2017CS231n李飞飞深度视觉识别笔记（六）——训练神经网络（上）

2018年07月16日 22:38:13 献世online 阅读数：289 更多

第六章 训练神经网络（上）

课时1 激活函数

我们已经讨论过如何使用计算图来表达一个函数，并且明确的探讨了神经网络，它包含若干个线性层，而层与层之间通过非线性函数进行连接实现堆叠；也介绍了卷积神经网络，它使用卷积层贯穿整个网络结构；这一章中将讨论训练神经网络的一些细节问题，包括：

（1）刚开始要如何建立起神经网络；

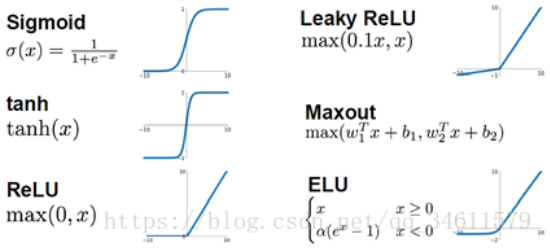
（2）要选择什么样的激活函数；

（3）如何做数据的预处理，权重初始化、正则化和梯度检查；

（4）讨论训练中的动态变化、训练监控过程、超参数优化；

（5）讨论模型评估和模型集成。

任意特定层产生的输出是输入数据在全连接层或者卷积层乘上权重值，然后将结果输入一个激活函数或者非线性单元，下面给出了一些激活函数：

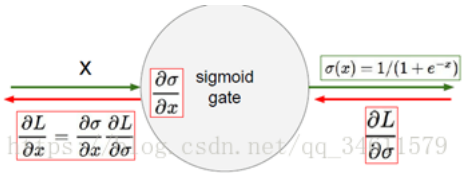


（1）sigmoid函数

函数表达式为：$\sigma(x)=\frac{1}{1+e^{-x}}$，每个元素输入到这个函数中，会被压缩在[0,1]范围内，如果有非常大的输入值，那么输出将会接近1；如果有绝对值很大的负的输入值，那么输出将会接近0。

存在的问题：

1）首先是饱和神经元将使梯度消失



当X的输入是一个很大的正值和很小的负值时，它们会位于sigmoid函数的平滑区域（从图像上看），这些区域会使梯度消失，从而无法得到梯度流的反馈。

2）sigmoid函数是一个非零中心的函数

意味着所有关于X的梯度，因为它们全是正数或负数，在更新的时候只能往一个方向移动，这种情况下梯度更新的效率会非常低，如下图中的例子：

假设最佳的W实际上是蓝色的向量，但是我们的梯度更新只能沿着红色的曲线走，目标也是得到最佳的W，但是这样的更新效率非常低。

3）使用指数函数的计算代价有点高。

（2）tanh函数

它与sigmoid函数很相似，只不过它的值被压缩在[-1,1]的范围内，所以主要的不同就是tanh函数是以0为中心，它就不会出现sigmoid函数的第二个问题，但是它仍会存在饱和使梯度消失的问题。

（3）ReLU函数

函数表达式为：$f(x)=\max(0,x)$，它在输入上按元素进行操作，如果输入的是负数，将会得到结果0；如果输入的是正数，结果还是所输入的数。看到ReLU函数的图像可以知道它在正数区域不会出现产生饱和现象，这是很大的优势，而且它的计算成本也不高，并且它比sigmoid函数和tanh函数收敛快得多（大约6倍）。

存在的问题：

1）还是出现了不以0为中心的问题；

2）负半轴还是会出现饱和现象。

（4）Leaky ReLU函数

它的函数形式为$f(x)=\max(0.01x,x)$，这个函数和ReLU函数的区别在于它的负区间有一个微小的负斜率，这就解决了之前的饱和问题，同样它也比sigmoid函数和tanh函数收敛快得多。

（5）ELU函数

它的函数形式为：

$$f(x)=\begin{cases} x & if\ x>0 \\ \alpha(\exp(x)-1) & if\ x\le 0 \end{cases}$$

它具有ReLU函数的所有优点，但是它的输出均值接近为0，与Leaky ReLU函数比较，它没有在负区间的倾斜，在负区间有饱和的问题。

（6）最大输出神经元（Maxout）

它的形式是：$\max(w_1^Tx+b_1,w_2^Tx+b_2)$，它的作用是泛化ReLU和Leaky ReLU，这是另一种线性机制，它不会饱和也不会消亡，但是它的问题在于会把每个神经元的参数数量翻倍；所以说，若每个神经元原来有权重集W，但现在有了W1和W2，相当于原来的两倍。

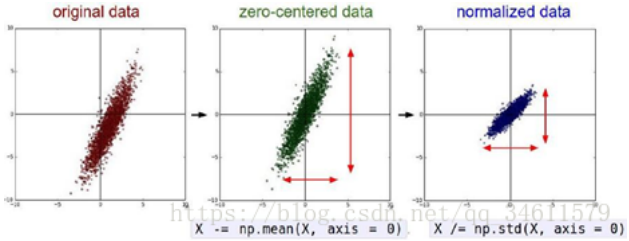
通常情况下：

1）使用ReLU时，要谨慎地调整学习速率；

2）尝试部分较为奇特的激活函数像Leaky ReLU/Maxout/ELU；

3）一般不会使用Sigmoid函数。

接下来讨论关于数据预处理的问题，一些数据预处理的标准类型是实现零均值化数据和用标准差归一化数据：



实际上，还会看到像是PCA或是数据的白化，一直以来坚持使用零均值化而不做归一化处理或是更复杂的处理；因为，一般来说，对于图像不会真的想要所有的输入，用像素值来举例，将其投影到一个更为低维的空间，这个空间有正在处理的各种新的特征，只想在空间上应用卷积网络并且得到原图像的空间结构。

在训练阶段决定均值，然后会将一样的均值应用到测试数据中去，会从训练数据中得到相同的经验均值来归一化。

总结来说，一般对于图像，做的就是零均值化的预处理，可以减去整张均值图像的值；对一些网络，也通过减去单通道的均值代替用整张均值图像来将图像集零中心化。

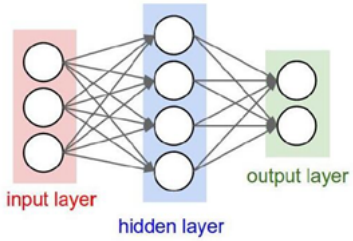
问题：减去单通道均值，通道是什么？

答：就是RGB，若图像是32*32*3的，宽高就是32*32，而深度就是三通道的RGB，所以会有一个红色、蓝色、绿色通道的均值。

问题：当减去均值图像的值时，均值是怎么得到的？

答：均值从所有的训练图像中得到。

接下来讨论如何初始化网络的权重，比如说有下图所示的两层神经网络：



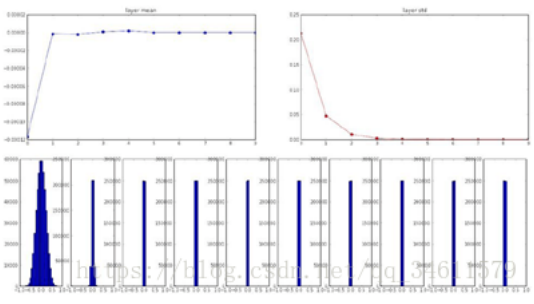
当使用零为权值，W的初始值会发生什么：每个神经元将在输入的数据上有相同的操作，接着将输出相同的数值并且得到相同的梯度；因为用相同的方式进行更新将得到完全相同的神经元，而我们期望的是不同的神经元学习到不同的知识。

所以，可以尝试将其改变为所有权重是一个小的随机数，从一个概率分布中抽样W=0.01*np.random.randn(D,H)，这里从标准高斯分布中抽样，这样的参数适合在小型网络中使用；下面给出一个例子：

初始化一个10层神经网络，每层有500个神经元，使用tanh非线性激活函数，用小的随机数来初始化，实现过程如下：

```
import numpy as np
import matplotlib.pyplot as plt

D = np.random.randn(1000, 500)
hidden_layer_sizes = [500] * 10
nonlinearities = ['tanh'] * len(hidden_layer_sizes)

act = {'relu': lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x)}
Hs = {}
for i in range(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i - 1]  # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 1.0  # layer initialization np.sqrt(fan_in)
    H = np.dot(X, W)  # matrix multiply
    H = act[nonlinearities[i]](H)  # nonlinearity
    Hs[i] = H  # cache result on this layer

print('input layer had mean %f and std %f' % (np.mean(D), np.std(D)))
layer_means = [np.mean(H) for i, H in Hs.items()]
layer_stds = [np.std(H) for i, H in Hs.items()]
for i, H in Hs.items():
    print('hidden layer %d had mean %f and std %f' % (i + 1, layer_means[i], layer_stds[i]))

# plot the means and std
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')
plt.show()

# plot the raw distributions
plt.figure()
for i, H in Hs.items():
    plt.subplot(1, len(Hs), i + 1)
    plt.hist(H.ravel(), 50, range=(-1, 1))
plt.show()
```

实现的效果图：

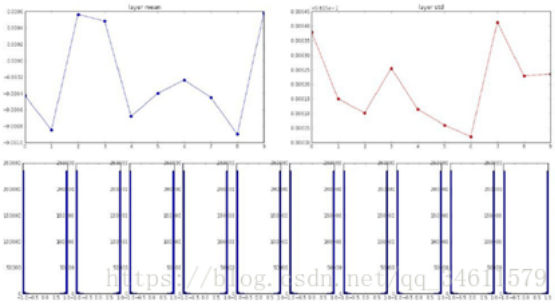


可以看到这里的输出，均值总是在0附近，所以因为tanh函数是以0为中心，这就变得很有意义；而标准差会不断缩小，快速逼近0；从上图中显示的每层均值和标准差，展示了每一层激活值的概率分布。

可以看到第一层有一个合理的高斯分布，而当每一层乘以一个W（一些小随机数）后，随着一次次乘法，迅速的缩小到最后是一堆0，所有的激活值都变成0，这并不是所期望的。

反向传播的梯度是上层的梯度乘以本层的梯度（W乘以X），反向传播的所有层基本上是在做上层梯度和权重的乘法来得到下层的梯度；由于这个原因，不断乘以W基本是相同的现象，所有数据越来越小，梯度和上层梯度趋向于0。

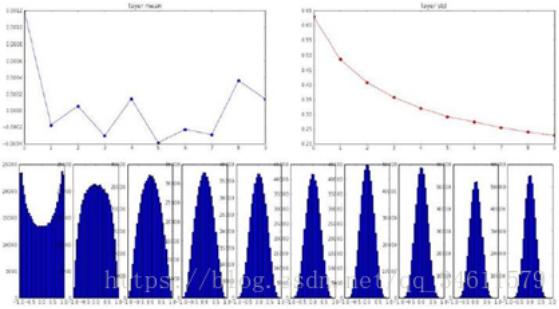
如果尝试用增大权重来解决这个问题，不在用0.01，而是用1来替代W = np.random.randn(fan_in, fan_out) * 1.0 ，会出现下面的情况：



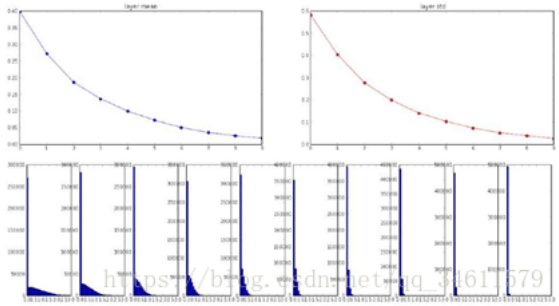
由于权重增大，就会始终处于饱和状态，不管是tanh的负方向还是正方向；如果观察激活值在每一层的分布，它们会趋向于-1或1。

初始化权重一个很好的经验是Xavier initialization。

W=np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)，这种方法在每一层都能大概获得一个高斯分布。



上述是使用tanh激活函数时的情况，当使用ReLU激活函数时，它会“消除”一半的神经元，所以实际是把得到的方差减半，将再次看到分布趋向于0。



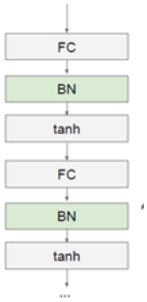
课时2 批量归一化

所谓的批量归一化是怎么工作的呢？

假设当前的批处理中有N个训练样本，并且假设每批是D维的，将对每个维度进行独立计算经验均值和方差，所以基本上每个特征元素通过批量处理都计算过了，并且对其进行归一化。

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

批量归一化通常是在完全连接或者卷积层之后插入的，不断在这些层上乘以W，基本上是通过与神经元、激活函数相连的输出来进行缩放，所以可以用相同的方法来完全连接卷积层，唯一的区别是在卷积层的情况下，不仅想要归一化所有的特征维度的独立训练实例，实际上想要归一化所有的特征维度和空间位置的所有特性。



但是这里有一个问题是我们并不清楚要在每个全连接层之后进行批量归一化操作，也并不清楚是否确定想要给这些tanh非线性函数输入一个单位高斯数据，所有强调的是在完成归一化操作之后需要进行额外的缩放操作；先做归一化，然后使用常量γ进行缩放，再用另一个因子β进行平移：

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

如果网络需要的话，它可以学习缩放因子γ，使之等于方差，可以学习β使之等于均值，在这种情况下就能恢复恒等映射，就像进行批量归一化一样。

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$
$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

因此，就有了让网络为了达到比较好的训练效果而去学习控制让tanh具有更高或更低饱和度的能力。

总结一下批量归一化的思想：

（1）提供输入，然后计算小批量均值，对每个输入的小批量都做这个操作，然后计算方差，通过均值和方差进行归一化，还有额外的缩放和平移因子，从而改进了整个网络的梯度流；

（2）它具有很高的鲁棒性，能在更广范围的学习率和不同初始值下工作；

（3）可以把它看作一种正则化的方法。

注意：在测试阶段，批量归一化层是不同的，它不需要重新计算均值和方差，只是把这当成了训练阶段。

课时3 观察学习过程

我们已经定义了网络结构，现在将要讨论如何监视训练并在训练过程中调整这些超参数以获得最好的学习结果。

一直以来第一件事是进行数据的预处理，然后要做的是初始化网络，网络进行前向传播并确实最终损失函数是合理的，接下来做的是加入0正则化项，然后开始训练，需要确定最优的学习率（一般设置在1e-3到1e-5之间）。

那么到底如何选择这些超参数呢？

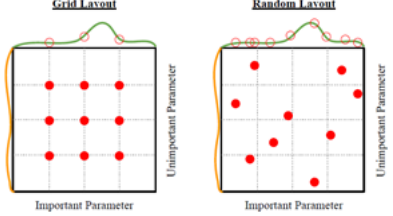
策略是对任何超参数执行交叉验证，交叉验证是在训练集上训练然后在验证集验证，观察这些超参数的实验结果。

首先，选择相当分散的数值，然后用几个epoch的迭代去学习，通过几个epoch可以知道哪些超参数有效，就可以做出相应的调整；

接着，花长时间来运行并在区间进行进一步精确搜索；

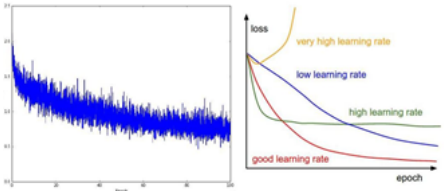
在训练循环中，找到一个像nan这样激增的技巧（nan表示设置的学习率过大），开始训练一些参数，在每一个迭代或epoch观察代价，如果出现一个远大于初始代价的值，就可以知道这不是一个正确的方向，它会迅速变得非常大跳出循环停止这个参数的巡练。

另一个就是使用网格寻找，对不同超参数的采样，可以对每个超参数的一组固定值采样，但是实际上不如使用一种随机排列的方式，对每一个超参数在一定范围内进行随机值采样，

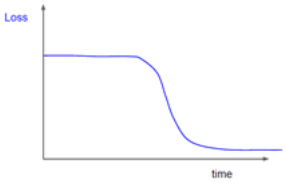


随机更加真实，通常会对我们真正有的维度进行稍微有效的降维，接着就可以得到已有的重要变量的样本，可以看到右图中上方画出的绿色的圈比左边的网格分布要更多（绿色的圈表示较好值的位置）。

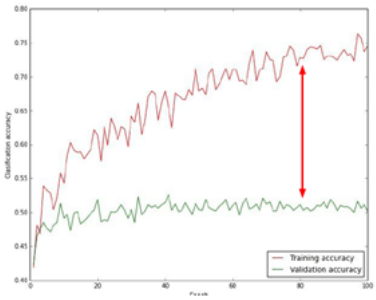
在观察每一条损失曲线的时候，学习率是重要的一个因素，可以很好的感受到哪些学习率是好的，哪些学习率是不好的，如下图所示：



另外要提到的是，当观察每一条损失曲线的时候，如果它在一定时间内很平滑，然后突然开始训练，这可能是初始值没有设好，如下图所示：



对于下面这张图，不断的积累经验就能很快的发现哪里有问题：



如果训练精度和验证精度间有一个很大的差值，意味着可能产生过拟合，可以试着增加正则项权重；如果没有差值，意味着没有过拟合，可以增加模型容量，这样也会提高精度。

总结：

（1）讨论了激活函数；（ReLU用来减去均值）

（2）数据预处理、权值初始化；（Xavier用来初始化）

（3）批量归一化；

（4）监测学习过程以及参数优化。（超参数随机采样）