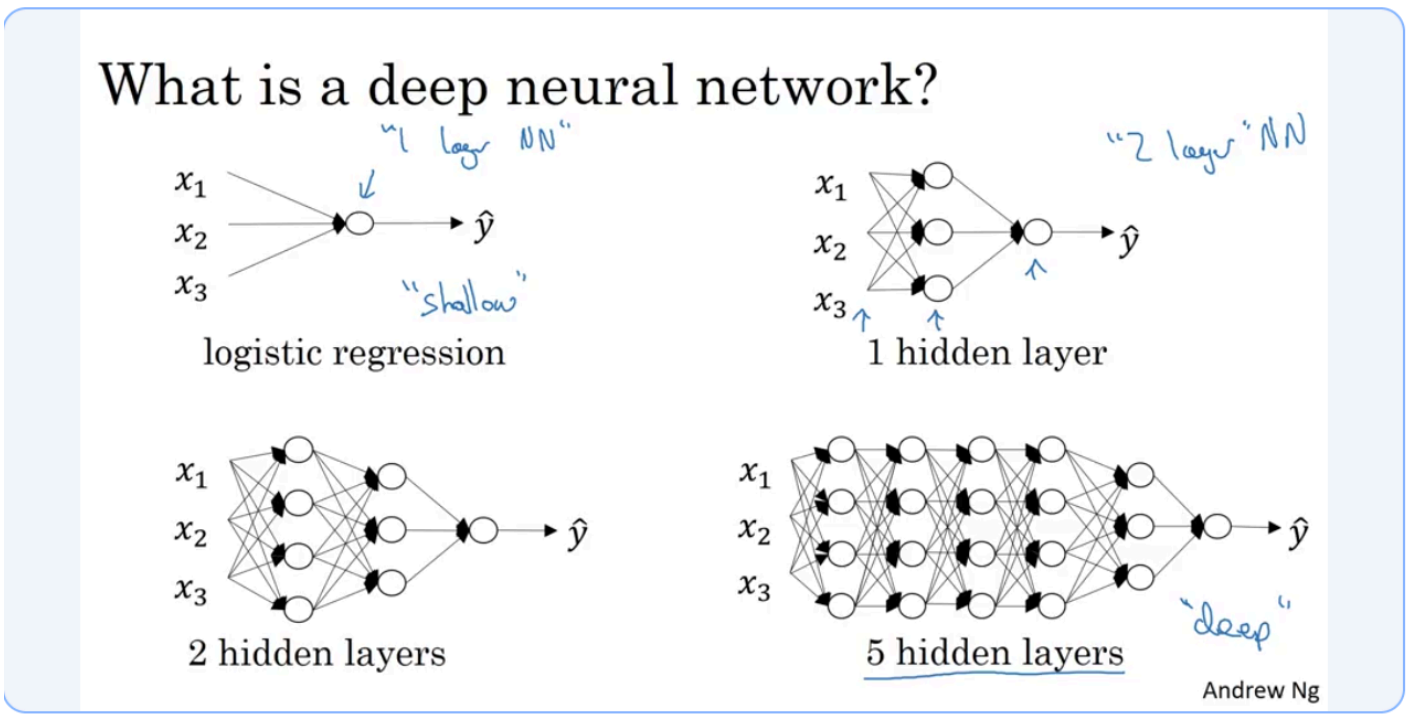
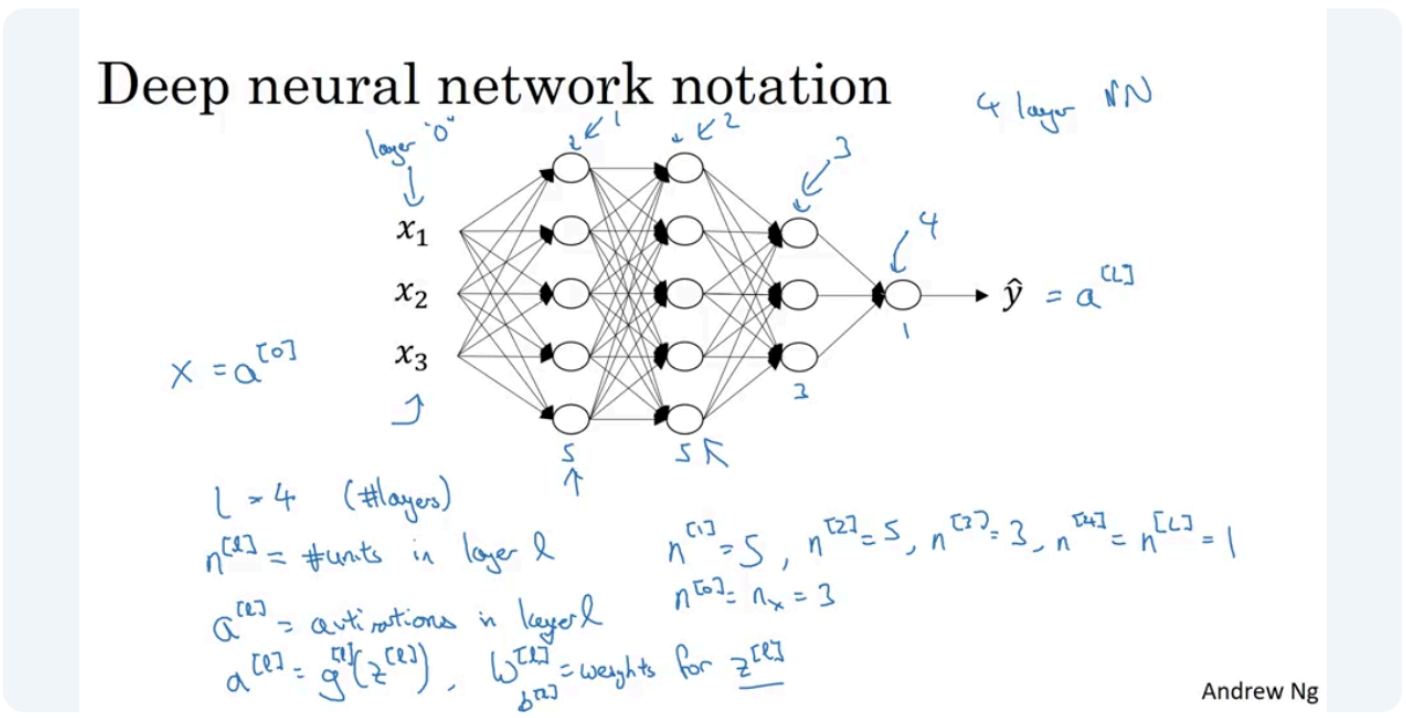
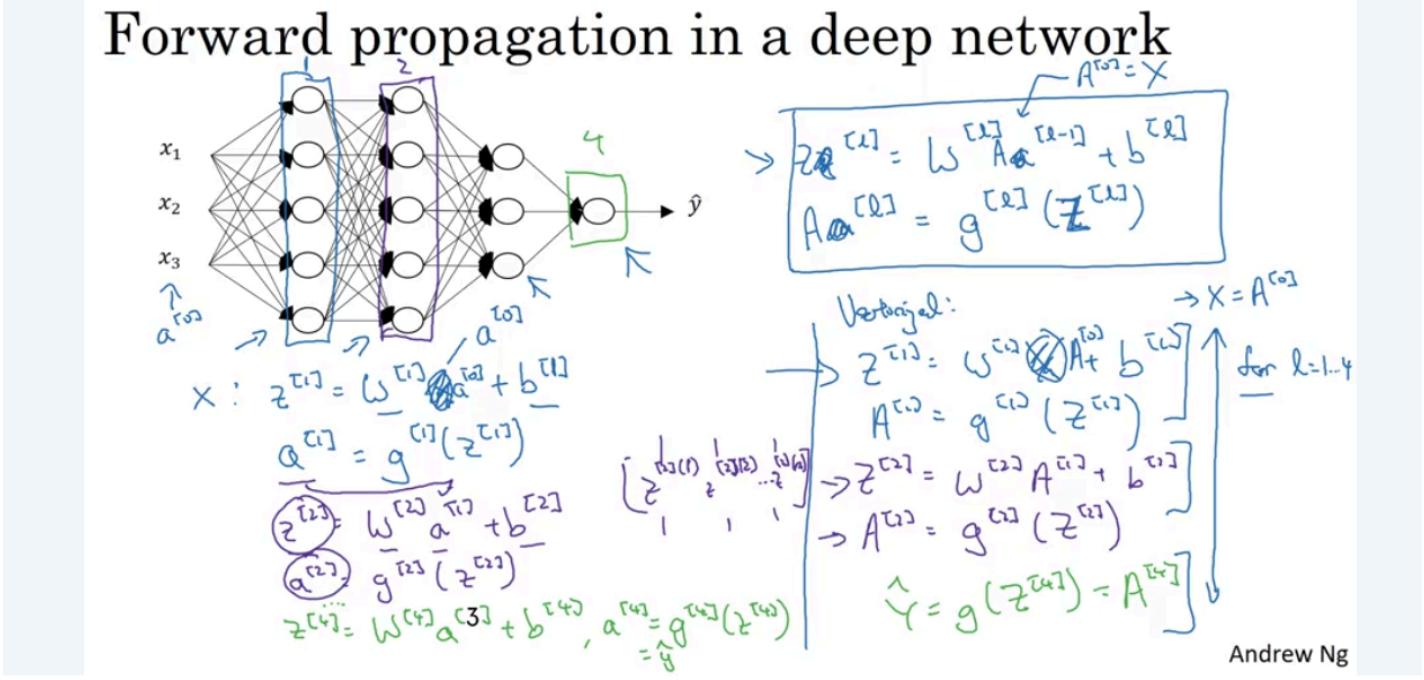
Deep L-Layer Neural Network



Notation:



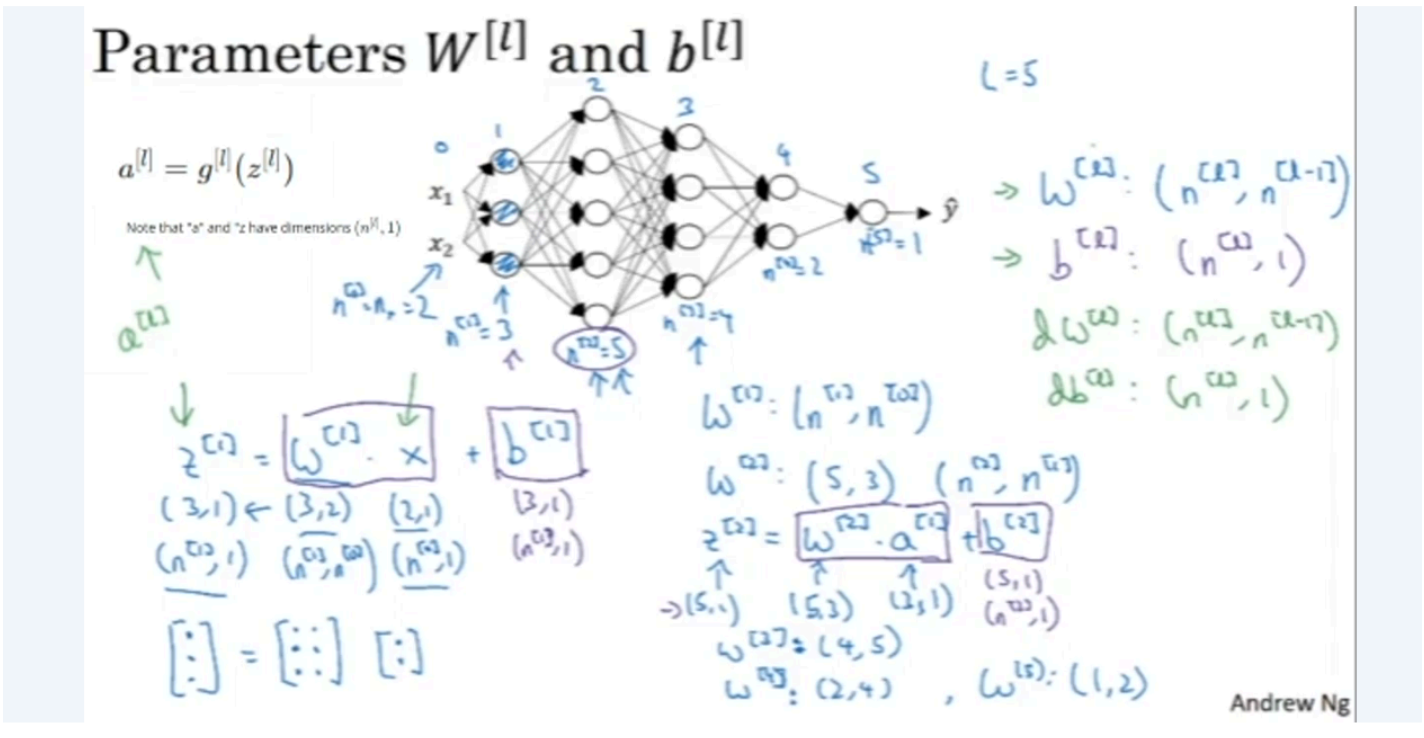
Forward Propagation in a Deep Network



loop through each layer

compute output of each layer → the output of layer L - 1 is the input of layer L

Getting your Matrix Dimensions Right



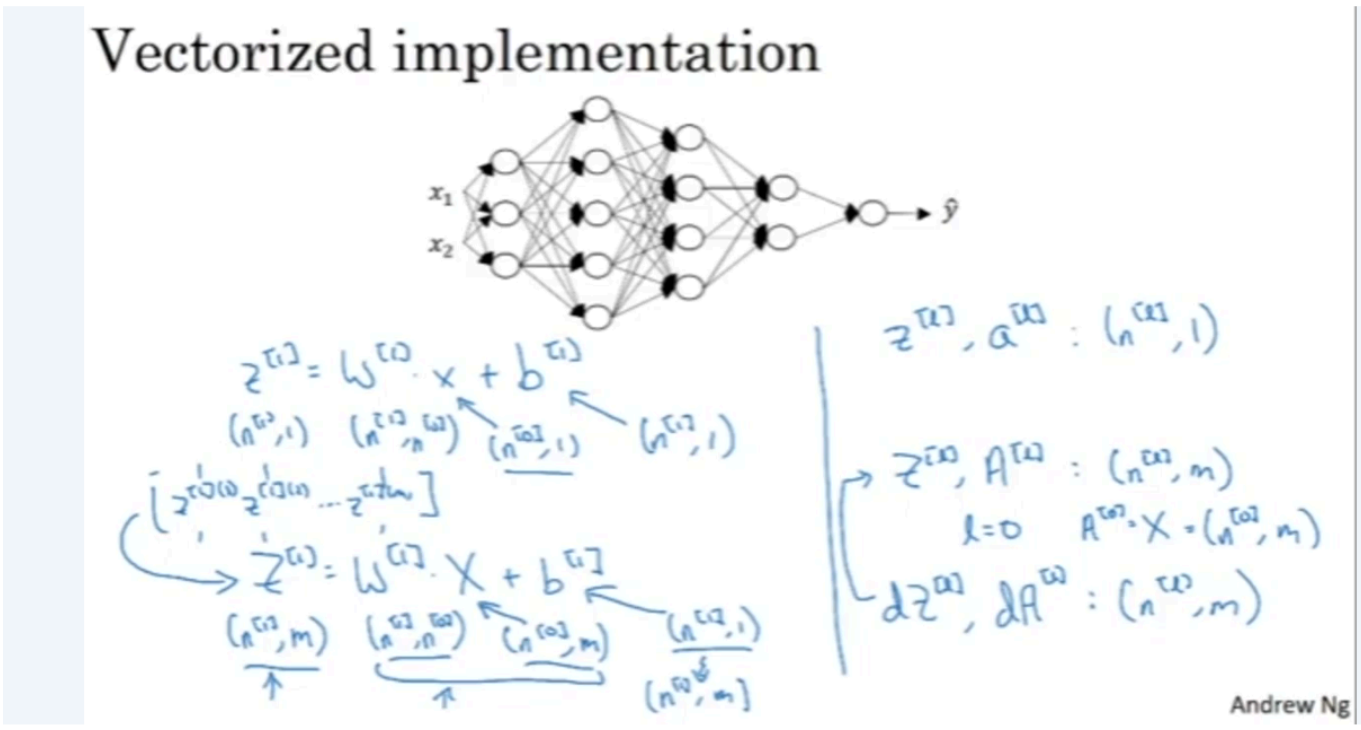
summary:

size of matrix in layer: l

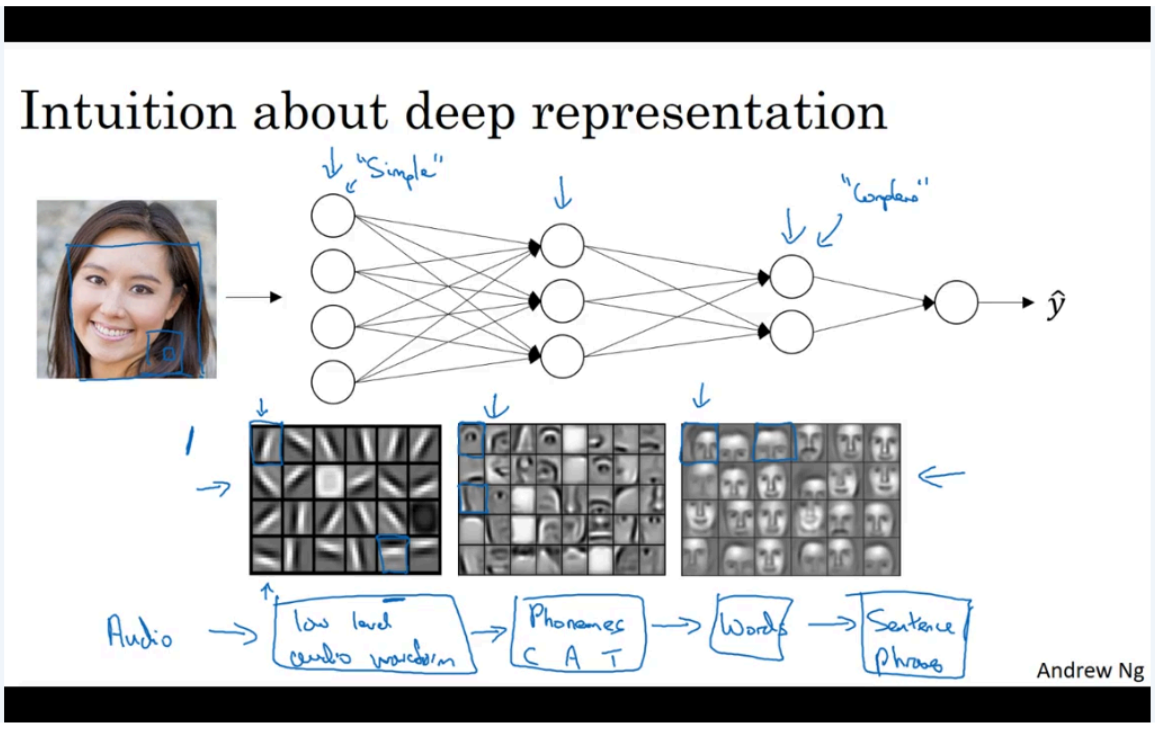
$$w^{[l]} : (n^{[l]}, n^{[l-1]})$$

$$b^{[l]} : (n^{[l]}, 1)$$

if you're implementing back-propagation, then the dimensions of dw should be the same as dimension of w. So dw should be the same dimension as w, and db should be the same dimension as b.



Why Deep Representations?



We've all been hearing that deep neural networks work really well for a lot of problems, and it's not just that they need to be big neural networks, is that specifically, they need to be deep or to have a lot of hidden layers. So why is that? Let's go through a couple examples and try to gain some intuition for why deep networks might work well. So first, what is a deep network computing? If you're building a system for face recognition or face detection, here's what a deep neural network could be doing. Perhaps you input a picture of a face then the first layer of the neural network you can think of as maybe being a feature detector or an edge detector. In this example, I'm plotting what a neural network with maybe 20 hidden units, might be trying to compute on this image.

So the 20 hidden units visualized by these little square boxes (the first image of 3 images below). So for example, this little visualization represents a hidden unit that's trying to figure out where the edges of that orientation are in the image. And maybe this hidden unit might be trying to figure out where are the horizontal edges in this image. And when we talk about convolutional networks in a later course, this particular visualization will make a bit more sense. But the form, you can think of the first layer of the neural network as looking at the picture and trying to figure out where are the edges in this picture. Now, let's think about where the edges in this picture by grouping together pixels to form edges. It can then detect the edges and group edges together to form parts of faces.

So for example, you might have a low neuron trying to see if it's finding an eye, or a different neuron trying to find that part of the nose. And so by putting together lots of edges, it can start to detect different parts of faces. And then, finally, by putting together different parts of faces, like an eye or a nose or an ear or a chin, it can then try to recognize or detect different types of faces. So intuitively, you can think of the earlier layers of the neural network as detecting simple functions, like edges. And then composing them together in the later layers of a neural network so that it can learn more and more complex functions. These visualizations will make more sense when we talk about convolutional nets. And one

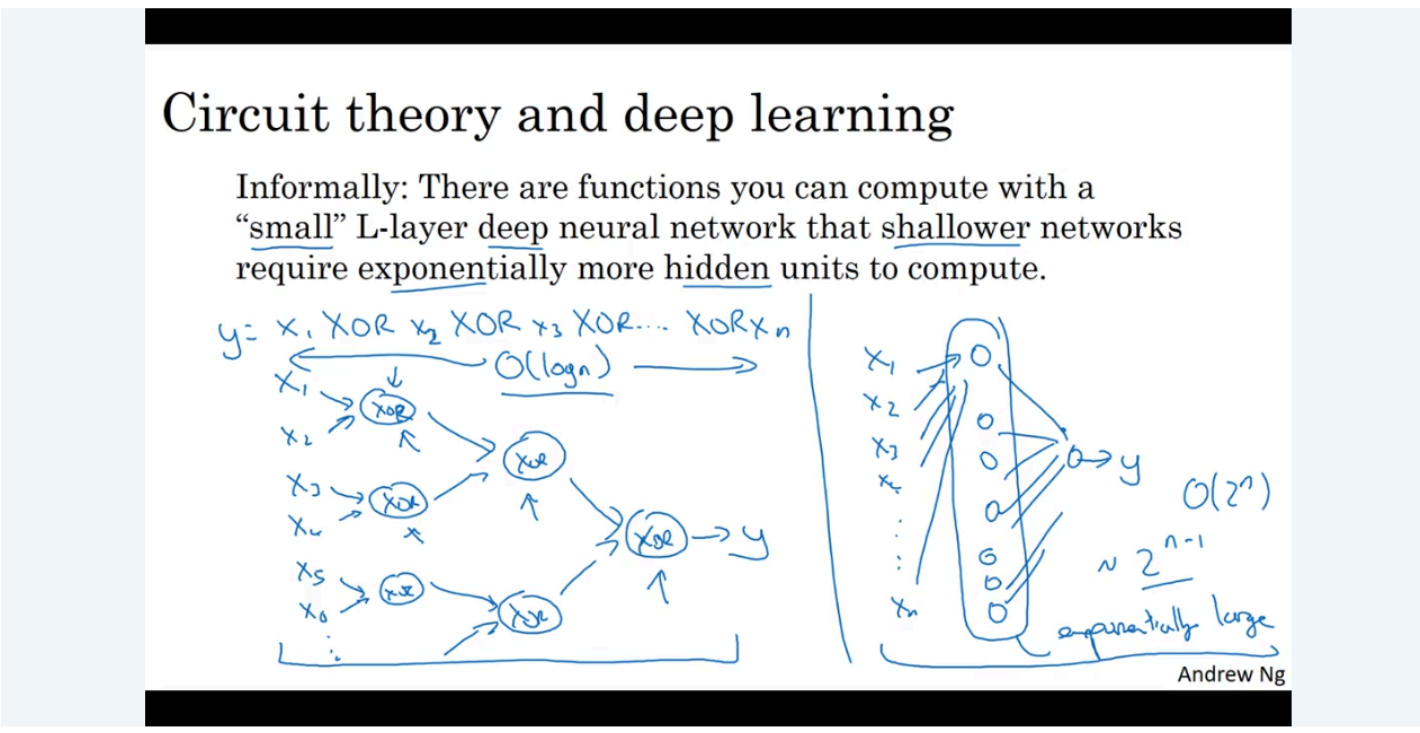
technical detail of this visualization, the edge detectors are looking in relatively small areas of an image, maybe very small regions like that.

And then the facial detectors you can look at maybe much larger areas of image. But the main intuition you take away from this is just finding simple things like edges and then building them up. Composing them together to detect more complex things like an eye or a nose then composing those together to find even more complex things. And this type of simple to complex hierarchical representation, or compositional representation, applies in other types of data than images and face recognition as well. For example, if you're trying to build a speech recognition system, it's hard to revisualize speech but if you input an audio clip then maybe the first level of a neural network might learn to detect low level audio wave form features, such as is this tone going up? Is it going down? Is it white noise or sniffling sound like

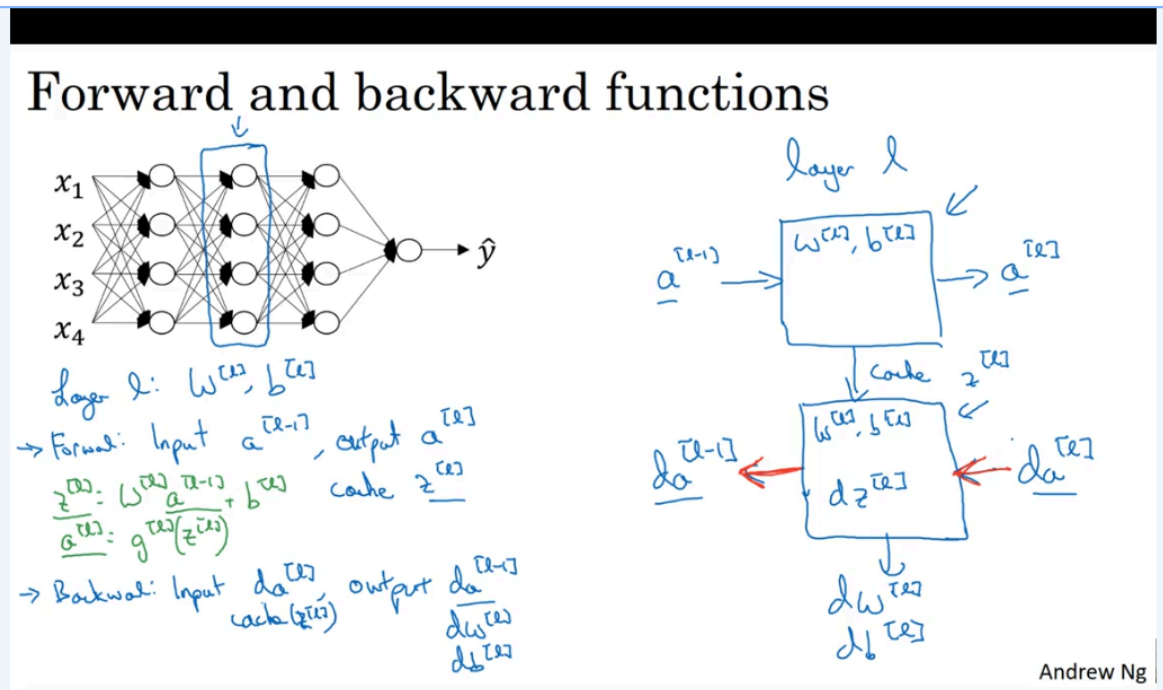
And what is the pitch? When it comes to that, detect low level wave form features like that. And then by composing low level wave forms, maybe you'll learn to detect basic units of sound. In linguistics they call phonemes. But, for example, in the word cat, the C is a phoneme, the A is a phoneme, the T is another phoneme. But learns to find maybe the basic units of sound and then composing that together maybe learn to recognize words in the audio. And then maybe compose those together, in order to recognize entire phrases or sentences.

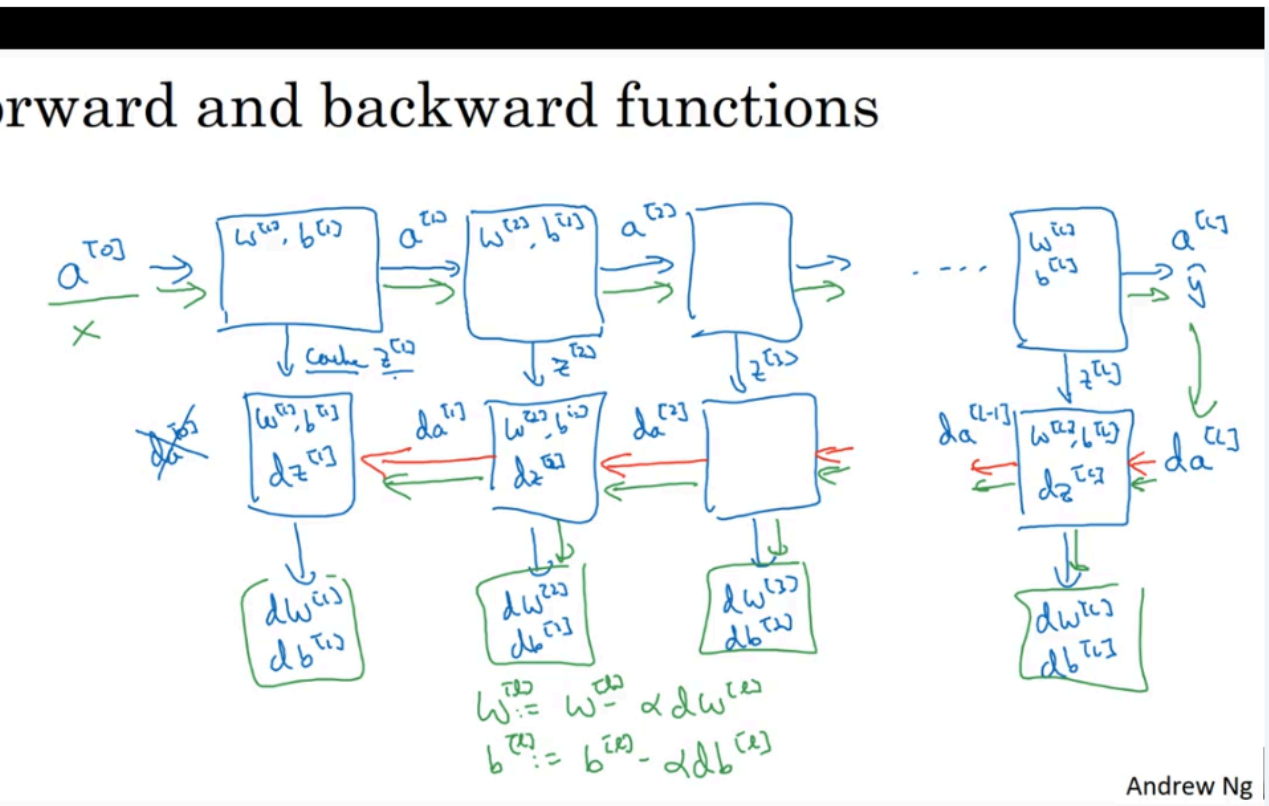
So deep neural network with multiple hidden layers might be able to have the earlier layers learn these lower level simple features and then have the later deeper layers then put together the simpler things it's detected in order to detect more complex things like recognize specific words or even phrases or sentences. The uttering in order to carry out speech recognition. And what we see is that whereas the other layers are computing, what seems like relatively simple functions of the input such as where the edge is, by the time you get deep in the network you can actually do surprisingly complex things. Such as detect faces or detect words or phrases or sentences. Some people like to make an analogy between deep neural networks and the human brain, where we believe, or neuroscientists believe, that the human brain also starts off detecting simple things like edges in what your eyes see then builds those up to detect more complex things like the faces that you see. I think analogies between deep learning and the human brain are sometimes a little bit dangerous. But there is a lot of truth to, this being how we think that human brain works and that the human brain probably detects simple things like edges first then put them together to from more and more complex objects and so that has served as a loose form of inspiration for some deep learning as well.

Circuit theory and deep learning



Building Blocks of Deep Neural Networks





Forward and Backward Propagation

Forward propagation for layer l

→ Input $a^{[l-1]}$

→ Output $a^{[l]}$, cache $(z^{[l]})$

$W^{[l]}, b^{[l]}$

$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

Vectorized:

$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$

$A^{[l]} = g^{[l]}(Z^{[l]})$

$a^{[0]}$

$A^{[0]}$

$X = A^{[0]}$ → [ ] → [ ] → [ ] →

Andrew

Backward propagation for layer l

→ Input $da^{[l]}$

→ Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$

$db^{[l]} = dz^{[l]}$

$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$

$dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$

$dz^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$

$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$

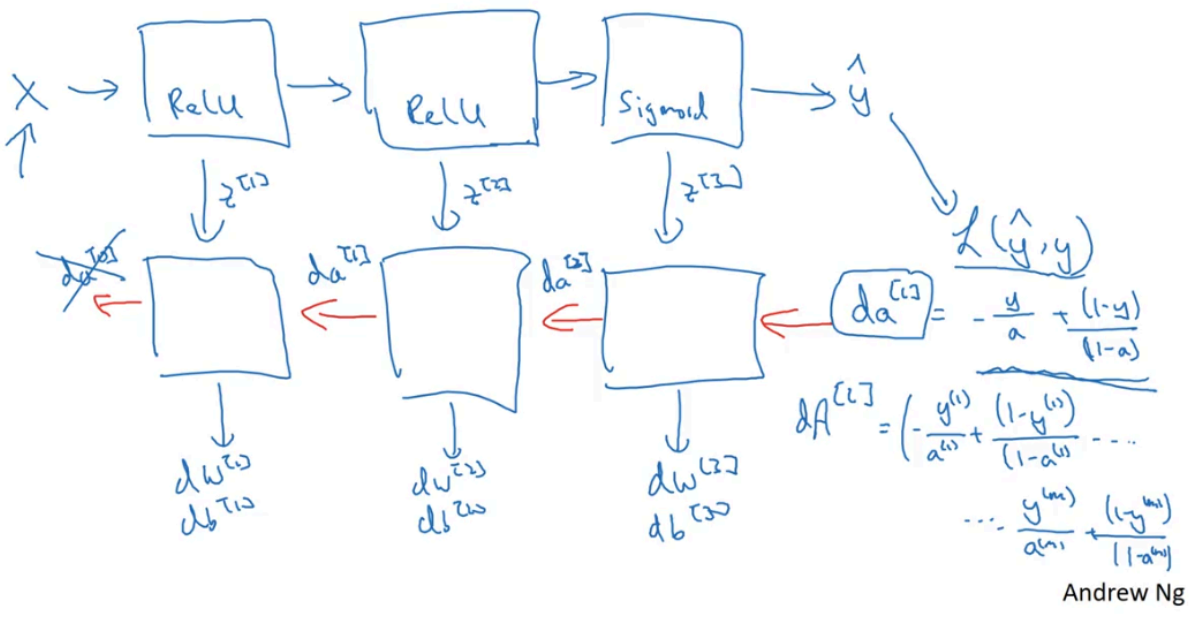
$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$

$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$

Andrew Ng

Summary:

Summary



Andrew Ng

During backpropagation, you pass backwards da^2 and da^1. Though this process could compute da^0, we don't need it, so you can discard it. This is how you implement forward and back propagation for a three-layer neural network.

There's one last important detail: for forward recursion, we initialize with the input data x. What about backward recursion? For logistic regression in binary classification, da[l] = -y/a + (1-y)/(1-a). This is the derivative of the loss function with respect to the output (y-hat).

If you know calculus, you can verify this formula by taking derivatives of the loss function l with respect to y-hat (or a). Use this formula for da in the final layer (capital L).

For a vectorized implementation, initialize backward recursion with da (capital A) for layer L, which applies the same formula across all training examples:
$[y_1/a_1 + (1-y_1)/(1-a_1), y_2/a_2 + (1-y_2)/(1-a_2), \ldots, y_m/a_m + (1-y_m)/(1-a_m)]$ according to the image

This is how you initialize the vectorized version of backpropagation.

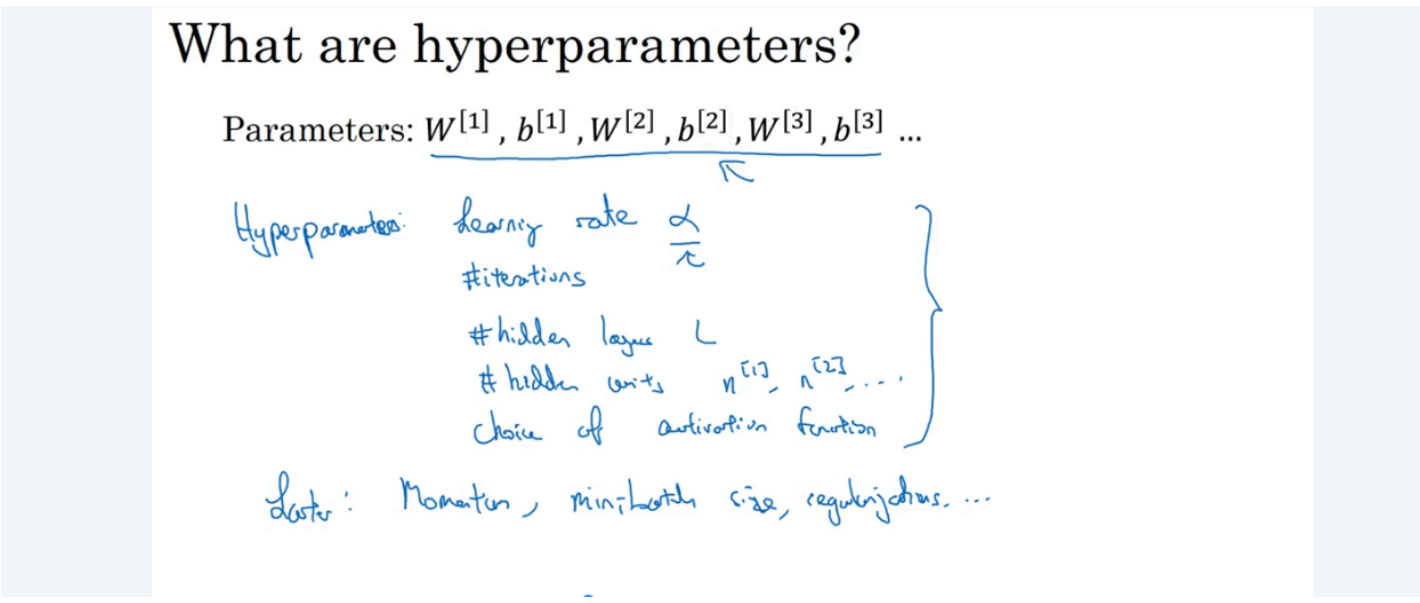
read more: https://community.deeplearning.ai/t/feedforward-neural-networks-in-depth/98811

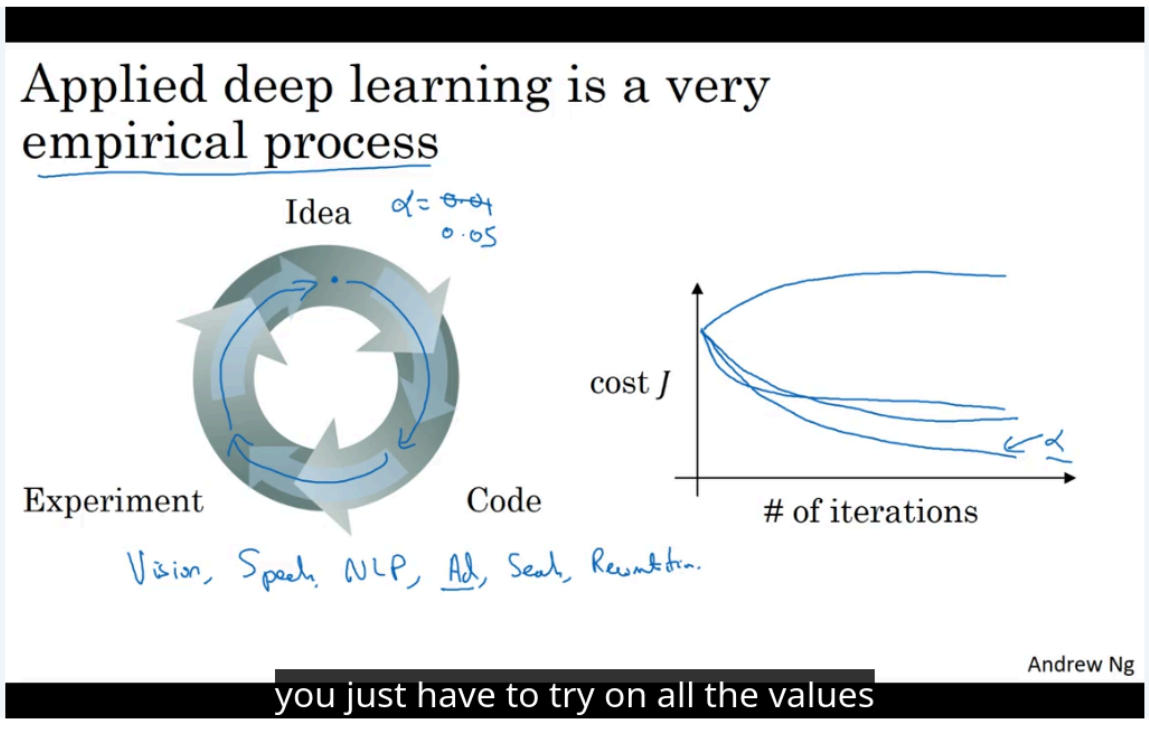
Parameters vs Hyperparameters

Being effective in developing your deep Neural Nets requires that you not only organize your parameters well but also your hyper parameters. So what are hyper parameters? let's take a look! So the parameters your model are W and B and there are other things you need to tell your learning algorithm, such as the learning rate alpha, because we need to set alpha and that in turn will determine how your parameters evolve or maybe the number of iterations of gradient descent you carry out. Your learning algorithm has oth numbers that you need to set such as the number of hidden layers, so we call that capital L, or the number of hidden units, such as 0 and 1 and 2 and so on. Then you also have the choice of activation function. do you want to use a RELU, or tangent or a sigmoid function especially in the hidden layers.

So all of these things are things that you need to tell your learning algorithm and so these are parameters that control the ultimate parameters W and B and so we call all of these things below hyperparameters.

Because these things like alpha, the learning rate, the number of iterations, number of hidden layers, and so on, these are all parameters that control W and B. So we call these things hyper parameters, because it is the hyper parameters that somehow determine the final value of the parameters W and B that you end up with. In fact, deep learning has a lot of different hyper parameters. In the later course, we'll see other hyper parameters as well such as the momentum term, the mini batch size, various forms of regularization parameters, and so on. If none of these terms at the bottom make sense yet, don't worry about it! We'll talk about them in the second course.





Clarification For: What does this have to do with the brain?

Clarification For: What does this have to do with the brain?

Note that the formulas shown in the next video have a few typos. Here is the correct set of formulas.

$$dZ^{[L]} = A^{[L]} - Y$$

$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L-1]^T}$$

$$db^{[L]} = \frac{1}{m} np.sum(dZ^{[L]}, axis = 1, keepdims = True)$$

$$dZ^{[L-1]} = W^{[L]^T} dZ^{[L]} * g'^{[L-1]}(Z^{[L-1]})$$

Note that * denotes element-wise multiplication)

$$\vdots$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g'^{[1]}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]^T}$$

Note that $A^{[0]^T}$ is another way to denote the input features, which is also written as X^T

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Go to next item ✓ Completed

What does this have to do with the brain?