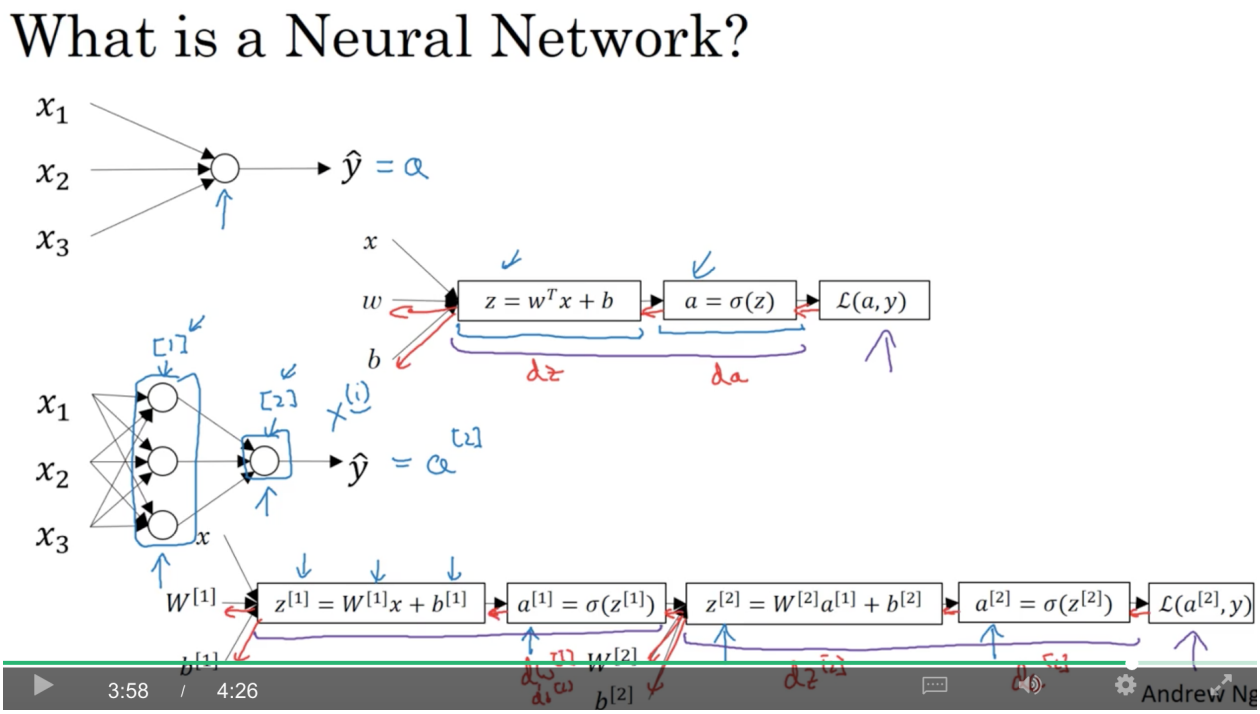
contents

neural networks overview

其中，每个神经元完成了$z = w^T x + b$以及$a = \sigma(z)$两个操作(a表示activation)，每一层的数据用上标[i]表示。



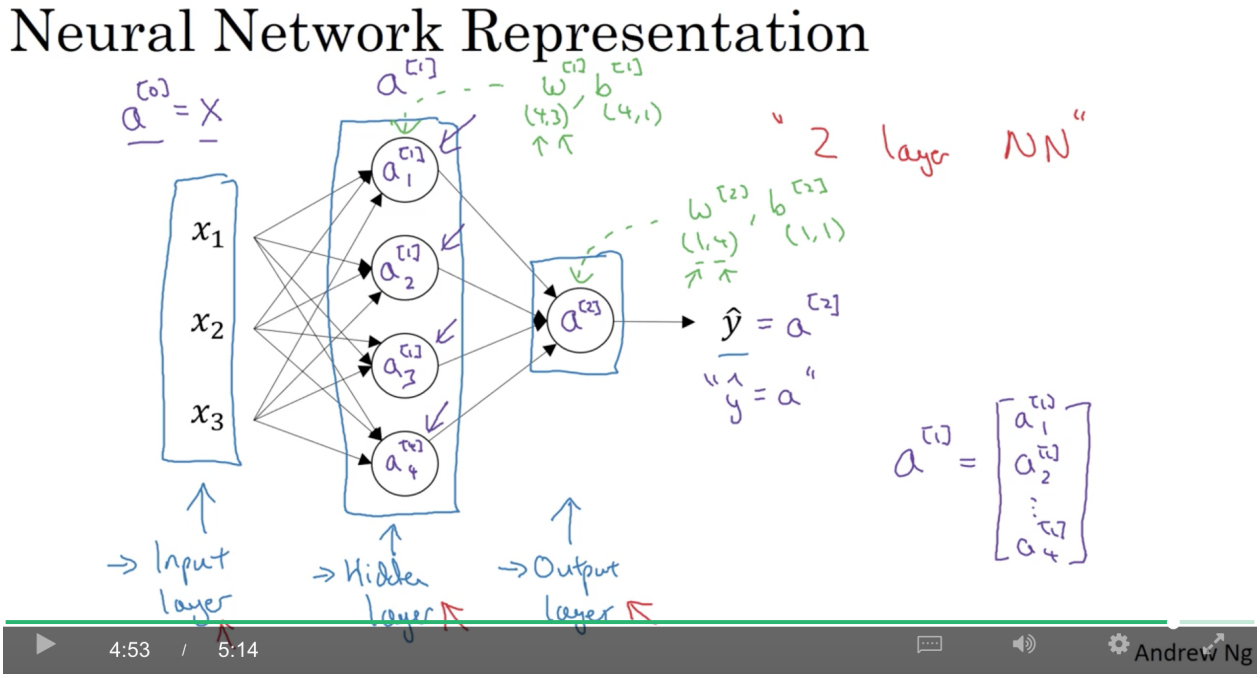
neural network representation

图示是一个2层nn（inputlayer不算在内，有hidden和output两层）。

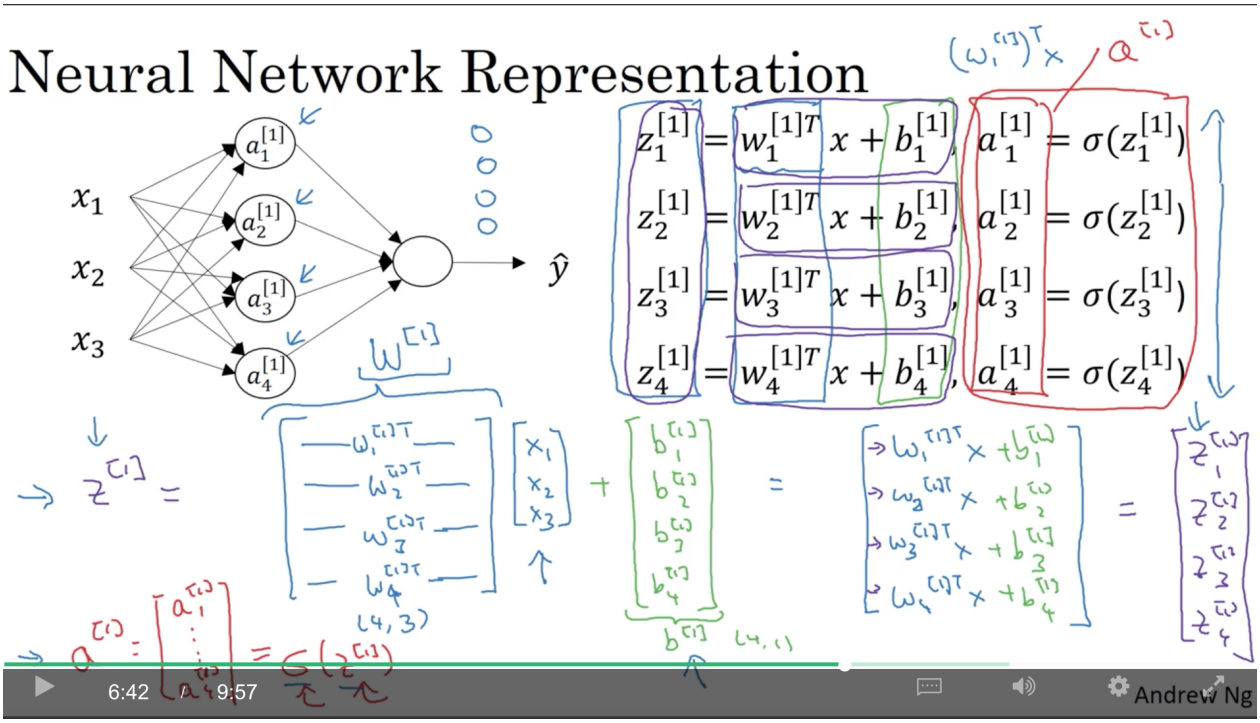
如果输入的x有3维，在lr中，$shape(w) = (1, 3)$，$shape(b) = (1, 1)$。

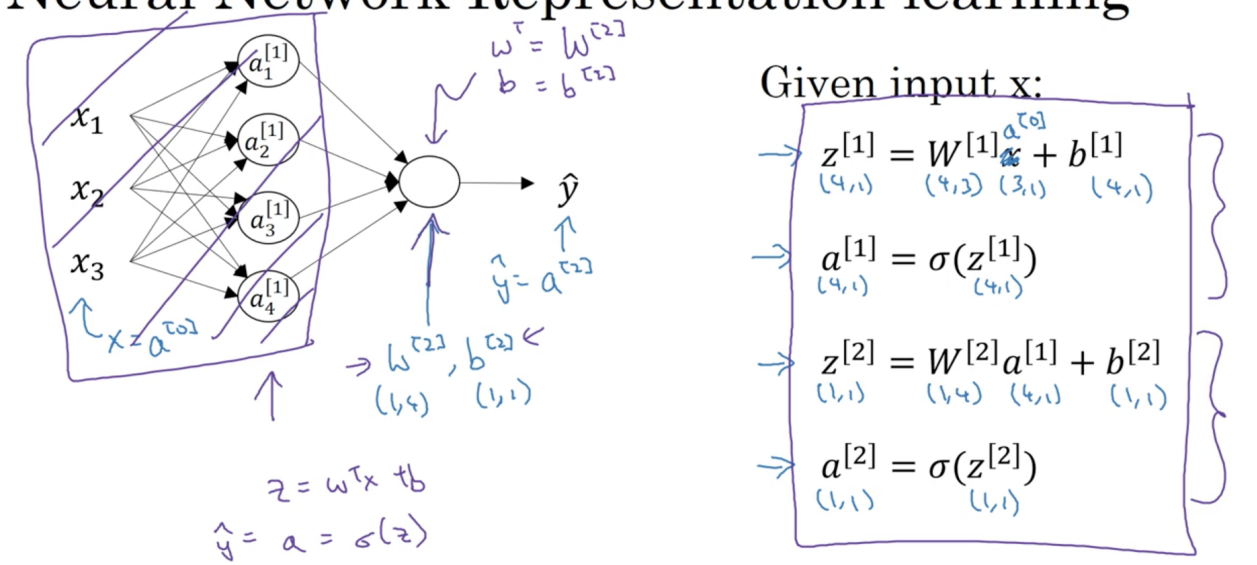
而在nn中，$shape(w^{[1]}) = (4, 3)$因为有4个神经元，输入是3维。同理$shape(b^{[1]}) = (4, 1)$。

而$shape(w^{[2]}) = (1, 4)$，因为只有1个神经元，输入是3维。同理$shape(b^{[2]}) = (1, 1)$。

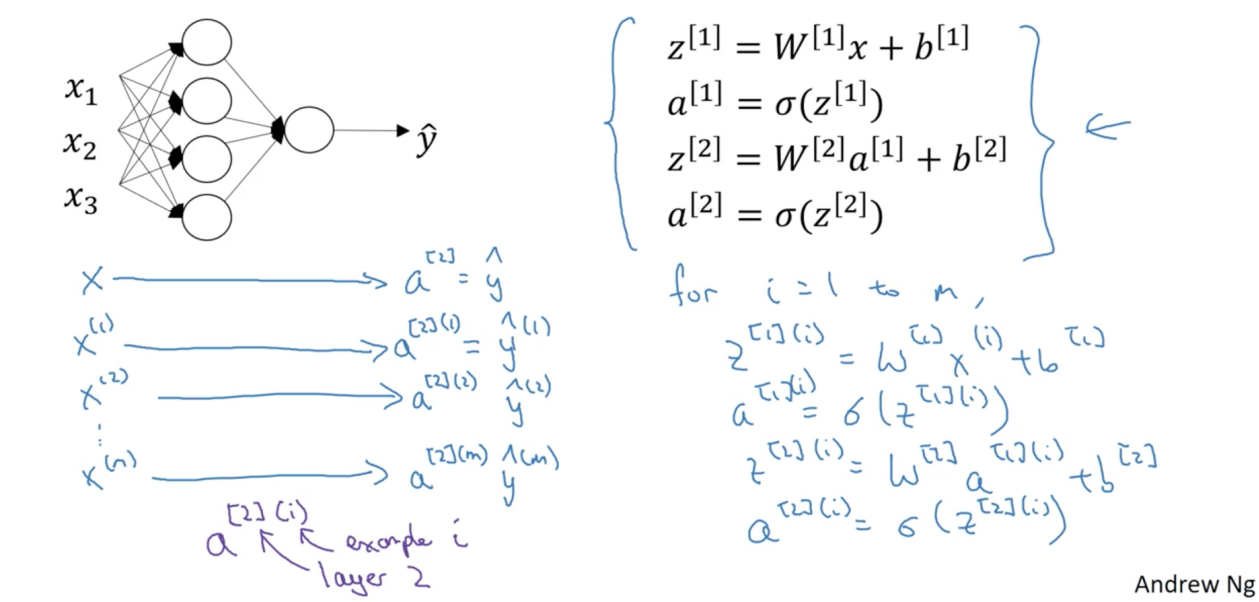


computing a neural network's output



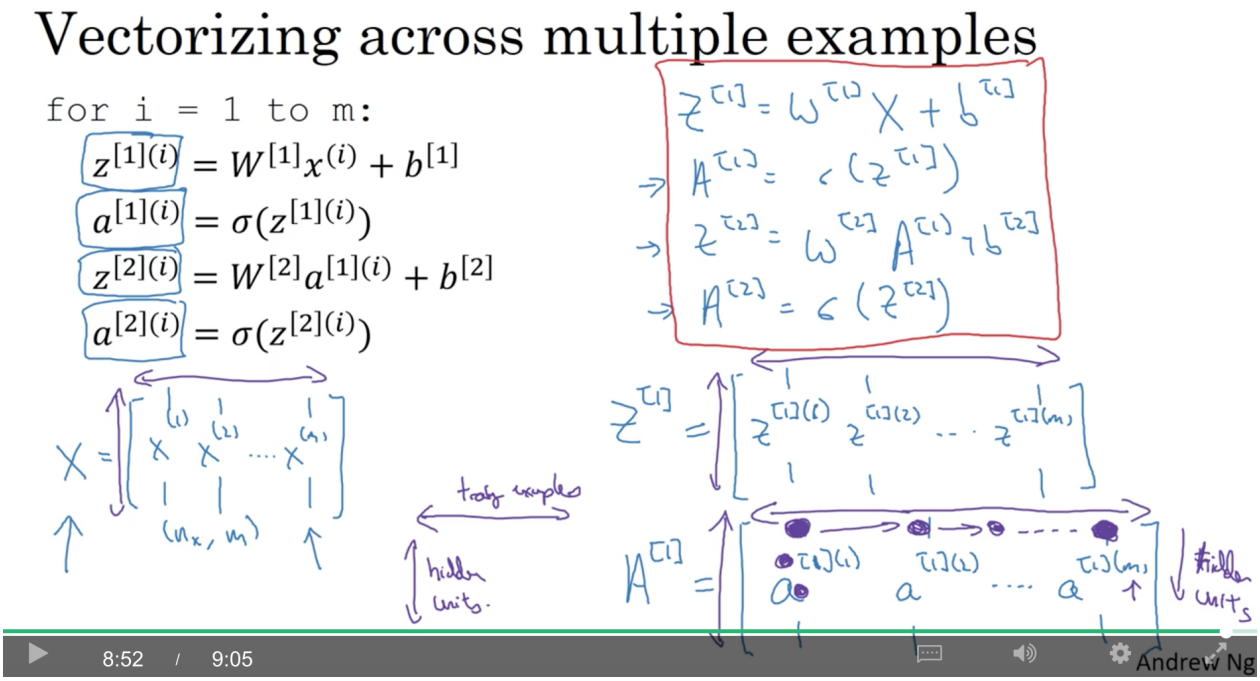


vectorizing across multiple examples

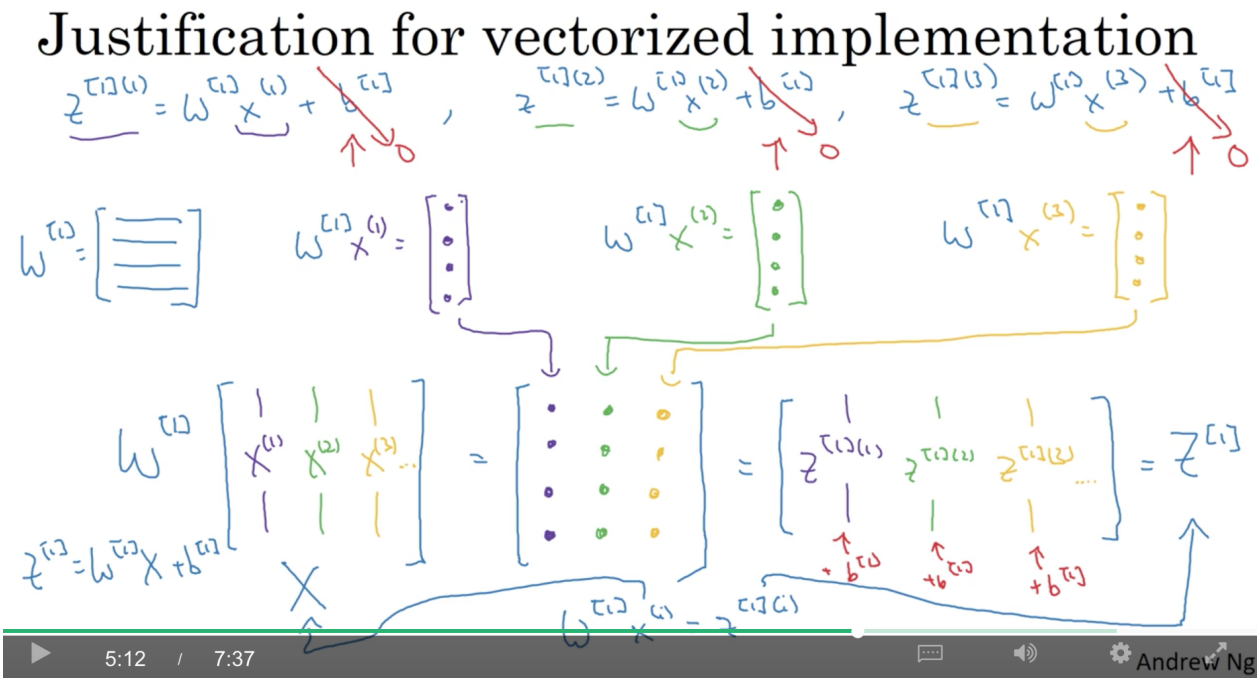


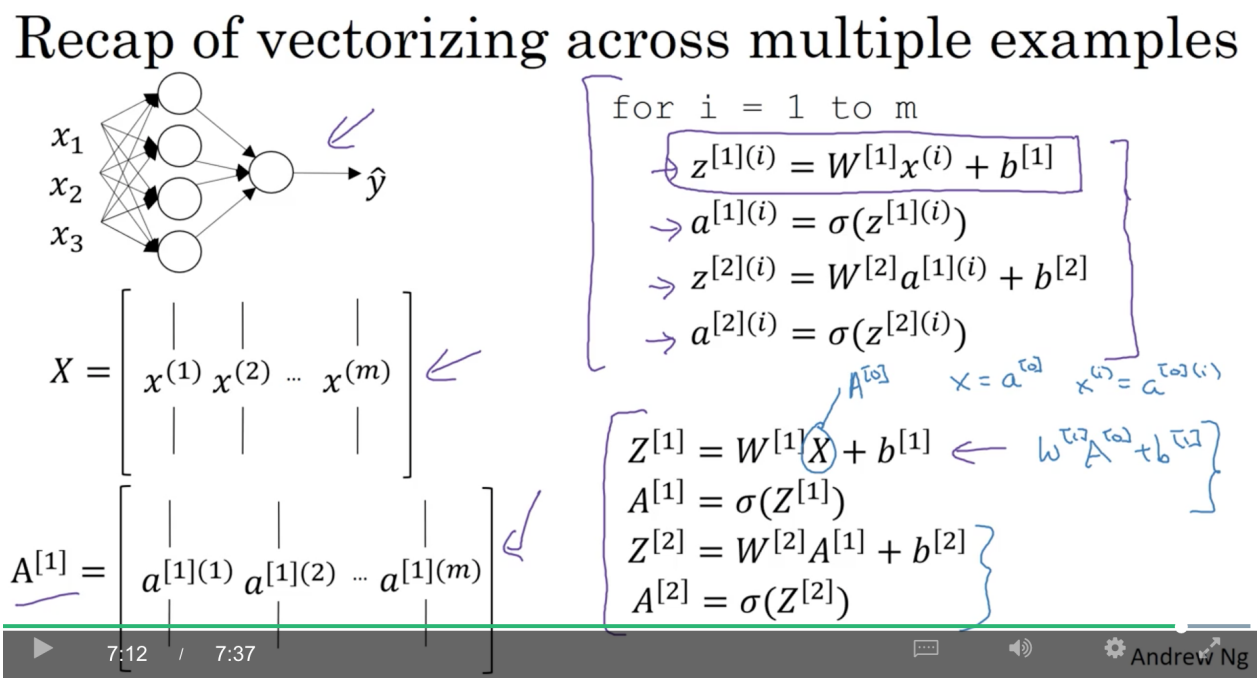
矩阵X纵向是x的维数（行数），横向是training examples的个数（列数）。

矩阵Z、A纵向是hidden units的个数（行数），横向是training examples的个数（列数）。



explanation for vectorized implementation

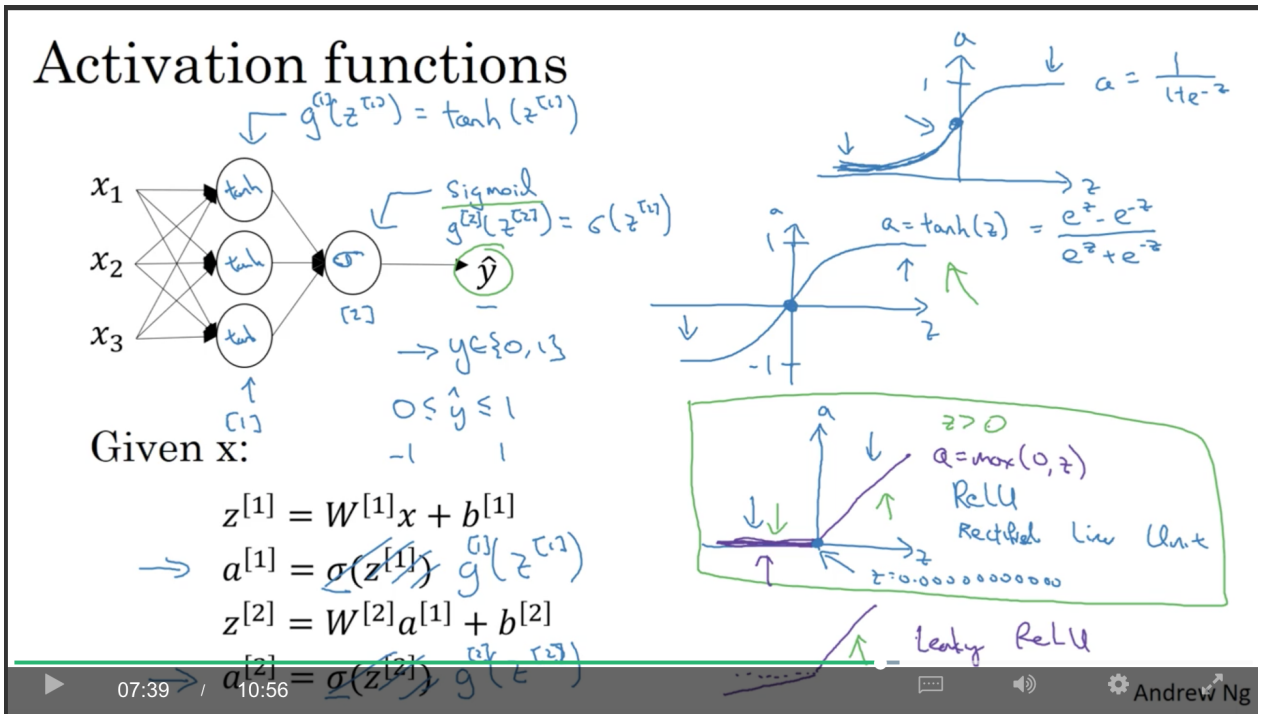




activation functions

一般来说，$tanh$效果比$sigmoid$好，因为均值是0。但对于outputlayer而言，一般$y \in \{0,1\}$，所以希望$0 \leq \hat{y} \leq 1$，所以会用$sigmoid$。

$ReLU(z) = max(0,z)$比$tanh$好，因为当$x \leq 0$时，梯度为0。而$leakyReLU$在$x \leq 0$时，梯度是接近0，效果会好一点，但在实践中还是$ReLU$居多。当$x > 0$时，梯度和$x \leq 0$差很多，所以训练速度会加快。

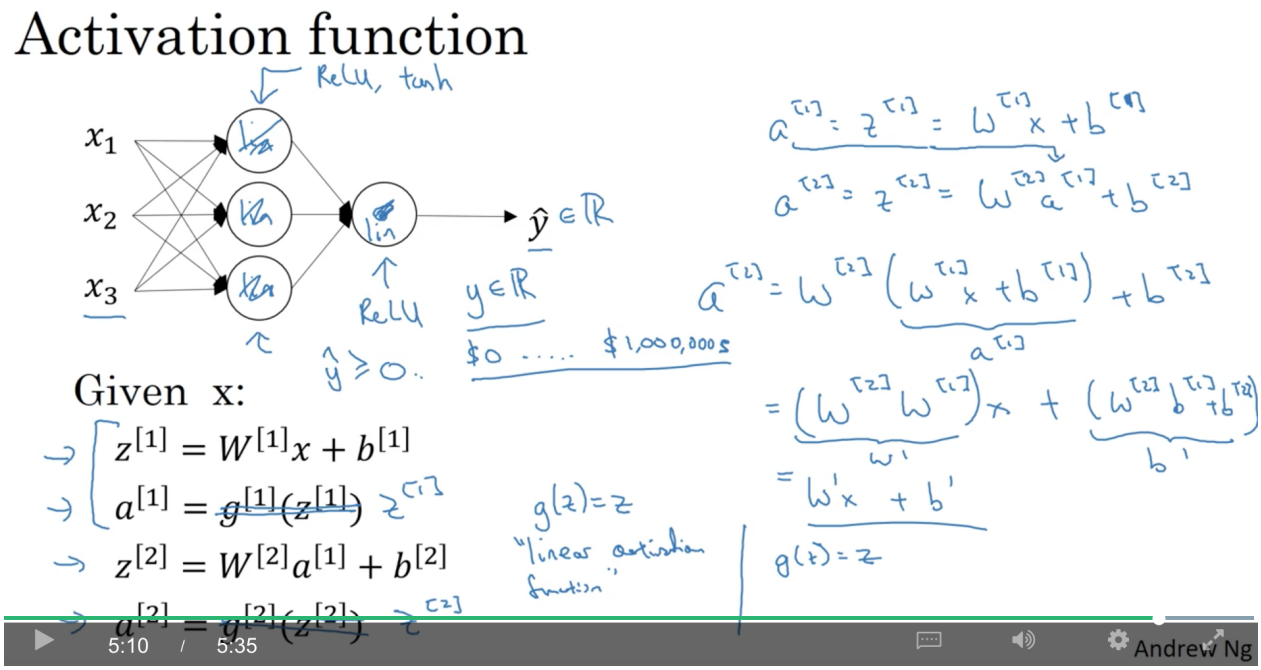


why do you need non-linear activation functions

linear activation: 因为$z = wx + b$，所以激活函数$g(z) = z = wx + b$就叫linear activation function，也叫identity activatioin function。

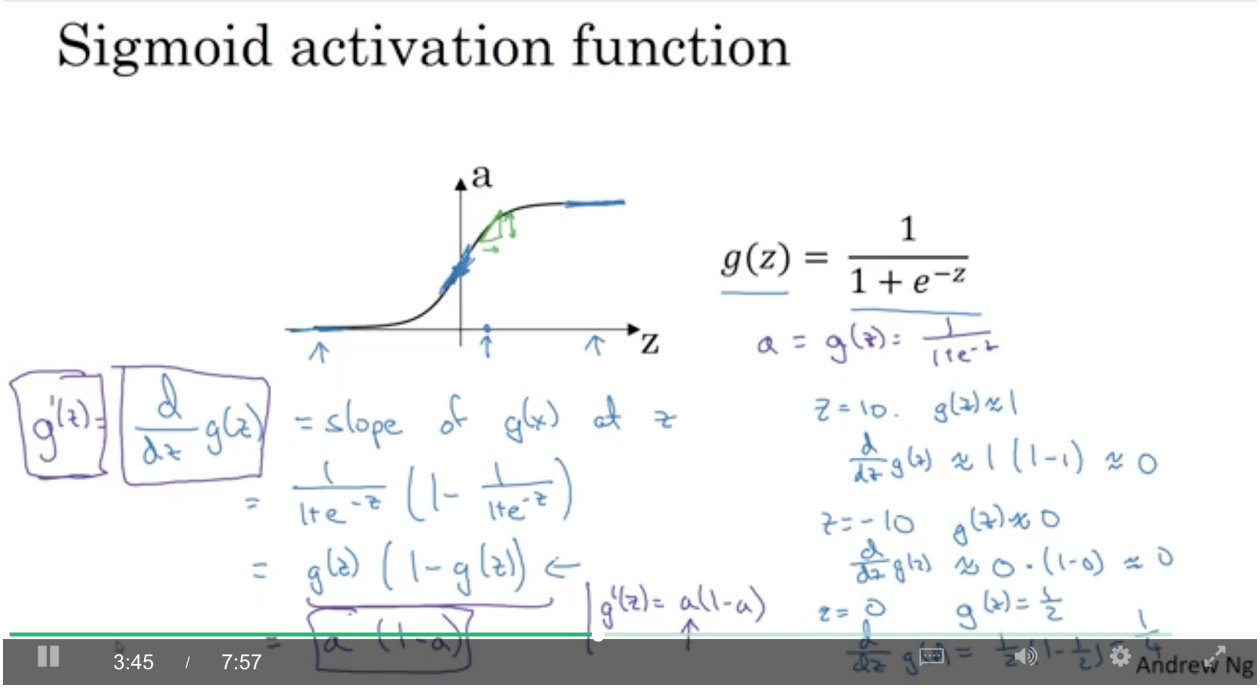
不要在hidden layer用linear activation functions，因为多个linear嵌套，实质上还是linear。

例外，当进行回归时，$y \in R$，可以hidden layer用$ReLU$，但output layer用linear activation。

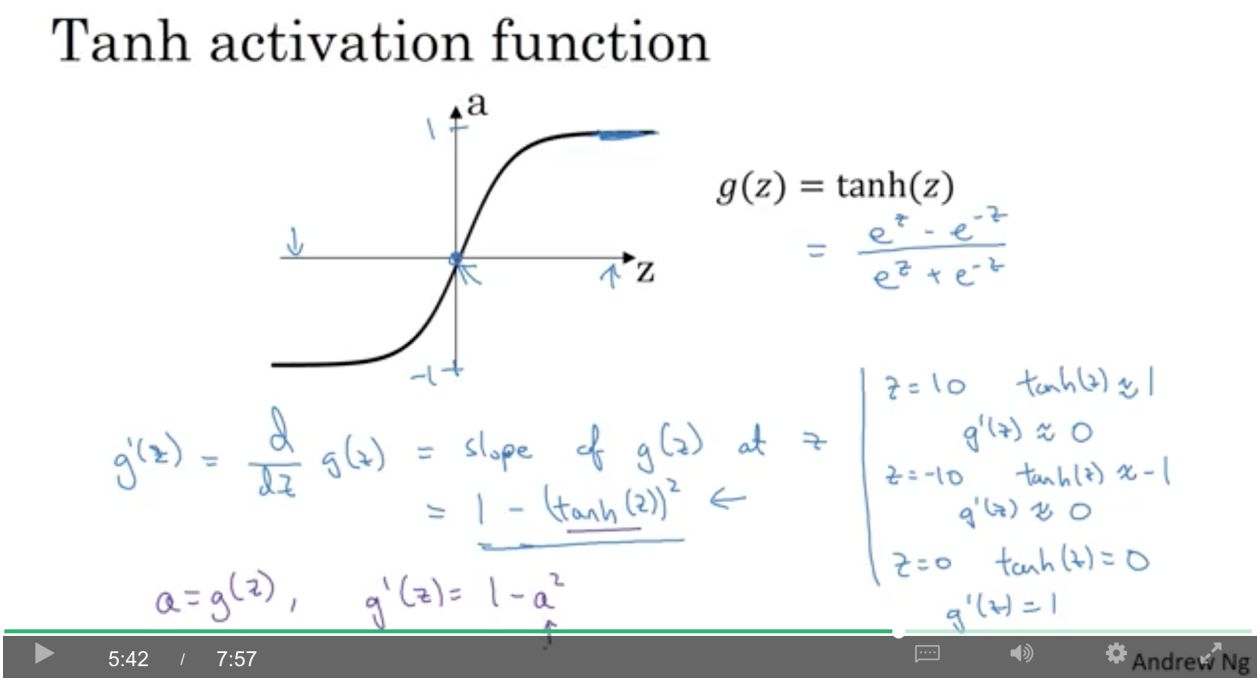


derivatives of activation functions

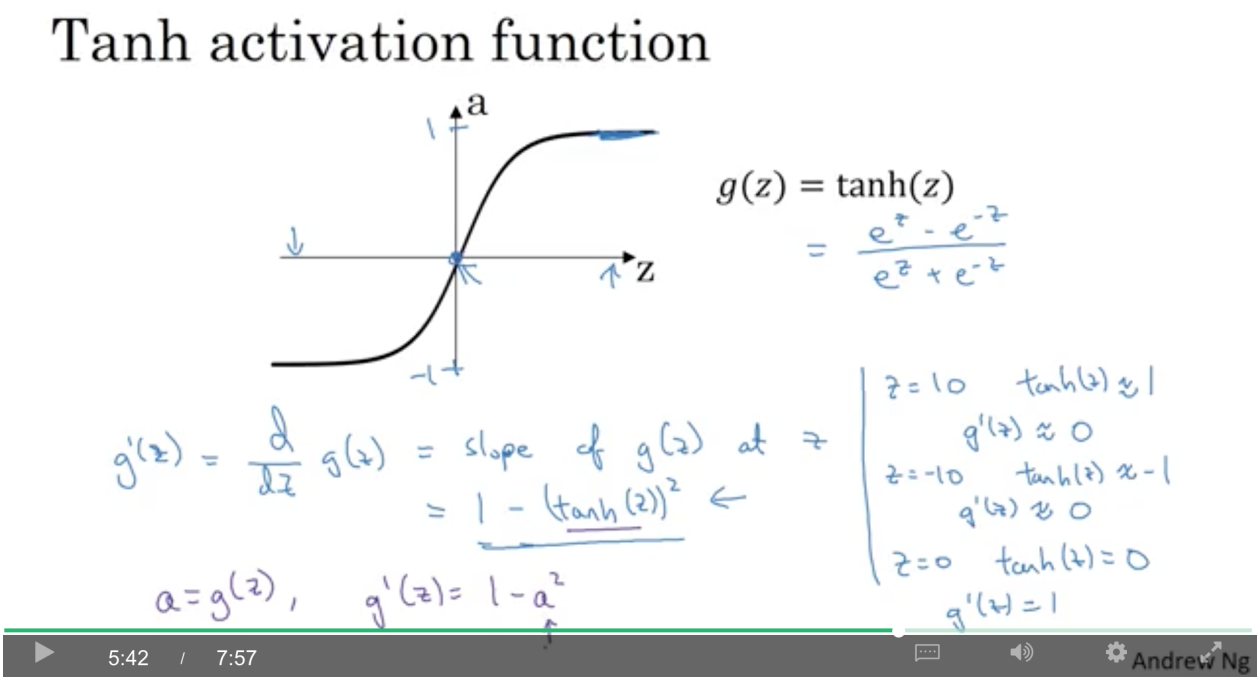
sigmoid的导数



tanh的导数

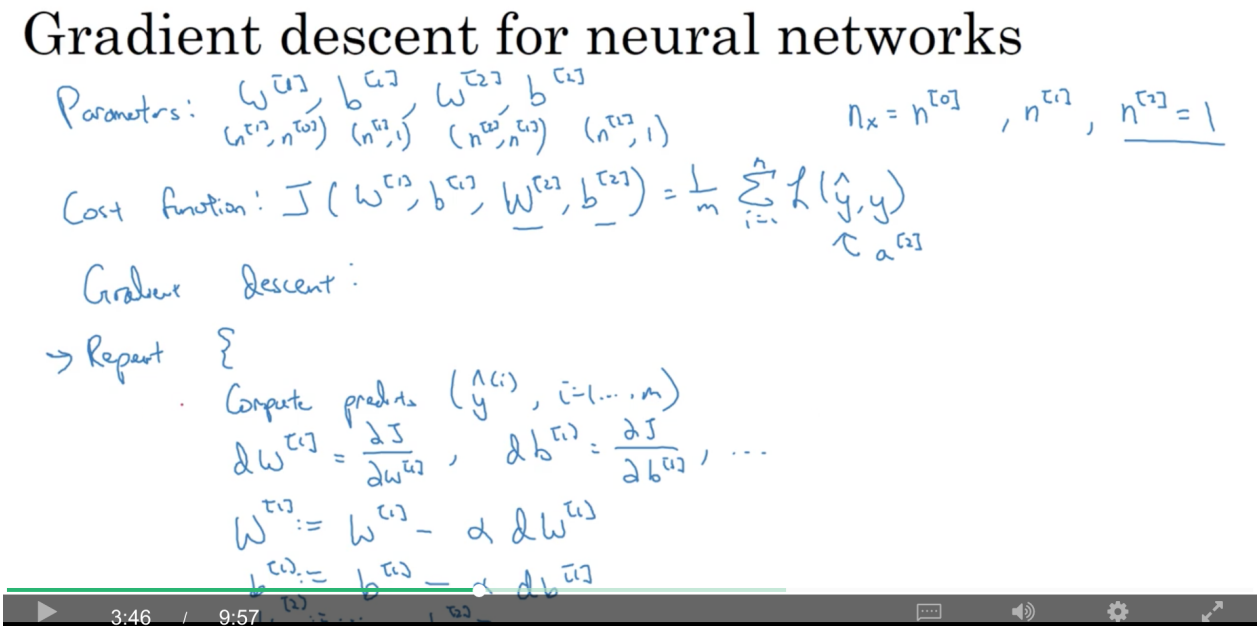


relu和leaky relu的导数(z=0时不可导，但在工程上，直接归入z>0部分)

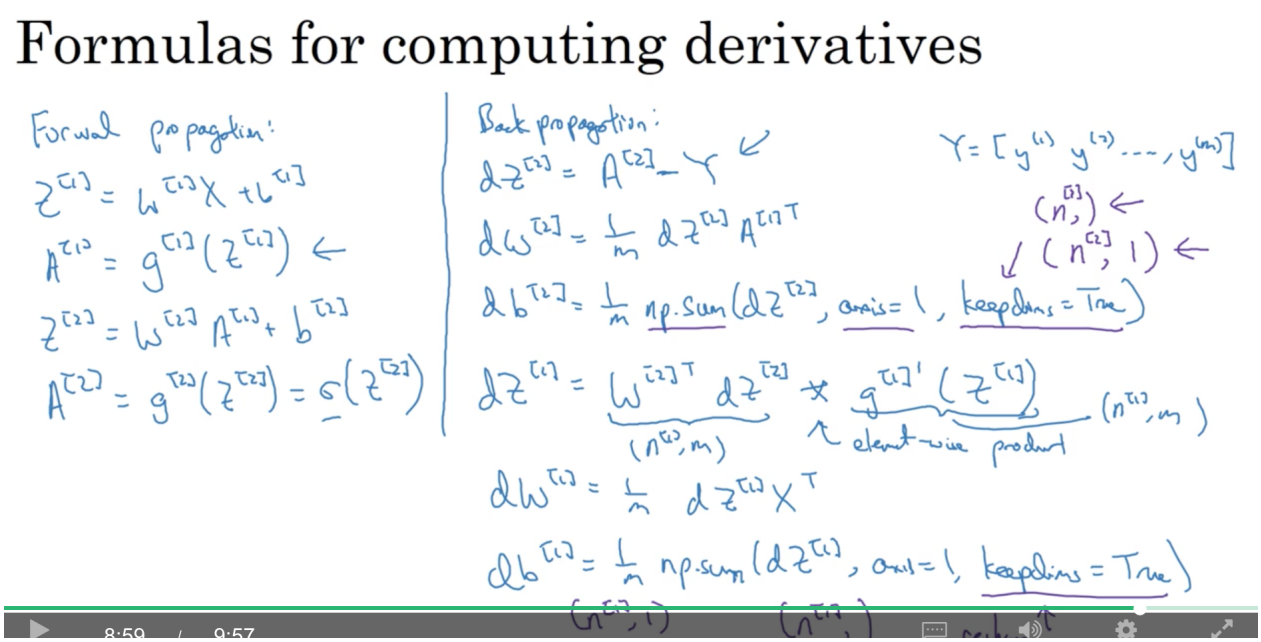


gradient descent for neural networks

记住每个W/b的shape!

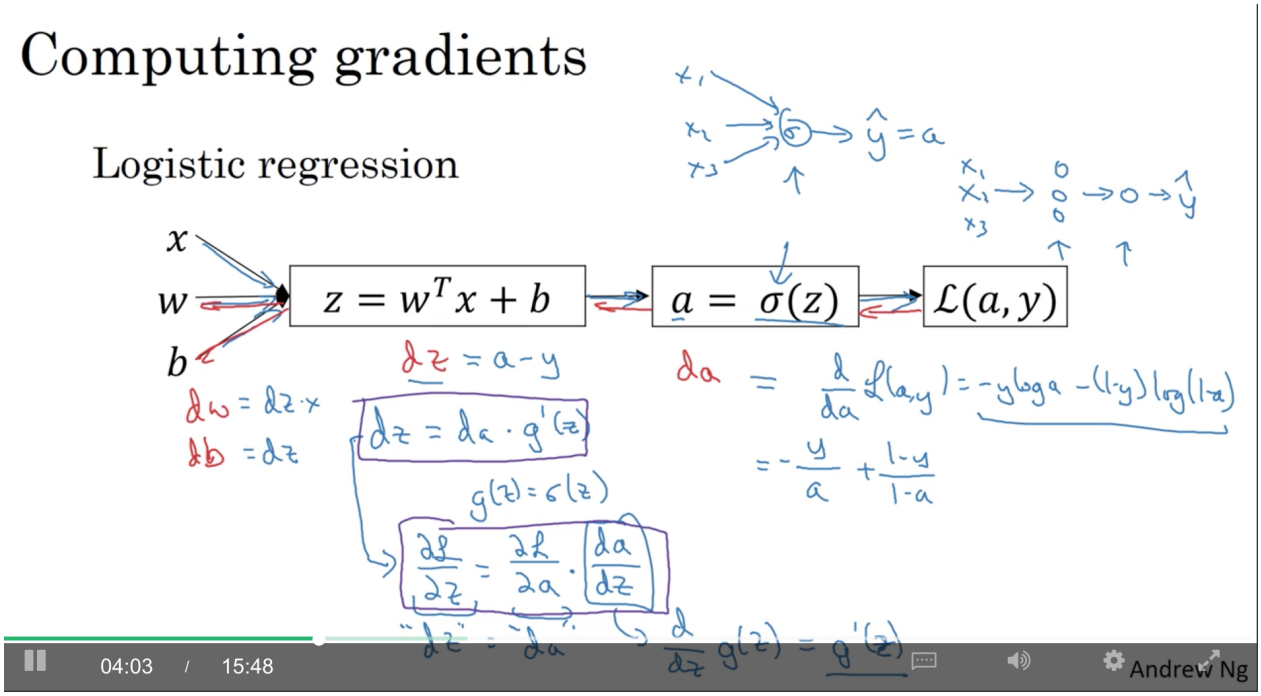


其中的keepdims=True表示，输出的$shape = (n^{[2]}, 1)$而非$(n^{[2]},)$ 另外求dz时，两项之前是element-wise product(np.multiply)，其中第二项就是对激活函数求在$z^{[1]}$的导数

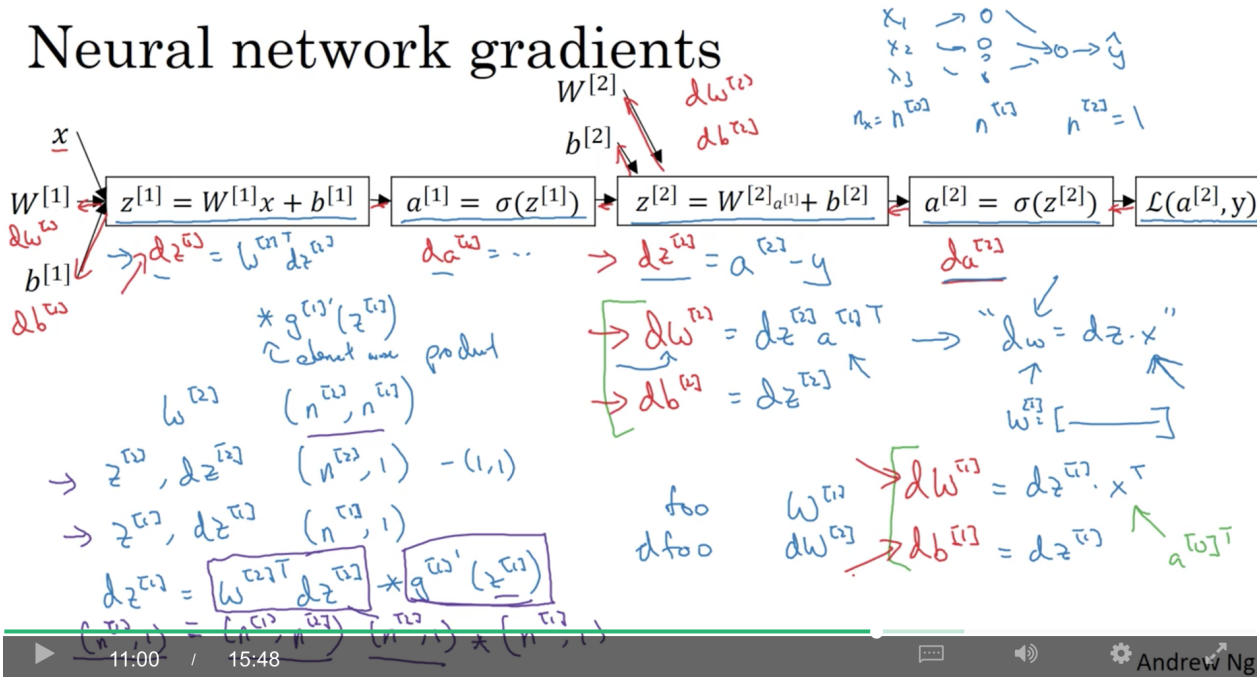


backpropagation intuition

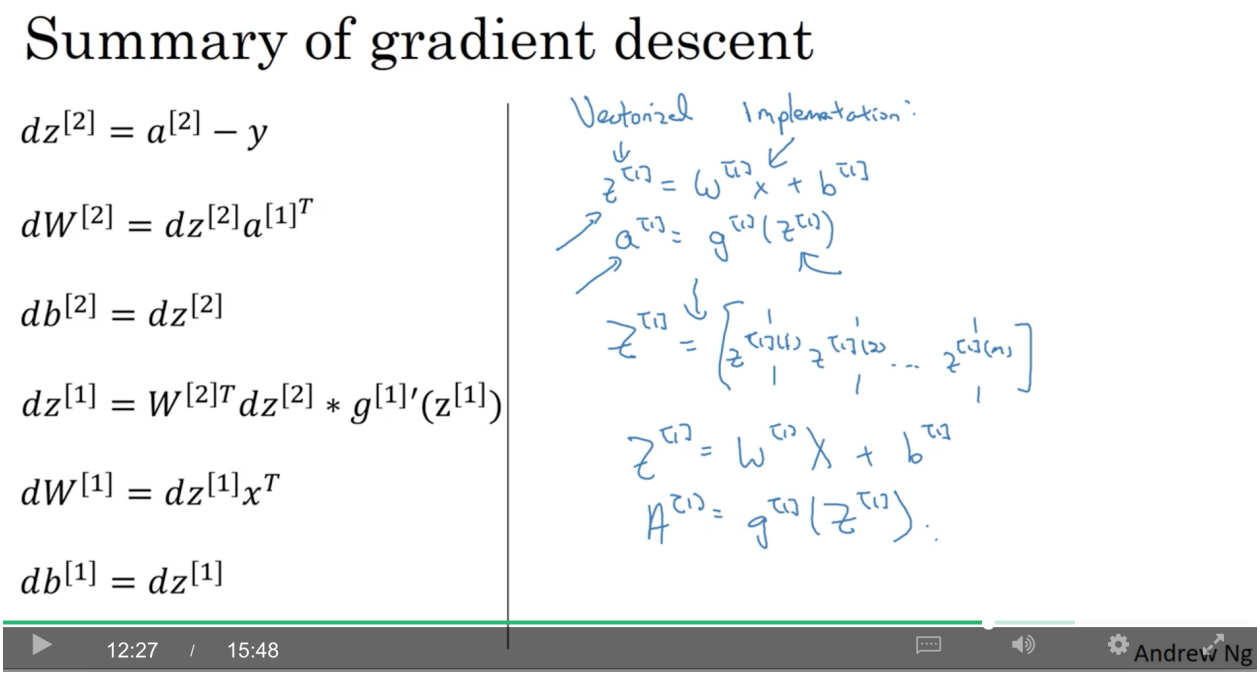
先回顾一下lr：

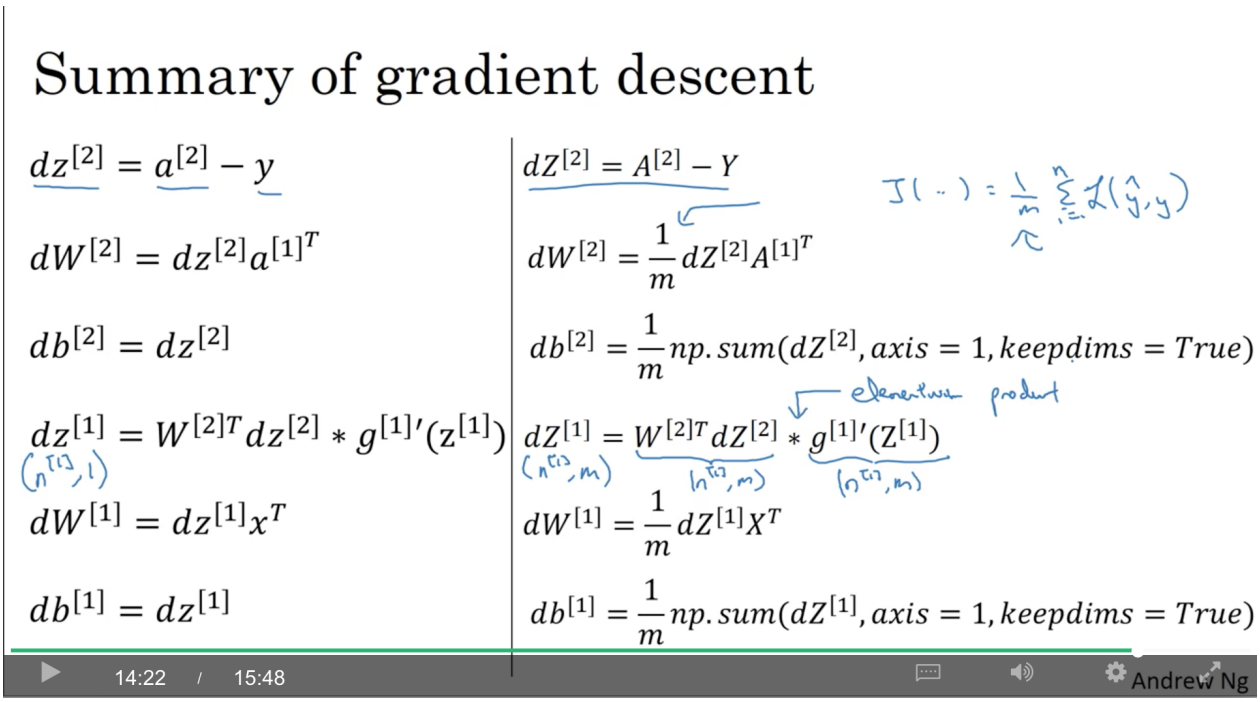


然后看nn：


Neural network gradients
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$
11:00 / 15:48
Andrew Ng

扩展到m个examples，并进行vectorize：

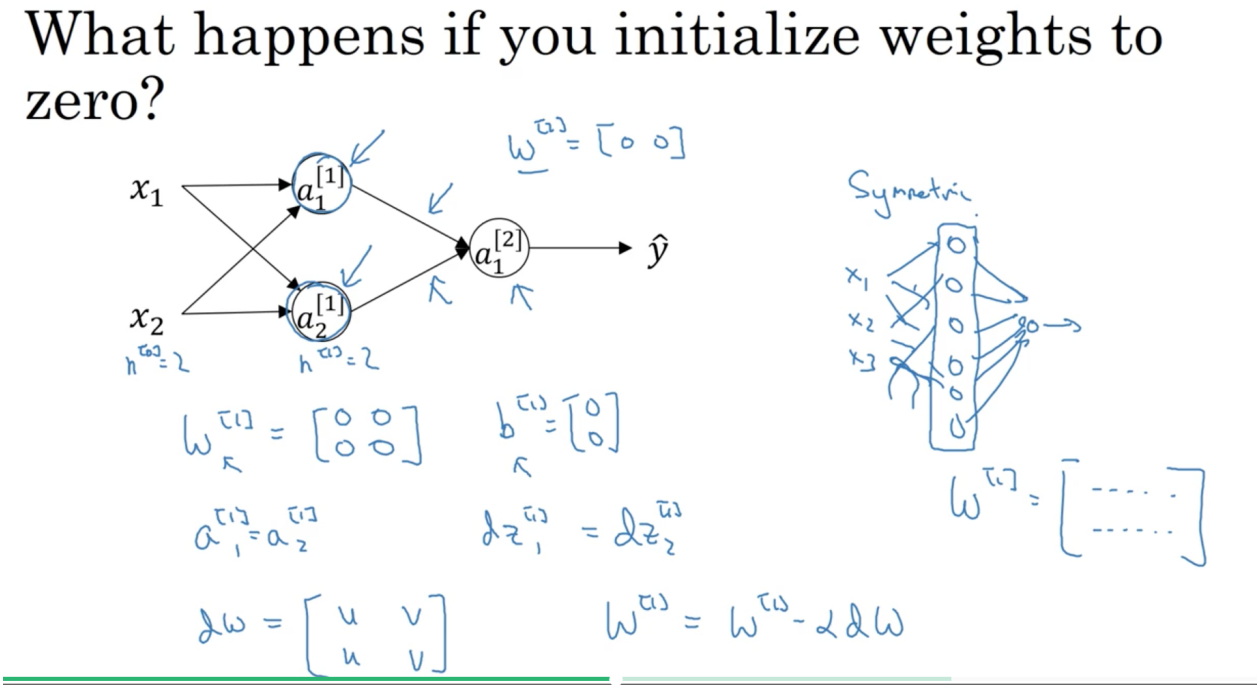




问：为何有的有1/m，有的没有。。。。。

random initialization

lr的训练，参数一般都初始化为0。但nn，如果初始化为0，会发现算出来的$a_1^{[1]}=a_2^{[1]}$，$dz_1^{[1]}=dz_2^{[1]}$，所以相当于每个神经元的influence是一样的(symetric)，所以$dw^{[1]}$这个矩阵的每一行都相等。 "Each neuron in the first hidden layer will perform the same computation. So even after multiple iterations of gradient descent each neuron in the layer will be computing the same thing as other neurons."



解决：随机初始化 只要w随机初始化了，b其实影响不大，0就可以了。如果用的是sigmoid/tanh的话，随机初始化时，尽量小，因为如果大的话，激活后会接近两端(无论是+无穷还是-无穷，梯度都接近0)，导致学习过程变得很慢。**对于浅层的网络，0.01就可以了，但如果是更深的网络，可能需要其他系数，这个选择过程，在后面的课程会讲。。。**