Deep Learning With Python

14-Day Mini-Course

Jason Brownlee

MACHINE
LEARNING
MASTERY

Jason Brownlee

Deep Learning With Python

14 Day Mini-Course

Deep Learning With Python

Edition: v1.17

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

Contents

Before We Get Started... 1

Lesson 01: Introduction to Theano 4

Lesson 02: Introduction to TensorFlow 5

Lesson 03: Introduction to Keras 6

Lesson 04: Crash Course in Multilayer Perceptrons 7

Lesson 05: First Neural Net in Keras 8

Lesson 06: Use Keras Models With Scikit-Learn 9

Lesson 07: Plot Model Training History 10

Lesson 08: Save Your Best Model During Training With Checkpointing 11

Lesson 09: Reduce Overfitting With Dropout Regularization 12

Lesson 10: Lift Performance With Learning Rate Schedules 13

Lesson 11: Crash Course in Convolutional Neural Networks 14

Lesson 12: Handwritten Digit Recognition 15

Lesson 13: Object Recognition in Small Photographs 17

Lesson 14: Improve Generalization With Data Augmentation 19

Final Word Before You Go... 21

Before We Get Started...

Deep learning is a fascinating field of study and the techniques are achieving world class results in a range of challenging machine learning problems. It can be hard to get started in deep learning. *Which library should you use and which techniques should you focus on?*

In this 14-part crash course you will discover applied deep learning in Python with the easy to use and powerful Keras library. This mini-course is intended for Python machine learning practitioners that are already comfortable with scikit-learn on the SciPy ecosystem for machine learning. Let's get started.

This is a long and useful guide. You might want to print it out.

Who Is This Mini-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. Don't panic if you don't match these points exactly, you might just need to brush up in one area or another to keep up.

- **Developers that know how to write a little code**. This means that it is not a big deal for you to get things done with Python and know how to setup the SciPy ecosystem on your workstation (a prerequisite). It does not mean your a wizard coder, but it does mean you're not afraid to install packages and write scripts.
- **Developers that know a little machine learning**. This means you know about the basics of machine learning like cross-validation, some algorithms and the bias-variance trade-off. It does not mean that you are a machine learning PhD, just that you know the landmarks or know where to look them up.

This mini-course is not a textbook on Deep Learning. It will take you from a developer that knows a little machine learning in Python to a developer who can get results and bring the power of Deep Learning to your own projects.

Mini-Course Overview (what to expect)

This mini-course is divided into 14 parts. Each lesson was designed to take the average developer about 30 minutes. You might finish some much sooner and other you may choose to go deeper and spend more time. You can complete each part as quickly or as slowly as you like. A comfortable schedule may be to complete one lesson per day over a two week period. Highly recommended. The topics you will cover over the next 14 lessons are as follows:

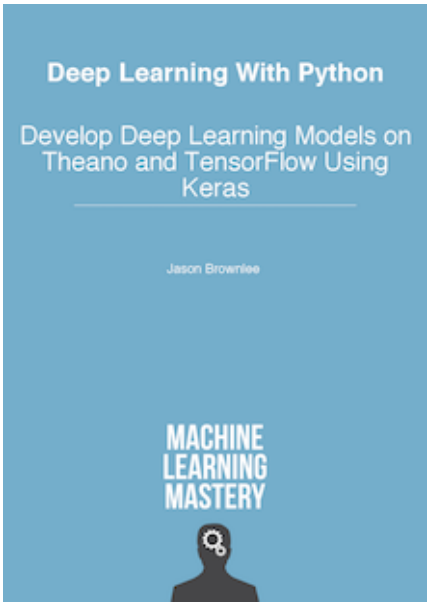
- **Lesson 1**: Introduction to Theano.
- **Lesson 2**: Introduction to TensorFlow.
- **Lesson 3**: Introduction to Keras.
- **Lesson 4**: Crash Course in Multilayer Perceptrons.
- **Lesson 5**: Develop Your First Neural Network in Keras.
- **Lesson 6**: Use Keras Models With Scikit-Learn.
- **Lesson 7**: Plot Model Training History.
- **Lesson 8**: Save Your Best Model During Training With Checkpointing.
- **Lesson 9**: Reduce Overfitting With Dropout Regularization.
- **Lesson 10**: Lift Performance With Learning Rate Schedules.
- **Lesson 11**: Crash Course in Convolutional Neural Networks.
- **Lesson 12**: Handwritten Digit Recognition.
- **Lesson 13**: Object Recognition in Small Photographs.
- **Lesson 14**: Improve Generalization With Data Augmentation.

This is going to be a lot of fun. You're going to have to do some work though, a little reading, a little research and a little programming. You want to learn deep learning right?

Here's a tip: All of the answers these lessons can be found on this blog http://MachineLearningMastery.com. Use the search feature.

Hang in there, don't give up!

If you would like me to step you through each lesson in great detail (and much more), take a look at my book: **Deep Learning With Python:**



Learn more here:
https://machinelearningmastery.com/deep-learning-with-python

Lesson 01: Introduction to Theano

Theano is a Python library for fast numerical computation to aid in the development of deep learning models. At it's heart Theano is a compiler for mathematical expressions in Python. It knows how to take your structures and turn them into very efficient code that uses NumPy and efficient native libraries to run as fast as possible on CPUs or GPUs.

The actual syntax of Theano expressions is symbolic, which can be off-putting to beginners used to normal software development. Specifically, expression are defined in the abstract sense, compiled and later actually used to make calculations. In this lesson your goal is to install Theano and write a small example that demonstrates the symbolic nature of Theano programs. For example, you can install Theano using `pip` as follows:

```
sudo pip install Theano
```

Listing 1: Install Theano with `pip`.

A small example of a Theano program that you can use as a starting point is listed below:

```
import theano
from theano import tensor
# declare two symbolic floating-point scalars
a = tensor.dscalar()
b = tensor.dscalar()
# create a simple expression
c = a + b
# convert the expression into a callable object that takes (a,b)
# values as input and computes a value for c
f = theano.function([a,b], c)
# bind 1.5 to 'a', 2.5 to 'b', and evaluate 'c'
result = f(1.5, 2.5)
print(result)
```

Listing 2: Small Example in Theano.

Learn more about Theano on the Theano homepage[1].

[1] http://deeplearning.net/software/theano/

Lesson 02: Introduction to TensorFlow

TensorFlow is a Python library for fast numerical computing created and released by Google. Like Theano, TensorFlow is intended to be used to develop deep learning models. With the backing of Google, perhaps used in some of it's production systems and used by the Google DeepMind research group, it is a platform that we cannot ignore. Unlike Theano, TensorFlow does have more of a production focus with a capability to run on CPUs, GPUs and even very large clusters.

In this lesson your goal is to install TensorFlow become familiar with the syntax of the symbolic expressions used in TensorFlow programs. For example, you can install TensorFlow using `pip`. There are many different versions of TensorFlow, specialized for each platform. Select the right version for your platform on the TensorFlow installation webpage[2].

```
sudo pip install TensorFlow
```

Listing 3: Install TensorFlow with `pip`.

A small example of a TensorFlow program that you can use as a starting point is listed below:

```
# Example of TensorFlow library
import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
# declare two symbolic floating-point scalars
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
# create a simple symbolic expression using the add function
add = tf.add(a, b)
# bind 1.5 to 'a', 2.5 to 'b', and evaluate 'c'
sess = tf.Session()
binding = {a: 1.5, b: 2.5}
c = sess.run(add, feed_dict=binding)
print(c)
```

Listing 4: Small Example in TensorFlow.

Learn more about TensorFlow on the TensorFlow homepage[3].

[2] https://www.tensorflow.org/versions/r0.9/get_started/os_setup.html
[3] https://www.tensorflow.org/

Lesson 03: Introduction to Keras

A difficulty of both Theano and TensorFlow is that it can take a lot of code to create even very simple neural network models. These libraries were designed primarily as a platform for research and development more than for the practical concerns of applied deep learning. The Keras library addresses these concerns by providing a wrapper for both Theano and TensorFlow. It provides a clean and simple API that allows you to define and evaluate deep learning models in just a few lines of code.

Because of the ease of use and because it leverages the power of Theano and TensorFlow, Keras is quickly becoming the go-to library for applied deep learning. The focus of Keras is the concept of a model. The life-cycle of a model can be summarized as follows:

1. Define your model. Create a Sequential model and add configured layers.
2. Compile your model. Specify loss function and optimizers and call the `compile()` function on the model.
3. Fit your model. Train the model on a sample of data by calling the `fit()` function on the model.
4. Make predictions. Use the model to generate predictions on new data by calling functions such as `evaluate()` or `predict()` on the model.

Your goal for this lesson is to install Keras. For example, you can install Keras using `pip`:

```
1 sudo pip install keras
```

Listing 5: Install Keras with `pip`.

Start to familiarize yourself with the Keras library ready for the upcoming lessons where we will implement our first model. You can learn more about the Keras library on the Keras homepage[4].

[4] http://keras.io/

Lesson 04: Crash Course in Multilayer Perceptrons

Artificial neural networks are a fascinating area of study, although they can be intimidating when just getting started. The field of artificial neural networks is often just called neural networks or Multilayer Perceptrons after perhaps the most useful type of neural network. The building block for neural networks are artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.

Neurons are arranged into networks of neurons. A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology. Once configured, the neural network needs to be trained on your dataset. The classical and still preferred training algorithm for neural networks is called stochastic gradient descent.

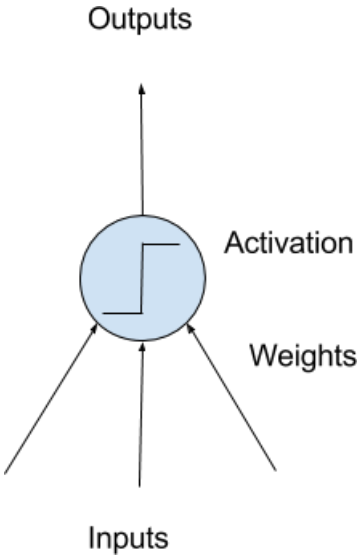


Figure 1: Model of a Simple Neuron

Your goal for this lesson is to become familiar with neural network terminology. Dig a little deeper into terms like neuron, weights, activation function, learning rate and more.

Lesson 05: First Neural Net in Keras

Keras allows you to develop and evaluate deep learning models in very few lines of code. In this lesson your goal is to develop your first neural network using the Keras library. Use a standard binary (two-class) classification dataset from the UCI Machine Learning Repository, like the Pima Indians[5] or the ionosphere datasets[6]. Piece together code to achieve the following:

1. Load your dataset using NumPy or Pandas.
2. Define your neural network model and compile it.
3. Fit your model to the dataset.
4. Estimate the performance of your model on unseen data.

To give you a massive kick start, below is a complete working example that you can use as a starting point. It assumes that you have downloaded the Pima Indians dataset to your current working directory with the filename `pima-indians-diabetes.csv`.

```
from keras.models import Sequential
from keras.layers import Dense
# Load the dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
# Define and Compile
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy' , optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# Evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Listing 6: First Neural Network in Keras.

Now develop your own model on a different dataset, or adapt this example.

[5] https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

[6] https://archive.ics.uci.edu/ml/datasets/Ionosphere

Lesson 06: Use Keras Models With Scikit-Learn

The scikit-learn library is a general purpose machine learning framework in Python built on top of SciPy. Scikit-learn excels at tasks such as evaluating model performance and optimizing model hyperparameters in just a few lines of code. Keras provides a wrapper class that allows you to use your deep learning models with scikit-learn. For example, an instance of `KerasClassifier` class in Keras can wrap your deep learning model and be used as an Estimator in scikit-learn.

When using the `KerasClassifier` class, you must specify the name of a function that the class can use to define and compile your model. You can also pass additional parameters to the constructor of the `KerasClassifier` class that will be passed to the `model.fit()` call later, like the number of epochs and batch size. In this lesson your goal is to develop a deep learning model and evaluate it using k-fold cross-validation. For example, you can define an instance of the `KerasClassifier` and the custom function to create your model as follows:

```
# Function to create model, required for KerasClassifier
def create_model():
  # Create model
  model = Sequential()
  ...
  # Compile model
  model.compile(...)
  return model

# create classifier for use in scikit-learn
model = KerasClassifier(build_fn=create_model, epochs=150, batch_size=10)
# evaluate model using 10-fold cross-validation in scikit-learn
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(model, X, Y, cv=kfold)
```

Listing 7: Use Keras Models in scikit-learn.

Learn more about using your Keras deep learning models with scikit-learn on the Wrappers for the Sciki-Learn API webpage[7].

[7] http://keras.io/scikit-learn-api/

Lesson 07: Plot Model Training History

You can learn a lot about neural networks and deep learning models by observing their performance over time during training. Keras provides the capability to register callbacks when training a deep learning model. One of the default callbacks that is registered when training all deep learning models is the `History` callback. It records training metrics for each epoch. This includes the loss and the accuracy (for classification problems) as well as the loss and accuracy for the validation dataset, if one is set.

The history object is returned from calls to the `fit()` function used to train the model. Metrics are stored in a dictionary in the history member of the object returned. Your goal for this lesson is to investigate the history object and create plots of model performance during training. For example, you can print the list of metrics collected by your history object as follows:

```
# list all data in history
history = model.fit(...)
print(history.history.keys())
```

Listing 8: Access Keras Model Training History.

You can learn more about the History object and the callback API in Keras[8].

[8] http://keras.io/callbacks/#history

Lesson 08: Save Your Best Model During Training With Checkpointing

Application checkpointing is a fault tolerance technique for long running processes. The Keras library provides a checkpointing capability by a callback API. The `ModelCheckpoint` callback class allows you to define where to checkpoint the model weights, how the file should be named and under what circumstances to make a checkpoint of the model. Checkpointing can be useful to keep track of the model weights in case your training run is stopped prematurely. It is also useful to keep track of the best model observed during training.

In this lesson, your goal is to use the `ModelCheckpoint` callback in Keras to keep track of the best model observed during training. You could define a `ModelCheckpoint` that saves network weights to the same file each time an improvement is observed. For example:

```
from keras.callbacks import ModelCheckpoint
...
checkpoint = ModelCheckpoint('weights.best.hdf5', monitor='val_accuracy',
    save_best_only=True, mode='max')
callbacks_list = [checkpoint]
# Fit the model
model.fit(..., callbacks=callbacks_list)
```

Listing 9: Checkpoint Model Weights During Training.

Learn more about using the `ModelCheckpoint` callback in Keras[9].

[9] http://keras.io/callbacks/#modelcheckpoint

Lesson 09: Reduce Overfitting With Dropout Regularization

A big problem with neural networks is that they can overlearn your training dataset. Dropout is a simple yet very effective technique for reducing dropout and has proven useful in large deep learning models. Dropout is a technique where randomly selected neurons are ignored during training. They are dropped-out randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

You can add a dropout layer to your deep learning model using the `Dropout` layer class. In this lesson your goal is to experiment with adding dropout at different points in your neural network and set to different probability of dropout values. For example, you can create a dropout layer with the probability of 20% and add it to your model as follows:

```
1 from keras.layers import Dropout
2 ...
3 model.add(Dropout(0.2))
```

Listing 10: Use Dropout In Your Models.

You can learn more about dropout in Keras[10].

[10] http://keras.io/layers/core/#dropout

Lesson 10: Lift Performance With Learning Rate Schedules

You can often get a boost in the performance of your model by using a learning rate schedule. Often called an adaptive learning rate or an annealed learning rate, this is a technique where the learning rate used by stochastic gradient descent changes while training your model. Keras has a time-based learning rate schedule built into the implementation of the stochastic gradient descent algorithm in the SGD class.

When constructing the class, you can specify the `decay` argument which is the amount that your learning rate (also specified) will decrease each epoch. When using learning rate decay you should bump up your initial learning rate and consider adding a large momentum value such as 0.8 or 0.9. Your goal in this lesson is to experiment with the time-based learning rate schedule built into Keras. For example, you can specify a learning rate schedule that starts at 0.1 and drops by 0.0001 each epoch as follows:

```
from keras.optimizers import SGD
...
sgd = SGD(lr=0.1, momentum=0.9, decay=0.0001, nesterov=False)
model.compile(..., optimizer=sgd)
```

Listing 11: Use a Learning Rate Schedule When Training Models.

You can learn more about the SGD class in Keras here[11].

[11] http://keras.io/optimizers/#sgd

Lesson 11: Crash Course in Convolutional Neural Networks

Convolutional Neural Networks are a powerful artificial neural network technique. They expect and preserve the spatial relationship between pixels in images by learning internal feature representations using small squares of input data. Feature are learned and used across the whole image, allowing for the objects in your images to be shifted or translated in the scene and still detectable by the network. It is this reason why this type of network is so useful for object recognition in photographs, picking out digits, faces, objects and so on with varying orientation. There are three types of layers in a Convolutional Neural Network:

- Convolutional Layers comprised of filters and feature maps.
- Pooling Layers that downsample the activations from feature maps.
- Fully-Connected Layers that plug on the end of the model and can be used to make predictions.

In this lesson you are to familiarize yourself with the terminology used when describing convolutional neural networks. This may require a little research on your behalf. Don't worry too much about how they work just yet, just learn the terminology and configuration of the various layers used in this type of network.

Lesson 12: Handwritten Digit Recognition

Handwriting digit recognition is a difficult computer vision classification problem. The MNIST dataset is a standard problem for evaluating algorithms on the problem of handwriting digit recognition. It contains 60,000 images of digits that can be used to train a model, and 10,000 images that can be used to evaluate it's performance.

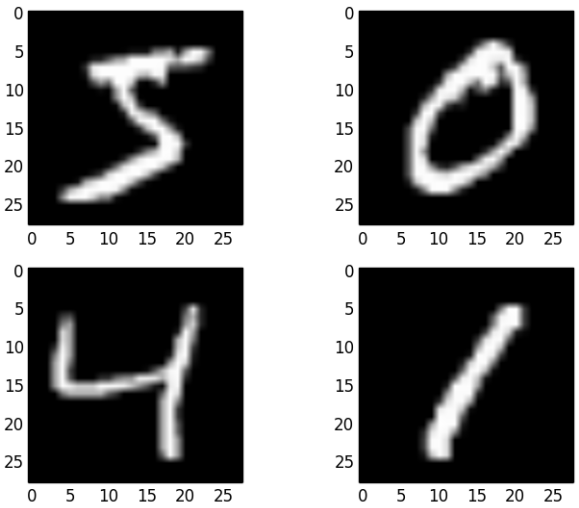


Figure 2: Examples from the MNIST dataset

State-of-the-art results can be achieved on the MNIST problem using convolutional neural networks. Keras makes loading the MNIST dataset dead easy. In this lesson your goal is to develop a very simple convolutional neural network for the MNIST problem comprised of one convolutional layer, one max pooling layer and one dense layer to make predictions. For example, you can load the MNIST dataset in Keras as follows:

```
from keras.datasets import mnist
...
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Listing 12: Load the MNIST Dataset.

It may take a moment to download the files to your computer. As a tip, the Keras `Conv2D` layer that you will use as your first hidden layer expects image data in the format width $\times$

height × channels, where the MNIST data has 1 channel because the images are grayscale and a width and height of 28 pixels. You can easily reshape the MNIST dataset as follows:

```
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))
```

Listing 13: Reshape the MNIST Dataset.

You will also need to one hot encode the output class value, that Keras also provides a handy helper function to achieve:

```
from keras.utils import np_utils
...
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

Listing 14: One Hot Encode Output Variables.

As a final tip, here is a model definition that you can use as a starting point:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='valid', input_shape=(28, 28, 1),
activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Listing 15: Example Convolutional Neural Network Model.

You can learn more about the convolutional neural network layers API on the Keras webpage[12].

[12] http://keras.io/layers/convolutional/

Lesson 13: Object Recognition in Small Photographs

Object recognition is a problem where your model must indicate what is in a photograph. Deep learning models achieve state-of-the-art results in this problem using deep convolutional neural networks. A popular standard dataset for evaluating models on this type of problem is called CIFAR-10. It contains 60,000 small photographs, each of one of 10 objects, like a cat, ship or airplane.

Figure 3: Small Sample of CIFAR-10 Images.

As with the MNIST dataset, Keras provides a convenient function that you can use to load the dataset, and it will download it to your computer the first time you try to load it. The dataset is a 163 MB so it may take a few minutes to download. Your goal in this lesson is to develop a deep convolutional neural network for the CIFAR-10 dataset. I would recommend a repeated pattern of convolution and pooling layers. Consider experimenting with drop-out and

long training times. For example, you can load the CIFAR-10 dataset in Keras and prepare it for use with a convolutional neural network as follows:

```
from keras.datasets import cifar10
from keras.utils import np_utils
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

Listing 16: Example Loading CIFAR-10 With Keras.

Lesson 14: Improve Generalization With Data Augmentation

Data preparation is required when working with neural network and deep learning models. Increasingly data augmentation is also required on more complex object recognition tasks. This is where images in your dataset are modified with random flips and shifts. This in essence makes your training dataset larger and helps your model to generalize the position and orientation of objects in images.

Keras provides an image augmentation API that will create modified versions of images in your dataset just-in-time. The `ImageDataGenerator` class can be used to define the image augmentation operations to perform which can be fit to a dataset and then used in place of your dataset when training your model. Your goal with this lesson is to experiment with the Keras image augmentation API using a dataset you are already familiar with from a previous lesson like MNIST or CIFAR-10. For example, the example below creates random rotations of up to 90 degrees of images in the MNIST dataset.

```
# Random Rotations
from keras.datasets import mnist
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# reshape to be [samples][pixels][width][height]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))
# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# define data preparation
datagen = ImageDataGenerator(rotation_range=90)
# fit parameters from data
datagen.fit(X_train)
# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9):
  # create a grid of 3 * 3 images
  for i in range(0, 9):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(X_batch[i].reshape(28, 28), cmap=pyplot.get_cmap('gray'))
  # show the plot
  pyplot.show()
  break
```

Listing 17: Example Using the Keras Image Augmentation to Rotate MNIST Images.

You can learn more about the Keras image augmentation API[13].

[13] http://keras.io/preprocessing/image/

Final Word Before You Go...

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

- You discovered deep learning libraries in Python including the powerful numerical libraries Theano and TensorFlow and the easy to use Keras library for applied deep learning.
- You built your first neural network using Keras and learned how to use your deep learning models with scikit-learn and how to retrieve and plot the training history for your models.
- You learned about more advanced techniques such as dropout regularization and learning rate schedules and how you can use these techniques in Keras.
- Finally, you took the next step and learned about and developed convolutional neural networks for complex computer vision tasks and learned about augmentation of image data.

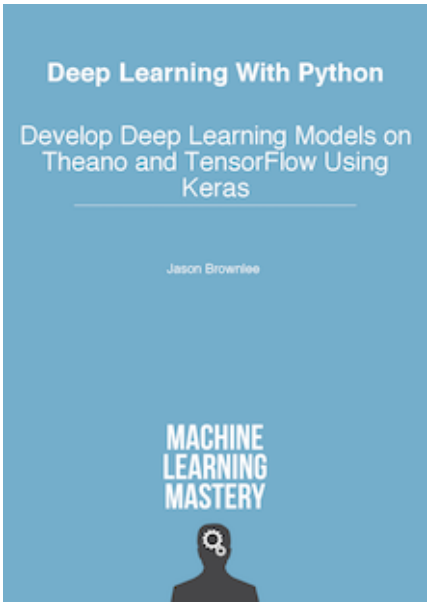
Don't make light of this, you have come a long way in a short amount of time. This is just the beginning of your machine learning journey with Python. Keep practicing and developing your skills.

How Did You Go With The Mini-Course?

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

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

If you would like me to step you through each lesson in great detail (and much more), take a look at my book: **Deep Learning With Python:**



Learn more here:
https://machinelearningmastery.com/deep-learning-with-python