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

The 5 Step Life-Cycle for Long Short-Term Memory Models in Keras

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

Deep learning neural networks are very easy to create and evaluate in Python with Keras, but you must follow a strict model life-cycle.

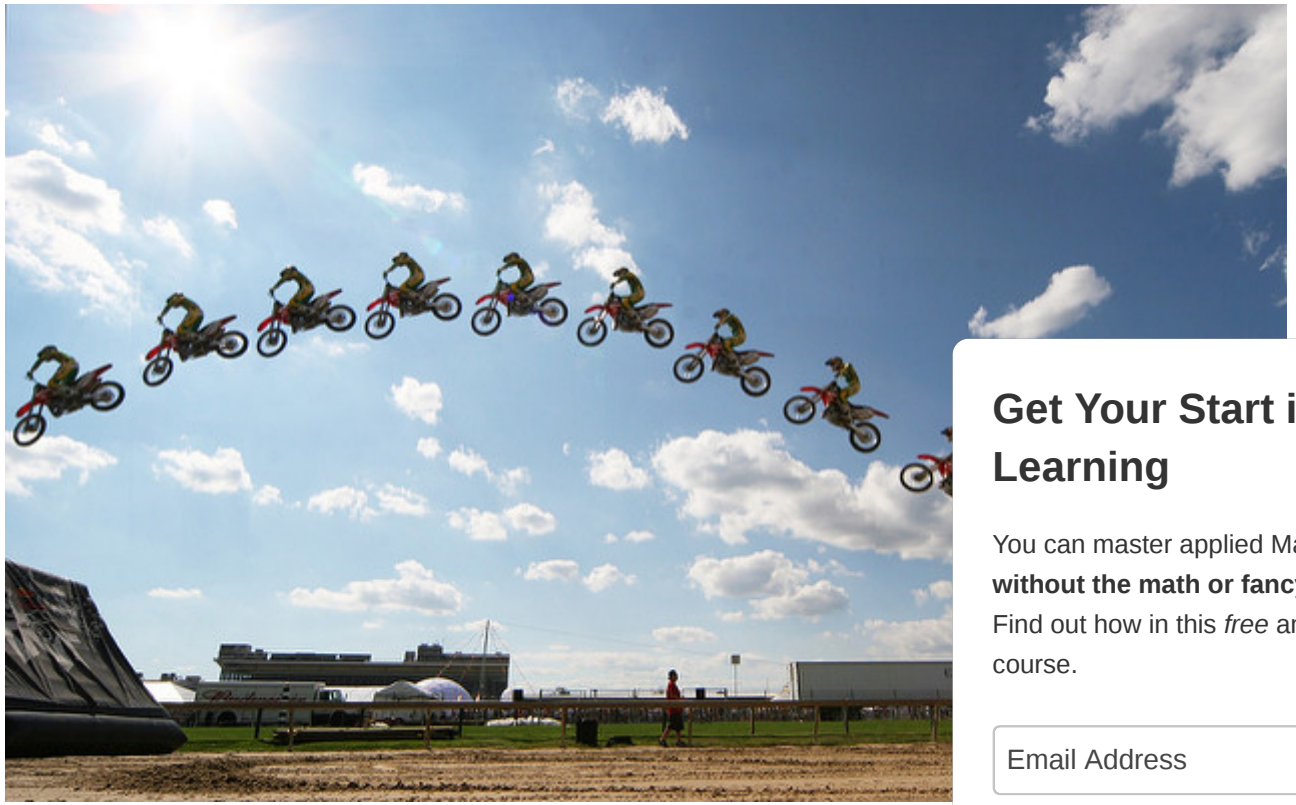
In this post, you will discover the step-by-step life-cycle for creating, training, and evaluating Long Short-Term Memory (LSTM) Recurrent Neural Networks in Keras and how to make predictions with a trained model.

After reading this post, you will know:

- How to define, compile, fit, and evaluate an LSTM in Keras.
- How to select standard defaults for regression and classification sequence prediction problems.
- How to tie it all together to develop and run your first LSTM recurrent neural network in Keras.

Let's get started.

- **Update June/2017**: Fixed typo in input resizing example.



The 5 Step Life-Cycle for Long Short-Term Memory Models in K
Photo by docmonstereyes, some rights reserved.

Overview

Below is an overview of the 5 steps in the LSTM model life-cycle in Keras that we are going to look at.

1. Define Network
2. Compile Network
3. Fit Network

Get Your Start in Machine Learning

4. Evaluate Network
5. Make Predictions

Environment

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

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

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

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

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

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

Step 1. Define Network

The first step is to define your network.

Neural networks are defined in Keras as a sequence of layers. The container for these layers is the

Get Your Start in Machine Learning

The first step is to create an instance of the Sequential class. Then you can create your layers and add them in the order that they should be connected. The LSTM recurrent layer comprised of memory units is called LSTM(). A fully connected layer that often follows LSTM layers and is used for outputting a prediction is called Dense().

For example, we can do this in two steps:

```
model = Sequential()
model.add(LSTM(2))
model.add(Dense(1))
```

But we can also do this in one step by creating an array of layers and passing it to the constructor of the Sequential.

```
layers = [LSTM(2), Dense(1)]
model = Sequential(layers)
```

The first layer in the network must define the number of inputs to expect. Input must be three-dimens
features.

- **Samples**. These are the rows in your data.
- **Timesteps**. These are the past observations for a feature, such as lag variables.
- **Features**. These are columns in your data.

Assuming your data is loaded as a NumPy array, you can convert a 2D dataset to a 3D dataset usin
columns to become timesteps for one feature, you can use:

```
data = data.reshape((data.shape[0], data.shape[1], 1))
```

If you would like columns in your 2D data to become features with one timestep, you can use:

```
data = data.reshape((data.shape[0], 1, data.shape[1]))
```

You can specify the input_shape argument that expects a tuple containing the number of timesteps and the number of features. For example, if we had two timesteps and one feature for a univariate time series with two lag observations per row, it would be specified as follows:

```
model = Sequential()
model.add(LSTM(5, input_shape=(2,1)))
model.add(Dense(1))
```


Get Your Start in Machine Learning

LSTM layers can be stacked by adding them to the Sequential model. Importantly, when stacking LSTM layers, we must output a sequence rather than a single value for each input so that the subsequent LSTM layer can have the required 3D input. We can do this by setting the return_sequences argument to True. For example:

```
model = Sequential()
model.add(LSTM(5, input_shape=(2,1), return_sequences=True))
model.add(LSTM(5))
model.add(Dense(1))
```

Think of a Sequential model as a pipeline with your raw data fed in at in end and predictions that come out at the other.

This is a helpful container in Keras as concerns that were traditionally associated with a layer can also be split out and added as separate layers, clearly showing their role in the transform of data from input to prediction.

For example, activation functions that transform a summed signal from each neuron in a layer can b[redacted]r-like object called Activation.

```
model = Sequential()
model.add(LSTM(5, input_shape=(2,1)))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

The choice of activation function is most important for the output layer as it will define the format that[redacted]

For example, below are some common predictive modeling problem types and the structure and sta[redacted] output layer:

- **Regression**: Linear activation function, or 'linear', and the number of neurons matching the nu[redacted]
- **Binary Classification (2 class)**: Logistic activation function, or 'sigmoid', and one neuron the o[redacted]
- **Multiclass Classification (>2 class)**: Softmax activation function, or 'softmax', and one output neuron per class value, assuming a one-hot encoded output pattern.

Step 2. Compile Network

Once we have defined our network, we must compile it.

Get Your Start in Machine Learning

Compilation is an efficiency step. It transforms the simple sequence of layers that we defined into a highly efficient series of matrix transforms in a format intended to be executed on your GPU or CPU, depending on how Keras is configured.

Think of compilation as a precompute step for your network. It is always required after defining a model.

Compilation requires a number of parameters to be specified, specifically tailored to training your network. Specifically, the optimization algorithm to use to train the network and the loss function used to evaluate the network that is minimized by the optimization algorithm.

For example, below is a case of compiling a defined model and specifying the stochastic gradient descent (sgd) optimization algorithm and the mean squared error (mean_squared_error) loss function, intended for a regression type problem.

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

Alternately, the optimizer can be created and configured before being provided as an argument to th

```
algorithm = SGD(lr=0.1, momentum=0.3)
model.compile(optimizer=algorithm, loss='mean_squared_error')
```

The type of predictive modeling problem imposes constraints on the type of loss function that can be

For example, below are some standard loss functions for different predictive model types:

- **Regression**: Mean Squared Error or 'mean_squared_error'.
- **Binary Classification (2 class)**: Logarithmic Loss, also called cross entropy or 'binary_crossen
- **Multiclass Classification (>2 class)**: Multiclass Logarithmic Loss or 'categorical_crossentropy'

The most common optimization algorithm is stochastic gradient descent, but Keras also supports a s ns that work well with little or no configuration.

Perhaps the most commonly used optimization algorithms because of their generally better performance are:

- **Stochastic Gradient Descent**, or 'sgd', that requires the tuning of a learning rate and momentum.
- **ADAM**, or 'adam', that requires the tuning of learning rate.
- **RMSprop**, or 'rmsprop', that requires the tuning of learning rate.

Finally, you can also specify metrics to collect while fitting your model in addition to the loss function. Generally, the most useful additional metric to collect is accuracy for classification problems. The metrics to collect are specified by name in an array.

For example:

```
model.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])
```

Step 3. Fit Network

Once the network is compiled, it can be fit, which means adapt the weights on a training dataset.

Fitting the network requires the training data to be specified, both a matrix of input patterns, X, and an array of matching output patterns, y.

The network is trained using the backpropagation algorithm and optimized according to the optimiza
compiling the model.

The backpropagation algorithm requires that the network be trained for a specified number of epoch

Each epoch can be partitioned into groups of input-output pattern pairs called batches. This defines
before the weights are updated within an epoch. It is also an efficiency optimization, ensuring that n
a time.

A minimal example of fitting a network is as follows:

```
history = model.fit(X, y, batch_size=10, epochs=100)
```

Once fit, a history object is returned that provides a summary of the performance of the model during
additional metrics specified when compiling the model, recorded each epoch.

Training can take a long time, from seconds to hours to days depending on the size of the network and the size of the training data.

By default, a progress bar is displayed on the command line for each epoch. This may create too much noise for you, or may cause problems for your environment, such as if you are in an interactive notebook or IDE.

You can reduce the amount of information displayed to just the loss each epoch by setting the verbose argument to 2. You can turn off all output by setting verbose to 1. For example:

```
history = model.fit(X, y, batch_size=10, epochs=100, verbose=0)
```

Step 4. Evaluate Network

Once the network is trained, it can be evaluated.

The network can be evaluated on the training data, but this will not provide a useful indication of the performance of the network as a predictive model, as it has seen all of this data before.

We can evaluate the performance of the network on a separate dataset, unseen during testing. This network at making predictions for unseen data in the future.

The model evaluates the loss across all of the test patterns, as well as any other metrics specified w accuracy. A list of evaluation metrics is returned.

For example, for a model compiled with the accuracy metric, we could evaluate it on a new dataset a

```
loss, accuracy = model.evaluate(X, y)
```

As with fitting the network, verbose output is provided to give an idea of the progress of evaluating th se argument to 0.

```
loss, accuracy = model.evaluate(X, y, verbose=0)
```

Step 5. Make Predictions

Once we are satisfied with the performance of our fit model, we can use it to make predictions on new data.

This is as easy as calling the predict() function on the model with an array of new input patterns.

For example:

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

The predictions will be returned in the format provided by the output layer of the network.

In the case of a regression problem, these predictions may be in the format of the problem directly, provided by a linear activation function.

For a binary classification problem, the predictions may be an array of probabilities for the first class that can be converted to a 1 or 0 by rounding.

For a multiclass classification problem, the results may be in the form of an array of probabilities (assuming a one hot encoded output variable) that may need to be converted to a single class output prediction using the argmax() NumPy function.

Alternately, for classification problems, we can use the predict_classes() function that will automatically convert uncrisp predictions to crisp integer class values.

```
predictions = model.predict_classes(X)
```

As with fitting and evaluating the network, verbose output is provided to given an idea of the progres is off by setting the verbose argument to 0.

```
predictions = model.predict(X, verbose=0)
```


End-to-End Worked Example

Let's tie all of this together with a small worked example.

This example will use a simple problem of learning a sequence of 10 numbers. We will show the net ct 0.1. Then show it 0.1 and expect it to predict 0.2, and so on to 0.9.

1. **Define Network**: We will construct an LSTM neural network with a 1 input timestep and 1 input feature in the visible layer, 10 memory units in the LSTM hidden layer, and 1 neuron in the fully connected output layer with a linear (default) activation function.
2. **Compile Network**: We will use the efficient ADAM optimization algorithm with default configuration and the mean squared error loss function because it is a regression problem.
3. **Fit Network**: We will fit the network for 1,000 epochs and use a batch size equal to the number of patterns in the training set. We will also turn off all verbose output.
4. **Evaluate Network**. We will evaluate the network on the training dataset. Typically we would eva

5. **Make Predictions**. We will make predictions for the training input data. Again, typically we would make predictions on data where we do not know the right answer.

The complete code listing is provided below.

```
# Example of LSTM to learn a sequence
from pandas import DataFrame
from pandas import concat
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
# create sequence
length = 10
sequence = [i/float(length) for i in range(length)]
print(sequence)
# create X/y pairs
df = DataFrame(sequence)
df = concat([df.shift(1), df], axis=1)
df.dropna(inplace=True)
# convert to LSTM friendly format
values = df.values
X, y = values[:, 0], values[:, 1]
X = X.reshape(len(X), 1, 1)
# 1. define network
model = Sequential()
model.add(LSTM(10, input_shape=(1,1)))
model.add(Dense(1))
# 2. compile network
model.compile(optimizer='adam', loss='mean_squared_error')
# 3. fit network
history = model.fit(X, y, epochs=1000, batch_size=len(X), verbose=0)
# 4. evaluate network
loss = model.evaluate(X, y, verbose=0)
print(loss)
# 5. make predictions
predictions = model.predict(X, verbose=0)
print(predictions[:, 0])
```


Running this example produces the following output, showing the raw input sequence of 10 numbers, the mean squared error loss of the network when making predictions for the entire sequence, and the predictions for each input pattern.

Outputs were spaced out for readability.

Get Your Start in Machine Learning

We can see the sequence is learned well, especially if we round predictions to the first decimal place.

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

4.54527471447e-05

[ 0.11612834 0.20493418 0.29793766 0.39445466 0.49376178 0.59512401
0.69782174 0.80117452 0.90455914]
```

Further Reading

- Keras documentation for Sequential Models.
- Keras documentation for LSTM Layers.
- Keras documentation for optimization algorithms.
- Keras documentation for loss functions.

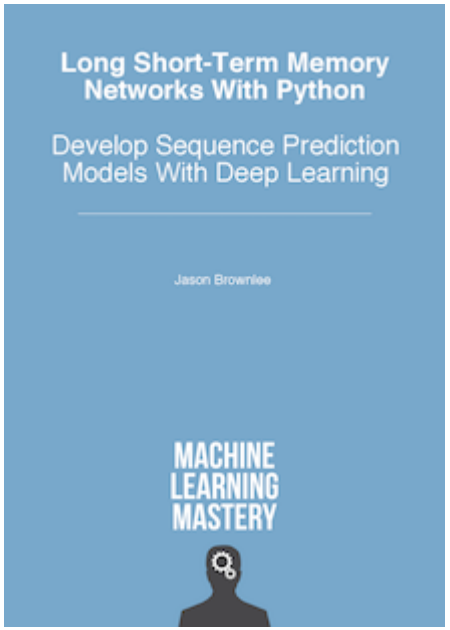
Summary

In this post, you discovered the 5-step life-cycle of an LSTM recurrent neural network using the Kera

Specifically, you learned:

- How to define, compile, fit, evaluate, and make predictions for an LSTM network in Keras.
- How to select activation functions and output layer configurations for classification and regressio
- How to develop and run your first LSTM model in Keras.

Do you have any questions about LSTM models in Keras, or about this post?
Ask your questions in the comments and I will do my best to answer them.



Discover how in my new Ebook:
Long Short-Term Memory Networks with Python

It provides **self-study tutorials** on topics like:
CNN LSTMs, Encoder-Decoder LSTMs, generative models, data preparation, making predictions and much more…

Finally Bring LSTM Recurrent Neural Networks to Your Sequence Predictions Projects

Skip the Academics. Just Results.

Click to learn more.

About Jason Brownlee

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

‹ How to Calculate Bootstrap Confidence Intervals For Machine Learning Results in Python

How to Learn to Echo Random Integers with Long Short-Term Memory Recurrent Neural Networks ›

Get Your Start in Machine Learning

10 Responses to *The 5 Step Life-Cycle for Long Short-Term Memory Models in Keras*

Peter Marelas June 7, 2017 at 7:47 am #

REPLY

Your missing ".shape" in your reshape statements.

Jason Brownlee June 8, 2017 at 7:35 am #

REPLY

Hi Peter, it still seems to work when shape is provided as integers rather than a tuple.

Birkey June 7, 2017 at 5:44 pm #

Nice summarisation!

and I'd like to add:

if timesteps = 2, and keep features as it is, then
– reshape(int(data[0]/2), 2, data[1])

if timesteps = 2, and keep samples as it is, then
– reshape(data[0], 2, int(data[1]/2))

klaas June 8, 2017 at 4:09 am #

REPLY

Thanks Birkey, that means I can have multiple timesteps and features at the same time right?

Get Your Start in Machine Learning

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

REPLY

Correct!

Jason Brownlee June 8, 2017 at 7:39 am #

REPLY

Nice!

Ali June 10, 2017 at 3:05 am #

Hi Jason,

As always, you write in the most excellent way. I really enjoyed it. Thank you.

Have you covered LSTM in Deep Learning Book?

Thanks

Jason Brownlee June 10, 2017 at 8:25 am #

Thanks. Yes, I have a few chapters on LSTMs in that book. I hope to release a new book d

Harsh July 13, 2017 at 4:04 am #

REPLY

Hello,
I have 3 classes and want to design a LSTM for 3-class classification. Any suggestion what I am doig wrong here.
I get the below error :
Input 0 is incompatible with layer lstm_1: expected ndim=3, found ndim=2

Get Your Start in Machine Learning

#below is the snippet from my code
print train_data.shape,train_labels.shape
(1199, 11) (1199, 3)

print test_data.shape, test_labels.shape
(592, 11) (592, 3)

##
model = Sequential()
model.add(LSTM(11,input_shape=(11,),return_sequences=True))
model.add(LSTM(11))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())

Jason Brownlee July 13, 2017 at 10:01 am #

LSTMs require 3D input. Ensure that your input data shape is 3D [samples, time steps, feat
with a tuple of (timesteps, features).

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.

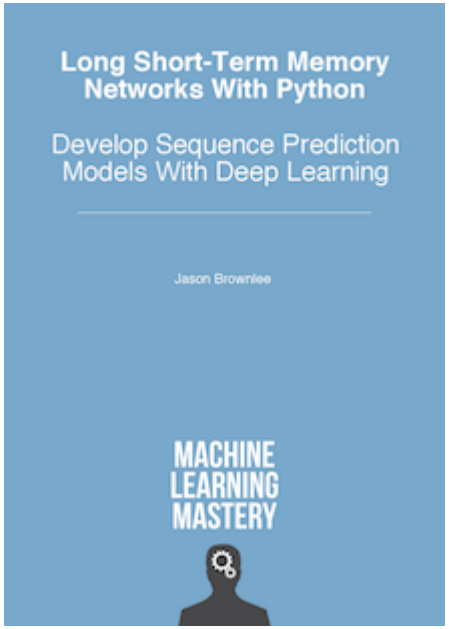
Read More

Deep Learning for Sequence Prediction

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

Get Your Start in Machine Learning

Get Started With LSTMs in Python Today!



Get Your Start in Machine Learning

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

Email Address

START MY EMAIL COURSE

POPULAR


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


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


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


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

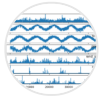
How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

Get Your Start in Machine Learning

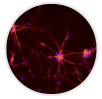
Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

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

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



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



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

Privacy | Contact | About

Get Your Start in Machine Learning