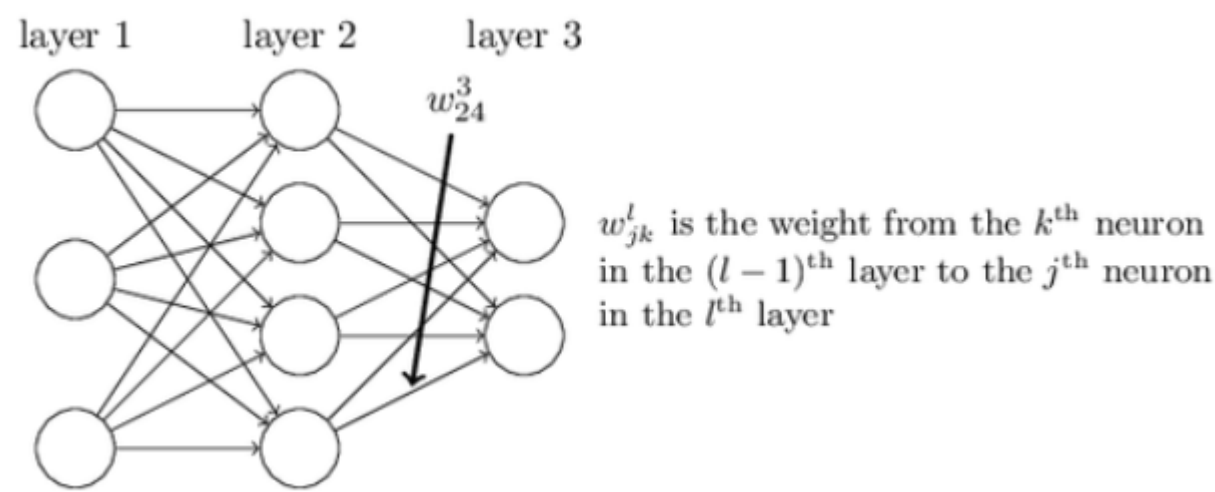
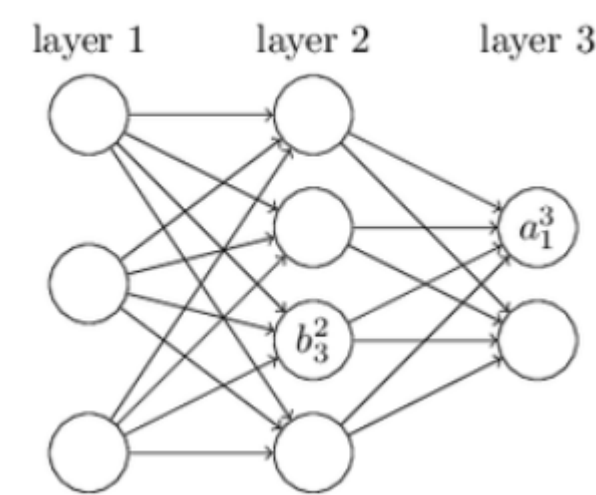


backpropagation算法原理

Backpropagation核心解决的问题: $\partial C/\partial w$ 和 $\partial C/\partial b$ 的计算, 针对cost函数C



ω_{24}^3 :表示第从第(3-1)层的第4个神经元到第3层的第2个神经元的权重weight



b_3^2 :表示第2层的第3个神经元的偏向bais

正向传播

公式:

$$\alpha_j^l = \sigma(\sum_k \omega_{jk}^l \alpha_k^{l-1} + b_j^l)$$

- 分为两步:
1. $\omega \alpha + b$ 加权求和
 2. 对加权求和整体使用sigmoid函数求出下层的输出值

数据结构

对于每一层(l), 定义一个权重矩阵 (weight matrix): ω^l , 这个权重矩阵包含当前层的所有神经元到前一层的
所有神经元的权重

ω_{jk}^l 表示第 l 层的第 j 个神经元到第 $l-1$ 层的第 k 个神经元的权重 weight

对于每一层(l), 定义一个偏向向量(bias vector): b^l

b_j^l 则表示 l 层的第 k 个神经元的 bias

同理, 对于 α : l 层的神经元向量 α^l , 每个神经元的值 α_j^l

Vector a function: $\sigma(v)_j = \sigma(v_j)$ 例如: $f(x) = x^2 f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} f(2) \\ f(3) \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \end{bmatrix}$ 则, 可以由 l 层的每一个元素的计算公式可以退出该层矩阵运算的公式:

$$\alpha_j^l = \sigma\left(\sum_k \omega_{jk}^l \alpha_k^{l-1} + b_j^l\right) \Rightarrow \alpha^l = \sigma(\omega^l \alpha^{l-1} + b^l)$$

其中 $\sum_k \omega_{jk}^l \alpha_k^{l-1}$ 可以用 $\omega_j^l \alpha^{l-1}$ 表示, 即第 l 层权重矩阵的第 j 个行向量与上一层神经元的列向量进行内积的结果, 令 $z^l = \omega^l \alpha^{l-1} + b^l$, 则 $\alpha^l = \sigma(z^l)$ 。简单点理解: 对于每层正向传播, 每层的值(向量形式)就是当前层的权重矩阵与上一层的值(向量)相乘再加上当前层的偏向(向量), 然后统一使用 sigmoid 函数转化

关于Cost函数的两个假设:

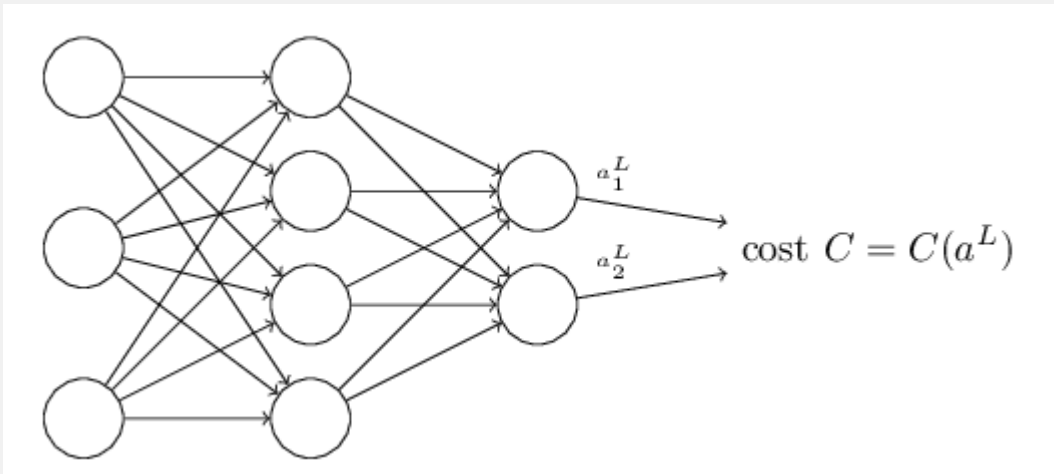
二次Cost函数

$$C = \frac{1}{2n} \sum_x \|y(x) - \alpha^L(x)\|^2$$

其中 $\alpha^L(x)$ 表示输出层的真实值所组成的向量, x 表示训练实例, n 表示输入实例的个数。

1. 第一个假设是成本函数可以写成单个训练样例 x 的成本函数 C_x 的平均值 $C = \frac{1}{n} \sum_x C_x$ 。二次成本函数就是这种情况, 单个训练样例的 Cost 函数为 $C_x = \frac{1}{2} \|y - a^L\|^2$

2. 第二个假设是它可以写成神经网络输出的函数：



二次Cost

函数满足这个要求，因为单个训练样例x的二次Cost可写为

$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2$$

介绍一个后面需要用到的公式

The Hadamard product, \odot , 向量对应元素相乘: $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$

backpropagation4个重要公式

Summary: the equations of backpropagation

$\delta^L = \nabla_a C \odot \sigma'(z^L)$ (BP1)

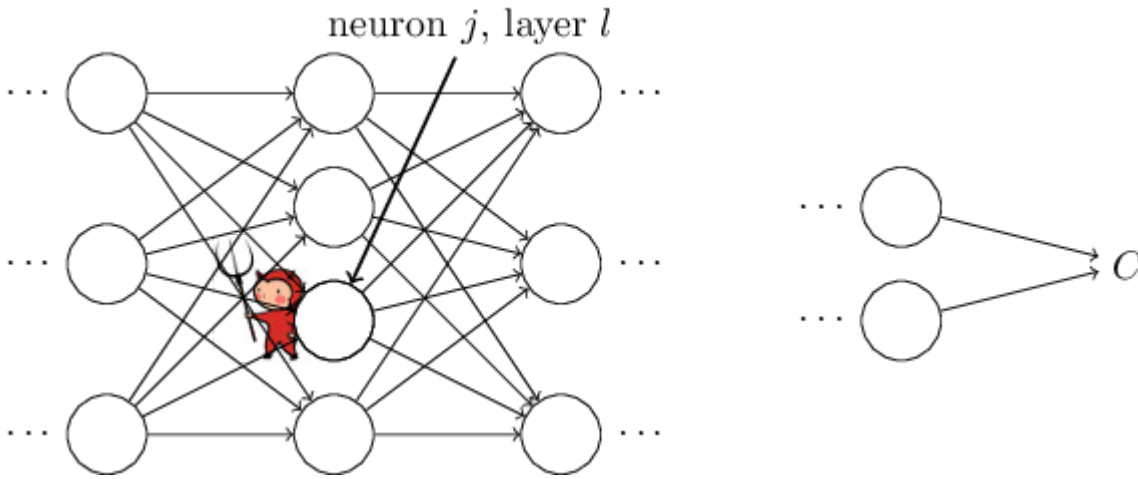
$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ (BP2)

$\frac{\partial C}{\partial b_j^l} = \delta_j^l$ (BP3)

$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ (BP4)

反向传播是关于如何改变网络中的权重和偏差来改变成本函数，这意味着需要计算偏导数 $\partial C / \partial w_{jk}^l$ 和 $\partial C / \partial b_j^l$ 。但为了计算这些，我们首先引入一个中间量， δ_j^l ，我们称之为在l层的第j个神经元的error。反向传播计算每一层的 δ_j^l ，然后将 δ_j^l 与 $\partial C / \partial w_{jk}^l$ 和 $\partial C / \partial b_j^l$ 关联起来。

为了理解错误是如何定义的，想象我们的神经网络中存在一个恶魔：



恶魔对第 l 层的第 j 个神经元添加一个变化量 Δz_j^l ,该神经元输出就变成 $\sigma(z_j^l + \Delta z_j^l)$ 。这种变化通过网络中的后续层传播,最终导致整体Cost的变化 $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$ (简单的高数知识)。如果这个恶魔是一个好人,它想要帮我们优化Cost,他会尝试一个更小的 Δz_j^l 使得损失函数更小。假设 $\frac{\partial C}{\partial z_j^l}$ 是一个很大的值(不管正负)。然后恶魔通过选择与 $\frac{\partial C}{\partial z_j^l}$ 有相反的符号的 Δz_j^l 来降低Cost。相反,如果 $\frac{\partial C}{\partial z_j^l}$ 接近于零,那么恶魔通过干扰加权输入 z_j^l 就几乎不能改变Cost,此时,这个神经元已经非常接近最优(再如何优化也不能改变Cost)。所以这里把 $\frac{\partial C}{\partial z_j^l}$ 定义为神经元error的度量。于是定义在 l 层的第 j 个神经元的error: δ_l^j

BP1

我们定义在输出层(L)的第 j 个神经元的error的方程为:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \quad (BP1)$$

解释:其中 $\frac{\partial C}{\partial a_j^L}$ 这部分衡量Cost相对于第 j 个神经元activation的输出的变化率, $\sigma'(z_j^L)$ 这部分衡量activation方程相对于中间变量 z_j^L 的变化率

转化为矩阵的表达形式

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

可以认为 $\nabla_a C$ 表示C相对于输出activation的变化率,根据上面定义的2次Cost方程,输出层的error可以写成:

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

BP2

因为下一层的error的变化会引起当前层error的变化,当前层(l 层)的error变化方程为(error传递公式):

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad (BP2)$$

其中 $(w^{l+1})^T$ 是第 $(l+1)^{\text{th}}$ 层的权重矩阵 w^{l+1} 的转置。第 $(l+1)^{\text{th}}$ 层处的error δ^{l+1} 乘以 $(l+1)^{\text{th}}$ 权重矩阵的转置 $(w^{l+1})^T$ 时，我们可以直观地认为网络向前传递error。然后 $\odot \sigma'(z^l)$ ，可以算出前一层的error。交替使用可以算出神经网络的所有层的error

BP3

Cost对偏向求偏导：

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l (BP3)$$

写成向量形式：

$$\frac{\partial C}{\partial b} = \delta$$

BP4

Cost对权重求偏导：

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

(BP4) 写成矩阵形式：

$$\frac{\partial C}{\partial w} = a_{\text{in}} \delta_{\text{out}}$$

[公式证明参考]

([http://neuralnetworksanddeeplearning.com/chap2.html#proof_of_the_four_fundamental_equations_\(optional\)](http://neuralnetworksanddeeplearning.com/chap2.html#proof_of_the_four_fundamental_equations_(optional)))