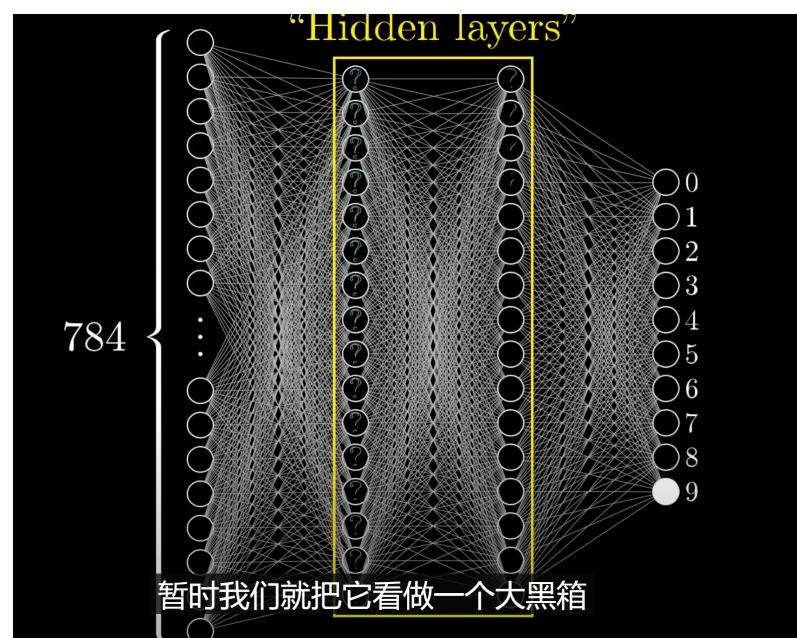
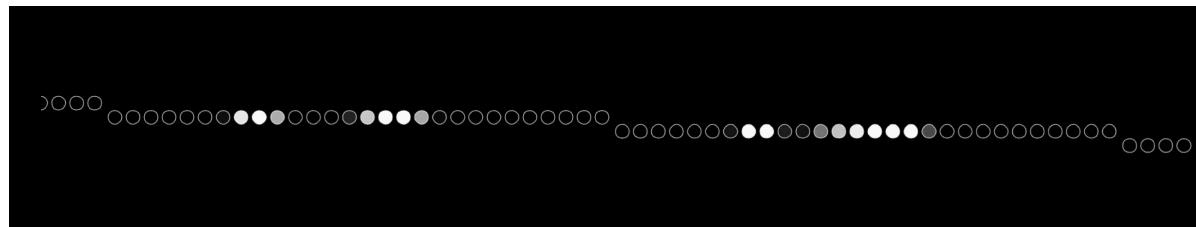
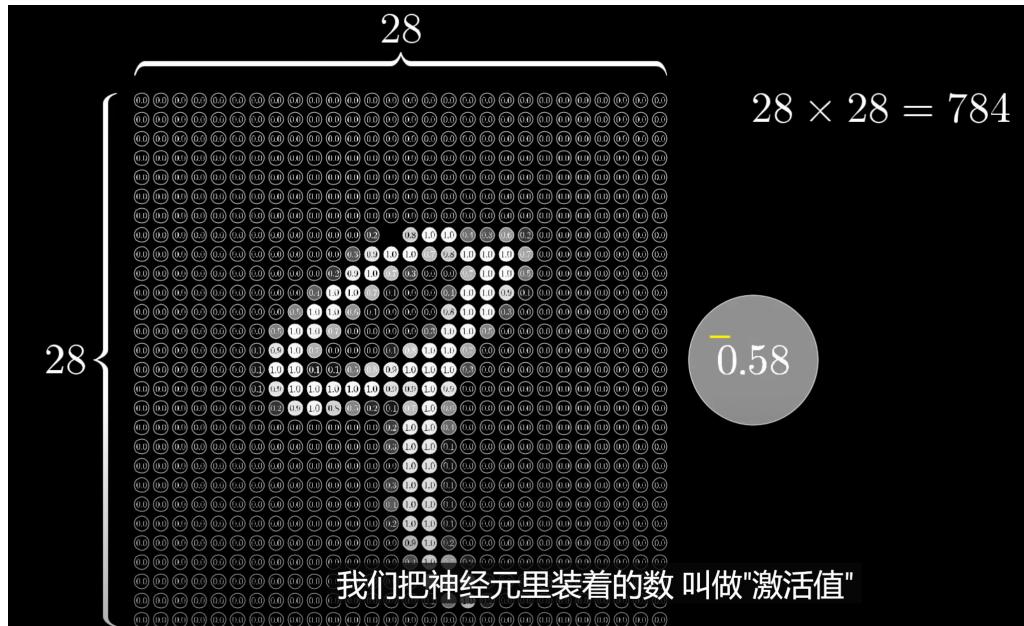


神经网络

1 什么是神经网络

首先一个图像把他分割成小单元，每个单元就是神经元，单元里的数字是‘激活值’

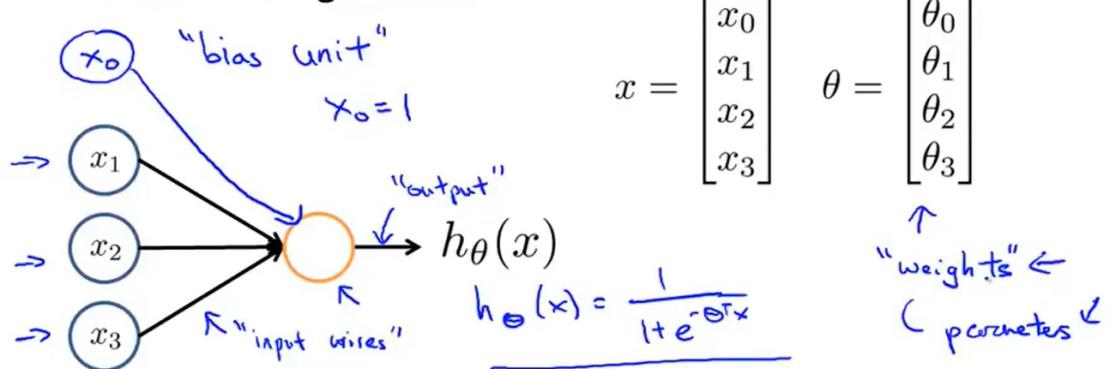


2 神经元模型

神经网络最基本成分是神经元网络

- 单个神经元

Neuron model: Logistic unit

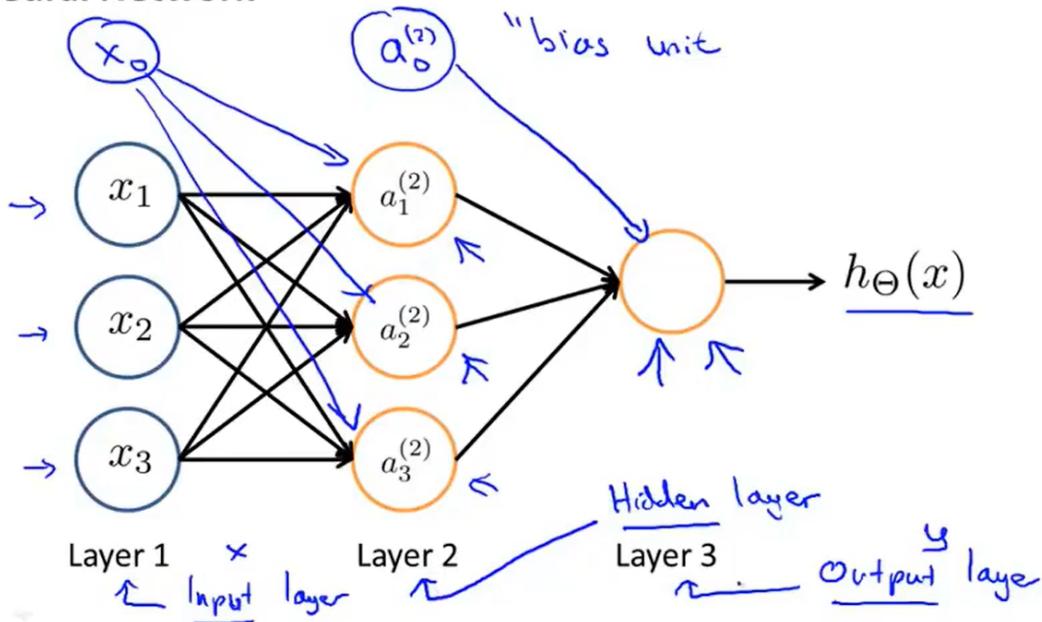


Sigmoid (logistic) activation function.

$$g(z) = \frac{1}{1 + e^{-z}}$$

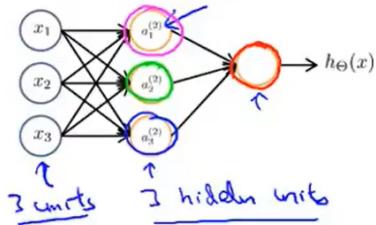
- 感知机与多层网络

Neural Network



从数学上定义神经网络的假设

Neural Network

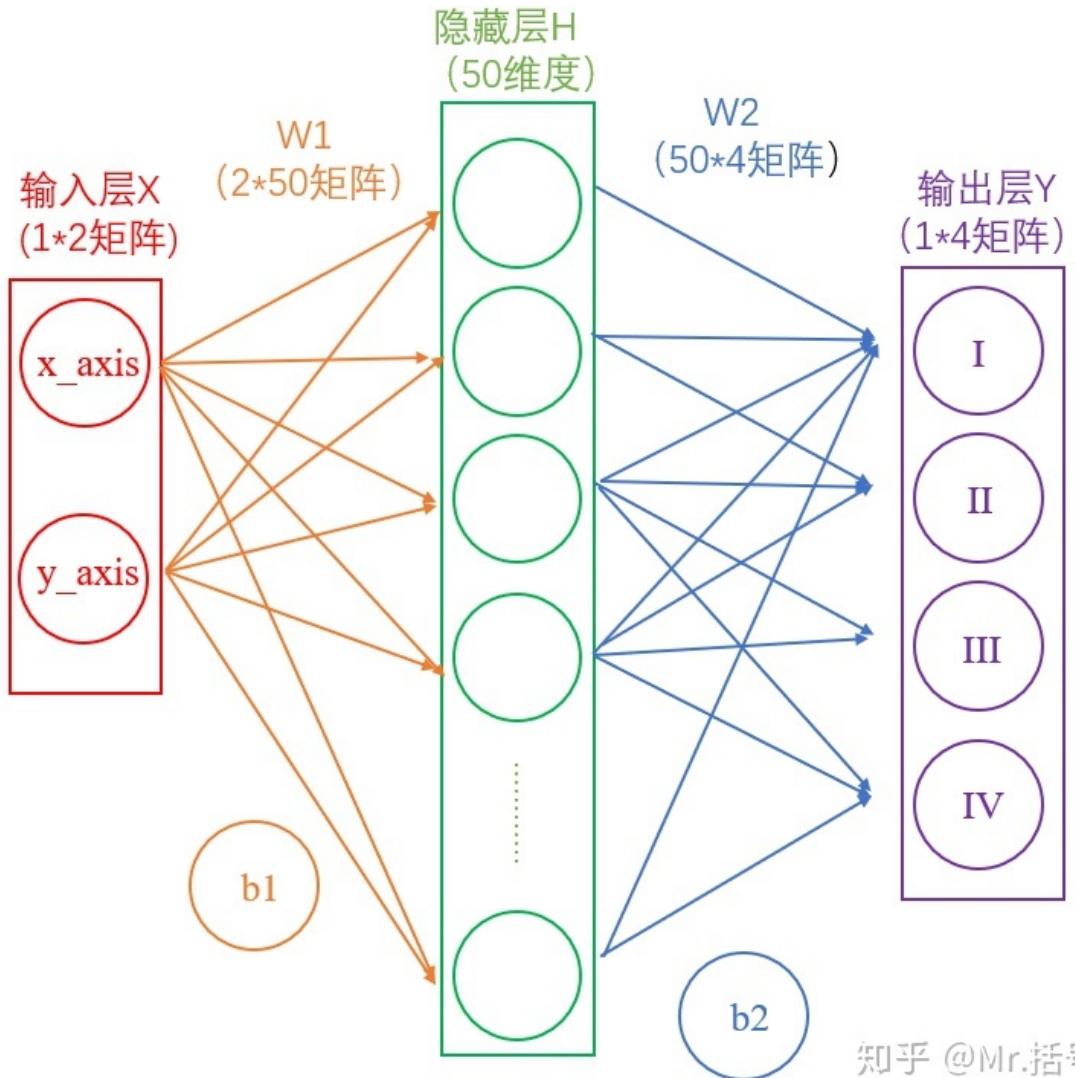


$\rightarrow a_i^{(j)}$ = “activation” of unit i in layer j
 $\rightarrow \Theta^{(j)}$ = matrix of weights controlling
 function mapping from layer j to
 layer $j + 1$

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$$

$$\begin{aligned}
 \rightarrow a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\
 \rightarrow a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\
 \rightarrow a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \\
 \rightarrow h_\Theta(x) &= a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})
 \end{aligned}$$

\rightarrow If network has s_j units in layer j , s_{j+1} units in layer $j + 1$, then $\Theta^{(j)}$
 will be of dimension $s_{j+1} \times (s_j + 1)$. \downarrow



知乎 @Mr.括号

- 推导

(一) 从输入层到隐藏层

$$H = X * W_1 + b_1$$

(二) 从隐藏层到输出层

$$Y = H * W_2 + b_2$$

(三) 对网络注入灵魂: 激活层

1. 阶跃函数: 当输入小于等于0时, 输出0; 当输入大于0时, 输出1。

2. Sigmoid: 当输入趋近于正无穷 / 负无穷时, 输出无限接近于1/0。

3. ReLU: 当输入小于0时, 输出0; 当输入大于0时, 输出等于输入。

(四) 输出正则化

$$S_i = \frac{e^i}{\sum_j e^j}$$

简单来说分三步进行: (1) 以e为底对所有元素求指数幂; (2) 将所有指数幂求和; (3) 分别将这些指数幂与该和做商。

这样求出的结果中, 所有元素的和一定为1, 而每个元素可以代表概率值。

我们将使用这个计算公式做输出结果正规化处理的层叫做“Softmax”层.

- 再衡量输出的好坏

- Softmax输出的结果是(90%,5%,3%,2%), 真实的结果是(100%,0,0,0)。虽然输出的结果可以正确分类, 但是与真实结果之间是有差距的, 一个优秀的网络对结果的预测要无限接近于100%, 为此, 我们需要将Softmax输出结果的好坏程度做一个“量化”

- 求对数的负数

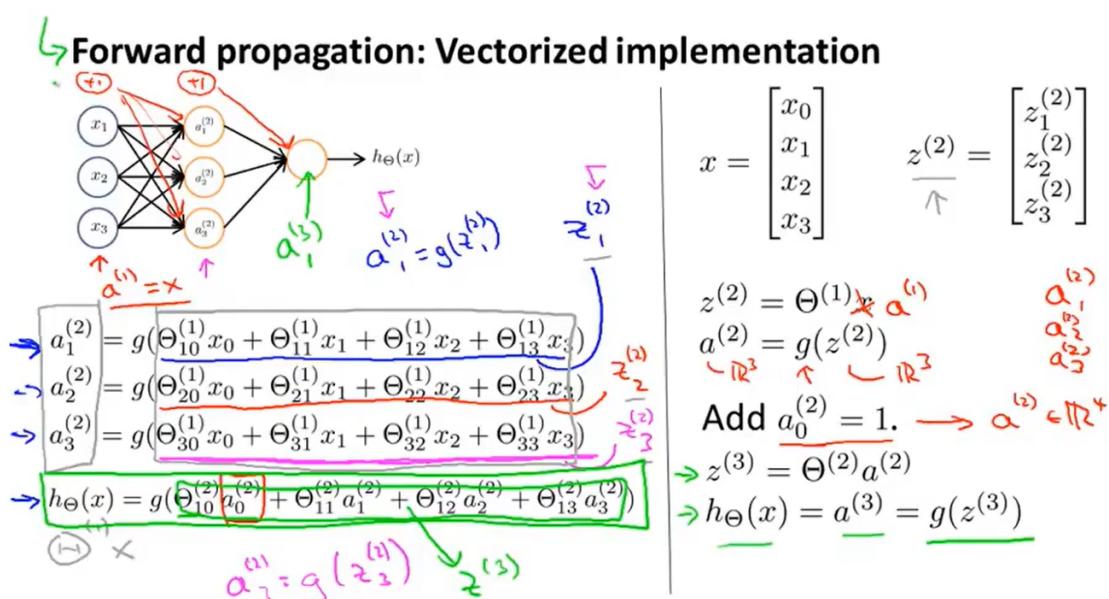
- 还是用90%举例, 对数的负数就是: $-\log 0.9 = 0.046$

可以想见, 概率越接近100%, 该计算结果值越接近于0, 说明结果越准确, 该输出叫做“交叉熵损失 (Cross Entropy Error) ”。

我们训练神经网络的目的, 就是尽可能地减少这个“交叉熵损失”。

- 前向传播

从输入单元的激活项开始, 然后进行前向传播给隐藏层, 计算隐藏层的激活项, 然后继续前向传播, 并计算输出层的激活项。

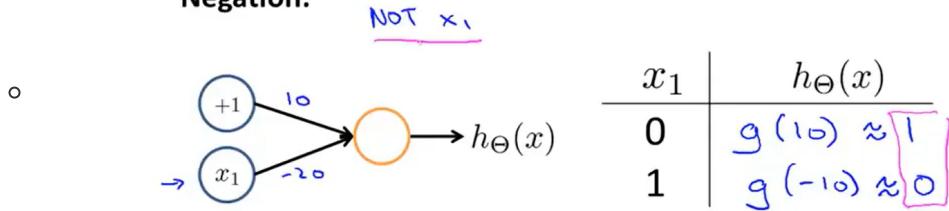


- 神经网络使用的不是原始特征来进行逻辑回归, 而是使用新学习的来作为新的特征进行逻辑回归训练
- 神经网络可以用来学习复杂的非线性假设模型

- 单层神经元（无中间层）的计算可用来表示逻辑运算

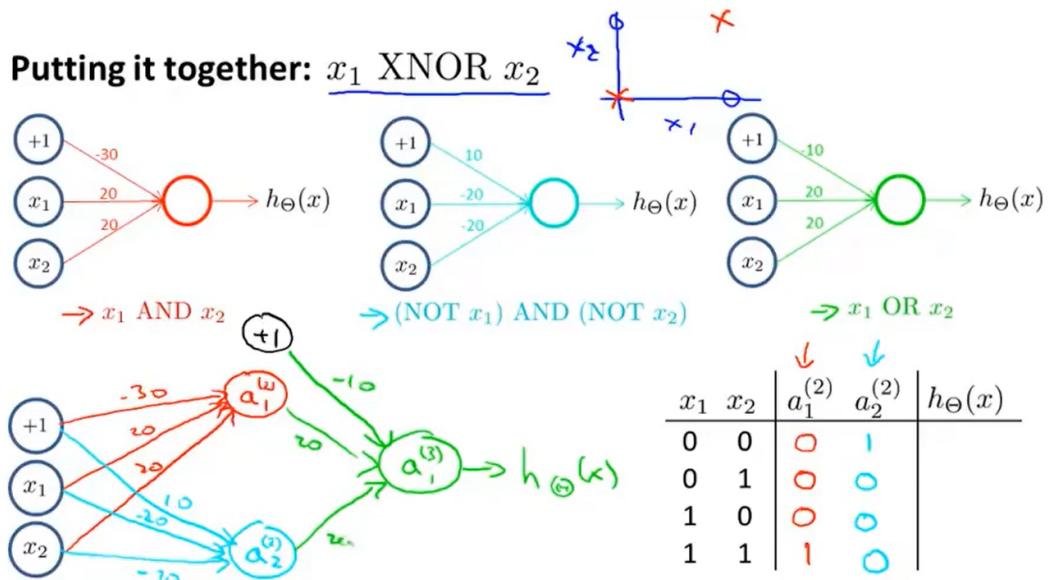
$$\rightarrow x_1 \text{ AND } x_2 \quad \rightarrow x_1 \text{ OR } x_2 \quad \{0, 1\}$$

Negation:



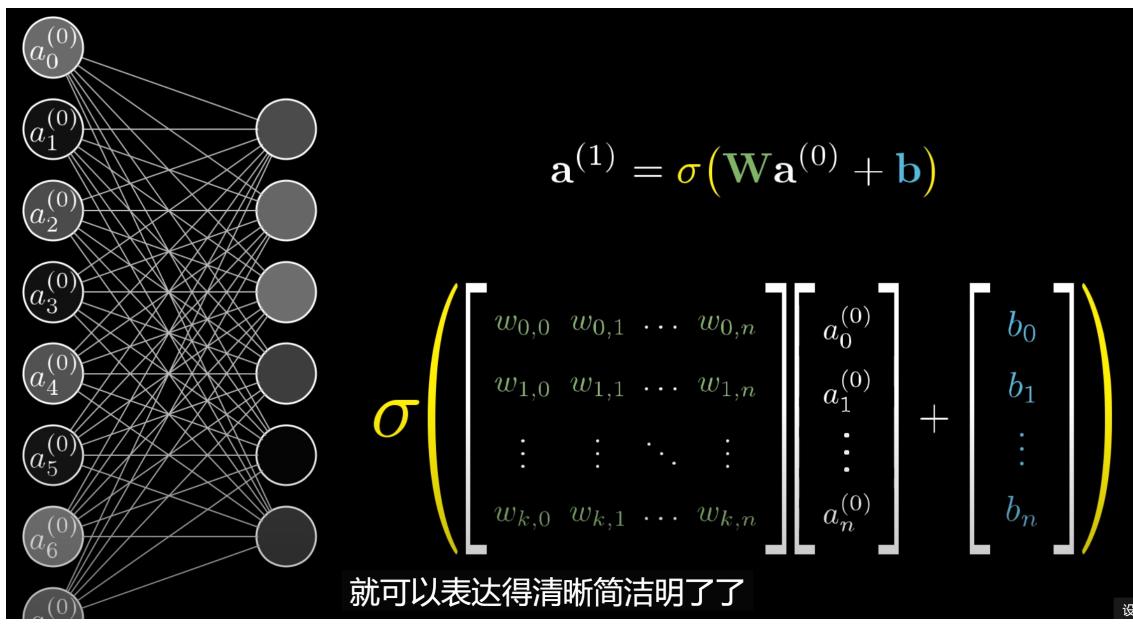
$$h_\Theta(x) = g(10 - 20x_1)$$

- 利用神经元来组合成更为复杂的神经网络以实现更为复杂的运算



代价函数

- 神经网络的定义分为两种：
 - 二分类: $S = 0, y = 0 \text{ or } 1$ 表示哪一类
 - k 分类: $S = 0, y_{-i} = 1$ 表示分到第 i 类
- 由于神经网络是多个输出变量，以及多个因变量，所以 $h_\Theta(x)$ 是一个一维度为 K 的向量，训练集中的因变量也是同样维度的一个向量。目的是希望通过代价函数来观察算法预测的结果与真实误差有多大，对于每一行的特征，都会给出 K 个预测，可以利用循环，对每一行特征都预测 K 个不同的结果，然后再利用循环在 K 个预测中选择可能性最高的一个，将其与 y 中的实际数据进行比较。
-



- Sigmoid

$$a_0^{(1)} = \sigma(w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \dots + w_{0,n} a_n^{(0)} + b_0)$$

↑
Bias

```

• class Network(object):
    def __init__(self, *args, **kwargs):
        pass

    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a
    
```

- 梯度下降

$$\vec{\mathbf{W}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$$

$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \end{bmatrix} \quad \begin{array}{l} w_0 \text{ should increase} \\ w_1 \text{ should increase} \\ w_2 \text{ should decrease} \\ \vdots \\ w_{13,000} \text{ should increase} \\ w_{13,001} \text{ should decrease} \end{array}$$

所以 你再看到这个复杂到爆炸的代价函数它的梯度向量时

反向传播函数

- 计算神经网络预测结果的时候，采用了一种正向传播，从第一层到最后一层。
- 现在计算代价函数的偏导数，需要采用反向传播算法：首先计算最后一层的误差，然后再一层一层反向求出各层的误差，直到倒数第二层
- 反向传播算法是指单个训练样本想怎样修改权重与偏置，不仅是说每个参数应该变大还是变小，还包括了这些变化的比例是多大，才能最快的降低代价

梯度检验

- 做梯度下降的时候，虽然代价不断在减小，但是最终的结果可能并不是最优解。
- 梯度的数值检验：通过估计梯度值来检验计算的导数值是否是我们要求的
 - 采用的方法：在代价函数上沿着切线的方向选择两个非常近的点然后计算两个点的平均值用以估计梯度。即对于某个 θ 计算出以 ϵ 为间隔之间的代价值，然后求两个代价的平均，用以估计在 θ 处的代价值

随机初始化

神经网络的随机初始化通常为正负 ϵ 之间的随机值

小结

- 网络结构：决定选择多少层以及决定每层分别有多少个单元
- 第一层的单元数是训练集的特征数量
- 最后一层单元数是训练集的结果的类的数量
- 如果隐藏层数大于1，确保每个隐藏层的单元个数相同，通常情况下隐藏层单元的个数越多越好
- 真正要决定的是隐藏层的层数和每个中间层的单元数

训练神经网络

- 参数的随机初始化
- 利用正向传播方法计算所有的 $h_{\theta}(x)$
- 编写计算代价函数的代码
- 利用反向传播的方法计算所有的偏导数
- 利用数值检验方法检验这些偏导数
- 利用优化算法来最小化代价函数