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

Time Series Forecasting with the Long Short-Term Memory Network in Python

by **Jason Brownlee** on April 7, 2017 in **Long Short-Term Memory Networks**

The Long Short-Term Memory recurrent neural network has the promise of learning long sequences of observations.

It seems a perfect match for time series forecasting, and in fact, it may be.

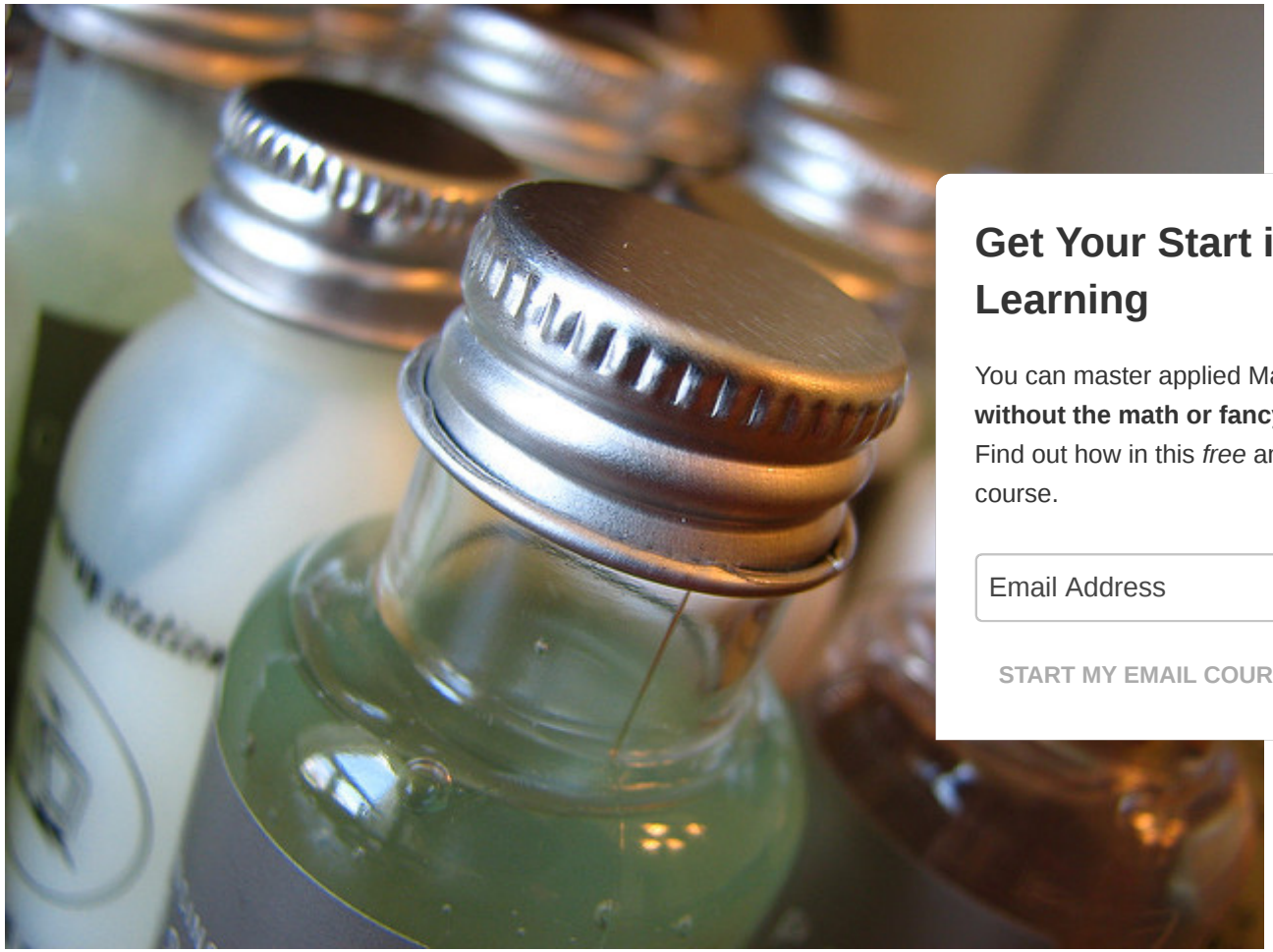
In this tutorial, you will discover how to develop an LSTM forecast model for a one-step univariate time series forecasting problem.

After completing this tutorial, you will know:

- How to develop a baseline of performance for a forecast problem.
- How to design a robust test harness for one-step time series forecasting.
- How to prepare data, develop, and evaluate an LSTM recurrent neural network for time series forecasting.

Let's get started.

- **Update May/2017**: Fixed bug in invert_scale() function, thanks Max.



Time Series Forecasting with the Long Short-Term Memory Network in Python
Photo by Matt MacGillivray, some rights reserved.

Tutorial Overview

This is a big topic and we are going to cover a lot of ground. Strap in.

This tutorial is broken down into 9 parts; they are:

1. Shampoo Sales Dataset
2. Test Setup
3. Persistence Model Forecast
4. LSTM Data Preparation
5. LSTM Model Development
6. LSTM Forecast
7. Complete LSTM Example
8. Develop a Robust Result
9. Tutorial Extensions

Python Environment

This tutorial assumes you have a Python SciPy environment installed. You can use either Python 2 c

You must have Keras (2.0 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 with your environment, see this post:

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

Shampoo Sales Dataset

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

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

You can download and learn more about the dataset here.

Download the dataset to your current working directory with the name "*shampoo-sales.csv*". Note that you may need to delete the footer information added by DataMarket.

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

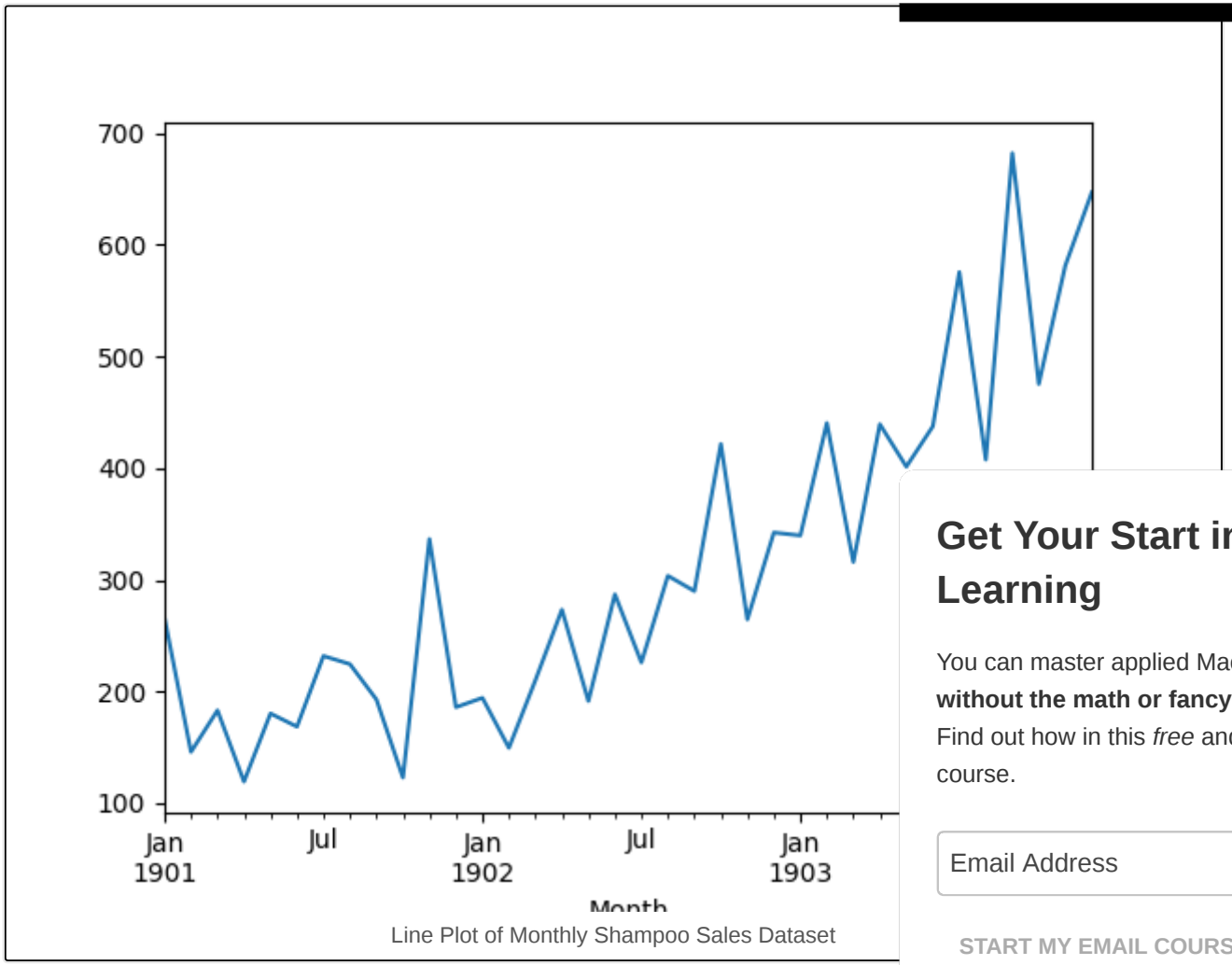
```
# load and plot dataset
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
# load dataset
def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=T
# summarize first few 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.

Get Your Start in Machine Learning



Line Plot of Monthly Shampoo Sales Dataset

Experimental Test Setup

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.

For example:

```
1 # split data into train and test
```

```
2 X = series.values
3 train, test = X[0:-12], X[-12:]
```

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

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.

For example:

```
1 # walk-forward validation
2 history = [x for x in train]
3 predictions = list()
4 for i in range(len(test)):
5     # make prediction...
```

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

Finally, all forecasts on the test dataset will be collected and an error score calculated to summarize (RMSE) will be used as it punishes large errors and results in a score that is in the same units as the

For example:

```
1 from sklearn.metrics import mean_squared_error
2 rmse = sqrt(mean_squared_error(test, predictions))
3 print('RMSE: %.3f' % rmse)
```


Persistence Model Forecast

A good baseline forecast for a time series with a linear increasing trend is a persistence forecast.

The persistence forecast is where the observation from the prior time step (t-1) is used to predict the observation at the current time step (t).

We can implement this by taking the last observation from the training data and history accumulated the current time step.

Get Your Start in Machine Learning

For example:

```
# make prediction
yhat = history[-1]
```

We will accumulate all predictions in an array so that they can be directly compared to the test dataset.

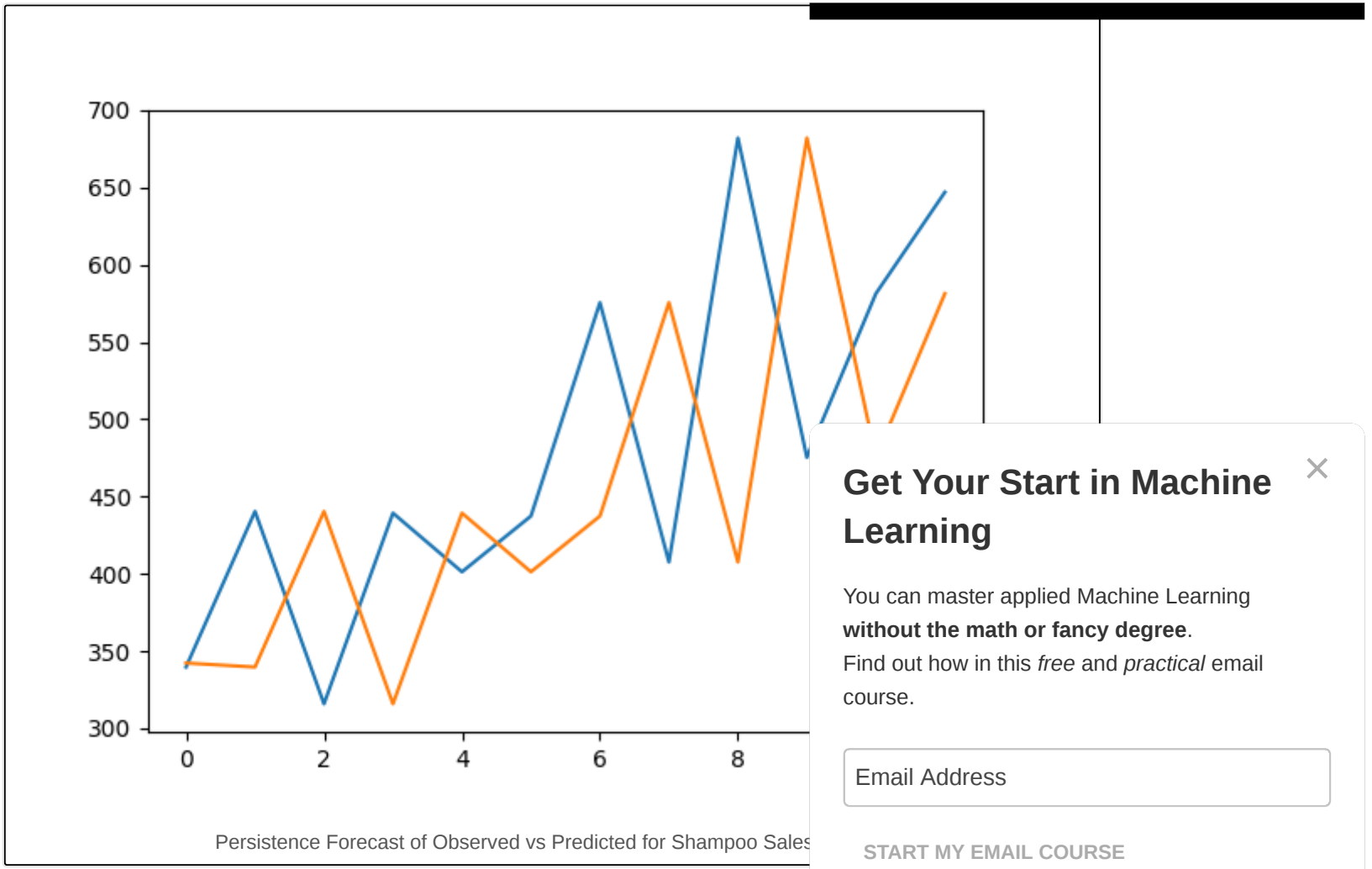
The complete example of the persistence forecast model on the Shampoo Sales dataset is listed below.

```
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from math import sqrt
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
# split data into train and test
X = series.values
train, test = X[0:-12], X[-12:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # make prediction
    predictions.append(history[-1])
    # observation
    history.append(test[i])
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(test)
pyplot.plot(predictions)
pyplot.show()
```


Running the example prints the RMSE of about 136 monthly shampoo sales for the forecasts on the test dataset.

```
RMSE: 136.761
```

A line plot of the test dataset (blue) compared to the predicted values (orange) is also created showing the persistence model forecast in context.



Persistence Forecast of Observed vs Predicted for Shampoo Sales

For more on the persistence model for time series forecasting, see this post:

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

Now that we have a baseline of performance on the dataset, we can get started developing an LSTM model for the data.

LSTM Data Preparation

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

This section is broken down into three steps:

1. Transform the time series into a supervised learning problem
2. Transform the time series data so that it is stationary.
3. Transform the observations to have a specific scale.

Transform Time Series to Supervised Learning

The LSTM model in Keras assumes that your data is divided into input (X) and output (y) componen

For a time series problem, we can achieve this by using the observation from the last time step (t-1) e step (t) as the output.

We can achieve this using the shift() function in Pandas that will push all values in a series down by a specified number places. We require a shift of 1 place, which will become the input variables. The time series as it stands will be the output variables.

We can then concatenate these two series together to create a DataFrame ready for supervised learning. The pushed-down series will have a new position at the top with no value. A NaN (not a number) value will be used in this position. We will replace these NaN values with 0 values, which the LSTM model will have to learn as "the start of the series" or "I have no data here," as a month with z

Get Your Start in Machine Learning

The code below defines a helper function to do this called *timeseries_to_supervised()*. It takes a NumPy array of the raw time series data and a lag or number of shifted series to create and use as inputs.

```
# 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.fillna(0, inplace=True)
    return df
```

We can test this function with our loaded Shampoo Sales dataset and convert it into a supervised learning problem.

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

# 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.fillna(0, inplace=True)
    return df

# 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
# transform to supervised learning
X = series.values
supervised = timeseries_to_supervised(X, 1)
print(supervised.head())
```


Running the example prints the first 5 rows of the new supervised learning problem.

```
           0           0
0   0.000000  266.000000
1  266.000000  145.899994
2  145.899994  183.100006
3  183.100006  119.300003
4  119.300003  180.300003
```

For more information on transforming a time series problem into a supervised learning problem, see the post:

- Time Series Forecasting as Supervised Learning

Transform Time Series to Stationary

The Shampoo Sales dataset is not stationary.

This means that there is a structure in the data that is dependent on the time. Specifically, there is an increasing trend in the data.

Stationary data is easier to model and will very likely result in more skillful forecasts.

The trend can be removed from the observations, then added back to forecasts later to return the prediction to the original scale and calculate a comparable error score.

A standard way to remove a trend is by differencing the data. That is the observation from the previo observation (t). This removes the trend and we are left with a difference series, or the changes to the

We can achieve this automatically using the diff() function in pandas. Alternatively, we can get finer g is, which is preferred for its flexibility in this case.

Below is a function called *difference()* that calculates a differenced series. Note that the first observa observation with which to calculate a differenced value.

```
# 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 also need to invert this process in order to take forecasts made on the differenced series back into their original scale.

The function below, called *inverse_difference()*, inverts this operation.

```
# invert differenced value
def inverse_difference(history, yhat, interval=1):
```

```
    return yhat + history[-interval]
```

We can test out these functions by differencing the whole series, then returning it to the original scale, as follows:

```
from pandas import read_csv
from pandas import datetime
from pandas import Series

# 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]

# 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
print(series.head())
# transform to be stationary
differenced = difference(series, 1)
print(differenced.head())
# invert transform
inverted = list()
for i in range(len(differenced)):
    value = inverse_difference(series, differenced[i], len(series)-i)
    inverted.append(value)
inverted = Series(inverted)
print(inverted.head())
```


Running the example prints the first 5 rows of the loaded data, then the first 5 rows of the differenced series, then finally the first 5 rows with the difference operation inverted.

Note that the first observation in the original dataset was removed from the inverted difference data. Besides that, the last set of data matches the first as expected.

```
Month
1901-01-01    266.0
```

```
3  1901-02-01    145.9
4  1901-03-01    183.1
5  1901-04-01    119.3
6  1901-05-01    180.3
7
8  Name: Sales, dtype: float64
9  0   -120.1
10 1     37.2
11 2    -63.8
12 3     61.0
13 4    -11.8
14 dtype: float64
15
16 0    145.9
17 1    183.1
18 2    119.3
19 3    180.3
20 4    168.5
21 dtype: float64
```

For more information on making the time series stationary and differencing, see the posts:

- How to Check if Time Series Data is Stationary with Python
- How to Difference a Time Series Dataset with Python

Transform Time Series to Scale

Like other neural networks, LSTMs expect data to be within the scale of the activation function used

The default activation function for LSTMs is the hyperbolic tangent (*tanh*), which outputs values betw me series data.

To make the experiment fair, the scaling coefficients (min and max) values must be calculated on the training dataset and applied to scale the test dataset and any forecasts. This is to avoid contaminating the experiment with knowledge from the test dataset, which might give the model a small edge.

We can transform the dataset to the range [-1, 1] using the MinMaxScaler class. Like other scikit-learn transform classes, it requires data provided in a matrix format with rows and columns. Therefore, we must reshape our NumPy arrays before transforming.

For example:

```
# transform scale
X = series.values
X = X.reshape(len(X), 1)
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(X)
scaled_X = scaler.transform(X)
```

Again, we must invert the scale on forecasts to return the values back to the original scale so that the results can be interpreted and a comparable error score can be calculated.

```
# invert transform
inverted_X = scaler.inverse_transform(scaled_X)
```

Putting all of this together, the example below transforms the scale of the Shampoo Sales data.

```
from pandas import read_csv
from pandas import datetime
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
# 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
print(series.head())
# transform scale
X = series.values
X = X.reshape(len(X), 1)
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(X)
scaled_X = scaler.transform(X)
scaled_series = Series(scaled_X[:, 0])
print(scaled_series.head())
# invert transform
inverted_X = scaler.inverse_transform(scaled_X)
inverted_series = Series(inverted_X[:, 0])
print(inverted_series.head())
```


Running the example first prints the first 5 rows of the loaded data, then the first 5 rows of the scaled data, then the first 5 rows with the scale transform inverted, matching the original data.

```
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
0   -0.478585
1   -0.905456
2   -0.773236
3   -1.000000
4   -0.783188
dtype: float64

0    266.0
1    145.9
2    183.1
3    119.3
4    180.3
dtype: float64
```

Now that we know how to prepare data for the LSTM network, we can start developing our model.

LSTM Model Development

The Long Short-Term Memory network (LSTM) is a type of Recurrent Neural Network (RNN).

A benefit of this type of network is that it can learn and remember over long sequences and does no on as input.

In Keras, this is referred to as stateful, and involves setting the "*stateful*" argument to "*True*" when de

By default, an LSTM layer in Keras maintains state between data within one batch. A batch of data is dataset that defines how many patterns to process before updating the weights of the network. State default, therefore we must make the LSTM stateful. This gives us fine-grained control over when state of the LSTM layer is cleared, by calling the *reset_states()* function.

The LSTM layer expects input to be in a matrix with the dimensions: [*samples, time steps, features*].

- **Samples**: These are independent observations from the domain, typically rows of data.
- **Time steps**: These are separate time steps of a given variable for a given observation.
- **Features**: These are separate measures observed at the time of observation.

We have some flexibility in how the Shampoo Sales dataset is framed for the network. We will keep it simple and frame the problem as each time step in the original sequence is one separate sample, with one timestep and one feature.

Given that the training dataset is defined as X inputs and y outputs, it must be reshaped into the Samples/TimeSteps/Features format, for example:

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

The shape of the input data must be specified in the LSTM layer using the "*batch_input_shape*" argument as a tuple that specifies the expected number of observations to read each batch, the number of time steps, and the number of features.

The batch size is often much smaller than the total number of samples. It, along with the number of epochs, defines how quickly the network learns the data (how often the weights are updated).

The final import parameter in defining the LSTM layer is the number of neurons, also called the num y simple problem and a number between 1 and 5 should be sufficient.

The line below creates a single LSTM hidden layer that also specifies the expectations of the input la

```
layer = LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=Tr
```

The network requires a single neuron in the output layer with a linear activation to predict the numbe

Once the network is specified, it must be compiled into an efficient symbolic representation using a b or Theano.

In compiling the network, we must specify a loss function and optimization algorithm. We will use "*m* ely matches RMSE that we will are interested in, and the efficient ADAM optimization algorithm.

Using the Sequential Keras API to define the network, the below snippet creates and compiles the network.

```
model = Sequential()
model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Once compiled, it can be fit to the training data. Because the network is stateful, we must control when the internal state is reset. Therefore, we must manually manage the training process one epoch at a time across the desired number of epochs.

By default, the samples within an epoch are shuffled prior to being exposed to the network. Again, this is undesirable for the LSTM because we want the network to build up state as it learns across the sequence of observations. We can disable the shuffling of samples by setting "*shuffle*" to "*False*".

Also by default, the network reports a lot of debug information about the learning progress and skill of the model at the end of each epoch. We can disable this by setting the "*verbose*" argument to the level of "*0*".

We can then reset the internal state at the end of the training epoch, ready for the next training iteration.

Below is a loop that manually fits the network to the training data.

```
for i in range(nb_epoch):
    model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
    model.reset_states()
```

Putting this all together, we can define a function called *fit_lstm()* that trains and returns an LSTM mo a supervised learning format, a batch size, a number of epochs, and a number of neurons.

```
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]), stat
    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
```

The batch_size must be set to 1. This is because it must be a factor of the size of the training and test datasets.

The *predict()* function on the model is also constrained by the batch size; there it must be set to 1 because we are interested in making one-step forecasts on the test data.

We will not tune the network parameters in this tutorial; instead we will use the following configuratio

- Batch Size: 1
- Epochs: 3000
- Neurons: 4

As an extension to this tutorial, you might like to explore different model parameters and see if you can improve performance.

- **Update**: Consider trying 1500 epochs and 1 neuron, the performance may be better!

Next, we will look at how we can use a fit LSTM model to make a one-step forecast.

LSTM Forecast

Once the LSTM model is fit to the training data, it can be used to make forecasts.

Again, we have some flexibility. We can decide to fit the model once on all of the training data, then p test data (we'll call this the fixed approach), or we can re-fit the model or update the model each time ste st data are made available (we'll call this the dynamic approach).

In this tutorial, we will go with the fixed approach for its simplicity, although, we would expect the dyn

To make a forecast, we can call the *predict()* function on the model. This requires a 3D NumPy array ray of one value, the observation at the previous time step.

The *predict()* function returns an array of predictions, one for each input row provided. Because we a NumPy array with one value.

We can capture this behavior in a function named *forecast()* listed below. Given a fit model, a batch-size used when fitting the model (e.g. 1), and a row from the test data, the function will separate out the input data from the test row, reshape it, and return the prediction as a single floating point value.

```
1 def forecast(model, batch_size, row):
2     X = row[0:-1]
3     X = X.reshape(1, 1, len(X))
4     yhat = model.predict(X, batch_size=batch_size)
5     return yhat[0,0]
```

During training, the internal state is reset after each epoch. While forecasting, we will not want to reset the internal state between forecasts. In fact, we would like the model to build up state as we forecast each time step in the test dataset.

This raises the question as to what would be a good initial state for the network prior to forecasting the test dataset.

In this tutorial, we will seed the state by making a prediction on all samples in the training dataset. In theory, the internal state should be set up ready to forecast the next time step.

We now have all of the pieces to fit an LSTM Network model for the Shampoo Sales dataset and evaluate its performance.

In the next section, we will put all of these pieces together.

Complete LSTM Example

In this section, we will fit an LSTM to the Shampoo Sales dataset and evaluate the model.

This will involve drawing together all of the elements from the prior sections. There are a lot of them,

1. Load the dataset from CSV file.
2. Transform the dataset to make it suitable for the LSTM model, including:
    1. Transforming the data to a supervised learning problem.
    2. Transforming the data to be stationary.
    3. Transforming the data so that it has the scale -1 to 1.
3. Fitting a stateful LSTM network model to the training data.
4. Evaluating the static LSTM model on the test data.
5. Report the performance of the forecasts.

Some things to note about the example:

- The scaling and inverse scaling behaviors have been moved to the functions *scale()* and *invert_scale()* for brevity.
- The test data is scaled using the fit of the scaler on the training data, as is required to ensure the min/max values of the test data do not influence the model.
- The order of data transforms was adjusted for convenience to first make the data stationary, the

Get Your Start in Machine Learning

- Differencing was performed on the entire dataset prior to splitting into train and test sets for convenience. We could just as easily collect observations during the walk-forward validation and difference them as we go. I decided against it for readability.

The complete 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
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.fillna(0, inplace=True)
    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, value):
    new_row = [x for x in X] + [value]
    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]), sta
    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

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]

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

# 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 the model
lstm_model = fit_lstm(train_scaled, 1, 3000, 4)
# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
lstm_model.predict(train_reshaped, batch_size=1)

# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)
    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # invert differencing
    yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
    # store forecast
    predictions.append(yhat)
    expected = raw_values[len(train) + i + 1]
    print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# report performance
rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
print('Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(raw_values[-12:])
pyplot.plot(predictions)
pyplot.show()
```


Running the example prints the expected and predicted values for each of the 12 months in the test dataset.

The example also prints the RMSE of all forecasts. The model shows an RMSE of 71.721 monthly shampoo sales, which is better than the persistence model that achieved an RMSE of 136.761 shampoo sales.

Random numbers are used in seeding the LSTM, and as a result, you may have a different result from a single run of the model. We cover this further in the next section.

```
Month=1, Predicted=351.582196, Expected=339.700000
Month=2, Predicted=432.169667, Expected=440.400000
Month=3, Predicted=378.064505, Expected=315.900000
Month=4, Predicted=441.370077, Expected=439.300000
Month=5, Predicted=446.872627, Expected=401.300000
Month=6, Predicted=514.021244, Expected=437.400000
Month=7, Predicted=525.608903, Expected=575.500000
Month=8, Predicted=473.072365, Expected=407.600000
Month=9, Predicted=523.126979, Expected=682.000000
Month=10, Predicted=592.274106, Expected=475.300000
Month=11, Predicted=589.299863, Expected=581.300000
Month=12, Predicted=584.149152, Expected=646.900000
Test RMSE: 71.721
```

A line plot of the test data (blue) vs the predicted values (orange) is also created, providing context for the model skill.

Line Plot of LSTM Forecast vs Expected Values

As an afternote, you can do a quick experiment to build your trust in the test harness and all of the transforms and inverse transforms.

Comment out the line that fits the LSTM model in walk-forward validation:

```
yhat = forecast_lstm(lstm_model, 1, X)
```

And replace it with the following:

```
yhat = y
```

This should produce a model with perfect skill (e.g. a model that predicts the expected outcome as the model output).

The results should look as follows, showing that if the LSTM model could predict the series perfectly, the inverse transforms and error calculation would show it correctly.

```
Month=1, Predicted=339.700000, Expected=339.700000
Month=2, Predicted=440.400000, Expected=440.400000
Month=3, Predicted=315.900000, Expected=315.900000
Month=4, Predicted=439.300000, Expected=439.300000
Month=5, Predicted=401.300000, Expected=401.300000
Month=6, Predicted=437.400000, Expected=437.400000
Month=7, Predicted=575.500000, Expected=575.500000
Month=8, Predicted=407.600000, Expected=407.600000
Month=9, Predicted=682.000000, Expected=682.000000
Month=10, Predicted=475.300000, Expected=475.300000
Month=11, Predicted=581.300000, Expected=581.300000
Month=12, Predicted=646.900000, Expected=646.900000
Test RMSE: 0.000
```


Develop a Robust Result

A difficulty with neural networks is that they give different results with different starting conditions.

One approach might be to fix the random number seed used by Keras to ensure the results are repr r the random initial conditions using a different experimental setup.

For more on randomness in machine learning, see the post:

- Embrace Randomness in Machine Learning

We can repeat the experiment from the previous section multiple times, then take the average RMSE as an indication of how well the configuration would be expected to perform on unseen data on average.

This is often called multiple repeats or multiple restarts.

We can wrap the model fitting and walk-forward validation in a loop of fixed number of repeats. Each iteration the RMSE of the run can be recorded. We can then summarize the distribution of RMSE scores.

```
# repeat experiment
repeats = 30
error_scores = list()
for r in range(repeats):
    # fit the model
    lstm_model = fit_lstm(train_scaled, 1, 3000, 4)
    # forecast the entire training dataset to build up state for forecasting
    train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
    lstm_model.predict(train_reshaped, batch_size=1)
    # walk-forward validation on the test data
    predictions = list()
    for i in range(len(test_scaled)):
        # make one-step forecast
        X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
        yhat = forecast_lstm(lstm_model, 1, X)
        # invert scaling
        yhat = invert_scale(scaler, X, yhat)
        # invert differencing
        yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
        # store forecast
        predictions.append(yhat)
    # report performance
    rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
    print('%d) Test RMSE: %.3f' % (r+1, rmse))
    error_scores.append(rmse)
```

The data preparation would be the same as before.

We will use 30 repeats as that is sufficient to provide a good distribution of RMSE scores.

The complete 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
from matplotlib import pyplot
import numpy
```

```
14
15 # date-time parsing function for loading the dataset
16 def parser(x):
17     return datetime.strptime('190'+x, '%Y-%m')
18
19 # frame a sequence as a supervised learning problem
20 def timeseries_to_supervised(data, lag=1):
21     df = DataFrame(data)
22     columns = [df.shift(i) for i in range(1, lag+1)]
23     columns.append(df)
24     df = concat(columns, axis=1)
25     df.fillna(0, inplace=True)
26     return df
27
28 # create a differenced series
29 def difference(dataset, interval=1):
30     diff = list()
31     for i in range(interval, len(dataset)):
32         value = dataset[i] - dataset[i - interval]
33         diff.append(value)
34     return Series(diff)
35
36 # invert differenced value
37 def inverse_difference(history, yhat, interval=1):
38     return yhat + history[-interval]
39
40 # scale train and test data to [-1, 1]
41 def scale(train, test):
42     # fit scaler
43     scaler = MinMaxScaler(feature_range=(-1, 1))
44     scaler = scaler.fit(train)
45     # transform train
46     train = train.reshape(train.shape[0], train.shape[1])
47     train_scaled = scaler.transform(train)
48     # transform test
49     test = test.reshape(test.shape[0], test.shape[1])
50     test_scaled = scaler.transform(test)
51     return scaler, train_scaled, test_scaled
52
53 # inverse scaling for a forecasted value
54 def invert_scale(scaler, X, value):
55     new_row = [x for x in X] + [value]
56     array = numpy.array(new_row)
57     array = array.reshape(1, len(array))
58     inverted = scaler.inverse_transform(array)
59     return inverted[0, -1]
60
```

```
# 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]), stateful=True))
    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

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]

# 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)

# repeat experiment
repeats = 30
error_scores = list()
for r in range(repeats):
    # fit the model
    lstm_model = fit_lstm(train_scaled, 1, 3000, 4)
    # forecast the entire training dataset to build up state for forecasting
    train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
    lstm_model.predict(train_reshaped, batch_size=1)
    # walk-forward validation on the test data
    predictions = list()
```

```
    for i in range(len(test_scaled)):
        # make one-step forecast
        X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
        yhat = forecast_lstm(lstm_model, 1, X)
        # invert scaling
        yhat = invert_scale(scaler, X, yhat)
        # invert differencing
        yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
        # store forecast
        predictions.append(yhat)
    # report performance
    rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
    print('%d) Test RMSE: %.3f' % (r+1, rmse))
    error_scores.append(rmse)

# summarize results
results = DataFrame()
results['rmse'] = error_scores
print(results.describe())
results.boxplot()
pyplot.show()
```

Running the example prints the RMSE score each repeat. The end of the run provides summary sta

We can see that the mean and standard deviation RMSE scores are 138.491905 and 46.313783 mo

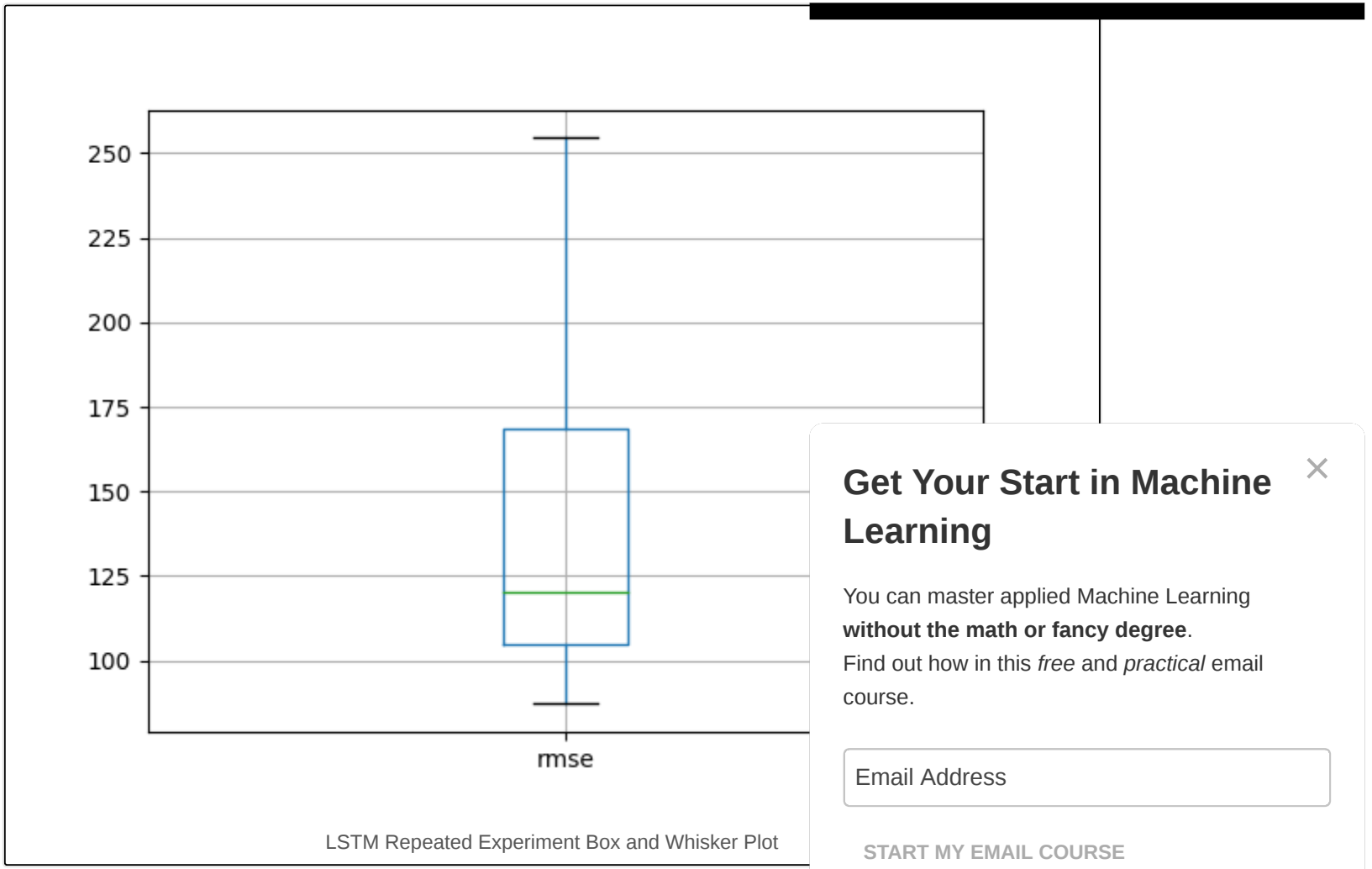
This is a very useful result as it suggests the result reported above was probably a statistical fluke. T y about as good as the persistence model on average (136.761), if not slightly worse.

This indicates that, at the very least, further model tuning is required.

```
1) Test RMSE: 136.191
2) Test RMSE: 169.693
3) Test RMSE: 176.553
4) Test RMSE: 198.954
5) Test RMSE: 148.960
6) Test RMSE: 103.744
7) Test RMSE: 164.344
8) Test RMSE: 108.829
9) Test RMSE: 232.282
10) Test RMSE: 110.824
11) Test RMSE: 163.741
12) Test RMSE: 111.535
13) Test RMSE: 118.324
```

```
14) Test RMSE: 107.486
15) Test RMSE: 97.719
16) Test RMSE: 87.817
17) Test RMSE: 92.920
18) Test RMSE: 112.528
19) Test RMSE: 131.687
20) Test RMSE: 92.343
21) Test RMSE: 173.249
22) Test RMSE: 182.336
23) Test RMSE: 101.477
24) Test RMSE: 108.171
25) Test RMSE: 135.880
26) Test RMSE: 254.507
27) Test RMSE: 87.198
28) Test RMSE: 122.588
29) Test RMSE: 228.449
30) Test RMSE: 94.427
              rmse
count   30.000000
mean   138.491905
std     46.313783
min     87.198493
25%    104.679391
50%    120.456233
75%    168.356040
max    254.507272
```

A box and whisker plot is created from the distribution shown below. This captures the middle of the



LSTM Repeated Experiment Box and Whisker Plot

This is an experimental setup that could be used to compare one configuration of the LSTM model or set up to another.

Tutorial Extensions

There are many extensions to this tutorial that we may consider.

Perhaps you could explore some of these yourself and post your discoveries in the comments below.

- **Multi-Step Forecast**. The experimental setup could be changed to predict the next *n*-time steps rather than the next single time step. This would also permit a larger batch size and faster training. Note that we are basically performing a type of 12 one-step forecast in this tutorial given the model is not updated, although new observations are available and are used as input variables.
- **Tune LSTM model**. The model was not tuned; instead, the configuration was found with some quick trial and error. I believe much better results could be achieved by tuning at least the number of neurons and number of training epochs. I also think early stopping via a callback might be useful during training.
- **Seed State Experiments**. It is not clear whether seeding the system prior to forecasting by predicting all of the training data is beneficial. It seems like a good idea in theory, but this needs to be demonstrated. Also, perhaps other methods of seeding the model prior to forecasting would be beneficial.
- **Update Model**. The model could be updated in each time step of the walk-forward validation. Experiments are needed to determine if it would be better to refit the model from scratch or update the weights with a few more training epochs including the new sample.
- **Input Time Steps**. The LSTM input supports multiple time steps for a sample. Experiments are needed to ... steps provides any benefit.
- **Input Lag Features**. Lag observations may be included as input features. Experiments are nee... benefit, not unlike an AR(k) linear model.
- **Input Error Series**. An error series may be constructed (forecast error from a persistence mode... an MA(k) linear model. Experiments are needed to see if this provides any benefit.
- **Learn Non-Stationary**. The LSTM network may be able to learn the trend in the data and make... see if temporal dependent structures, like trends and seasonality, left in data can be learned and...
- **Contrast Stateless**. Stateful LSTMs were used in this tutorial. The results should be compared...
- **Statistical Significance**. The multiple repeats experimental protocol can be extended further to... whether the difference between populations of RMSE results with different configurations are st...

Summary

In this tutorial, you discovered how to develop an LSTM model for time series forecasting.

Specifically, you learned:

- How to prepare time series data for developing an LSTM model.
- How to develop an LSTM model for time series forecasting.
- How to evaluate an LSTM model using a robust test harness.

Can you get a better result?
Share your findings in the comments below.

About Jason Brownlee

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

View all posts by Jason Brownlee →

‹ Seasonal Persistence Forecasting With Python

How to Seed State for LSTMs for Time Series Forecasting in Python ›

235 Responses to *Time Series Forecasting with the Long Short-Term Memory Network in Python*

Chang April 7, 2017 at 4:42 pm #

REPLY

I've been working on multi-step-ahead forecast after reading your ebook and following one step
seems that seq2seq model is used, but I want to configure simple lstm for multi step ahead prediction. C

Jason Brownlee April 9, 2017 at 2:53 pm #

Yes, I have some seq2seq examples scheduled to come out on the blog soon.

Supriya April 9, 2017 at 9:49 am #

Hi Jason,
Thank you for this blog. I am working on multi-step-ahead forecast using recursive prediction technique and I have some difficulty. Blog on this particular topic would be really helpful.
Also, is it possible to somehow implement recursive technique in ARIMA?.

Jason Brownlee April 9, 2017 at 3:01 pm #

REPLY

Absolutely, you can take the predictions as history and re-fit the ARIMA.

Get Your Start in Machine Learning

Thanks for the suggestion, it would make a good blog post.

REPLY

Jack Dan July 28, 2017 at 5:13 am #

I am wondering if I have predictions (not good prediction) and refitting without validating with the test set, wouldn't it give false prediction again?

REPLY

Jason Brownlee July 28, 2017 at 8:34 am #

It may, design an experiment to test the approach.

Peter Marelas April 9, 2017 at 2:45 pm #

Hi Jason,

LSTM remember sequences but is there a way to encode calendar effects in the network so that it reme within the sequence and each cycles? A concrete example would be a time series that exhibits specific e year, example first Monday of every month and/or last day of every month. I am thinking whether we car these events better?

REPLY

Guillaume July 10, 2017 at 8:12 pm #

Hello Peter,

You might want to check out the X-11 method to separate trend, seasonal, and random change to your sequence. Then apply an algorithm to each part.

You can look at the following article :
Study of the Long-term Performance Prediction Methods Using the Spacecraft Telemetry Data

from Hongzeng Fang
(Sorry but I can't find a free dl page anymore ..).

Jason Brownlee July 11, 2017 at 10:29 am #

REPLY

Thanks for the tip.

Kunpeng Zhang April 10, 2017 at 5:04 am #

REPLY

Hi Jason,
Nice post. Is there any tutorial available on multivariate time series forecasting problem?
I got two sets of data: traffic flow data and weather data. I am thinking to predict the traffic flow using the
I'd like to learn if I get weather condition involved what will happen for my model.
Could you kindly give me some advice?
Thank you.

Jason Brownlee April 10, 2017 at 7:40 am #

I do not have a multivariate regression example, but I hope to post one soon.

Kunpeng Zhang April 12, 2017 at 10:42 am #

REPLY

Great. Thank you in advance.

galdo trouchky April 12, 2017 at 7:46 pm #

Get Your Start in Machine Learning

That would be awesome, indeed

Wenhan Wang September 29, 2017 at 10:08 pm #

REPLY

Hi Jason. It is really a nice example of using LSTM. I'm working on it.
I also have the same question, are you going to give an example on multivariate time series forecasting? I'm struggling on it.

Jason Brownlee September 30, 2017 at 7:41 am #

REPLY

I have an example here:
https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

Gabriel Beauplet April 13, 2017 at 12:21 am #

You did a great job. This is very detailed ! Bravo 😉

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

I'm glad you found it useful Gabriel.

Hand April 19, 2017 at 1:39 pm #

REPLY

Great tutorial,

How do we get a prediction on a currently not existing point in the future?

Jason Brownlee April 20, 2017 at 9:22 am #

REPLY

yhat = model.predict(X)

Hans April 20, 2017 at 1:44 am #

REPLY

I mean without a y.

Error measurements are cool, but I also want to make a prediction for a next step.
That's my main intention to read this tutorial.

Since there are other tutorials out there, lacking the same issue, it would be great to have a complete ex

I'm aware of the fact that every problem should have its own solution/model, but with a real result (or a c
practical/reusable- especially for python-ml beginners trying to predict some values in the future.

Jason Brownlee April 20, 2017 at 9:30 am #

Fit your model on the entire training dataset then predict the next time step as:

```
1 yhat = model.predict(X)
```

Donna May 22, 2017 at 7:07 am #

REPLY

Hi Jason, thanks for your tutorial. But can you specify how to fit the model on the entire training set and do the prediction for future point?
Thanks.

Jason Brownlee May 22, 2017 at 7:56 am #

REPLY

That is just a little too much hand holding Donna. What are you having trouble with exactly?

Hans April 20, 2017 at 2:14 pm #

REPLY

Thank you Jason,

I'm new to Python (mainly coding other languages) and just beginning to understand the code- thanks to your outstanding detailed descriptions.

In the last weeks I tried out two other tutorials and failed exactly at this point (making a first own test-appl[illegible]

A) Could you please suggest a location/row number in the code in one of the examples for that line?

B) Is there a magic trick available to avoid the date conversion and work with real dates in own data sets
In the moment I would have to transform my data to that date format of the used raw data.

I'm afraid to break the code logic.

But I also try to research every crucial part of the code separately for example here:
https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html

My current Python skills are on the level of this tutorial
https://www.youtube.com/watch?v=N4mEzFDjqtA .

Jason Brownlee April 21, 2017 at 8:30 am #

REPLY

Hi Hans,

What do you mean by "first own test-appliance with a result". I don't follow.

Do you mean make a prediction? If so you can make a prediction by fitting the model no all of your data and calling model.predict(X).

Pandas is really flexible when it comes to loading date data. A good approach for you might be to sp
loading your data. See this post for an example that does this:

http://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/

Hans April 20, 2017 at 3:39 pm #

REPLY

Worth to mention (latest Windows-Info 4.2017):

There is an issue with Keras/Anaconda on Windows.
To run the last example above on Win, we have to manually reinstall Keras.
Further informations can be found here:
https://github.com/llSourcell/How-to-Predict-Stock-Prices-Easily-Demo/issues/3#issuecomment-288981625
…otherwise it throws compiling errors.

Jason Brownlee April 21, 2017 at 8:30 am #

Does this tutorial work for you?
http://machinelearningmastery.com/setup-python-environment-machine-learning-deep-learning-anac

Hans April 21, 2017 at 11:12 am #

I already saw this tutorial written by you.

On Windows (not a Mac) it's a slightly different story to build such an environment, even with An

It begins with the fact that there is no Tensorflow with a version <= 3 for Windows and ends with the Keras-hint.

We needed several days to discuss it on Github and find the right setup in the context of another RNN-script (see the link).

I use virtual conda-environments and your scripts are running on windows if the keras-hint is implemented.
Before I had the same issue with some of your scripts then with the discussed one on Github (compile error).

Hans April 20, 2017 at 5:02 pm #

REPLY

Update, what I mean:

I guess one could trigger “forecast_lstm(model, batch_size, X)” or “yhat = model.predict(X)” from the end of the last two example scripts.

But how to do that in regard to the trained model?

“Month=13, Predicted=???”

Do I have to define a new “fictional” X? And if so how?

Jason Brownlee April 21, 2017 at 8:32 am #

You must load the new input data for which a prediction is required as X and use your alrea
predict() function.

I’m eager to help, but perhaps I don’t understand the difficulty exactly?

Hans April 21, 2017 at 10:31 am #

Hello Jason,

I knew I shouldn’t mention the date conversion part :-). Meanwhile I managed it with “return datetime.str
I have all of your example script parts in separate python file version. So I can test out modifications for

```
python basic_data_loading.py
python persistence_forecast_model.py
python transform_time_series_to_supervised.py
python transform_time_series_to_statonary_remove_trends.py
python transform_scales.py
python complete_example.py
```

Own data loading is solved. That was a relatively easy task.

Hans April 21, 2017 at 10:56 am #

REPLY

Since the transforming and performance measurement parts are running (I guess they will do even with integer data)
I now have to build a part lets call it:

“python predict_one_value.py”

Of course I have to load my own data that’s clear.

The question is where to trigger the function

```
yhat = model.predict(X)
```

in the context of one of your example-scripts and finally say:
print(yhat). That’s all.

I guess a short example snippet could solve the problem.
Could you provide an example- It would help a lot?

Currently I also don’t understand the role of X completely.
In the context of “datetime.strptime” it seems to be a date only, If I print it out.
So if I would have training data of:
– 01.12.1977
– 02.12.1977
– 03.12.1977
I would guess I could say something like “yhat = model.predict(“1977-12-04″)”.
The question is where and when in which code context.

Thank you.

Hans April 21, 2017 at 12:06 pm #

Get Your Start in Machine Learning

Update:

Currently I use the the code from “complete example” (“without robust result”).
If I comment out from line 106 to line 127 and then at the end of the script say:

```
# report one value in the future
test = datetime.strptime('2017-04-15', '%Y-%m-%d')
yhat = model.predict(test)
print(yhat)
```

I get the error message ‘model is not defined’. So trying…

```
# report one value in the future
test = datetime.strptime('2017-04-15', '%Y-%m-%d')
yhat = lstm_model.predict(test)
print(yhat)
```

…throws the error “Data should be a Numby array”.

I guess maybe I could also append a new date to the raw data (without a y),
but I’m not sure If this would be right.

The best way to get this running would be an example snipped in context.

Jason Brownlee April 22, 2017 at 9:22 am #

This post will give you a clearer idea of how to use neural nets in Keras including making predictions:
http://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/

Yes, input data to making a prediction must be a 2D numpy array. It will not be date data, it will be past obs in the case of time series forecasting.

Hans April 22, 2017 at 11:41 am #

Get Your Start in Machine Learning

Thank you,

I will read it ;-).

Hans April 22, 2017 at 12:46 pm #

I have read the tutorial and tried out the pima indians diabetes example.
I guess I got it and understand the 5 steps (mostly).

Unfortunately this does not answer my question. Or do I miss something?
In my problem I have only one input like in the tutorial on this site.

When you say:

“yhat = model.predict(X)”

would give a forecast for a next step.
What about a step which is not in the training data nor in the test data?

I have a SVM model which proves and predicts based on my raw data (realized in anoth
Lets say I have 100 items training data, 10 items test data.
It will printout 10 predictions and additionally corresponding performance data.
The last printed prediction is for a future step which lies in the future.

How would this be archived in your example?
Do I have to shift something?

Jason Brownlee April 23, 2017 at 5:14 am #

To make predictions beyond your dataset, you must feed in the last few observations from your dataset as input (X) to predict what happens next (y).

This post might clear up your thinking on X and y:
http://machinelearningmastery.com/time-series-forecasting-supervised-learning/

Get Your Start in Machine Learning

Hans April 23, 2017 at 1:27 pm #

>To make predictions beyond your dataset, you must feed in the last few observations from your dataset as input (X) to >predict what happens next (y).

Is there an example available where this is done?

Jason Brownlee April 24, 2017 at 5:32 am #

Yes, the “LSTM Forecast” section of this very post.

Hans April 24, 2017 at 1:55 pm #

Assuming that “X = row[0:-1]” is an observation,
how do we sample/collect the last few observations, to make a forecast.?

Jason Brownlee April 25, 2017 at 7:43 am #

It depends on your model, if your model expects the last one observation as inp
it as X to model.predict(X).

If your model requires the last two lag obs as inputs, retrieve them, define them as a one row, two column matrix and provide them to model.predict().

And so on. I hope that helps.

Arun July 13, 2017 at 10:37 am #

HI Jason,

Even i started with machine learning and have similar kind of doubt so in one step forecasting we can only get one time step future observation correct ? and to get the prediction i have provided last input observation and then the value obtained from model.predict(X) has to be again scaled and inversed correct ?

PFB the code:
X = test_scaled[3,-1:] (my last observation)
yhat = forecast_lstm(lstm_model, 1, X)
yhat = invert_scale(scaler, X, yhat)
yhat = inverse_difference(raw_values, yhat, 1)
print(yhat)

Can you please guide me if i am going in a right way ?

Jason Brownlee July 13, 2017 at 4:53 pm #

Yes, one step forecasting involves predicting the next time step.

You can make multi-step forecasts, learn more in this post:
http://machinelearningmastery.com/multi-step-time-series-forecasting/

Yes, to make use of the prediction you will need to invert any data transforms performed

Arun Menon July 13, 2017 at 11:32 pm #

Thank you Jason ..

Get Your Start in Machine Learning

Arun Menon July 13, 2017 at 11:48 pm #

Hi Jason,

Being said that, i have another clarification , so when i forecast the next time step using this model , using the below code:
X = test_scaled[3,-1:] (my last observation)
yhat = forecast_lstm(lstm_model, 1, X)
yhat = invert_scale(scaler, X, yhat)
yhat = inverse_difference(raw_values, yhat, 1)
print(yhat)

in the above code, let yhat be the prediction of future time step, can i use the result of yhat and use the same model to predict one more step ahead in the future ? is this what we call as the recursive multistep forecast ?

Jason Brownlee July 14, 2017 at 8:30 am #

Yes. This is recursive.

Arun Menon July 14, 2017 at 1:40 am #

Hi Jason,

Can i use the below code and use the recursive multistep forecast
for eg :
yhat value can be used as an input to the same model again to get the next future step and so on ?

X = test_scaled[3,-1:] (my last observation)
yhat = forecast_lstm(lstm_model, 1, X)
yhat = invert_scale(scaler, X, yhat)
yhat = inverse_difference(raw_values, yhat, 1)
print(yhat)

Arun Menon July 14, 2017 at 11:59 am #

Hi Jason,

In predictive analytics using this ML technique, how many future steps should we able to predict , is there any ideal forecasting range in future for eg if i have a data for the last 10 days or so , and i want to forecast the future , the less the future time steps are set, the better the result as the error will be minimum right. Can i use the same code for predicting time series data in production for network traffic for future 3 days ? requirement given for me was to predict the network bandwidth for the next entire week given the data for past 1 year.

Your comments and suggestions always welcome 🙂

Regards,
Arun

Jason Brownlee July 15, 2017 at 9:37 am #

It depends on the problem and the model.

Generally, the further in the future you want to predict, the worse the performance of the

Hans April 21, 2017 at 6:37 pm #

Hello,

can we normalize the RMSE-value(s)?
And if so how?

Jason Brownlee April 22, 2017 at 9:24 am #

REPLY

Normalize the skill score?

Yes, but you will need to know the largest possible error.

Hans April 22, 2017 at 12:55 pm #

REPLY

I'm feeding your example with values ranged from 1 to 70.

There is no increasing trend in my raw data.

When it comes to predictions the script predicts values above 70.
Regarding to the other tutorial (5 Step Life-Cycle) I think it has to do with the compile part (model.compile).
But I'm not sure. Could you provide a comprehensible hint in regard of the example script on this site?

Fabian April 23, 2017 at 10:33 am #

Hi Jason,
assuming you had multiple features (you can add one feature to the shampoo dataset) and wanted to us
that I put into the model? Is it a 3 dimensional array, where the features are lists of values and each obs
(which is also a list of values)?

Jason Brownlee April 24, 2017 at 5:31 am #

Good question, it would be a 3D array with the dimensions [samples, timesteps, features].

Rahul April 26, 2017 at 5:33 am #

REPLY

Right now the model gives only one step forecast. What if I wanted to create a model which giv

Jason Brownlee April 26, 2017 at 6:24 am #

REPLY

I will have a post on this see, until then see this post:
http://machinelearningmastery.com/multi-step-time-series-forecasting/

Kunpeng Zhang April 27, 2017 at 1:07 am #

REPLY

Hi Jason, I have a question.
raw_values = series.values
diff_values = difference(raw_values, 1)
print(len(raw_values)) 36
print(len(diff_values)) 35

So, after difference we lose the first value?

Jason Brownlee April 27, 2017 at 8:43 am #

Correct.

Guido van Steen April 27, 2017 at 2:27 am #

REPLY

Dear Jason,

Thank you very much for this extremely useful and interesting post.

I may be missing something, but I think there is one omission: By differencing you loose the trend including its start and end level. Later on you try restore the trend again, but in your code it seems you fail to restore the end level. IMO the end level of the current observations should be added to all the predictions.

Thanks again!

Get Your Start in Machine Learning

Jason Brownlee April 27, 2017 at 8:44 am #

REPLY

What do you mean by the end level? Sorry I don't follow, perhaps you could restate your comment?

Raul April 27, 2017 at 3:32 am #

REPLY

Great tutorial!
Quick question: On the scaling section of the tutorial you say that

"To make the experiment fair, the scaling coefficients (min and max) values must be calculated on the tra
and any forecasts. This is to avoid contaminating the experiment with knowledge from the test dataset, v

However, if the max of your sample is on the test dataset the scaling with parameters from the training s
can one deal with that?

Thanks!

Jason Brownlee April 27, 2017 at 8:46 am #

Correct.

One good approach is to estimate the expected min and max values that are possible for the domai

If even then you see values out of range, you can clamp them to the bounds 0/1.

Guido van Steen April 27, 2017 at 6:20 am #

REPLY

Dear Jason,

Get Your Start in Machine Learning

To be more precise: you should add the difference between the start level and the end level of the train set. This is because the current code effectively replicates the train set. By this I means that it starts at the same level as the train set. However, it should start at the end level of the train set.

Kind regards,

Guido

Guido van Steen April 27, 2017 at 3:11 pm #

REPLY

I will try to restate my comment:

Currently the predictions (of your test set) start at the same level as the observations (in your train set). Therefore, there is a shift between the last observed value (in your train set) and the first predicted value (of your test set). The size of this shift level of the observations (in your train set). You should correct for this shift by adding it to the predict

Jason Brownlee April 28, 2017 at 7:35 am #

Isn't this made moot by making the data stationary?

Deeps April 28, 2017 at 4:59 am #

Hello,

1. can you please explain me the below 2 lines in detail.

model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
model.add(Dense(1))

2. I want to know the layer architecture. What are the number of neurons in hidden layer?
3. If I want to add one more hidden layer, how the syntax looks like?
4. What could be the reason for test rmse is less than train rmse?

Jason Brownlee April 28, 2017 at 7:56 am #

REPLY

The line defines the input layer and the first LSTM hidden layer with “neurons” number of memory units.

You can stack LSTMs, the first hidden layer must return sequences (return_sequence=True) and you can add a second LSTM layer. I have an example here:
http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Better performance on the test data than the training data may be a sign of an unstable/underfit model.

Fabio May 2, 2017 at 9:43 pm #

Hallo Jason, thank you for this post! I bought the first version of your book and I have seen you
Very good! 🙂 Something have I yet not clear.

“Samples: These are independent observations from the domain, typically rows of data.
Time steps: These are separate time steps of a given variable for a given observation.”

I could understand the case where the time step parameter is 1 as in your book and in this example but
1…

My hypothesys, sure wrong 🙂

Perhaps when a timestep is made of n observations one could give the value n to it…but then I would e

“model.add(LSTM(4, input_shape=(1, look_back)))”

the LSTM would use (look_back * timesteps) rows for every step to predict the next row…

I cannot also understand why you say ‘of a given variable’…a row is normally built by the values of many variables, isn’t it?

Could you give me an example with timesteps > 1? Thank you!

Jason Brownlee May 3, 2017 at 7:36 am #

Hi Fabio,

The structure of the input to the LSTM is [samples, timesteps, features]

If you have multiple observations at one time step, then these are represented as features.

Does that help?

Fabio May 7, 2017 at 10:28 pm #

REPLY

Hello Jason,

unfortunately If don't have a concrete example I cannot fully understand…The examples in your
always on timesteps=1…if I'm not wrong. For example how could be adapted the szenario desc

Thank you very much!

PS. In the meantime I bought also your book on time series 🙂

Jason Brownlee May 8, 2017 at 7:44 am #

See this post on multi-step forecasting:
http://machinelearningmastery.com/multi-step-time-series-forecasting/

Nitin May 5, 2017 at 8:11 am #

REPLY

Thanks Jason for the wonderful tutorial!
I am using your tutorial to apply LSTM network on some syslog/network log data.

I have syslog data(a specific event) for each day for last 1 year and so I am using LSTM network for time series analysis.

As I understand from your tutorial.
1. A batch of data is a fixed-sized number of rows from the training dataset that defines how many patte

network. Based on the batch_size the Model takes random samples from the data for the analysis. For time series this is not desirable, hence the batch_size should always be 1.
2. By default, the samples within an epoch are shuffled prior to being exposed to the network. This is undesirable for the LSTM because we want the network to build up state as it learns across the sequence of observations. We can disable the shuffling of samples by setting “shuffle” to “False“.

Scenario1 –
Using above two rules/guidelines – I ran several trials with different number of neurons, epoch size and different layers and got better results from the baseline model(persistence model).

Scenario2-
Without using above guidelines/rules – I ran several trials with different number of neurons, epoch size and different layers and got even better results than Scenario 1.

Query – Setting shuffle to True and Batch_size values to 1 for time series. Is this a rule or a guideline?

It seems logical reading your tutorial that the data for time series should not be shuffled as we do not wa
the results are better if I let the data be shuffled.
At the end what I think, what matters is how I get better predictions with my runs.
I think I should try and put away “theory” over concrete evidence, such as metrics, elbows, RMSEs,etc.

Kindly enlighten.

Jason Brownlee May 5, 2017 at 11:25 am #

Random samples are not taken, LSTMs require sequence data to be ordered – they learn c

The “shuffle” argument must be set to “False” on sequence prediction problems if sequence data is s

Masum May 7, 2017 at 12:44 am #

REPLY

Dear Jason,

I have two hours time series data which consists of 120 observations, using LSTM RNN how can I predict next 20 observation while putting all my data to training section.

We normally split the original data set in two data set (test dataset and validation dataset) for checking our model performance. I would like to see that my model is only taking help from training dataset to produce an output that does match with validation data set. What I understand from several of your blogs that we are forecasting single value and using that single forecasting along with the help of validation dataset we are forecasting rest of values. I believe I getting lost there? How it is going to work when have only past and current data (suppose no validation data) and we want to predict the next half an hour.

For example, suppose I have data of a product price from 12 to 1:30pm which consists of 90 observations and using these past 90 observations can we forecast the price of that product during 1:31 to 2:00pm(otherwise next 30 observations) .

Would you please help me to solve the confusion that I have? By the way I am going through your books time series forecasting with Python and deep learning with Python.

Jason Brownlee May 7, 2017 at 5:42 am #

You can make a multi-step model with an LSTM by using it directly (predict n timesteps in a model again and again and use predictions as inputs for subsequent predictions).

More on multi-step forecasting here:
http://machinelearningmastery.com/multi-step-time-series-forecasting/

Does that help?

masum May 7, 2017 at 5:57 am #

Thanks for quick replay

Looks like you sleep very less as you have provided feedback on early morning from your site.

Thanks for being so kind to your students. May god bless you.

By the way do you have any blog or example for second option that you have provided me(use the same model again and again and use predictions as inputs for subsequent predictions). Obviously I would like to see that I am not using any data from validation dataset or not getting any feedback from validation data. Model should only from past and current data along with current predictions. I hope you got this confused student.

Thanking you

Jason Brownlee May 8, 2017 at 7:41 am #

REPLY

Sorry, I don't have an example of using the model recursively. I believe you could adapt the above example.

masum May 9, 2017 at 9:10 am #

REPLY

would you please give me some hint where I have to change the code to make the model recursive as I am not very good at coding.

Jason Brownlee May 10, 2017 at 8:36 am #

Predictions would be used as history (input) to make predictions on the subsequent

Max May 9, 2017 at 11:10 am #

Thanks Jason for the wonderful tutorial!
A little problem here:

```
def invert_scale(scaler, X, value):
new_row = [x for x in X] + [yhat]
...
```

Does it should be

```
def invert_scale(scaler, X, value):
new_row = [x for x in X] + [value]
...
```

Jason Brownlee May 10, 2017 at 8:38 am #

REPLY

Thanks Max.

Charlie May 17, 2017 at 4:08 am #

REPLY

I'm getting the following error when running the first block of code you provide.

TypeError: strptime() argument 1 must be str, not numpy.ndarray

It seems like this a problem with how Python 3.6 handles byte strings.

Jason Brownlee May 17, 2017 at 8:42 am #

I have only tested the code on Python 2.7 and Python 3.5, perhaps try one of those?

Otherwise, you may need to adapt the example for your specific platform, if an API change is indeed

Logan May 18, 2017 at 10:28 am #

Hello, Your examples are great, thanks.

I have a question:

1- What do we do with the array of predicted values when we use lstm_model.predict() with train data in the example? It seems like they are not being used. Is it important for the net?

Jason Brownlee May 19, 2017 at 8:10 am #

REPLY

Thanks Logan.

What do you mean exactly? Is there a specific section and line of code you are referring to in the tutorial?

Logan May 27, 2017 at 2:47 am #

REPLY

Yes, line 101 lstm_model.predict(train_reshaped, batch_size=1) on section Complete LSTM Example.

By the way, I have another question: When the differentiation is applied to the entire dataset, isn't it giving information about the test dataset to the model?

Thanks. You've been helping me a lot.

Jason Brownlee June 2, 2017 at 11:58 am #

In this case yes, but in practice, we can apply the same method to the test data usi

Brandon May 19, 2017 at 12:34 pm #

Hi Jason,

Thanks so much for this helpful tutorial. Could you clarify which of all the parameters need to be updated to change the lag value? I haven't been able to get it to work.

Thanks!

Get Your Start in Machine Learning

Jason Brownlee May 20, 2017 at 5:34 am #

REPLY

Great question.

Change the parameters to the timeseries_to_supervised() function

Nirikshith May 29, 2017 at 8:23 pm #

REPLY

Hi Jason,

I have been following you for quite sometime now. Another great example for time series. I remember trying out sliding window approach in one of the earlier blogs. I am still waiting for you to write up a blog on involving external features(it can be as simple as (is_ implemented.

can you point out where we can add the above external variables in the above example and the format p new to python its very confusing with these reshapes and pre -format required for LSTM s

#aspring data scientist, student

Jason Brownlee June 2, 2017 at 12:25 pm #

You can add them as separate features.

I am working on an example at the moment. I hope to post it when it gives good results.

Donato June 1, 2017 at 10:00 pm #

REPLY

Hi Jason, nice example! I have your similar problem, but I don't know how resolve. I have a two different class to predict, in different files. I have to training my neural network with this different files. The problem is that each file has different number of samples, and I have to use the same LSTM for each file.How I can generalize the number of input for each LSTM??

This is my problem:
https://drive.google.com/file/d/0B5hOtU0Xa45RUDJJWHVyeHVNQWM/view?usp=sharing

Jason Brownlee June 2, 2017 at 12:59 pm #

REPLY

Maybe this post will help you better understand your prediction problem and frame it as supervised learning:
http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/

Donato June 5, 2017 at 8:24 pm #

REPLY

Thanks for the answere, I have understand which is my problem. It depends from the tr
when I insert a new file and remake the fit, the previusly information are forgotten. Do you have
And where each one has the supervisioned learning, apply to time-series and predict informatio

Jason Brownlee June 6, 2017 at 9:32 am #

Yes, this example shows how to update an existing model with more data:
http://machinelearningmastery.com/update-lstm-networks-training-time-series-forecasting/

Pengyuan Shao June 8, 2017 at 10:02 pm #

REPLY

Dear Jason:
You did a great job!!
Thank you for sharing the great post with us.
I'm a beginner to machine learning, these days i am busy on doing a project to predict the movement of ship on the sea.
I found this post is very usefull and easy to understand.
I have one question that:

Get Your Start in Machine Learning

if the number of neurons is the past steps used to predict next step?
Thank you for answer in advance.

Best wishes

REPLY

Jason Brownlee June 9, 2017 at 6:24 am #

I'm not sure what you mean.

The number of neurons defines the size/complexity of the model. The model is used to make predictions.

Perhaps you could reframe your question?

Macarena June 10, 2017 at 7:19 pm #

Hi,
Congrats for the blog, and thanks! It helped me a lot.
I have one question related to the data transformation. I understand the step to make that one input is th
info. But I have a different case (classification) and I don't know how could I apply this.
In my case my inputs are images (frames from a video) what have a temporal dependency, and my outp
images through a CNN network and then take the features and those would be my LSTM input. In that c
done here? I thought the LSTM would save that info by itself, but now I'm not so sure.

Thank you in advance.

REPLY

Jason Brownlee June 11, 2017 at 8:23 am #

I don't have an example of working with video data.

Generally, the principles would be the same. As a classification problem, you would need to change the output layer to use one of the log loss activation functions. The input would be from a CNN or similar, e.g. [cnn]->[lstm]->[dense].

Get Your Start in Machine Learning

Let me know how you go.

Kim Miller June 15, 2017 at 6:14 am #

REPLY

In your ARIMA tutorial you also use the shampoo data set, but measure accuracy by MSE. Making the change to RMSE:

```
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
```

I get:

```
> Test RMSE: 83.417
```

Can we conclude that ARIMA is far better than an LSTM for this problem so far in our research?

Jason Brownlee June 15, 2017 at 8:55 am #

No, these are demonstrations. Both algorithms would require tuning to present the best ver
model selection.

Kim Miller June 15, 2017 at 12:28 pm #

REPLY

We are writing the timeseries_to_supervised function to accept a lag:

```
def timeseries_to_supervised(data, lag=1):
```

Is this feature to essentially the Window Method + reshaping described in your July 21, 2016 tutorial?

But we don't seem to use that parameter, always "timeseries_to_supervised(diff_values, 1)"

Using lag values up to around 50 (on the airline data) and changing predict() like so:

```
# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0:look_back].reshape(len(train_scaled), 1, look_back)
# train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
lstm_model.predict(train_reshaped, batch_size=1)
```

seems to make good predictions with very low RMSE on test.

Am I possibly breaking the model now so it's cheating somehow?

Kim Miller June 16, 2017 at 5:27 am #

look_back = 60
epochs = 15

```
Month=1, Predicted=321.813110, Expected=315.000000
Month=2, Predicted=310.719757, Expected=301.000000
Month=3, Predicted=363.746643, Expected=356.000000
...
Month=46, Predicted=458.424683, Expected=461.000000
Month=47, Predicted=418.427124, Expected=390.000000
Month=48, Predicted=412.831085, Expected=432.000000
Test RMSE: 18.078
```

for test harness

```
yhat = y
Test RMSE: 0.000
```

It seems there is contamination of training data in the test forecast in the sense that forecast_lstm(model, batch_size, X) is being given X observations, each with 60 historical past observations that overlap into training data. But so what, right? That's history that training saw, and as it moves into the test data it adds history only seen in the test set. But that's how real life observations work it seems: You always have all of the past to look at.

Finally, you say, "In this tutorial, we will go with the fixed approach for its simplicity, although, we would expect the dynamic approach to result in better model skill." For a small dataset like we have, a large lag seems as if it gives us a semi-dynamic approach in some ways?

Solid model with a seemingly good RMSE?
Full code: http://tinyurl.com/y74ypvde
Prediction v. Actual Plot: http://tinyurl.com/y9ouajdm

Kim Miller June 16, 2017 at 7:26 am #

On the Shampoo dataset:
look_back = 18 (lag)
epochs = 30
Test RMSE: 110.594
Prediction v. Actual Plot: http://tinyurl.com/ycoy6aes

Jason Brownlee June 16, 2017 at 8:12 am #

Very sharp!

Jason Brownlee June 16, 2017 at 8:10 am #

REPLY

Nice.

Hmmm, I believe by dynamic I was referring to updating the model with new data.

If you have the resources, I would recommend exploring whether updating the model as new ob
example here:

http://machinelearningmastery.com/update-lstm-networks-training-time-series-forecasting/

Pedro June 15, 2017 at 5:22 pm #

REPLY

Hi Jason,

You're doing a great job helping people to learn how to apply machine learning. One week ago, I didn't know anything about time series, but now I'm able to play a little bit with it. Thanks you!

Regarding this post, I've a doubt. When we say

'To make the experiment fair, the scaling coefficients (min and max) values must be calculated on the training dataset and applied to scale the test dataset and any forecasts. This is to avoid contaminating the experiment with knowledge from the test dataset, v

shouldn't we do

scaler = scaler.fit(X[:-12])

instead of

scaler = scaler.fit(X) ?

If we use X we are using data from the test set, no?

Thanks for your attention and keep up with the good work!

Jason Brownlee June 16, 2017 at 7:51 am #

Correct. Ideally, we would use domain knowledge to estimate the expected min/max of values that could ever be observed.

Eric June 20, 2017 at 6:57 pm #

REPLY

Hello Jason,

After using the robust code (last one) I got theses results

1) Test RMSE: 180.438
2) Test RMSE: 110.352
3) Test RMSE: 119.655
4) Test RMSE: 170.720
5) Test RMSE: 211.877
6) Test RMSE: 101.453
7) Test RMSE: 105.532
8) Test RMSE: 149.351
9) Test RMSE: 88.118
10) Test RMSE: 138.013
11) Test RMSE: 265.045
12) Test RMSE: 135.861
13) Test RMSE: 167.766 … (rest is omitted, it was taking too long).
Being a beginner in machine learning and using your tutorial to learn about it could you tell me if theses
have) while everything before was ok. If theses are not, could you tell my what could be the reason of su

Thank you for your attention and please do continue making tutorial like that, it's really helpful!

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

Hi Eric, see this post to better understand the stochastic nature of neural network algorithm
http://machinelearningmastery.com/randomness-in-machine-learning/

See this post for more details on how to develop a robust estimate of model skill:
http://machinelearningmastery.com/evaluate-skill-deep-learning-models/

See this post on how to fix the random number seed:
http://machinelearningmastery.com/reproducible-results-neural-networks-keras/

Eric June 21, 2017 at 1:02 pm #

Get Your Start in Machine Learning

I see, thank you very much for the links and solutions!

Jason Brownlee June 22, 2017 at 6:03 am #

REPLY

You're welcome.

Eric June 23, 2017 at 5:50 pm #

After reading the links what I understood was:
– It's pretty hard (impossible) to always get the same result due to a lot of randomness t
– Only by repeating the operation a lot (like you did on last example, with 30 example (b
help you to determine how much result can be different and find a range of "acceptable
– Seeding the random number generator can help.

So to continue my project I thought about seed the random number generator like you e
I have a server I can let it do it overnight and go for 100) and determine the average and

Does it seem ok to begin with?
Am I lacking something?

Thank you for your attention and for the help you provide to everyone.

Jason Brownlee June 24, 2017 at 8:00 am #

See this post on how to estimate the number of repeats:
http://machinelearningmastery.com/estimate-number-experiment-repeats-stochastic-machine-learning-algorithms/

I would not recommend fixing the random number seed for experimental work. Your results will misleading.

Get Your Start in Machine Learning

Dhineshkumar June 30, 2017 at 7:18 pm #

REPLY

Hi Jason,

Thank you so much for the tutorial.

I have few doubts though.

1. I am working on a problem where the autocorrelation plot of the detrended data show me that the value at time t is significantly correlated to around past 100 values in the series. Is it ideal to take the batch size of 100 to model the series?

2. You mentioned that less than 5 memory units is sufficient for this example. Can you please give me some idea on how to choose the number of memory units for a particular problem like the above? On what other factors does this number depend on?

Kindly clarify.
Thanks

Jason Brownlee July 1, 2017 at 6:33 am #

Try passing in 100-200 time steps and see how you go.

Systematically test a suite of different memory units to see what works best.

Dhineshkumar July 1, 2017 at 2:42 pm #

Thank you Jason.

Jason Brownlee July 2, 2017 at 6:26 am #

REPLY

You're welcome.

Sophie July 4, 2017 at 3:04 pm #

REPLY

Hey Jason! Thanks a lot for this tutorial, it really helped.

I used this tutorial as is for predicting the cost of an item which is of the the range of a dollar and few cents.

My dataset has 262 rows, i.e 0 to 261.

When I run the model, the graph captures even the most intricate trends beautifully, BUT there seems to be a lag of 1 time step in the predicted data.

The predicted values of this month almost exactly matches the expected value of the previous month. And this trend is followed throughout.

The indexing is the only thing I've changed,
to

```
train, test = supervised_values[0:200], supervised_values[200:]
rmse = sqrt(mean_squared_error(raw_values[200:], predictions))
pyplot.plot(raw_values[200:])
```

are the only lines of code I've really changed

Jason Brownlee July 6, 2017 at 10:11 am #

Consider starting with a simpler linear model:
http://machinelearningmastery.com/start-here/#timeseries

I do not believe LSTMs are the best or first choice for autoregression models:
http://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecasting/

Viorel Emilian Teodorescu July 9, 2017 at 2:09 am #

REPLY

Hi, Jason

Doesn't LSTM have a squashing gate at input? With outputs in between (-1, 1)? Then why do we need to prepare the input data to be between (-1, 1) if the first input gate will do this for us?

Am I missing something?

REPLY

Jason Brownlee July 9, 2017 at 10:56 am #

Yes, you can rescale input to [0,1]

Hans July 9, 2017 at 2:58 am #

Considering

unseenPredict = lstm_model.predict(X)

…how do we structure X to get a one step forward prediction of unseen data?

Or can we change some offsets in script "Complete LSTM Example" to get the same effect, and if so hov

Paba July 10, 2017 at 8:45 pm #

Hi Jason,
Thanks for the excellent tutorial. I tried to modify the above code to include multiple timesteps and multiple lags in the model. I can give these parameters as input and paralleling run the script for different configurations to select the most accurate model. What do you think of the modifications I have done to the following functions? I am especially concerned about the time_steps included in the model, is that correct?

```
def timeseries_to_supervised(data, lag=1, time_steps=0):
df = DataFrame(data)
columns = [df.shift(i) for i in range(1, lag+1)]
#considers the number of time_steps, t, involved and
#add next t x columns next to each x columnn
```

```
#question?? if time_steps = 3, does that mean y should start from y_4 and
#we trim the last 3 values from the dataset?
if time_steps > 0 :
columns_df = concat(columns,axis=1)
#note that I have multipled i by -1 to perform left shift rather than right shift
timestep_columns = [columns_df.shift(i*-1) for i in range(1, time_steps+1)]
timestep_columns_df =concat(timestep_columns, axis=1)
columns.append(timestep_columns_df)
columns.append(df)
df = concat(columns, axis=1)
df.fillna(0, inplace=True)
return df

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

def forecast_lstm(model, batch_size, X, lag,time_steps):

X = X.reshape(1,time_steps+1,lag)
pad = np.zeros(shape=(batch_size-1, time_steps+1, lag))
padded = np.vstack((X, pad))
yhat = model.predict(padded, batch_size=batch_size)
return yhat[0,0]
```

Jason Brownlee July 11, 2017 at 10:30 am #

REPLY

Great work!

Kim Miller July 13, 2017 at 9:01 am #

REPLY

I want to extend this idea to several features x the lag values for each x time observations. Does it seem reasonable to give MinMaxScaler that 3D object? How does the y truth fit in what I give MinMaxScaler, since it's only 2D?

Jason Brownlee July 13, 2017 at 10:04 am #

No, I would recommend scaling each series separately.

Kim Miller July 13, 2017 at 11:51 am #

Above you seem to scale y along with X. But with multiple features, the rest of which ar
fit y_train by itself before transforming y_train and y_val? Then that's actually the only scaler obj

Jason Brownlee July 13, 2017 at 4:58 pm #

REPLY

I would recommend scaling each series before any shifting of the series to make it a supervised learning problem.

I hope that answers your question, please shout if I misunderstood.

Kim Miller July 14, 2017 at 2:17 am #

I think the words “each series separately” threw me. I assume we can still scale all incoming values (all X’s and the time series that will later become y and shifted, one of the X’s) using a single scaler object. Then we create the lag values from scaled values. Finally, that single scaler object is used to invert y predictions. I have that right?

Jason Brownlee July 14, 2017 at 8:33 am #

Each series being a different “feature” or “column” or “time series” or whatever we call it.

The coefficients (min/max or mean/stdev) used in the scaling of each series will need to
apply the operation to input data later. You can save the coefficients or the objects that v

Sasha July 13, 2017 at 4:15 pm #

Jason, great tutorial, thank you!

A question: why do you loop through epochs instead of just setting an appropriate number of epochs wit
be neater?

Jason Brownlee July 13, 2017 at 5:02 pm #

So that I can manually manage the resetting of the internal state of the network.

Sasha July 13, 2017 at 5:22 pm #

REPLY

Ah, I see, it has to do with “stateful” definition being different in keras, it’s well explained

kotb July 13, 2017 at 6:59 pm #

REPLY

Hi, DR jason

thank you very much for this tutorial.

I want to have multiple time steps , but i don't know how to modify function "timeseries_to_supervised()".
I found another post of you talking about this , but you use function "create_dataset()"
i modify this function as follow:

```
def create_dataset(dataset, look_back=1):
dataset = np.insert(dataset,[0]*look_back,0)
dataX, dataY = [], []
for i in range(len(dataset)-look_back):
a = dataset[i:(i+look_back)]
dataX.append(a)
dataY.append(dataset[i + look_back])
dataY=numpy.array(dataY)
dataY = np.reshape(dataY,(dataY.shape[0],1))
dataset = np.concatenate((dataX,dataY),axis=1)
return dataset
```

please, check my modification , is it right OR what?

Jason Brownlee July 14, 2017 at 8:25 am #

REPLY

See this post:
http://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

Eric July 20, 2017 at 2:22 pm #

Get Your Start in Machine Learning

Hi Jason,

Sorry if it seem stupid but there is a part that I don't understand.

To predict you use: "yhat = model.predict(X, batch_size=batch_size)"
But as we see X is :
train, test = supervised_values[0:-12], supervised_values[-12:]
scaler, train_scaled, test_scaled = scale(train, test)
yhat = forecast_lstm(lstm_model, 1, X)

So X is the 12 value (after being passed in the scale function) that we want to predict. Why are we using them since in normal case we wouldn't know theirs values.

Once again really thank you for your tutorial, really helped me in my training on machine learning.

Jason Brownlee July 21, 2017 at 9:29 am #

In the general case, you can pass in whatever you like in order to make a prediction.

For example, if your model was defined by taking 6 days of values and predicting the next day and y
today and 5 days prior.

Does that help?

Eric July 21, 2017 at 10:48 am #

I think it does help me. In my case I have values for each minute and I have to predict the next week (so more or less 10K of prediction).
I have data from the last year so there isn't any problem with my training, just wondered what I should do at the prediction part (so I can just instead of feeding it test_scaled send him my training set again?)
Thank you for your help and quick reply!

Jason Brownlee July 22, 2017 at 8:29 am #

REPLY

Yes, if your model is setup to predict given some fixed set of lags, you must provide those lags to predict beyond the end of the dataset. These may be part of your training dataset.

Eric July 24, 2017 at 5:55 pm #

I don't think it's in my training dataset, for this part I'm pretty much doing it like you (the lag appearing when turning a sequence to a supervised learning problem). I'm pretty much feeding my model like you do. The problem for me being to know what to feed it when calling the predict command. Sending it "train_scaled" was a bad idea (got poor result, predicting huge values when it should predict low and predicting low instead of predicting high). I'm working on it but every hint is accepted. Or learning/understanding.

Jason Brownlee July 25, 2017 at 9:36 am #

The argument to the predict() function is the same as the argument to the fit() fu

The data must be scaled the same and have the same shape, although the number of s

Obviously, you don't need to make predictions for the training data, so the data passed t prediction you require. This really depends on the framing of your problem/model.

Does that help?

Eric July 25, 2017 at 12:09 pm #

Thank your for your quick reply! I think I'm understanding it better but I have some part I have trouble to understand.

The input is X right?
If I follow your tutorial on input/output (http://machinelearningmastery.com/time-series-fo

an example (the database register the value each 3 minute and we want to predict the next value (so to begin for example it would be 12:24:00)):
Date,congestion
2016-07-08 12:12:00,92
2016-07-08 12:15:00,80
2016-07-08 12:18:00,92
2016-07-08 12:21:00,86

This is (part of) my training data, when turning it into supervised training data (and shifting) I get:
X, y
?, 92
92, 80
80, 92
92, 86
86, ?

The problem is that I don't know X for predicting, I only know the X I use for my training
results (test_scaled).

What is the input I should feed it? I can't feed it my test_scaled since in real situation I w

Sorry if my question seem stupid and thank you for taking time to explain it.

Jason Brownlee July 26, 2017 at 7:45 am #

It depends how you have framed your model.

If the input (X) is the observation at t-1 to predict t, then you input the last observation to predict the next time step.

Eric July 26, 2017 at 11:49 am #

It is, each of my input X is the observation at t-1 (pretty similar to the shampoo

Thank you for your answer, you answered my question, I shouldn't have any problem now!

Thanks for the tutorial too, they really helped me!

Jason Brownlee July 26, 2017 at 4:01 pm #

Glad to hear that.

Eric July 26, 2017 at 6:49 pm #

Just to be sure that I didn't made any mistake in my reasoning, if I take my exa

X, y
?, 92 / T-3
92, 80 / T-2
80, 92 / T-1
92, 86 / T
86, ? / T+1

to predict the next step (T+1) I have to use "yhat = model.predict(X, , batch_size=batch_
Then I'll get the value predicted for T+1 (that I have to send to invert_scale and differenc
If I want to predict farther then I continue (sending the scaled/reshaped value predicted a
wanted).

Thanks for your time and answers!

Jason Brownlee July 27, 2017 at 7:58 am #

Correct.

Get Your Start in Machine Learning

Eric August 4, 2017 at 12:05 pm #

Thank you Jason with your help I managed to predict value of each minute of the next week. I had 2 question though:
First: I removed the code for testing sets (since I wouldn't have it in reality), the only thing I have are the testing set in the excel file (the last 10000 lines)
When using this code (to train):

```
# transform data to be stationary
raw_values = data.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 = supervised_values[:-10000]

# transform the scale of the data
scaler, train_scaled = scale(train)
```

and this one (to predict):

```
# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
lstm_model.predict(train_reshaped)

# walk-forward validation on the test data
predictions = list()
predictionFeeder = list() #Used to feed the model with the value of T-1
X, y = train_scaled[0, 0:-1], train_scaled[0, -1]
predictionFeeder.append(X) #Give the last value of training

for i in range(0, 10000):
# make one-step forecast
yhat = forecast_lstm2(lstm_model, predictionFeeder[i])
predictionFeeder.append(yhat)
# invert scaling
yhat2 = invert_scale(scaler, testa[i + 1], yhat)
```


Get Your Start in Machine Learning

yhat3 = inverse_difference(raw_values, yhat2, 10000 + 1 – i)
predictions.append(yhat3)
and train a model (25 epoch) then predict the results I get result that are way too good (RMSE of 2 or less and prediction that have less than 5% of error).
Being used of thing going wrong for no reasons I decide to remove the testing data from the excel (even though it shouldn't change anything since I'm not using them (I've even set the variable to None at first)). Then when I do that the prediction is way less good and have some lag (though, if you remove the lag you still have a good results, just way less better than before).
Why is that?
My 2nd question is about lag, we can see on the prediction that while the shape of both chart (prediction and reality) look alike the prediction is raising/lowering before the reality, do you have any idea to fix it? Do you think changing the lag or timestep would help?

Once again thank you for your help, I don't think I would have achieved this much without your tutorials.

Jason Brownlee August 4, 2017 at 3:43 pm #

Sorry, I cannot debug your code for you.

Perhaps you are accidentally fitting the model on the training and test data then evaluati

I would encourage you to evaluate different lag values to see what works best for your p

Eric August 4, 2017 at 4:02 pm #

Don't worry, I wouldn't ask you to debug.

Maybe, I don't know, I did remove the variable to be sure to never have affected the testing set and using it but since I'm human I may have made errors.

So changing lag would help me for theses gap between reality and prediction raise. Thank you I'll do that.

Thanks for your answer!

Josep July 21, 2017 at 8:08 pm #

REPLY

Can I buy your books physically(not Ebook)?Thanks

Josep July 21, 2017 at 8:15 pm #

REPLY

Sorry, now I have read that is not possible. Thanks anyway. Your explanations and tutorials are amazing. Congratulations!

Jason Brownlee July 22, 2017 at 8:33 am #

REPLY

Thanks Josep.

Pawel July 26, 2017 at 7:59 pm #

Hi,
Thanks for very good tutorial. I have one question/doubt.

in the following part of the code:

# invert differencing
yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)

should not we relay on predicted value instead of already known raw_values? In your example for validation we always refer to the test value(known) while calling inverse difference. But in reality we will have only the predicted values(used as X), and of course known starting point(t=0). Or I missed something?

my proposal:
invert differencing – (starting from 2nd loop cycle (1st woul be the starting point (raw_values[-1]))

yhat = inverse_difference(predictions, yhat, 1)

Thanks in advance
Pawel

Jason Brownlee July 27, 2017 at 8:00 am #

REPLY

We could, but in this case the known observations are available and not in the future and it is reasonable to use them for the rescaling (inverting).

Pawel July 27, 2017 at 5:03 pm #

REPLY

Hi, Thanks for explanation, got it now. Cause I train the model used e.g. MAY data (15 seconds samples) and then used that model to predict whole JUNE data. Afterwards I compared predicted data VS data that I got from JUNE, and I have to say that model does not work, after few prediction there is huge “off sync”,

In the validation phase as described in your case I got RMSE 0.11 so not bad, but in reality whe there is a problem.

Do you know how to improve the model? should I use multiple step forecats, or lag features, inp

Thanks a lot.
Pawel

Jason Brownlee July 28, 2017 at 8:29 am #

I would recommend braining storming, then try everything you can think of.

I have a good list here:
http://machinelearningmastery.com/improve-deep-learning-performance/

Also, compare results to well-tuned MLPs with a window.

Surya July 27, 2017 at 6:49 pm #

Hey Jason, I am not following one point in your post. You wanted to train a state-full LSTM but reset_states() is executed after every epoch. That means, the states from previous batch are not used in the current batch. How does it make the network state-full?

Thanks

REPLY

Jason Brownlee July 28, 2017 at 8:30 am #

State is maintained between the samples within one epoch.

Niklas August 2, 2017 at 12:03 am #

Hi Jason,

thanks for the great tutorial. I have one question. Wouldn't it be better to use a callback for resetting the example an EarlyStopping Monitor while training, here is what I changed:
class resetStates(Callback):
def on_epoch_end(self, epoch, logs=None):
self.model.reset_states()

model.fit(X, y, epochs=nb_epoch, batch_size=batch_size, verbose=1, shuffle=False, callbacks=[resetSta
verbose=1, mode='min')])

REPLY

Jason Brownlee August 2, 2017 at 7:54 am #

Yes, that is a cleaner implementation for those problems that need to reset state at the end of each epoch.

REPLY

Bharath August 6, 2017 at 2:38 am #

Get Your Start in Machine Learning

Hello, can we extend this for anomaly detection techniques ?

Jason Brownlee August 6, 2017 at 7:40 am #

REPLY

Perhaps, I have not used LSTMs for anomaly detection, I cannot give you good advice.

Perhaps you would frame it as sequence classification?

Daniel Ruiz August 7, 2017 at 1:36 pm #

REPLY

Hi Jason,

In the persistence model plot there is a one time interval lag. When making single step predictions, is it p
to happen? It seems like the model places a huge weight on time interval x[t-1].
Here is an example of the dataset I am analyzing:

iteration: 969
Month=970, Predicted=-7.344685, Expected=280.000000
iteration: 970
Month=971, Predicted=73.259611, Expected=212.000000
iteration: 971
Month=972, Predicted=137.053028, Expected=0.000000

Expected should be 280 and 212 (high magnitudes), and the model captures this more or less with 73 a

Thanks!

Jason Brownlee August 8, 2017 at 7:41 am #

REPLY

LSTMs are not great at autoregression problems and often converge to a persistence type model.

Daniel Ruiz August 8, 2017 at 8:57 am #

REPLY

Ok thanks. What model would be a good alternative to capture this issue? I ran into the same problem with ARIMA. It could just be a difficult dataset to predict.

Jason Brownlee August 8, 2017 at 5:07 pm #

REPLY

I recommend starting with a well-tuned MLP + Window and see if anything can do better.

Eric August 9, 2017 at 11:02 am #

Hi Jason,

Thanks to you I managed to get a working LSTM network who seem to have a good accuracy (and so a

But I've got a problem, do you know what could be the cause of extreme delay between reality values ar as the reality but increase/decrease way before reality)?

Best regards and please continue what you are doing, it's really useful.

Jason Brownlee August 10, 2017 at 6:40 am #

Hi Eric, it might be a bug in the way you are evaluating the model.

Eric August 10, 2017 at 10:50 am #

REPLY

It might be the case, I had to make some changes to the prediction system to use the la this moment. (Also I have to use Tflearn instead of Tensorflow but it shouldn't be a problem sinc

tensorflow).

Thank you for your answer!

Jason Brownlee August 10, 2017 at 4:40 pm #

REPLY

Hang in there Eric!

Eric August 10, 2017 at 5:10 pm #

Thank you!

Well.. I have a gap of 151 (reference to pokemon?).

Just to try I removed theses 151 values from my training set, I now have no gap of value
epoch training). I know that this is no way a fix but make me wonder where did I fail..

Eric August 10, 2017 at 7:21 pm #

Could it be that while my training set is on 400K of value my prediction start 15
399849) of the training set (which is strange since the information from training tell me t
It would mean that my machine is trying to predict some point of time used for training.
Or it would mean that the 151 last data weren't used at all for training (I tried to reduce the number of data but it's the same problem).

Jason Brownlee August 11, 2017 at 6:41 am #

The algorithm is trained sample by sample, batch by batch, epoch by epoch. The last sample is what is key.

Eric August 11, 2017 at 12:24 pm #

Thanks for the reply.
When I think about it my prediction is in advance compared to the reality.
So my model start his prediction not where I finished my training but after (which is strange since my training end where my testing begin and that each input are a value (and each line correspond to a minute like the the line of shampoo data sets correspond to month)).
I must have made a mistake somewhere.
Thanks for you answer I think I'm on something!

Jason Brownlee August 12, 2017 at 6:45 am #

Hang in there!

- srn -

Stone August 9, 2017 at 12:16 pm #

One thing I don't understand in these models is, how can I predict future.

Say, I have trained my model and let's say we're living August 8, 2017. Now model.predict() seems to re
case values from test_scaled. So what do I give it when I want to get a prediction for August 9, 2017, wh
Can I ask the model to give me forecasts for the next 5 days?

Eric August 9, 2017 at 2:11 pm #

REPLY

I may be wrong but in my case I trained my model on the data I had (let's say one year, until August 8, 2017) and to predict the value T you send the value T-1 (so in this case the value of August 8 2017), you'll get the value of August 9.
Then if you want August 10 (T+1) you send the value of August 9 (T). But this code here show a one step forecast implementation. Maybe if you want to predict more you should look for multi step forecasting? I'm think there is some example of it on this website.

Jason would give you a better answer I think.

Jason Brownlee August 10, 2017 at 6:47 am #

REPLY

Nice one Eric!

Yes, I'll add, if the model was trained to predict one day from the prior day as input, then when finalized, the model will need the observation from the prior day in order to predict the next day.

It all comes down to how you frame the prediction problem that determines what the inputs you need.

Jason Brownlee August 10, 2017 at 6:43 am #

If you would like to predict one day, the model needs to be trained to predict one day.

You have to decide what the model will take as input in order to make that prediction, e.g. the last m

Then, when it's trained, you can use it to predict future values using perhaps the last few values fron predict one day to predict may days in a recursive manner where predictions become inputs for subs http://machinelearningmastery.com/multi-step-time-series-forecasting/

You can also make multi-step forecasts directly, here is an example: http://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-netw

Does that make things clearer?

Stone August 10, 2017 at 11:08 am #

REPLY

Thanks Eric and Jason. That's exactly how I've done it, but it seems to me, that the prediction that I get is not one timestep forward (T+1), but it's a prediction of T, which doesn't make sense. I'm using close price of a stock as my data. I'll have to check again, if it's really is making a prediction for the future, as you insist. and I will check the Jason's links.

Anyways, thanks Jason for the excellent tutorials! 🙂

Stone August 10, 2017 at 12:22 pm #

REPLY

Here's a sample of the close prices:
2017-05-30 5.660
2017-05-31 5.645
2017-06-01 5.795
2017-06-02 5.830

As a matter of fact, it seems to me that the predicted values lags one time step behind from expected ones.

Day=1, Predicted=5.705567, Expected=5.660000
Day=1, Predicted=5.671651, Expected=5.645000
Day=1, Predicted=5.657278, Expected=5.795000
Day=1, Predicted=5.805318, Expected=5.830000

Here I'm going forwarding one day at a time and as you see the predicted ones are closer to what w

Let's examine the second line:
Day=1, Predicted=5.671651, Expected=5.645000

The expected price (which is the actual price on 2017-05-31) is what I give the model on 2017-05-31 model to predict is closer to 5.79 than 5.67 (which is closer to the actual price the day before!). See changed anything in the framework except the data.

Jason Brownlee August 10, 2017 at 4:41 pm #

That is a persistence model and the algorithm will converge to that (e.g. predict the input as the output) if it cannot do better.

Perhaps explore tuning the algorithm.

Also, I believe security prices are a random walk and persistence is the best we can do anyway:
http://machinelearningmastery.com/gentle-introduction-random-walk-times-series-forecasting-python/

Firnas August 12, 2017 at 1:27 pm #

REPLY

Please do some tutorials RNN, NN with more categorical values in the dataset? please, I could not find many resources using the categorical values.

Thanks

Jason Brownlee August 13, 2017 at 9:45 am #

REPLY

I have a few tutorials on the blog – for example, text data input or output are categorical variables.

What are you looking for exactly?

nandini August 16, 2017 at 5:23 pm #

is it possible to write the code RNN Regression ,what is the activation function i need to give fo
classification to regression?

Jason Brownlee August 17, 2017 at 6:36 am #

Yes. The output would be ‘linear’ for regression.

Maria Jimenez August 17, 2017 at 1:20 am #

REPLY

Hi Dr. Brownlee,

I am trying to understand this code. I am a beginner working with time-series and LSTM.

My questions about your code:

1. What does “transform data to be stationary” mean?
2. Why do you create diff? What does it mean?
3. If the raw_values are 36 data, why diff has only 35?

I appreciate your response in advance,

Maria

REPLY

Maria Jimenez August 17, 2017 at 1:43 am #

Never mind, I already understand. 🙂

Jason Brownlee August 17, 2017 at 6:46 am #

Glad to hear it.

Jason Brownlee August 17, 2017 at 6:46 am #

Consider using the blog search.

More about stationary data here:
http://machinelearningmastery.com/time-series-data-stationary-python/

More about differencing here:
http://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

I hope that helps.

Sourabh August 18, 2017 at 4:41 am #

Get Your Start in Machine Learning

Thanks, for the awesome blogs,
But, I have a doubt, for time series forecasting, classical methods like autoregression and ARIMA, or this machine learning approach using LSTM RNN model, which is the better approach? Both are good on their own, So, what should be our first choice while forecasting any dataset? How should we choose between them?

REPLY

Jason Brownlee August 18, 2017 at 6:27 am #

It depends on your problem. Try a few methods and see what works best.

I recommend starting with linear methods like ARIMA, then try some ML methods like trees and such, then an MLP, then perhaps an RNN.

Bilal August 19, 2017 at 10:29 pm #

When I changed train/test ration to 0.8/0.2 in LSTM code, it took half an hour to predict, and wit
taking hours(i still couldn't get result, i don't know how long it'll take). Can you please suggest me how I

Jason Brownlee August 20, 2017 at 6:06 am #

I have some ideas here:
http://machinelearningmastery.com/handle-long-sequences-long-short-term-memory-recurrent-neura

REPLY

Arun August 22, 2017 at 1:45 am #

Hi Jason, for long sequence like 4000 records, can you please help me to understand the changes from code point of view as per your current example, the above link is for classification and i am looking for sequence prediction problem.

Jason Brownlee August 22, 2017 at 6:46 am #

REPLY

You must split your sequence into subsequences, for example, see this post for ideas:
https://machinelearningmastery.com/truncated-backpropagation-through-time-in-keras/

Arun Menon August 23, 2017 at 12:25 am #

Thank you Jason, can you please help us to understand with a complete code example ? The above link just have an idea but not a code implementation, I want to understand from code point of view.

Regards,
Arun

hirohi August 21, 2017 at 12:54 pm #

Is the evaluation method in this post really “walk-forward”? I’ve read your post about time-series validation for not time series data. In both, we make many models to learn, so these methods are not su this post.

Jason Brownlee August 21, 2017 at 4:26 pm #

Yes, we are using walk-forward validation here.

Walk-forward validation is required for sequence prediction like time series forecasting.

See this post for more about the method and when to use it:
https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

hirohi August 22, 2017 at 11:16 am #

REPLY

Thank you for your reply. Sorry, that's not what I mean it. I've read the post you pasted above, in which you split data into train/test 2820 times and build 2820 models. But, in this LSTM post, you split data and build a LSTM network only once, so I suggest the test in this article is not walk forward. ordered(not shuffle) hold-out test?

hirohi August 22, 2017 at 11:17 am #

REPLY

sorry, please delete this comment

Jason Brownlee August 23, 2017 at 6:36 am #

Above, the model is evaluated 30 times.

Within the evaluation of one model, we use walk forward validation and samples are not shuffled.

I recommend re-reading the tutorial and code above.

hirohi August 23, 2017 at 12:26 pm #

Thank you for your reply! Oh, repeat=30! Sorry, I misunderstood. But I think in the back this LSTM article, the size of train data is fixed, right? Sorry for so many questions.

Gauthier August 23, 2017 at 10:38 pm #

REPLY

Thanks for your work, it is seriously awesome.

I wanted to see the chart comparing training loss vs validation loss per epoch, see the chart here: https:/

“From the plot of loss, we can see that the model has comparable performance on both train and validation datasets (labeled test). If these parallel plots start to depart consistently, it might be a sign to stop training at an earlier epoch.”

However, in this case, the test loss never really gets lower and both lines are not parallel at all. for this problem, and in general, does that mean that no meaningful test-set pattern is learned from the training, as seems to be corroborated by the RMSE being close on average to the persistent/dummy method?

Is this the end of the road? Have NN’s failed in their tasks? If not, where to go next?

The main thing I changed in the code was the end of the fit_lstm function:
for i in range(nb_epoch):
e=model.fit(X, y, epochs=10, batch_size=batch_size, verbose=0,validation_split=0.33, shuffle=False)
model.reset_states()
return model, e

Jason Brownlee August 24, 2017 at 6:42 am #

Thanks Gauthier.

It may mean that the model under provisioned for the problem.

Gauthier August 24, 2017 at 6:37 pm #

OK… So in this case would you suggest to include more neurons for example, or try an

Jason Brownlee August 25, 2017 at 6:41 am #

REPLY

Both would be a safe bet.

Muni August 28, 2017 at 12:19 pm #

Get Your Start in Machine Learning

Json,

Thanks for your blogs and tutorials. Being a research scholar, when I was in dilemma how to learn ML, your tutorials gave me an excellent start to carry out my work in Machine Learning and Deep Learning.

I tried to work with the example in this page, But, I couldnt download the shampoo_sales dataset from datamarket.com. Could you please put the link to the datasets in the blog? If any procedure to download, please let me know.

Jason Brownlee August 29, 2017 at 5:01 pm #

REPLY

Thanks, I'm glad to hear that.

Here is the full dataset:

```
"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
```

```
29 "3-04",439.3
30 "3-05",401.3
31 "3-06",437.4
32 "3-07",575.5
33 "3-08",407.6
34 "3-09",682.0
35 "3-10",475.3
36 "3-11",581.3
37 "3-12",646.9
```

mikiwate September 3, 2017 at 3:35 pm #

REPLY

Hi Jason,

I am getting NAN on the 12 month:::

Month=1, Predicted=440.400000, Expected=440.400000
Month=2, Predicted=315.900000, Expected=315.900000
Month=3, Predicted=439.300000, Expected=439.300000
Month=4, Predicted=401.300000, Expected=401.300000
Month=5, Predicted=437.400000, Expected=437.400000
Month=6, Predicted=575.500000, Expected=575.500000
Month=7, Predicted=407.600000, Expected=407.600000
Month=8, Predicted=682.000000, Expected=682.000000
Month=9, Predicted=475.300000, Expected=475.300000
Month=10, Predicted=581.300000, Expected=581.300000
Month=11, Predicted=646.900000, Expected=646.900000
Month=12, Predicted=646.900000, Expected=nan

from the code below:

```
predictions = list()
for i in range(len(test_scaled)):
# make one-step forecast
X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
#yhat = forecast_lstm(lstm_model, 1, X)
```

```
yhat=y
# 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)
expected = raw_values[len(train) + i + 1]
print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))
```

I do not understand as to why this is happening.

thanks for the feedback.

Jason Brownlee September 3, 2017 at 3:49 pm #

Interesting.

Does this happen every time?

I wonder if it is platform specific. Are you on a 32-bit machine? What version of Python?

Eric September 3, 2017 at 6:30 pm #

Dear Jason,

I had a question, what are we supposed to send as first parameter of inverse_difference if we go with the idea that we don't know the future?

Regardless what I do, if I give it the testing set then it have perfect accuracy and values. If I send the training set it will look like the last len(test_scaled) training values.

Best regards

Jason Brownlee September 4, 2017 at 4:26 am #

REPLY

You would pass a list of past observations so that you can reverse the difference operation.

Eric September 4, 2017 at 9:42 am #

REPLY

Dear Mr Jason,

Sorry, I have trouble understanding, what do you mean by “list of past observations”? Would that be what we predict (the predictions list)?
It would then follow what “Pawel” asked and said previously?
“my proposal:
invert differencing – (starting from 2nd loop cycle (1st would be the starting point (raw_values[-

yhat = inverse_difference(predictions, yhat, 1)”

Best regards,
Eric

Jason Brownlee September 7, 2017 at 12:32 pm #

Sorry, I mean it is the true or real values observed in the domain at prior time steps

Eric September 7, 2017 at 5:04 pm #

Dear Mr Jason,

Thank you, so let’s imagine a 500 000 entry file.
My training would be the 490 000 first line. I want to predict the last 10 000 lines.

For predicting I send the last line of the training set (then for the next prediction I send m
the difference).

To get the true value I send as argument of “Invert_difference” my training set, or more specially, my last 10 000 thousands line?

Best regards,
Eric Godard.

Jason Brownlee September 9, 2017 at 11:40 am #

I would not recommend prediction 10K time steps. I expect skill would be very poor.

Nevertheless, observation 490000 would be used to invert observation 490001, then the 490001 transformed prediction would be used to decode 490002 and so on.

Does that help?

Eric September 8, 2017 at 10:09 am #

Sorry, I may have badly worded myself here.

My dataset have one entry per minute, one day is 1440 entry, I have years worth of data
I send the last day of the training set (trained_scaled[-1440:]) as argument of inverse_di

Best regards,

Jason Brownlee September 9, 2017 at 11:49 am #

Perhaps this post will help you better understand the different transform:
https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

Brian September 16, 2017 at 8:55 pm #

Get Your Start in Machine Learning

I keep getting inconsistent result of RMSE every time I rerun the code, so I added

np.random.seed(1) # for consistent results

to the very top of my code so now every run produces consistent RMSE.

REPLY

Jason Brownlee September 17, 2017 at 5:27 am #

It is hard to get reproducible results, see this post for more information Brian:
https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

Irene September 18, 2017 at 8:48 pm #

When I try to load the data after executing the following code line with Python 3.6 I get the follo

series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)

TypeError: ufunc 'add' did not contain a loop with signature matching types dtype('<U32') dtype('<U32') d

Can you help me, please?

Jason Brownlee September 19, 2017 at 7:37 am #

Ouch, I have not seen that error before.

Perhaps confirm that the data is CSV and that you have removed the footer.

Perhaps ensure that your environment was installed correctly:
http://machinelearningmastery.com/setup-python-environment-machine-learning-deep-learning-anaconda/

If that does not help, perhaps try posting to stackoverflow?

Priyank September 18, 2017 at 10:05 pm #

REPLY

Hello Jason,

Thanks for sharing this post, very helpful as currently I am working on a time series project. I have one query.. How can we save a model with best RSME and use it to further prediction.

Thanks

Jason Brownlee September 19, 2017 at 7:44 am #

REPLY

This post shows you how to save your model and use it to make predictions:
https://machinelearningmastery.com/save-load-keras-deep-learning-models/

Alex September 21, 2017 at 10:27 am #

Thanks Jason for the great tutorial!

Can you please clarify this statement for me: “The batch_size must be set to 1. This is because it must b

I don’t understand how 1 in this case is a factor of the training and test datasets?

Further, for time series data, can we have a batch size greater than 1? If not, what was the relevance of

Thank you

Jason Brownlee September 21, 2017 at 4:21 pm #

REPLY

Yes, it is a constraint for the chosen framing of the problem so that we can make one-step forecasts with a stateful LSTM.

It is not a constraint generally.

Nidhi September 22, 2017 at 9:58 am #

REPLY

Great post Jason, very granularly explained LSTM example. Thanks a lot. I am trying to train a model to predict scientific notation data like "9.543301358961403E-9", could you suggest a good way to rescale this data that fits LSTM the best?

Jason Brownlee September 23, 2017 at 5:34 am #

REPLY

For input data, rescale values to the range 0-1.

vardhan October 2, 2017 at 4:22 pm #

Hi, I have gone through many of your posts and still confusing when BPTT is applied?? Does B
we take more than one previous inputs to predict the current output??

And in this example, the parameters are updated at each epoch (batch gradient descent) completely cor
is this right??

Jason Brownlee October 3, 2017 at 5:39 am #

This post provides an intro to BPTT:
https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

BPTT is applied when you have more than one time step in your input sequence data.

Aljo Jose October 3, 2017 at 11:42 pm #

Get Your Start in Machine Learning

Hi Jason,
Thank you for great tutorial. Please let me know the difference between below two –

1) model.fit(trainX, trainY, epochs=3000, batch_size=1, verbose=0)

2) for i in range(3000):
model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
model.reset_states()

Jason Brownlee October 4, 2017 at 5:47 am #

REPLY

Not much.

Leave a Reply

Name (required)

Email (will not be published) (required)

Get Your Start in Machine Learning

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

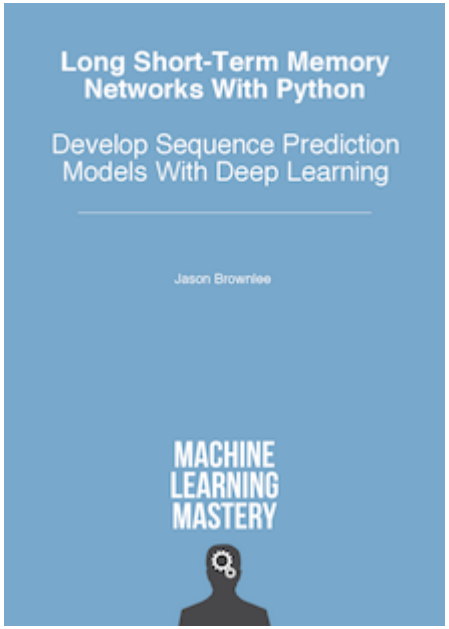
Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Deep Learning for Sequence Prediction

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

Get Started With LSTMs in Python Today!



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

POPULAR

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

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

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

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

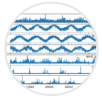
How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

Get Your Start in Machine Learning

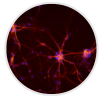
Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

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

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



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



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

Get Your Start in Machine Learning