深度学习之 c++ 实现 Back Propagation

本轮的主要内容是教会初学者如何用 c++来实现深度学习中至关重要的一个环节----反向传播算法的实现　　(部分内容参考：http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm)。

主要内容：

　　介绍反向传播算法的作用

　　反向传播算法实现步骤

　　c++实现的一个例子

反向传播算法的作用

假设我们有一个固定的训练集$\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\ldots,(x^{(m)},y^{(m)})\}$，包含 m 个训练样本。我们可以使用批量梯度下降法来训练我们的网络。

对于单个样本，我们可以将其训练误差定义为：

$$J(W,b;x,y)=\frac{1}{2}\left\|h_{W,b}(x)-y\right\|^2$$

对于整个训练集，其误差为：

$$J(W,b)=\left[\frac{1}{m}\sum_{i=1}^{m}J(W,b;x^{(i)},y^{(i)})\right]+\frac{\lambda}{2}\sum_{l=1}^{n_l}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

这里，后面一项是正则化项，λ是权重衰减项。

首先需要明确的是：**我们的目的是最小化误差函数$\boldsymbol{J(W,b)}$**。J是 w,b 的函数。训练一个伸进网络，我们会选择某种方法(可以参考《如何优雅地训练神经网络》)来初始化我们的权重。

根据梯度下降法的思想，沿着梯度的反方向是误差函数下降最快的方向。因此，可以得出更新系数的公式：

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

这里，α是学习速率，可以理解为一个常数先不作讨论，特别地我们会发现$J(W,b)$关于 w,b 的导数显得格外的重要。**因为只要有了这两个导数，系数更新就易如反掌了。那么这个导数如何求呢？反向传播算法就是来解决这个问题的。**

反向传播算法的主要步骤

反向传播算法的主要步骤如下：

1、 进行一次前向传播(feedforward pass)，计算出各层的输出激活值$l_2, l_2, \dots, l_{n_l}$

2、 对于网络的输出层n_l的每个神经元，计算：

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -\left(y_i - a_i^{n_l}\right) \cdot \mathrm{f}'\left(z_i^{(n_l)}\right)$$

3、 对于$l = n_l - 1, n_l - 2, \dots, 3, 2$层：

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) \mathrm{f}'\left(z_i^{(l)}\right)$$

注意下标的顺序(可以结合“反向”二字理解下标)

4、 计算偏导数

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}$$

以上便是反向传播算法的主要步骤。

C++实现的一个例子见：https://github.com/xupeng082008/DeepLearning-Backpropagation