3.9 神经网络的梯度下降（Gradient descent for neural networks）

在这个视频中，我会给你实现反向传播或者说梯度下降算法的方程组，在下一个视频我们会介绍为什么这几个特定的方程是针对你的神经网络实现梯度下降的正确方程。

你的单隐层神经网络会有$W^{[1]}$，$b^{[1]}$，$W^{[2]}$，$b^{[2]}$这些参数，还有个n_x表示输入特征的个数，$n^{[1]}$表示隐藏单元个数，$n^{[2]}$表示输出单元个数。

在我们的例子中，我们只介绍过的这种情况，那么参数:

矩阵$W^{[1]}$的维度就是$(n^{[1]}, n^{[0]})$，$b^{[1]}$就是$n^{[1]}$维向量，可以写成$(n^{[1]}, 1)$，就是一个的列向量。 矩阵$W^{[2]}$的维度就是$(n^{[2]}, n^{[1]})$，$b^{[2]}$的维度就是$(n^{[2]}, 1)$维度。

你还有一个神经网络的成本函数，假设你在做二分类任务，那么你的成本函数等于：

Cost function:

公式： $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

loss function 和之前做 **logistic** 回归完全一样。

训练参数需要做梯度下降，在训练神经网络的时候，随机初始化参数很重要，而不是初始化成全零。当你参数初始化成某些值后，每次梯度下降都会循环计算以下预测值：

$\hat{y}^{(i)}, (i = 1,2, \ldots, m)$

公式 3.28： $dW^{[1]} = \frac{dJ}{dW^{[1]}}, db^{[1]} = \frac{dJ}{db^{[1]}}$

公式 3.29： $dW^{[2]} = \frac{dJ}{dW^{[2]}}, db^{[2]} = \frac{dJ}{db^{[2]}}$

其中

公式 3.30： $W^{[1]} \Longrightarrow W^{[1]} - adW^{[1]}, b^{[1]} \Longrightarrow b^{[1]} - adb^{[1]}$

公式 3.31： $W^{[2]} \Longrightarrow W^{[2]} - adW^{[2]}, b^{[2]} \Longrightarrow b^{[2]} - adb^{[2]}$

正向传播方程如下（之前讲过）：

forward propagation：

(1) $z^{[1]} = W^{[1]}x + b^{[1]}$

(2) $a^{[1]} = \sigma(z^{[1]})$

(3) $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

(4) $a^{[2]} = g^{[2]}(z^{[z]}) = \sigma(z^{[2]})$

反向传播方程如下:

back propagation：

公式 3.32： $dz^{[2]} = A^{[2]} - Y, Y = [y^{[1]} \quad y^{[2]} \quad \cdots \quad y^{[m]}]$

公式 3.33： $dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$

公式 3.34： $db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis = 1, keepdims = True)$

公式 3.35：

$$dz^{[1]} = \underbrace{W^{[2]T} \mathrm{d}z^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]'}}_{activation\ function\ of\ hidden\ layer} * \underbrace{(z^{[1]})}_{(n^{[1]},m)}$$

公式 3.36： $dW^{[1]} = \frac{1}{m} dz^{[1]} x^T$

公式 3.37： $\underset{(n^{[1]},1)}{db^{[1]}} = \frac{1}{m} np.sum(dz^{[1]}, axis = 1, keepdims = True)$

上述是反向传播的步骤，注：这些都是针对所有样本进行过向量化，Y是$1 \times m$的矩阵；这里 `np.sum` 是 **python** 的 **numpy** 命令，`axis=1` 表示水平相加求和，`keepdims` 是防止 python 输出那些古怪的秩数$(n,)$，加上这个确保阵矩阵$db^{[2]}$这个向量输出的维度为$(n,1)$这样标准的形式。

目前为止，我们计算的都和 **Logistic** 回归十分相似，但当你开始计算反向传播时，你需要计算，是隐藏层函数的导数，输出在使用 **sigmoid** 函数进行二元分类。这里是进行逐个元素乘积，因为$W^{[2]T} dz^{[2]}$和$(z^{[1]})$这两个都为$(n^{[1]}, m)$矩阵；

还有一种防止 **python** 输出奇怪的秩数，需要显式地调用 `reshape` 把 `np.sum` 输出结果写成矩阵形式。

以上就是正向传播的 4 个方程和反向传播的 6 个方程，这里我是直接给出的，在下个视频中，我会讲如何导出反向传播的这 6 个式子的。如果你要实现这些算法，你必须正确执行正向和反向传播运算，你必须能计算所有需要的导数，用梯度下降来学习神经网络的参数；你也可以许多成功的深度学习从业者一样直接实现这个算法，不去了解其中的知识。