Contents

contents

1 1. shallow neural network

1.1 1.1. 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。

1.2 1.2. 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 $shape(b^{[2]}) = (1, 1)$。

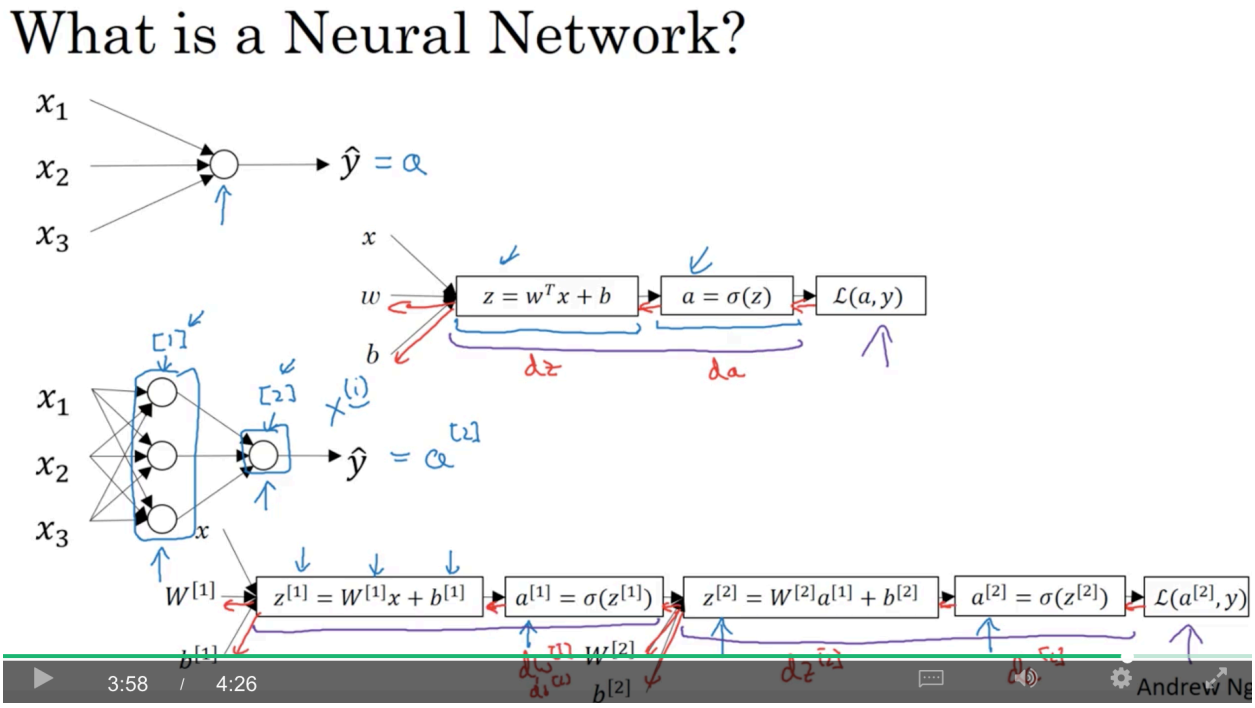


Figure 1: neural-networks-overview

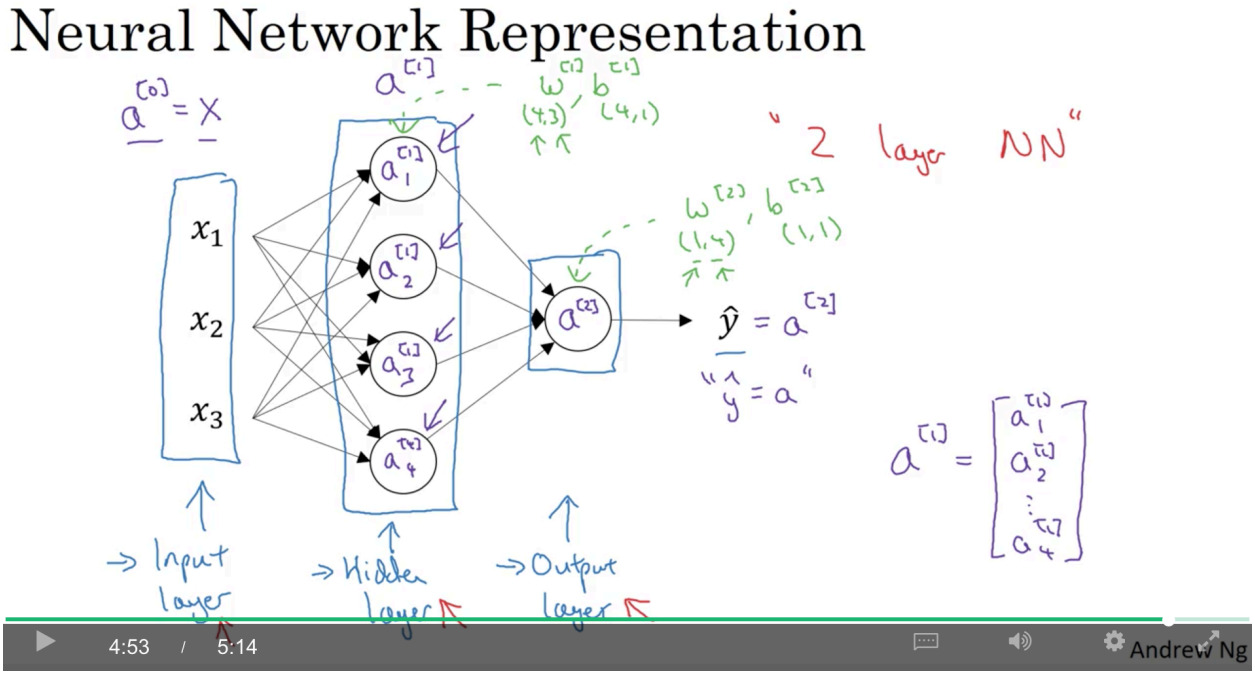
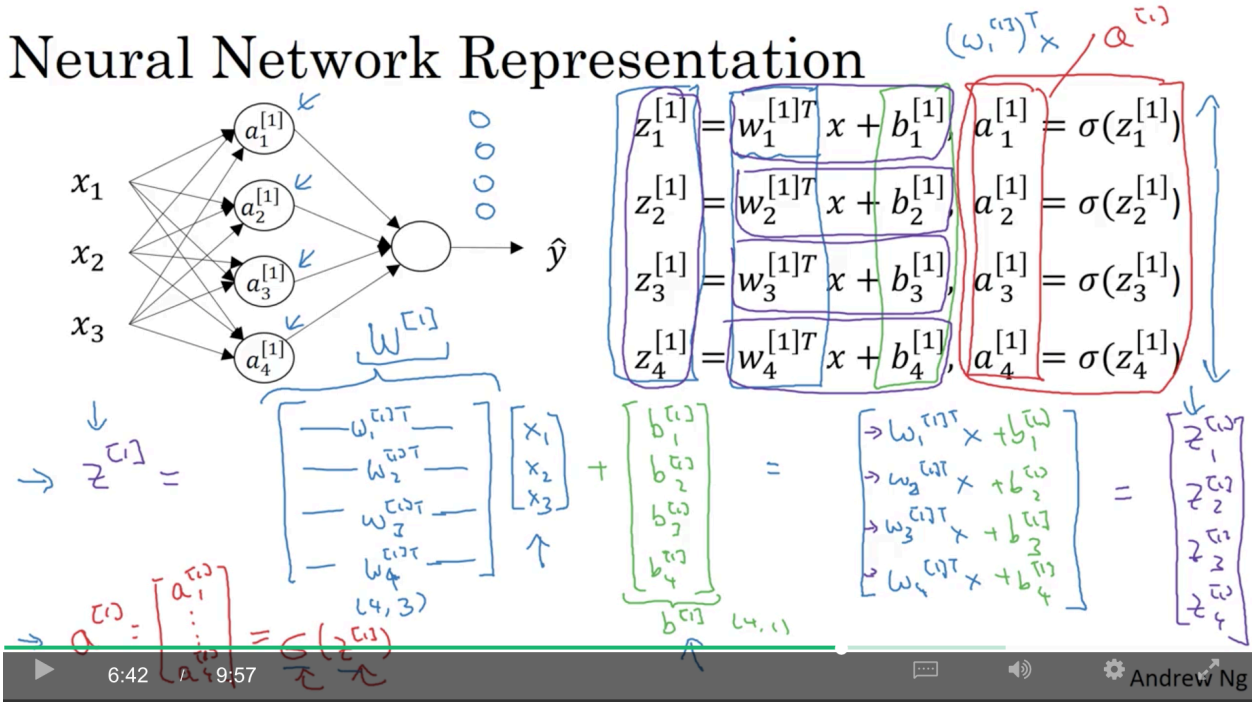
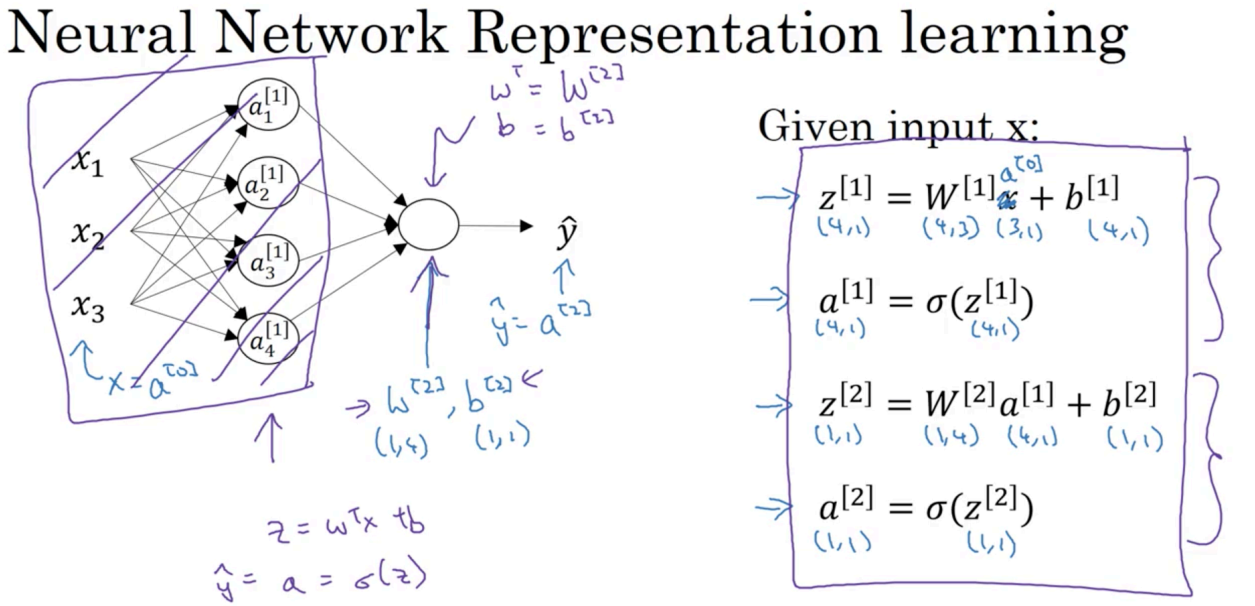


Figure 2: neural-network-representation

1.3 1.3. computing a neural network's output





1.4 1.4. 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）。

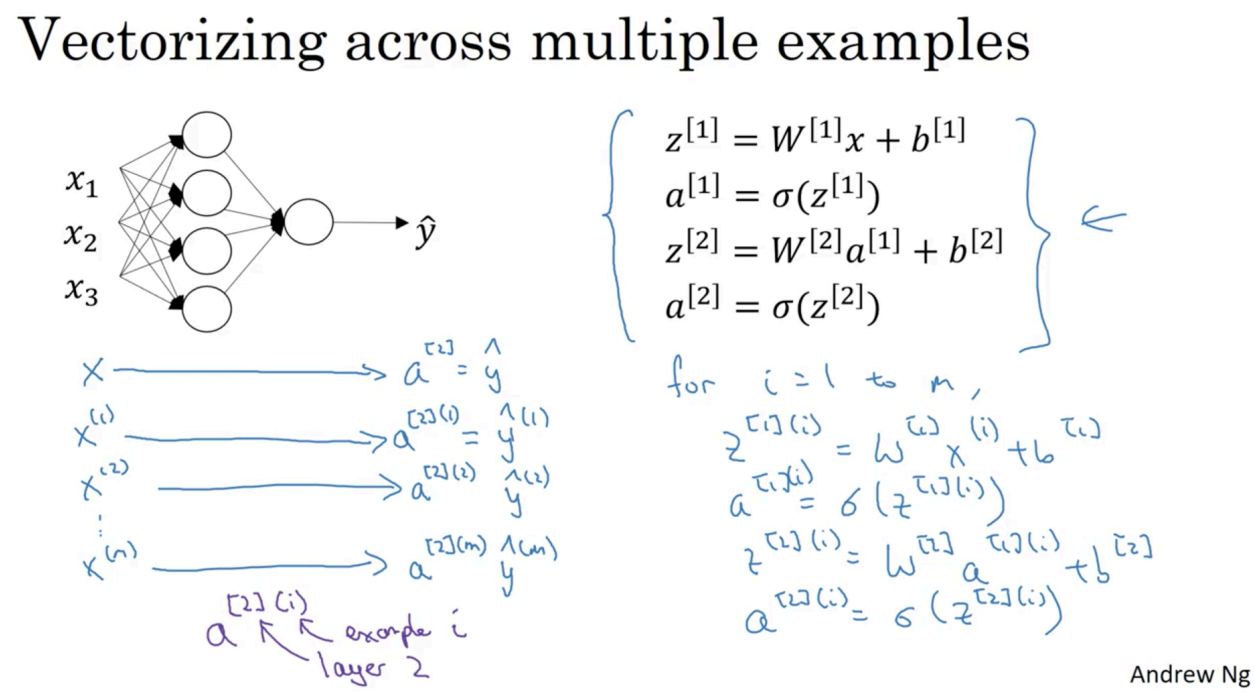


Figure 3: vectorizing-across-multiple-examples-for-loop

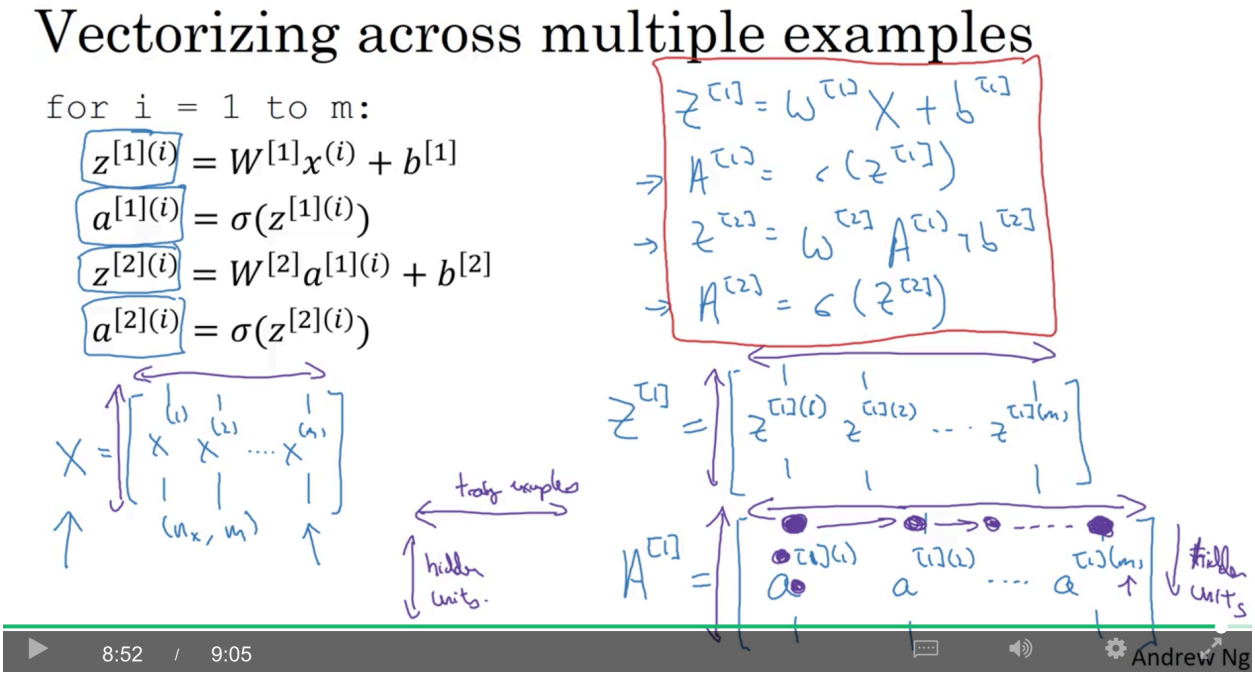
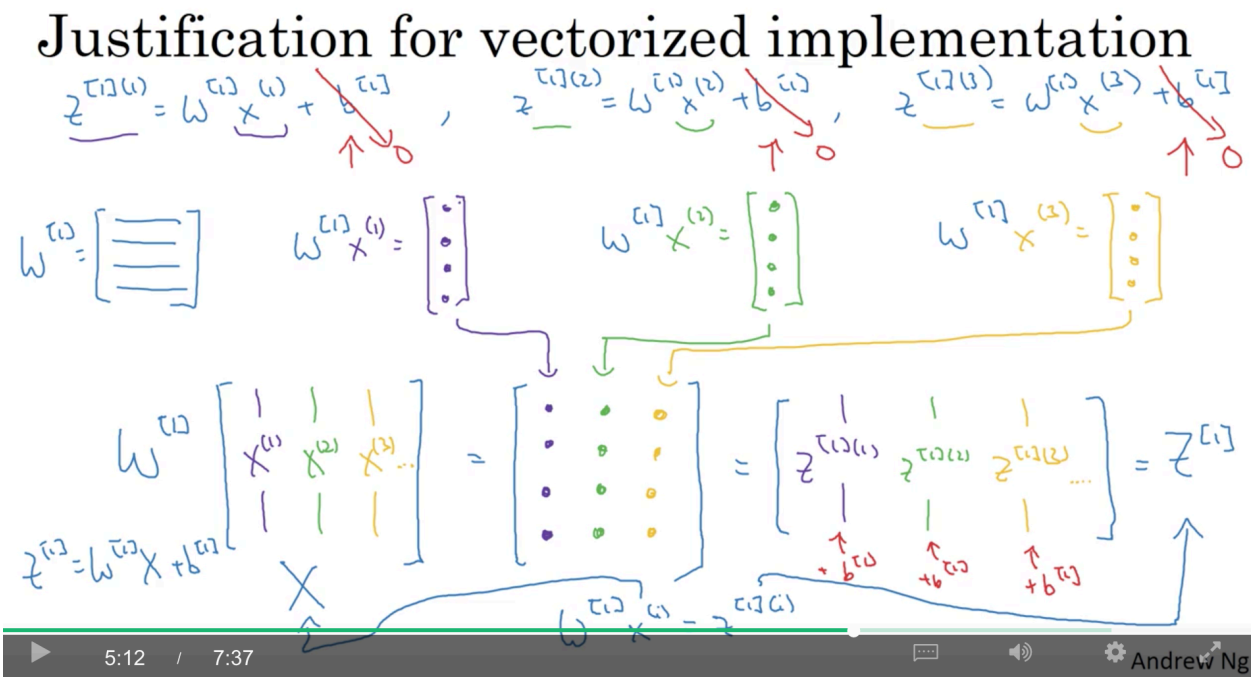
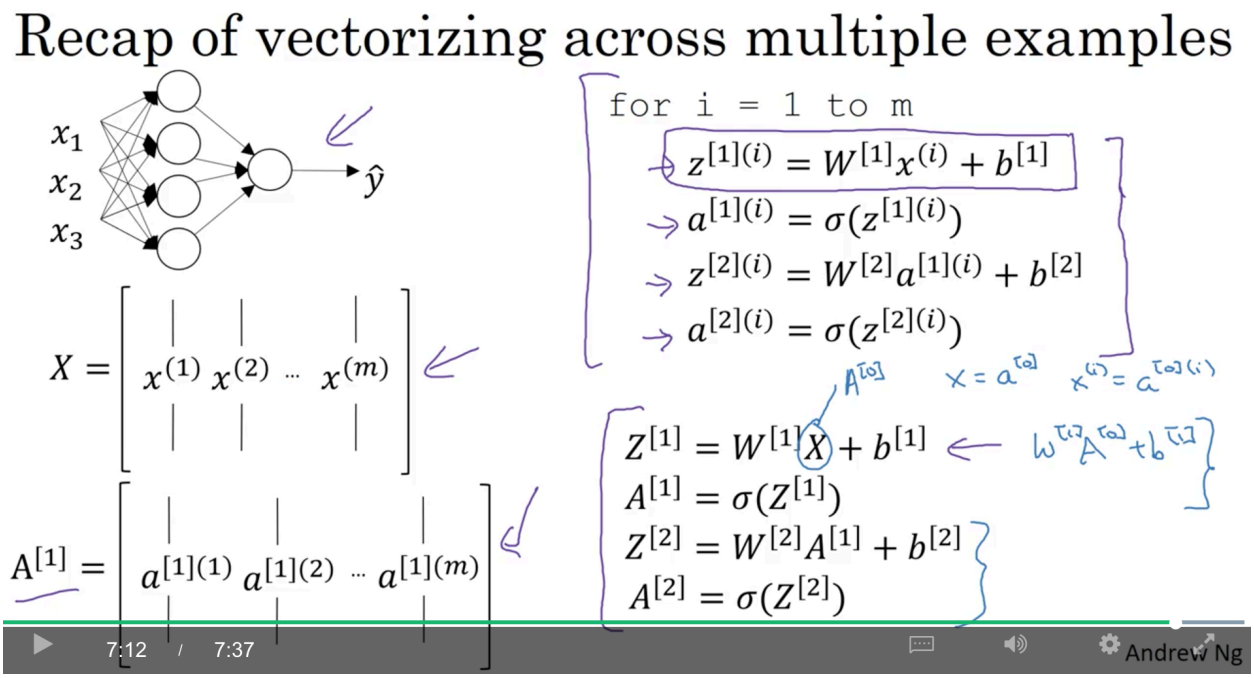


Figure 4: vectorizing-across-multiple-examples-vectorization

1.5 1.5. explanation for vectorized implementation





1.6 1.6. 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。

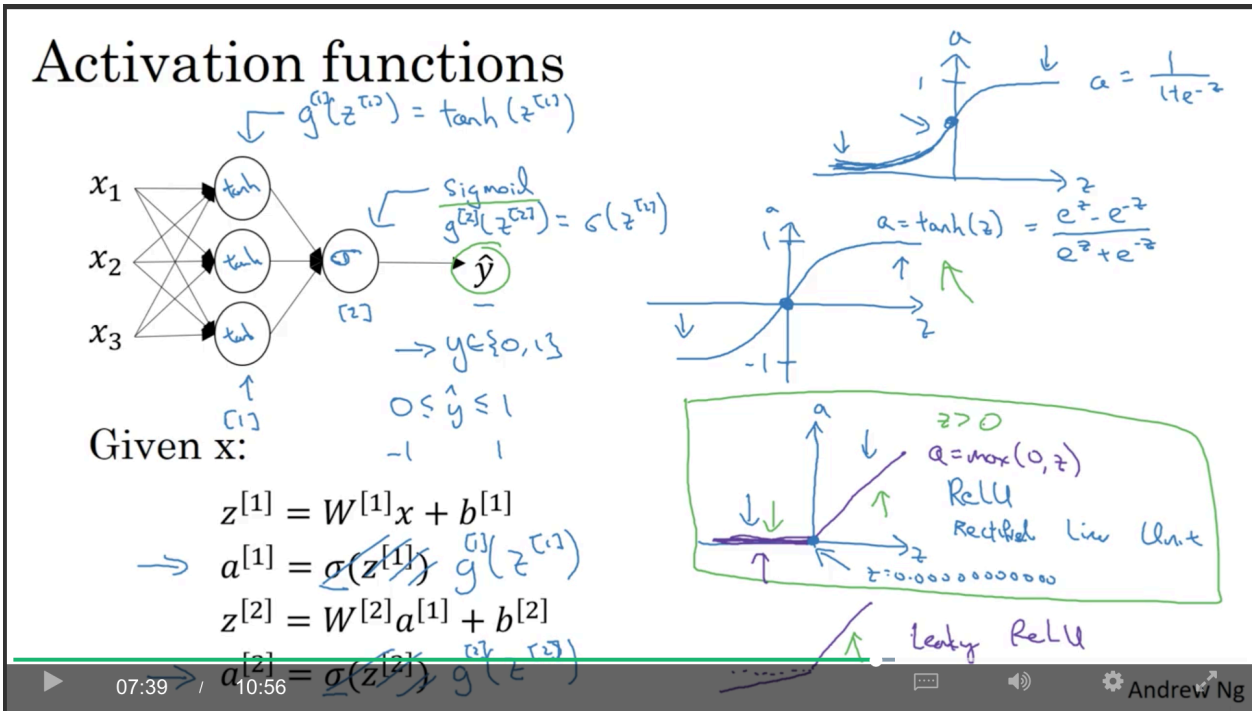


Figure 5: activation-functions

1.7 1.7. 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。

1.8 1.8. 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)

1.9 1.9. 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

1.10 1.10. backpropagation intuition

先回顾一下 lr:

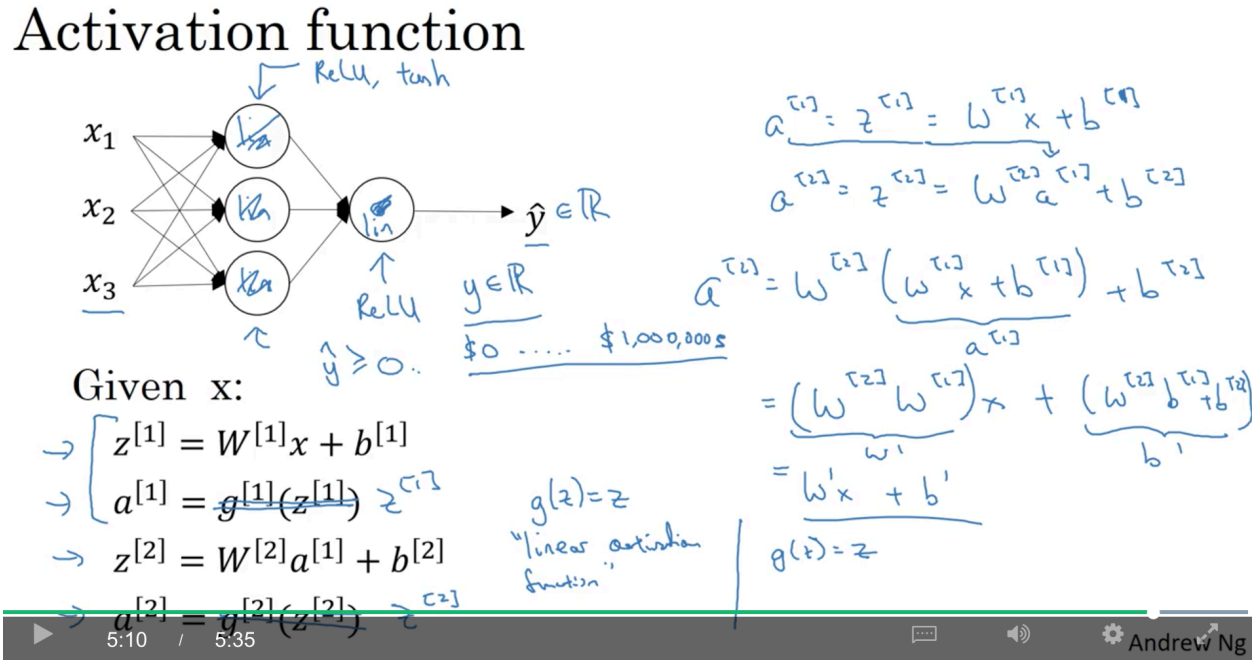


Figure 6: why-use-non-linear-functions.png

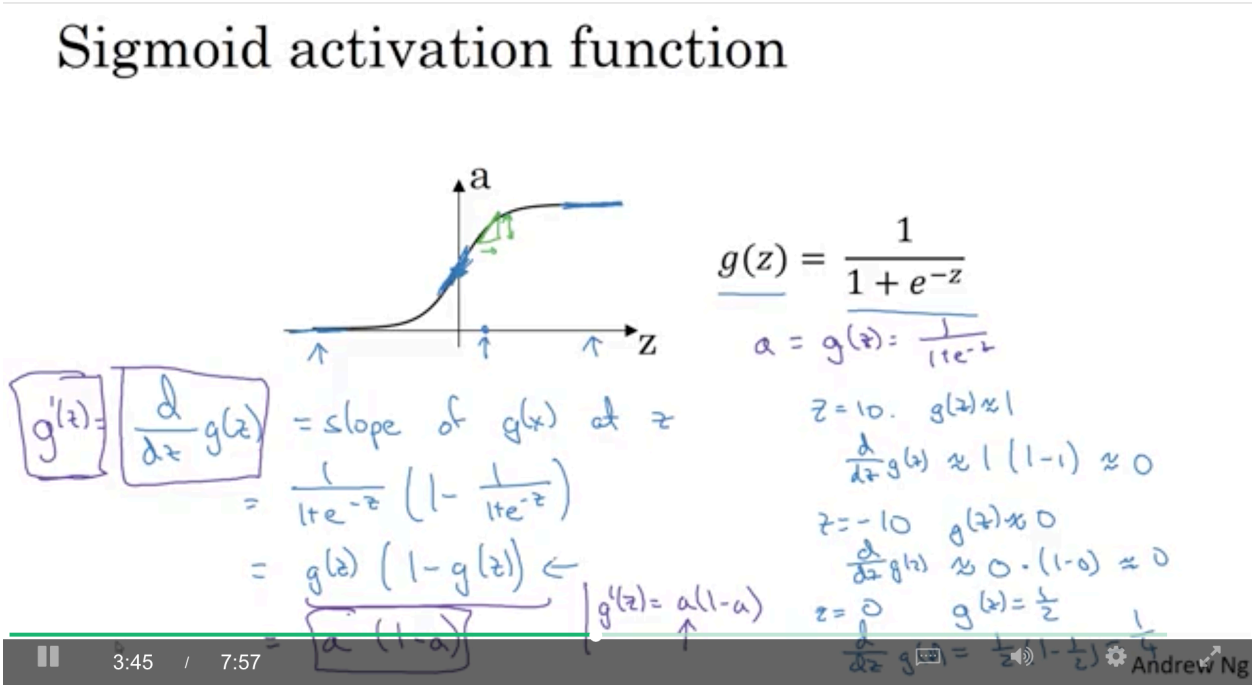


Figure 7: derivative-of-sigmoid.png

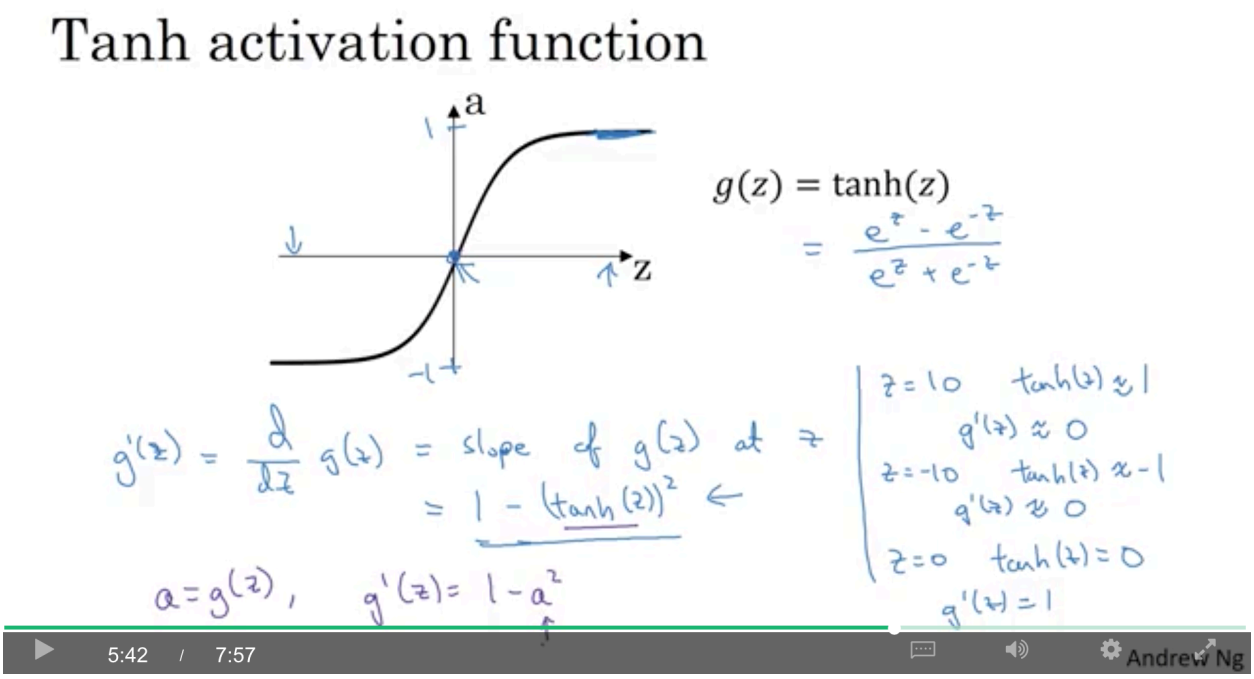


Figure 8: derivative-of-tanh.png

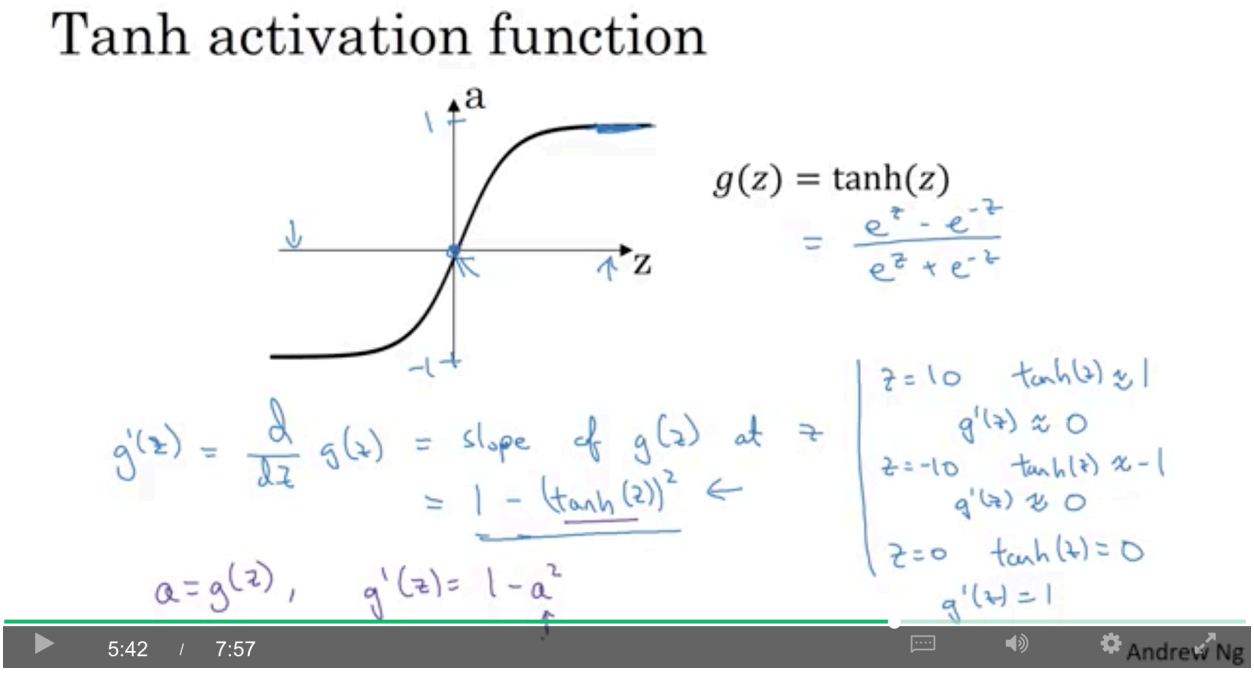


Figure 9: derivative-of-tanh.png

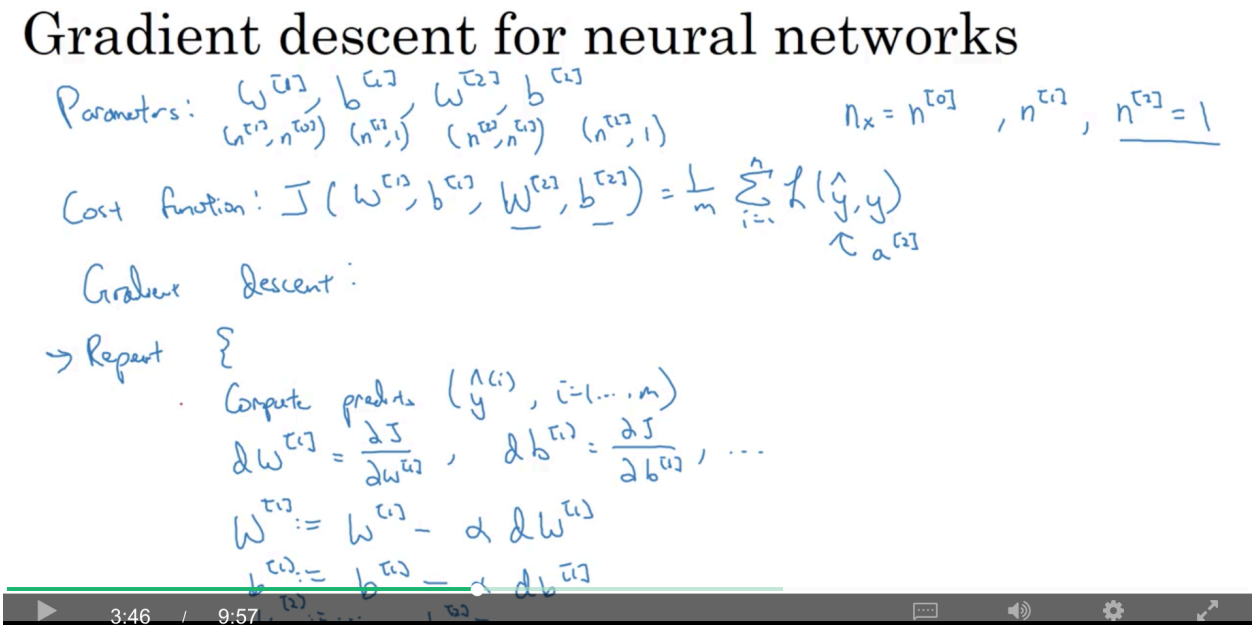


Figure 10: gradient-descent-for-neural-networks

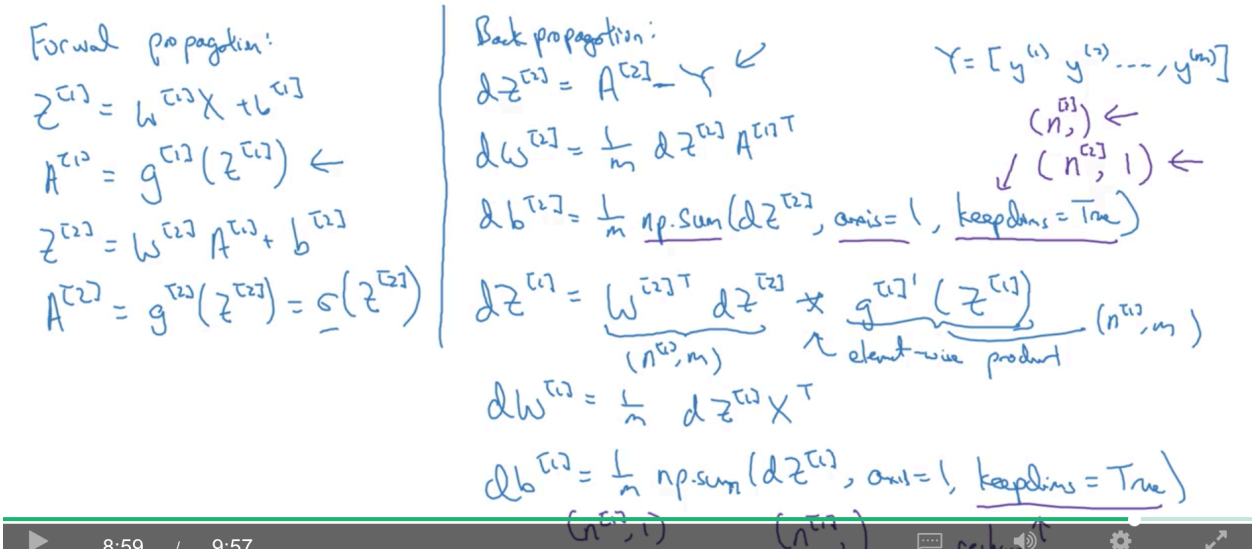


Figure 11: forward-propagation-and-back-propagation

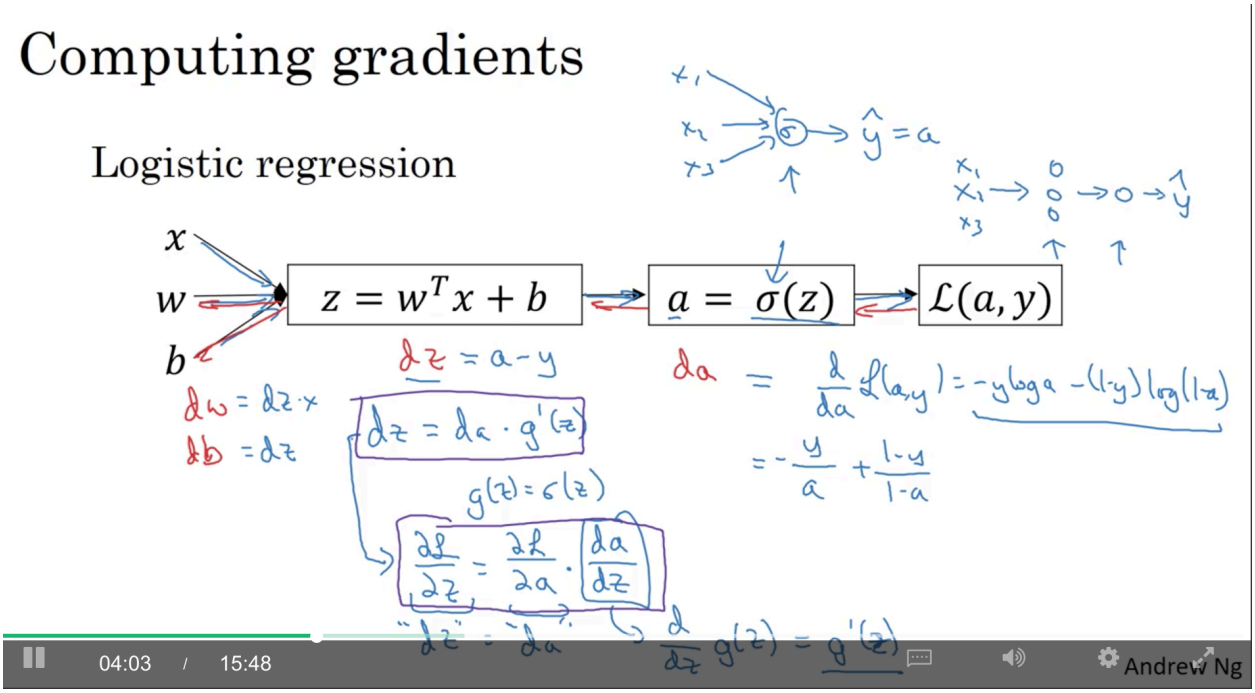


Figure 12: back-propagation-lr

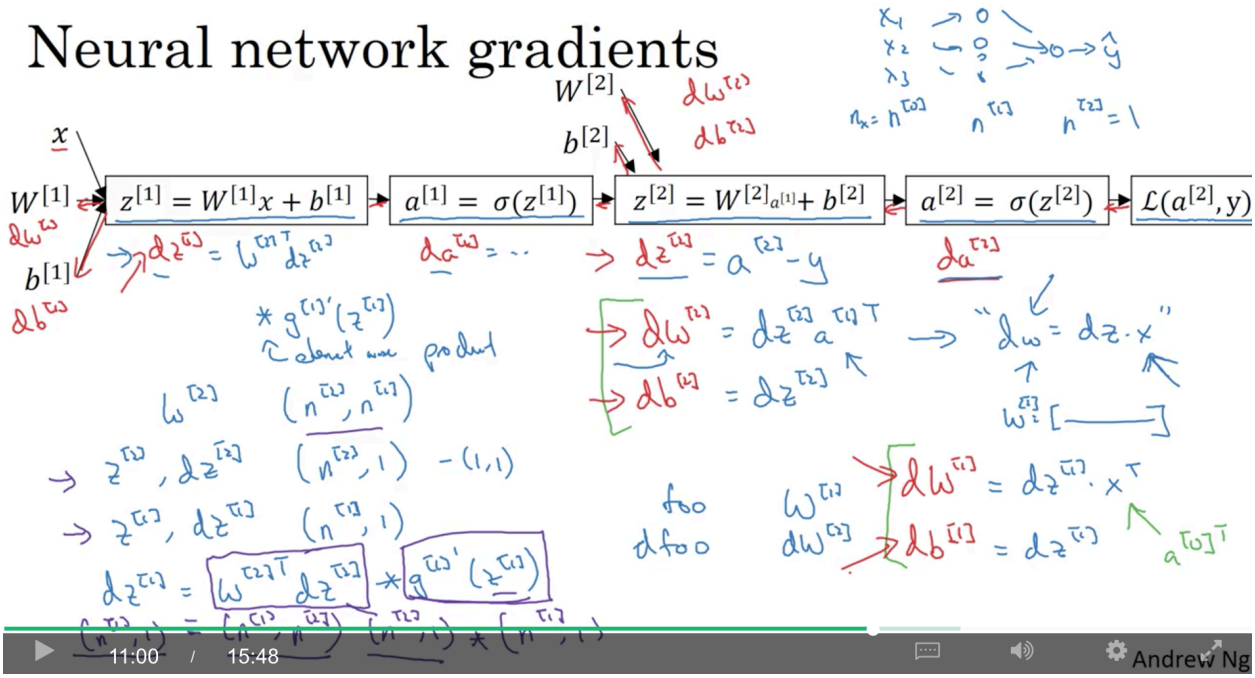
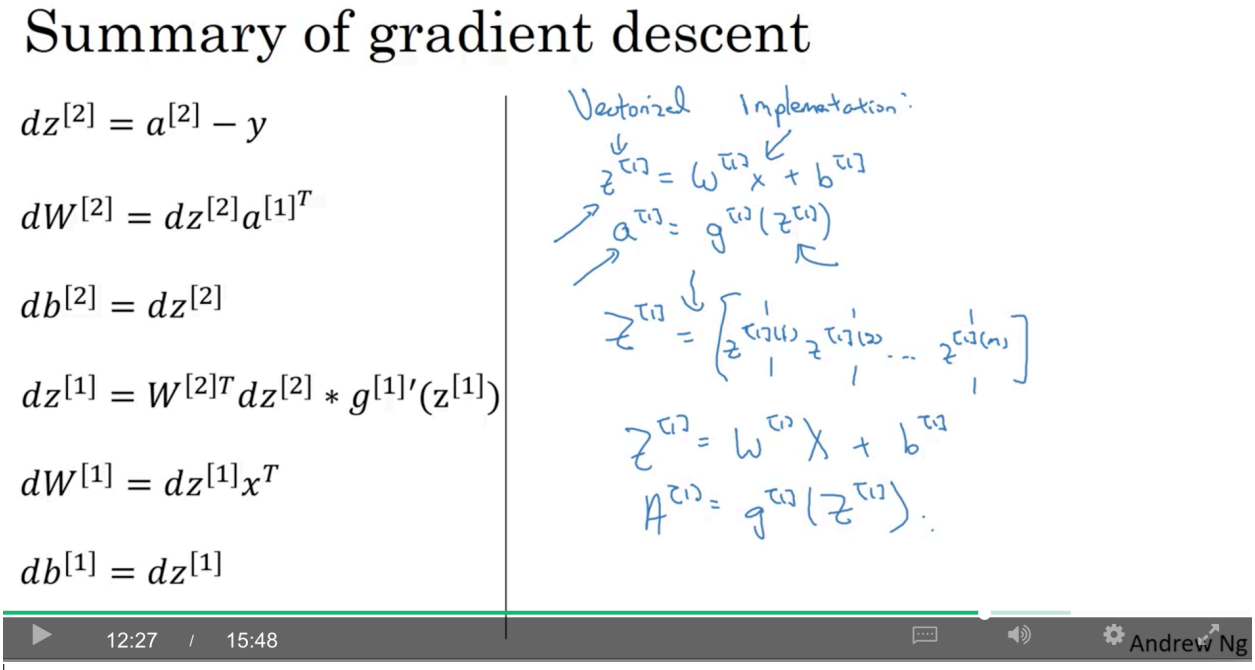
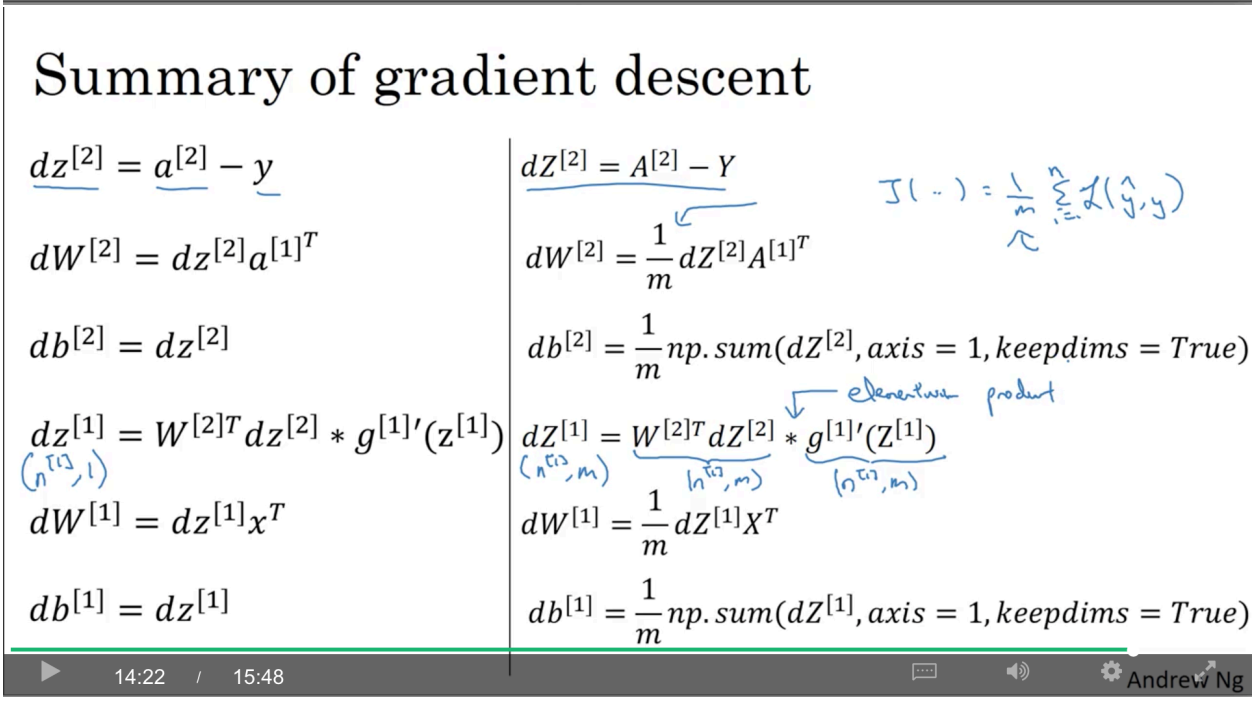


Figure 13: back-propagation-nn

然后看 nn：

扩展到 m 个 examples，并进行 vectorize：





问：为何有的有 1/m，有的没有。。。。。

1.11 1.11. 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."

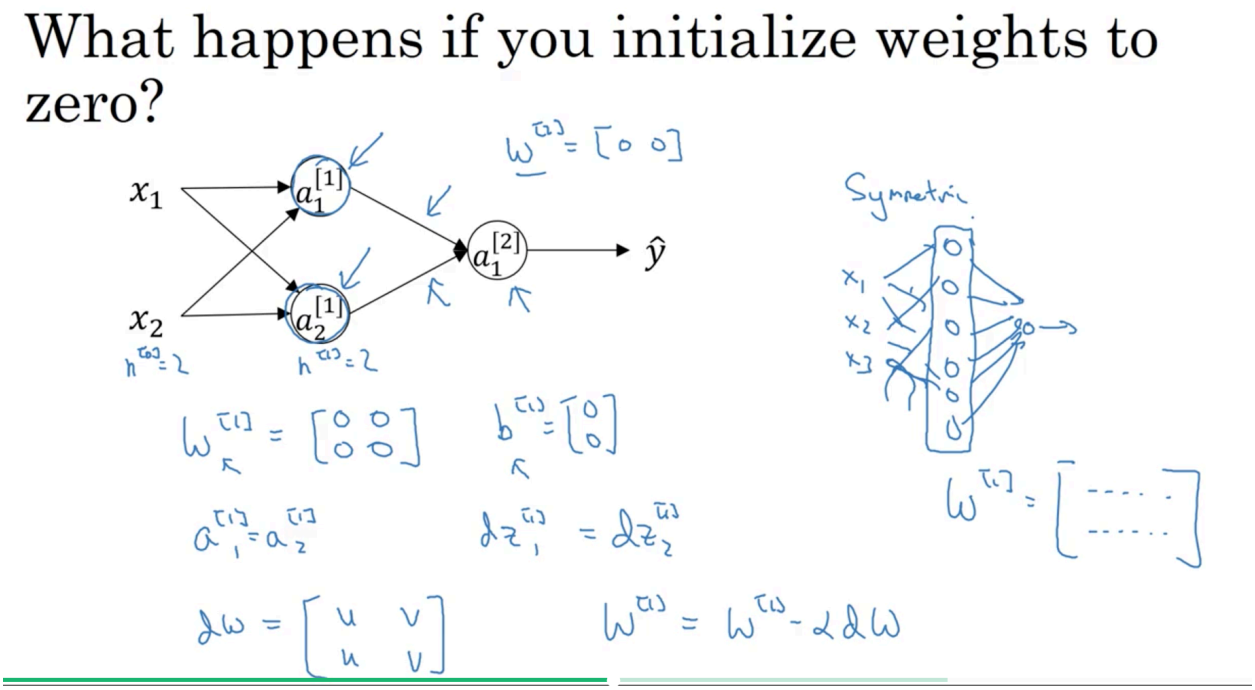


Figure 14: if-initialize-zero

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。。。**

Random initialization

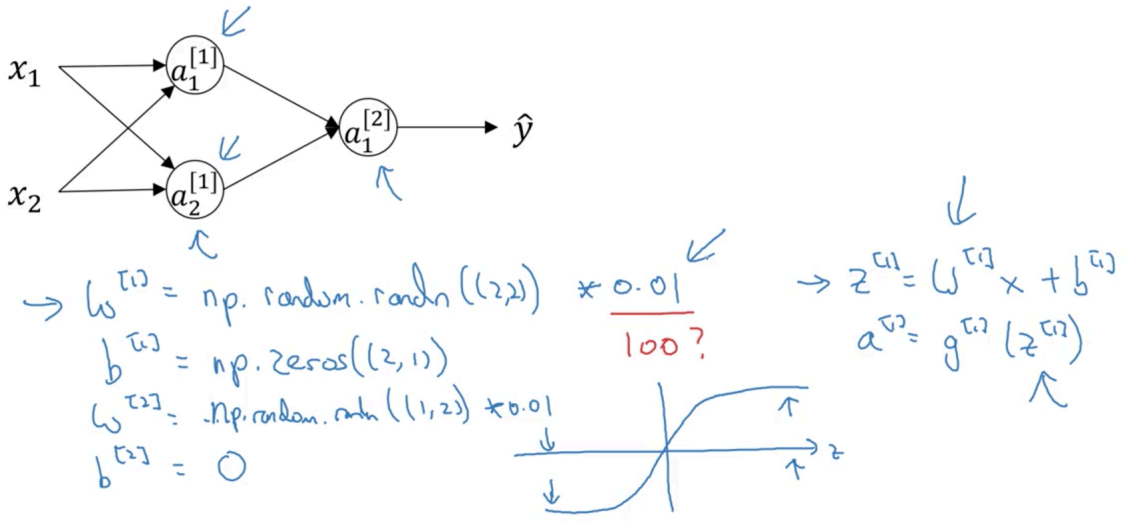


Figure 15: random-initialization