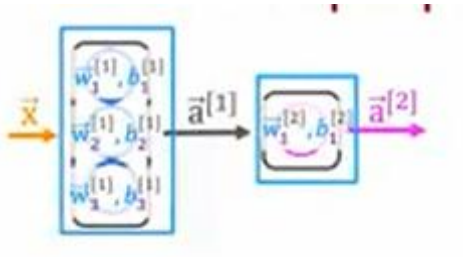
Etai before in a single layer

if you had to implement forward propagation yourself from scratch in python, how would you go about doing so, in addition to gaining intuition about what's really going on in libraries like TensorFlow and PyTorch. If ever some day you decide you want to build something even better than TensorFlow and PyTorch, maybe now you have a better idea home, I don't really recommend doing this for most people.

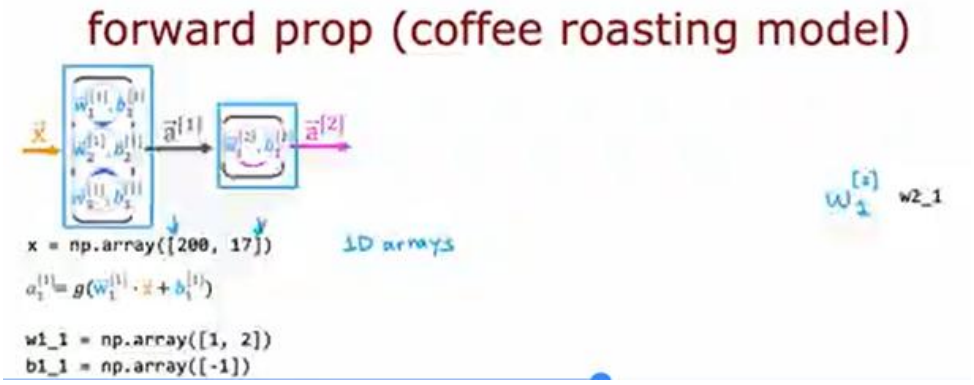
But maybe someday, someone will come up with an even better framework than TensorFlow and PyTorch and whoever does that may end up having to implement these things from scratch themselves. So let's take a look, on this slide I'm going to go through quite a bit of code and you see all this code again later in the optional lab as was in the practice lab. So don't worry about having to take notes on every line of code or memorize every line of code.

You see this code written down in the Jupiter notebook in the lab and the goal of this video is to just show you the code to make sure you can understand what it's doing. So that when you go to the optional lab and the practice lab and see the code there, you know what to do so don't worry about taking detailed notes on every line.



If you can read through the code on this slide and understand what it's doing, that's all you need. So let's take a look at how you implement forward prop in a single layer, we're going to continue using the coffee roasting model shown here. And let's look at how you would take an input feature vector x, and implement forward prop to get this output a2.

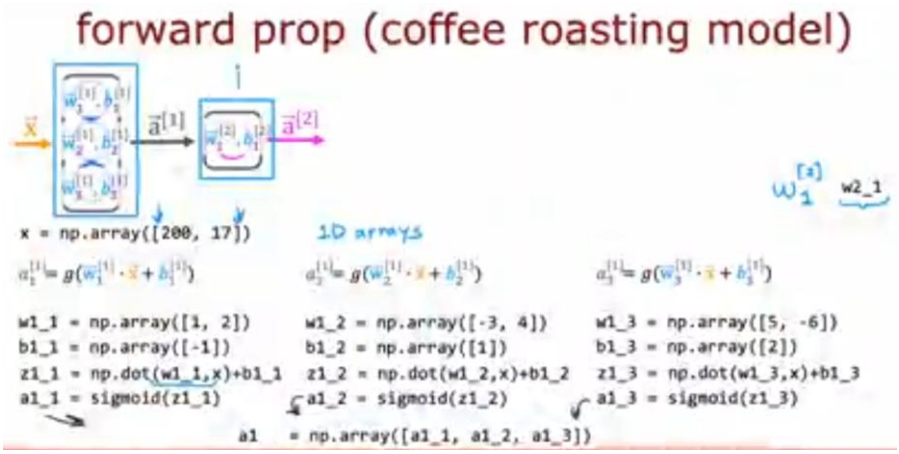
In this python implementation, I'm going to use 1D arrays to represent all of these vectors and parameters, which is why there's only a single square bracket here. This is a 1D array in python rather than a 2D matrix, which is what we had when we had double square brackets. So the first value you need to compute is, a super strip square bracket 1 subscript 1, which is the first activation value of a1 and that's g of this expression over here. So I'm going to use the convention on this slide that at a term like w2, 1, I'm going to represent as a variable w2 and then subscript 1.



This underscore one denotes subscript one, denotes subscript one so w2 means w superscript 2 in square brackets and then subscript 1. So, to compute a1_1, we have parameters w1_1 and b1_1, which are say 1_2 and -1. You would then compute z1_1 as the dot product between that parameter w1_1 and the input x, and added to b1_1 and then finally a1_1 is equal to g, the sigmoid function applied to z1_1.

Next let's go on to compute a1_2, which again by the convention I described here is going to be a1_2, written like that. So similar as what we did on the left, w1_2 is two parameters -3, 4, b1_2 is the term, b 1, 2 over there, so you compute z as this term in the middle and then apply the sigmoid function and then you end up with a 1_2, and finally you do the same thing to compute a1_3.

Now, you've computed these three values, a1_1, a1_2, and a1_3, and we like to take these three numbers and group them together into an array to give you a1 up here, which is the output of the first layer. And so you do that by grouping them together using a np array as follows, so now you've computed a_1, let's implement the second layer as well.



So you compute, the output a2, so a2 is computed using this expression and so we would have parameters w2_1 and b2_1 corresponding to these parameters. And then you would compute z as the dot product between w2_1 and a1, and add b2_1 and then apply the sigmoid function to get a2_1 and that's it, that's how you implement forward prop using just python and np.

Now, there are a lot of expressions in this page of code that you just saw, let's in the next video look at how you can simplify this to implement forward prop for a more general neural network, rather than hard coding it for every single neuron like we just did.