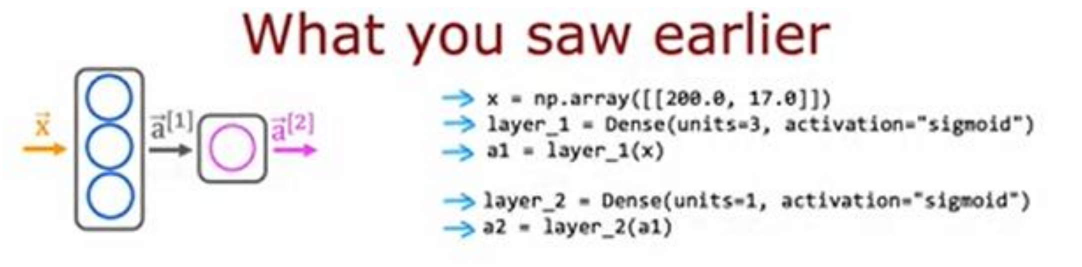
Building a neural network

So you've seen a bunch of tensor flow code by now learned about how to build a layer in tensor flow, how to do forward prop through a single layer in tensor flow. And also learned about data in TensorFlow. Let's put it all together and talk about how to build a neural network in TensorFlow. This is also the last video on tensor flow for this week.

And in this video you also learn about a different way of building a neural network, that will be even a little bit simpler than what you've seen so far. So let's dive in what you saw previously was. If you want to do forward prop, you initialize the data X create layer one then compute a one, then create layer two and compute a two.



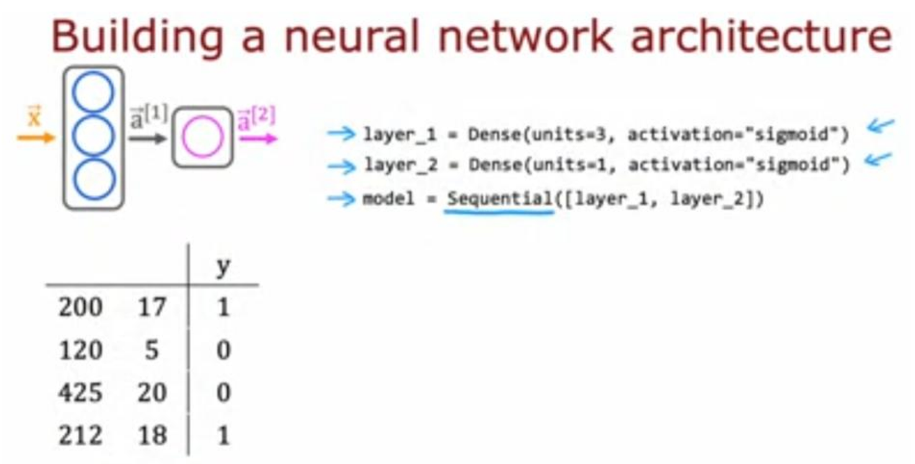
What you saw earlier

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation="sigmoid")
a1 = layer_1(x)

layer_2 = Dense(units=1, activation="sigmoid")
a2 = layer_2(a1)
```

So this was an explicit way of carrying out forward prop one layer of computation at the time. It turns out that tensor flow has a different way of implementing forward prop as well as learning. Let me show you a different way of building a neural network in TensorFlow, which is that same as before you're going to create layer one and create layer two.

But now instead of you manually taking the data and passing it to layer one and then taking the activations from layer one and pass it to layer two. We can instead tell tensor flow that we would like it to take layer one and layer two and string them together to form a neural network.

That's what the sequential function in TensorFlow does which is it says, Dear TensorFlow, please create a neural network for me by sequentially string together these two layers that I just created. It turns out that with the sequential framework tensor flow can do a lot of work for you.



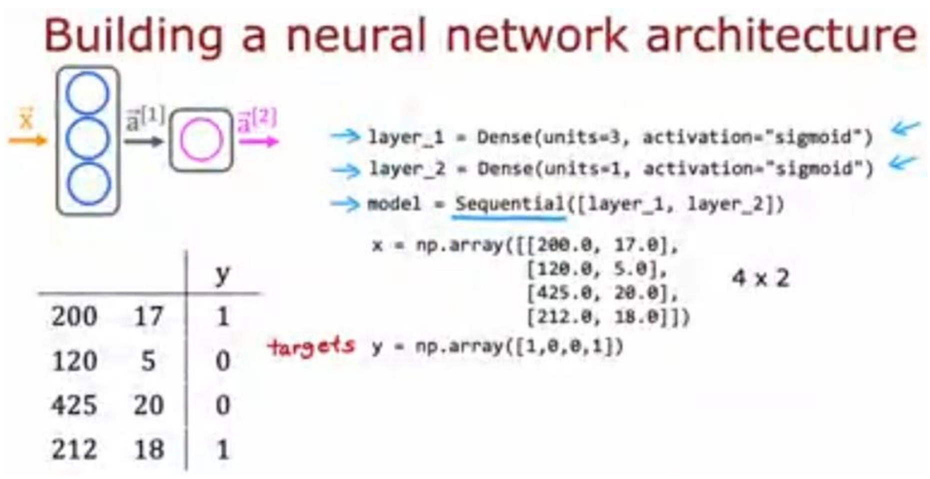
Building a neural network architecture

```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])
```

		y
200	17	1
120	5	0
425	20	0
212	18	1

Let's say you have a training set like this on the left. This is for the coffee example. You can then take the training data as inputs X and put them into a numpy array. This here is a four by two matrix and the target labels. Y can then be written as follows.

And this is just a one dimensional array of length four Y this set of targets can then be stored as a 1-D array like this 1001 corresponding to four train examples. And it turns out that given the data, X and Y stored in this matrix X and this array, Y.



Building a neural network architecture

```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])

x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

4 x 2

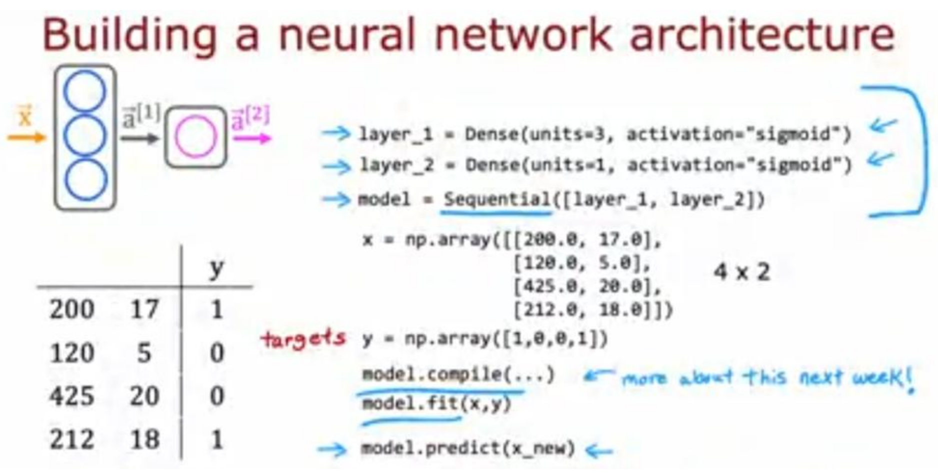
targets
```
y = np.array([1,0,0,1])
```

		y
200	17	1
120	5	0
425	20	0
212	18	1

If you want to train this neural network, all you need to do is call to functions you need to call model dot compile with some parameters. We'll talk more about this next week, so don't worry about it for now. And then you need to call model dot fit X Y, which tells tensor flow to take this neural network that are created by sequentially string together layers one and two, and to train it on the data, X and Y.

But we'll learn how but we'll learn the details of how to do this next week and then finally how do you do inference on this neural network? How do you do forward prop if you have a new example, say X new, which is NP array with these two features than to carry out forward prop instead of having to do it one layer at a time yourself, you just have to call model predict on X new and this will output the corresponding value of a two for you given this input value of X.

So model predicts carries out forward propagation and carries an inference for you, using this neural network that you compiled using the sequential function. Now I want to take these three lines of code on top and just simplify it a little bit further, which is when coding in Tensorflow.



Building a neural network architecture

```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])

x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

4 x 2

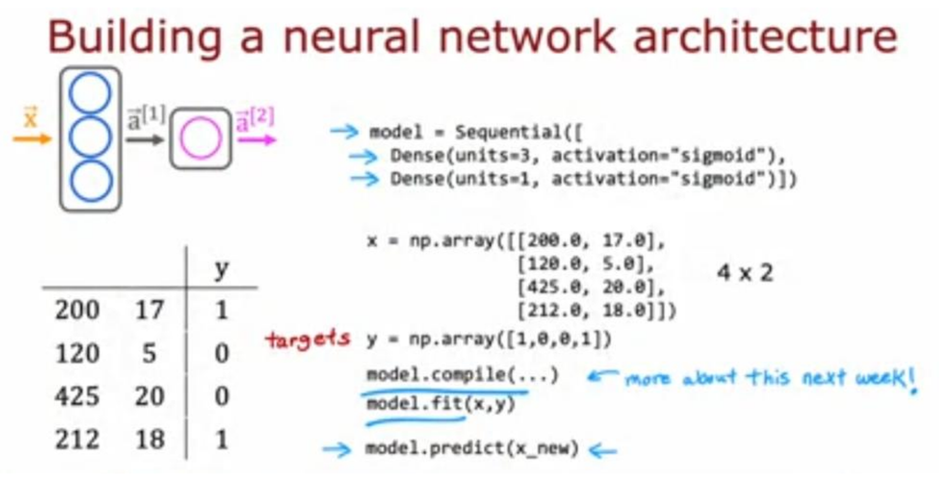
targets
```
y = np.array([1,0,0,1])
model.compile(...)
model.fit(x,y)

model.predict(x_new)
```

more about this next week!

		y
200	17	1
120	5	0
425	20	0
212	18	1

By convention we don't explicitly assign the two layers to two variables, layer one and layer two as follows. But by convention I would usually just write a code like this, when we say the model is a sequential model of a few layers strung together. Sequentially where the first layer one is a dense layer with three units and activation of sigmoid and the second layer, is a dense layer with one unit and again a sigmoid activation function.

So if you look at others tensor flow code, you often see it look more like this rather than having an explicit assignment to these layer one and layer two variables. And so that's it. This is pretty much the code you need in order to train as well as to inference on a neural network in TensorFlow. Where again we'll talk more about the training bits of this two combined the compiler and the fit function next week.



Building a neural network architecture

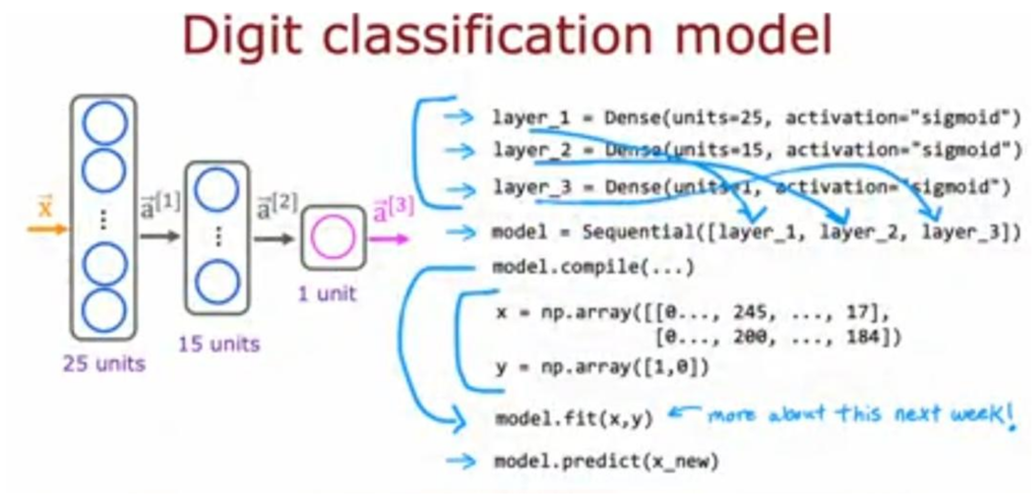
		y
200	17	1
120	5	0
425	20	0
212	18	1

```
model = Sequential([
  Dense(units=3, activation="sigmoid"),
  Dense(units=1, activation="sigmoid")])

x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
y = np.array([1,0,0,1])
model.compile(...)
model.fit(x,y)

model.predict(x_new)
```

Let's redo this for the digit classification example as well. So previously we had X, in this input layer one is a layer a one equals. They want to apply to X and so on through layer two and layer three in order to try to classify a digit, with this new coding convention with using tensor flow sequential function, you can instead specify what are layer one, layer two, layer three and tell tensor flow to string the layers together for you into a new network and same as before.

You can then store the data in the matrix and run the compile function and fit the model as follows. Again, more on this next week. Finally to do inference or to make predictions you can use model predict on X new and similar to what you saw before with the coffee classification network by convention, instead of assigning layer one, layer two, layer three, explicitly like this, we would more commonly just take these layers and put them directly into the sequential function.

So you end up with this more compact code which just tell tensor flow, create a model for me that sequentially strings together these three layers and then the rest of the code works the same as before.



Digit classification model

```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)
x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])
model.fit(x,y)
model.predict(x_new)
```

So that's how you have built a neural network in TensorFlow. Now I know that when you're learning about these techniques, sometimes someone may ask you to implement these five lines of code and then you type five lines of code and then someone says congratulations with just five lines of code. You built this crazy complicated state of the art neural network and sometimes that makes you wonder, what exactly did I do with just these five lines of codes?

One thing I want you to take away from the machine learning specialization is the ability to use cutting edge libraries like tensor flow to do your work efficiently. But I don't really want you to just call five lines of code and not really also know what the code is actually doing underneath the hood. So in the next video I'll let you go back and share with you how you can implement from scratch by yourself. forward propagation in python, so that you can understand the whole thing for yourself in practice.

Most machine learning engineers don't actually implement forward propagation in python that often we just use libraries like tensorflow and pytorch, but because I want you to understand how these algorithms work yourself so that if something goes wrong, you can think through for yourself, what you might need to change, what's likely to work, what's less likely to work.

Let's also go through what it would take for you to implement for propagation from scratch because that way, even when you're calling a library and having it run efficiently and do great things in your application, I want you in the back of your mind to also have that deeper understanding of what your code is actually doing, so that let's go on to the next video.