assignment2 我的答案

xzx

2016 年 12 月 4 日

1 Q1: Fully-connected Neural Network

1.1 /cs231n/layers.py

1.1.1 affine_forward & affine_backward

affine_forward 没什么好说的，就是正常的相乘就完了。

affine_backward 需要注意的地方是再对 dout 求和的时候，参数 keepdims=True 很有用。假设 dout 是 (N,M) 的矩阵，

```
np.sum(dout, axis=0)#返回一个 (M,) 的矩阵
np.sum(dout, axis=0,keepdims=True)#返回一个 (1,M) 的矩阵
```

1.1.2 /cs231n/classifiers/fc_net/py

TwoLayerNet 与 assignment1 不同的是，这里的两层网络加上了一个 relu 层。在做作业的时候，开始忘掉了 relu 层，导致不论怎么调 hyper parameter 训练集上面的 accuracy 最高只能到达 43%。因为 affine 层是线性层，线性层不论怎么叠加的效果都是线性层，所以没有 relu 的情况下准确率很难提高

1.2 update rules

课程中介绍了几种主流的更新 W 的方法。想象当前所在的位置是山谷中随机的一个位置，目标是走到山谷的底部。
最简单的就是 SGD，每次按照偏导数的方向 (下山的方向) 走一小步，直到走到结束条件为止。
第二种方法是 Momentum。假如一个小球从山谷的某个地方向下滚，这个小球会有一个当前速度，还有一个加速度的方向，它下一步的速度方向应当是由当前速度和当前加速度一起决定的。所以再计算的时候要记录下当前的速度，把偏导数的方向当做加速度的方向。

```
v = mu*v - learning_rate*dx
x += v
```

这里可以把 mu 认为是在向下滚动时候碰到的摩擦力第三种方法是 AdaGrad。在视频中可以看到 SGD 的一个缺点是它会沿着偏导数较大的方

向走很远，但是偏导数较小的方向走的步伐会很小，这样在向谷底走的时候，会产生很大的震荡。

```
#Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

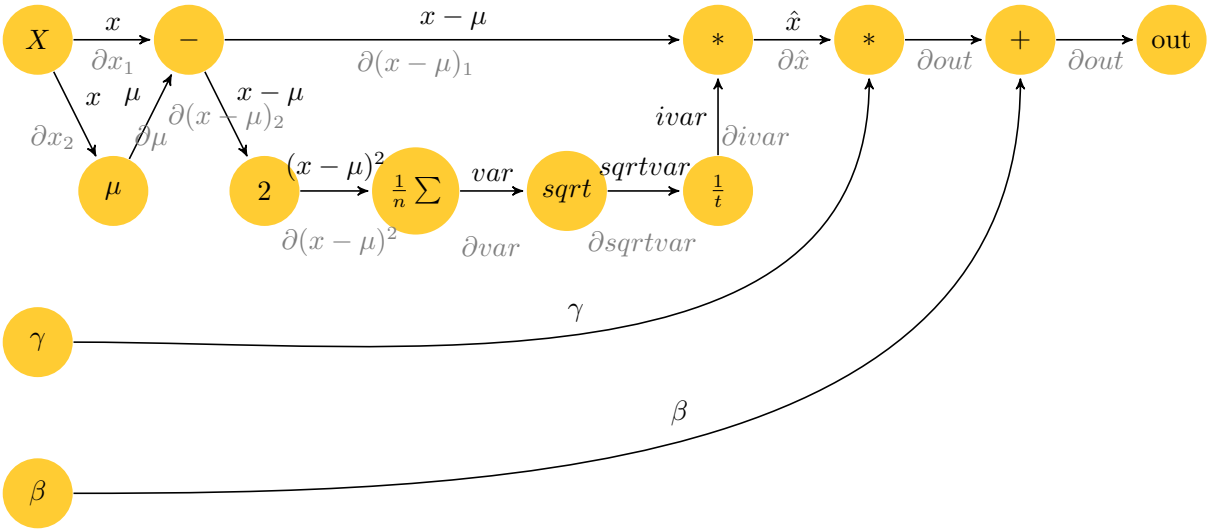
cache 的作用就是不让更新的步伐沿着偏导数较大的方向走太远。加上 10^{-7} 仅仅是为了防止除数为零。当更新次数很多的时候，cache 会逐渐变得很大，更新的步长就会逐渐变小。

在 AdaGrad 的基础上，Hinton 提出了 RMSProp。思路是差不多的，仅仅是 cache 变化了一下。

将 AdaGrad 和 Momentum 合在一起，就是 Adam。

2 Q2: Batch Normalization

在训练神经网络的时候，会出现一个问题，在较深层的神经元，信号数值的分布很难看。这就导致了训练出来的神经网络对 learning_rate 非常敏感。Batch Normalization 的思路就是在每一层加上一个 Batch_Norm 层，保证每层神经元得到的数据都符合正态分布。这样做的结果是会降低神经网络的表现力，所以要让 Batch Norm 学习 shift 和 scale，shift 是正太分布的中心，scale 是方差（代码里面是 gamma 和 beta）。



这个图差不多是整个作业里面最复杂的一个反向传播的图了。借鉴了网上

的一个博客，具体地址忘记了。。。
$\beta = \sum \partial out$ 这个应该不难理解。
而 ∂out 仅仅是经过了一个加法运算，所以再加号左边仍然不变.
另外在前面，第一个节点 X 跟减号的节点有分叉。对于分叉的节点，偏导数就把各个分支的偏导数加起来。剩下的就按照反向传播往前传就好了。