CS231n学习#2：最优化与反向传播

原创 Zerg_Wang 于 2019-02-01 13:28:21 发布 266 收藏 1 编辑 版权
分类专栏： Machine Learning

1.最优化

最优化的过程就是不断更改W的参数使目标函数更小的过程，换句话说，对于公式

$Loss = f(W) + \lambda R(W)$，我们的目标就是找到一个能使Loss取到最小值的W。

那应该如何找到这个W呢？

算法一：随机生成多个W，选择Loss最小的，这个方法简单易行，但效率过于低下。

算法二：对于某个随机生成W，每次随机生成一个微小值δW，当$W + \delta W$的Loss更小时更新W，虽然比算法一好，但效率还是太低。

算法三：跟随梯度。实际上可以通过计算得到W能使Loss减小的更新的方向。用中学数学的话说，就是对Loss求导，在导数小于0的地方寻求W。而梯度就是一个向量，表示某一函数在该点处的方向导数沿着该方向取得最大值，即函数在该点处沿着该方向（此梯度的方向）变化最快，变化率最大（为该梯度的模）。对于初等函数，梯度即其斜率（导数），对于多变量函数，梯度为其各个变量的偏导数所形成的向量。

在该算法下，我们需要计算出损失函数的梯度，然后按梯度的负方向更新即可，用公式表示：

$$W_{new} = W - \alpha \frac{\partial L}{\partial W}$$

下面有两种计算梯度的方法：

数值梯度法

即根据以下公式计算：

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

考虑到可能有多个变量（维度），因此逐一枚举变量，对每个变量求偏导即可。

因为h的值不能真的取0，一般取到1e-5即可。

另外，采用对称导数（Symmetric Derivative）计算会更精确：

$$\lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}.$$

然而，在按梯度更新时，还需要考虑一个超参数：学习率（Learning rate，也称为步长）。即W沿梯度负方向的更新程度。若学习率过低，虽然稳健但效率不高，若学习率过高，易“矫枉过正”。

我们以$y = 3x^2 + 6x$这个函数为例，这里的y代表Loss，x代表W，现在采用跟随梯度的方法找到使y最小的x值（显然x=-1的时候y取到最小值），假设初始化x=2，在此处梯度（导数的值）为18，如果学习率设为0.1，则更新后：

$$x_{new} = x - 0.1 * 18 = 0.2$$

显然，当x=0.2时，y的值的确变小了，但如果学习率为0.01，新的x值为1.82，虽然y也变小了，但效率明显不如学习率为0.1的时候高，若将学习率设为1，则新的x=-16，y的值反而变大了。因此在实际应用中，学习率的设定非常重要。

分析梯度法

数值梯度法虽然简单易行，但每次更新，都要对W中所有参数进行梯度计算，如果面对神经网络中千万级别的参数，使用数值梯度法效率无疑是极低的，此外，数值梯度法得到的梯度是近似值，存在误差。因此应采用分析梯度法。

分析梯度法其实就是对损失函数直接求偏导，存储得到的偏导数，在每次更新权重计算梯度时直接代入W的值即可。

以SVM为例：（下面公式的将Wj和Wr视为变量，并非Xi）

$$L_i = \sum_{j \neq r}^{k} max(0, W_j X_i - W_r X_i + t)$$

对Wj求偏导，得：（若括号中的不等式成立，则括号内的值为1，否则为0）

$$\frac{\partial L_i}{\partial W_j} = X_i(W_j X_i - W_r X_i + t > 0)$$

对Wr求偏导，得：

$$\frac{\partial L_i}{\partial W_r} = -X_i(\sum_{j \neq r}^{k}(W_j X_i - W_r X_i + t > 0))$$

以带对数的Softmax为例：

$$L_i = -ln(\frac{e^{W_r X_i}}{\sum_{j=1}^{k} e^{W_j X_i}})$$

对Wj求偏导，得：（为了防止混淆，原来在西格玛函数处的j临时换成t）

$$\frac{\partial L_i}{\partial W_j} = \frac{e^{W_j X_i}}{\sum_t^k e^{W_t X_i}} X_i$$

对Wr求偏导，得：

$$\frac{\partial L_i}{\partial W_r} = \frac{e^{W_r X_i}}{\sum_j^k e^{W_j X_i}} X_i - X_i$$

小批量数据梯度下降（Mini-batch gradient descent）

在跟随梯度更新权重时，实际上没有必要拿整个数据集的图片数据参与梯度的运算。一方面是因为整个数据集可能过于庞大，全部参与运算效率极低；另一方面是因为数据集中的数据可能较为近似，全部参与运算可能会有重复的现象（其实数据集中的图片是没有重复的），有鉴于此，我们可以挑选一部分数据作为数据集的近似来更新梯度。

那每次更新权重要多少数据参与呢？这也是一个超参数，一般由存储器的限制决定，也可以设置为32、64、128等2的倍数，因为向量化操作时数据量为2的倍数的话效率会更高。

2.反向传播

对于一个函数（例如前文提到的损失函数），按照给出的W与x计算出Loss，这个过程称为前向传播（之后学到的CNN中的各种卷积、池化计算都是前向传播），而根据这些函数求其梯度的过程就是反向传播。

分析梯度法固然好用，但具有应用意义的神经网络并不仅仅像上面的损失函数那么简单，面对极为复杂的神经网络，有什么更为简便的方法求出其函数的梯度呢？这就需要使用链式法则（Chain Rule，其实这也是大学高数的内容 ）

链式法则需要用到中间函数，举个例子：

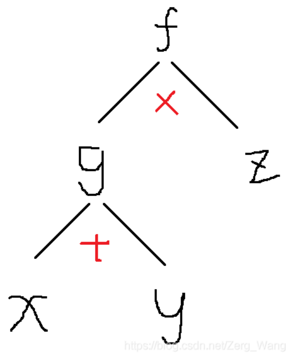
$$f(x, y, z) = (x + y)z$$

显然，利用乘法分配律去括号之后可直接得结果，但按链式法则的方法，可把这个函数看作以下两者的乘积：

$$f(x, y, z) = z * g(x, y)$$

$$g(x, y) = x + y$$

其中，g(x,y)就是中间函数，然后我们可以画出这张表：



红色字符表示变量间的运算关系，然后根据上图：

$$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

因为：

$$\frac{\partial f}{\partial g}=z,\frac{\partial g}{\partial x}=1$$

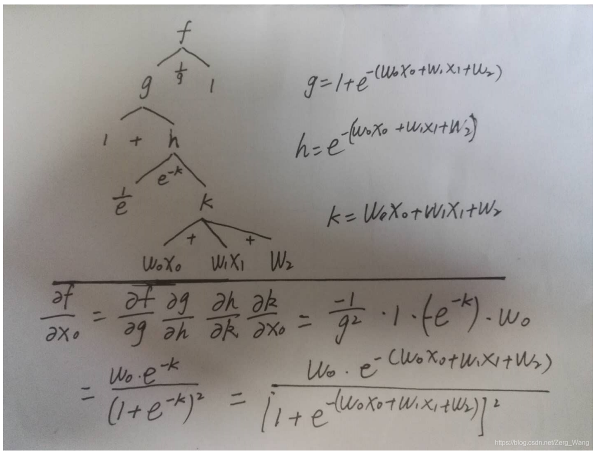
所以：

$$\frac{\partial f}{\partial x}=z$$

同理也可得到f关于y和z的偏导数，下面再看一个更为复杂的例子：（该函数表示一个带sigmoid激活函数的双层神经网络）

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

画图并对x0求偏导：



利用链式法则求梯度的关键在于：要充分利用中间函数，将原本复杂的函数拆分成多个易于求导的函数（这里可以使用模块化的思想，把复杂函数拆分成一个个模块，从而降低求导难度，提高效率，例如上式中被分为了加法模块、乘法模块、指数模块等）

再提一点，在损失函数中经常会遇到求最大值函数，其偏导如下：

$$f(x,y)=max(x,y)$$

$$\frac{\partial f}{\partial x}=x>y,\frac{\partial f}{\partial y}=y>x$$

当x>y时f对x的偏导为1，否则为0，y同理。

需要注意的是，如果变量在前向传播的表达式中多次出现，偏导的结果要将这些变量累加起来，例如：

$$f(x,y)=\frac{x+y}{2x+3y}$$

设：

$$g = x + y, h = \frac{1}{2x+3y}, f = gh$$

则有：

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial x} = \frac{1}{2x+3y} + \frac{-2(x+y)}{(2x+3y)^2}$$

实际运算中，变量往往不是一个实数，而是一个矩阵。虽然以上的运算均适用于矩阵，但运算时还是要关注维度以及矩阵的转置操作。

本篇博文知识基于斯坦福大学CS231n课程，经博主学习、整理而来，仅为个人理解，若有错误，欢迎批评指正，谢谢！

参考来源：

http://cs231n.stanford.edu/

https://zhuanlan.zhihu.com/p/21930884

附上cs231n的网易云课堂地址：

https://study.163.com/course/courseMain.htm?courseId=1004697005