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

Multi-step Time Series Forecasting with Long Short-Term Memory Networks in Python

by **Jason Brownlee** on May 10, 2017 in **Long Short-Term Memory Networks**

The Long Short-Term Memory network or LSTM is a recurrent neural network that can learn and forecast long sequences.

A benefit of LSTMs in addition to learning long sequences is that they can learn to make a one-shot multi-step forecast which may be useful for time series forecasting.

A difficulty with LSTMs is that they can be tricky to configure and it can require a lot of preparation to get the data in the right format for learning.

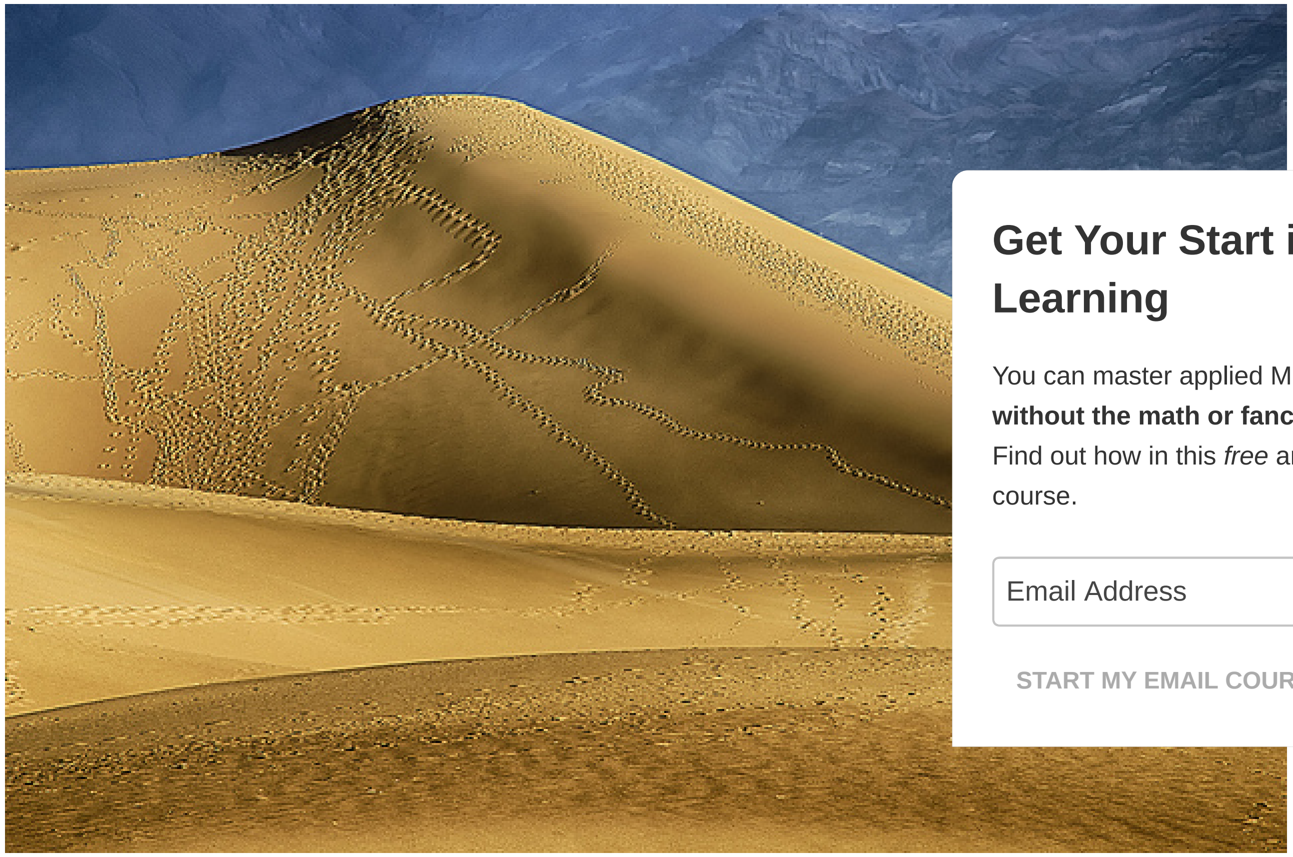
In this tutorial, you will discover how you can develop an LSTM for multi-step time series forecasting

[redacted]

After completing this tutorial, you will know:

- How to prepare data for multi-step time series forecasting.
- How to develop an LSTM model for multi-step time series forecasting.
- How to evaluate a multi-step time series forecast.

Let’s get started.



Multi-step Time Series Forecasting with Long Short-Term Memory Networks in Python
Photo by Tom Babich, some rights reserved.

Tutorial Overview

[redacted]

This tutorial is broken down into 4 parts; they are:

1. Shampoo Sales Dataset
2. Data Preparation and Model Evaluation
3. Persistence Model
4. Multi-Step LSTM

Environment

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

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

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

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

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

Next, let's take a look at a standard time series forecasting problem that we can use as context for th

Shampoo Sales Dataset

[redacted]

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).

You can download and learn more about the dataset here.

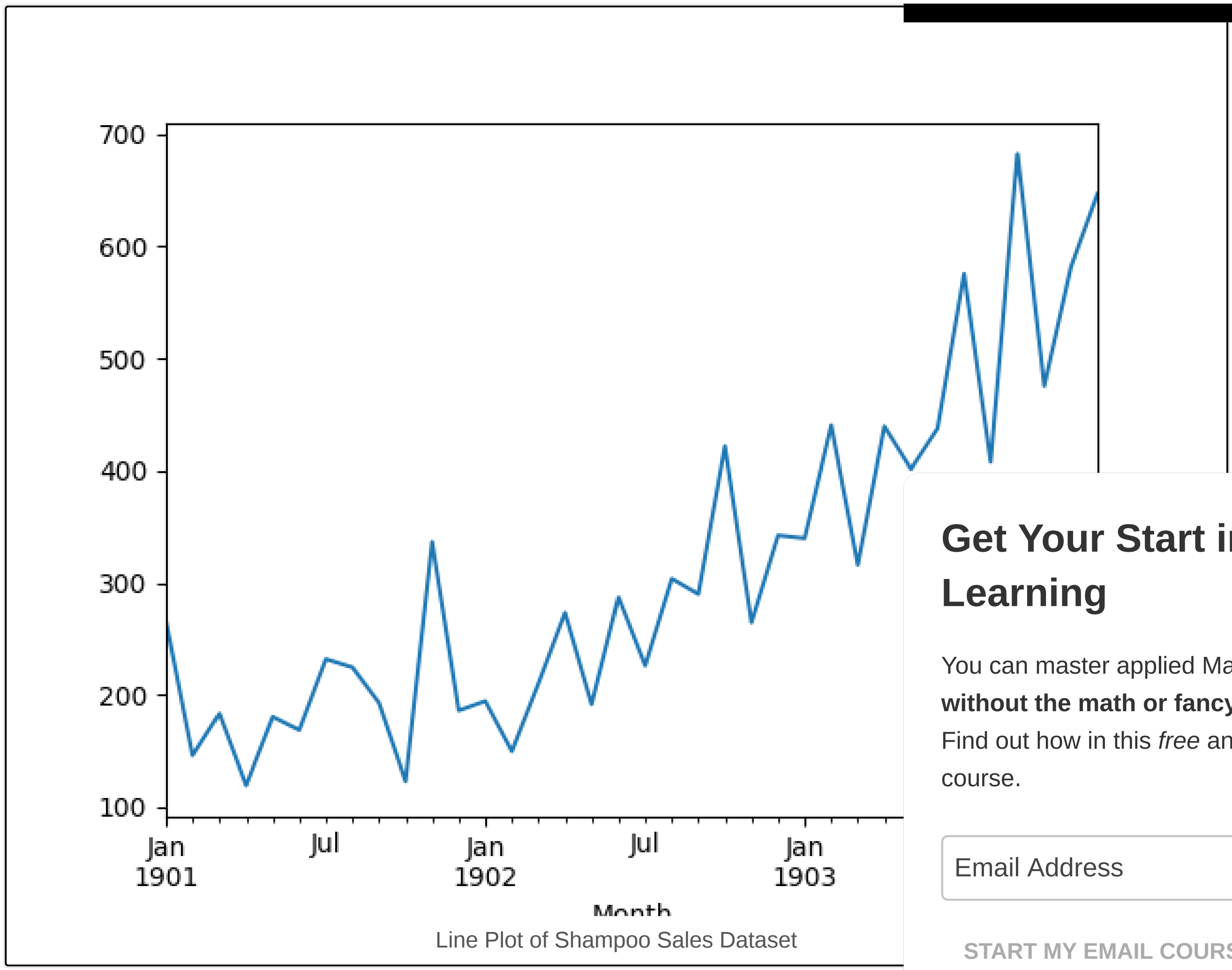
The example below loads and creates a plot of the loaded dataset.

```
# load and plot dataset
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
# load dataset
def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=T
# 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 model configuration and test harness used in the experiment.

Data Preparation and Model Evaluation

This section describes data preparation and model evaluation used in this tutorial

Data Split

[redacted]

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.

For reference, the last 12 months of observations are as follows:

```
"3-01",339.7
"3-02",440.4
"3-03",315.9
"3-04",439.3
"3-05",401.3
"3-06",437.4
"3-07",575.5
"3-08",407.6
"3-09",682.0
"3-10",475.3
"3-11",581.3
"3-12",646.9
```

Multi-Step Forecast

We will contrive a multi-step forecast.

For a given month in the final 12 months of the dataset, we will be required to make a 3-month forec

That is given historical observations (t-1, t-2, … t-n) forecast t, t+1 and t+2.

Specifically, from December in year 2, we must forecast January, February and March. From Januar the way to an October, November, December forecast from September in year 3.

A total of 10 3-month forecasts are required, as follows:

```
Dec,    Jan, Feb, Mar
Jan,    Feb, Mar, Apr
Feb,    Mar, Apr, May
Mar,    Apr, May, Jun
Apr,    May, Jun, Jul
May,    Jun, Jul, Aug
```

```
Jun,    Jul, Aug, Sep
Jul,    Aug, Sep, Oct
Aug,    Sep, Oct, Nov
Sep,    Oct, Nov, Dec
```

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 for the next month 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.

All forecasts on the test dataset will be collected and an error score calculated to summarize the skil The root mean squared error (RMSE) will be used as it punishes large errors and results in a score that i monthly shampoo sales.

Persistence Model

A good baseline for time series forecasting is the persistence model.

This is a forecasting model where the last observation is persisted forward. Because of its simplicity,

You can learn more about the persistence model for time series forecasting in the post:

- How to Make Baseline Predictions for Time Series Forecasting with Python

Prepare Data

The first step is to transform the data from a series into a supervised learning problem.

That is to go from a list of numbers to a list of input and output patterns. We can achieve this using a *d().*

For more on this function, see the post:

[redacted]

- How to Convert a Time Series to a Supervised Learning Problem in Python

The function is listed below.

```python
# convert time series into supervised learning problem
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

The function can be called by passing in the loaded series values an *n_in* value of 1 and an n_out va

```python
supervised = series_to_supervised(raw_values, 1, 3)
```

Next, we can split the supervised learning dataset into training and test sets.

We know that in this form, the last 10 rows contain data for the final year. These rows comprise the test set and the rest of the data makes up the training dataset.

We can put all of this together in a new function that takes the loaded series and some parameters and returns a train and test set ready for modeling.

```python
# transform series into train and test sets for supervised learning
```

```
def prepare_data(series, n_test, n_lag, n_seq):
    # extract raw values
    raw_values = series.values
    raw_values = raw_values.reshape(len(raw_values), 1)
    # transform into supervised learning problem X, y
    supervised = series_to_supervised(raw_values, n_lag, n_seq)
    supervised_values = supervised.values
    # split into train and test sets
    train, test = supervised_values[0:-n_test], supervised_values[-n_test:]
    return train, test
```

We can test this with the Shampoo dataset. The complete example is listed below.

```
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from pandas import datetime

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

# convert time series into supervised learning problem
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

# transform series into train and test sets for supervised learning
```

[redacted]

```
def prepare_data(series, n_test, n_lag, n_seq):
    # extract raw values
    raw_values = series.values
    raw_values = raw_values.reshape(len(raw_values), 1)
    # transform into supervised learning problem X, y
    supervised = series_to_supervised(raw_values, n_lag, n_seq)
    supervised_values = supervised.values
    # split into train and test sets
    train, test = supervised_values[0:-n_test], supervised_values[-n_test:]
    return train, test

# load dataset
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
# configure
n_lag = 1
n_seq = 3
n_test = 10
# prepare data
train, test = prepare_data(series, n_test, n_lag, n_seq)
print(test)
print('Train: %s, Test: %s' % (train.shape, test.shape))
```

Running the example first prints the entire test dataset, which is the last 10 rows. The shape and siz

```
[[ 342.3  339.7  440.4  315.9]
 [ 339.7  440.4  315.9  439.3]
 [ 440.4  315.9  439.3  401.3]
 [ 315.9  439.3  401.3  437.4]
 [ 439.3  401.3  437.4  575.5]
 [ 401.3  437.4  575.5  407.6]
 [ 437.4  575.5  407.6  682. ]
 [ 575.5  407.6  682.   475.3]
 [ 407.6  682.   475.3  581.3]
 [ 682.   475.3  581.3  646.9]]
Train: (23, 4), Test: (10, 4)
```

We can see the single input value (first column) on the first row of the test dataset matches the observation in the shampoo-sales for December in the 2nd year:

```
"2-12",342.3
```

We can also see that each row contains 4 columns for the 1 input and 3 output values in each observation.

Make Forecasts

[redacted]

The next step is to make persistence forecasts.

We can implement the persistence forecast easily in a function named *persistence()* that takes the last observation and the number of forecast steps to persist. This function returns an array containing the forecast.

```
# make a persistence forecast
def persistence(last_ob, n_seq):
    return [last_ob for i in range(n_seq)]
```

We can then call this function for each time step in the test dataset from December in year 2 to September in year 3.

Below is a function *make_forecasts()* that does this and takes the train, test, and configuration for the dataset as arguments and returns a list of forecasts.

```
# evaluate the persistence model
def make_forecasts(train, test, n_lag, n_seq):
    forecasts = list()
    for i in range(len(test)):
        X, y = test[i, 0:n_lag], test[i, n_lag:]
        # make forecast
        forecast = persistence(X[-1], n_seq)
        # store the forecast
        forecasts.append(forecast)
    return forecasts
```

We can call this function as follows:

```
forecasts = make_forecasts(train, test, 1, 3)
```

Evaluate Forecasts

The final step is to evaluate the forecasts.

We can do that by calculating the RMSE for each time step of the multi-step forecast, in this case giving us 3 RMSE scores. The function below, *evaluate_forecasts()*, calculates and prints the RMSE for each forecasted time step.

```
# evaluate the RMSE for each forecast time step
def evaluate_forecasts(test, forecasts, n_lag, n_seq):
    for i in range(n_seq):
        actual = test[:,(n_lag+i)]
        predicted = [forecast[i] for forecast in forecasts]
        rmse = sqrt(mean_squared_error(actual, predicted))
        print('t+%d RMSE: %f' % ((i+1), rmse))
```

[redacted]

We can call it as follows:

```
evaluate_forecasts(test, forecasts, 1, 3)
```

It is also helpful to plot the forecasts in the context of the original dataset to get an idea of how the RMSE scores relate to the problem in context.

We can first plot the entire Shampoo dataset, then plot each forecast as a red line. The function *plot_forecasts()* below will create and show this plot.

```
# plot the forecasts in the context of the original dataset
def plot_forecasts(series, forecasts, n_test):
    # plot the entire dataset in blue
    pyplot.plot(series.values)
    # plot the forecasts in red
    for i in range(len(forecasts)):
        off_s = len(series) - n_test + i
        off_e = off_s + len(forecasts[i])
        xaxis = [x for x in range(off_s, off_e)]
        pyplot.plot(xaxis, forecasts[i], color='red')
    # show the plot
    pyplot.show()
```

We can call the function as follows. Note that the number of observations held back on the test set is 0
supervised learning input/output patterns as was used above.

```
# plot forecasts
plot_forecasts(series, forecasts, 12)
```

We can make the plot better by connecting the persisted forecast to the actual persisted value in the

This will require adding the last observed value to the front of the forecast. Below is an updated version of the *plot_forecasts()* function with this improvement.

```
# plot the forecasts in the context of the original dataset
def plot_forecasts(series, forecasts, n_test):
    # plot the entire dataset in blue
    pyplot.plot(series.values)
    # plot the forecasts in red
    for i in range(len(forecasts)):
        off_s = len(series) - 12 + i - 1
        off_e = off_s + len(forecasts[i]) + 1
        xaxis = [x for x in range(off_s, off_e)]
        yaxis = [series.values[off_s]] + forecasts[i]
        pyplot.plot(xaxis, yaxis, color='red')
    # show the plot
    pyplot.show()
```

Complete Example

We can put all of these pieces together.

The complete code example for the multi-step persistence forecast is listed below.

```
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from math import sqrt
from matplotlib import pyplot

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

# convert time series into supervised learning problem
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
```

```python
    if dropnan:
        agg.dropna(inplace=True)
    return agg

# transform series into train and test sets for supervised learning
def prepare_data(series, n_test, n_lag, n_seq):
    # extract raw values
    raw_values = series.values
    raw_values = raw_values.reshape(len(raw_values), 1)
    # transform into supervised learning problem X, y
    supervised = series_to_supervised(raw_values, n_lag, n_seq)
    supervised_values = supervised.values
    # split into train and test sets
    train, test = supervised_values[0:-n_test], supervised_values[-n_test:]
    return train, test

# make a persistence forecast
def persistence(last_ob, n_seq):
    return [last_ob for i in range(n_seq)]

# evaluate the persistence model
def make_forecasts(train, test, n_lag, n_seq):
    forecasts = list()
    for i in range(len(test)):
        X, y = test[i, 0:n_lag], test[i, n_lag:]
        # make forecast
        forecast = persistence(X[-1], n_seq)
        # store the forecast
        forecasts.append(forecast)
    return forecasts

# evaluate the RMSE for each forecast time step
def evaluate_forecasts(test, forecasts, n_lag, n_seq):
    for i in range(n_seq):
        actual = test[:,(n_lag+i)]
        predicted = [forecast[i] for forecast in forecasts]
        rmse = sqrt(mean_squared_error(actual, predicted))
        print('t+%d RMSE: %f' % ((i+1), rmse))

# plot the forecasts in the context of the original dataset
def plot_forecasts(series, forecasts, n_test):
    # plot the entire dataset in blue
    pyplot.plot(series.values)
    # plot the forecasts in red
    for i in range(len(forecasts)):
        off_s = len(series) - n_test + i - 1
        off_e = off_s + len(forecasts[i]) + 1
        xaxis = [x for x in range(off_s, off_e)]
        yaxis = [series.values[off_s]] + forecasts[i]
        pyplot.plot(xaxis, yaxis, color='red')
    # show the plot
    pyplot.show()

# load dataset
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
# configure
n_lag = 1
n_seq = 3
n_test = 10
# prepare data
train, test = prepare_data(series, n_test, n_lag, n_seq)
# make forecasts
forecasts = make_forecasts(train, test, n_lag, n_seq)
# evaluate forecasts
evaluate_forecasts(test, forecasts, n_lag, n_seq)
# plot forecasts
plot_forecasts(series, forecasts, n_test+2)
```

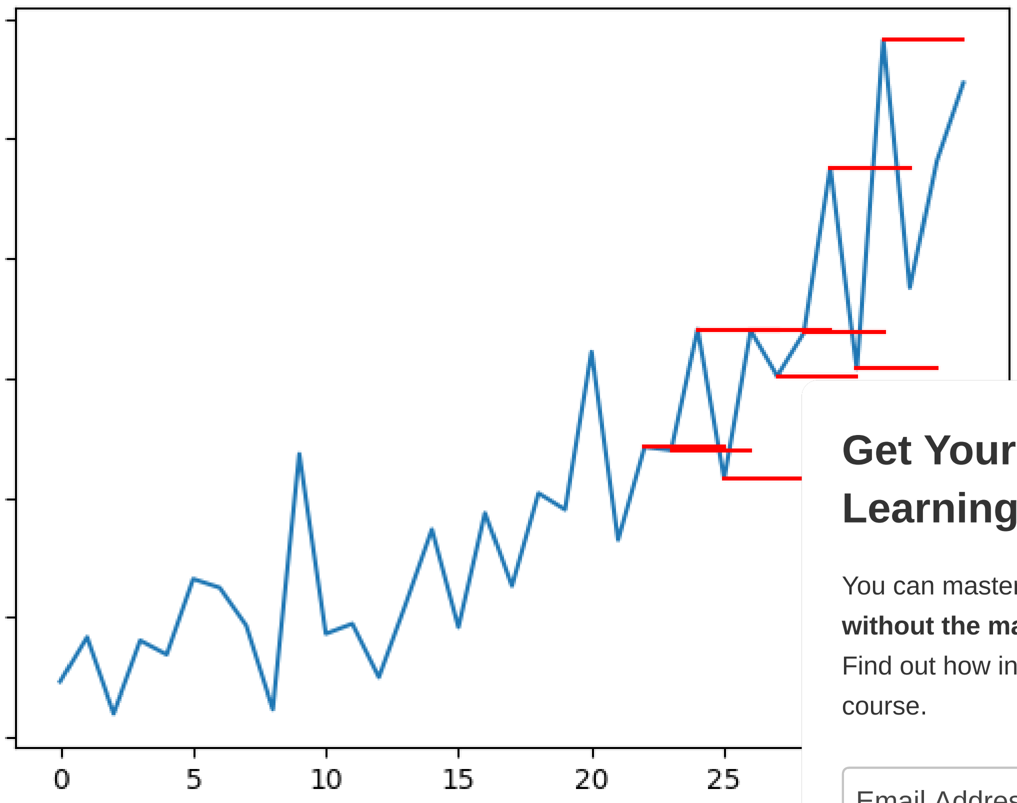
Running the example first prints the RMSE for each of the forecasted time steps.

This gives us a baseline of performance on each time step that we would expect the LSTM to outper

```
t+1 RMSE: 144.535304
t+2 RMSE: 86.479905
t+3 RMSE: 121.149168
```

The plot of the original time series with the multi-step persistence forecasts is also created. The lines
forecast.

This context shows how naive the persistence forecasts actually are.



Line Plot of Shampoo Sales Dataset with Multi-Step Persistence F

Multi-Step LSTM Network

In this section, we will use the persistence example as a starting point and look at the changes needed to fit an LSTM to the training data and make multi-step forecasts for the test dataset.

Prepare Data

The data must be prepared before we can use it to train an LSTM.

Specifically, two additional changes are required:

1. **Stationary**. The data shows an increasing trend that must be removed by differencing.
2. **Scale**. The scale of the data must be reduced to values between -1 and 1, the activation function of the LSTM units.

We can introduce a function to make the data stationary called *difference()*. This will transform the series of values into a series of differences, a simpler representation to work with.

```
# 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)
```

We can use the *MinMaxScaler* from the sklearn library to scale the data.

Putting this together, we can update the *prepare_data()* function to first difference the data and resca
learning problem and train test sets as we did before with the persistence example.

The function now returns a scaler in addition to the train and test datasets.

```
# transform series into train and test sets for supervised learning
def prepare_data(series, n_test, n_lag, n_seq):
    # extract raw values
    raw_values = series.values
    # transform data to be stationary
    diff_series = difference(raw_values, 1)
    diff_values = diff_series.values
    diff_values = diff_values.reshape(len(diff_values), 1)
    # rescale values to -1, 1
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaled_values = scaler.fit_transform(diff_values)
    scaled_values = scaled_values.reshape(len(scaled_values), 1)
    # transform into supervised learning problem X, y
    supervised = series_to_supervised(scaled_values, n_lag, n_seq)
    supervised_values = supervised.values
    # split into train and test sets
    train, test = supervised_values[0:-n_test], supervised_values[-n_test:]
    return scaler, train, test
```

We can call this function as follows:

[redacted]

```
# prepare data
scaler, train, test = prepare_data(series, n_test, n_lag, n_seq)
```

Fit LSTM Network

Next, we need to fit an LSTM network model to the training data.

This first requires that the training dataset be transformed from a 2D array [*samples, features*] to a 3D array [*samples, timesteps, features*]. We will fix time steps at 1, so this change is straightforward.

Next, we need to design an LSTM network. We will use a simple structure with 1 hidden layer with 1 LSTM unit, then an output layer with linear activation and 3 output values. The network will use a mean squared error loss function and the efficient ADAN

The LSTM is stateful; this means that we have to manually reset the state of the network at the end
1500 epochs.

The same batch size must be used for training and prediction, and we require predictions to be mad s
that a batch size of 1 must be used. A batch size of 1 is also called online learning as the network w
training pattern (as opposed to mini batch or batch updates).

We can put all of this together in a function called *fit_lstm()*. The function takes a number of key para
and the function returns a fit LSTM model ready for forecasting.

```
# fit an LSTM network to training data
def fit_lstm(train, n_lag, n_seq, n_batch, nb_epoch, n_neurons):
    # reshape training into [samples, timesteps, features]
    X, y = train[:, 0:n_lag], train[:, n_lag:]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    # design network
    model = Sequential()
    model.add(LSTM(n_neurons, batch_input_shape=(n_batch, X.shape[1], X.shape[2]), stateful=True))
    model.add(Dense(y.shape[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    # fit network
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=n_batch, verbose=0, shuffle=False)
        model.reset_states()
    return model
```

[redacted]

The function can be called as follows:

```
# fit model
model = fit_lstm(train, 1, 3, 1, 1500, 1)
```

The configuration of the network was not tuned; try different parameters if you like.

Report your findings in the comments below. I'd love to see what you can get.

Make LSTM Forecasts

The next step is to use the fit LSTM network to make forecasts.

A single forecast can be made with the fit LSTM network by calling *model.predict()*. Again, the data r [*samples, timesteps, features*].

We can wrap this up into a function called *forecast_lstm()*.

```
# make one forecast with an LSTM,
def forecast_lstm(model, X, n_batch):
    # reshape input pattern to [samples, timesteps, features]
    X = X.reshape(1, 1, len(X))
    # make forecast
    forecast = model.predict(X, batch_size=n_batch)
    # convert to array
    return [x for x in forecast[0, :]]
```

We can call this function from the *make_forecasts()* function and update it to accept the model as an

```
# evaluate the persistence model
def make_forecasts(model, n_batch, train, test, n_lag, n_seq):
    forecasts = list()
    for i in range(len(test)):
        X, y = test[i, 0:n_lag], test[i, n_lag:]
        # make forecast
        forecast = forecast_lstm(model, X, n_batch)
        # store the forecast
        forecasts.append(forecast)
    return forecasts
```

This updated version of the *make_forecasts()* function can be called as follows:

```
# make forecasts
forecasts = make_forecasts(model, 1, train, test, 1, 3)
```

Invert Transforms

After the forecasts have been made, we need to invert the transforms to return the values back into the original scale.

This is needed so that we can calculate error scores and plots that are comparable with other models, like the persistence forecast above.

We can invert the scale of the forecasts directly using the *MinMaxScaler* object that offers an *inverse_transform()* function.

We can invert the differencing by adding the value of the last observation (prior months' shampoo sa
the value down the forecast.

This is a little fiddly; we can wrap up the behavior in a function name *inverse_difference()* that takes ne
forecast as arguments and returns the inverted forecast.

```
# invert differenced forecast
def inverse_difference(last_ob, forecast):
    # invert first forecast
    inverted = list()
    inverted.append(forecast[0] + last_ob)
    # propagate difference forecast using inverted first value
    for i in range(1, len(forecast)):
        inverted.append(forecast[i] + inverted[i-1])
    return inverted
```

Putting this together, we can create an *inverse_transform()* function that works through each forecas first inverting the scale and then inverting the
differences, returning forecasts to their original scale.

```
# inverse data transform on forecasts
def inverse_transform(series, forecasts, scaler, n_test):
    inverted = list()
    for i in range(len(forecasts)):
        # create array from forecast
        forecast = array(forecasts[i])
        forecast = forecast.reshape(1, len(forecast))
        # invert scaling
        inv_scale = scaler.inverse_transform(forecast)
        inv_scale = inv_scale[0, :]
        # invert differencing
        index = len(series) - n_test + i - 1
        last_ob = series.values[index]
        inv_diff = inverse_difference(last_ob, inv_scale)
        # store
        inverted.append(inv_diff)
    return inverted
```

We can call this function with the forecasts as follows:

```
# inverse transform forecasts and test
forecasts = inverse_transform(series, forecasts, scaler, n_test+2)
```

We can also invert the transforms on the output part test dataset so that we can correctly calculate the RMSE scores, as follows:

```
actual = [row[n_lag:] for row in test]
actual = inverse_transform(series, actual, scaler, n_test+2)
```

We can also simplify the calculation of RMSE scores to expect the test data to only contain the outpu

```
def evaluate_forecasts(test, forecasts, n_lag, n_seq):
    for i in range(n_seq):
        actual = [row[i] for row in test]
        predicted = [forecast[i] for forecast in forecasts]
        rmse = sqrt(mean_squared_error(actual, predicted))
        print('t+%d RMSE: %f' % ((i+1), rmse))
```

Complete Example

We can tie all of these pieces together and fit an LSTM network to the multi-step time series forecas

The complete code listing is provided 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
from matplotlib import pyplot
from numpy import array

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

# convert time series into supervised learning problem
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

# 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)

# transform series into train and test sets for supervised learning
def prepare_data(series, n_test, n_lag, n_seq):
    # extract raw values
    raw_values = series.values
    # transform data to be stationary
    diff_series = difference(raw_values, 1)
    diff_values = diff_series.values
    diff_values = diff_values.reshape(len(diff_values), 1)
    # rescale values to -1, 1
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaled_values = scaler.fit_transform(diff_values)
    scaled_values = scaled_values.reshape(len(scaled_values), 1)
    # transform into supervised learning problem X, y
    supervised = series_to_supervised(scaled_values, n_lag, n_seq)
    supervised_values = supervised.values
    # split into train and test sets
    train, test = supervised_values[0:-n_test], supervised_values[-n_test:]
    return scaler, train, test

# fit an LSTM network to training data
def fit_lstm(train, n_lag, n_seq, n_batch, nb_epoch, n_neurons):
    # reshape training into [samples, timesteps, features]
    X, y = train[:, 0:n_lag], train[:, n_lag:]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    # design network
    model = Sequential()
    model.add(LSTM(n_neurons, batch_input_shape=(n_batch, X.shape[1], X.shape[2]), stat
    model.add(Dense(y.shape[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    # fit network
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=n_batch, verbose=0, shuffle=False)
        model.reset_states()
    return model

# make one forecast with an LSTM,
def forecast_lstm(model, X, n_batch):
    # reshape input pattern to [samples, timesteps, features]
    X = X.reshape(1, 1, len(X))
    # make forecast
    forecast = model.predict(X, batch_size=n_batch)
    # convert to array
    return [x for x in forecast[0, :]]

# evaluate the persistence model
def make_forecasts(model, n_batch, train, test, n_lag, n_seq):
    forecasts = list()
    for i in range(len(test)):
        X, y = test[i, 0:n_lag], test[i, n_lag:]
        # make forecast
        forecast = forecast_lstm(model, X, n_batch)
        # store the forecast
        forecasts.append(forecast)
    return forecasts

# invert differenced forecast
def inverse_difference(last_ob, forecast):
    # invert first forecast
    inverted = list()
    inverted.append(forecast[0] + last_ob)
    # propagate difference forecast using inverted first value
    for i in range(1, len(forecast)):
        inverted.append(forecast[i] + inverted[i-1])
    return inverted

# inverse data transform on forecasts
def inverse_transform(series, forecasts, scaler, n_test):
    inverted = list()
    for i in range(len(forecasts)):
        # create array from forecast
        forecast = array(forecasts[i])
        forecast = forecast.reshape(1, len(forecast))
        # invert scaling
        inv_scale = scaler.inverse_transform(forecast)
        inv_scale = inv_scale[0, :]
        # invert differencing
        index = len(series) - n_test + i - 1
        last_ob = series.values[index]
        inv_diff = inverse_difference(last_ob, inv_scale)
        # store
        inverted.append(inv_diff)
    return inverted

# evaluate the RMSE for each forecast time step
def evaluate_forecasts(test, forecasts, n_lag, n_seq):
    for i in range(n_seq):
        actual = [row[i] for row in test]
        predicted = [forecast[i] for forecast in forecasts]
        rmse = sqrt(mean_squared_error(actual, predicted))
        print('t+%d RMSE: %f' % ((i+1), rmse))

# plot the forecasts in the context of the original dataset
def plot_forecasts(series, forecasts, n_test):
    # plot the entire dataset in blue
    pyplot.plot(series.values)
    # plot the forecasts in red
    for i in range(len(forecasts)):
        off_s = len(series) - n_test + i - 1
        off_e = off_s + len(forecasts[i]) + 1
        xaxis = [x for x in range(off_s, off_e)]
        yaxis = [series.values[off_s]] + forecasts[i]
        pyplot.plot(xaxis, yaxis, color='red')
    # show the plot
    pyplot.show()

# load dataset
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
# configure
n_lag = 1
n_seq = 3
n_test = 10
n_epochs = 1500
n_batch = 1
n_neurons = 1
# prepare data
scaler, train, test = prepare_data(series, n_test, n_lag, n_seq)
# fit model
model = fit_lstm(train, n_lag, n_seq, n_batch, n_epochs, n_neurons)
# make forecasts
forecasts = make_forecasts(model, n_batch, train, test, n_lag, n_seq)
# inverse transform forecasts and test
forecasts = inverse_transform(series, forecasts, scaler, n_test+2)
actual = [row[n_lag:] for row in test]
actual = inverse_transform(series, actual, scaler, n_test+2)
# evaluate forecasts
evaluate_forecasts(actual, forecasts, n_lag, n_seq)
# plot forecasts
plot_forecasts(series, forecasts, n_test+2)
```

Running the example first prints the RMSE for each of the forecasted time steps.

We can see that the scores at each forecasted time step are better, in some cases much better, thar

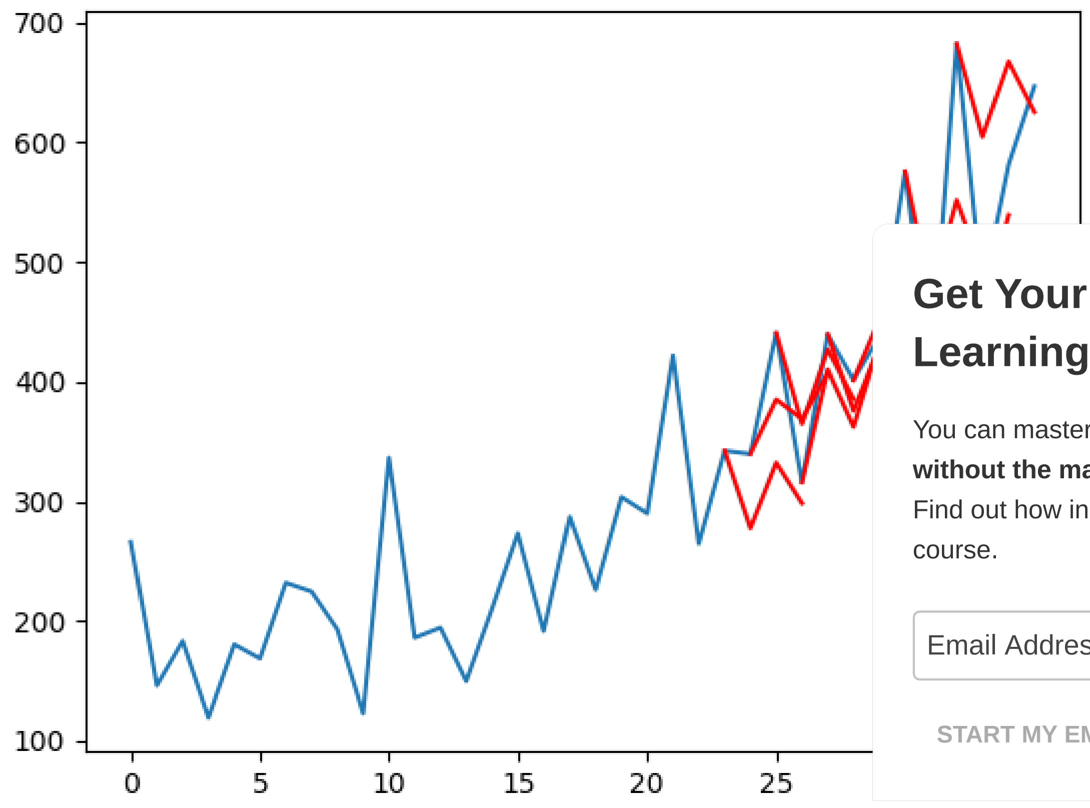
This shows that the configured LSTM does have skill on the problem.

It is interesting to note that the RMSE does not become progressively worse with the length of the forecast horizon, as would be expected. This is marked by the fact that the t+2 appears easier to forecast than t+1. This may be because the downward tick is easier to predict than the upward tick noted in the series (this could be confirmed with more in-depth analysis of the results).

```
t+1 RMSE: 95.973221
t+2 RMSE: 78.872348
t+3 RMSE: 105.613951
```

A line plot of the series (blue) with the forecasts (red) is also created.

The plot shows that although the skill of the model is better, some of the forecasts are not very good and that there is plenty of room for improvement.



Line Plot of Shampoo Sales Dataset with Multi-Step LSTM Forecasts

Extensions

There are some extensions you may consider if you are looking to push beyond this tutorial.

- **Update LSTM**. Change the example to refit or update the LSTM as new data is made available. [redacted] retrain with a new observation.
- **Tune the LSTM**. Grid search some of the LSTM parameters used in the tutorial, such as number of epochs, number of neurons, and number of layers to see if you can further lift performance.
- **Seq2Seq**. Use the encoder-decoder paradigm for LSTMs to forecast each sequence to see if this offers any benefit.
- **Time Horizon**. Experiment with forecasting different time horizons and see how the behavior of the network varies at different lead times.

Did you try any of these extensions?
Share your results in the comments; I'd love to hear about it.

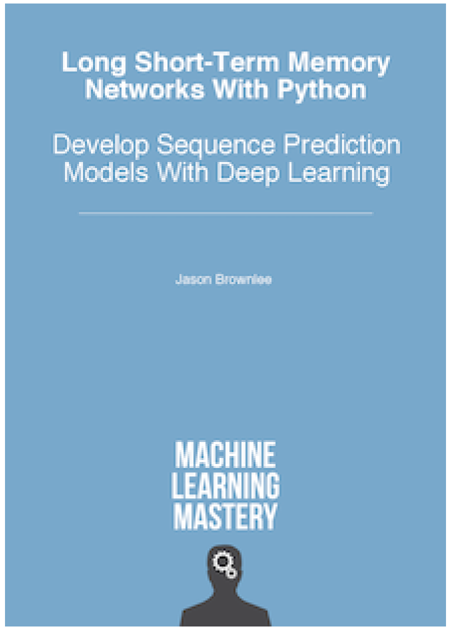
Summary

In this tutorial, you discovered how to develop LSTM networks for multi-step time series forecasting.

Specifically, you learned:

- How to develop a persistence model for multi-step time series forecasting.
- How to develop an LSTM network for multi-step time series forecasting.
- How to evaluate and plot the results from multi-step time series forecasting.

Do you have any questions about multi-step time series forecasting with LSTMs?
Ask your questions in the comments below and I will do my best to answer.


Long Short-Term Memory Networks With Python
Develop Sequence Prediction Models With Deep Learning
Jason Brownlee
MACHINE LEARNING MASTERY

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional devel
to helping developers get started and get good at applied machine learning. Learn more.
View all posts by Jason Brownlee →

< How to Convert a Time Series to a Supervised Learning Problem in Python

Demonstration of Memory with a Long Short-Term Memory Network in Python >

130 Responses to *Multi-step Time Series Forecasting with Long Short-Term Memory Networks in Python*

Masum May 10, 2017 at 6:48 am # REPLY

Thanks

you are the best

Did not had to wait for long. Asked for it in different blog few days back

Jason Brownlee May 10, 2017 at 8:53 am #

I hope you find the post useful!

Masum May 10, 2017 at 9:59 am #

I believe so. Things are getting deeper here.

Will we get recursive LSTM MODEL for multi step forecasting soon?

Will eagerly wait for that blog.

Thanks

Jason Brownlee May 11, 2017 at 8:22 am # REPLY

Maybe.

Masum May 11, 2017 at 8:43 am #

Sir,

Hope to see that soon.

jvr May 17, 2017 at 1:27 am #

REPLY

Thanks a lot for this post. I was trying to make this for my thesis since september, with no well results. But I'm having trouble: I'm not able to compile. Maybe you or someone who reads this is able to tell me why this happens: I'm getting the following error when running the code:

The TensorFlow library wasn't compiled to use SSE instructions, but these are available on your machin

The TensorFlow library wasn't compiled to use SSE2 instructions, but these are available on your machi

The TensorFlow library wasn't compiled to use SSE3 instructions, but these are available on your machi
.
The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your mac

The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your mac

The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machin

Obviously it has something to do with Tensorflow (I have read about this problem and I think its becase i
to fix it).

Thank you in advance.

Get Your Start in Machine Learning

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

Email Address

START MY EMAIL COURSE

Jason Brownlee May 17, 2017 at 8:38 am #

REPLY

These are warnings that you can ignore.

Get Your Start in Machine Learning

Shamsul May 17, 2017 at 9:17 pm # REPLY

Sir,

Can we say that multiple output strategy (avoiding 1.direct, 2. Recursive, 3.direct recursive hybrid strategies) have been used here ?

Am I right ?

Jason Brownlee May 18, 2017 at 8:36 am # REPLY

I think the LSTM has implemented a direct strategy.

jinhua zhang May 18, 2017 at 11:26 am #

Hi,Jason,
Your article is very useful! I have a problem, if the data series are three-dimensional data, the 2th line is
data(all include the train and test data),Do they can run the" difference"and "tansform"?
Thank you very much!

Jason Brownlee May 19, 2017 at 8:11 am #

Great question.

You may want to only make the prediction variable stationary. Consider perform three tests:

– Model as-is
– Model with output variable stationary
– Model with all variables stationary (if others are non-stationary)

REPLY

jvr May 21, 2017 at 10:21 pm #

I have discovered how to do it by asking some people. The object series is actually a Pandas Series. It's a vector of information, with a named index. Your dataset, however, contains two fields of information, in addition to the time series index, which makes it a DataFrame. This is the reason why the tutorial code breaks with your data.

To pass your entire dataset to MinMaxScaler, just run difference() on both columns and pass in the transformed vectors for scaling. MinMaxScaler accepts an n-dimensional DataFrame object:

```
ncol = 2
diff_df = pd.concat([difference(df[i], 1) for i in range(1,ncol+1)], axis=1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_values = scaler.fit_transform(diff_df)
```

So, with this, we can use as many variables as we want. But now I have a big doubt.

When the transform or dataset into a supervised learning problem, we have a distribution in columns
http://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

I mean, for a 2 variables dataset as yours, we can set, for example, this values:

```
n_lags=1
n_seq=2
```

so we will have a supervised dataset like this:

var1(t-1) var2(t-1) var1(t) var2 (t) var1(t+1) var2 (t+1)

so, if we want to train the ANN to forecast var2 (which is the target we want to predict) with the var1 ,
we have to separate them and here is where my doubt begins.

In the part of the code:

```
def fit_lstm(train, n_lag, n_seq, n_batch, nb_epoch, n_neurons):
# reshape training into [samples, timesteps, features]
X, y = train[:, 0:n_lag], train[:, n_lag:]
X = X.reshape(X.shape[0], 1, X.shape[1])
```

I think that if we want to define X, we should use:

[redacted]

X=train[:,0:n_lag*n_vars]

this means we are selecting this as X from the previous example:

var1(t-1) var2(t-1)

(number of lags*number of variables), so: X=train[:,0:1*2]=train[:,0:2]

but…

Y=train[:,n_lag*n_vars:] is the vector of ¿targets?

the problem is that, on this way, we are selecting this as targets:

var1(t) var2(t) var1(t+1) var2(t+1)

so we are including var1 (which we don't have the aim to forecast, just use as input).

I would like to know if there is any solution to solve this in order to use the variable 1,2…n-1 just as i

Hope this is clear :/

jvr May 19, 2017 at 3:16 am #

Thanks for the previous clarification. I have a dubt in relation to the section "fit network" in the c
graph (validation vs training) in order to see if the network is or not overfitted, but due to the "model.rese
val_loss from de history sentence. Is there any way to solve this?

thank you in advance 🙂

jvr May 19, 2017 at 3:45 am #

REPLY

I reply to myself, if someone is also interested.

Just creating 2 list (or 1, but i see it more clear on this way) and returning then on the function. Then, outside, just plot them. I'm sorry for the question, maybe the answer is obvious, but I'm starting on python and I'm not a programmer.

```
# fit network
loss=list()
val_loss=list()
for i in range(nb_epoch):
history=model.fit(X, y, epochs=1, batch_size=n_batch,shuffle=True, validation_split=val_split)
eqm=history.history['loss']
eqm_val=history.history['val_loss']
loss.append(eqm)
val_loss.append(eqm_val)
model.reset_states()

return model,loss,val_loss

# fit model
model,loss,val_loss=fit_lstm(train, n_lag, n_seq, n_batch, n_epochs, n_neurons)

pyplot.figure()
pyplot.plot(loss)
pyplot.plot(val_loss)
pyplot.title('cross validation')
pyplot.ylabel('MSE')
pyplot.xlabel('epoch')
pyplot.legend(['training', 'test'], loc='upper left')
pyplot.show()
```

Jason Brownlee May 19, 2017 at 8:23 am #

Nice to see you got there jvr, well done.

Jason Brownlee May 19, 2017 at 8:22 am #

REPLY

History is returned when calling model.fit().

We are only fitting one epoch at a time, so you can retrieve and accumulate performance each epoc[redacted]
(save/graph/return it) at the end of the loop.

Does that help?

jvr May 19, 2017 at 9:17 pm #

REPLY

It does help, thank you.

Now I'm trying to find a way to make the training process faster and reduce RMSE, but it's pretty dificult (the idea is to make results better than in the NARx model implemented in the Matlab Neural Toolbox, but results and computational time are hard to overcome).

Jason Brownlee May 20, 2017 at 5:37 am #

LSTMs often need to be trained longer than you think and can greatly benefit from

DJ June 2, 2017 at 1:42 am #

Hi,

Thanks for the great tutorial, I'm wondering if you can help me clarify the reason you have

```
model.reset_states()
```

(line 83)
when fitting the model, I was able to achieve similar results without the line as well.

Thanks!

Jason Brownlee June 2, 2017 at 1:02 pm # REPLY

It clears the internal state of the LSTM.

anurag August 30, 2017 at 3:41 pm # REPLY

I have tried experimenting with and without mode.reset_states(), using some other dataset.
I am doing multistep prediction for 6-10 steps, I am able to get better results without model.reset_states().

Am i doing something wrong, or it completely depends on dataset to dataset.

Thanks in advance.

Jason Brownlee August 30, 2017 at 4:20 pm #

It completely depends on the dataset and the model.

anurag August 31, 2017 at 6:42 pm #

Thank you so much. 🙂

DJ June 2, 2017 at 4:11 pm # REPLY

Thanks for the quick reply Jason :-). I've seen other places where reset is done by using callbacks parameter in model.fit.

```
class ResetStatesCallback(Callback):
```

[redacted]

def __init__(self):
self.counter = 0

def on_batch_begin(self, batch, logs={}):
if self.counter % max_len == 0:
self.model.reset_states()
self.counter += 1

Then the callback is used by as follows:

model.fit(X, y, epochs=1, batch_size=1, verbose=2,
shuffle=False, callbacks=[ResetStatesCallback()])

The ResetStatesCallback snippet was obtained from:
http://philipperemy.github.io/keras-stateful-lstm/

Please let me know what you think.

Thanks!

Jason Brownlee June 3, 2017 at 7:21 am #

Yes, there are many ways to implement the reset. Use what works best for your application

REPLY

QQ June 2, 2017 at 5:00 pm #

Hi Jason, greate post, and I have some questions:

1. in your fit_lstm function, you reset each epoch state, why?
2. why you iterate each epoch by yourself, instead of using model.fit(X, y, epochs)

thx Jason

[redacted]

```
# fit an LSTM network to training data
def fit_lstm(train, n_lag, n_seq, n_batch, nb_epoch, n_neurons):
# reshape training into [samples, timesteps, features]
X, y = train[:, 0:n_lag], train[:, n_lag:]
X = X.reshape(X.shape[0], 1, X.shape[1])
# design network
model = Sequential()
model.add(LSTM(n_neurons, batch_input_shape=(n_batch, X.shape[1], X.shape[2]), stateful=True))
model.add(Dense(y.shape[1]))
model.compile(loss='mean_squared_error', optimizer='adam')
# fit network
for i in range(nb_epoch):
model.fit(X, y, epochs=1, batch_size=n_batch, verbose=0, shuffle=False)
model.reset_states()
return model
```

Jason Brownlee June 3, 2017 at 7:23 am #

The end of the epoch is the end of the sequence and the internal state should not carry ove

I run the epochs manually to give fine grained control over when resets occur (by default they occur

J June 7, 2017 at 12:48 am #

I'd like to clarify line 99 in the LSTM example:

—– plot_forecasts(series, forecasts, n_test+2)

Is the n_test + 2 == n_test + n_lag – n_seq?

Thanks,
J

[redacted]

jvr June 15, 2017 at 11:49 pm # REPLY

I'd also like to know why using n_test + 2

M August 8, 2017 at 3:07 am # REPLY

I thought it should be n_test + 2 == n_test+n_seq-1 (regardless of n_seq). It would be great if someone could clarify that.

Mrtn October 4, 2017 at 8:36 pm #

M, you are right. Otherwise the RMS is incorrectly calculated and plotting is not alig

Get Your Start in Machine Learning

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

Email Address

START MY EMAIL COURSE

Kao June 10, 2017 at 5:46 pm #

Hi jason,
When I applied your code into a 22-year daily time series, I find out that the LSTM forecast result is simil
horizontal bar. I'm sure I did not mess those two methods, I wonder what cause this?

My key configure as follows:
n_lag = 1
n_seq = 3
n_test = 365*3

and my series length is 8035.

Jason Brownlee June 11, 2017 at 8:21 am #

Get Your Start in Machine Learning

You will need to tune the model to your problem. [redacted]

Kao June 25, 2017 at 6:55 pm # REPLY

Thanks to your tutorial, I've been tuning the parameters such as numbers of epochs and neurons these days. However, I noticed that you mentioned the grid search method to get appropriate parameters, could you please explain how to implement it into LSTM? I'm confused about your examples on some other tutorial which has a model class, seems unfamiliar to me.

Jason Brownlee June 26, 2017 at 6:07 am # REPLY

See this example on how to grid search with LSTMs manually:
http://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecastin

Kao June 28, 2017 at 1:25 am #

Thanks, I've just finished one test. What does it mean if error oscillates violently diminishing? Can I tune the model better, or LSTM is incapable of this time series?

Jason Brownlee June 28, 2017 at 6:28 am #

You may need a larger model (more layers and or more neurons).

MM June 13, 2017 at 6:44 am # REPLY

Jason,

Thank you for these tutorials. These are the best tutorials on the web. One question: what is the best wa [redacted]

Thank you

Jason Brownlee June 13, 2017 at 8:31 am #

REPLY

Thanks MM.

No one can tell you the "best" way to do anything in applied machine learning, you must discover it through trial and error on your specific problem.

MM June 13, 2017 at 9:29 am #

Jason,

Understood. Let me re-phrase the question. In a practical application, one would be interested ir
dataset, "3-12". How would you suggest doing that?

Jason Brownlee June 14, 2017 at 8:41 am #

Fit your model to all of the data then call predict() passing whatever lag inputs your

MM June 13, 2017 at 10:24 am #

REPLY

Jason,

Should the line that starts the offset point in plot_forecasts() be

off_s = len(series) – n_test + i + 1

not

off_s = len(series) – n_test + i – 1

[redacted]

Michael June 21, 2017 at 4:03 am #

REPLY

Hi Jason,

Thanks for your excellent tutorials!

I have followed a couple of your articles about LSTM and did learn a lot, but here is a question in my mind: can I introduce some interference elements in the model? For example for shampoo sale problem, there may be some data about holiday sales, or sales data after an incident happens. If I want to make prediction for sales after those incidents, what can I do?

What's more, I noticed that you will parse date/time with a parser, but you did not really introduce time fe prediction for next Monday or next January, how can I feed time feature?

Thanks!

Jason Brownlee June 21, 2017 at 8:18 am #

Yes, see this post for ideas on adding additional features:
http://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/

Michael June 22, 2017 at 5:53 pm #

Thanks for clarification.

I have two more specific questions:
1) In inverse_transform, why index = len(series) – n_test + i – 1?

2) In fit_lstm, you said "reshape training into [samples, timesteps, features]", but I think the code in line 74 is a little different from your format:

```
73 X, y = train[:, 0:n_lag], train[:, n_lag:]
74 X = X.reshape(X.shape[0], 1, X.shape[1])
```

In line 74, I think it should be X = X.reshape(X.shape[0], X.shape[1], 1)

Jason Brownlee June 23, 2017 at 6:52 am #

REPLY

Hi Michael,

Yes, the offset finds one step prior to the forecast in the original time series. I use this motif throughout the tutorial.

In the very next line I say: “We will fix time steps at 1, so this change is straightforward.”

Michael June 22, 2017 at 6:01 pm #

Hi Jason,

I would like to know how to do short term and long term prediction with minimum number of models?

For example, I have a 12-step input and 12-step output model A, and a 12-step input and 1-step output
next first time step than model B?

What’s more, if we have 1-step input and 1-step output model, it is more error prone to long term predict
if we have multi-step input and 1-step output mode it is still more more error prone long term. So how to

Jason Brownlee June 23, 2017 at 6:53 am #

REPLY

I would recommend developing and evaluating each model for the different uses cases. LSTMs are quite resistant to assumptions and rules of thumb I find in practice.

jzx June 25, 2017 at 1:17 pm # REPLY

Hello, thanks for your tutorial
If my prediction model is three time series a, b, c, I would like to use a, b, c to predict the future a, how can I build my LSTM model.
thank you very much!

Jason Brownlee June 26, 2017 at 6:05 am # REPLY

Each of a, b, and c would be input features. Remember, the shape or dimensions of input data is [samples, timesteps, features].

Kedar June 26, 2017 at 6:03 pm #

Does stationarizing data really help the LSTM? If so, what is the intuition behind that? I mean, I y
for LSTM's?

Jason Brownlee June 27, 2017 at 8:27 am #

Yes in my experience, namely because it is a simpler prediction problem.

I would suggest trying a few different "views" of your sequence and see what is easiest to model / ge

Michael June 28, 2017 at 5:47 pm # REPLY

Hi Jason,

I want to train a model with the following input size: [6000, 4, 2] ([samples, timestamps, features])

For example, I want to predict shampoo's sale in next two years. If I have other feature like economy ind[redacted] index data in the above format? So my input will be a 3d vector. How should I modify the model to train?

I always get such error: ValueError: Error when checking target: expected dense_1 to have 2 dimensions, but got array with shape (6000, 2, 2).

The error comes from this line: model.fit(X, y, epochs=1, batch_size=n_batch, verbose=0, shuffle=False). Can you provide some advices? Thanks!

Jason Brownlee June 29, 2017 at 6:32 am # REPLY

Reshape your data to be [6000, 4, 2]

Update the input shape of the network to be (4,2)

Adjust the length of the output sequence you want to predict.

shamsul July 11, 2017 at 11:31 am #

sir,

To make one forecast with an LSTM, if we write

oneforecast = forecast_lstm(model, X, n_batch)

it says: undefined X

what should be the value of X? we know the model and n_batch value?

would you help?

Jason Brownlee July 12, 2017 at 9:38 am # REPLY

X would be the input sequence required to make a prediction, e.g. lag obs.

masum July 12, 2017 at 8:06 am #

REPLY

sir,

what if I want to tell the model to learn from train data (23 samples here) and want to forecast only 3 steps forward (Jan, Feb, Mar). I want to avoid persistence model in this case and only require 3 step direct strategy. hope you got that.

any help would be grateful.

tarin (past data)= forecast (Jan, Feb, Mar)

Jason Brownlee July 12, 2017 at 9:54 am #

Perhaps I misunderstand, but this is the model presented in the tutorial. It predicts 3 time st

---

masum July 12, 2017 at 11:00 am #

```
# evaluate the persistence model
def make_forecasts(model, n_batch, train, test, n_lag, n_seq):
forecasts = list()
for i in range(len(test)):
X, y = test[i, 0:n_lag], test[i, n_lag:]
# make forecast
forecast = forecast_lstm(model, X, n_batch)
# store the forecast
forecasts.append(forecast)
return forecasts
```

here if i would like to make only one forecast for 3 steps (jan,feb,march) what i have to change. i do not need the rest of the month(april, may, june, july,aug,……dec). one predictions or forecast for 3 steps.

hope you got me

[redacted]

Jason Brownlee July 13, 2017 at 9:47 am # REPLY

Pass in only what is required to make the prediction for those 3 months.

masum July 13, 2017 at 10:16 am #

sir,

will be kind enough to simplify a little bit more.

I did not get it.

Devakar Kumar Verma July 24, 2017 at 4:23 am #

I am getting an error while parsing the date at time of loading the data from csv file.
The error is:
ValueError: time data '1901-Jan' does not match format '%Y-%m'

Anyone please help me to resolve this issue.

Jason Brownlee July 24, 2017 at 6:56 am # REPLY

I'm sorry to hear that. Confirm you have copied the code exactly and the data file does not have any extra footer information.

p July 30, 2017 at 8:05 pm # REPLY

[redacted]

hi
I have so this problem
i have downloaded the dataset from the link in the text
i think this error has occured because the data of our csv file is not in correct format!
can anyone give us the dataset plz???

Jason Brownlee July 31, 2017 at 8:15 am #

REPLY

Here is the raw data ready to go:

```
"Month","Sales"
"1-01",266.0
"1-02",145.9
"1-03",183.1
"1-04",119.3
"1-05",180.3
"1-06",168.5
"1-07",231.8
"1-08",224.5
"1-09",192.8
"1-10",122.9
"1-11",336.5
"1-12",185.9
"2-01",194.3
"2-02",149.5
"2-03",210.1
"2-04",273.3
"2-05",191.4
"2-06",287.0
"2-07",226.0
"2-08",303.6
"2-09",289.9
"2-10",421.6
"2-11",264.5
"2-12",342.3
"3-01",339.7
"3-02",440.4
"3-03",315.9
"3-04",439.3
"3-05",401.3
"3-06",437.4
```

```
"3-07",575.5
"3-08",407.6
"3-09",682.0
"3-10",475.3
"3-11",581.3
"3-12",646.9
```

Dongchan October 9, 2017 at 9:26 am # REPLY

Sir,

I have the same issue. How can I fix the parser to resolve this error?

Devakar Kumar Verma July 24, 2017 at 2:34 pm #

@Jason,
Data file doesn't have any footer and i had simply copy paste the code but dateparser throwing the error

Jason Brownlee July 25, 2017 at 9:27 am #

Sorry, I don't have any good ideas. It may be a Python environment issue?

Josep July 31, 2017 at 8:15 pm # REPLY

Hi Jason,
Great explanation again. I have a doubt about this piece of code:

# evaluate the persistence model
def make_forecasts(model, n_batch, train, test, n_lag, n_seq):

[redacted]

forecasts = list()
for i in range(len(test)):
X, y = test[i, 0:n_lag], test[i, n_lag:]
# make forecast
forecast = forecast_lstm(model, X, n_batch)
# store the forecast
forecasts.append(forecast)
return forecasts

Why do you pass the parameter "n_seq" to the function if it has no use inside the function?

Jason Brownlee August 1, 2017 at 7:59 am #

Good point, thanks.

Nara August 1, 2017 at 10:12 pm #

Hi,
How would I go about forecasting for a complete month. (Assuming I have daily data).
Assuming I have around 5 years data 1.8k data points to train.

I would like to use one year old data to forecast for the whole of next month?

To do this should I change the way this model is trained?
Is my understanding correct that this model tries to predict the next value by only using current value?

Jason Brownlee August 2, 2017 at 7:50 am #

REPLY

Yes, frame the data so that it predicts a month, then train the model.

The model can take as input whatever you wish, e.g. a sequence of the last month or year.

[redacted]

Nara August 3, 2017 at 3:12 am #

REPLY

Hey, thanks for the reply.

This post really helped me.
Now the next question is how do we enhance this to consider exogenous variables while forecasting?
If I simply add exogenous variable values at this step:
train, test = supervised_values[0:-n_test], supervised_values[-n_test:], (and obviously make appropriately changes to batch_input_shape in model fit.)
Would it help improve predictions?
What is the correct way of adding independent variables.

I have gone through this post of yours.
http://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
It was helful but how to do this using neural networks that has LSTM?
Can you please point me in the right direction?

Jason Brownlee August 3, 2017 at 6:55 am #

Additional features can be provided directly to the model as new features.

See this post on framing the problem, then reshape the results:
http://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python

Kiran August 4, 2017 at 2:09 pm #

REPLY

Hi Jason, thanks for writing up such detailed explanations.
I am using an LSTM layer for a time series prediction problem.
Everything works fine except for when I try to use the inverse_transform to undo the scaling of my data. I get the following error:

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

Not really sure how I can get past this problem. Could you please help me with this ? [redacted]

Jason Brownlee August 4, 2017 at 3:45 pm # REPLY

It looks like you are tring to perform an inverse transform on NaN values.

Perhaps try some print statements to help track down where the NaN values are coming from.

Kiran August 5, 2017 at 12:01 pm # REPLY

Thank you for the reply. Yes, there are some NaN values in my predictions. Does that i

Jason Brownlee August 6, 2017 at 7:36 am #

Your model might be receiving NaN as input, check that.

It may be making NaN predictions with good input, in which case it might have had trouble d clipping that can address this.
https://keras.io/optimizers/

Figure out which case it is first though.

Kiran August 14, 2017 at 11:05 pm #

Thanks ! My inputs do not have any NaN. Will check out gradient clipping.

Jason Brownlee August 15, 2017 at 6:37 am #

Let me know how you go Kiran.

Nara August 8, 2017 at 9:34 pm #

REPLY

Hi Jason,

When I try step by step forecast. i.e. forecast 1 point and then use this back as data and forecast the next point, my predictions become constant after just 2 steps, sometimes from the beginning itself.

https://datascience.stackexchange.com/questions/22047/time-series-forecasting-with-rnnstateful-lstm-pr
In detail there. Can you say why this is happening? And which forecast method is usually better. Step by

Also can you comment on when can ARIMA/ linear models perform better than netowrks/RNN?

Jason Brownlee August 9, 2017 at 6:30 am #

Using predictions as input is bad as the errors will compound. Only do this if you cannot ge

If your model has a linear relationship it will be better to model it with a linear model with ARIMA, the

Nara August 11, 2017 at 10:09 pm #

REPLY

But that is how ARIMA models predict right?
They do point by point forecast. And from my results ARIMA(or STL ARIMA or even XGBOOST) is doing pretty well when compared to RNN. 🙁

But i haven't considered stationarity and outlier treatment and I see that RNN performs pathetically when the data is non stationary/has outliers.

Is this expected? I have read that RNN should take care of stationarity automatically?

Also, will our results be bad if we do first order differencing even when there is no stationarity in [redacted]

And as for normalization, is it possible that for some cases RNN does well without normalizing?
When is normalization usually recommended? When standard deviation is huge?

Jason Brownlee August 12, 2017 at 6:49 am # REPLY

I have found RNNs to not perform well on autoregression problems, and they do better with more data prep (e.g. removing anything systematic). See this post:
http://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecasting/

Generally, don't difference if you don't need to, but test everything to be sure.

Standardization if the distribution is Gaussian, normalization otherwise. RNNs like LSTMs ne relu.

Nara August 13, 2017 at 1:34 am #

Oh then a hybrid model using residuals from ARIMA for RNN should work well
The residuals will not have any seasonal components.(even scaling should be well take
Or here also do you expect MLPs to work better?

Jason Brownlee August 13, 2017 at 9:55 am #

It is hard to know for sure, I recommend using experiments to collect data to know for sure, rather than guessing.

Nights August 13, 2017 at 5:37 am # REPLY

I think there is an issue with inverse differencing while forecasting for multistep.(to deal with non stationa[redacted]
This example is adding previously forecasted(and inverse differenced) value to the currently forecasted value.Isn't this method wrong when we have 30 points to forecast as it keeps adding up the results and hence the output will continuously increase.

Below is the output I got.
https://ibb.co/d1oyNF

Instead should I just add the last known real observation to all the forecasted values? I dont suppose that would work either.

Jason Brownlee August 13, 2017 at 9:58 am #

REPLY

It could be an issue for long lead times, as the errors will compound.

If real obs are available to use for inverse differencing, you won't need to make a forecast for such a

Consider contrasting model skill with and without differencing, at least as a starting point.

Sandra August 14, 2017 at 5:46 pm #

Hi, thank you for your helpful tutorial.

I have a question regarding a seq to seq timeseries forcasting problem with multi-step lstm.

I have created a supervised dataset of (t-1), (t-2), (t-3)..., (t-look_back) and (t+1), (t+2), (t+3)..., (t+look_
timesteps.

We have tried your complete example code of doing a dense(look_ahead) last layer but received not so good results. This was done using both a stateful and non-stateful network.

We then tried using Dense(1) and then repeatvector(look_ahead), and we get the same (around average) value for all the look_ahead timesteps. This was done using a non-stateful network.

Then I created a stepwise prediction where look_ahead = 1 always. The prediction for t+2 is then based on the history of (t+1)(t)(t-1)... This has given me better results, but only tried for non-stateful network.

[redacted]

My questions are:
– Is it possible to use repeatvector with non-stateful networks? Or must network be stateful? Do you have any idea why my predictions are all the same value?
– What do network you recommend for this type or problem? Stateful or non stateful, seq to seq or stepwise prediction?

Thanks in advance!
Sandra

Jason Brownlee August 15, 2017 at 6:32 am # REPLY

Very nice work Sandra, thanks for sharing.

The RepeatVector is only for the Encoder-Decoder architecture to ensure that each time step in the
encoding vector from the Encoder. It is not related to stateful or stateless models.

I would develop a simple MLP baseline with a vector output and challenge all LSTM architectures to
LSTM and a seq2seq model. I would also try the recursive model (feed outputs as inputs for repeati

It sounds like you're trying all the right things.

Now, with all of that being said, LSTMs may not be very good at simple autoregression problems. I c
autoregression. See this post:
http://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecas

I hope that helps, let me know how you go.

Oscar August 16, 2017 at 1:28 am # REPLY

Hi Jason,
Thanks for your tutorials. I'm trying to learn ML and your webpage is very useful!

I'm a bit confuse with the inverse_difference function. Specifically with the last_ob that I need to pass.

Let's say I have the following:

[redacted]

Raw Data difference scaled Forecasted values
raw_val1=.4
raw_val2=.35 -.05 -.045 [0.80048585, 0.59788215, -0.13518856]
raw_val3=.29 -.06 -.054 [0.65341175, 0.37566081, -0.14706305]
raw_val4=.28 -.01 -.009 [[0.563694, -0.09381149, 0.03976132]

When passing the last_ob to the inverse_difference function which observation do I need to pass to the function, raw_val2 or raw_val1?

My hunch is that I need to pass raw_val2. Is that correct?

Also, in your example, in the line:

forecasts = inverse_transform(series, forecasts, scaler, n_test+2)

What's the reason of this n_test+2?

Thanks in advance!
Oscar

Jaskaran August 17, 2017 at 10:57 am #

Hi Jason,
Great work.

I had a question. When reshaping X for lstm (samples,timesteps,features) why did you model the proble t
it be timesteps = lag window size
and the output dense layer have the size of horizon_window. This will give much better results in my opi

Here is a link which will make my question more clear:
https://stackoverflow.com/questions/42585356/how-to-construct-input-data-to-lstm-for-time-series-multi-step-horizon-with-exte

Jason Brownlee August 17, 2017 at 4:54 pm #

REPLY

I model the problem with no timesteps and lots of features (multiple obs at the same time).

I found that if you frame the problem with multiple time steps for multiple features, performance was [redacted] type network here.

LSTMs are not great at autoregression, but this post was the most requested I've ever had.

More on LSTM suitability here:
http://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecasting/

Jaskaran August 18, 2017 at 5:59 am # REPLY

So Jason,

Correct me if I am wrong but the whole point of RNN+LSTM learning over time(hidden states de

Essentially, this is just an autoregressive neural network. There is no storage of states over time

Jason Brownlee August 18, 2017 at 6:31 am #

Yes, there is no BPTT because we are only feeding in one time step.

You can add more history, but results will be worse. It turns out that LSTMs are poor at autoı
http://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series

Nevertheless, I get a lot of people asking how to do it, so here it is.

hanoun August 18, 2017 at 11:37 am # REPLY

Hi, I try to use this example to identify the shape switch an angle , its useful to use this tutorial and how I can test the model I train it,
Regards,
Hanen

REPLY

A August 19, 2017 at 7:53 am #

Hi there – I love your blog and these tutorials! They're really helpful.

I have been studying both this tutorial and this one: http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/. I have applied both codes to a simple dataset I'm working with (date, ROI%). Both codes run fine with my data, but I'm having a problem that has me completely stumped:

With this code, I'm able to actually forecast the future ROI%. With the other, it does a lot better at modeling the past data, but I can't figure out how to get it to forecast the future. Both codes have elements I need, but I can't seem to figure out how to bring them together.

Any insight would be awesome! Thank you!

Jason Brownlee August 20, 2017 at 6:04 am #

What is the problem exactly?

Ankit August 22, 2017 at 11:34 pm #

Jason, first of all, I would like to thank you for the work you've done. It has been tremendously h

I have a question and seeking your expert opinion.

How to handle a time series data set with multiple and variable granularity input of each time step. for ins

Date | Area | Product category | Orders | Revenue | Cost

so, in this case, there would be multiple records for a single day aggregated on date and this is the granularity I want.

How should this kind of data be handled, since these features will contribute to the Revenue and Orders?

Jason Brownlee August 23, 2017 at 6:53 am #

[redacted]

You could standardize the data and feed it into one model or build separate models and combine their predictions.

Try a few methods and see what works best for your problem.

Daniel August 24, 2017 at 2:07 am # REPLY

I am using this framework for my first shot at an LSTM network for monitoring network response times. The data I'm working with currently is randomly generated by simulating API calls. What I'm seeing is the LSTM seems to always predict a return to what looks like the mean of the data. Is this a function of the data being stochastic?

Separate question: since LSTM's have a memory component built into the neurons, what are the advantages/disadvantages of using a larger n_in/n_lag than 1?

Jason Brownlee August 24, 2017 at 6:48 am #

THe problem might be too hard for your model, perhaps tune the LSTM or try another algor

A key benefit of LSTMs is that they the lag can extend much longer than other methods, e.g. hundre something like:

yhat = f(t-1, …, t-500)

And the model can reproduce something it saw 500 time steps ago if needed.

Daniel August 26, 2017 at 3:34 am # REPLY

Thanks. I am playing with some toy data now just to make sure I'm understanding how this works.

I am able to model a cosine wave very nicely with a 5 neuron, 100 epoch training run against np.cos(range(100)) split into 80/20 training set. This is with the scaling, but without the difference. I feed in 10 inputs, and get 30 outputs.

Does calling model.predict change the model? I am calling repeatedly with the same 10 inputs [redacted] the predicted wave cycles through different amplitudes.

Daniel August 26, 2017 at 4:09 am #

REPLY ↩

Ah ok, I got it. Since stateful is on, I would need to do an explicit reset_states between predictions. Makes sense, I think! Stateful was useful for training, but since I won't be "online learning" and since I feed the network lag in the features, I should not rely on state for predictions.

Jason Brownlee August 26, 2017 at 6:48 am #

Nice work!

Yes, generally scaling is important, but if your cosine wave values are in [0,1] then you're go

Daniel August 26, 2017 at 6:03 am #

I have a simple question. Trying to set up an a different toy problem, with data generate

validation). No matter how many layers, neurons, epochs that I train over, the results tend to be

lower values, but it diverges quickly and and approaches some fixed y=400 for higher values.

Do you have any ideas why this would happen?

Jason Brownlee August 26, 2017 at 6:51 am #

REPLY ↩

May be error accumulating. You're giving the LSTM a hard time.

Daniel September 1, 2017 at 2:47 am # REPLY

Can I get your input on this issue I'm having? I would really like to make sure that I'm not implementing incorrectly. If there are network parameters I need to do, I can go through that exercise. But, I am not feeling confident about what I am on the right path with this problem.
https://stackoverflow.com/questions/45982445/keras-lstm-time-series-multi-step-predictions-has-same-output-for-any-input

lucius September 1, 2017 at 6:14 pm # REPLY

Hi, there is a problem with the code. when doing data processing, i.e. calculate difference and min max scale. you should not use all data. in more real situation, you can only do this to train data. since you have no idea about test data.

So I changed the code, cut the last 12 month as test. then only use 24 months data for difference, min m
27.

Then I continue to use 25 months data for difference, min max scale, fit the model and predict for month
…

The final result is worse than baseline.!

Jason Brownlee September 2, 2017 at 6:04 am #

Correct, this is a simplification I implemented to keep the tutorial short and understandable.

Eldar M. September 17, 2017 at 1:47 am # REPLY

Hi Jason, I was able to get slightly better results with a custom loss function (weighted mse)

def weighted_mse(yTrue,yPred):

[redacted]

ones = K.ones_like(yTrue[0,:])
idx = K.cumsum(ones)
return K.mean((1/idx)*K.square(yTrue-yPred))

credit goes to Daniel Möller on Stack Overflow as I was not able to figure out the tensor modification steps on my own and he responded to my question there

Jason Brownlee September 17, 2017 at 5:28 am # REPLY

Nice one! Thanks for sharing.

Alex September 23, 2017 at 1:53 am #

def make_forecasts(model, n_batch, train, test, n_lag, n_seq):
forecasts = list()
for i in range(len(test)):
X, y = test[i, 0:n_lag], test[i, n_lag:]
# make forecast
forecast = forecast_lstm(model, X, n_batch)
# store the forecast
forecasts.append(forecast)
return forecasts

What is the point of the "train" data set as parameter in this function if it is not used?
Thanks

Jason Brownlee September 23, 2017 at 5:43 am # REPLY

Yep, looks like its not used. You can probably remove it.

Fei September 24, 2017 at 1:51 am # REPLY

Hello, It is very useful tutorial. I am starter for the python and programming. May I convert input of model into 4 or more than one variable? and change the n_batch into other number not 1?

Jason Brownlee September 24, 2017 at 5:17 am # REPLY

Sure.

Fei September 26, 2017 at 4:33 am #

But ,When I change the n_batch size, the model does not work. By the way, you said manually ?

Fabian September 29, 2017 at 7:41 pm #

Hi Jason,
thanks a lot for your tutorials on LSTMs.
Do you have a suggestion how to model the network for a multivariate multi-step forecast? I read your a
combining both seems to be more tricky as the output of the dense layer gets a higher dimension.

In words of your example here: if I want to forecast not only shampoo but also toothpaste sales T time steps ahead, how can I achieve the forecast to have the dimension 2xT? Is there an alternative to the dense layer?

Jason Brownlee September 30, 2017 at 7:38 am # REPLY

I see. You could have two neurons in the output layer of your network, as easy as that.

Camille September 30, 2017 at 9:07 am # REPLY

Thanks for this great tutorial. Do you think this technique is applicable on the case of a many-to-many prediction?

A toy scenario: Imagine a machine with has 5 tuning knobs [x1, x2, x3, x4, x5] and as a result we can read 2 values [y, z] as a response to a change of any of the knobs.

I am wondering if I can use LSTM to predict y and z at with a single model instead of building one model for y and another for z? I am planning to follow this tutorial but I will love to hear what you think about it.

Jason Brownlee October 1, 2017 at 9:02 am #

Yes, LSTMs can easily be configured to support multiple input series and output a vector or

For example of taking multiple series as input, see this post:
https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

Jean-Marc September 30, 2017 at 12:08 pm #

Hi Jason, thank you very much for this tutorial. I am just starting with LSTM and your series on
A question about multi-output forecasting: how to deal with a multi-output when plotting the true data ver
Let's say I have a model to forecast the next 10 steps (t, t+1…,t+9).
Using the observation at time:
–> t=0, the model will give a forecast for t =1,2,3,4,5,6,7,8,9,10
and similarly, at
–> t=1, a forecast will be outpout for t=2,3,4,5,6,7,8,9,10,11
etc…
There is overlap in the timestep for the forecast from t=0 and from t=1. For example, if I want to know the value at t=2, should I use the forecast from t=1 or from t=0, or a weighted average of the forecast?

May be using only the forecast from t=1 enough, because it already includes the history of the time serie

Get Your Start in Machine Learning

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

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

[redacted]

Jason Brownlee October 1, 2017 at 9:06 am # REPLY

I'm not sure I follow. Perhaps you might be better off starting with linear models then move to an LSTM to lift skill on a framing/problem that is already working:
https://machinelearningmastery.com/start-here/#timeseries

mr October 1, 2017 at 9:53 pm # REPLY

The:
return datetime.strptime('190'+x, '%Y-%m')

gives me:

ValueError: time data '1901/1' does not match format '%Y-%m'

Thanks in advance

Jason Brownlee October 2, 2017 at 9:38 am #

Perhaps confirm that you downloaded the dataset in CSV format.

wmbm October 4, 2017 at 10:29 pm # REPLY

So you don't actually need to split the data into test and training sets because you don't use the training set in this code. So this then becomes an unsupervised problem?

Jason Brownlee October 5, 2017 at 5:23 am # REPLY

No, it is a supervised learning model.

We use walk-forward validation. Learn more about it here:
https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

wmbm October 5, 2017 at 6:26 am # REPLY

my mistake, I was look at just the multi-step persistence model. Thanks!

Jason Brownlee October 5, 2017 at 5:15 pm #

No problem.

Noah yao October 16, 2017 at 2:33 pm #

sorry i am confuse about the function inverse_transform why you use n_test+2 in the function b

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Get Your Start in Machine Learning

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

Email Address

START MY EMAIL COURSE

Welcome to Machine Learning Mastery

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

Read More

Deep Learning for Sequence Prediction

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

Get Your Start in Machine Learning

POPULAR

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

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

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

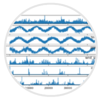
Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

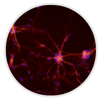
How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

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

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

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


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


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

Privacy | Contact | About

Get Your Start in Machine Learning

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

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning