Deep Learning for Time Series Forecasting

7-Day Crash-Course

Jason Brownlee

MACHINE LEARNING MASTERY

Disclaimer

Deep Learning for Time Series Forecasting Crash Course

Edition: v1.4

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started...

Time series forecasting is challenging, especially when working with long sequences, noisy data, multi-step forecasts and multiple input and output variables. Deep learning methods offer a lot of promise for time series forecasting, such as the automatic learning of temporal dependence and the automatic handling of temporal structures like trends and seasonality. In this crash course, you will discover how you can get started and confidently develop deep learning models for time series forecasting problems using Python in 7 days. Let's get started.

Who Is This Crash-Course For?

Before we get started, let's make sure you are in the right place. The list below provides some general guidelines as to who this course was designed for.

You need to know:

- You need to know the basics of time series forecasting.
- You need to know your way around basic Python, NumPy and Keras for deep learning.

You do NOT need to know:

- You do not need to be a math wiz!
- You do not need to be a deep learning expert!
- You do not need to be a time series expert!

This crash course will take you from a developer that knows a little machine learning to a developer who can bring deep learning methods to your own natural language processing project. This crash course assumes you have a working Python 2 or 3 SciPy environment with at least NumPy and Keras 2 installed. If you need help with your environment, you can follow the step-by-step tutorial here:

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

Crash-Course Overview

This crash course is broken down into 7 lessons. You could complete one lesson per day (recommended) or complete all of the lessons in one day (hardcore). It really depends on the time you have available and your level of enthusiasm. Below are 7 lessons that will get you started and productive with deep learning for time series forecasting in Python:

- **Lesson 01**: Promise of Deep Learning.
- **Lesson 02**: How to Transform Data for Time Series.
- **Lesson 03**: MLP for Time Series Forecasting.
- **Lesson 04**: CNN for Time Series Forecasting.
- **Lesson 05**: LSTM for Time Series Forecasting.
- **Lesson 06**: CNN-LSTM for Time Series Forecasting.
- **Lesson 07**: Encoder-Decoder LSTM Multi-step Forecasting.

Each lesson could take you 60 seconds or up to 30 minutes. Take your time and complete the lessons at your own pace. Ask questions and even put results online and share your results. The lessons expect you to go off and find out how to do things. I will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help on and about the deep learning, time series forecasting and the best-of-breed tools in Python (hint, I have all of the answers directly on this blog, use the search box). I do provide more help in the form of links to related material because I want you to build up some confidence and inertia. Post your results online, I'll cheer you on!

Hang in there, don't give up!

Lesson 01: Promise of Deep Learning

In this lesson, you will discover the promise of deep learning methods for time series forecasting. Generally, neural networks like Multilayer Perceptrons or MLPs provide capabilities that are offered by few algorithms, such as:

- **Robust to Noise**. Neural networks are robust to noise in input data and in the mapping function and can even support learning and prediction in the presence of missing values.
- **Nonlinear**. Neural networks do not make strong assumptions about the mapping function and readily learn linear and nonlinear relationships.
- **Multivariate Inputs**. An arbitrary number of input features can be specified, providing direct support for multivariate forecasting.
- **Multi-step Forecasts**. An arbitrary number of output values can be specified, providing direct support for multi-step and even multivariate forecasting.

For these capabilities alone, feedforward neural networks may be useful for time series forecasting.

Your Task

For this lesson you must suggest one capability from both Convolutional Neural Networks and Recurrent Neural Networks that may be beneficial in modeling time series forecasting problems. Post your answer online. I would love to see what you discover.

More Information

- The Promise of Recurrent Neural Networks for Time Series Forecasting.

In the next lesson, you will discover how to transform time series data for time series forecasting.

Lesson 02: How to Transform Data for Time Series

In this lesson, you will discover how to transform your time series data into a supervised learning format. The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (X) and an output variable (y) and you use an algorithm to learn the mapping function from the input to the output. The goal is to approximate the real underlying mapping so well that when you have new input data, you can predict the output variables for that data. Time series data can be phrased as supervised learning. Given a sequence of numbers for a time series dataset, we can restructure the data to look like a supervised learning problem. We can do this by using previous time steps as input variables and use the next time step as the output variable. For example, the series:

```
1, 2, 3, 4, 5, ...
```

Listing 1: Example of a time series.

Can be transformed into samples with input and output components that can be used as part of a training set to train a supervised learning model like a deep learning neural network.

```
X              y
[1, 2, 3]     [4]
[2, 3, 4]     [5]
...
```

Listing 2: Example of a time series transformed into samples.

This is called a sliding window transformation as it is just like sliding a window across prior observations that are used as inputs to the model in order to predict the next value in the series. In this case the window width is 3 time steps.

Your Task

For this lesson you must develop Python code to transform the daily female births dataset into a supervised learning format with some number of inputs and one output. You can download the dataset from here: `daily-total-female-births.csv` Post your code online. I would love to see what you can come up with.

More Information

- Time Series Forecasting as Supervised Learning.

- How to Convert a Time Series to a Supervised Learning Problem in Python.
- How to Prepare Univariate Time Series Data for Long Short-Term Memory Networks.

In the next lesson, you will discover how to develop a Multilayer Perceptron deep learning model for forecasting a univariate time series.

Lesson 03: MLP for Time Series Forecasting

In this lesson, you will discover how to develop a Multilayer Perceptron model or MLP for univariate time series forecasting. We can define a simple univariate problem as a sequence of integers, fit the model on this sequence and have the model predict the next value in the sequence. We will frame the problem to have 3 inputs and 1 output, for example: `[10, 20, 30]` as input and `[40]` as output.

First, we can define the model. We will define the number of input time steps as 3 via the `input_dim` argument on the first hidden layer. In this case we will use the efficient Adam version of stochastic gradient descent and optimizes the mean squared error (`'mse'`) loss function. Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction. The complete example is listed below.

```
# univariate mlp example
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
# define dataset
X = array([[10, 20, 30], [20, 30, 40], [30, 40, 50], [40, 50, 60]])
y = array([40, 50, 60, 70])
# define model
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=3))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=2000, verbose=0)
# demonstrate prediction
x_input = array([50, 60, 70])
x_input = x_input.reshape((1, 3))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Listing 3: Example of an MLP for univariate time series forecasting.

Running the example will fit the model on the data then predict the next out-of-sample value. Given `[50, 60, 70]` as input, the model correctly predicts `80` as the next value in the sequence.

Your Task

For this lesson you must download the daily female births dataset, split it into train and test sets and develop a model that can make reasonably accurate predictions on the test set. You can download the dataset from here: You can download the dataset from here: `daily-total-female-births.csv` Post your code online. I would love to see what you can come up with.

More Information

- Crash Course On Multilayer Perceptron Neural Networks.
- Time Series Prediction With Deep Learning in Keras.
- Exploratory Configuration of a Multilayer Perceptron Network for Time Series Forecasting.

In the next lesson, you will discover how to develop a Convolutional Neural Network model for forecasting a univariate time series.

Lesson 04: CNN for Time Series Forecasting

In this lesson, you will discover how to develop a Convolutional Neural Network model or CNN for univariate time series forecasting. We can define a simple univariate problem as a sequence of integers, fit the model on this sequence and have the model predict the next value in the sequence. We will frame the problem to have 3 inputs and 1 output, for example: `[10, 20, 30]` as input and `[40]` as output.

An important difference from the MLP model is that the CNN model expects three-dimensional input with the shape `[samples, timesteps, features]`. We will define the data in the form `[samples, timesteps]` and reshape it accordingly. We will define the number of input time steps as 3 and the number of features as 1 via the `input_shape` argument on the first hidden layer. We will use one convolutional hidden layer followed by a max pooling layer. The filter maps are then flattened before being interpreted by a `Dense` layer and outputting a prediction.

The model uses the efficient Adam version of stochastic gradient descent and optimizes the mean squared error (`'mse'`) loss function. Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction. The complete example is listed below.

```
# univariate cnn example
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
# define dataset
X = array([[10, 20, 30], [20, 30, 40], [30, 40, 50], [40, 50, 60]])
y = array([40, 50, 60, 70])
# reshape from [samples, timesteps] into [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(3, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=1000, verbose=0)
```

```
# demonstrate prediction
x_input = array([50, 60, 70])
x_input = x_input.reshape((1, 3, 1))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Listing 4: Example of a CNN for univariate time series forecasting.

Running the example will fit the model on the data then predict the next out-of-sample value. Given `[50, 60, 70]` as input, the model correctly predicts `80` as the next value in the sequence.

Your Task

For this lesson you must download the daily female births dataset, split it into train and test sets and develop a model that can make reasonably accurate predictions on the test set. You can download the dataset from here: You can download the dataset from here: `daily-total-female-births.csv` Post your code online. I would love to see what you can come up with.

More Information

- Crash Course in Convolutional Neural Networks for Machine Learning.

In the next lesson, you will discover how to develop a Long Short-Term Memory network model for forecasting a univariate time series.

Lesson 05: LSTM for Time Series Forecasting

In this lesson, you will discover how to develop a Long Short-Term Memory Neural Network model or LSTM for univariate time series forecasting. We can define a simple univariate problem as a sequence of integers, fit the model on this sequence and have the model predict the next value in the sequence. We will frame the problem to have 3 inputs and 1 output, for example: `[10, 20, 30]` as input and `[40]` as output.

An important difference from the MLP model, and like the CNN model, is that the LSTM model expects three-dimensional input with the shape `[samples, timesteps, features]`. We will define the data in the form `[samples, timesteps]` and reshape it accordingly. We will define the number of input time steps as 3 and the number of features as 1 via the `input_shape` argument on the first hidden layer. We will use one `LSTM` layer to process each input sub-sequence of 3 time steps, followed by a `Dense` layer to interpret the summary of the input sequence.

The model uses the efficient Adam version of stochastic gradient descent and optimizes the mean squared error (`'mse'`) loss function. Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction. The complete example is listed below.

```
# univariate lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
# define dataset
X = array([[10, 20, 30], [20, 30, 40], [30, 40, 50], [40, 50, 60]])
y = array([40, 50, 60, 70])
# reshape from [samples, timesteps] into [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=1000, verbose=0)
# demonstrate prediction
x_input = array([50, 60, 70])
x_input = x_input.reshape((1, 3, 1))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Listing 5: Example of a LSTM for univariate time series forecasting.

Running the example will fit the model on the data then predict the next out-of-sample value. Given `[50, 60, 70]` as input, the model correctly predicts `80` as the next value in the sequence.

Your Task

For this lesson you must download the daily female births dataset, split it into train and test sets and develop a model that can make reasonably accurate predictions on the test set. You can download the dataset from here: You can download the dataset from here: `daily-total-female-births.csv` Post your code online. I would love to see what you can come up with.

More Information

- A Gentle Introduction to Long Short-Term Memory Networks by the Experts.
- Crash Course in Recurrent Neural Networks for Deep Learning.

In the next lesson, you will discover how to develop a hybrid CNN-LSTM model for a univariate time series forecasting problem.

Lesson 06: CNN-LSTM for Time Series Forecasting

In this lesson, you will discover how to develop a hybrid CNN-LSTM model for univariate time series forecasting. The benefit of this model is that the model can support very long input sequences that can be read as blocks or subsequences by the CNN model, then pieced together by the LSTM model. We can define a simple univariate problem as a sequence of integers, fit the model on this sequence and have the model predict the next value in the sequence. We will frame the problem to have 4 inputs and 1 output, for example: `[10, 20, 30, 40]` as input and `[50]` as output.

When using a hybrid CNN-LSTM model, we will further divide each sample into further subsequences. The CNN model will interpret each sub-sequence and the LSTM will piece together the interpretations from the subsequences. As such, we will split each sample into 2 subsequences of 2 times per subsequence. The CNN will be defined to expect 2 time steps per subsequence with one feature. The entire CNN model is then wrapped in `TimeDistributed` wrapper layers so that it can be applied to each subsequence in the sample. The results are then interpreted by the `LSTM` layer before the model outputs a prediction.

The model uses the efficient Adam version of stochastic gradient descent and optimizes the mean squared error (`'mse'`) loss function. Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction. The complete example is listed below.

```
# univariate cnn-lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import TimeDistributed
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
# define dataset
X = array([[10, 20, 30, 40], [20, 30, 40, 50], [30, 40, 50, 60], [40, 50, 60, 70]])
y = array([50, 60, 70, 80])
# reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
X = X.reshape((X.shape[0], 2, 2, 1))
# define model
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'),
    input_shape=(None, 2, 1)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
```

```
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=500, verbose=0)
# demonstrate prediction
x_input = array([50, 60, 70, 80])
x_input = x_input.reshape((1, 2, 2, 1))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Listing 6: Example of a CNN-LSTM for univariate time series forecasting.

Running the example will fit the model on the data then predict the next out-of-sample value. Given `[50, 60, 70, 80]` as input, the model correctly predicts `90` as the next value in the sequence.

Your Task

For this lesson you must download the daily female births dataset, split it into train and test sets and develop a model that can make reasonably accurate predictions on the test set. You can download the dataset from here: You can download the dataset from here: `daily-total-female-births.csv` Post your code online. I would love to see what you can come up with.

More Information

- CNN Long Short-Term Memory Networks.
- How to Use the `TimeDistributed` Layer for Long Short-Term Memory Networks in Python.

In the next lesson, you will discover how to develop an Encoder-Decoder LSTM network model for multi-step time series forecasting.

Lesson 07: Encoder-Decoder LSTM Multi-step Forecasting

In this lesson, you will discover how to develop an Encoder-Decoder LSTM Network model for multi-step time series forecasting. We can define a simple univariate problem as a sequence of integers, fit the model on this sequence and have the model predict the next two values in the sequence. We will frame the problem to have 3 inputs and 2 outputs, for example: `[10, 20, 30]` as input and `[40, 50]` as output.

The LSTM model expects three-dimensional input with the shape `[samples, timesteps, features]`. We will define the data in the form `[samples, timesteps]` and reshape it accordingly. The output must also be shaped this way when using the Encoder-Decoder model. We will define the number of input time steps as 3 and the number of features as 1 via the `input_shape` argument on the first hidden layer. We will define an LSTM encoder to read and encode the input sequences of 3 time steps. The encoded sequence will be repeated 2 times by the model for the two output time steps required by the model using a `RepeatVector` layer. These will be fed to a decoder `LSTM` layer before using a `Dense` output layer wrapped in a `TimeDistributed` layer that will produce one output for each step in the output sequence.

The model uses the efficient Adam version of stochastic gradient descent and optimizes the mean squared error (`'mse'`) loss function. Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction. The complete example is listed below.

```
# multi-step encoder-decoder lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
# define dataset
X = array([[10, 20, 30], [20, 30, 40], [30, 40, 50], [40, 50, 60]])
y = array([[40,50],[50,60],[60,70],[70,80]])
# reshape from [samples, timesteps] into [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))
y = y.reshape((y.shape[0], y.shape[1], 1))
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(3, 1)))
model.add(RepeatVector(2))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
```

```
# fit model
model.fit(X, y, epochs=100, verbose=0)
# demonstrate prediction
x_input = array([50, 60, 70])
x_input = x_input.reshape((1, 3, 1))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Listing 7: Example of an Encoder-Decoder LSTM for multi-step time series forecasting.

Running the example will fit the model on the data then predict the next two out-of-sample values. Given `[50, 60, 70]` as input, the model correctly predicts `[80, 90]` as the next two values in the sequence.

Your Task

For this lesson you must download the daily female births dataset, split it into train and test sets and develop a model that can make reasonably accurate predictions on the test set. You can download the dataset from here: You can download the dataset from here: `daily-total-female-births.csv` Post your code online. I would love to see what you can come up with.

More Information

- Encoder-Decoder Long Short-Term Memory Networks.
- 4 Strategies for Multi-step Time Series Forecasting.
- Multi-step Time Series Forecasting with Long Short-Term Memory Networks in Python.

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come. You discovered:

- The promise of deep learning neural networks for time series forecasting problems.
- How to transform a time series dataset into a supervised learning problem.
- How to develop a Multilayer Perceptron model for a univariate time series forecasting problem.
- How to develop a Convolutional Neural Network model for a univariate time series forecasting problem.
- How to develop a Long Short-Term Memory network model for a univariate time series forecasting problem.
- How to develop a Hybrid CNN-LSTM model for a univariate time series forecasting problem.
- How to develop an Encoder-Decoder LSTM model for a multi-step time series forecasting problem.

This is just the beginning of your journey with deep learning for time series forecasting. Keep practicing and developing your skills. Take the next step and check out my book on *Deep Learning for Time Series Forecasting*.

How Did You Go With The Crash-Course?

Did you enjoy this crash-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

Take the Next Step

Looking for more help with Deep Learning for Time Series Forecasting?

Grab my new book:
Deep Learning for Time Series Forecasting
https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/