Stock price prediction using Long short-term memory (LSTM) Recurrent Neural Network

October 20, 2022

Shyamal Raj
19CE10064
Indian Institute of Technology, Kharagpur

Overview

• Engineered a multi-layer **long short-term memory** (LSTM) RNN to the input 'TSLA' stock close price for a period of 5 years.

• Pre-processed the data using MinMaxScaler from sklearn by scaling each feature to a value between -1 and 1.

• Achieved a **correlation coefficient** between real values and predicted values for train data to be **1.00** and for test data to be **0.97**

TECH STACK: PYTHON, PYTORCH, SKLEARN.PREPROCESSING, NUMPY, PANDAS, MATPLOTLIB, YFINANCE

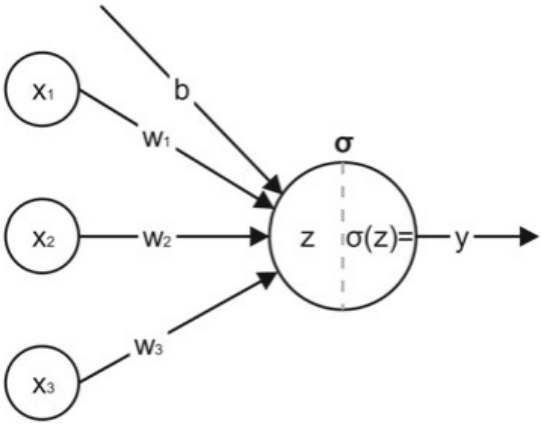
So, What exactly is a Recurrent Neural Network?

For that, let's back up a little...

Introduction

Simple fundamental basic building block components that make up every neural network are "**neurons**". A neuron is a component that accepts inputs.

Neurons stack up to form layers and these layers stack up further to form **neural networks**. Depending upon the number of layers, neural network can be deep or shallow.

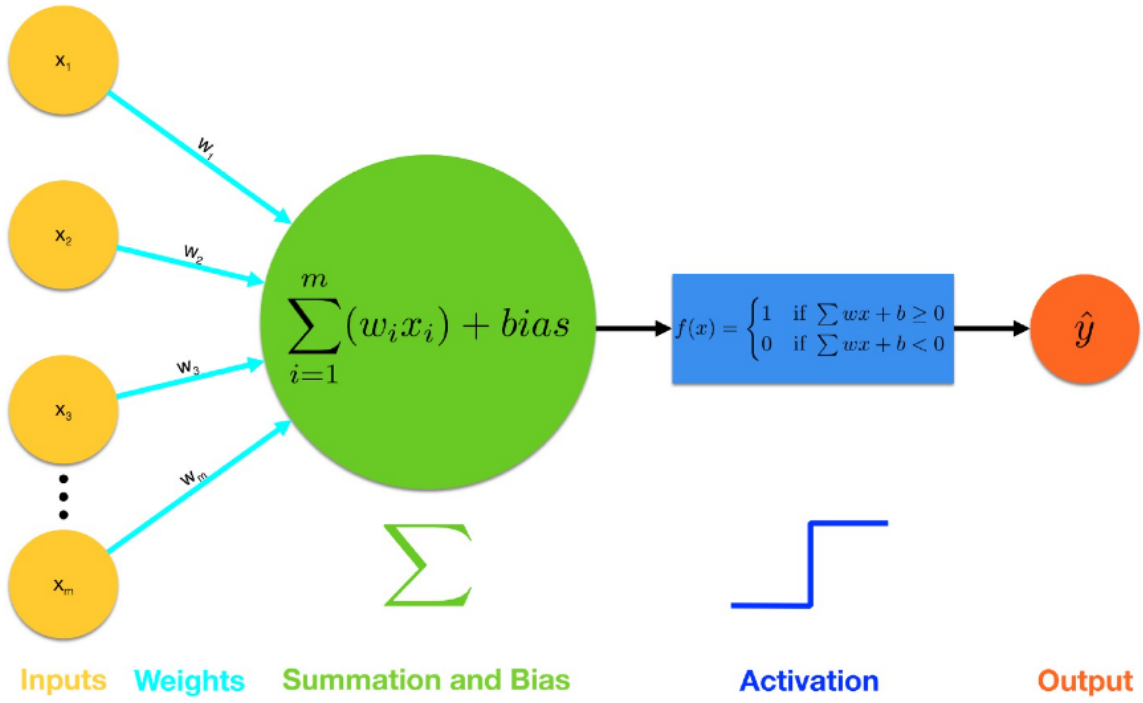


A simple representation of a Neuron

Source- Intro. to Deep Learning : From Logical Calculus to Artificial Intelligence,S Skansi

A neuron is a weighted combination of inputs plus a bias passed through "*some*" **non-linear activation function** (sigmoid function, ReLU, Leaky ReLU, tanh, etc.)

The non-linear part is what distinguishes it from traditional statistical modeling. The non-linear activation functions are great at learning complex relationships between inputs and predicting the output.

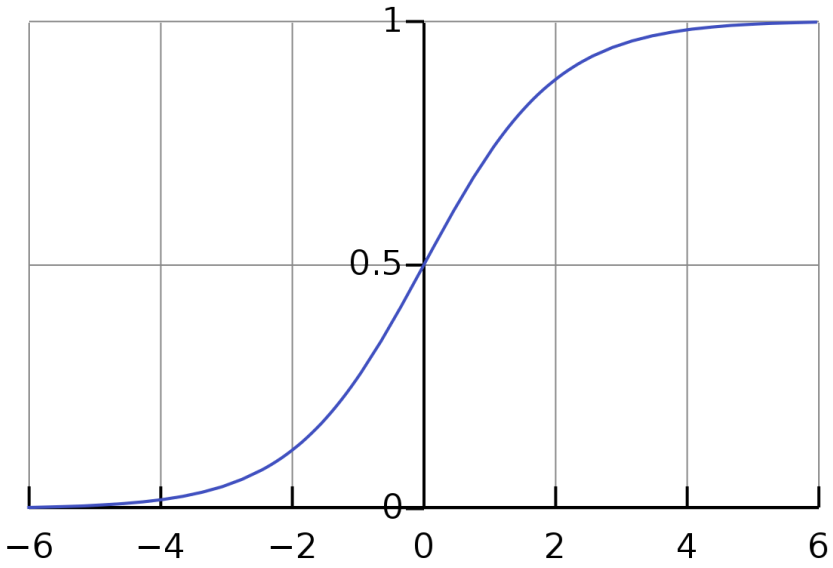


Source: Kang, N.

The Sigmoid "Gate"

Sigmoid Activation function

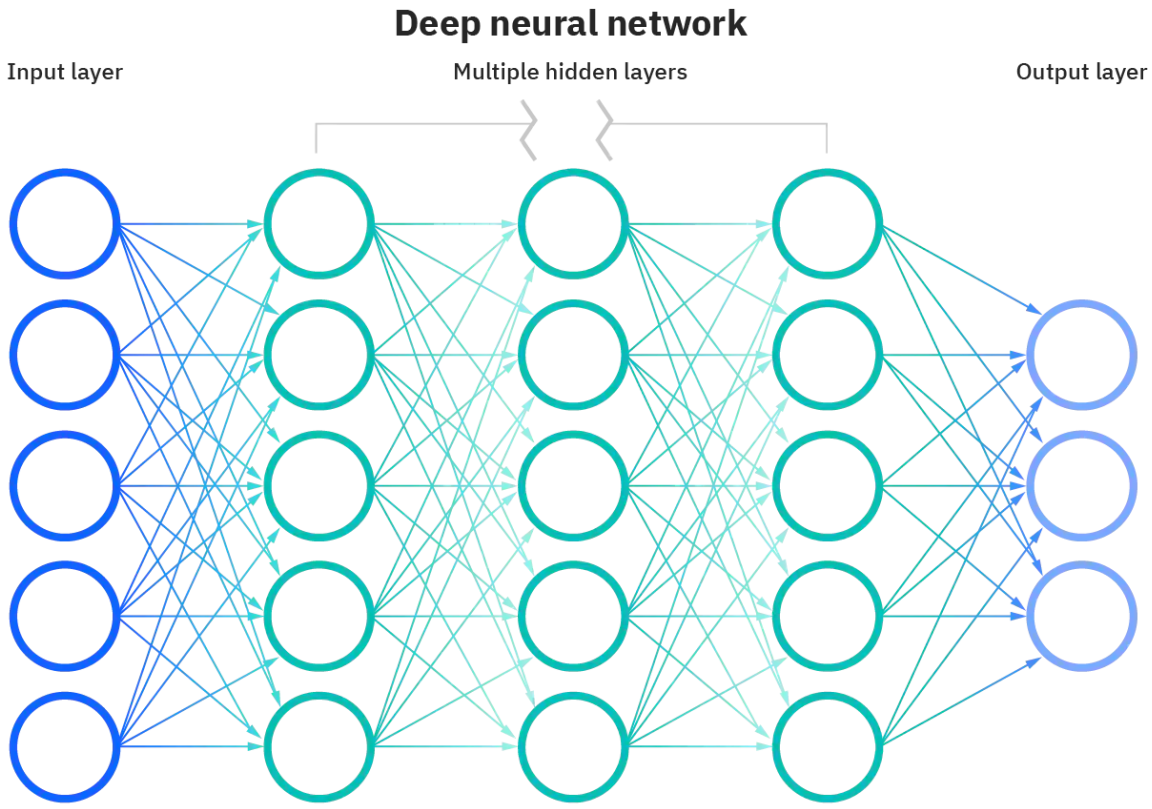
$$\sigma = \frac{1}{1 + e^{-x}}$$



Most values are close to 0 or 1

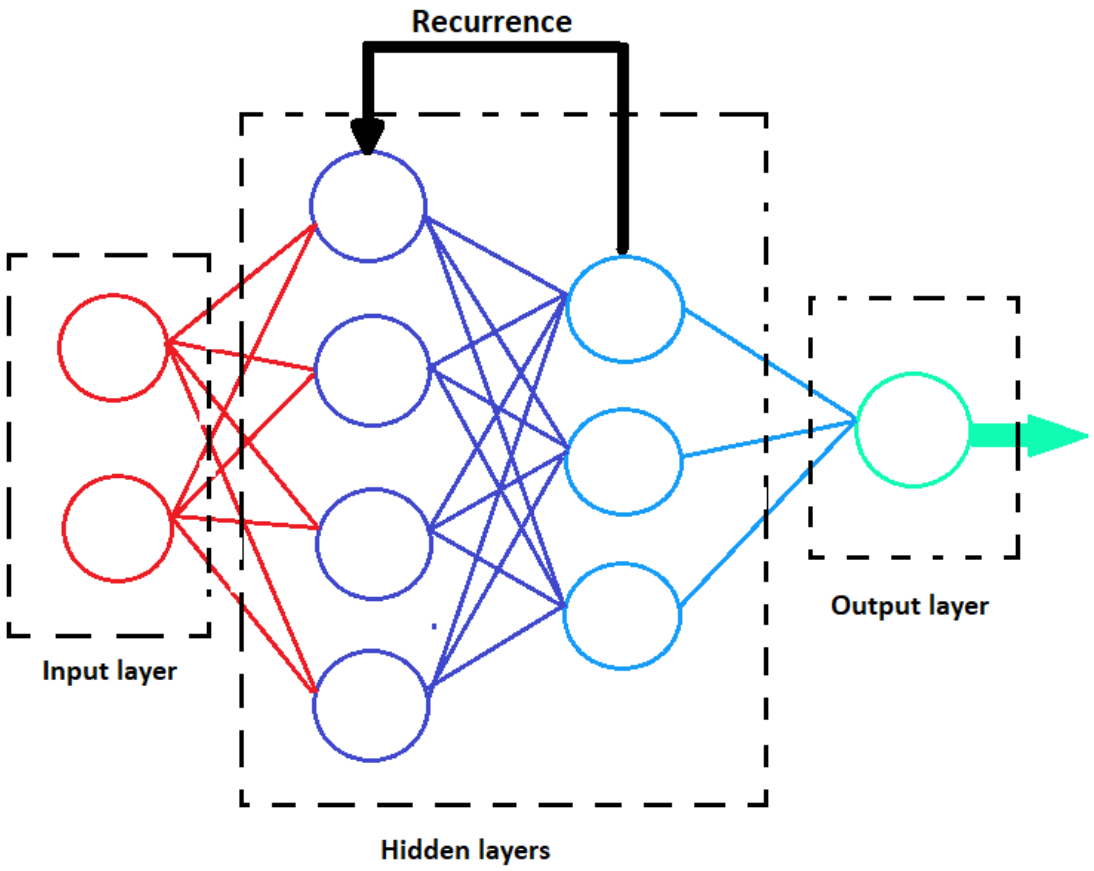
Therefore, Multiplying something with sigmoid can "switch off" (0) or "switch on" (1) the something; Acting as a gate

The Deep Neural Network



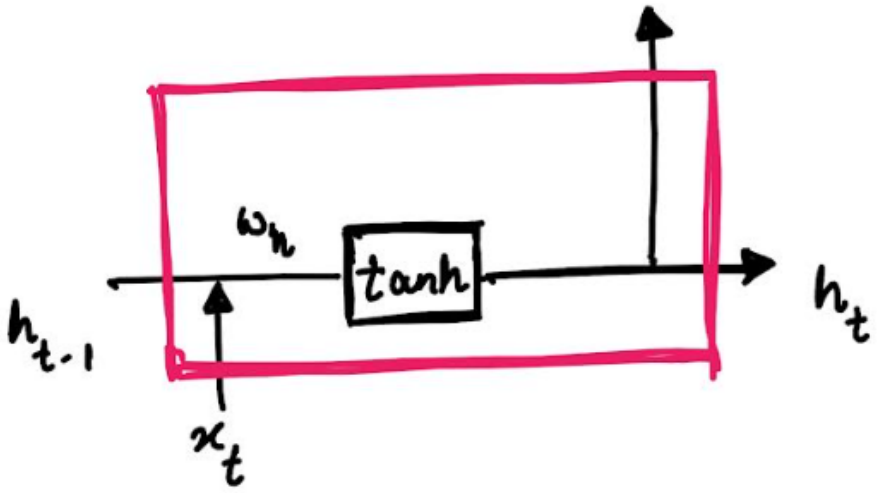
The Recurrent Neural Network

The output from some nodes can influence subsequent input to the same nodes in a recurrent neural network, which belongs to a family of artificial neural networks where connections between nodes can establish a cycle.



Recurrent Neural Network

A simplified RNN Network



RNN Architecture

$$h_t = tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

or,

$$h_t = tanh\left(w_h [x_t ; h_{t-1}]\right)$$

Limitations of RNN

Highly sensitive to recent inputs (therefore, works best when important characteristics are a few time-steps back)

May have vanishing and Exploding gradient risk

Long Short-term memory (LSTM)

Recurrent neural networks that can learn order dependency in sequence prediction tasks are known as Long Short-Term Memory (LSTM) networks.

Includes a **memory trace** specifically; Works better for **longer sequences** or complex patterns

However, These networks are complex and require more **training data** and **computation time.**

For each element in the input sequence, each layer computes the following function:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi})$$

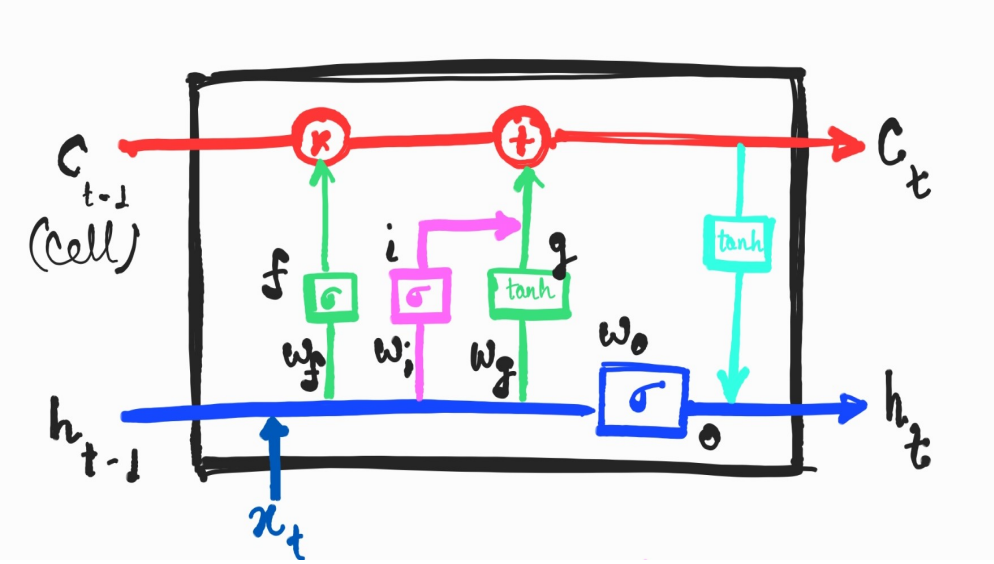
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf})$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg})$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho})$$

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

$$h_t = o_t * \tanh(c_t)$$



LSTM Architecture (explained in the algorithm section)

LSTM network is applied to predict the stock price of 'TSLA' (Tesla, Inc.) [close price] for a period of 5 years

About Tesla, Inc.

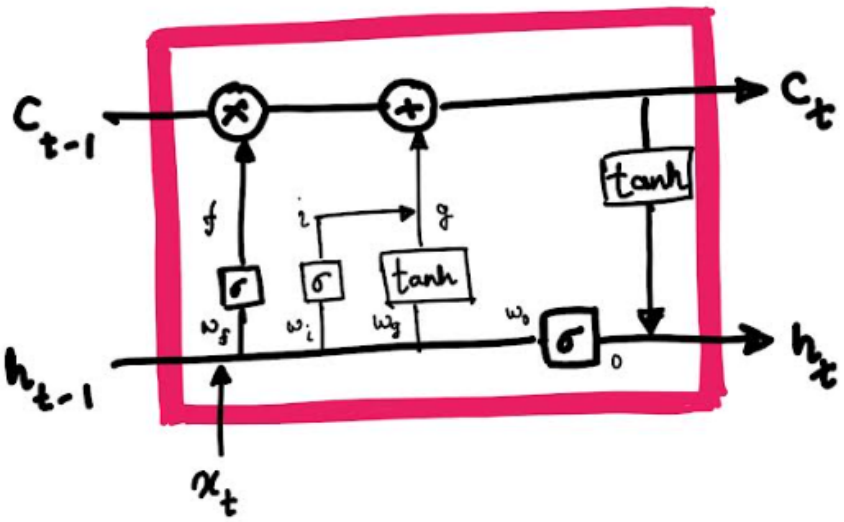
Tesla, Inc. designs, develops, manufactures, leases, and sells electric vehicles, and energy generation and storage systems in the United States, China, and internationally. The company operates in two segments, Automotive, and Energy Generation and Storage.

TSLA close price for a period of 5 years

Source: Yahoo! Finance

Image generated from code

Algorithm

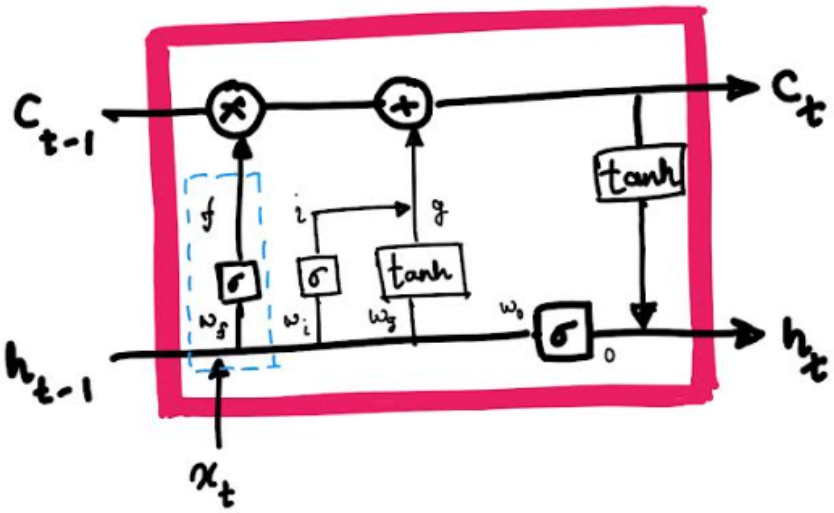


LSTM Architecture

Forget gate - Combines current input and previous hidden states to decide which previous samples to forget and which to remember

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf})$$

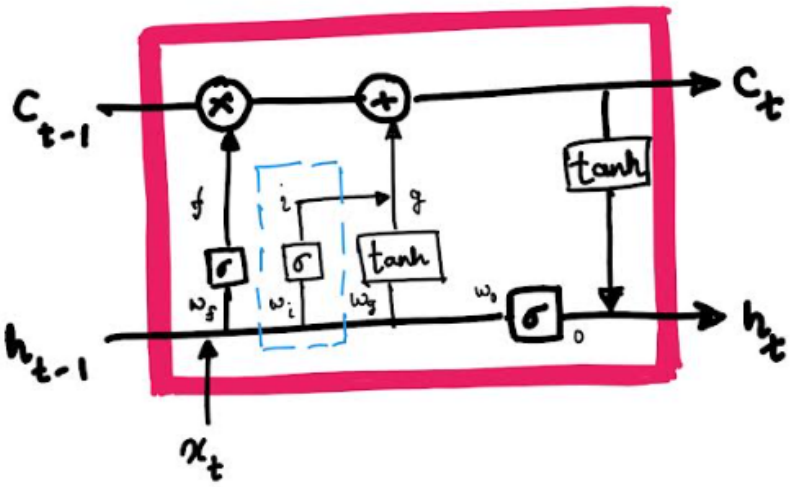
f is ~0 or ~1



Input gate - Combines current input and previous hidden states to decide which previous samples to forget and which to remember

i is ~0 or ~1

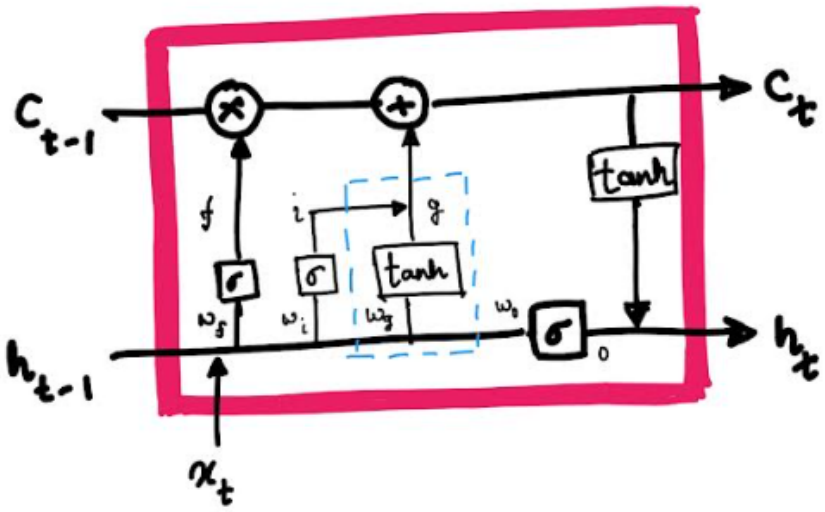
$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi})$$



Cell gate - Combines current input and previous hidden states to compute values

-1 < g < 1

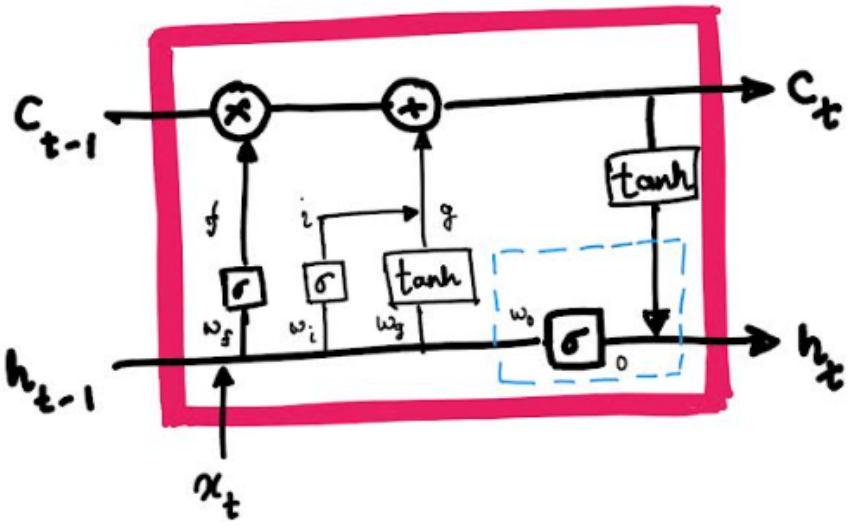
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg})$$



Output gate - Combines current input and previous hidden states and current cell states to compute the output of the LSTM cell

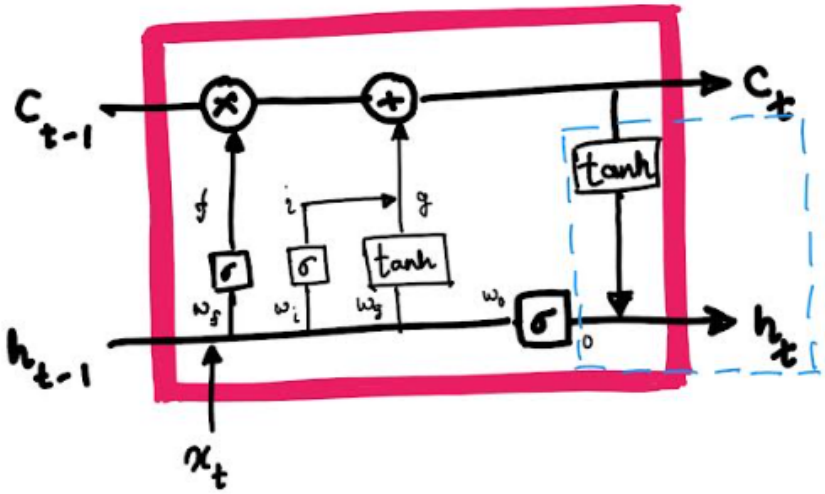
0 < o < 1

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho})$$



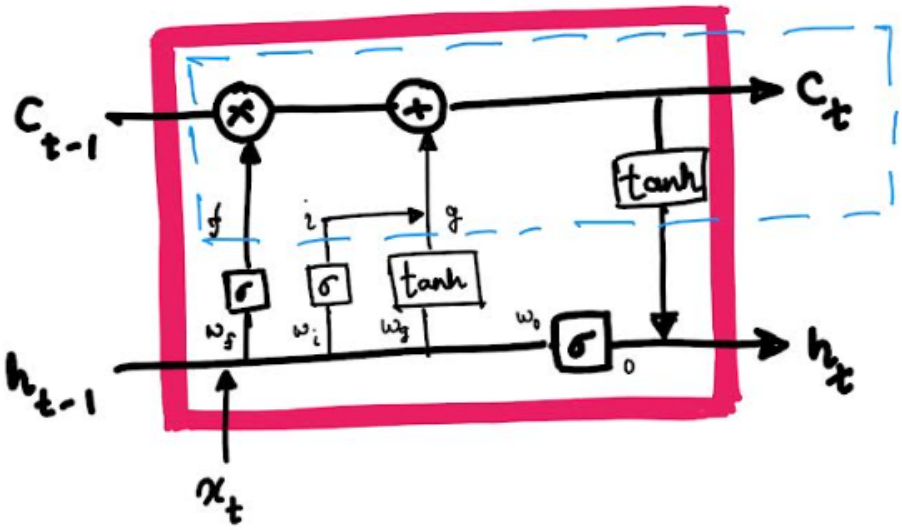
Hidden state - combination of previous hidden state and current input and newly computed cell state

$$h_t = o_t * \tanh(c_t)$$



Cell state - to remember previous cell state plus to remember current cell state

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$



Mean squared error loss function -

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

```
loss = nn.MSELoss()
input = torch.randn(3, 5, requires_grad=True)
target = torch.randn(3, 5)
output = loss(input, target)
output.backward()
```

(MSE code Pytorch)

Using Stochastic Gradient Descent to optimize the network parameters

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

α = learning rate

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)
```

Algorithm Code

(Importing Libraries and Modules)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf

import torch
import torch.nn as nn

from sklearn.preprocessing import MinMaxScaler
import sys
```

Creating the LSTM network class

```
class LSTMnet(nn.Module):
    def __init__(self, input_size, num_hidden, num_layers, output_size):
        super().__init__()

        self.input_size = input_size
        self.num_hidden = num_hidden
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size, num_hidden,
                            num_layers, batch_first=True)

        self.out = nn.Linear(num_hidden, output_size)

    def forward(self, x):
        h = torch.zeros(self.num_layers, x.size(
            0), self.num_hidden).requires_grad_()
        c = torch.zeros(self.num_layers, x.size(
            0), self.num_hidden).requires_grad_()

        y, (h_, c_) = self.lstm(x, (h.detach(), c.detach()))

        o = self.out(y)
        return o, (h_, c_)
```

Network Parameters used for training the model

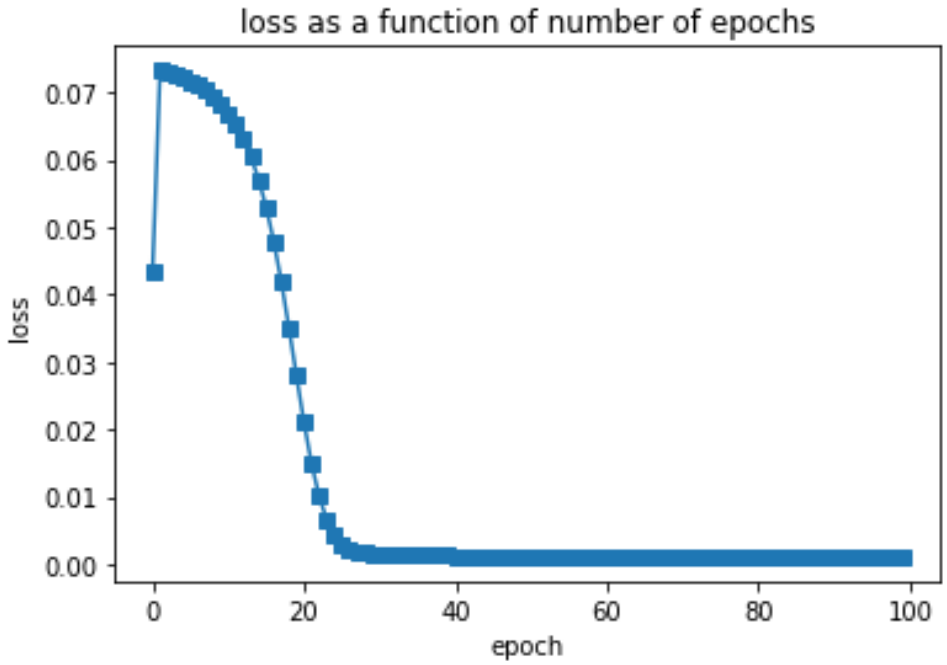
```
# Network parameters
input_size = 1
hidden_size = 20
num_layers = 2
output_size = 1
seqlength = 7
batchsize = 1
```

Training the model

```
for timeI in range(N-seqlength):
    X = data[timeI:timeI+seqlength].view(seqlength, 1, 1)
    y = data[timeI+seqlength].view(1, 1)

    yHat, hidden_states = net(X)
    finalValue = yHat[-1]

    loss = lossfun(finalValue, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    seglosses.append(loss.item())
```

Training Results

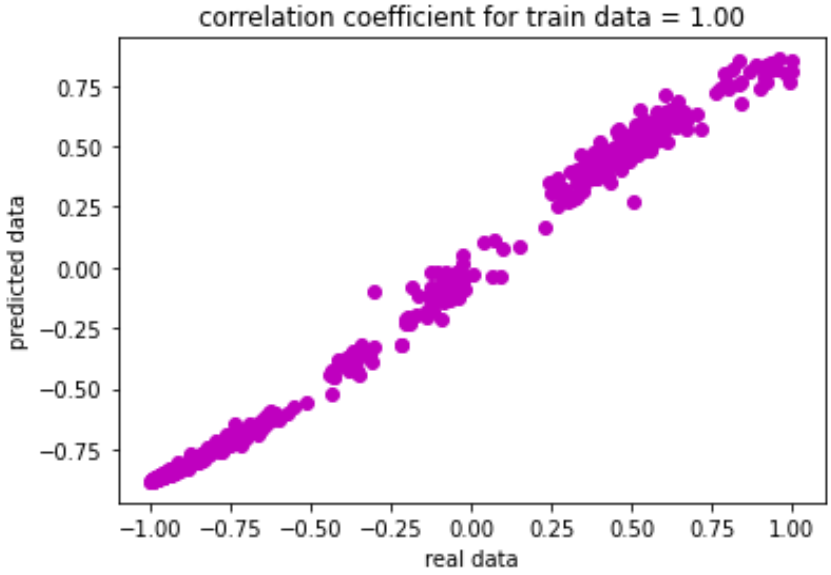
Loss decreases as the training progresses and reaches close to zero



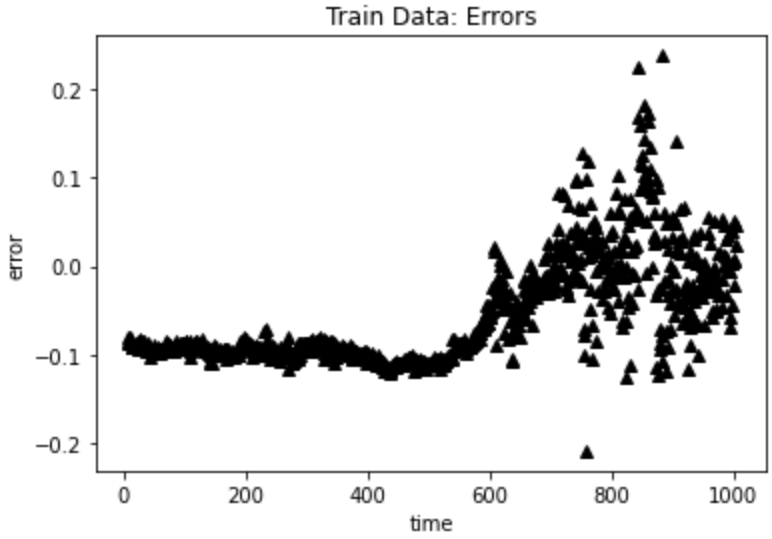
Results

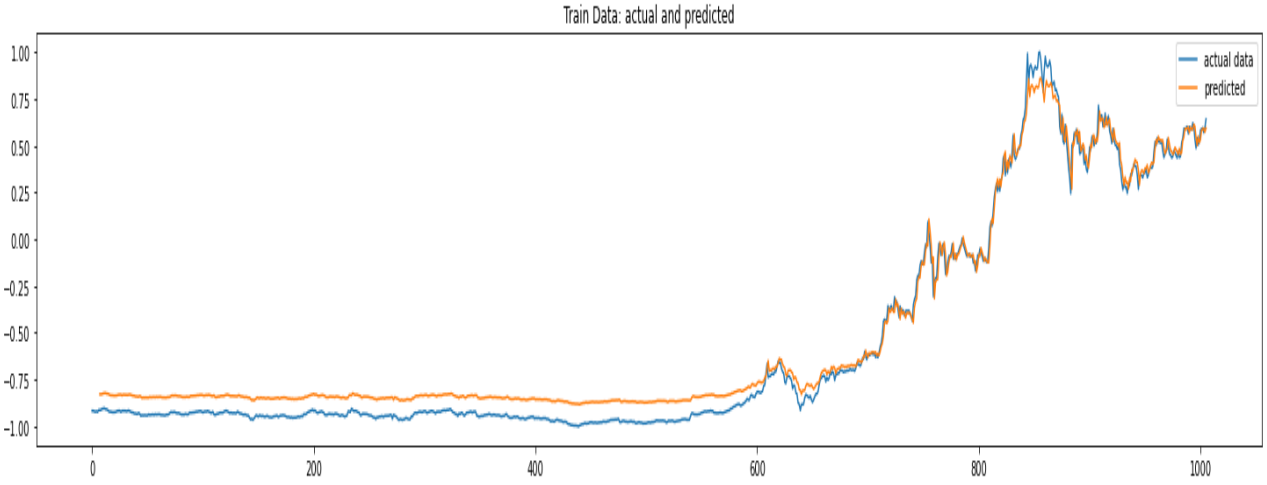
After training the network:

The correlation coefficient between real values and predicted values for **train data** is 1.00

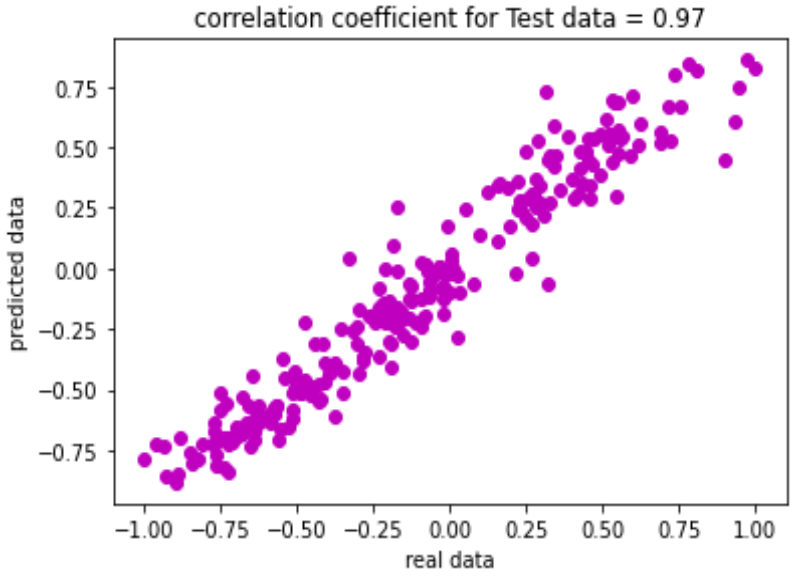


Initially, there is some negative error but after some training, the error is clustered around 0.0





The correlation coefficient between real values and predicted values for **test data** is 0.97



For test data, there are a few outlier errors; however, the majority of data point for error is crowded near about 0.0

References:

1. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. 9(8), 1735–1780 (1997)
2. Introduction to Deep Learning : From Logical Calculus to Artificial Intelligence, Sandro Skansi
3. J.L. Elman, Finding structure in time. Cogn. Sci. 14, 179–211 (1990)