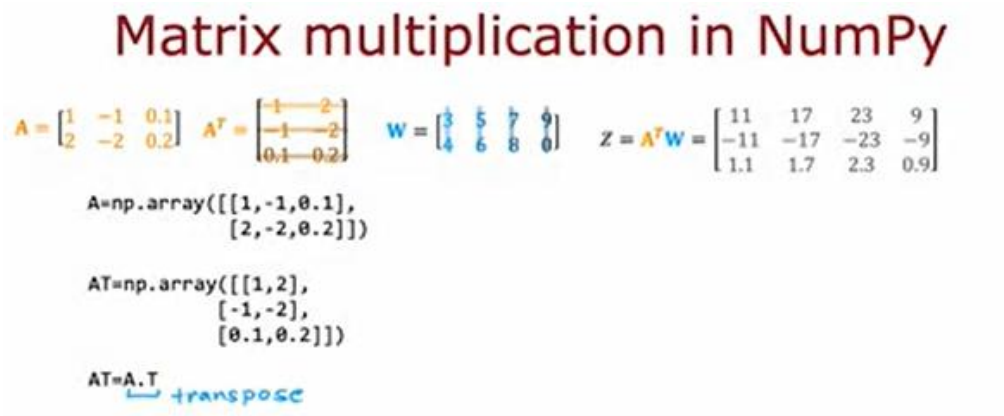
Matrix multiplication code

Without further ado, let's jump into the vectorize implementation of a neural network. We'll look at the code that you have seen in a earlier video, and hopefully, Matmul, that is that matrix multiplication calculation, will make more sense. Let's jump in. You saw previously how you can take the matrix A and compute A transpose times W resulting in this matrix here, Z.

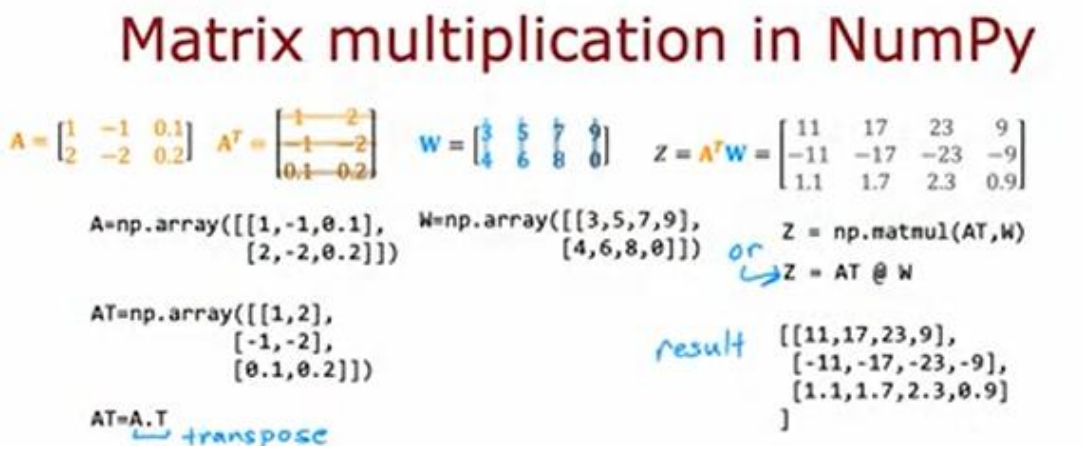
In code if this is the matrix A, this is a NumPy array with the elements corresponding to what I wrote on top, then A transpose, which I'm going to write as AT, is going to be this matrix here, with again the columns of A now laid out in rows instead.

By the way, instead of setting up AT this way, another way to compute AT in NumPy, we will write AT equals A.T. That's the transpose function that takes the columns of a matrix and lays them on the side.



In code, here's how you initialize the matrix W as another 2D NumPy array. Then to compute Z equals A transpose times W, you will write Z equals np.matmul, AT, W, and that will compute this matrix Z over here, giving you this result down here.

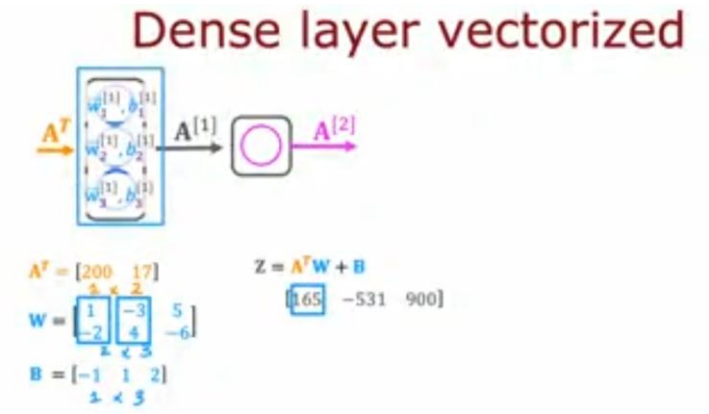
By the way, if you read other's code, sometimes you see Z equals AT and then the @ W. This is an alternative way of calling the matmal function. Although I find using np.matmul to be clearer. The call you see in this class, we just use the matmal function like this rather than this @. Let's look at what a vectorized implementation of forward prop looks like.



I'm going to set A transpose to be equal to the input feature values 217. These are just the usual input feature values, 200 degrees roasting coffee for 17 minutes. This is a one by two matrix. I'm

going to take the parameters w_1, w_2, and w_3, and stack them in columns like this to form this matrix W. The values b_1, b_2, b_3, I'm going to put it into a one by three matrix, that is this matrix B as follows.

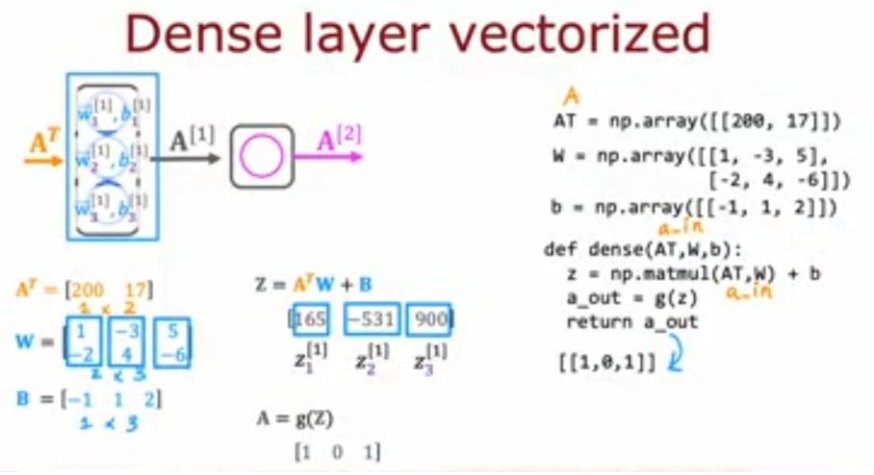
Then it turns out that if you were to compute Z equals A transpose W plus B, that will result in these three numbers and that's computed by taking the input feature values and multiplying that by the first column and then adding B to get 165. Taking these feature values, dot-producting with the second column, that is a weight w_2 and adding b_2 to get negative 531.



These feature values dot product with the weights w_3 plus b_3 to get 900. Feel free to pause the video if you wish to double-check these calculations. But this gives you is the values of z^1_1, Z^1_2, and Z^1_3.

Then finally, if the function g applies the sigmoid function to these three numbers element-wise, that is, applies the sigmoid function to 165, to negative 531, and to 900, then you end up with A equals g of this matrix Z ends up being 1,0,1.

It's 1,0,1 because sigmoid of 165 is so close to one that up to numerical round off is based to one and these are bases 0 and 1. Let's look at how you implement this in code. A transpose is equal to this, is this one by two array of 217.



The matrix W is this two by three matrix, and B, this is one by three matrix. The way you can implement forward prop in a layer is dense input A transpose W b is equal to z equals matmul A transpose times W plus b. That just implements this line of code.

Then a_out that is the output of this layer is equal to g, the activation function applied element-wise to this matrix Z. You return a_out, and that gives you this value. In case you're comparing this

slide with the slide a few videos back, there was just one little difference, which was by convention, the way this is implemented in TensorFlow, rather than calling this variable A,T, we were calling it A_in, which is why this too is the correct implementation of the code.

There is a convention in TensorFlow that individual examples are actually laid out in rows in the matrix X rather than in the matrix X transpose which is why the code implementation actually looks like this in TensorFlow.

But this explains why with just a few lines of code you can implement forward prop in the neural network and moreover, get a huge bonus because modern computers are very good at implementing matrix multiplications such as matmul efficiently. That's the last video this week.