反向传导算法

From Ufldl

假设我们有一个固定样本集 $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$，它包含 m 个样例。我们可以用批量梯度下降法来求解神经网络。具体来讲，对于单个样例 (x, y)，其代价函数为：

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

这是一个（二分之一的）方差代价函数。给定一个包含 m 个样例的数据集，我们可以定义整体代价函数为：

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

以上关于$J(W,b)$定义中的第一项是一个均方差项。第二项是一个规则化项（也叫权重衰减项），其目的是减小权重的幅度，防止过度拟合。

[注：通常权重衰减的计算并不使用偏置项 $b_i^{(l)}$，比如我们在 $J(W,b)$ 的定义中就没有使用。一般来说，将偏置项包含在权重衰减项中只会对最终的神经网络产生很小的影响。如果你在斯坦福选修过CS229（机器学习）课程，或者在YouTube上看过课程视频，你会发现这个权重衰减实际上是课上提到的贝叶斯规则化方法的变种。在贝叶斯规则化方法中，我们将高斯先验概率引入到参数中计算MAP（极大后验）估计（而不是极大似然估计）。]

权重衰减参数 λ 用于控制公式中两项的相对重要性。在此重申一下这两个复杂函数的含义：$J(W,b;x,y)$ 是针对单个样例计算得到的方差代价函数；$J(W,b)$ 是整体样本代价函数，它包含权重衰减项。

以上的代价函数经常被用于分类和回归问题。在分类问题中，我们用 $y = 0$ 或 1，来代表两种类型的标签（回想一下，这是因为 sigmoid激活函数的值域为 $[0,1]$；如果我们使用双曲正切型激活函数，那么应该选用 -1 和 $+1$ 作为标签）。对于回归问题，我们首先要变换输出值域（译者注：也就是 y），以保证其范围为 $[0,1]$ （同样地，如果我们使用双曲正切型激活函数，要使输出值域为 $[-1,1]$）。

我们的目标是针对参数 W 和 b 来求其函数 $J(W,b)$ 的最小值。为了求解神经网络，我们需要将每一个参数 $W_{ij}^{(l)}$ 和 $b_i^{(l)}$ 初始化为一个很小的、接近零的随机值（比如说，使用正态分布 $Normal(0,\epsilon^2)$ 生成的随机值，其中 ϵ 设置为 0.01 ），之后对目标函数使用诸如批量梯度下降法的最优化算法。因为 $J(W,b)$ 是一个非凸函数，梯度下降法很可能会收敛到局部最优解；但是在实际应用中，梯度下降法通常能得到令人满意的结果。最后，需要再次强调的是，要将参数进行随机初始化，而不是全部置为 0。如果所有参数都用相同的值作为初始值，那么所有隐藏层单元最终会得到与输入值有关的、相同的函数（也就是说，对于所有 i，$W_{ij}^{(1)}$都会取相同的值，那么对于任何输入 x 都会有：$a_1^{(2)} = a_2^{(2)} = a_3^{(2)} = \ldots$ ）。随机初始化的目的是使对称失效。

梯度下降法中每一次迭代都按照如下公式对参数 W 和b 进行更新：

$$\begin{aligned} W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b) \end{aligned}$$

其中 α 是学习速率。其中关键步骤是计算偏导数。我们现在来讲一下反向传播算法，它是计算偏导数的一种有效方

法。

我们首先来讲一下如何使用反向传播算法来计算 $\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y)$ 和 $\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y)$，这两项是单个样例 (x,y) 的代价函数 $J(W,b;x,y)$ 的偏导数。一旦我们求出该偏导数，就可以推导出整体代价函数 $J(W,b)$ 的偏导数：

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

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

以上两行公式稍有不同，第一行比第二行多出一项，是因为权重衰减是作用于 W 而不是 b。

反向传播算法的思路如下：给定一个样例 (x,y)，我们首先进行“前向传导”运算，计算出网络中所有的激活值，包括 $h_{W,b}(x)$ 的输出值。之后，针对第 l 层的每一个节点 i，我们计算出其“残差” $\delta_i^{(l)}$，该残差表明了该节点对最终输出值的残差产生了多少影响。对于最终的输出节点，我们可以直接算出网络产生的激活值与实际值之间的差距，我们将这个差距定义为 $\delta_i^{(n_l)}$ （第 n_l 层表示输出层）。对于隐藏单元我们如何处理呢？我们将基于节点（译者注：第 $l+1$ 层节点）残差的加权平均值计算 $\delta_i^{(l)}$，这些节点以 $a_i^{(l)}$ 作为输入。下面将给出反向传导算法的细节：

1. 进行前馈传导计算，利用前向传导公式，得到 $L_2, L_3, \ldots$ 直到输出层 L_{n_l} 的激活值。
2. 对于第 n_l 层（输出层）的每个输出单元 i，我们根据以下公式计算残差：

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

[译者注：

$$\begin{aligned}\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{n_l}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 \\ &= \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2 = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - f(z_j^{(n_l)}))^2 \\ &= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})\end{aligned}$$

]

3. 对 $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ 的各个层，第 l 层的第 i 个节点的残差计算方法如下：

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

{译者注：

$$\begin{aligned}
\delta_i^{(n_l-1)} &= \frac{\partial}{\partial z_i^{n_l-1}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2 \\
&= \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - a_j^{(n_l)})^2 = \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - f(z_j^{(n_l)}))^2 \\
&= \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} f(z_j^{(n_l)}) = \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot f'(z_j^{(n_l)}) \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{(n_l-1)}} \\
&= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{n_l-1}} = \sum_{j=1}^{S_{n_l}} \left(\delta_j^{(n_l)} \cdot \frac{\partial}{\partial z_i^{n_l-1}} \sum_{k=1}^{S_{n_l-1}} f(z_k^{n_l-1}) \cdot W_{jk}^{n_l-1} \right) \\
&= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot W_{ji}^{n_l-1} \cdot f'(z_i^{n_l-1}) = \left(\sum_{j=1}^{S_{n_l}} W_{ji}^{n_l-1} \delta_j^{(n_l)} \right) f'(z_i^{n_l-1})
\end{aligned}$$

将上式中的$n_l - 1$与n_l的关系替换为l与$l+1$的关系，就可以得到：

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

以上逐次从后向前求导的过程即为“反向传导”的本意所在。]

4. 计算我们需要的偏导数，计算方法如下：

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

最后，我们用矩阵-向量表示法重写以上算法。我们使用“$\bullet$” 表示向量乘积运算符（在Matlab或Octave里用“.*”表示，也称作阿达马乘积）。若 $a = b \bullet c$，则 $a_i = b_i c_i$。在上一个教程中我们扩展了 $f(\cdot)$ 的定义，使其包含向量运算，这里我们也对偏导数 $f'(\cdot)$ 也做了同样的处理（于是又有 $f'([z_1, z_2, z_3]) = [f'(z_1), f'(z_2), f'(z_3)]$ ）。

那么，反向传播算法可表示为以下几个步骤：

1. 进行前馈传导计算，利用前向传导公式，得到 $L_2, L_3, \ldots$直到输出层 L_{n_l} 的激活值。
2. 对输出层（第 n_l 层），计算：

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. 对于 $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ 的各层，计算：

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \bullet f'(z^{(l)})$$

4. 计算最终需要的偏导数值：

$$\begin{aligned}
\nabla_{W^{(l)}} J(W,b;x,y) &= \delta^{(l+1)} (a^{(l)})^T, \\
\nabla_{b^{(l)}} J(W,b;x,y) &= \delta^{(l+1)}.
\end{aligned}$$

实现中应注意：在以上的第2步和第3步中，我们需要为每一个 i 值计算其 $f'(z_i^{(l)})$。假设 $f(z)$ 是sigmoid函数，并且我们已经在前向传导运算中得到了 $a_i^{(l)}$。那么，使用我们早先推导出的 $f'(z)$表达式，就可以计算得到 $f'(z_i^{(l)}) = a_i^{(l)}(1 - a_i^{(l)})$。

最后，我们将对梯度下降算法做个全面总结。在下面的伪代码中，$\Delta W^{(l)}$ 是一个与矩阵 $W^{(l)}$ 维度相同的矩阵，$\Delta b^{(l)}$ 是一个与 $b^{(l)}$ 维度相同的向量。注意这里“$\Delta W^{(l)}$”是一个矩阵，而不是“Δ 与 $W^{(l)}$ 相乘”。下面，我们实现批量梯度下降法中的一次迭代：

1. 对于所有 l，令 $\Delta W^{(l)} := 0$，$\Delta b^{(l)} := 0$ （设置为全零矩阵或全零向量）
2. 对于 $i = 1$ 到 m，
   a. 使用反向传播算法计算 $\nabla_{W^{(l)}} J(W,b;x,y)$ 和 $\nabla_{b^{(l)}} J(W,b;x,y)$。
   b. 计算 $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W,b;x,y)$。
   c. 计算 $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W,b;x,y)$。
3. 更新权重参数：

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

现在，我们可以重复梯度下降法的迭代步骤来减小代价函数 $J(W,b)$ 的值，进而求解我们的神经网络。

中英文对照

反向传播算法 Backpropagation Algorithm
（批量）梯度下降法 (batch) gradient descent
（整体）代价函数 (overall) cost function
方差 squared-error
均方差 average sum-of-squares error
规则化项 regularization term
权重衰减 weight decay
偏置项 bias terms
贝叶斯规则化方法 Bayesian regularization method
高斯先验概率 Gaussian prior
极大后验估计 MAP
极大似然估计 maximum likelihood estimation
激活函数 activation function
双曲正切函数 tanh function
非凸函数 non-convex function
隐藏层单元 hidden (layer) units
对称失效 symmetry breaking
学习速率 learning rate
前向传导 forward pass
假设值 hypothesis
残差 error term
加权平均值 weighted average
前馈传导 feedforward pass
阿达马乘积 Hadamard product
前向传播 forward propagation

中文译者

王方（fangkey@gmail.com），林锋（xlfg@yeah.net），许利杰（csxulijie@gmail.com）

Language : English

- This page was last modified on 2 December 2016, at 01:30.