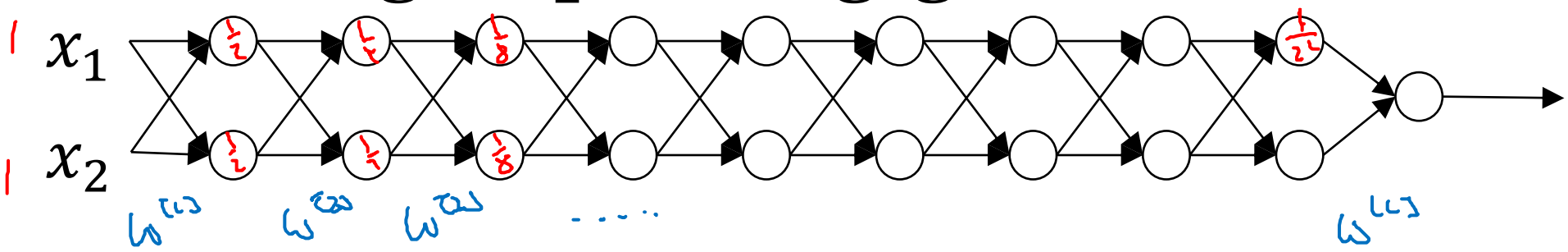
deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients



$L = 150$

$g(z) = z$ $\quad b^{[l]} = 0$

$\hat{y} = w^{[L]} w^{[L-1]} w^{[L-2]} \cdots w^{[3]} w^{[2]} w^{[1]} x$

$a^{[3]}$

1.5^L

0.5^L

$w^{[l]} > I$

$w^{[l]} < I \quad \begin{bmatrix} 0.9 & \\ & 0.9 \end{bmatrix}$

$z^{[1]} = w^{[1]} x$

$a^{[1]} = g(z^{[1]}) = z^{[1]}$

$a^{[2]} = g(z^{[2]}) = g(w^{[2]} a^{[1]})$

$w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ (0.5)

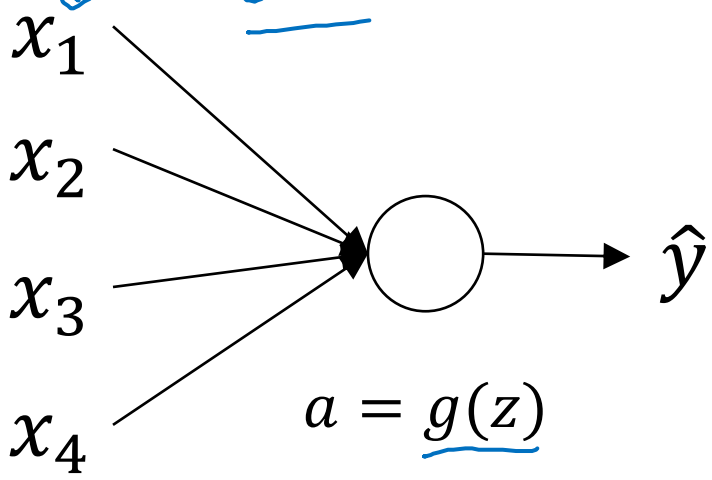
$\hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x$ (0.5, 0.5)

$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example



$w^{[l]}$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$ ~~+b~~

large n → smaller w_i

$Var(w_i) = \frac{1}{n} \frac{2}{n}$

$w^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

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

Other variants:

tanh

$\frac{1}{n^{[l-1]}}$

Xavier initialization

$\frac{2}{n^{[l-1]} + n^{[l]}}$

Andrew Ng

Single neuron example
在上一个视频中 你们看到了非常深的神经网络 会出现梯度消失和梯度爆炸问题 事实上 有一种针对此问题的部分解决方法 虽然不能完全解决它 但帮助很大 该方法就是更好 更细致地随机初始化你的神经网络 为了理解这个方法 我们从初始化单个神经元开始 然后再把它应用到深度网络中 现在让我们从单个神经元的例子说起 稍后再去考虑深度神经网络 单个的神经元可能会输入4个特征 从x1到x4 然后用a=g(z)激活 最后得到y 稍后在更深的网络中你会发现 这些输入是在一些a[l]层的右边 但现在我们先把它当成x 所以z就等于w1*x1+w2*x2+···+wn*xn 我们让b等于0 因此在这里忽略b项 因此 为了不让z项太大或者太小 你会注意到n项的数值越大 你就会希望Wi的值越小 对吗? 因为z是wi*xi的加和 因此 如果加和了很多这样的项 就会希望每一项就尽可能地小 一个合理的做法是 让变量wi等于1/n 这里的n是指输入一个神经元的特征数 在实践中 你可以使用np.random.randn函数 去设置某一层的权重矩阵W 还要把矩阵的形状参数输入到这里 然后乘以 (1/输入每个神经元的特征数)的平方根 然后乘以 (1/输入每个神经元的特征数)的平方根 在这里就是n[l−1] 因为这是输入每个神经元的输入单元的数量 事实上如果你使用ReLU激活函数 就不要在这里使用 1/n 而是把2/n当做变量 会工作地好一些 因此你会经常在初始化时看到这项 特别是使用ReLU激活函数时 此时g(z)等于ReLU(z) 当然这还取决于你对随机变量的熟悉程度 事实上 使用高斯随机变量并乘以这一项的平方根 事实上 使用高斯随机变量并乘以这一项的平方根 就和 2/n 这一项是相等的 我在这儿写n[l−1]而不是n的原因是 在这个例子中 逻辑回归中有n个输入特征 但是在更一般的例子中 每层中的每个单元都有n[l−1]个输入 所以 如果输入经过激活的特征平均值为0并且标准差为1 这就能使z项也呈现相同的分布性质 虽然这样不能完全解决问题 但它降低了梯度消失和梯度爆炸问题的程度 因为这种做法通过设置权重矩阵W 使得W不会比1大很多 也不会比1小很多 因此梯度不会过快地膨胀或者消失 我现在说一些其他的变体 这里我刚刚描述的是假设使用ReLU作为激活函数 这里我刚刚描述的是假设使用ReLU作为激活函数 另外还有一些变体 如果你使用tanh激活函数 那么在一篇论文中显示 与使用2做常数项不同的是 使用1作为常数项更好 因此用1而不是用2 然后计算它的平方根 这个平方根的项就代表了下面这一项 当使用tanh激活函数的时候 就应这样操作 这就是Xavier初始化方法 还有另外一个版本 是Yoshua Bengio和合作者发明的 你可能在一些论文中会见到 它是用这个公式 是有一些理论证明 它是用这个公式 是有一些理论证明 因此 我想说如果你用ReLU这种更为常见的激活函数 因此 我想说如果你用ReLU这种更为常见的激活函数 我会用这个公式 如果你用tanh激活函数 就尝试这个版本 另外一些也会用到这个 但是在实践中 我认为所有的这些公式只是给你一个出发点 它让权重矩阵的初始化变量有一个默认值 它让权重矩阵的初始化变量有一个默认值 如果你希望有这些变量 这些变量参数可以成为超参数的一部分 你可以通过调优确定使用哪个版本 因此你就有另外一个参数 与这一个公式相乘 调优这个乘数作为超参数中的一部分 有时候调优这个超参数会影响模型的规模 这通常不是我要最先调优的超参数之一 但是我经常在调优它的时候遇到一些问题 它会有一些帮助 但是相比于其他可以调优的参数 它的帮助通常会下降 我希望这节课能给一些关于梯度消失或梯度爆炸问题的直观感受 我希望这节课能给一些关于梯度消失或梯度爆炸问题的直观感受 以及如果使用适合的方法去初始化权重 希望这些能使你的权重不会过快地爆炸或者消失 因此训练一个合适的深度网络时 因此训练一个合适的深度网络时 权重或者梯度不会爆炸或消失太严重 但你训练深度网络时 这么做算是一个小技巧 可以帮助你更快地训练神经网络