What is LSTM?

LSTM stands for Short Term Long Term Memory. It is a model or an architecture that extends the memory of recurrent neural networks. Typically, recurrent neural networks have “short-term memory” in that they use persistent past information for use in the current neural network. Essentially, the previous information is used in the current task. This means that we do not have a list of all of the previous information available for the neural node.

Forecast Time Series with LSTM I hope you have understood what time series forecasting means and what are LSTM models. Now I will be heading towards creating a machine learning model to forecast time series with LSTM in Machine Learning.

For this task to forecast time series with LSTM, I will start by importing all the necessary packages we need:

```
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# fix random seed for reproducibility
numpy.random.seed(7)
```

Now let’s load the data, and prepare the data so that we can use it on the LSTM model, you can download the dataset from kaggle i'm using "airline passenger Data".

```
# load the dataset
dataframe = pandas.read_csv('/content/archive (1).zip', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

Now, I will split the data into training sets and test sets:

```
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))
```

```
97 48
```

Now before training the data on the LSTM model, we need to prepare the data so that we can fit it on the model, for this task I will define a helper function:

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

Now, we need to reshape the data before applying it into the LSTM model:

```
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Now as all the tasks are completed concerning data preparation to fit into the LSTM model, it time to fit the data on the model and let's train the model:

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

```
Epoch 1/100
95/95 - 4s - loss: 0.0487 - 4s/epoch - 39ms/step
Epoch 2/100
95/95 - 0s - loss: 0.0252 - 168ms/epoch - 2ms/step
Epoch 3/100
95/95 - 0s - loss: 0.0183 - 195ms/epoch - 2ms/step
Epoch 4/100
95/95 - 0s - loss: 0.0162 - 180ms/epoch - 2ms/step
Epoch 5/100
95/95 - 0s - loss: 0.0153 - 182ms/epoch - 2ms/step
Epoch 6/100
95/95 - 0s - loss: 0.0141 - 174ms/epoch - 2ms/step
Epoch 7/100
95/95 - 0s - loss: 0.0128 - 170ms/epoch - 2ms/step
Epoch 8/100
95/95 - 0s - loss: 0.0117 - 172ms/epoch - 2ms/step
Epoch 9/100
95/95 - 0s - loss: 0.0103 - 186ms/epoch - 2ms/step
Epoch 10/100
95/95 - 0s - loss: 0.0089 - 172ms/epoch - 2ms/step
Epoch 11/100
95/95 - 0s - loss: 0.0074 - 162ms/epoch - 2ms/step
Epoch 12/100
95/95 - 0s - loss: 0.0061 - 168ms/epoch - 2ms/step
Epoch 13/100
95/95 - 0s - loss: 0.0049 - 167ms/epoch - 2ms/step
Epoch 14/100
95/95 - 0s - loss: 0.0040 - 171ms/epoch - 2ms/step
Epoch 15/100
95/95 - 0s - loss: 0.0033 - 182ms/epoch - 2ms/step
Epoch 16/100
95/95 - 0s - loss: 0.0028 - 175ms/epoch - 2ms/step
Epoch 17/100
95/95 - 0s - loss: 0.0025 - 161ms/epoch - 2ms/step
Epoch 18/100
95/95 - 0s - loss: 0.0023 - 162ms/epoch - 2ms/step
Epoch 19/100
95/95 - 0s - loss: 0.0021 - 166ms/epoch - 2ms/step
Epoch 20/100
95/95 - 0s - loss: 0.0021 - 158ms/epoch - 2ms/step
Epoch 21/100
95/95 - 0s - loss: 0.0021 - 166ms/epoch - 2ms/step
Epoch 22/100
95/95 - 0s - loss: 0.0020 - 159ms/epoch - 2ms/step
Epoch 23/100
95/95 - 0s - loss: 0.0020 - 165ms/epoch - 2ms/step
Epoch 24/100
95/95 - 0s - loss: 0.0021 - 163ms/epoch - 2ms/step
Epoch 25/100
95/95 - 0s - loss: 0.0021 - 177ms/epoch - 2ms/step
Epoch 26/100
95/95 - 0s - loss: 0.0021 - 175ms/epoch - 2ms/step
Epoch 27/100
95/95 - 0s - loss: 0.0020 - 175ms/epoch - 2ms/step
Epoch 28/100
95/95 - 0s - loss: 0.0021 - 171ms/epoch - 2ms/step
Epoch 29/100
95/95 - 0s - loss: 0.0021 - 157ms/epoch - 2ms/step
```

Now, let's make predictions and visualize the time series trends by using the matplotlib package in python:

```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

```
3/3 [==============================] - 1s 4ms/step
2/2 [==============================] - 0s 5ms/step
```