Need help with LSTMs in Python? Take the FREE Mini-Course.

How to Use Features in LSTM Networks for Time Series Forecasting

by **Jason Brownlee** on April 19, 2017 in **Long Short-Term Memory Networks**

The Long Short-Term Memory (LSTM) network in Keras supports multiple input features.

This raises the question as to whether lag observations for a univariate time series can be used as features for an LSTM and whether or not this improves forecast performance.

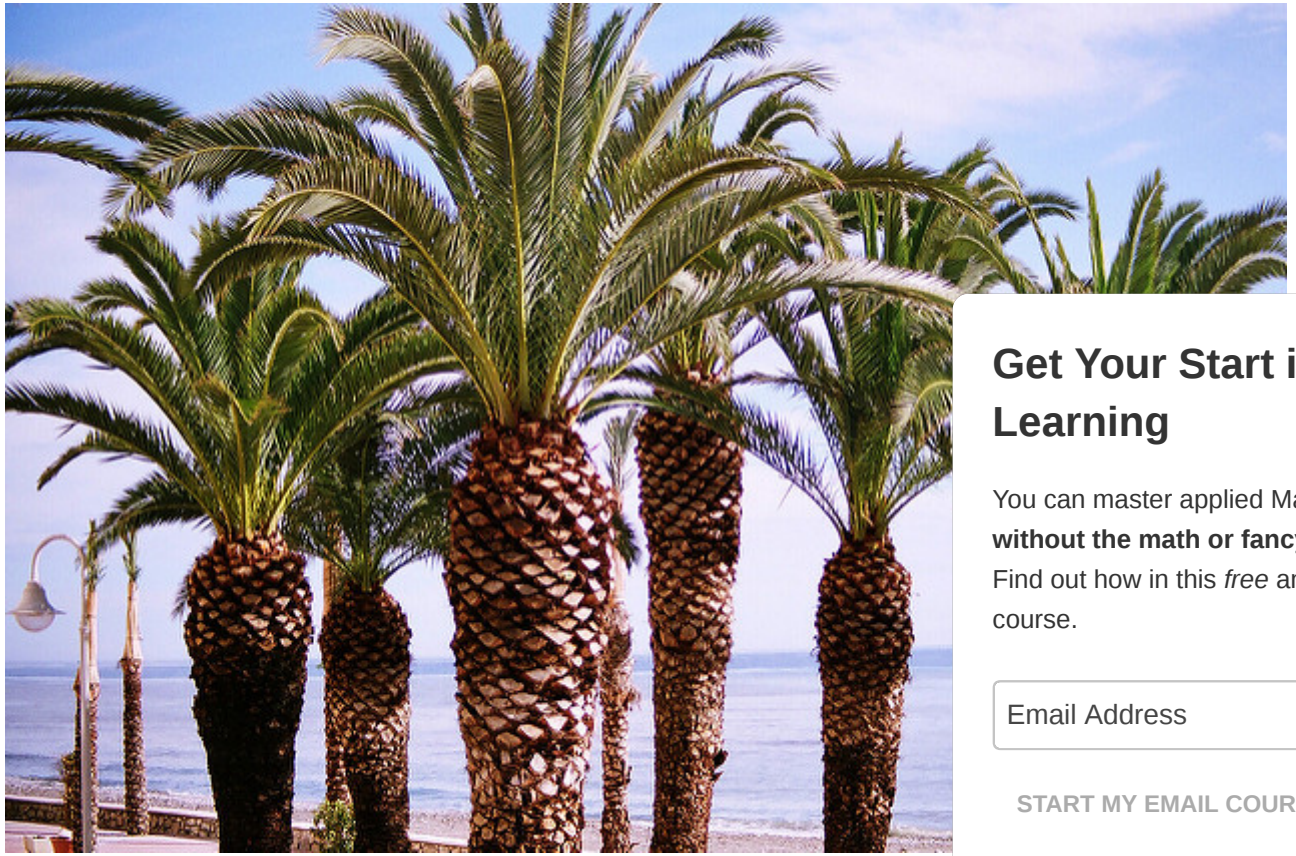
In this tutorial, we will investigate the use of lag observations as features in LSTM models in Python.

After completing this tutorial, you will know:

- How to develop a test harness to systematically evaluate LSTM features for time series forecast

- The impact of using a varied number of lagged observations as input features for LSTM models.
- The impact of using a varied number of lagged observations and matching numbers of neurons for LSTM models.

Let's get started.



How to Use Features in LSTM Networks for Time Series Forecasting
Photo by Tom Hodgkinson, some rights reserved.

Tutorial Overview

This tutorial is divided into 4 parts. They are:

1. Shampoo Sales Dataset

Get Your Start in Machine Learning

2. Experimental Test Harness
3. Experiments with Timesteps
4. Experiments with Timesteps and Neurons

Environment

This tutorial assumes you have a Python SciPy environment installed. You can use either Python 2 or 3 with this example.

This tutorial assumes you have Keras v2.0 or higher installed with either the TensorFlow or Theano backend.

This tutorial also assumes you have scikit-learn, Pandas, NumPy, and Matplotlib installed.

If you need help setting up your Python environment, see this post:

- How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda

Shampoo Sales Dataset

This dataset describes the monthly number of sales of shampoo over a 3-year period.

The units are a sales count and there are 36 observations. The original dataset is credited to Makrid

Get Your Start in Machine Learning

You can download and learn more about the dataset here.

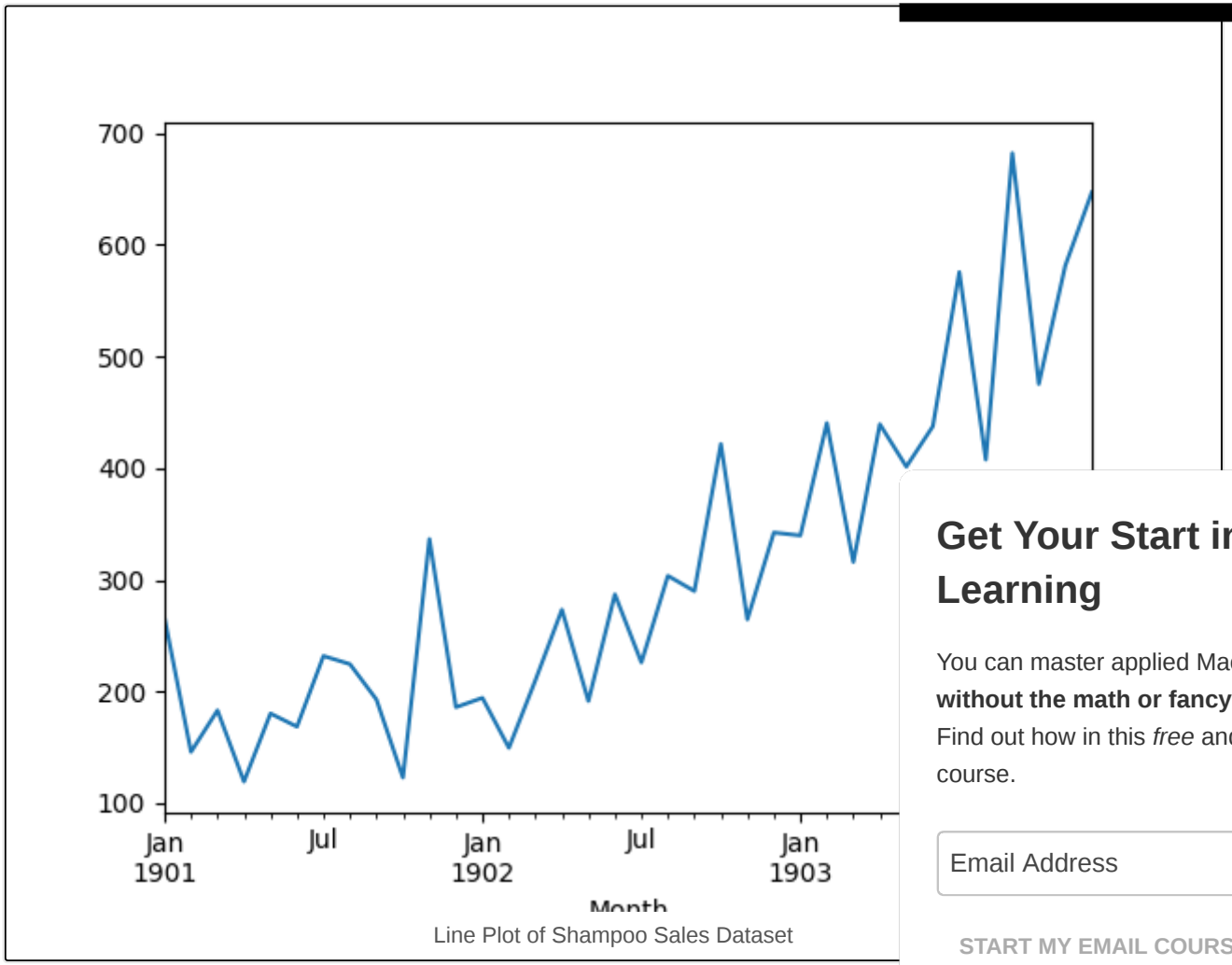
The example below loads and creates a plot of the loaded dataset.

```
# load and plot dataset
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
# load dataset
def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
# summarize first few rows
print(series.head())
# line plot
series.plot()
pyplot.show()
```

Running the example loads the dataset as a Pandas Series and prints the first 5 rows.

```
Month
1901-01-01 266.0
1901-02-01 145.9
1901-03-01 183.1
1901-04-01 119.3
1901-05-01 180.3
Name: Sales, dtype: float64
```

A line plot of the series is then created showing a clear increasing trend.



Line Plot of Shampoo Sales Dataset

Next, we will take a look at the LSTM configuration and test harness used in the experiment.

Experimental Test Harness

This section describes the test harness used in this tutorial.

Data Split

Get Your Start in Machine Learning

We will split the Shampoo Sales dataset into two parts: a training and a test set.

The first two years of data will be taken for the training dataset and the remaining one year of data will be used for the test set.

Models will be developed using the training dataset and will make predictions on the test dataset.

The persistence forecast (naive forecast) on the test dataset achieves an error of 136.761 monthly shampoo sales. This provides a lower acceptable bound of performance on the test set.

Model Evaluation

A rolling-forecast scenario will be used, also called walk-forward model validation.

Each time step of the test dataset will be walked one at a time. A model will be used to make a forec ue from the test set will be taken and made available to the model for the forecast on the next time step

This mimics a real-world scenario where new Shampoo Sales observations would be available each g month.

This will be simulated by the structure of the train and test datasets.

All forecasts on the test dataset will be collected and an error score calculated to summarize the skil E) will be used as it punishes large errors and results in a score that is in the same units as the forecas

Data Preparation

Before we can fit an LSTM model to the dataset, we must transform the data.

The following three data transforms are performed on the dataset prior to fitting a model and making a forecast.

1. **Transform the time series data so that it is stationary**. Specifically, a lag=1 differencing to remove the increasing trend in the data.
2. **Transform the time series into a supervised learning problem**. Specifically, the organization of data into input and output patterns where the observation at the previous time step is used as an input to forecast the observation at the current time step

Get Your Start in Machine Learning

3. **Transform the observations to have a specific scale**. Specifically, to rescale the data to values between -1 and 1 to meet the default hyperbolic tangent activation function of the LSTM model.

These transforms are inverted on forecasts to return them into their original scale before calculating and error score.

LSTM Model

We will use a base stateful LSTM model with 1 neuron fit for 500 epochs.

A batch size of 1 is required as we will be using walk-forward validation and making one-step forecasts for each of the final 12 months of test data.

A batch size of 1 means that the model will be fit using online training (as opposed to batch training or mini-batch training). As a result, it is expected that the model fit will have some variance.

Ideally, more training epochs would be used (such as 1000 or 1500), but this was truncated to 500 to

The model will be fit using the efficient ADAM optimization algorithm and the mean squared error los

Experimental Runs

Each experimental scenario will be run 10 times.

The reason for this is that the random initial conditions for an LSTM network can result in very differe ed.

Let's dive into the experiments.

Experiments with Features

We will perform 5 experiments; each will use a different number of lag observations as features from 1 to 5.

A representation with a 1 input feature would be the default representation when using a stateful LSTM. Using 2 to 5 features is contrived. The hope would be that the additional context from the lagged observations may improve performance of the predictive model.

The univariate time series is converted to a supervised learning problem before training the model. T ber of input variables (*X*) used to predict the next observation (*y*). As such, for each feature used in the r

Get Your Start in Machine Learning

from the beginning of the dataset. This is because there are no prior observations to use as features for the first values in the dataset.

The complete code listing for testing 1 input feature is provided below.

The features parameter in the *run()* function is varied from 1 to 5 for each of the 5 experiments. In addition, the results are saved to file at the end of the experiment and this filename must also be changed for each different experimental run, e.g. *experiment_features_1.csv*, *experiment_features_2.csv*, etc.

```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
import matplotlib
import numpy
from numpy import concatenate

# date-time parsing function for loading the dataset
def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]
```

```
39
40 # scale train and test data to [-1, 1]
41 def scale(train, test):
42     # fit scaler
43     scaler = MinMaxScaler(feature_range=(-1, 1))
44     scaler = scaler.fit(train)
45     # transform train
46     train = train.reshape(train.shape[0], train.shape[1])
47     train_scaled = scaler.transform(train)
48     # transform test
49     test = test.reshape(test.shape[0], test.shape[1])
50     test_scaled = scaler.transform(test)
51     return scaler, train_scaled, test_scaled
52
53 # inverse scaling for a forecasted value
54 def invert_scale(scaler, X, yhat):
55     new_row = [x for x in X] + [yhat]
56     array = numpy.array(new_row)
57     array = array.reshape(1, len(array))
58     inverted = scaler.inverse_transform(array)
59     return inverted[0, -1]
60
61 # fit an LSTM network to training data
62 def fit_lstm(train, batch_size, nb_epoch, neurons):
63     X, y = train[:, 0:-1], train[:, -1]
64     X = X.reshape(X.shape[0], 1, X.shape[1])
65     model = Sequential()
66     model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), sta
67     model.add(Dense(1))
68     model.compile(loss='mean_squared_error', optimizer='adam')
69     for i in range(nb_epoch):
70         model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
71         model.reset_states()
72     return model
73
74 # make a one-step forecast
75 def forecast_lstm(model, batch_size, X):
76     X = X.reshape(1, 1, len(X))
77     yhat = model.predict(X, batch_size=batch_size)
78     return yhat[0,0]
79
80 # run a repeated experiment
81 def experiment(repeats, series, features):
82     # transform data to be stationary
83     raw_values = series.values
84     diff_values = difference(raw_values, 1)
85     # transform data to be supervised learning
```

```
    supervised = timeseries_to_supervised(diff_values, features)
    supervised_values = supervised.values[features:,:]
    # split data into train and test-sets
    train, test = supervised_values[0:-12, :], supervised_values[-12:, :]
    # transform the scale of the data
    scaler, train_scaled, test_scaled = scale(train, test)
    # run experiment
    error_scores = list()
    for r in range(repeats):
        # fit the base model
        lstm_model = fit_lstm(train_scaled, 1, 500, 1)
        # forecast test dataset
        predictions = list()
        for i in range(len(test_scaled)):
            # predict
            X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
            yhat = forecast_lstm(lstm_model, 1, X)
            # invert scaling
            yhat = invert_scale(scaler, X, yhat)
            # invert differencing
            yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
            # store forecast
            predictions.append(yhat)
        # report performance
        rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
        print('%d) Test RMSE: %.3f' % (r+1, rmse))
        error_scores.append(rmse)
    return error_scores

# execute the experiment
def run():
    # load dataset
    series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, sque
    # experiment
    repeats = 10
    results = DataFrame()
    # run experiment
    features = 1
    results['results'] = experiment(repeats, series, features)
    # summarize results
    print(results.describe())
    # save results
    results.to_csv('experiment_features_1.csv', index=False)

 # entry point
run()
```

Run the 5 different experiments for the 5 different numbers of features.

You can run them in parallel if you have sufficient memory and CPU resources. GPU resources are not required for these experiments and runs should be complete in minutes to tens of minutes.

After running the experiments, you should have 5 files containing the results, as follows:

- *experiment_features_1.csv*
- *experiment_features_2.csv*
- *experiment_features_3.csv*
- *experiment_features_4.csv*
- *experiment_features_5.csv*

We can write some code to load and summarize these results.

Specifically, it is useful to review both descriptive statistics from each run and compare the results fo

Code to summarize the results is listed below.

```
from pandas import DataFrame
from pandas import read_csv
from matplotlib import pyplot
# load results into a dataframe
filenames = ['experiment_features_1.csv', 'experiment_features_2.csv',
    'experiment_features_3.csv','experiment_features_4.csv','experiment_features_5.csv']
results = DataFrame()
for name in filenames:
    results[name[11:-4]] = read_csv(name, header=0)
# describe all results
print(results.describe())
# box and whisker plot
results.boxplot()
pyplot.show()
```

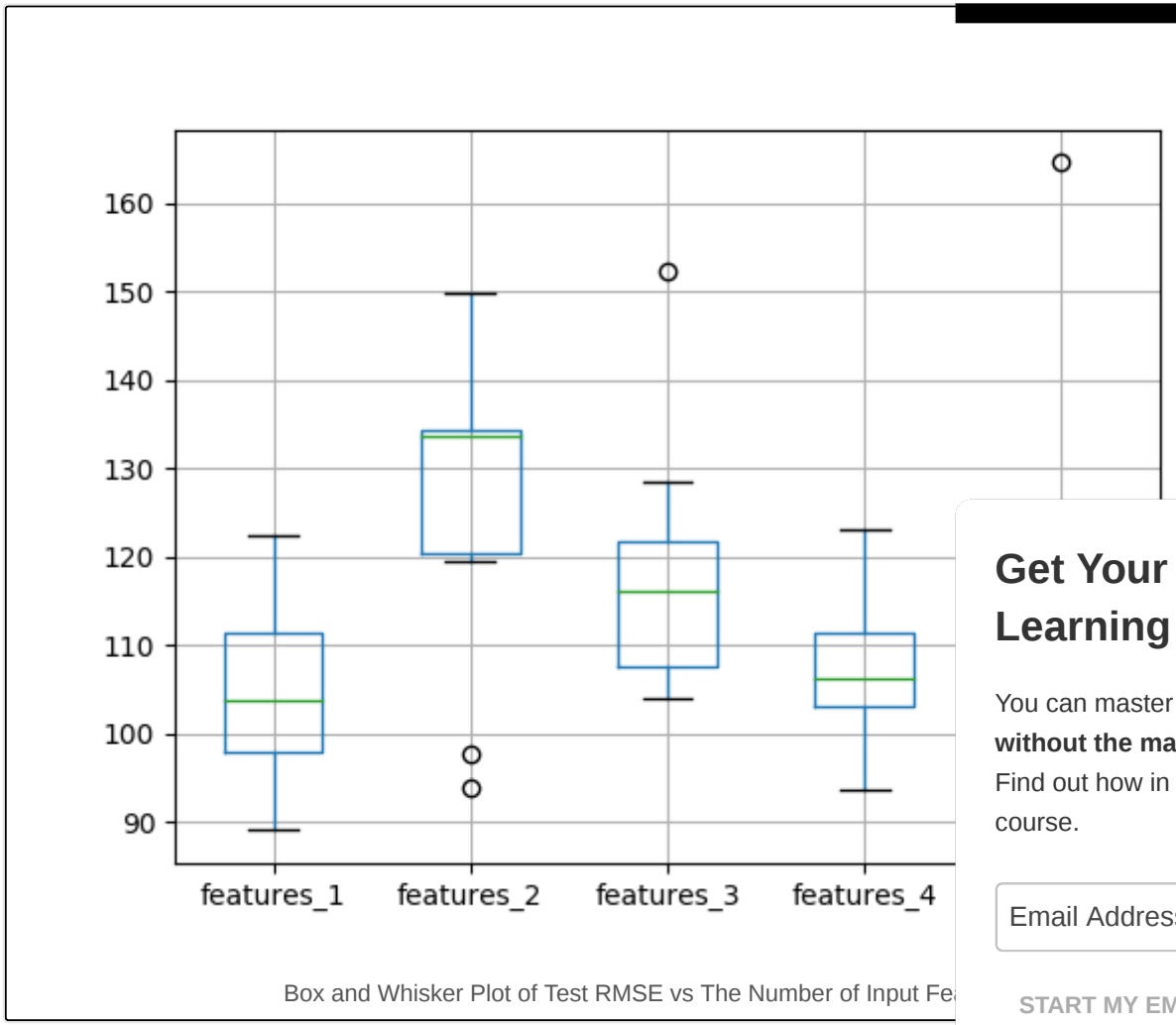
Running the code first prints descriptive statistics for each set of results.

We can see from the average performance alone that the default of using a single feature resulted in the best performance. This is also shown when reviewing the median test RMSE (50th percentile).

```
       features_1  features_2  features_3  features_4  features_5
count   10.000000   10.000000   10.000000   10.000000   10.000000
mean   104.588249  126.597800  118.268251  107.694178  116.414887
std     10.205840   18.639757   14.359983    8.683271   18.806281
min     89.046814   93.857991  103.900339   93.702085   98.245871
25%     97.850827  120.296634  107.664087  102.992045  105.660897
50%    103.713285  133.582095  116.123790  106.116922  112.950460
75%    111.441655  134.362198  121.794533  111.498255  117.926664
max    122.341580  149.807155  152.412861  123.006088  164.598542
```

A box and whisker plot comparing the distributions of results is also created.

The plot tells the same story as the descriptive statistics. The test RMSE seems to leap up with 2 features and trend upward as the number of features is increased.



Box and Whisker Plot of Test RMSE vs The Number of Input Fe

The expectation of decreased error with the increase of features was not observed, at least with the dataset and LSTM configuration used.

This raises the question as to whether the capacity of the network is a limiting factor. We will look at this in the next section.

Experiments with Features and Neurons

The number of neurons (also called units) in the LSTM network defines its learning capacity.

It is possible that in the previous experiments the use of one neuron limited the learning capacity of the network such that it was not capable of making effective use of the lagged observations as features.

We can repeat the above experiments and increase the number of neurons in the LSTM with the increase in features and see if it results in an increase in performance.

This can be achieved by changing the line in the experiment function from:

```
1 lstm_model = fit_lstm(train_scaled, 1, 500, 1, features)
```

to

```
1 lstm_model = fit_lstm(train_scaled, 1, 500, features, features)
```

In addition, we can keep the results written to file separate from the results from the first experiment example, changing:

```
1 results.to_csv('experiment_features_1.csv', index=False)
```

to

```
1 results.to_csv('experiment_features_1_neurons.csv', index=False)
```

Repeat the same 5 experiments with these changes.

After running these experiments, you should have 5 result files.

- *experiment_features_1_neurons.csv*
- *experiment_features_2_neurons.csv*
- *experiment_features_3_neurons.csv*
- *experiment_features_4_neurons.csv*
- *experiment_features_5_neurons.csv*

As in the previous experiment, we can load the results, calculate descriptive statistics, and create a box and whisker plot. The complete code listing is below.

```
1 from pandas import DataFrame
```

```
from pandas import read_csv
from matplotlib import pyplot
# load results into a dataframe
filenames = ['experiment_features_1_neurons.csv', 'experiment_features_2_neurons.csv',
    'experiment_features_3_neurons.csv','experiment_features_4_neurons.csv','experiment_features_5_neurons.csv']
results = DataFrame()
for name in filenames:
    results[name[11:-12]] = read_csv(name, header=0)
# describe all results
print(results.describe())
# box and whisker plot
results.boxplot()
pyplot.show()
```

Running the code first prints descriptive statistics from each of the 5 experiments.

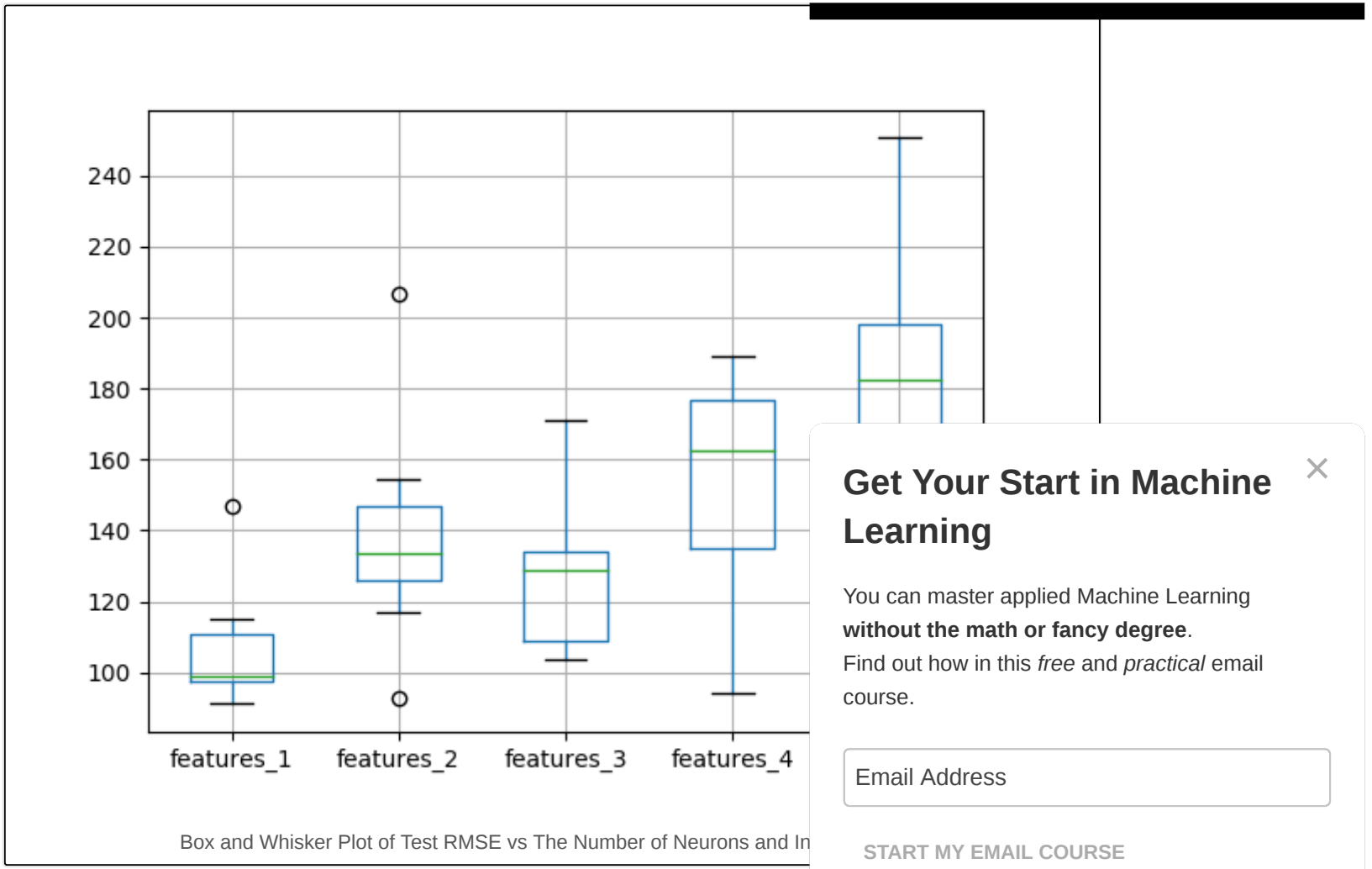
The results tell a different story to the first set of experiments with a one neuron LSTM. The average
neurons and the number of features is set to one, then error increases as neurons and features are

```
       features_1  features_2  features_3  features_4  features_5
count   10.000000   10.000000   10.000000   10.000000   10.000000
mean   106.219189  138.411111  127.687128  154.281694  175.951500
std     16.100488   29.700981   21.411766   30.526294   44.839217
min     91.073598   92.641030  103.503546   94.063639  117.017109
25%     97.263723  125.748973  108.972440  134.805621  142.146601
50%     99.036766  133.639168  128.627349  162.295657  182.406707
75%    110.625302  146.896608  134.012859  176.969980  197.913894
max    146.638148  206.760081  170.899267  188.911768  250.685187
```

A box and whisker plot is created to compare the distributions.

The trend in spread and median performance almost shows a linear increase in test RMSE as the nu

The linear trend may suggest that the increase network capacity is not given sufficient time to fit the data. Perhaps an increase in the number of epochs would be required as well.



Box and Whisker Plot of Test RMSE vs The Number of Neurons and In

Experiments with Features and Neurons More Epochs

In this section, we repeat the above experiment to increase the number of neurons with the number of features but double the number of training epochs from 500 to 1000.

This can be achieved by changing the line in the experiment function from:

```
1 lstm_model = fit_lstm(train_scaled, 1, 500, features, features)
```

to

```
lstm_model = fit_lstm(train_scaled, 1, 1000, features, features)
```

In addition, we can keep the results written to file separate from the results from the previous experiment by adding a "*1000*" suffix to the filenames, for example, changing:

```
results.to_csv('experiment_features_1_neurons.csv', index=False)
```

to

```
results.to_csv('experiment_features_1_neurons1000.csv', index=False)
```

Repeat the same 5 experiments with these changes.

After running these experiments, you should have 5 result files.

- *experiment_features_1_neurons1000.csv*
- *experiment_features_2_neurons1000.csv*
- *experiment_features_3_neurons1000.csv*
- *experiment_features_4_neurons1000.csv*
- *experiment_features_5_neurons1000.csv*

As in the previous experiment, we can load the results, calculate descriptive statistics, and create a l
below.

```
from pandas import DataFrame
from pandas import read_csv
from matplotlib import pyplot
# load results into a dataframe
filenames = ['experiment_features_1_neurons1000.csv', 'experiment_features_2_neurons1000.csv',
    'experiment_features_3_neurons1000.csv','experiment_features_4_neurons1000.csv','experiment_features_5_neurons1000.csv']
results = DataFrame()
for name in filenames:
    results[name[11:-16]] = read_csv(name, header=0)
# describe all results
print(results.describe())
# box and whisker plot
results.boxplot()
pyplot.show()
```

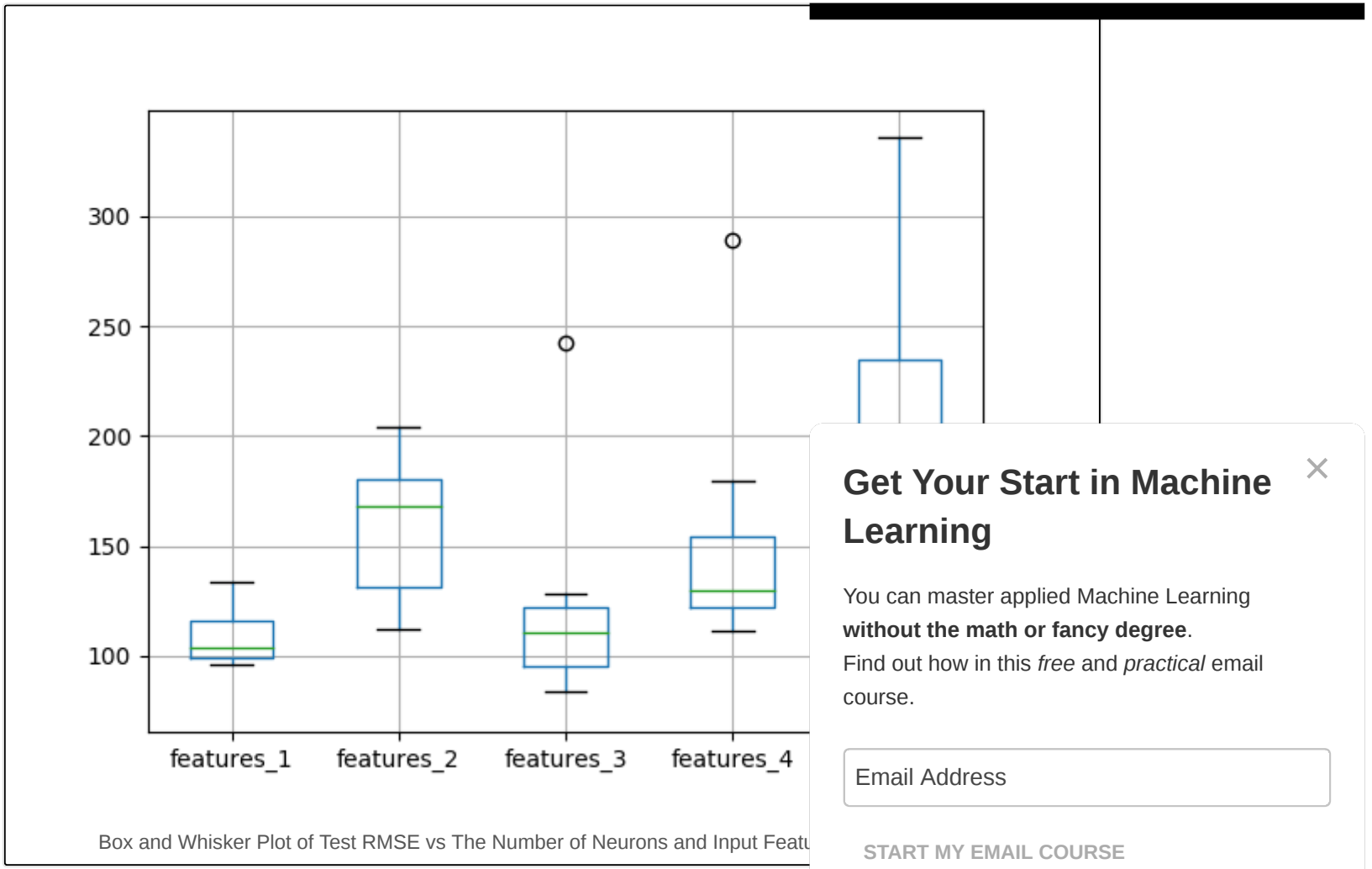
Running the code first prints descriptive statistics from each of the 5 experiments.

The results tell a very similar story to the previous experiment with half the number of training epochs. On average, a model with 1 input feature and 1 neuron outperformed the other configurations.

```
       features_1  features_2  features_3  features_4  features_5
count   10.000000   10.000000   10.000000   10.000000   10.000000
mean   109.262674  158.295172  120.340623  149.741882  201.992209
std     13.850525   32.288109   45.219564   53.121113   82.986691
min     95.927393  111.936394   83.983325  111.017837   78.040385
25%     98.754253  130.875314   95.198556  122.287208  148.840499
50%    103.990988  167.915523  110.256517  129.552084  188.498836
75%    116.055435  180.679252  122.158321  154.283676  234.519359
max    133.270446  204.260072  242.186747  288.907803  335.595974
```

A box and whisker plot was also created to compare the distributions. In the plot, we see the same t

At least on this problem and with the chosen LSTM configuration, we do not see any clear benefit in



Box and Whisker Plot of Test RMSE vs The Number of Neurons and Input Featu

Extensions

This section lists some areas for further investigation that you may consider exploring.

- **Diagnostic Run Plots**. It may be helpful to review plots of train and test RMSE over epochs for multiple runs for a given experiment. This might help tease out whether overfitting or underfitting is taking place, and in turn, methods to address it.
- **Increase Repeats**. Using 10 repeats results in a relatively small population of test RMSE results. It is possible that increasing repeats to 30 or 100 (or even higher) may result in a more stable outcome.

Did you explore any of these extensions?
Share your findings in the comments below; I'd love to hear what you found.

Summary

In this tutorial, you discovered how to investigate using lagged observations as input features in an LSTM network.

Specifically, you learned:

- How to develop a robust test harness for experimenting with input representation with LSTMs.
- How to use lagged observations as input features for time series forecasting with LSTMs.
- How to increase the learning capacity of the network with the increase of input features.

You discovered that the expectation that "*the use of lagged observations as input features improves* [redacted] e chosen problem and LSTM configuration.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer them.

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional develo to helping developers get started and get good at applied machine learning. Learn more.

View all posts by Jason Brownlee →

‹ How to Use Timesteps in LSTM Networks for Time Series Forecasting

Stateful and Stateless LSTM for Time Series Forecasting with Python ›

Get Your Start in Machine Learning

14 Responses to *How to Use Features in LSTM Networks for Time Series Forecasting*

Zach May 5, 2017 at 7:40 am #

REPLY

When using multiple features, is a stateful model the same as a stateless one with timesteps? Can you use timesteps in a stateful model with multiple features?

Jason Brownlee May 5, 2017 at 11:23 am #

It really depends on how you structure your data.

Stateless with all data in a batch or all time steps in a sample within a batch is the same as stateful.

I hope that makes things clearer. Also see this post:
http://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/

Jason Ho May 5, 2017 at 1:30 pm #

Hi,Dr.Jason Brownlee.I am confused that difference between time_steps and features in a LSTI

Jason Brownlee May 6, 2017 at 7:32 am #

REPLY

Great question.

Time steps are the past observations that the network will learn from (e.g. backpropagation through time). I have a few posts on BPTT coming out on the blog soon.

Features are different measurements taken at a given time step (e.g. pressure and temperature).

Does that help?

Jason Ho May 11, 2017 at 12:34 pm #

REPLY

Thanks for Dr.Jason Brownlee's reply.

So we can take time steps and features just 2 type of LSTM's input?

And for multi-dimension input , just one neuron in input layer , does it really work? And does the number of input layer's neuron of LSTM need to be same as the input's dimension(time steps' dimension +features' dimension) like the traditional BP Neural Network?

Jason Brownlee May 12, 2017 at 7:33 am #

No, the number of memory cells in the input layer does not have to match the numb

Jason Ho May 11, 2017 at 12:52 pm #

Compared to traditional method ,like AR, MA and ARIMA , What does the LSTM's advantages i

Jason Brownlee May 12, 2017 at 7:35 am #

Great question.

The promise of LSTMs (I have a scheduled post on this) is that they can learn the temporal dependence. That you just feed in sequences and do not have to specify the required window/lag (e.g. outcome from an ACF/PACF analysis).

Jason Ho May 12, 2017 at 1:59 pm #

REPLY

And except that the LTSM do not have to specify the required window/lag ,is the LSTM's prediction accuracy better than ARIMA or the LSTM can do better than other tradition method thanks to its special model struction and deep layers?

Jason Brownlee May 13, 2017 at 6:11 am #

REPLY

It may be, that is problem specific (e.g. no free lunch).

Siddharth May 31, 2017 at 10:51 pm #

Thanks Jason, yet again a very well structured blog post. I am learning a lot from this set of pos
I have two questions for you:

1) What affect does adding more layers (going deeper) to the network have? More specifically, how do y
when to make the network deeper (more layers)?

2) This might be a broad question, but specifically for time series forecasting, what "size" of data do you
for a problem, is it possible to transfer learn from another LSTM?

Looking forward to your response!

Jason Brownlee June 2, 2017 at 12:50 pm #

REPLY

More layers mean more hierarchical representational capacity, but slower to train and perhaps more likely to overfit. Try different configurations on your problem.

As much data as you can get.

You may be able to do transfer learning, I have not read anything about it for time series with LSTMs

Ukesh June 14, 2017 at 6:59 am #

REPLY

Hi Jason,
Thank you for your wonderful post.

I am new to deep learning and LSTM. I have a very simple question. I have taken a sample of demands for 50 time steps and I am trying to forecast the demand value for the next 10 time steps using the sample to train the model.

But unfortunately, the closest I came is splitting the sample demands into 67 training % and 33 testing % and my forecast is only forecasting for the 33%. Can anybody help me with this issue?

I have borrowed your code and put it in the in github explaining this issue.

Thank you in advance.

https://github.com/ukeshchawal/hello-world/blob/master/trial.ipynb

Jason Brownlee June 14, 2017 at 8:53 am #

See this post for ways of evaluating time series forecasting methods:
http://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

Leave a Reply

Get Your Start in Machine Learning

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

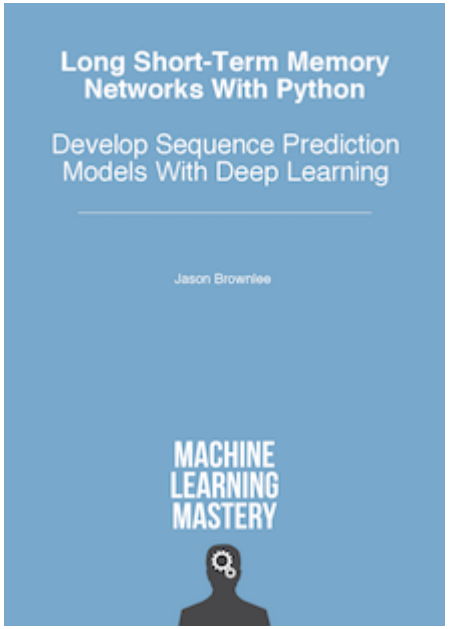
Read More

Deep Learning for Sequence Prediction

Cut through the math and research papers.
Discover 4 Models, 6 Architectures, and 14 Tutorials.

Get Your Start in Machine Learning

Get Started With LSTMs in Python Today!



POPULAR


Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
JULY 21, 2016


Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016


Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016


Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

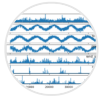
START MY EMAIL COURSE

Get Your Start in Machine Learning

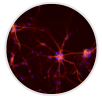
Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017

Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016



Multivariate Time Series Forecasting with LSTMs in Keras
AUGUST 14, 2017



How to Implement the Backpropagation Algorithm From Scratch In Python
NOVEMBER 7, 2016

Get Your Start in Machine Learning