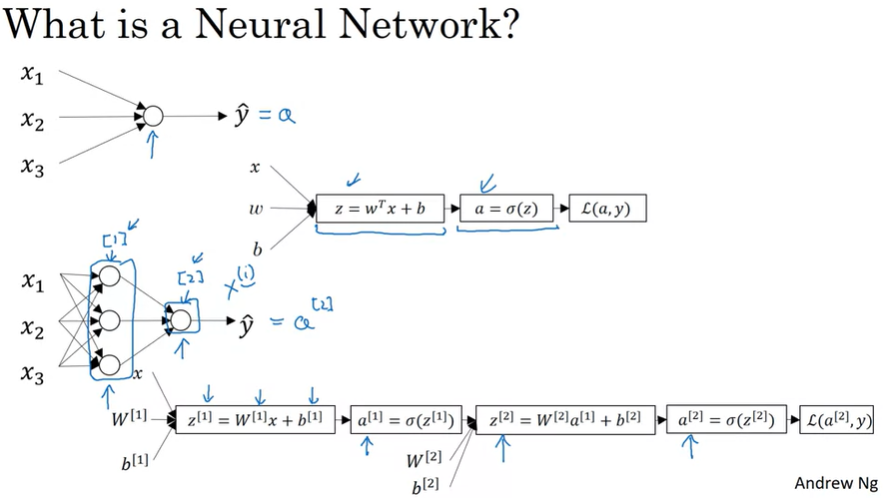
Author: Anish Mahapatra  
Shallow Neural Networks

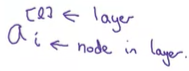
Superscripts will be used to denote the later. [1] would indicate layer 1, [2] would indicate layer 2 and so on.  
Superscripts with () brackets are used to indicate training examples.



For logistic regression, we had the z followed by a calculation. In this neural network, we jut do it multiple times, as a z followed by a calculation, and a z followed by a calculation and then we finally compute the loss in the end.

**Neural Network Representation**

Input Layer -> [Hidden layer] -> Output Layer  
a -> activations (values that different layers are passing on to the neural network.

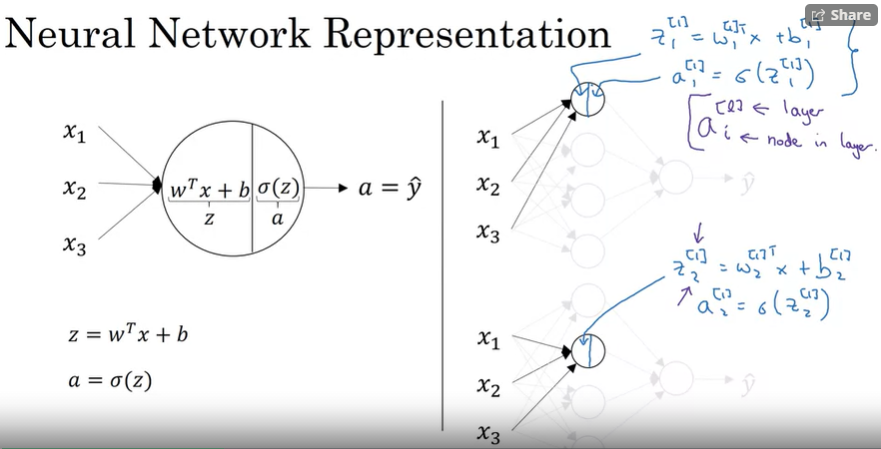


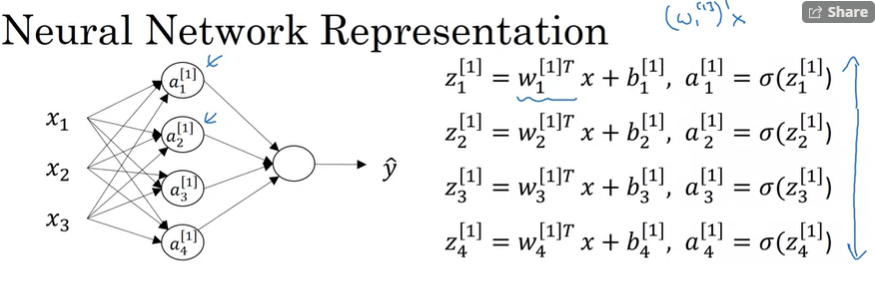


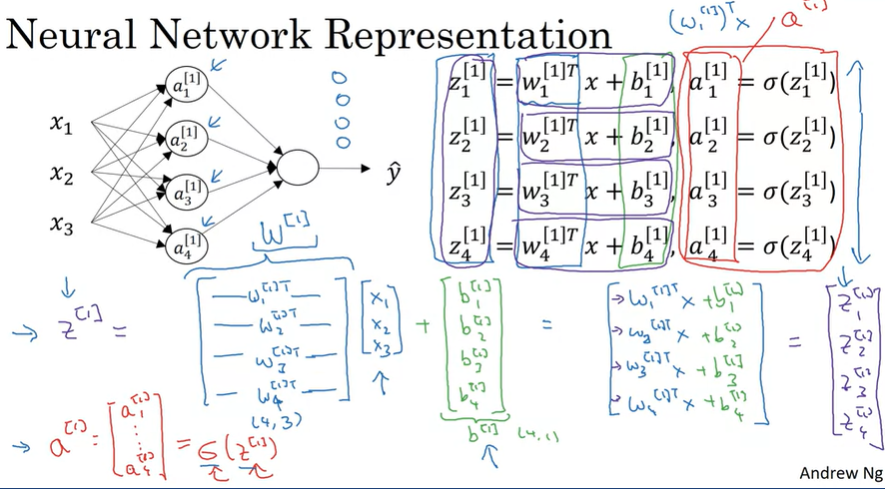
Number of layers in a neural network: INPUT Layer is not counted. So the above would be a 2 – Layer – Neural network.  
Layers 0, 1, 2 -> 2 Layer NN

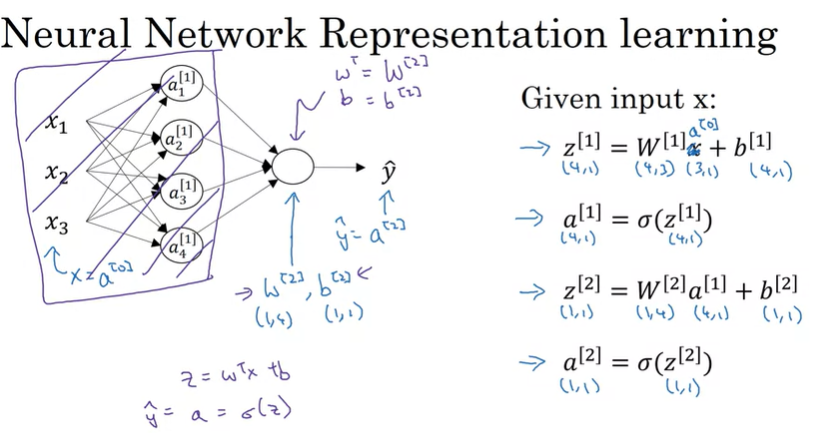
w[2] (input), b[2] (output)

w[2] dimension is (1,4) because the hidden layer has 4 hidden units and the output layer has just one hidden unit

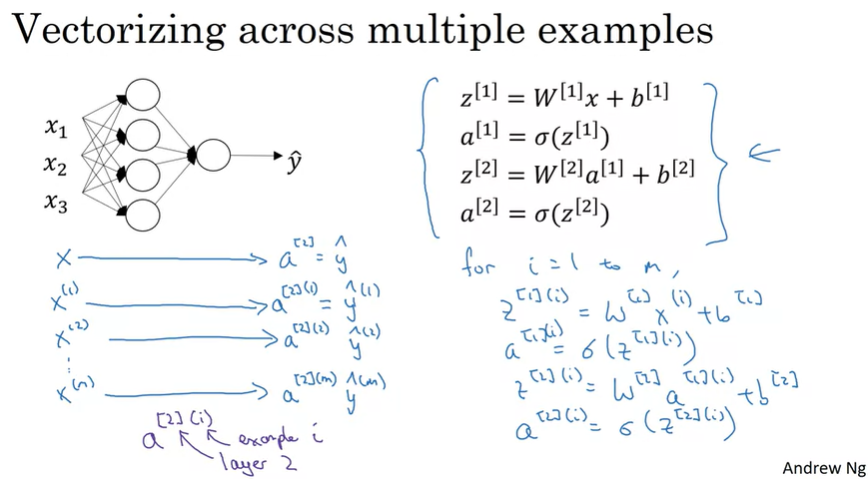


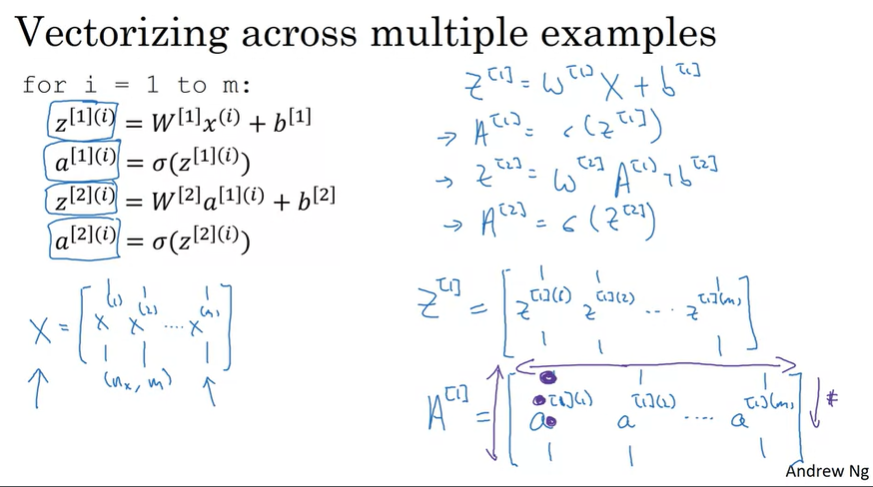


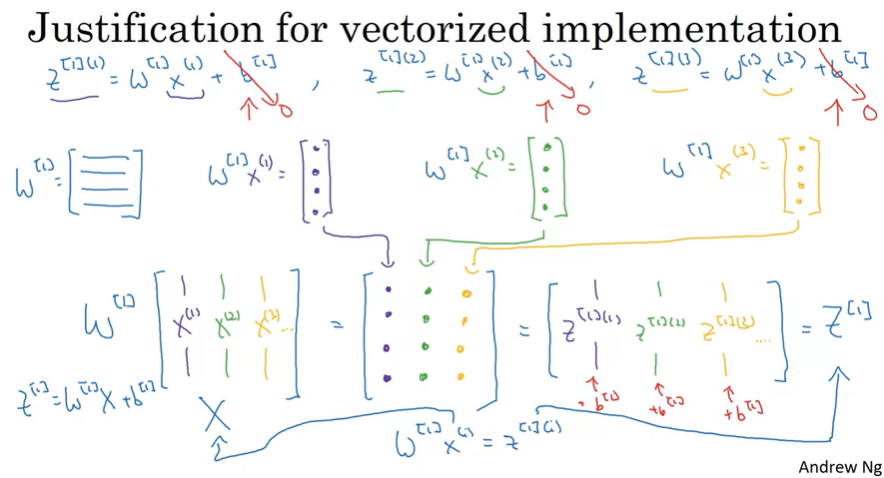


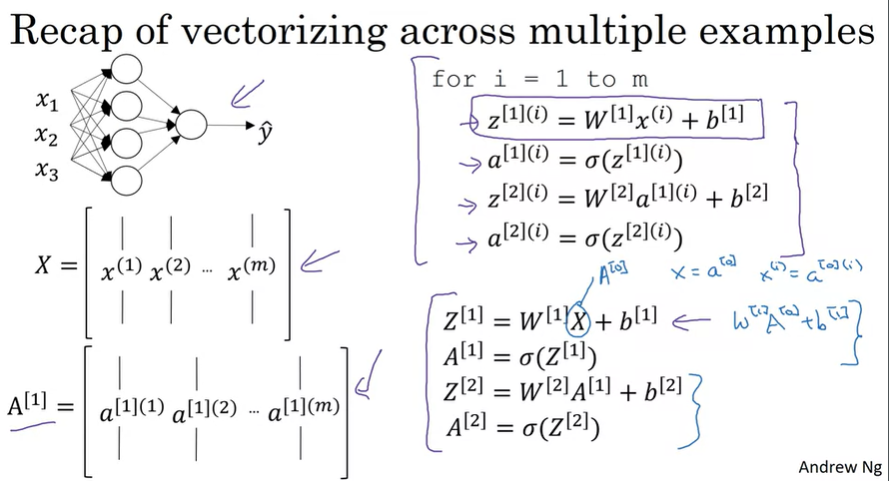




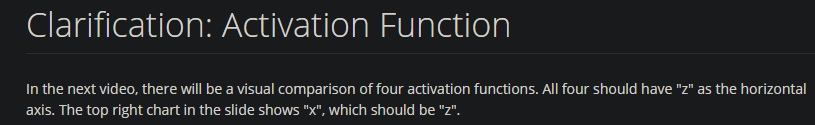








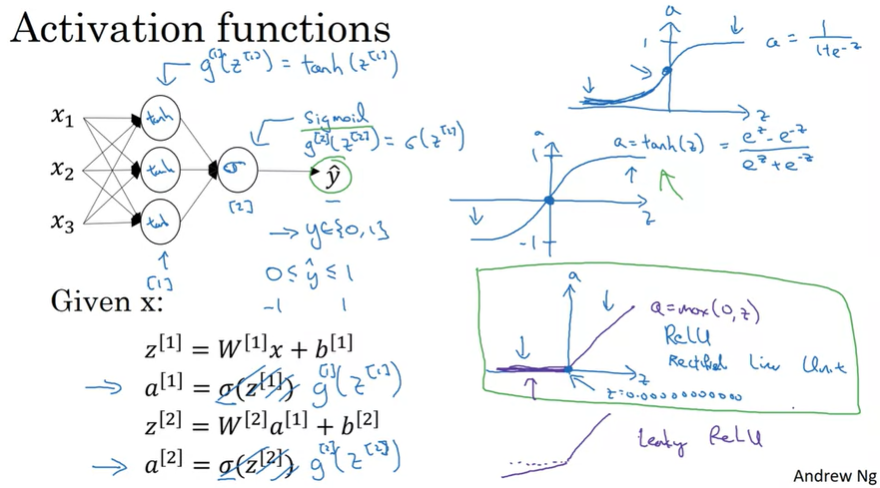
Activation Functions:

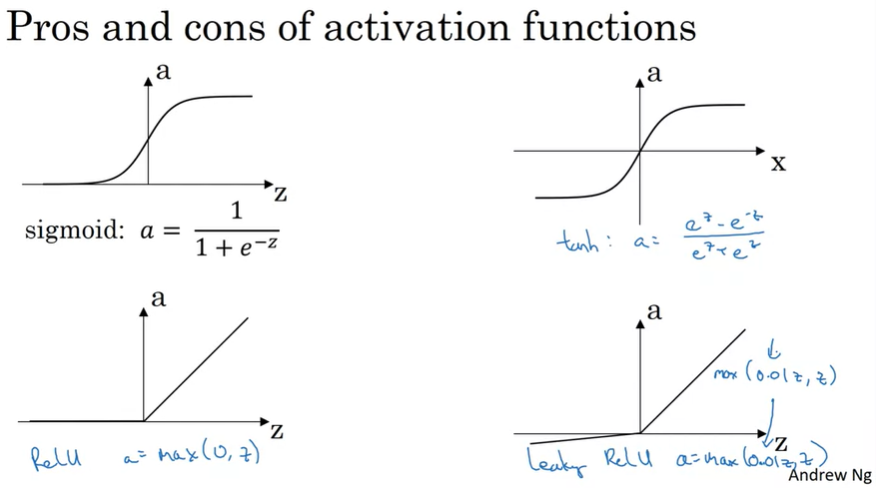


The sigmoid activation function can be used for binary classification (almost never use this)

The ReLU (Rectified Linear Unit) is the most popular activation function

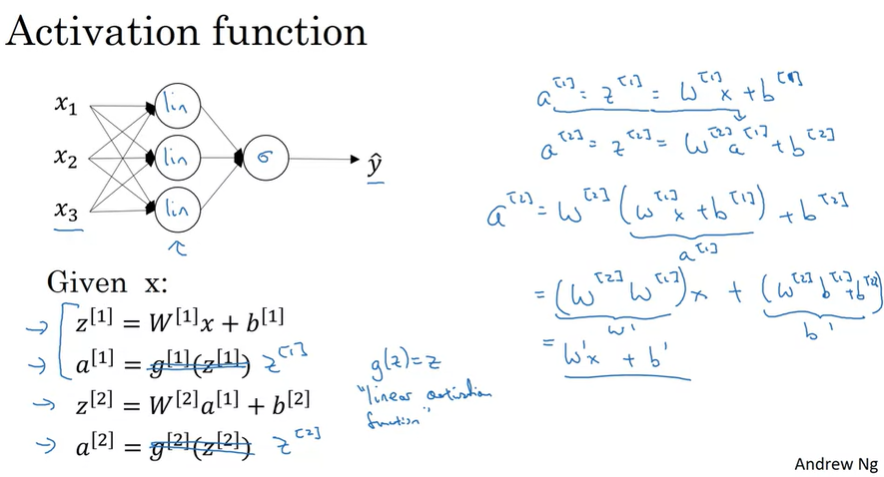
Tan h can also be used and has a higher preference than the sigmoid function as it centers itself at 0

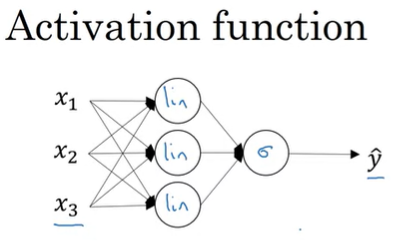




**Why do we need to use a** **non-linear activation function?**

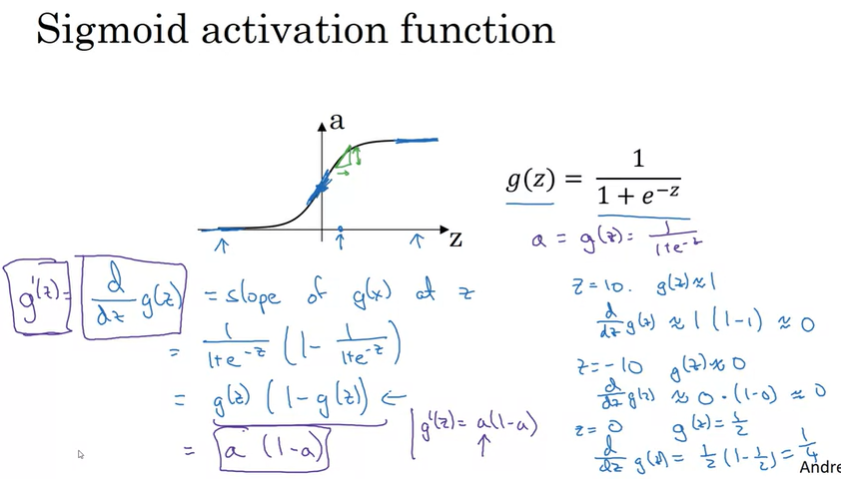
The composition of two or more linear functions is itself a linear function, which is not as useful for deep learning.

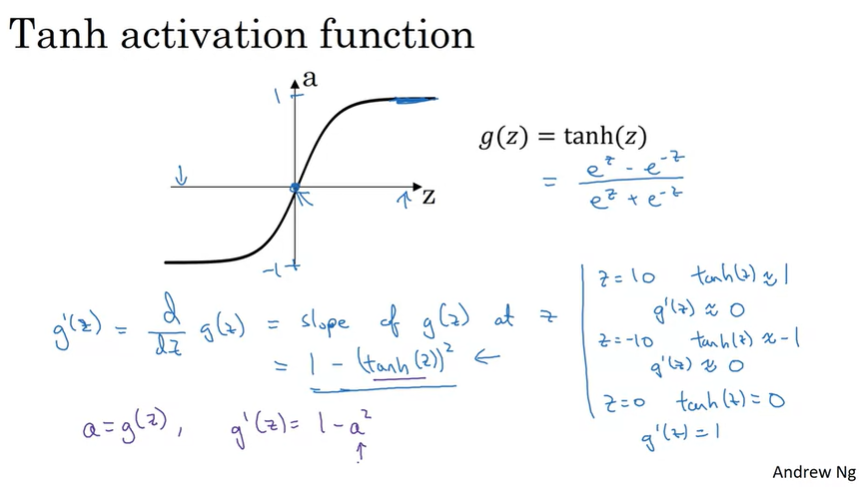


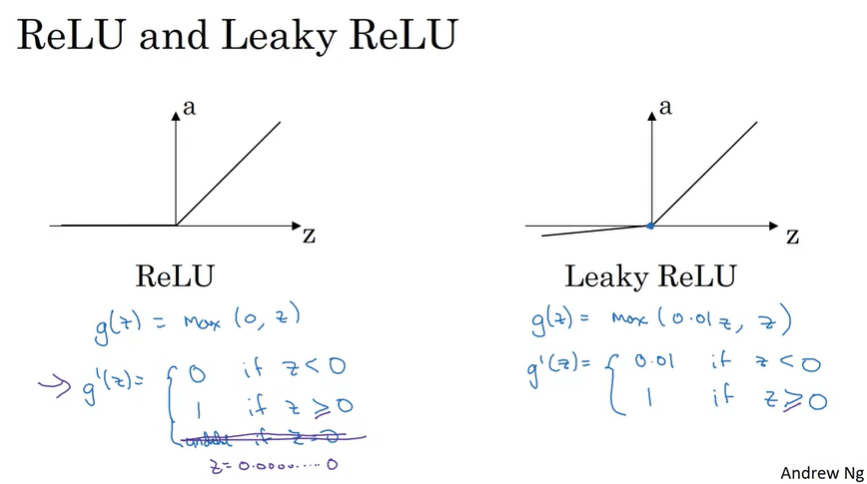


The one place where you might want to use a linear function: if you are doing machine learning on a regression problem  
(usually in the input layer)

Derivatives of activation function

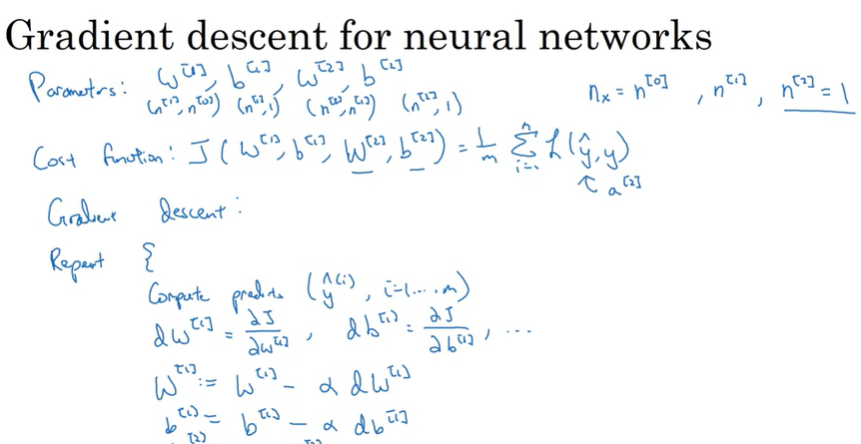




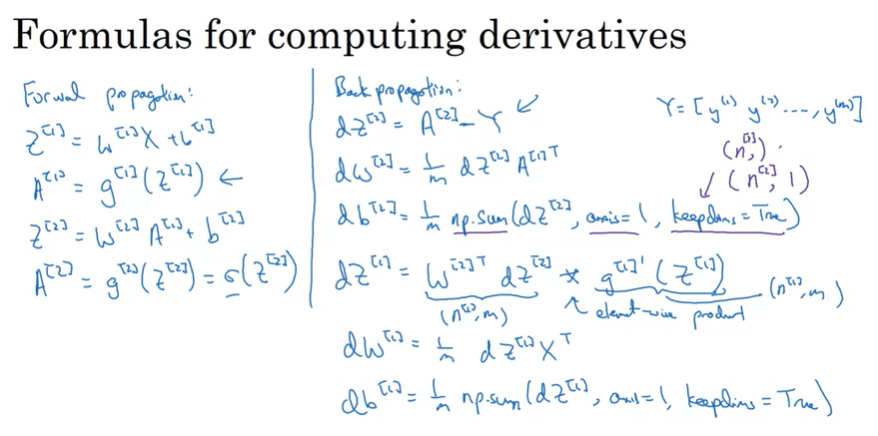


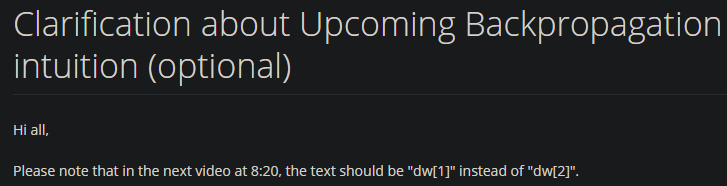
**Gradient Descent for Neural Networks**

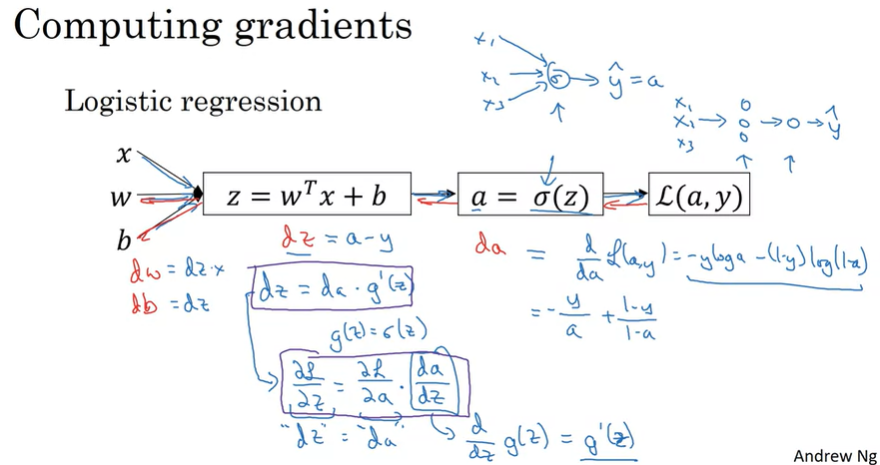
Equations to get back propagation working!

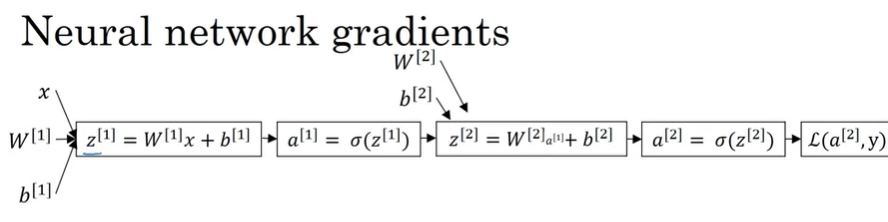


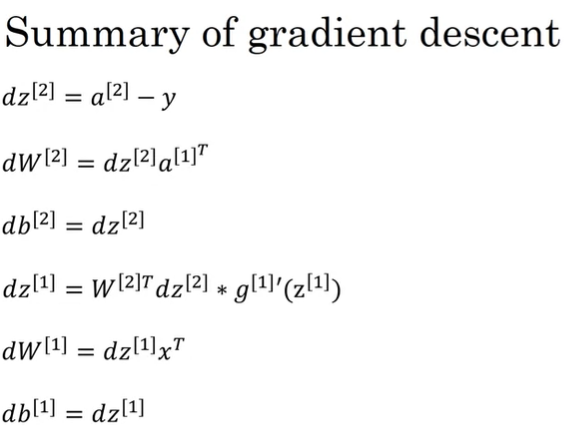
Formulas for forward propagation

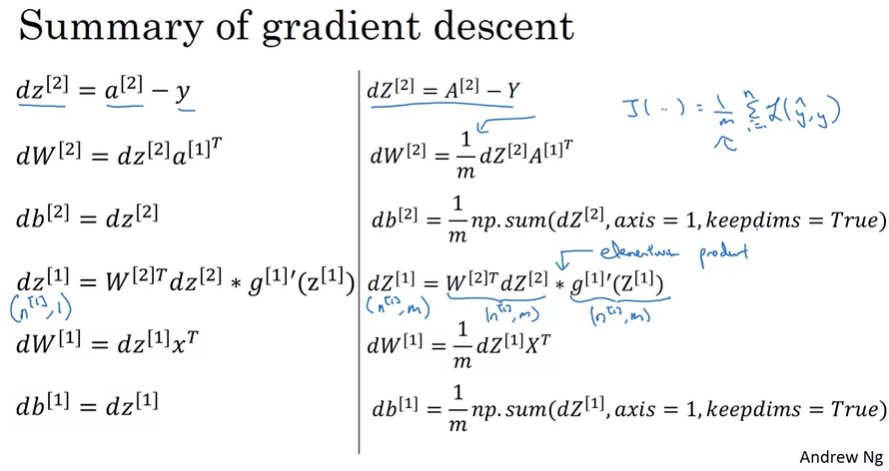












How to initialize the weights of the neural network while training it?

It turns out initializing weights randomly is much better than initializing it with zeros. If the hidden layers have the same value, i.e., 0, then when back propagation happens, each node is still computing the exact same value and they have equal effect on the output.

So, no matter how long you run gradient descent, they all continue to compute the exact same function.

W can be initialized to small random values &  
b can be initialized to zeros

If w is very large, learning will take longer

