Deep Learning In Python Using Tensorflow & Keras

Fredrik Carlsson - 2018

Introduction **2**

Getting Started **2**

Keras & Backend 2

NumPy 3

Setting up the training environment **3**

The Neural Network 4

Training Data 5

Making Predictions 5

Training Our Neural Network 6

Putting It All Together **6**

Generating Training Data 6

Predicting and Training 7

Future Work **8**

Introduction

There currently exists quite a few different deep learning libraries and frameworks out there, all competing for your attention. Often backed by big corporations we for example have:

- **Tensorflow** - Google
- **PyTorch** - Facebook
- **CNTK** - Microsoft

Of Course there also exists some open source competitors namely **Theano.** But fact remains that big corporations are locked into an battle to get you to learn their Deep Learning toolkit.

Naturally we are then faced with the question, "*Where do we start?*", and seeing how trends go in deep learning we can conclude that most tutorials now a days start with another option called **Keras**.

Keras is a sort of wrapper and API for other frameworks, giving you a nice and simple interface for creating deep learning models. Whilst you let Keras figure out how to convert your Keras code into something compatible to your selected backend framework. As of now you can switch between *Tensorflow*, *Theano* and *CNTK* as your backend in Keras, without having to change any of your Keras code.

Getting Started

Python Environment

First things first, and we will have to have a way of writing and executing Python code. For this we of course need to install python. But also an text editor and getting comfortable of how to install Python libraries. I will be using PyCharm, since this pretty much solves all of named tasks above.

Keras & Backend

In addition to installing Keras, we will have to select a backend. For this tutorial I will use Tensorflow as my backend for Keras, seeing that it's the default option. If you wish to do the same you will need to install Tensorflow before getting started. But any backend should suffice.

NumPy

The last thing we will need before getting started is a library called NumPy, which is a commonly used matrix/vector library for scientific computations. Seeing that we will have to represent all of our training and labeled data in the form of NumPy arrays, it's essential we get comfortable with this library. Getting used to NumPy can be a somewhat of a threshold, but trust me it will be worth it. It might be good to have a look at some NumPy tutorial if you have never used this library before.

Setting up the training environment

Here on and forward I presume that you're somewhat comfortable using Python, and will thus not explain the Pythonic tricks that might creep into the code. But even if you're not used to Python you should hopefully be able to follow along quite fine.

Taking things one step at a time we will need to import the libraries we have installed.The use of these libraries will become clear later on.

```
import keras.layers as Layers
import keras.optimizers as Optimizers
from keras import Sequential
import numpy as np
import random
```

The Neural Network

There exists several ways of creating deep neural networks using Keras, but we will make use of the Sequential Model. In essence this allows us to simply define and stack layers upon each other into one deep learning model.

Knowing this we're now ready to create our Neural Network! To keep things organized we will define a function that creates and returns our neural network. For simplicity's sake and to minimise the amount of Pythonistic code I will hardcode the parameters to this network. But feel free to create a more sophisticated function.

```
def createNewSeqModel():
   model = Sequential()
   weights = 'random_uniform'

   model.add(Layers.Dense(50, input_shape=(1,),
activation='relu', kernel_initializer=weights))

   model.add(Layers.Dense(50, activation='relu',
kernel_initializer=weights))

   model.add(Layers.Dense(1, activation='linear',
kernel_initializer=weights))

   model.compile(Optimizers.adam(), loss='mse')
   return model
```

Here we create an Sequential object named **model** and then proceed to add layers to it. The final phenotype of our network has the following structure.

1. Input layer with size: **1**
2. Hidden Layer with size **50**
3. Hidden Layer with size **50**
4. Output Layer with size **1**

Take note that we define the size of our input layer by adding the argument **input_shape** in the first layer we add to our **model**!

Also at every layer we define two parameters: ***activation*** and ***kernel_initializer***. ***Activation*** is somewhat self-explanatory and the ***kernel_initializer*** defines how our weights for that layer should be initialized.
Finally we specify how we wish to train our model. Here I have used the **adam** optimizer which is like **S**tochastic **G**radient **D**escent(***SGD***) on steroids.
Finally we set the loss function to **M**ean **S**quared **E**rror(***MSE***).

Training Data

Reminding ourselves that a Neural Network only works with numbers this means that if we wish to process things like text, images, sound or any cool data we will need to find a way to convert that data into numbers. For example there exists out-of-the box solutions for working with images, and many other datatypes.

For this tutorial however we will work with numbers directly, in the hopes that you get to understand how to use Keras for any other data that you have lying around. There exists a large amount of cool datasets on this website. That being said we will keep things simple and create our own data.

Since the network we have created takes **one** input and returns exactly **one** output, we need to generate data that matches this format. To keep things flexible we create a function that can generate data for us.

```
def createTrainingData(amount, maxNum, func):
    inData = [random.randint(0, maxNum) for i in range(amount)]
    results = [func(d) for d in inData]
    return inData, results
```

Not being used to Python this might look confusing. But essentially this functions creates **amount** of random samples and labels. That's given from the function **func** in the range of [0, **maxNum]**…....

Making Predictions

As mentioned earlier when we're passing data to and from our model we will do this using NumPy. Trying however to minimise how much NumPy you need to comprehend in this tutorial we will once again create a seperate function that does this for us.

```
def predictWithModel(model, samples):
    return model.predict(np.array(samples))
```

The thing to keep in mind is that **predictWithModel** expects the samples to come in the form of a list, where every element is on full sample: **[*s1*, *s2*, *s3*, ...]**
This however means that if we have for example have a network that expects two input features the list passed our predict functions should be a list where every element is a list containing two elements: **[*s1=[f1, f2]*, *s2=[f1, f2]*, *s3=[f1, f2]*, ...]**

Training Our Neural Network

Keeping things nice and tidy we once again create function for this that manages our NumPy conversion and starts the training.

```
def trainModel(model, samples, labels, epochs, batchSize):
   model.fit(np.array(samples), np.array(labels),
epochs=epochs, batch_size=batchSize)
```

epochs defines how many iterations of training that we wish to do.
batchSize defines how many samples we should batch into one calculation of the gradient .

Putting It All Together

We should now have all of the necessary functions to create a simple script so that we can see how well our model performs on the training data. You should now have all of the following functions:

1. ***createNewSeqModel()***
2. ***createTrainingData(amount, maxNum, func)***
3. ***predictWithModel(model, samples)***
4. ***trainModel(model, samples, labels, epochs, batchSize)***

Let's put them to the test!

Generating Training Data

First we need to decide what type of function we wish that our Neural Network should approximate. Yet again we will start of simple and say that we wish to approximate the function: ***f(x) = 3*x***
Later you should of course change this to see how complex functions you can predict, and how you need to change the network architecture to match this.

```
func = lambda x: x*3
samples, labels = createTrainingData(10000, 1000, func)
evalSamples, evalLabels = createTrainingData(5, 100, func)
print(evalSamples)
print(evalLabels)
```

You might think that predicting such a easy function is useless, but as a somewhat wise man once said "*That's just negative thinking*" - Johan Montelius.

Predicting and Training

Now that we have our training data we should create our network and put it to the test.

```
myNet = createNewSeqModel()

#Test model on evaluation data
evalPredictions = predictWithModel(myNet, evalSamples)
print("Predictions:\n", evalPredictions)
print("Labels:\n", evalLabels)
```

Almost certainly your model will not perform very well, hence we will need to do some training. So let's fit it to the training data and see if it seems to perform any better afterwards.

```
#Fit the model to our training data
trainModel(myNet, samples, labels, 100, 100)

#Test model on evaluation data
evalPredictions = predictWithModel(myNet, evalSamples)
print("Predictions:\n", evalPredictions)
print("Labels:\n", evalLabels)
```

Most likely it should perform quite well after only a 100 epochs!
Congratulations you have just successfully created your first Neural Network.

Future Work

Different Functions

Try changing the function that your neural network predicts and see how this changes things. You could for example try the following:

- `func = lambda x: x*3 + 20`
    - Did it manage to predict the: ***+ 20***?
- Try several input variables

Hyperparameters

Hyperparameters refers to the parameters that does not change during training. Unlike for example the weights and the biases. Some different parameters that you can tweak are:

- Amount of training samples
- The range of the training samples
- Different activation functions
- Amount and the size of the layers
- Amount of training epochs
- The batch size

Import an interesting data set

As mentioned earlier there exists tons of cool datasets to be found here. Try one out and see if you can create a model that performs well.

Other cool things to lookup

- Convolutional Neural Networks
- Recurrent Neural Networks
- Autoencoders