**Problem Description**

The problem we are going to look at in this tutorial is predicting the future of sequential data, climatic data, using Time Series Forecasting model called Long Short Term Memory (LSTM) networks, which using the time series data values to make predictions about future values on our historical data points. According to several online sources, the LSTM model has improved Google’s speech recognition, greatly improved machine translations on Google Translate, and the answers of Amazon’s Alexa. This neural system is also employed by Facebook, reaching over 4 billion LSTM-based translations per day as of 2017 [1].

The tutorial is organized as follows. Section II presents the structure of the Long Short-Term Memory (LSTM) neural network model. In Section III, Training Data for LSTM Model and, finally, LSTM Model Development discussed. In last Section, we show how to handle missing values of dataframe.

**2. The structure of the Long Short-Term Memory (LSTM) neural network**

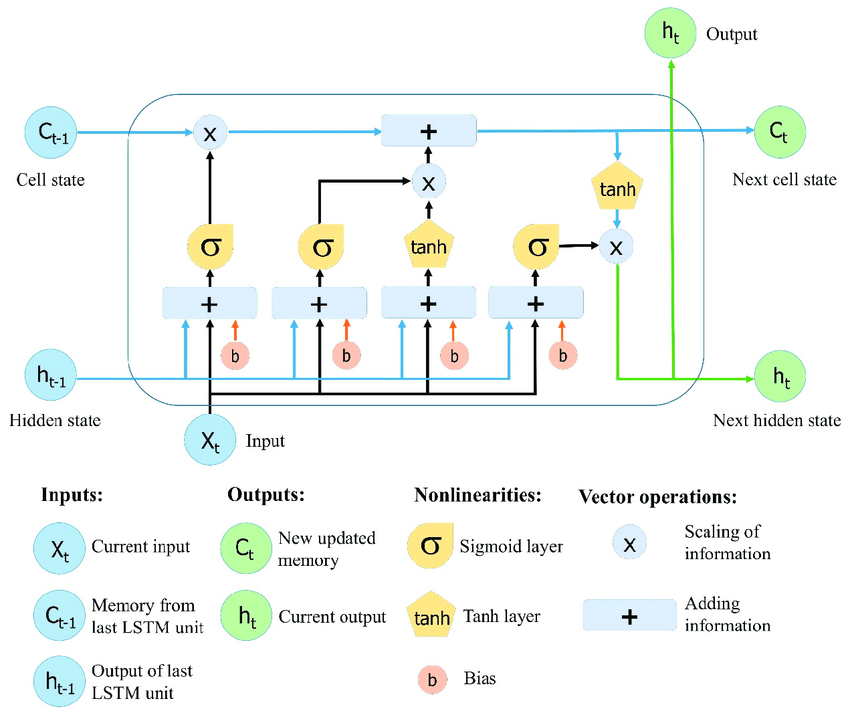
2.1. An Artificial Neural Network (ANN) is a computing system inspired by the human nervous system. An ANN model is a data-driven mathematical model that has the ability to solve problems through machine-learning neurons [2].

2.2. The Recurrent Neural Network (RNN) consists of an input layer, one or more hidden layers, and an output layer. RNNs have chain-like structures of repeating modules with the idea behind using these modules as a memory to store important information from previous processing steps, as shown in figure 1 [2].



Figure 1.Recurrent Neural Network (RNN) scheme [3].

2.3. Long Short-Term Memory (LSTM) is a special kind of RNN, capable of learning long-term dependencies and remembering information for prolonged periods of time as a default. The repeating module has a different structure, where instead of a single neural network like a standard RNN, it has four interacting layers with a unique method of communication. The structure of the LSTM neural network is shown in Figure 2 [2].



**Figure 2.** The structure of the Long Short-Term Memory (LSTM) neural network. Reproduced from Yan [4].

#### **3. Training Data for LSTM Model**

#### **3.1. Convert Training Data**

In order to train LSTM on our data, we need to convert our data into the shape accepted by the LSTM. We need to convert our data into three-dimensional format. The first dimension is the number of records or rows. The second dimension is the number of time steps and the last dimension is the number of indicators. Since we are only using one feature, the number of indicators will be one. Execute the following script:

features\_set = []

features\_set = np.array(features\_set)

features\_set = np.reshape(features\_set, (features\_set.shape[0], features\_set.shape[1], 1))

In the script above we create a list: feature\_set. We need to convert the feature\_set list to the numpy array before we can use it for training.

### **3.2. Transform Time Series to Scale**

LSTMs expect data to be within the scale of the activation function used by the network.

The default activation function for LSTMs is the hyperbolic tangent (tanh), which outputs values between 0 and 1. This is the preferred range for the time series data.

We can transform the dataset to the range [0, 1] using the [MinMaxScaler class](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html), where we need to reshape our NumPy arrays before transforming and use MinMaxScaler class from the sklear.preprocessing library to scale our data between 0 and 1, [5].

The feature\_range parameter is used to specify the range of the scaled data.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range = (0, 1))

df\_scaled = scaler.fit\_transform(df)

Now is the time to create our LSTM. The LSTM model that we are going to create will be a sequential model with multiple layers. We will add four LSTM layers to our model followed by a dense layer that predicts the future climatic data.

## 3.3. LSTM Model Development

The shape of the input data must be specified in the LSTM layer using the “batch\_input\_shape” argument as a tuple that specifies the expected number of observations to read each batch, the number of time steps, and the number of features.

The batch size is often much smaller than the total number of samples. The import parameter in defining the LSTM layer is the number of neurons, also called the number of memory units or blocks. Once the network is specified, it must be compiled into an efficient symbolic representation using a backend mathematical library, such as TensorFlow.

from keras.models import Sequential

from keras.layers import  Dense,LSTM,Dropout

In the script above we imported the Sequential class from keras.models library and Dense, LSTM, and Dropout classes from keras.layers library.

**3.4. Build the model**

As a first step, we need to instantiate the Sequential class. This will be our model class and we will add LSTM, Dropout and Dense layers to this model. Execute the following script:

model = Sequential()

To add a layer to the sequential model, the add method is used. Inside the add method, we passed our LSTM layer. The first parameter to the LSTM layer is the number of neurons or nodes that we want in the layer. The second parameter is return\_sequences, which is set to true since we will add more layers to the model. The first parameter to the input\_shape is the number of time steps while the last parameter is the number of indicators.

Let's now add a dropout layer to our model. Dropout layer is added to avoid over-fitting, which is a phenomenon where a machine learning model performs better on the training data compared to the test data [6].

**Now, it’s time to build the model.**We will build the **LSTM** with 50 neurons and **4 hidden layers**. Finally, we will assign 1 neuron in the output layer for predicting the normalized stock price. We will use the MSE loss function and the Adam stochastic gradient descent optimizer.

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(forecast\_features\_set.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50))

model.add(Dropout(0.2))

##### **3.5. Creating Dense Layer**

To make our model more robust, we add a dense layer at the end of the model. The number of neurons in the dense layer will be set to 1 since we want to predict a single value in the output [6].

model.add(Dense(units = 1))

##### **3.6. Model Compilation**

Finally, we need to compile our LSTM before we can train it on the training data. The following script compiles our model.

model.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

We call the compile method on the Sequential model object, which is "model" in our case. In compiling the network, we must specify a loss function and optimization algorithm. We will use “mean\_squared\_error*”* as the loss function as it closely matches RMSE that we will are interested in, and the efficient ADAM optimization algorithm [6].

**4. Handelling missing values**

Data may be corrupt or unavailable, but it is also possible that your data has variable length sequences by definition. Those sequences with fewer timesteps may be considered to have missing values [7].

In this tutorial, we discover how to handle data with missing values for sequence prediction problems in Python. In Python, specifically Pandas, we mark missing values as NaN which we can replace all NaN values with a specific value which is the average of values between the NAN sequence.

Pandas**dataframe.ffill()** and **dataframe.bfill()** functions are used to find the average of values between the NAN sequence where dataframe. ‘ffill’ stands for ‘forward fill’ and will propagate last valid observation forward [8] and **dataframe.bfill()** stands for backward snd finally Pandas**dataframe.div()** is used to find the floating division of the backward and forward values of the NAN sequence.

dataframe= dataframe.ffill().add(dataframe.bfill()).div(2)

Note that in the case that forward value of the NAN sequence is NAN we use backward value to fill missing values.

**References**

1. A review on the long short-term memory model. Greg Van Houdt, Carlos Mosquera & Gonzalo Nápoles. Artificial Intelligence Review volume 53, pages5929–5955(2020).

2. Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. Xuan-Hien Le, Hung Viet Ho, Giha Lee and Sungho Jung. Water 2019, 11, 1387; doi:10.3390/w11071387.

3. Olah, C. Understanding LSTM Networks. Available online: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 26 June 2018).

4. Yan, S. Understanding LSTM and Its Diagrams. Available online: https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714 (accessed on 26 June 2018).

5. Time Series Forecasting with the Long Short-Term Memory Network in Python by Jason Brownlee on April 7, 2017 in Deep Learning for Time Series.

6. Time Series Analysis with LSTM using Python's Keras Library By Usman Malik. Available online : <https://stackabuse.com/time-series-analysis-with-lstm-using-pythons-keras-library/>

7. How to Handle Missing Timesteps in Sequence Prediction Problems with Python by Jason Brownlee on June 21, 2017 in Deep Learning for Time Series.

8. Online documentary available : https://www.geeksforgeeks.org/python-pandas-dataframe-ffill/