《神经网络与深度学习》第二章知识点

张翼鹏　杨思敏

October 8, 2019

目录

摘要

上一章我们已经介绍了神经网络的结构、梯度下降法的应用，并初步构建了神经网络识别手写数字的代码框架，但其中有一个重要部分并未做具体介绍－－反向传播算法。本章将围绕反向传播法的工作原理展开一系列讲解。

0 回顾

关于神经网络识别手写数字的案例，我们目前所掌握的信息：

1. 对于每个样本（即手写图片），我们拥有784个输入记为x（$28*28$个像素的灰度值）和1个期望输出记为$y(x)$（即该样本表示哪个数字。在该神经网络中，$y(x)$为十维向量，只有一个维度的值是1，其余都是0）。

2. 每个样本被输入进神经网络，经过与w和b的前向传播，最终输出一个激活值向量a，即神经网络对于该样本属于哪个数字所做出的判断。a与$y(x)$越接近，表示神经网络的判断越准确。

3. 于是我们定义代价函数：

$$C(w,b)=\frac{1}{2n}\sum_{x}\|y(x)-a\|^2$$

   代价函数C表示所有样本的实际输出与其期望输出的均方误差的平均值。C越小，则表示我们训练出的神经网络效果越好（不考虑过拟合）。

4. C是关于w和b的函数，我们要做的就是通过梯度下降法调整w和b，使C达到最小值。

1 什么是反向传播法

首先我们先回顾一下用梯度下降法调整神经网络权重和偏置的公式：

$$w_k'=w_k-\eta\frac{\partial C}{\partial w_k}$$

$$b_l'=b_l-\eta\frac{\partial C}{\partial b_l}$$

而反向传播的核心就是求一个$\frac{\partial C}{\partial w}$（或$\frac{\partial C}{\partial b}$）的表达式，它让我们细致领悟到如何通过改变权重和偏置来改变整个网络的行为。

2 推导反向传播法前的准备

2.1 使用矩阵快速计算输出

当我们有了第$l-1$层的所有激活值、第l层的所有权重和偏置，就可以用下式将两层的激活值联系起来（即前向传播）：

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$$

我们用矩阵的形式改写上式，引入权重矩阵w^l、偏置向量b^l以及向量化函数σ，上式可以被改写为：

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

矩阵化的表达式省略了索引下标，形式上更加美观简洁；且在实际应用中可以利用编程软件的矩阵库中提供的矩阵乘法、向量加法等快速方法，提高了算法的运算速度。另外值得注意的是，在上式中，我们计算了中间量z^l，称为第l层神经元的带权输入：

$$z^l \equiv w^l a^{l-1} + b^l \qquad a^l = \sigma(z^l)$$

在接下来的讨论中，我们会充分利用带权输入z^l

2.2 关于代价函数的两个假设

代价函数形式如下：

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

L表示神经网络的层数，$a^L(x)$表示对于单个样本x，网络最后输出的激活值的向量。

假设1: 代价函数可以被写成每个训练样本x的代价函数C_x的均值，即$C = \frac{1}{n}\sum_x C_x$。对于目前我们所用的二次代价函数，$C_x = \frac{1}{2}\left\| y - a^L \right\|^2$。

原因: 反向传播实际上是对一个独立的训练样本x计算$\frac{\partial C_x}{\partial w}$和$\frac{\partial C_x}{\partial b}$，然后再通过在所有训练样本上取平均值得到$\frac{\partial C}{\partial w}$和$\frac{\partial C}{\partial b}$。实际上，有了这个假设我们可以认为训练样本$x$已经固定住了，可以丢掉其下标，将代价函数$C_x$看作$C$。

假设2: 代价可以写成关于神经网络输出的函数（即C是关于a^L的函数）。

原因: 推导反向传播算法时，我们要以a^L为起点，推出我们想要的表达式。

2.3 Hadamard乘积

反向传播算法会用到一种运算，$s \odot t$。例如：

$$\begin{bmatrix}1\\2\end{bmatrix} \odot \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1*3\\2*4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$$

3 反向传播的四个基本方程

3.1 误差的概念

在开始推导反向传播之前，首先引入一个中间量δ_j^l，称为第l层第j个神经元上的误差。**反向传播将给出计算误差δ_j^l的方法，然后将其关联到$\frac{\partial C}{\partial w_{jk}^l}$和$\frac{\partial C}{\partial b_j^l}$上。**

在第l层第j个神经元上，当输入a^{l-1}进来时，如果我们给带权输入z_j^l增加一个很小的变化Δz_j^l，会使该神经元输出由$\sigma(z_j^l)$变为$\sigma(z_j^l + \Delta z_j^l)$。这个变化会向网络后边的层进行传播，最终导致整个代价产生$\frac{\partial C}{\partial z_j^l}\Delta z_j^l$的改变。

于是，我们可以根据$\frac{\partial C}{\partial z_j^l}$的值来找到能够优化代价的$\Delta z_j^l$：若$\frac{\partial C}{\partial z_j^l}$的绝对值很大，则可以选择与其符号相反的$\Delta z_j^l$来降低代价；若$\frac{\partial C}{\partial z_j^l}$的值接近0，那么并不能通过改变$\Delta z_j^l$来改善多少代价，我们可以认为这时神经元已经接近最优了。所以这里有一种启发性的认识，称$\frac{\partial C}{\partial z_j^l}$是神经元的误差的度量。

按照上述描述，我们定义第l层第j个神经元上的误差δ_j^l为：

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

3.2 四个基本方程

方程1: **输出层误差δ^L的方程：**

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

写成矩阵形式：

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

由于二次代价函数$C = \frac{1}{2}\sum_j (y_j - a_j)^2$，所以$\frac{\partial C}{\partial a_j^L} = (a_j - y_j)$，写成矩阵形式有$\nabla_a C = (a^L - y)$，于是方程1最终的矩阵形式为：

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

解释: 右式第一项$\frac{\partial C}{\partial a_j^L}$表示代价随第$j$个输出激活值的变化而变化的速度；右式第二项$\sigma'(z_j^L)$表示激活函数$\sigma$在$z_j^L$处变化的速度。

方程2: **用下一层的误差δ^{l+1}来表示当前层的误差δ^l：**

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

解释: 我们把转置后的权重矩阵$(w^{l+1})^T$作用到第$l+1$层的误差δ^{l+1}上，可以凭直觉把它看作是在沿着神经网络反向移动误差。然后我们进行Hadamard乘积运算$\odot\sigma'(z^l)$，让第$l+1$层的误差通过第l层的激活函数反向传递回来得到第l层的误差。

方程3: **代价函数关于任意偏置的改变率：**

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

简记为（其中δ和b都是针对同一个神经元）：

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

方程4: **代价函数关于任意权重的改变率：**

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

简记为：

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

解释: 当a_{in}很小时，$\frac{\partial C}{\partial w}$也会趋向很小，结果就是作用在低激活值上的权重学习会非常缓慢。

我们将这四个方程组合在一起，还可以总结出一个规律：当$\sigma(z_j^l)$近似为0或1时，$\sigma'(z_j^l) \approx 0$。所以如果某神经元的输出处于低激活值（≈ 0）或高激活值（≈ 1）时，我们称该神经元已经饱和了，对应的权重和偏置的学习将会非常缓慢（甚至终止）。

3.3 四个方程的证明

证明略。

我们可以将反向传播看成是一种系统性地应用多元微积分中的链式法则来计算代价函数梯度的方法。

4 反向传播的优点

为了计算$\frac{\partial C}{\partial w}$和$\frac{\partial C}{\partial b}$，还有另一个常见的方法——差分法。下面只拿权重举例，将代价看作权重的函数$C = C(w)$，给所有权重编号$w_1, w_2, \cdots$，有以下近似公式：

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j) - C(w)}{\epsilon}$$

其中ϵ是一个很小的正数，e_j是在第j个方向上的单位向量。

这个方法概念易懂，代码易于实现。为什么我们不用，因为它运行起来非常缓慢。

假设我们的神经网络总共有一百万个权重，对于计算每个$\frac{\partial C}{\partial w_j}$，为了得到$C(w+\epsilon e_j)$的值，我们需要进行一次完整的前向传播。这意味着我们为了计算梯度，要计算一百万次代价函数，需要一百万次前向传播。

而反向传播聪明的地方就是它确保我们可以同时计算所有的偏导数$\frac{\partial C}{\partial w_j}$，仅使用一次前向传播加一次后向传播，计算代价约为两次前向传播。

5 反向传播的全局观

至此我们已经了解并证明了反向传播，但其依然显得有些神秘。反向传播算法究竟是怎么被发现的？是否有一个推理的思路可以引导我们发现反向传播算法？我们能否从直觉上进一步理解反向传播究竟在做什么？下面将从全局的角度，讨论反向传播法的思路。

我们假设对网络中的某个权重w^l_{jk}做了一些微小的改动Δw^l_{jk}，这个改动会使得对应神经元的输出激活值a^l_j发生改变，然后会使下一层所有激活值发生改变，一层一层影响下去，到达输出层，最终使得代价函数发生改变。所以代价函数的改变ΔC和Δw^l_{jk}就按照下式联系起来：

$$\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk}$$

这里给出了一个计算$\frac{\partial C}{\partial w^l_{jk}}$的思路，即细致地追踪$w^l_{jk}$的微小变化如何导致$C$的变化，如果我们可以精确地使用易计算的量来表达每种关系，那么我们就能够计算$\frac{\partial C}{\partial w^l_{jk}}$了。

其实反向传播的思想非常简单。举个例子，如果要求y关于x的导数，最简单的方法就是找到y与x的关系式，然后直接求导即可。反向传播法也是一样。比如求δ^L时用到的$a^L - y$，就是C对a^L直接求导得到的。这也是为什么反向传播法需要从后向前运行，因为我们若改变第l层的某个权重，考察它对代价的影响，那么这跟前$l-1$层是没有任何关系的。所以我们先从输出层出发，因为改变输出层的权重对它之前的所有层都没有影响，我们只需考虑输出层与代价的关系，于是很容易就能写出C与输出层权重（或偏置）的关系，也就很容易能计算出偏导数。但由于神经网络结构复杂，很难写出代价与所有权重的关系式，所以我们需要用到一些微积分的知识，比如链式法则，对这个思路进行一些改进。

下面我们尝试一下这个方法。Δw^l_{jk}首先导致a^l_j发生变化：

$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

a^l_j的变化会导致第$l+1$层所有激活值发生变化，我们取其中一个激活值a^{l+1}_q：

$$\Delta a^{l+1}_q \approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \Delta a^l_j$$

联立上面两个式子得：

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

然后这个变化Δa_q^{l+1}又会去影响下一层激活值。我们可以想象出一条从w_{jk}^l到C的路径，假设激活值的序列如下$a_j^l, a_q^{l+1}, \cdots a_n^{L-1}, a_m^L$，那么结果的表达式就是

$$\Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

当然，从w_{jk}^l到C存在很多路径，为了计算C的全部改变，我们需要对所有可能的路径进行求和：

$$\Delta C \approx \sum_{mn\cdots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

将上式与本节第一个式子联立得：

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mn\cdots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$

我们如果按照上式求解$\frac{\partial C}{\partial w_{jk}^l}$，先算出等式右端所有偏导数的显示表达式，再进行简化，最后我们会发现其实就是在做反向传播。

更深层地说，**反向传播其实是一种巧妙地追踪权重（和偏置）微小变化如何传播并抵达输出层影响代价函数的技术。**