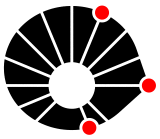

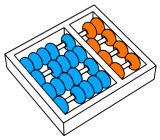

Utilização de Redes Neurais para Previsões no Mercado de Ações

R. Castelão

Relatório Técnico - IC-PFG-18-01
Projeto Final de Graduação
2018 - Junho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Utilização de Redes Neurais para Previsões no Mercado de Ações

R. Castelão*

Resumo

Esse trabalho tem como objetivo desenvolver um sistema capaz de realizar projeções dos preços de abertura de ações de empresas participantes da BOVESPA e NASDAQ. Para atingir tais objetivos foram criados scripts em Python utilizando bibliotecas específicas de aprendizado de máquina para criar duas propostas de solução: uma baseada em redes neurais recorrentes com tecnologia LSTM e outra com base na ferramenta open source do Facebook, o Prophet. Com base nos resultados obtidos, percebeu-se que a utilização de redes neurais recorrentes apresenta uma assertividade maior, chegando a um erro de 5.21 RMSE para o dataset de teste no melhor caso, para ações da American Airlines Group (AAL). Foi também constatado que a tarefa de previsão pode apresentar um erro razoável, porém, a identificação de tendências futuras é muito bem retratada pelo algoritmo, o que mostra que o sistema possui utilidade prática. A pesquisa também constatou que o Prophet é uma ferramenta bem interessante no mercado, sendo utilizada na tomada de decisões dentro do próprio Facebook. Os resultados obtidos foram bem relevantes se olhado para um horizonte de previsão de aproximadamente 50 dias, apresentando erro menor que 10 RMSE.

1 Introdução

Com a popularização da Inteligência Artificial, diversos investidores e pesquisadores tem buscando criar soluções, o mais assertivas possíveis, para realizar projeções sobre o comportamento de ações de forma a obter maior lucro a partir da negociação de ações com conhecimento prévio.

Além disso, com o número cada vez maior de engenheiros e cientistas de computação trabalhando no mercado financeiro, cresceu a variedade de aplicações dos conhecimentos adquiridos na universidade nessas áreas. Um bom meio de se fazer isso é pensar em soluções tecnológicos com base nas dificuldades enfrentadas nesses setores.

Finalmente, redes neurais tem sido utilizadas parta comprovar/refutar a Hipótese de Mercados Eficientes, que afirma que nenhum sistema poderia ser superior ao mercado, porque, se tal sistema existisse, de alguma forma ele se tornaria público e qualquer indivíduo poderia utilizá-lo, fato que diminuiria seu ganho potencial.

Dado o cenário apresentado, o objetivo principal deste projeto foi a criação de um sistema baseado em aprendizado de máquina para prever os valores de abertura de ações (a empresa

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

mais trabalhada durante o projeto foi a American Airlines Group) e as tendências desses preços. Para isso foi necessário desenvolver um script em Python, fortemente baseados na biblioteca *keras* e *sklearn*, para implementar uma rede neural recorrente baseada na tecnologia LSTM (*Long Short-Term Memory*, ou em português, Memória de Longo e Curto Prazo). Arquiteturas distintas de redes neurais foram testadas e suas performances na tarefa foram mensuradas.

As previsões também foram testadas através da ferramenta Prophet. Para isso foi criado um script em Python com utilização da biblioteca *fbprophet* para a criação das rotinas de treino e cross validation do modelo. Sua performance também foi testada e avaliada, além de comparada com a performance das redes neurais.

As próximas seções descrevem o problema e detalham a metodologia e resultados obtidos. O conteúdo está distribuído da seguinte forma:

- No capítulo 2 estão descritos os fundamentos do mercado financeiro, bem como as hipóteses que regem essa área de estudo.
- No capítulo 3 estão descritos o funcionamento das redes neurais recorrentes. É detalhado o que são modelos sequenciais, as formulações matemáticas das redes neurais e o funcionamento de perceptrons LSTM.
- O capítulo 4 apresenta o Prophet, bem como seu funcionamento e motivações para utilização.
- O capítulo 5 contém a explicação detalhada da metodologia utilizada para criar os scripts em Python, tanto para a implementação das redes neurais, como para o Prophet. No capítulo 6 é mostrado os resultados obtidos com ambas as tecnologias. Foram feitos diversos testes com diferentes cenários de redes neurais e medido seus desempenhos.
- O capítulo 7 descreve as conclusões obtidas com o estudo.

1.1 Objetivos

Esse projeto tem como objetivo estudar técnicas de aprendizado de máquina para atuar na previsão dos preços de abertura de ações para empresas participantes da BOVESPA (Bolsa de Valores de São Paulo) e NASDAQ (Associação Nacional de Corretores de Títulos de Cotações Automáticas).

2 Fundamentos do Mercado Financeiro

2.1 A Hipótese de Mercados Eficientes

A Hipótese de Mercados Eficientes descrita por Fama (1970, 1991) se insere dentro da Moderna Teoria de Finanças. Essa teoria parte do pressuposto que, no processo de tomada de decisões, o homem é capaz de analisar todas as informações disponíveis e considerar todas as hipóteses para a solução do problema [1].

Assim, um mercado de capitais é eficiente se ele explora todas as informações disponíveis para a precificação dos ativos. Portanto, os preços desse mercado refletem todas as informações disponíveis de forma lógica e instantânea. Logo, como não há um lapso de tempo entre a disponibilidade da informação e sua disseminação no mercado, não há como um investidor obter vantagem sobre outro investidor [2].

Fama (1970, 1991) definiu três formas de eficiência de mercado:

- **Forma forte:** qualquer informação que se relaciona com o valor da ação, conhecida por pelo menos um investidor, estará refletida no preço da ação. Assim, os preços das ações refletem plenamente todas as informações existentes no mercado, sejam elas públicas ou privadas.

- **Forma semi-forte:** os preços das ações refletem todas as informações disponíveis aos investidores. Dessa forma, não é possível obter vantagem no mercado com base na análise dessas informações, porque outros analistas já fizeram isso e os preços já foram ajustados.

- **Forma fraca:** os preços das ações refletem todas as informações contidas no histórico de preços, além de volumes já negociados.

2.2 Testes de Eficiência de Mercado

De acordo com o modelo *Random Walk* de Disorntetiwat [3], as ações não se comportam de maneira previsível, mas sim de maneira aleatória, e por isso a melhor expectativa de preço para o dia seguinte é o próprio período atual.

A fim de examinar quais estratégias de investimento serão mais bem sucedidas no sentido de gerar retorno, foram criados diversos tipos de testes de eficiência de mercado. Alguns testes também avaliam os custos de transações e de prováveis execuções.

Grandes estudos feitos a partir de 1970 tentam refutar o modelo do Random Walk. O ramo de Finanças Comportamentais, por exemplo, tenta mostrar que os preços não variam de forma aleatória, mas sim que seguem o princípio de retorno a média, além de serem altamente influenciados pela subjetividade do comportamento humano.

2.3 Métodos de Previsão de Ações

Desde que o modelo de Random Walk não seja completamente verdade, é válido estudar e criar métodos de previsão do mercado de ações, a fim de obter vantagens transacionais. Esses métodos estão em constante evolução nos últimos anos, e graças ao crescente poder computacional, diversas anomalias no histórico de preços estão sendo explicadas e adicionadas em modelos teóricos.

Alguns desses métodos são apresentados em seguida. Com eles é possível ter uma noção de como evoluímos até as análises de redes neurais.

2.3.1 Análise Técnica

Afirma que os preços se movem em tendências ditadas pelas constantes mudanças de atitudes de investidores em resposta a diferentes forças. Utiliza o histórico de movimento de preços e volumes e outras características, ignorando fatores externos, para prever o comportamento de ações [2].

Na análise técnica, os investidores tentam identificar anomalias e tendências através do estudo de gráficos. Acredita-se que essas tendências tem comportamento cíclico e são baseadas nas leis de oferta e demanda. Há inúmeros indicadores técnicos derivados desse tipo de análise a partir dos quais podem ser inferidas regras de aplicação e entradas em redes neurais, como: indicadores de filtro, indicadores de volume, indicadores de momento, análise de curva de tendência, médias móveis, análise de padrões etc.

2.3.2 Análise Fundamentalista

Procura informações não só referentes ao histórico de preços e volume, mas faz um estudo profundo sobre a empresa, no qual se avalia seu panorama setorial, suas conjunturas macro e microeconômicas, demonstrações financeiras, capital humano da organização, balanço patrimonial etc.

Esse modelo tem vantagem sobre a análise técnica pelo fato de que sua abordagem permite prever mudanças antes que elas apareçam efetivamente nos gráficos. Como as companhia são comparadas entre si e seus panoramas de crescimento são comparados de acordo com o ambiente econômico, os investidores tornam-se mais familiares com o aspecto geral da empresa e mercado.

Esse tipo de abordagem é mais difícil de automatizar, utilizando redes neurais, por exemplo, pela complexidade das análises, subjetividade das informações coletadas, presença de fatores desconhecidos dos analistas e demora até que o resto do mercado interprete as informações da mesma forma.

2.3.3 Séries Temporais

Séries temporais são sequências de vetores $x(t)$, onde t representa o tempo. Analisar esses elementos é identificar as características e propriedades que descrevem o fenômeno gerador da série [4].

A maior aplicação da análise de séries temporais é a geração de modelos de previsão. A partir de informações passadas da série e outras variáveis importantes ao problema é possível inferir com certo grau de confiabilidade valores futuros. O uso desses modelos de previsão são fundamentais para diminuir os riscos nas tomadas de decisão.

As séries temporais foram extensamente utilizadas na implementação desse projeto, e são trabalhadas especialmente com redes neurais recorrentes, que serão explicadas na próxima seção.

2.3.4 Teoria do Caos

A teoria do caos parte do pressuposto de que não existe ordem em casualidade e portanto acredita que um processo tem, ao mesmo tempo, características determinísticas e aleatórias [5]. Ela tenta explicar sistemas que são caóticos, ou seja, processos não-lineares com características aleatórias. Portanto, sistemas de redes neurais são perfeitos para interpretar esses sistemas, pois conseguem olhar tanto para o conjunto de informações determinísticas, que são as informações históricas, como para as informações aleatórias, que utilizam parâmetros estatísticos de uma dada função.

2.3.5 Sistemas Computacionais Especialistas

Existem diversas outras técnicas computacionais para auxiliar investidores a tomar decisões no mercado de ações, entre elas estão os sistemas computacionais especialistas. Esses são usados na criação de regras de negociação baseadas em indicadores técnicos, ou na previsão do comportamento de ações com o auxílio de redes neurais.

A maior dificuldade nesses sistemas é que nem mesmo os investidores mais experientes tem conhecimento em fórmulas as regras com as quais esses sistemas trabalham, visto que é muito difícil um entendimento completo do mercado. As redes neurais, no entanto, apresentam aqui a vantagem de extraírem essas regras sem que seja necessário o completo entendimento delas.

3 As Redes Neurais Recorrentes

3.1 Modelos Sequenciais

Redes neurais artificiais (RNA) representam um modelo de processamento que tem como propósito imitar o funcionamento do nosso cérebro para resolver problemas. Atualmente, graças ao rápido avanço tecnológico, podemos ver RNAs sendo usadas em uma variedade de aplicações no mercado [6]. Os maiores motivos desse avanço foram:

1. O avanço das GPUs tem melhorado drasticamente nosso poder computacional nos últimos anos, o que nos possibilitou aumentar vastamente a profundidade e tamanho das camdas dos neurônios (perceptrons no contexto de redes neurais artificiais).

2. O fato de que agora redes neurais podem processar dados sequenciais tanto no input como no output. Esse é o modo como nosso cérebro trabalha. Ele não soluciona problemas de classificação binários para entender ideias complexas, mas sim formula pensamentos baseados em sequências de informações e gera uma resposta também baseada em sequências.

O processamento de dados sequenciais é possível graças aos modelos de sequenciamento, categoria nas quais as redes neurais recorrentes - estrutura utilizada nesse projeto - se encaixam.

Algumas aplicações onde modelos sequenciais são úteis nos dias de hoje são:

- **Reconhecimento de voz:** a entrada na rede é um clip de áudio, enquanto a saída é um texto. Ambos são dados sequenciais. A entrada por ser um dado que varia no tempo, e a saída pode ser uma sequência de palavras.
- **Geração de música:** o input aqui pode ser diverso (um conjunto vazio, um inteiro representando o gênero musical, um conjunto de parâmetros denotando aspectos musicais, etc.). O output é uma música, caracterizada por uma sequência de notas musicais distribuídas no tempo.
- **Classificação de sentimento:** o input aqui é um texto que será analisado e portanto um dado sequencial, enquanto o output é uma classificação, por exemplo, número de estrelas de 0 a 5 para um filme.
- **Análise de sequência de DNA:** o DNA é representado por quatro letras do alfabeto, A, C, G e T, portanto podemos analisar em qual região existe, por exemplo, alguma proteína.
- **Tradução:** o input é um texto em uma língua, e o output o texto anterior traduzido para uma outra linguagem.
- **Reconhecimento de atividade em vídeos:** o input é uma sequência de frames de vídeo e o output é a descrição do que as imagens estão mostrando.
- **Predição do comportamento do mercado de ações:** o input é a variação das ações ao longo do tempo, e o output é o comportamento que essas ações terão nos próximos dias.

3.2 Redes Neurais Recorrentes

Para entender sobre redes neurais recorrentes, é interessante entender sobre redes neurais feedforward. Essa é uma estrutura de aprendizado de máquina onde a informação se move em apenas uma direção: camada de entrada (input layer) para as camadas intermediárias (hidden layers) para a camada de saída (output layer), como mostrado na Figura 1. Como a informação se move de forma unidirecional através da rede, ela nunca passa duas vezes no mesmo perceptron (nó da rede) [7].

Redes neurais feedforward não são boas em resolver problemas que envolvem dados sequenciais. Isso porque elas não possuem nenhuma memória do input que receberam previamente, apenas o atual, não existindo assim noção de ordem no tempo ou relação entre duas entradas sequenciais.

Imagine um problema que receba como input um texto incompleto e tente adivinhar a próxima palavra que melhor se encaixa no final da sentença. Considere o texto:

"Hoje o dia está nublado, com diferentes cores de nuvens no..."

Uma rede neural feedforward pode identificar que a palavra que completa a frase seja "céu", isso porque a rede não precisa de nenhum conhecimento prévio para inferir isso. Já para a frase:

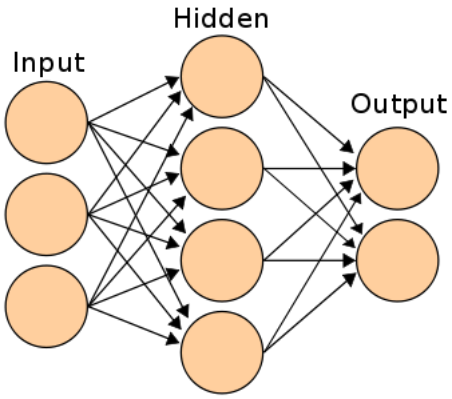

Figura 1: Arquitetura básica de uma rede neural artificial.

"Assim que chegaram a França, João percebeu que não entendia uma palavra do que era dito pelas pessoas de lá. Por sorte, Ana estava com ele. Ela é fluente em..."

Uma rede feedforward não consegue inferir com precisão qual a próxima palavra. A rede pode até compreender que a palavra que segue a sentença represente um idioma, mas, para saber qual, ela deve levar em consideração informações passadas. Mesmo sabendo que anteriormente o país da França foi referenciado, ela não consegue carregar essa informação para as partes mais à frente no texto. Uma rede neural recorrente já conseguiria inferir que a palavra seguinte é "francês", porque ela consegue "lembrar"do conhecimento adquirido anteriormente.

Melhor explicando, o que uma rede neural recorrente faz é que cada feature x de um input alimenta a rede neural A (como a da Figura 2) e gera um output $\hat{y}$, quando a rede lê a próxima feature, ela carrega alguma informação do passo anterior para gerar um output e assim por diante até o final do dado sequencial, como é mostrado na Figura 2 [8, 9].

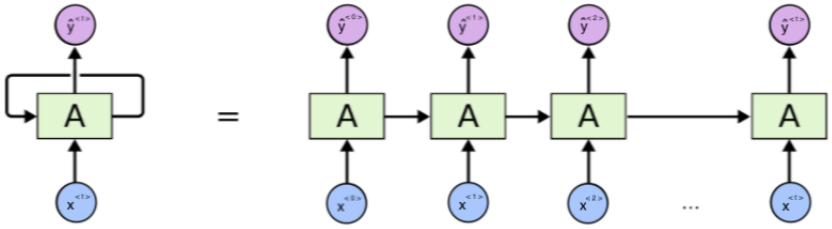

Figura 2: Arquitetura básica de uma rede neural recorrente. A imagem à esquerda é sua forma simplificada, enquanto à direita é sua forma "desenrolada".

A Figura 3 apresenta, com com maiores detalhes, a estrutura de uma rede neural recorrente (RNN - Recurrent Neural Network). Toda vez que a rede neural computa o valor de $\hat{y}^{<t>}$ ela passa para o próximo step uma variável de ativação $a^{<t>}$, que é levada em consideração no cálculo de $\hat{y}^{<t+1>}$. A RNN analisa os dados sequenciais da esquerda para

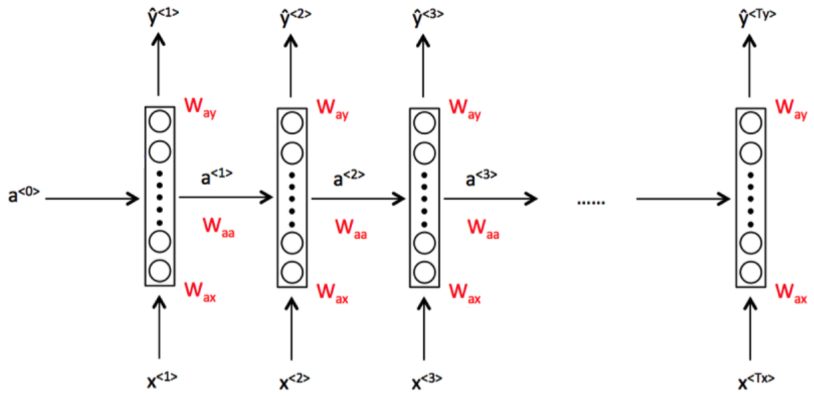

Figura 3: Rede neural recorrente com indicação das matrizes de pesos utilizadas para predição.

a direita e os pesos usados para cada passo, dados pela matriz W_{ax}, são compartilhados entre todos os passos. As conexões horizontais são governadas pela matriz de pesos W_{aa}, enquanto a responsável pelo output é a matriz W_{ay}.

Como já foi citado, a RNN usa informações não apenas do input correspondente, mas de todos os outros anteriores, assim para prever $\hat{y}^{<3>}$ foi levada em consideração informações de $x^{<3>}$, $x^{<2>}$ e $x^{<1>}$. Porém, não são levadas em consideração informações posteriores ao input correspondente, esse é o caso das redes neurais recorrentes bidirecionais (BRNN - Bidirectional Recurrent Neural Network).

A RNN funcionando em forward propagation é regida então pelas seguinte fórmulas:

$$a^{<0>} = \vec{0} \tag{1}$$

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{2}$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y) \tag{3}$$

onde $a^{<t>}$ é a variável de ativação, $g()$ é a função de ativação sendo utilizada, b_a é um fator de correção para o cálculo de $a^{<t>}$ e b_y é o fator de correção para o cálculo de $\hat{y}^{<t>}$.

3.3 Backpropagation Através do Tempo

O exemplo anterior funciona em forward propagation. Para que a rede aprenda através do backpropagation, é preciso uma função de custo. Para isso, será definida seguinte função:

$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}log\hat{y}^{<t>} - (1 - y^{<t>})log(1 - \hat{y}^{<t>}) \tag{4}$$

que é a função de custo da regressão logística padrão.

Para definir uma função de custo geral da sequência inteira, basta fazer o somatório das funções de custo de 1 até T_y, nesse caso $T_x = T_y$. Portanto:

$$L(\hat{y}, y) = \sum_{t=1}^{T_y} (\hat{y}^{<t>}, y^{<t>}) \tag{5}$$

O cômputo do backpropagation é realizada coforme apresentado pelas setas vermelhas da Figura 4.

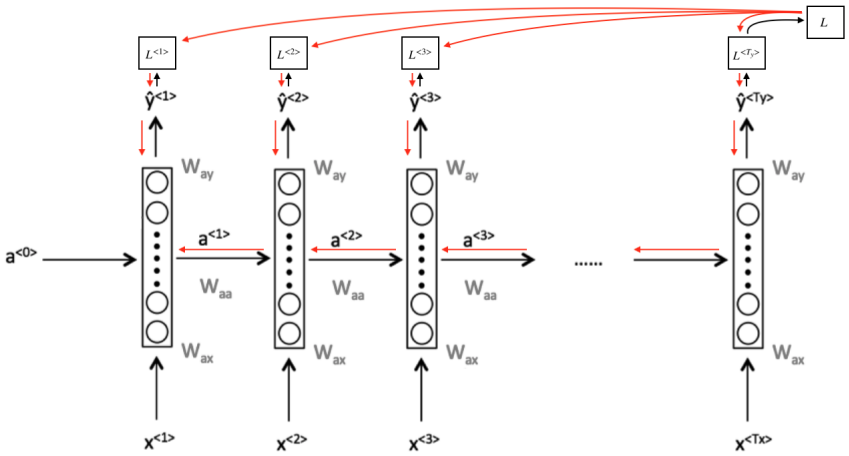

Figura 4: Rede neural recorrente com backpropagation, mostrando a função de custo L para cada timestep, assim como a função geral e as direções da backpropagation em vermelho.

3.4 Unidades de Memória de Longo-Curto Prazo (LSTM)

Até então é possível perceber que o output $\hat{y}$ foi influenciado principalmente pelos valores da sequência próximos a si. Por outro lado, há situações em que as features possuem longas dependências com dados mais distantes. RNNs básicas não são tão eficientes em capturar essas dependências de longo prazo, porque essas redes não conseguem propagar para os timesteps mais no passado o efeito dos pesos. Assim, para focar no problema conhecido como *"vanishing gradients"*, quando a rede "esquece" o que aconteceu anteriormente e não consegue propagar as dependências através da sequência inteira, são usadas as LSTMs.

As LSTMs são um tipo especial de RNN, introduzidas por Hochreiter & Schmidhuber (1997). Elas funcionam extremamente bem em uma grande quantidade de problemas e são extensamente utilizadas no mercado financeiro [10].

Diferentemente das RNNs padrões, as LSTMs possuem um módulo de repetição, até então composto por uma única camada de rede neural, um pouco diferente. Esses módulos são compostos por quatro camadas que interagem de forma especial, como pode ser visto na Figura 5.

O core da LSTM é o estado da célula, que é a linha horizontal que atravessa o topo do diagrama. Ela atravessa a cadeia inteira sofrendo apenas pequenas alterações causadas pelas estruturas conhecidas como "gates". Esses são meios que opcionalmente deixar a informação atravessar a cadeia, e são compostos por uma camada sigmoid e uma operação

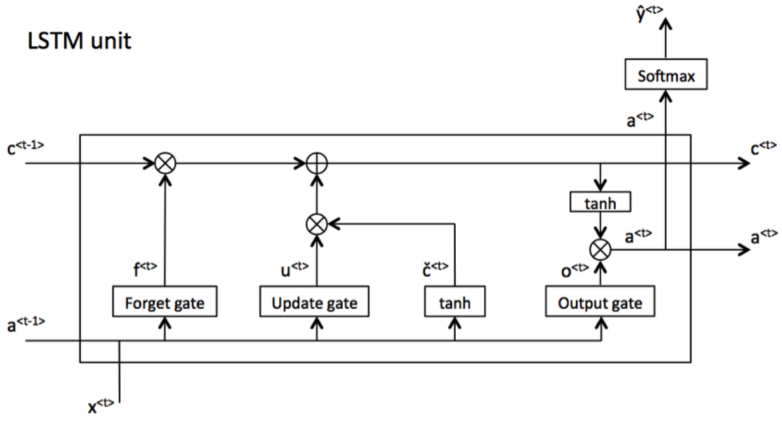

Figura 5: Unidade de LSTM mostrando todos os componentes estruturais como estado da célula, gates e operações matemáticas.

de multiplicação. O sinal sigmoid tem como saída valores de 0 a 1, que representam deixar o sinal passar inteiramente ou não passar, respectivamente. Uma LSTM possui três gates como esse para proteger e controlar o estado da célula [11, 12].

O primeiro desses gates é o "forget gate", que olha para o valor de $a^{<t-1>}$ e $x^{<t>}$ e gera um número de 0 a 1 para cada número no estado da célula . A fórmula para o forget gate é:

$$f_t = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{6}$$

onde W_f é uma matriz de parâmetros que recebe $a^{<t-1>}$ e $x^{<t>}$, e b_f é um valor de correção.

O próximo passo é para decidir qual nova informação será armazenado no estado da célula. Para isso há duas partes: a primeira é uma camada sigmoid chamada "input gate", que decide quais valores serão atualizados, a segunda é uma camada tanh que cria um vetor de novos valores candidatos, $C^{\tilde{<t>}}$, que podem ser adicionadas ao estado, sendo:

$$u_t = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{7}$$

$$C^{\tilde{<t>}} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{8}$$

onde W_u e W_c são matrizes de parâmetros que recebem $a^{<t-1>}$ e $x^{<t>}$, e b_u e b_c são valores de correção.

Na sequência, o antigo $C^{<t-1>}$ é atualizado com um novo $C^{<t>}$, multiplicando o estado antigo por f_t e depois adicionando $u_t * C^{\tilde{<t>}}$. Portanto:

$$C^{<t>} = f_t * C^{<t-1>} + u_t * C^{\tilde{<t>}} \tag{9}$$

Finalmente, é decidido qual será o output. Primeiro, roda-se uma camada sigmoid que decide quais partes do estado farão parte da saída. Depois, submete-se o estado da célula

a uma função tanh (resultando em valores entre -1 e 1) e multiplica-se a saída da função sigmoid, de acordo com:

$$o^{<t>} = \sigma(W_o[a^{<t-1>}, c^{<t>}] + b_o) \tag{10}$$

$$a^{<t>} = o_t * tanh(C^{<t>}) \tag{11}$$

Essa estrutura de gates permitindo o fluxo de informações de um estado de célula é o que permite a solução de diversos problemas relacionados a modelos sequenciais com dependências de longo prazo, como é o caso do problema de previsões do mercado de ações proposto nesse relatório.

4 Prophet

4.1 Introdução ao Prophet

O Prophet [13] é uma ferramenta de previsões open source do Facebook e disponibilizada tanto em Python como em R. Ela foi concebida para facilitar a tarefa de forecasting que é extremamente complexa tanto para máquinas quando para experientes cientistas de dados. O Prophet é otimizado para realizar previsões na área de negócios, tomando como base as tarefas encontradas no Facebook, que possuem as seguintes características:

- Observações horárias, diárias ou semanais com pelo menos alguns meses ou anos de base histórica.
- Sazonalidades presentes: dia da semana e tempo do ano.
- Feriados importantes que são previamente conhecidos, como campeonatos de futebol.
- Um número razoável de informações faltantes e outliers.
- Mudança de comportamento históricas, como quebra de uma companhia ou lançamento de um produto.
- Tendências que não são lineares, onde a tendência atinge um limite natural ou um ponto de saturação.

Portanto, essa ferramenta torna-se interessante para realizar previsões no mercado de ações, que é um ambiente que se enquadra em quase todas as características listadas acima.

4.2 Como o Prophet Funciona

Como parte central, o Prophet funciona como um modelo de regressão aditivo com quatro componentes principais:

- Uma tendência de curva de crescimento linear ou logística. O Prophet detecta mudanças em tendências selecionando diferentes pontos dos dados.

- Um componente sazonal anual modelado usando séries de Fourier.
- Um componente sazonal semanal modelado usando variáveis explicativas.
- Um número razoável de informações faltantes e outliers.
- Uma lista de feriados provida pelo usuário.

O que resulta em um modelo bem interessante para previsões que possuem alta sazonalidade e que algumas informações, como feriados, já foram pré-estabelecidas.

4.3 Utilização do Prophet em Séries Temporais

O projeto em questão não usou o Prophet na sua forma mais completa, por exemplo, através de inputs de uma lista de feriados como parâmetro. O Prophet foi utilizado neste contexto para fazer previsões de séries temporais, mais precisamente, flutuação do mercado de ações.

O modelo, que consiste em dados históricos do preço de abertura de algumas ações, foi treinado através do Prophet. Após esse passo foi aplicada a técnica de cross validation, utilizando as ferramentas de diagnóstico do Prophet. Avaliar performance em séries temporais é diferente do que fazer isso em outros tipos de datasets com variáveis independentes, porque esses podem ser divididos em seções de treino, validação e teste enquanto aqueles não podem. Para realizar a cross validation, o Prophet usa o que é chamado de **dados de cutoff.** Para cada dado de cutoff, o modelo é treinado usando dados exclusivamente antes desse dado. As previsões são então realizadas em um intervalo de tempo após o cutoff. Esse intervalo é conhecido como horizonte e as métricas de erro são calculadas comparando as previsões com os valores reais [13]. Esses parâmetros podem ser melhor interpretados com a Figura 6.

Através do utilitário *performance_metrics* do Prophet é possível calcular o erro sobre a previsão em diferentes métricas, como erro quadrático relativo (RSE), a raíz do erro quadrático médio (RMSE), erro absoluto médio (MAE) e erro percentual absoluto médio (MAPE). Esse erro é calculado como função da distância do cutoff, dada pelo horizonte, como podemos ver através da Figura 7, que é o output da função *plot_cross_validation_metric* [14].

O Prophet consegue ser uma ferramenta bem poderosa para análises de séries temporais, tendo performance superior a muitas soluções interessantes no mercado, como a ARIMA.

5 Metodologia

5.1 Redes Neurais Recorrentes baseadas em LSTM

5.1.1 Dados de Entrada

Para este projeto, os dados utilizados referem-se a séries históricas de ações da BOVESPA (Bolsa de Valores de São Paulo) e NASDAQ (Associação Nacional de Corretores de Títulos de Cotações Automáticas), que transaciona milhares de ações de diferentes empresas.

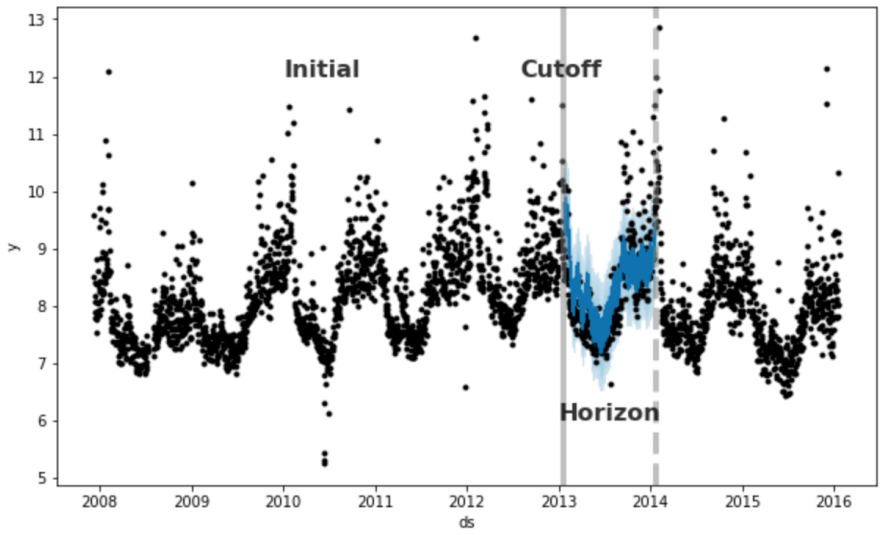

Figura 6: Procedimento de Cross Validation do Prophet atuando sobre uma base de dados. Pode-se ver os parâmetros de cutoff e o horizonte, bem como o parâmetro chamado de inicial, que representa o tamanho do primeiro cutoff. Os pontos pretos são os dados reais e o azul é a previsão realizada pelo Prophet.

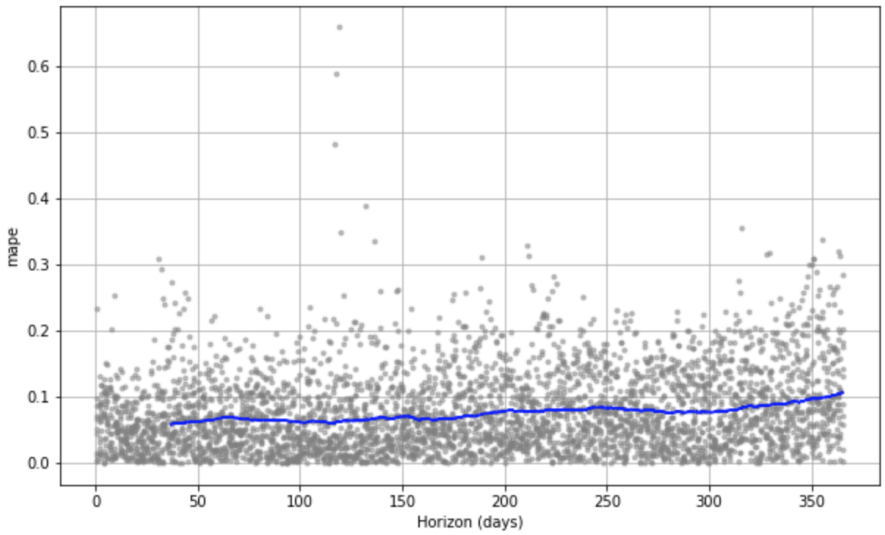

Figura 7: Erro percentual absoluto médio (MAPE) da previsão em cross validation do Prophet para quanto mais distante do cutoff a previsão fica, ou seja, em função do horizonte.

Esses dados são facilmente coletados do *Yahoo! Finance*, que disponibiliza os dados históricos em arquivos .csv com as informações de abertura, máxima, mínima, fechamento, fechamento ajustado e volume em uma escala de tempo diária.

Para tratar esses dados, foi criado um script em Python para limpar os dados e adicioná-los a uma estrutura de DataFrame da biblioteca *pandas*, como pode ser observado na Figura 8. Visualizando esses dados em forma de gráfico (Figura 9), através da biblioteca *plotly*, é possível perceber que os valores são muito próximos entre si, e é por esse motivo que a rede neural construída leva em consideração uma série temporal univariada, e que, portanto, só possui um único escalar como elemento de observação.

Date	Open	High	Low	Close	Adj Close	Volume
2005-09-27	21.05000	21.40000	19.10000	19.30000	18.58854	961200
2005-09-28	19.30000	20.53000	19.20000	20.50000	19.74430	5747900
2005-09-29	20.40000	20.58000	20.10000	20.21000	19.46499	1078200
2005-09-30	20.26000	21.05000	20.18000	21.01000	20.23550	3123300
2005-10-03	20.90000	21.75000	20.90000	21.50000	20.70743	1057900

Figura 8: DataFrame tratado com a biblioteca *pandas* mostrando os primeiros valores da base de dados do *Yahoo! Finance* para as ações da American Airlines Group (AAL).

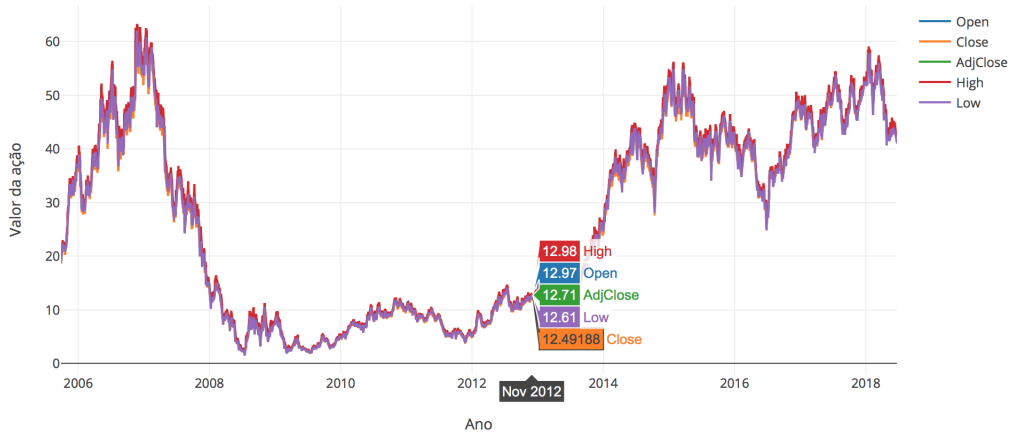

Figura 9: Gráfico das principais variáveis dos dados históricos da empresa American Airlines Group plotado com o auxílio da biblioteca *plotly*.

5.1.2 Normalização dos Dados

Como a rede neural trabalha com uma série temporal univariada, mais especificamente com os dados de abertura das ações, o DataFrame foi transformado em um Array da biblioteca *numpy* com apenas essa variável indexada pela data.

Para normalizar esses dados foi utilizada a função *MinMaxScalar* da biblioteca *sklearn*, própria para machine learning. Essa função tem como objetivo mapear os dados a serem normalizados no intervalo [0, 1]. Assim, o maior valor do array que está sendo normalizado, receberá o valor de 1, e o menor, o valor de 0, os valores intermediários serão transformados em números dentro desse intervalo.

A normalização é um passo importante para se trabalhar com redes neurais, além de agilizar o processo de gradiente descendente, evita que o algoritmo tome decisões erradas devido ao grande intervalo de valores das variáveis utilizadas.

5.1.3 Defasagem dos Dados

Uma próxima modificação que os dados passam é no sentido de transformar a série temporal em um problema de regressão. Para isso, é feito o que é conhecido como "defasar" o dado, ou seja, para cada linha do nosso dataset, vamos adicionar uma quantidade pré-determinada, chamada de *lag*, de variáveis contendo informações de dados anteriores no tempo. Por exemplo, se o *lag* for igual a 3, para cada linha de informação, no tempo t, de abertura da ação, serão adicionadas mais duas variáveis com os valores de abertura de $t-1$ e $t-2$, fazendo com que cada linha seja uma tupla contendo [t, t-1, t-2].

Desta forma, a rede neural consegue olhar para mais do que um dia para fazer a previsão, porém, há controvérsias se defasar o dado realmente melhora a performance da rede neural [15].

5.1.4 Criação da Rede Neural

O problema proposto faz uso de séries temporais com um input muito grande, e para a previsão ser o mais assertiva possível, a rede neural deve levar em consideração os inputs desde o começo do dataset. Por esse motivo, foi escolhido uma rede neural com base em LSTMs.

Para a criação da rede, foi utilizada a biblioteca *keras*, que é especializada em deep learning. Primeiro foi instanciado um modelo sequencial, visto que estamos trabalhando com séries temporais. Em seguida, foram instanciados um número de perceptrons com módulo de repetição de uma LSTM. Esses perceptrons possuem função de ativação *softmax* e função recorrente de ativação *relu*.

A função de ativação é aquela que aparece na equação 3 e que funciona como output para o valor de $y^{<t>}$. A função *softmax* atribuída a essa ativação tem como objetivo mapear os valores no intervalo [0, 1], assim como a função *sigmoid*. Porém, ela também divide cada output até que a soma de todos eles seja 1. Matematicamente a função de *softmax* está representada na equação 12, onde z é o vetor de inputs e j é o índice dos elementos do vetor [16].

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{12}$$

Já a função de ativação relu é aquela que aparece na equação 2 e que funciona para gerar o valor de $a^{<t>}$. A função *relu* aplicada aqui utiliza *rectified linear units*, que gera

um valor igual a 0, caso o input seja negativo, e um valor igual ao input, caso seja maior ou igual a 0. Esse tipo de função de ativação está sendo cada vez mais usado no lugar da função *sigmoid*. Ela é a forma mais simples de ativação não-linear existente, pois quando o input é positivo, a derivada é simplesmente 1, portanto não há o efeito *squeezing* encontrado em erros com backpropagation como há na função *sigmoid*. Um estudo da Universidade de Toronto [17] comprovou que relu resulta em um treinamento muito mais rápido para redes muito grandes. Matematicamente, a função relu está representada na equação 13:

$$f(x) = max(x, 0) \tag{13}$$

Em seguida, foi instanciado um perceptron do tipo *Dense* com 1 *unit*, que recebe os inputs da rede de LSTMs e devolve um array de tamanho 1 com o valor da previsão do preço de abertura da ação. A função de ativação utilizada aqui é a *linear*, que simplesmente devolve o valor do input recebido.

A Figura 10 mostra a estrutura da rede neural construída. O número de features, $x^{<t>}$, é dado pelo tamanho da defasagem dos dados, *lags*. A quantidade de perceptrons LSTM é definida no modelo, e portanto a quantidade de valores preditos pela camada LSTM, $y^{<t>}$. O modelo é composto por um perceptron do tipo Dense que recebe as informações de todos os $y^{<t>}$ e retorna a resposta final, r. As funções de ativação estão indicadas nas setas de fluxo da informação.

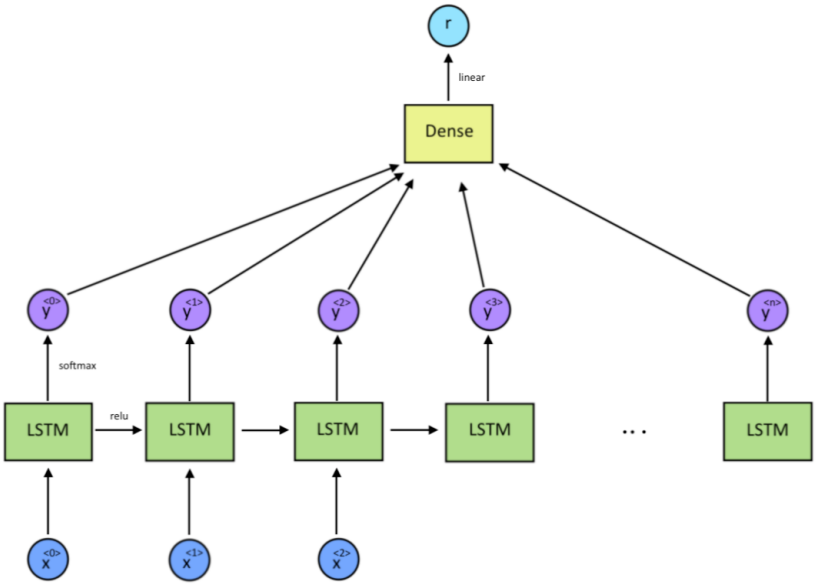

Figura 10: Gráfico com o modelo de rede neural construído no projeto. As features de entrada são dadas por $x^{<t>}$, as previsões da camada LSTM por $y^{<t>}$, e a previsão final por r. As funções de ativação estão indicadas nas setas de fluxo da informação.

5.1.5 Treinamento da Rede Neural

Após a criação da rede neural, foi feita a compilação dessa rede através do comando *compile* da biblioteca *keras*. A função de custo, que é a função que tenta ser minimizada pela rede neural, escolhida foi a de erro quadrático médio.

Após compilar a rede, foi realizado o treinamento dela com o dataset de treino, que corresponde aos primeiros 70% da série histórica de ações da empresa escolhida. Esses 70% precisam ser sequenciais a partir do ponto mais antigo dos dados, já que estamos lidando com um problema de séries temporais. Foi também passado como input do treinamento, o dataset de validação, ou teste, que são os próximos 30% da série histórica.

Além do dataset, são passados mais dois inputs: *epoch* e *batch size*. A *epoch*, ou época, é a quantidade de vezes que os dados de treino são passados inteiramente pela rede neural para treiná-la. Portanto, se o número de épocas é definido como 1, a rede neural treina o modelo passando os dados apenas uma vez por seus perceptrons. Porém, se o número de épocas é 2, a rede neural passa os dados de treino duas vezes pelos perceptrons, a fim de deixar a rede mais bem treinada. Após cada época a rede é testada com os dados de validação, se a função de custo não diminuir, o *learning rate* α do passo de *gradiente descendente* é dividido pela metade, a fim de treinar o modelo melhor [18].

O *batch size* é o tamanho do lote do dataset que é passado para a rede neural Uma vez que é impraticável passar todo o dataset para a rede neural de uma vez só, pelo grande volume de dados, o mesmo é dividido em diversas partes que são os *batches*.

Em seguida, é necessário entender se esse modelo está com um bom fit, ou seja, não está sofrendo *overfitting* e nem *underfitting*. *Overfitting* é quando o modelo foi demasiadamente treinado, a ponto de conseguir fazer previsões com erros baixíssimos para os dados que foram treinados, mas quando um novo dado é apresentado, o erro é alto. A rede aprendeu a memorizar os exemplos de treino, mas não aprendeu a generalizar esse conhecimento para novas informações. O problema de *underfitting* é justamente o oposto, é quando a rede foi tão pouco treinada que não consegue fazer previsões assertivas.

Para endereçar esse problema foi analisado o comportamento do modelo revisitando sua performance a cada época. Quando a rede é treinada, ela retorna uma variável chamada *history*, que contém informações da função de custo definida na compilação. Essas informações são colhidas a cada época tanto para os exemplos de treino, quanto para os dados de validação. Com posse dessas informações é possível identificar se o modelo está bem treinado ou não. Se o modelo está *underfit* ele apresentará uma performance ruim tanto no treino, quanto na validação. Se o modelo está *overfit* ele terá uma excelente performance no conjunto de treino mas, a medida que o modelo continua a ser treinado, a acurácia no conjunto de validação piora pois a rede se torna especialista no conjunto de treinamento. Um bom modelo é aquele treinado até o ponto em que a função de custo retorna curvas de decaimento próximas para os exemplos de treino como para os de validação (Figura 11) [19].

5.1.6 Predição dos Valores

Finalmente, foram realizadas as predições com a função *predict* da biblioteca *keras*. O retorno dessa função já é um array de previsões. Esse array precisa passar por uma trans-

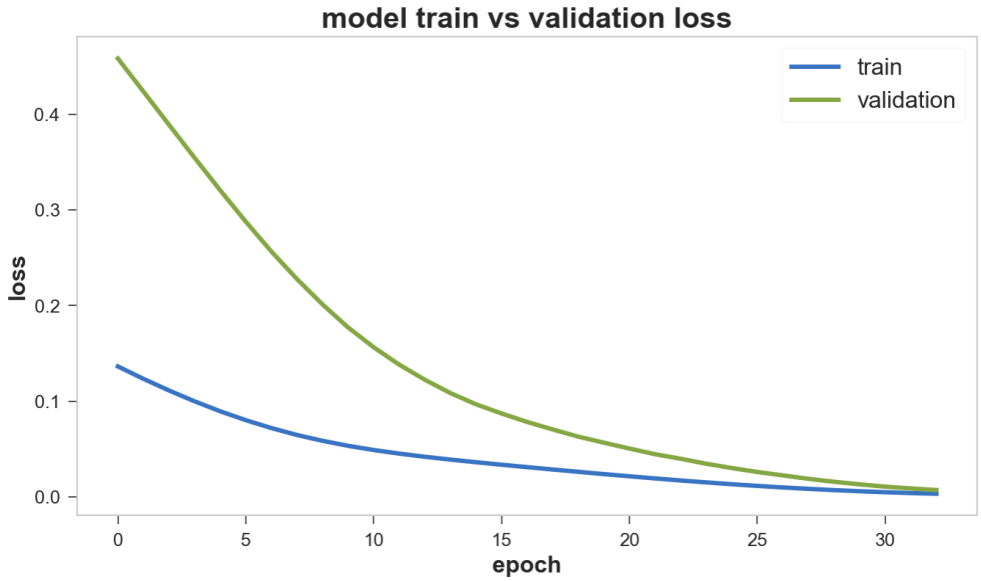

Figura 11: Gráfico da função do custo atingida pelo treinamento do modelo de rede neural baseada em LSTMs com um número ótimo de épocas, atingindo o *best fit*.

formação inversa a fim de retornar seu valor para um formato inverso ao obtido pela função *MinMaxScaler* na fase de normalização.

Para cada dataset, são feitas as predições dos dados de treino, de teste e os dados reais. O gráfico obtido é apresentado na Figura 12.

5.2 Prophet

5.2.1 Dados de Entrada

O procedimento aplicado para tratar os dados de entrada para o Prophet são iguais aos apresentados para a rede neural recorrente na seção anterior. Aqui o dataset também vem da fonte *Yahoo! Finance* e a variável escolhida para treinar o modelo e ser prevista é a de abertura da ação ao longo dos dias. Também trabalhamos com séries temporais univariadas no Prophet.

5.2.2 Treinamento e Previsão

Como mencionado anteriormente, o Prophet foi criado com o intuito de facilitar a tarefa de fazer previsões, gerando valores confiáveis e assertivos. Por esse motivo, é extremamente fácil treinar um modelo e criar previsões a partir dele com essa ferramenta. Para treinar o modelo, basta utilizar a função *fit*, que recebe como parâmetro o dataset de treino. Com a função *make_future_dataframe* é possível determinar o número de dias no futuro que terão seus valores previstos, portanto essa função recebe como parâmetro essa quantidade de dias, chamada de periods.

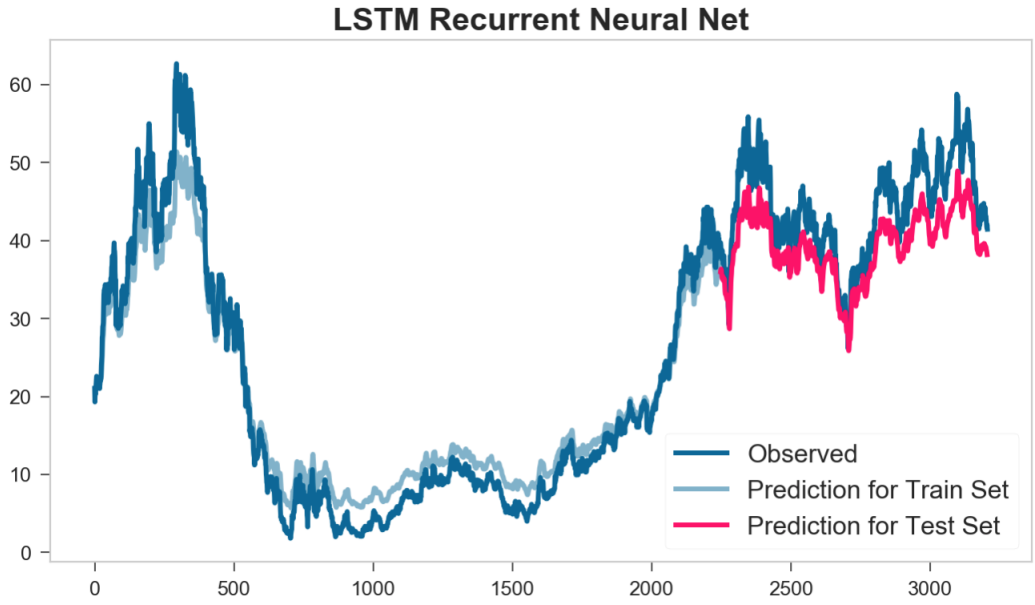

Figura 12: Gráfico com as previsões obtidas pela rede neural para os dados de treino e de teste, além da curva dos dados reais.

Finalmente, é feita a predição desses valores futuros com a função predict. Essa função retorna um DataFrame com as informações de previsão, $\hat{y}$, o limite superior da previsão, $\hat{y}_{upper}$, e o limite inferior, $\hat{y}_{lower}$.

A visualização da previsão é feita plotando-se as informações retornadas pela função *predict*. Na Figura 13 é possível ver um exemplo da previsão feita pelo Prophet. Os pontos pretos representam os valores reais do dataset utilizado, no caso em questão os valores dos preços das ações de uma empresa específica, a linha azul representa os valores previstos pelo modelo e o sombreamento azul em torno da linha representa os limites superior e inferior da previsão.

5.3 Cross Validation

Foi explicado anteriormente que o Prophet inclui a funcionalidade de cross validation para medir o erro da predição usando séries históricas. Também mencionamos que isso é feito através da seleção de pontos de cutoff. O modelo é treinado para todos os pontos da série antes do ponto de cutoff e em seguida faz a previsão para um intervalo de tempo após esse ponto, conhecido como horizonte. Após a previsão, o cross validation pega o próximo ponto de cutoff que está a uma quantidade de pontos a frente do último cutoff, essa quantidade é definida pela variável *period*. O primeiro ponto de cutoff é definido pela variável *initial* [14].

Para iniciar o método de cross validation, as variáveis *initial*, *period* e *horizon* devem ser definidas. O padrão utilizado no projeto foi que *initial* é 60% do tamanho do dataset completo, enquanto *period* e *horizon* são 5% e 10%, respectivamente.

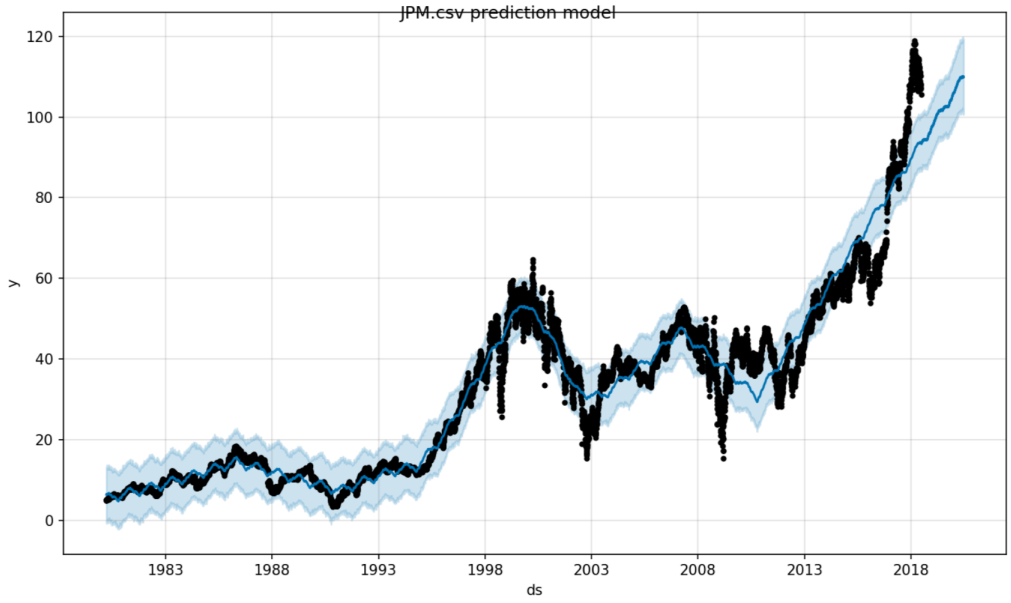

Figura 13: Gráfico com a previsão obtida pelo Prophet para uma série temporal univariada. Os pontos pretos representam os valores reais do dataset, a linha azul representa os valores previstos pelo modelo e o sombreamento azul em torno da linha representa os limites superior e inferior da previsão.

Em seguida, o modelo é treinado com a função *fit*, detalhada anteriormente. Para rodar a ferramenta de diagnóstico de cross validation é utilizada a função *diagnostics.cross_validation* e são passados como parâmetros o modelo treinado e as variáveis *initial*, *period* e *horizon*.

Para imprimir o diagnóstico com o erro desejado é usada a função *plot_cross_validation_metric*, que recebe como parâmetro o output da função *diagnostics.cross_validation*, a variável *metric*, que define o tipo de erro a ser utilizado na medição, e a variável *rolling_window*, que define a razão de dias que será usada para calcular os erros usando médias móveis. Por padrão, a *rolling_window* é 10% do tamanho do dataset, o que significa que 10% das predições serão incluídas em cada janela.

Plotando-se o gráfico de cross validation referente a previsão feita na Figura 13 para o tipo de erro raiz do erro quadrático médio (RMSE), tem-se o gráfico da Figura 14.

6 Experimentos e Resultados

6.1 Introdução

Este capítulo apresenta os experimento realizados e a análise dos resultados obtidos na implementação do projeto. As simulações foram iniciadas com a rede neural recorrente baseada em LSTM. Foram gerados vários testes com parâmetros distintos. Durante os testes, foi sendo alterado o número de defasagem dos dados, o número de épocas a fim de

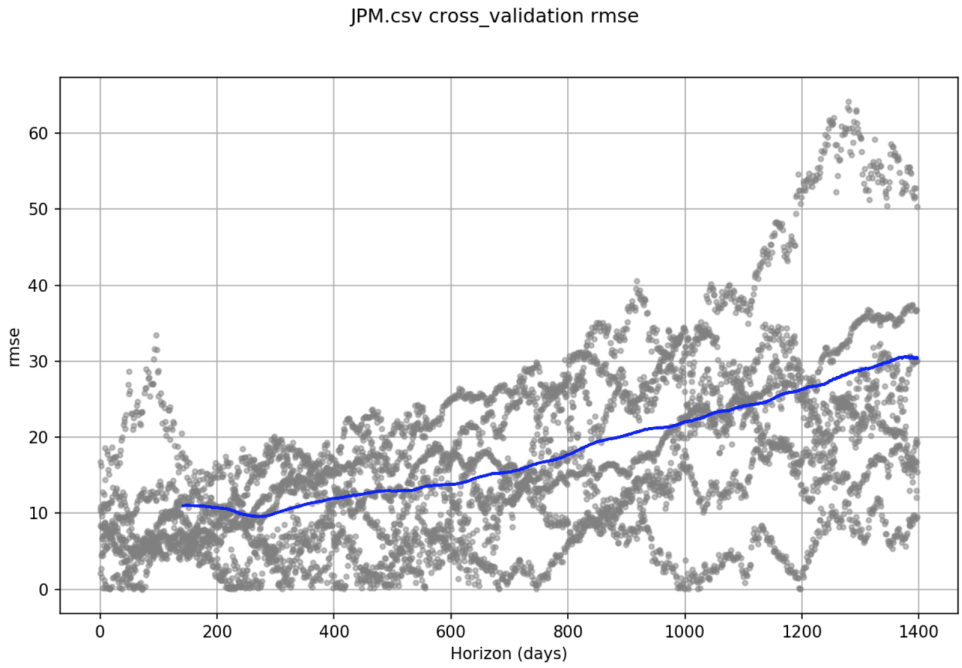

Figura 14: Gráfico de cross validation realizado pelo Prophet para o modelo treinado com os mesmos dados da Figura 13. Os valores das variáveis *initial*, *period*, *horizon* e *rolling_window* são, respectivamente, 60%, 5%, 10% e 10% do tamanho total do dataset. Foi utilizada a raiz do erro quadrático médio (RMSE) como medição.

atingir o *best fit* do modelo, as funções de ativação e ativação recorrente dos blocos LSTM, o número de perceptrons LSTM, etc. Em seguida, foi feita uma simulação da previsão e do erro RMSE obtidos pelo Prophet para o mesmo conjunto de dados usados para a rede neural.

O objetivo dos experiementos é entender o impacto dos diversos parâmetros na previsão dos dados. Além disso, gerar material suficiente para compararmos os resultados obtidos pela rede neural LSTM e aqueles do Prophet.

6.2 Rede Neural Recorrente

O primeiro teste com a rede neural recorrente baseada em LSTM foi realizado com a série histórica de preços da American Airlines Group. Esse dataset possui 3207 pontos distribuídos em uma base de tempo diária que vai de 27-09-2005 até 22-06-2018. A variável do dataset que foi utilizada na rede é a de abertura do preço da ação, assim, o sistema trata uma série temporal univariada.

Em seguida, a normalização *MinMaxScaler* é aplicada, além da defasagem dos dados em 3 unidades de tempo, *lags*. O dataset é dividido em treino e teste, sendo o treino os primeiros 70% da série temporal e o teste os últimos 30%.

Para a criação da rede neural nesse primeiro teste, foram instanciados 5 perceptrons

do tipo LSTM com função de ativação e ativação recorrente dos tipos *relu* e *hard_sigmoid*, respectivamente. Além disso, foi criado um perceptron do tipo *Dense* com 1 *unit* e com função de ativação linear para gerar a previsão do preço de abertura da ação.

O modelo foi treinado levando em consideração 5 *epochs* e *batch size* 100. Quando o gráfico da função de custo dos dados de treino e teste foram plotados, o resultado da Figura 15 foi observado, o que indica que o modelo está underfi pois ambas as curvas ainda estão em decaimentot.

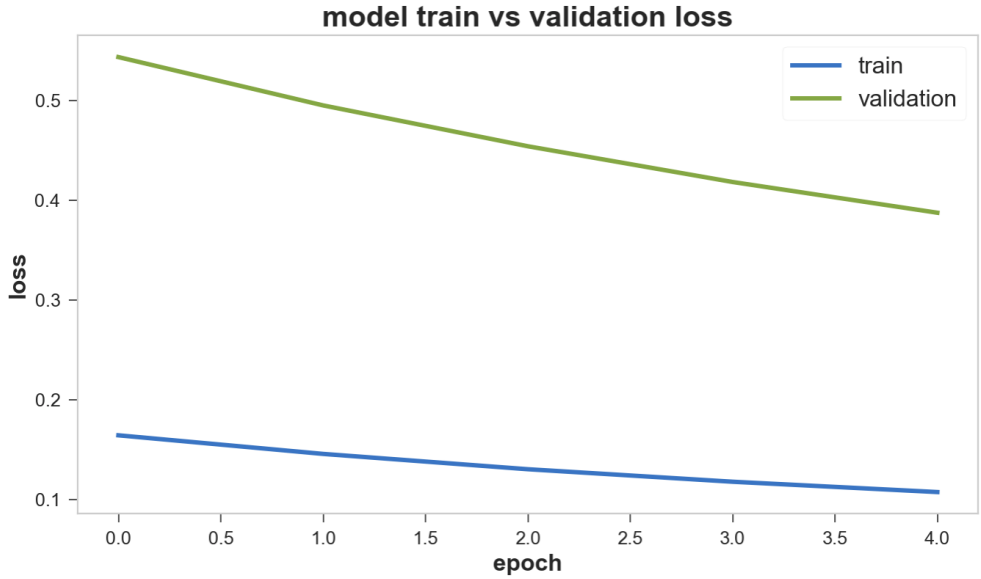

Figura 15: Gráfico da função de custo dos dados de treino e teste, da empresa American Airlines Group em função do número de épocas do modelo. Pode-se perceber que o modelo está apresentando *underfitting*.

Foi então aumentado o número de épocas para 12 para tentar deixar o modelo melhor treinado. O gráfico obtido (Figura 16) mostra que o modelo está próximo ao *best fit*, uma vez que as funções de custo do treino e do teste aproximam-se na época 12.

Em seguida, os dados de treino e teste foram passados como input para a rede neural prever os valores de abertura das ações. Para os exemplos de treino, o RMSE obtido foi de 7.64, enquanto que para os exemplos de teste, o RMSE foi de 9.29, o que faz sentido, uma vez que o modelo é treinado com os dados de treino e, portanto, deve apresentar melhor resultado na previsão desse grupo.

A Figura 17 mostra uma visualização dos dados obtidos pela predição da rede neural LSTM com base nos dados anteriores. A Figura 18 mostra a mesma curva, porém apenas para os dados de teste.

O segundo teste, tem como principal objetivo mudar o número da defasagem dos dados de 3 para 10 *lags*. Para obter o *best fit* o número de épocas foi reduzido para 3. Porém, como é possível observar na Figura 19, a curva de custo dos dados de treino cai muito abruptamente, o que pode não ser bom para o modelo.

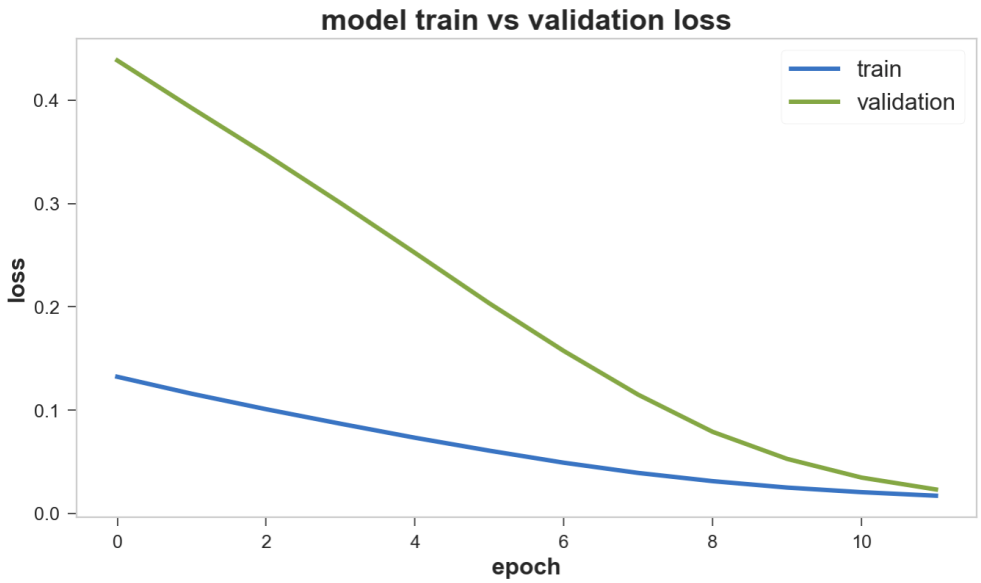

Figura 16: Gráfico da função de custo dos dados de treino e teste, da empresa American Airlines Group em função do número de épocas do modelo. Pode-se perceber que o modelo está em seu *best fit*.

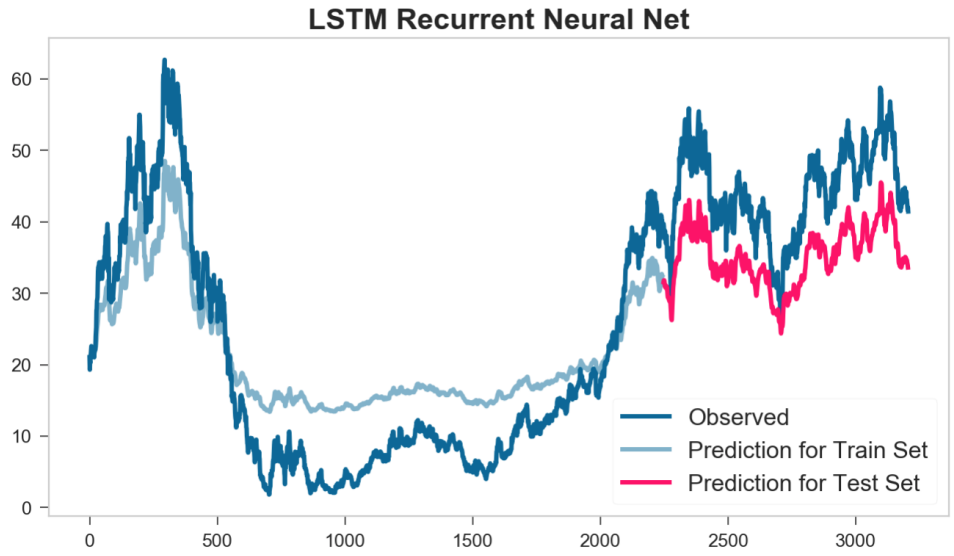

Figura 17: Gráfico da predição da rede neural LSTM em seu primeiro teste.

Nesse segundo teste, a rede obteve RMSE igual a 7.30 para os exemplos de treino e RMSE de 13.10 para os dados de teste. Em relação ao teste anterior, esse apresentou um fit melhor para o treino, o que faz sentido com o aumento de *lags*, mas tem uma piora na

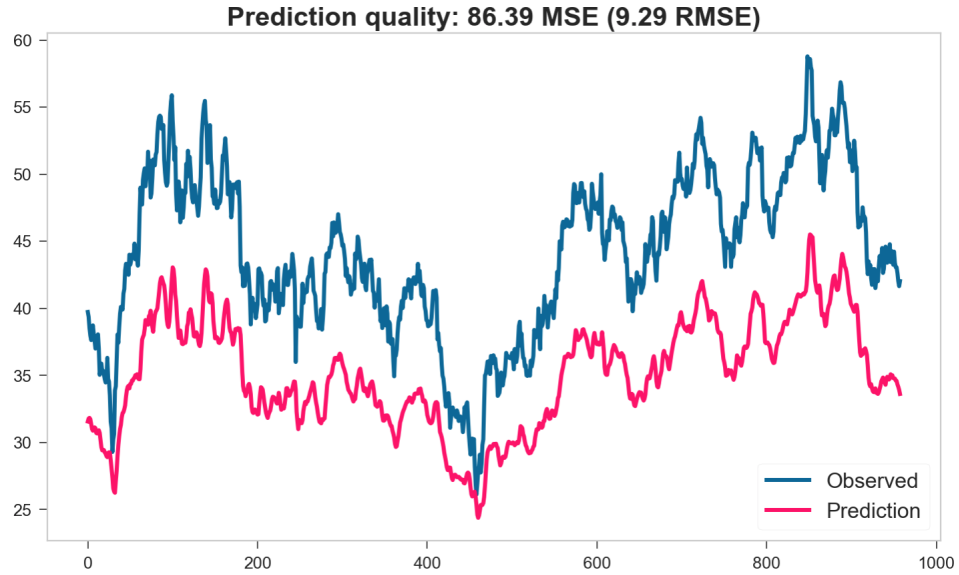

Figura 18: Gráfico da predição dos dados de teste da rede neural LSTM em seu primeiro teste.

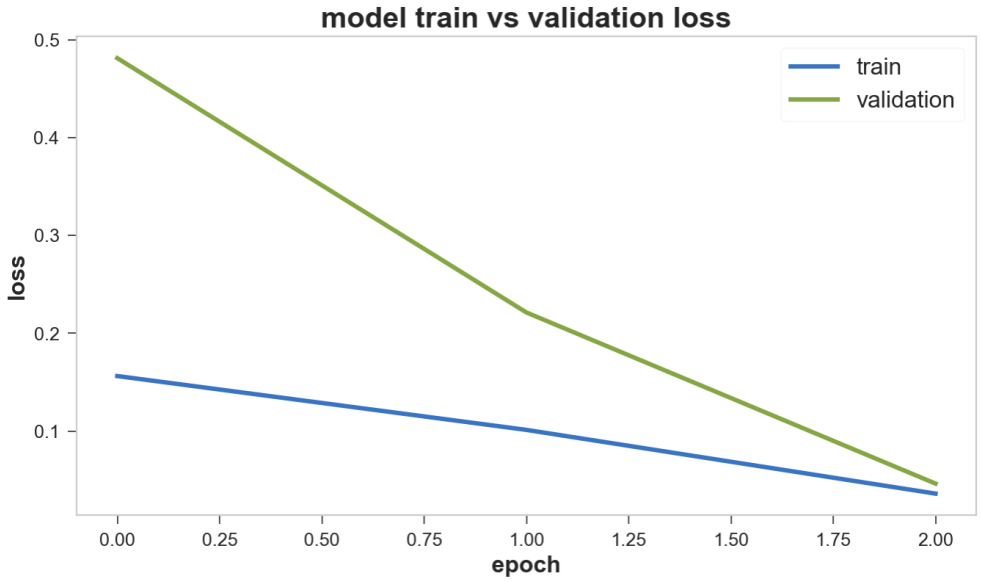

Figura 19: Gráfico da função de custo dos dados de treino e teste, da empresa American Airlines Group em função do número de épocas do modelo. Os dados utilizados nesse modelo estão com defasagem igual a 10 *lags*. Pode-se perceber que a curva de custo dos dados de teste cai muito abruptamente, o que pode não ser bom para a previsão de valores.

previsão dos dados de teste. Esse caso mostra que o aumento da defasagem dos dados não é benéfico para a assertividade do modelo.

Para o próximo teste, o valor de defasagem de 3 *lags* foi retomado, porém, agora as funções de ativação e ativação recorrente dos blocos LSTM são *softmax* e *relu*, respectivamente. Isso porque essas funções são teoricamente mais adequadas para esse tipo de problema. O número de épocas escolhido foi 10, por apresentar o melhor fit.

O erro deste experimento para os dados de treino foi de 6.20 RMSE, e para os dados de teste, de 8.09 RMSE. Já é possível ver uma melhora considerável nesse teste em relação aos anteriores, o que mostra a importância em escolher o melhor tipo de função de ativação.

O último teste leva em consideração um número muito maior de blocos LSTM. Enquanto os testes anteriores tinham 5 perceptrons LSTM, esse possui 300. O número de épocas ideal é 33, alcançando assim o melhor fit. O erro dessa rede para os dados de treino é de apenas 3.35 RMSE, e para os dados de teste, de 5.21 RMSE. Isso mostra a importância da complexidade da rede para prever resultados mais assertivos. Os gráficos da previsão dos preços de abertura da American Airlines Group por essa rede pode ser observado nas Figuras 20 e 21.

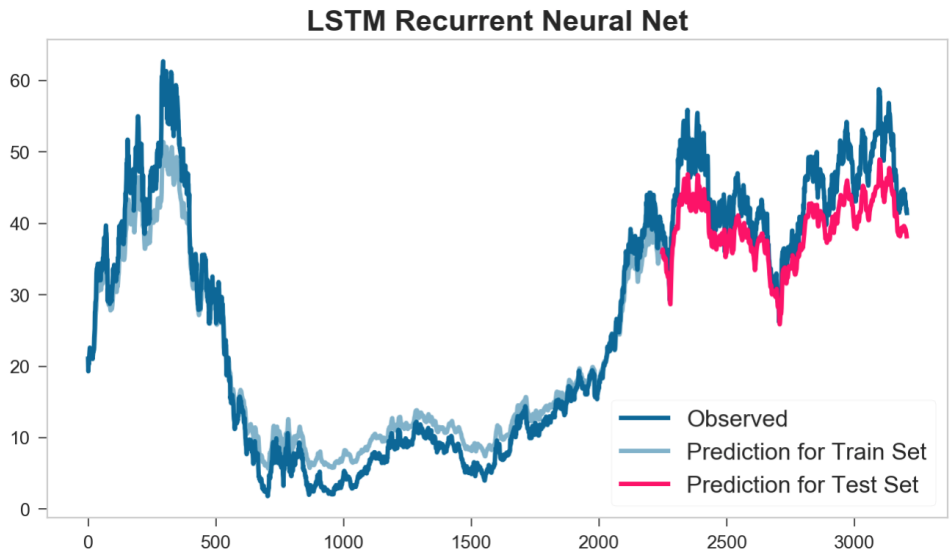

Figura 20: Gráfico da previsão do modelo de rede neural com 300 perceptrons LSTM com funções de ativação e ativação recorrente iguais a *softmax* e *relu*, respectivamente.

A partir dos experimentos foi possível perceberque todos os modelos tem uma taxa de acerto de direção muito alta. Mesmo os primeiros testes, em que os modelos possuem um erro RMSE considerável. Uma taxa de acerto de direção implica que o modelo pode não acertar a previsão no valor correto, mas ele consegue identificar melhor os períodos em que uma ação vai subir ou descer. Claro que fatores externos que interfiram diretamente no valor das ações, não serão previstos nesse tipo de modelo.

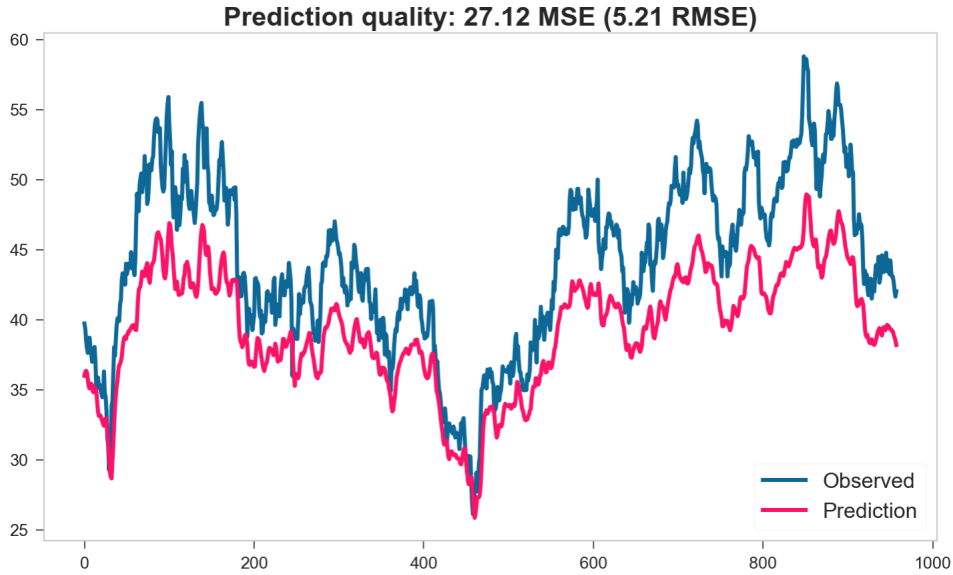

Figura 21: Gráfico da previsão dos dados de teste do modelo de rede neural com 300 perceptrons LSTM com funções de ativação e ativação recorrente iguais a *softmax* e *relu*, respectivamente.

6.3 Prophet

Para os experimentos com Phophet, a mesma base de dados e o mesmo arquivo utilizado para os testes com as redes neurais LSTM referentes as séries históricas do preço de abertura das ações da American Airlines Group foram aplicados.

Através do modelo de predição do Prophet, quando dado como entrada essa série temporal univariada, pedido para fazer a previsão do modelo inteiro e ainda mais 2 anos de dados futuros, que não existem mapeados, o gráfico da Figura 22 é obtido.

Submetendo o dataset ao método de cross validation apresentado anteriormente com os parâmetros *initial*, *period*, *horizon*, *rolling_window* e *error_metric* iguais a 60%, 5%, 10%, 10% e RMSE, respectivamente, o gráfico do erro calculado pela raiz do erro quadrático médio é representado na Figura 23.

7 Conclusão

A tarefa de prever preços no mercado de ações é extremamente complexa e envolve muito mais variáveis do que simplesmente a série histórica de preços. No entanto, é possível perceber bons resultados utilizando-se de redes neurais recorrentes com base em LSTM sendo treinados por séries temporais. Também é possível perceber resultados razoáveis através da ferramenta open source do Facebook, o Prophet.

Neste projeto, os resultados alcançados com as redes neurais recorrentes foram muito positivos, especialmente pelo fato de ser possível variar muitas característica da rede e

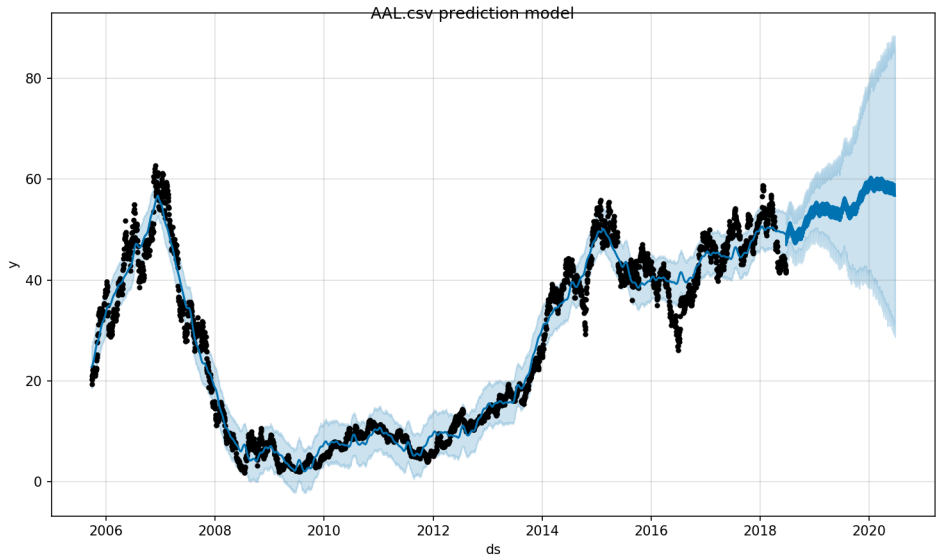

Figura 22: Gráfico da previsão da abertura das ações da American Airlines Group pelo Prophet. Também é tentada fazer a previsão para os próximos dois anos de dados não mapeados, não podendo ser calculado o erro desse período.

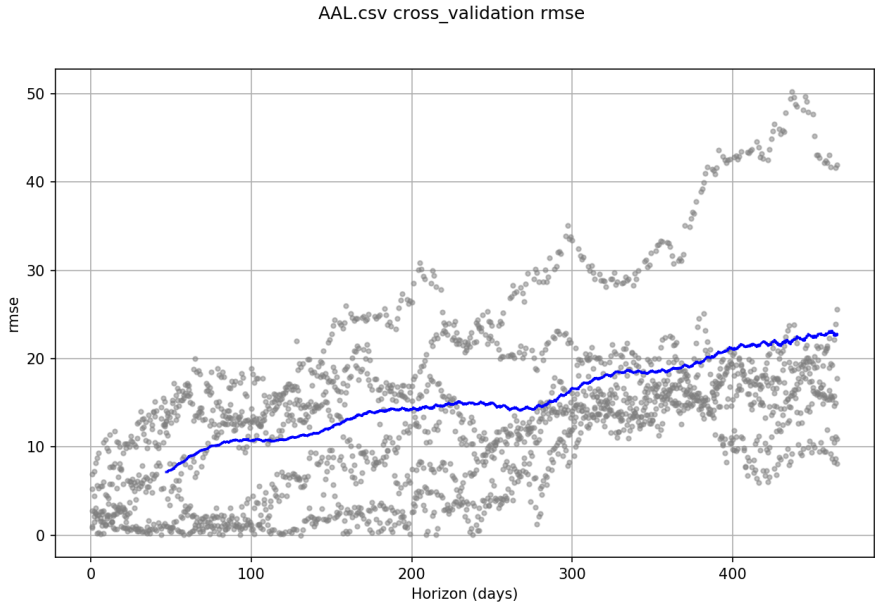

Figura 23: Gráfico do erro RMSE analisado pelo método de cross validation realizado sobre os dados de abertura das ações da American Airlines Group.

visualizar as diferentes performances de cada cenário. Foi possível variar a defasagem dos dados, as funções de ativação da rede, o número de perceptrons etc. Ao final, chegou-se a um cenário ótimo com erro no dataset de treino, que era de 30% do dataset total, de 5.21 RMSE. No entanto, mesmo a previsão não sendo extremamente assertiva, foi percebido pela Figura 21, que a rede neural consegue prever muito bem a tendência das ações, ou seja, quando ela irá subir e quando irá cair, o que pode sim auxiliar muito a tomada de decisões de investidores.

O Prophet é uma ferramenta com menos liberdade de variação dos parâmetros, uma vez que age como uma caixa-preta em muitas das operações. Ainda assim, é possível ver um erro relativamente bom, menor que 10 RMSE para horizontes não muito distantes, por volta de 50 dias. Essa ferramenta ainda possui diversos utilitários que não foram explorados no projeto, como análise de tendências.

Como melhorias, poderíamos treinar outros tipos de redes neurais recorrentes e utilizar outras ferramentas de previsão disponíveis no mercado, como é o caso da ARIMA. Ainda, é possível levar em consideração outros variáveis na previsão de preços de ações. Uma ação que seria muito interessante, seria fazer uma análise de sentimento em notícias relacionadas as empresas, como report de bancos, e identificar se as empresas estão indo bem ou mal com base no texto analisado. A conclusão retirada desses reports pode ser adicionada como um conjunto de variáveis para a rede neural, a fim de deixar o modelo mais preciso.

O link com os algoritmos de previsões no mercado de ações em Python tanto para a rede neural recorrente baseada em LSTM quanto para a implementação com o Prophet podem ser encontrados no seguinte git: Predicting Stock Prices with Machine Learning Algorithms.

Referências

[1] Gabriel, F. S., Ribeiro, R. B., and de Sousa Ribeiro, K. C. (2013), *Hipóteses de mercado eficiente: um estudo de eventos a partir da redução do IPI*, Revista de Gestão, Finanças e Contabilidade.

[2] do Carmo Roque, R., and de Mello, F. L. (2009), *Estudo sobre a empregabilidade da previsão do índice BOVESPA usando Redes Neurais Artificiais.*

[3] Malkiel, B. G., and Fama, E. F. (1970), *Efficient Capital Markets; a review of theory and empirical work*, Journal of Finance.

[4] Dorffner, G. (1996), *Neural networks for time series processing*, Neural Network World.

[5] Lawrence, R. (1997), *Using neural networks to forecast stock market prices*, University of Manitoba.

[6] Srivastava, T. (2018), "A Must-Read Introduction to Sequence Modelling (with use cases)", available at: www.analyticsvidhya.com/blog/2018/04/sequence-modelling-an-introduction-with-practical-use-cases/ (acessed 28 June 2018).

[7] Sinha, N. (2018), "Understanding LSTM and its Quick Implementation in Keras for Sentiment Analysis", available at: https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47 (acessed 28 June 2018).

[8] Cavaioni, M. (2014), "DeepLearning series: Sequence Models", available at: https://medium.com/machine-learning-bites/deeplearning-series-sequence-models-7855babeb586 (acessed at 27 June 2018).

[9] NG, A., Katanforoosh, K. and Mourri, Y. B. (2018), "Sequence Models", available at: www.coursera.org/learn/nlp-sequence-models (acessed 28 June 2018).

[10] Olah, C. (2015), "Understanding LSTM Networks", available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (acessed 28 June 2018).

[11] Chen, E. (2017), "Exploring LSTMs", available at: http://blog.echen.me/2017/05/30/exploring-lstms/ (acessed 29 June 2018).

[12] Hochreiter, S. and Schmidhuber, J. (1997), *Long Short-Term Memory*, Johannes Kepler University Linz.

[13] Taylor, S. J., and Letham, B. (2017), *Forecasting at scale*, The American Statistician, PeerJ Preprint.

[14] Facebook Open Source (2018), "Prophet Documentation - Diagnostics", available at: https://facebook.github.io/prophet/docs/diagnostics.html (acessed 2 July 2018).

[15] Brownlee, J. (2017), "How to Use Timesteps in LSTM Networks for Time Series Forecasting", available at: https://machinelearningmastery.com/use-timesteps-lstm-networks-time-series-forecasting/ (acessed 30 June 2018).

[16] Zhang, C., Yan, J., Li, C., Rui, X., Liu, L., and Bie, R. (2016), *On estimating air pollution from photos using convolutional neural network*, Proceedings of the 2016 ACM on Multimedia Conference.

[17] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012), *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems.

[18] Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010), *Recurrent neural network based language model*, Eleventh Annual Conference of the International Speech Communication Association.

[19] Brownlee, J. (2017) "How to Diagnose Overfitting and Underfitting of LSTM Models", available at: https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/ (acessed 30 June 2018).