Deep Learning With Python

Develop Deep Learning Models on Theano and TensorFlow Using Keras

Jason Brownlee

MACHINE LEARNING MASTERY

Disclaimer

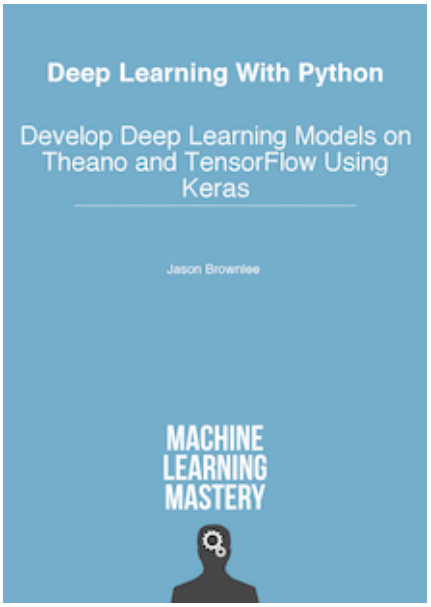
Copyright

Deep Learning With Python

Edition: v1.18

This is Just a Sample

Thank-you for your interest in **Deep Learning With Python**.
This is just a sample of the full text. You can purchase the complete book online from:
https://machinelearningmastery.com/deep-learning-with-python/



Contents

Copyright i

1 Welcome 1
- 1.1 Deep Learning The Wrong Way . . . 1
- 1.2 Deep Learning With Python . . . 2
- 1.3 Book Organization . . . 2
- 1.4 Requirements For This Book . . . 5
- 1.5 Your Outcomes From Reading This Book . . . 6
- 1.6 What This Book is Not . . . 7
- 1.7 Summary . . . 7

2 Develop Your First Neural Network With Keras 8
- 2.1 Tutorial Overview . . . 8
- 2.2 Pima Indians Onset of Diabetes Dataset . . . 9
- 2.3 Load Data . . . 10
- 2.4 Define Model . . . 10
- 2.5 Compile Model . . . 11
- 2.6 Fit Model . . . 12
- 2.7 Evaluate Model . . . 12
- 2.8 Tie It All Together . . . 13
- 2.9 Make Predictions . . . 14
- 2.10 Summary . . . 15

Chapter 1

Welcome

Welcome to Deep Learning With Python. This book is your guide to deep learning in Python. You will discover the Keras Python library for deep learning and how to use it to develop and evaluate deep learning models. In this book you will discover the techniques, recipes and skills in deep learning that you can then bring to your own machine learning projects.

Deep learning does have a lot of fascinating math under the covers, but you do not need to know it to be able to pick it up as a tool and wield it on important projects and deliver real value. From the applied perspective, deep learning is quite a shallow field and a motivated developer can quickly pick it up and start making very real and impactful contributions. This is my goal for you and this book is your ticket to that outcome.

1.1 Deep Learning The Wrong Way

If you ask a deep learning practitioner how to get started with neural networks and deep learning, what do they say? They say things like

- You must have a strong foundation in linear algebra.
- You must have a deep knowledge of traditional neural network techniques.
- You really must know about probability and statistics.
- You should really have a deep knowledge of machine learning.
- You probably need to be a PhD in computer science.
- You probably need 10 years of experience as a machine learning developer.

You can see that the *common sense* advice means that it is not until after you have completed years of study and experience that you are ready to actually start developing and evaluating machine learning model for your machine learning projects.

I think this advice is dead wrong.

1.2 Deep Learning With Python

The approach taken with this book and with all of Machine Learning Mastery is to flip the traditional approach. If you are interested in deep learning, start by developing and evaluating deep learning models. Then if you discover you really like it or have a knack for it, later you can step deeper and deeper into the background and theory, as you need it in order to serve you in developing better and more valuable results. This book is your ticket to jumping in and making a ruckus with deep learning.

I have used many of the top deep learning platforms and libraries and I chose what I think is the best-of-breed platform for getting started and very quickly developing powerful and even state-of-the-art deep learning models in the Keras deep learning library for Python. Unlike R, Python is a fully featured programming language allowing you to use the same libraries and code for model development as you can use in production. Unlike Java, Python has the SciPy stack for scientific computing and scikit-learn which is a professional grade machine library.

There are two top numerical platforms for developing deep learning models, they are Theano developed by the University of Montreal and TensorFlow developed at Google. Both were developed for use in Python and both can be leveraged by the super simple to use Keras library. Keras wraps the numerical computing complexity of Theano and TensorFlow providing a concise API that we will use to develop our own neural network and deep learning models.

You will develop your own and perhaps your first neural network and deep learning models while working through this book, and you will have the skills to bring this amazing new technology to your own projects. It is going to be a fun journey and I can't wait to start.

1.3 Book Organization

This book is broken down into three parts.

- **Lessons** where you learn about specific features of neural network models and how to use specific aspects of the Keras API.
- **Projects** where you will pull together multiple lessons into an end-to-end project and deliver a result, providing a template for your own projects.
- **Recipes** where you can copy and paste the standalone code into your own project, including all of the code presented in this book.

1.3.1 Lessons and Projects

Lessons are discrete and are focused on one topic, designed for you to complete in one sitting. You can take as long as you need, from 20 minutes if you are racing through, to hours if you want to experiment with the code or ideas and improve upon the presented results. Your lessons are divided into five parts:

- Background.
- Multilayer Perceptrons.
- Advanced Multilayer Perceptrons and Keras.

- Convolutional Neural Networks.
- Recurrent Neural Networks.

1.3.2 Part 2: Background

In this part you will learn about the Theano, TensorFlow and Keras libraries that lay the foundation for your deep learning journey. This part of the book includes the following lessons:

- Introduction to the Theano Numerical Library.
- Introduction to the TensorFlow Numerical Library.
- Introduction to the Keras Deep Learning Library.

The lessons will introduce you to the important foundational libraries that you need to install and use on your workstation. At the end of this part you will be ready to start developing models in Keras on your workstation.

1.3.3 Part 3: Multilayer Perceptrons

In this part you will learn about feedforward neural networks that may be deep or not and how to expertly develop your own networks and evaluate them efficiently using Keras. This part of the book includes the following lessons:

- Crash Course In Multilayer Perceptrons.
- Develop Your First Neural Network With Keras.
- Evaluate The Performance of Deep Learning Models.
- Use Keras Models With Scikit-Learn For General Machine Learning.

These important lessons are tied together with three foundation projects. These projects demonstrate how you can quickly and efficiently develop neural network models for tabular data and provide project templates that you can use on your own regression and classification machine learning problems. These projects include:

- Project: Multiclass Classification Problem.
- Project: Binary Classification Problem.
- Project: Regression Problem.

At the end of this part you will be ready to discover the finer points of deep learning using the Keras API.

1.3.4 Part 4: Advanced Multilayer Perceptrons

In this part you will learn about some of the more finer points of the Keras library and API for practical machine learning projects and some of the more important developments in applied neural networks that you need to know in order to deliver world class results. This part of the book includes the following lessons:

- Save Your Models For Later With Network Serialization.
- Keep The Best Models During Training With Checkpointing.
- Understand Model Behavior During Training By Plotting History.
- Reduce Overfitting With Dropout Regularization.
- Lift Performance With Learning Rate Schedules.

At the end of this part you will know how to confidently wield Keras on your own machine learning projects with a focus on the finer points of investigating model performance, persisting models for later use and gaining lifts in performance over baseline models.

1.3.5 Part 5: Convolutional Neural Networks

In this part you will receive a crash course in the dominant model for computer vision machine learning problems and some natural language problems and how you can best exploit the capabilities of the Keras API for your own projects. This part of the book includes the following lessons:

- Crash Course In Convolutional Neural Networks.
- Improve Model Performance With Image Augmentation.

The best way to learn about this impressive type of neural network model is to apply it. You will work through three larger projects and apply CNN to image data for object recognition and text data for sentiment classification.

- Project: Handwritten Digit Recognition.
- Project: Object Recognition in Photographs.
- Project: Movie Review Sentiment Classification.

After completing the lessons and projects in this part you will have the skills and the confidence of complete and working templates and recipes to tackle your own deep learning projects using convolutional neural networks.

1.3.6 Part 6: Recurrent Neural Networks

In this part you will receive a crash course in the dominant model for data with a sequence or time component and how you can best exploit the capabilities of the Keras API for your own projects. This part of the book includes the following lessons:

- Crash Course In Recurrent Neural Networks.
- Multilayer Perceptron Models for Time Series Problems.
- LSTM Models for Time Series Problems.
- Understanding State in LSTM Models for Sequence Prediction.

The best way to learn about this complex type of neural network model is to apply it. You will work through two larger projects and apply RNN to sequence classification and text generation.

- Project: Sequence Classification of Movie Reviews.
- Project: Text Generation With Alice in Wonderland.

After completing the lessons and projects in this part you will have the skills and the confidence of complete and working templates and recipes to tackle your own deep learning projects using recurrent neural networks.

1.3.7 Conclusions

The book concludes with some resources that you can use to learn more information about a specific topic or find help if you need it as you start to develop and evaluate your own deep learning models.

1.3.8 Recipes

Building up a catalog of code recipes is an important part of your deep learning journey. Each time you learn about a new technique or new problem type, you should write up a short code recipe that demonstrates it. This will give you a starting point to use on your next deep learning or machine learning project.

As part of this book you will receive a catalog of deep learning recipes. This includes recipes for all of the lessons presented in this book, as well as the complete code for all of the projects. You are strongly encouraged to add to and build upon this catalog of recipes as you expand your use and knowledge of deep learning in Python.

1.4 Requirements For This Book

1.4.1 Python and SciPy

You do not need to be a Python expert, but it would be helpful if you knew how to install and setup Python and SciPy. The lessons and projects assume that you have a Python and SciPy

environment available. This may be on your workstation or laptop, it may be in a VM or a Docker instance that you run, or it may be a server instance that you can configure in the cloud. You will be guided as to how to install the deep learning libraries Theano, TensorFlow and Keras in Part II of the book. If you have trouble, you can follow the step-by-step tutorial in Appendix **??**.

1.4.2 Machine Learning

You do not need to be a machine learning expert, but it would be helpful if you knew how to navigate a small machine learning problem using scikit-learn. Basic concepts like cross-validation and one hot encoding used in lessons and projects are described, but only briefly. There are resources to go into these topics in more detail at the end of the book, but some knowledge of these areas might make things easier for you.

1.4.3 Deep Learning

You do not need to know the math and theory of deep learning algorithms, but it would be helpful to have some basic idea of the field. You will get a crash course in neural network terminology and models, but we will not go into much detail. Again, there will be resources for more information at the end of the book, but it might be helpful if you can start with some idea about neural networks.

Note: All tutorials can be completed on standard workstation hardware with a CPU. A GPU is not required. Some tutorials later in the book can be speed up significantly by running on the GPU and a suggestion is provided to consider using GPU hardware at the beginning of those sections. You can access GPU hardware easily and cheaply in the cloud and a step-by-step procedure is taught on how to do this in Appendix **??**.

1.5 Your Outcomes From Reading This Book

This book will lead you from being a developer who is interested in deep learning with Python to a developer who has the resources and capabilities to work through a new dataset end-to-end using Python and develop accurate deep learning models. Specifically, you will know:

- How to develop and evaluate neural network models end-to-end.
- How to use more advanced techniques required for developing state-of-the-art deep learning models.
- How to build larger models for image and text data.
- How to use advanced image augmentation techniques in order to lift model performance.
- How to get help with deep learning in Python.

From here you can start to dive into the specifics of the functions, techniques and algorithms used with the goal of learning how to use them better in order to deliver more accurate predictive models, more reliably in less time. There are a few ways you can read this book. You can dip

into the lessons and projects as your need or interests motivate you. Alternatively, you can work through the book end-to-end and take advantage of how the lessons and projects build in complexity and range. I recommend the latter approach.

To get the very most from this book, I recommend taking each lesson and project and build upon them. Attempt to improve the results, apply the method to a similar but different problem, and so on. Write up what you tried or learned and share it on your blog, social media or send me an email at `jason@MachineLearningMastery.com`. This book is really what you make of it and by putting in a little extra, you can quickly become a true force in applied deep learning.

1.6 What This Book is Not

This book solves a specific problem of getting you, a developer, up to speed applying deep learning to your own machine learning projects in Python. As such, this book was not intended to be everything to everyone and it is very important to calibrate your expectations. Specifically:

- **This is not a deep learning textbook**. We will not be getting into the basic theory of artificial neural networks or deep learning algorithms. You are also expected to have some familiarity with machine learning basics, or be able to pick them up yourself.
- **This is not an algorithm book**. We will not be working through the details of how specific deep learning algorithms work. You are expected to have some basic knowledge of deep learning algorithms or how to pick up this knowledge yourself.
- **This is not a Python programming book**. We will not be spending a lot of time on Python syntax and programming (e.g. basic programming tasks in Python). You are expected to already be familiar with Python or a developer who can pick up a new C-like language relatively quickly.

You can still get a lot out of this book if you are weak in one or two of these areas, but you may struggle picking up the language or require some more explanation of the techniques. If this is the case, see the Getting More Help chapter at the end of the book and seek out a good companion reference text.

1.7 Summary

It is a special time right now. The tools for applied deep learning have never been so good. The pace of change with neural networks and deep learning feels like it has never been so fast, spurred by the amazing results that the methods are showing in such a broad range of fields. This is the start of your journey into deep learning and I am excited for you. Take your time, have fun and I'm so excited to see where you can take this amazing new technology.

1.7.1 Next

Let's dive in. Next up is Part II where you will take a whirlwind tour of the foundation libraries for deep learning in Python, namely the numerical libraries Theano and TensorFlow and the library you will be using throughout this book called Keras.

Chapter 2

Develop Your First Neural Network With Keras

Keras is a powerful and easy-to-use Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in a few short lines of code. In this lesson you will discover how to create your first neural network model in Python using Keras. After completing this lesson you will know:

- How to load a CSV dataset ready for use with Keras.
- How to define and compile a Multilayer Perceptron model in Keras.
- How to evaluate a Keras model on a validation dataset.

Let's get started.

2.1 Tutorial Overview

There is not a lot of code required, but we are going to step over it slowly so that you will know how to create your own models in the future. The steps you are going to cover in this tutorial are as follows:

1. Load Data.
2. Define Model.
3. Compile Model.
4. Fit Model.
5. Evaluate Model.
6. Tie It All Together.
7. Make Predictions.

2.2 Pima Indians Onset of Diabetes Dataset

In this tutorial we are going to use the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset available for free download from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years. It is a binary classification problem (onset of diabetes as 1 or not as 0). The input variables that describe each patient are numerical and have varying scales. Below lists the eight attributes for the dataset:

1. Number of times pregnant.
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skin fold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index.
7. Diabetes pedigree function.
8. Age (years).
9. Class, onset of diabetes within five years.

Given that all attributes are numerical makes it easy to use directly with neural networks that expect numerical inputs and output values, and ideal for our first neural network in Keras. This dataset will also be used for a number of additional lessons coming up in this book, so keep it handy. Below is a sample of the dataset showing the first 5 rows of the 768 instances:

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
```

Listing 2.1: Sample of the Pima Indians Dataset.

The dataset file is available in your bundle of code recipes provided with this book. Alternatively, you can download the Pima Indian dataset from the UCI Machine Learning repository and place it in your local working directory, the same as your Python file[1]. Save it with the file name:

```
pima-indians-diabetes.csv
```

Listing 2.2: Pima Indians Dataset File.

The baseline accuracy if all predictions are made as *no onset of diabetes* is 65.1%. Top results on the dataset are in the range of 77.7% accuracy using 10-fold cross-validation[2]. You can learn more about the dataset from the published dataset details[3].

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

[2] http://www.is.umk.pl/projects/datasets.html

[3] https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names

2.3 Load Data

The first step is to define the functions and classes we intend to use in this tutorial. We will use the NumPy library to load our dataset and we will use two classes from the Keras library to define our model. The imports required are listed below.

```
# first neural network with keras tutorial
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
...
```

Listing 2.3: Load Libraries.

Now we can load our Pima Indians dataset. You can now load the file directly using the NumPy function `loadtxt()`. There are eight input variables and one output variable (the last column). Once loaded we can split the dataset into input variables (X) and the output class variable (Y).

```
...
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
```

Listing 2.4: Load The Dataset Using NumPy.

We are now ready to define our neural network model.

2.4 Define Model

Models in Keras are defined as a sequence of layers. We create a `Sequential` model and add layers one at a time until we are happy with our network topology. The first thing to get right is to ensure the input layer has the right number of inputs. This can be specified when creating the first layer with the `input_dim` argument and setting it to 8 for the 8 input variables.

How do we know the number of layers to use and their types? This is a very hard question. There are heuristics that we can use and often the best network structure is found through a process of trial and error experimentation. Generally, you need a network large enough to capture the structure of the problem if that helps at all. In this example we will use a fully-connected network structure with three layers.

Fully connected layers are defined using the `Dense` class. We can specify the number of neurons in the layer as the first argument and specify the activation function using the `activation` argument. We will use the rectifier (`relu`) activation function on the first two layers and the `sigmoid` activation function in the output layer. It used to be the case that sigmoid and `tanh` activation functions were preferred for all layers. These days, better performance is seen using the rectifier activation function. We use a sigmoid activation function on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5. We can piece it all together by adding each layer. The first hidden layer has 12 neurons and expects

8 input variables (e.g. `input_dim=8`). The second hidden layer has 8 neurons and finally the output layer has 1 neuron to predict the class (onset of diabetes or not).

```
...
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Listing 2.5: Define the Neural Network Model in Keras.

The figure below provides a depiction of the network structure.

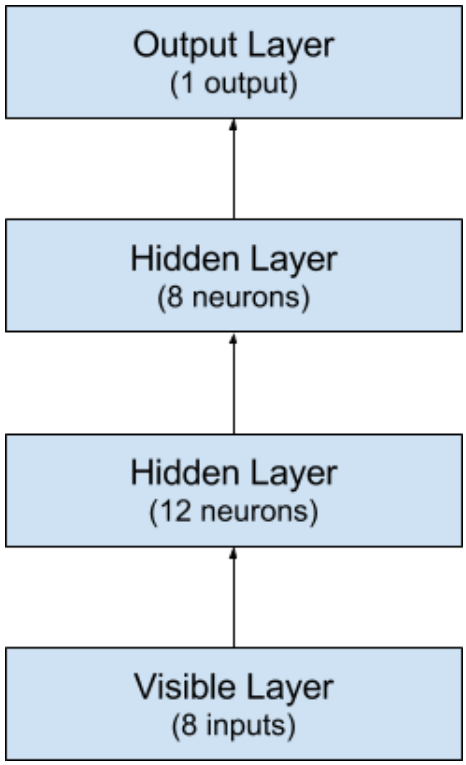


Figure 2.1: Visualization of Neural Network Structure.

2.5 Compile Model

Now that the model is defined, we can compile it. Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware. When compiling, we must specify some additional properties required when training the network. Remember training a network means finding the best set of weights to make predictions for this problem.

We must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network and any optional metrics we would like to collect and report during training. In this case we will use logarithmic loss, which for a binary classification problem is defined in Keras as `binary_crossentropy`. We will also use the

efficient gradient descent algorithm `adam` for no other reason that it is an efficient default. Learn more about the Adam optimization algorithm in the paper *Adam: A Method for Stochastic Optimization*[4]. Finally, because it is a classification problem, we will collect and report the classification accuracy as the metric.

```
...
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Listing 2.6: Compile the Neural Network Model.

2.6 Fit Model

We have defined our model and compiled it ready for efficient computation. Now it is time to execute the model on some data. We can train or fit our model on our loaded data by calling the `fit()` function on the model.

The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the `epochs` argument. We can also set the number of instances that are evaluated before a weight update in the network is performed called the batch size and set using the `batch_size` argument. For this problem we will run for a small number of epochs (150) and use a relatively small batch size of 16. Again, these can be chosen experimentally by trial and error.

```
...
# Fit the model
model.fit(X, Y, epochs=150, batch_size=16)
```

Listing 2.7: Fit the Neural Network Model to the Dataset.

This is where the work happens on your CPU or GPU.

2.7 Evaluate Model

We have trained our neural network on the entire dataset and we can evaluate the performance of the network on the same dataset. This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no idea of how well the algorithm might perform on new data. We have done this for simplicity, but ideally, you could separate your data into train and test datasets for the training and evaluation of your model.

You can evaluate your model on your training dataset using the `evaluation()` function on your model and pass it the same input and output used to train the model. This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.

```
...
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Listing 2.8: Evaluate the Neural Network Model on the Dataset.

[4] http://arxiv.org/abs/1412.6980

2.8 Tie It All Together

You have just seen how you can easily create your first neural network model in Keras. Let's tie it all together into a complete code example.

```
# first neural network with keras tutorial
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=16)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Listing 2.9: Complete Working Example of Your First Neural Network in Keras.

You can copy all of the code into your Python file and save it as `keras_first_network.py` in the same directory as your data file `pima-indians-diabetes.csv`. You can then run the Python file as a script from your command line (command prompt) as follows:

```
python keras_first_network.py
```

Listing 2.10: Example of running a Python script.

Running this example, you should see a message for each of the 150 epochs printing the loss and accuracy for each, followed by the final evaluation of the trained model on the training dataset. It takes about 10 seconds to execute on my workstation running on the CPU with a Theano backend.

Note: Your specific results may vary given the stochastic nature of the learning algorithm. Consider running the example a few times and compare the average performance.

```
...
768/768 [==============================] - 0s 63us/step - loss: 0.4817 - acc: 0.7708
Epoch 147/150
768/768 [==============================] - 0s 63us/step - loss: 0.4764 - acc: 0.7747
Epoch 148/150
768/768 [==============================] - 0s 63us/step - loss: 0.4737 - acc: 0.7682
Epoch 149/150
768/768 [==============================] - 0s 64us/step - loss: 0.4730 - acc: 0.7747
Epoch 150/150
768/768 [==============================] - 0s 63us/step - loss: 0.4754 - acc: 0.7799
```

```
768/768 [==============================] - 0s 38us/step
Accuracy: 76.56
```

Listing 2.11: Output of Running Your First Neural Network in Keras.

Neural networks are a stochastic algorithm, meaning that the same algorithm on the same data can train a different model with different skill each time the code is run. This is a feature, not a bug. The variance in the performance of the model means that to get a reasonable approximation of how well your model is performing, you may need to fit it many times and calculate the average of the accuracy scores. For more on this approach to evaluating neural networks, see the post: For example, below are the accuracy scores from re-running the example 5 times:

```
Accuracy: 75.00
Accuracy: 77.73
Accuracy: 77.60
Accuracy: 78.12
Accuracy: 76.17
```

Listing 2.12: Output of Running the model five different times.

We can see that all accuracy scores are around 77% and the average is 76.924%.

2.9 Make Predictions

We can adapt the above example and use it to generate predictions on the training dataset, pretending it is a new dataset we have not seen before. Making predictions is as easy as calling the `predict()` function on the model. We are using a sigmoid activation function on the output layer, so the predictions will be a probability in the range between 0 and 1. We can easily convert them into a crisp binary prediction for this classification task by rounding them. For example:

```
...
# make probability predictions with the model
predictions = model.predict(X)
# round predictions
rounded = [round(x[0]) for x in predictions]
```

Listing 2.13: Example of predicting probabilities for each example.

Alternately, we can call the `predict_classes()` function on the model to predict crisp classes directly, for example:

```
...
# make class predictions with the model
predictions = model.predict_classes(X)
```

Listing 2.14: Example of predicting class labels for each example.

The complete example below makes predictions for each example in the dataset, then prints the input data, predicted class and expected class for the first 5 examples in the dataset.

```
# first neural network with keras make predictions
from numpy import loadtxt
from keras.models import Sequential
```

```
from keras.layers import Dense
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10, verbose=0)
# make class predictions with the model
predictions = model.predict_classes(X)
# summarize the first 5 cases
for i in range(5):
  print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))
```

Listing 2.15: Complete Working Example of Fitting a Model in Keras and Using it to Make Predictions.

Running the example does not show the progress bar as before as we have set the verbose argument to 0. After the model is fit, predictions are made for all examples in the dataset, and the input rows and predicted class value for the first 5 examples is printed and compared to the expected class value. We can see that most rows are correctly predicted. In fact, we would expect about 76.9% of the rows to be correctly predicted based on our estimated performance of the model in the previous section.

Note: Your specific results may vary given the stochastic nature of the learning algorithm. Consider running the example a few times and compare the average performance.

```
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 0 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)
```

Listing 2.16: Output of Making Predictions with Keras.

2.10 Summary

In this lesson you discovered how to create your first neural network model using the powerful Keras Python library for deep learning. Specifically you learned the five key steps in using Keras to create a neural network or deep learning model, step-by-step including:

- How to load data.
- How to define a neural network model in Keras.

- How to compile a Keras model using the efficient numerical backend.
- How to train a model on data.
- How to evaluate a model on data.

2.10.1 Next

You now know how to develop a Multilayer Perceptron model in Keras. In the next section you will discover different ways that you can evaluate your models and estimate their performance on unseen data.

This is Just a Sample

Thank-you for your interest in **Deep Learning With Python**.
This is just a sample of the full text. You can purchase the complete book online from:
https://machinelearningmastery.com/deep-learning-with-python/