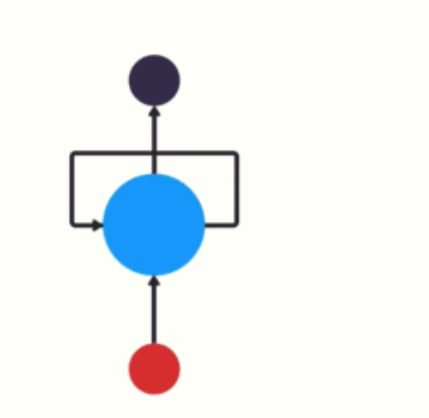
Machine Learning
Lab 6

What is LSTM and Why is it Important for Time Series?

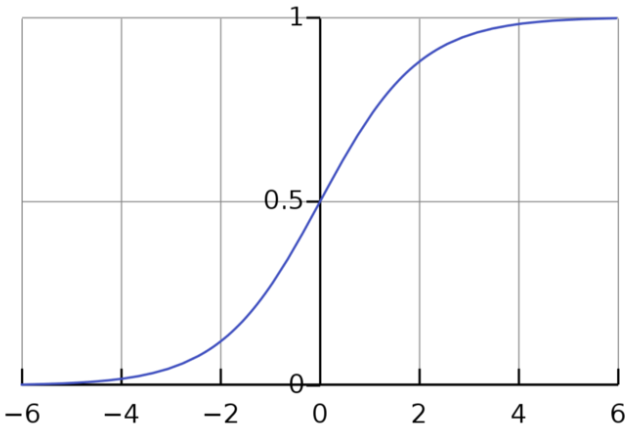
Long short-term memory (LSTM) is an artificial repetitive neural network (RNN) architecture used in the field of deep learning. Although it is not different from RNN in terms of working logic, it allows much longer sequences to work. According to RNN, it can largely solve the gradient problem.



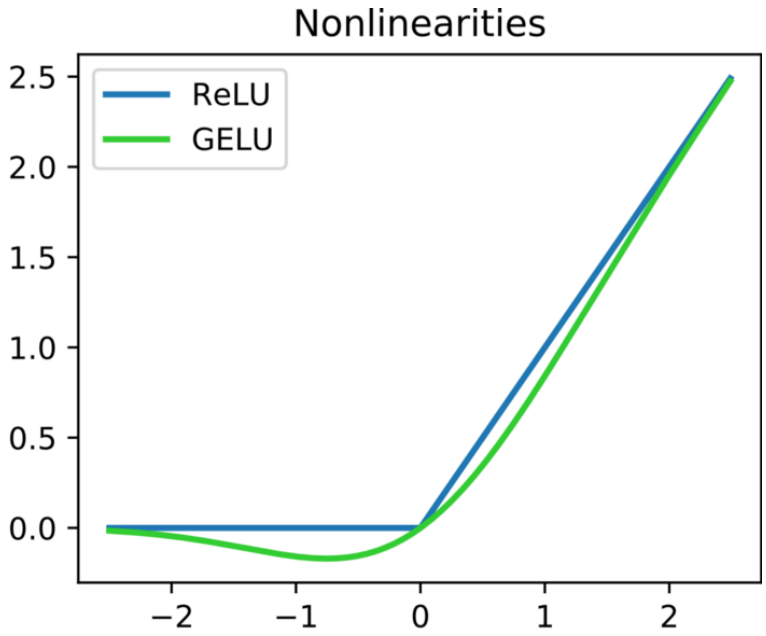
In short, LSTM models can store information for a certain period of time. Thanks to this feature of LSTM, using LSTM is extremely useful when dealing with time series or sequential data. Of course, as this article is part of the Time Series article series, an in-depth explanation of LSTM is beyond the scope of this article. But its use can give you some idea.

Preferred and Possible Activation Functions

When we know what it is to predict, the activation function we prefer can actually appear spontaneously. If we want to proceed without our knowledge at this stage, it will return to us in a way that will increase the training time.
For example, a sigmoid works well for a classifier, which we can understand most simply by looking at the graph for the sigmoid.



On the other hand, if we want to go easy, ReLU is computationally easier than tanh and sigmoid, as it includes simpler mathematical operations, and requires less power.



Loading Data

After installing the required libraries and loading the data, we perform the separation. Here, the Test dataset will be data that we have reserved for use at the end, to use the values that we will predict with the model. We will separate the training dataset into x_train and y_train in the future, but before all that, we scale the data. (Scaling the data is very important when working on neural networks, this model shortens the training time considerably.)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime as dt

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM

# Load Data
florida = pd.read_csv('github/florida_file_.csv')
florida.head()
florida.tail(5)
florida["Date"] = pd.to_datetime(florida["Date"])

florida = florida[["Date", "Avg_Temp"]]
# florida = florida["Avg_Temp"].resample('MS').mean()
florida = florida.fillna(florida.bfill())
florida.columns = ['Date', 'Avg_Temp']

train = florida[:-225]
len(train)
test = florida[-225:]
len(test)
train_dates = pd.to_datetime(train['Date'])
test_dates  = pd.to_datetime(test['Date'])
```

Prepare Data

After scaling all the values in the data set between 0 and 1, we specify the number of days to predict. After this value, we separate the data as x_train and y_train. With the value of 225 we have given, the model will work in such a way that it examines 225 data and predicts the next one, then examines 225 data again and tries to predict the next one.

After filling our lists, we convert them to NumPy arrays. then we reshape x_train so that x_train can work with the neural network.

```
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data =
scaler.fit_transform(train['Avg_Temp'].values.reshape(-1,1))

prediction_days = 225

x_train = []
y_train = []

for x in range(prediction_days, len(scaled_data)):
    x_train.append(scaled_data[x-prediction_days:x, 0])
    y_train.append(scaled_data[x, 0])


x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

Build the Model

In this step, we set up our model, since it is a basic neural network, we set up a Sequential model directly. Here, after choosing LSTM layers, we applied Dropout to increase efficiency and eliminate useless nodes, so that after each LSTM layer, we didn't move with the worst-performing nodes to the next step. Here we have eliminated 20% of the total number of nodes.

Attention here!!! We have to give the input_shape value to our first LSTM layer, because our model does not know the size of the data it will process, we do not need to enter this value in the next steps. In the last step, we returned a single value with the Dense layer, so this estimated value will be the average temperature predicted by our model.

```
model = Sequential()

model.add(LSTM(units =128, activation='relu', return_sequences=True,
input_shape = (x_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(units =128, activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units =128, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1)) # Prediction of the next value
```

Then we compile our model, we prepare the test data set that our model will see with the completed model. Finally, we reshape x_test and bring it to the appropriate format for our model.

```
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

history = model.fit(x_train, y_train, epochs = 25, batch_size=32,
validation_split=0.1)
```

Prediction

Then we perform a predict operation on our model. Then we normalize these estimated values with the inverse_transform operation in order to compare these predicted values with the actual values we have.

```
actual_temp = test['Avg_Temp'].values
total_temp = pd.concat((train['Avg_Temp'], test['Avg_Temp']),axis=0)

model_inputs = total_temp[len(total_temp)-len(test)-
prediction_days:].values
model_inputs = model_inputs.reshape(-1,1)
model_inputs = scaler.transform(model_inputs)


# Make Predictions on Test Data
x_test = []

for x in range(prediction_days, len(model_inputs)):
    x_test.append(model_inputs[x-prediction_days:x, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

pred = model.predict(x_test)
pred = scaler.inverse_transform(pred)
```

Activity

Understand this LSTM case study;

Develop some codes to evaluate the prediction performance;

Develop some codes to visualise the prediction results;

What if we want to change the number of days for input?