How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting

by **Jason Brownlee** on April 12, 2017 in **Deep Learning for Time Series**

Last Updated on August 5, 2019

Configuring neural networks is difficult because there is no good theory on how to do it.

You must be systematic and explore different configurations both from a dynamical and an objective results point of a view to try to understand what is going on for a given predictive modeling problem.

In this tutorial, you will discover how you can explore how to configure an LSTM network on a time series forecasting problem.

After completing this tutorial, you will know:

- How to tune and interpret the results of the number of training epochs.
- How to tune and interpret the results of the size of training batches.
- How to tune and interpret the results of the number of neurons.

Discover how to build models for multivariate and multi-step time series forecasting with LSTMs and more in my new book, with 25 step-by-step tutorials and full source code.

Let's get started.

- **Updated Apr/2019**: Updated the link to dataset.

How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting
Photo by David Saddler, some rights reserved.

Tutorial Overview

This tutorial is broken down into 6 parts; they are:

1. Shampoo Sales Dataset
2. Experimental Test Harness
3. Tuning the Number of Epochs
4. Tuning the Batch Size
5. Tuning the Number of Neurons
6. Summary of Results

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.

The 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 Makridakis, Wheelwright, and Hyndman (1998).

- Download the dataset.

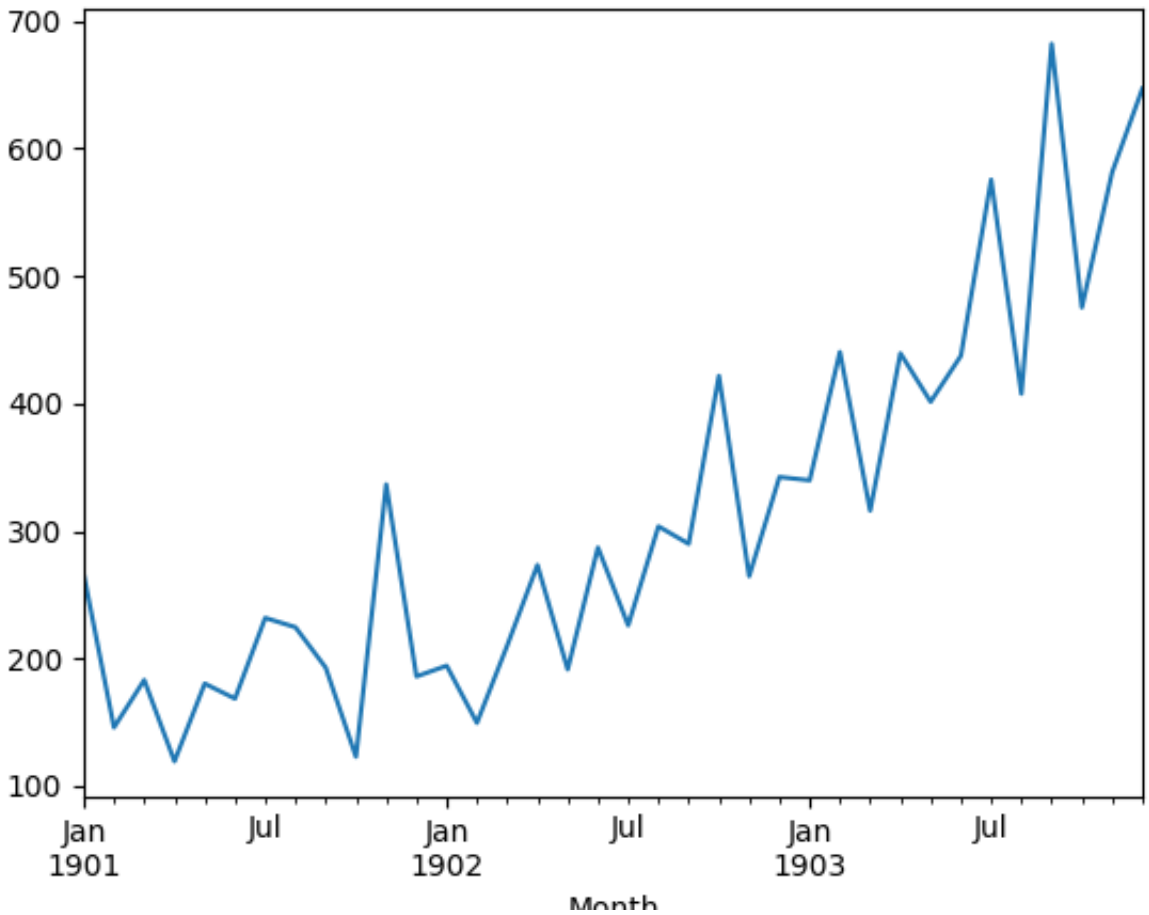
The example below loads and creates a plot of the loaded dataset.

```
# 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,
# 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

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 forecast for the time step, then the actual expected value 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 month and used in the forecasting of the following month.

This will be simulated by the structure of the train and test datasets. We will make all of the forecasts in a one-shot method.

All forecasts on the test dataset will be collected and an error score calculated to summarize the skill of the model. The root mean squared error (RMSE) will be used as it punishes large errors and results in a score that is in the same units as the forecast data, namely monthly shampoo sales.

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 time step
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.

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 different results each time a given configuration is trained.

A diagnostic approach will be used to investigate model configurations. This is where line plots of model skill over time (training iterations called epochs) will be created and studied for insight into how a given configuration performs and how it may be adjusted to elicit better performance.

The model will be evaluated on both the train and the test datasets at the end of each epoch and the RMSE scores saved.

The train and test RMSE scores at the end of each scenario are printed to give an indication of progress.

The series of train and test RMSE scores are plotted at the end of a run as a line plot. Train scores are colored blue and test scores are colored orange.

Let's dive into the results.

Tuning the Number of Epochs

The first LSTM parameter we will look at tuning is the number of training epochs.

The model will use a batch size of 4, and a single neuron. We will explore the effect of training this configuration for different numbers of training epochs.

Diagnostic of 500 Epochs

The complete code listing for this diagnostic is listed below.

The code is reasonably well commented and should be easy to follow. This code will be the basis for all future experiments in this tutorial and only the changes made in each subsequent experiment will be listed.

```
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
# be able to save images on server
matplotlib.use('Agg')
from matplotlib import pyplot
import numpy

# 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)
    df = df.drop(0)
    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)

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
```

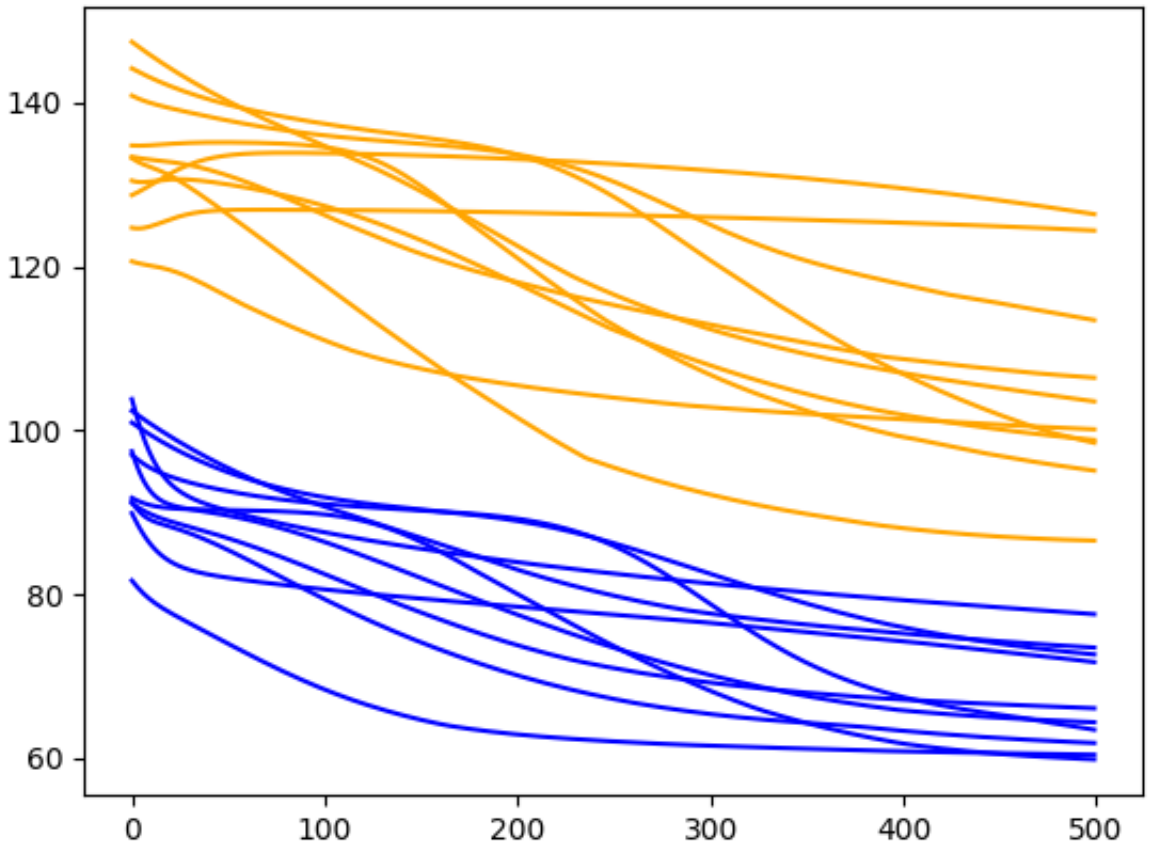
```
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, yhat):
    new_row = [x for x in X] + [yhat]
    array = numpy.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# evaluate the model on a dataset, returns RMSE in transformed units
def evaluate(model, raw_data, scaled_dataset, scaler, offset, batch_size):
    # separate
    X, y = scaled_dataset[:,0:-1], scaled_dataset[:,-1]
    # reshape
    reshaped = X.reshape(len(X), 1, 1)
    # forecast dataset
    output = model.predict(reshaped, batch_size=batch_size)
    # invert data transforms on forecast
    predictions = list()
    for i in range(len(output)):
        yhat = output[i,0]
        # invert scaling
        yhat = invert_scale(scaler, X[i], yhat)
        # invert differencing
        yhat = yhat + raw_data[i]
        # store forecast
        predictions.append(yhat)
    # report performance
    rmse = sqrt(mean_squared_error(raw_data[1:], predictions))
    return rmse

# fit an LSTM network to training data
def fit_lstm(train, test, raw, scaler, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    # prepare model
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), statefu
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    # fit model
    train_rmse, test_rmse = list(), list()
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
        # evaluate model on train data
        raw_train = raw[-(len(train)+len(test)+1):-len(test)]
        train_rmse.append(evaluate(model, raw_train, train, scaler, 0, batch_size))
        model.reset_states()
        # evaluate model on test data
        raw_test = raw[-(len(test)+1):]
        test_rmse.append(evaluate(model, raw_test, test, scaler, 0, batch_size))
        model.reset_states()
```

```
    history = DataFrame()
    history['train'], history['test'] = train_rmse, test_rmse
    return history

# run diagnostic experiments
def run():
    # load dataset
    series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=
    # transform data to be stationary
    raw_values = series.values
    diff_values = difference(raw_values, 1)
    # transform data to be supervised learning
    supervised = timeseries_to_supervised(diff_values, 1)
    supervised_values = supervised.values
    # 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)
    # fit and evaluate model
    train_trimmed = train_scaled[2:, :]
    # config
    repeats = 10
    n_batch = 4
    n_epochs = 500
    n_neurons = 1
    # run diagnostic tests
    for i in range(repeats):
        history = fit_lstm(train_trimmed, test_scaled, raw_values, scaler, n_batch, n_epoch
        pyplot.plot(history['train'], color='blue')
        pyplot.plot(history['test'], color='orange')
        print('%d) TrainRMSE=%f, TestRMSE=%f' % (i, history['train'].iloc[-1], history['tes
    pyplot.savefig('epochs_diagnostic.png')

# entry point
run()
```

Running the experiment prints the RMSE for the train and the test sets at the end of each of the 10 experimental runs.

```
0) TrainRMSE=63.495594, TestRMSE=113.472643
1) TrainRMSE=60.446307, TestRMSE=100.147470
2) TrainRMSE=59.879681, TestRMSE=95.112331
3) TrainRMSE=66.115269, TestRMSE=106.444401
4) TrainRMSE=61.878702, TestRMSE=86.572920
5) TrainRMSE=73.519382, TestRMSE=103.551694
6) TrainRMSE=64.407033, TestRMSE=98.849227
7) TrainRMSE=72.684834, TestRMSE=98.499976
8) TrainRMSE=77.593773, TestRMSE=124.404747
9) TrainRMSE=71.749335, TestRMSE=126.396615
```

A line plot of the series of RMSE scores on the train and test sets after each training epoch is also created.



Diagnostic Results with 500 Epochs

The results clearly show a downward trend in RMSE over the training epochs for almost all of the experimental runs.

This is a good sign, as it shows the model is learning the problem and has some predictive skill. In fact, all of the final test scores are below the error of a simple persistence model (naive forecast) that achieves an RMSE of 136.761 on this problem.

The results suggest that more training epochs will result in a more skillful model.

Let's try doubling the number of epochs from 500 to 1000.

Diagnostic of 1000 Epochs

In this section, we use the same experimental setup and fit the model over 1000 training epochs.

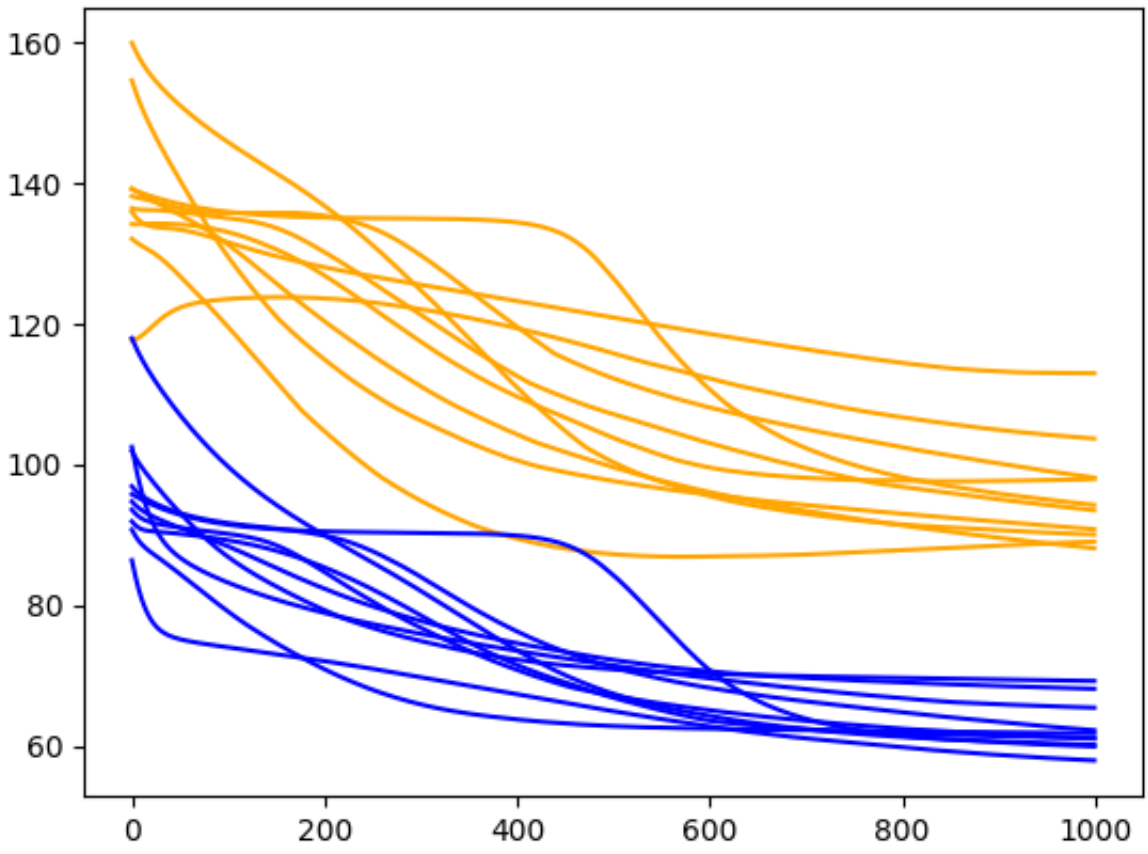
Specifically, the *n_epochs* parameter is set to *1000* in the *run()* function.

```
n_epochs = 1000
```

Running the example prints the RMSE for the train and test sets from the final epoch.

```
0) TrainRMSE=69.242394, TestRMSE=90.832025
1) TrainRMSE=65.445810, TestRMSE=113.013681
2) TrainRMSE=57.949335, TestRMSE=103.727228
3) TrainRMSE=61.808586, TestRMSE=89.071392
4) TrainRMSE=68.127167, TestRMSE=88.122807
5) TrainRMSE=61.030678, TestRMSE=93.526607
6) TrainRMSE=61.144466, TestRMSE=97.963895
7) TrainRMSE=59.922150, TestRMSE=94.291120
8) TrainRMSE=60.170052, TestRMSE=90.076229
9) TrainRMSE=62.232470, TestRMSE=98.174839
```

A line plot of the test and train RMSE scores each epoch is also created.



Diagnostic Results with 1000 Epochs

We can see that the downward trend of model error does continue and appears to slow.

The lines for the train and test cases become more horizontal, but still generally show a downward trend, although at a lower rate of change. Some examples of test error show a possible inflection point around 600 epochs and may show a rising trend.

It is worth extending the epochs further. We are interested in the average performance continuing to improve on the test set and this may continue.

Let's try doubling the number of epochs from 1000 to 2000.

Diagnostic of 2000 Epochs

In this section, we use the same experimental setup and fit the model over 2000 training epochs.

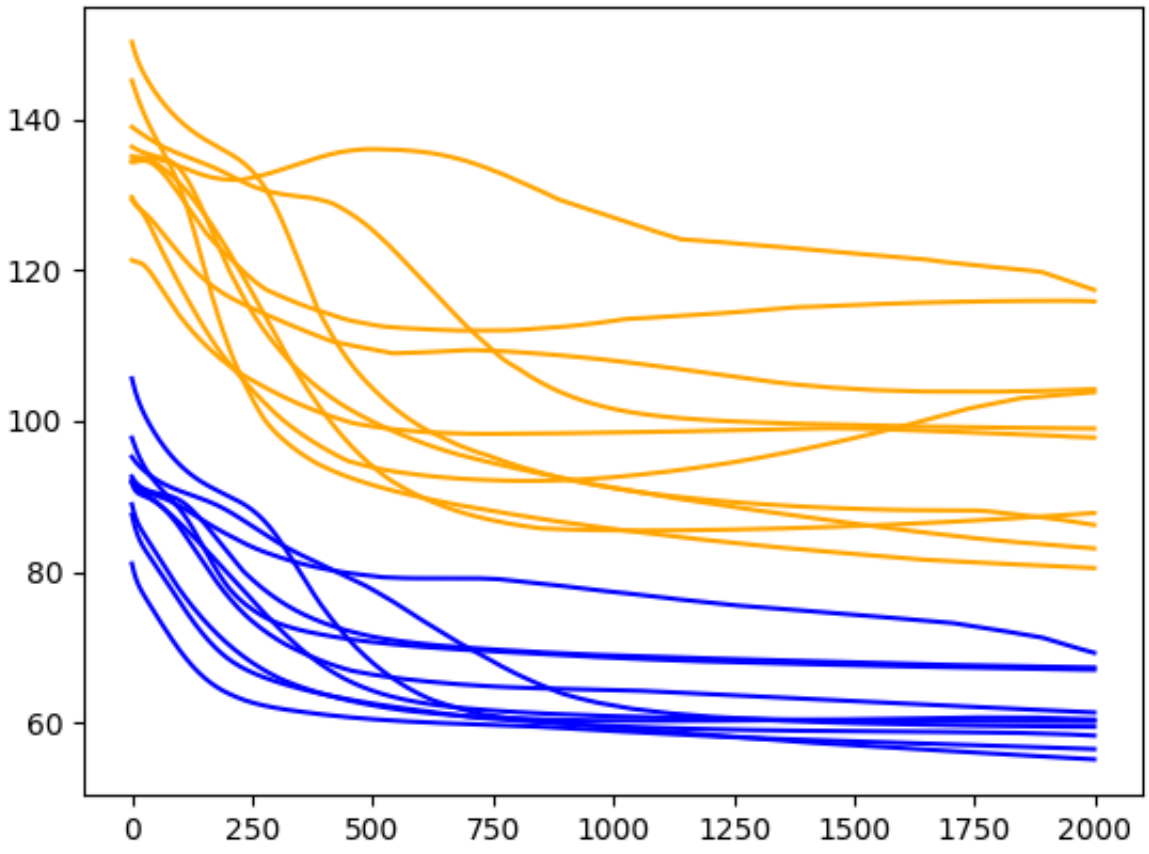
Specifically, the *n_epochs* parameter is set to 2000 in the *run()* function.

```
n_epochs = 2000
```

Running the example prints the RMSE for the train and test sets from the final epoch.

```
0) TrainRMSE=67.292970, TestRMSE=83.096856
1) TrainRMSE=55.098951, TestRMSE=104.211509
2) TrainRMSE=69.237206, TestRMSE=117.392007
3) TrainRMSE=61.319941, TestRMSE=115.868142
4) TrainRMSE=60.147575, TestRMSE=87.793270
5) TrainRMSE=59.424241, TestRMSE=99.000790
6) TrainRMSE=66.990082, TestRMSE=80.490660
7) TrainRMSE=56.467012, TestRMSE=97.799062
8) TrainRMSE=60.386380, TestRMSE=103.810569
9) TrainRMSE=58.250862, TestRMSE=86.212094
```

A line plot of the test and train RMSE scores each epoch is also created.



Diagnostic Results with 2000 Epochs

As one might have guessed, the downward trend in error continues over the additional 1000 epochs on both the train and test datasets.

Of note, about half of the cases continue to decrease in error all the way to the end of the run, whereas the rest show signs of an increasing trend.

The increasing trend is a sign of overfitting. This is when the model overfits the training dataset at the cost of worse performance on the test dataset. It is exemplified by continued improvements on the training dataset and improvements followed by an inflection point and worsting skill in the test dataset. A little less than half of the runs show the beginnings of this type of pattern on the test dataset.

Nevertheless, the final epoch results on the test dataset are very good. If there is a chance we can see further gains by even longer training, we must explore it.

Let's try doubling the number of epochs from 2000 to 4000.

Diagnostic of 4000 Epochs

In this section, we use the same experimental setup and fit the model over 4000 training epochs.

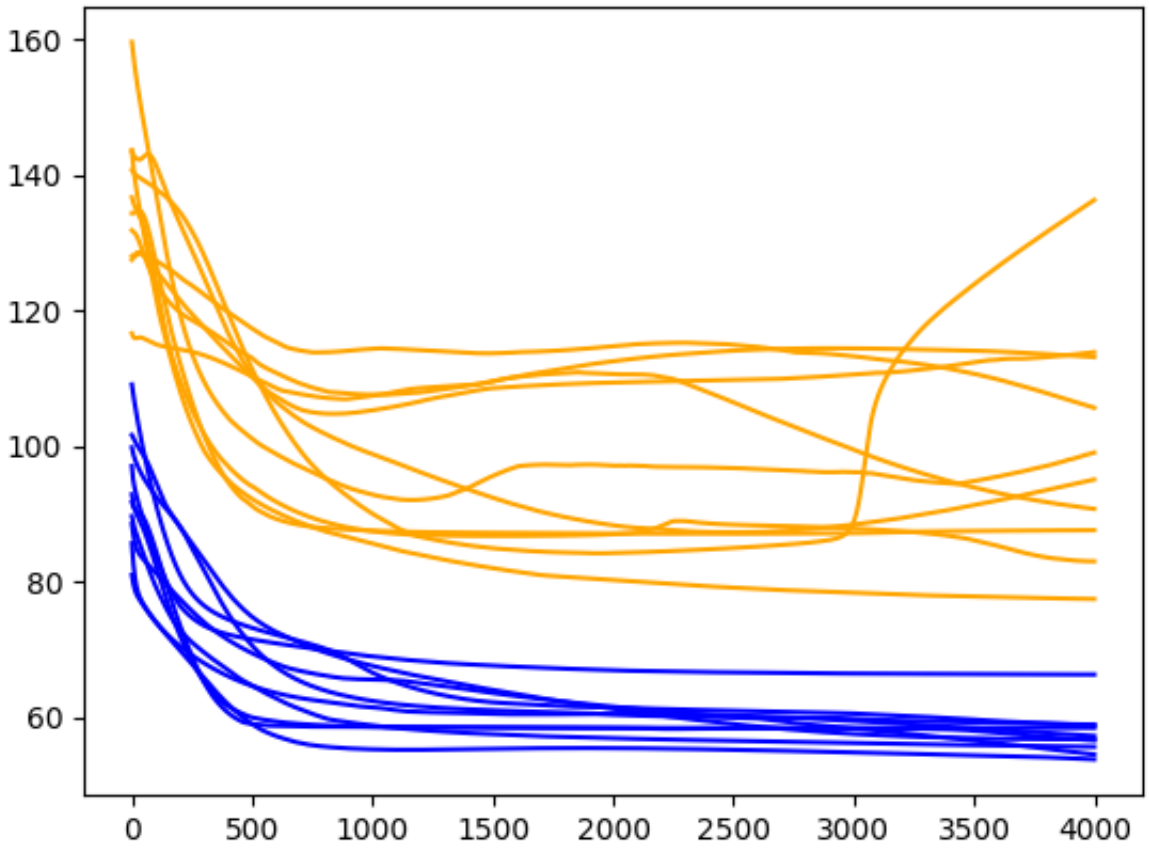
Specifically, the *n_epochs* parameter is set to 4000 in the *run()* function.

```
n_epochs = 4000
```

Running the example prints the RMSE for the train and test sets from the final epoch.

```
0) TrainRMSE=58.889277, TestRMSE=99.121765
1) TrainRMSE=56.839065, TestRMSE=95.144846
2) TrainRMSE=58.522271, TestRMSE=87.671309
3) TrainRMSE=53.873962, TestRMSE=113.920076
4) TrainRMSE=66.386299, TestRMSE=77.523432
5) TrainRMSE=58.996230, TestRMSE=136.367014
6) TrainRMSE=55.725800, TestRMSE=113.206607
7) TrainRMSE=57.334604, TestRMSE=90.814642
8) TrainRMSE=54.593069, TestRMSE=105.724825
9) TrainRMSE=56.678498, TestRMSE=83.082262
```

A line plot of the test and train RMSE scores each epoch is also created.



Diagnostic Results with 4000 Epochs

A similar pattern continues.

There is a general trend of improving performance, even over the 4000 epochs. There is one case of severe overfitting where test error rises sharply.

Again, most runs end with a “good” (better than persistence) final test error.

Summary of Results

The diagnostic runs above are helpful to explore the dynamical behavior of the model, but fall short of an objective and comparable mean performance.

We can address this by repeating the same experiments and calculating and comparing summary statistics for each configuration. In this case, 30 runs were completed of the epoch values 500, 1000, 2000, 4000, and 6000.

The idea is to compare the configurations using summary statistics over a larger number of runs and see exactly which of the configurations might perform better on average.

The complete code example is listed below.

```
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
# be able to save images on server
matplotlib.use('Agg')
from matplotlib import pyplot
import numpy

# 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)
    df = df.drop(0)
    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]

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
```

```
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, yhat):
    new_row = [x for x in X] + [yhat]
    array = numpy.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), statefu
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model

# run a repeated experiment
def experiment(repeats, series, epochs):
    # transform data to be stationary
    raw_values = series.values
    diff_values = difference(raw_values, 1)
    # transform data to be supervised learning
    supervised = timeseries_to_supervised(diff_values, 1)
    supervised_values = supervised.values
    # 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 model
        batch_size = 4
        train_trimmed = train_scaled[2:, :]
        lstm_model = fit_lstm(train_trimmed, batch_size, epochs, 1)
        # forecast the entire training dataset to build up state for forecasting
        train_reshaped = train_trimmed[:, 0].reshape(len(train_trimmed), 1, 1)
        lstm_model.predict(train_reshaped, batch_size=batch_size)
        # forecast test dataset
        test_reshaped = test_scaled[:,0:-1]
        test_reshaped = test_reshaped.reshape(len(test_reshaped), 1, 1)
        output = lstm_model.predict(test_reshaped, batch_size=batch_size)
        predictions = list()
        for i in range(len(output)):
            yhat = output[i,0]
            X = test_scaled[i, 0:-1]
            # 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


# load dataset
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True
# experiment
repeats = 30
results = DataFrame()
# vary training epochs
epochs = [500, 1000, 2000, 4000, 6000]
for e in epochs:
    results[str(e)] = experiment(repeats, series, e)
# summarize results
print(results.describe())
# save boxplot
results.boxplot()
pyplot.savefig('boxplot_epochs.png')
```

Running the code first prints summary statistics for each of the 5 configurations. Notably, this includes the mean and standard deviations of the RMSE scores from each population of results.

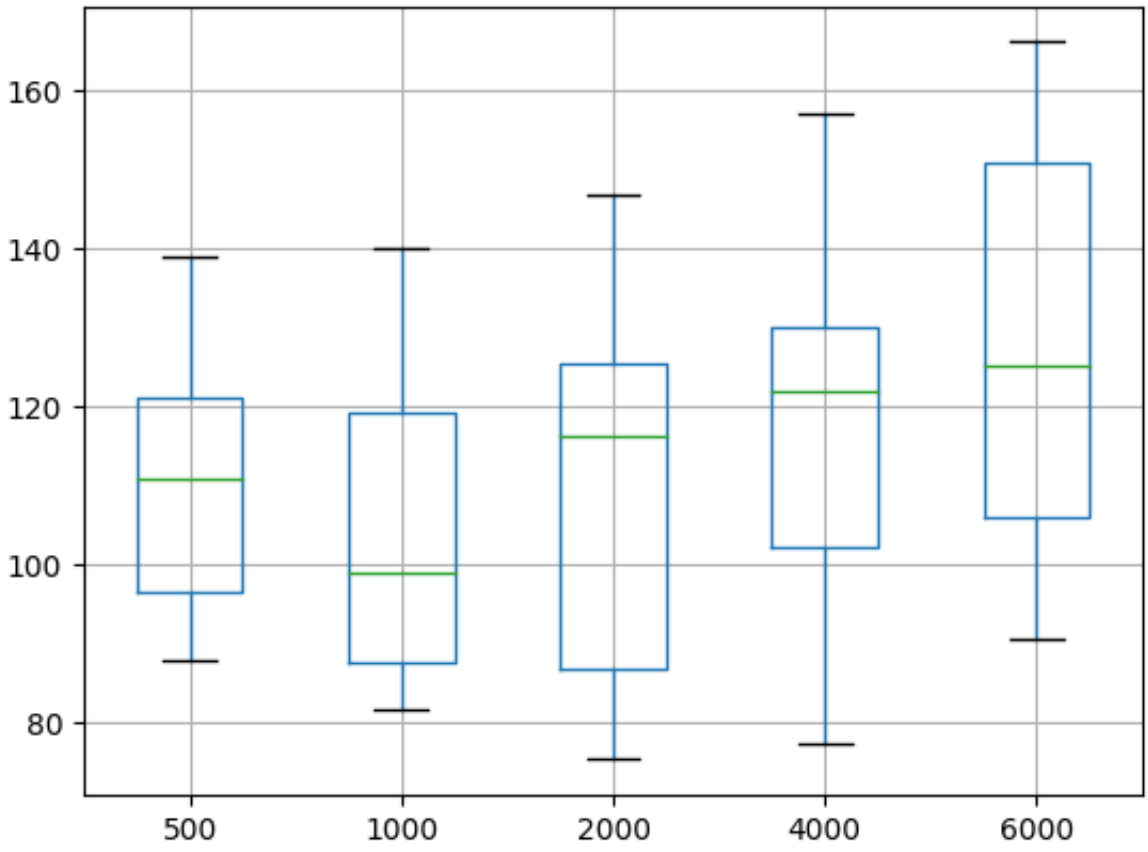
The mean gives an idea of the average expected performance of a configuration, whereas the standard deviation gives an idea of the variance. The min and max RMSE scores also give an idea of the range of possible best and worst case examples that might be expected.

Looking at just the mean RMSE scores, the results suggest that an epoch configured to 1000 may be better. The results also suggest further investigations may be warranted of epoch values between 1000 and 2000.

```
             500        1000        2000        4000        6000
count   30.000000   30.000000   30.000000   30.000000   30.000000
mean   109.439203  104.566259  107.882390  116.339792  127.618305
std     14.874031   19.097098   22.083335   21.590424   24.866763
min     87.747708   81.621783   75.327883   77.399968   90.512409
25%     96.484568   87.686776   86.753694  102.127451  105.861881
50%    110.891939   98.942264  116.264027  121.898248  125.273050
75%    121.067498  119.248849  125.518589  130.107772  150.832313
max    138.879278  139.928055  146.840997  157.026562  166.111151
```

The distributions are also shown on a box and whisker plot. This is helpful to see how the distributions directly compare.

The green line shows the median and the box shows the 25th and 75th percentiles, or the middle 50% of the data. This comparison also shows that the choice of setting epochs to 1000 is better than the tested alternatives. It also shows that the best possible performance may be achieved with epochs of 2000 or 4000, at the cost of worse performance on average.



Box and Whisker Plot Summarizing Epoch Results

Next, we will look at the effect of batch size.

Tuning the Batch Size

Batch size controls how often to update the weights of the network.

Importantly in Keras, the batch size must be a factor of the size of the test and the training dataset.

In the previous section exploring the number of training epochs, the batch size was fixed at 4, which cleanly divides into the test dataset (with the size 12) and in a truncated version of the test dataset

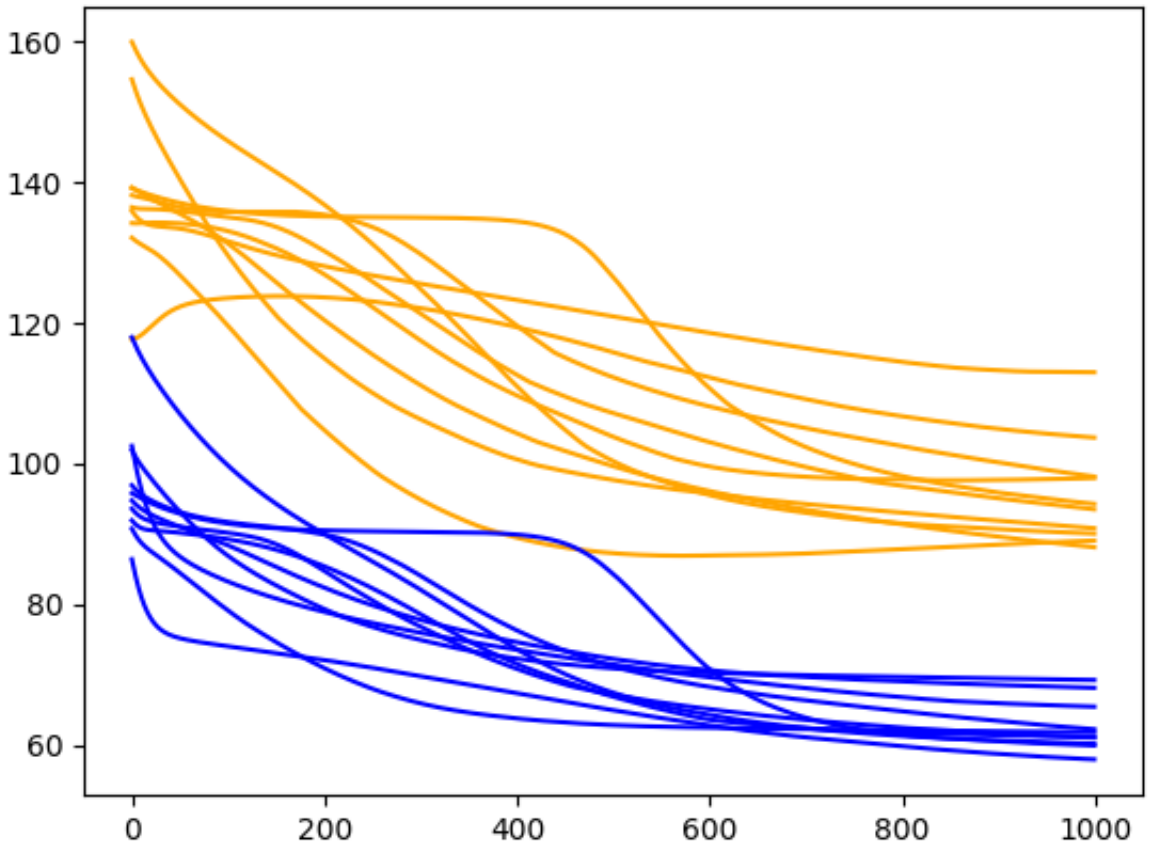
(with the size of 20).

In this section, we will explore the effect of varying the batch size. We will hold the number of training epochs constant at 1000.

Diagnostic of 1000 Epochs and Batch Size of 4

As a reminder, the previous section evaluated a batch size of 4 in the second experiment with a number of epochs of 1000.

The results showed a downward trend in error that continued for most runs all the way to the final training epoch.



Diagnostic Results with 1000 Epochs

Diagnostic of 1000 Epochs and Batch Size of 2

In this section, we look at halving the batch size from 4 to 2.

This change is made to the *n_batch* parameter in the *run()* function; for example:

```
n_batch = 2
```

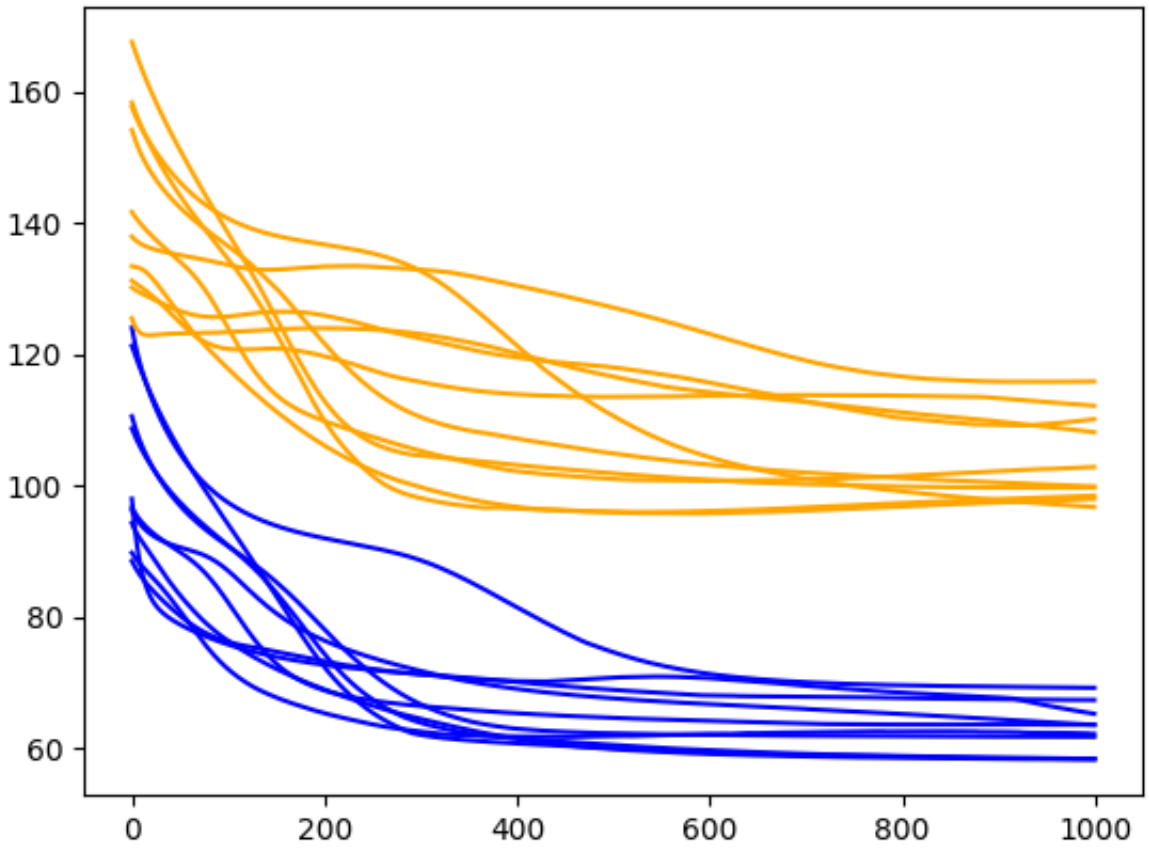
Running the example shows the same general trend in performance as a batch size of 4, perhaps with a higher RMSE on the final epoch.

The runs may show the behavior of stabilizing the RMES sooner rather than seeming to continue the downward trend.

The RSME scores from the final exposure of each run are listed below.

```
0) TrainRMSE=63.510219, TestRMSE=115.855819
1) TrainRMSE=58.336003, TestRMSE=97.954374
2) TrainRMSE=69.163685, TestRMSE=96.721446
3) TrainRMSE=65.201764, TestRMSE=110.104828
4) TrainRMSE=62.146057, TestRMSE=112.153553
5) TrainRMSE=58.253952, TestRMSE=98.442715
6) TrainRMSE=67.306530, TestRMSE=108.132021
7) TrainRMSE=63.545292, TestRMSE=102.821356
8) TrainRMSE=61.693847, TestRMSE=99.859398
9) TrainRMSE=58.348250, TestRMSE=99.682159
```

A line plot of the test and train RMSE scores each epoch is also created.



Diagnostic Results with 1000 Epochs and Batch Size of 2

Let's try having the batch size again.

Diagnostic of 1000 Epochs and Batch Size of 1

A batch size of 1 is technically performing online learning.

That is where the network is updated after each training pattern. This can be contrasted with batch learning, where the weights are only updated at the end of each epoch.

We can change the *n_batch* parameter in the *run()* function; for example:

```
n_batch = 1
```

Again, running the example prints the RMSE scores from the final epoch of each run.

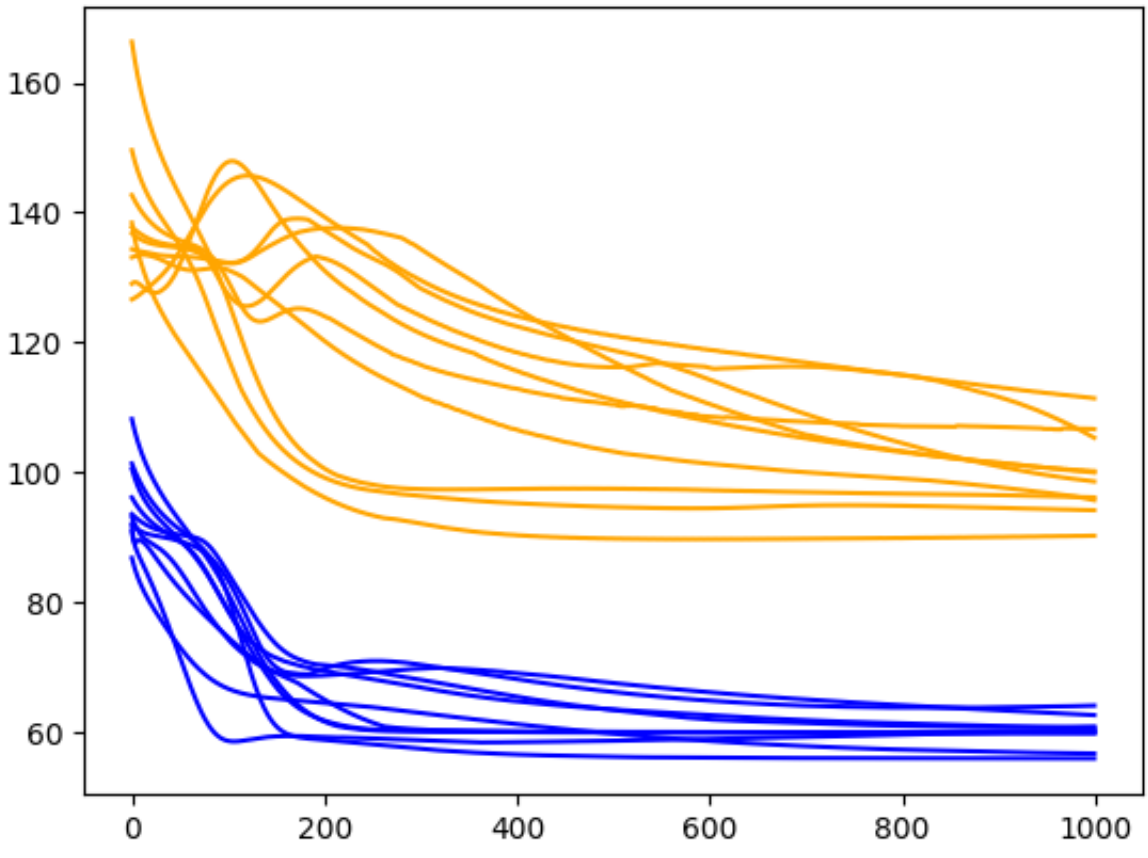
```
0) TrainRMSE=60.349798, TestRMSE=100.182293
1) TrainRMSE=62.624106, TestRMSE=95.716070
```

```
2) TrainRMSE=64.091859, TestRMSE=98.598958
3) TrainRMSE=59.929993, TestRMSE=96.139427
4) TrainRMSE=59.890593, TestRMSE=94.173619
5) TrainRMSE=55.944968, TestRMSE=106.644275
6) TrainRMSE=60.570245, TestRMSE=99.981562
7) TrainRMSE=56.704995, TestRMSE=111.404182
8) TrainRMSE=59.909065, TestRMSE=90.238473
9) TrainRMSE=60.863807, TestRMSE=105.331214
```

A line plot of the test and train RMSE scores each epoch is also created.

The plot suggests more variability in the test RMSE over time and perhaps a train RMSE that stabilizes sooner than with larger batch sizes. The increased variability in the test RMSE is to be expected given the large changes made to the network give so little feedback each update.

The graph also suggests that perhaps the decreasing trend in RMSE may continue if the configuration was afforded more training epochs.



Diagnostic Results with 1000 Epochs and Batch Size of 1

Summary of Results

As with training epochs, we can objectively compare the performance of the network given different batch sizes.

Each configuration was run 30 times and summary statistics calculated on the final results.

```
...

# run a repeated experiment
def experiment(repeats, series, batch_size):
    # transform data to be stationary
    raw_values = series.values
    diff_values = difference(raw_values, 1)
    # transform data to be supervised learning
    supervised = timeseries_to_supervised(diff_values, 1)
    supervised_values = supervised.values
    # 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 model
        train_trimmed = train_scaled[2:, :]
        lstm_model = fit_lstm(train_trimmed, batch_size, 1000, 1)
        # forecast the entire training dataset to build up state for forecasting
        train_reshaped = train_trimmed[:, 0].reshape(len(train_trimmed), 1, 1)
        lstm_model.predict(train_reshaped, batch_size=batch_size)
        # forecast test dataset
        test_reshaped = test_scaled[:,0:-1]
        test_reshaped = test_reshaped.reshape(len(test_reshaped), 1, 1)
        output = lstm_model.predict(test_reshaped, batch_size=batch_size)
        predictions = list()
        for i in range(len(output)):
            yhat = output[i,0]
            X = test_scaled[i, 0:-1]
            # 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


# load dataset
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True,
# experiment
```

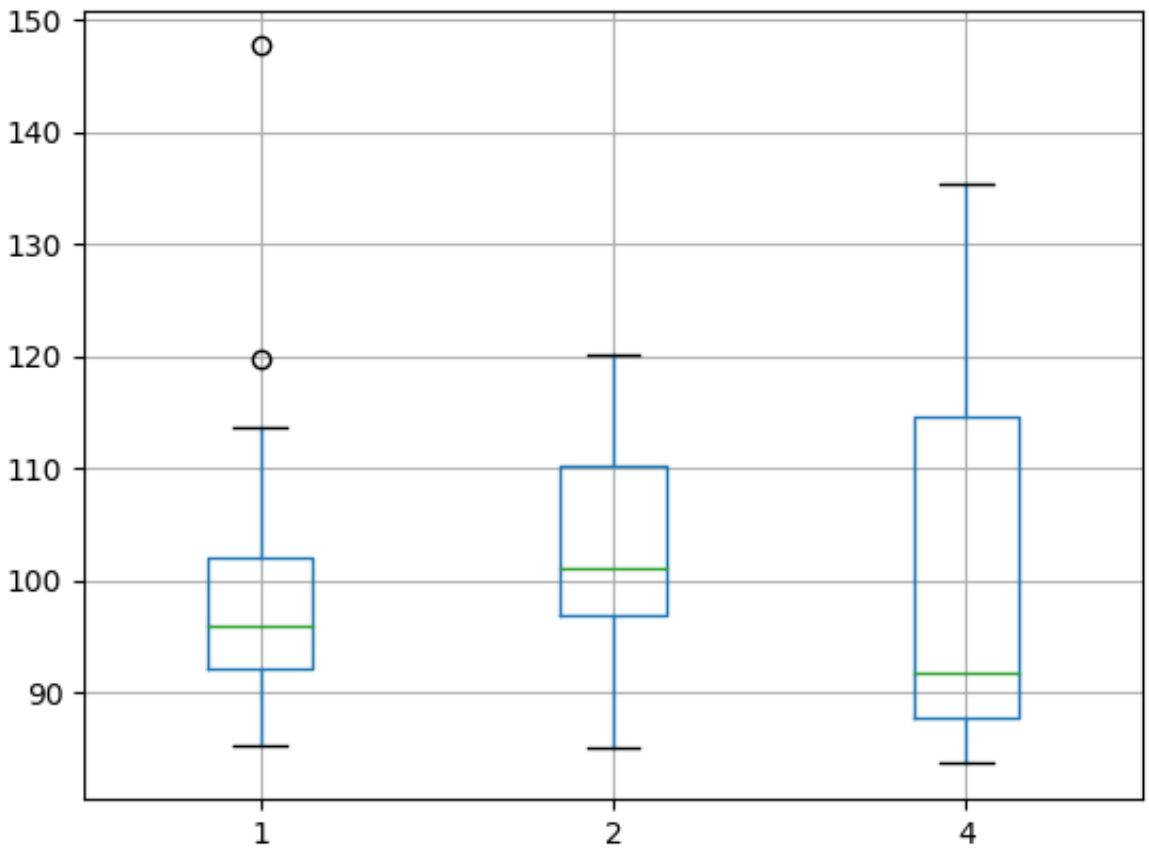
```
repeats = 30
results = DataFrame()
# vary training batches
batches = [1, 2, 4]
for b in batches:
    results[str(b)] = experiment(repeats, series, b)
# summarize results
print(results.describe())
# save boxplot
results.boxplot()
pyplot.savefig('boxplot_batches.png')
```

From the mean performance alone, the results suggest lower RMSE with a batch size of 1. As was noted in the previous section, this may be improved further with more training epochs.

```
                1           2           4
count   30.000000   30.000000   30.000000
mean    98.697017  102.642594  100.320203
std     12.227885    9.144163   15.957767
min     85.172215   85.072441   83.636365
25%     92.023175   96.834628   87.671461
50%     95.981688  101.139527   91.628144
75%    102.009268  110.171802  114.660192
max    147.688818  120.038036  135.290829
```

A box and whisker plot of the data was also created to help graphically compare the distributions. The plot shows the median performance as a green line where a batch size of 4 shows both the largest variability and also the lowest median RMSE.

Tuning a neural network is a tradeoff of average performance and variability of that performance, with an ideal result having a low mean error with low variability, meaning that it is generally good and reproducible.



Box and Whisker Plot Summarizing Batch Size Results

Tuning the Number of Neurons

In this section, we will investigate the effect of varying the number of neurons in the network.

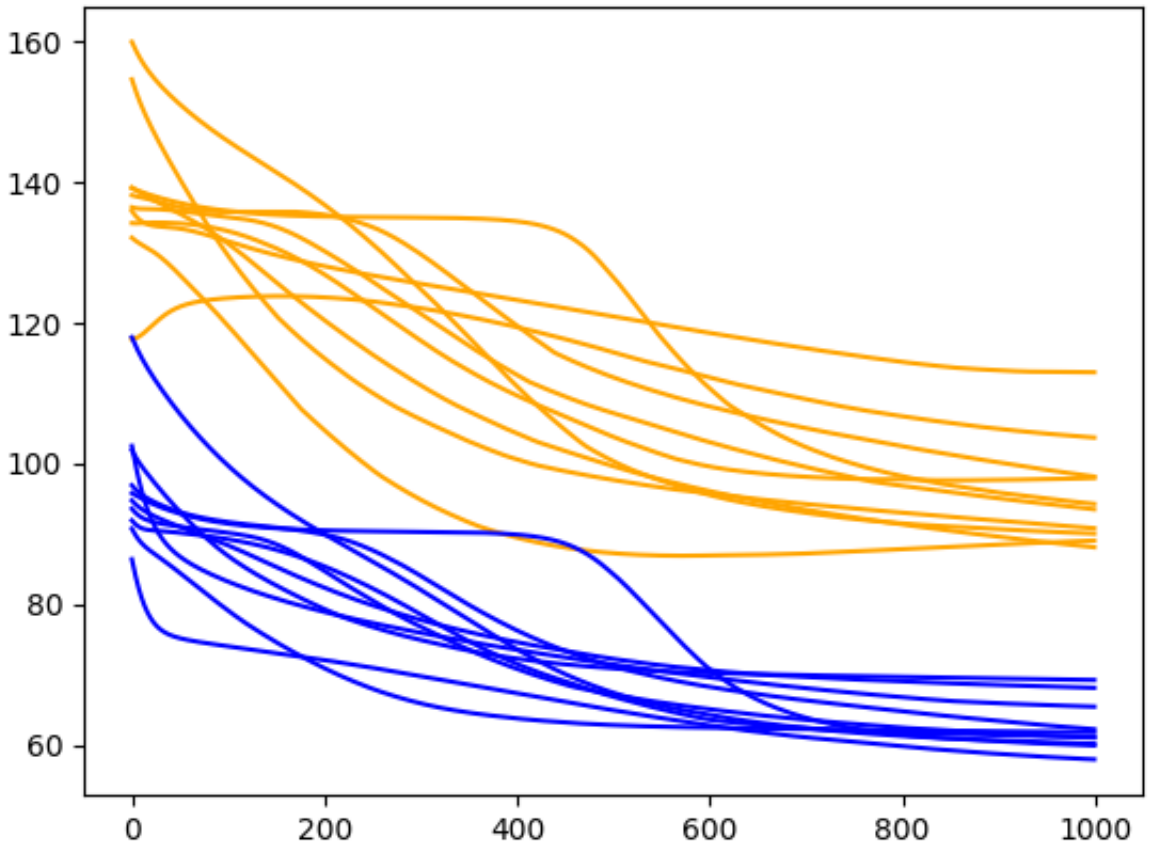
The number of neurons affects the learning capacity of the network. Generally, more neurons would be able to learn more structure from the problem at the cost of longer training time. More learning capacity also creates the problem of potentially overfitting the training data.

We will use a batch size of 4 and 1000 training epochs.

Diagnostic of 1000 Epochs and 1 Neuron

We will start with 1 neuron.

As a reminder, this is the second configuration tested from the epochs experiments.



Diagnostic Results with 1000 Epochs

Diagnostic of 1000 Epochs and 2 Neurons

We can increase the number of neurons from 1 to 2. This would be expected to improve the learning capacity of the network.

We can do this by changing the *n_neurons* variable in the *run()* function.

```
n_neurons = 2
```

Running this configuration prints the RMSE scores from the final epoch of each run.

The results suggest a good, but not great, general performance.

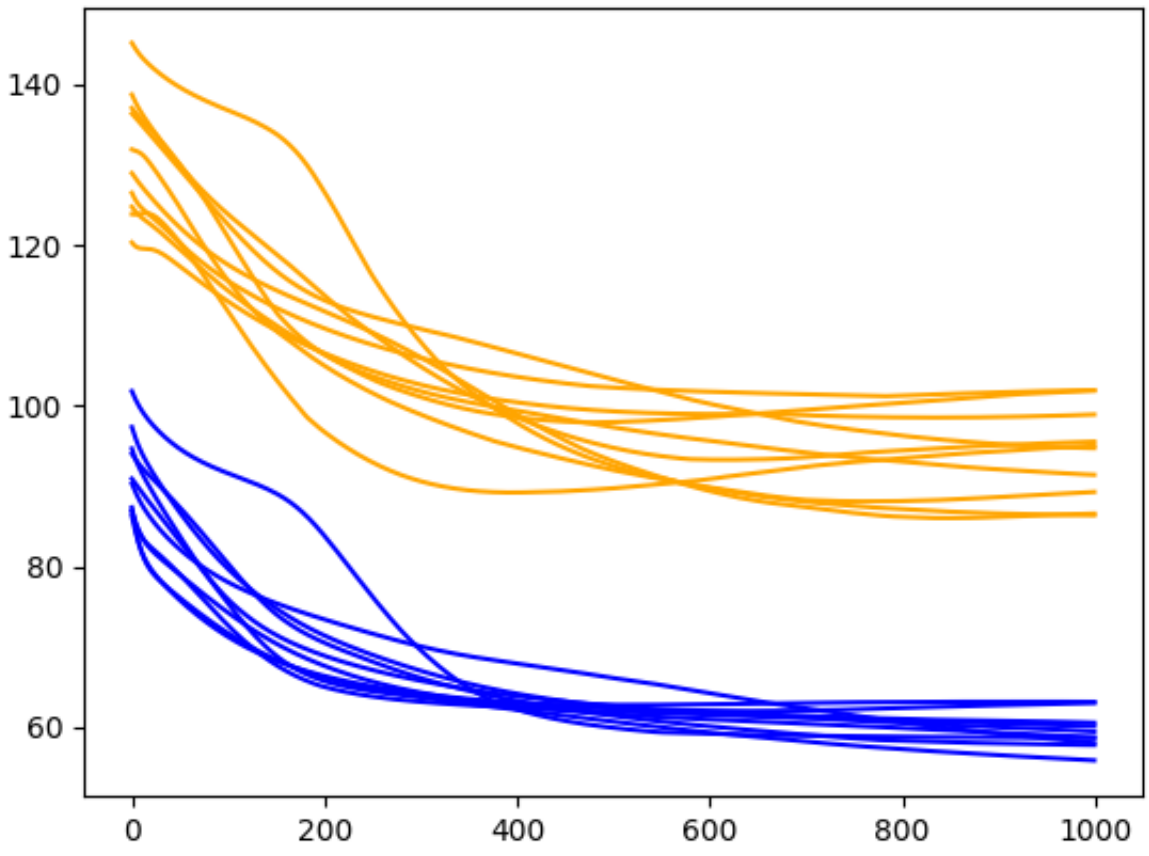
```
0) TrainRMSE=59.466223, TestRMSE=95.554547
1) TrainRMSE=58.752515, TestRMSE=101.908449
2) TrainRMSE=58.061139, TestRMSE=86.589039
3) TrainRMSE=55.883708, TestRMSE=94.747927
4) TrainRMSE=58.700290, TestRMSE=86.393213
```

```
5) TrainRMSE=60.564511, TestRMSE=101.956549
6) TrainRMSE=63.160916, TestRMSE=98.925108
7) TrainRMSE=60.148595, TestRMSE=95.082825
8) TrainRMSE=63.029242, TestRMSE=89.285092
9) TrainRMSE=57.794717, TestRMSE=91.425071
```

A line plot of the test and train RMSE scores each epoch is also created.

This is more telling. It shows a rapid decrease in test RMSE to about epoch 500-750 where an inflection point shows a rise in test RMSE almost across the board on all runs. Meanwhile, the training dataset shows a continued decrease to the final epoch.

These are good signs of overfitting of the training dataset.



Diagnostic Results with 1000 Epochs and 2 Neurons

Let's see if this trend continues with even more neurons.

Diagnostic of 1000 Epochs and 3 Neurons

This section looks at the same configuration with the number of neurons increased to 3.

We can do this by setting the *n_neurons* variable in the *run()* function.

```
n_neurons = 3
```

Running this configuration prints the RMSE scores from the final epoch of each run.

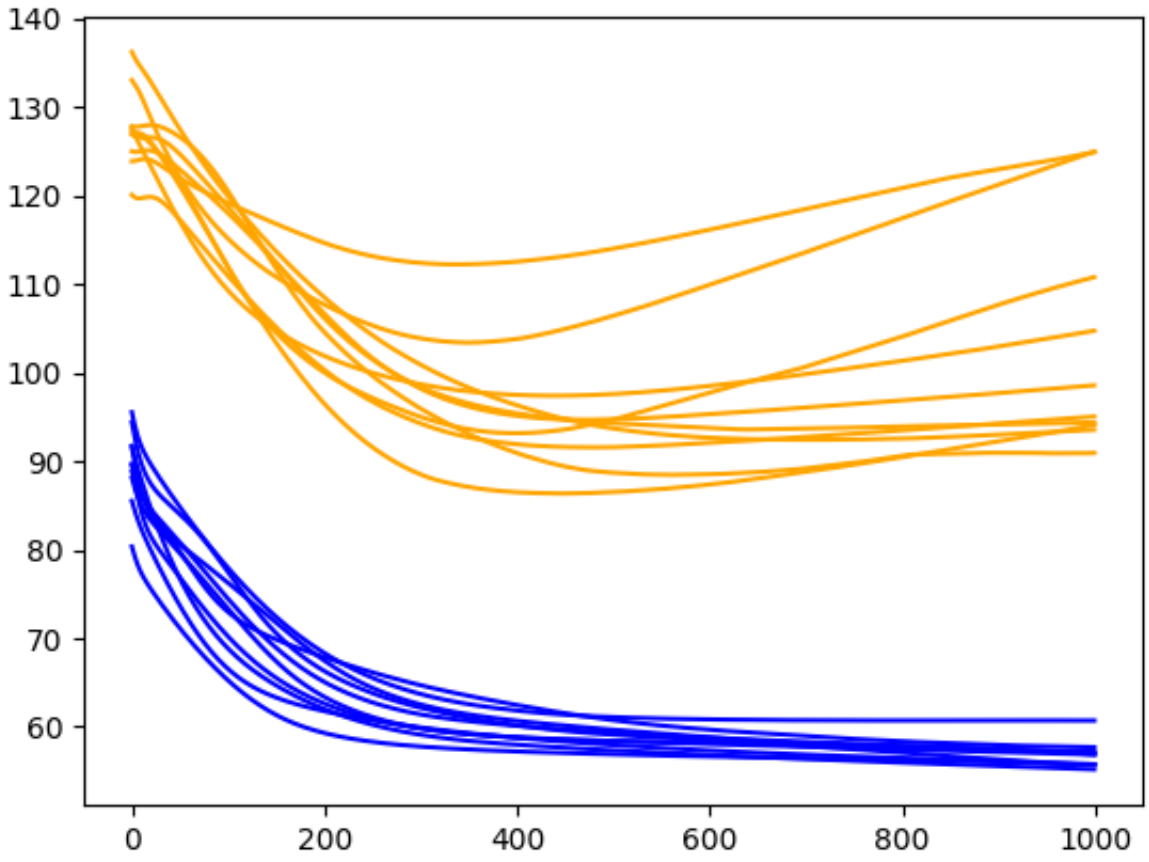
The results are similar to the previous section; we do not see much general difference between the final epoch test scores for 2 or 3 neurons. The final train scores do appear to be lower with 3 neurons, perhaps showing an acceleration of overfitting.

The inflection point in the training dataset seems to be happening sooner than the 2 neurons experiment, perhaps at epoch 300-400.

These increases in the number of neurons may benefit from additional changes to slowing down the rate of learning. Such as the use of regularization methods like dropout, decrease to the batch size, and decrease to the number of training epochs.

```
0) TrainRMSE=55.686242, TestRMSE=90.955555
1) TrainRMSE=55.198617, TestRMSE=124.989622
2) TrainRMSE=55.767668, TestRMSE=104.751183
3) TrainRMSE=60.716046, TestRMSE=93.566307
4) TrainRMSE=57.703663, TestRMSE=110.813226
5) TrainRMSE=56.874231, TestRMSE=98.588524
6) TrainRMSE=57.206756, TestRMSE=94.386134
7) TrainRMSE=55.770377, TestRMSE=124.949862
8) TrainRMSE=56.876467, TestRMSE=95.059656
9) TrainRMSE=57.067810, TestRMSE=94.123620
```

A line plot of the test and train RMSE scores each epoch is also created.



Diagnostic Results with 1000 Epochs and 3 Neurons

Summary of Results

Again, we can objectively compare the impact of increasing the number of neurons while keeping all other network configurations fixed.

In this section, we repeat each experiment 30 times and compare the average test RMSE performance with the number of neurons ranging from 1 to 5.

```
...

# run a repeated experiment
def experiment(repeats, series, neurons):
    # transform data to be stationary
    raw_values = series.values
    diff_values = difference(raw_values, 1)
    # transform data to be supervised learning
    supervised = timeseries_to_supervised(diff_values, 1)
    supervised_values = supervised.values
```

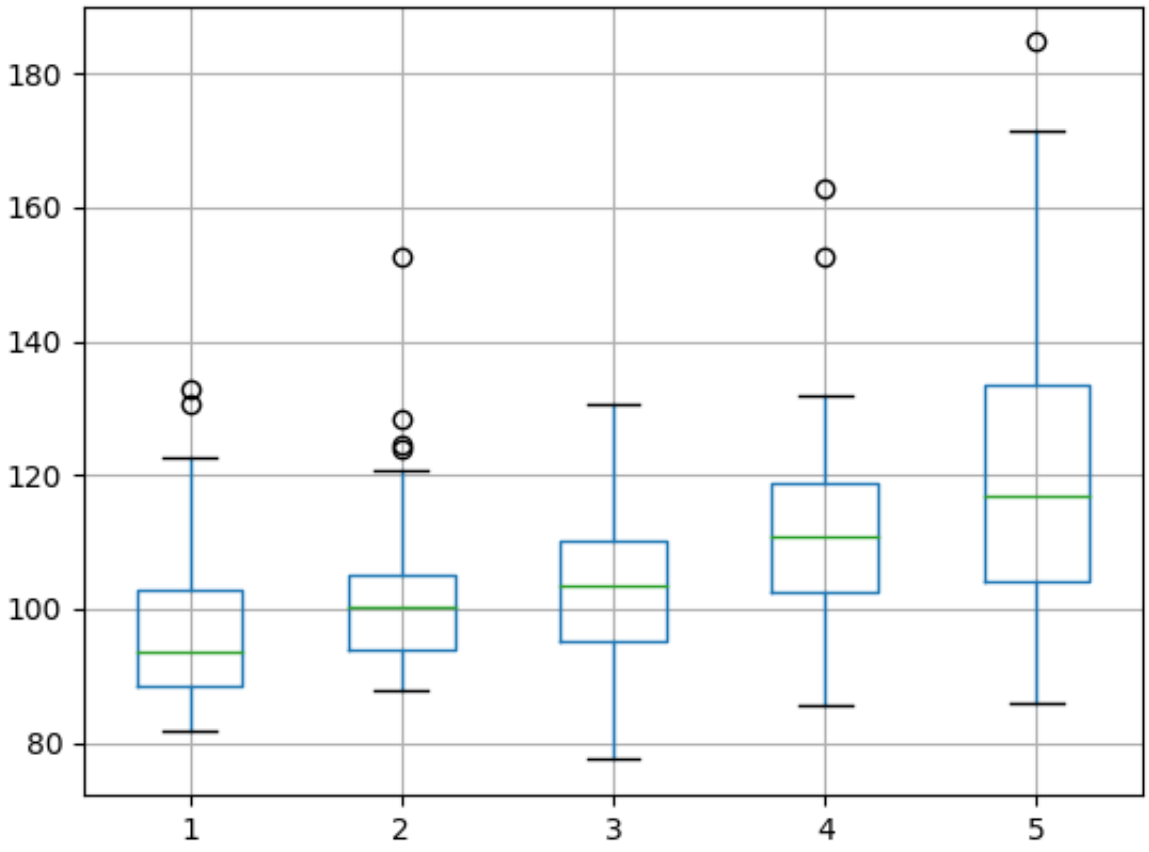
```
    # split data into train and test-sets
    train, test = supervised_values[0:-12], supervised_values[-12:]
    # transform the scale of the data
    scaler, train_scaled, test_scaled = scale(train, test)
    # run experiment
    error_scores = list()
    for r in range(repeats):
        # fit the model
        batch_size = 4
        train_trimmed = train_scaled[2:, :]
        lstm_model = fit_lstm(train_trimmed, batch_size, 1000, neurons)
        # forecast the entire training dataset to build up state for forecasting
        train_reshaped = train_trimmed[:, 0].reshape(len(train_trimmed), 1, 1)
        lstm_model.predict(train_reshaped, batch_size=batch_size)
        # forecast test dataset
        test_reshaped = test_scaled[:,0:-1]
        test_reshaped = test_reshaped.reshape(len(test_reshaped), 1, 1)
        output = lstm_model.predict(test_reshaped, batch_size=batch_size)
        predictions = list()
        for i in range(len(output)):
            yhat = output[i,0]
            X = test_scaled[i, 0:-1]
            # 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


# load dataset
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True,
# experiment
repeats = 30
results = DataFrame()
# vary neurons
neurons = [1, 2, 3, 4, 5]
for n in neurons:
    results[str(n)] = experiment(repeats, series, n)
# summarize results
print(results.describe())
# save boxplot
results.boxplot()
pyplot.savefig('boxplot_neurons.png')
```

Running the experiment prints the summary statistics for each configuration.

From the mean performance alone, the results suggest a network configuration with 1 neuron as having the best performance over 1000 epochs with a batch size of 4. This configuration also shows the tightest variance.

```
1                1           2           3           4           5
2  count    30.000000   30.000000   30.000000   30.000000   30.000000
3  mean     98.344696  103.268147  102.726894  112.453766  122.843032
4  std      13.538599   14.720989   12.905631   16.296657   25.586013
5  min      81.764721   87.731385   77.545899   85.632492   85.955093
6  25%      88.524334   94.040807   95.152752  102.477366  104.192588
7  50%      93.543948  100.330678  103.622600  110.906970  117.022724
8  75%     102.944050  105.087384  110.235754  118.653850  133.343669
9  max     132.934054  152.588092  130.551521  162.889845  184.678185
```

The box and whisker plot shows a clear trend in the median test set performance where the increase in neurons results in a corresponding increase in the test RMSE.



Box and Whisker Plot Summarizing Neuron Results

Summary of All Results

We completed quite a few LSTM experiments on the Shampoo Sales dataset in this tutorial.

Generally, it seems that a stateful LSTM configured with 1 neuron, a batch size of 4, and trained for

1000 epochs might be a good configuration.

The results also suggest that perhaps this configuration with a batch size of 1 and fit for more epochs may be worthy of further exploration.

Tuning neural networks is difficult empirical work, and LSTMs are proving to be no exception.

This tutorial demonstrated the benefit of both diagnostic studies of configuration behavior over time, as well as objective studies of test RMSE.

Nevertheless, there are always more studies that could be performed. Some ideas are listed in the next section.

Extensions

This section lists some ideas for extensions to the experiments performed in this tutorial.

If you explore any of these, report your results in the comments; I'd love to see what you come up with.

- **Dropout**. Slow down learning with regularization methods like dropout on the recurrent LSTM connections.
- **Layers**. Explore additional hierarchical learning capacity by adding more layers and varied numbers of neurons in each layer.
- **Regularization**. Explore how weight regularization, such as L1 and L2, can be used to slow down learning and overfitting of the network on some configurations.
- **Optimization Algorithm**. Explore the use of alternate optimization algorithms, such as classical gradient descent, to see if specific configurations to speed up or slow down learning can lead to benefits.
- **Loss Function**. Explore the use of alternative loss functions to see if these can be used to lift performance.
- **Features and Timesteps**. Explore the use of lag observations as input features and input time steps of the feature to see if their presence as input can improve learning and/or predictive capability of the model.
- **Larger Batch Size**. Explore larger batch sizes than 4, perhaps requiring further manipulation of the size of the training and test datasets.

Summary

In this tutorial, you discovered how you can systematically investigate the configuration for an LSTM network for time series forecasting.

Specifically, you learned:

- How to design a systematic test harness for evaluating model configurations.
- How to use model diagnostics over time, as well as objective prediction error to interpret model behavior.
- How to explore and interpret the effects of the number of training epochs, batch size, and number of neurons.

Do you have any questions about tuning LSTMs, or about this tutorial?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

‹ How to Seed State for LSTMs for Time Series Forecasting in Python
How to Update LSTM Networks During Training for Time Series Forecasting ›

153 Responses to *How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting*

John Richards April 12, 2017 at 8:51 am #

REPLY

Awesome Work!

Jason Brownlee April 12, 2017 at 9:34 am #

REPLY

Thanks John!

Calvin December 22, 2019 at 5:42 pm #

REPLY

Thanks for such a detailed and helpful tutorial. I would like ask: when tune hyperparamters, why in the order of epoch number, batch size, then Number of neurons? Is there anything special about this order? Thanks a lot!

Jason Brownlee December 23, 2019 at 6:45 am #

REPLY

Probably start with the setting a large capacity for the model then tune the learning rate:
https://machinelearningmastery.com/framework-for-better-deep-learning/

Kunpeng Zhang April 12, 2017 at 10:51 am #

REPLY

line 137, in
results[str(e)] = experiment(repeats, series, e)

line 104, in experiment
lstm_model = fit_lstm(train_trimmed, batch_size, epochs, 1)
line 81, in fit_lstm
model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
File “C:\Anaconda3\lib\site-packages\keras\models.py”, line 654, in fit
str(kwargs))
TypeError: Received unknown keyword arguments: {‘epochs’: 1}
Is there something wrong?

Jason Brownlee April 13, 2017 at 9:54 am #

REPLY

You need to upgrade to Keras v2.0 or higher.

Logan May 17, 2017 at 12:28 am #

REPLY

You should replace model.fit(X, y, epochs=1, …) for model.fit(X, y, nb_epoch=1…) It worked perfectly for me.

Jason Brownlee May 17, 2017 at 8:25 am #

REPLY

The example was updated to use Keras v2.0 which changed the “nb_epochs” argument to now be “epochs”.

Leo April 12, 2017 at 6:12 pm #

REPLY

Nice blog post once again. What is the size of the data set that you are using? What is the maximum data size that one can use with your example if I have a system with 8GB RAM?

Jason Brownlee April 13, 2017 at 9:56 am #

REPLY

The examples use small <1 MB datasets.

The size of supported datasets depends on data types, number of rows and number of columns.

Consider doing some experiments with synthetic data to find the limits of your system.

Lorenzo April 13, 2017 at 1:08 am #

REPLY

Amazing tutorial as always. Would you have an example of running the LSTM on a multivariate regression problem like the Walmart's on Kaggle (https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting) ?

Jason Brownlee April 13, 2017 at 10:02 am #

REPLY

There should be some examples on the blog soon.

Tykimos April 13, 2017 at 4:28 am #

REPLY

Great work! I have a question regarding batch_size. I think that only one batch_size is reasonable for this case because stateful is true. If batch_size is 4, there are 4 states separately. That means 4 states don't share each other. Although every sample has connection with the next sample sequently, every sample is connected the quaternary next sample.

For example,
1 > 5 > 9
2 > 6 > 10
3 > 7 > 11
4 > 8 > 12

Jason Brownlee April 13, 2017 at 10:12 am #

REPLY

Sorry, I'm not sure I understand your question, perhaps you can restate it.

We are using stateful LSTMs and regardless of the batch size, state is only reset when we reset it explicitly with a call to model.reset_states()

tykimos April 14, 2017 at 1:56 am #

REPLY

I think that the batch_size depends on not user tuning parameter but dataset design relative to the number of sequence set.

When batch_size is 4 and stateful is true, output of LSTM is weight_count * batch_size. This means that there are 4 different and independent states. It seems good for the following dataset :
A1-B1-C1-D1-A2-B2-C2-D2-A3-B3-C3-D3
– sample is 12
– batch_size is 4
– sequence set is 4
Because A isn't relative to others in sequence, the next of A1 should be not B1 but A2. For this we have to choose batch_size as 4.

For each epoch,
A1-A2-A3 and reset
B1-B2-B3 and reset
C1-C2-C3 and reset
D1-D2-D3 and reset

Your example is one sequence set like:
A1-A2-A3-A4-A5-A6-A7-A8-A9-A10-A11-A12
If we set batch_size as 4, state is transferred as the following:
A1-A5-A9 and reset
A2-A6-A10 and reset
A3-A7-A11 and reset
A4-A8-A12 and reset

So, I think that batch_size should be 1 in your dataset. Other batch_size values are invalid.

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

REPLY

I agree with the first part, it makes sense to reset state at the natural end of a sequence.

I also agree that with the dataset used that there is no natural end to the sequence other than the end of the data, so no state resets are needed.

I disagree that you "have to have" a batch size of 1. We can have a batch size of 1. We can also have a batch size equal to the sequence length.

My previous point is that batch size does not matter when stateful=True because we manage exactly when the state is reset. It does not matter what batch size is used, as long as the state is reset at the natural end of the sequence.

Yes, you are right, there is a bug. The regime of resetting state after each batch of 4 input patterns does not make sense. I'll schedule time to fix the examples and re-run the experiments.

Thanks for pointing this out and thanks for having the patients to help me see what you saw.

UPDATE: There is no fault, all weight updates are occurring within one epoch regardless of batch size.

Jason Brownlee April 24, 2017 at 8:02 am #

Update I have taken a much closer look at this code (it was written months ago).

I believe there is no fault.

Take a close look at the loop for running manual epochs.

Although the batch sizes vary, we are performing one entire epoch + all weight updates before resetting state.

Sebastian April 13, 2017 at 7:51 pm #

REPLY

Great blog post!

Jason Brownlee April 14, 2017 at 8:44 am #

REPLY

Thanks Sebastian.

Jesse April 26, 2017 at 4:54 pm #

REPLY

Hi Jason,

Cool post! I already learnt a lot from your blogs.

I have 2 questions regarding this post, I hope you can help me:

1. Why aren't you specifying activation functions in your model layers (for the LSTM layer for example?) I read on the keras docs that no activation function is used if you don't pass one. It seems to me that a tanh activation would fit the [-1,1] scaling?
2. Do you always remove the increasing trend from the data?

Thanks a lot!

Jason Brownlee April 27, 2017 at 8:35 am #

REPLY

I am using the default for LSTMs which are tanh and sigmoid.

Yes, making time series data stationary is a recommended in general.

Sebastian June 16, 2017 at 12:10 am #

REPLY

Hi Jason,

very nice post, but still got a question.

When reversing difference for predicitions, isn't it wrong to reverse it using the raw values? Instead, shouldn't you use the last prediction to reverse the difference?
Like, instead of yhat = yhat + history[-interval] it is yhat = yhat + predictions[-interval]?

Or am I misunderstanding something?

Freetings

Jason Brownlee June 16, 2017 at 8:02 am #

REPLY

Yes, but if you have the real observations for a past time step, we should use them instead to better reflect the "real" level.

Does that help?

Sepideh September 30, 2017 at 7:42 am #

REPLY

Thank you Dr. Jason for the great blog. Is it possible to use GPU for this example? If so, how can I apply it! I appreciate your help. The reason I want to use GPU is that I need to get results faster. I am using google compute engine with 6vCPUs and 39 GB memory.

Jason Brownlee September 30, 2017 at 7:48 am #

REPLY

Perhaps. The GPU configuration is controlled by the library used by Keras, such as TensorFlow.

I have found that it is better to run models sequentially on the GPU and instead use multiple GPUs/servers to test configurations.

Farzad December 7, 2017 at 1:54 am #

REPLY

Many thanks for your great tutorial.
I have a question regarding updating the model after making a prediction on each timestep of the test dataset. Could you please explain which line of the code is responsible for doing this; I mean the line that implements this statement: "… then the actual expected value from the test set will be taken and made available to the model for the forecast on the next time step."

Jason Brownlee December 16, 2017 at 5:14 am #

REPLY

The model is not updated after making a prediction in this post.

This is called updating a model, see an example here:
https://machinelearningmastery.com/update-lstm-networks-training-time-series-forecasting/

By that line, I meant that we are adding observations to the history used by the model for making the next prediction.

Suraj Nepal January 3, 2018 at 4:27 pm #

REPLY

Thank you so much for this wonderful tutorial.

Can you please help me to implement Grid search and Bayesian optimization approach for LSTM hyper parameter tuning.

Thank you in advance.

Jason Brownlee January 4, 2018 at 8:03 am #

REPLY

What problem are you having exactly?

Suraj Nepal January 4, 2018 at 1:19 pm #

REPLY

I want to implement proper hyper parameter tuning mechanism for LSTM time series forecasting.

But I am facing difficulties using grid search for time series model.
Also I dont find proper guidelines for using Bayesian optimization in LSTM time series forecasting.
Thank you in advance

Jason Brownlee January 4, 2018 at 3:27 pm #

REPLY

Sorry I don't have an example of using Bayesian optimization for tuning LSTMs.
Thanks for the suggestion.

Edward January 4, 2018 at 5:03 am #

REPLY

Hi Jason!
Can we use GridSearchCV to tune LSTM Hyperparameters with Keras for Time Series Forecasting?
or we can not do this because for GridSearchCV we need cross validation but in this case we can not use cross validation (time seies)

Jason Brownlee January 4, 2018 at 8:16 am #

REPLY

I would recommend writing your own for-loop and using walk-forward validation to test time series forecasting models.
https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

Edward January 5, 2018 at 3:22 am #

REPLY

thanks a lot Jason

Galen February 14, 2019 at 9:49 pm #

REPLY

hi Jason,

I think it should be possible to use gridsearchCV with LSTMs, since there is a built-in validation_split in keras.Sequential models.

gridsearchCV would be useful since it can run in parallel (also on databricks).

Do you have any simple examples of this? I get a strange error when I run grid.fit

thanks

‘model = KerasRegressor(build_fn=builder.build_LSTM_simple, verbose=verbose)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X, y)’

————————————————————————
AttributeError Traceback (most recent call last)
in ()
—-> 1 grid_result = grid.fit(X, y)

~/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py in fit(self, X, y, groups, **fit_params)
638 error_score=self.error_score)
639 for parameters, (train, test) in product(candidate_params,
–> 640 cv.split(X, y, groups)))
641
642 # if one choose to see train score, “out” will contain train score info

~/anaconda3/lib/python3.6/site-packages/sklearn/externals/joblib/parallel.py in __call__(self, iterable)
787 # consumption.
788 self._iterating = False
–> 789 self.retrieve()
790 # Make sure that we get a last message telling us we are done
791 elapsed_time = time.time() – self._start_time

~/anaconda3/lib/python3.6/site-packages/sklearn/externals/joblib/parallel.py in retrieve(self)
697 try:
698 if getattr(self._backend, 'supports_timeout', False):
–> 699 self._output.extend(job.get(timeout=self.timeout))
700 else:
701 self._output.extend(job.get())

~/anaconda3/lib/python3.6/multiprocessing/pool.py in get(self, timeout)
642 return self._value
643 else:
–> 644 raise self._value
645
646 def _set(self, i, obj):

~/anaconda3/lib/python3.6/multiprocessing/pool.py in _handle_tasks(taskqueue, put, outqueue, pool, cache)
422 break
423 try:
–> 424 put(task)
425 except Exception as e:
426 job, idx = task[:2]

~/anaconda3/lib/python3.6/site-packages/sklearn/externals/joblib/pool.py in send(obj)
369 def send(obj):
370 buffer = BytesIO()
–> 371 CustomizablePickler(buffer, self._reducers).dump(obj)
372 self._writer.send_bytes(buffer.getvalue())
373 self._send = send

~/anaconda3/lib/python3.6/site-packages/statsmodels/graphics/functional.py in _pickle_method(m)

AttributeError: 'function' object has no attribute 'im_self'

Galen February 15, 2019 at 12:30 am #

REPLY

Ok, I found that updating scikit-learn to the latest version fixed this, so I am now able to do a gridsearchCV on an LSTM.

Jason Brownlee February 15, 2019 at 8:06 am #

Glad to hear it.

Jason Brownlee February 15, 2019 at 8:06 am #

REPLY

I show how to manually grid search LSTMs here:
https://machinelearningmastery.com/how-to-grid-search-deep-learning-models-for-time-series-forecasting/

aniya February 5, 2018 at 8:09 pm #

REPLY

hi jason ,
could you tell me about LSTM in which loss function ‘mean_squared_error’ is used.why it is used ..in that why cannot we measure accuracy…

Jason Brownlee February 6, 2018 at 9:13 am #

REPLY

This post will help clarify things for you:
https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

Sacha_yacoubi February 10, 2018 at 7:11 am #

REPLY

Dear Jason,
To make sure I am interpreting things as I should , what is th difference between neurons , layers and hidden states?

I am asking because the inner structure of an LSTM is completely different from any other NN and is composed of mulptiple gates rather than “neurons”.
Am I missing Something here?

Thank you very much

Jason Brownlee February 10, 2018 at 9:01 am #

REPLY

A unit is a neuron and a layer is a group of units. This is the same as an MLP.

Hidden state is a like an additional weight or local variable inside each neuron.

Does that help?

Sacha_yacoubi February 10, 2018 at 10:53 am #

REPLY

ok thank you Jason. I understand then that a neuron in LSTM does exactly the same thing as in feedforward NN, that is: computing the activation function. The difference is that in LSTM, the neuron is far more complex and is composed of 4 gates and processes a hidden vector that is recursively fed to the neuron itself.

Lu_Zweig February 12, 2018 at 4:21 am #

REPLY

Dear all ,
How should I reshape my dataset if the objective is to use the last M points to predict the next N points for multiple time series at once. In my project, I am dealing with 20 time series at the same time.
Thank you very much for you help!!

Jason Brownlee February 12, 2018 at 8:32 am #

REPLY

See this post:
https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/

And perhaps this post:
https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

joseph February 13, 2018 at 12:29 pm #

REPLY

Hi Jason,

May i know why do we need to reset the model after evaluating?i know that we need to reset when performing model.fit , but why do so on evaluating as well? thank you

Jason Brownlee February 14, 2018 at 8:13 am #

REPLY

It will have state from making predictions that might not make sense for the next use of the model.

char March 10, 2018 at 6:15 am #

REPLY

Why do you trim the data by 2 in the experiment function?

Jason Brownlee March 10, 2018 at 6:36 am #

REPLY

I might be removing rows with nan values given the data shift.

Doctor March 13, 2018 at 1:44 am #

REPLY

It looks like there is no one optimal combination of Epochs, Batch Size and Neurons. Is this a correct observation?

Also, does tuning so much on the given training and testing set lead to overfitting to these sets? Should there be training-validation-test combination where you would check findings with the test set in the end?

Jason Brownlee March 13, 2018 at 6:30 am #

REPLY

Yes, and results vary from problem to problem in general.

It can lead to overfitting.

Yes, this post will clarify things:
https://machinelearningmastery.com/difference-test-validation-datasets/

Eric March 20, 2018 at 11:20 pm #

REPLY

Hi Jason,

Great post! I'm curious as to why you scale the testing and training sets separately here, but do so as one complete data set in this post (https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/).

thanks

Jason Brownlee March 21, 2018 at 6:35 am #

REPLY

Sometimes I do it in one step because I am focusing the tutorial on some other aspect.

Kevin Millan March 23, 2018 at 12:08 pm #

REPLY

Hello Jason, thank you for your dedication I have learned a lot from you.

Currently I'm trying to predict the energy output of wind turbines for an interval of every 15 minutes using LSTM. Are there any optimization methods, loss functions among other parameters you would particularly suggest for such a case study? my data set is a data set of dimensions (57000, 17). thank you

Jason Brownlee March 24, 2018 at 6:18 am #

REPLY

Perhaps try MAE or MSE loss functions as a start?

Raffy April 1, 2018 at 2:02 am #

REPLY

Hi Sir, thank you very much for all your content. This is my first time learning machine learning specifically LSTM.

Is it possible to tune these hyperparameters simultaneously?

ei. for each epoch (say 500, 1000, 3000) try batch sizes of 1,2,3 and for each batch size set neurons to 1,2,3,4

With this method, can we assess which of the following combinations yield better result? I am aware that this is would take a lot of time to do but I am curious if this is plausible and sensible to perform.

Jason Brownlee April 1, 2018 at 5:50 am #

REPLY

Yes, but it would require a lot of computational resources.

Elon April 1, 2018 at 9:55 pm #

REPLY

Hi Jason,

First, thank you for your great content. I am a beginner of the RNN, and I am a little confused with this line:

transform train
> train = train.reshape(train.shape[0], train.shape[1])
train_scaled = scaler.transform(train)

Why do I need to reshape this, it seems the shape of it hasn't change after the reshape.

Thank you 🙂

Jason Brownlee April 2, 2018 at 5:22 am #

REPLY

Perhaps it is unneeded, try removing this line and see what happens?

Skye April 6, 2018 at 6:38 pm #

REPLY

Hi Jason,

Thank a lot for your great tutorial and books! I am working on a time-series forecasting problem using LSTM and want to ask you some questions:

1. I have tuned hyperparameters according to your tutorial. But the validation loss always lower than training loss. What‘s the reason for this?

2. By the time of 30th epochs it has converged, is this normal?

3. Why choose ‘adam’ optimizer?

Looking forward to your reply~ Thank you very much!

Jason Brownlee April 7, 2018 at 6:22 am #

REPLY

That is odd that test loss is better than training. Perhaps the model is underspecified?

There are no rules about number of epochs and convergence. It is what it is for your problem and model config.

Adam is fast, sgd is slow:
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

There is no best optimizer though, use what you prefer.

Skye April 7, 2018 at 3:14 pm #

REPLY

Hi Jason,

Thanks for your reply!

But what is the meaning of ‘the model is underspecified’? The model and the method I use are almost the same as those in the article.

Jason Brownlee April 8, 2018 at 6:15 am #

REPLY

Perhaps make sure you have copied all of the code.

Skye April 8, 2018 at 11:15 am #

REPLY

I have not difference the series because there is no trend in this series. I calculated statistics of partitioned time series data and used ADF test, the result is as follows, so I think it a stationary series.

mean1=-0.130372, mean2=-0.077736
variance1=0.539531, variance2=0.652681

ADF statistic: -9.851064
p-value: 0.000000
Critical values:
1%: -3.434
5%: -2.863
10%: -2.568

Does it matter?

Jason Brownlee April 9, 2018 at 6:02 am #

REPLY

Perhaps try modeling and see if it matters.

Skye April 9, 2018 at 10:27 am #

REPLY

Ok. I will try it. Thank you a lot.

Jason Brownlee April 10, 2018 at 6:10 am #

REPLY

You're welcome.

Max April 11, 2018 at 4:47 pm #

REPLY

Hi Jason

Thanks for your sharing. I tried your codes on my jupyter notebook. Tuning epochs worked properly. However, fit_lstm fn in tuning epochs gives a dataframe as an output.
So it is not the same fit_lstm fn in tuning batch_size. Which should have a model to predict as an output.

In my notebook it gives such an error:

'AttributeError: 'DataFrame' object has no attribute 'predict'

Am I missing something or is it because of my Keras version?

Jason Brownlee April 12, 2018 at 8:33 am #

REPLY

I recommend using the latest version of all of the libraries and run intensive code like this from the command line.

Max April 11, 2018 at 4:49 pm #

REPLY

Sorry

I found my mistake u can delete my comment.

Thanks

Jason Brownlee April 12, 2018 at 8:34 am #

REPLY

I'm glad to hear that you fixed your issue.

Jason C. April 30, 2018 at 1:14 am #

REPLY

Just noticed I may encountered the same problem. Could you advise me how did you fix it? Thanks.

Jason April 30, 2018 at 1:10 am #

REPLY

Hello Jason,

I would like to know how I can save a model and use it to forecast.

In your “Summary of Results” of the last section (the codes above “Summary of All Results”), line 21: I want to save the last model which function “fit_lstm” output.

I modified your code to achieve this but give me error “AttributeError: ‘Series’ object has no attribute ‘predict'”, I figured out that the error was due to the model I saved was not a correct one to use, though I checked the model’s value was “<keras.models.Sequential object at 0x0000026EC…" which looks right.

Below is what I did:

```
# fit an LSTM network to training data
def fit_lstm(train, test, raw, scaler, batch_size, nb_epoch, neurons):
#### omitted copy of your codes here#####
# fit model
train_rmse, test_rmse = list(), list()
for i in range(nb_epoch):
model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
model.reset_states()
#### omitted copy of your codes here#####
history = DataFrame()
history['train'], history['test'] , history['model']= train_rmse, test_rmse, model
return history
### save last model to dataframe "history"

# run diagnostic experiments
def run(repeats, n_epochs, n_batch, n_neurons):
# fit and evaluate model
train_trimmed = train_scaled[2:, :]
results_run = DataFrame()
error_scores = list()
# run diagnostic tests
for i in range(repeats):
history = fit_lstm(train_trimmed, test_scaled, raw_values, scaler, n_batch, n_epochs, n_neurons)
#### omitted copy of your codes here#####
if i==0:
model_tune = history['model']
results_run['error_scores'], results_run['model_tune'] = error_scores, model_tune
return results_run

### Below out of function and loops ####
```

results_temp = run(repeats, n_epochs, n_batch, n_neurons)
model_tune_all = results_temp['model_tune']
model_tune = model_tune_all[-1:]
get model_tune is <keras.models.Sequential object at 0x0000026EC…
forecasts = make_predictions(model_tune, n_batch, test_scaled, n_lag)
AttributeError: 'Series' object has no attribute 'predict'

end

thanks for your help in advance!

Jason Brownlee April 30, 2018 at 5:38 am #

REPLY

This post shows you how to save a model:
https://machinelearningmastery.com/save-load-keras-deep-learning-models/

This post shows you how to make predictions:
https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/

Felix May 1, 2018 at 11:34 pm #

REPLY

If I use the sequence length 3. x1,x2,x3 -> x4 = y1. Should I set batch size 3 for the fitting? I am not using stateful. The sequences are not dependent on each other so I cant have the LSTM take information from another sequence.

Jason Brownlee May 2, 2018 at 5:44 am #

REPLY

Batch size has to do with the number of samples/sequence before updating weights and resetting state.

You can reset after each sequence by using a stateful LSTM and calling reset_states()

Felix May 1, 2018 at 10:45 pm #

REPLY

Your posts have helped me a lot so I want to say thank you! However I have two questions. I

am training my LSTM on the sequence x0, x1, x2 -> y0 = x3 and I am using stateless. The reason I use statess is because I want to reset the state after a sequence beacause the sequences are not dependant on each other. My first question is if I am thinking correctly about stateless?

Second question, the training batch size you define when building the LSTM, is that the batch size which the states reset? Or is it resetted after each sequence?

With other words, should I train with in my case batch size 3 or can I use any batch size?

Jason Brownlee May 2, 2018 at 5:42 am #

REPLY

Makes sense.

State is reset at the end of each batch.

You can have more control by using “stateful”, run the epoch yourself and call reset after each sample/sequence.

Hazem May 5, 2018 at 7:58 pm #

REPLY

Thank you very much for your efforts
How to set the volume batch_size with this model
The number of rows in a dataset of 10000 data
Number and features 12
11 input
1 output

net = tflearn.input_data([None, 11])
net = tflearn.embedding(net, input_dim=10000, output_dim=128)
net = tflearn.lstm(net, 128, dropout=0.8)
net = tflearn.fully_connected(net, 1, activation=’sigmoid’)
net = tflearn.regression(net, optimizer=’adam’, learning_rate=0.001,
loss=’binary_crossentropy’)

# Training
model = tflearn.DNN(net)

Jason Brownlee May 6, 2018 at 6:28 am #

REPLY

The batch size is the number of samples to be processed in an epoch before weights are updated.

You can learn more here:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-a-batch-and-an-epoch

It is common to set the batch size as a factor of the number of samples in the training dataset.

A popular value is 32, that appears to work well:
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Devarsh June 20, 2018 at 12:56 pm #

REPLY

Hi, i am currently building a model for time series forecastings using LSTM , is there a way to perform dynamic searches for best parameters, like gridsearch etc, we cant use grid search and etc, becaue the sequence is of importance. I would highly appreciate it if you could guide me towards the dynamic parameter tuning for LSTM time series model in python.

Jason Brownlee June 21, 2018 at 6:08 am #

REPLY

Generally LSTMs are poor at time series forecasting.

Nevertheless, I have tutorials on grid searching on the blog, for example:
https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/

Patrão August 27, 2018 at 1:47 am #

REPLY

Hi Jason

Thank you very much for putting so much time and effort to provide the knowledge for other developers to become awesome at Machine Learning.

You have many posts of using LSTM for time series- so why you say that “Generally LSTMs are poor at time series forecasting”?

Jason Brownlee August 27, 2018 at 6:13 am # REPLY

Experience shows this. You can get much better results using a CNN, CNN-LSTM or ConvLSTM on time series forecasting.

For univariate forecasting problems, all deep learning methods are out-performed by classical methods like SARIMA and ETS. Even naive methods.

I have book in the works to really make this concrete.

Allen Hu August 28, 2018 at 7:41 am #

Hi Jason,

Appreciate your tutor and reply.

“For univariate forecasting problems, all deep learning methods are out-performed by classical methods like SARIMA and ETS. Even naive methods.” — But how about multivariate forecasting problems? I mean, if I have several features (lag1 of multivariate time series), and I want to use them to predict one feature (a time series) among the features, is deep learning methods still not better than the SARIMA?

Actually, I’m a guy who studied ARIMA, VAR, ETS first, then came to deep learning. At the beginning of my deep learning study, I thought I found a new continent, but now I’m feeling sad.

Wish you have a nice day.

Best,

Allen

Jason Brownlee August 28, 2018 at 11:07 am #

I have found CNNs, CNN-LSTMs and ConvLSTMs can perform well for multivariate data.

Allen Hu August 29, 2018 at 6:58 am #

Hi Jason,

Appreciate your answer — “I have found CNNs, CNN-LSTMs and ConvLSTMs can perform well for multivariate data.”

Moreover, could I ask, how are they performed for multivariate data compare to VAR(vector autoregression)? Thank you.

By the way, when you replied Patrão — “I have book in the works to really make this concrete.”, which book are you referring? I already bought your 12 books yesterday, so I can found CNN-LSTMs in book “long_short_term_memory_networks_with_python”, but I can’t find ConvLSTMs and the applications on multivariate time series data by using CNNs, CNN-LSTMs, and ConvLSTMs in this book. Also, when I take a quick review of the book “time_series_forecasting_with_python”, I can’t find any deep learning methods on multivariate time series because this book is all about classical time series methods. I’m not making a complaint here, but I really need someone to guide me to make my hands dirty onmultivariate time series data by leveraging CNNs, CNN-LSTMs, and ConvLSTMs methods. Thank you!

Best,

Allen

Jason Brownlee August 29, 2018 at 8:17 am #

It really depends on the problem. Sometime better.

A new book titled “Deep Learning for Time Series Forecasting”. It will be released very soon.

Allen Hu August 29, 2018 at 1:35 pm #

Hi Jason,

Thank you!

WOW, I can’t wait to read it! Hope it will release soon.

Best,

Allen

Jason Brownlee August 30, 2018 at 6:23 am #

Here's a link (e.g. a quite launch):
https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/

Jamy June 22, 2019 at 12:42 am #

Hi, Jason. Do you have some examples of using a CNN, CNN-LSTM or ConvLSTM on time series forecasting for univariate forecasting problems? Thank you so much!

Jason Brownlee June 22, 2019 at 6:45 am #

Yes, many examples. Start here:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

And here:
https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/

georgesun88 August 21, 2018 at 12:21 am #

REPLY

Hi Jason.
Many thanks for your great tutorials.
Could you please let me know if you have an example using Scikit-Optimize (skopt) to tune the hyper-parameters with the dataset?

Jason Brownlee August 21, 2018 at 6:17 am #

REPLY

Not at this stage. What do you need help with exactly?

Rachael August 31, 2018 at 7:08 pm #

REPLY

Hi Jason,

I have a question about the correct time to scale data. In your LSTM book, you state “It would not be appropriate to scale the series after it has been transformed into a supervised learning problem as each series would be handled differently, which would be incorrect.” However, I see in this example, that the scaling occurs after the data has been transformed to a supervised learning problem. In this case, does the timing not matter? If so, would you please explain why?

Thanks!

Jason Brownlee September 1, 2018 at 6:17 am #

REPLY

Good question, I discuss the order of transforms in tis recent post:
https://machinelearningmastery.com/machine-learning-data-transforms-for-time-series-forecasting/

Sivaram September 13, 2018 at 10:42 am #

REPLY

Thank you Jason for the great post.

In my use case of LSTM, I am observing something unusual. It intuitively makes sense to have the train RMSE to be less than the test RMSE. But in my case, I have the train RMSE to be greater than the test RMSE.

For instance,
After running for 4 epochs
Train RMSE: 0.097 Test RMSE: 0.046

The split of my training and testing data is 2/3 and 1/3.
I can think of the following cases why this is happening

a) The test cases are very similar to the train cases, therefore the prediction accuracy is high
b) The number of test cases are not enough.

What are your thoughts on this? Do you think it is a problem?

Sivaram

Jason Brownlee September 13, 2018 at 1:59 pm #

REPLY

I have seen this:
https://machinelearningmastery.com/faq/single-faq/what-if-model-skill-on-the-test-dataset-is-better-than-the-training-dataset

Jack September 18, 2018 at 1:12 pm #

REPLY

Hi Jason,
Thank you very much for this tutorial. I have one question and that is how we can evaluate the model performance since we can't do a k-fold cross validation.Is it OK just testing it on the test dataset?

Jason Brownlee September 18, 2018 at 2:25 pm #

REPLY

Use walk-forward validation:
https://machinelearningmastery.com/faq/single-faq/how-do-i-use-cross-validation-for-time-series-forecasting

Adam October 17, 2018 at 10:37 pm #

REPLY

This post has been very helpful! I do have a question. I am attempting to adapt what you have here to a different dataset, one that does not have much of a discernible trend. I have that, in other algorithms with my dataset, that taking the log10 is the only transformation i need to apply to get good output. I am wondering how i might configure the

```
for i in range(len(output)):
        yhat = output[i,0]
        # invert scaling
        yhat = invert_scale(scaler, X[i], yhat)

        #invert differencing
        yhat = yhat + raw_data[i]
        # store forecast
        predictions.append(yhat)
```

in the evaluate formula to work for inverting the log10, rather than the difference of yhat and raw_data[i]? Or is taking the log unnecessary for this type of model?

thanks!

Jason Brownlee October 18, 2018 at 6:30 am #

REPLY

A log10 can be inverted by raising 10 to the power of the result, for example:

x = log10(value)
value = 10^x

Does that help?

Adam October 17, 2018 at 11:17 pm #

REPLY

Nevermind! ignore my last question. I figured it out.

Thanks!

Cecilia November 9, 2018 at 7:22 am #

REPLY

Hi Jason!

Thanks for all the tutorials, they have been very helpful!

However, there's still something I don't quite get about this piece of code:

```
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), st
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model
```

Which one is the part in which we are fitting the model? Is it:
1)LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True)
or 2) model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)?

I'm confused because in the second part is looks like we are setting the epochs to 1. Can you explain what we are doing there?

Thanks in advance!

REPLY

Jason Brownlee November 9, 2018 at 2:00 pm #

The calls to model.fit() update the weights of the model.

Each call fits one epoch and we enumerate this nb_epoch times, e.g. we run the epochs manually.

REPLY

Rajesh January 11, 2019 at 5:54 pm #

Dear Jason

I'm having some trouble understanding the code in this bit of the script. I'm trying to understand each part of your entire code very carefully, but for some reason, I have a hard time visualizing this part. I apologize if this is a trivial question:

a) What exactly is train_scaled[2:, :]?
b) at the end of the lst.fit part why do we specify the number 1? what does that refer to?

code:

```
for r in range(repeats):
# fit the model
train_trimmed = train_scaled[2:, :]
lstm_model = fit_lstm(train_trimmed, batch_size, epochs, 1)
```

Thank you,
Rajesh

REPLY

Jason Brownlee January 12, 2019 at 5:39 am #

train_trimmed is the scaled training data.

I don't understand your second question sorry, can you please elaborate?

REPLY

Kiran February 6, 2019 at 4:55 pm #

Hello,

Your tutorials are really great.

I am new to RNN and LSTM. I am solving a problem (supervised regression), which predicts a parameter using 23 features. Please suggest a starter numbers for input/output neurons, layers, activation fn, loss, optimizer, epochs.

Thank you

Jason Brownlee February 7, 2019 at 6:36 am #

REPLY

Perhaps start here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Rajesh February 7, 2019 at 4:36 am #

REPLY

Dear Jason,
I figured out my other questions, but found another one:

What does this do?

forecast the entire training dataset to build up state for forecasting
…
lstm_model.predict(train_reshaped, batch_size=batch_size)

I'm not quite sure why you would be forecasting training data (since you're using it to train your model) – why not just forecast the test data directly? What is the thought process behind this?

Thank you,
Rajesh

Jason Brownlee February 7, 2019 at 6:44 am #

REPLY

It is an attempt to build up internal state before forecasting. I don't think it is required.

Rajesh February 7, 2019 at 1:48 pm #

REPLY

Thank you Jason :). I assumed it wasn't required, but needed to maker sure.
With your tutorials I've managed to build a number of lstm's with multiple input, multiple output, multiple lag timesteps, and multiple predictions into the future. All differenced, hot-encoded when needed, and scaled. I'm now just working on creating a harness that can systemically search out different parameter and hyperparameter configurations. I've decided to buy one of your books to show my appreciation. Any suggestions as to a book with lots of project examples (especially similar to problems found in meteorological study)?

Thank you,
Rajesh

Jason Brownlee February 7, 2019 at 2:08 pm #

REPLY

Thanks for your support!

Perhaps this would be the best fit:
https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/

Rajesh February 8, 2019 at 12:31 am #

REPLY

Hello Jason,
done! Looks like a great book. I'm going to go through it linearly like you suggest in your preface. You mention that it might be ok to shoot you an email regarding some specific questions (if they're not answered by checking them out in the materials mentioned in the further reading sections) – I wouldn't want to bother you too much, however. Is there perhaps an online message board associated with each book?

Thank you,
Rajesh

Jason Brownlee February 8, 2019 at 7:53 am #

REPLY

No message board at this stage, email is fine.

Muhammad March 3, 2019 at 4:10 pm #

REPLY

Hi Jason,

Right now I am trying to implement the LSTM algorithm to a different time-series data. I have tried to build the model using R instead of python. Although, I came across to a tutorial (https://www.r-bloggers.com/time-series-deep-learning-forecasting-sunspots-with-keras-stateful-lstm-in-r/) where it suggested numbers of data exploration, one of it is ACF and PACF review.
It does not seem you have done the same ACF and PACF analysis prior to constructing the model. Have you confirmed it in advance that the data is suitable for LSTM model? Or is there another way to construct the model even though the data does not in fulfill the ACF criteria?

Thank you,
Muhammad

Jason Brownlee March 4, 2019 at 6:57 am #

REPLY

I recommend this process:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

zhangzhe March 6, 2019 at 8:18 pm #

REPLY

Hi Jason,

I'm a little confused. In the Summary of Results section, I know that this code is not required. But I wonder how important is the state of the network to the later prediction?

Lstm_model. Predict (train_reshaped, batch_size = batch_size)

In keras, LSTM is trained with a certain batch_size, so I have to input batch_size in the end when I predict? For example, if the state stored by the network is batch_size, what will happen if the data I input is not that size?

Jason Brownlee March 7, 2019 at 6:47 am #

REPLY

Much less than expected. I even have a post somewhere that tests this and finds it has no effect.

Yes, with stateful LSTM, the batch size must be the same, although I have a work around here:
https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/

Aaron March 18, 2019 at 1:56 pm #

REPLY

Hi Jason,

Got a few questions about your choice of data preparation methodology –

1- What is the purpose of the lag? Also, why did you choose differencing to remove the trend? Ultimately, are you trying to find the difference between two days? How would you convert the difference back into un-differenced values?
2- The supervised learning aspect, gives you 2 columns of data where the value in the first column is supposed to be the observation from the previous time step to be used as input to find the value in the second column?

Thanks

Jason Brownlee March 18, 2019 at 2:12 pm #

REPLY

Good questions, the answers are all on the blog.

More on the representation of a time series problem as a supervised learning problem here:
https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

Specifics on time series representation with lstms here:
https://machinelearningmastery.com/prepare-univariate-time-series-data-long-short-term-memory-networks/

More on differencing here:
https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

Aaron March 19, 2019 at 5:39 am #

REPLY

Thanks Jason – again appreciate all the guidance.

Read those articles, and definitely making a clearer picture of how my LSTM will be setup.

A quick follow up on the difference transformations – using t-1 -> t-48 to predict t -> t+12 I feel that for my data certain trends and time of year are important to the output, and so will have day of the year, and day of the week as part of my feature set. Since this is a univariate prediction problem I only need to predict the value, but utilizing multiple features at every one of the

previous timesteps. The data is non-linear and generally increasing over time, but not always.

Every feature needs to be normalized according to that specific feature, correct? (Getting somewhat side-tracked here but…) For day of the year, I would use min-max scaler for between -1 and 1 does that make any sense to do?

Ok sorry, back to my original thought, if I am predicting a sequence, how would the inverse difference work if the previous timestep value would be that of a predicted value? My thinking was to just do min-max scaler for each sample set, or is that overkill?

Thanks again

Jason Brownlee March 19, 2019 at 9:02 am #

REPLY

The difference is inverted using the last known observation then propagated down through all predictions.

Aaron March 19, 2019 at 11:11 pm #

Ahh – ok, so using the predicted values to calculate the difference going forward. Next I have to figure out whether or not I am actually going to use an encoder/decoder model and which loss functions and optimizers to use

Aaron March 19, 2019 at 11:34 pm #

Oh sorry – one more question on the difference transformation – the network is trained on the inverse difference values correct?

So –
– Difference input data
– Scale input data
– Make prediction
– Unscale output data
– Inverse difference of unscaled output data
– use RMSE to determine the error
– Train network based on error

Or is the Keras LSTM handling the optimization and we are simply unscaling and

running the inverse difference to get the output, then using RMSE to determine how accurate the network is?

Jason Brownlee March 20, 2019 at 8:31 am #

Looks good, also try with/without each transform to see if they are required.

More on the order of transforms for time series here:
https://machinelearningmastery.com/machine-learning-data-transforms-for-time-series-forecasting/

lingyaw April 8, 2019 at 9:47 pm #

REPLY

Hi Jason,

For Shampoo sales, you have only one sample. I remember you said that ” if there is one sample and the batch size is > 1, The batch size will be 1″.

Why the batch size is 2, 4 here when only one sample (24 observations) is available.

When batch_size is four, does it mean that four same samples (training examples)will be propagated into the network ? Since one epoch = one forward pass and one backward pass of all the training examples. Weights parameters will be updated four times in one batch ???

Luke April 22, 2019 at 5:55 am #

REPLY

Hi Jason,

I have enjoyed your blog posts, thank you for all the hard work. As my program is building it occurred to me that we are isolating each tuning variable and then assessing the performance based on static values for the other values.
So my questions are:

1) Is there an order which in tuning these isolated variables? Such as, tune batch size first since it is dependent on being a divisor of test/train data. Then train the number of epochs (for whichever reason), then train the number of neurons? This is the intuitive/manual approach without destroying my computational time.

2) I want to eventually create an automated script to optimize the tuning parameters for me, but I want

to figure out a way that doesn't take days to run due to computational limitations. (I'm trying to avoid Grid searching: analyze all possibilities of X, given possibilities Y/Z; then analyze all possibilities of Y, given possibilities of X/Z; then analyze all possibilities of Z, given variables of Y/X). My first thought is to try random starting values and then apply Newton's Min/Max finding? Any method that has worked well for you to date?

Either way, I will continue to test the suggested tuning changes and start working on the tuning automation. I'll let you know of my results once I get them.

All the best,
-Luke

Jason Brownlee April 22, 2019 at 6:26 am #

REPLY

Yes, probably model capacity perhaps with early stopping then learning rate.

More here:
https://machinelearningmastery.com/start-here/#better

Luke April 23, 2019 at 10:59 am #

REPLY

Thank you! I'll take a look

Jieyu Chen May 2, 2019 at 12:23 am #

REPLY

n_neurons is the parameter units???

Jason Brownlee May 2, 2019 at 8:03 am #

REPLY

Yes.

Anas Aburawwash May 16, 2019 at 11:55 pm #

REPLY

Hey Jason,

Thank you so much for this useful article.

I wounder if I could apply the concept of LSTM between the output layer units. To explain more, I am predicting the time series based on properties of the system.

Jason Brownlee May 17, 2019 at 5:55 am #

REPLY

I believe you are referring to a multi-step forecasting problem, perhaps this will help:
https://machinelearningmastery.com/faq/single-faq/how-do-you-use-lstms-for-multi-step-time-series-forecasting

Andre Araujo July 7, 2019 at 5:50 am #

REPLY

Hi Jason,

Do you believe that assume that you have the best features to train your model (time-series). Is it good approach build a model that stop only when all instances reach a max error?

Here is a approach wit forward neural network:
https://colab.research.google.com/drive/1h0hIscYRVlCBFdTHXUfNhnq_eAA3Q_qA

In LSTM is possible stop early when the error stop to decrease?

Jason Brownlee July 7, 2019 at 7:56 am #

REPLY

LSTMs are pretty terrible at time series forecasting, see this:
https://machinelearningmastery.com/findings-comparing-classical-and-machine-learning-methods-for-time-series-forecasting/

Lopa August 15, 2019 at 4:29 am #

REPLY

Hi Jason,

Thanks for this useful material. Can you please tell me why you have used train_trimmed (it is the scaled data)

Jason Brownlee August 15, 2019 at 8:27 am #

REPLY

I may have removed some unusable data/NaNs.

Lopamudra August 15, 2019 at 4:39 am #

REPLY

I think it's because the data is differenced

JustAddMoreLayers August 23, 2019 at 4:37 pm #

REPLY

Is there any technical reason to refer to the first input of the LSTM layer as "neurons" instead of "units"? Per the Keras documentation: units: Positive integer, dimensionality of the output space. Is it just a matter of taste?

Jason Brownlee August 24, 2019 at 7:45 am #

REPLY

Nope, habit. I've been hacking on neural nets for more than two decades and we always used to call them neurons regardless of the network type.

They're units.

Carol October 2, 2019 at 1:00 am #

REPLY

Hi Jason,
you only tunning one value of batch_size because you divide your sample equally between train and test. Correct?

But what happens if you have different lenghts for test and training sample?

Do I have to tune both of them?

Jason Brownlee October 2, 2019 at 8:01 am #

REPLY

The model will require each sample is the same length, but the number of samples can vary between datasets.

Arjun November 14, 2019 at 10:54 pm #

REPLY

I have a out of context question. Is data extraction from scratch a job for data analysts or is it done by us developers itself? Preprocessing is not a big issue but formation of data seems like a tedious task.

Jason Brownlee November 15, 2019 at 7:52 am #

REPLY

Yes, collecting data is challenging. It might even be harder than modeling these days.

Arjun November 15, 2019 at 3:03 pm #

REPLY

yes, but I am asking is it a job for data analysts or developers only?

Jason Brownlee November 16, 2019 at 7:18 am #

REPLY

Either, both, neither. It depends on the company and the developer and the data analyst.

Radhouane Baba November 20, 2019 at 12:20 am #

REPLY

can we also change the input so that it is no more sequential?
for example: the input of a wednesday will have: (3 last wednesdays + last week)

Jason Brownlee November 20, 2019 at 6:18 am #

REPLY

Sure, you can define the input any way you wish.

Arjun November 20, 2019 at 3:25 pm #

REPLY

```
def createDelayedColumns(series, times):
    cols = []
    column_index = []
    for time in times:
        cols.append(series.shift(-time))
        lag_fmt = "t+{time}" if time > 0 else "t{time}" if time < 0 else "t"
        column_index += [(lag_fmt.format(time=time), col_name)
                         for col_name in series.columns]
    df = pd.concat(cols, axis=1)
    df.columns = pd.MultiIndex.from_tuples(column_index)
    return df
```

what is happening with "t +{time}"?
what is the purpose of this snippet?

Jason Brownlee November 21, 2019 at 6:01 am #

REPLY

I don't think I wrote that – and its not from this post.

Perhaps ask the author?

Qizal Ashfaq January 12, 2020 at 1:10 am #

REPLY

how to make multivariate time series data stationary?only need to make outputs stationary or whole dataset should be stationary ?need some tutorial for this.

Jason Brownlee January 12, 2020 at 8:05 am #

REPLY

Each series is made stationary separately.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome!
My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.
Read more

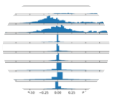
Never miss a tutorial:

Picked for you:

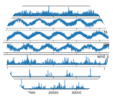
How to Develop LSTM Models for Time Series Forecasting

How to Develop Convolutional Neural Network Models for Time Series Forecasting

How to Develop Multi-Step LSTM Time Series Forecasting Models for Power Usage



How to Develop 1D Convolutional Neural Network Models for Human Activity Recognition



Multivariate Time Series Forecasting with LSTMs in Keras