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

by **Jason Brownlee** on July 21, 2016 in **Deep Learning**

	11	170

Time series prediction problems are a difficult type of predictive modeling problem.

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

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

In this post, you will discover how to develop LSTM networks in Python using the Keras deep learning library to address a demonstration time-series prediction problem.

After completing this tutorial you will know how to implement and develop LSTM networks for your own time series prediction problems and other more general sequence problems. You will know:

- About the International Airline Passengers time-series prediction problem.
- How to develop LSTM networks for regression, window and time-step based framing of time series prediction problems.
- How to develop and make predictions using LSTM networks that maintain state (memory) across very long sequences.

In this tutorial, we will develop a number of LSTMs for a standard time series prediction problem.

The problem and the chosen configuration for the LSTM networks are for demonstration purposes only they are not optimized.

These examples will show you exactly how you can develop your own differently structured LSTM networks for time series predictive modeling problems.

Let's get started.

- **Update Oct/2016**: There was an error in the way that RMSE was calculated in each example. Reported RMSEs were just plain wrong. Now, RMSE is calculated directly from predictions and both RMSE and graphs of predictions are in the units of the original dataset. Models were evaluated using Keras 1.1.0, TensorFlow 0.10.0 and scikit-learn v0.18. Thanks to all those that pointed out the issue, and to Philip O'Brien for helping to point out the fix.
- **Update Mar/2017**: Updated example for Keras 2.0.2, TensorFlow 1.0.1 and Theano 0.9.0.
- **Update Apr/2017**: For a more complete and better explained tutorial of LSTMs for time series forecasting see the post Time Series Forecasting with the Long Short-Term Memory Network in Python.

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
Photo by Margaux-Marguerite Duquesnoy, some rights reserved.

Problem Description

The problem we are going to look at in this post is theInternational Airline Passengers prediction problem.

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

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

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

```
"Month","International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60"
"1949-01",112
"1949-02",118
"1949-03",132
"1949-04",129
"1949-05",121
```

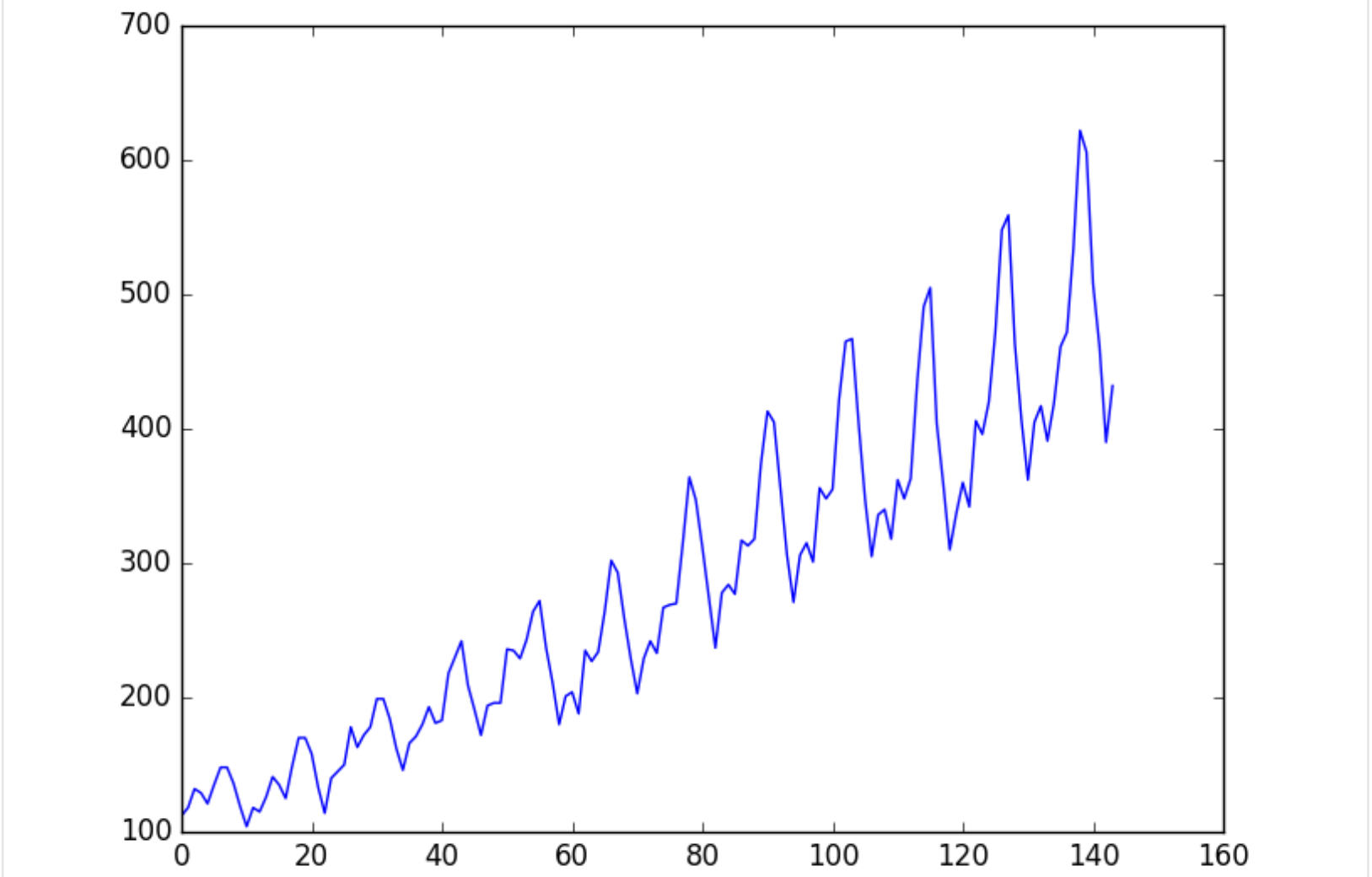
We can load this dataset easily using the Pandas library. We are not interested in the date, given that each observation is separated by the same interval of one month. Therefore, when we load the dataset we can exclude the first column.

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

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

You can see an upward trend in the dataset over time.

You can also see some periodicity to the dataset that probably corresponds to the Northern Hemisphere vacation period.



Plot of the Airline Passengers Dataset

We are going to keep things simple and work with the data as-is.

Normally, it is a good idea to investigate various data preparation techniques to rescale the data and to make it stationary.

Long Short-Term Memory Network

The Long Short-Term Memory network, or LSTM network, is a recurrent neural network tha
and overcomes the vanishing gradient problem.

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

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

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

There are three types of gates within a unit:

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

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

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

LSTM Network for Regression

We can phrase the problem as a regression problem.

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

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

Before we get started, let's first import all of the functions and classes we intend to use. This assumes a working SciPy environment with the Keras deep learning library installed.

```
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

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

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

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

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

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

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

After we model our data and estimate the skill of our model on the training dataset, we nee
unseen data. For a normal classification or regression problem, we would do this using cro

With time series data, the sequence of values is important. A simple method that we can u
datasets. The code below calculates the index of the split point and separates the data into
that we can use to train our model, leaving the remaining 33% for testing the model.

```
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))
```

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

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

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

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

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

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

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

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

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

```
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

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

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

```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

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

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

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

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

Note that we invert the predictions before calculating error scores to ensure that performance is reported in the same units as the original data (thousands of passengers per month).

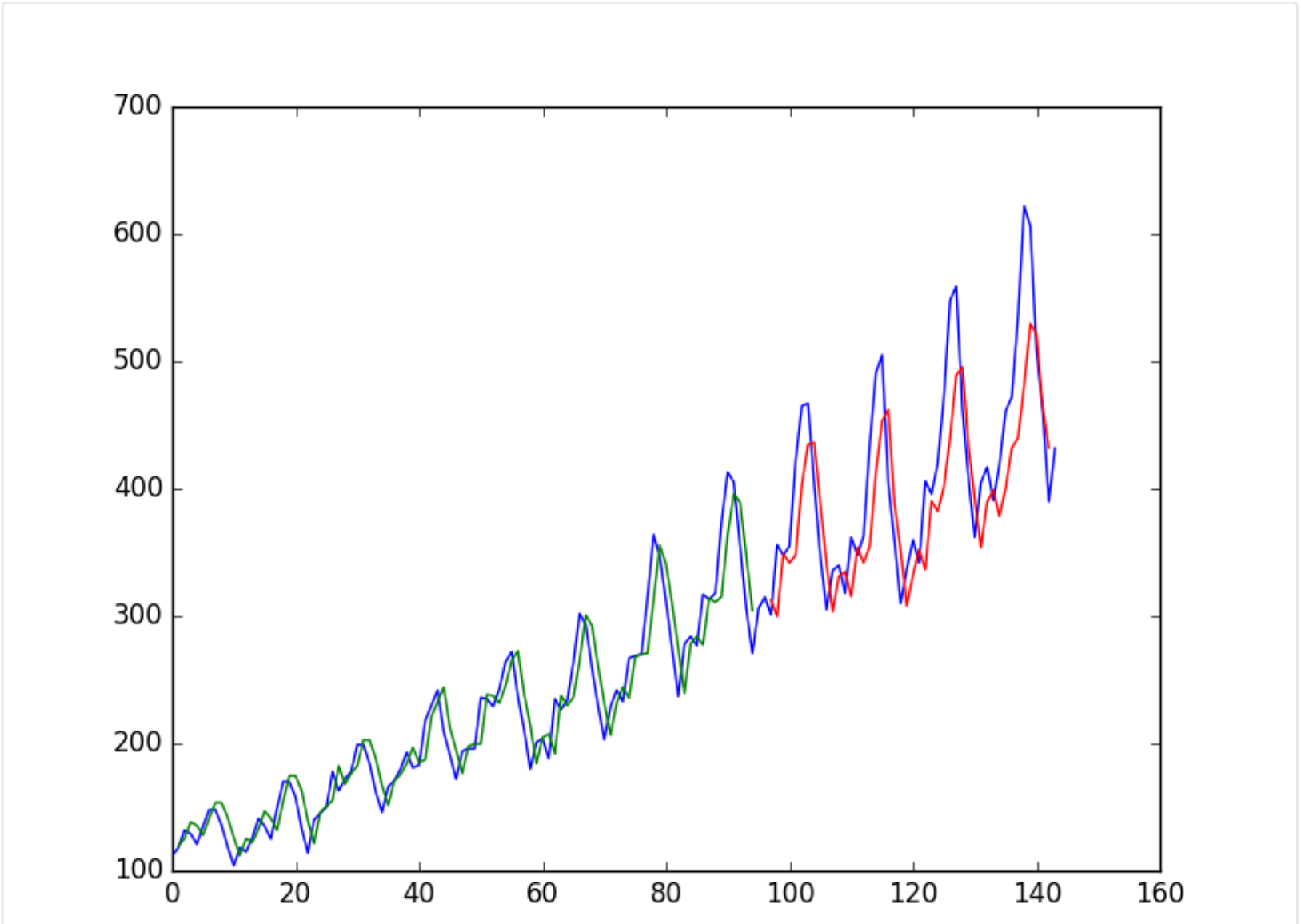
```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

Finally, we can generate predictions using the model for both the train and test dataset to g

Because of how the dataset was prepared, we must shift the predictions so that they align on the x-axis with the original dataset. Once prepared, the data is plotted, showing the original dataset in blue, the predictions for the training dataset in green, and the predictions on the unseen test dataset in red.

```
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

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



LSTM Trained on Regression Formulation of Passenger Prediction Problem

For completeness, below is the entire code example.

```
# LSTM for international airline passengers problem with regression framing
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engin
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
```

```
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Running the example produces the following output.

```
...
Epoch 95/100
0s - loss: 0.0020
Epoch 96/100
0s - loss: 0.0020
Epoch 97/100
0s - loss: 0.0020
Epoch 98/100
0s - loss: 0.0020
Epoch 99/100
0s - loss: 0.0020
Epoch 100/100
0s - loss: 0.0020
Train Score: 22.93 RMSE
Test Score: 47.53 RMSE
```

We can see that the model has an average error of about 23 passengers (in thousands) on the training dataset, and about 52 passengers (in thousands) on the test dataset. Not that bad.

LSTM for Regression Using the Window Method

We can also phrase the problem so that multiple, recent time steps can be used to make the prediction for the next time step.

This is called a window, and the size of the window is a parameter that can be tuned for each problem.

For example, given the current time (t) we want to predict the value at the next time in the sequence (t+1), we can use the current time (t), as well as the two prior times (t-1 and t-2) as input variables.

When phrased as a regression problem, the input variables are t-2, t-1, t and the output va

The **create_dataset()** function we created in the previous section allows us to create this f g the **look_back** argument from 1 to 3.

A sample of the dataset with this formulation looks as follows:

```
X1  X2  X3  Y
112 118 132 129
118 132 129 121
132 129 121 135
129 121 135 148
121 135 148 148
```

We can re-run the example in the previous section with the larger window size. The whole code listing with just the window size change is listed below for completeness.

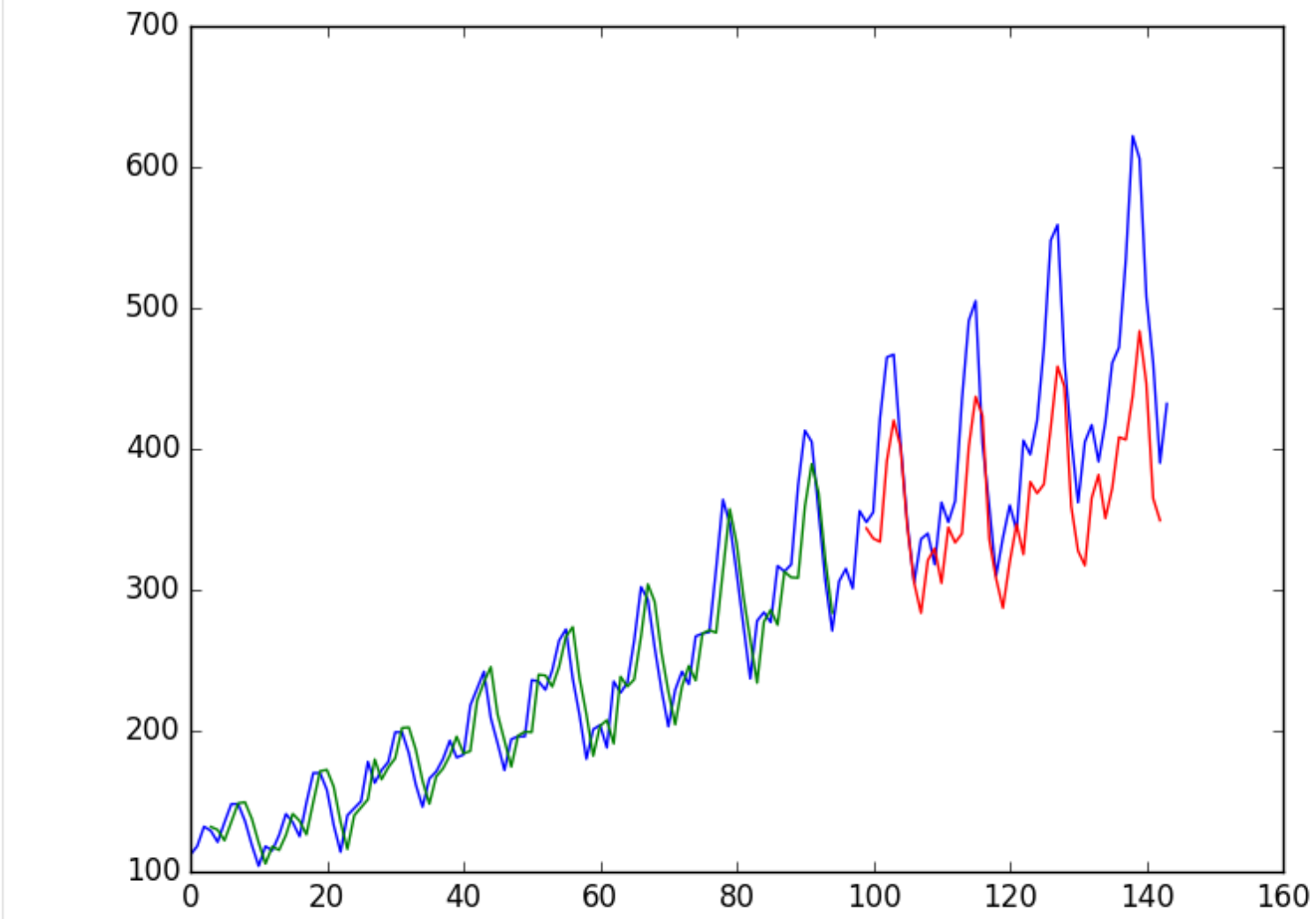
```
# LSTM for international airline passengers problem with window regression framing
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Running the example provides the following output:

```
...
Epoch 95/100
0s - loss: 0.0021
Epoch 96/100
0s - loss: 0.0021
Epoch 97/100
0s - loss: 0.0021
Epoch 98/100
0s - loss: 0.0021
Epoch 99/100
0s - loss: 0.0022
Epoch 100/100
0s - loss: 0.0020
Train Score: 24.19 RMSE
Test Score: 58.03 RMSE
```

We can see that the error was increased slightly compared to that of the previous section.

not tuned: this is just a demonstration of how to frame a prediction problem.



LSTM Trained on Window Method Formulation of Passenger Prediction Problem

LSTM for Regression with Time Steps

You may have noticed that the data preparation for the LSTM network includes time steps.

Some sequence problems may have a varied number of time steps per sample. For example, you may have measurements of a physical machine leading up to a point of failure or a point of surge. Each incident would be a sample the observations that lead up to the event would be the time steps, and the variables observed would be the features.

Time steps provide another way to phrase our time series problem. Like above in the window example, we can take prior time steps in our time series as inputs to predict the output at the next time step.

Instead of phrasing the past observations as separate input features, we can use them as time steps of the one input feature, which is indeed a more accurate framing of the problem.

We can do this using the same data representation as in the previous window-based example, except when we reshape the data, we set the columns to be the time steps dimension and change the features dimension back to 1. For example:

```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
```

The entire code listing is provided below for completeness.

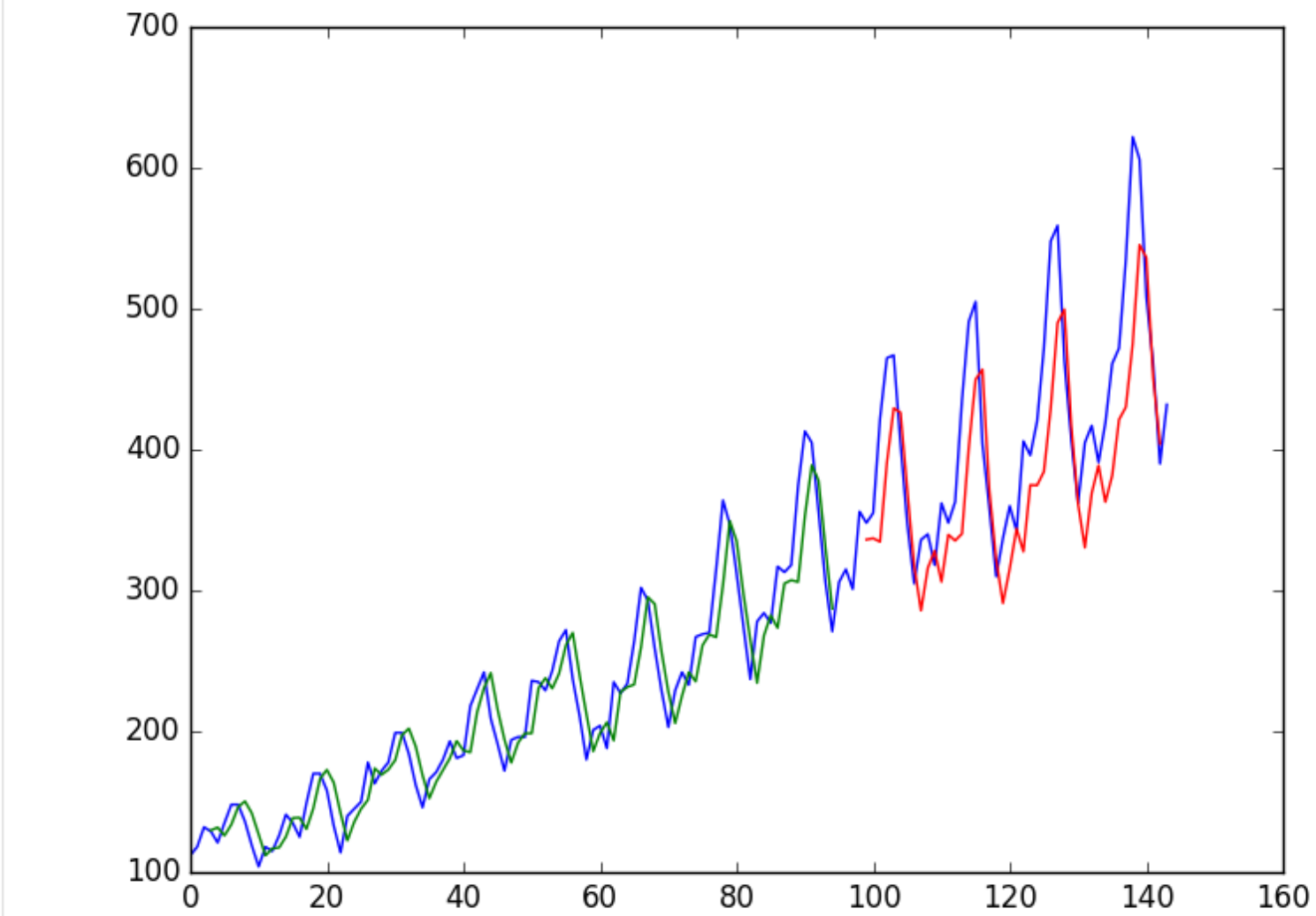
```
# LSTM for international airline passengers problem with time step regression f
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
```

```
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(look_back, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Running the example provides the following output:

```
...
Epoch 95/100
1s - loss: 0.0021
Epoch 96/100
1s - loss: 0.0021
Epoch 97/100
1s - loss: 0.0021
Epoch 98/100
1s - loss: 0.0020
Epoch 99/100
1s - loss: 0.0021
Epoch 100/100
1s - loss: 0.0020
Train Score: 23.69 RMSE
Test Score: 58.88 RMSE
```

We can see that the results are slightly better than previous example, although the structure of the input data makes a lot more sense.



LSTM Trained on Time Step Formulation of Passenger Prediction Problem

LSTM with Memory Between Batches

The LSTM network has memory, which is capable of remembering across long sequences.

Normally, the state within the network is reset after each training batch when fitting the model, as well as each call to **model.predict()** or **model.evaluate()**.

We can gain finer control over when the internal state of the LSTM network is cleared in Keras by making the LSTM layer "stateful". This means that it can build state over the entire training sequence and even maintain that state if needed to make predictions.

It requires that the training data not be shuffled when fitting the network. It also requires explicit resetting of the network state after each exposure to the training data (epoch) by calls to **model.reset_states()**. This means that we must create our own outer loop of epochs and within each epoch call **model.fit()** and **model.reset_states()**. For example:

```
for i in range(100):
    model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
```

Finally, when the LSTM layer is constructed, the **stateful** parameter must be set **True** and instead of specifying the input dimensions, we must hard code the number of samples in a batch, number of time steps in a sample and number of features in a time step by setting the **batch_input_shape** parameter. For example:

```
model.add(LSTM(4, batch_input_shape=(batch_size, time_steps, features), stateful=True))
```

This same batch size must then be used later when evaluating the model and making predictions. For example:

```
model.predict(trainX, batch_size=batch_size)
```

We can adapt the previous time step example to use a stateful LSTM. The full code listing

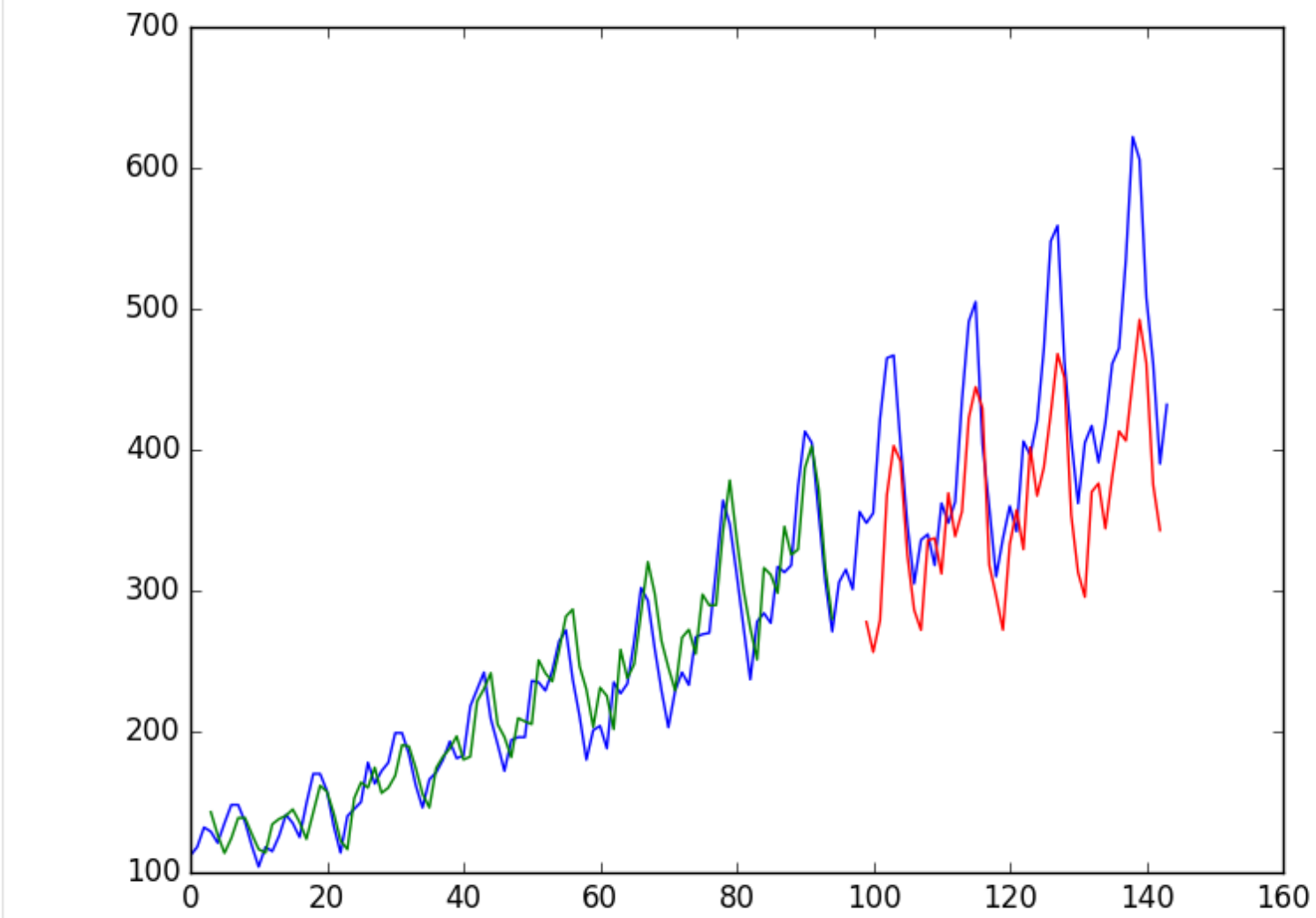
```
# LSTM for international airline passengers problem with memory
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
```

```
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# create and fit the LSTM network
batch_size = 1
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
    model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Running the example provides the following output:

```
...
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0016
Train Score: 20.74 RMSE
Test Score: 52.23 RMSE
```

We do see that results are worse. The model may need more modules and may need to be re of the problem.



Stateful LSTM Trained on Regression Formulation of Passenger Prediction Problem

Stacked LSTMs with Memory Between Batches

Finally, we will take a look at one of the big benefits of LSTMs: the fact that they can be successfully trained when stacked into deep network architectures.

LSTM networks can be stacked in Keras in the same way that other layer types can be stacked. One addition to the configuration that is required is that an LSTM layer prior to each subsequent LSTM layer must return the sequence. This can be done by setting the **return_sequences** parameter on the layer to **True**.

We can extend the stateful LSTM in the previous section to have two layers, as follows:

```
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
```

The entire code listing is provided below for completeness.

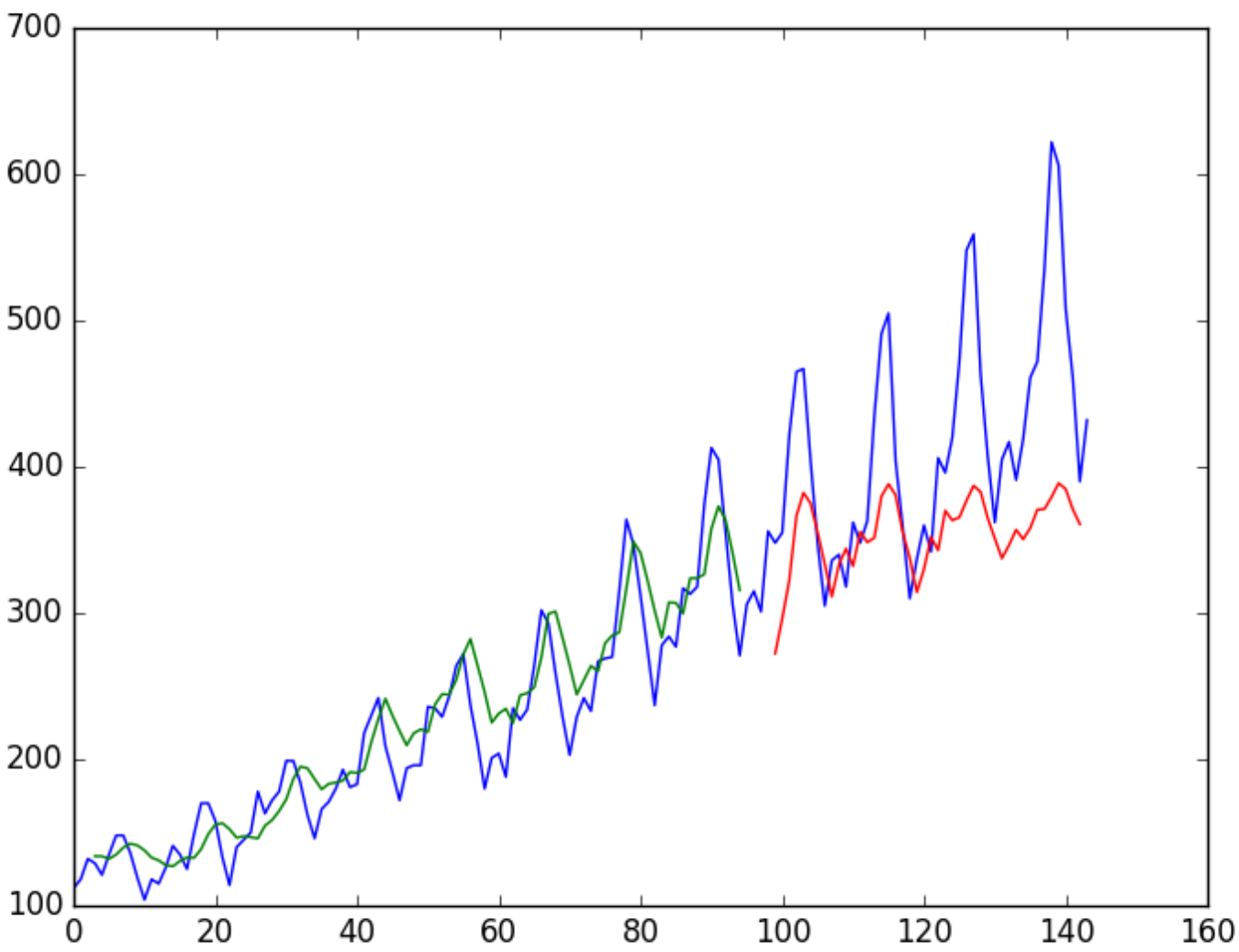
```
# Stacked LSTM for international airline passengers problem with memory
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engin
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

```
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# create and fit the LSTM network
batch_size = 1
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
    model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Running the example produces the following output.

```
...
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0017
Epoch 1/1
1s - loss: 0.0016
Train Score: 20.49 RMSE
Test Score: 56.35 RMSE
```

The predictions on the test dataset are again worse. This is more evidence to suggest the need for additional training epochs.



Stacked Stateful LSTMs Trained on Regression Formulation of Passenger Prediction Problem

Summary

In this post, you discovered how to develop LSTM recurrent neural networks for time series prediction in Python with the Keras deep learning network.

Specifically, you learned:

- About the international airline passenger time series prediction problem.
- How to create an LSTM for a regression and a window formulation of the time series problem.
- How to create an LSTM with a time step formulation of the time series problem.
- How to create an LSTM with state and stacked LSTMs with state to learn long sequences.

Do you have any questions about LSTMs for time series prediction or about this post?
Ask your questions in the comments below and I will do my best to answer.

Related Posts

For a more complete and better explained tutorial of LSTMs for time series forecasting see the post:

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

Looking for some more tutorials on LSTMs in Python with Keras? Take a look at some of these:

- Time Series Prediction With Deep Learning in Keras
- Understanding Stateful LSTM Recurrent Neural Networks in Python with Keras
- Text Generation With LSTM Recurrent Neural Networks in Python with Keras
- Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras

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 →

‹ How To Estimate A Baseline Performance For Your Machine Learning Models in Weka

How To Use Regression Machine Learning Algorithms in Weka ›

460 Responses to *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*

Shilin Zhang July 21, 2016 at 12:49 pm #

REPLY

I like Keras, the example is excellent.

Jason Brownlee July 21, 2016 at 1:55 pm #

REPLY

Thanks.

Robin Schäfer November 4, 2016 at 6:41 am #

REPLY

Hi, thanks for your awesome tutorial!

I just don’t get one thing… If you’d like to predict 1 step in the future, why does the red line stop before the blue line does?

So for example, we have the testset untill end of the year 1960. How can i predict the future year? Or passangers at the 1/1/1961 (if dataset ends at 12/31/1960).

Best,

Robin

Jason Brownlee November 4, 2016 at 11:16 am #

REPLY

Great question, there might be a small bug in how I am displaying the predictions in the plot.

Terrence November 15, 2016 at 12:06 am #

Hey, great tutorial.

I have the same question about the future prediction. The “testPredict” has tw… running, how would I obtain the values for the a prediction 1 or 2 days ahead

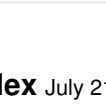
Terrence

zhang February 3, 2017 at 7:46 am #

I think in the ”create_dataset” function, the range should be “len(dataset)-look_back” but not “len(dataset)-look_back-1”. No “1”should be subtracted here.

Shiva May 11, 2017 at 5:35 am #

Hi Jason,
how to fix this bug? what modifications you need to make in the code to predict the values for 1/31/1961, if the dataset ends at 12/31/1960?



Shiva May 11, 2017 at 5:33 am #

REPLY

Hi Robin,
Are you up with a solution for the bug? as you rightly said, the testpredict is smaller than test. How do you modify the code so that it predicts the value on 1/1/1961?

Shovon Sengupta February 16, 2017 at 1:00 am #

REPLY

Hello Jason,

Thanks for sharing this great tutorial! Can you please also suggest the way to get the forward forecast based on the LSTM method. For example, if we want to forecast the value of the series for the next few weeks (ahead of current time–As we usually do for the any time series data), then what would be process to do that.

Regards
Shovon

Jason Brownlee February 16, 2017 at 11:07 am #

REPLY

Hi Shovon,

I would suggest reframing the problem to predict a long output sequence, then develop a model on this framing of the problem.

Alex July 21, 2016 at 1:04 pm #

REPLY

Hi, thanks for the walkthrough. I’ve tried modifying the code to run the network for online prediction, but I’m having some issues. Would you be willing to take a look at my SO question? http://stackoverflow.com/questions/38313673/lstm-with-keras-for-mini-batch-training-and-online-testing

Cheers,
Alex

Jason Brownlee July 23, 2016 at 2:15 pm #

REPLY

Sorry Alex, you’re question is a little vague.

It’s of the order “I have data like this…, what is the best way to model this problem”. It’s a tough StackOverflow question because it’s an open question rather than a specific technical question.

Generally, my answer would be “no idea, try lots of stuff and see what works best”.

I think your notion of online might also be confused (or I’m confused). Have you seen onli t
it as far as I know. You train your model then you make predictions. Unless of course you
not online learning, it is just static model with state, the weights are not updated in an onli

It might be worth stepping back from the code and taking some time to clearly define I/O c
right kind of algorithm/setup you need to solve it.

Tommy Johnson July 26, 2016 at 2:26 am #

REPLY

Hello Dr. Brownlee,
I have a question about the difference between the Time Steps and Windows method. Am I c

shape of the data you feeding into the model? If so, can you give some intuition why the Time Steps method works better? If I have two sequences (For example, if I have 2 noisy signals, one noisier than the other), and I'm using them both to predict a sequence, which method do you think is better?

Best

Jason Brownlee July 26, 2016 at 5:58 am #

REPLY

Hi Tommy,

The window method creates new features, like new attributes for the model, where as timesteps are a sequence within a batch for a given feature.

I would not say one works better than another, the examples in this post are for demonstration only and are not tuned.

I would advise you to try both methods and see what works best, or frame your problem in the way that best makes sense.

Pedro Ferreira July 29, 2016 at 1:48 am #

REPLY

Hi Jason,

What are the hyperparameters of your network?

Thanks,
Pedro

Jason Brownlee July 29, 2016 at 6:30 am #

REPLY

Hi Pedro, the hyperparameters for each network are available right there in the code.

Evgeni Stavinov January 19, 2017 at 11:30 pm #

REPLY

Is it possible to perform hyperparameter optimization of the LTSM, for example using hyperopt?

Jason Brownlee January 20, 2017 at 10:21 am #

REPLY

I don't see why not Evgeni., Sorry I don't have an example.

Jack Kinkade July 30, 2016 at 7:41 pm #

REPLY

Hi Jason,

Interesting post and a very useful website! Can I use LSTMS for time series classification, for a binary supervised problem? My data is arranged as time steps of 1 hr sequences leading up to an event and the occurrence and non-occurrence of the event are labelled in each instance. I have done a bit of research and have not seen many use cases in the literature. Do you think a different recurrent neural net or simpler MLP might work better in this case? Most of my the research done in my area has got OK results(70% accuracy) from feed forward neural networks and i thought to try out recurrent neural nets, specifically LSTMs to improve my accuracy.

Jason Brownlee July 31, 2016 at 7:12 am #

I don't see why not Jack.

I would suggest using a standard MLP with the window method as the baseline, then deve
LSTMs to generally perform better if there is information in the long sequences.

This post on binary classification may help, you can combine some of the details with the
output layer and loss function):
http://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learn

Peter Ostrowski July 31, 2016 at 11:19 pm #

REPLY

Hi Jason,

Thanks for this example. I ran the first code example (lookback=1) by just copying the code and can reproduce your train and test scores precisely, however my graph looks differently. Specifically for me the predicted graph (green and red lines) looks as if it is shifted by one to the right in comparison to what I see on this page. It also looks like the predicted graph starts at x=0 in your example, but my predicted graph starts at 1. So in my case it looks like the prediction is almost like predicting identity? Is there a way for me to verify what I could have done wrong?

Thanks,
Peter

Jason Brownlee August 1, 2016 at 6:26 am #

REPLY

Thanks Peter.

I think you're right, I need to update my graphs in the post.

Peter Ostrowski August 2, 2016 at 12:05 am #

REPLY

Hi Jason,

when outputting the train and test score, you scale the output of the model.evaluate with the minmaxscaler to match into the original scale. I am not sure if I understand that correctly. The data values are between 104 and 622, the trainScore (which is the mean squared error) will be scaled into that range using a linear mapping, right? So your transformed trainscore can never be lower than the minimum of the dataset, i.e. 104. Shouldn't the square root of the trainScore be transformed and then the minimum of the range be subtracted and squared again to get the mean square error in the original domain range? Like numpy.square(scalar.inverse_transform([[nump.sqrt(trainScore)]])-scaler.data_min_)

Thanks,
Peter

Jason Brownlee August 3, 2016 at 8:33 am #

REPLY

Hi Peter, you may have found a bug, thanks.

I believe I thought the default evaluation metric was RMSE rather than MSE and I was using the scaler to transform the RMSE score back into original units.

I will update the examples ASAP.

Update: All estimates of model error were updated to first convert the error score to RMSE and then invert scale transform back to original units.

seiya.kumada August 2, 2016 at 3:17 pm #

REPLY

Thank you for your excellent post.

I have one question.
In your examples, you are discussing a predictor such as {x(t-2),x(t-1),x(t)} -> x(t+1).
I want to know how to implement a predictor like {x(t-2),x(t-1),x(t)} -> {x(t+1), x(t+2)}.
Could you tell me how to do so?

Jason Brownlee August 3, 2016 at 5:54 am #

REPLY

This is a sequence in and sequence out type problem.

I believe you prepare the dataset in this form and model it directly with LSTMs and MLPs.

I don't have a worked example at this stage for you, but I believe it would be straight forwa

Sachin August 2, 2016 at 6:08 pm #

REPLY

Hi,

First of all thanks for the tutorial. An excellent one at that.

However, I do have some questions regarding the underlying architecture that I'm trying to reconcile with what I've done learnt about. I posted a question here: http://stackoverflow.com/questions/38714959/understanding-keras-lstms which I felt was too long to post in this forum.

I would really appreciate your input, especially the question on time_steps vs features argument.

Thanks,
Sachin

Jason Brownlee August 3, 2016 at 6:01 am #

REPLY

If I understand correctly, you want more elaboration on time steps vs features?

Features are your input variables. In this airline example we only have one input variable, but we can contrive multiple input variables using past time steps in what is called the window method. Normally, multiple features would be a multivariate time series.

Timesteps are the sequence through time for a give attribute. As we comment in the tutorial, this is the natural mapping of the problem onto LSTMs for this airline problem.

You always need a 3D array as input for LSTMs [samples, features, timesteps], but you can reduce each dimension to one if needed. We explore this ability in reframe the problem in the tutorial above.

You also ask about the point of stateful. It is helpful to have memory between batches over one training run. If we keep all of out time series samples in order, the method can learn the relationships between values across batches. If we did not enable the stateful parameter, the algorithm we no knowledge beyond each batch, much like a MLP.

I hope that helps, I'm happy to dig into a specific topic further if you have more questions.

Sachin August 4, 2016 at 3:54 pm #

REPLY

So does that mean (in reference to the LSTM diagram in http://colah.github.io/posts/2015-08-Understanding-LSTMs/) that the cell memory is not passed between consecutive lstms if stateful=false (i.e. set to zero)? Or do you mean cell memory is reset to zero between consecutive batches (In this tutorial batch_size is 1). Although I guess I should point out that the hidden layer values are passed on, so it will still be different to a MLP (wouldn't it?)

On a side note, the fact that the output has to be a factor of batch_size seems to be confounding. Feels like it limits me to using a batch_size of one.

Jason Brownlee August 5, 2016 at 8:38 am #

REPLY

If stateful is set to false (the default), then I understand according to the Keras documentation that the state within each LSTM node is reset after each batch, either for prediction or training.

This is useful if you do not want to use LSTMs in a stateful manner of you want to train with all of the required memory to learn from within each batch.

This does tie into the limit on batch size for prediction. The TF/Theano structures created from this network definition are optimized for the batch size.

Mango Freezz October 16, 2016 at 6:18 am #

REPLY

I'm super confused here. If the LSTM node is reset after each batch (in this c… backprop session, the LSTM starts with a fresh state without any memory of previous… the case, how could it possibly learn anything?

E.g., let's say on both time step 10 and 15 the input value is 150, how does the netwo… 130 while the only input is 150 and the LSTM start with a fresh state?

ARandomPerson December 6, 2016 at 9:35 am #

Hi Mango, I think you're right. If the number of time-steps is one and the… recurrent property of the LSTM at all.

Nuno Fonseca August 4, 2016 at 8:52 pm #

REPLY

Hi!

First of all, thank you for that great post

I have just one small question: For some research work i am working on, I need to make a prediction, so I've been looking for the best possible solution and I am guessing its LSTM…

The app. that I am developing is used in a learning environment, so to predict is the probability of a certain student will submit one solution for a certain assignment…

I have data from previous years in this format:

A1 A2 A3 A4 …
Student 1 – Y Y Y Y N Y Y N
Student 2 – N N N N N Y Y Y
…

Where Y means that the student has submitted, and N otherwise…

From what I understood, the best to achieve what I need is by using the solution described in the section "LSTM For Regression Using the Window Method" where my data will be something like

I1 I2 I3 O
N N N N
Y Y Y Y

And when I present a new case like Y N N the "LSTM" will make a prediction according to what has been learnt in the training moment.

Did I understand it right? Do you suggest another way?

Sorry for the eventually dumb question…

Best regards

Jason Brownlee August 5, 2016 at 5:30 am #

REPLY

Looks fine Nuno, although I would suggest you try to frame the problem a few different ways and see what gives you the best results.

Also compare results from LSTMs to MLPs with the window method and ensure it is worth the additional complexity.

Dunhui Xiao August 7, 2016 at 6:59 am #

REPLY

Hi Jason,
Very interesting. Is there a function to descale the scaled data (0-1)? You show the data from 0-1. I want to see the original scale data. This is a easy question. But, it is better to show the original scale data, I suppose.

Jason Brownlee August 7, 2016 at 8:46 am #

REPLY

Great point.

Yes, you can save the MinMaxScaler used to scale the training data and use it later to scale new input data and in turn descale predictions. The call is scaler.inverse_transform() from memory.

Pacchu August 9, 2016 at 5:09 am #

Why is the shift necessary for plotting the output? Isn't it unavailable information at ti

---

Jason Brownlee August 15, 2016 at 9:46 am #

Hi Pacchu, the framing of the problem is to predict t+1, given t, and possibly som

I hope that is clearer.

Mat August 10, 2016 at 12:06 am #

REPLY

Does the output simply mimics the input ? (the copy is shifted by one)
Just like here : https://github.com/fchollet/keras/issues/2856 ?

Jason Brownlee August 15, 2016 at 9:47 am #

REPLY

No, the output is a prediction of the next time step given prior time steps.

André C. Andersen May 14, 2017 at 1:45 am #

REPLY

Have you tried to use the input value as a prediction? It produces an RMSE similar to what you are getting, 48.66.

Jason Brownlee May 14, 2017 at 7:31 am #

REPLY

Yes, this is call persistence, here is an example:
http://machinelearningmastery.com/persistence-time-series-forecasting-with-python/

Shaifu August 10, 2016 at 6:35 pm #

REPLY

Hi sir

I tried your code for time series prediction. On passing either univariate or multivariate data, the predictions of the target variable are same. Should’nt there be a difference in the predicted values. I expect the predictions to improve with the multivariate data. Please shed some light on this.

Jason Brownlee August 15, 2016 at 9:47 am #

REPLY

The performance of the model is dependent on both the framing of your problem and how the model is configured.

Madhav August 17, 2016 at 4:02 pm #

REPLY

Hi Jason,

Thanks for the wonderful tutorial. It felt great following your code and implementing my first LSTM network. Looking forward to learning a lot more!!

Can we extend time series forecasting problems to multiple time series? I have the following problem in my mind. Suppose we have stock prices of 100 companies (instead of one) and we wanna forecast what happens in the next month for all the companies? Is it possible to use LSTMs and RNNs for such multiple time series problems?

Jason Brownlee August 18, 2016 at 7:15 am #

REPLY

Forecasting stock prices is not my area of expertise. Nevertheless, LSTMs can be used for multiple regression as well as sequence prediction (if you want to predict multiple steps ahead). Give it a shot.

Mango December 18, 2016 at 10:08 pm #

i guess i have the same idea in mind as Madhav..^^ i want to predict multiple tim
the city(since i assume that the neighboured grids influence each other to some extend)..
you share me some tricks or experience? Thankyou~

Liu August 18, 2016 at 2:11 am #

REPLY

I guess the function learnt is only an one-step lag identity (mimic) prediction.

If the code of your basic version runs, it will look like this:

http://imgur.com/BvPnwGu

I change the csv (setting all the data points after some time to be 400 until the end) and run the same code, it will look like this:

http://imgur.com/TvuQDRZ

If it is truly learning the dynamics of the pattern, the prediction should not look like a strict line. At least the previous information before the 400 values will pull down the curve a little bit.

Liu August 18, 2016 at 3:44 am #

REPLY

Typo: a *straight line

Clarification: Of course what I said may not be correct. But I think this is an alarming concern to interrupt what the LSTM is really learning behind the scene.

Jason Brownlee August 18, 2016 at 8:01 am #

REPLY

A key to the examples is that the LSTM is capable of learning the sequence, not just the input-output pairs. From the sequence, it is able to predict the next step.

Nicholas August 19, 2016 at 7:16 am #

REPLY

Hi Liu,

after investigating a bit, I have concluded that the 1 time-step LSTM is indeed the trivial identity function (you can convince yourself by reducing the layer to 1 neuron, and adding ad-hoc data to the test set, as you have). But if we think about it, this makes alot of sense that the ‘mimic’ function would minimize MSE for such a simple network – it doesn’t see enough time steps to learn the sequence, anyways.

However, if you increase the number of timesteps, you will see that it can reach lower MSE on the test set by slowly moving away from the mimic function to actually learning the sequence, although for low #’s of neurons the approximation will be rougher-looking. I recommend experimenting with the look_back amount, and adding more layers to see how the MSE can be reduced further!

Liu August 20, 2016 at 8:47 am #

REPLY

Hi Nicholas,

Thanks for the comment!

I guess the problem (or feature you can say) in the first example is that ‘time-step’ is set to 1 if I understand the API correctly:

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

It means it is feeding sequence of length 1 to the network in each training. Therefore, the RNN/LSTM is not unrolled. The updated internal state is not used anywhere (as it is also resetting the states of the machine in each batch).

I agree with what you said. But by setting timestep and look_back to be 1 as in the first example, it is not learning a sequence at all.

For other readers, I think it worths to look at http://stackoverflow.com/questions/38714959/understanding-keras-lstms

---

Gilles September 2, 2016 at 5:44 pm #

REPLY

Hi Nicholas,

This is a very good point, thanks for mentioning it.

I have implemented an LSTM on a 1,500 points sample size and indeed sometimes I difference with a “mimic” function.

A lot of work to predict the value t+1 while value at t would have been a good enough

Will try more experiments as I have more data.

Logan April 6, 2017 at 4:44 am #

REPLY

hey Liu, it's a very good observation. I still on the basics and I think these sort of information is really important if we want to build something with quality. Thanks.
Thanks for the tutorial as well.

Chris August 20, 2016 at 12:07 am #

REPLY

Hi Jason,

Thanks for this amazing tut, could you please tell me about what is the main role of batch_size in model.fit() and output shape of LSTM layer parameter ?
I read somewhere that using batch_size is depend on our dataset why you chose batch_size = 1 for fitting model and what is the effect of choosing it's value on calculating gradient of the model?

thanks

Jason Brownlee August 20, 2016 at 6:09 am #

REPLY

Great question Chris.

The batch_size is the number of samples from your train dataset shown to the model at a time. After batch_size samples are run through the network and error calculated, an update to the weights is performed. Too many and the updates are too big, too few, and the updates are too noisy. The hardware you use is also a factor for batch_size and you want to ensure you can fit the batch of samples in memory (e.g. so your GPU can get at them).

I chose a batch_size of 1 because I want to explore and demonstrate LSTMs on time series working with only one sample at a time, but potentially vary the number of features or time steps.

Hany El-Ghaish August 22, 2016 at 8:46 am #

REPLY

Hi Jason,

Thanks for this series. I have a question for you.
I want to apply a multi-classification problem for videos using LSTM. also, video samples have a different number of frames.
Dataset: samples of videos for actions like boxing, jumping, hand waving, etc.. (Dataset like UCF1101) . each class of action has one label.

so, each video has a label.
Really, I do not know how to describe the data set to LSTM when a number of frames sequence are different from one action to another and I do not know how to use it when a number of frames are fixed as well.

if you have any example of how to use:
LSTM, stacked of LSTM, or CNN with LSTM with this problem this will help me too much.
I wait for your recommendations
Thanks

Harsha August 30, 2016 at 7:47 pm #

REPLY

Hi Jason. Thanks for such a wonderful tutorial. it helped me a lot to get an insight on LSTM's. I too have a similar question. Can you please comment on this question.

Alvin August 26, 2016 at 3:02 am #

REPLY

Hi Jason,

Thanks for this great tutorial! I would like to ask, suppose I have a set of X points : X1, X2, .. X
represented by Y, and I have 60 days data (Y1 until Y60), how do I do time series forecast usi
Y65. Do you have any sample or coding references?

Thanks in advance

Jason Brownlee August 26, 2016 at 10:34 am #

REPLY

I believe you could adapt one of the examples in your post directly to your problem. Little effort required.

Consider normalizing or standardizing your input and output values when working with neural networks.

Alvin August 30, 2016 at 8:31 am #

REPLY

Hi Jason,

I just found out the question that I have is a multi step ahead prediction, where all the X contributes to Y, and I would like to predict ahead the value of Y n days ahead. Is the example that you gave in this tutorial still relevant?

Thanks!

Jason Brownlee August 31, 2016 at 8:42 am #

REPLY

Hi Alvin,

Yes, you could trivially extend the example to do sequence-to-sequence prediction as it is called in recurrent neural network parlance.

Alvin August 31, 2016 at 1:03 pm #

Hi Jason,

Thanks for your reply. I still would like to clarify after looking at the sequence to sequence concept. Assuming I would like to predict the daily total sales (Y), given x1 such as the total number of customers, total item A sold as x2, total item B sold as x3 and so on for the next few items, is sequence to sequence suitable for this?

Thanks

Alvin August 31, 2016 at 6:05 pm #

Hi Jason,

I have another question. Looking at your example for the Window method, on line 35:
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

what if I would like to change the time steps to more than 1? What other parts of codes I would need to change? Currently when I change it, it says
ValueError: total size of new array must be unchanged.

Thanks

Alvin September 13, 2016 at 11:23 am #

Hi Jason,

For using stateful LSTM, to predict multiple steps, I came across suggestions to feed the output vector back into the model for the next timestep's prediction. May I know how to get an output vector from based

Jason Brownlee September 14, 2016 at 10:04 am #

Hi Alvin,

The LSTM will maintain internal state so that you only need to provide the nex
does require that you provide your data in consistent batch sizes, but when ex
down to 1.

I hope that helps.

DRD August 29, 2016 at 2:25 am #

REPLY

Apparently, when using the tensor flow backend, you have to specify input_length in the LTSM constructor. Otherwise, you get an exception. I assume it would just be input_length=1

DRD August 29, 2016 at 2:29 am #

REPLY

So like this:
model.add(LSTM(4, input_dim=look_back, input_length=1))

This references the first example where number of features and timesteps is 1. Here input_length corresponds to timesteps.

Alvin August 30, 2016 at 8:32 am #

REPLY

Hi DRD,

Is this the setting used to solve multi step ahead prediction?

Thanks in advance!

DRD September 1, 2016 at 6:19 am #

REPLY

HI,

Haven't tried it yet, but in the section titled: "LSTM With Memory Between Batches

" input_length should be 3. Basically the same as look_back

Nick August 31, 2016 at 1:34 am #

REPLY

Hi Jason,
I applied your technique on stock prediction:
But, I am having some issues.
I take all the historical prices of a stock and frame it the same way the airline passenger prices are in a .csv file.

I use a look_back=20 and I get the following image:

https://postimg.org/image/p53hw2nc7/

Then I try to predict the next stock price and the prediction is not accurate.

Why is the model able to predict the airline passengers so precisely ?

https://postimg.org/image/z0lywvru1/

Thank you

Jason Brownlee August 31, 2016 at 9:47 am #

REPLY

I would suggest tuning the number of layers and number of blocks to suits your problem.

Nader September 1, 2016 at 7:25 am #

REPLY

Thank you.
I will play around the network.

In general, For input_dim (windows size), is a smaller or larger number better ?

Marcel August 31, 2016 at 8:29 pm #

REPLY

Hi Jason,

First off, thanks again for this great blog, without you I would be nowhere, with LSTM, and life

I am running a LSTM model that works, but when I make predictions with “model.predict” it spits out 4000 predictions, which look fine. However, when I run “model.predict” again and save those 4000 predictions again, they are different. From prediction 50 onward, they are all essentially the same, but the first few (that are very important to me) are very different. To give you an idea, the correlation between the first 10 predictions of both rounds is 0.11.

Please help!

Marcel August 31, 2016 at 10:55 pm #

REPLY

The problem wasn’t with numpy.random.seed(0) as I originally thought. I’ve tested this over and over, and even if on the exact same data, predictions are always different/inconsistent for the first few predictions, and only “converge” to some consistent predictions after about 50 predictions have been made previously (on the same or different input data).

Marcel September 1, 2016 at 1:15 am #

REPLY

It seems like I have made an error by neglecting to include “model.reset_states()” after one line of calling model.predict()

Jason Brownlee September 1, 2016 at 8:05 am #

REPLY

I’m glad to hear you worked it out Marcel.

A good lesson for all of to remember or calls to reset state.

Nader September 1, 2016 at 9:30 pm #

REPLY

In the part “LSTM For Regression with Time Steps”,

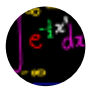
should’t the reshaping be in the form:

[Samples, Features, Time] = (trainX, (trainX.shape[0], trainX.shape[1], 1]

Because in the previous two section:
“LSTM Network For Regression” and
“LSTM For Regression Using the Window Method” we used:

[Samples, Time Steps, Features] = (trainX, (trainX.shape[0], 1, trainX.shape[1])

Thank you



sachin September 2, 2016 at 3:57 pm #

REPLY

Hi Jason,

Correct me if I’m wrong, but you don’t want to reset_state in the last training iteration do you? Basically my logic is that you want to carry through the last ‘state’ onto the test set because they occur right after the other.

Cheers,
Sachin

Jason Brownlee September 3, 2016 at 6:57 am #

REPLY

You do. The reason is that you can seed the network with state later when you a

Megs September 3, 2016 at 10:04 pm #

REPLY

Hello Jason,

Am I correct if I was to use Recurrent Neural Networks to predict Dengue Incidences against data on temperature, rainfall, humidity, and dengue incidences.. If so, how would I go about in the processing of my data. I already have the aforementioned data at hand and I have tried using a feed forward neural network using pybrain. It doesn't seem to get the trend hence my trying of Recurrent Neural Network.

Thank you!

Christoph September 5, 2016 at 4:22 am #

REPLY

I am a little bit confused regarding the "statefulness".

If I use a Sequential Model with LSTM layers and stateful set to false. Will this still be a recurrent network that feeds back into my nodes? How would I compare it to the standard LSTM model proposed by Hochreiter et al. (1997)? Do I have to use the stateful layers to mimic the behaviour presented in the original paper?

In essence, I have a simple time series of sales forecasts that show a weekly and partly a yearly pattern. It was easy to create a simple MLP with the Dense layer and the time window method. I put some sales values from the last week, the same week day a few weeks back and the sales of the days roughly a year before into my feature vector. Results are pretty good so far.

I now want to compare it to an LSTM approach. I am however not sure how I can model the weekly and yearly pattern correctly and if I need to use the stateful LSTM or not. Basically I want to use the power of an LSTM to predict a sequence of a longer period of time and hope that the forecasts will be better than with a standard (and much faster) MLP.

Nathan George September 7, 2016 at 3:27 pm #

REPLY

These lines don't make sense to me:

```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Isn't it [samples, fetaures, timesteps] ?
When you switch to lookback = 3, you still use trainX.shape[0], 1, trainX.shape[1] as your reshape, and aren't the timesteps the lookback? I noticed the Keras model doesn't work unless you reshape in this way, which is really strange to me. Why do we have to have a matrix full of 1×1 vectors which hold another vector inside of them? Is this in case we have more features or something? Are there ever any cases where the '1' in the reshape would be any other number?

Christoph September 8, 2016 at 10:30 pm #

REPLY

I think the example given here is wrong in the sense, that each data point represents a sample with one single timestep. This doesn't make sense at all if you think about the strengths of recurrent networks and especially LSTM. How is the LSTM going to memorize if there's only one timestep for each sample?

Since we are working on a single time series, it should probably be the other way around, with one sample and n timesteps. One could also try and reduce the number of timesteps and split the time series up into samples of a week or a month depending on the data.

Not sure if this is all correct what I just said, but I get the feeling that this popular tutorial isn't a 100% correct and therefore a bit misleading for the community.

Jason Brownlee September 9, 2016 at 7:20 am #

I do provide multiple examples just so you can compare the methods, both le

For this specific problem, the timestep approach is probably the best way to model it, template for your own problem, you have a few options to choose from.

Oswaldo September 9, 2016 at 10:35 am #

REPLY

If I want to predict t+1 in a test set with t, the prediction doesn't make sense. If I shift the prediction, it make sense but I end up predicting t with t, using a next-step model that learnt the sequence. Nice exercise to get used with the implementation, but what's the point in real life? I really want a t+1 prediction that match t+1 (not t) in the test set, What I'm missing?

Easwar September 12, 2016 at 1:21 am #

REPLY

Hi Jason,

This is an excellent tutuorial. I have a question. You have one LSTM (hidden) layer with 4 neurons. What if I construct a LSTM layer with only 1 neuron. Why should I have 4 neurons ? I suppose this is different from having two or layers (depth) ? Depth in my understanding is if you have more layers of LSTM.

If you have 4 LSTM neurons in first layers, does input get fed to all the 4 neurons in a fully connected fashion ? Can you explain that ?

Best Regards,
Easwar

Jason Brownlee September 12, 2016 at 8:33 am #

REPLY

Great question Easwar.

More neurons means more representational capacity at that layer. More layers means more opportunity to abstract hierarchical features.

The number of neurons (LSTM call them blocks) and layers to use is a matter of trial and error (tuning).

Sam February 19, 2017 at 2:41 pm #

REPLY

If we reduce the number of neurons BELOW the number of features fed into an RNN, then does the model simply use as many features as the neuron number allows ?
For example, if I have 10 features but define a model with only 5 neurons in the initial layer(s), would the model only use the FIRST 5 features ?

Thanks,
Sam

Jason Brownlee February 20, 2017 at 9:28 am #

REPLY

No, I expect it will cause an error. Try it and see.

Sam February 23, 2017 at 12:05 pm #

NO, surprisingly it works very well and gives great prediction results.
Is there a requirement that each feature have a neuron ?

Max Clayer September 13, 2016 at 4:38 am #

REPLY

Hi I have found when running your raw example on the data, the training data seems
same as your graph in your first example, why could this be?

Stijn September 19, 2016 at 1:29 am #

REPLY

Hi Jason,

Nice blog post.

I noticed however, that when you do not scale the input data and switch the activation of the L
comparable to the feedforward models in your other blog post (http://machinelearningmastery
in-python-with-keras/). The performance becomes: Train Score: 23.07 RMSE, Test Score: 48.

Moreover, when you run the feedforward models in the other blog post with scaling of the input data their performance degrades.

Any idea why scaling the dataset seems to worsen the performance performance?

Cheers,

Stijn

Jason Brownlee September 19, 2016 at 7:44 am #

REPLY

Interesting finding Stijn, thanks for reporting it.

I need to experiment more myself. Not scaling and using ReLu would go hand in hand.

V September 23, 2016 at 2:48 pm #

REPLY

Hi Jason – actually I was able to verify Stjin's results (could you please delete my inquiry to him).

But I am curious about this:

Train Score: 22.51 RMSE
Test Score: 49.75 RMSE

The error is almost twice as large on the out of sample data, what does that mean about the quality of our model?

V September 23, 2016 at 2:35 pm #

REPLY

Hi Stijn – I wasn't able to replicate your results, could you please post your code. Thanks!

Max Clayer October 4, 2016 at 3:56 am #

REPLY

http://stats.stackexchange.com/questions/59630/test-accuracy-higher-than-training-how-to-interpret

Jason Brownlee October 4, 2016 at 7:27 am #

REPLY

Great link, thanks Max.

Jakob Aungiers September 22, 2016 at 10:52 pm #

REPLY

Hey Jason,

As far as I can tell (and you'll have to excuse me if I'm being naive) this isn't predicting the next timestep at all? Merely doing a good job at mimicking the previous timestep?

For example the with the first example, if we take the first timestep of trainX (trainX[0]) the prediction from the model doesn't seem to be trying to predict what t+1 (trainX[1]) is, but merely mimics what it thinks fits the model at that particular timestep (trainX[0]) i.e. tries to copy the current timestep. Same for trainX[1], the prediction is not a prediction of trainX[2] but a guess at trainX[1]… Hence which the graphs in the post (which as you mentioned above you need to update) look like they're forwardlooking, but running the code actually produces graphs which have the predictions shifted t+look_back.

How would you make this a forward looking graph? Hence also, I tried to predict multiple futur t prediction with testX[0] and then feeding the next predictions with the prior predictions but the downwards curve. Not predicting the next timesteps at all.

Am I being naive to the purpose of this blog post here?

All the best, love your work,
Jakob

Jeremy Irvin September 24, 2016 at 11:08 am #

REPLY

Hi Jakob,

I believe you are correct.

I have tried these methods on many different time series and the same mimicking behavior occurs – the training loss is somehow minimized by outputting the previous timestep.

A similar mimicking behavior occurs when predicting multiple time steps ahead as well (for example, if predicting two steps ahead, the model learns to output the previous two timesteps).

There is a small discussion on this issue found here – https://github.com/fchollet/keras/issues/2856 – but besides that I haven't discovered any ways to combat this (or if there is some underlying problem with Keras, my code, etc.).

I am in the process of writing a blog to uncover this phenomenon in more detail. Will follow up when I am done.

Any other advancements or suggestions would be greatly appreciated!

Thanks,
Jeremy

Max Clayer October 4, 2016 at 4:27 am #

REPLY

Are you simply using t-1 to predict t+1 in the time window, if so I don't think there is enough data being fed into the neural network to learn effectively. with a bigger time window I notice that the model does start to fit better.

Dominic September 23, 2016 at 2:55 am #

REPLY

Hi, Jason
Thank you for your post.
I am still confused about LSTM for regression with window method and time steps.
Could you explain more about this point. Could you use some figures to show the difference between them?
Many thanks!

Dominic September 23, 2016 at 6:36 am #

REPLY

As my understanding, the LSTM for regression with window method is the same as a standard MLP method, which has 3 input features and one output as the example. Is this correct? What's the difference?

Wolfgang September 24, 2016 at 4:20 pm #

REPLY

Thank you very much for the detailed article!

Does anybody have a scientific source for the time window? I can't seem to find one.

Jason Brownlee September 25, 2016 at 8:01 am #

REPLY

Great question.

I don't know of any, but there must be something written up about it back in the early days. I also expect it's in old neural net texts.

Brian September 25, 2016 at 7:14 am #

REPLY

Have you experimented with having predictors (multivariate time series) versus a uni

Jason Brownlee September 25, 2016 at 8:05 am #

Yes, you can have multiple input features for multiple regression.

Brian September 26, 2016 at 12:37 am #

REPLY

Any chance you will add this type of example?

Jason Brownlee September 26, 2016 at 6:59 am #

REPLY

I will in coming weeks.

Jacques Rand September 27, 2016 at 11:40 pm #

Me too will be interested in using multivariate(numerical) data !
Been trying for a few days , but the “reshaping/shaping/data-format-blackmagic” always breaks
Purely cause I don’t yet understand it !
Otherwise great example !

Jason Brownlee September 28, 2016 at 7:41 am #

Understood, I’ll prepare tutorials.

Richard Ely October 3, 2016 at 4:31 pm #

Sir, Awesome work!!!

I am very interested in cross-sectional time series estimation… How can that be done?

I am starting your Python track, but will eventually target data with say 50 explanatory variables, with near infinite length of time series observations available on each one. Since the explanatory variables are not independent normal OLS is useless and wish to learn your methods.

I would be most interested in your approach to deriving an optimal sampling temporal window and estimation procedure.

Jason Brownlee October 4, 2016 at 7:20 am #

Sorry Richard, I don’t know about cross-sectional time series estimation.

For good window sizes, I recommend designing a study and resolving the question empirically.

Zhang Wenjie December 25, 2016 at 8:04 am #

Hi Jason

As you mentioned before, you will prepare the tutorial for multiple input features for multiple regression. Could you provide us the link to that tutorial?

Jason Brownlee December 26, 2016 at 7:44 am #

It will be part of a new book I am writing. I will put a version of it on th

Multiple regression is straight forward with LSTMs, remember input is defined
inputs are features.

For multiple output multi-step regression you can recurse the LSTM or chang

Zach May 2, 2017 at 2:15 am #

Did you ever create a tutorial for multivariate LSTM? I can’t seem to

Bob September 27, 2016 at 12:22 pm #

REPLY

I get this error when I run your script with the Theano backend:

ValueError: ('The following error happened while compiling the node', forall_inplace,cpu,scan_fn}(Elemwise{maximum,no_inplace}.0, Subtensor{int64:int64:int8}.0, IncSubtensor{InplaceSet;:int64:}.0, IncSubtensor{InplaceSet;:int64:}.0, Elemwise{Maximum}[(0, 0)].0, lstm_1_U_o, lstm_1_U_f, lstm_1_U_i, lstm_1_U_c), '\n', 'numpy.dtype has the wrong size, try recompiling')

Any idea what might be happening?

Jason Brownlee September 28, 2016 at 7:35 am #

REPLY

Hi Bob, it might be a problem with your backend. Try reinstalling Theano or Tensorflow – whichever you are using. Try switching up the one you are using.

Philip September 28, 2016 at 9:32 pm #

REPLY

Excellent article, really insightful. Do you have an article which expands on this to forecast data? An approach that would mimic that of say arima.predict in statsmodels? So ideally we train/fit the model on historical data, but then attempt to predict future values?

Jason Brownlee September 29, 2016 at 8:36 am #

REPLY

Thanks Philip, I plan to prepare many more time series examples.

Waldemar October 19, 2016 at 12:39 pm #

REPLY

Hi,Mr.Brownlee!
I don't see future values on your plots, as I understand your model don't predict the Future, only describe History. Can you give advice, how can I do this? And how I can print predictive values?
Thanks a lot!

Tucker Siegel September 29, 2016 at 10:51 am #

REPLY

Great article Jason.
I was just wondering if there was any way I could input more than 1 feature , and have 1 output, which is what I am trying to predict? I am trying to build a stock market predictor. And yes I know, it is nearly impossible to predict the stock market, but I am just testing this, so lets say we live in a perfect world and it can be predicted. How would I do this?

Jason Brownlee September 30, 2016 at 7:44 am #

REPLY

Hi Tucker,

You can have multiple features as inputs.

The input structure for LSTMS is [samples, time steps, features], as explained above. In fact, there are examples of what you are looking for above (see the section on the window method).

Just specify the features (e.g. different indicators) in the third dimension. You may have 1
dimension).

I hope that helps.

Tucker Siegel September 30, 2016 at 1:33 pm #

I did what you said, but now it wants to output 2 sequences out of the activat
1. Basically what I am doing is trying to use open and close stock data, and use it to p
sequences and have an output of 1. I hope I explained that right. What should I do?

Joe October 1, 2016 at 11:24 pm #

REPLY

Jason, you mentioned that LSTMs input shape must be [samples, time stamps, features]. What if my time series is sampled (t, x), i.e. each sample has its own time stamp, and the time stamps are NOT evenly spaced. Do I have to generate another time series in which all samples are evenly spaced? Is there any way to handle the original time series?

Jason Brownlee October 2, 2016 at 8:19 am #

REPLY

Really good question Joe. Thanks. I have not thought about this.

My instinct would be to pad the time series, fill in the spaces with zeros and make the time series steps equidistant. At least, I would try that against not doing it and evaluate the effect on performance.

Pho King November 5, 2016 at 3:38 pm #

REPLY

Take samples in blocks via Sklearn.model_selection.TimeSeriesSplit

Rio October 6, 2016 at 4:07 am #

REPLY

What an excellent article!
Recently I used LSTM to predict stock market index where the data is fluctuating and has no seasonal pattern like the air passanger data. I was just wondering about how does LSTM (or every gate) decide when to forget or keep a certain value of this type of series data. Any explanation about this? Thank you.

Jason Brownlee October 6, 2016 at 9:40 am #

REPLY

Great question Rio. I would love to work through how the gates compute/output on a specific problem.

I think this would be a new blog post in the future.

Rio October 7, 2016 at 1:15 pm #

REPLY

I'm looking forward to that post, Jason. Thank you

SalemAmeen October 6, 2016 at 10:27 pm #

REPLY

I used "LSTM For Regression Using the Window Method" with the following parameters
look_back = 20
LSTM(20,input_dim=look_back)

I got the following results

Train_Score: 113.67 RMSE
Test_Score: 122.88 RMSE

I computed R-squared and I got 0.93466300136

In addition I tried changing the hyperparameters in the other two models but R-squared was le

Cas May 9, 2017 at 5:53 pm #

Dear SalemAmeen,

I am currently working with lstm recurrent neural networks and I am curious how you calcu
about the R-squared calculation with me? Thank you very much!

Kind regards
Cas

Jason Brownlee May 10, 2017 at 8:45 am #

REPLY

If you collect your predictions, you can calculate r^2 using the sklearn library:
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

You can also use a custom metric to report r^2 each epoch:
https://keras.io/metrics/

Randy October 7, 2016 at 11:03 pm #

REPLY

Hi, Jason
First of all, this is really a fantastic post and thank you so much!
I've got confused on the "model.predict(x,batch_size)".
I can't figure what it means "predict in a batched way" on the keras official website.
My situation is like:
batch_size=10
I have a test sample [x_1] \in R^{2}, and I put it into the function,
[x_2] = model.predict([x_1],batch_size=batch_size)
(Let's skip the numpy form issue)
Then, subsequently, I put [x_2] into it, similarly, and I get [x_3] = model.predict([x_2],batch_size=batch_size), and so on, till x_10.
I don't know if the function "predict" treats [x_1],[x_2],…[x_3] as in a batch ?
I guess it does.(although I didn't put them into the function at one time)

Otherwise, I've tried another way to compute [x_2],…[x_10] and I got the same as above.
Another strategy is like:
[x_2] = model.predict([x_1],batch_size=batch_size)
[x_3] = model.predict([[x_1],[x_2],batch_size=batch_size)
[x_4] = model.predict([[x_1],[x_2],[x_3]],batch_size=batch_size)
…

What's the difference between the two ways?
Thanks!

Randy October 7, 2016 at 11:55 pm #

REPLY

btw, I am also confused at "batch".
If batch_size=1, does that mean there's no relation between samples? I mean the state s_t won't be sent to affect the next step s_(t+1).
So, why we need RNN?

Jason Brownlee October 8, 2016 at 10:40 am #

REPLY

It's a good point.

We need RNN because the state they can maintain gives results better than methods that do not maintain state.

As for batch_size=1 during calls to model.predict() I have not tested whether indeed state is lost as in training, but I expect it may be. I expect one would need batch_size=n_samples and replay data each time a prediction is needed.

I must experiment with this and get back to you.

Jason Brownlee October 8, 2016 at 10:37 am #

I'm not sure I understand the "two ways" you're comparing, sorry.

The batch size is the number of records that the network will process at once – as in load

In training, this is the constraint on data before weight update. In test, it is data before con

Does that help at all?

Randy October 13, 2016 at 3:11 pm #

REPLY

yes, it does!

I appreciate it !!

dubi dubi October 8, 2016 at 3:09 pm #

REPLY

This is a great post! Thanks for the guidance. I'm wondering about performance. I've setup my network very similarly to yours, just have a larger data set (about 2500 samples, each with 218 features). Up to about 20 epochs runs in a reasonable amount of time, but anything over that seems to take forever.

I've set-up random forests and MLPs, and nothing has run so slowly. I can see all CPUs are being used, so am wondering whether Keras and/or LSTM has performance issues with larger data sets.

Jason Brownlee October 9, 2016 at 6:48 am #

REPLY

Great question.

LSTMs do use more resources than classical networks because of all the internal gates. No significant, but you will notice it at scale.

I have not performed any specific studies on this, sorry.

Jason Wills October 12, 2016 at 4:02 pm #

REPLY

Hi Jason,

I am confusing about deep learning and machine learning in Stock Market , forex . There are a lot of models which analyses via chart using amibroker or metastock which redraw the history price and take the prediction in that model . Does it call the machine learning or deep learning ? How is it when we could do farther to make better prediction via deep learning if it's right ?

Jason Brownlee October 13, 2016 at 8:35 am #

REPLY

Hi Jason, it sounds like you are already using predictive models.

It may fair to call them machine learning. Deep learning is one group of specific techniques you may or may not be using.

There are may ways to improve results, but it is trial and error. I offer some ideas here:
http://machinelearningmastery.com/improve-deep-learning-performance/

Sorry, I don't know the specifics of stockmarket data.

Alexander October 13, 2016 at 12:46 pm #

REPLY

Hi Jason,

These models do not predict, they extrapolate current value 1 step ahead in more or less obscured way. As seen on the pictures, prediction is just shifted original data. For this data one can achieve much better RMSE 33.7 without neural net, in just one line of code:

trainScore = math.sqrt(mean_squared_error(x[:-1], x[1:]))

Jason Brownlee October 14, 2016 at 8:56 am #

REPLY

Hi Alexander, thanks.

This is good motivation for me/community to go beyond making LSTMs "just work" for tim
and even competitively on even very simple problems.

It's an exciting open challenge.

Leftriver October 18, 2016 at 12:45 pm #

REPLY

This is a nice tutorial for starters. Thank you.

However, I have some concerns about the create_dataset function. I think it just make a simpl

When look_back=1, the function is simply equivalent to: dataX = dataset[:len(dataset)-look_ba

When look_back is larger than 1, the function is wrong: after each iteration, dataX is appended by more than 1 data, but dataY is appended by just 1 data. Finally, dataX will be look_back times larger than dataY.

Is that what create_dataset supposed to do?

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

pang wenfeng October 21, 2016 at 3:23 pm #

REPLY

Thanks for your great article! I have a question that when I use the model “Stacked LSTMs With Memory Between Batches” with my own data, I found that cpu is much faster than gpu.
May data contains many files and each files’ size is about 3M. I put each file into the model to
trian one by one. I guess that the data is too small so the gpu is useless, but I can’t sure. I use thano backend and I can sure that the type of the data is float32. So I want to know what reason would make this happen, or the only reason is the data too small? Thank you very much and best wishes to you.

Rajesh October 22, 2016 at 4:00 pm #

REPLY

Hi Jason,

Excellent tutorial. I am new to time series prediction. I have a basic question. In this example(international-airline-passengers) model predicted the values on test data from 1957-01 to 1960-12 time period.

How to predict the passengers in next one year from 1961-01 – 1961-12.

How to pass input values to model, so it will predict the passengers count in each month for next one year.

Rajesh October 23, 2016 at 5:53 pm #

REPLY

Hi,

Any inputs to solve below question

How to predict the passengers in next one year from 1961-01 – 1961-12.

Regards,
Rajesh

NicoAd October 22, 2016 at 8:33 pm #

REPLY

Hi,

“The dataset is available for free from the DataMarket webpage as a CSV download with the filename “international-airline-passengers.csv“.”

Not anymore I guess.

Any other way to get the file?

Thanks,
Nicolas

Jason Brownlee October 23, 2016 at 10:11 am #

Yes, it is still available NicoAd.

Visit: https://datamarket.com/data/set/22u3/international-airline-passengers-monthly-totals

Click “Export” on the left-hand side, and choose the CSV file format.

NicoAd October 27, 2016 at 9:05 pm #

REPLY

Thanks !

Hans April 25, 2017 at 5:31 am #

REPLY

Just a new location:

https://github.com/ufal/npfl114/blob/master/labs06/international-airline-passengers.tsv

Brian October 25, 2016 at 3:51 am #

REPLY

Hi,

Great article on LSTM and keras. I was really struggling with this, until I read through your examples. Now I have a much better understanding and can use LSTM on my own data.

One thing I'd like to point out. The reuse of trainY and trainX on lines 55 & 57.
Line 55 trainY = scaler.inverse_transform([trainY])

This confused me a lot, because the model can't run fit() or predict() again, after this is done. I was struggling to understand why it could not do a second predict or fit. Until i very carefully read each line of code.

I think renaming the above variables would make the example code clearer.

Unless I am missing something….. and being a novice programmer that's very possible.

Thanks again for the great work.
Brian.

Jason Brownlee October 25, 2016 at 8:32 am #

REPLY

Thanks Brian, I'm glad the examples were useful to get you started.

Great point about remaining variables.

Joaco October 28, 2016 at 12:09 pm #

REPLY

Hi, Jason, thank you for the example.
I have used the method on my own data. The data is about the prediction of the average temperature per month. I want to predict more than one month. But I can only predict one month now. Because the inputs are X1 X2 X3, the result is only y. I want to kown how to modify the code to use ,like, X1 X2 X3 X4 X5 X6 to predict Y1 Y2 Y3.
I don't know if I have made it clear. I hope you will help me.
Thank you very much.

Jason Brownlee October 29, 2016 at 7:33 am #

REPLY

Hi Joaco, that is a sequence to sequence problem. I will prepare an example soon.

Je December 27, 2016 at 10:34 am #

Hi Jason, Joaco,
I am also looking forward to a similar example ☐

Je

Donato June 1, 2017 at 9:09 pm #

Sorry do you have prepare this example? Thanks!

Jason Brownlee June 2, 2017 at 12:58 pm #

REPLY

Yes, see here:
http://machinelearningmastery.com/learn-add-numbers-seq2seq-recurrent-neural-networks/

Nida October 28, 2016 at 1:21 pm #

REPLY

Nice post Jason!
I read that GRU has a less complex architecture than the LSTM has, but many people still use LSTM for forecasting. I'd like to ask, what are the advantages LSTM compared to GRU? Thank you

Jason Brownlee October 29, 2016 at 7:35 am #

REPLY

Hi Nida, I would defer to model skill in most circumstances, rather than concerns of computational complexity – unless that is a requirement of your project.

Agreed, we do want the simplest and best performing model possible, so perhaps evaluate GRUs and then see of LSTMs can out perform them on your problem.

Tim October 28, 2016 at 9:49 pm #

REPLY

I'm a total newbie to Keras and LSTM and most things NN, but if you'll excuse that, I'd like to run this idea past you just to see if I'm talking the same language let alone on the same page… :

I'm interested in time-series prediction, mostly stocks / commodities etc, and have encountered the same problem as others in these comments, namely, how is it prediction if it's mostly constrained to within the time-span for which we already have data?

With most ML algorithms I could train the model and implement a shuffle, ie get the previous day's prediction for today and append it in the input-variable column, get another prediction, … repeat. The worst that would happen is a little fudge around the last day in the learning dataset.
That seems rather laborious if we want to predict how expensive gold is going to be in 6 months' time.
(Doubly so, since in other worlds (R + RSNNS + elman or jordan), the prediction is bound-up with training so a prediction would involve rebuilding the entire NN for every day's result, but we digress.)

I saw somewhere Keras has a notion of "masking", assigning a dummy value that tells the training the values are missing. Would it be possible to use this with LSTM, just append a bunch of 180 mask zeroes, let it train itself on this and then use the testing phase to impute the last values, thereby filling in the blanks for the next 6 months?

It would also be possible to run an ensemble of these models and draw a pretty graph similar to arima.predict with varying degrees of confidence as to what might happen.

Waffle ends.

Jason Brownlee October 29, 2016 at 7:45 am #

REPLY

Interesting idea.

My thoughts go more towards updating the model. A great thing about neural nets is that they are updatable. This means that you can prepare just some additional training data for today/this week and update the weights with the new knowledge, rather than training them from scratch.

Again, the devil is in the detail and often updating may require careful tuning and perhaps balance of old data to avoid overfitting.

Lazaros October 30, 2016 at 6:39 pm #

REPLY

Dear Jason,

I am trying to implement your code in order to make forecasting on a time-series that i am rec
length of my dataset is continuously increasing. Is there any way to read the last N rows from
below in order to succeed it.

```
def create_dataset(dataset, look_back=1):
dataX, dataY = [], []
for i in range(len(dataset)-look_back-1):
a = dataset[i:(i+look_back), 0]
dataX.append(a)
```

```
dataY.append(dataset[i + look_back, 0])
return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = pandas.read_csv('timeseries.csv', usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
```

Jason Brownlee October 31, 2016 at 5:29 am #

REPLY

This post might help with loading your CSV into memory:
http://machinelearningmastery.com/load-machine-learning-data-python/

If you load your data with Pandas, you can use DataFrame.tail() to select the last n records of your dataset, for example:

```
# Load CSV using Pandas
from pandas import read_csv
filename = 'pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data = data.tail(10)
print(data.shape)
```

Alisson Pereira November 3, 2016 at 9:12 am #

REPLY

Hello, I would like to use that your model. But the problem I am using sliding window size greater than one. Type {[x-2], [x-1], [x]} ==> [x + 1]. But I found several problems in training. For example, when I turn your trainX in {[x-2], [x-1], [x]} and trainY in [x + 1], the keras tells me that the input and the target must have same size. Can you help me with this?

Jason Brownlee November 4, 2016 at 9:02 am #

REPLY

Hi Alisson, I think the error suggests that input and target do not have the same number of rows.

Check your prepared data, maybe even save it to file and look in a text editor or excel.

Alisson Pereira November 9, 2016 at 3:55 am #

REPLY

Thaks, Jason. I was able to solve my problem. But see, the use of the ReLu function in the memory cell and the sigmoid function on the output showed strange behavior. You have some experience with this setting.
Congratulations on the work, this page has helped me a lot.

Soren November 4, 2016 at 2:20 am #

REPLY

Hi Jason,

Thanks for your great content.

As you did i upgraded to Keras 1.1.0 and scikit-learn v0.18. however i run Theano v.0.9.0dev3 as im on Windows 10. Also im on Anaconda 3.5. (installed from this article: http://ankivil.com/installing-keras-theano-and-dependencies-on-windows-10/)

Your examples run fine on my setup – but i seem to be getting slightly different results.

For eamples in your first example: # LSTM for international airline passengers problem with w

Train Score: 22.79 RMSE
Test Score: 48.80 RMSE

Should i be getting exact the same results as in your tutorial? If yes, any idea what i should be

Best regards
Soren

Jason Brownlee November 4, 2016 at 9:12 am #

REPLY

Great work Soren!

Don't worry about small differences. It is hard to get 100% reproducible results with Keras/Theano/TensorFlow at the moment. I hope the community can work something out soon.

pemfir November 4, 2016 at 2:00 pm #

REPLY

great post ! thank you so much. I was wondering how can be adapt the code to make multiple-step-ahead prediction. One of the commenters suggested defining the out-put like [x(t+1),x(t+2),x(t+3),…x(t+n)] , but is there a way to make prediction recursively ? More specifically, to build an LSTM with only one output. We first predict x(t+1), then use the predicted x(t+1) as the input for the next time step to predict x(t+2) and continue doing so 'n' times.

Bill November 5, 2016 at 12:11 pm #

REPLY

Hi Jason,

I am wondering how to apply LSTM to real time data. The first change I can see is the data normalisation. Concretely, a new sample could be well out of min max among previous observations. How would you go about this problem?

Thanks.

Noque November 6, 2016 at 2:58 am #

REPLY

Could it be that in:

```
# calculate root mean squared error
math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
```

you mean :

```
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[:], trainPredict[:,0]))
```

Kit December 21, 2016 at 9:30 am #

REPLY

trainY is an array of (one) array. Compare:

```
print len(trainYi[0]) # 720
print len(trainYi[:]) # 1

print len(trainPredict[:,0]) # 720
```

So the original code is the right one.

sherlockatszx November 6, 2016 at 6:05 am #

REPLY

Hi jason, i got a question,that have stacked me for many days,how can i add hidden layer into the LSTM model(By using model.add(LSTM()).what i tried : say ,in your first code example, I assume 'model.add(LSTM(4, input_dim=look_back))' this line was to create a hidden layer in the LSTM model. So i thought:oh , 1 hidden layer is so easy , why don't add one hidden layer into it .So i try to add one layer. After the code:'model.add(LSTM(4, input_dim=look_back))' , i try many ways to insert one hidden layer , such as : just copy model.add(LSTM(4,input_dim=look_back)) and insert after it . I try many ways ,but it always got the error that got the wrong input _ dimension. So can you show me how to add one hidden layer in example 1st . Or , i don't got the LSTM mod

Jason Brownlee November 7, 2016 at 7:07 am #

See the section titled "Stacked LSTMs With Memory Between Batches".

It gives an example of multiple hidden layers in an LSTM network.

sherlockatszx November 7, 2016 at 1:26 pm #

REPLY

Thanks . I got that.

However, I got another question: compared to the other article you published 'time series prediction with deep learning '(http://machinelearningmastery.com/time-series-prediction-with-deep-learning-in-python-with-keras/?utm_source=tuicool&utm_medium=referral) , It seems that 'LSTM' model doesn't predict as well as the simple neurons. Does that mean LSTM may not a good choice for some specific time series structure?

Jason Brownlee November 8, 2016 at 9:49 am #

REPLY

I would not agree, these are just demonstration projects and were not optimized for top performance.

These examples show how LSTMs could be used for time series projects (and how to use MLPs for time series projects), but not optimally tuned for the problem.

Noque November 9, 2016 at 1:28 am #

REPLY

Hi, great post! Thanks

How could I set the input if I have several observations (time series) with same length of the same feature and I want to predict t+1? Would I concatenate them all? In that case the last sample of one observation would predict the first one of the next.. Or should I explicitly assign the length of each time series to the batch_size?

Mauro November 10, 2016 at 9:44 am #

REPLY

Hi, you're predicting one day after your last entry, if i want to predict a day five days after what should i do?

Jason Brownlee November 11, 2016 at 9:56 am #

REPLY

Hi Mauro, that would be a sequence to sequence prediction.

Sorry, I don't have an example just yet.

Ron November 16, 2016 at 7:01 am #

REPLY

Hi

This is a great example. I am quite new in deep learning and keras. But this website has been very helpful. I want to learn more.

Like many commenters, I am also requesting to find out: how to predict future time periods. Is that possible? How can I achieve this using the example above? If there are multiple series or Ys and there are categorical predictors, how can I accomodate that?

Please help, and am very keen to learn this via other channels in this website if required. Please let me know.

Many thanks

Jason Brownlee November 16, 2016 at 9:34 am #

REPLY

The example does indeed predict future values.

You can adapt the example and call model.predict to make a prediction for new data, rather than just evaluate the performance of predictions on new data as in the example.

Nico AD November 17, 2016 at 3:21 am #

REPLY

Hi,

I tried to predict future values, but have trouble finding the right way to do it

I work with the window method so my current data à t-3 t-2 t-1 t looks like this

[100,110,108]

If I try

data = [[100,110,108]]

model.predict(data)

I get the following error :

Attribute error : ‘list’ object has no attribute “shape”

I guess the format is not correct, and I need sort of reshape.

but for me the line

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1])) is not clear.

and if I try to apply the same transform on my new data I get this error :

IndexError : tuple index out of range

could you provide an exemple dealing with new data ?

Thanks !

Nico AD November 17, 2016 at 3:54 am #

REPLY

I think I almost here but still have a last error (my window side is 8)

```
todayData = numpy.array([[1086.22,827.62,702.94,779.5,711.8399999999999,1181.25,1908.69,2006.39]])
todayData = todayData.astype('float32')
todayData = scaler.fit_transform(todayData)
print "todayData scaled " + str(todayData)

todayData = scaler.inverse_transform(todayData)
print "todayData inversed " + str(todayData)
todayData = numpy.reshape(todayData, (todayData.shape[0], 1, todayData.shape[1]))

predictTomorrow = model.predict(todayData)
predictTomorrow = scaler.inverse_transform([predictTomorrow])
print "prediction" + str(predictTomorrow)
```

the inverse_transform line on predictTomorrow generate the following error

ValueError : Found array with dim 3 . Estimator expected <= 2

again a reshape issue ☐

Jason Brownlee November 17, 2016 at 9:55 am #

REPLY

I am working on a new example Nico, it may be a new blog post.

Nico AD November 22, 2016 at 8:08 pm #

thanks Jason. I tried various things with no luck. for me some part of the tutorial (like the reshape part / scaling) are pure magic ☐ trying to get some help from the keras community on gitter ☐

Nico AD November 23, 2016 at 2:05 am #

finally got it , I need to reshape in (1,1,8) (where 8 is the look_back size)

Jason Brownlee November 23, 2016 at 9:00 am #

Well done Nico.

Ron November 16, 2016 at 7:15 am #

Hi Jason

Which book gives complete examples/codes with time series keras? I want to predict future ti
variables? Is that achievable?

Please let me know if you have any resources / book that I can purchase.

Many thanks

Jason Brownlee November 16, 2016 at 9:35 am #

REPLY

Deep Learning With Python:
https://machinelearningmastery.com/deep-learning-with-python/

Sarah November 19, 2016 at 7:54 am #

REPLY

Hi Jason

Thank you for your great tutorial,
I have a question about number of features. How could I have input with 5 variables?

Thank you in advance

Sarah

Jason Brownlee November 19, 2016 at 8:52 am #

REPLY

Hi Sarah, LSTMs take input in the form [samples, timesteps, features], e.g. [n, 1, 5].

You can prepare your data in this way, then set the input configuration of your network appropriately, e.g. input_dim=5.

Adam November 19, 2016 at 1:24 pm #

REPLY

Nice tutorial, thanks.
I think the line
for i in range(len(dataset)-look_back-1):
should be
for i in range(len(dataset)-(look_back-1)):

Adam November 19, 2016 at 1:52 pm #

REPLY

Actually, I think its
for i in range(len(dataset)-look_back):
and
testPredictPlot[train_size+(look_back-1):len(dataset)-1, :] = testPredict

Jason Brownlee November 22, 2016 at 6:46 am #

REPLY

Thanks Adam, I'll take a look.

Ben November 23, 2016 at 8:55 am #

REPLY

Hi Adam, nice blog ! I only have a small suggestion for shifting data: use the shift() m

Jason Brownlee November 23, 2016 at 9:07 am #

Great suggestion Ben. I have been using this myself recent to create a lagged da

Weixian November 23, 2016 at 3:08 pm #

REPLY

Hi Jason,

As i am new to RNN, i would like to ask about the difference in stateful:

for i in range(100):
model.fit(trainX, trainY, nb_epoch=1, batch_size=batch_size, verbose=2, shuffle=False)
model.reset_states()

and the stateless:

model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=2)

Do the range of 100(nb_epoch=1) the same as nb_epoch=100?
What is the difference between these 2?

Jason Brownlee November 24, 2016 at 10:37 am #

REPLY

Good question Weixian.

In the stateful case, we are running each epoch manually and resetting the state of the network at the end of each epoch.

In the stateless case, we let the Keras infrastructure run the loop over epochs for us and we're not concerned with resetting network state.

I hope that is clear.

Weixian November 24, 2016 at 4:21 pm #

REPLY

Hi Jason,

Thanks for the reply.

In this case for the stateful:
if i reset the network, would the next input from the last trained epoch?

For the stateless:
Does it loop from the epochs that was previously trained?

How does the 2 affect the data trained or tested?

Jason Brownlee November 25, 2016 at 9:32 am #

REPLY

Sorry, I don't understand your questions. Perhaps you could provide more context?

Weixian November 28, 2016 at 7:36 pm #

Hi Jason,

stateful:
I mean like the training results of the last epoch [Y1] output for example A
Would the [X2] input of the network be A from the last epoch?

Stateless:
How would the top situation be different from the epoch=2?

Jason Brownlee November 29, 2016 at 8:49 am #

Yes, you need to have the same inputs in both cases. The difference
stateful.

Quinn November 23, 2016 at 5:08 pm #

REPLY

Hi Jason
Thank you for your LSTM tutorial.

But i found that an error always occurred, when i ran the first code in 'model.add(LSTM(4, inpu

The error is : TypeError: super() argument 1 must be type, not None

So, why?

Thanks

Jason Brownlee November 24, 2016 at 10:38 am #

REPLY

Check your white space Quinn, it's possible to let extra white space sneak in when doing the copy-paste.

Vedhas November 23, 2016 at 9:57 pm #

REPLY

Many thanks for this article. I am trying to wrap my head around

trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))

(from LSTM for Regression with Time Steps section), since this is exactly what I need.

Let's say I have 4 videos (prefix v) , of different lengths (say, 2,3,1 sec) (prefix t) , and for every 1 sec, I get a feature vector of length 3 (prefix f).

So, as I understand my trainX would be like this, right? –>

trainX=np.array (
[
[[v1_t1_f1, v1_t1_f2, v1_t1_f3],
[v1_t2_f1, v1_t2_f2, v1_t2_f3]],

[[v2_t1_f1, v2_t1_f2, v2_t1_f3],
[v2_t2_f1, v2_t2_f2, v2_t2_f3],
[v2_t3_f1, v2_t3_f2, v2_t3_f3],],

[[v3_t1_f1, v3_t1_f2, v3_t1_f3]])
]
(=[[v1], [v2], [v3]], and v is Nt x Nf python list?)

If I have v1, v2,v3, how do I start with an **empty** xTrain and update them **recursively** to xTrain, so that xTrain can be used by Keras?

I have tried np.append, np.insert, np.stack methods, but no success as yet, I always get some error. Kindly help!!!

Vedhas November 23, 2016 at 10:18 pm #

REPLY

If I make my 'v1′,'v2′,v3′ ..'v19' as np arrays, and trainX as a list =[v1, v2, v3…v19] using trainX.append(vn) –> and eventually outside of for loop: trainX=np.array(trainX), I get following error.

File "/usr/local/lib/python2.7/dist-packages/keras/engine/training.py", line 100, in standardize_input_data
str(array.shape))
Exception: Error when checking model input: expected lstm_input_1 to have 3 dimensions, but got array with shape (19, 1)

Which makes sense since, Keras must be expecting input to have 3 dimensions = (sample,tstep, features).

But how do I fix this???

Your comment is awaiting moderation.

Vedhas November 23, 2016 at 10:18 pm #

REPLY

If I make my 'v1′,'v2′,v3′ ..'v19' as np arrays, and trainX as a list =[v1, v2, v3…v19] using trainX.append(vn) –> and eventually outside of for loop: trainX=np.array(trainX), I get following error.

File "/usr/local/lib/python2.7/dist-packages/keras/engine/training.py", line 100, in standardize_
str(array.shape))
Exception: Error when checking model input: expected lstm_input_1 to have 3 dimensions, bu

Which makes sense since, Keras must be expecting input to have 3 dimensions = (sample,tst

But how do I fix this???

Ilias November 25, 2016 at 12:08 pm #

REPLY

Guestion about the stateful data representation.
If I understood correctly prepare_data makes repeats the previous look_back sequences.

For example the original data
1
2
3
4
5
6

will become
1 2 3 -> 4
2 3 4 ->5
3 4 5 ->6

Then when you reshape for the stateful LSTM don't you feed these sequences like this ?
batch 1 sequences [1, 2, 3] -> predict 4
batch 2 sequences [2, 3, 4] -> predict 5
batch 3 sequences [3, 4, 5] -> predict 6

In the stateful RNN shouldn't it be two batches only that continue one from the next:
batch 1 sequences [1, 2, 3] -> predict 4
batch 2 sequences [3, 4, 5] -> predict 6

Or alternatively you can have it to return the full state and predict all of them
batch 1 sequences [1, 2, 3] -> predict [2, 3, 4]
batch 2 sequences [3, 4, 5] -> predict [4, 5, 6]

Thanks,
Ilias

Ilias November 25, 2016 at 12:09 pm #

REPLY

Sorry i mean for the stateful RNN
batch 1 sequences [1, 2, 3] -> predict 4
batch 2 sequences [4, 5, 6] -> predict 7 (not 3,4,5)

Luca November 25, 2016 at 8:27 pm #

REPLY

Hi!
First of all, thanks for the tutorial. I'm trying to predict data that are very similar to the example ones. I was playing with the code you gave, but then something very strange happened: if I fit a model using the flight data and i use those hyper parameters to predict white noise I receive a very accurate results. Example:

#Data Generation:

dataset = numpy.random.randint(500, size=(200,1))
dataset = dataset.astype('float32')

#Data Prediction:
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

prediction in red:

https://s17.postimg.org/oavua7uq7/download.png

How could that be possible? White noise should be not predictable, what am I doing wrong?

Luca November 25, 2016 at 8:43 pm #

Sorry, i was doing something very stupid, just ignore my latest post.

Cheers

Luca November 25, 2016 at 10:09 pm #

Ok, sorry again for the last correction, the result was obtained using:

plt.plot(dataset,"b",model.predict(dataset[:,:,numpy.newaxis]),"r")

so I'am actually predicting white noise, how could that be possible?

Jason Brownlee November 26, 2016 at 10:38 am #

REPLY

Hi Luca, glad you're making progress.

If results are too good to be true, they usually are. There will be a bug somewhere.

Vedhas November 28, 2016 at 10:03 pm #

Kindly reply to my question above as well, please?

How do I shape trainX for 4 videos (v1,..v4) , of different lengths (2,3,1 sec) and for every 1 sec, I get a feature vector [f1 f2 f3] ?

Jason Brownlee November 29, 2016 at 8:50 am #

Sorry, I don't have examples of working with video data. Hopefully soon.

Vedhas November 30, 2016 at 4:18 am #

oh, it is not about videos.. Question is about 'instances/samples' in general…
I am saying,
Instance1 through instance4 correspond to 2,3,1,5 feature vectors in time respectively, each of dimension 3. How do I shape these to train LSTM?

That is the whole idea behind the section "LSTM for Regression with Time Steps" above, right?

Features of instance1 should not be considered when training LSTM on instance2! Just as your paragraph says:

"Some sequence problems may have a varied number of time steps per sample. For example, you may have measurements of a physical machine leading up to a point of failure or a point of surge. Each incident would be a sample the observations that lead up to the event would be the time steps, and the variables observed would be the features."

I don't need *examples of working with video data.* Kindly advise only on how to shape trainX I mentioned above.

Shu December 20, 2016 at 3:51 am #

REPLY

look careful, isn't there +1 shift in your white noise prediction?)))
same as in charts in tutorial?
best prediction for weather tomorrow is: it'll be exact the same as today. see?

Prakash November 27, 2016 at 1:24 pm #

REPLY

I see many factors for your handling this time series prediction:

-Number of LSTM blocks
-Lookback number
-Epochs
-Activation
-Optimizer

Can you show the order of importance for these in creating a prediction model? Also, you hav

Jason Brownlee November 28, 2016 at 8:41 am #

REPLY

Great question Prakash. I would say framing of the problem and network topology as the biggest levers.

I have more info on improving performance in deep learning here:
http://machinelearningmastery.com/improve-deep-learning-performance/

I chose 4 neurons (memory modules) with a little trial and error.

C November 30, 2016 at 12:51 am #

REPLY

Hi Jason,
When I try your “Stacked LSTMs with Memory Between Batches” example as it is, I found the following error. I wonder if you could help to explain what went wrong and how to rectify it please?
Thank you.

ValueError Traceback (most recent call last)
in ()
41 model = Sequential()
42 model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
—> 43 model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
44 model.add(Dense(1))
45 model.compile(loss=’mean_squared_error’, optimizer=’adam’)

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/keras/models.py in add(self, layer)
322 output_shapes=[self.outputs[0]._keras_shape])
323 else:
–> 324 output_tensor = layer(self.outputs[0])
325 if type(output_tensor) is list:
326 raise Exception(‘All layers in a Sequential model ‘

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/keras/engine/topology.py in __call__(self, x, mask)
515 if inbound_layers:
516 # This will call layer.build() if necessary.
–> 517 self.add_inbound_node(inbound_layers, node_indices, tensor_indices)
518 # Outputs were already computed when calling self.add_inbound_node.
519 outputs = self.inbound_nodes[-1].output_tensors

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/keras/engine/topology.py in add_inbound_node(self, inbound_layers, node_indices, tensor_indices)
569 # creating the node automatically updates self.inbound_nodes
570 # as well as outbound_nodes on inbound layers.
–> 571 Node.create_node(self, inbound_layers, node_indices, tensor_indices)
572
573 def get_output_shape_for(self, input_shape):

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/keras/engine/topology.py in create_node(cls, outbound_layer, inbound_layers, node_indices, tensor_indices)
153
154 if len(input_tensors) == 1:
–> 155 output_tensors = to_list(outbound_layer.call(input_tensors[0], mask=input_masks[0]))
156 output_masks = to_list(outbound_layer.compute_mask(input_tensors[0], input_masks[0]))
157 # TODO: try to auto-infer shape if exception is raised by get_output_shape_for.

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/keras/layers/recurrent.py in call(self, x, mask)
225 constants=constants,
226 unroll=self.unroll,
–> 227 input_length=input_shape[1])
228 if self.stateful:
229 updates = []

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/keras/backend/tensorflow_backen
go_backwards, mask, constants, unroll, input_length)
1304 loop_vars=(time, output_ta) + states,
1305 parallel_iterations=32,
-> 1306 swap_memory=True)
1307 last_time = final_outputs[0]
1308 output_ta = final_outputs[1]

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/tensorflow/python/ops/control_flow
shape_invariants, parallel_iterations, back_prop, swap_memory, name)

```
2634 context = WhileContext(parallel_iterations, back_prop, swap_memory, name)
2635 ops.add_to_collection(ops.GraphKeys.WHILE_CONTEXT, context)
-> 2636 result = context.BuildLoop(cond, body, loop_vars, shape_invariants)
2637 return result
2638

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/tensorflow/python/ops/control_flow_ops.py in BuildLoop(self, pred, body, loop_vars,
shape_invariants)
2467 self.Enter()
2468 original_body_result, exit_vars = self._BuildLoop(
-> 2469 pred, body, original_loop_vars, loop_vars, shape_invariants)
2470 finally:
2471 self.Exit()

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/tensorflow/python/ops/control_flow_ops.py in _BuildLoop(self, pred, body,
original_loop_vars, loop_vars, shape_invariants)
2448 for m_var, n_var in zip(merge_vars, next_vars):
2449 if isinstance(m_var, ops.Tensor):
-> 2450 _EnforceShapeInvariant(m_var, n_var)
2451
2452 # Exit the loop.

/home/nbuser/anaconda3_410/lib/python3.5/site-packages/tensorflow/python/ops/control_flow_ops.py in _EnforceShapeInvariant(merge_var,
next_var)
584 “Provide shape invariants using either the shape_invariants ”
585 “argument of tf.while_loop or set_shape() on the loop variables.”
–> 586 % (merge_var.name, m_shape, n_shape))
587 else:
588 if not isinstance(var, (ops.IndexedSlices, sparse_tensor.SparseTensor)):

ValueError: The shape for while_2/Merge_2:0 is not an invariant for the loop. It enters the loop with shape (1, 4), but has shape (?, 4) after one
iteration. Provide shape invariants using either the shape_invariants argument of tf.while_loop or set_shape() on the loop variables.
```

Icy December 1, 2016 at 8:14 pm #

REPLY

Hi, Jason.
Thank you for your LSTM tutorial! I would like to use it to do some predictions, however, the input_dim is two variables, and the output_dim is one, just like: input: x(t) y(t) output: y(t+1) .I have known that you answered it with the window method. I still have no idea, any suggestions?

Benjamin S. Skrainka December 2, 2016 at 6:41 am #

REPLY

This is an informative and fun article. Thanks!

However, for this application, ARIMA and exponential smoothing perform better out of the box without any tuning.

```
# Compare ARIMA vs. NN
library(forecast)
library(ModelMetrics)

df.raw <- read.csv('international-airline-passengers.csv', stringsAsFactors=FALSE)
df <- ts(df.raw[,2], start=c(1949,1), end=c(1960,12), frequency=12)

# Same train/test split as example
train.size <- floor(length(df) * 0.67)
ts.train <- ts(df[1:train.size], start=c(1949,1), frequency=12)
ts.test <- ts(df[(train.size+1):length(df)], end=c(1960,12), frequency=12)

ts.fit <- auto.arima(ts.train)
ets.fit <- ets(ts.train)

fcast <- forecast(ts.fit, 4*12)
y_hat <- fcast$mean

# Simple ARIMA vs. NN has RMSE = 8.83/26.47 vs. 22.61/51.58 for train/test
ModelMetrics::rmse(ts.train, fcast$fitted)
ModelMetrics::rmse(ts.test, y_hat)

# ETS is even better ... RMSE 7.25/23.05
ets.fcast <- forecast(ets.fit, 4*12)
ModelMetrics::rmse(ts.train, ets.fcast$fitted)
```

```
ModelMetrics::rmse(ts.test, ets.fcast$mean)
```

Jason Brownlee December 2, 2016 at 8:21 am #

REPLY

Agreed Benjamin. The post does show how LSTMs can be used, just not a very good use on this dataset.

Hans April 25, 2017 at 6:04 am #

REPLY

Is this Python too?

libra December 10, 2016 at 8:35 pm #

REPLY

I have a question is how to predict the data outside the dataset

Jason Brownlee December 11, 2016 at 5:27 am #

REPLY

Hi libra, train your model your training data and make predictions by calling model.predict().

The batch size/pattern dimensions must match what was used to train the network.

Nilavra Pathak December 15, 2016 at 1:38 am #

REPLY

Hi, does the dataset need to be continuous … if i have intermittent missing data then is it supposed to work ?

Jason Brownlee December 15, 2016 at 8:29 am #

REPLY

You can use 0 to pad and to mark missing values Nilavra.

Also, try consider imputing and see how that affects performance.

Aubrey Li December 16, 2016 at 6:02 pm #

REPLY

Hi Jason,

This is a wonderful tutorial. As a beginner, just wondering, how do I know when I should add a layer and when I should add more neurons in a layer?

Jason Brownlee December 17, 2016 at 11:09 am #

REPLY

Great question.

More layers offer more “levels of abstraction” or indirection, depending on how you want to conceptualize.

More nodes/modules in a layer offers more “capacity” at one level of abstraction.

Increasing the capacity of the network in terms of layers or neurons in a layer will both rec
(learning rate).

What is the magic bullet for a given problem? There’s none. Find a big computer with lots
sample of the dataset to see what works well.

Aubrey Li December 17, 2016 at 10:25 pm #

Thanks for the reply, another question is, is there a typical scenario we shoul

Jason Brownlee December 18, 2016 at 5:31 am #

REPLY

When you need more representation capacity.

It's a vague answer , because it's a hard question to answer objectively, sorry.

Je December 20, 2016 at 5:32 am #

REPLY

Hi Jason,
Many thanks for the tutorial. Very useful indeed.

Following up the question from Aubrey Li and your response to that, does it mean that if I double the number of LSTM nodes (from four to eight), it will perform better?. In other words, how did you decide that number of LSTM nodes to be of 4 and not 6 or 8?

Thanks ☐

Regards

Je

Jason Brownlee December 20, 2016 at 7:26 am #

REPLY

It may perform better but may require a lot more training.

It may also not converge or it may overfit the problem.

Sadly, there is no magic bullet, just a ton of trial and error. This is why we must develop a strong test harness for a given problem and a strong baseline performance for models to out-perform.

Je December 21, 2016 at 6:02 am #

REPLY

Thanks Jason. Please keep throwing all these nice and very informative blogs / tutorials.

Je

Jason Brownlee December 21, 2016 at 8:47 am #

REPLY

Thanks Je.

nrcjea001 December 20, 2016 at 6:34 pm #

REPLY

Hi Jason

I've been struggling with a particular problem and I am hoping you can assist. Basically, I'm running a stateful LSTM following the same logic and code as you've discussed above and in addition I've played around a bit by adding for example a convolutional layer. My issue is with the mean squared error given at the last epoch (where verbose=2 in model.fit) compared to the mean squared error calculated from trainPredict as in the formula you provide above. Please correct me if I am wrong, but my intuition tells me that these two mean square errors should be the same or at least approximately equal because we are predicting on the training set. However, in my case the mean square error calculated from trainPredict is nearly 50% larger than the mean square error at the last epoch of model.fit. Initially, I thought this had something to do with the resetting of states, but this seems not to be the case with only small differences noticed through my investigation. Does anything come to mind of why this may be? I feel like there is something obvious I'm missing here.

```
model.compile(loss='mean_squared_error', optimizer=ada)
for i in range(500):
XXm = model.fit(trainX, trainY, nb_epoch=1, batch_size=bats, verbose=0,
shuffle=False)
model.reset_states()
print(XXm.history)
```

at epoch 500: {'loss': [0.004482088498778204]}

```
trainPredict = model.predict(trainX, batch_size=bats)
mean_squared_error(trainY, trainPredict[:,0])
Out[68]: 0.0064886363673947768
```

Thanks

Jason Brownlee December 21, 2016 at 8:36 am #

REPLY

I agree with your intuition, I would expect the last reported MSE to match a manually calculated MSE. Also, it is not obvious from a quick scan where you might be going wrong.

Start off by confirming this expectation on a standalone small network with a contrived or well understand dataset. Say one hidden layer MLP on the normalized boston house price dataset.

This is a valuable exercise because it cuts out all of the problem specific and technique specific code and concerns and gets right to the heart of the matter.

Once achieved, now come back to your project and cut it back to the bone until it achieves the same outcome.

Let me know how you go.

nrcjea001 December 22, 2016 at 11:36 pm #

REPLY

Hi Jason

Thanks for getting back to me.

I followed your suggestion by running a simple MLP using the housing dataset but I'm still seeing differences. Here is my code as well as the output:

```
%reset -f
import numpy
seed = 50
numpy.random.seed(seed)
import pandas
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import mean_squared_error

dataframe = pandas.read_csv("housing.csv", delim_whitespace=True,
header=None)
dataset = dataframe.values

X = dataset[:,0:13]
Y = dataset[:,13]

model = Sequential()
model.add(Dense(13, input_dim=13, init='normal', activation='relu'))
model.add(Dense(1, init='normal'))
model.compile(loss='mean_squared_error', optimizer='adam')
```

nep=100
mhist = model.fit(X, Y, nb_epoch=nep, batch_size=3, verbose=0)
print ‘MSE on last epoch:’, mhist.history[“loss”][nep-1]

PX=model.predict(X)
print ‘Calculated MSE:’, mean_squared_error(Y, PX)

MSE on last epoch: 30.7131816067
Calculated MSE: 28.8423397398

Please advise. Thanks

REPLY

nrcjea001 December 23, 2016 at 12:43 am #

Apologies. I forgot to scale. Used a MinMaxScaler

scalerX = MinMaxScaler(feature_range=(0, 1))
scalerY = MinMaxScaler(feature_range=(0, 1))
X = scalerX.fit_transform(dataset[:,0:13])
Y = scalerY.fit_transform(dataset[:,13])

MSE on last epoch: 0.00589414117318
Calculated MSE: 0.00565485540125

The difference is about 4%. Perhaps this is negligible?

REPLY

Jason Brownlee December 23, 2016 at 5:32 am #

Might be small differences due to random number generators and platform differences.

REPLY

David Holmgren December 22, 2016 at 10:07 am #

Hi Jason,

Thank you for an excellent introduction to using LSTM networks for time series prediction; I learned a great deal from this article. One question I did have: if I wanted to plot the difference between the data and prediction, would it be correct to use something like (in the case of the training data):

plt.plot(trainY[0]-trainPredict[:,0]),plt.show()

Once again, many thanks.

Regards,
David

REPLY

unknnw0afa December 23, 2016 at 1:22 pm #

For the codes with stacked ltsm, I’m getting the following error. Copy paste the whole thing doesn’t work either. Any help?

The shape for while_1/Merge_2:0 is not an invariant for the loop. It enters the loop with shape (1, 4), but has shape (?, 4) after one iteration. Provide shape invariants using either the shape_invariants argument of tf.while_loop or set_shape() on the loop variables.

REPLY

Jason Brownlee December 24, 2016 at 4:32 am #

Ouch, I’ve not seen that before.

Perhaps try StackOverflow or the google group for the backend that you’re using?

REPLY

Søren Pallesen December 25, 2016 at 9:18 pm #

Hi Jason.

Thanks for all you valuable advice here.

I have trained a model for time series prediction on a quite big data set, which took 12 hours f

The results (validation accuracy) stayed flat for the first 90 epochs and then began to move up

Now wonder how to add more training on top of a trained model in Keras without loosing the training gained from the first 100 epochs?

Best regards
Søren

Jason Brownlee December 26, 2016 at 7:47 am #

REPLY

Hi Søren,

You can save the weights of your network. Then later load them and continue training / refining weights.

See here for more info on saving and loading a network:
http://machinelearningmastery.com/save-load-keras-deep-learning-models/

Je December 27, 2016 at 10:31 am #

REPLY

Hi Jason,
Another question towards the normalisation. Here, we are lucky to have all the data for training and testing. And this has enabled us to normalise the data (MinMaxScaler). However, in real-life, we may not have all the data in one go and in fact it is very likely the case that we will be receiving data from streams. In such cases, we will never has the max or min or even the sum. How do we handle this case (so that we can feed the RNN with the normalised values?).

One obvious solution, perhaps, is calculating this over the running data. But that will be an expensive approach. Or something to do with stochastic sampling strategy ? Any help Jason?

Thanks in advance

Kind Regards

Je

Jason Brownlee December 28, 2016 at 7:03 am #

REPLY

Great question Je.

For normalization we need to estimate the expected extremes of the data (min/max). For standardization we need to estimate the expected mean and standard deviation. These can be stored and used any time to validate and prepare data.

For more on normalizing and standardizing time series data, see this post:
http://machinelearningmastery.com/normalize-standardize-time-series-data-python/

Je December 28, 2016 at 11:02 pm #

REPLY

Hi Jason,
Thanks for the response and for the pointer. Useful – I have to say. ☐

Kind Regards

Je

Jason Brownlee December 29, 2016 at 7:17 am #

REPLY

Glad to hear it!

Je December 28, 2016 at 11:08 pm #

REPLY

Thanks Jason for the response and for the pointer. Useful – I have to say. ☐

Shaun L January 5, 2017 at 2:21 am #

REPLY

Hi Jason,

Great article! I got a lot of benefits from your work.

One question here, lots of LSTM code like yours use such

trainX[1,2,3,4] to target trainY[5]
trainX[2,3,4,5] to target trainY[6]
…

It is possible to make trainY also be time series? like

trainX[1,2,3,4] to target trainY[5,6]
trainX[2,3,4,5] to target trainY[6,7]
…
So the prediction will be done at once rather than 5 and then 6.

Best regards,

Jason Brownlee January 5, 2017 at 9:24 am #

REPLY

Yes, Shaun.

Reform the dataset with two output variables. Then change the number of neurons in the output layer to 2.

I will have an example of this on the blog in coming weeks.

Shaun L January 7, 2017 at 1:27 am #

REPLY

Thanks, I look forward to your example! I really wonder the advantages and disadvantages in doing so.

Joaco January 9, 2017 at 6:49 pm #

REPLY

Hi Jason, I am here again. I have achieved my goal to predict more than one day in this period of time. But now I have another question. I make X=[x1,x2…x30] and Y=[y1,y2…y7], which means I use 30 days to predict 7 days. When predicting y2, actually I used the real value. So here is the question. How can I put my predicted number,like y2,to the X sequence to predict y3? I am looking forward to your answer.
Thank you very much

Kavitha January 11, 2017 at 12:11 am #

REPLY

Hi Jason, a great tutorial. I'm a newbie, and trying to understand this code. My understanding of Keras is that time steps refers to the number of hidden nodes that the system back propagates to through time, and input dimensions refers to the number of 'features' for a given input datum (e.g. if we had 2 categorical values, the input dimensions would be 2). So what confuses me about the code is that it tries to model past values (look back) as the number of input dimensions. Timesteps is always set to 1. In that case isn't the system not behaving like a recurrent network at all but more like an MLP? Thanks!

Jason Brownlee January 11, 2017 at 9:28 am #

REPLY

Hi Kavitha,

The tutorial demonstrates a number of ways that you can use LSTMs, including using lag variables as input features and lag variables as time steps.

Kavitha January 16, 2017 at 10:21 am #

Got it, thank you!

Nishat January 12, 2017 at 2:56 pm #

Hi Jason, I am looking for a machine learning algorithm that can learn the timing issu
predict an output.

Jason Brownlee January 13, 2017 at 9:08 am #

REPLY

Sounds like an interesting problem Nishat.

This post might help you frame your problem for predictive modeling:
http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/

sss January 13, 2017 at 5:43 pm #

REPLY

I think this is wrong :len(dataset)-look_back-1
it should be len(dataset)-look_back

Jakub January 17, 2017 at 9:04 pm #

REPLY

Hi,

I would like to point out that the graphics

LSTM Trained on Regression Formulation of Passenger Prediction Problem

is the most confusing part of the article.

The red line is NOT the actual prediction for 1,2,3, etc. steps ahead. As we can see from the data, you need to know the REAL value just at the time T to predict T+1, it is not based on your prediction in this setup.

If you need to do a prediction for more steps ahead a different approach is needed.

I am still grateful for the parts of the code you have provided, but this part led me way away from my goal.

Faezeh January 24, 2017 at 3:34 am #

REPLY

Hi Jakub, do you have any idea on what approach to take for multi-step ahead prediction?

Salvo January 19, 2017 at 11:09 pm #

REPLY

Hi,
I would control the input of the internal gate of the cell memory. is it a possible thing to do?
In case of yes, what are the function that allow it? Thanks!

Jason Brownlee January 20, 2017 at 10:20 am #

REPLY

I don't believe this is the case in Keras Salvo. I'm happy to be corrected though.

Salvo January 21, 2017 at 2:57 am #

REPLY

Thanks for your help! These articles are very useful for my studies!

Jason Brownlee January 21, 2017 at 10:34 am #

I'm glad to hear that Salvo.

Nader January 20, 2017 at 4:23 am #

REPLY

in the "LSTM for Regression with Time Steps"
how can we add more layers to the model ?

```
model = Sequential()
model.add(LSTM(4, input_dim=1))
```

```
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=2)
```

How can I add another Layer or more Layers ?

Jason Brownlee January 20, 2017 at 10:24 am #

REPLY

Hi Nader,

Set the batch_input_shape on each layer and set the return_sequences argument on all layers except the output layer.

I'd recommend carefully re-reading the words and code in the section titled "Stacked LSTMs with Memory Between Batches".

I hope that helps.

Anthony January 21, 2017 at 1:25 am #

REPLY

Jason,
Thanks for the nice blog. What Hardware configurations are required for running this program?

Jason Brownlee January 21, 2017 at 10:34 am #

REPLY

Hi Anthony, you're welcome.

A normal PC without a GPU is just fine for running small LSTMs like those in this tutorial.

Nikola Tanković January 21, 2017 at 9:34 pm #

REPLY

I have a small question. I dont see how look_back feature is relevant. If I put look_back to zero or one but increase memory units to lets say 20, I get much better results because the network itself "learns" to look back as much as its needed. Can you replicate that? Isn't that the whole point of LSTM?

Sam January 23, 2017 at 6:57 am #

REPLY

How do you recommend we include additional features, such as moving averages, standard deviation,etc.. ?

Also, how would we tune the Stacked LSTMs with Memory Between Batches to achieve better accuracy ?

Anthony January 23, 2017 at 4:04 pm #

REPLY

Thanks Jason for a wonderful post. Your code uses keras which has tensorflow working in the background. Tensorflow is not available under Windows platform. Is there any way one could run this code in windows?
I am using Anaconda.

Jason Brownlee January 24, 2017 at 11:00 am #

Hi Anthony, absolutely. Use the Theano backend instead:

```
1 pip install theano
```

Hans April 25, 2017 at 6:17 am #

Correction 04.2017: Its available on Windows/Anaconda

Akhilesh Kumar January 23, 2017 at 7:27 pm #

REPLY

I think the way data is normalized in this tutorial is not correct. The shows hetroskadicity and hence needs advanced method of normalization.

Jason Brownlee January 24, 2017 at 11:01 am #

REPLY

I agree Akhilesh.

The series really should have been made stationary first. A log or box-cox transform and then differenced.

S Wollner January 24, 2017 at 2:06 am #

REPLY

I'm sorry to tell you that this is no prediction.
Your LSTM network learned to save the value from t-1 and retrieve it at time t.

Try one thing… train this model on that dataset… and test this on a hole different Timeseries. E.g. a sincurve.

You will get the inputted sincurve with an offset of 1 timestep out. Maybe with some distortion in it.

I can get the same results with an stupid arima model…

This is no prediction at all. It just a stupid system.

Kind Regards,
S. Wollner

Jason Brownlee January 24, 2017 at 11:06 am #

REPLY

Thanks S. Wollner,

It is a trivial perhaps even terrible prediction example, but it does show how to use the LSTM features of Keras.

I hope to provide some updated examples soon.

Hans April 25, 2017 at 4:42 pm #

REPLY

If the example is not predicting anything, is this article somehow misleading for those trying to predict with this code?

Hans April 25, 2017 at 5:01 pm #

REPLY

Now I have two adapted versions of the example, feeded with own data.

One from Wollner and one from Jason. Both are running and plotting.

With some additions, I'm even able to forecast unseen data- BUT…

As beginner, how can I decide whether I'm dealing with real predictions or not?

Jason Brownlee April 26, 2017 at 6:20 am #

Compare to a persistence model:
http://machinelearningmastery.com/persistence-time-series-forecasting-with-p

Hans April 26, 2017 at 3:58 pm #

Thank you Jason.

S. Wollner January 26, 2017 at 8:24 am #

REPLY

Hi again,

I've updated your example so that a real prediction is possible.

What I did:
set look_back to 25
add a linear activation to the Dense layer
and changed trainings settings like batch size

optionally I added detrending and stationarity of signal (Currently it's commented out)

Here is the code:

```
import numpy
import math
import matplotlib.pyplot as plt
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Activation
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

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

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

#plt.plot(dataset);
#plt.show();

# normalize the dataset
#dataset = numpy.log10(dataset) # stationary signal
#dataset = numpy.diff(dataset, n=1, axis=0) # detrended signal
dataset = (dataset – numpy.min(dataset)) / (numpy.max(dataset) – numpy.min(dataset)) # normalized signal

#plt.plot(dataset);
#plt.show();

# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) – train_size
train, test = dataset[:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))

# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
dataX, dataY = [], []
for i in range(len(dataset)-look_back-1):
a = dataset[i:(i+look_back), 0]
dataX.append(a)
dataY.append(dataset[i + look_back, 0])
return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t and Y=t+1
look_back = 25
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(100, input_dim = look_back))
model.add(Dense(1))
model.add(Activation("linear"))

model.compile(loss='mean_squared_error', optimizer='adam')
#model.compile(loss="mean_squared_error", optimizer="rmsprop")
model.fit(trainX, trainY, nb_epoch=100, batch_size=25, validation_data=(testX, testY), verbos
score = model.evaluate(testX, testY, verbose=0)
```

```
print('Test score:', score)

# make predictions
trainPredict = model.predict(trainX, verbose=0)
testPredict = model.predict(testX, verbose=0)

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict

# plot baseline and predictions
plt.plot(dataset)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Kind regards
S. Wollner

Kay February 2, 2017 at 10:52 am #

REPLY

Hello Wollner,
I tried to follow your code, however i got the prediction as a straight line. Where do you think i went wrong.
Thank you.

Luis January 27, 2017 at 1:10 am #

REPLY

Jason,

Thank you for this excellent post. I have reproduce the example and also used a real time-series data set successfully. But I have a simple question:

How I can generate a sequence of predict new values? I mean future values (no the test values), for example, the six first months of the year 1961; values for 1961-01, 1961-02,…1961-06.

Luis

Jason Brownlee January 27, 2017 at 12:09 pm #

REPLY

Hi Luis, you can make predictions on new data by calling y = model.predict(X)

shazz January 27, 2017 at 5:54 am #

REPLY

Hi Jason,
I hope I don't ask for something already in the comments, but at least I did not see it. All my apologizes else.

Based on your dataset, let's assume that we have more features than only the number of passengers per unit of time, for example the "current" weather, fuel price,… whatever.
If I want to use them in the training, the idea is the same, for each one I "copy" the n loopback

Thanks

Jason Brownlee January 27, 2017 at 12:24 pm #

Hi Shazz,

I would recommend creating a new dataset using DataFrame.shift() rather than the crude

Sam January 27, 2017 at 6:06 am #

REPLY

Hello S. Wollner:

I too notice the predictions to be simply mimicking the last known value.
Thanks for posting your code.

I had a couple questions on your post:

1. How did you set the batch_size ? It appears that matches the lookback.
Is that intentional ?

2. Similarly, how did you know how to set the number of neurons to 100 via this line:
model.add(LSTM(100, input_dim = look_back)) ?
That appears to be 4*look_back ?

S. Wollner January 27, 2017 at 11:57 pm #

REPLY

Hi Sam,

To 1.:
No it doesn't have to match the look_back amount.
Here is a similar question and a good answer
http://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

Short, you devide your time series into pieces for training. In this case:
Training_set_size = (data_size * train_size – forecast_amount) / look_back
That is the set for training your network. Now you devide it by the batch_size.
Each batch should have more or less the same size.
In the link above you will see pros and cons about the size of each batch.

To 2.:
Try and error, like almost everything with neuronal networks. That's parameter optimization. There is no true config for all problems. The more neurons you have, the more powerful your network can be. The problem is you also need a larger trainingset.

Normally you iterate through the number of neurons. E.g. you start at 2 and go up till 100 in a step size of 2 neurons. For each step you calculate at least 35 networks (statistical expression) and calculate the mean and variance over the error of train- and testset.
Plot all the results in a graph and take the network with less complexity and best TEST rate (not train!). Consider variance and mean!!!

That's a paper from our research group. In this paper you'll see such a graphic for lvq networks (Figure 4).
http://isommer.informatik.fh-schmalkalden.de/publications/2002_Sci_doc.pdf

Kind regards,
S. Wollner

Sam January 29, 2017 at 6:40 am #

REPLY

Thanks S. Wollner for the guidance.
I'm currently trying to use this LSTM RNN to predict monthly stock returns.
Again though I cannot beat the naive benchmark of simply predicting
t+1 = t or predicting the future return is simply the last known/given return at time t.

I'm wondering what else I can tune /change in the LSTM RNN to remove
the" mimicking" effect ?

---

berkmeister January 29, 2017 at 11:17 pm #

The major difficulty here is that the time series is non stationary – it is both mean trer
hard to forecast using this time series.

You get around this by scaling using the entire dataset, therefore violating the in-sample out-o
into the future, i.e. your test set, for scaling – which unfortunately is not possible in real life.

Jason Brownlee February 1, 2017 at 10:17 am #

REPLY

Hi berkmeister,

The level can be made stationary with order one differencing.

The variance can be made stationary with log or box-cox transforms.

Both methods can be used on test and training data.

Abdulaziz Almalaq January 31, 2017 at 10:14 am #

REPLY

Hi Jason,

Many thanks to your post and tutorial. I really got the most beneficial of ideas to apply the LSTM to my problem.

Jason Brownlee February 1, 2017 at 10:35 am #

REPLY

I'm glad to hear that.

Sam February 2, 2017 at 6:21 am #

REPLY

I believe I have made the stock data in my dataset stationary by taking the first difference of the log of the prices.

However, if I want to include additional features such as volatility, a moving average, etc… would those be computed on the ORIGINAL stock prices or on the newly calculated log differences, which are stationary ?

Jason Brownlee February 2, 2017 at 2:03 pm #

REPLY

Great question Sam.

I don't work with security prices myself, but I expect you will want those measures on the original data.

From a feature engineering perspective, I'd recommend testing alternatives and use what results in the most accurate predictive models.

Regarding predicting security prices in the short term, consider using a persistence model:
http://machinelearningmastery.com/gentle-introduction-random-walk-times-series-forecasting-python/

Sam February 7, 2017 at 10:32 am #

REPLY

Another question I had was on performing 2 or more day ahead forecasts on a stationary time series with first differences.
For example, if we want to forecast 5 days ahead of day (instead of 1 day), would we instead use the differences between t and t-5 ?

Jason Brownlee February 8, 2017 at 9:32 am #

REPLY

Hi Sam,

Forecasts would be made one time step at a time. The differences can then be inverted from the last known observation across each of the predicted time steps.

I hope that answers your question.

Sam February 9, 2017 at 5:49 am #

REPLY

Unfortunately I'm not able to follow.

Suppose we have the following stock price history

Date Price Difference
1/2/2017 100
1/3/2017 102 2
1/4/2017 104 2

1/5/2017 105 1
1/6/2017 106 1
1/7/2017 107 1
1/8/2017 108 1

If we want to forecast what the price will be on January 8th STARTING from January 3rd (a 5 day horizon),
how would build the differences to make the series stationary? If we continue with first differences, then I believe we would only be forecasting the change from Jan 7th to Jan 8, which is still
a 1 day change, not a 5 day ?

Thanks again.

Jason Brownlee February 9, 2017 at 7:30 am #

REPLY

Hi Sam,

Off the cuff: The LSTM can forecast a 5-day horizon by having 5 neurons in the output layer and learn from differenced data. The difference inverse can be applied from the last know observation and propagated along the forecast to get back to domain values.

Sam February 10, 2017 at 4:53 am #

Alright, so if I understand correctly, the 5 outputs from the output layer would correspond to the differences between days 0-1,1 -2, 2-3,3-4, 4-5 respectively?

Thanks for your patience.

Jason Brownlee February 10, 2017 at 9:54 am #

Correct Sam.

Sam February 11, 2017 at 3:52 am #

One more question on that:
Would I also need to modify the target values (trainY) so they
contained 5 targets per sample, instead of just one ? That is to match up the 5 RNN outputs ?

Thanks,

Jason Brownlee February 11, 2017 at 5:06 am #

Yes.

Kim February 8, 2017 at 3:10 am #

REPLY

Hi, Jason
I have some question about using multivariable.

Did I understand correctly?
for example, if i have three variables and one window (just one day, continuous data)
data structure is In this way,

variable1 variable2 variable3 output1
(input_shape=(1, 3))

and, if i have three variables and two windows (two day, continuous data)
data structure is In this way,

variable1(t-1) variable2(t-1) variable3(t-1) output1(t-1)
variable1 variable2 variable3 output1
(input_shape=(2, 3))

is it right way? thank in advance

YS_XIE February 11, 2017 at 1:52 am #

REPLY

Many thanks to your post and tutorial. I really got the most beneficial of ideas to apply the LSTM to my problem.

I have some questions:
1): How to save the test data and predict dat to a text file?
2): How to save the output image ?

Thanks a lot.

Jason Brownlee February 11, 2017 at 5:05 am #

REPLY

You can save data to a file using Python IO functions, npy functions for saving the matrix, or wrap it in a dataframe and save that.

You can save a plot using the matplotlib function savefig().

YS_XIE February 11, 2017 at 11:56 am #

REPLY

Thanks for your quickly reply. I have resolved the problem.

Jason Brownlee February 12, 2017 at 5:33 am #

REPLY

I'm glad to hear that.

Tony Zhang February 15, 2017 at 12:24 am #

REPLY

Hi, Jason
It's a great tutorial. I have learnt a lot from it. Thank you very much.
By the way, is it possible to use the LSTM-RNN to obtain the predictions with a probability distribution? I think it will be even better if LSTM-RNN can do this.
Please let me know if I have the wrong thinking.

Jason Brownlee February 15, 2017 at 11:36 am #

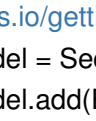
Sure Tony, you could use a sigmoid on the output layer and interpret it as a prob

Tony Zhang February 15, 2017 at 12:26 pm #

REPLY

Thank you for your quickly reply.

Maybe I have asked in a wrong way. I mean like the example above, is it possible we get the probability distributions of the predicted future passengers at the same time? In other words, how confident we are sure about the prediction accuracies.



Jason Brownlee February 16, 2017 at 11:01 am #

REPLY

Not directly Tony.

Amw 5G February 15, 2017 at 6:19 am #

REPLY

Thank you for this, it has been a great help in debugging my own keras RNN code. A suggestion for your root LSTM for Regression with Time Steps model, as examples of what else you could do:

First, incorporate the month number as a predictor. This helps with the obvious seasonality in the time series. You can do this by creating an N-by-lookback shaped matrix where the value equals the month number (0 for January, …, 11 for December). I did it by adjusting the create_dataset function to look like
def create_dataset(dataset, look_back=1):
dataX, dataY, dataT = [], [], []
for i in range(len(dataset)-look_back-1):
a = dataset[i:(i+look_back), 0]
dataX.append(a)
dataY.append(dataset[i + look_back, 0])
b = [x % (12) for x in range(i, i+(look_back))] #12 because that's how many months are in a year
dataT.append(b)
return numpy.array(dataX), numpy.array(dataY), numpy.array(dataT)

Then, feed this into an embedding layer to create a one-hot vector of 12 dimensions. Your model will now have two branches (one for the time series and one for the vector of month indicators), that merge into a single model containing the RNN. F. Chollet has some examples at https://keras.io/getting-started/sequential-model-guide/. I did it with
month_model = Sequential()
month_model.add(Embedding(12, 12, input_length=look_back))
month_model.add(GRU(output_dim=4, return_sequences=True))
month_model.add(TimeDistributed(Dense(look_back)))

series_model = Sequential()
series_model.add(Dense(look_back, input_shape=(look_back,1)))
model = Sequential()
model.add(Merge([month_model, series_model], mode='concat', concat_axis=-1))
model.add(GRU(4, return_sequences=False))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

Lastly, I used a callback to ensure that the model didn't overfit during the 100 epochs. Specifically, if the loss on the validation set stopped decreasing, the model would early terminate. I found I didn't really need even half of the epochs, thus saving some time. E.g.,
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model.fit([np.array(trainT.squeeze()),trainX], trainY.squeeze(), validation_data=([np.array(testT.squeeze()),testX],testY.squeeze()), nb_epoch=100, batch_size=1, verbose=2, callbacks=[early_stopping])

Using a lookback of 3, my training set had a RMSE of 7.38, and 17.69 for the validation set. Which I think is a pretty decent improvement with minimal additional work.

Jason Brownlee February 15, 2017 at 11:38 am #

REPLY

Very nice Amw, thanks!

DAN February 21, 2017 at 11:11 pm #

Hello, I was testing your code, when you run the function create_dataset, it gives
trainX, trainY = create_dataset(train, look_back)

ValueError: too many values to unpack (expected 2)

Jason Brownlee February 22, 2017 at 10:03 am #

REPLY

Hi Dan,

Perhaps check that you do not have any extra white space and that you have not modified the example.

KJ February 23, 2017 at 9:34 am #

REPLY

I think you are missing some Keras dependencies. Make sure you have:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Embedding
from keras.layers import GRU
from keras.layers import Merge
from keras.layers import TimeDistributed
from keras.callbacks import EarlyStopping
```

DAN February 28, 2017 at 1:36 am #

REPLY

The error comes from: trainX,trainY = create_dataset(train, look_back), If I remove trainY, it works…wtf

KJ February 23, 2017 at 9:32 am #

REPLY

Nice idea.

What do you feed into trainPredict and testPredict?

trainPredict = model.predict(numpy.array(trainT.squeeze()),trainX], batch_size=batch_size)
gives me an invalid syntax error.

HD February 18, 2017 at 2:33 am #

REPLY

Hello Sir,

This guide is amazing, but how can we use that to predict out of sample ?

Thank you

Jason Brownlee February 18, 2017 at 8:42 am #

REPLY

Train your model on all your historical data then call model.predict().

HD February 20, 2017 at 9:25 pm #

REPLY

It didn't work for me. I've done: train_size = int(len(dataset) * 1)
and removed all the test data but still it does predict inside the samples.
Is there a function that I should add ?

Chris February 23, 2017 at 4:24 am #

Thank you for this great post.
I just have two questions:

1) What would be the code to add to show the accuracy in percentage?
2) How can I code the red line to show the next prediction at the end of the chart?

Thank you in advance.

Jason Brownlee February 23, 2017 at 8:55 am #

REPLY

Hi Chris,

It is a regression problem so accuracy does not make sense. If it were a classification problem, the activation function in the output layer would have to be changed to sigmoid or similar.

You can predict the next out of sample value by training the model on all available data and calling model.predict()

REPLY

Chris February 24, 2017 at 2:37 am #

Ok, I will try.
Thank you Jason ☺

REPLY

KJ February 23, 2017 at 3:29 pm #

I really appreciate the time you put into this very detailed explanation of time series prediction. I think there are a couple of errors in your code, which appear to be confusing a lot of people:

1. When you read the file, skipfooter should = 2, not 3, as there are only two lines of data after the last value. The existing code prevents the last value from being added to dataset.

2. Similarly, in create_dataset, you should not subtract 1 in the range of the for loop. Again, this prevents the last value from being added to dataX and dataY.

3. Finally, you should not use the the shift train prediction, and the shift test prediction is incorrect. This has the effect of making it appear that the prediction at t=0 instead of t+1. I think this is why many people have been asking how to predict the last value.

Here is my code which both demonstrates and fixes these issues. Please feel free to tell me I'm wrong.

```
# Stacked LSTM for international airline passengers problem with memory
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
dataX, dataY = [], []
for i in range(len(dataset)-look_back): # – 1 IS WRONG. IT PREVENTS LAST VALUE OF DATASET FROM BEING USED FOR dataX AND dataY
a = dataset[i:(i+look_back), 0]
dataX.append(a)
dataY.append(dataset[i + look_back, 0])
return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)

# load the dataset
# No header on data
dataframe = pandas.read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=2) # 3 IS WRONG AS THERE ARE ONLY TWO LINES IN THE FOOTER. THIS PREVENTS THE LAST VALUE FROM BEING READ.
dataset = dataframe.values
dataset = dataset.astype('float32')

print()
print('Last 5 values from dataset')
print(dataset[dataset.shape[0] – 5])
print(dataset[dataset.shape[0] – 4])
print(dataset[dataset.shape[0] – 3])
print(dataset[dataset.shape[0] – 2])
print(dataset[dataset.shape[0] – 1])
print()

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

# split into train and test sets
```

```
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) – train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))

# create and fit the LSTM network
batch_size = 1
model = Sequential()
model.add(LSTM(12, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
model.add(LSTM(12, stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

for i in range(1):
model.fit(trainX, trainY, nb_epoch=1, batch_size=batch_size, verbose=0, shuffle=False) # I JUST SET VERBOSE=0 SO IT IS EASIER TO SEE
THE PRINTED DATA
model.reset_states()

# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# print the data
testX = scaler.inverse_transform(testX[:,0,0])
print()
print('testX','testY', 'testPredict')
print(testX[testX.shape[0] – 5], round(testY[0, testY.shape[1] – 5]), round(testPredict[testPredict.shape[0] – 5, 0]))
print(testX[testX.shape[0] – 4], round(testY[0, testY.shape[1] – 4]), round(testPredict[testPredict.shape[0] – 4, 0]))
print(testX[testX.shape[0] – 3], round(testY[0, testY.shape[1] – 3]), round(testPredict[testPredict.shape[0] – 3, 0]))
print(testX[testX.shape[0] – 2], round(testY[0, testY.shape[1] – 2]), round(testPredict[testPredict.shape[0] – 2, 0]))
print(testX[testX.shape[0] – 1], round(testY[0, testY.shape[1] – 1]), round(testPredict[testPredict.shape[0] – 1, 0]))
print()

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting THIS IS ALSO WRONG
#trainPredictPlot = numpy.empty_like(dataset)
#trainPredictPlot[:, :] = numpy.nan
#trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

trainPredictPlot = trainPredict
testPredictPlot = testPredict

# plot last 25 predictions
plt.figure(figsize=(10,4))
plt.title('Last 25 Predictions')
datasetPlot = dataset[len(dataset) – 25:len(dataset),:]
plt.plot(scaler.inverse_transform(datasetPlot), color='b', label='Actual')
testPredictPlot = testPredictPlot[len(testPredictPlot) – 25:len(testPredictPlot),:]
plt.plot(testPredictPlot, color='r', label='Prediction')
plt.grid(True)
plt.legend()
plt.show()

# shift test predictions for plotting THIS IS ALSO WRONG
testPredictPlot = numpy.empty_like(dataset)
```

```
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back):len(dataset) – 1, :] = testPredict

# plot baseline and predictions
plt.figure(figsize=(10,4))
plt.title('All Data')
plt.plot(scaler.inverse_transform(dataset), color='b', label='Actual')
plt.plot(trainPredictPlot, color='g', label='Training')
plt.plot(testPredictPlot, color='r', label='Prediction')
plt.grid(True)
plt.legend()
plt.show()
```

Justin February 24, 2017 at 12:26 pm #

REPLY

Jason,
I have looked through most of the comments, and not seen this pointed out, but in the "Memory Between Batches" examples, you should not do the reset_state() in between doing predict on the training and test sets. The beginning of your test set should know that it is at the "end" of the training set, rather than at null.

Thank you very much for all this, by the way, it's been super helpful.

m0rtal February 24, 2017 at 9:37 pm #

REPLY

Are you sure?

Quote:
It requires that the training data not be shuffled when fitting the network. It also requires explicit resetting of the network state after each exposure to the training data (epoch) by calls to model.reset_states(). This means that we must create our own outer loop of epochs and within each epoch call model.fit() and model.reset_states().

m0rtal February 24, 2017 at 9:34 pm #

REPLY

If this is a prediction, should'n plot be continued like this?
http://i90.fastpic.ru/big/2017/0224/31/9aa244bfcaebaa4cc8255d858e12d731.png

Please correct me if I'm wrong, we are predicting for look_back periods? Or just t+1?

KJ February 26, 2017 at 1:20 am #

REPLY

It is prediction, but the code as written doesn't predict anything into the future. My updated code http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/#comment-389969 predicts one value into the future based on the existing data.

You would have to add to either code if you want to predict more than one value into the future.

KJ February 25, 2017 at 5:26 am #

REPLY

I'd appreciate it if you would delete the code from my post on February 23, 2017 at 3:29 pm. I've made a couple of improvements to it, so it will make your comments section a lot shorter if people don't have to read through all that old

The improvements are:

look_back = look_back + 1, ie. if the look_back is set to 1, each X will contain t-1 & t, rather th
value, set look_back = 0.

The look_back is added to x before it is split into trainX and trainY. This allows predictions to b
might argue that this is letting the model use data in the training set to make predictions, but s
problem.

The final value in the dataset is added to testX, so the model makes a prediction for 1961-01.

There are a lot of commented out print statements to allow people to see exactly what the dat

Here is the updated code:

```
# Stacked LSTM for international airline passengers problem with memory
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back):
dataX, dataY = [], []
for i in range(len(dataset)- look_back):
dataX.append(dataset[i:(i + look_back), 0])
dataY.append(dataset[i + look_back, 0])
return numpy.array(dataX), numpy.array(dataY)

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

# load the dataset
# https://datamarket.com/data/set/22u3/international-airline-passengers-monthly-totals-in-thousands-jan-49-dec-60#!ds=22u3&display=line
# if using a different dataset, skipfooter=2 may need to have a different value
# and if the csv file does not have a header, add header=None
dataframe = pandas.read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=2)
dataset = dataframe.values
dataset = dataset.astype('float32')

'''
# this can be deleted
# print first and last values from dataset to verify data
print()
print('First 15 values from dataset:')
for i in range(15):
print(dataset[i])
print()
print('Last 15 values from dataset:')
for i in range(dataset.shape[0] – 15, dataset.shape[0]):
print(dataset[i])
'''

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

# reshape into X=t and Y=t+1
look_back = 10
look_back += 1
trainX, trainY = create_dataset(dataset, look_back)

# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) – train_size
trainX, testX = trainX[0:train_size,:], trainX[train_size:len(trainX),:]
trainY, testY = trainY[0:train_size], trainY[train_size:len(trainY)]

'''
# this can be deleted
# print trainX and trainY to verify data
print()
print('trainX + trainY')
for i in range(trainX.shape[0]):
print(i + look_back, end=' ')
for c in range(look_back):
print(trainX[i,c], end=' ')
print(trainY[i],' ')

print()
print('testX + testY')
for i in range(testX.shape[0]):
print(i+ look_back + trainX.shape[0], end=' ')
```

```
for c in range(look_back):
print(testX[i,c], end=' ')
print(testY[i],'')
'''

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))

# create and fit the LSTM network
batch_size = 1
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
model.add(LSTM(4, stateful=True)) # There is no need for an batch_input_shape in the second layer
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

for i in range(100):
model.fit(trainX, trainY, nb_epoch=1, batch_size=batch_size, verbose=0, shuffle=False) # I JUST SET VERBOSE=0 SO IT IS EASIER TO SEE
THE PRINTED DATA
model.reset_states()

# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()

# add last entry from dataset to be able to predict unknown testY
a = dataset[len(dataset) – look_back:len(dataset), 0]
a = numpy.reshape(a, (1, a.shape[0]))
a = numpy.reshape(a, (a.shape[0], a.shape[1], 1))
testX = numpy.append(testX, a, 0)

testPredict = model.predict(testX, batch_size=batch_size)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# print next predicted value
print()
print('Prediction for 1961-01:', round(testPredict[len(testPredict) – 1, 0]))
'''

# this can be deleted
# print testX, testY and testPredict to confirm code is working
train = scaler.inverse_transform(trainX[:,0,0])
train = numpy.reshape(train, (len(train), 1))
test = scaler.inverse_transform(testX[:,0,0])
test = numpy.reshape(test, (len(test), 1))
for c in range(1, look_back):
train1 = scaler.inverse_transform(trainX[:,c,0])
train1 = numpy.reshape(train1, (len(train1), 1))
train = numpy.append(train, train1, axis = 1)
test1 = scaler.inverse_transform(testX[:,c,0])
test1 = numpy.reshape(test1, (len(test1), 1))
test = numpy.append(test, test1, axis = 1)
trainX = train
testX = test
# print number of rows in data
print()
print('Number of values in:')
print('trainX','trainY', 'trainPredict')
print(trainX.shape[0], ' ', trainY.shape[1], ' ', trainPredict.shape[0])
print('testX','testY', 'testPredict')
print(testX.shape[0], ' ', testY.shape[1], ' ', testPredict.shape[0])
# print trainX, trainY and trainPredict to verify data
print()
print('trainX' , ' trainY', 'trainPredict')
for i in range(trainX.shape[0]):
print(i + look_back, end=' ')
for c in range(look_back):
```

```
print(trainX[i, c], end=' ')
print(round(trainY[0,i]), end=' ')
print(round(trainPredict[i,0]), ' ')
# print testX, testY and testPredict to verify data
print()
print('testX' , ' testY', 'testPredict')
for i in range(testX.shape[0]):
print(i + look_back + trainX.shape[0], end=' ')
for c in range(look_back):
print(testX[i, c], end=' ')
if(i < testX.shape[0] – 1):
print(round(testY[0,i]), end=' ')
else:
print(' ', end=' ')
print(round(testPredict[i,0]), " ")
'''

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print()
print('Train Score: %.2f RMSE' % (trainScore))
print()
# THIS HAS BEEN REMOVED, BECAUSE WHEN THE LAST ENTRY IS ADDED TO testX,
# testPredict HAS MORE VALUES THAN testY, WHICH WILL THROW AN EXCEPTION
#testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
#print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset) # create array with same shape as dataset
trainPredictPlot[:, :] = numpy.nan # fill with nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset) # create array with same shape as dataset
testPredictPlot[:, :] = numpy.nan # fill with nan
test = ['nan']
test = numpy.reshape(test, (len(test), 1))
testPredictPlot = numpy.append(testPredictPlot, test, 0)
testPredictPlot[len(trainPredict) + look_back:len(dataset) + 1, :] = testPredict

# plot baseline and predictions
plt.figure(figsize=(10,4))
plt.title('All Data')
plt.plot(scaler.inverse_transform(dataset), color='b', label='Actual')
plt.plot(trainPredictPlot, color='g', label='Training')
plt.plot(testPredictPlot, color='r', label='Prediction')
plt.grid(True)
plt.grid(b=True, which='minor', axis='both')
plt.minorticks_on()
plt.legend()
plt.show()

# zoom in on the test predictions
plt.figure(figsize=(10,4))
plt.title('Test Predictions')
'''
# the next two lines plot the data set to confirm that testY is the same as the dataset
# this can be deleted
datasetPlot = dataset[len(dataset) – len(testPredict) + 1:len(dataset),:] # subtract 24, because
plt.plot(scaler.inverse_transform(datasetPlot), color='y', label='dataset')
'''
testYPlot = numpy.reshape(testY, [testY.shape[1], 1]) # testY is [0, 47] need to change shape
testYPlot = testYPlot[testYPlot.shape[0] – len(testPredict) + 1:testYPlot.shape[0]] # subtract 2
plt.plot(testYPlot, color='b', label='testY')
testPredictPlot = testPredict[len(testPredict) – len(testPredict):len(testPredict), :]
plt.plot(testPredictPlot, color='r', label='Prediction')
plt.grid(True)
plt.grid(b=True, which='minor', axis='both')
plt.minorticks_on()
plt.legend()
plt.show()
```

Hans April 25, 2017 at 9:27 am #

REPLY

It is nearly impossible to reproduce the original formatting of this code.
Could you provide it formatted or on Github?

Robert February 26, 2017 at 3:15 pm #

REPLY

Thanks for the awesome tutorial, Jason, and for being so helpful in the comments!

I'd like to predict into the future using time series data that has multiple observations from each date (and goes beyond a single year), and that also has multiple other features besides just the date and the label. I would also like to predict more than 1 step into the future, and to predict multiple dependent variables simultaneously if possible. I saw how to use additional features from other comments. I also think I saw how to predict more than 1 step into the future.

Could you explain how I can use data with multiple observations from the same date? I'm really stuck on understanding that. If you have the time, could you also explain how to simultaneously predict multiple dependent variables?

One other thing…how do I normalize the data when I have multiple variables, not all of which are numbers?

Jason Brownlee February 27, 2017 at 5:49 am #

REPLY

Hi Robert,

Great question. I'm working on more examples like this at the moment.

Generally, multiple input features is multiple multivariate time series forecasting. You can structure you data so that each column is a new feature in the LSTM format of [samples, timesteps, features].

A time-horizon of more than one timestep is called multi-step time series forecasting. Again, you can structure your data so that the output has multiple columns and then specify the output layer of your network with that many neurons.

This post might help with restructuring your data:
http://machinelearningmastery.com/time-series-forecasting-supervised-learning/

This post might help with scaling your data:
http://machinelearningmastery.com/normalize-standardize-time-series-data-python/

I hope that helps.

Donald February 28, 2017 at 6:45 pm #

REPLY

Hello,

If my input file have multiple columns "open, high, low, close, volume", how can I adapt the script to use all columns for training and then make prediction for "close" column?

Jason Brownlee March 1, 2017 at 8:34 am #

REPLY

Sorry Donald, I do not have a good tutorial for multivariate time series for you to work from. Not yet anyway.

Generally, the principles are the same as the univariate case. See this post on how to structure your data:
http://machinelearningmastery.com/time-series-forecasting-supervised-learning/

Nilavra Pathak March 6, 2017 at 11:47 am #

REPLY

Hi,

Nice tutorial. I recently used an LSTM forecasting.

I have a couple of questions.

1. When we are providing the training set is the LSTM which is being trained equal to the leng
data is a sequence of 10000 values then is the LSTM equal to a 10000 length unrolled recurre
/2015-08-Understanding-LSTMs/

If so then shouldn't we part up the data of sequences of smaller length like your tutorial in http
lstm-recurrent-neural-networks-python-keras/

2. I see that you are normalizing the entire dataset, and the data seems to be a monotonically increasing one. So any future data may not be scaled within that range .
How will you deal with such a case, where future data value is not within the range of observed data.

Jason Brownlee March 7, 2017 at 9:29 am #

REPLY

Hi Nilavra,

I'd recommend trying many different representations for a given perdiction problem. Try breaking it into sub-sequences, try using lag obs as features and time steps, see what works best for your problem.

It is a good idea to difference a dataset with a trend (changing level). Also, it is a good idea to power transform a dataset with an increasing variance (e.g. ideally boxcox or log). The dataset in this example could use both transforms.

Viral Mehta March 7, 2017 at 7:41 am #

REPLY

Why do we see a discontinuity between train prediction (green) and test prediction (red)?

anthony March 8, 2017 at 10:16 pm #

REPLY

Is your lstm code performing forecasts for multiple time steps?

Arslan March 12, 2017 at 5:41 pm #

REPLY

I have a multivariate time series in which I have fields such as business days, holidays, product launch data etc. in addition to the Y (variable to forecast). How can I implement this model using a LSTM?

I tried to modify the data such that my Nxm training set (trainX) contains m lags for N rows (using 'create_dataset' method from the code), and then concatenated the additional information (business days, holidays, product launch data etc.) as columns. However, I realize that , it does not make sense. I will somehow have to pass a data vector in place of every Y value and its m lags (containing the additional information: holidays, product launch data and other info). If this makes sense to you, please advise how to go about with this. Thanks

Jason Brownlee March 13, 2017 at 7:39 am #

REPLY

Hi Arslan, see this post on how to structure your data as supervised learning for multivariate time series:
http://machinelearningmastery.com/time-series-forecasting-supervised-learning/

lotusirous March 15, 2017 at 3:27 am #

REPLY

Thank you for your great article.

However, in your example. Why don't you consider preprocessing methods such as: log and difference dataset? Most of research papers showed that preprocessing data can improve prediction performance.

Jason Brownlee March 15, 2017 at 8:13 am #

REPLY

I agree, the example would be better if the data was made stationary first (log tra
fuller examples in upcoming blog posts.

Hobart March 16, 2017 at 1:18 am #

REPLY

Thanks for great article!!!! I now probably can practice LSTM in code! ☐

I got one question when I try to define my LSTM network. Say if I have some different datas, c
Shall I use different batches or different features? (I know feature is designed for this, but it se

dataX, dataY, dataZ, dataA all hold scalar value.

Another question is practical (basically python related). Say I have dataX and try to predcit da

build training dataset like [3,2,].

Thanks

Jason Brownlee March 16, 2017 at 8:01 am #

REPLY

There are many ways to represent time series data with LSTMs.

I'd recommend using a stateful LSTM with 1 lag variable as input and let it learn the sequence. Then try all other structures (timesteps/features) you can think of to see if you can out-perform your baseline.

Yes, consider using the Pandas shift() function. I offer examples in my recent time series blog posts.

Stefan March 18, 2017 at 1:31 am #

REPLY

Awesome post and simple (as it should be)!

What I'd like to know from this is if one could try to predict *one* signal given *n* other signals?

That would mean that the network learns about non-linear correlations between multiple signals and outputs a desired target signal. Can this be done using LSTM networks?

Jason Brownlee March 18, 2017 at 7:48 am #

REPLY

I don't see why not, as long is the signals are correlated.

klaas March 19, 2017 at 12:51 am #

REPLY

I have a similar dataset. However when trying to reshape the X array into the LSTM required format i receive an error : "lstm tuple index out of range". Searched on Google but can't find the solution that works. Any suggestions?

Jason Brownlee March 19, 2017 at 6:12 am #

REPLY

Check your data carefully ensure the shape you are requesting makes sense.

Remember, LSTMs require data in [samples, timesteps, features] format.

Apurv Verma March 19, 2017 at 9:46 pm #

REPLY

I feel like there is a delay effect happening. The shifted predicted values are closer to the value at current timestamp. Do you think there is an explanation of why that particular effect is being observed?

linamede March 21, 2017 at 2:04 am #

REPLY

Would it be possible to learn to predict more than one steps ahead at once? For example, instead for giving past 3 data, learn to predict the future 3 data. (instead of only one future datum, as in your example)

Jason Brownlee March 21, 2017 at 8:42 am #

Yes, you will need to reframe your problem and specify 3 neurons in the output la

Sandi March 23, 2017 at 3:45 pm #

REPLY

Nice blog Jason!

In this tutorial you always give 4 output dimensions
model.add(LSTM(4, input_dim=look_back))

When I set the output dimension = 1, like:

model.add(LSTM(1, input_dim=look_back))

it returned error maessage:
TypeError: Cannot convert Type TensorType(float32, 3D) (of Variable Subtensor{:int64:}.0) into Type TensorType(float32, (False, False, True)).
You can try to manually convert Subtensor{:int64:}.0 into a TensorType(float32, (False, False, True)).

Any suggestions?

Jason Brownlee March 24, 2017 at 7:52 am #

REPLY

Sorry, I have not seen this fault.

Pablo Estrada March 24, 2017 at 12:07 pm #

REPLY

Hello! Thanks a lot for your post! I have a small question? How do I modify the network in order to predict more than one day? For example, getting the next week of the passengers What are the specific parameters I should modify to make the prediction longer?

Jason Brownlee March 25, 2017 at 7:32 am #

REPLY

You can reframe the problem to predict multiple days.

You can also call the model again and again and use predictions as input observations.

This post will give you some ideas:
http://machinelearningmastery.com/multi-step-time-series-forecasting/

Mete YALCİNER March 25, 2017 at 4:44 am #

REPLY

Thank you jason good article

But code alittle bit incorrect , if we use the network in time series problems tests array must start end of train array in first example test array starts end of train array+1 in anothers exaples test array stars end of train array+3 …. etc if our inputs is small like 1 and 3 no problem but if our inputs big like 170 ,200 it is big problem…… Think please we predict stocks prices our tests stars 150 days later and will give incorrect results, coz market is changable……

than
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

must change like this

train, test = dataset[0:train_size,:], dataset[train_size-look_back-1:len(dataset),:]

and
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict

must change like this

testPredictPlot[len(trainPredict)+look_back:len(dataset)-1, :] = testPredict

Mete YALCİNER March 25, 2017 at 5:02 am #

REPLY

so sorry i must add so that □ i have a predict problem i tryed last tree mounts with A
e-3 , when i see your article i decided try DeepLearning with keras now i cannot belive my eye
much……

Shailen March 25, 2017 at 6:27 am #

Hi Jason, Can this technique work for dynamical problems as well, where the time-va
Ultimately the model should be able to provide/predict time-varying response for any arbitrary

Freddie March 26, 2017 at 5:20 pm #

REPLY

Hi!

I'm trying to create a curve predictive model to generate new curves (for example, sinusoidal curves), but the problem is that when I train my model with curves with different periods, the model can only generate curves with the same period (like an average of the periods of the training set) which could be the problem?

thanks

Jason Brownlee March 27, 2017 at 7:53 am #

REPLY

It is hard to know.

Perhaps the problem requires a more sophisticated model or a simpler representation, or perhaps the problem is too difficult.

George March 27, 2017 at 7:30 am #

REPLY

Hello Jason and thanks for your very nice tutorials.

I have this kind of problem and I wanted to ask you in which category it fits.

Let's say I have a device which measures temperatures.The temperatures will have a "specific" range (I mean the source of the temprerature will be a steady source).

Now, I will take some thousands of temperatures vaues in the beginning and then, I want to be able to train my network , and when it sees a temperature which is not a good fit in the preious values, to reject them.

What kind of problem is this?Time series? Kernel density?Unsupervised cluster?

Thank you very much!

Jason Brownlee March 27, 2017 at 7:59 am #

REPLY

This sounds like "anomaly detection" or "change detection", even "outlier detection".

These terms will help you find good algorithms to try.

George March 27, 2017 at 8:27 am #

REPLY

Great!Thanks a lot!

Jason Brownlee March 28, 2017 at 8:18 am #

REPLY

You're welcome George.

Kunpeng Zhang March 29, 2017 at 2:18 am #

REPLY

Hi Jason,
thank you very much for your great post.
I have a naive confusion regarding the 'Test Score' part.
Why we use
'testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))' instead of
'testScore = math.sqrt(mean_squared_error(test[:286], testPredict))' ?
What's the difference between the two?
And also I am aware that you use 'testScore = model.evaluate(testX, testY, verbose=0)' to trig
Which one is the best to evaluate our model? Could you give me some advice?
Thank you for your work. I appreciate it a lot.
Best regards.

marm March 30, 2017 at 7:56 am #

REPLY

Hi Jason, That is really helpful post. I have a question about random walk data, Can

also read your post about prediction of random walk but still believe LSTM is much better than other methods. Can you give me some advice?

Jason Brownlee March 30, 2017 at 8:59 am #

REPLY

You can, but I doubt you will do better than a persistence model.

In fact, if the data is a true random walk, then you will not do better.

Fisher March 30, 2017 at 10:29 am #

REPLY

Hi Jason,

Nice talk! Very clear and useful, being my first guide to implement LSTM in time series prediction. I've followed many of your articles, thx for your sharing!

Only one different opinion, it seems that you do Min-Max firstly on the whole dataset before training and prediction, which I think may be improper. Because "test set" should be treated like "online" data in real world, i.e. we never know what's coming next into our model when we do prediction. If you scale test set first, then you are actually putting information of test set into training process, this will improve model performance on test set, but obviously, it's improper. So I think the correct pipeline should be:

Step 1. Split dataset into train set and test set
Step 2. Train Scaler on train set and convert train set. i.e. trainSet_convert = scaler.fit_transform(train set)
Step 3. Train LSTM on trainSet_convert .
Step 4. Use scaler to transforms test set, i.e. testSet_transformed = scaler.transform(test set)
Step 5. Use LSTM to do prediction and evaluation on testSet_transformed

Here Step 1.~3. are training stage, Step 4.~5. are prediction stage. The main difference compared with your pipeline is that, scaler transforms test set on prediction stage, not training stage.

Jason Brownlee March 31, 2017 at 5:50 am #

REPLY

Yes, ideally you would perform scaling on the training dataset and use the coefficients from training to scale test (e.g. min/max or stdev/mean).

Andrea March 31, 2017 at 8:00 pm #

REPLY

Hi Jason,

Thanks a lot for the useful articles, really good job!

I just have one question: also if I've already seen "4 Strategies for Multi-Step Time Series Forecasting" I still can't get how to predict more future values than my initial dataset.

For example: If my dataset is composed by 100 values, how I can predict the next 10 values and display it in the same way you have done in this example?

Thank you in advance!

Jason Brownlee April 1, 2017 at 5:54 am #

REPLY

You can configure the LSTM to have 10 units in the output layer.

You can treat it as a many-to-one RNN or a many-to-many and use a seq2seq paradigm.

I have tutorials covering this scheduled for the blog.

Mike Williamson April 7, 2017 at 1:43 am #

REPLY

I went through this example, but I feel like there is something I am not understanding
advance.

E.g., if we look at the generated test data (testPredict from the line

testPredict = model.predict(testX, batch_size=batch_size)

), we see that we are using testX to generate predictions. Well, testX is merely a feature conta

day we're next trying to predict.

Then, once a prediction is made, that prediction is actually *ignored* when doing to subsequent prediction, and instead we use the actual data from testX.

So, is this model really only predicting one day in advance? If so, how is it useful? What am I missing here? I know that LSTM RNN's are very powerful, but this example is not convincing me.

(FWIW, I reproduced this example by correcting it and truly using the predictions to feed the subsequent predictions, and the predictive capablility falls apart. After a few days, it just "stabilizes" and stops moving altogether.)

pawan April 9, 2017 at 4:06 am #

REPLY

I agree with you! I have raised same question in my comment below. Almost all tutorial on using ML for time series forecasting is doing this type of mistake

---

Jason Brownlee April 9, 2017 at 2:45 pm #

REPLY

This is called walk forward validation and assumes new data is available each day, at least in this domain.

Anis April 7, 2017 at 11:43 pm #

REPLY

Thank you for this tuto.
There is one thing that I didn't understand.
I want to interpret the performance of my algo using the RMSE score.
But I was always working with scores in the range of 0 and 1 which is not the case in the first example(LSTM for regression).
So please , I want some details of the score of RMSE and its significance.

Thank you !!!

pawan April 9, 2017 at 4:00 am #

REPLY

RMSE may or may not be between 0 to 1, depending upon the scale of target variable. RMSE mathematically is sum of sqaure of residuals, which in a way tells how much of variance model is not able to explain.

Since, we know the variance in original data we can use it to divide the RMSE, which gives Normalized RMSE or NRMSE, for a good model this quantity should always be less than 1, the closer it is to 0 the better it is. If it's value is more than, it means that our model is introducing more variance than actually present in data, which is not the goal of any modeling exercise.

In statistics, 1- NRMSE is R-square, which is used to assess the goodness of a model. if R-square is closer to 1, model almost explains all the variance observed, if it is closer to 0, model does not explain any variance and in this case 'mean' of target variable would be much better predictor the than model itself.

Hope this helps!

Jason Brownlee April 9, 2017 at 2:55 pm #

REPLY

This post shows how to implement RMSE from scratch and other performance measures:
http://machinelearningmastery.com/time-series-forecasting-performance-measures-with-python/

pawan April 9, 2017 at 3:54 am #

REPLY

When forecasting for test data, I think we should not assume that lagged data is avai
available, i think we are over-stating the power of LSTM for time series forecasting.

As per me, right method to test would be –
1. When forecasting for time t + 1, we can take t and t – 1 values from train data. Here, t being
2. But when forecasting for t + 2, t + 1 and t has to be used, in reality we don't know t + 1, as v
should be used instead.
3. Step 2 will have to be repeated for time t + 3, t + 4, …..
4. Once we have forecast using steps 2 and 3 for test period, we should compare it with the a

If above is followed, we will get much higher error than we are currently getting by assuming t

It would be good to know your thought on this

Jason Brownlee April 9, 2017 at 2:59 pm #

REPLY

I agree in the case when we remaking a multi-step forecast.

Anis April 10, 2017 at 10:33 pm #

REPLY

Thank you for yours responses.
I asked this question because I follow the course Algo LSTM with Keras but using another dataset.
And I get as result RMSE= 0.01 but when I plot the result i found that the prediction graph are not really fiiting with our dataset.

Yasmin April 12, 2017 at 4:13 pm #

REPLY

Hey, thanks for such a detailed post. I thought you might like to check out a guest post we recently had on using LTSM RNNs in Keras Tendorflow for trend prediction using just 3 steps – let us know what you think…
https://www.freelancermap.com/freelancer-tips/11865-trend-prediction-with-lstm-rnns-using-keras-tensorflow-in-3-steps

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

REPLY

Thanks for the link.

JD April 13, 2017 at 10:56 pm #

REPLY

Great tutorial! Thank you for your effort and time Sir.

I think this is the one of few tutorials that actually talk about how to manage data matrix.

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

REPLY

Thanks JD.

Srinivas April 15, 2017 at 2:38 am #

REPLY

Thanks a lot for the great tutorial Jason. This is super helpful.

However, I have one question. For the stateful LSTM when we are running the epochs manually, I understand why you are doing the model.reset_states() after each epoch.

But when we are finally doing the prediction, we run a predict on the train data and then do a reset_states() before doing a predict on the test data.

I think we do not need this as the test data is right after the train data temporally. So, I think we can use the previous state of the internal memory of the LSTM layer. What do you think?

Thanks,
Srinivas

Jason Brownlee April 15, 2017 at 9:40 am #

REPLY

We may.

See this post on seeding state in LSTMs:
http://machinelearningmastery.com/seed-state-lstms-time-series-forecasting-python/

Dan April 17, 2017 at 6:06 am #

REPLY

I updated to the latest version of keras and tensorflow and get extremly bad results. Any suggestions for that Jason? Did you use the latest version of tensorflow?

REPLY

Jason Brownlee April 18, 2017 at 8:26 am #

Consider increasing the number of training epochs.

Consider running the example multiple times and take the average of results.

REPLY

Moha April 24, 2017 at 3:05 pm #

Hi, The way this post wrote very well summarise.Thanks. I'm a beginner to ML student in campus in SL. I would like to understand the code snippet what each line does, where can I get the clear idea on that ?

REPLY

Jason Brownlee April 25, 2017 at 7:44 am #

This might be a good place to start:
http://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/

REPLY

Deepak April 24, 2017 at 3:13 pm #

Thanks for the detailed blog. this helped me apply this to predicting gender using name with minor modification to accomodate many-to-one architecture. the link can be found here
https://medium.com/@prdeepak.babu/deep-learning-gender-from-name-lstm-recurrent-neural-networks-448d64553044

Thanks again! keep writing !!

REPLY

Jason Brownlee April 25, 2017 at 7:44 am #

Well done!

Thanks for sharing the link.

REPLY

Hans April 25, 2017 at 7:57 am #

What about the so called mimicking effect described in several comments here Jason?

I'm new to this topic and slightly concerned.
Could you provide an elaborated statement in regard to this 'critic' (t+1 etc.)?

REPLY

Jason Brownlee April 26, 2017 at 6:18 am #

I refer you to this updated tutorial and compare skill to a persistence model:
http://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

Hans April 26, 2017 at 4:06 pm #

Thank you Jason.

REPLY

Hans April 28, 2017 at 12:01 am #

I have compared the performance, running the code on

http://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-r

..with the performance of the recommended baseline model, consisting a persistence

A) Feeded with the shampoo example data

The results are:

baseline score: 133.16 RMSE

LSTM simple example score: 142.44 RMSE > 133.16 RMSE

LSTM complete example score: 107.214 RMSE < 133.16 RMSE

B) Feeded with airline data the results are:

baseline score: 47.81 RMSE

LSTM simple example score: 44.87 RMSE 47.81 RMSE

What does this mean in regard to the so called mimicking effect?
And the criticism that there is actually no prediction involved.
I've got different results and multiple options to interpret them.
I'm a little bit confused now.

REPLY

Hans April 28, 2017 at 12:13 am #

Sorry, the parser of the forum has stripped out some of my code.

B) Fed with airline data the results are:

baseline score: 47.81 RMSE

LSTM simple example score: 44.87 RMSE, is less then baseline 47.81 RMSE

LSTM complete example score: 61.369 RMSE, is greater then baseline 47.81 RMSE

What does this mean in regard to the so called mimicking effect?
And the criticism that there is actually no prediction involved.
I've got different results and multiple options to interpret them.
I'm a little bit confused now.

REPLY

Jason Brownlee April 28, 2017 at 7:42 am #

It may suggest the LSTM is more skillful than persistence.

Neural networks will give different results each time they are run. I recommend re-running an experiment many times (30) and taking the average performance.

More on the stochastic nature of neural nets here:
http://machinelearningmastery.com/randomness-in-machine-learning/

I also recommend tuning the LSTM to the problem, this post may help:
http://machinelearningmastery.com/improve-deep-learning-performance/

REPLY

Hans April 25, 2017 at 5:12 pm #

Trying to adapt some versions from the user comments I get the error message:

"TypeError: ('Keyword argument not understood:', 'input_dim')"

If I try instead "input_shape" it says "int object is not iterateable"

Does somebody has a solution for this?

REPLY

Hans April 26, 2017 at 4:05 pm #

Update:

There seems to be an undocumented change in the Keras-API.
Got very few infos out of the web.

REPLY

Alex April 28, 2017 at 3:48 am #

Hello,

This a good tutorial.
The only thing which is not clear is how to predict a value that we don't know it.In this exemple we can predict only the value in our data set and we are not able to predict futur values !!!!!!!!!!!!

Please I want some clarifications

REPLY

Jason Brownlee April 28, 2017 at 7:54 am #

You can predict a future value by feeding in the last observations as input (X) to the function:

yhat = model.predict(X)

REPLY

Hans April 28, 2017 at 12:43 pm #

I created a 'Jason-checked' example here…

http://machinelearningmastery.com/time-series-prediction-with-deep-learning-in-python-with-keras/#comment-397444

REPLY

Kaoutar April 28, 2017 at 6:49 pm #

Hello Jason,

Thanks for sharing this great tutorial! Can you please also suggest the way to get the forecast For example, if we want to forecast the value of the series for the next few years (ahead of current time–As we usually do for the any time series data), i have a data from 2000 to 2016 ,and i want to have the forecast for the 4 next years . Thanks

REPLY

Jason Brownlee April 29, 2017 at 7:22 am #

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

REPLY

Daniel Luque April 30, 2017 at 4:06 am #

Hi Jason!, first your work is awesome □ and I want to thank you for sharing your knowledge, I have a question, how the model can predict future valued that are not in the data set?. All the examples I have seen in the internet just mimic the last part of the data set but not predict anything. I want to predict future movements but unless I know it can be done I have not find any way to do it.
Hope you can help me to do this.
Thanks so much!.

REPLY

Jason Brownlee April 30, 2017 at 5:35 am #

Fit the model on all available data and then call model.predict(X) where X are the last few observations.

REPLY

cdsj May 1, 2017 at 2:27 am #

hi jason，i follow your tutorial,program can run ,but when i change little ,the result is
i add the "metrics=['accuracy']":
model.compile(loss='mean_squared_error', optimizer='adam',metrics=['accuracy'])
when running,loss is down,but acc not change.why?thank you !

REPLY

Jason Brownlee May 1, 2017 at 5:59 am #

This is the challenge of applied machine learning.

Alex May 1, 2017 at 7:42 pm #

REPLY

Hello again ,

When I set look_back=50 and I want to predict the 51 value , I found that I must provid 52 value and no 50 values to predict the futur.

It is supposed to work like that ???

Alex May 3, 2017 at 7:29 pm #

REPLY

I think that my question is important. I am confusing with this algorithme.
When we set look_back = 50 and I provide a dataset with 50 measure to predict the 51 measure, I get an error.
I only can make prediction of the 53th value using 52th measure in dataset and look_back = 50

This is normal ????

Alex May 4, 2017 at 11:37 pm #

REPLY

If we add the metrics accuracy
model.compile(loss='mean_squared_error', optimizer='adam' ,metrics=['accuracy'])
we will ntice that the accuracy is not good

Edward May 7, 2017 at 3:37 pm #

REPLY

Hello Jason, I have been reading a lot on your blog, very helpful indeed. So first my thanks, I'm fairly new to python and machine/deep learning, but with your examples I got a good starting point.

I have taken one of your lstm examples and adapted it , also changing to GRU and got decent results, however I would like to improve them. Unfortunately I couldn't find anything on the internet which would fit what I need.

So my hope is that you could point me in the right direction or help…I am trying to model a time series , now my problem is that it behaves differently on different days of the week and also on bank holidays.

I would like to "add"information to any time series step such as Day of week , week , distance to next bank holiday and which bank holiday is the next one. I have found tf. sequence example but not sure if that is what I am looking for.
Thanks in advance and Best Regards
Ed

Jason Brownlee May 8, 2017 at 7:42 am #

REPLY

I would recommend providing this additional context (day or type of day) as a separate input variable.

Edward May 12, 2017 at 1:40 am #

REPLY

HI Jason, thanks for the reply.I have done exactly that now, still fiddling abit to see the effects, but I hope that ultimately with the rihgt "tuning" that this will improve the results.
Best Regards
Ed

Jason Brownlee May 12, 2017 at 7:44 am #

Hang in there Ed!

Shiva May 11, 2017 at 10:46 pm #

Hi Jason,
When it comes to displaying the predictions in the plot, my testpredict has fewer values than t
to make in the code for text predict to predict the values for 1/31/1961, if the dataset ends at 1

Jason Brownlee May 12, 2017 at 7:42 am #

REPLY

You could trim off the test data for which you did not make predictions.

Antonio May 14, 2017 at 12:09 am #

REPLY

Hi Jason, excellent example.

I think there is a terrible mistake when you divide the dataset between training and test. I rather split that samples randomly (but sequentially). Just give it a try and check the mse:

60% – train set, 20% – validation set, 20% – test set:
train, validate, test = np.split(df.sample(frac=1), [int(.6*len(df)), int(.8*len(df))])

or

70% – train set, 30% – test set:
train, test = train_test_split(df, test_size=0, random_state=42)

Antonio May 14, 2017 at 12:10 am #

REPLY

Just be sure they get sorted:

train.sort_index(inplace=True)
test.sort_index(inplace=True)

#optional
validate.sort_index(inplace=True)

Jason Brownlee May 14, 2017 at 7:28 am #

REPLY

Great tip! Thanks for sharing Antonio.

Antonio May 14, 2017 at 10:17 pm #

REPLY

I think that it’s better if you try with more data. I don’t beleive 141 samples are good enoght to train a model… I just tried with another dataset with +3K samples and it worked really good (small RMSE) without overfitting the NN. Anyhow, these are really good posts, it helped to to get started with keras and LSTM from zero to hero in in few mintues. Just wonder if you have another post describing in details how to build the keras model…

Jason Brownlee May 15, 2017 at 5:52 am #

Thanks Antonio, I’m really glad to hear that.

Jason Brownlee May 14, 2017 at 7:27 am #

REPLY

Yes, you can see more about testing strategies for time series / sequence prediction here:
http://machinelearningmastery.com/backtest-machine-learning-models-time-series-foreca

André C. Andersen May 14, 2017 at 1:35 am #

REPLY

Thank you for you contribution. However, I'm sorry to say it, but this post doesn't seem to show a successful LSTM model. A RMSE of 56.35 is worse than if you simply pick the input value as your prediction, which gives you a RMSE of 48.66. You can check it by running:

testX_ = scaler.inverse_transform(testX.reshape(1,-1))
print("Untrained:", numpy.sqrt(mean_squared_error(testX_.flatten(), testY.flatten())))
print("Trained:", numpy.sqrt(mean_squared_error(testPredict.flatten(), testY.flatten())))

This is hinted at in the graphs as well, where you can see your predictions goes in one-lag lockstep with the inputs, they should overlap exactly. Remember, a good time series prediction has the ability to predict change.

Jason Brownlee May 14, 2017 at 7:30 am #

REPLY

I agree, see this post for a better approach:
http://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

Xiangpeng Wan May 15, 2017 at 12:13 pm #

REPLY

Hi, thanks for you Blog, that help me a lot in my project and understanding LSTM, by the way, I notice that in creat_dataset function, there is no need to minus one in i's range. Or you will loose one element.

Jason Brownlee May 16, 2017 at 8:33 am #

REPLY

Thanks.

Justin Jones May 20, 2017 at 1:37 am #

REPLY

Jason:
Thank you so much for the great tutorial. Your blog has really taken my skills to the next level as working with Theano was very challenging in the past.

Quick question:
How can I modify you code to take in sequences of 2 real valued numbers (x, y coordinates) and with a look_back window of 3 to predict the time at t+1. The output should be x, y coordinates.

Here is the code I am using, though I am getting an error:
Error when checking model input: expected simple_rnn_1_input to have shape (None, None, 3) but got array with shape (1470, 3, 2)

Here is the main areas I have changed:
-pandas.read_csv reads in 2 columns of data
-Create dataset creates sets that are num_samples x time_steps x features.
-I am using the regression with time steps sample.
-Change Dense(1) to Dense(2) since expecting 2 outputs.

Jason:
Thank you so much for the great tutorial. Your blog has really taken my skills to the next level as working with Theano was very challenging in the past.

Quick question:
How can I modify you code to take in sequences of 2 real valued numbers (x, y coordinates) and with a look_back window of 3 to predict the time at t+1. The output should be length 2 for the regression of x, y coordinates of the prediction.

Here is the code I am using, though I am getting an error:
Error when checking model input: expected simple_rnn_1_input to have shape (None, None,

Here is the main areas I have changed:
-pandas.read_csv reads in 2 columns of data
-Create dataset creates sets that are num_samples x time_steps x features.
-I am using the regression with time steps sample.

# SimpleRNN that learns on a x, y
import numpy
import matplotlib.pyplot as plt
import pandas
import math

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
dataX, dataY = [], []
for i in range(len(dataset)-look_back-1):
a = dataset[i:(i+look_back), 0]
dataX.append(a)
dataY.append(dataset[i + look_back, 0])
return numpy.array(dataX), numpy.array(dataY)

def create_dataset2(dataset, look_back=1):
dataX, dataY = [], []
for i in range(len(dataset)-look_back-1): # go from 0 to dataset length – lookback +1
features=[]
set_of_features=[]
a = dataset[i:(i+look_back), 0] # slice dataset from i to i+ lookback, column 0 (which is x)
b = dataset[i:(i+look_back), 1] # slice dataset from i to i+ lookback, column 1 (which is y)
for X1, X2 in zip(a, b): # create tuples, the list will be tuples of length lookback
features=[X1,X2]
set_of_features.append(features)

dataX.append(set_of_features)
# now do the target (should be a tuple right after the lookback)
dataY.append([dataset[i + look_back, 0], dataset[i + look_back, 1]])
return numpy.array(dataX), numpy.array(dataY)

# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = pandas.read_csv('spiral.csv', usecols=[1,2], engine='python', skiprows=0, delimiter='\t') # tab delimited here, use the 2 columns
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) – train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset2(train, look_back)
testX, testY = create_dataset2(test, look_back)
# reshape input to be [samples, time steps, features]
#trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
#testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], trainX.shape[2]))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], testX.shape[2]))

# create and fit the RNN network
model = Sequential()
model.add(SimpleRNN(4, input_dim=look_back))
model.add(Dense(2)) # was 1
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
```

```
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Jason Brownlee May 20, 2017 at 5:39 am #

REPLY

I have a post on multivariate forecasting with LSTMs scheduled on the blog for a few weeks time.

Justin Jones May 20, 2017 at 2:19 am #

REPLY

Hi Jason:
Can you tell me how to modify the code to take in sequences of x,y coordinates (real values) and output the x, y coordinate at t+1?

Thank you for your time.

Jason Brownlee May 20, 2017 at 5:40 am #

REPLY

Frame the problem:
http://mlmastery.staging.wpengine.com/convert-time-series-supervised-learning-problem-python/

Then ensure you define your LSTM with 2 inputs on the input layer and 2 outputs on the output layer.

View May 21, 2017 at 7:51 am #

REPLY

Hi. Great tutorial. Need help in adapting this to my specific problem. I have 2d data set of the dimension 100*30. where 100 is number of different company stocks and 30 is the stock price of each company for 30 consecutive days. Please help in understaning the input dimension of the LSTM in keras for predicting the stock price of 100 different companies in next 10 days.

Thanks

Jason Brownlee May 22, 2017 at 7:50 am #

REPLY

The expected input structure is [samples, timesteps, features].

Each series is a different feature. Time steps are the ticks or movements. You may need to split each series into sub-sequences, e.g. 200-400 timesteps long. In that case, each sub-sequence will be a sample.

I hope that helps.

Ethels May 23, 2017 at 9:56 pm #

Hello Jason,

I discovered that in http://machinelearningmastery.com/time-series-forecasting-long-short-term
of differencing, yet the example has seasonal and trend variations. Is it ok to just model your L
without differencing, what is the implications?

Thank you very much

Jason Brownlee May 24, 2017 at 4:55 am #

REPLY

No, it is best to difference the data and in this case also use a log transform.

ismail May 27, 2017 at 2:07 am #

REPLY

Hi Jason,

That is really amazing work, but is all the cases you showed, the data is always numeric. I have regression problem (label is a float) but I have multi type data (strings and numerics). How can I use string properly instead of mapping them a random integer ?
Thank you

Jason Brownlee June 2, 2017 at 11:54 am #

REPLY

Strings will have to be mapped to integers (chars or words), sorry.

Luis May 28, 2017 at 12:03 am #

REPLY

Hi, Jason.

I see that LSTM is a powerful model which can get better results than classic time series models like ARIMA in some circumstances, but… is it possible to have something similar to confidence intervals for the predictions? is only possible to obtain point forecasts?

Jason Brownlee June 2, 2017 at 12:05 pm #

REPLY

Good question, I'm not sure off hand.

Perhaps you could use the bootstrap method with multiple repeats of fitting the model and making predictions?

Chester May 30, 2017 at 2:10 am #

REPLY

Hi Dr.Brownlee,

Thanks for the tutorial!

I am struggling to understand the shifting done on the predicted data, namely why do we shift by that particular amount?

Thank you!

hutauf May 30, 2017 at 7:43 am #

REPLY

Your plot looks like your prediction for both test and train data is lacking 1 datapoint behind – which is exactly what is known to the network. A "predictor" that would just return the input, so lets assume this function:
def predict(x):
return x
would also "follow" the curve, but is not very intelligent. A better prediction would be to use the prediction of t=t0 as an input to the prediction at t=t0+1, then use this output as input for t=t0+2 and so on. Just like what you did here:
http://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-ke

I hope you could grasp what I mean. Thanks for your thoughts.

Roshan Silva May 31, 2017 at 4:10 am #

REPLY

Awesome post!!!!

I'm trying to apply this for a dengue prediction system. The problem is to determine number of
prediction), given the attributes (humidity, temperature for that week).
I set trainX to input features and trainY to number of dengue patients.
So the trainX has the format (attribute1, attribute2, .. attributen) whereas trainY is a vector cor
The model predicts some output, but the mean square error is very high.

Is there any way to improve?

and in dengue prediction, we should relay on time steps since number of dengue patients in a given week depends on number dengue patients in the previous week. But the problem is unlike the problem in this post, I don't have the actual number of patients for a given week for the test data set. In this case, how can I use "LSTM for Regression with Time Steps" for my problem?

Thanks in advance

Jason Brownlee June 2, 2017 at 12:40 pm #

REPLY

This post may give you some ideas:
http://machinelearningmastery.com/improve-deep-learning-performance/

Ammar Sohail May 31, 2017 at 4:43 pm #

REPLY

Hi Jason,

You are amazing.

I am very new to machine learning and Keras. I have one task to be completed. In it, my input is vector of size 19 (features) and I have 2000 samples (rows of data) and I would want to predict single output value. I am little bit confused in how to create and fit LSTM network that is suitable for this scenario.
What would be the parameters to the LSTM layer while adding it to the model like, units, input_shape?
Basically its a time series regression analysis using LSTMs.

Can you please provide me with some help.

Many thanks

Jason Brownlee June 2, 2017 at 12:45 pm #

REPLY

The input shape is [samples, timesteps, features], your number of features is 19.

Sara June 8, 2017 at 12:27 pm #

REPLY

Hi Jason

Thanks for your beautiful article.I tried this with some other data set(it has some negative values too), the result was excellent but the only problem was in predicting negative part!I mean in some place at future the line is going below x-axis(negative part) in actual graphs but when the algorithm tries to predict that it goes until x=0 and then return to the positive area of the y-axis.Could you help me why this happens?

Many thanks

sara June 9, 2017 at 4:49 am #

REPLY

And also I saw this question (a lot of time) above in the article but I could not find any answer for that.What should we do if we want to predict more step in the future like future 2-month prediction?If there is any other tutorial about this, please let me know.

Jason Brownlee June 9, 2017 at 6:31 am #

REPLY

Call model.predict()

You can better understand how to make predictions with LSTMs here:
http://machinelearningmastery.com/5-step-life-cycle-long-short-term-memory-models-

Jason Brownlee June 9, 2017 at 6:16 am #

Consider scaling your data first to the range 0-1.

Goran Alioz June 9, 2017 at 2:43 am #

REPLY

HI Jason and All,
Sorry if my question is too basic, how can I separate the training from the test please? I need to run them separately but failed to understand what output comes out of the training and what I need to feed in to the predict function in another file? Any Clues? I mean if it's possible in the first place of course.
Many thanks,
Goran

Jason Brownlee June 9, 2017 at 6:28 am #

REPLY

See this post for a tutorial:
http://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

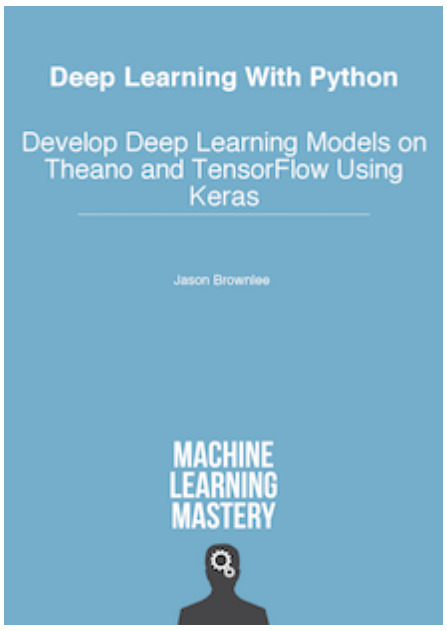
Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Finally Get Started With Deep Learning

Sick of the fancy math and need for super computers?
Looking for step-by-step tutorials?
Want end-to-end projects?

Get Started With Deep Learning in Python Too



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

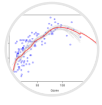

Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016

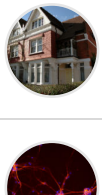

Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016


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


How to Run Your First Classifier in Weka
FEBRUARY 17, 2014


Tutorial To Implement k-Nearest Neighbors in Python From Scratch
SEPTEMBER 12, 2014


A Tour of Machine Learning Algorithms
NOVEMBER 25, 2013


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

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

Privacy | Contact | About