Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras

ABSTRACT

Technology is developing at a rapid pace and the luxury of the human being is increasing. Now a days, we can travel to almost all famous and important cities by plane. Due to more demand and profits in airline services, many airline services have been established to provide service to the customers and to gain profits.

As an airline service, it is very important to know the expected number of travellers for the next months in order to change the frequency of flights between respective routes in order to increase the profits and decrease the loses and can provide excellent service to the customers by increasing the number of flights during more demand and decrease the frequency of flights during less demand.

Now, we are going to build a model which can predict the expected number of passengers for the next month using LSTM Recurrent Neural Network in keras Deep Learning library having Tensor flow as back-end. Using the past 12 years international airline passengers data we are going to build a model, which can predict the expected number of passengers for next month. So that the airline services can use this model and plan their flights according to need based on expected number of passengers.

INTRODUCTION

It is very important to decide which neural network to use while building a model. we are going to work on prediction problem. Time series prediction problems are a difficult type of predictive modeling problem.

Unlike regression predictive modeling, time series also adds the complexity of a sequence dependence among the input variables.

A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained.

Another important thing we need to consider to while building a model training data. The performance and accuracy of the model is mostly based on the data we used to train the network. So, we should use correct data to train the model. In this model we are going to use past 12 years international airline passenger data.

In this project, we are going to predict the expected number of passengers for next month using the LSTM Recurrent Neural Network by training the network using the past 12 years international airline passengers data. So, it makes easier for the airline services to plan the frequency of flights for the next month.

PROBLEM STATEMENT

The problem we are going to look at in this project is the International Airline Passengers prediction problem. This is a problem where, given a year and a month, the task is to predict the number of international airline passengers in units of 1,000. The data ranges from January 1949 to December 1960, or 12 years, with 144 observations.

Irrespective of what our objective is, while working on any project, we should be able to provide a fully legitimate final “soft copy” of a project. This particular project provides a complexity of unlike regression predictive modeling, time series also adds the complexity of a sequence dependence among the input variables.

Building from bottom up, there are a number of things to consider and get right.

- In the beginning, it is very important to look at the data to make it suitable for training the neural network, which is called data cleaning
- we need to choose the neural network which works best for our problem
- we need to create the model
- Fit the model
- Evaluate the model

- Make predictions

LITERATURE REVIEW

The Literature Review is split into the three components:

They are

(1) Deep Learning

(2) Python

(3) keras

1. Deep Learning

Deep learning is a subset of a subset of the way that machines think. If we want to know about Deep learning we need to have a basic idea on Artificial Intelligence and Machine Learning. So, what is AI and Machine Learning.

Artificial intelligence

Artificial intelligence is the broadest term we use for technology that allows machines to mimic human behavior. AI can include logic, if-then rules, and more.

Machine Learning

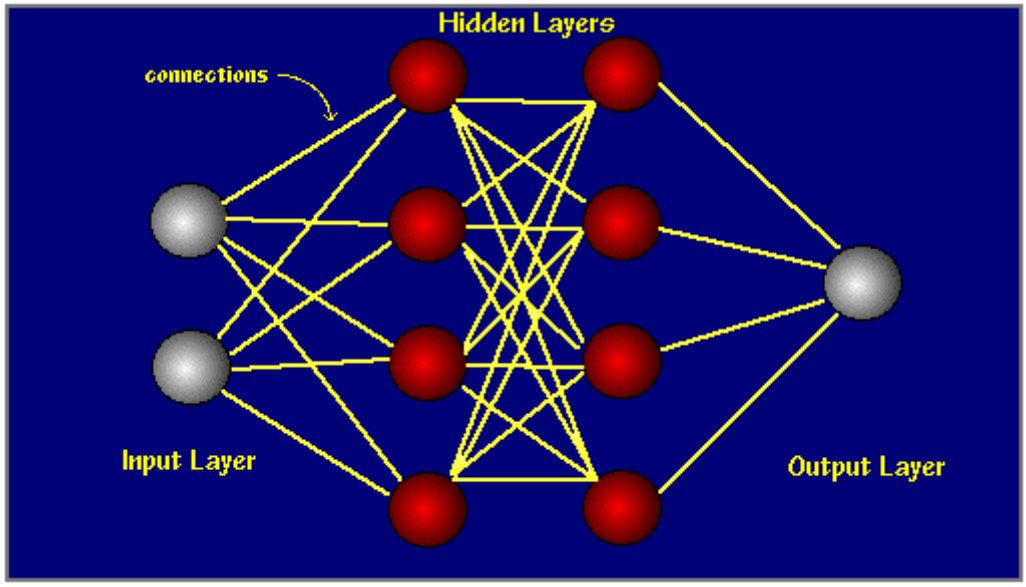
One subset of AI technology is machine learning. Machine learning allows computers to use statistics and use the experience to improve at tasks.

Deep Learning

A subset of machine learning is deep learning, which uses algorithms to train computers to perform tasks by exposing neural nets to huge amounts of data, allowing them to predict responses without needing to actually complete every task.

Neural Networks

Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.



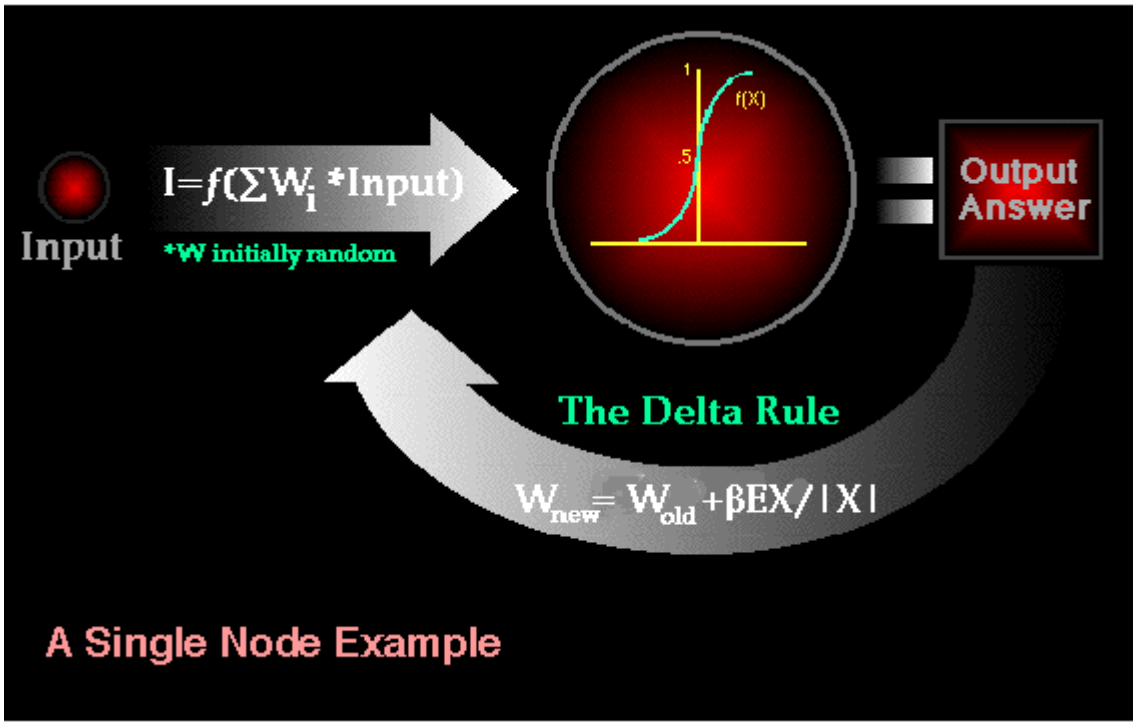
Most ANNs contain some form of 'learning rule' which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs[1] learn by example as do their biological counterparts; a child learns to recognize dogs from examples of dogs.

Although there are many different kinds of learning rules used by neural networks, this demonstration is concerned only with one; the delta rule. The delta rule is often utilized by the most common class of ANNs called BPNNs[2]. Back propagation is an abbreviation for the backwards propagation of error.

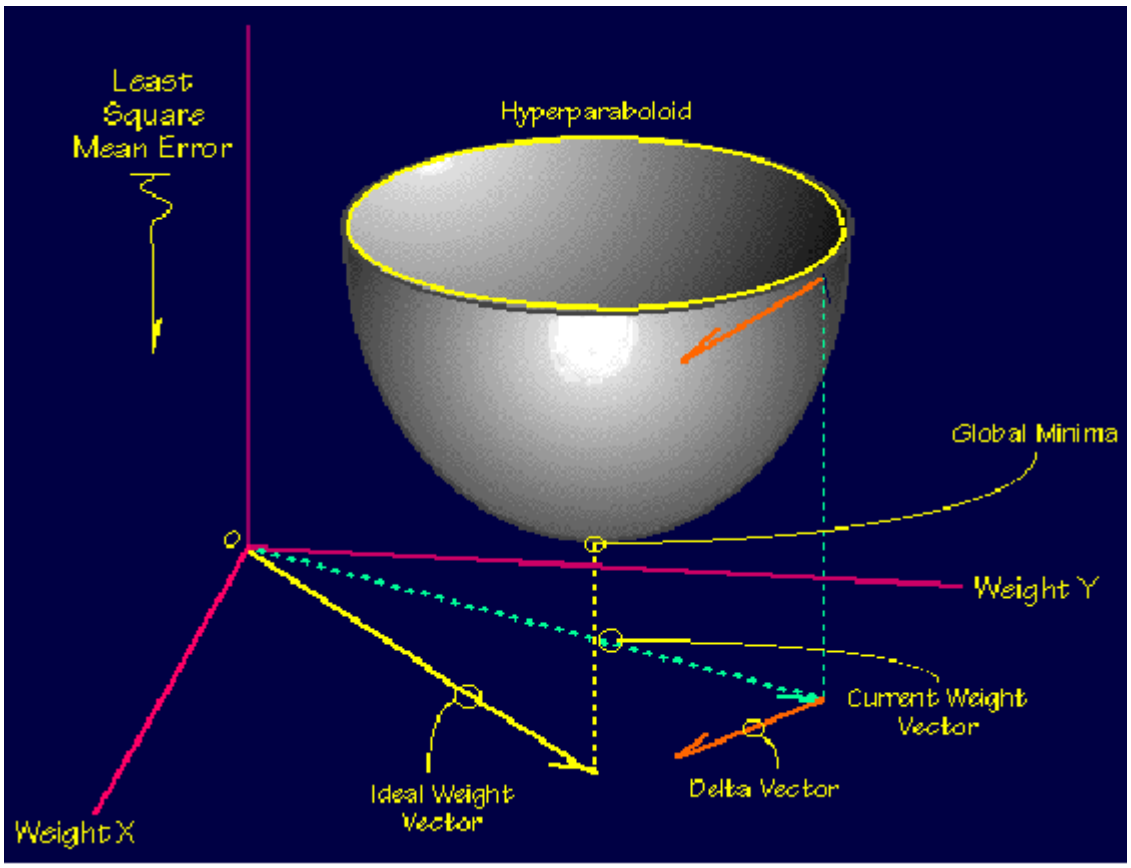
With the delta rule, as with other types of back propagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be.

It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. More graphically, the process looks something like figure below:

1 ANN – Artificial Neural Neutron.
2 BPNN – Back Propagation Neural Neutron.



Note also, that within each hidden layer node is a sigmoidal activation function which polarizes network activity and helps it to stablize. Back propagation performs a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is that theoretical solution with the lowest possible error. The error surface itself is a hyper paraboloid but is seldom 'smooth' as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minum' which is not the the best overall solution.



Since the nature of the error space can not be known a prioi, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data. To do this,

the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no back propagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

It is also possible to over-train a neural network, which means that the network has been trained exactly to respond to only one type of input; which is much like rote memorization. If this should happen then learning can no longer occur and the network is referred to as having been "grandmothered" in neural network jargon. In real-world applications this situation is not very useful since one would need a separate grandmothered network for each new kind of input. We can find more information in this page[i].

2. Python

Installation of Python is required for this project. Python is a programming language that is dynamic and object-oriented. It emphasises on readability of the code and thereby, features indentation to delimit blocks of code.

Python can be downloaded from official python website[ii]. Version 3.6 has been used in our project.

3. Keras

In this project, we are going to use keras deep learning library to create a model. Keras is a high-level neural networks API, written in Python and capable of running on top of Tensor Flow, CNTN or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. We can find more information in keras documentation[iii]

REQUIREMENTS

Software Requirements

Anaconda Cloud

Anaconda is an open-source package manager, environment manager, and distribution of the Python and R programming languages. It is commonly used for large-scale data processing, scientific computing, and predictive analytics, serving data scientists, developers, business analysts, and those working in DevOps.

Anaconda offers a collection of over 720 open-source packages, and is available in both free and paid versions. The Anaconda distribution ships with the conda command-line utility. You can learn more about Anaconda

and `conda` by reading the Anaconda Documentation pages[iv].

Installing Anaconda cloud

The best way to install Anaconda is to download the latest Anaconda installer bash script, verify it, and then run it and you can see this page[v] for installation steps

a. Python

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

To install python using command line

```
$ conda install -c anaconda python
```

for more information , you can see this page

b. Pandas

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

To install pandas using command line

```
$ conda install -c conda-forge pandas
```

for more information, you can see this documentation[vi]

c. Jupyter

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

You can find jupyter by launching the anaconda navigator. For more information, you can see this documentation[vii]

d. Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code

To install Numpy using command line

```
$ conda install -c anaconda numpy
```

for more information, you can see this documentation[viii]

e. Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and Ipython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

To install matplotlib using command line

```
$ conda install  -c anaconda matplotlib
```

for more information, you can see this documentation[ix]

f. Scikit - learn

Scikit - Learn It is used for machine learning in python.

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib

To install scikit – learn using command line

```
$ conda install  -c anaconda scikit-learn
```

for more information, you can see this documentation[x]

g. Tensor flow

Tensor Flow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment ofcomputation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. We are not using tensor flow directly. But, we are using it as a back-end for keras

To install tensor flow using command line

```
$ conda install -c anaconda tensorflow-gpu
```

for more information, you can see this documentation[xi]

h. Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of Tensoe Flow, CNTN or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras allows us

- For easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- To Runs seamlessly on CPU and GPU.

To install keras using command line

```
$ conda install -c anaconda keras-gpu
```

CONCEPTUAL MODEL

Long Short-Term Memory Network

The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Back propagation Through Time and overcomes the vanishing gradient problem.

As such, it can be used to create large recurrent networks that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results.

Instead of neurons, LSTM networks have memory blocks that are connected through layers.

A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block's state and output. A block operates upon an input sequence and each gate within a block uses the sigmoid activation units to control whether they are triggered or not, making the change of state and addition of information flowing through the block conditional.

There are three types of gates within a unit:

- **Forget Gate**: conditionally decides what information to throw away from the block.
- **Input Gate**: conditionally decides which values from the input to update the memory state.
- **Output Gate**: conditionally decides what to output based on input and the memory of the block.

Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.

You can see how you may achieve sophisticated learning and memory from a layer of LSTMs, and it is not hard to imagine how higher-order abstractions may be layered with multiple such layers.

For advanced information, you can see this research paper[xii]

METHODOLOGY AND APPROACH

Outlined below is the methodology adopted to achieve the end result

The problem we are going to look at in this project is the International Airline Passengers prediction problem.

This is a problem where, given a year and a month, the task is to predict the number of international airline passengers in units of 1,000. The data ranges from January 1949 to December 1960, or 12 years, with 144 observations.

The dataset is available for free from the DataMarket webpage as a CSV download[xiii] with the filename "*international-airline-passengers.csv*".

Below is a sample of the first few lines of the file.

"Month","International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60"

"1949-01",112

"1949-02",118

"1949-03",132

"1949-04",129

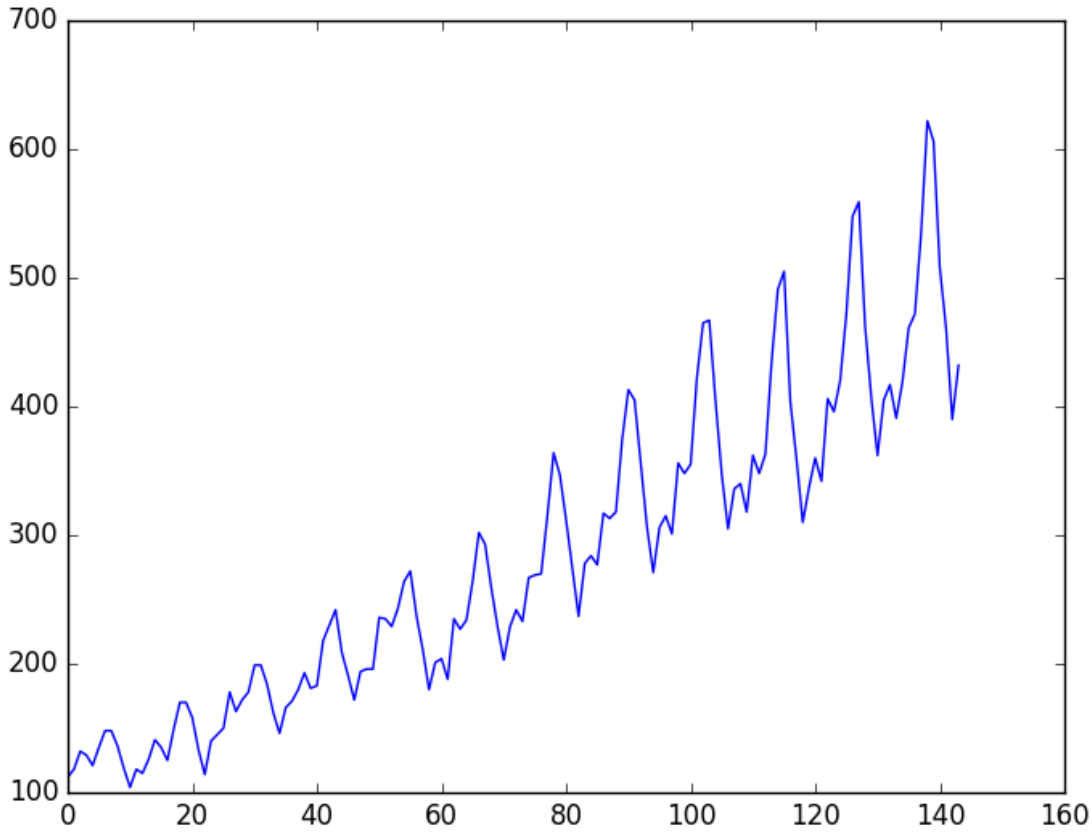
"1949-05",121

We can load this dataset easily using the Pandas library. We are not interested in the date, given that each observation is separated by the same interval of one month. Therefore, when we load the dataset we can exclude the first column.

The downloaded dataset also has footer information that we can exclude with the **skipfooter** argument to **pandas.read_csv()** set to 3 for the 3 footer lines. Once loaded we can easily plot the whole dataset. The code to load and plot the dataset is listed below.

```
import pandas
import matplotlib.pyplot as plt
dataset = pandas.read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
plt.plot(dataset)
plt.show()
```

You can see an upward trend in the dataset over time. You can also see some periodicity to the dataset that probably corresponds to the Northern Hemisphere vacation period



LSTM Network for Regression

We can phrase the problem as a regression problem.

That is, given the number of passengers (in units of thousands) this month, what is the number of passengers next month?

We can write a simple function to convert our single column of data into a two-column dataset: the first column containing this month's (t) passenger count and the second column containing next month's (t+1) passenger count, to be predicted.

Before we get started, let’s first import all of the functions and classes we intend to use.

```
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
```

Before we do anything, it is a good idea to fix the random number seed to ensure our results are reproducible.

```
# fix random seed for reproducibility
numpy.random.seed(7)
```

We can also use the code from the previous section to load the dataset as a Pandas data frame. We can then extract the NumPy array from the data frame and convert the integer values to floating point values, which are more suitable for modeling with a neural network.

```
# load the dataset
dataframe = pandas.read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
```

LSTMs are sensitive to the scale of the input data, specifically when the sigmoid (default) or tanh activation functions are used. It can be a good practice to rescale the data to the range of 0-to-1, also called normalizing. We can easily normalize the dataset using the **MinMaxScaler**[3] preprocessing class from the scikit-learn library.

```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

After we model our data and estimate the skill of our model on the training dataset, we need to get an idea of the skill of the model on new unseen data. For a normal classification or regression problem, we would do this using cross validation.

With time series data, the sequence of values is important. A simple method that we can use is to split the ordered dataset into train and test datasets. The code below calculates the index of the split point and separates the data

3 Minmax scaler - http://scikit-learn.org/stable/modules/preprocessing.html

into the training datasets with 67% of the observations that we can use to train our model, leaving the remaining 33% for testing the model.

```
# 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))
```

Now we can define a function to create a new dataset, as described above.

The function takes two arguments: the **dataset**, which is a NumPy array that we want to convert into a dataset, and the **look_back**, which is the number of previous time steps to use as input variables to predict the next time period — in this case defaulted to 1.

This default will create a dataset where X is the number of passengers at a given time (t) and Y is the number of passengers at the next time (t + 1).

It can be configured, and we will by constructing a differently shaped dataset in the next section.

```
# 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)
```

Let's take a look at the effect of this function on the first rows of the dataset (shown in the unnormalized form for clarity).

X	Y
112	118
118	132
132	129
129	121
121	135

if you compare these first 5 rows to the original dataset sample listed in the previous section, you can see the X=t and Y=t+1 pattern in the numbers.

Let’s use this function to prepare the train and test datasets for modeling.

```
# 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)
```

The LSTM network expects the input data (X) to be provided with a specific array structure in the form of: *[samples, time steps, features]*.

Currently, our data is in the form: [*samples*, *features*] and we are framing the problem as one time step for each sample. We can transform the prepared train and test input data into the expected structure using **numpy.reshape()** as follows:

```
# 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]))
```

We are now ready to design and fit our LSTM network for this problem.

The network has a visible layer with 1 input, a hidden layer with 4 LSTM blocks or neurons, and an output layer that makes a single value prediction. The default sigmoid activation function is used for the LSTM blocks. The network is trained for 100 epochs and a batch size of 1 is used.

```
# 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)
```

Once the model is fit, we can estimate the performance of the model on the train and test datasets. This will give us a point of comparison for new models.

Note that we invert the predictions before calculating error scores to ensure that performance is reported in the same

units as the original data (thousands of passengers per month).

```
# 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]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

Finally, we can generate predictions using the model for both the train and test dataset to get a visual indication of the skill of the model.

Because of how the dataset was prepared, we must shift the predictions so that they align on the x-axis with the original

dataset. Once prepared, the data is plotted, showing the original dataset in blue, the predictions for the training dataset in green, and the predictions on the unseen test dataset in red

```
# 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()
```

OUTPUT AND RESULTS

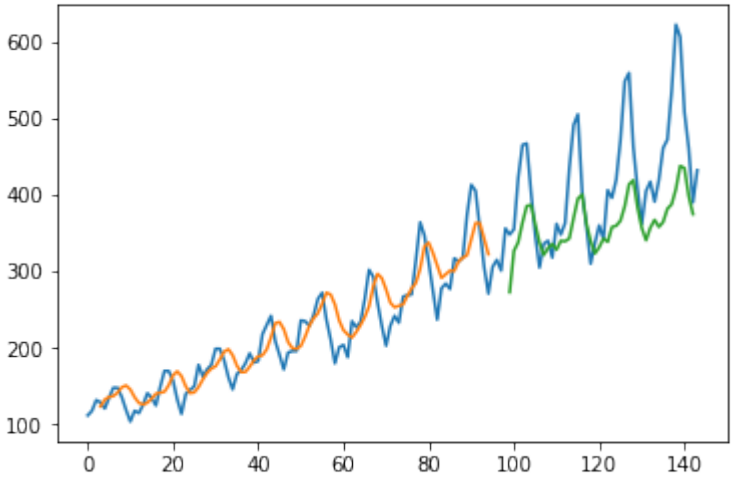
We can see that model did an excellent job of fitting both the training and test datasets

```
Epoch 1/1
 - 9s - loss: 0.0057
Epoch 1/1
 - 2s - loss: 0.0141
Epoch 1/1
 - 2s - loss: 0.0099
Epoch 1/1
 - 2s - loss: 0.0072
Epoch 1/1
 - 2s - loss: 0.0060
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
```

```
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0055
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0054
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0051
Epoch 1/1
 - 2s - loss: 0.0051
Epoch 1/1
 - 2s - loss: 0.0051
Epoch 1/1
 - 2s - loss: 0.0051
Epoch 1/1
 - 2s - loss: 0.0051
Epoch 1/1
```

```
 - 2s - loss: 0.0051
Epoch 1/1
 - 2s - loss: 0.0050
Epoch 1/1
 - 2s - loss: 0.0050
Epoch 1/1
 - 2s - loss: 0.0050
Epoch 1/1
 - 2s - loss: 0.0050
Epoch 1/1
 - 2s - loss: 0.0050
Epoch 1/1
 - 2s - loss: 0.0049
Epoch 1/1
 - 2s - loss: 0.0049
Epoch 1/1
 - 2s - loss: 0.0049
Epoch 1/1
 - 2s - loss: 0.0049
Epoch 1/1
 - 2s - loss: 0.0048
Epoch 1/1
 - 2s - loss: 0.0048
Epoch 1/1
 - 2s - loss: 0.0048
Epoch 1/1
 - 2s - loss: 0.0048
Epoch 1/1
 - 2s - loss: 0.0047
Epoch 1/1
 - 2s - loss: 0.0047
Epoch 1/1
 - 2s - loss: 0.0047
Epoch 1/1
 - 2s - loss: 0.0046
Epoch 1/1
 - 2s - loss: 0.0046
Epoch 1/1
 - 2s - loss: 0.0046
Epoch 1/1
 - 2s - loss: 0.0045
Epoch 1/1
 - 2s - loss: 0.0045
Epoch 1/1
 - 2s - loss: 0.0045
Epoch 1/1
 - 2s - loss: 0.0044
Epoch 1/1
 - 2s - loss: 0.0044
Epoch 1/1
 - 2s - loss: 0.0044
Epoch 1/1
 - 2s - loss: 0.0043
Epoch 1/1
 - 2s - loss: 0.0043
Epoch 1/1
 - 2s - loss: 0.0042
Epoch 1/1
 - 2s - loss: 0.0042
Epoch 1/1
 - 2s - loss: 0.0042
Epoch 1/1
 - 2s - loss: 0.0041
Epoch 1/1
```

```
 - 2s - loss: 0.0041
Epoch 1/1
 - 2s - loss: 0.0040
Epoch 1/1
 - 2s - loss: 0.0040
Epoch 1/1
 - 2s - loss: 0.0040
Epoch 1/1
 - 2s - loss: 0.0039
Epoch 1/1
 - 2s - loss: 0.0038
Epoch 1/1
 - 2s - loss: 0.0038
Epoch 1/1
 - 2s - loss: 0.0037
Epoch 1/1
 - 2s - loss: 0.0036
Train Score: 29.79 RMSE
Test Score: 79.47 RMSE
```



We can see that the model has an average error of about 29 passengers (in thousands) on the training dataset, and about 79 passengers (in thousands) on the test dataset, which is really good.

CONCLUSION

The model was successfully created by using LSTM Recurrent Neural Network and trained and it produced excellent results, We can see that the model has an average error of about 29 passengers (in thousands) on the training dataset, and about 79 passengers (in thousands) on the test dataset, which is really good. If we train the model with more efficient data, there is more change to increase the efficiency of the model and we can get better results.

For detailed information about neural networks , you can read this online book[xiv] by Michael Nielsen.

For online lecture videos, you can watch Cs231n tutorial[xv] from stanford university in youtube.

REFERENCES

i . Basics of neural networks - http://neuralnetworksanddeeplearning.com/

ii . python documentation - https://www.python.org/doc/

iii . Keras documentation - https://keras.io/

iv . Anaconda cloud - https://docs.anaconda.com/anaconda-cloud/

v . Anconda Cloud installation - https://docs.anaconda.com/anaconda/install/linux

vi. pandas documentation - https://pandas.pydata.org/pandas-docs/stable/

vii . Jupyter documentation - https://pandas.pydata.org/pandas-docs/stable/

viii. Numpy documentation - https://docs.scipy.org/doc/

ix . Matplotlib documentation - https://matplotlib.org/contents.html

x . Scikit – learn documentation - http://scikit-learn.org/stable/documentation.html

xi . Tensor flow documentation - https://www.tensorflow.org/tutorials/

REFERENCES

xii . Research paper by google on LSTM - https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf

xiii . International airline Data set - https://datamarket.com/data/set/22u3/international-airline-passengers-monthly-totals-in-thousands-jan-49-dec-60#!ds=22u3&display=line

xiv . Neural Networks and Deep Learning book - http://neuralnetworksanddeeplearning.com/about.html

xv . CS 231n tutorial - https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv