Introduction to Deep Learning - Deep Learning basics with Python, TensorFlow and Keras p.1

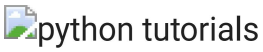
Welcome everyone to an updated deep learning with Python and Tensorflow tutorial mini-series.

Since doing the first deep learning with TensorFlow course a little over 2 years ago, much has changed. It's nowhere near as complicated to get started, nor do you need to know as much to be successful with deep learning.

If you're interested in more of the details with how TensorFlow works, you can still check out the previous tutorials, as they go over the more raw TensorFlow. This is more of a deep learning quick start!

To begin, we need to find some balance between treating neural networks like a total black box, and understanding every single detail with them.

Let's show a typical model:

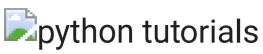


A basic neural network consists of an input layer, which is just your data, in numerical form. After your input layer, you will have some number of what are called "hidden" layers. A hidden layer is just in between your input and output layers. One hidden layer means you just have a neural network. Two or more hidden layers? Boom, you've got a deep neural network!

Why is this? Well, if you just have a single hidden layer, the model is going to only learn linear relationships.

If you have many hidden layers, you can begin to learn non-linear relationships between your input and output layers.

A single neuron might look as follows:



So this is really where the magic happens. The idea is a single neuron is just sum of all of the inputs x weights, fed through some sort of activation function. The activation function is meant to simulate a neuron firing or not. A simple example would be a stepper function, where, at some point, the threshold is crossed, and the neuron fires a 1, else a 0. Let's say that neuron is in the first hidden layer, and it's going to communicate with the next hidden layer. So it's going to send it's 0 or a 1 signal, multiplied by the weights, to the next neuron, and this is the process for all neurons and all layers.

The mathematical challenge for the artificial neural network is to best optimize thousands or millions or whatever number of weights you have, so that your output layer results in

what you were hoping for. Solving for this problem, and building out the layers of our neural network model is exactly what TensorFlow is for. TensorFlow is used for all things "operations on tensors." A tensor in this case is nothing fancy. It's a multi-dimensional array.

To install TensorFlow, simply do a:

```
pip install --upgrade tensorflow
```

Following the release of deep learning libraries, higher-level API-like libraries came out, which sit on top of the deep learning libraries, like TensorFlow, which make building, testing, and tweaking models even more simple. One such library that has easily become the most popular is Keras.

Keras has become so popular, that it is now a superset, included with TensorFlow releases now! If you're familiar with Keras previously, you can still use it, but now you can use `tensorflow.keras` to call it. By that same token, if you find example code that uses Keras, you can use with the TensorFlow version of Keras too. In fact, you can just do something like:

```
import tensorflow.keras as keras
```

For this tutorial, I am going to be using TensorFlow version 1.10. You can figure out your version:

```
import tensorflow as tf

print(tf.__version__)
```

```
1.10.0
```

Once we've got tensorflow imported, we can then begin to prepare our data, model it, and then train it. For the sake of simplicity, we'll be using the most common "hello world" example for deep learning, which is the mnist dataset. It's a dataset of hand-written digits, 0 through 9. It's 28x28 images of these hand-written digits. We will show an example of using outside data as well, but, for now, let's load in this data:

```
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

When you're working with your own collected data, chances are, it wont be packaged up so nicely, and you'll spend a bit more time and effort on this step. But, for now, woo!

What exactly do we have here? Let's take a quick peak.

So the `x_train` data is the "features." In this case, the features are pixel values of the 28x28 images of these digits 0-9. The `y_train` is the label (is it a 0,1,2,3,4,5,6,7,8 or a 9?)

The testing variants of these variables is the "out of sample" examples that we will use. These are examples from our data that we're going to set aside, reserving them for testing the model.

Neural networks are exceptionally good at fitting to data, so much so that they will commonly over-fit the data. Our real hope is that the neural network doesn't just memorize our data and that it instead "generalizes" and learns the actual problem and patterns associated with it.

Let's look at this actual data:

```
print(x_train[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
```

```
[  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
  78   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

Alright, could we visualize this?

```
import matplotlib.pyplot as plt

plt.imshow(x_train[0],cmap=plt.cm.binary)
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

Okay, that makes sense. How about the value for `y_train` with the same index?

```
print(y_train[0])
```

```
5
```

It's generally a good idea to "normalize" your data. This typically involves scaling the data to be between 0 and 1, or maybe -1 and positive 1. In our case, each "pixel" is a feature, and each feature currently ranges from 0 to 255. Not quite 0 to 1. Let's change that with a handy utility function:

```
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

Let's peak one more time:

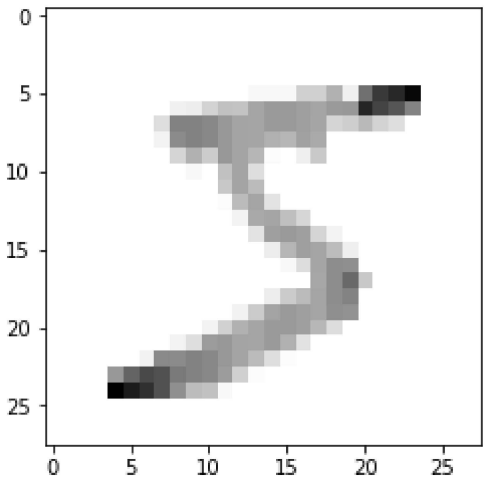
```
print(x_train[0])

plt.imshow(x_train[0],cmap=plt.cm.binary)
plt.show()
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
  0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
  0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
  0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
  0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
  0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
  0.         0.         0.         0.        ]
```

```
      [0.         0.         0.         0.         0.         0.
       0.         0.04500225 0.4219755  0.45852825 0.43408872 0.37314701
       0.33153488 0.32790981 0.28826244 0.26543758 0.34149427 0.31128482
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.1541463  0.28272888 0.18358693 0.37314701
       0.33153488 0.26569767 0.01601458 0.         0.05945042 0.19891229
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.0253731  0.00171577 0.22713296
       0.33153488 0.11664776 0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.20500962
       0.33153488 0.24625638 0.00291174 0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.01622378
       0.24897876 0.32790981 0.10191096 0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.04586451 0.31235677 0.32757096 0.23335172 0.14931733 0.00129164
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.10498298 0.34940902 0.3689874  0.34978968 0.15370495
       0.04089933 0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.06551419 0.27127137 0.34978968 0.32678448
       0.245396   0.05882702 0.         0.         0.         0.
       0.         0.         0.         0.        ]
```

```
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.02333517 0.12857881 0.32549285
       0.41390126 0.40743158 0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.32161793
       0.41390126 0.54251585 0.20001074 0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.06697006 0.18959827 0.25300993 0.32678448
       0.41390126 0.45100715 0.00625034 0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.05110617 0.19182076 0.33339444 0.3689874  0.34978968 0.32678448
       0.40899334 0.39653769 0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.04117838 0.16813739
       0.28960162 0.32790981 0.36833534 0.3689874  0.34978968 0.25961929
       0.12760592 0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.         0.         0.04431706 0.11961607 0.36545809 0.37314701
       0.33153488 0.32790981 0.36833534 0.28877275 0.111988   0.00258328
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.         0.
       0.05298497 0.42752138 0.4219755  0.45852825 0.43408872 0.37314701
       0.33153488 0.25273681 0.11646967 0.01312603 0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
      [0.         0.         0.         0.         0.37491383 0.56222061
       0.66525569 0.63253163 0.48748768 0.45852825 0.43408872 0.359873
       0.17428513 0.01425695 0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.        ]
```

```
 [0.         0.         0.         0.         0.92705966 0.82698729
  0.74473314 0.63253163 0.4084877  0.24466922 0.22648107 0.02359823
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```



Alright, still a 5. Now let's build our model!

```
model = tf.keras.models.Sequential()
```

A sequential model is what you're going to use most of the time. It just means things are going to go in direct order. A feed forward model. No going backwards...for now.

Now, we'll pop in layers. Recall our neural network image? Was the input layer flat, or was it multi-dimensional? It was flat. So, we need to take this 28x28 image, and make it a flat 1x784. There are many ways for us to do this, but keras has a Flatten layer built just for us, so we'll use that.

```
model.add(tf.keras.layers.Flatten())
```

This will serve as our input layer. It's going to take the data we throw at it, and just flatten it for us. Next, we want our hidden layers. We're going to go with the simplest neural network layer, which is just a `Dense` layer. This refers to the fact that it's a densely-connected layer, meaning it's "fully connected," where each node connects to each prior and subsequent node. Just like our image.

```
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
```

This layer has 128 units. The activation function is `relu`, short for rectified linear. Currently, `relu` is the activation function you should just default to. There are many more to test for sure, but, if you don't know what to use, use relu to start.

Let's add another identical layer for good measure.

```
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
```

Now, we're ready for an output layer:

```
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

This is our final layer. It has 10 nodes. 1 node per possible number prediction. In this case, our activation function is a softmax function, since we're really actually looking for something more like a probability distribution of which of the possible prediction options this thing we're passing features through of is. Great, our model is done.

Now we need to "compile" the model. This is where we pass the settings for actually optimizing/training the model we've defined.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Remember why we picked relu as an activation function? Same thing is true for the Adam optimizer. It's just a great default to start with.

Next, we have our `loss` metric. Loss is a calculation of error. A neural network doesn't actually attempt to maximize accuracy. It attempts to minimize loss. Again, there are many choices, but some form of categorical crossentropy is a good start for a classification task like this.

Now, we fit!

```
model.fit(x_train, y_train, epochs=3)
```

```
Epoch 1/3
60000/60000 [==============================] - 7s 119us/step - loss: 0.262
Epoch 2/3
60000/60000 [==============================] - 5s 89us/step - loss: 0.1074
Epoch 3/3
60000/60000 [==============================] - 5s 89us/step - loss: 0.0715


<tensorflow.python.keras.callbacks.History at 0x2871a387cf8>
```

As we train, we can see loss goes down (yay), and accuracy improves quite quickly to 98-99% (double yay!)

Now that's loss and accuracy for in-sample data. Getting a high accuracy and low loss might mean your model learned how to classify digits in general (it generalized)...or it simply memorized every single example you showed it (it overfit). This is why we need to test on out-of-sample data (data we didn't use to train the model).

```
val_loss, val_acc = model.evaluate(x_test, y_test)
print(val_loss)
print(val_acc)
```

```
10000/10000 [==============================] - 0s 34us/step
0.09623032998903655
0.9722
```

Full code up to this point, with some notes:

In [ ]:

```
import tensorflow as tf  # deep learning library. Tensors are just multi-d

mnist = tf.keras.datasets.mnist  # mnist is a dataset of 28x28 images of h
(x_train, y_train),(x_test, y_test) = mnist.load_data()  # unpacks images

x_train = tf.keras.utils.normalize(x_train, axis=1)  # scales data between
x_test = tf.keras.utils.normalize(x_test, axis=1)  # scales data between 0

model = tf.keras.models.Sequential()  # a basic feed-forward model
model.add(tf.keras.layers.Flatten())  # takes our 28x28 and makes it 1x784
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))  # a simple f
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))  # a simple f
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))  # our outp

model.compile(optimizer='adam',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calcu
              metrics=['accuracy'])  # what to track

model.fit(x_train, y_train, epochs=3)  # train the model

val_loss, val_acc = model.evaluate(x_test, y_test)  # evaluate the out of
print(val_loss)  # model's loss (error)
print(val_acc)  # model's accuracy
```

It's going to be very likely your accuracy out of sample is a bit worse, same with loss. In fact, it should be a red flag if it's identical, or better.

Finally, with your model, you can save it super easily:

```
model.save('epic_num_reader.model')
```

Load it back:

```
new_model = tf.keras.models.load_model('epic_num_reader.model')
```

finally, make predictions!

```
predictions = new_model.predict(x_test)
```

```
print(predictions)
```

```
[[4.2427444e-09 1.6043286e-08 1.2955404e-06 ... 9.9999785e-01
  9.2534069e-09 1.3064107e-07]
 [2.8164232e-08 1.0317165e-04 9.9989593e-01 ... 1.0429282e-09
  2.1738730e-07 3.1334544e-12]
 [2.0918677e-07 9.9984252e-01 2.9539053e-05 ... 5.4588450e-05
  6.1221406e-05 1.4490828e-07]
 ...
 [2.3763378e-09 8.4034167e-07 2.0469349e-08 ... 1.1768799e-05
  4.7667407e-07 1.0428642e-04]
 [4.5215497e-06 6.1512014e-06 4.7418069e-08 ... 1.1381173e-06
  7.6969003e-04 8.6409798e-08]
 [6.3100595e-07 2.5021842e-07 2.4660849e-07 ... 2.5157806e-10
  5.8900000e-08 2.3496944e-09]]
```

Uwotm8?

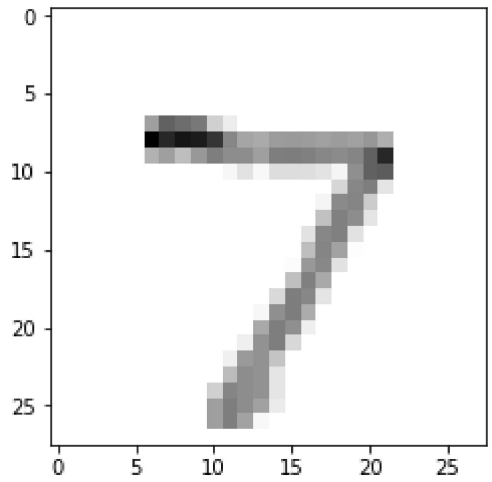
That sure doesn't start off as helpful, but recall these are probability distributions. We can get the actual number pretty simply:

```
import numpy as np

print(np.argmax(predictions[0]))
```

7

There's your prediction, let's look at the input:

```
plt.imshow(x_test[0],cmap=plt.cm.binary)
plt.show()
```



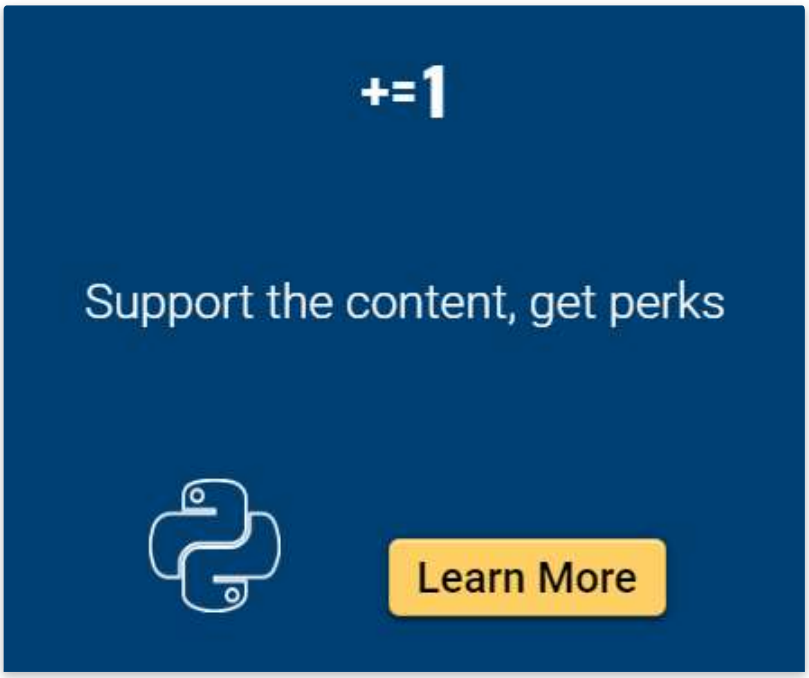
Awesome! Okay, I think that covers all of the "quick start" types of things with Keras. This is just barely scratching the surface of what's available to you, so start poking around Tensorflow and Keras documentation.

Also check out the Machine Learning and Learn Machine Learning subreddits to stay up to date on news and information surrounding deep learning.

If you have further questions too, you can join our Python Discord. Til next time.

The next tutorial:

Loading In Your Own Data - Deep Learning Basics With Python, TensorFlow And Keras P.2



Introduction to Deep Learning - Deep Learning basics with Python, TensorFlow and Keras p.1

Loading in your own data - Deep Learning basics with Python, TensorFlow and Keras p.2

Convolutional Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.3

Analyzing Models with TensorBoard - Deep Learning basics with Python, TensorFlow and Keras p.4

Optimizing Models with TensorBoard - Deep Learning basics with Python, TensorFlow and Keras p.5

How to use your trained model - Deep Learning basics with Python, TensorFlow and Keras p.6

Recurrent Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.7

Creating a Cryptocurrency-predicting finance recurrent neural network - Deep Learning basics with Python, TensorFlow and Keras p.8

Normalizing and creating sequences for our cryptocurrency predicting RNN - Deep Learning basics with Python, TensorFlow and Keras p.9

Balancing Recurrent Neural Network sequence data for our crypto predicting RNN - Deep Learning basics with Python, TensorFlow and Keras p.10

Cryptocurrency-predicting RNN Model - Deep Learning basics with Python, TensorFlow and Keras p.11

You've reached the end!

Contact: Harrison@pythonprogramming.net.

Support this Website!
Hire me
Facebook
Twitter
Google+

Legal stuff:

Terms and Conditions
Privacy Policy

Programming is a superpower.