

1.4 Estrutura do Texto

O presente trabalho está estruturado de tal forma que inicia-se com a introdução, cujo objetivo é informar ao leitor sobre o problema e suas as características, assim como mencionar trabalhos já realizados sobre o assunto, justificar o motivo de se abordar este problema e como o mesmo será abordado. Posteriormente, no segundo capítulo, serão apresentados os referenciais teóricos, onde, apresenta-se os assuntos tratados no trabalho e a base teórica das ferramentas utilizadas, tais como: séries temporais, mercado financeiro, indicadores técnicas, redes RNN e LSTM.

Em seguida, tem-se um terceiro capítulo dedicado a proposta de abordagem do problema, onde é apresentada a solução detalhado ao problema exposto e, em sequência, o quarto capítulo dedicado aos resultados obtidos, onde são expostos os recursos utilizados no desenvolvimento deste projeto, os experimentos efetuados, as métricas e os resultados obtidos. Por fim, dedica-se

o quinto capítulo para a exposição das conclusões do trabalho, onde a eficiência da solução proposta é comparada a outras soluções propostas em outros trabalhos, assim como possíveis melhorias.

2 EMBASAMENTO TEÓRICO

Este capítulo tem por finalidade estabelecer os conceitos teóricos necessários usados no trabalho. Portanto, o capítulo inicia com com uma explicação conceitual sobre o que são séries temporais. Em seguida, são explicadas características do mercado financeiro, ferramentas de análise e possíveis estratégias utilizadas no mercado. Por fim, são explicadas redes neurais recorrentes, seus pontos chaves e o porquê da escolha de utiliza-las.

2.1 Séries Temporais

Séries Temporais são sequências de dados que tem relações temporais entre os pontos observados. A característica fundamental de uma série temporal é que os pontos observados adjacentes são necessariamente dependentes (OLIVEIRA, 2012). O Gráfico 1 apresenta um exemplo de uma série temporal composta pelos valores das ações da empresa *Google* ao longo dos anos. Em azul tem-se os valores de fechamento das ações e abaixo, tem-se em vermelho e verde um histograma correspondendo ao volume de ações negociadas no tempo.

Gráfico 1 – Exemplo de uma série temporal

Fonte: Yahoo (2019b).

As séries temporais aparecem em vários problemas, como: (*i*) demanda elétrica de uma cidade, (*ii*) variação da temperatura de uma cidade, (*iii*) preço de ações na bolsa de valores e etc. Onde são classificadas como: (*i*) contínuas ou discretas onde fenômenos contínuos são eventos onde

há infinitas observações de valores entre dois valores numéricas quaisquer enquanto fenômenos discretos tem um número finito de valores, uma variável discreta é sempre numérica. (*ii*) determinísticas ou estocásticas onde séries determinísticas tem seus valores escritos por funções matemáticas, a lei de formação de séries determinísticas é da forma $y = f(tempo)$. Enquanto séries estocásticas são descritas como um função matemática dependente do tempo com uma componente aleatória associada a essa função, por exemplo, $y = f(tempo, \delta)$, onde a componente aleatória é δ. Ainda sobre séries estocásticas, existe uma sub-divisão em series estacionárias e não-estacionárias. Séries estacionárias são séries que apresentam valores estatísticos constantes para cada realização da série. Na prática, a maioria das séries presentes na natureza são não-estacionárias, isto é, seus valores estatísticos variam para cada realização da serie. A proposta de se estudar séries estacionárias esta relacionada à análise estatística, uma vez que existe uma necessidade de se simplificar certas séries para poder compreendê-las (OLIVEIRA, 2012). Há vários modelos para se analisar séries temporais utilizando métodos lineares e não-lineares. Uma abordagem baseado em modelos não lineares é o uso de redes neurais.

2.2 Análise de Ativos no Mercado Financeiro

A análise de ativos no mercado financeiro pode ser divida em estilos de implementação. Estas implementações vão de análises puramente fundamentalistas assim como puramente técnicas. Isto é, na análise fundamentalista, opera-se no mercado financeiro através de índices de empresas, métricas como lucro líquido de uma empresa ao ano, número de funcionários, atividade econômica no setor em que atua, assim como notícias e acontecimentos do país e do mundo. Já a análise técnica gráfica, busca observar padrões na flutuação do preço de ativos a fim de operar comprando e vendendo ativos. Ao se analisar o mercado financeiro deve-se estar ciente de como cada mercado se comporta, isto é, diferentes lugares, tem diferentes características de flutuação, horário de pico, volume negociado, quantidade de ativos assim como estar ciente de características de corretoras como a comissão à ser paga por negociação, além de características que variam com tempo como o *slippage* e o *spread*.

O *slippage* ocorre quando é executada uma ordem, seja de compra ou venda, no mercado financeiro, mas o preço efetivo não é o mesmo do posto na ordem. Isto pode ocorrer em

qualquer momento, porém tal comportamento é acentuado devido a baixa liquidez de ativos ou quando o investidor opera, seja compra ou venda, em grandes volumes e não há compradores ou vendedoras disponíveis para negociar o montante desejado, desta forma, a ordem não é executada imediatamente, tendo que ser executada em preço diferente possivelmente com outro valor. Isto é, o preço e quantidade de ações que o investidor deseja movimentar, não é compatível com o mercado. A diferença em preço entre a força vendedora e compradora é chamada de *spread*.

2.2.1 Dados OHLCV

Os dados históricos obtidos para negociação de ativos no mercado financeiro são dados por cinco séries temporais as quais compõem o OHLCV, isto é, *Open-High-Low-Close-Volume* - OHLCV, desta forma, os dados são formados por: preço de abertura, preço máximo, preço mínimo, preço de fechamento e volume de ações negociadas. Entretanto, pode ocorrer que os dados de volume não estejam presentes, nesta situação, tem-se o OHLC. No Gráfico 2, são mostrados dados históricos OHLC do ativo CMIG3 da empresa Companhia Energética de Minas Gerais expostos em forma de barras e o volume exposto através do histograma verde. Cada barra do gráfico denominada “vela” representa os preços OHLC para um dado momento no tempo.

Gráfico 2 – Exemplo de velas OHLC

Fonte: Produção do próprio autor.

2.3 Indicadores Técnicos no Mercado Financeiro

Ao se estudar o comportamento da flutuação de um ativo no mercado financeiro, busca-se obter ferramentas gráficas que tentem expressar características do mercado através de funções matemáticas.

Tais funções matemáticas usadas em análise gráfica compõem os chamados indicadores técnicos, que, através do preço de abertura, fechamento, preço máximo e preço mínimo, extraem informações de momento, tendência e volatilidade. Indicadores técnicos atuam como operadores de séries temporais, com o propósito de sintetizar ou extrair informações específicas. Um dos indicadores de maior uso é o média móvel simples (SMA, do inglês *Simple Moving Average*) com sua definição dada pela equação (2.1), onde, t é o período em análise e preço_{t+T} é o preço de fechamento em dado momento $t+T$.

$$SMA_t(\text{preço}) = \frac{\text{preço}_T + \cdots + \text{preço}_{t+T}}{t} \tag{2.1}$$

2.3.1 Indicadores de Momento

Tem o objetivo de fornecer momentos apropriados para o investidor se posicionar no mercado, isto é, fornecer momentos ótimos de compra e venda de ativos na bolsa de valores a fim de maximizar o retorno. A ferramenta indica que, caso esteja com um valor perto do seu máximo, então, força compradora é excessiva, isto é, o movimento pode estar prestes a mudar seu momento uma vez que, agora com muitos compradores em relação à vendedores, o preço tende a diminuir. No momento oposto, isto é, quando o indicador estiver perto do seu valor mínimo, tem-se então força vendedora excessiva podendo levar a subida do preço. Dentre eles, os mais comumente utilizados são o taxa de variação (*ROC*, do inglês *Rate of Change*) dado pela equação (2.2) e o $Williams\%R$ dado pela equação (2.3).

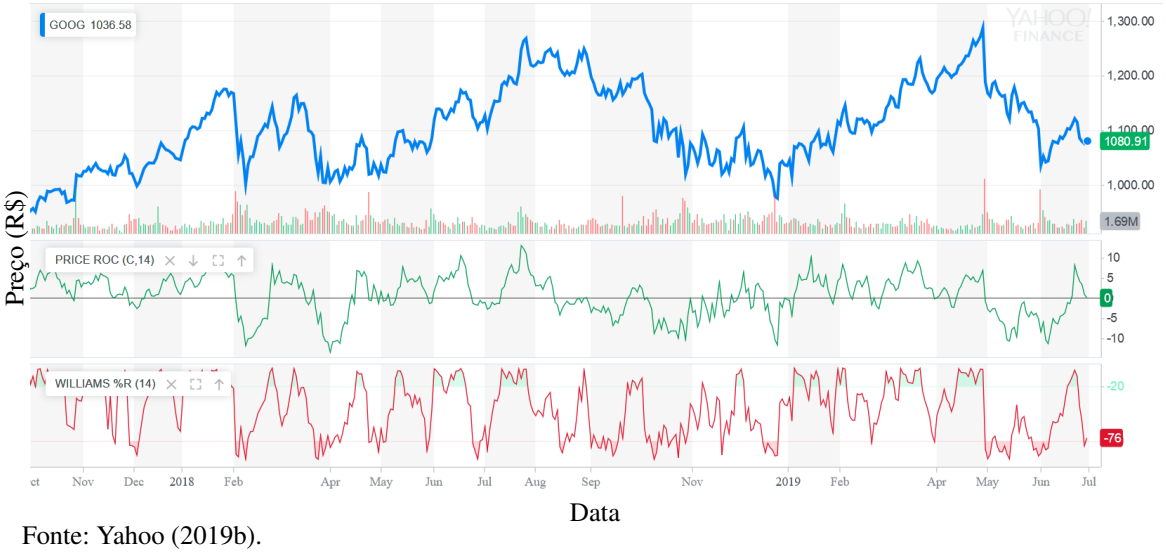
$$ROC = \left(\frac{\text{preço}_t}{\text{preço}_{t-n}} - 1\right) \times 100, \tag{2.2}$$

$$William\%R_t = \left(\frac{\text{preço}_{\text{máximo}_n} - \text{preço}_{\text{fechamento}_t}}{\text{preço}_{\text{máximo}_n} - \text{preço}_{\text{mínimo}_n}}\right) \times 100. \tag{2.3}$$

Onde: t é o período em análise n é o período de calculo do indicador; preço-máximo$_n$ é valor máximo nos últimos n períodos; preço-mínimo$_n$ é valor mínimo nos últimos n períodos; preço-fechamento$_t$ é o valor de fechamento para o tempo t.

No Gráfico 3 é demonstrado um exemplo dos indicadores em relação ao ativo GOOG da empresa Google. O valor do preço de fechamento do ativo é dado pela linha azul, já as linhas verde e vermelho, são os indicadores *ROC* e *William%R*, respectivamente.

Gráfico 3 – Valores de fechamento com indicadores de momento



Fonte: Yahoo (2019b).

2.3.2 Indicadores de Tendência

Tem como objetivo mostrar a direção do mercado, isto é, se existe tendência de subida ou descida, provendo assim, uma direção geral do mercado. Dentre os indicadores de tendência, o mais comumente utilizado é o Convergência/Divergência de Média Móvel (MACD, do inglês *Moving Average Convergence/Divergence*). Nota-se que n na equação (2.4) se refere ao período de calculo do indicador.

$$EMA_t = k \times \text{preço}_t + (1-k) \times EMA_{t-1}, \text{ onde } k = \frac{n+1}{2}, \tag{2.4}$$

$$Sinal_{MACD} = EMA_{curta} - EMA_{longa}, \tag{2.5}$$

$$Sinal_{\text{médio}} = EMA_{\text{médio}}, \tag{2.6}$$

$$Histograma_{MACD} = Sinal_{MACD} - Sinal_{\text{médio}} \tag{2.7}$$

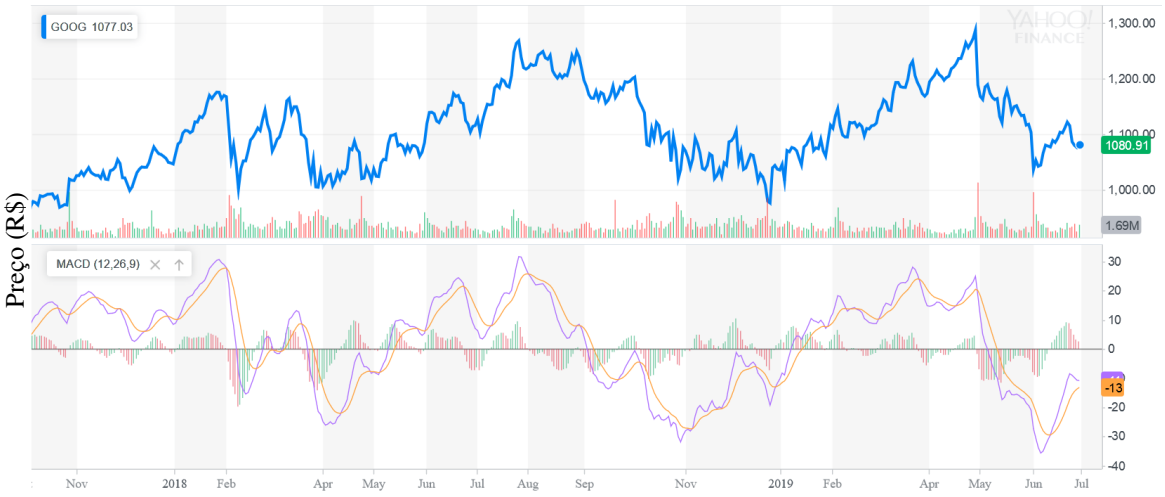
Neste indicador, é obtido, a partir de duas médias móveis com períodos distintos, um sinal e um histograma. O sinal MACD é obtido através da diferença entre as duas médias móveis, como demonstra a equação (2.5), somado a isto, utiliza-se outra média móvel com um período entre os períodos utilizados no calculado do sinal médio, equação (2.6). A média móvel clássica na utilização do MACD é a média móvel exponencial (EMA, do inglês *Exponential Moving Average*) dada pela equação (2.4).

A análise deste indicador é feita através do comportamento de: convergência, divergência e cruzamento entre o $Sinal_{MACD}$ e $Sinal_{\text{médio}}$, assim como o cruzamento destes sinais com o eixo zero, isto é, o momento em que um sinal passa de uma valor positivo a um valor negativo ou vice-versa.

O histograma é uma forma gráfica de mensurar o comportamento de divergência e convergência entre os sinais, facilitando a visualização da diferença dos sinais ao longo do tempo. O cruzamento de sinais tende a ser um momento de interesse, pois o mesmo significa que ocorreu uma variação na tendência. Caso os sinais se cruzem e o $sinal_{MACD}$ cruze o eixo tal que o sinal passe de um valor positivo a um valor negativo, ou vice-versa, tem-se uma uma tendência à valorização, onde, quanto maior a diferença entre os sinais, maior a velocidade em que esta tendência esta se formando, tal diferença é visualizada através do histograma. Caso o $sinal_{MACD}$ cruze o eixo de um valor positivo para um valor negativo, tem-se uma tendência à desvalorização do preço do ativo.

No Gráfico 4 é demonstrado um exemplo dos indicadores em relação ao ativo GOOG da empresa Google. O valor do preço de fechamento do ativo é dado pela linha azul, já as linhas amarela e roxa são os sinais MACD e o sinal médio, respectivamente.

Gráfico 4 – Valores de fechamento com indicador técnico MACD



Fonte: Yahoo (2019b).

2.3.3 Indicadores de Volatilidade

Estes indicadores tentam observar comportamentos de volatilidade de um ativo, analisando quanto o ativo variou em dado momento em relação ao passado, a fim de determinar se o comportamento esta fora do padrão visto no passado.

Os indicadores de volatilidade mais comumente utilizados são os indicadores Bandas de Bollinger e o alcance verdadeiro médio (ATR, do inglês *Average True Range*).

O indicador Banda de *Bollinger* gera três sinais, a banda média, superior e inferior. Dadas pelas equações (2.8), (2.9), (2.10), onde $\mathbb{STD}$ é o operador de desvio padrão.

$$\text{Bollinger}_{\text{Média}} = SMA_t(\text{preço-fechamento}) \tag{2.8}$$

$$\text{Bollinger}_{\text{Superior}} = \text{Banda}_{\text{Média}} + 2\mathbb{STD} \times (\text{Banda}_{\text{Média}}) \tag{2.9}$$

$$\text{Bollinger}_{\text{Inferior}} = \text{Banda}_{\text{Média}} - 2\mathbb{STD} \times (\text{Banda}_{\text{Média}}) \tag{2.10}$$

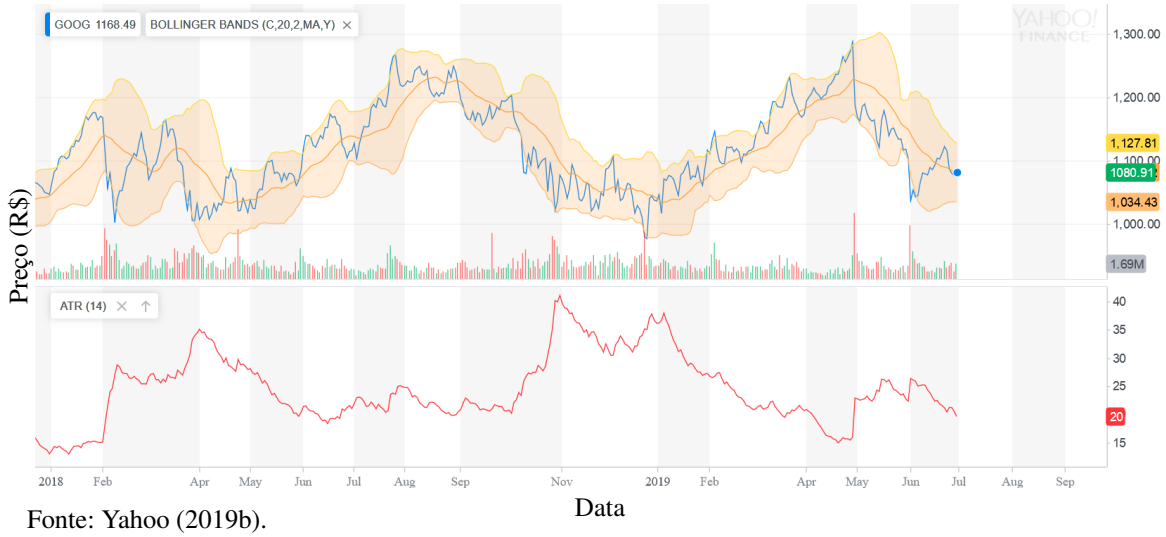
O indicador ATR é dado pela equação (2.12), onde t é o período de análise e ATR_t é o maior valor entre: a diferença em valor absoluto do preço máximo atual e de fechamento anterior, a diferença em valor absoluto preço mínimo atual e fechamento anterior, e a diferença entre o preço máximo atual e o mínimo atual.

$$ATR_t = \max \left\{ \begin{bmatrix} \text{preço-máximo}_t - \text{preço-mínimo}_t \\ |\text{preço-máximo}_t - \text{preço-fechamento}_{t-1}| \\ |\text{preço-mínimo}_t - \text{preço-fechamento}_{t-1}| \end{bmatrix} \right\} \tag{2.11}$$

$$ATR = SMA_t(ATR_t) \tag{2.12}$$

No Gráfico 5 é demonstrado um exemplo dos indicadores em relação ao ativo GOOG da empresa Google. O valor do preço de fechamento do ativo é dado pela linha azul, já a zona amarela é a área delimitada pelas bandas de Bollinger a linha vermelha é o ATR.

Gráfico 5 – Valores de fechamento com indicador técnico ATR



Fonte: Yahoo (2019b).

2.4 Computação Financeira e *Algorithmic Trading*

Computação financeira é uma extensão de ciência da computação que envolve assuntos relacionados ao mercado financeiro. Assim como a matemática, a utilização de métodos computacionais na economia traz a capacidade da simulação e visualização de modelos matemáticos e econômicos.

Investidores ou instituições que contam com sistemas computacionais e bancos de dados sobre o mercado financeiro, conseguem tirar proveito de modelos matemáticos probabilísticos utilizando

algoritmos para extrair informações úteis sobre o mercado sem ter que passar pelas tecnicalidades de se fazer cada cálculo. Tarefa que um sistema computacional faz rapidamente (MARKOWITZ, 1991).

Há vários nichos de conhecimento que tiram vantagem disso, como melhores formas de administrar créditos, avaliar propostas financeiras ou automação de compra e venda em preços de ações.

Robô de investimento (do inglês, *Algorithmic trading*) é uma forma de automatizar o processo de compra e venda no mercado financeiro. Isto é, criar um algorítimo que através de uma estratégia e regras do tipo *if - then* sejam feitas as ordens de compra e venda. A vantagem de se usar este tipo de método reside no fato de que por serem sistemas computacionais, operam em uma velocidade maior que a de seres humanos e de forma automatizada. Tornando o sistema capaz de gerar lucros de forma rápida e independente de interação humana, contando que uma estratégia eficiente seja utilizada.

Um exemplo que pode ser dado é, criar um sistema onde um computador análise médias móveis de uma ação, constantemente procurando por pontos onde há criação de uma tendência de compra. E com o tempo de análise mais rápido que a de seres humanos, comprar as ações no começo da tendência, aumentando assim, os lucros.

Através da capacidade de simular cenários com bancos de dado, que advém da computação financeira, um sistema de *Algorithmic trading* pode executar *backtesting*. O *backtesting* é o procedimento de teste de estratégia. Com a possibilidade de gerar e simular cenários reais, antes de ingressar no mercado financeiro com capital investido. O *backtesting* possibilita ao usuário simular, sobre os dados reais do histórico de preço de quantas ações julgar necessário, o rendimento e a eficiência da estratégia. Com as simulações feitas, pode-se ainda gerar relatórios que definem níveis de risco, lucro percentual e quando se fez lucro ou quando se fez prejuízo.

2.5 RNN

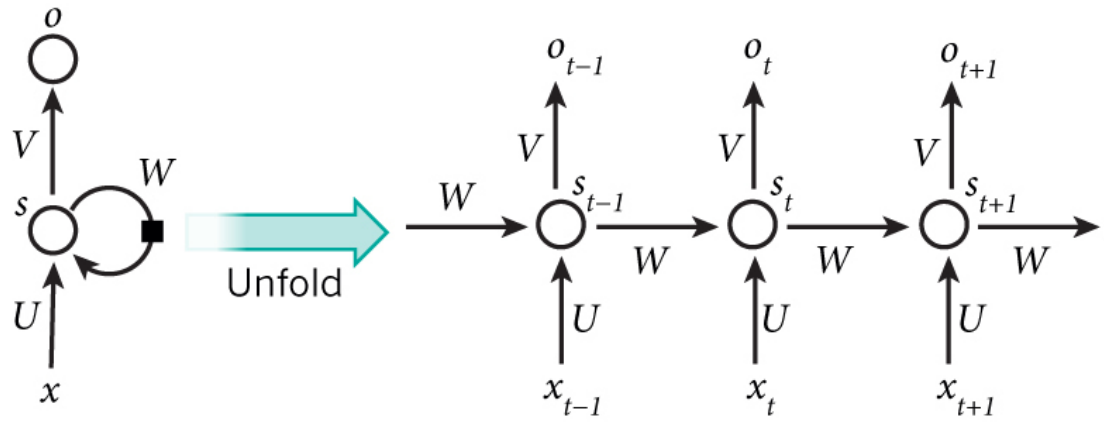
Uma rede RNN tem como característica o uso de nós recorrentes em sua estrutura, isto é, a saída da célula é re-alimentada como uma entrada dela mesma. Como apresentado na Figura 1, onde na esquerda tem-se representação de uma célula RNN e na direita seu uso para entradas com dependências temporais mostrando como são esquematizadas suas conexões, uma relação recorrente básica definida por uma RNN é dada pelas equações (2.13) e (2.14):

$$s_t = f(Ux_t + Ws_{t-1}) \tag{2.13}$$

$$o_t = Vs_t. \tag{2.14}$$

Onde, x é a entrada, p é a saída, s é o estado oculto e U, V e W são os pesos associados a estrutura da rede, e f é uma função não linear para a ativação da saída[1]. Nota-se, pela equação (2.14), que para redes RNN, a saída da célula no tempo t é dado pela entrada no tempo t e seu estado no tempo $t-1$. Desta forma, associa-se a informações a serem processadas no tempo já que durante o processo de atualização dos parâmetros via treinamento supervisionado, o gradiente do estado $t-1$ é usado para atualizar o parâmetro no tempo t.

Figura 1 – Modelo de uma RNN



Fonte: LeCun, Bengio e Hinton (2015).

[1] Usualmente, utiliza-se como função de ativação não linear as funções tangente hiperbólica e a *rectified linear unit* - ReLU

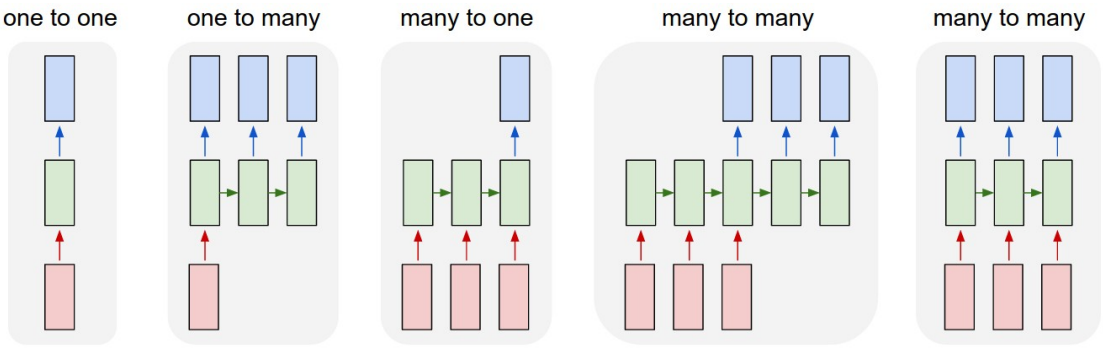
A habilidade de uma RNN em criar dependências temporais reside na propagação e influência destes gradientes ao longo da rede assim como mencionado acima. Entretanto, existem limitações a essas relações temporais a medida que o gradiente se propaga, ele tende a ficar com valores próximos de zero, deixando de influenciar a rede e por sua vez, perdendo longas relações temporais entre as entradas. Isto é conhecido como o Problema do desvanecimento do gradiente (PVG, do inglês *Vanishing Gradient*) (HU, 2018).

2.5.1 Tipos de arquiteturas de RNN

Como demostra a Figura 2, onde os retângulos vermelhos representam as entradas da rede, os retângulos verdes os estados ocultos da rede e os retângulos azuis as saídas da rede, a literatura define quatro tipos arquiteturas RNN, listadas a seguir: (*i*) muitos para muitos (do inglês, *Many to Many*), (*ii*) muitos para um (do inglês, *Many to One*), (*iii*) um para muitos (do inglês, *One to Many*), e (*iiii*) um para um (do inglês, *One to One*).

Como o nome sugere, tais arquiteturas formalizam como os dados estão correlacionados, isto é, caso queira obter uma sequência de dados no tempo na saída, deve ser utilizar uma arquitetura "para muitos", ou seja, processa-se a entrada de tal forma que obtenha uma sequência de saída. Seguindo a lógica, caso queira-se apenas um valor a partir de uma sequência, utiliza-se uma arquitetura "para um".

Figura 2 – Tipos arquiteturas RNN



Fonte: Olah (2015).

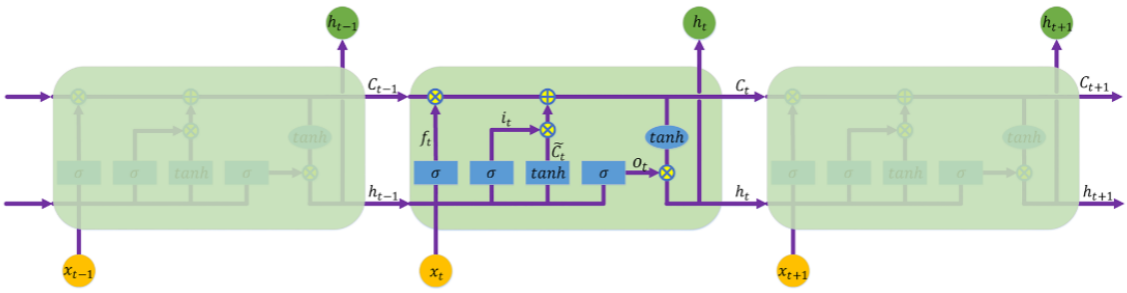
2.6 LSTM

Redes LSTM são um tipo de RNN com modificações na estrutura de suas camadas. Há uma diferença na estrutura da saída em relação a saída de redes recorrentes mostradas pelas equações (2.13) e (2.14). Em uma rede neural LSTM, busca-se propagar a entrada no tempo t assim como o estado da rede no passo anterior através do uso de estruturas denominadas portões.

As modificações buscam contornar o PVG, ou seja, criar um modelo que define relações temporais mais distantes entre as entradas. Não há modificações no conceito de redes recorrentes, isto é, a rede continua a trabalhar com o estado s_t no tempo t influenciando o estado s_{t+1} no tempo $t+1$ e assim sucessivamente. Não há mudança na lógica por trás do funcionamento, apenas é aumentada a robustez da estrutura de cada célula.

Como apresentado na Figura 3, onde é demonstrado uma rede LSTM desdobrada no tempo para 3 entradas com suas conexões esquematizadas demonstrando o fluxo de informação na célula, o diferencial reside na estrutura interna da célula.

Figura 3 – Rede LSTM desdobrada no tempo



Fonte: Bao, Yue e Rao (2017).

A mesma é composta por portões que trabalham como comportas de dados. Tais estruturas de controle trabalham de forma cooperativa com o objetivo de controlar o fluxo de informação, operando como uma memória, definindo o que a rede deve guardar e o que deve esquecer ao longo do tempo. Especificamente, Uma célula LSTM contém três tipos de portões de controle: (*i*) portão para decidir qual informação não é relevante, o portão de esquecimento(do inglês, *Forget Gate*); (*ii*) portão para definir quais entradas serão utilizadas para atualizar as células

de memória, o portão de entrada (do inglês, *Input Gate*); (*iii*) portão que decide quais serão as saídas da célula, levando em conta qual a entrada e qual o estado da célula de memória, o portão de saída (do inglês, *Output Gate*) (BROWNLEE, 2017).

2.6.1 Processamento na célula LSTM

As equações que definem a estrutura interna de uma célula LSTM são mostradas a seguir:

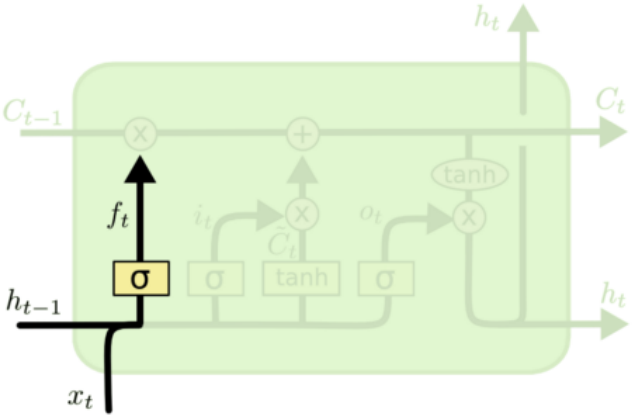
$$
\begin{aligned}
i_t &= \sigma(\mathbf{W}_{ii}x_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}h_{t-1} + \mathbf{b}_{hi}), && (2.15)\\
f_t &= \sigma(\mathbf{W}_{if}x_t + \mathbf{b}_{if} + \mathbf{W}_{hf}h_{t-1} + \mathbf{b}_{hf}, && (2.16)\\
o_t &= \sigma(\mathbf{W}_{io}x_t + \mathbf{b}_{io} + \mathbf{W}_{ho}h_{t-1} + \mathbf{b}_{ho}), && (2.17)\\
\widetilde{C}_t &= \tanh(\mathbf{W}_{ig}x_t + \mathbf{b}_{ig} + \mathbf{W}_{hg}h_{t-1} + \mathbf{b}_{hg}), && (2.18)\\
C_t &= f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t, && (2.19)\\
h_t &= o_t \odot \tanh(C_t). && (2.20)
\end{aligned}
$$

Onde: $\odot$ é o operador do produto Hadamard; *tanh* é a função tangente hiperbólica; σ é a função sigmoide; h_t, x_t e C_t são a saída, a entrada e o o estado da célula LSTM no instante t, respectivamente; i_t, f_t e o_t são as saídas do *Input Gate*, *Forget Gate* e do *Output Gate*, respectivamente.

A Figura 4 mostra o efeito da equação (2.16), que representa o *Forget Gate*. Aqui, f_t permite controlar quanto da informação do estado no tempo anterior, C_{t-1}, influencia o estado atual C_t[2].

[2] Nota-se que a função sigmoide retorna um valor no intervalo de $[0,1]$, que define quanto do estado da célula no tempo passado deseja-se usar.

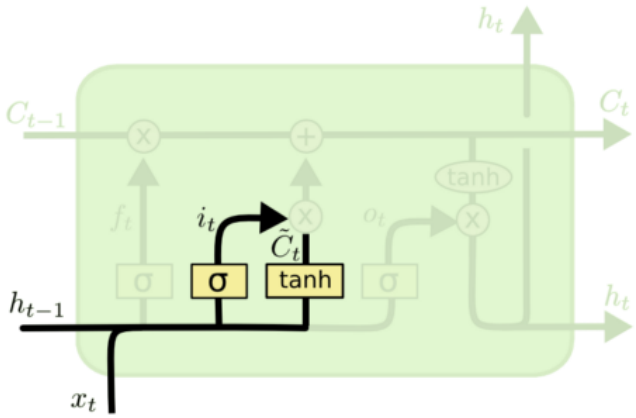
Figura 4 – Fluxo de informação referente ao *Forget Gate*



Fonte: Olah (2015).

A Figura 5 mostra o efeito das equações (2.15) e (2.18), que representam o *Input Gate* e o *buffer* de estados, respectivamente. Aqui, i_t e $\widetilde{C}_t$ permitem adicionar novas informações na atualização do estado atual C_t.

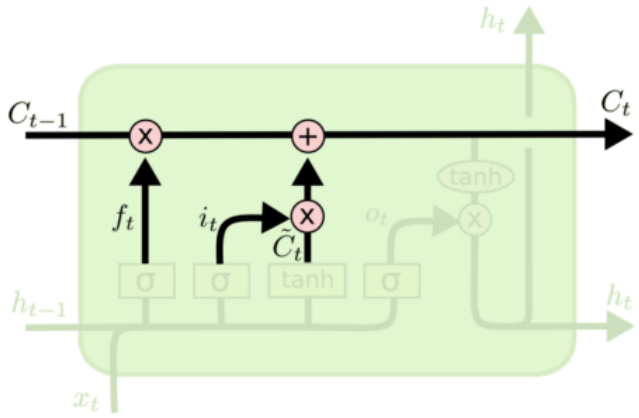
Figura 5 – Fluxo de informação referente ao *Input Gate*



Fonte: Olah (2015).

A Figura 6 mostra o efeito da equação (2.19), que representam a atualização do estado da célula. De forma intuitiva pode-se dizer que: (*i*) o termo $f_t \odot C_{t-1}$ representa a informação que deseja-se manter (o porcentagem de C_{t-1} que passa para C_t). (*ii*) o termo $i_t \odot \widetilde{C}_t$ representa a informação que deseja-se atualizar (a porcentagem de $\widetilde{C}_t$ que passa para C_t).

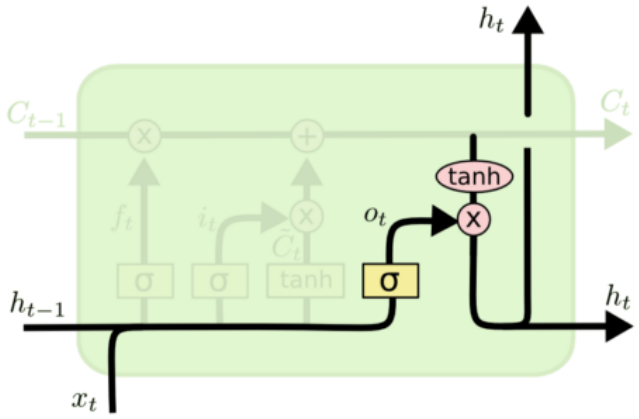
Figura 6 – Atualização do estado da célula



Fonte: Olah (2015).

A Figura 7 mostra o efeito da equação (2.20), que representam o *Output Gate*. O valor de o_t atua como um filtro que pondera quanto da informação do estado atual passado pela função não linear tangente hiperbólica será usado para determinar a saída da célula h_t (ver equação (2.20)).

Figura 7 – Fluxo de informação referente ao *Output Gate*



Fonte: Olah (2015).

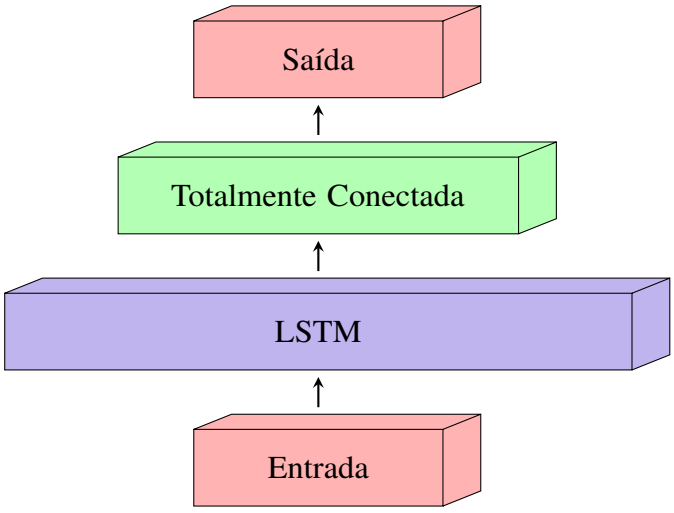
2.6.2 Tipos de arquiteturas de LSTM

A camada LSTM pode ser utilizada com outras camadas com o objetivo de aumentar a robustez do modelo. Com isso, é possível construir modelos LSTM com camadas totalmente conectadas (KARIM, 2017), camadas convolucionais (SAINATH, 2015), camadas de *Embedding* (WANG, 2016) e etc. Segue os detalhes das diferentes arquiteturas utilizadas neste trabalho.

A rede LSTM com camada totalmente conectada é a primeira arquitetura LSTM, proposta em 1997, fornece bons resultados em problemas de predição de sequências. Como demostrado na Figura 8, é uma rede neural LSTM com uma camada totalmente conectada, (BROWNLEE, 2017). No diagrama, as setas entre blocos indicam o fluxo de informação dentro da rede neural. Os blocos azuis são uma representação de redes LSTM desdobradas do tempo, já o bloco verde representa a camada totalmente conectada.

O objetivo da camada totalmente conectada é transformar uma LSTM *Many-to-Many* para uma LSTM *Many-to-One*. Como é mostrado na Figura 2, em uma rede LSTM *Many-to-Many*, para cada sequência de entrada é gerada uma sequência de saída, que ao ser aplicada a uma camada totalmente conectada é transformada em uma única saída. Com isso, cria-se uma saída onde a rede neural leva em consideração toda a informação aprendida no tempo.

Figura 8 – Diagrama de redes LSTM totalmente conectadas

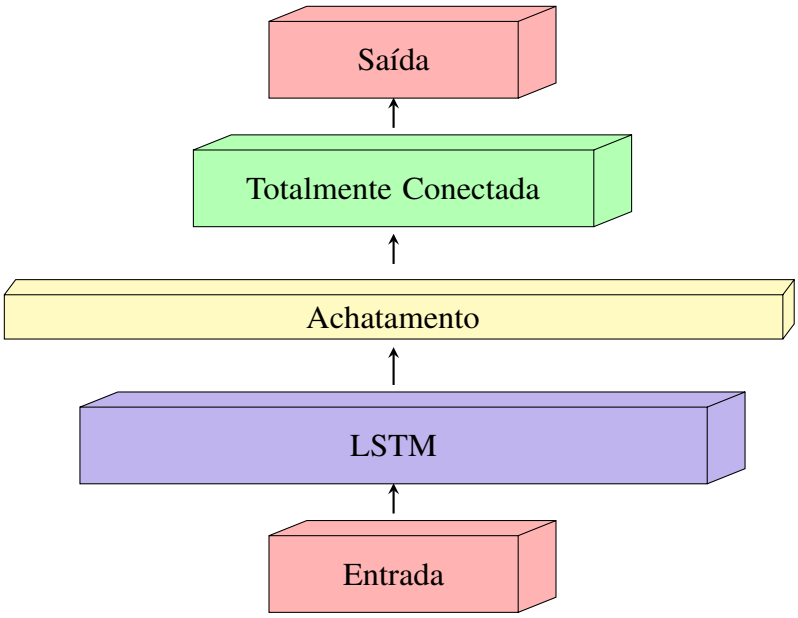


Fonte: Produção do próprio autor.

A rede LSTM com camada totalmente conectada achatada é uma variante da LSTM com camada totalmente conectada. A arquitetura não só utiliza o conjunto das saídas no tempo, como utiliza as características de todas as saídas no tempo para compor o conjunto de dados a ser totalmente conectado. É demonstrado pela Figura 9 este variante, onde, as setas entre blocos indicam o fluxo de informação dentro da rede neural, os blocos azuis são uma representação de redes LSTM desdobradas do tempo, o bloco verde representa a camada totalmente conectada e o bloco amarelo representa a operação de achatamento.

Executa-se o achatamento dos dados em uma dimensão e conecta-se estes dados achatados, desta forma, a saída gerada é o resultado de toda a informação que circula pelo modelo. Nota-se que esta arquitetura, assim como a vista a rede LSTM com camada totalmente conectada, visa a forçar o otimizador durante o treinamento da rede neural à levar em conta toda informação passada a rede, não só temporalmente, mas também espacialmente.

Figura 9 – Diagrama de redes LSTM totalmente conectadas achatadas

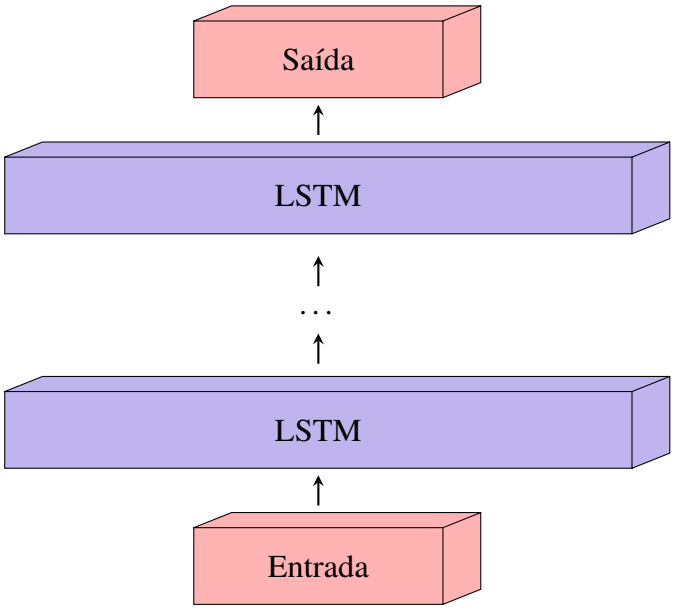


Fonte: Produção do próprio autor.

Em uma rede LSTM multicamada tem-se várias redes neurais postas de tal forma que a saída de uma se torna a entrada de outra como demonstra a Figura 10. No diagrama, segue a estrutura de uma rede LSTM multicamada, as setas entre blocos indicam o fluxo de informação dentro da rede neural. Os blocos azuis são uma representação de redes LSTM desdobradas do tempo.

Assim, tem-se um modelo LSTM com camadas LSTM ocultas, estas camadas ocultas adicionais são entendidas como uma forma de recombinar as representações aprendidas nas camadas anteriores e criar novas representações em níveis maiores de abstração (BROWNLEE, 2017).

Figura 10 – Diagrama de redes LSTM multicamadas



Fonte: Produção do próprio autor.

2.7 *Autoencoder* LSTM

Autoencoders são à junção de um *encoder* e um *decoder*, isto é, uma estrutura que busca comprimir a representação da entrada através da redução da dimensionalidade no meio de sua estrutura e após esta redução, expandir a mesma para seu formato original.

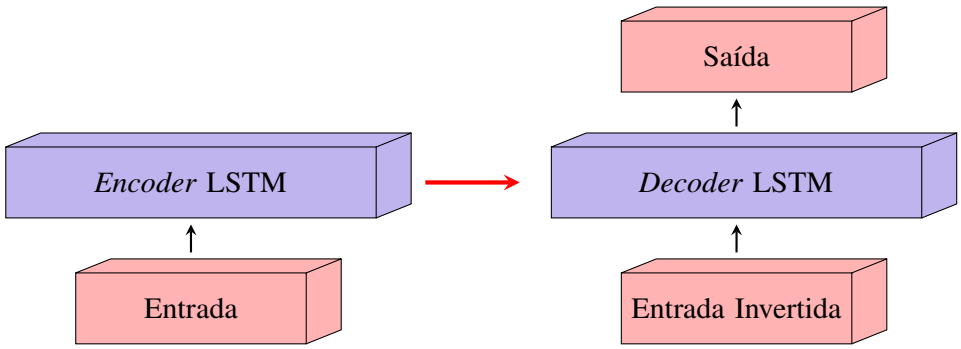
Ao se criar uma estrutura onde é feita a redução da entrada e depois a reconstrução desta, tem-se um modelo que necessariamente passa a aprender características relevantes dos dados. Isto ocorre pois, durante o treinamento, deseja-se minimizar a diferença entre a entrada e a saída. Desta forma, a rede tende a aprender as características mais relevantes da entrada para conseguir reconstruí-la, ou seja, a única forma da rede ser capaz de reconstruir a entrada no *decoder* é através de informações relevantes.

Devido a esta redução da dimensionalidade, *autoencoders* formam ferramentas de seleção de características (do inglês, *Feature selection*), isto é, uma ferramenta para a seleção de características representativas dos dados de entrada. Para maior detalhe ver o APÊNDICE A. Com isso, *autoencoders* LSTM podem ser utilizados para prever futuros *frames* em um vídeo analisando os

últimos *frames* do vídeo, aprendendo quais informações são relevantes para o vídeo e criando um novo *frame* com estas informações (SRIVASTAVA; MANSIMOV; SALAKHUDINOV, 2015).

Neste contexto, um *autoencoder* LSTM tem uma arquitetura do tipo *Many to Many* onde a saída desejada é a mesma informação na entrada, ou sua informação invertida. Na Figura 11 é exemplificada o mecanismo comentado acima, transferência do espaço latente esta destacada pela seta vermelha demonstrando que o fluxo de informação do *encoder* para o *decoder* se dá através dos estados internos da LSTM, não de sua saída.

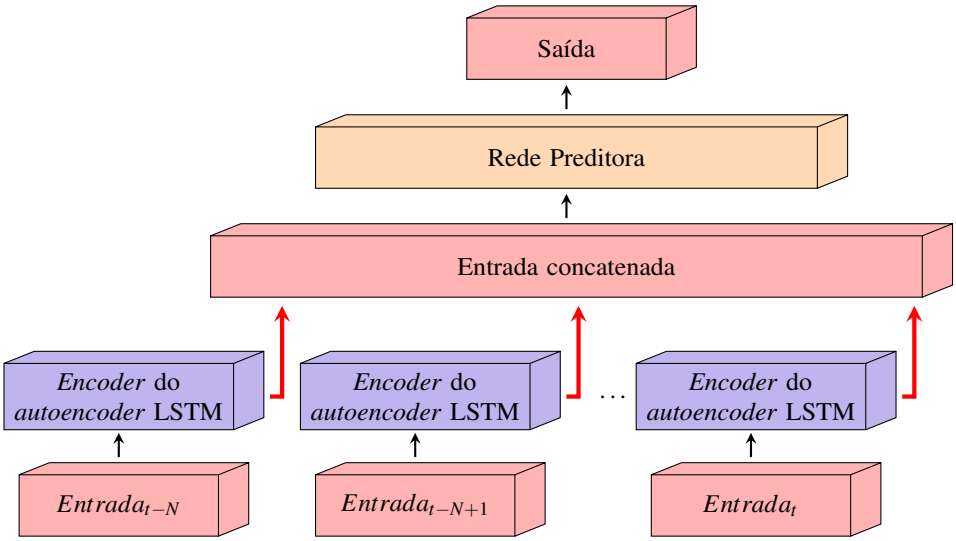
Figura 11 – Estrutura de um *autoencoder* LSTM



Fonte: Produção do próprio autor.

Com o uso de redes *autoencoders* pode-se, inicialmente, criar uma rede cujo objetivo é selecionar características relevantes dos dados obtidos e posteriormente, utilizar o *encoder* (parte do modelo cuja saída forma o espaço latente) como entrada de outro modelo. Este tipo de estrutura, define um modelo composto, onde, utilizando o *encoder* da rede *autoencoder* LSTM como forma de pré-processamento da entrada, seleciona-se as características mais relevantes e posteriormente, alimenta-se uma outra rede com estes dados. Nota-se que o bloco “Rede Preditora” pode ser qualquer rede neural, não só redes LSTM como os vistos nas subseções anteriores da Seção 2.6. As setas entre blocos indicam o fluxo de informação dentro da rede neural. Os blocos azuis são uma representação de redes *encoder* LSTM desdobradas do tempo. As entradas e saídas são determinadas pelos blocos vermelhos e a rede preditora é determinada pela cor laranja.

Figura 12 – Diagrama de redes compostas



Fonte: Produção do próprio autor.

2.8 Resumo

Deste capítulo pode-se concluir que: O mercado financeiro pode ser visto como um problema de análise de séries temporais. Para fazer tal análise, pode-se utilizar métodos envolvendo redes neurais. Em especial, redes neurais recorrentes são boas candidatas à modelar e arquitetar sistemas capazes de tal análise devido a sua capacidade de lidar com séries temporais. Uma arquitetura diferente, que propõe uma forma de lidar com os problemas existentes em redes neurais recorrentes, a arquitetura LSTM, apresenta características bem vindas à problemas de séries temporais, onde podem haver relações temporais mais longas, que redes recorrentes comuns não poderiam observar. O que segue deste trabalho são as partes constitutivas do sistema de *trading* desenvolvido. Portanto, Os seguintes capítulos estão dedicados especificamente às etapas de metodologia e desenvolvimento do modelo proposto.

3 SOLUÇÃO PROPOSTA

Neste capítulo é apresentado o sistema proposto para a solução do problema em estudo. Sendo assim, o capítulo divide-se em duas seções além desta introdução. Na primeira seção, busca-se explicar em detalhes como cada etapa foi executada, seus resultados e o motivo de sua execução. Desta forma, é descrito os passos dados para a produção da solução ao problema proposto. Posterior à isso, na segunda seção sintetiza-se o sistema proposta de tal forma que fique claro como o sistema opera.

3.1 Solução Proposta

3.1.1 Visão Geral

Através dos estudos feitos, observou-se características especificas no problema de predição em séries temporais devido à sua não estacionariedade. Buscou-se criar uma método robusto de pré-processamento para que fosse possível extrair mais informações e formatar estas informações para que aumentasse a robustez da solução. Não só isso, como testou-se diferentes arquiteturas especificadas na Seção 2.6 para que fosse possível decidir qual arquitetura apresenta melhores resultados em termos de predição.

O sistema construído parte inicialmente do treinamento *offline* com um conjunto de dados, onde treina-se diferentes modelos com o objetivo de aprender os padrões presentes em séries temporais. Os melhores modelos são avaliados em termos de erro de validação através de métricas de regressão.

Posteriormente, testa-se tais modelos treinados em um conjunto de teste. Durante os testes, além de criar comparações cabíveis entre os modelos, cria-se comparações com outros métodos de regressão clássicos assim como outros métodos de processamento de dados. Por fim, os melhores modelos são analisados em cenário real através do *backtesting* onde testa-se a rentabilidade com o auxílio de métricas do mercado financeiro, isto é, utiliza-se do poder de criar predições de cada

modelo e julga-se, com a taxa de comissão estipulada no mercado e o *slippage* do mercado, se é possível obter lucro.

A solução baseia-se em um regime de operação que busca criar sinais de entrada e saída do mercado em uma janela de quinze minutos até trinta minutos, uma vez que, o comportamento de ativos pode ser visto como um sistema randômico para escalas de tempo pequenas (MAHATA; PRASAD, 2019). Entretanto, não se pode escolher uma escala de tempo grande pois a medida que se aumenta a escala de tempo, aumenta-se também a influência de eventos e notícias do país e do mundo, onde ocorre aumento súbito de volume negociado, que por sua vez leva a maior volatilidade.

O projeto foi implementado de forma a criar uma lógica de funcionamento do sistema semelhante a lógica de microsserviços onde a filosofia de organização de uma aplicação se baseia em pequenos serviços acoplados uns aos outros.

Para tanto, separa-se o projeto em diferentes partes. Cria-se um serviço dedicado ao treinamento de modelos onde todo o processo é automático. Modelos são criados, treinados, testados, salvos, carregados e disponibilizam suas informações automaticamente. Assim, diferentes arquiteturas podem ser testadas rapidamente. Não só isso, como a inferência dos modelos é disponibilizada como um serviço através de um servidor. Os modelos são carregados e permanecem prontos para executar predições através de requisições de comunicação via protocolo HTTP recebidas pelo servidor. Com serviços voltados a treinamento e inferência de diferentes modelos para regressão, cria-se um serviço que mede a qualidade dos modelos através de métricas de análise de regressão da literatura, assim como a qualidade em se gerar lucros na bolsa de valores através do já mencionado *backtesting*.

Para esta lógica de microsserviços, foram construídos algoritmos em Python com o uso de bibliotecas amplamente utilizadas, o que facilita o suporte ao projeto uma vez que tais bibliotecas são gerenciadas por uma grande comunidade de usuários e empresas. Com esta estrutura, busca-se uma forma de rápida de prototipagem já que diferentes modelos podem ser criados, testados e postos em produção rapidamente. Desta forma, é possível definir características reais de operação como porcentagens de comissão, *spread* e *slippage* que ocorrem no mercado financeiro brasileiro, o B3, e características fundamentais dos modelos LSTM a fim de gerar um relatório de atividades

de *backtesting* em diferentes ativos com cada posição tomada pelo sistema além de um conjunto de métricas econômicas a fim de facilitar a análise do sistema.

3.1.2 Pré-processamento

Um dos problemas de técnicas comuns de extrapolação em séries temporais, principalmente em dados econômicos, é como abordar a falta de periodicidade dos dados. Para tratar este problema, foi executado um procedimento de tratamento dos dados históricos recebidos, isto é, criou-se um procedimento que recebe os preços OHLC de ações e a partir disto, gera-se um conjunto de dados prontos para a rede neural utilizar. Note que durante os experimentos o pré-processamento de dados estipulado aqui também será avaliado.

Em resumo, são utilizados indicadores técnicos como ferramentas para extração de características e os divide em características que apresentam teor acumulativo. A partir disso, aplica-se métodos distintos para que ambos possam ser utilizados. A método será descrito em detalhes abaixo.

Define-se $\mathbf{p}_{fechamento} \in \mathbb{R}^{H\times 1}$, $\mathbf{p}_{abertura} \in \mathbb{R}^{H\times 1}$, $\mathbf{p}_{mximo} \in \mathbb{R}^{H\times 1}$, $\mathbf{p}_{minimo} \in \mathbb{R}^{H\times 1}$, como o valor dos preços de fechamento, abertura, máximo e mínimo do dados históricos de um ativo, respectivamente, considerando H dados históricos.

Então, a partir dos dados $\mathbf{p}_{fechamento}, \mathbf{p}_{abertura}, \mathbf{p}_{mximo}, \mathbf{p}_{minimo}$ determinamos os indicadores econômicos: MACD, Williams %R, ROC, Bandas de Bollinger, ATR e SMA, tal que: (*i*) o operador *MACD* calcula o indicador MACD via a equação (2.6); (*ii*) o operador $William$ calcula o indicador Williams %R via a equação (2.3); (*iii*) o operador *ROC* calcula o indicador ROC via a equação (2.2); (*iv*) o operador $Bollinger_{inf}$ calcula o indicador Banda inferior de Bollinger via a equação (2.10); (*v*) o operador $Bollinger_{sup}$ calcula o indicador Banda superior de Bollinger via a equação (2.9); (*vi*) o operador ATR calcula o indicador ATR via a equação (2.12) e (*vii*) o operador *SMA* calcula o indicador Médias Móveis Simples via a equação (2.1). Com isso tem-se o seguinte conjunto de indicadores:

$$\mathbf{i}_{MACD} = MACD(\mathbf{p}_{fechamento}) \in \mathbb{R}^{H' \times 1}, \tag{3.1}$$

$$[\mathbf{i}_{W_7}, \mathbf{i}_{W_{21}}] = William(\mathbf{p}_{fechamento}, \mathbf{p}_{mximo}, \mathbf{p}_{minimo}) \in \mathbb{R}^{H' \times 1} \tag{3.2}$$

$$[\mathbf{i}_{ROC_7}, \mathbf{i}_{ROC_{21}}] = ROC(\mathbf{p}_{fechamento}) \in \mathbb{R}^{H' \times 2} \tag{3.3}$$

$$\mathbf{i}_{B_{inf}} = Bollinger_{inf}(\mathbf{p}_{fechamento}) \in \mathbb{R}^{H' \times 1} \tag{3.4}$$

$$\mathbf{i}_{B_{sup}} = Bollinger_{sup}(\mathbf{p}_{fechamento}) \in \mathbb{R}^{H' \times 1} \tag{3.5}$$

$$[\mathbf{i}_{ATR_7}, \mathbf{i}_{ATR_{21}}] = ATR(\mathbf{p}_{fechamento}, \mathbf{p}_{mximo}) \in \mathbb{R}^{H' \times 2}, \tag{3.6}$$

$$[\mathbf{i}_{SMA_7}, \mathbf{i}_{SMA_{21}}] = SMA(\mathbf{p}_{fechamento}) \in \mathbb{R}^{H' \times 2}. \tag{3.7}$$

Cabe indicar que o período curto utilizado equivale a sete e o período longo equivale a vinte e um. Portanto, para garantir que todos indicadores tenham a mesma quantidade de informação, tem-se $H' = H - 21$. Com os indicadores calculados, agrupa-os em indicadores não cumulativos ($\mathbf{I}_P$) e cumulativos ($\mathbf{I}_A$), especificamente:

$$\mathbf{I}_P = [\mathbf{i}_{MACD}\ \mathbf{i}_{W_7}\ \mathbf{i}_{W_{21}}\ \mathbf{i}_{ROC_7}\ \mathbf{i}_{ROC_{21}}\ \mathbf{i}_{ATR_7}\ \mathbf{i}_{ATR_{21}}] \in \mathbb{R}^{H' \times 7} \tag{3.8}$$

$$\mathbf{I}_A = \left[\mathbf{i}_{B_{inf_{14}}}\ \mathbf{i}_{B_{sup_{14}}}\ \mathbf{i}_{SMA_7}\ \mathbf{i}_{SMA_{21}}\ \mathbf{p}_{fechamento}\ \mathbf{p}_{abertura}\ \mathbf{p}_{mximo}\ \mathbf{p}_{mnimo}\right] \in \mathbb{R}^{H' \times 8} \tag{3.9}$$

Sobre os indicadores onde percebe-se efeitos de tendência, aplica-se o operador $Detrend(\bullet)$[1] que retira o comportamento acumulativo dos indicadores $\mathbf{I}_A$, isto é, retira os efeitos da tendência de mercado presentes em $\mathbf{I}_A$. Este tratamento, perde um valor histórico, por tanto $H'' = H' - 1$. Este processo é representado na a equação (3.10), onde $\mathbf{V}_A$ representa os indicadores cumulativos agora sem tendência.

$$\mathbf{V}_A = Detrend(\mathbf{I}_A) \in \mathbb{R}^{H'' \times 8} \tag{3.10}$$

Com os dois conjuntos de indicadores $\mathbf{V}_A$ e $\mathbf{I}_P$ e os preços de fechamento $\mathbf{p}_{fechamento}$, são construídos os conjuntos de dados de entrada $\mathbf{X}$ e saída $\mathbf{y}$, onde, cada coluna de $\mathbf{X}$ representa as 15 características obtidas em um instante especifico, e $\mathbf{y}$ representa o alvo para o treinamento do modelo.

[1] O operador $Detrend(\bullet)$ calcula a diferença entre cada dois elementos consecutivos de cada coluna da matriz de entrada, ou seja, $\mathbf{Y} = Detrend(\mathbf{X})$, implicara que, $y_{ij} = x_{i+1,j} - x_{i,j}$.

$$\mathbf{X} = \begin{bmatrix} \mathbf{I}_P^T \\ \mathbf{V}_A^T \end{bmatrix} \in \mathbb{R}^{15\times H''} = [\mathbf{x}_1 \cdots \mathbf{x}_t \cdots \mathbf{x}_{H''}] \ , \ \mathbf{x}_t \in \mathbb{R}^{1\times H''} \tag{3.11}$$

$$\mathbf{y} = Detrend(\mathbf{p}^T_{fechamento}) \in \mathbb{R}^{1\times H''} = [y_1, \cdots, y_t, \cdots, y_{H''}] \ , \ y_t \in \mathbb{R} \tag{3.12}$$

Usando $\mathbf{X}$ e $\mathbf{y}$, compõem-se então, um conjunto de pares entrada-saída em forma de janela deslizante $\mathscr{T} = \left\{(\mathsf{x}_1, \mathsf{y}_1), (\mathsf{x}_2, \mathsf{y}_2), \cdots, (\mathsf{x}_{H'/B}, \mathsf{y}_{H'/B})\right\}$, tal que cada entrada x_k é uma janela de comprimento B dos indicadores e a saída correspondente, y_k é o preço de fechamento da ação no instante seguinte ao ultimo conjunto de observações capturadas na janela k-ésima, especificamente:

$$\mathsf{x}_1 = [\mathbf{x}_1 \cdots \mathbf{x}_B] \qquad \mathsf{y}_1 = [0, \cdots, 0, y_{B+1}] \tag{3.13}$$

$$\mathsf{x}_2 = [\mathbf{x}_{B+1} \cdots \mathbf{x}_{2B}] \qquad \mathsf{y}_2 = [0, \cdots, 0, y_{2B+1}] \tag{3.14}$$

$$\mathsf{x}_{H''/B} = \left[\mathbf{x}_{H'-(B+1)} \cdots \mathbf{x}_{H''}\right] \quad \mathsf{y}_{H''/B} = [0, \cdots, 0, y_{H''+1}] \tag{3.15}$$

Por fim, normaliza-se cada janela aplicando o operador de normalização Min-Max denominado aqui como $Norm$[2] no conjunto entrada-saída. Nota-se que y_i é normalizado em relação ao intervalo de x_i, $[Max(\mathsf{x}_i), Min(\mathsf{x}_i)]$, para que tenha-se o mesmo intervalo de normalização para cada janela. Com a normalização por janela, busca-se acelerar o tempo de treinamento minimizando efeitos de divergência, (BHANJA; DAS, 2018). Tem-se então:

$$\mathsf{x}_1 = Norm([\mathbf{x}_1 \cdots \mathbf{x}_B]) \qquad \mathsf{y}_1 = Norm([0, \cdots, 0, y_{B+1}]) \tag{3.16}$$

$$\mathsf{x}_2 = Norm([\mathbf{x}_{B+1} \cdots \mathbf{x}_{2B}]) \qquad \mathsf{y}_2 = Norm([0, \cdots, 0, y_{2B+1}]) \tag{3.17}$$

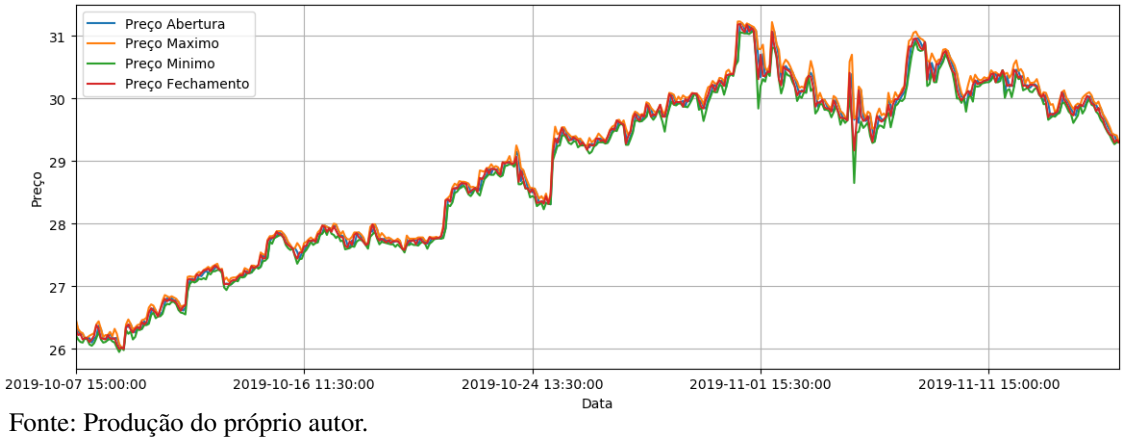
$$\mathsf{x}_{H''/B} = Norm(\left[\mathbf{x}_{H'-(B+1)} \cdots \mathbf{x}_{H''}\right]) \quad \mathsf{y}_{H''/B} = Norm([0, \cdots, 0, y_{H''+1}]) \tag{3.18}$$

O processo de divisão em janelas deslizantes força a rede neural aprender relações dentro de cada janela. Como forma de ajudar na visualização do algoritmo proposto, segue nos Gráficos 6

[2] O operador $Norm$ normaliza entre 0 e 1 cada elemento da janela de entrada, ou seja, $\mathsf{x}_{normi} = Norm(\mathsf{x}_i)$, implicara que, $\mathsf{x}_{normi} = \frac{\mathsf{x}_i - Min(\mathsf{x}_i)}{Max(\mathsf{x}_i) - Min(\mathsf{x}_i)}$, aqui, os operadores $Min(\mathsf{x}_i)$ e $Max(\mathsf{x}_i)$ retornam o valor mínimo e máximo da janela de entrada.

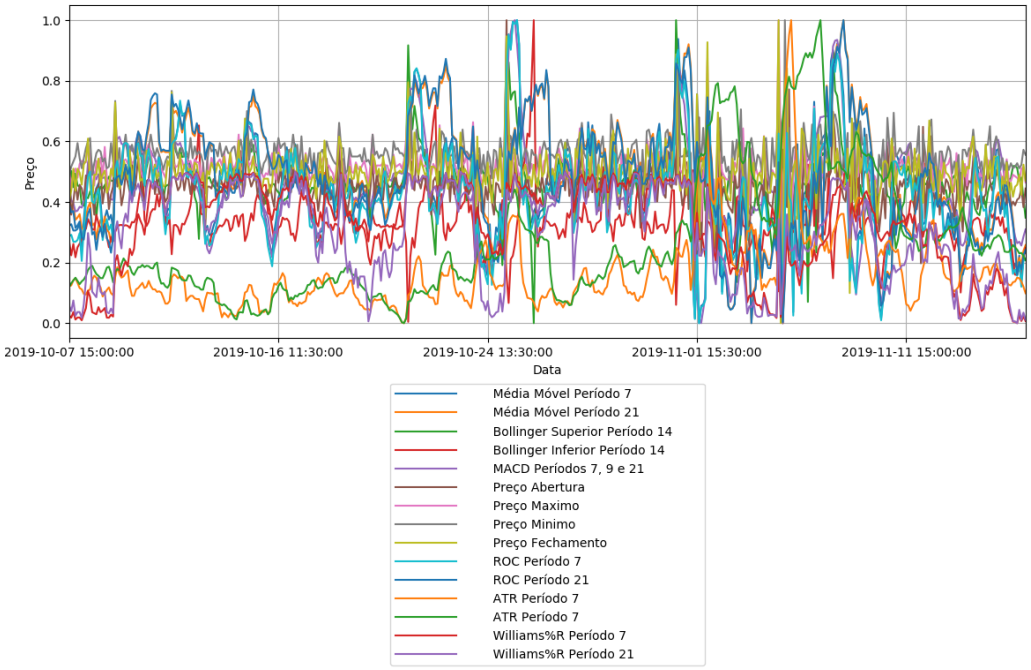
e 7, o comparativo de uma janela de dados com valores reais e os seus valores depois de serem processados.

Gráfico 6 – Dados recebidos



Fonte: Produção do próprio autor.

Gráfico 7 – Dados processados

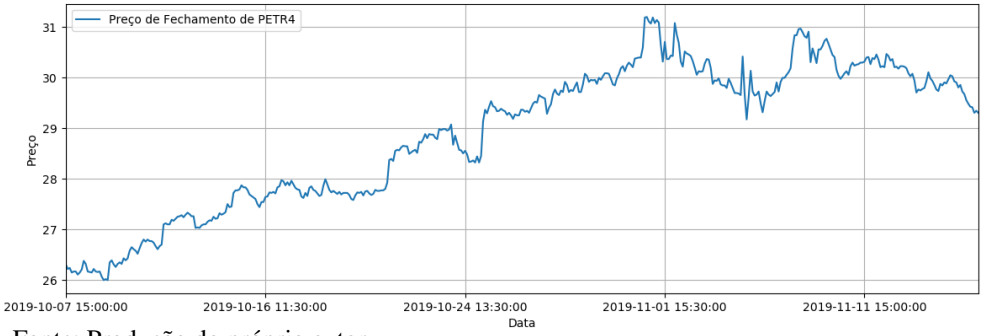


Fonte: Produção do próprio autor.

Nos Gráficos 8 e 9 evidencia-se o efeito da função *Detrend* sobre os dados, foi executado *Detrend* sobre o preço de fechamento das ações da Petrobras (PETR4) obtidos via Yahoo (2019a), nota-se que variações no tempo se transformam em valores absolutos em volta de uma faixa central. A vantagem de se operar desta forma reside em se retirar a força de tendência de um movimento, obtendo a variação entre tempo t e $t-1$ de forma absoluta. Isto é, obtém-se a

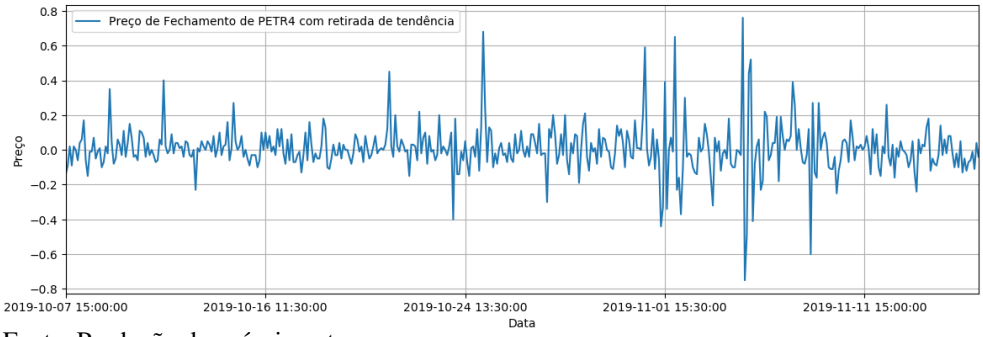
real variação de um movimento, o que leva a facilitar a análise de periodicidade de uma série temporal.

Gráfico 8 – Ação da Petrobras antes da remoção de tendência



Fonte: Produção do próprio autor.

Gráfico 9 – Ação da Petrobras após da remoção de tendência



Fonte: Produção do próprio autor.

É possível checar a validez deste operador através do teste da hipótese nula pelo método de *Augmented Dickey-Fuller* para checar se a série temporal apresenta raiz unitária utilizando a aproximação de MacKinnon da probabilidade de significância denominada valor-p. Caso um sinal seja estacionário, seu valor-p será um número menor que $0,5$ (DICKEY; FULLER, 1979).

No Quadro 1 segue a comparação do valor-p e a estacionariedade dos valores de fechamento das ações da Petrobras de 2016 até 2019 ao se executar o operador $Detrend$, dados obtidos via Yahoo (2019a). Note que, ao se observar o valor-p, para a mesma série temporal após a aplicação do operador temos forte comportamento estacionário devido ao baixo valor-p obtido.

Quadro 1 – Valor-p de séries temporais antes e depois do operador *Detrend*

Tipo	Valor-p	Estacionariedade
PETR4 Normal	$0,670$	Não Estacionário
PETR4 com *Detrend*	$2,025 \times 10^{-30}$	Estacionário

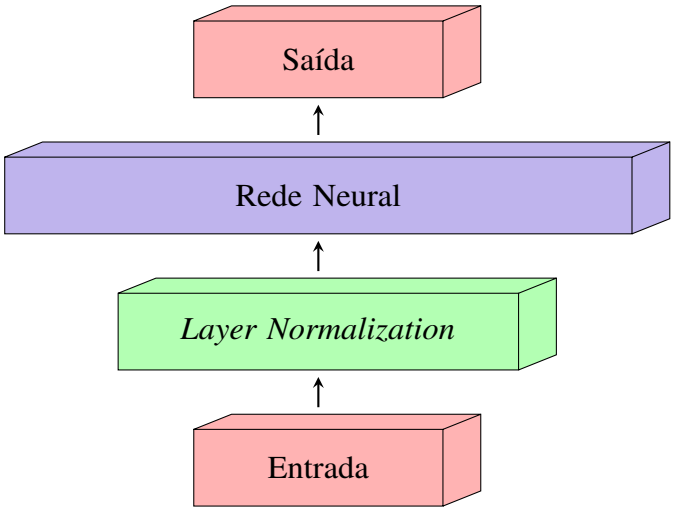
Fonte: Produção do próprio autor.

3.1.3 Construção dos Modelos

São propostos dois tipos de modelos: modelos preditores e compostos, ambos serão explicados a seguir. Modelos preditores São redes LSTM treinadas diretamente com os dados, isto é, são os modelos LSTM que treinam diretamente com os dados OHLC já pré-processados. Também, é proposto adicionar aos mesmos modelos uma camada de normalização de camada (do inglês, *Layer Normalization*), onde busca-se modelos que possam ser treinados com maior taxa de aprendizado sem que isto tenha efeitos como explosão de gradiente ou *overfitting*, desta forma, torna-se possível convergir ao mínimo global mais rápido quando comparado a um modelo sem esta camada (BA; KIROS; HINTON, 2016). como demonstra a Figura 13, segue a estrutura de uma rede qualquer com a camada de *Layer Normalization*. Os blocos azuis são uma representação de redes neurais quaisquer tendo em vista que esta técnica pode ser utilizada com qualquer arquitetura, já o bloco verde representa a camada de *Layer Normalization*. Esta camada antecede o resto das camadas, podendo ser vista como uma etapa de pré-processamento.

Modelos compostos são definidos pela junção de um *autoencoder* LSTM e um modelo preditor. Aqui, a estrutura de um modelo composto é semelhante ao modelo proposto na Figura 11, isto é, tem-se uma arquitetura LSTM *Many to Many* que passa seu estado e sua saída, $\mathbf{c}_t$ e $\mathbf{h}_t$ como entrada de uma rede LSTM com arquitetura *Many to One*, uma vez que as predições serão feitas apenas um passo no futuro. Desta forma, busca-se inicialmente aprender características importantes sobre a composição de séries temporais de tal forma que a representação de espaço latente do *autoencoder* LSTM seja utilizado como entrada para o modelo preditor. Isto é, no modelo composto, temos um modelo preditor sendo treinado com o espaço latente de um *autoencoder*, não com os os dados OHLC processados.

Figura 13 – Diagrama de redes neurais com *Layer Normalization*



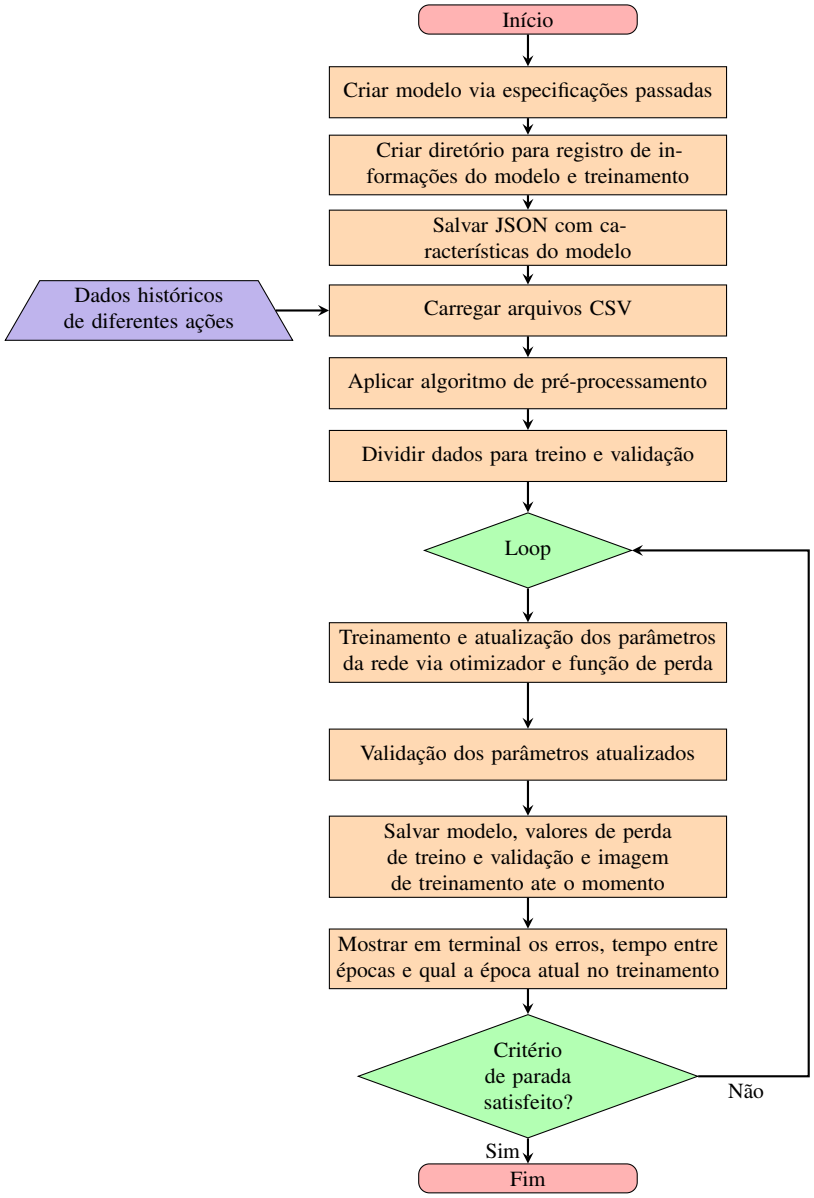
Fonte: Produção do próprio autor.

3.1.4 Etapa de treinamento

O treinamento é feito em três etapas: (*i*) treina-se os modelos preditores através do aprendizado supervisionado, utilizando-se os dados OHLC processados; (*ii*) com os diferentes modelos treinados, escolhe-se o melhor modelo. Do melhor modelo, obtém-se o modelo preditor a ser utilizar no sistema de *algorithmic trading* assim como obtém a melhor arquitetura a ser utilizada na construção dos modelos *autoencoder*; (*iii*) treina-se o *autoencoder* através do aprendizado *Self Supervised* com a sequência alvo sendo a sequência de entrada e alimenta-se o *decoder* com a entrada inversa para que seja possível explorar relações próximas no tempo. Feito isso, passa-se para a etapa do treinamento do modelo composto utilizando o *autoencoder* treinado.

O fluxograma de treinamento de todos os modelos criados é mostrado na Figura 14 onde o critério de parada e determinado por qual dos critérios ser alcançado primeiro: Valor de erro de validação passou a aumentar (*Overfitting*) ou treinamento foi executado pelo número de épocas estabelecidas pelo usuário. Vale ressaltar que todos os dados OHLC usados estão em arquivos no formato valores separados por virgula (CSV, do inglês *Comma-separated values*), isto é, arquivos de texto cuja separação de colunas se dá pelo uso de virgula.

Figura 14 – Fluxograma de Treinamento de redes neurais



Fonte: Produção do próprio autor.

3.1.5 Etapa de inferência

Agora, com os melhores modelos selecionados, passa-se à etapa de inferência, onde, o modelo passa a operar como estratégia de *algorithmic trading* em uma plataforma controlada de tal forma que seu lucro seja usado como métrica de sucesso.

Para que as predições dos modelos possam ser utilizadas, é necessário que os resultados passem

por um pós- processamento onde os resultados reais são obtidos através do processo inverso ao pré-processamento. Assim, o preço de fechamento futuro, $\mathbf{p}_{fechamento}[t+1]$, é calculado através da equação (3.19):

$$\hat{\mathbf{p}}_{fechamento}[t+1] = \mathbf{p}_{fechamento}[t] + Denorm(\hat{y}). \tag{3.19}$$

Onde $\mathbf{p}_{fechamento}[t]$ é o preço de fechamento no instante atual, $\hat{y}$ é a predição feita pelo modelo. $Denorm(\bullet)$ é o operador de desnormalização[3] Intuitivamente a equação (3.19) mostra que o preço de fechamento predito depende do preço de fechamento do instante anterior à predição mais o efeito de tendência à predição representado por $Denorm(\hat{y})$.

3.1.6 *Backtesting* e *Algorithmic Trading*

A plataforma de *backtesting* trabalha independente do modelo usado, contando que os modelos providenciem suas informações no mesmo padrão. Esta forma é preferível uma vez que o objetivo do *backtesting* é simular um sistema real. Assim, como seria em execução em tempo real, o sistema utiliza as predições do modelo criado para decidir se vale a pena ou não se posicionar no mercado referente aos parâmetros do mercado financeiro. Desta forma, garante-se que o sistema só opere caso exista um certo grau de certeza de obter um lucro.

Para a execução do *backtesting*, execução dos testes e obtenção de diferentes métricas de operação utiliza-se bibliotecas escritas em Python amplamente utilizadas na comunidade. Porém, para a utilização na bolsa de valores, necessita-se de uma forma de comunicação padronizada capaz de funcionar independente da plataforma escolhida para se operar na bolsa de valores.

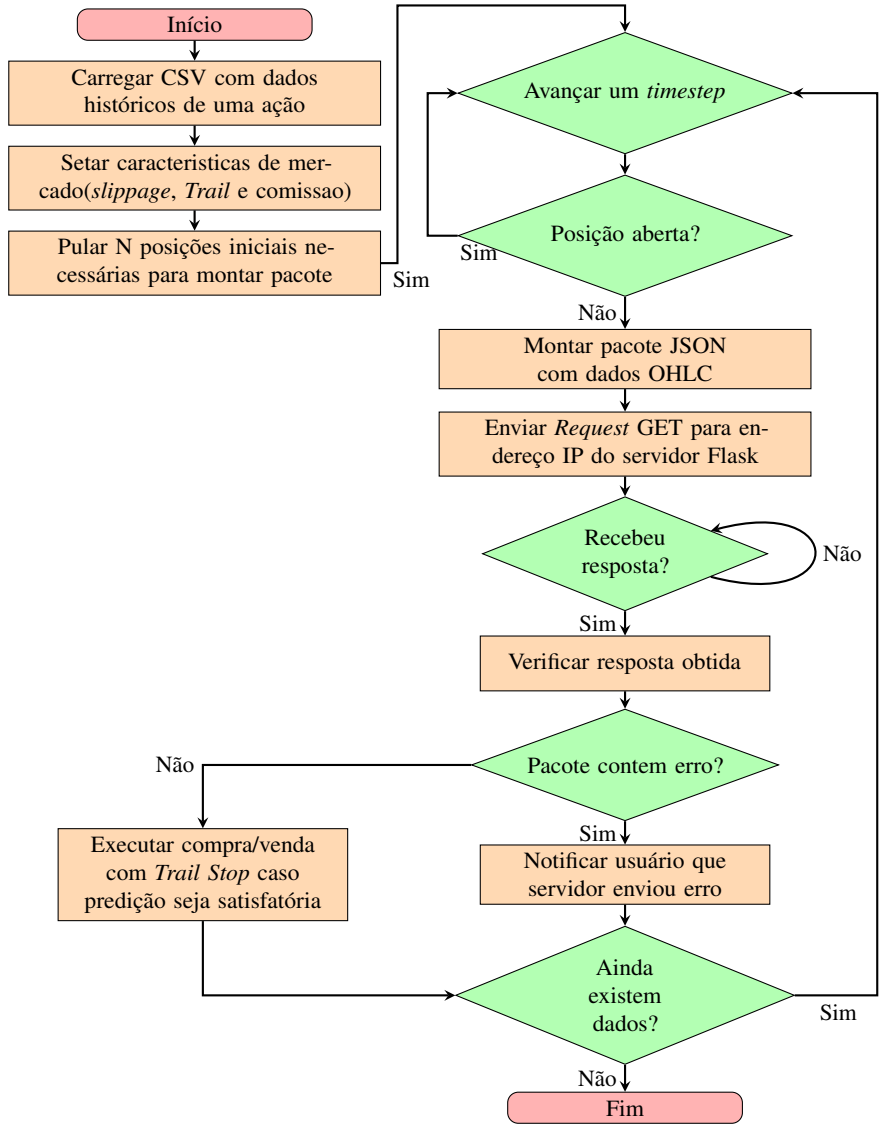
Para tanto, propõe-se uma forma unificada de interface com diferentes modelos, isto é, a fim de manter a interoperabilidade do sistema, utiliza-se o protocolo HTTP para comunicação entre

[3] O operador $Denorm(\bullet)$ inverte o processo de normalização o valor entre o intervalo normalizado no pré-processamento $[Min(Norm_{intervalo}), Max(Norm_{intervalo})]$ para o intervalo $[Min(\mathrm{x}), Max(\mathrm{x})]$, ou seja, $y_i = Denorm(y_{normi})$, implicara que, $y_{denormi} = Min(\mathrm{x}_i) + \frac{(Max(Norm_{intervalo}) - Min(\mathrm{x}_i)) * (y - Min(\mathrm{x}_i))}{Max(\mathrm{x}_i) - Min(\mathrm{x})}$, aqui, os operadores $Min(\mathrm{x}_i)$ e $Max(\mathrm{x}_i)$ retornam o valor mínimo e máximo da janela de entrada x_i.

o modelo, a plataforma de *backtesting* e a plataforma para operação na bolsa de valores. É construído um servidor em Python, utilizando a biblioteca Flask, para expor diferentes modelos através de requisições do tipo GET. A utilização do servidor como forma de comunicação com o sistema, garante ao usuário escolher em qual plataforma deseja-se operar, contando que esta plataforma tenha a habilidade de trocar informações via protocolo HTTP com um servidor.

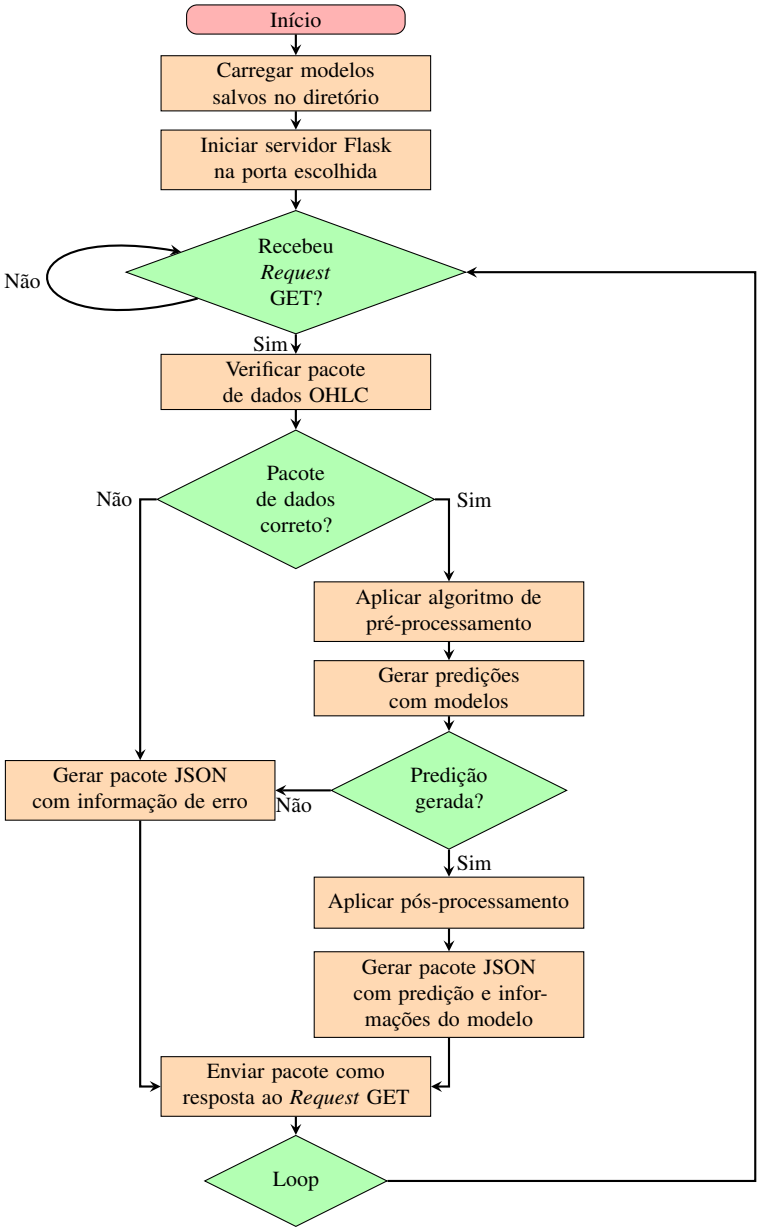
O processo de *backtesting* se dá pelo fluxograma definido na Figura 15, nota-se que a operação de compra e venda finaliza automaticamente caso um valor previamente estipulado seja alcançado. Já o processo de execução do servidor é definido pelo fluxograma mostrado na Figura 16 onde é demonstrado o processo de recebimento e envio de informações via protocolo HTTP. Note que o servidor envia ao usuário possíveis falhas no servidor seja nos dados enviados pelo usuário, como em cenários onde algum modelo falhe Devido a forma de comunicação com os modelos ser dada via protocolo de comunicação HTTP, a maneira de funcionamento descrita acima garante que os modelos possam ser utilizados da mesma forma, seja em um ambiente simulado, seja em um cenário real de execução. O uso de um servidor para comunicação garante que a inicialização e o processamento dos modelos esteja encapsulado em um serviço.

Figura 15 – Fluxograma do processo de *Backtrading*



Fonte: Produção do próprio autor.

Figura 16 – Fluxograma da operação do servidor enviando dados



Fonte: Produção do próprio autor.

Para a execução do *backtesting*, estipula-se valores de comissão, *slippage* e *spread* para que o modelo atue conforme o mercado. Além disso, cria-se uma estratégia de operação com o uso de um parada de perda com trilha (do inglês, *Trail Stop Loss*), isto é, caso o valor real de uma ação varie contra o movimento predito pelo modelo, será executada a ordem de fechamento de posição. Esta forma de atuação previne o sistema de confiar em demasiado no poder de predição do modelo.

Para o *backtesting*, seleciona-se os ativos mais negociados na bolsa de valores, o B3, para que

possa ser avaliada a rentabilidade dos modelos. O motivo da escolha de ações mais negociadas está associado ao *spread* e *slippage* a ser utilizado. Em ativos onde há maior volume de negociação tem-se um *spread* menor, uma vez que há um grande volume de compradores e vendedores se posicionando no mercado. O que torna a diferença entre preço de venda e preço de compra menor devido a oferta e a procura.

Além disto, com maior volume de negociações, tem-se um *slippage* menor pelo mesmo motivo Vale ressaltar que estes parâmetros mudam com o tempo e com a ação sendo analisada, portanto, escolhe-se setar um valor porcentual pequeno para ambos tal que seja simulado um cenário de alto fluxo de negociação, isto é, valores baixos. A taxa de corretagem varia de acordo com a corretora utilizada pelo usuário, sendo que há corretoras onde não é aplicada a taxa de corretagem. Para tanto, escolhe-se uma taxa de corretagem percentual para que cada posição tomada no mercado financeiro seja taxado. Note que os valores percentuais são calculados a partir do total gasto em uma operação, seja de compra ou venda. O Quadro 2 mostra o valor dos parâmetros definidos para o *backtesting* criado. Com estes valores busca-se criar um sistema que se assemelha ao cenário real na bolsa de valores.

Quadro 2 – Parâmetros dos testes durante o *backtesting*

Parâmetro	Valor
Slippage	$0,1\%$
Spread	$0,1\%$
Commissão	$0,25\%$
Capital Inicial	*R*\$ 100 mil

Fonte: Produção do próprio autor.

A estratégia de *Trailing Stop Loss* consiste em criar um intervalo de segurança para cada posição tomada, onde, ações são tomadas automaticamente caso o valor da ação saia deste intervalo. Isto é, para cada posição tomada tem-se um intervalo $[Limite_{inferior}, Limite_{superior}]$, assim como definido abaixo, caso tenha sido efetuado uma compra:

$$Limite_{superior} = \text{Preço}_{compra} \times (100\% + Stop_{Gain}\%) \tag{3.20}$$

$$Limite_{inferior} = \text{Preço}_{compra} \times (100\% - Stop_{Loss}\%) \tag{3.21}$$

E como é definido abaixo, no caso de uma venda:

$$Limite_{superior} = \text{Preço}_{venda} \times (100\% - Stop_{Gain}\%) \quad (3.22)$$

$$Limite_{inferior} = \text{Preço}_{venda} \times (100\% + Stop_{Loss}\%) \quad (3.23)$$

Se o preço de uma ação for maior que o preço do $Limite_{superior}$ ou menor que o $Limite_{inferior}$ automaticamente fecha-se a posição. Além disso, cria-se um regime de atualização dos limites caso o preço desta ação tenda à aumentar o lucro. Ou seja, caso tenha sido executado uma venda e a ação reduziu de valor ou caso tenha sido executado uma compra a ação aumentou de valor, deve-se atualizar o valor dos limites para que os limites possam acompanhar a tendência ao lucro. Desta forma, utiliza-se o parâmetro *Acréscimo*, tal que o valor dos limites seja atualizado, no caso de uma compra, para:

$$Limite_{superior} = Limite_{superior} \times (100\% + \text{Acréscimo}\%) \quad (3.24)$$

$$Limite_{inferior} = Limite_{inferior} \times (100\% + \text{Acréscimo}\%) \quad (3.25)$$

E para o caso de uma venda:

$$Limite_{superior} = Limite_{superior} \times (100\% - \text{Acréscimo}\%) \quad (3.26)$$

$$Limite_{inferior} = Limite_{inferior} \times (100\% - \text{Acréscimo}\%) \quad (3.27)$$

Além disso, cria-se um intervalo mínimo de variação para que o modelo cogite operar. Isto é, utiliza-se da predição do modelo e o valor atual da ação para calcular um intervalo mínimo de não atuação, uma vez que predições sinalizando baixas variações no preço de uma ação, indicam que não irá valer a pena se posicionar no mercado devido a taxas e riscos associados a esta ação. Utiliza-se o parâmetro *Bias* para determinar os valores mínimos de atuação tal que Diferença$\% > Bias$ ou Diferença$\% < -Bias$, onde:

$$\text{Diferença}\% = 100 \times \left| \frac{\text{Preço}_{predito} - \text{Preço}_{atual}}{\text{Preço}_{Atual}} \right| \quad (3.28)$$

A estratégia de *algorithmic trading* à ser utilizada necessita de quatro parâmetros: *Bias*, *Acréscimo*, $Stop_{Loss}$ e $Stop_{Gain}$. Com o uso de algoritmos, é possível testar vários valores percentuais para cada parâmetro, a fim de determinar a melhor combinação. Para tanto, utiliza-se um modelo regressor perfeito, isto é, um modelo que sabe o valor futuro para decidir os valores ideais para os parâmetros. Utiliza-se as possíveis combinações dos intervalos do Quadro 3 para obter os melhores valores de cada parâmetro. Desta forma, obtém-se 980 combinações de parâmetros a serem testados. Nota-se que a escolha dos parâmetros tenta maximizar o lucro em relação ao número de posições tomadas, uma vez que quanto mais posições são tomadas, mais expõe-se ao risco.

Quadro 3 – Intervalo de parâmetros a serem testados durante *Backtesting*

Parâmetro	Intervalo
Acréscimo	$[2,5\%; 3,75\%; 5\%; 6,25\%; 7,5\%]$
Bias	$[2,5\%; 5\%; 7,5\%; 10\%]$
StopLoss	$[5\%; 12,5\%; 20\%; 27,5\%; 35\%; 42,5\%; 50\%]$
StopGain	$[5\%; 12,5\%; 20\%; 27,5\%; 35\%; 42,5\%; 50\%]$

Fonte: Produção do próprio autor.

A metodologia a ser aplicada na avaliação de *Backtesting* começa pela definição dos parâmetros da estratégia *Trail Stop Loss*, onde, são determinados os melhores parâmetros da estratégia de *Trail Stop Loss* com o uso de um regressor ideal[4]. Em seguida, é executado o *Backtesting* com a estratégia *Trail Stop Loss*, onde os melhores modelos LSTM treinados são utilizados juntos à estratégia *Trail Stop Loss* a fim de analisar a rentabilidade do sistema de *Algorithmic Trading*.

Posteriormente, ocorre a execução do *Backtesting* com outras estratégias onde, no caso seleciona-se duas estratégias amplamente utilizadas: *(i)* cruzamento de Médias Móveis, estratégia que faz uso de duas médias móveis, onde opera-se no cruzamento do sinal entre uma média móvel de período longo e outra de período curto e *(ii)* comprar e Segurar (do inglês, *Buy and Hold*), estratégia onde compra-se uma ação e apenas deixe-a valorizar com o tempo.

Por fim, são feitas comparações entre as estratégias utilizadas no *Backtesting* utilizando os resultados em diferentes cenários das estratégias definidas. Serão comparados os resultados das

[4] Um regressor ideal é um modelo onde as suas predições são dadas por $\hat{y}[t+1] = y[t+1]$. Isto é, o modelo sabe o que irá acontecer no futuro.

estratégias levando em consideração o lucro médio obtido assim como o número de posições tomadas.

3.2 Resumo

Deste capítulo pode-se concluir que: O processo de se gerar predições via modelos LSTM requer um devido tratamento feito sobre os dados de entrada. Para isso, formaliza-se um método de extração de características na tentativa de facilitar o treinamento da rede LSTM. Propõe-se dois tipos de arquiteturas diferentes para que ambas sejam testadas e em conjunto à isso, é proposto um método para testar a rentabilidade de tais modelos em um ambiente controlado. Isto é, executa-se o processo de *backtesting* com parâmetros reais a fim de se obter um esquema de atuação que se assemelha ao cenário real. Modelos são servido via interface HTTP para que todo o trabalho possa ser executado por outras plataformas contando que a plataforma disponibilize o consumo de informações via HTTP. A proposta finaliza com a definição da estratégia a ser utilizada em conjunto ao modelo. O que segue deste trabalho os resultados obtidos das partes constitutivas do sistema desenvolvido, sendo elas, os resultados obtidos dos modelos propostos, o sistema de *backtrading* e servidor.

4 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos do sistema como um todo, isto é, serão mostrados os resultados na etapa de construção e treino dos modelos, assim como os desafios e conclusões tiradas dos resultados obtidos. Além disso, será feita as comparações cabíveis entre os modelos. Posteriormente, será mostrado o resultado das melhores redes neurais obtidas e suas características. Seguindo o raciocínio, será mostrado os resultados obtidos da plataforma de *backtesting* construída e seu funcionamento utilizando o servidor criado para acessar as predições via comunicação HTTP.

4.1 Recursos Computacionais

A solução proposta baseia-se na utilização da linguagem Python (ROSSUM, 1995), com foco na biblioteca de *Machine Learning* Pytorch (PASZKE, 2019) para a construção e testes, treinos e validação de modelos. Além disso, utiliza-se a biblioteca Backtrader (RODRIGUEZ, 2019) para se efetuar o *backtesting* . Em conjunto às ferramentas mencionadas, é utilizada a biblioteca Flask (RONACHER, 2019) com o intuito de criar o servidor para comunicação com os modelos treinados. As bibliotecas Numpy (OLIPHANT, 2006), Matplotlib (HUNTER, 2007) e Pandas (MCKINNEY, 2010) foram amplamente utilizadas durante a etapa de pré-processamento dos dados e análise dos dados OHLC. Para as comparações com outros métodos de regressão, utiliza-se a biblioteca Scikit-Learn (PEDREGOSA, 2011).

Os recursos de *Hardware* utilizados podem se dividir em dois. Sendo o primeiro uma máquina pessoal com as seguintes configurações: (*i*) sistema operacional Ubuntu 16.04; (*ii*) processador Intel Core i7-4770, 3.40GHz com 4 núcleos físicos; (*iii*) memória RAM de 16 GB; (*iv*) unidade de armazenamento de 1TB (disco rígido); (*v*) placa de vídeo Nvidia Geforce GTX 1060, com 3 GB de memória dedicada. E o segundo a plataforma gratuita da Google chamada de Google Colaboratory. Cada usuário pode utilizar até 12 horas de sessão ininterrupta de máquinas com uma GPU da Nvidia, as configurações de cada sessão variam mas em média consegue-se uma sessão com as seguintes configurações: (*i*) sistema operacional Linux; (*ii*) processador Intel Xeon de 2.3GHz; (*iii*) memória RAM de 12 GB; (*iv*) unidade de armazenamento de 300GB (disco

rígido); (*v*) placa de vídeo Nvidia Tesla K80, com 12 GB de memória dedicada e 2496 núcleos CUDA.

Foram utilizados 3 sessões no Google Colaboratory (ALPHABET, 2019a) e o computador pessoal, desta forma consegue-se treinar concomitantemente até 4 modelos de redes LSTM. As sessões no Google Colaboratory embora temporárias, foram suficientes para executar testes de diferentes modelos uma vez que os recursos de *Hardware* da sessão são superiores aos recursos do computador pessoal.

Os dados utilizados foram obtidos através das plataformas Kaggle (ALPHABET, 2019b) e o Metatrader (METAQUOTES, 2019). Do Kaggle, obtém os dados históricos de todos os ativos da bolsa de valores da Nova Iorque (NYSE, do inglês *New York Stock Exchange*), utiliza-se estes dados devido sua facilidade em obtenção em relação aos valores do mercado brasileiro e devido ao seu maior tamanho em relação ao B3, compondo um banco de dados mais diverso. Com isso obtém-se um banco de dados composto do valor dos ativos da NYSE desde a abertura da empresa à bolsa até a data de 16 de novembro de 2018, com o dado de abertura, fechamento, máximo e mínimo diário. Compondo assim, aproximadamente 2.3 GB de dados.

Com esses dados, escolhe-se ativos com maior volatilidade e foca-se na escala de tempo de 2 de fevereiro de 2007 até 16 de novembro de 2018. O motivo desta pré seleção dos dados é devido a uma inspeção sobre os valores dos ativos. Nota-se que para valores passados, a flutuação do preço era consideravelmente menor, uma vez que o banco de dados compõe informações desde a data de 17 de junho de 1977. Com a escolha de vinte e seis ativos presentes no banco de dados, consegue-se 56 mil leituras de abertura, fechamento, máximo e mínimo diário para treino e validação dos resultados. O horário de funcionamento do B3 onde o usuário pode comprar e vender ações no mercado de ativos começa às 10 horas da manhã e termina as 17 horas da tarde horário de Brasília. Note que um dia de negociações representa 7 horas de dados (B3, 2019).

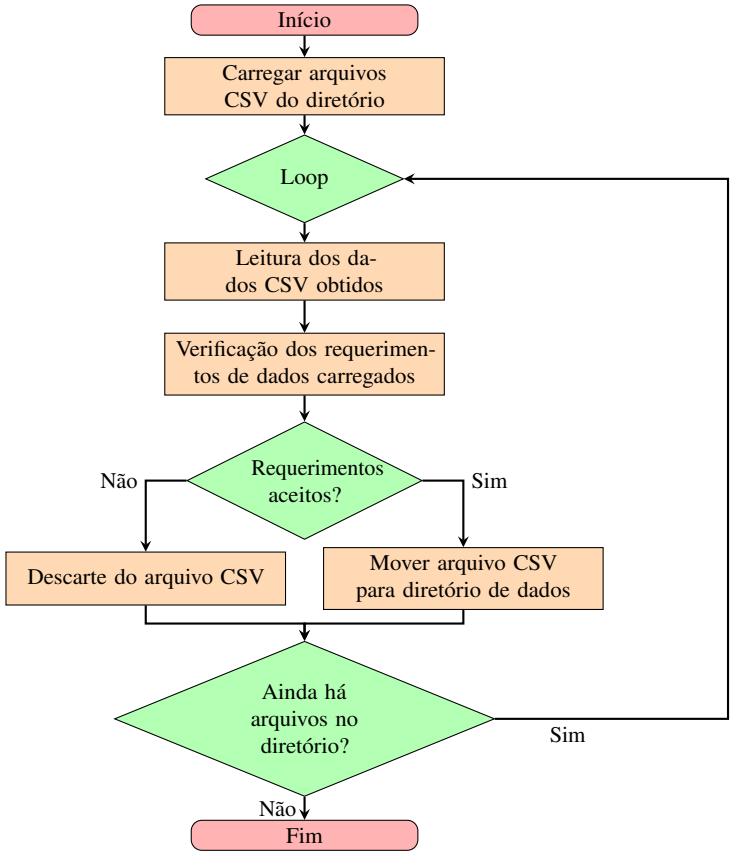
Nota-se que é possível obter os dados da bolsa de valores B3 oficialmente pelo *site* oficial. Porém, os dados históricos estão em formatação não padronizada o que torna o acesso custoso, uma vez que os dados devem ser formatados manualmente para cada acesso. Não só isso, mas os dados obtidos pelo site oficial não contam com os efeitos que eventos e notícias tem sobre os preços de ativos. Por exemplo, empresas podem decidir dividir sua ação por duas, processo chamado

de *split*, neste cenário, o valor da ação seria dividido pela metade. Graficamente, este cenário seria lido como uma desvalorização no preço, porém esta desvalorização não é originada da depreciação, utilizar fontes de dados onde este tipo de cenário não é corrigido fere a capacidade de uma rede neural em gerar predições a medida que a desvalorização vista graficamente não representa um comportamento real de desvalorização.

Com isso, prefere-se utilizar a ferramenta de negociações, o MetaTrader. Com esta plataforma, é possível obter, não só diferentes ativos de diferentes bolsas no mundo, mas também, em diferentes escalas temporais. Por se tratar de uma plataforma para negociação, não compartilha dos mesmos problemas mencionados acima do site oficial do B3. Diferente dos dados obtidos do Kaggle, os dados desta plataforma representam dados históricos atualizados de ativos da bolsa de valores brasileira. Vale ressaltar que o MetaTrader funciona vinculado à uma corretora, entretanto, pode-se criar uma conta fictícia, conhecida como conta Demo para obter os dados e simular operações na bolsa de valores.

Para o uso correto dos dados, as informações mínimas que o arquivo deve conter são os dados históricos de preço de abertura, fechamento, máximo e mínimo. Tais informações são necessárias para a implementação do algoritmo de pré-processamento e extração de características mencionados. Com os dados obtidos, passa-se inicialmente pelo processo de checagem de valores errôneos. Isto ocorre quando, em um dado momento, acontece uma avaria no processo de coleta dos dados e no arquivo CSV é registrado uma entrada não numérica ou vazia. Inicialmente foi utilizado o processo de alimentar para frente (do inglês, *feed foward*) onde o valor errado é substituído pelo valor anterior à ele. Porém, ao se executar este método, cria-se alterações nos valores históricos. Desta forma, optou-se pelo descarte dos arquivos que não satisfazem os dois critérios já que o banco de dados é grande o suficiente para o descarte dos dados errôneos. O fluxograma deste tratamento dos arquivos se dá pela Figura 17. Note que todos os arquivos utilizados são encontrados em formato CSV, com isso, padroniza-se o processo de filtragem, nota-se que o descarte dos arquivos e possível devido a quantidade de dados OHLC disponíveis na internet.

Figura 17 – Fluxograma de filtragem dos dados históricos



Fonte: Produção do próprio autor.

4.2 Detalhes de Implementação

Foi utilizado como otimizador a variante AdamW do otimizador Adam. Tal variante foi criado objetivando corrigir falhas no cálculo da penalidade via norma L^2 no otimizador, de tal forma que consiga melhorar a eficiência do algoritmo (LOSHCHILOV; HUTTER, 2017). Como indicado na literatura, o uso da norma L^2 no otimizador e camadas de *Dropout* na rede LSTM, ajudam à generalização do modelo, isto é, o modelo tende a convergir para uma conjunto de parâmetros que os levam à melhores resultados.

A função de perda utilizada é o erro quadrático médio (MSE, do inglês *Mean Squared Error*). O motivo de seu uso é dado pela volatilidade do mercado financeiro, desta forma, a rede neural deve ser treinada de tal maneira que diferenças mínimas entre predição e valor alvo sejam penalizadas. Com a função de perda MSE, temos uma métrica com sensibilidade ao erro pois a diferença entre a saída esperada e a predita é elevada ao quadrado, desta forma, mesmo erros pequenos são

levados em consideração. Isto é chamado de ser sensível à *outliers*, ou ainda, valores aberrantes.

4.3 Métricas

Como método de comparação entre modelos, somado à métrica de treinamento e validação utilizada, utilizou-se aqui a métrica erro absoluto mediano (MdAE, do inglês *median absolute error*) para determinar os melhores modelos preditores. Esta métrica permite julgar um modelo sem tomar em conta sua sensibilidade a *outliers*. Quanto menor, melhor o modelo. Com estas métricas busca-se analisar a sensibilidade do modelo assim como a sua capacidade geral de observar a tendência. Como o objetivo final do projeto é criar um é criar um sistema que atua na bolsa de valores, após a seleção dos melhores modelos, utilizou-se o lucro obtido, assim como e o número de posições tomadas durante o teste em *backtesting* como métricas para avaliar a eficácia do sistema como uma estratégia de *Algorithm Trading*.

4.4 Experimentos

4.4.1 Hiper-parâmetros utilizados em treinamento

Os hiper-parâmetros demonstrados no Quadro 4 apresentaram os melhores resultados durante testes preliminares e por isso, foram os hiper-parâmetros utilizados durante treinamento. Note que Taxa de aprendizado$_{\text{LN}}$ referencia-se a taxa de aprendizado para os modelos com *Layer Normalization* uma vez que esta técnica visa o aumento da taxa de aprendizado sem compromisso à convergência.

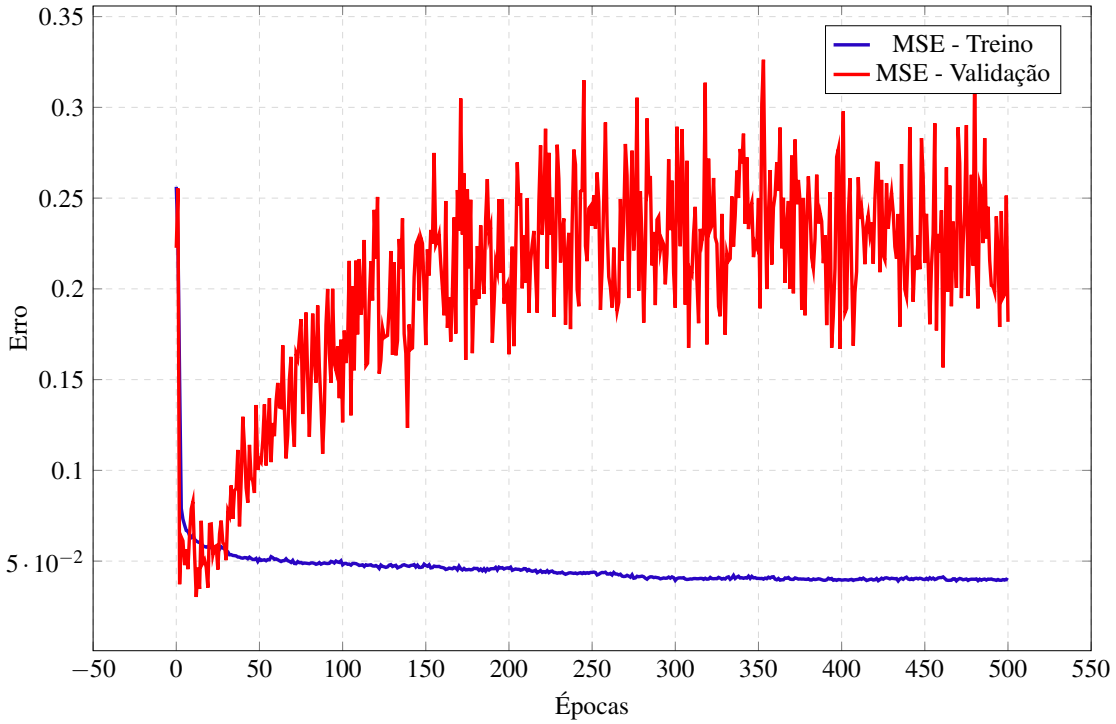
Quadro 4 – Hiper-parâmetros usados no treinamento dos modelos

Hiper-parâmetro	Valor
Tamanho de *Batch*	24
Taxa de aprendizado	$0,0003$
Taxa de aprendizado$_{\text{LN}}$	$0,0005$
Otimizador	AdamW
Função de Erro	MSE

Fonte: Produção do próprio autor.

No Gráfico 10, onde a linha azul representa o erro de treinamento e a linha vermelha representa erro de validação, nota-se a crescente divergência entre as duas métricas a medida que o treinamento ocorre. Foi observado que alguns modelos convergiam durante o treino, mas divergiam durante a validação, evidenciando um processo de *overfitting*. Este comportamento ocorreu com maior frequência a medida que o tamanho do *batch* aumentou, demonstrando a influência deste parâmetro na etapa de treinamento. Quando se constato este comportamento o treinamento do modelo era finalizado.

Gráfico 10 – Treinamento com *overfitting*



Fonte: Produção do próprio autor.

4.4.2 Experimentos Efetuados

Para o desenvolvimento do projeto, foram efetuados cinco experimentos. Os experimentos demonstram sequencialmente o processo para a escolha de parâmetros dos modelos, a análise da influência de cada parâmetro e as comparações cabíveis entre a metodologia proposta e outras técnicas utilizadas para regressão. No primeiro experimento, é efetuada a comparação das diferentes arquiteturas LSTM atuando como preditores considerando um tamanho de janela especifica. Este experimento buscou analisar qual das arquiteturas obtém maior capacidade

em gerar predições corretas sobre a mesma quantidade de informação disponível. No segundo experimento, com a melhor arquitetura LSTM atuando como preditor, buscou-se avaliar a quantidade de informação a ser usada pelo modelo a fim de gerar melhores predições, isto é, determinar o tamanho da janela deslizante adequada para o modelo. No terceiro experimento, com a melhor arquitetura para o modelo preditor e um tamanho da janela deslizante definido, passou-se a treinar o modelo composto (conformado por um *autoencoder* LSTM). e avaliar o ganho ou perda de desempenho através das métricas de MAE e MdAE, procurando determinar se o uso de um *autoencoder* de fato aumenta a capacidade de gerar predições corretas em relação ao modelo preditor. No quarto experimento, com o modelo preditor e composto treinados e comparados como especificado no segundo e terceiro experimento. Passou-se a: (*i*) analisar e validar a metodologia de tratamento de dados; (*ii*) efetuar uma comparação entre o modelo preditor e composto em relação a modelos regressores clássicos da literatura. Nesse contexto, o experimento buscou comparar a eficácia da metodologia de pré-processamento proposta como solução ao problema de gerar predições em séries temporais e validar a escolha de modelos LSTM através de comparações cabíveis em relação a outros modelos que não necessariamente tem uma estrutura neural.

A fim de criar modelos profundos capazes de abstrair informações temporais, cria-se modelos LSTM multicamada de 6 camadas LSTM. Note que na estrutura da rede LSTM, adiciona-se uma taxa de *Dropout* de 25% em todos os modelos testados. Esta taxa, como vista na literatura, aumenta a robustez dos modelos por atuar como um regularizador. Para facilitar a leitura dos resultados, define-se no Quadro 5, as abreviações de cada modelo implementado durante os experimentos.

Quadro 5 – Abreviações dos modelos preditores utilizados

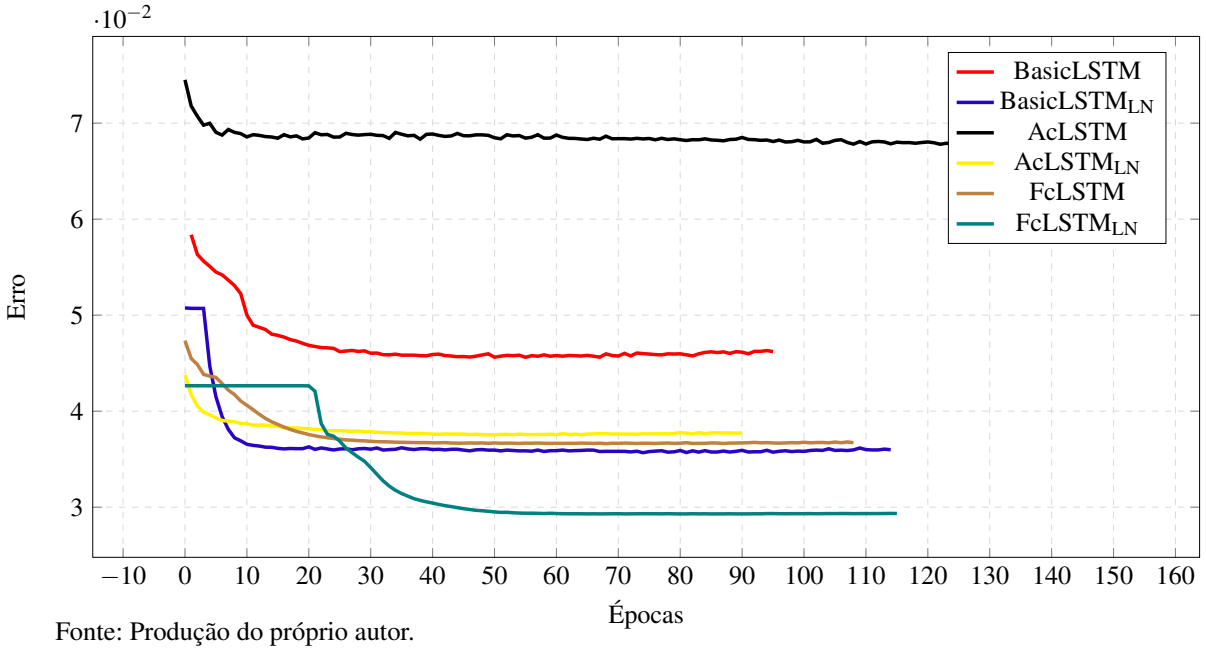
Abreviação	Arquitetura
BasicLSTM	LSTM básica
FcLSTM	LSTM totalmente conectada
AcLSTM	LSTM totalmente conectada achatada
$\text{BasicLSTM}_{\text{LN}}$	LSTM básica com *Layer Normalization*
$\text{FcLSTM}_{\text{LN}}$	LSTM totalmente conectada com *Layer Normalization*
$\text{AcLSTM}_{\text{LN}}$	LSTM totalmente conectada achatada com *Layer Normalization*

Fonte: Produção do próprio autor.

4.4.3 Primeiro Experimento

A metodologia desta etapa consiste em treinar todos os modelos preditores indicados no Quadro 5, considerando um mesmo tamanho de janela, especificamente de 64 amostras. O Gráfico 11 mostra os resultados da validação durante o treinamento. Na figura, os resultados do valor de erro MSE de validação obtido ao longo do treinamento de diversos modelos, note que a legenda esta contida na imagem. Pode ser observado que os modelos tiveram diferentes durações de treinamento, isto ocorre devido ao mecanismo de parada antecipada (do inglês, *Early Stopping*) onde o treinamento foi interrompido caso o erro de validação estagnasse ou aumentasse.

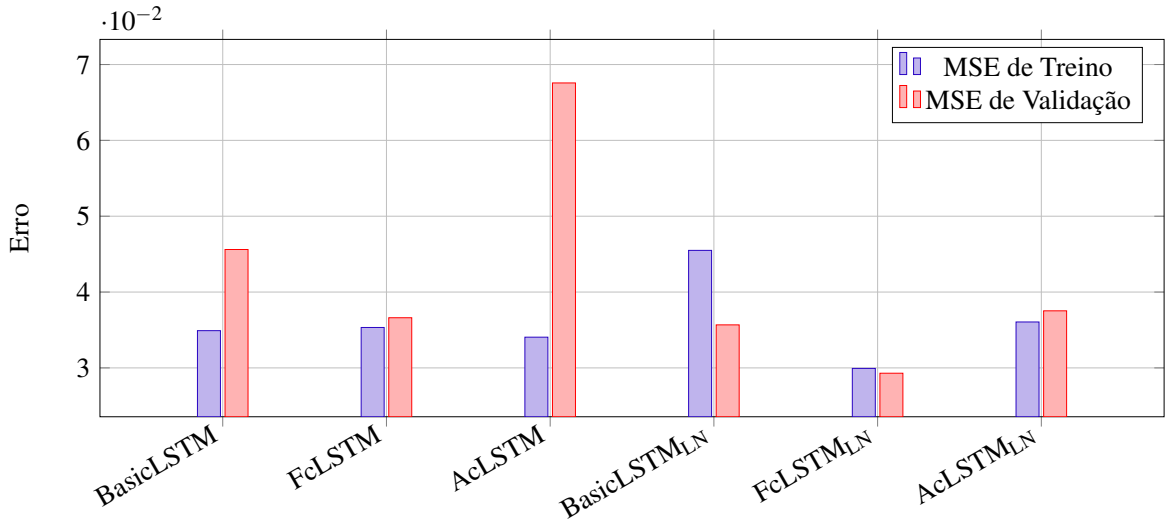
Gráfico 11 – Erro de validação durante treinamento



Fonte: Produção do próprio autor.

O Quadro 6 e o Gráfico 12 resumem o resultado do treinamento de todos os modelos sobre os conjuntos de treino e validação. No Quadro, os erros de treino e validação das diferentes arquiteturas treinadas para o tamanho da janela fixado. Em destaque, tem se o menor erro de validação e treino onde o erro de validação é utilizado como métrica para julgar performance entre as arquiteturas. Os resultados estão ordenados em ordem tal que o último erro de validação seja o melhor já que quanto menor o erro, melhor o modelo. Observou-se que o modelo com melhores resultados foi o modelo multicamada LSTM totalmente conectado com *Layer Normalization*, isto é, o modelo FcLSTM$_{LN}$.

Gráfico 12 – Erro mínimo de treino e validação para diferentes arquiteturas



Fonte: Produção do próprio autor.

Quadro 6 – Erro mínimo de treino e validação para diferentes arquiteturas

Modelo	Erro treino	Erro validação
AcLSTM	$0,034051$	$0,067572$
BasicLSTM	$0,034910$	$0,045611$
AcLSTM$_{LN}$	$0,036055$	$0,037525$
FcLSTM	$0,035330$	$0,036614$
BasicLSTM$_{LN}$	$0,045495$	$0,035673$
FcLSTM$_{LN}$	$\mathbf{0,029932}$	$\mathbf{0,029298}$

Fonte: Produção do próprio autor.

4.4.4 Segundo Experimento

Considerando-se unicamente como modelo preditor a arquitetura FcLSTM$_{LN}$, passou-se a analisar o tamanho de janela deslizante a ser utilizada para o treinamento deste modelo. Note que uma janela grande não será necessariamente capaz de gerar melhores resultados que uma janela menor. Isto pode ser analisado como a importância de observações passadas sobre dados futuros, ou seja, muitas observações no passado podem criar uma janela com características irrelevantes à observações no futuro. Da mesma forma, poucas observações podem ferir a capacidade do modelo em criar predições já que menos características são passadas e portanto, características essenciais podem ser perdidas.

Uma ressalva vale ser feita em relação ao tamanho da janela, o sistema proposto visa operar em uma janela de quinze a trinta minutos, isto é, cada uma das amostras dos dados de uma janela, representa um intervalo de quinze a trinta minutos em leitura. Como demonstra o Quadro 7, devido ao período diário de funcionamento de 7 horas, uma janela menor que 14 amostras apenas contém movimentações de um dia de negociações, assim como uma janela maior que 140 amostras contém informações de uma semana passada. No Quadro, mostra-se o número de dados que podem ser coletados de quaisquer ativos na bolsa de valores brasileira em um dia e uma semana, isto se dá pelo horário de funcionamento da bolsa, de 10 horas até as 17 horas. Foi definido testar três possíveis valores de janela, onde cada valor de janela é maior que os dados de uma dia e menor que os dados de uma semana, sendo estes valores: 32, 64 e 128.

Quadro 7 – Número de amostras de dados OHLC

Taxa de amostragem	Amostras no dia	Amostras na semana
30 minutos	14 amostras	70 amostras
15 minutos	28 amostras	140 amostras

Fonte: Produção do próprio autor.

Portanto, utilizando a arquitetura $\text{FcLSTM}_{\text{LN}}$, treina-se três instâncias dela, uma para cada tamanho de janela especificada. Com isso, foram obtidos os resultados resumidos no Quadro 8, evidenciando-se que, para o modelo, o tamanho de janela que gera o melhor resultado é a de 128 amostras. No Quadro, os erros de treino e validação da melhor arquitetura treinada para diferentes tamanhos de janela. Em destaque, tem se o menor erro de validação e treino onde o erro de validação é utilizado como métrica para julgar performance entre as janelas. Os resultados estão ordenados em ordem tal que o último erro de validação seja o melhor já que quanto menor o erro, melhor o modelo.

Quadro 8 – Erro mínimo de treino e validação para diferentes janelas

Tamanho Janela	Erro treino	Erro validação
32	$0,066814$	$0,065431$
64	$0,029932$	$0,029298$
128	$\mathbf{0,022827}$	$\mathbf{0,022828}$

Fonte: Produção do próprio autor.

Finalmente, como resultado dos dois primeiros experimentos, o modelo preditor selecionado foi a arquitetura $\text{FcLSTM}_{\text{LN}}$ com seus hiper-parâmetros definidos no Quadro 9. Dos dois primeiros

experimentos, notou-se que, para o problema em pauta, a variação no tamanho da janela dos dados foi o fator de maior impacto sobre a desempenho do modelo em gerar predições corretas. Isto é, o erro MSE na validação foi mais sensível a variação do tamanho da janela.

Quadro 9 – Hiper-parâmetros do modelo preditor

Parâmetro	Valor
Tamanho de janela	128
Número de camadas	6
Dropout%	25%
Tipo	$\text{FcLSTM}_{\text{LN}}$

Fonte: Produção do próprio autor.

4.4.5 Terceiro Experimento

Neste experimento, buscou-se treinar o modelo composto e avaliar a influência do uso de um *autoencoder* LSTM para reduzir a representação dos dados através do espaço latente. A estrutura do modelo composto foi: três *encoders*, cujas saídas são concatenadas e passadas para o modelo preditor, tanto o *encoder* do autoencoder LSTM como o modelo preditor são modelos $\text{FcLSTM}_{\text{LN}}$, com seus hiper-parâmetros indicados no Quadro 10.

Como comentário sobre o tamanho da janela do autoencoder LSTM, o objetivo do *autoencoder* é buscar características relevantes para utilizar seu espaço latente em um modelo preditor, portanto, não é de interesse utilizar o tamanho da janela do *autoencoder* LSTM pela mesma lógica do modelo preditor. Ao utilizar uma janela de 128, o modelo composto passará a analisar uma janela necessariamente maior e múltipla de 128, isto é, o modelo passará a observar informações fora do tamanho de uma semana de dados. Para tanto, optou-se por criar um *autoencoder* LSTM com uma janela de tamanho 64 para construir um modelo composto com uma janela de 192 amostras. Desta forma, o modelo composto passará a ter uma janela de 192 amostras, onde os *encoders* do *autoencoder* LSTM irão reduzir esta entrada para 3 amostras que serão as entradas para o modelo preditor, as características dos modelos encontram-se no Quadro 10.

Como comentário sobre o treinamento do *autoencoder* LSTM, para o treinamento, define-se um *encoder* com tamanho de janela deslizante de 64 e um *decoder* com uma janela de tamanho 6. A entrada do *decoder* é invertida para que relações próximas no tempo possam ser exploradas. A

cada iteração, o *encoder* passa seu estado latente para o *decoder*. Com o modelo treinado, não há mais motivos para usar *decoder*, apenas há interesse no espaço latente do *autoencoder* uma vez que o mesmo será utilizado como entrada para o modelo preditor.

Como comentário sobre o operação de *TimeDistributed*, para o uso do *encoder* no modelo preditor, congelou-se os pesos do *encoder* e o replicou três vezes. Isto é, para cada janela com tamanho 192, aplicou-se três *encoders* com seus pesos congelados, então as saídas de cada *encoders* foram concatenadas e, só assim, alimentadas ao modelo preditor. Desta forma, força-se que o modelo preditor espere o tempo necessário para a obtenção do resultado dos três *encoders* anteriores a ele, esta operação é denominada de *TimeDistributed*. Nota-se que é necessário congelar os pesos do modelo para que não ocorra atualização de seus parâmetros.

Quadro 10 – Hiper-parâmetros dos modelos utilizados na construção do modelo composto

Parâmetro	*Encoder*	Preditor
Número de camadas	4	6
Tamanho da Janela	64	3
Dropout%	25%	25%
Arquitetura	$\text{FcLSTM}_{\text{LN}}$	$\text{FcLSTM}_{\text{LN}}$
Número de características	15	4

Fonte: Produção do próprio autor.

O Quadro 11 mostra os resultados do modelo composto versus o modelo preditor. Percebe-se que o modelo composto não foi capaz de criar predições melhores que o melhor preditor, o que significa que o *autoencoder* LSTM não foi capaz de apropriadamente reduzir em seu espaço latente informações úteis para a reconstrução dos dados. No quadro, os erros de treino e validação do modelo preditor determinado em experimentos anteriores e o modelo composto, tem se o menor erro de validação e treino onde o erro de validação é utilizado como métrica para julgar a acurácia entre as arquiteturas. Os resultados estão ordenados em ordem tal que o último erro de validação seja o melhor já que quanto menor o erro, melhor o modelo.

Além disso, percebeu-se dos experimentos que alterações no tamanho da janela ou até mesmo em profundidade não foram capazes obter um modelo que pudesse convergir a um erro menor.

Quadro 11 – Erro mínimo de treino e validação entre modelo preditor e composto

Modelo	Erro treino	Erro validação
Modelo Composto	$0,031460$	$0,039667$
Modelo Preditor	$\mathbf{0,029932}$	$\mathbf{0,029298}$

Fonte: Produção do próprio autor.

4.4.6 Quarto experimento

Neste experimento, procurou-se comparar diferentes arquiteturas e diferentes técnicas de pré-processamento a fim de avaliar a metodologia de pré-processamento e os modelos propostos.

Em relação às técnicas de pré-processamento, foram treinados modelos com *Norm* global ou sem o operador *Detrend*, isto é: (*i*) todos os dados foram normalizados antes de serem divididos em janelas usadas no treinamento, portanto, todos os dados foram normalizados de acordo com o valor máximo e mínimo de todo o conjunto de dados e (*ii*) os dados não passaram pelo processo de remoção de tendência. Para facilitar a leitura dos resultados, o Quadro 12 mostra as abreviações dos diferentes processamentos avaliados.

Quadro 12 – Abreviações dos tratamentos utilizados

Abreviação	Procedimento
Normal	Procedimento como definido
Global	Operador *Norm* antes da divisão em janela
Trend	Sem operador *Detrend*
Close	Apenas preço de fechamento como característica

Fonte: Produção do próprio autor.

Em relação à comparação com técnicas clássicas da literatura, foram utilizadas as técnicas de: regressão ingênua[1], regressão *Ridge*, Média Móvel Integrada Autorregressiva Sazonal (SARIMAX, do inglês *Seasonal Autoregressive Integrated Moving Average*), modelo *Perceptron* multicamada e regressão por árvore de decisão (denominado *DecisionTree*). Todas as técnicas testadas tiveram como entrada uma janela de tamanho 128, assim como os modelos LSTM, e

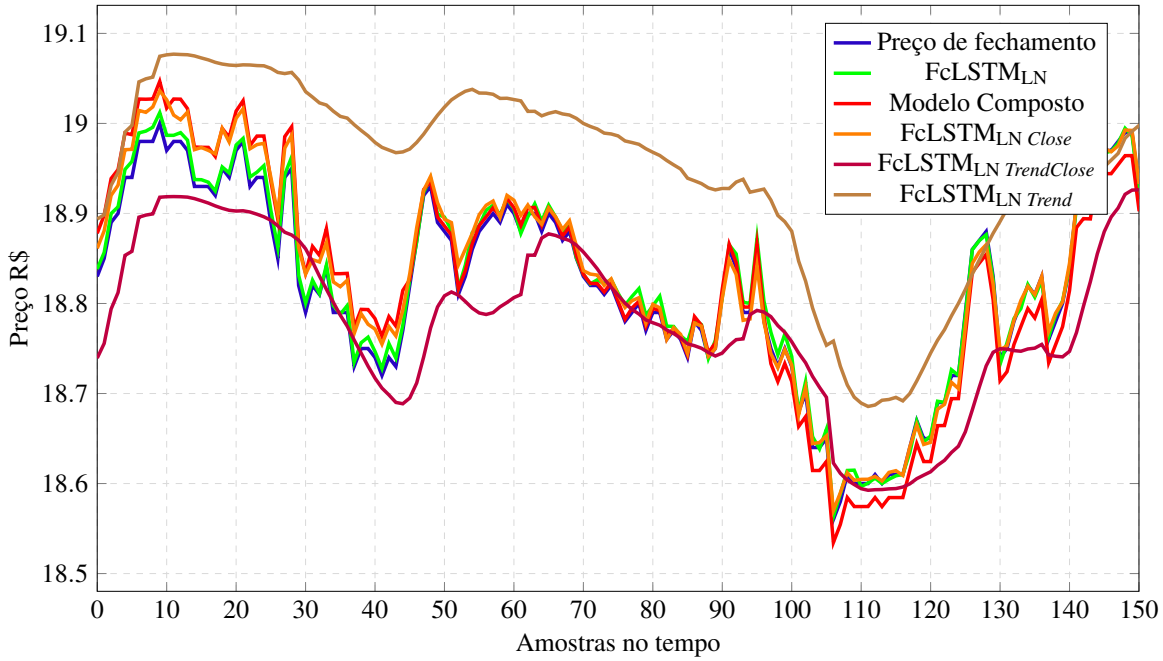
[1] Na técnica chamada de regressão ingênua (do inglês, *naive regression*) é suposto que todo valor futuro é igual ao valor observado no passado imediato, assim como demonstra a seguinte relação $\hat{y}[t+1] = y[t]$.

seus hiper-parâmetros ótimos encontrados via o algoritmo *GridSearch*[2], onde encontra-se os melhores parâmetros via força bruta.

Aqui, foi utilizado um conjunto de dados de teste com ativos da B3. Especificamente, os ativos selecionados do mercado brasileiro foram: Usiminas (USIM5), Petrobras (PETR4), Oi (OIBR3), Ambev (ABEV3), Gerdau (GOAU4) e Marfrig (MRFG3). O resultado da análise é resumido através da média das métricas MdAE e MSE obtidas nos ativos indicados.

No tocante às técnicas de processamento, os resultados desta etapa de avaliação ficam resumidos nos Gráficos 13 e 14 e nos Quadros 13 e 14. As figuras comparam as predições dos diferentes modelos em relação a linha de referência denominada preço de fechamento. Já os quadros comparam os resultados médias de cada modelo através das métricas utilizadas. Nota-se que a metodologia proposta é válida ao passo que obteve menor erro MSE e MdAE quando comparada aos variantes. Outro detalhe observado é o impacto do comportamento não-estacionário dos dados. detalhe percebido ao comparar os resultados dos modelos onde se aplica o operador *Detrend* e onde não se aplica.

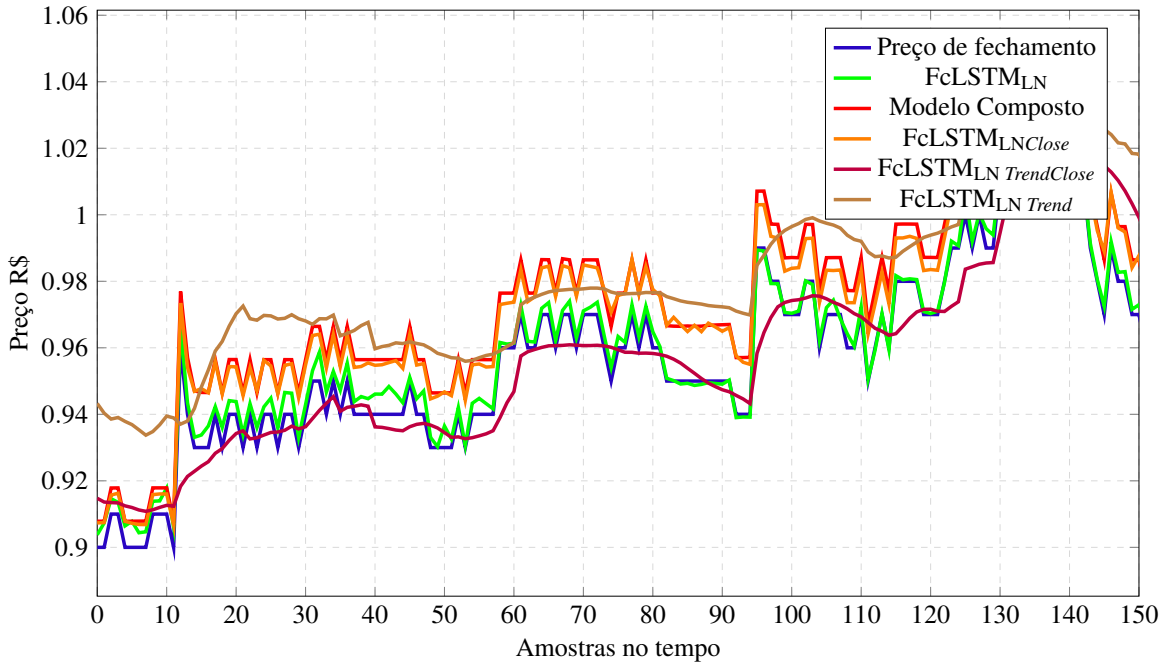
Gráfico 13 – Comparação de predições entre modelos com o ativo ABEV3



Fonte: Produção do próprio autor.

[2] Para o *GridSearch foi utilizado a biblioteca scikit-learn em Python onde o processo de busca de parâmetros ótimos é automatizado.*

Gráfico 14 – Comparação de predições entre modelos com o ativo OIBR3



Fonte: Produção do próprio autor.

Quadro 13 – Erro MdAE para diferentes modelos e diferentes ações

Erro (10^{-3})	FcLSTM$_{LN}$	FcLSTM$_{LN}$ *Global*	FcLSTM$_{LN}$ *Trend*	FcLSTM$_{LN}$ *TrendClose*	FcLSTM$_{LN}$ *Close*	Modelo Composto
ABEV3	5,85	6,19	147,96	52,11	10,24	25,77
GOAU4	6,06	33,81	122,92	78,98	10,93	3,49
MRFG3	13,80	131,29	200,27	65,89	42,13	56,26
OIBR3	2,24	27,33	23,52	5,73	13,77	16,44
PETR4	18,48	222,58	576,30	198,71	75,37	73,63
USIM5	6,23	77,06	131,97	40,13	36,09	29,42
Média	**8,78**	83,04	200,49	73,59	31,42	34,17

Fonte: Produção do próprio autor.

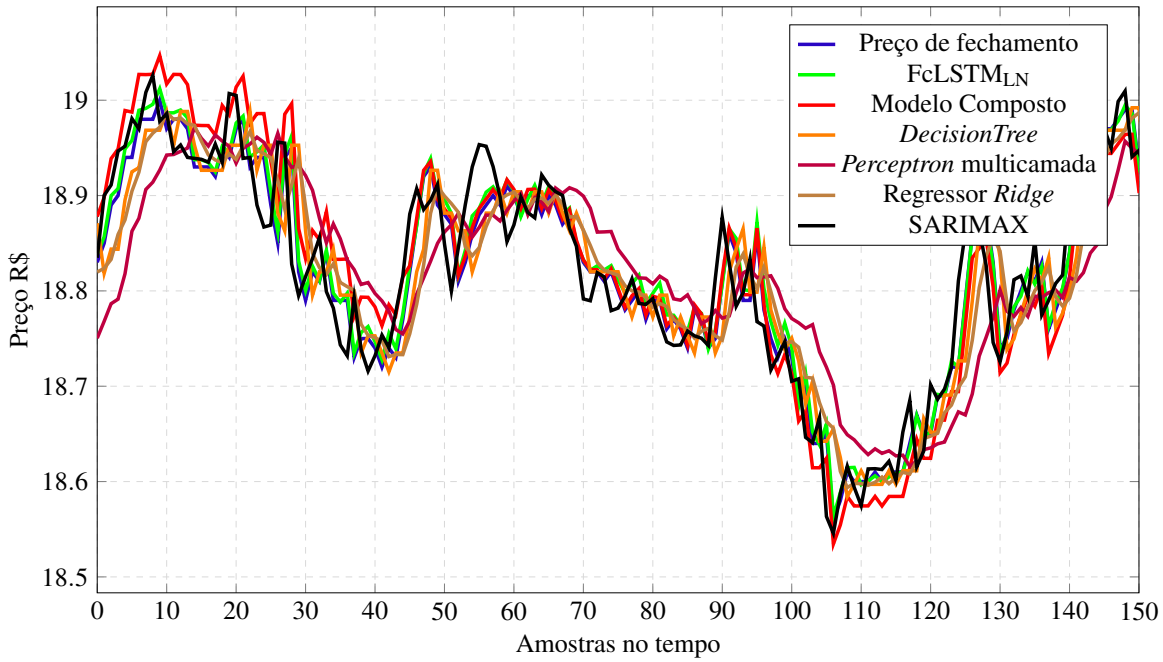
Quadro 14 – Erro MSE para diferentes modelos e diferentes ações

Erro (10^{-3})	FcLSTM$_{LN}$	FcLSTM$_{LN}$ *Global*	FcLSTM$_{LN}$ *Trend*	FcLSTM$_{LN}$ *TrendClose*	FcLSTM$_{LN}$ *Close*	Modelo Composto
ABEV3	0,07	2,33	28,10	5,36	0,38	0,95
GOAU4	0,06	0,87	19,40	9,77	0,24	0,05
MRFG3	0,37	13,48	53,00	9,48	2,59	3,62
OIBR3	0,01	0,67	0,73	0,13	0,19	0,24
PETR4	0,88	40,09	444,33	65,63	14,45	5,59
USIM5	0,09	4,04	46,66	3,40	1,40	0,89
Média	**0,25**	10,25	98,70	15,63	3,21	1,89

Fonte: Produção do próprio autor.

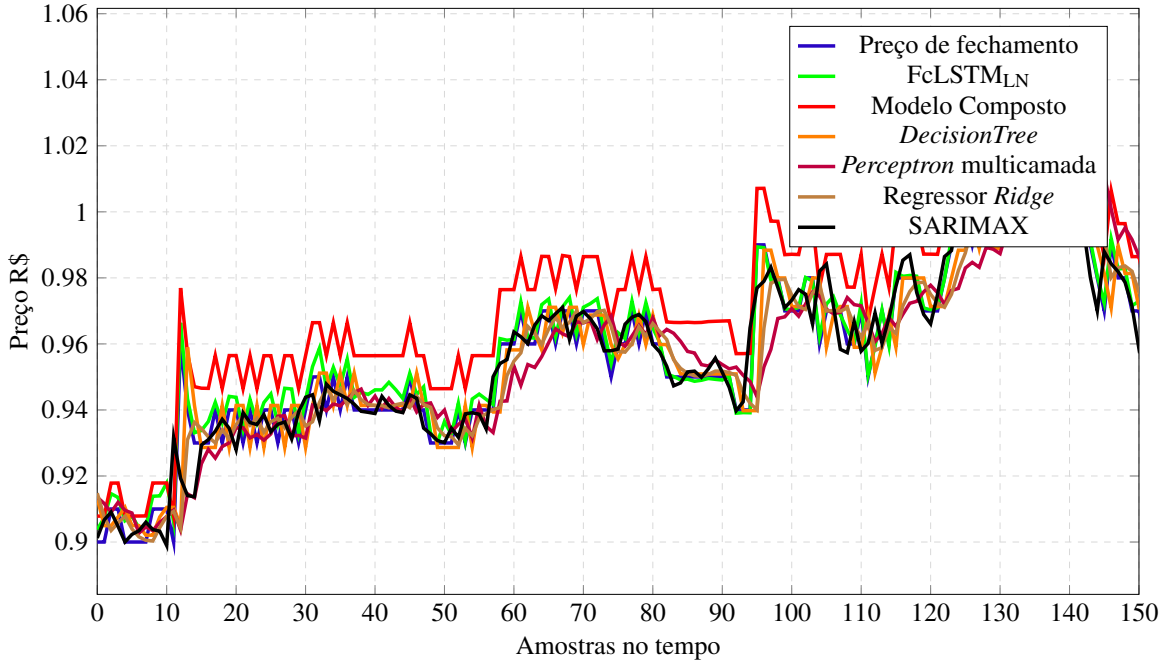
No que diz respeito à comparação com técnicas clássicas da literatura, os resultados desta etapa de avaliação ficam resumidos nos Gráficos 15 e 16 e nos Quadros 15 e 16. As figuras comparam as predições dos diferentes modelos em relação a linha de referência denominada preço de fechamento. Já os quadros comparam os resultados médias de cada modelo através das métricas utilizadas. Nota-se que a metodologia proposta é válida ao passo que obteve menor erro MSE e MdAE quando comparada aos variantes. Observa-se que o modelo preditor proposto (FcLSTM$_{LN}$) obtém melhores resultados que os métodos clássicos. Entretanto, nota-se que o modelo composto não obteve resultados satisfatórios, uma vez que outros métodos clássicos de regressão obtiveram melhores resultados.

Gráfico 15 – Comparação de predições entre LSTM e regressores clássicos com o ativo ABEV3



Fonte: Produção do próprio autor.

Gráfico 16 – Comparação de predições entre LSTM e regressores clássicos com o ativo OIBR3



Fonte: Produção do próprio autor.

Quadro 15 – Erro MdAE para diferentes modelos clássicos em ações

Erro (10^{-3})	SARIMAX	Naive	Ridge	MLP	DecisionTree	$FcLSTM_{LN}$	Modelo Composto
ABEV3	25,90	20,00	22,80	42,44	22,55	5,85	25,77
GOAU4	22,00	20,00	20,46	32,62	18,68	6,06	3,49
MRFG3	35,27	30,00	37,82	57,33	36,64	13,80	56,26
OIBR3	5,14	10,00	5,38	7,63	8,07	2,24	16,44
PETR4	84,00	70,00	85,07	147,39	70,92	18,48	73,63
USIM5	25,00	20,00	22,56	43,79	21,15	6,23	29,42
Média	32,88	28,33	32,35	55,20	29,67	**8,78**	34,1

Fonte: Produção do próprio autor.

Quadro 16 – Erro MdAE para diferentes modelos clássicos em ações

Erro (10^{-3})	SARIMAX	Naive	Ridge	MLP	DecisionTree	$FcLSTM_{LN}$	Modelo Composto
ABEV3	1,82	1,37	1,89	4,25	1,51	0,07	0,38
GOAU4	1,35	1,17	1,51	3,49	1,30	0,06	0,24
MRFG3	4,13	3,79	4,75	9,73	4,63	0,37	2,59
OIBR3	0,07	0,10	0,10	0,15	0,10	0,01	0,19
PETR4	30,70	29,38	31,98	56,71	30,17	0,88	14,45
USIM5	1,97	1,46	2,01	5,17	1,57	0,09	1,40
Média	6,67	6,21	7,04	13,25	6,55	**0,25**	3,21

Fonte: Produção do próprio autor.

4.4.7 Comparação de resultados via *Backtesting*

Em relação aos parâmetros da estratégia *Trail Stop Loss*, assim como utilizado no quarto experimento, foram utilizados os ativos do mercado brasileiro USIM5, PETR4, OIBR3, ABEV3, GOAU4 e MRFG3 a fim de determinar os melhores parâmetros da estratégia *Trail Stop Loss*, resumidos na Tabela 17.

Quadro 17 – Parâmetros otimizados da estratégia *Trail Stop Loss*

Parâmetro	Valor ótimo
Acréscimo	$2,5\%$
Bias	$7,5\%$
$Stop_{\text{Loss}}$	5%
$Stop_{\text{Gain}}$	50%

Fonte: Produção do próprio autor.

Com relação à execução do *Backtesting* com a estratégia *Trail Stop Loss* sintonizado com o regressor ideal, os resultados obtidos são referentes a um mês de execução de *backtesting* no período entre outubro de 2019 e novembro de 2019. Neste teste com o regressor ideal, foram executadas 31 operações de compra e 39 operações de venda, com isto, tem se 69 posições tomadas durante o período de 20 dias o que leva a uma média de 3.45 operações$/dia$.

O Quadro 18 demonstra que variação percentual de lucro depende da taxa de comissão utilizada. Isto se dá pelo fato de 69 posições terem sido tomadas, é preferível tomar poucas posições grandes ao invés de muitas posições pequenas devido à exposição ao risco e taxas associadas a cada operação. No mercado financeiro brasileira, há corretoras que oferecem contas com taxa de corretagem zero, isto é, o usuário opera sem ser taxado por operação. Alterando os parâmetros de *backtesting* para simular este tipo de corretora, pode-se testar as estratégias em *backtesting* com menos taxas por operação, o que torna viável sistemas de *Algorithmic trading* com muito posicionamento. Para tanto, as comparações e as simulações que seguem são executadas com taxa de corretagem zero.

Quadro 18 – Diferença em lucro devido à taxa de corretagem

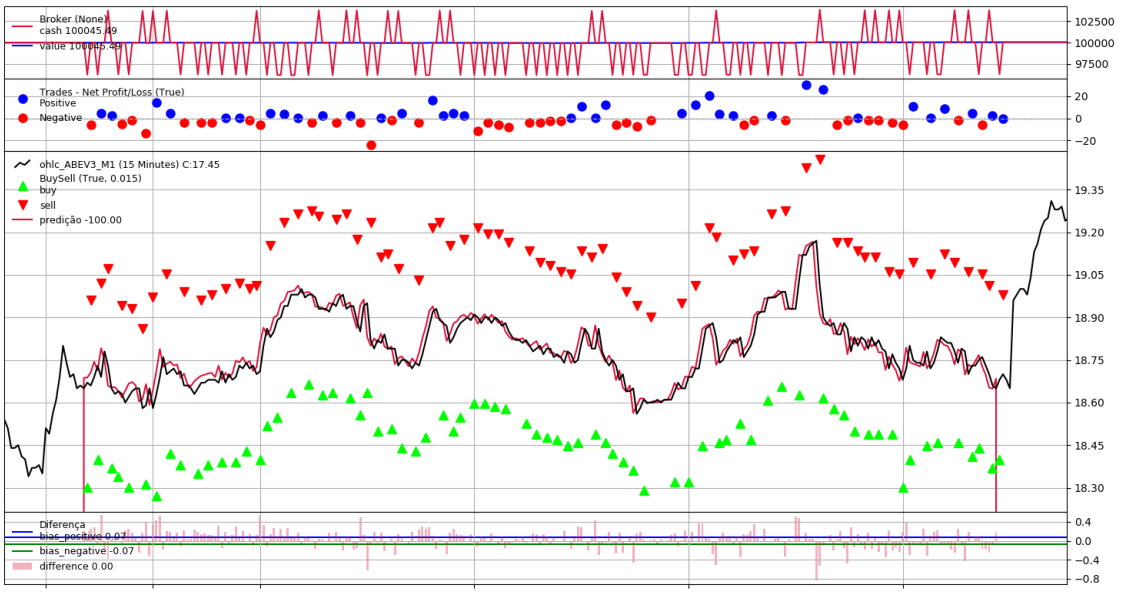
Tipo de operação	Número de posições	Lucro%	Ações por posição
Com comissão	69	$-0,583\%$	200
Sem comissão	69	$+0,131\%$	200

Fonte: Produção do próprio autor.

Em relação a execução do *Backtesting* com a estratégia *Trail Stop Loss* e os modelos predito e composto, utilizando o modelo preditor e o modelo composto executou-se o *backtesting*, obtendo-se o regime de operação para ambos modelos como demonstrado nos Gráficos 17 e 18, respectivamente. A figura demonstra o comportamento da estratégia de *algorithmic trading*

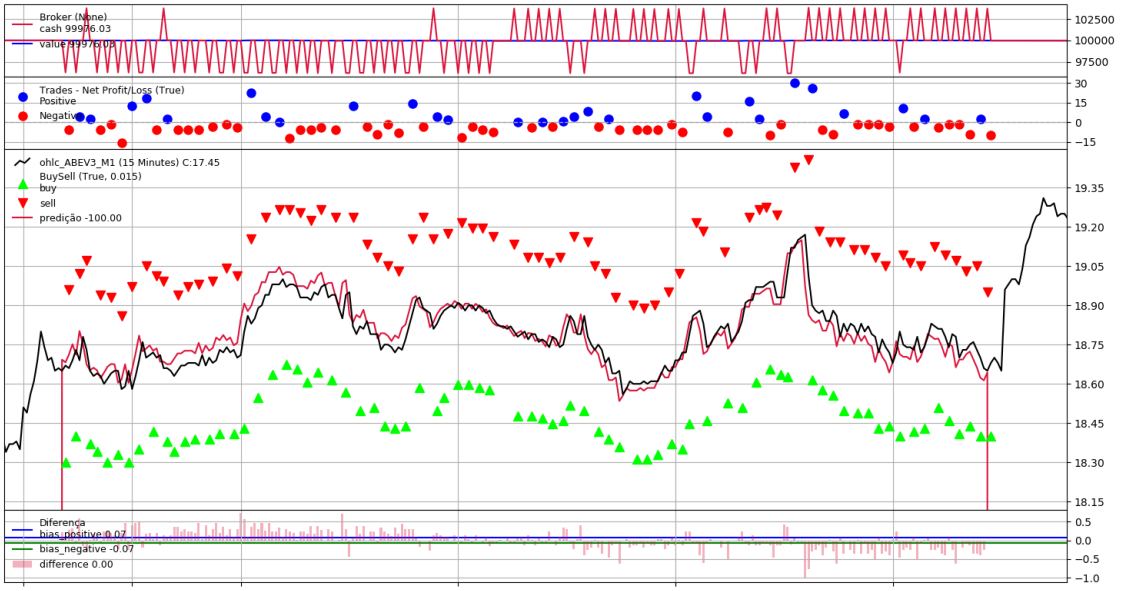
com os modelos atuando na ação da empresa Ambev, nota-se pelo número de bolas azuis e vermelhas que ambos atuam em excesso. Para todas as execuções de *backtesting* obtém-se um resumo gráfico das decisões tomadas na estratégia, onde: (*i*) o preço de fechamento da ação é representado pela linha preta; (*ii*) a predição feita pelo modelo é representado pela linha vermelha. Além disto, abaixo do gráfico é exposto o valor da variável *Diferença%* como mencionado na Proposta. Somado a isto, a cada posição tomada pelo modelo, graficamente tem-se: (*i*) triângulos vermelhos para os momentos em que ações foram vendidas; (*ii*) triângulos verdes para os momentos ações foram comparadas; (*iii*) bolas azuis para as operações onde obteve-se lucro e (*iv*) bolas vermelhas para as operações onde obteve-se prejuízo. Nota-se que os modelos operam em demasiado, idealmente, o modelo operaria uma vez, ou seja, comprando no valor mínimo e vendendo no valor máximo. O número de posições tomadas levou a redução do lucro devido a taxações mesmo após a retirada da taxa de corretagem, entretanto, ainda foi possível obter lucro.

Gráfico 17 – Análise de operação do modelo preditor no ativo ABEV3



Fonte: Produção do próprio autor.

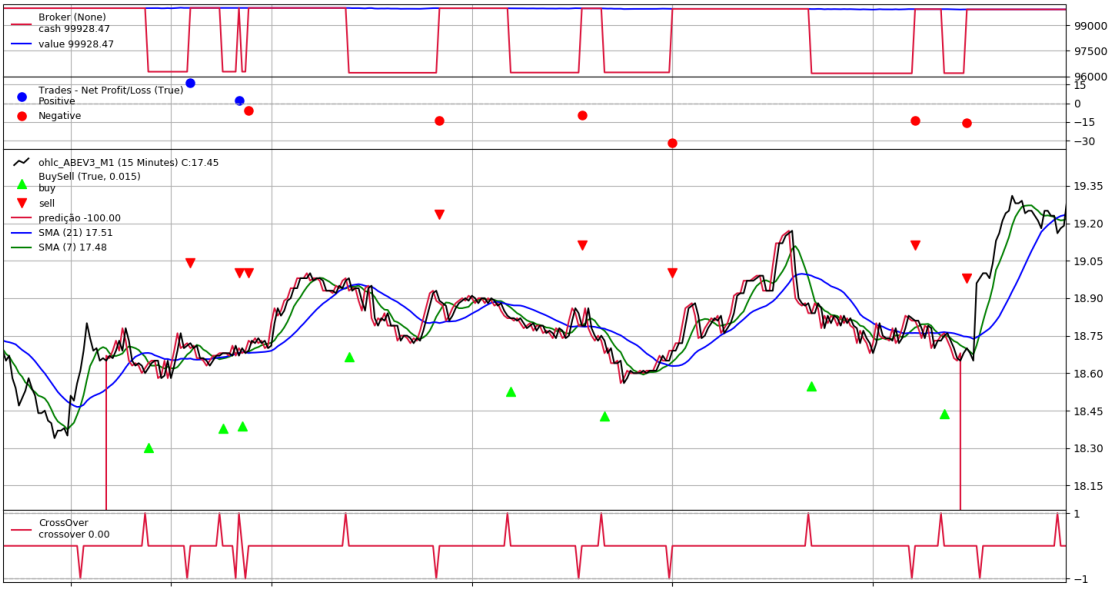
Gráfico 18 – Análise de operação do modelo composto no ativo ABEV3



Fonte: Produção do próprio autor.

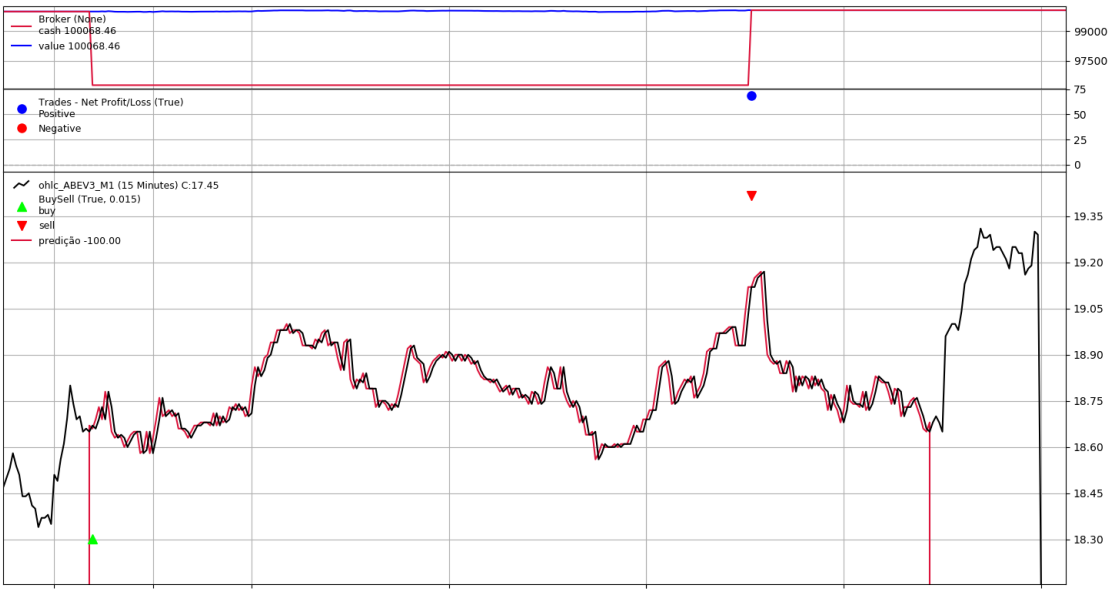
Ao que se refere à execução do *Backtesting* com outras estratégias, utilizando a mesma ferramenta para a execução do *backtesting*, cria-se as estratégias de Cruzamento de Médias Móveis, onde optou-se por usar médias móveis de período sete e vinte e um e a estratégia de *Buy and Hold* a qual não há parâmetros a serem definidos uma vez que a estratégia consiste apenas em comprar o ativo e deixar valorizar até um valor máximo. Desta forma, obtém-se um regime de operação como demonstrado nos Gráficos 19 para a estratégia de *Buy and Hold* e 20 para a estratégia de Cruzamento de Médias Móveis. Para todas as execuções de *backtesting* obtém-se um resumo gráfico das decisões tomadas na estratégia, onde: (*i*) o preço de fechamento da ação é representado pela linha preta; (*ii*) a linha vermelha representa a predição do modelo regressor; (*iii*) a média móvel longa, de período 21, é representado pela linha azul; (*iv*) a média móvel longa, de período 21, é representado pela linha verde; Como mencionado na execução do *backtesting* comos modelos LSTM, abaixo do gráfico é exposto o valor da variável *Diferença%* e para cada posição tomada pelo modelo, tem-se: (*i*) triângulos vermelhos para os momentos em que ações foram vendidas; (*ii*) triângulos verdes para os momentos ações foram comparadas; (*iii*) bolas azuis para as operações onde obteve-se lucro e (*iv*) bolas vermelhas para as operações onde obteve-se prejuízo.

Gráfico 19 – Análise de operação com cruzamento de médias móveis no ativo ABEV3



Fonte: Produção do próprio autor.

Gráfico 20 – Análise de operação com *Buy and Hold* no ativo ABEV3



Fonte: Produção do próprio autor.

No quesito de comparação entre as estratégias utilizadas no *Backtesting*, como feito no quarto experimento, aqui, executa-se o processo de *backtesting* em um conjunto de ações a fim de determinar o lucro porcentual médio de atuação de cada estratégia e quantas posições foram tomadas por estratégia. Através dos resultados sumarizados na Tabela 1, percebe-se que para uma janela de tempo curta, o sistema de *algorithmic trading* com o uso de redes LSTM e estratégia de *Trail Stop Loss* foi capaz de obter lucro médio maior que outras estratégias mesmo que tomando

mais posições no mercado financeiro.

Tabela 1 – Lucro e número de operações em diferentes ações

	Modelo Preditor		Modelo Composto		*Buy and Hold*		Cruzamento de Médias Móveis	
Ações	Lucro(%)	$N_{Posições}$	Lucro(%)	$N_{Posições}$	Lucro(%)	$N_{Posições}$	Lucro(%)	$N_{Posições}$
ABEV3	$+0,0455\%$	74	$-0,0240\%$	77	$+0,0682\%$	1	$-0,0721\%$	8
GOAU4	$+0,1838\%$	80	$+0,1701\%$	79	$+0,1833\%$	1	$+0,0710\%$	6
MRFG3	$+0,3158\%$	77	$+0,0918\%$	82	$-0,2600\%$	1	$-0,0920\%$	7
OIBR3	$-0,0063\%$	87	$-0,0251\%$	88	$+0,0123\%$	1	$+0,0110\%$	6
PETR4	$+1,1281\%$	76	$+0,4643\%$	79	$+0,2722\%$	1	$-0,1651\%$	5
USIM5	$+0,2978\%$	81	$+0,1837\%$	83	$+0,0412\%$	1	$-0,0800\%$	7
Média	$+\mathbf{0,32745}\%$	79,16	$+0,143466\%$	81,32	$+0,0526\%$	1	$-0,054516\%$	6,5

Fonte: Produção do próprio autor.

4.5 Resumo

Deste capítulo pode-se concluir que: Através do treinamento e da escolha de bons modelos treinados, é possível obter lucro atuando no mercado financeiro através de *algorithmic trading* com o uso de redes neurais LSTM, mesmo operando com *spread* e *slippage*. Nota-se que o uso do método de pré-processamento resultou em melhora dos resultados. Além disso, observa-se que através de algoritmos consegue-se efetuar o *backtesting* com diferentes estratégias com diferentes parâmetros facilmente, sendo possível criar comparações cabíveis sobre o desempenho do modelo e estratégias utilizadas no mercado financeiro. Nota-se embora tenha sido obtido lucro percentual. Há muitas compras e vendas feitas pelo modelo, idealmente, o modelo operaria comprando no valor mínimo e vendendo no valor máximo. O que segue deste trabalho são as conclusões sobre o desenvolvimento do trabalho.

5 CONCLUSÕES

O objetivo principal deste trabalho foi propor um sistema de *Algorithmic trading* utilizando redes neurais LSTM. A técnica proposta é baseada fundamentalmente na habilidade de redes LSTM de criarem dependências temporais em sua estrutura. Junto a estas redes, através da formalização de um processo de tratamento dos dados e o uso de bibliotecas de ML da atualidade, cria-se um sistema capaz de operar automaticamente na bolsa de valores sendo capaz de se ajustar a dinamicidade do mercado.

Como demonstrado na seção anterior, foi possível criar um sistema de *algorithmic trading* capaz de obter lucro com o uso da estratégia de *Trail Stop Loss* somado ao uso de redes LSTM para gerar predições sobre os valores futuros. Além disso, foi possível demonstrar que a metodologia de processamento de dados é válida ao passo que modelos treinados com outras formas de processamento de dados obtiveram resultados inferiores ao modelos aqui propostos quando avaliados em relação às métricas estabelecidas.

A partir dos valores das métricas obtidas ao longo dos testes realizados, observou-se que, contrário ao que era esperado ao desenvolver a pesquisa, o modelo composto, definido por um *autoencoder* LSTM como forma de processamento dos dados, não levou à melhores resultados quando comparados com o modelo $\text{FcLSTM}_{\text{LN}}$. Este resultado pode ser visto como a incapacidade do modelo composto, com as características utilizadas, de reduzir as representações temporais dos dados em seu espaço latente. Além disso, ao se analisar os resultados e comparar as métricas de modelos com a metodologia proposta e outras metodologias utilizadas, percebe-se um aumento no desempenho. Isto pode ser visto como a importância e validade de utilizar indicadores técnicos como ferramentas de extração de características.

Outra forma de interpretar esta informação está no próprio uso destas ferramentas por indivíduos que operam na bolsa de valores. Isto é, os indicadores utilizados como forma de extração de características são as mesmas ferramentas utilizadas para guiar a decisão de indivíduos que atuam na bolsa de valores. Portanto, há a possibilidade de que os indicadores em si, para uma janela pequena de tempo, sejam dados mais relevantes que os preços, uma vez que a variação do valor de uma ação é composta por diferenças entre forças compradoras e forças vendedoras, que

por sua vez, possam estar com sua decisão parcialmente influenciada pelos indicadores usados no projeto.

Devido a divisão do projeto em microsserviços e o uso de protocolos de comunicação amplamente utilizados pela internet, torna-se possível construir ou até mesmo alugar um servidor e hospedar os modelos, sendo eles acessados através de requisições GET de qualquer lugar do mundo. Além disso, o trabalho buscou construir estruturas que visam a facilitar a construção, treinamento e testes de diferentes modelos em problemas de regressão em séries temporais onde há uma ou mais que uma série temporal a ser considerada como entrada.

Como trabalhos futuros: serão pesquisadas outras técnicas e outros modelos a serem construídos e testados. Desta forma, busca-se outras redes neurais com maior robustez para avaliar dados temporais. Futuramente, pretende-se explorar mecanismos de atenção e técnicas de *ensemble* onde explora-se a utilização de um conjunto de modelos juntos para gerar um único modelo com desempenho superior aos modelos individuais.

REFERÊNCIAS BIBLIOGRÁFICAS

ALPHABET. **Colaboratory Documentação**. 2019a. Disponível em: https://colab.research.google.com/notebooks/welcome.ipynb. Acesso em: 9 jun. 2019.

ALPHABET. **Kaggle Documentação**. 2019b. Disponível em: https://www.kaggle.com/docs. Acesso em: 8 abr. 2019.

B3. **Boletim Diário do Mercado**. 2019. Disponível em: http://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/consultas/boletim-diario/boletim-diario-do-mercado. Acesso em: 7 jun. 2019.

BA, J.; KIROS, J. R.; HINTON, G. **Layer Normalization**. 2016. Disponível em: https://davidstutz.de/layer-normalization-ba-et-al. Acesso em: 3 abr. 2019.

BAO, W.; YUE, J.; RAO, Y. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. **PLoS ONE**, v. 12, p. 1–24, jul. 2017.

BHANJA, S.; DAS, A. **Impact of Data Normalization on Deep Neural Network for Time Series Forecasting**. 2018. Disponível em: https://arxiv.org/abs/1812.05519. Acesso em: 3 abr. 2019.

BROWNLEE, J. **Long Short-Term Memory Networks With Python**: Develop sequence prediction models with deep learning. [S.l.]: Jason Brownlee, 2017.

CHEN, C. J.; MIIKKULAINEN, R. Creating melodies with evolving recurrent neural networks. *In:* INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 3., 2001, Piscataway. **Proceedings [...]**, Piscataway: [s.n], p. 2241–2246, fev. 2001.

CHOLLET, F. **Building Autoencoders in Keras**. 2016. Disponível em: https://blog.keras.io/building-autoencoders-in-keras.html. Acesso em: 12 abr. 2019.

DALKEY, N.; HELMER, O. An experimental application of the delphi method to the use of experts. **Management Science**, v. 9, n. 3, p. 458–467, abr. 1963.

DICKEY, D.; FULLER, W. Distribution of the estimators for autoregressive time series with a unit root. **Journal of the American Statistical Association (JASA)**, v. 74, p. 427–431, jun. 1979.

ELLIOT, C. H. H. A. **Time Series Prediction: Predicting Stock Price**. 2017. Disponível em: https://arxiv.org/abs/1710.05751. Acesso em: 4 abr. 2019.

ER, M. J. Face recognition with radial basis function (rbf) neural networks. **IEEE Transactions on Neural Networks**, v. 13, n. 3, p. 697–710, ago. 2002.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, USA: MIT Press, 2016.

GRAVES, A.; MOHAMED, A. R.; HINTON, G. Speech recognition with deep recurrent neural networks. *In:* INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 1., 2013, Vancouver. **Proceedings [...]**, Vancouver: [s.n], p. 6645–6649, mar. 2013.

HANSSON, M. **[Seminário]**, **On stock return prediction with LSTM networks**. Lund, Suécia: Lund University, 2017.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997.

HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. **Proceedings of the National Academy of Sciences**, v. 79, n. 8, p. 2554–2558, abr. 1982.

HU, Y. **Overcoming the vanishing gradient problem in plain recurrent networks**. 2018. Disponível em: https://arxiv.org/abs/1801.06105. Acesso em: 3 abr. 2019.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, v. 9, n. 3, p. 90–95, 2007.

JOUNI, H. Neural network architecture for breast cancer detection and classification. *In:* INTERNATIONAL MULTIDISCIPLINARY CONFERENCE ON ENGINEERING TECHNOLOGY, 1., 2016, Beirut. **Proceedings [...]**, Beirut: [s.n], p. 37–41, nov. 2016.

KARIM, F. Lstm fully convolutional networks for time series classification. **IEEE Access**, v. 6, p. 1662–1669, dez. 2017.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group., v. 521, n. 7553, p. 436–444, maio 2015.

LOSHCHILOV, I.; HUTTER, F. **Decoupled Weight Decay Regularization**. 2017. Disponível em: https://arxiv.org/abs/1711.05101. Acesso em: 4 abr. 2019.

MAHATA, M. N. A.; PRASAD, D. Identification of short-term and long-term time scales in stock markets and effect of structural break. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 524, n. 2, p. 1236–1250, jun. 2019.

MAKNICKIENE, A. M. N. Application of neural network for forecasting of exchange rates and forex trading. *In:* INTERNATIONAL SCIENTIFIC CONFERENCE OF BUSINESS AND MANAGEMENT, 7., 2012, Vilnius. **Proceedings [...]**, Vilnius: [s.n], p. 123–127, maio. 2012.

MARKOWITZ, H. M. Foundations of portfolio theory. **Journal of Finance**, v. 46, n. 2, p. 469–477, jun. 1991.

MCKINNEY, W. Data structures for statistical computing in python. *In:* PYTHON IN SCIENCE CONFERENCE, 9., 2010, Austin. **Proceedings [...]**, Austin: [s.n], p. 51–56, jan. 2010.

METAQUOTES. **Metatrader Documentação**. 2019. Disponível em: https://www.metatrader5.com/pt/terminal/help/startworking. Acesso em: 8 abr. 2019.

NELSON, D. M. Q. **Uso de Redes Neurais Recorrentes para previsão de séries temporais financeiras**. 2017. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, Belo Horizonte, Brasil, 2017. Disponível em: https://repositorio.ufmg.br/handle/1843/ESBF-AM2NTS. Acesso em: 12 abr. 2019.

OLAH, C. **The Unreasonable Effectiveness of Recurrent Neural Networks**. 2015. Disponível em: http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Acesso em: 20 abr. 2019.

OLIPHANT, T. E. **A guide to NumPy**. [S.l.]: Trelgol Publishing USA, 2006. v. 1.

OLIVEIRA, S. Um estudo em séries temporais na análise da receita nominal de vendas de veículos e motos. **Conexão Ciência (Online)**, Centro Universitário de Formiga, Formiga, Minas Gerais, p. 1–5, nov. 2012.

PASZKE, A. Pytorch: An imperative style, high-performance deep learning library. In: WALLACH, H. (Ed.). **Advances in Neural Information Processing Systems 32**. [S.l.]: Curran Associates, Inc., 2019. p. 8024–8035.

PEDREGOSA, F. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PERWEJ, Y.; CHATURVEDI, A. Neural networks for handwritten english alphabet recognition. **International Journal of Computer Applications**, Foundation of Computer Science, v. 20, n. 7, p. 1–5, abr. 2011.

RODRIGUEZ, D. **Backtrader Documentação**. 2019. Disponível em: https://www.backtrader.com/docu/. Acesso em: 11 jun. 2019.

RONACHER, A. **Flask Documentação**. 2019. Disponível em: https://flask-ptbr.readthedocs.io/en/latest/. Acesso em: 5 jun. 2019.

ROSSUM, G. **Python Reference Manual**. Amsterdam, Holanda: Centre for Mathematics and Computer Science, 1995.

SAINATH, T. N. Convolutional, long short-term memory, fully connected deep neural networks. *In:* INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 1., 2015, Brisbane. **Proceedings [...]**, Brisbane: [s.n], p. 4580–4584, abr. 2015.

SILVER, D. Mastering the game of go without human knowledge. **Nature**, Nature Publishing Group, v. 550, n. 7676, p. 354, out. 2017.

SRIVASTAVA, N.; MANSIMOV, E.; SALAKHUDINOV, R. **Unsupervised learning of video representations using lstms**. 2015. Disponível em: https://arxiv.org/abs/1502.04681. Acesso em: 5 abr. 2019.

WANG, Y. Attention-based lstm for aspect-level sentiment classification. *In:* CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, 1., 2016, Austin. **Proceedings [...]**, Austin: [s.n], p. 606–615, nov. 2016.

YAHOO. **Dados Históricos**. 2019a. Disponível em: https://finance.yahoo.com/quote/PETR4.SA/history?p=PETR4.SA. Acesso em: 1 nov. 2019.

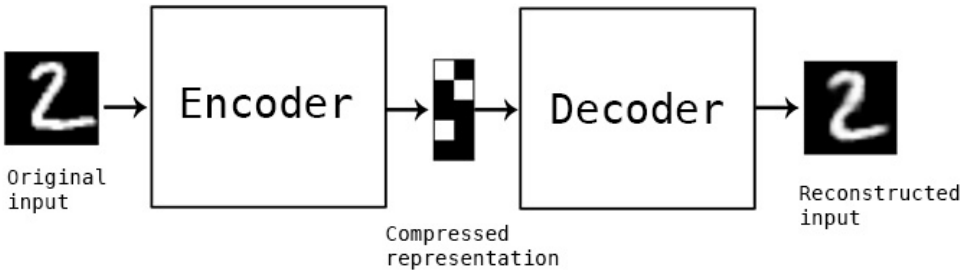
YAHOO. **Gráficos de Mercado**. 2019b. Disponível em: http://finance.yahoo.com/chart/GOOG. Acesso em: 7 abr. 2019.

ZHOU, S. **What happens in Sparse Autoencoder**. 2016. Disponível em: https://medium.com/@syoya/what-happens-in-sparse-autencoder-b9a5a69da5c6. Acesso em: 12 abr. 2019.

APÊNDICE A – AUTOENCODER

Autoencoders são redes neurais que treinam através de um algoritmo supervisionado chamado de auto-supervisionado (do inglês, *self-supervised*). Diferente de redes neurais que tentam aprender relações nos dados de tal forma à classificar ou regredir relações entre os dados, *autoencoders* tentam aprender a recriar a entrada na saída (GOODFELLOW; BENGIO; COURVILLE, 2016). Como demonstra a Figura 18, *autoencoders* são à junção de um *encoder* e um *decoder*, onde o *encoder* comprimi a informação da entrada de tal forma que ocorra redução da dimensionalidade da entrada e o *decoder* opera fazendo o processo inverso, isto é, passa a aprender à recriar a entrada através da representação compactada dos dados, aumentando a dimensionalidade dos dados.

Figura 18 – Modelo de um Autoencoder



Fonte: Chollet (2016).

Desta forma, *autoencoders* executam extração de características (do inglês, *feature extraction*) dos dados, onde a rede neural funciona executando a redução não-linear da dimensão dos dados. Com isso, pode-se definir uma rede neural *autoencoder* como uma rede cujas camadas ocultas compostas pelo *encoder* e pelo *decoder* podem ser descritas como uma função de reconstrução da entrada onde:

$$h = f(x) \tag{A.1}$$

$$r = g(h) \tag{A.2}$$

$$r = g(f(x)) \tag{A.3}$$

Com o *encoder* sendo descrito por (A.1) e o *decoder* por (A.2), obtém-se o *autoencoder*

sendo descrito por (A.3). Nota-se que o objetivo é obter um *autoencoder* onde a saída r é diferente da entrada x. Uma vez que pretende-se criar uma rede neural que aprenda a priorizar as características que mais impactam a reconstrução da entrada. Uma forma de se obter uma rede *autoencoder* que priorize características mais importantes, é forçar que h tenha dimensão menor que a entrada x (GOODFELLOW; BENGIO; COURVILLE, 2016). Dado o processo de aprendizado, cuja o objetivo é minimizar uma função de perda definida por:

$$L(x) = (x, g(f(x))) \tag{A.4}$$

Outra forma de treinar redes *autoencoder* de tal forma que se aprenda caraterísticas mais importantes e que mantenha o modelo generalizado é através do uso de regularização, isto é, acrescentar ao calcula da função de perda uma forma de penalização (GOODFELLOW; BENGIO; COURVILLE, 2016). Sendo esta penalidade definida por $\Omega(h)$, temos:

$$L(x) = (x, g(f(x)) + \Omega(h)) \tag{A.5}$$

Técnicas de regularização podem ser vistas como uma forma de reduzir o erro em teste através de aumento de erro durante treinamento, isto ocorre devido à adição da penalidade $\Omega(h)$, conhecida como decaimento de peso (do inglês, *weight decay*), à função de perda. Tal função Ω varia de acordo com o tipo de regularização que se usa. Uma forma de se modificar o peso da penalidade sobre a função L como demonstrado na equação (A.5) é através de um peso α, tal que a contribuição da penalidade seja descrita por $\alpha\Omega(x)$ (GOODFELLOW; BENGIO; COURVILLE, 2016).

Os mais comumente usados tipos de regularização devido à sua simplicidade são os métodos L^1 e L^2. Onde regularização L^2, conhecida como regressão do cume (do inglês, *ridge regression*), tem seu parâmetro de penalidade descrito pela equação (A.6), onde β_j é o peso. Já a regularização L^1, conhecida como regressão lasso (do inglês, *lasso regression*), tem seu parâmetro de penalidade descrito pela equação (A.7) (GOODFELLOW; BENGIO; COURVILLE, 2016).

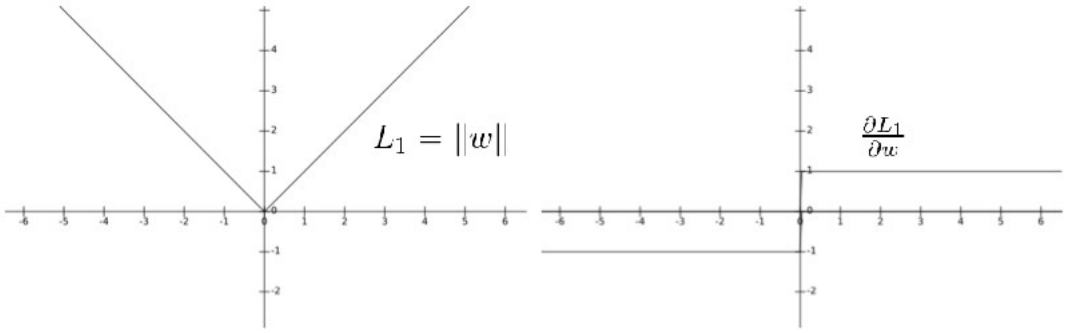
$$\alpha\Omega(h) = \alpha \sum_{j=1}^{p} \beta_j^2 \tag{A.6}$$

$$\alpha\Omega(h) = \alpha \sum_{j=1}^{p} ||\beta_j|| \tag{A.7}$$

Nota-se a diferença entre as duas regularizações onde, L^2 tende à acrescentar uma penalidade quadrática enquanto L^1 tende à acrescentar uma penalidade absoluta. Desta forma, L^1 tende a tornar características não relevantes próximas de zero, enquanto L^2 não. No Gráfico 21 evidencia-se a regularização L^1, já no Gráfico 22, a regularização L^2. Nas duas imagens, na esquerda tem-se a função de penalidade e na direita sua derivada, onde w é o vetor de peso.

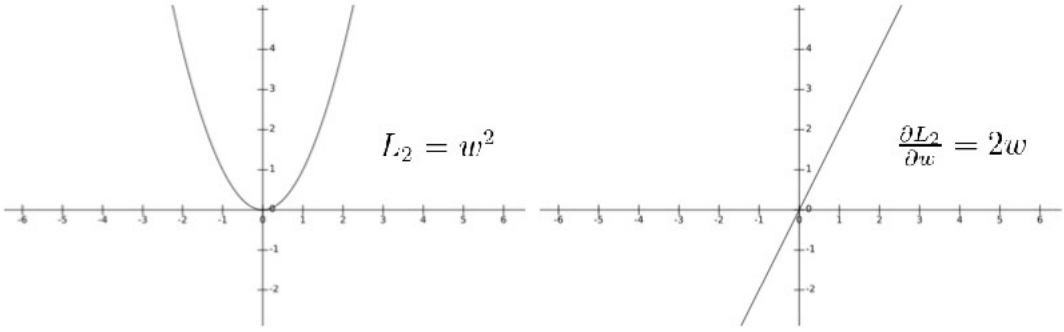
Nota-se no Gráfico 21 como o impacto da parcela de penalidade L^1 comporta-se durante *back-propagation*, sua derivada contém valor contante, portanto, através da escolha de α em $\alpha\Omega(h)$ pode-se zerar a influência de um peso, ou característica, durante o treinamento. Assim como no Gráfico 22, nota-se que o comportamento da penalidade, tende a aumentar quadraticamente a influência de pontos fora da curva durante o erro.

Gráfico 21 – Função de regularização L^1



Fonte: Zhou (2016).

Gráfico 22 – Função de regularização L^2



Fonte: Zhou (2016).

Esta diferença faz com que a regularização L^1 execute seleção de características(do inglês, *feature selection*), isto é, seleção das características. Com isso, obtém se um modelo mais simples, cuja

saída depende de menos características. Já a regularização L^2 tende a aumentar o impacto de pontos fora da curva, uma vez que o seu erro é elevado ao quadrado. Desta forma, a rede neural aumenta sua robustez, ao passo que passa-se a treinar a rede com grande impacto de pontos fora da curva, aumentando o erro durante treinamento, o que leva à maior generalização durante teste.

ANEXO A – CÓDIGO

```
"""
+-----------------------------------------------------+
| MAIN CODE |
+-----------------------------------------------------+
@author: Rafael Costa de Almeida
@copyright: Rafael Costa de Almeida
"""
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from utils.support import System, ignite_trainer, AdamW # scripts para
                                    treino dos modelos
from models import torch_models # scripts com arquitetura de modelos
import torch # pytorch
import copy
import sys

epochs = 150 # exemplo de epochs para se executar treinamento
num_layers = 6 # numero de camada dos modelos

if __name__ == "__main__":
    input_size = int(sys.argv[1]) # recebe do usu rio o tamanho da
                                        janela deslizante
    # chama o construtor dos modelos a serem treinados
    models = [torch_models.LayerNormLiLSTM, torch_models.
                                        LayerNormBasicLSTM, torch_models
                                        .LayerNormFCLSTM],
                torch_models.LiLSTM, torch_models.BasicLSTM,
                                                torch_models.FCLSTM]

    # nomeia cada modelo a ser treinado para salvar em disco suas
                                        informa  es
    names = ["LayerNormLiLSTM_", "LayerNormBasicLSTM_", "
                                        LayerNormFCLSTM_"],
                "LiLSTM_", "BasicLSTM_", "FCLSTM_"]
```

```
# cria inst ncia da classe que administra o treino e
                                    inicializa  o do pre-
                                    processamento
syss = System(model_prefix='Master', num_layers=num_layers,
                                    input_size=input_size,
 global_normalize_interval=None, local_normalize_interval=(1, 1e-5))

# configura ambiente para treino como checagem de GPU e etc
syss.set_enviroment()

# cria os bancos de dados com o pre processamento
syss.generate_dataloaders(syss.data_folder)

# sufixos e prefixos a serem adicionados ao nome para facilitar
                                    identifica  o
sufixs = str(epochs)
if syss.global_normalize_interval is not None:
    sufixs += "G"
if syss.local_normalize_interval is not None:
    sufixs += "L"
sufixs +=str(input_size)

# loop para cria  o e treinamento dos modelos
for model_type, name in zip(models, names):
    syss.model_prefix= name+sufixs

    # cria modelo de acordo com o construtor e os par metros
                                        setados
    model = model_type(input_dim=syss.input_dim,
                hidden_dim=1, output_dim=syss.output_dim,
                num_layers=syss.num_layers, dropout=syss.dropoutRate
                                                    ,
                input_size=syss.input_size, output_size=syss.
                                                    output_size,
                device=syss.device)

    # mostra modelo construido para usuario
    print(model)
```

```
        # treina modelo
        result = syss.train_model(model, epochs)
```