Predição de preço de ativos financeiros com RNN - LSTM

Micael C. Fernandes Lima[1]

[1]Instituto de Tecnologia (ITEC) – Universidade Federal do Pará (UFPA)
Rua Augusto Corrêa, 01 - Guamá. CEP 66075-110. Belém - Pará - Brasil

micael.lima@itec.ufpa.br

Abstract. *Machine learning is an extremely useful technology for data mining and problem solving. One of the ways to develop this learning is by using neural networks. This article shows an application that uses Recurrent Neural Networks to predict the price movement of a financial asset. This way, through a python language code and the use of several libraries and programming tools, it is expected to obtain graphs that indicate the future movement of the asset's price and demonstrate the learning of the neural network, that is, that the application reaches its established objective.*

Resumo. *Aprendizado de máquina é uma tecnologia extremamente útil para mineração de dados e a resolução de problemas. Uma das formas de desenvolver esse aprendizado é utilizando redes neurais. Este artigo mostra uma aplicação que utiliza Redes Neurais Recorrentes para prever a movimentação do preço de um ativo financeiro. Desse modo, por meio de um código na linguagem python e o uso de diversas bibliotecas e ferramentas de programação, espera-se obter gráficos que indiquem a movimentação futura do preço do ativo e que demonstre o aprendizado da rede neural, ou seja, que a aplicação alcance seu objetivo estabelecido.*

1. Introdução

Inicialmente é essencial conhecer como as redes neurais funcionam e sua estrutura básica. Redes neurais são sistemas de computação com nós interconectados que funcionam como os neurônios do cérebro humano. Usando algoritmos, elas podem reconhecer padrões escondidos e correlações em dados brutos, agrupá-los e classificá-los, e – com o tempo – aprender e melhorar continuamente. O objetivo original da abordagem de rede neural era criar um sistema computacional capaz de resolver problemas como um cérebro humano.

Existem diversas categorias de redes neurais, onde cada tipo possui vantagens e desvantagens e podem ser utilizadas para atacar problemas específicos e ser usados das mais diversas formas. Podemos destacar os principais tipos de redes neurais:

- Redes neurais convolucionais (RNCs ou CNNs) contêm cinco tipos de camadas: de entradas, de convolução, de agrupamento, as completamente conectadas e as de saída. Cada camada tem um propósito específico, como de resumo, conexão ou ativação. As redes neurais convolucionais popularizaram a classificação de imagens e a detecção de objetos. Entretanto, RNCs também foram aplicadas em outras áreas como previsão e processamento de linguagem natural.
- Redes neurais recorrente (RNRs ou RNN) usam informações sequenciais, como dados de registro de data e hora de um sensor ou uma frase dita. Essas informações são

compostas por uma sequência de termos. Diferentemente das redes neurais tradicionais, as entradas de uma rede neural recorrente não são independentes umas das outras, e os resultados para cada elemento dependem da computação dos elementos precedentes. RNRs são utilizadas na previsão e aplicação de séries temporais, análise de sentimento e outras aplicações de texto.

- Redes neurais feedforward consiste no cascateamento em camadas de um neurônio chamada perceptron, cada perceptron em uma camada é conectado por pesos (arestas) sinápticos a todo perceptron da camada seguinte. A informação é entregue de maneira antecipada de uma camada à seguinte seguindo sempre em frente. As redes feed foward são mais utilizadas para problemas de classificação, não sendo muito efetiva para problemas de séries temporais
- Redes neurais autoencoder são utilizadas para criar abstrações chamadas encoders, criados a partir de um conjunto estipulado de entradas. Apesar de similares às redes neurais mais tradicionais, autoencoders procuram modelar as entradas por si só e, portanto, o método é considerado não supervisionado. A premissa dos autoencoders é diminuir a sensibilidade ao que é irrelevante e aumentar ao que é. Conforme camadas são adicionadas, outras abstrações são formuladas em camadas mais altas (camadas mais próximas ao ponto onde uma camada decodificadora é introduzida). Essas abstrações podem, então, ser usadas por classificadores lineares ou não lineares.

2. Características da rede LSTM

2.1. Redes Neurais Recorrente (RNN)

Nas redes feedforward, exemplos de entrada são alimentados na rede e transformados em uma saída, com a aprendizagem supervisionada, a saída seria por exemplo um rótulo, um nome aplicado à entrada. Ou seja, elas mapeiam dados brutos para categorias, reconhecendo padrões que podem sinalizar, por exemplo, que uma imagem de entrada deve ser rotulada como “gato” ou “cachorro”.

Já as redes recorrentes tomam como entrada não apenas o exemplo de entrada atual que veem, mas também o que perceberam anteriormente no tempo. Na Figura 1 podemos notar a diferença de arquitetura nos entre os dois tipos de redes.

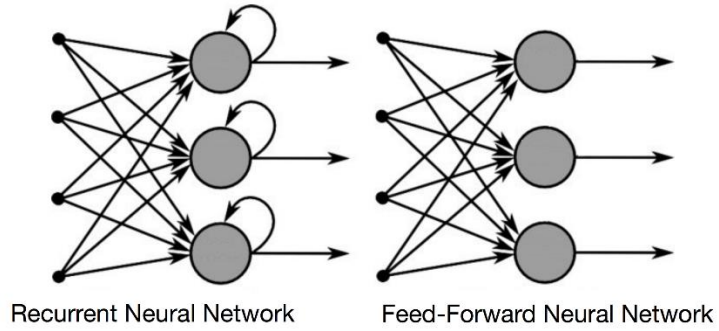

Figura 1. Diagrama das redes RNN e feedfoward

As redes recorrentes são diferenciadas das redes feedforward pelo loop de feedback conectado às suas decisões anteriores, ingerindo suas próprias saídas momento após momento como entrada. Costuma-se dizer que as redes recorrentes têm memória. A

adição de memória às redes neurais tem uma finalidade: há informações na própria sequência e as redes recorrentes a utilizam para executar tarefas que as redes de feedforward não conseguem. Podemos descrever matematicamente o ato de levar a memória adiante na RNN da seguinte forma:

$$h_t = \phi(Wx_t + Uh_{t-1})$$

O estado oculto na etapa de tempo t é h_t. É uma função da entrada na mesma etapa de tempo x_t, modificada por uma matriz de peso W (como a que usamos para redes feedforward) adicionada ao estado oculto do passo de tempo anterior h_{t-1} multiplicado por seu próprio estado oculto – a matriz de estado oculto U, também conhecida como matriz de transição e semelhante a uma cadeia de Markov. As matrizes de peso são filtros que determinam quanta importância deve ser dada tanto à entrada atual quanto ao estado oculto do passado. O erro que eles geram retornará por meio de retropropagação (backpropagation) e será usado para ajustar seus pesos até que o erro não diminua mais.

A soma da entrada de peso e do estado oculto é comprimida pela função φ – função de ativação – que é uma ferramenta padrão para condensar valores muito grandes ou muito pequenos em um espaço logístico, bem como tornar os gradientes viáveis para retropropagação.

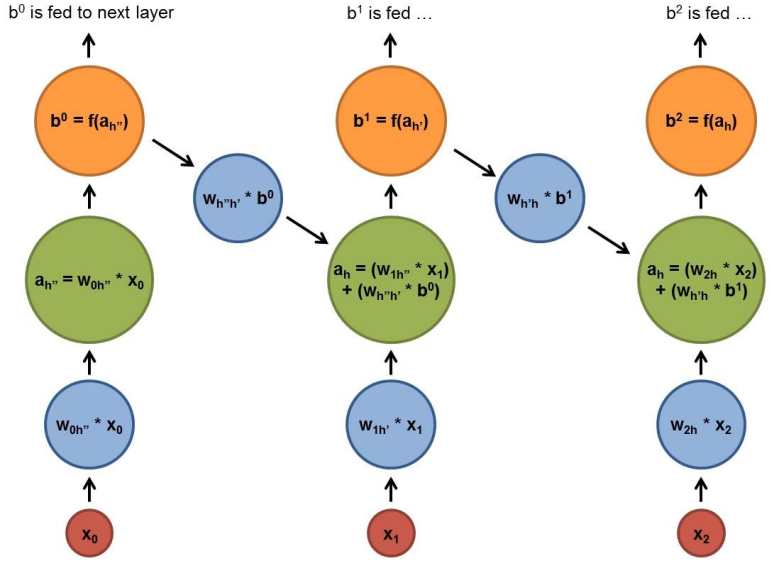

Figura 2. Exemplo de um funcionamento de um RNN

No diagrama acima, cada **x** é um exemplo de entrada, **W** são os pesos que filtram as entradas, $\mathbf{a_h}$ é a ativação da camada oculta (um produto entre a entrada ponderada e o estado oculto anterior), e cada **b** é a saída da camada oculta após ter sido ativada, ou seja, o resultado da função de ativação.

2.2. Redes LSTM

A memória de longo prazo (LSTM) é uma arquitetura especial de rede neural recorrente (RNN). Ao contrário das redes neurais feedforward padrão, a LSTM tem conexões de feedback. Ele pode processar não apenas pontos de dados únicos (como imagens), mas também sequências inteiras de dados (como fala ou vídeo). Por exemplo, o LSTM é aplicável a tarefas como reconhecimento de escrita à mão não segmentada e conectada,

reconhecimento de fala e detecção de anomalias no tráfego da rede ou IDSs (sistemas de detecção de intrusão).

Os LSTMs são projetados para evitar o problema de dependência de longo prazo. Seu comportamento padrão consiste em lembrar informações por longos períodos de tempo visando selecionar as melhores informações para construir seu "aprendizado". Todas as redes neurais recorrentes têm a forma de uma cadeia de módulos repetitivos da rede neural. Em RNNs padrão, este módulo de repetição terá uma estrutura muito simples, como uma única camada de camada.

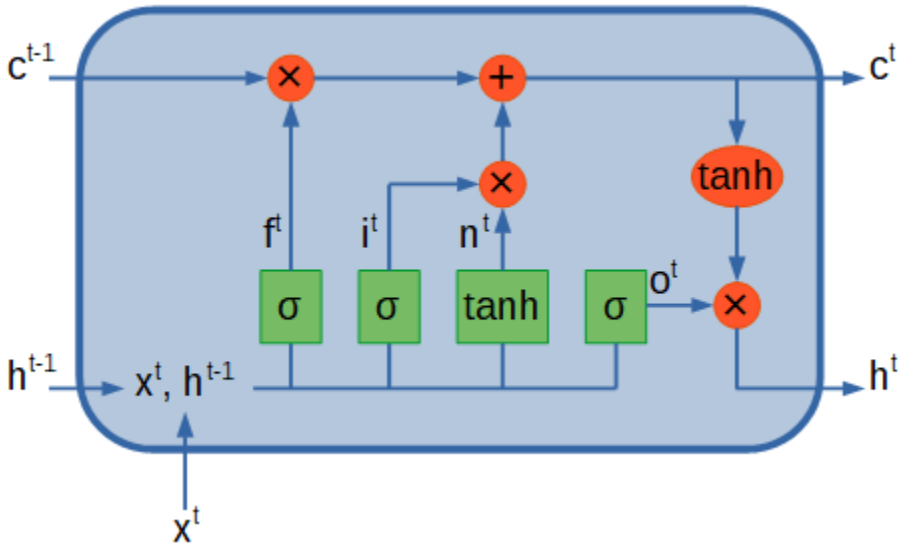

Figura 3. Exemplo de uma estrutura LSTM padrão

Acima, podemos ver a propagação direta dentro de uma célula LSTM. É consideravelmente mais complicado do que o simples RNN. Ele contém quatro redes ativadas pela função sigmóide (σ) ou pela função tanh, todas com seu próprio conjunto diferente de parâmetros.

Cada uma dessas redes, também conhecidas como portas, tem uma finalidade diferente. Eles transformarão o estado da célula para a etapa de tempo t (c^t) com as informações relevantes que devem ser passadas para a próxima etapa de tempo. Os círculos / elipse laranja são transformações em elementos das matrizes que os precedem. Como dito, cada porta tem uma função, dentre as quais:

- Forget gate layer (f)**:** Decide quais informações esquecer do estado da célula usando uma função σ que modula as informações entre 0 e 1. Ela esquece tudo que é 0, lembra tudo que é 1 e tudo no meio são possíveis candidatos.
- Camada de porta de entrada (i): Esta também pode ser uma porta de lembrança. Ele decide quais dos novos candidatos são relevantes para este intervalo de tempo também com a ajuda de uma função σ.
- Nova camada de porta candidata (n): Cria um novo conjunto de candidatos a serem armazenados no estado da célula. A relevância desses novos candidatos será modulada pela multiplicação elemento a elemento com a camada de porta de entrada.
- Camada da porta de saída (o): determina quais partes do estado da célula são geradas. O estado da célula é normalizado por meio de uma função tanh e é multiplicado

elemento a elemento pela porta de saída que decide qual novo candidato relevante deve ser gerado pelo estado oculto.

Há também equações matemáticas que modelam essas portas e fazem a célula funcionar, essas equações podem ser vistas na Figura 4. A ultima equação remete a uma propagação direta na rede para obter sua saída ou previsão.

$$f^t = \sigma\left(\left(X^t\right)^T \cdot W_f + B_f\right)$$
$$i^t = \sigma\left(\left(X^t\right)^T \cdot W_i + B_i\right)$$
$$n^t = \tanh\left(\left(X^t\right)^T \cdot W_n + B_n\right)$$
$$o^t = \sigma\left(\left(X^t\right)^T \cdot W_o + B_o\right)$$
$$c^t = \left(f^t \times c^{t-1}\right) + \left(i^t \times n^t\right)$$
$$h^t = o^t \times \tanh\left(c^t\right)$$
$$\hat{y} = h^t \cdot W_h + B_h$$

Figura 4. Equações modeladoras da estrutura LSTM

Toda a arquitetura da rede LSTM é construída para lidar com uma sequência de entrada de três etapas de tempo e para prever uma etapa de tempo no futuro, conforme mostrado na Figura 5. Colocar as entradas e os parâmetros na forma de vetor e matriz pode ajudar a entender a dimensionalidade dos cálculos.

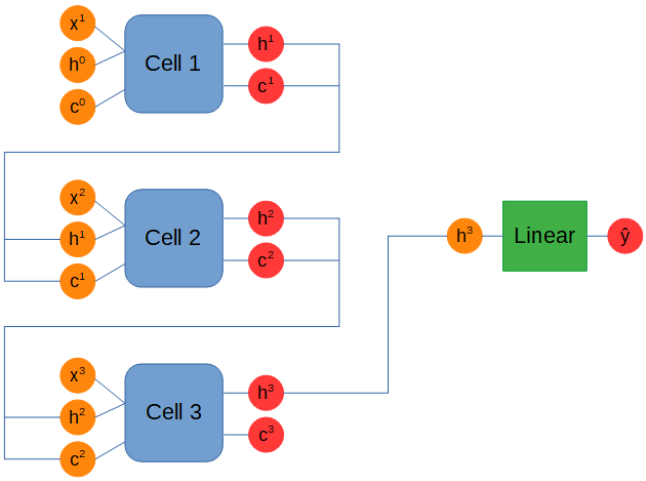

Figura 5. Representação de um LSTM com três entradas e uma saída

3. Visualização dos dados

A aplicação visa prever a movimentação futura no preço de um ativo financeiro, que nesse caso é um criptoativo, o Ethereum. Os dados envolvidos referem-se à movimentação diária de preço do Ethereum pelo dólar americano (ETH/USD), que consistem no preço de abertura (Open), máxima do dia (High), mínima do dia (Low) e o preço de fechamento do dia (Close). Os dados [Ethereum Data, Kaggle] foram obtidos no site Kaggle, um repositório gratuito de bancos de dados. Utilizando a biblioteca apropriada do python, podemos visualizar os dados brutos na Figura 6.

	Date	Open	High	Low	Close
0	2015-08-07	2.831620	3.536610	2.521120	2.772120
1	2015-08-08	2.793760	2.798810	0.714725	0.753325
2	2015-08-09	0.706136	0.879810	0.629191	0.701897
3	2015-08-10	0.713989	0.729854	0.636546	0.708448
4	2015-08-11	0.708087	1.131410	0.663235	1.067860
...	...	...	...	...	...
2176	2021-07-22	1994.822876	2044.116455	1954.297852	2025.202759
2177	2021-07-23	2025.105713	2129.441162	2000.436279	2124.776611
2178	2021-07-24	2123.961182	2197.649414	2107.323486	2189.218750
2179	2021-07-25	2187.145508	2194.438232	2108.829834	2191.373779
2180	2021-07-26	2177.329590	2384.400635	2177.329590	2340.090820

2181 rows × 5 columns

Figura 6. Dados brutos da movimentação diária do ETH/USD

É necessário tratar esses dados para que a rede LSTM trabalhe da forma mais otimizada possivel. Na Figura 7 podemos ver um gráfico com a variação diária do preço de fechamento do ETH/USD.

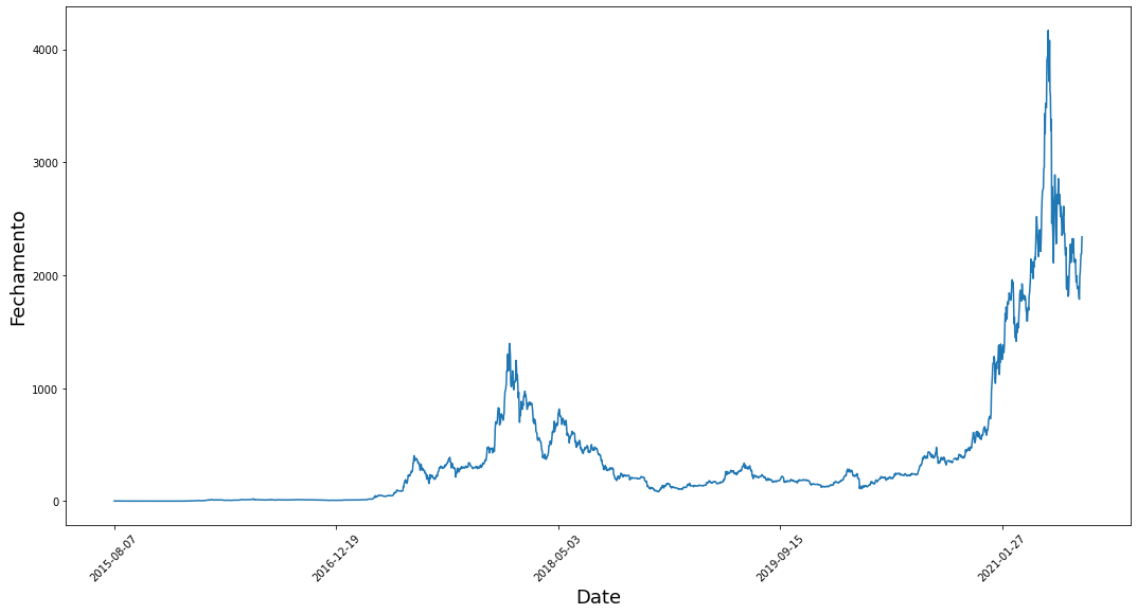

Figura 7. Gráfico da variação diária do ETH/USD

O principal parâmetro de aprendizagem para a rede LSTM é preço de fechamento do dia. Com isso, de acordo com a execução do código, a rede tentará prever o proximo preço de fechamento do ativo.

4. O código principal da aplicação

4.1. Importações e bibliotecas

O código foi construído na linguagem python, onde se encontra também o uso de bibliotecas eficientes para o aprendizado de máquina, preparação e visualização de dados. As bibliotecas Keras, numpy, Tensorflow e sklearn contém as funções necessárias para modelar, dividir e treinar os dados. A biblioteca pandas contém ferramentas úteis para a obtenção e visualização de dados, já a biblioteca matplotlib foi utilizada para obter os gráficos (plots) desse documento.

4.2. Preparação dos dados

Inicialmente, visto que a rede LSTM é sensível a escala dos dados, é importante normalizar os dados e escalona-los para uma faixa de valores menor, desse modo poderemos alcançar um aprendizado mais otimizado. Na Figura 8 podemos ver na saída do código a diferença dos dados originais para os normalizados, foi utilizado uma função da biblioteca sklearn para normalizar os dados.

```
df1   # ETH/USD close price
```

```
0          2.772120
1          0.753325
2          0.701897
3          0.708448
4          1.067860
           ...
2176    2025.202759
2177    2124.776611
2178    2189.218750
2179    2191.373779
2180    2340.090820
Name: Close, Length: 2181, dtype: float64
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

```
print(df1)  # ETH/USD close price normalized
```

```
[[5.60734561e-04]
 [7.64097046e-05]
 [6.40717214e-05]
 ...
 [5.25106541e-01]
 [5.25623549e-01]
 [5.61301941e-01]]
```

Figura 8. Normalização dos dados (preço de fechamento)

É importante definir partes dos dados para treino e para teste e as janelas de tempo para o aprendizado. As janelas de tempo definem o que a máquina usará para aprender (features) e também defina a resultado esperado para assim haver a retro propagação do erro na célula LSTM e a correção de seus pesos sinápticos. Na Figura 9 está a parte do código responsável por separar os dados de treino e de teste, neste caso, 60% dos dados será usado para treino e o restante para teste.

```
# splitting dataset into train and test data
training_size=int(len(df1)*0.60)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

```
training_size,test_size   # shows the size of each group of data
```

```
(1308, 873)
```

Figura 9. Separação dos dados de treino e de teste (60% e 40%)

Utilizando a biblioteca numpy, foi criada uma função "create_dataset" (Figura 10) que estabelece as janelas de tempo e retorna uma tupla que contém o conjunto de dados que será usado como features (x) e o conjunto de dados que será usado como o resultado real (y). Nessa aplicação, a janela de tempo foi definida como 100 dias, desse modo, a rede aprendera com 100 dias de dados e assim tentará prever o preço no dia 101.

```
import numpy
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
  dataX, dataY = [], []
  for i in range(len(dataset)-time_step-1):
    a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
    dataX.append(a)
    dataY.append(dataset[i + time_step, 0])
  return numpy.array(dataX), numpy.array(dataY)
```

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

Figura 10. Função create_dataset e sua execução

Podemos ver também na Figura 10 o estabelecimento dos dados X_train, X_test, Y_train e Y_test onde eles representam as features e resultados reais de treino e teste, respectivamente. Com os dados gerados, modelados e organizados resta apenas instanciar a estrutura da rede LSTM e executa-la (treinamento).

4.3. Criação e treinamento da rede neural

Como visto anteriormente, a aplicação trabalha com a uma rede recorrente especial LSTM que é excelente para problemas de séries temporais, que é o caso da previsão de preço de um ativo financeiro.

Utilizando a biblioteca tensorflow e sklearn, é possivel instanciar os componentes da nossa rede. A Figura 11 mostra essa parte do código.

```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')

```

```
model.summary() # show the LSTM network structure
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
lstm_10 (LSTM)               (None, 100, 50)           20200
_________________________________________________________________
lstm_11 (LSTM)               (None, 50)                20200
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

Figura 11. Criação da rede LSTM

Primeiramente é necessário criar o modelo da rede, nesse caso, será em modelo de camadas sequencialmente interligadas (em forma de pilha). A estrutura dessa rede consiste em 3 camadas LSTM, a rede possui 100 features (entrada) e apenas 1 saída. Vale lembrar que cada camada possui seu bias, pesos sinápticos e uma estrutura de retro propagação que é criada ao se instanciar e configurar o modelo.

```
metadata = model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=200,batch_size=64,verbose=1)
```

```
Epoch 1/200
19/19 [==============================] - 10s 254ms/step - loss: 0.0014 - val_loss: nan
Epoch 2/200
19/19 [==============================] - 4s 191ms/step - loss: 3.2161e-04 - val_loss: nan
Epoch 3/200
19/19 [==============================] - 4s 192ms/step - loss: 2.4507e-04 - val_loss: nan
Epoch 4/200
19/19 [==============================] - 4s 192ms/step - loss: 2.1705e-04 - val_loss: nan
Epoch 5/200
19/19 [==============================] - 4s 189ms/step - loss: 2.0038e-04 - val_loss: nan
Epoch 6/200
19/19 [==============================] - 4s 191ms/step - loss: 1.9002e-04 - val_loss: nan
Epoch 7/200
19/19 [==============================] - 4s 188ms/step - loss: 1.8493e-04 - val_loss: nan
Epoch 8/200
19/19 [==============================] - 4s 194ms/step - loss: 1.7467e-04 - val_loss: nan
Epoch 9/200
19/19 [==============================] - 4s 197ms/step - loss: 1.7273e-04 - val_loss: nan
Epoch 10/200
```

Figura 12. Treinamento da rede neural

Na Figura 12 vemos a execução da função “fit” responsável por treinar a rede e torna-la capaz de prever algum resultado. Essa função recebe como parâmetros, dentre outras coisas, os dados anteriormente organizados e o número de iterações para treinar a rede. É escolhido 200 iterações, pois, deduzido maneira empírica, foi o suficiente para a rede ter um erro pequeno de predição e um bom aprendizado. Com o treino realizado, é interessante a visualização dos resultados.

5. Resultados

Por meio da execução do código é possivel ver a evolução da rede neural pelo numero de iterações. A Figura 13 mostra o gráfico de erro versus o número de iterações de treino da rede. Isso mostra a evolução do aprendizado ao longo do tempo, uma vez que o erro diminui com o passar das iterações, o que mostra que a rede neural está aprendendo.

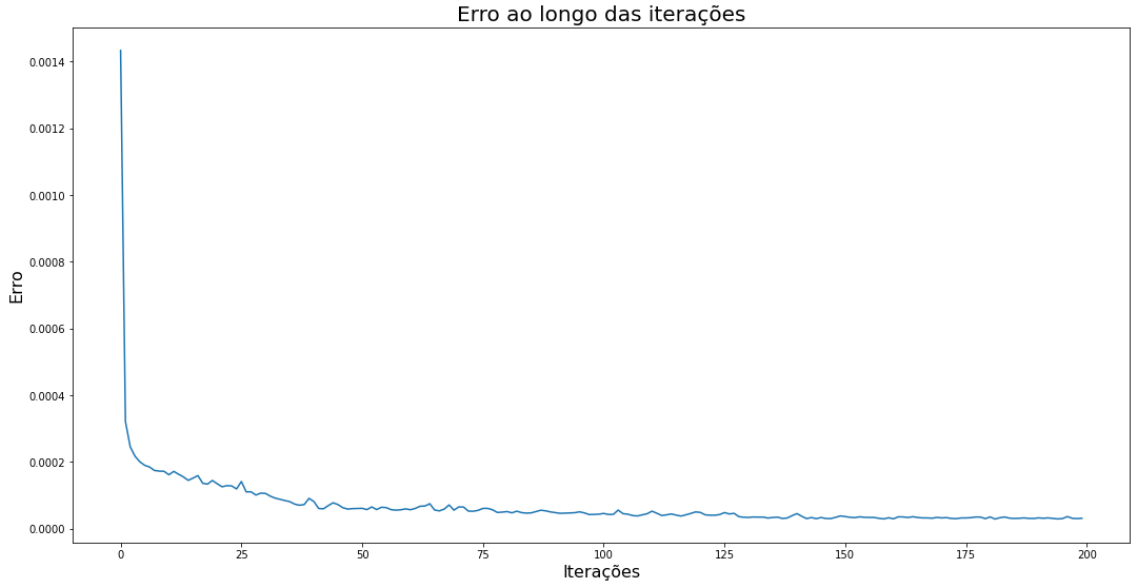

Figura 13. Gráfico do erro da rede LSTM

Figura 14. Resultado de previsão da rede neural sobre o preço do ETH/USD

A Figura 14 mostra o resultado alcançado pela rede após 200 iterações. A linha verde mostra a previsão da rede neural, a linha laranja mostra a parte de treino utilizada pela rede. Apesar de não conseguir ser exata, a previsão da rede se aproxima consideravelmente da realidade da movimentação do preço.

6. Conclusão

Como visto nos resultados, a rede neural LSTM conseguiu concretizar seu objetivo. A movimentação do preço do ativo ETH/USD foi devidamente modelada e padronizada pela rede neural. O gráfico de erro versus o número de iterações nos remete a capacidade da máquina aprender e a que taxa ela faz isso, podemos afirmar que a taxa de aprendizado da rede foi alta uma vez que o gráfico citado desce bruscamente em direção a zero.

Com tudo isso em mente, a implementação da aplicação de predição de preços de ativos financeiro permitiu desenvolver um maior entendimento acerca de problemas com séries temporais e de redes neurais LSTM, bem como contribuiu para o aprendizado de modelagem de dados.

Referências

Understanding LSTM Networks, Christopher Olah, 2017,<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Stock Price Prediction And Forecasting Using Stacked LSTM- Deep Learning – YouTube, Krish Naik, 2020, <https://www.youtube.com/watch?v=H6du_pfuznE>

Como o LSTM melhora o RNN (ichi.pro), acessado em outubro de 2021, <https://ichi.pro/pt/como-o-lstm-melhora-o-rnn-34021890806049>

Sepp Hochreiter e Jürgen Schmidhuber, “Long Short-Term Memory” in Neural Computation, 1997, DOI: 10.1162 / neco.1997.9.8.1735.