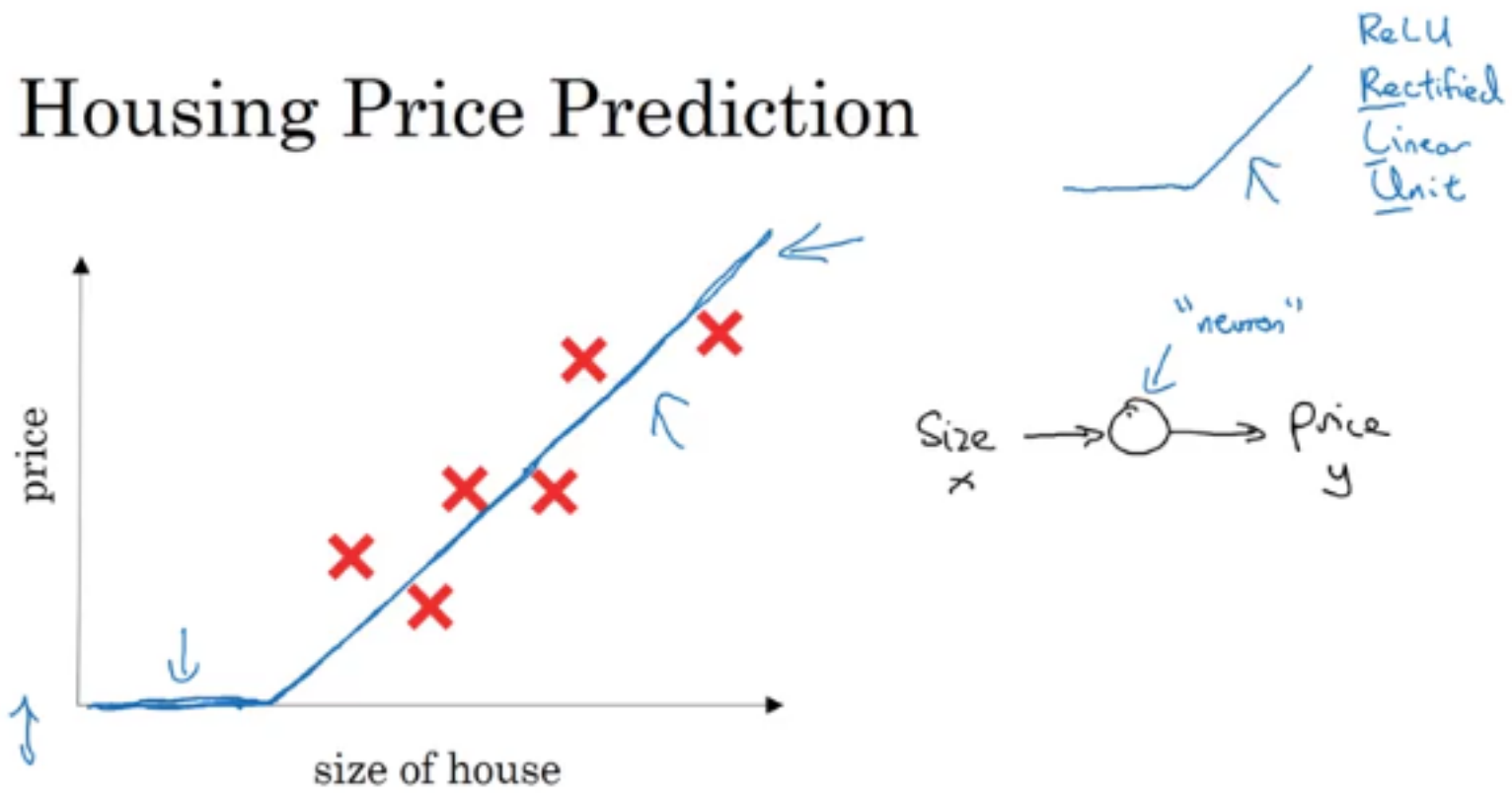
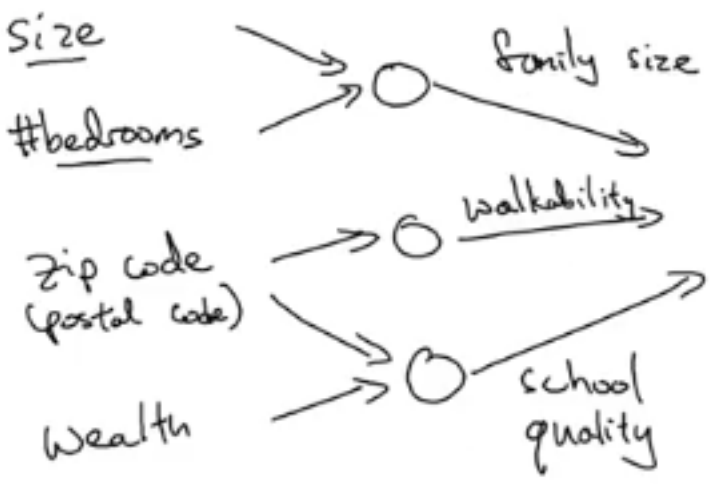


This function which goes to zero sometimes and then it'll takes of as a straight line. This function is called a ReLU function which stands for rectified linear units. So R-E-L-U. And rectify just means taking a max of 0 which is why you get a function shape like this.

You don't need to worry about ReLU units for now but it's just something you see again later in this course. So if this is a single neuron, neural net-work, really a tiny little neural network, a larger neural network is then formed by taking many of the single neurons and stacking them together.

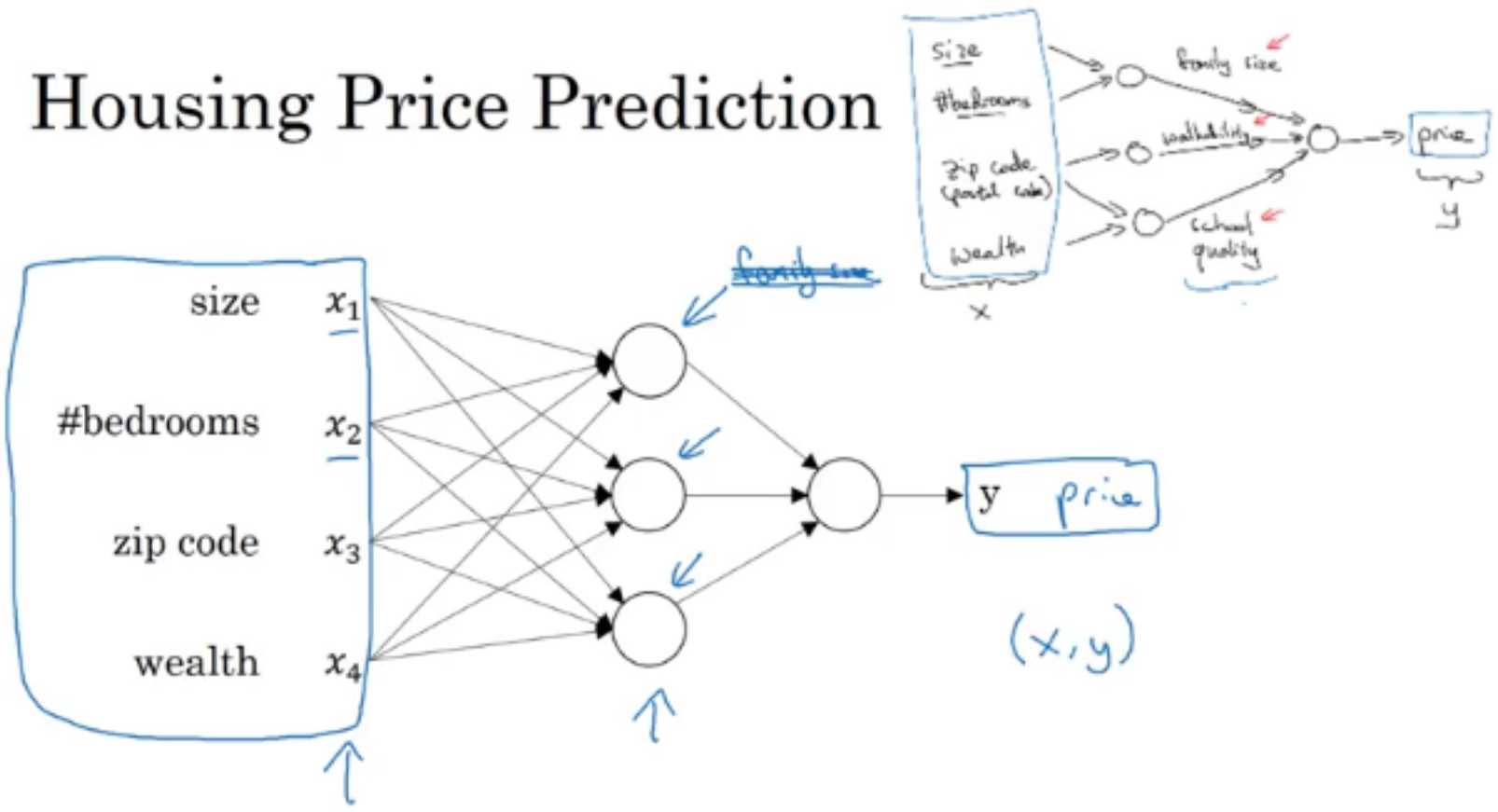
So, if you think of this neuron that's



you how good is the school quality. So each of these little circles I'm drawing, can be one of those ReLU, rectified linear units or some other slightly non linear function.

So that based on the size and number of bedrooms, you can estimate the family size, their zip code, based on walkability, based on zip code and wealth can estimate the school quality.

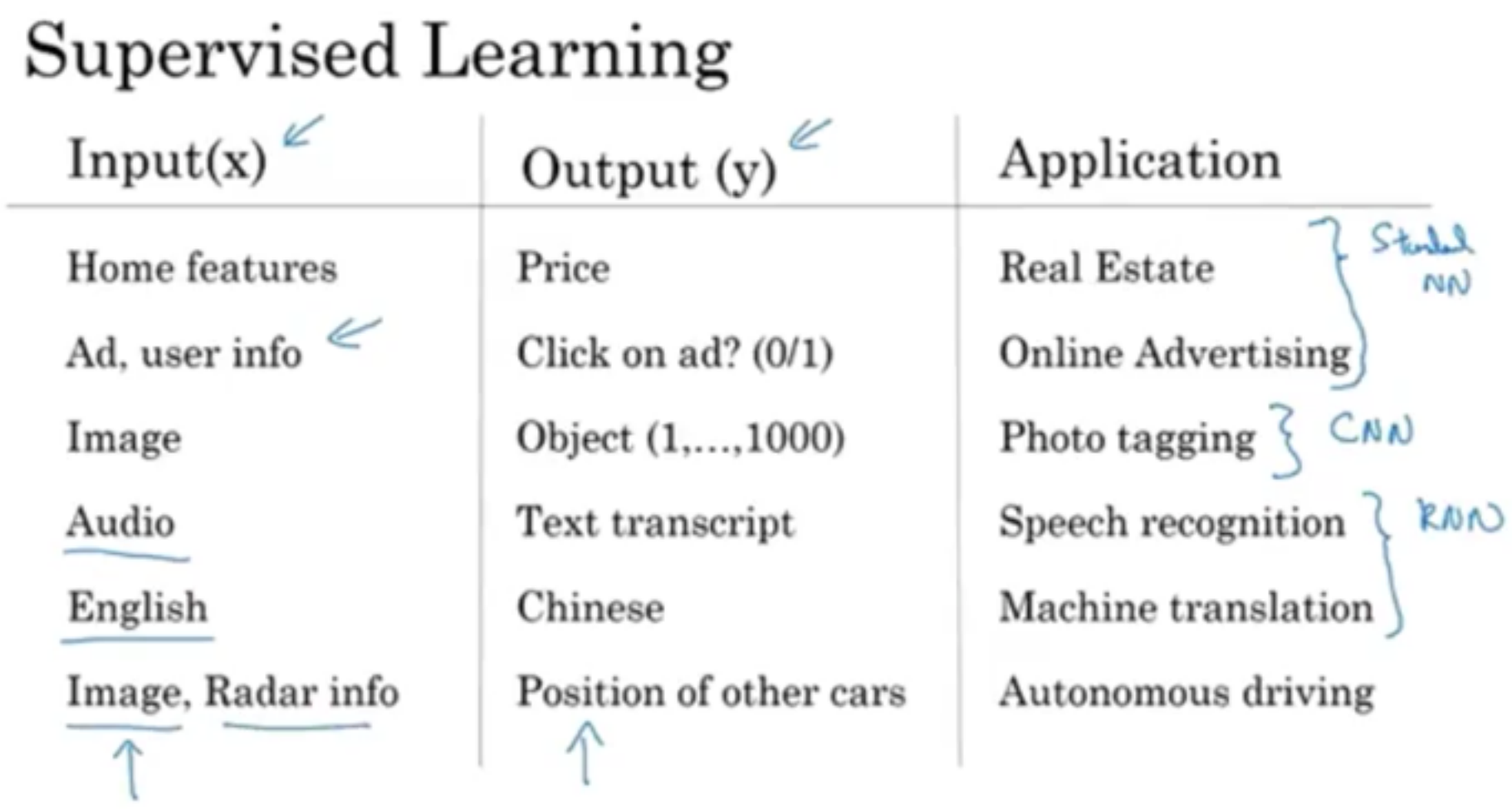
And then finally you might think that well the way people decide how much they're will to pay for a house, is they look at the things that really matter to them. In this case family size, walkability and school quality and that



And the remarkable thing about neural networks is that, given enough data about x and y, given enough training examples with both x and y, neural networks are remarkably good at figuring out functions that accurately map from x to y. So, that's a basic neural network.

In turns out that as you build out your own neural networks, you probably find them to be most useful, most powerful in supervised learning incentives, meaning that you're trying to take an input x and map it to some output y, like we just saw in the housing price prediction example.

In the next video let's go over some



Chinese, the alphabets or the words come one at a time.

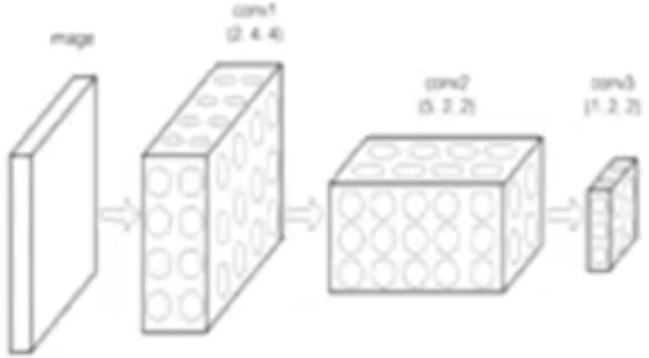
So language is also most naturally represented as sequence data. And so more complex versions of RNNs are often used for these applications.

And then, for more complex applications, like autonomous driving, where you have an image, that might suggest more of a CNN convolution neural network structure and radar info which is something quite different.

You might end up with a more custom, or some more complex, hybrid neural network architecture. So, just to be a bit more concrete about what are the

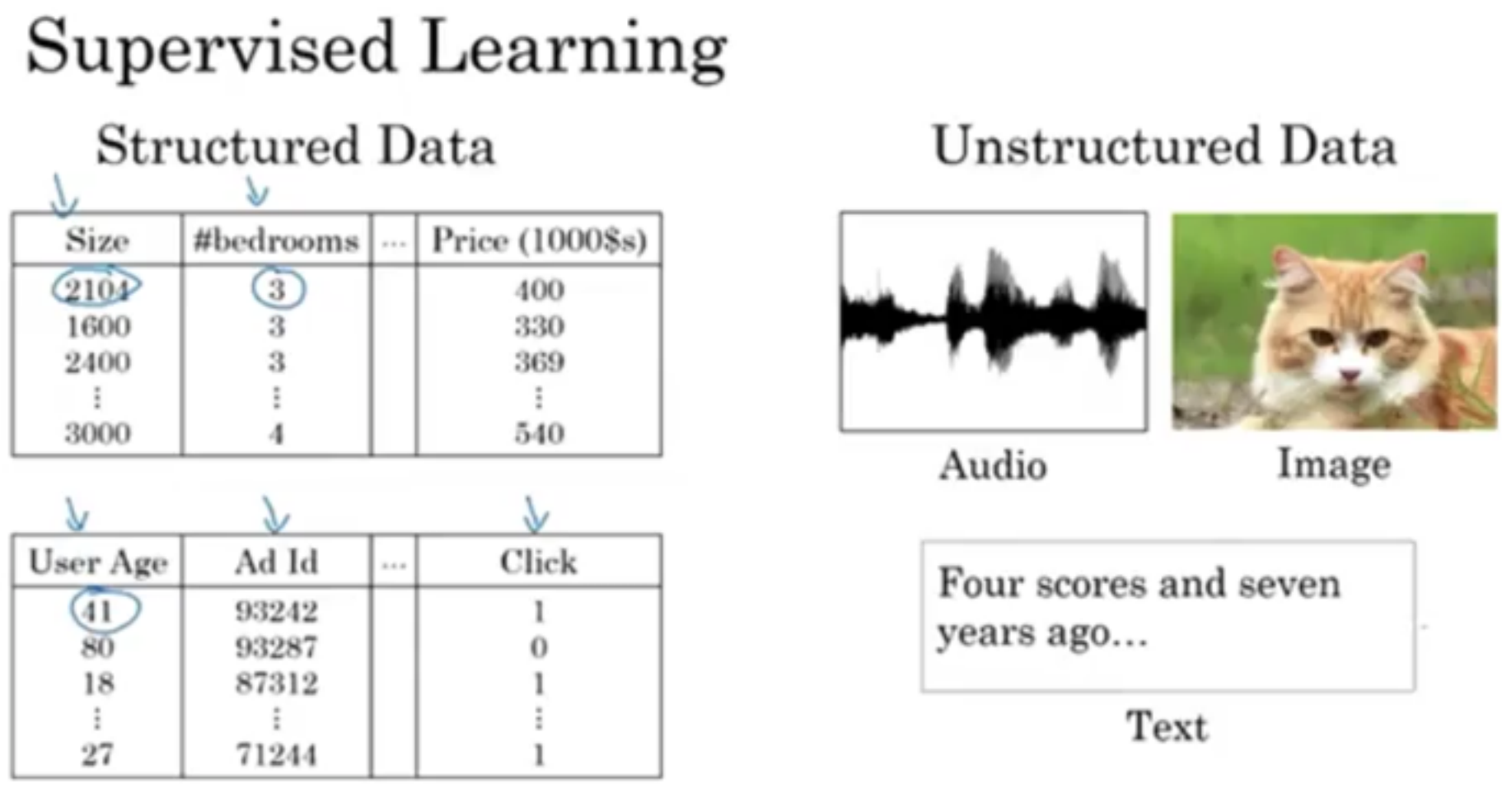
Neural Network examples

Standard NN



Convolutional NN

Recurrent NN

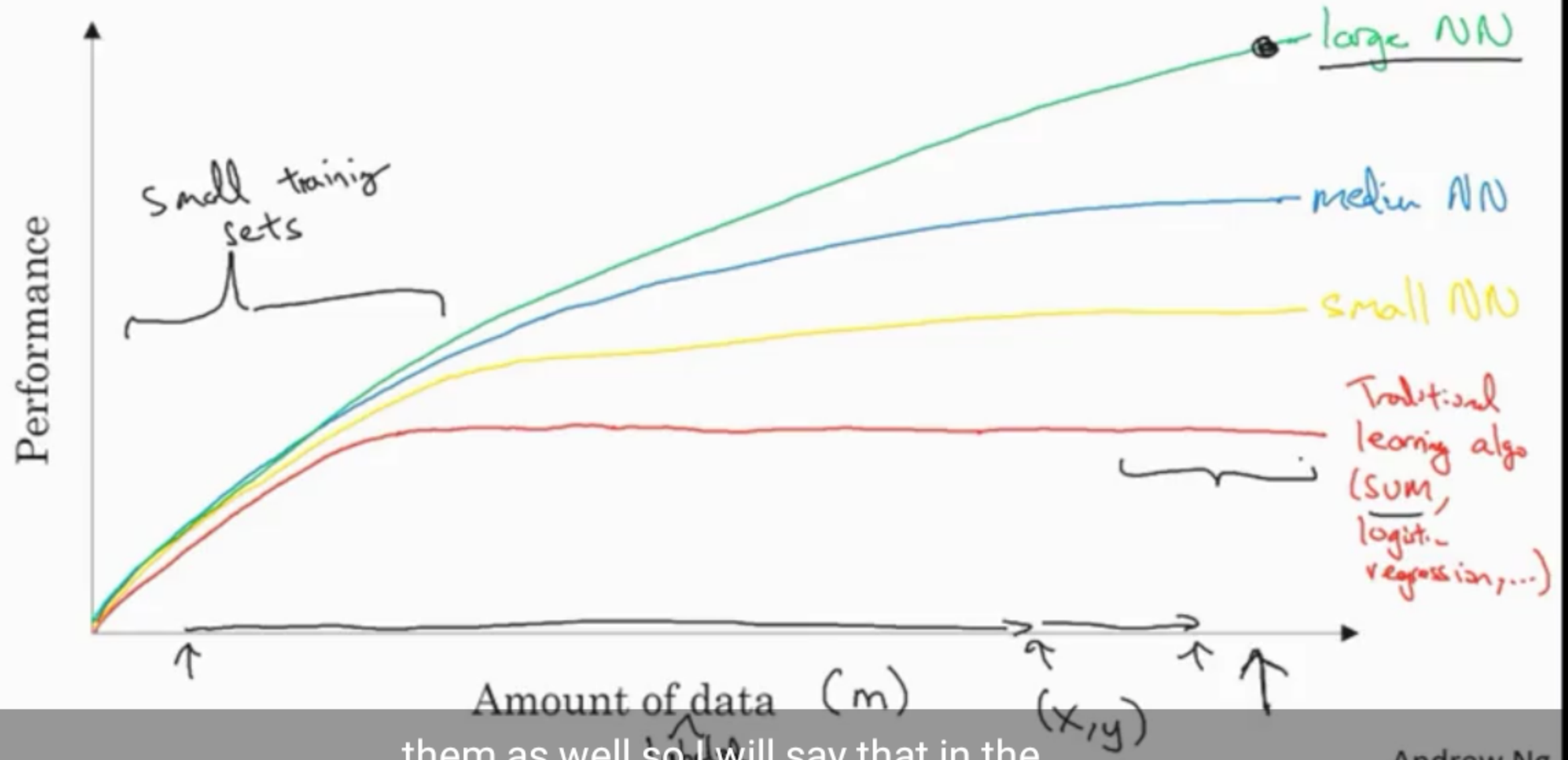


Historically, it has been much harder for computers to make sense of unstructured data compared to structured data. And the fact the human race has evolved to be very good at understanding audio cues as well as images.

And then text was a more recent invention, but people are just really good at interpreting unstructured data.

And so one of the most exciting things about the rise of neural networks is that, thanks to deep learning, thanks to neural networks, computers are now much better at interpreting unstructured data as well

Scale drives deep learning progress



them as well so I will say that in the

Andrew Ng

I hope you enjoyed this course and
to help you complete it I want to
make sure that there are few course
resources that you know about first if
you have any questions or you want to
discuss anything with the classmates
or the teaching staff including me
or if you want to file a bug report the
best place to do that is the discussion
forum the teaching staff and I will
be monitoring that regularly and
this is also a good place for you to
get answers to your questions from
your classmates or if you wish try to
answer your classmates questions
to get to the discussion forum from

Logistic regression on m examples

$J=0$; $dw_1=0$; $dw_2=0$; $db=0$

For $i=1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})\right]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

dw_3 : dw_n

$db \mathrel{+}= dz^{(i)}$

$n=2$

$J /= m$

$dw_1 /= m$; $dw_2 /= m$; $db /= m$.

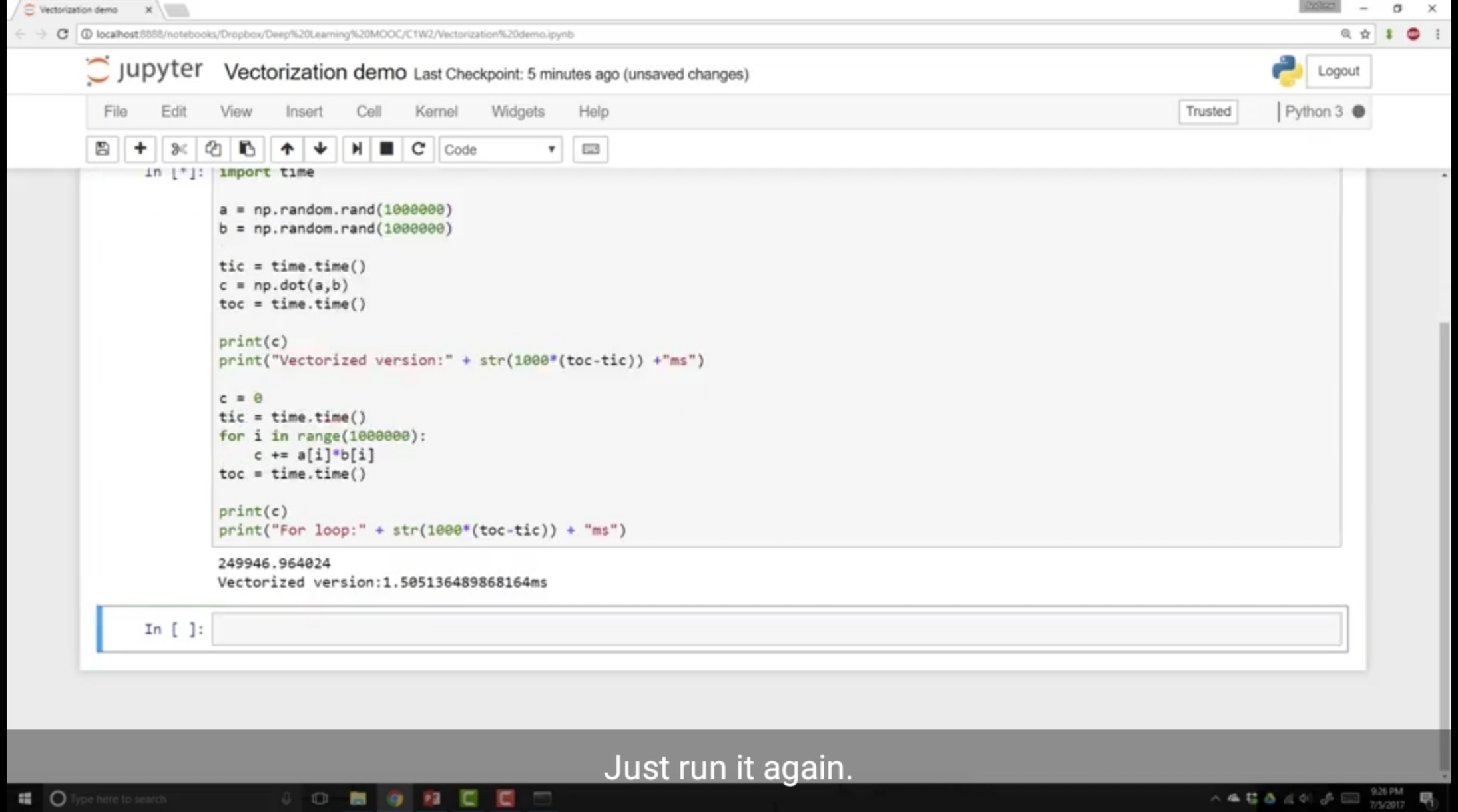
$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha\, dw_1$$

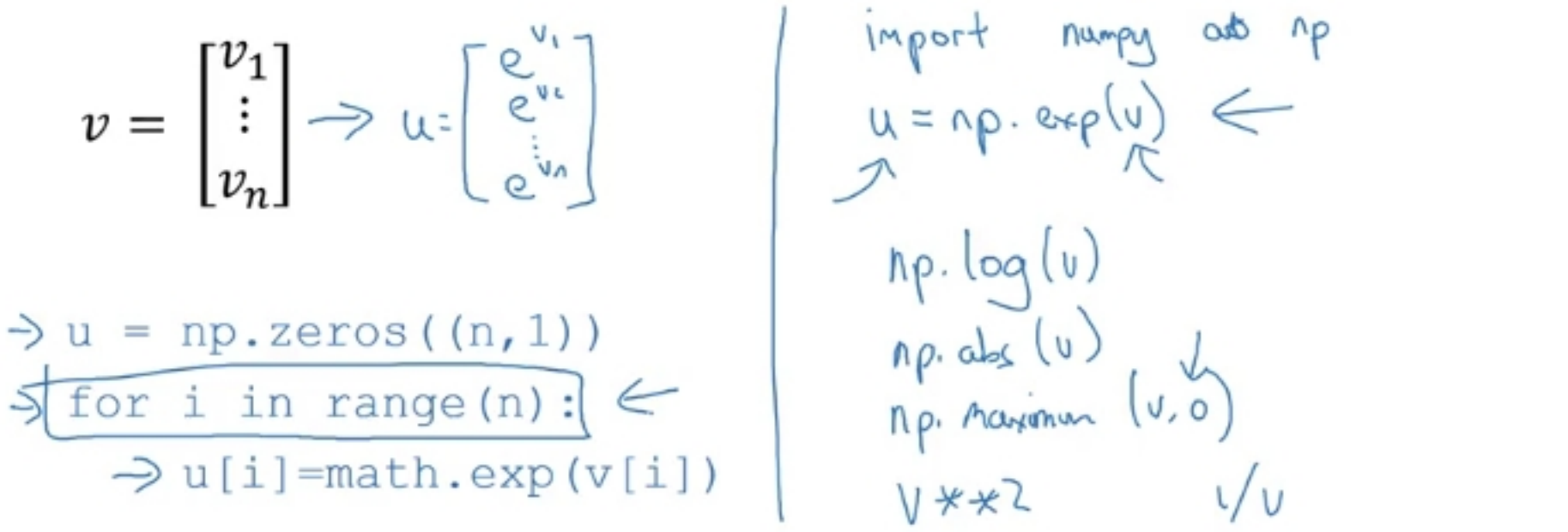
$$w_2 := w_2 - \alpha\, dw_2$$

$$b := b - \alpha\, db.$$

Andrew Ng



Just run it again.



Andrew Ng

the vector value functions.

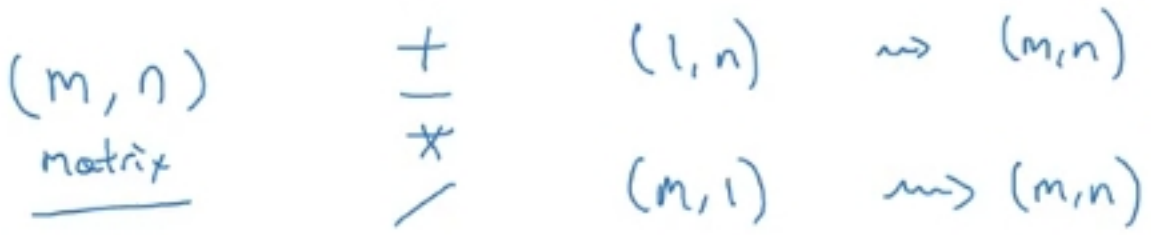
So np.log (v) will compute the element-wise log, np.abs computes the absolute value, np.maximum computes the element-wise maximum to take the max of every element of v with 0. v**2 just takes the element-wise square of each element of v. One over v takes the element-wise inverse, and so on.

So, whenever you are tempted to write a for-loop take a look, and see if there's a way to call a NumPy built-in function to do it without that for-loop.

So, let's take all of these learnings and apply it to our logisti regression

Broadcasting in Python

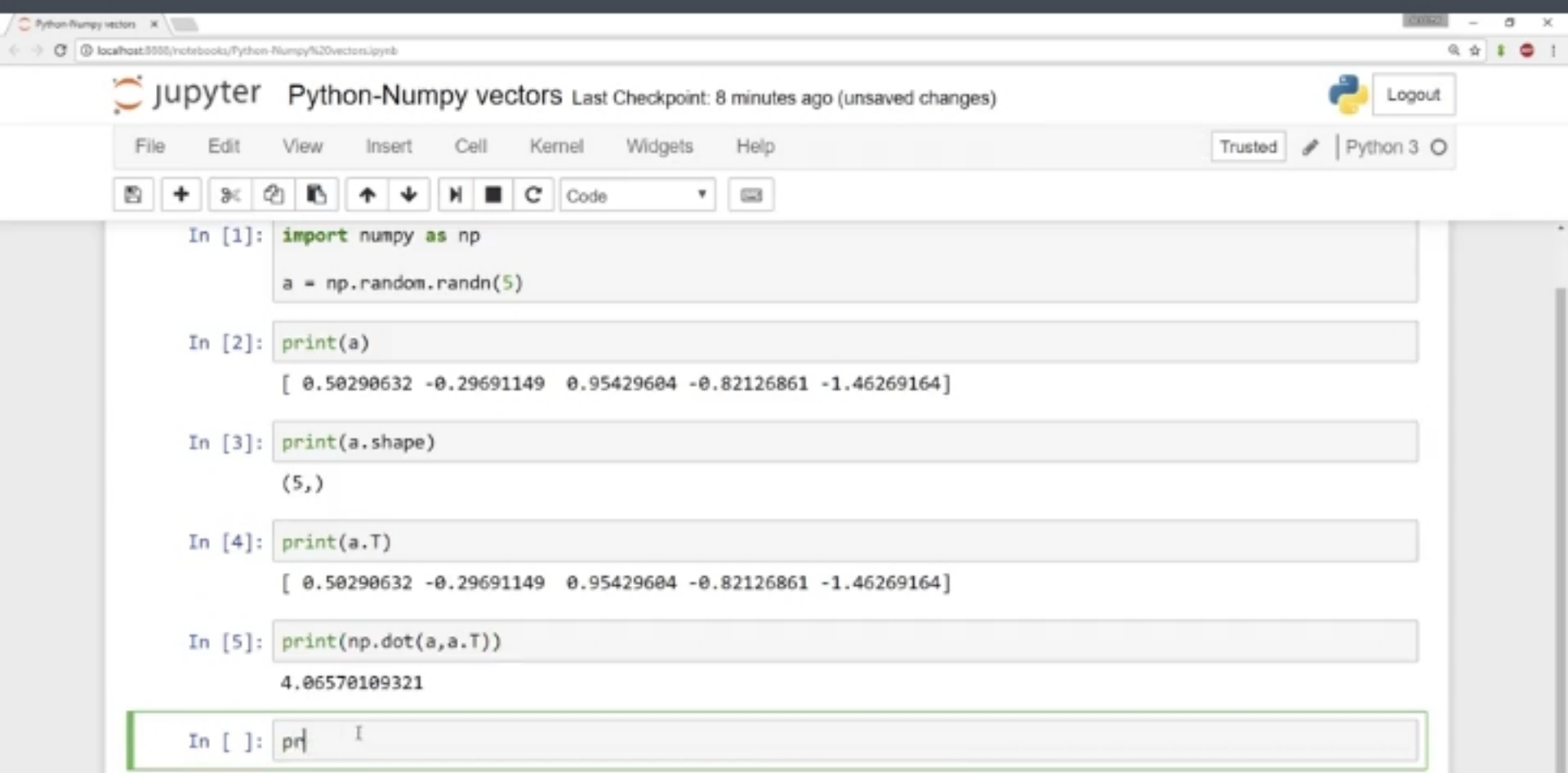
General Principle



If you have an (m,n) matrix and you add or subtract or multiply or divide with a (1,n) matrix, then this will copy it n times into an (m,n) matrix. And then apply the addition, subtraction, and multiplication of division element wise.

If conversely, you were to take the (m,n) matrix and add, subtract, multiply, divide by an (m,1) matrix, then also this would copy it now n times. And turn that into an (m,n) matrix and then apply the operation element wise.

Just one of the broadcasting, which is if you have an (m,1) matrix, so that's really a column vector like [1,2,3], and



So what I would recommend is that when you're coding new networks, that you just not use data structures where the shape is 5, or n, rank 1 array. Instead, if you set a to be this, (5,1), then this commits a to be (5,1) column vector.

And whereas previously, a and a transpose looked the same, it becomes now a transpose, now a transpose is a row vector. Notice one subtle difference. In this data structure, there are two square brackets when we print a transpose. Whereas previously, there was one square bracket.

Python/numpy vectors

```
a = np.random.randn(5)
```

a.shape = (5,)

"rank 1 array"

} Don't use

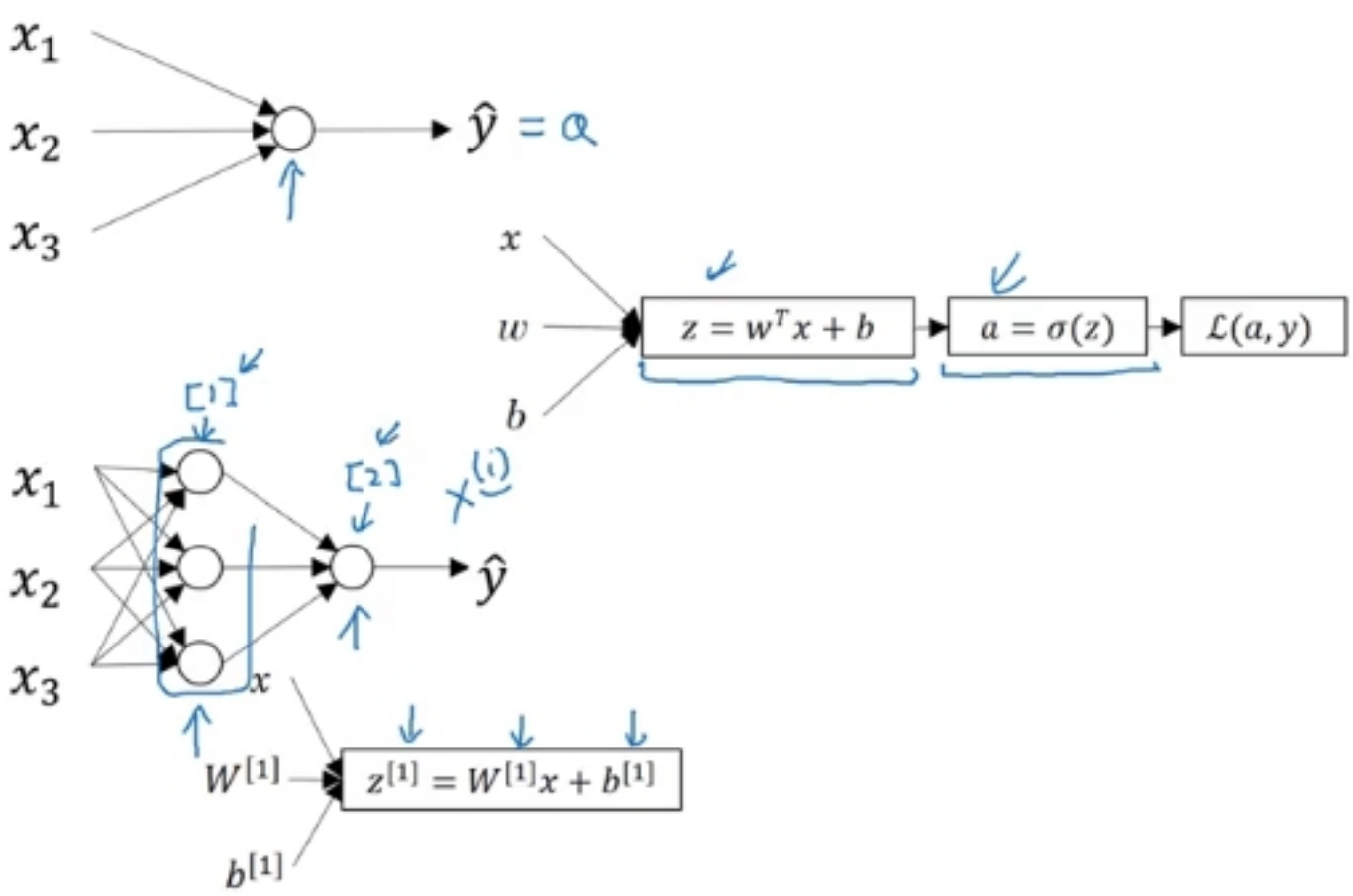
```
a = np.random.randn(5,1)
```
→ a.shape = (5,1) column vector ✓

```
a = np.random.randn(1,5)
```
→ a.shape = (1,5) row vector ✓

```
assert(a.shape == (5,1))
```
←

a = a.reshape((5,1))

What is a Neural Network?



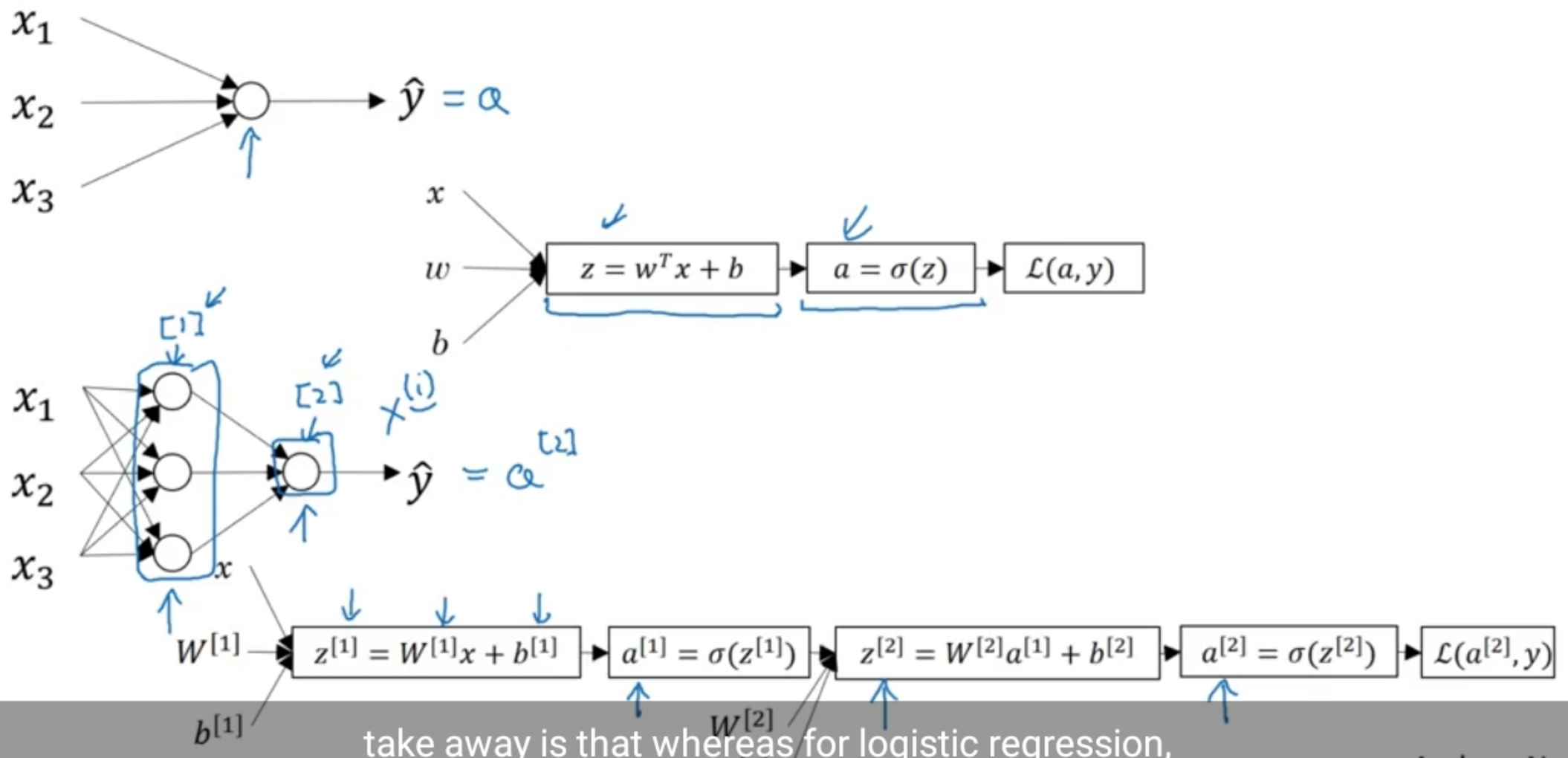
Andrew Ng

The superscript square brackets, like we have here, are not to be confused with the superscript round brackets which we use to refer to individual training examples.

So, whereas x superscript round bracket I refer to the ith training example, superscript square bracket one and two refer to these different layers; layer one and layer two in this neural network.

But so going on, after computing z_1 similar to logistic regression, there'll be a computation to compute a_1, and that's just sigmoid of z_1, and then you compute z_2 using another linear

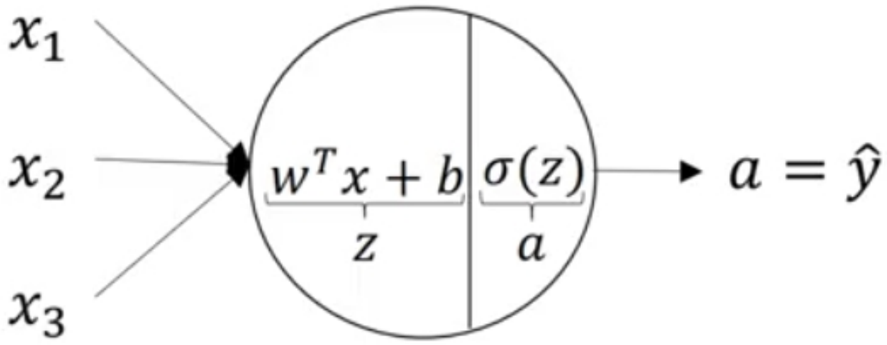
What is a Neural Network?



take away is that whereas for logistic regression,

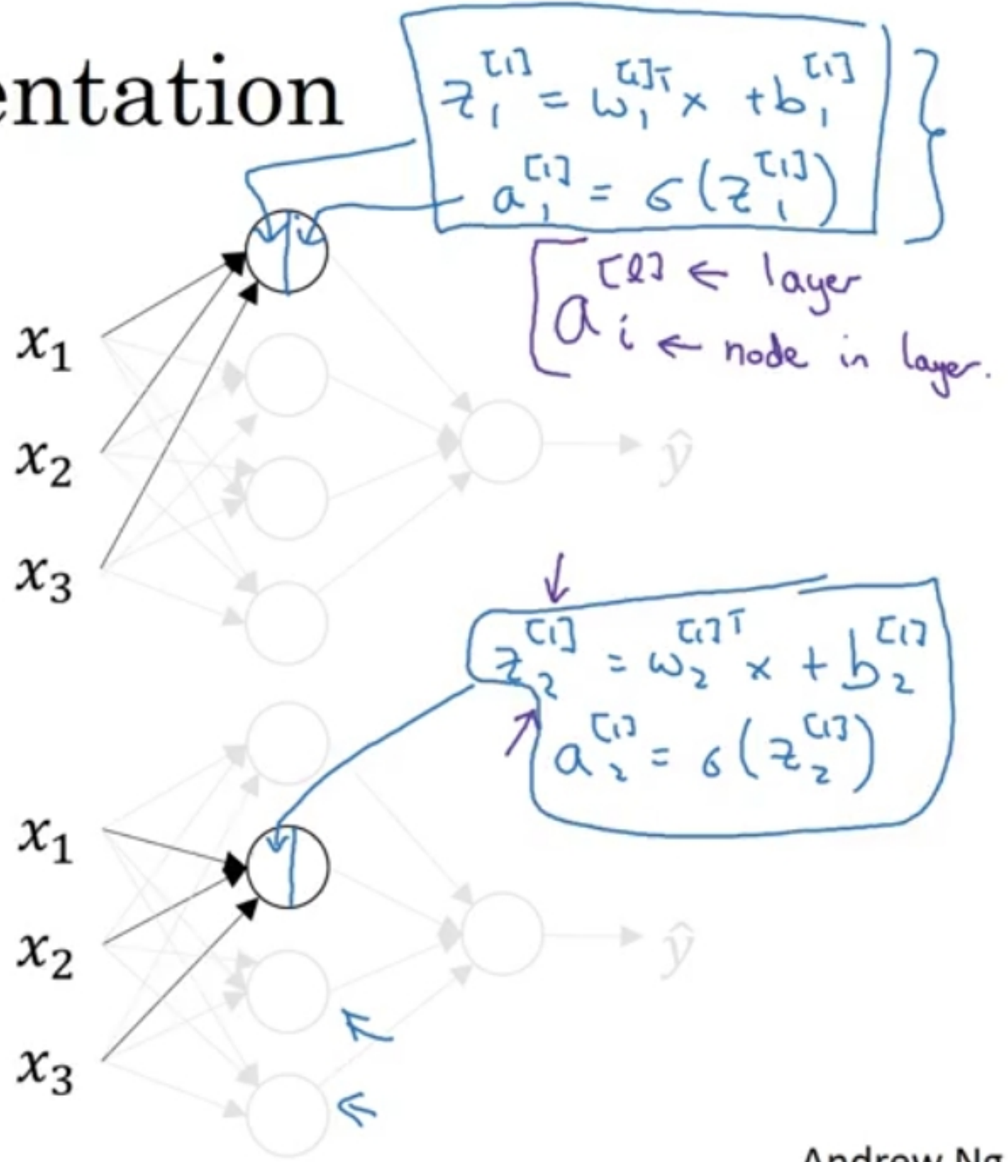
Andrew Ng

Neural Network Representation



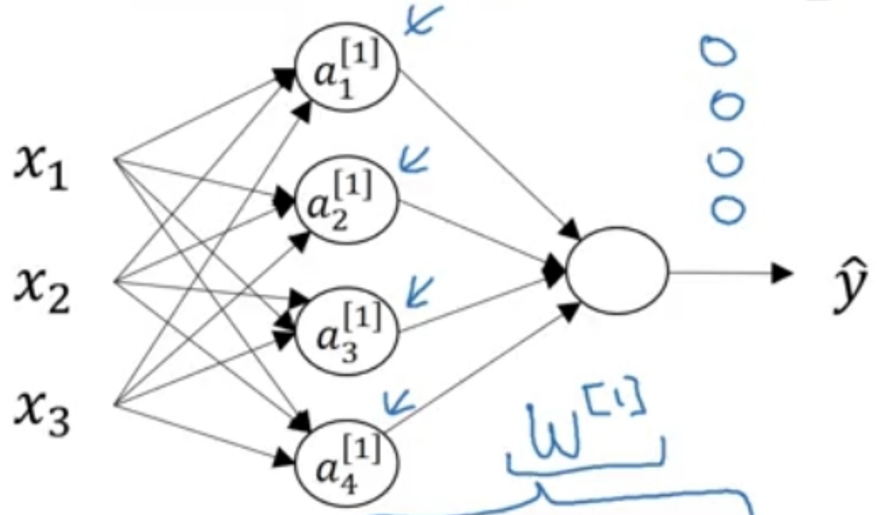
$$z = w^T x + b$$

$$a = \sigma(z)$$



Andrew Ng

Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$(w_1^{[1]})^T x$ $a^{[1]}$

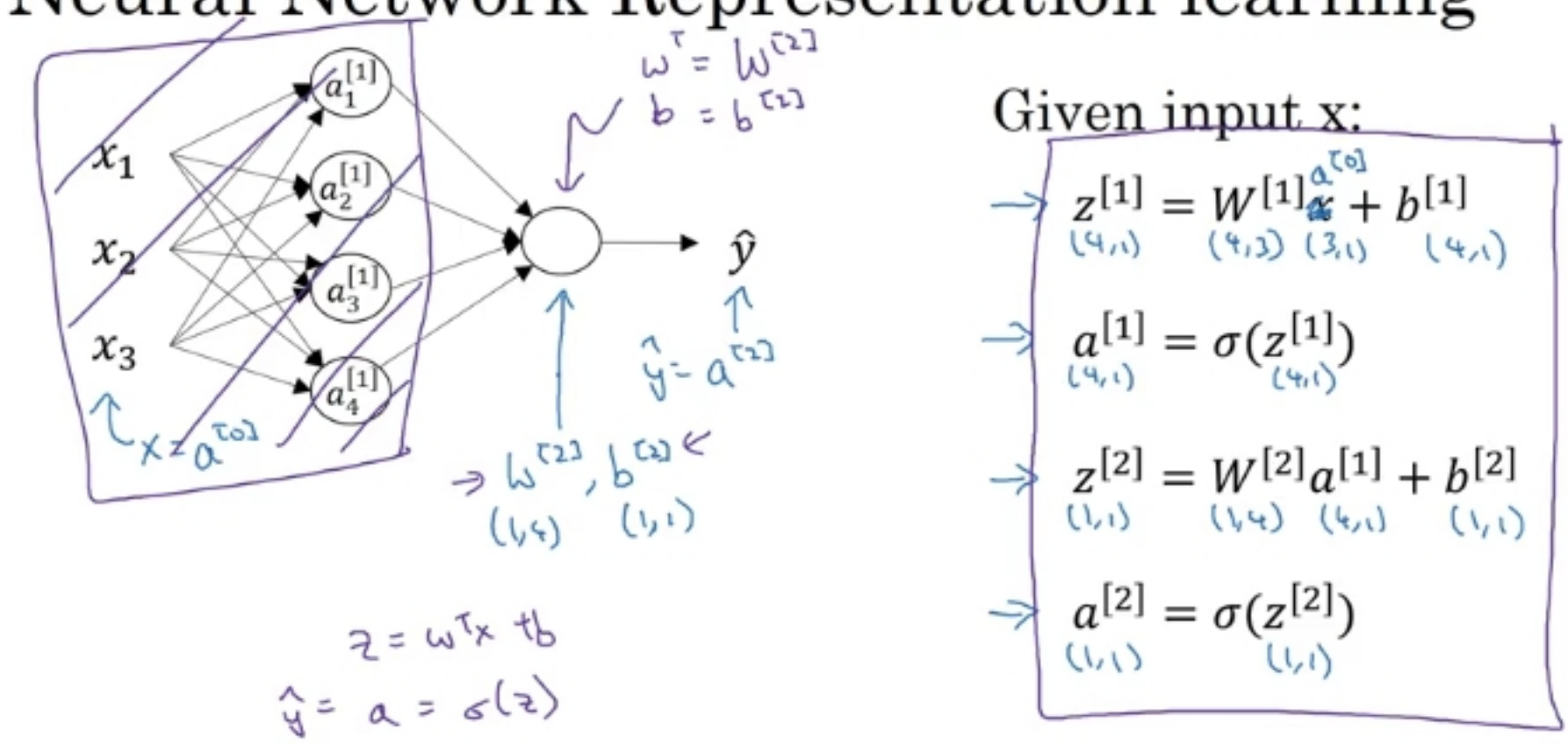
$W^{[1]}$ (4, 3)

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$b^{[1]}$ (4, 1)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Andrew Ng



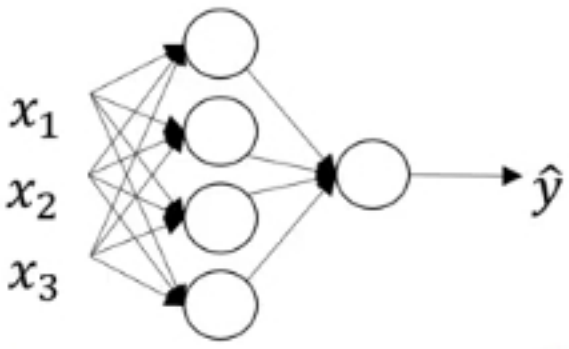
Andrew Ng

When you have a neural network with one hidden layer, what you need to implement, is to computer this output is just these four equations.

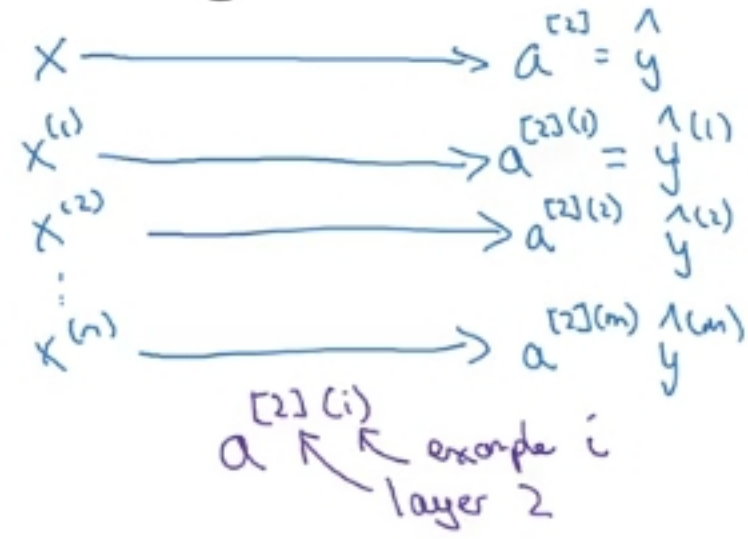
You can think of this as a vectorized implementation of computing the output of first these for logistic regression units in the hidden layer, that's what this does, and then this logistic regression in the output layer which is what this does.

I hope this description made sense, but the takeaway is to compute the output of this neural network, all you need is those four lines of code. So now, you've seen how given a single input feature, vector a, you can with

Vectorizing across multiple examples



$$
\begin{cases}
z^{[1]} = W^{[1]}x + b^{[1]} \\
a^{[1]} = \sigma(z^{[1]}) \\
z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} = \sigma(z^{[2]})
\end{cases}
$$

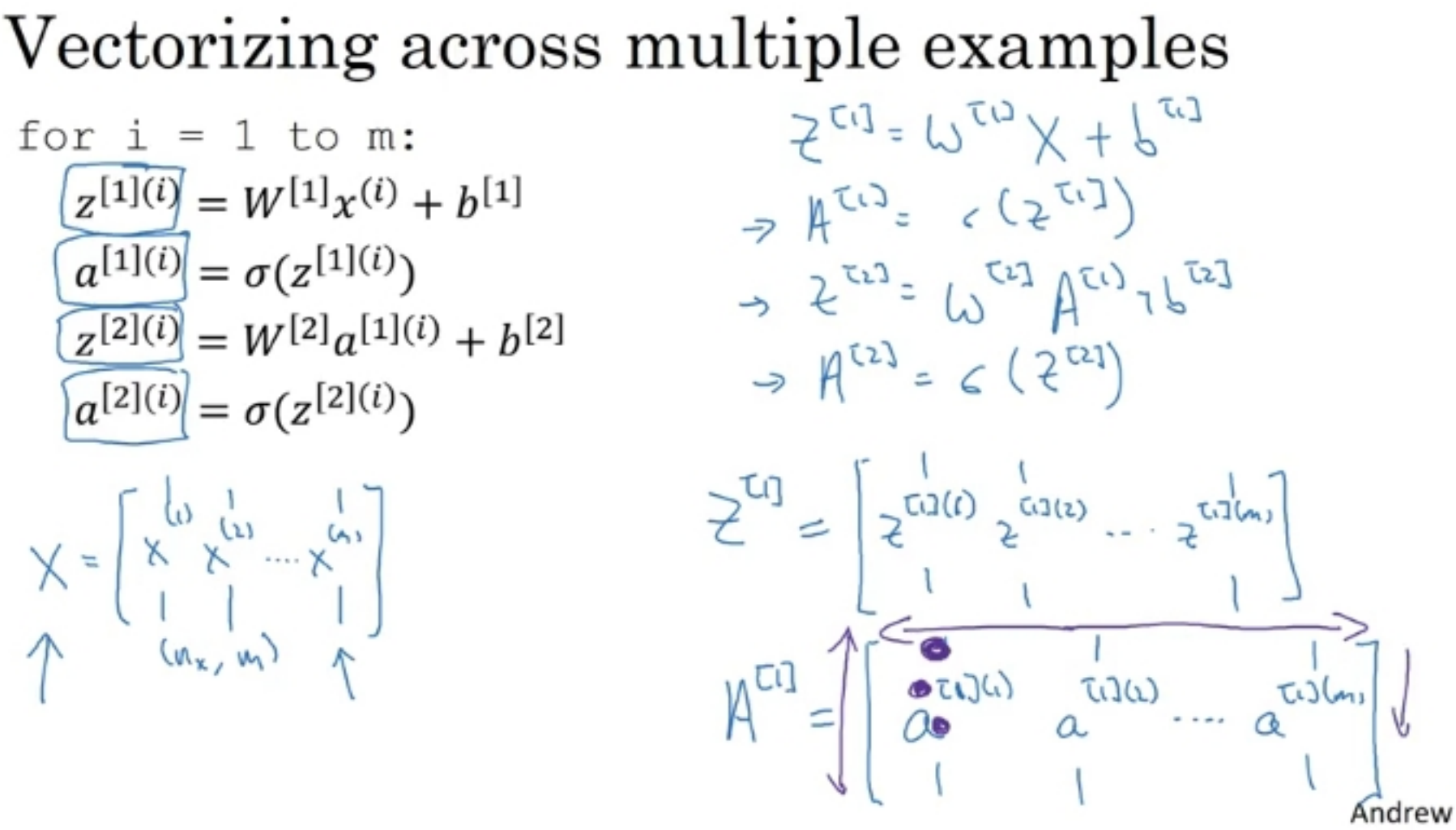


Andrew Ng

The round bracket i refers to training example i, and the square bracket 2 refers to layer 2, okay. So that's how the square bracket and the round bracket indices work.

And so to suggest that if you have an unvectorized implementation and want to compute the predictions of all your training examples, you need to do for i = 1 to m. Then basically implement these four equations, right?

You need to make a z[1](i) = W(1) x(i) + b[1], a[1](i) = sigma of z[1](1). z[2](i) = w[2]a[1](i) + b[2] andZ2i equals w2a1i plus b2 and a[2](i) = sigma point of z[2](i).



example, when you sweep from left to right you're scanning through the training cells. And vertically this vertical index corresponds to different nodes in the neural network.

So for example, this node, this value at the top most, top left most corner of the mean corresponds to the activation of the first heading unit on the first training example.

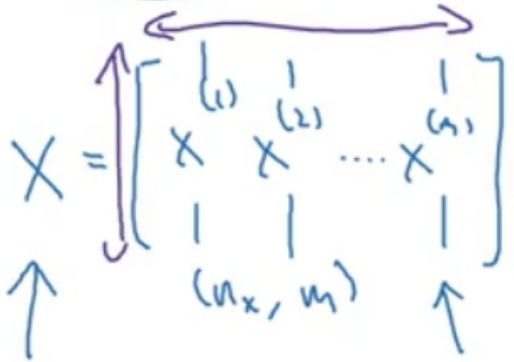
One value down corresponds to the activation in the second hidden unit on the first training example, then the third heading unit on the first training sample and so on. So as you scan down this is your indexing to the hidden units number.

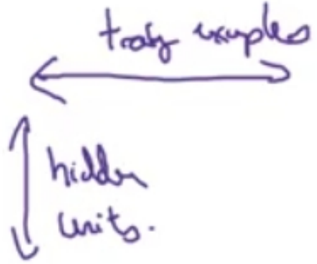
Vectorizing across multiple examples

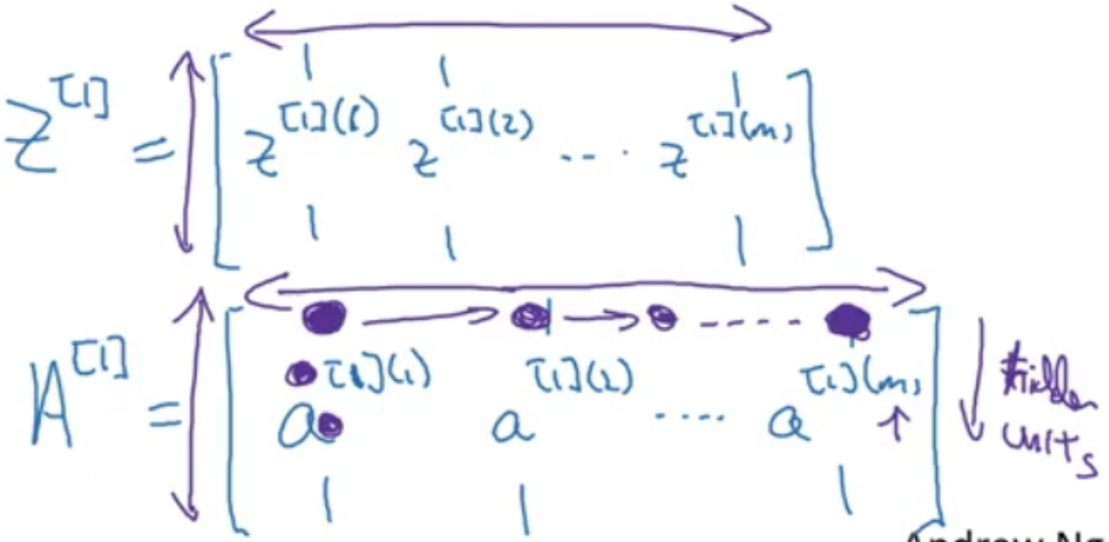
```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

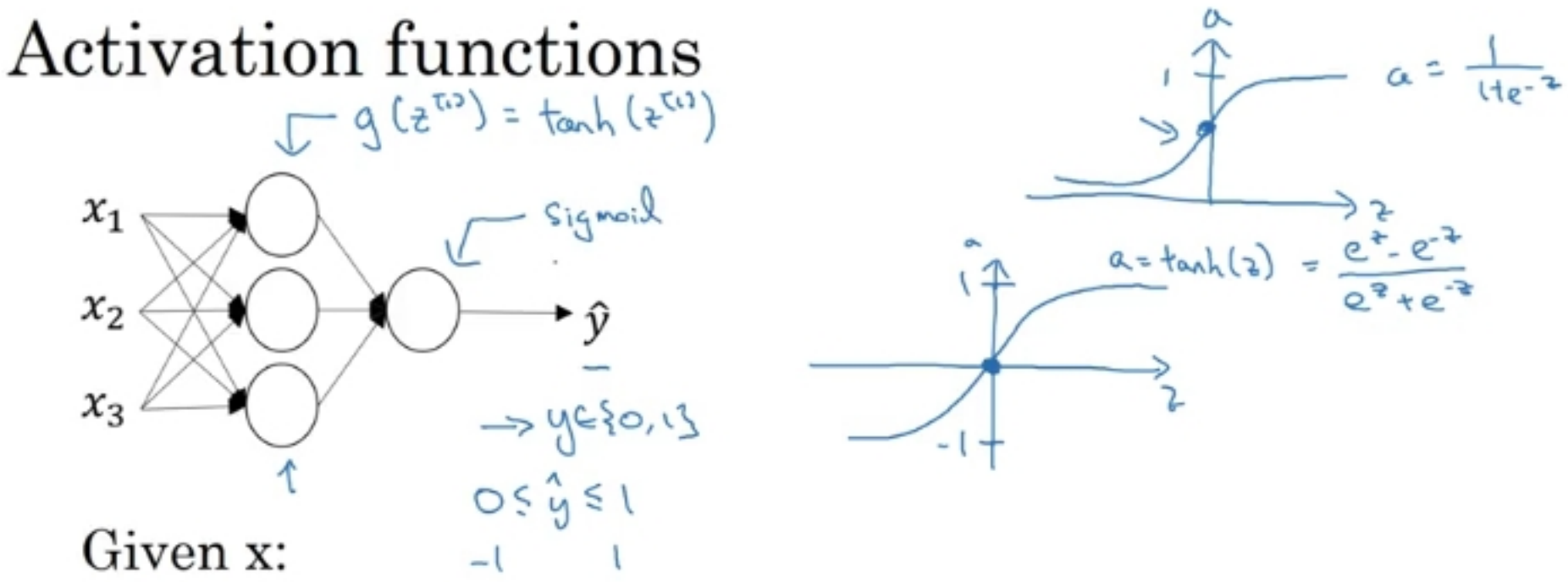
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$







Andrew Ng



Andrew Ng

So the one exception where I would use the sigmoid activation function is when you are using binary classification, in which case you might use the sigmoid activation function for the output layer. So g of z 2 here is equal to sigma of z 2.

And so what you see in this example is where you might have a tanh activation function for the hidden layer, and sigmoid for the output layer. So deactivation functions can be different for different layers.

And sometimes to note that activation functions are different for different layers, we might use these square