Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

Bacharelado em Sistemas de Informação

Estudo da Acurácia das Redes Neurais Recorrentes LSTM para Séries Temporais Financeiras PETR4

Rodolfo Viegas de Albuquerque

Recife

setembro de 2020

Resumo

As séries temporais são um conjunto de dados ordenados no tempo, vários campos do conhecimento utilizam-nas para seus estudos. Mas, não só as ciências usam esse tipo de análise, o mercado financeiro se vale dos estudos de séries temporais para a tomada de decisão, por exemplo, a compra ou venda de ações. Para analisar as séries são necessários métodos cada vez mais acurados. As redes neurais recorrentes (RNN) do tipo LSTM, dada sua estrutura de acumular e repassar informações anteriores, podem suprir essa exigência de melhor acurácia. O presente estudo visa aplicar a RNN às séries temporais da Petrobras do tipo preferenciais (PETR4) comparando a acurácia com o modelo *multilayer perceptron* (MLP), assim verificando qual modelo possui mais precisão nas predições dessas séries em dado corte histórico.

Palavras-chaves: Deep Learning; Séries Temporais; Redes Neurais Recorrentes; Finanças; Petrobras; PETR4.

1. Introdução

1.1 Apresentação e Motivação

As séries temporais estão em todos as campos do conhecimento: economia, finanças, climatologia etc; possuindo grande importância para a ciência e para o mercado. Brockwell e Davis (2016) definem-nas como "um conjunto de observações x_t, cada uma registrada num tempo específico *t*".

Operadores do mercado financeiro se valem da análise das séries temporais, estas têm o comportamento (tendências, sazonalidades) analisado por aqueles para tomada de decisão - seja compra ou venda de algum ativo financeiro (ações, títulos, moedas). A Petrobras, como exemplo, possui na bolsa de valores dois tipos de ações: as do tipo ordinárias, com o símbolo PETR3, que dá ao dono poder de voto em assembléias de acionistas; e a do tipo preferencial, cujo símbolo é PETR4, estas ao contrário daquelas não que possuem direito a voto, mas ganham um valor maior da partilhas dos dividendos (fatia dos lucros repartidos entre os acionistas). Os dados dos valores dessas ações, do modo que são organizados, são séries temporais. Para a análise das séries, como a PETR3 e PETR4, os métodos de aprendizado profundo podem ser usados para verificar como ativos comportam-se com o passar do tempo.

Falar em aprendizado profundo (*deep learning*) é falar em redes neurais artificiais (RNA ou ANN). Estas são algoritmos que funcionam como aproximadores universais de funções, foram desenhadas para simular o funcionamento das redes no cérebro. Elas são compostas por um conjunto de neurônios organizados em camadas que recebem dados (entradas ou *inputs*) e geram novos dados (saídas ou *outputs*). Cada neurônio é ligado a outros neurônios, antes e após, através de pesos (parâmetros) que definirão os valores que serão passados a eles e deles para neurônios à frente. As redes, ao receberem os dados (entradas e saídas), aprendem como esses se comportam - definindo regras - podendo generalizar para novos valores de entrada não vistos e retornando novas saídas.

Há várias arquiteturas de RNA, para lidar com valores em sequência (textos ou séries temporais) o tipo Recorrente (RNN) é o mais adequado. Essas redes, com algumas modificações no algoritmo de *backpropagation*, conseguem reter mais informações pretéritas e passá-las aos neurônios à frente.

Conforme o exposto este artigo propõe utilizar as Redes Neurais Recorrente e verificar se ela consegue ter uma melhor acurácia que o modelo de rede neural artificial "multilayer perceptron" (MLP) para a modelagem e predição das séries temporais PETR4, assim verificando para este caso específico se elas podem ou não ser mais uma opção de ferramenta para os interessados em negociar as ações PETR4.

1.2 Objetivos Gerais

Como objetivo geral será estudo e aplicação das redes neurais recorrentes (RNN) nas séries temporais das ações da Petrobras PETR4.

1.3 Objetivos Específicos

Descrever o funcionamento do algoritmo "backpropagation" nas redes neurais;

Descrever o funcionamento interno da LSTM;

Tomar notas dos resultados da acurácia e outras métricas na aplicação das RNN para predição das séries temporais das ações da Petrobras PETR4.

2. Trabalhos Relacionados

Papadoulos et al. (2016) fizeram um estudo das séries temporais PETR4 no intervalo de 2008 a 2013, sendo que os dados estimados são de 2012 a 2013, intervalo este que, segundo os autores, possui menor oscilação que o intervalo de 2008 a 2013. Os autores realizaram 10 experimentos com redes neurais artificiais multicamadas (MLP), no melhor resultado a variância para o intervalo estimado foi de 3,115080788, a variância dos valores reais é 3,136943537, com diferença de 0,021862749. O autor do artigo também utilizou a média da diferença de variâncias dos 10 modelos, resultando no valor de 0,353725489.

3. Referencial Teórico

3.1 Séries Temporais

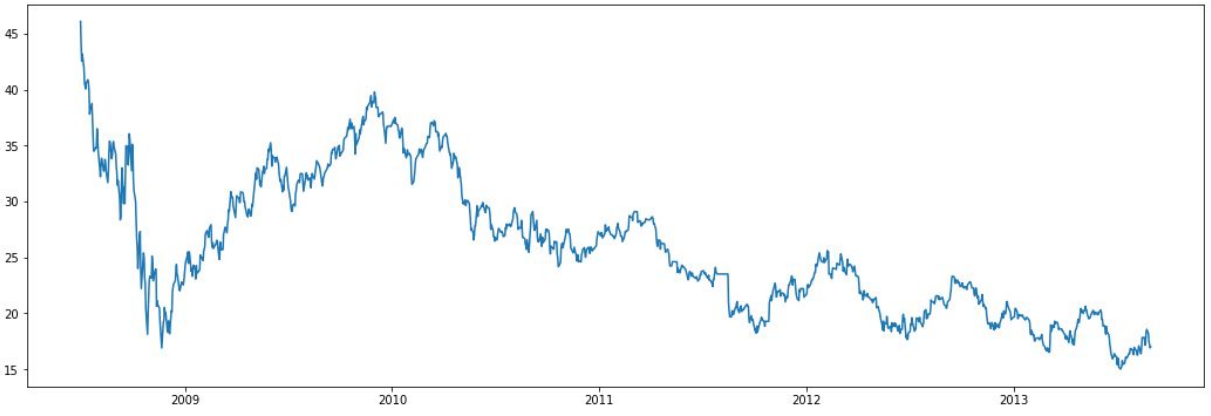
Brockwell e Davis (2016) definem uma série temporal como "um conjunto de observações x_t, cada uma registrada num tempo específico *t*." Há dois tipos, grosso modo, de dados de séries temporais: discretos e contínuos. Séries discretas são aquelas que os dados são discretos, podendo ser contados. Já as contínuas são compostas de dados contínuos, ou seja, que têm valores infinitos dentro de um intervalo de valores qualquer.

São muitos os campos que se valem das séries temporais; como: finanças, economia, climatologia, sociologia etc. Seja para a tomada de decisão ou validação de hipóteses essas áreas do saber utilizam as séries temporais e as técnicas para analisá-las e tecer conclusões. Em finanças, por exemplo, operadores de corretoras estudam o comportamento de séries com valores dos preços de ações, valores de moedas, ou títulos.

3.2 Ações da Petrobras

O site Investidor Petrobras afirma que há dois tipos de ações da Estatal: as ações ordinárias, cujo símbolo é PETR3 e as ações preferenciais, o seu símbolo é PETR4. As do tipo ordinária dão direito ao detentor de votar em assembléias de acionistas; já a do segundo tipo não possue direito a voto, mas o detentor na hora da partilha dos lucros (dividendos) recebe uma fatia maior que o detentor da PETR3. Ambos, ainda segundo o site, são majoritariamente negociadas na Bolsa de Valores de São Paulo.

Figura 1 - Ações da PETR4 de julho de 2008 a setembro de 2013 (dados preenchidos)



Compilação do autor

3.3 Deep Learning

Deep learning (também chamado de aprendizado profundo), como explica Andrew Trask (2019), é um subconjunto de *machine learning*, que é uma área da inteligência artificial que visa criar algoritmos capazes de aprender. Deep learning, com um subcampo dos métodos de *machine learning*, "primeiramente usando redes neurais artificiais, que são uma classe de algoritmos vagamente inspirado pelo cérebro humano " Trask (2019).

A indústria e os serviços utilizam os algoritmos de redes neurais em variadas aplicações como: visão computacional, reconhecimento de fala, traduções de textos e previsão de séries temporais.

3.3.1 Perceptron

A unidade base para as redes neurais é chamada de perceptron, que foi criada por Frank Rosenblatt nos anos 50. Essa rede recebe, segundo Mitchell (1997), um vetor de entradas x_i cada um sendo multiplicado por parâmetros w_i, também chamados de pesos, com o adicional de w_0 , o viés (ou *bias*), sempre multiplicado por 1. Se o valor calculado for maior que zero será retornado 1; senão -1.

$$o(x_1,...,x_n) = \begin{cases} 1, \text{ se: } w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 \ ,...w_n \cdot x_n > 0 \\ -1, \text{ do contrário} \end{cases}$$

Alternativamente pode-se utilizar uma visão vetorial:

$$o(\vec{x}) = sgn(\vec{x} \cdot \vec{w})$$

onde

$$\text{sgn}(y) = \begin{cases} 1, \text{ se } y > 0 \\ -1 \text{ do contrário} \end{cases}$$

Para aprender com os dados o perceptron, de modo iterativo, corrige seus pesos em relação ao acerto de seu *output o* em comparação com ao valor real, ou *target*, que define a classe da observação.

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

Em que η é um valor real conhecido como taxa de aprendizado, Mitchell (1997) ainda afirma que, via demonstração, se a taxa de aprendizado for suficientemente pequena e os dados linearmente separáveis o perceptron encontrará uma regra que consiga classificar os dados.

3.3.2 Regra Delta

Se os exemplos não são linearmente separáveis o algoritmo de treino do perceptron pode falhar, para suprir tal dificuldade há regra delta, esta usa o gradiente descendente, para provê melhores resultados. A introdução da regra delta é importante porque essa será base para o algoritmo de retropropagação.

Mitchell (1997) afirma que um melhor entendimento da regra delta é treinar um perceptron "não-liminarizado", ou seja, sem a ideia da função retornando 1 ou -1 mediante o resultado o produto do vetor de pesos pelo vetor de entradas. Neste caso pode-se representar o cálculo como:

$$o(\vec{x}) = \vec{x} \cdot \vec{w}$$

A regra, para treinar o vetor de pesos necessita de uma medida que avalie o quão próximos o valor predito do *target* está. São várias equações que mensuram o erro de treino, para este artigo usaremos esta:

$$E(\vec{w}) = \sum_{d \in D} (t_d - o_d)^2$$

Onde D é o conjunto de exemplos para o treino com d um exemplo qualquer, t_d o *target*, o_d a saída (*output*) calculada para cada exemplo d.

Para calcular, baseado na função erro exposta acima, é preciso derivar essa para cada peso em questão, criando assim um vetor de derivadas parciais da função erro para respectivos pesos, dando uma direção que minimize o erro como um todo e encontre uma regra geral que classifique corretamente o predito em comparação com o real.

$$\nabla E(\vec{w}) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}]$$

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

Essa regra pode também ser representada para cada peso do vetor de pesos.

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Para obter a derivada do erro *E* para respectivo peso:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_n} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d})$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

O variável x_{id} representa uma característica x_i de um exemplo *d*. E, com a fórmula acima, podemos ter a atualização dos pesos:

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

De modo iterativo, calcule, para cada exemplo, o valor predito pelo perceptron, calcule em seguida o Δw_i para atualizar os pesos; repita o processo até que a quantidade de épocas (rodadas dentro de um laço repetindo o processo exposto) acabe.

3.3.3 Multilayer Perceptron

Para a etapa de desenvolver uma rede multicamadas é necessário escolher qual será a unidade que a baseie. Tom Mitchell (1997) recomenda uma combinação melhorada do perceptron de Rosenblatt com a regra delta, com a diferença que a saída não será mais uma

função linear, que apenas varie entre -1 e 1, mas que seja não-linear. A função sigmóide, ou função logística, serve para o novo papel e com a linearidade é possível achar regras que descrevem problemas não-lineares, que são os mais comuns no mundo real.

Esta unidade nova será representada com as seguintes equações:

$$o = \sigma(\vec{w} \cdot \vec{x})$$

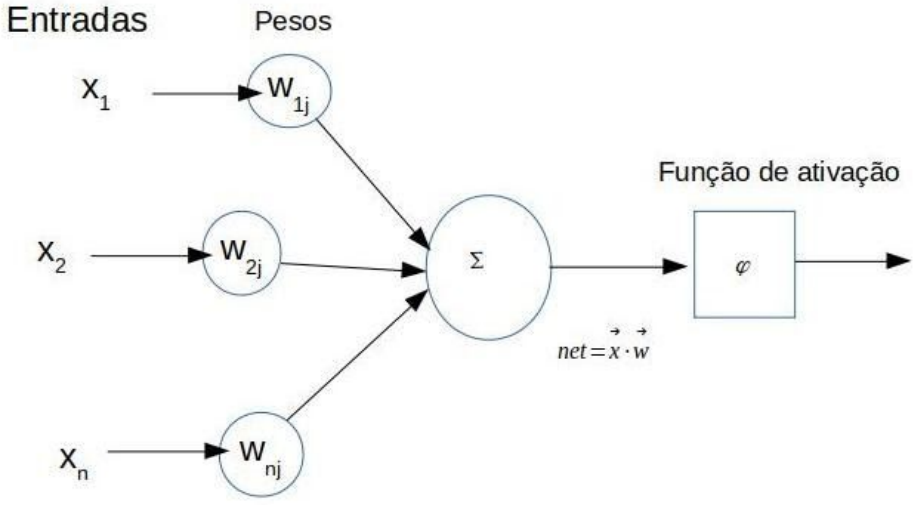
Em que a σ representa a função sigmóide e y, ou *net*, o valor de resultante da multiplicação das entradas pelos pesos:

$$\sigma(y) = \frac{1}{1+e^{-y}}$$

A sigmóide, que será utilizada para modelar problemas não-lineares, pode fazer bem o papel. Por ser contínua é diferenciável, assim podendo aplicar o gradiente descendente. Sua derivada é representada como:

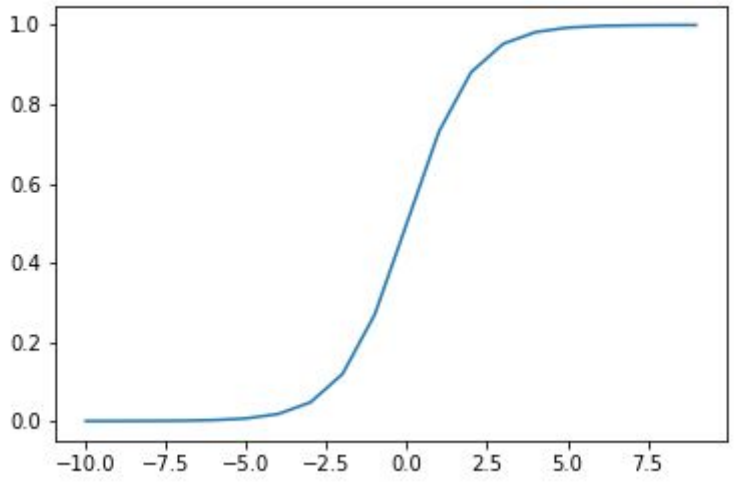
$$\frac{\partial \sigma(y)}{\partial y} = \sigma(y)(1 - \sigma(y))$$

Figura 2 - Perceptron



Fonte: Compilação do autor

Figura 3 - Curva Sigmóide



Fonte: Compilação do autor

3.3.4 Backpropagation

O processo inicia com o *feedforwards* que são entradas x_i multiplicando com os pesos w_i e somando com o viés b_0, terminando comparado o erro do previsto pelo *target*, via uma função de custo. A etapa de retropropagação é similar à usada com a regra delta, mas com a diferença de que mais pesos serão atualizados e o uso da função de ativação.

A função de erro será similar a usada anteriormente, mas considerando uma camada de saída com mais de um neurônio (visando mais áreas para o MLP modelar) Mitchell (1997) a reescreve como:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

Em que "*outputs* é o conjunto de unidades de saída na redes, e t_{kd} e o_{kd} são o *target* e o valor da saída associados com o *k*-ésima unidade de saída e o exemplo de treino *d*." Michell (1997).

Para propagar os erros na volta é necessário derivar a função de erro *E* em relação aos pesos da última da camada e das camadas escondidas. Estas são as derivadas parciais do erro *E* em relação aos pesos da última e da primeira camada escondida:

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

Em que δ é $-\frac{\partial E}{\partial net}$ seja para os pesos *k*, da camada de saída, ou *h*, as camadas escondidas.

Tom Mitchell (1997) descreve o algoritmo:

Backpropagation(*training_exemples*, η, n_{in}, n_{out}, n_{hidden}):

Cada exemplo no conjunto, que será atribuído ao parâmetro training_exemples, terá a forma $\langle \vec{x}, \vec{t} \rangle$. $\vec{x}$ *é o vetor de valores de entrada da rede e* $\vec{t}$ *os valores target para as respectivas entradas.*

A taxa de aprendizado é representada pela letra grega η, n_{in} *o número de entradas,* n_{out} *número de neurônios na camada de saída e* n_{hidden} *número de neurônios na camada escondida.*

- Crie um rede feedforward com n_{in} entradas, n_{hidden} unidades escondidas e n_{out} unidades de saída;
- Inicialize os pesos com valores aleatórios pequenos (entre -5 e 5, p. ex.)
- Até que a condição de término seja satisfeita, faça
    - Para cada $\langle \vec{x}, \vec{t} \rangle$ no *training_exemples*, faça

      1 - alimente a rede com um vetor $\vec{x}$ e calcule a saída o_{u} de cada unidade *u* na rede;

      2 - Para cada unidade *k* na última camada, calcule o termo de erro δ_k :
      $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
      3 - Para cada unidade na camada escondida, calcule o termo de δ_h :
      $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$
      4 - Atualize os pesos w_{ij} da rede:
      $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$
      onde:
      $$\Delta w_{ji} = \eta \delta_j x_{ji}$$

3.4 LSTM (Long Short Term Memory)

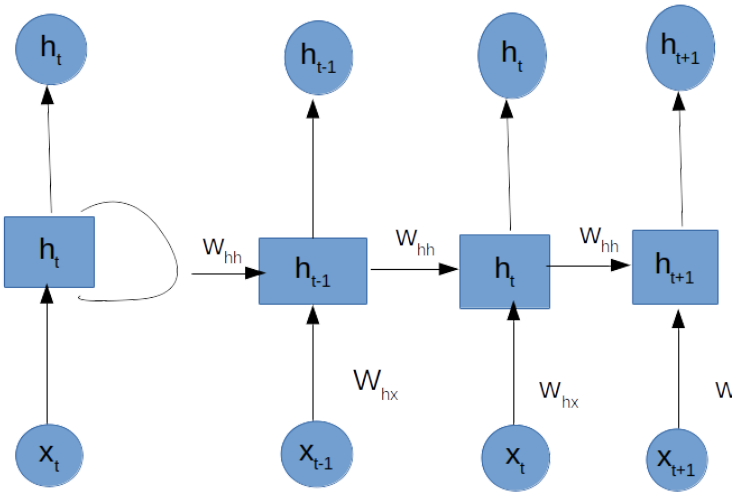
As redes neurais recorrentes são uma modificação do algoritmo “backpropagation” que consegue armazenar informações do passado, através de novo mecanismo o “estado escondido” h_t e mais parâmetros aprendendo e generalizando para sequências - sejam textos ou séries temporais. Uma importante característica das RNNs é o compartilhamento dos parâmetros, que segundo Goodfellow, Bengio e Courville (2016) é o que fará com que a rede possa aprender e generalizar sequência de variados tamanhos e formas. Mas essa rede sofre para aprender longas sequência por causa do “vanishing gradient” ou “exploding gradient”. A LSTM é uma versão melhorada das redes neurais recorrentes que lida melhor com tais problemas.

O neurônio da RNN compartilha os mesmos pesos para calcular os novos *input* x_t e os novos estados h_t. A equação que modifica os neurônios:

$$h_t = tanh(W_{hh}h_{t-1} + W_{hx}x_t)$$

Mas a cada novo estado h_t calculado o gradiente tende a desaparecer ou explodir.

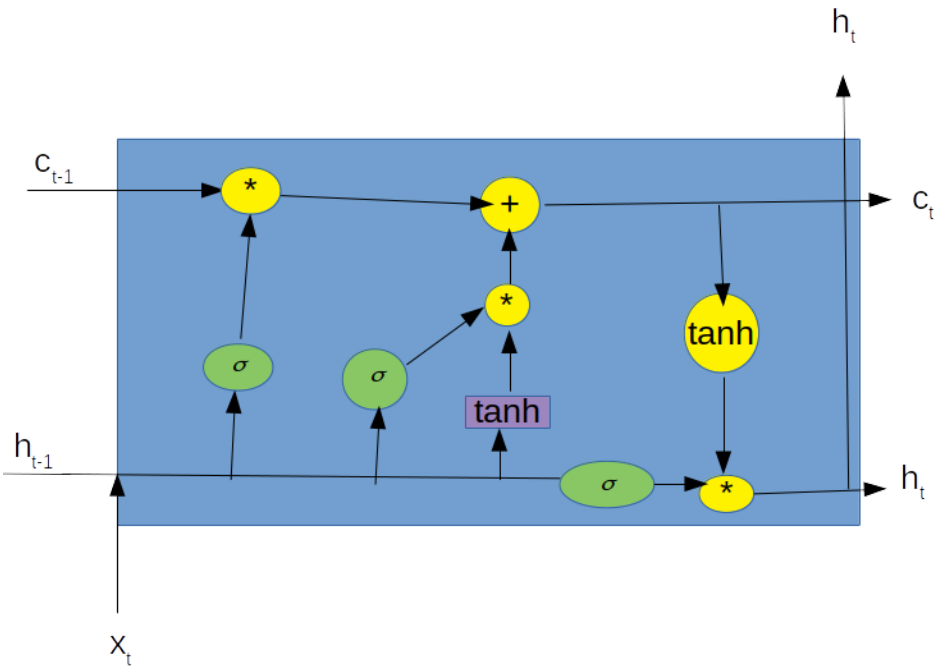
Figura 4 - Rede Neural Recorrente



Fonte: Compilado pelo autor

As LSTMs possuem um novo estado: o *Cell State* - que segundo Olah (2015) é a chave para o sucesso dessas redes; além de três “portas” que controlam as novas informações que podem ou não ser passadas ao *Cell State* e um “candidato” a novo *Cell State*. Essas quatro últimas inovações funcionam como camadas comuns: com pesos, entradas e funções de ativação (tangente hiperbólica para o candidato e sigmóide para as portas)

Figura 5 - representação interna unidade de memória da Long Short Term Memory



Fonte: Compilado pelo autor

4.0 Metodologia

Inicialmente os dados, em arquivo CSV, foram retirados do *site Yahoo! Finance* do período de janeiro de 2000 a junho de 2020. Os dados são divididos em sete colunas, mas este trabalho, junto com o trabalho relacionado, tem como foco a coluna de valor de *fechamento* (Close). Como Papadoulos et al. escolheram o corte de 2008 a 2013, foi aplicada a LSTM nesse intervalo de tempo.

Há dados faltando no *dataset*, para lidar com tal dificuldade foi feito um processo de imputação do valor à frente. Tal prática, segundo a documentação do Pandas, é bastante comum para lidar com valores faltantes em séries temporais.

Em seguida a série foi particionada em duas: o conjunto de treino e o conjunto de teste - este possuindo 20% dados da série; aquele 80%.

Após limpeza e divisão dos dados é necessário verificar qual forma que esses serão escalados. Escalar dados é ajustar os valores entre um intervalo, geralmente entre 0 e 1. De acordo com Brownlee (2017) projetos de *machine learning* funcionam melhor quando são

normalizados ou estandardizados. Brownlee recomenda que dados que se não distribuam normalmente devem ser normalizados (se esses têm distribuição normal, então devem ser estandardizados). Para saber se a série deve ser normalizada ou estandardizada é possível através de: visual, em que se verifica se a distribuição da série é próxima na forma à distribuição normal e com testes como o K-quadrado e o teste Shapiro-Wilk. Se o valor *p* resultante dos testes for menor ou igual a 0,05 a série não é normal. Finalizando, assim, a fase de pré-processamento.

Na parte de seleção de arquiteturas de redes neurais este trabalho utilizou seleção de modelos aleatória, selecionando especificamente a configuração de hiperparâmetros: quantidade de camadas (de 1 a 3), quantidades de neurônios nas camadas (de 11 a 150), função de ativação (*mse* e *mae*), número de épocas (de 5 a 20) e taxa de aprendizado (de 0,001 a 0,005). Setenta modelos foram treinados e validados. Após a seleção de um modelo há um segundo particionamento, agora no conjunto de treino; para validar o modelo é necessária a validação cruzada de séries temporais, esta, como recomenda Brownlee (2016) se dá com vários conjuntos de treino e de validação. O tamanho do conjunto de validação terá tamanho fixo enquanto o de treino aumentará mediante a quantidade de operações desejadas. Foram escolhidas 5 etapas de treino e validação.

A cada vez que os dados eram divididos entre treino e validação, estes conjuntos foram normalizados - para valores entre -1 e 1 correspondendo a função de ativação da LSTM, a tangente hiperbólica - de modo separado. Para em seguida serem preparados para o treino criando uma matriz **X** com cinco colunas com preços de ações de cinco dias seguidos - de x_t a x_{t-5}, correspondendo ao tamanho do *lag* que Papadoulos et al. (2016) utilizaram.E também vetor **y**, que será o *target*, com valores do sexto dia.

Hiperparâmetros sorteados e dados divididos há o treinamento da rede. Após esta fase a rede será avaliada com o conjunto de validação, que terá transformado em uma matriz com *lag* igual a 5 e a parte separada como *target*. O objetivo da validação é que a rede preveja os valores após cada linha de 5 dias, ou seja o do 6º dia. Todos esses valores serão comparados, após serem desnormalizados, aos valores do *target*, valores reais, do conjunto de validação. A comparação se dá pelas métricas de avaliação erro médio absoluto (MAE) e raiz do erro médio quadrático (RMSE).

Após os 70 modelos serem treinados e validados os 10 que tiverem os menores MAE e RMSE passaram a fase de teste que é predição dos valores *target* do conjunto de teste. Este

foi, como os outros dados, normalizado entre -1 e 1 transformado em matriz de cinco colunas e criando o seu próprio vetor **y** *target.*

5. Procedimentos

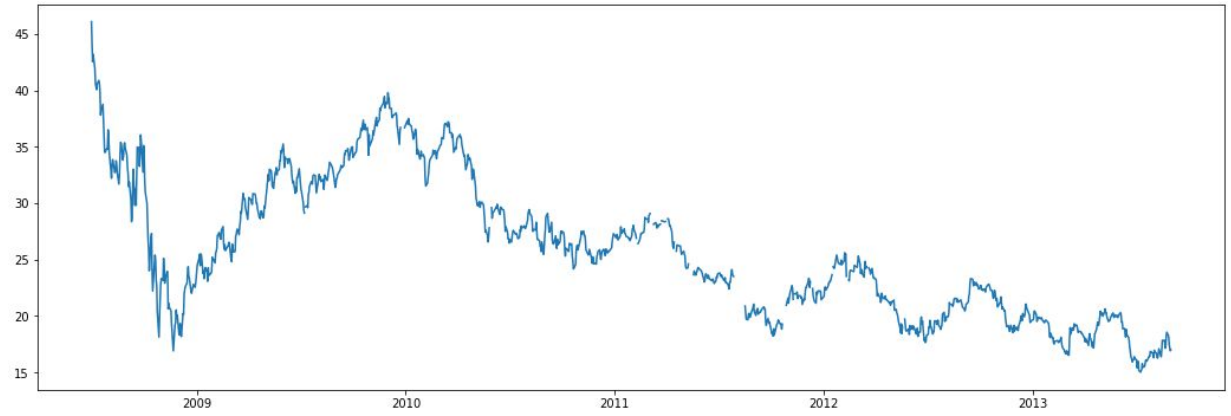
Como primeiro passo foi analisados os dados em falta na série. A série baixada no site *Yahoo! Finance* possui 5143 dias - de julho de 2000 a junho de 2020 - observados e em falta 117 dias. No corte histórico analisado ao todos são 1282 dias com 40 dias com valores em falta. Com grande concentração desses dias em falta no ano de 2011, 32 dias neste ano.

Figura 6 - Tabela Pandas com as 10 primeiras observações.

Date	Open	High	Low	Close	Adj Close	Volume
2008-07-01	45.500000	46.770000	45.380001	46.090000	37.044971	18256700.0
2008-07-02	46.340000	46.840000	43.799999	43.980000	35.349056	21150500.0
2008-07-03	44.259998	44.700001	42.279999	42.549999	34.199684	16835500.0
2008-07-04	42.450001	43.490002	42.000000	43.200001	34.722126	9253600.0
2008-07-07	42.380001	43.970001	41.810001	41.930000	33.701366	17909400.0
2008-07-08	41.599998	41.930000	40.000000	40.610001	32.640411	25528600.0
2008-07-10	39.889999	40.450001	39.020000	40.049999	32.190308	29497500.0
2008-07-11	40.150002	41.259998	39.990002	40.599998	32.632374	20777700.0
2008-07-14	41.099998	41.189999	40.700001	40.880001	32.857430	13011900.0
2008-07-15	40.349998	40.950001	39.560001	40.549999	32.592182	20875300.0

Compilação do autor

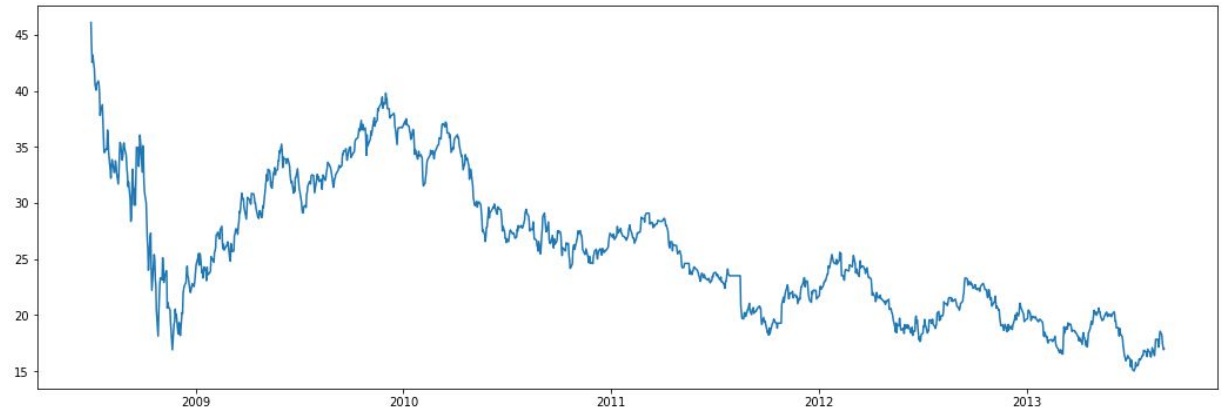
Figura 7 - Corte completo das ações da Petrobras com dados em falta



Compilação do autor

Os autores do artigo relacionado não citam como lidaram com os dados faltante. Este trabalho optou com preenchê-los com dados do dia anterior, como recomenda a documentação do Pandas.

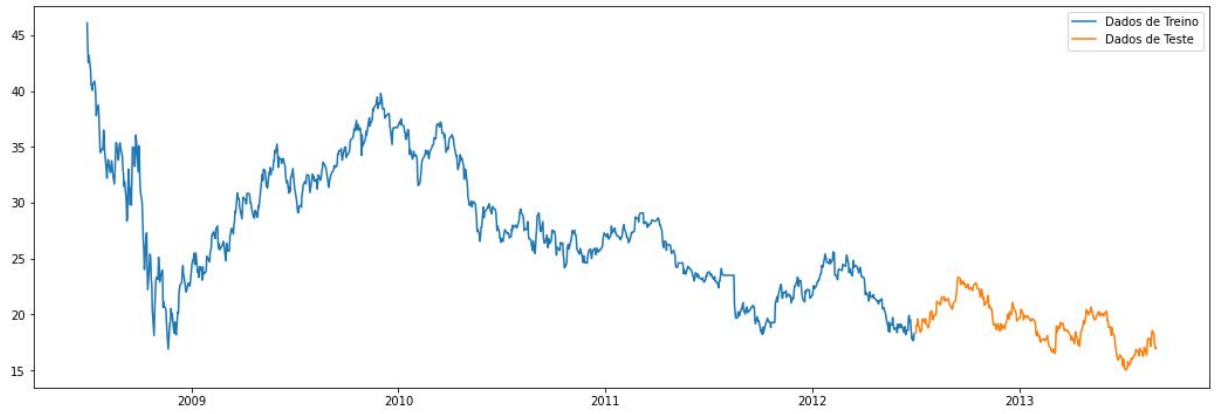
Figura 8 - Corte completo das ações da Petrobras com dados preenchidos



Compilação do autor

Após a imputação de dados ausentes a fase de definição de qual será o tamanho dos conjuntos de teste e foi feita. Seguindo a separação feita por Papadoulos et al. (2016), de 80% para treino e 20% para teste a série ficou particionada desta forma:

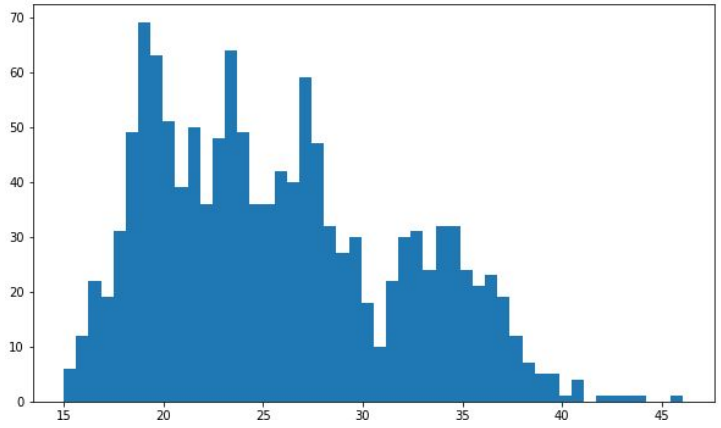
Figura 9 - Dados divididos em conjunto de treino e teste



Compilação do autor

Em seguida os teste para verificar se a série deve ser o ou não normalizada. A primeira verificação foi de modo visual, com o plotagem da distribuição dos dados da PETR4, como segue:

Figura 10 - Distribuição dos dados da série com evidência de não-normalidade



Compilação do autor

A forma da distribuição, com uma assimetria à esquerda é uma evidência de que a série não é normal, porém os testes são mais confiáveis: O teste K-quadrado de D'Agostino retornou um valor *p* igual a zero; idem o teste Shapiro-Wilk. Indicando, assim, uma forte possibilidade de que os dados não distribuem-se gaussianamente. E, portanto, serão normalizados.

Através de busca aleatória, foram treinados e testados 70 configurações de LSTMs. Desta busca os dez melhores serão utilizados. Os 5 melhores são mostrados na tabela abaixo:

Figura 11 - Tabela com os 5 melhores modelos

		modelo	Nº de épocas	Nº camadas	Nº neur 1	Nº de neur 2	Nº de neur 3	Taxa de apr.	Loss
MAE	RMSE								
0.597844	**0.742150**	45	10	2	103	112	0	0.001460	mean_squared_error
0.612021	**0.755028**	44	9	2	97	20	0	0.001655	mean_absolute_error
0.634587	**0.778035**	22	5	2	150	119	0	0.001280	mean_absolute_error
0.636293	**0.773435**	48	8	2	97	98	0	0.001464	mean_absolute_error
0.651653	**0.818682**	25	7	1	63	0	0	0.002369	mean_squared_error

Compilação do autor

6. Resultados

Após as fases de pré-processamento e treinamento, a fase de teste é iniciada, em que os dados de treino e validação são concatenados, o modelos com hiperparâmetros selecionados são retreinados com conjunto de treino completo, para assim ser testado com o conjunto de dados não vistos. Os resultados são os seguintes:

A variância do conjunto de teste é 3.7127192074653736, a variância dos dados preditos pelo modelo mais acurado é 3.435279, a diferença de variâncias é 0.2774403149482838. Os valores divulgados por Papadoulos et al. (2016) são: variância do real de 3,136943537, variância do predito igual a 3,115080788 e a diferença 0,021862749.

Papadoulos et al. (2016) também utilizou a média da diferença das variâncias dos 10 modelos com o resultado de 0,353725489. Este trabalho conseguiu uma média de 0.5747976595756642. A tabela que segue expõe as diferenças de variância dos 10 modelos selecionados e retreinados com o conjunto de treino completo:

Figura 12 - Tabela com os 10 modelos mais acurados

	modelo	Variância Real	Variância Predita	Diferença
0	1.0	3.712719	3.428286	0.284434
1	2.0	3.712719	2.844689	0.868031
2	3.0	3.712719	2.988983	0.723736
3	4.0	3.712719	3.151158	0.561561
4	5.0	3.712719	3.104853	0.607867
5	6.0	3.712719	3.191998	0.520721
6	7.0	3.712719	3.132777	0.579942
7	8.0	3.712719	3.051089	0.661630
8	9.0	3.712719	3.122978	0.589741
9	10.0	3.712719	3.362405	0.350314

Compilado pelo autor

7. Conclusão

O objetivo deste trabalho foi verificar se as redes neurais recorrentes do tipo LSTM poderia ser uma boa ferramenta para previsão de séries temporais financeiras. A série escolhida foi a dos preços das ações preferenciais da Petrobras PETR4, para avaliar se a rede é ou não adequada ao proposto o artigo de Papadoulos et al. (2016), que também utiliza a mesma série, esse estudou a aplicação do MLP.

Dados os resultados, para essa série no corte histórico de 2008 a 2013 (e uma dificuldade em cruzar os dados obtidos com os utilizados pelos autores do artigo relacionado), com 70 modelos selecionados randomicamente, foi constatado que o MLP possui resultados superiores aos da LSTM. A constatação se deve a métrica de diferença de variâncias, em que a diferença da variância

Como recomendação para estudos futuros utilizar Grid Search para a busca de modelos melhores, um número maior de buscas iterativas (no pressuposto de disponibilidade de maior poder computacional), testar com outras arquiteturas e outras séries temporais.

Referências Bibliográficas

Papadoulos, T.B., et al. (2016) **Redes Neurais Artificiais na Predição do Preço Futuro de Papéis com Alta e Baixa Volatilidade na Bovespa e na Bolsa de Valores de Nova Iorque**. XVII Congresso Nacional de Administração e Contabilidade. Rio de Janeiro.

Investidor Petrobras (2020). **Negociação das Ações** (Brasil). Disponível em: https://www.investidorpetrobras.com.br/acoes-dividendos-e-divida/acoes/. Acesso Em 01/10/2020

Brockwell, P. J.; Davis, R.A. (2016). **Introduction to Time Series and Forecasting,** 3º edição., Springer. Suíça.

Trask, A.W. (2019). **Grokking Deep Learning**, 1ª edição, Manning. Estados Unidos da América.

Mitchell, T.M. (1997) **Machine Learning**, 1ª edição, McGraw-Hill. Estados Unidos da América.

Goodfellow, I.; Bengio, Y.; Courville, A. (2016). Deep Learning, MIT-Press. Disponível em: http://www.deeplearningbook.org/. Acesso em 15/10/2020.

Olah, C. (2015) **Understanding LSTM Networks,** Disponível em: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Acesso em: 12/10/2020

-python/. Acesso em: 12/10/2020.

Brownlee, J. (2017) **How to Scale Data for Long Short-Term Memory Networks in Python**. Disponível em: https://machinelearningmastery.com/how-to-scale-data-for-long-short-term-memory-networks-in-python/. Acesso em: 13/10/2020.

_____. (2016) **How To Backtest Machine Learning Models for Time Series Forecasting** Disponível em: https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/. Acesso em 18/10/2020.

Pandas documentation (2020). Disponível em: https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html. Acesso em: 13/10/2020.