干货|如何调试神经网络（深度神经网络）？

<http://blog.csdn.net/dcxhun3/article/details/53924524>

神经网络的调试基本上难于绝大多数的程序，因为大部分的神经网络的错误不会以类型错误或运行时错误显现，他们只是使得网络难以收敛。

如果你是一个新人，这可能会让你非常沮丧。一个有经验的网络训练者可以系统的克服这些困难，尽管存在着大量似是而非的错误信息，比如：

你的网络训练的不太好。

对缺少经验的人来说，这个信息令人却步；但对有经验的人来说，这是一个非常好的错误消息。它意味着样板代码已经偏移了正确道路，而且是时候去深入发掘一下原因了。

### 如何应对NaN

“为什么出现了NaN？”有时候，这个问题的答案很复杂。但大多数情况是，NaN在前100轮迭代中就出现了，这时候这个答案就非常简单：学习率太高了。当学习率很高的时候，在训练的前100轮迭代中就会出现NaN。尝试不断的把学习率除以三，直到前一百轮迭代中不再得到NaN，当这样的尝试起作用的时候，就得到了不错的初始学习率。我的经验是，最优的学习率一般在你得到NaN的学习率的1-10倍以下。

如果是在100轮迭代后出现了NaN，可能有两个更深层的原因

**- 原因1**

如果你训练的是RNN，请确保使用的是“梯度剪裁（clip gradient ）”，这可以把全局的梯度二范数限制在一定的范围内。RNN倾向于在训练早期产生梯度，其中10%或者更少的batch会出现学习尖峰，这些尖峰上的梯度值非常大。如果没有限制幅度，这些尖峰就可能导致NaN。

clip gradient工作的具体细节如下：

1.在solver中先设置一个clip\_gradient 训练中会把所有权值的梯度diff相加，如果这个值大于clip   
2. gradient，就会求一个缩放因子 s = clip\_gradient/sum\_diff   
3. 将所有的权值梯度乘以这个缩放因子，这时得到的梯度才是最后的梯度信息

**- 原因2**

如果你写了定制化的later，这个问题很可能是由这些定制化的layer中一些除零错误引发的。还有一个著名的产生NaN的layer就是softmax layer。 softmax的计算在分母和分子中都含有指数函数exp(x)，当inf除以inf时就会产生NaN。所以要确定你用的是一个稳定版本的softmax的实现。

**当神经网络不再学习的时候该怎么做？**

当你不再碰到NaN的时候，很可能就会遇到这样一种情况，你的网络平滑的训练的几千次，但是loss却在前几百个回合后不再减小。如果你是初次构建代码库的话，超过2000次的等待很难给出答案。这不是因为所有网络都能在2000次迭代内开始学习，而是因为你在编码中引入bug的几率很高，与其等待长时间的迭代，不如早早的进入调试模式。现在你应该不断缩小问题的范围，直到你的网络可以在2000次迭代内开始学习。幸运的是，有两个不错的维度来减小复杂度：

1、 把训练集的样本量减小到10。任何一个可用的网络通畅都能在几百个迭代后过拟合十个样本。而bug则会阻止过拟合发生。如果网络仍然不能过拟合十个样本，再次确认样本和label是正确对应的。然后把batch size设为1来检查batch计算中的错误。在code中加入一些log输出以确保是以你期望的方式运行的。一般来说，通过暴力排查总会找到这些错误。一旦网络可以拟合10个样本，它也可以拟合100个。如果现在可以训练但不如预期，则可以进入下一个步骤了

2、解决你感兴趣的问题的最简单版本。如果你在做句子翻译，先建立一个目标语言的语言模型。当上一步成功了，只给出三个源语言的单词，尝试着去预测翻译的第一个词。如果你打算从图像中检测物体，训练回归网络之前试着去分类图像中有多少个物体。在得到网络可以解决的好的子问题以及花费最少的时间来使用代码挂接数据之间存在着平衡点。创造力可以起到帮助作用。

为一个新的想法扩大网络的小技巧就是缓慢的缩小上述两步中的简化。这是坐标上升法的一种形式，而且十分有用。一开始，你可以证明这个网络可以记住少量的样本，然后可以证明它在一个简化版的子问题中可以在验证集上具有泛化能力。缓步提升难度，稳步前进。

有些时候你会发现有些问题本身十分困难，难以在2000次迭代内完成学习。这很棒，但这也很少需要以前那些难度的问题训练时间的十倍以上的时间。如果真需要这么多时间，可以尝试寻找一个中间的复杂度。

**微调超参数**

也许你的网络现在开始学习东西了，但你可能发现它不能解决这个问题中最困难的部分。超参数的调整就是其中的关键。也许有人下载了一个cnn包然后在上面跑自己的数据集，并告诉你超参数的调整并不会带来改变。你要认识到他们在用已有的框架解决已有的问题。如果你在使用新架构解决新问题，则必须调试超参数来获得一个良好的配置。

1.可视化是关键。不要怕浪费时间去写一些好用的训练过程中的可视化工具。如果你还是从terminal中打印出来的loss裸眼的做可视化，那那你该考虑一下升级了。

2.权值初始化很重要。一般来说，大一点幅度的初始化权值会好一些，但太大了就会导致NaN。因此初始化权值应该要和学习率一起调试。

3.确保权值看起来是“健康的”。要了解这是什么意思，我推荐ipython   
notebook打开现有网络的权值。花一些时间来熟悉在标准数据集（如ImageNet或Penn TreeBank）上训练的成熟网络中的组件的权值直方图应该是什么样子。

4.神经网络不是对输入的尺度不敏感的，尤其当它使用SGD训练而不是其他的二阶方法训练，因为SGD不是一个尺度无关的方法。在确定缩放尺度之前，花点时间来尝试多次缩放输入的数据和输出。

5.在训练结束之前减小学习率总能带来提升。最佳的decay策略是：在k个epoch后，每n个epoch之后学习率除以1.5（k > n）。

6.使用超参数配置文件，也可以把超参数写在code里。我使用<https://github.com/Russell91/tensorbox>   
中的json文件，使用命令行加载。应该避免因为超参数的问题来重构代码。

7.随机的搜索超参数。随机搜索可以组合出你想不到的超参数组合， 并且能减少你考虑在已有超参数训练带来什么样的影响时花费的大量精力。

## 总结

调试神经网络花费的劳力比调试传统程序要多，因为几乎所有的错误都会反应成一个问题：网络的表现不够好。尽管如此，二分查找仍然起作用。

1.调整问题的难度

2.使用少量的样本

这两个方法可以帮助你找到最初的问题。然后超参数调整和长时间的等待就可以解决你剩下的问题了。