**权重初始化**

**错误：全零初始化。**让我们从应该避免的错误开始。在训练完毕后，虽然不知道网络中每个权重的最终值应该是多少，但如果数据经过了恰当的归一化的话，就可以假设所有权重数值中大约一半为正数，一半为负数。这样，一个听起来蛮合理的想法就是把这些权重的初始值都设为0吧，因为在期望上来说0是最合理的猜测。这个做法错误的！

因为如果网络中的每个神经元都计算出同样的输出，然后它们就会在反向传播中计算出同样的梯度，从而进行同样的参数更新。换句话说，如果权重被初始化为同样的值，神经元之间就失去了不对称性的源头。

**小随机数初始化。**因此，权重初始值要非常接近0又不能等于0。解决方法就是将权重初始化为很小的数值，以此来打破对称性。其思路是：如果神经元刚开始的时候是随机且不相等的，那么它们将计算出不同的更新，并将自身变成整个网络的不同部分。小随机数权重初始化的实现方法是：W = 0.01 \* np.random.randn(D,H)。其中randn函数是基于零均值和标准差的一个高斯分布来生成随机数的。根据这个式子，每个神经元的权重向量都被初始化为一个随机向量，而这些随机向量又服从一个多变量高斯分布，这样在输入空间中，所有的神经元的指向是随机的。也可以使用均匀分布生成的随机数，但是从实践结果来看，对于[算法](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)的结果影响极小。

***警告****。*并不是小数值一定会得到好的结果。例如，一个神经网络的层中的权重值很小，那么在反向传播的时候就会计算出非常小的梯度（因为梯度与权重值是成比例的）。这就会很大程度上减小反向传播中的“梯度信号”，在深度网络中，就会出现问题。

**使用1/sqrt(n)校准方差。**上面做法存在一个问题，随着输入数据量的增长，随机初始化的神经元的输出数据的分布中的方差也在增大。我们可以除以输入数据量的平方根来调整其数值范围，这样神经元输出的方差就归一化到1了。也就是说，建议将神经元的权重向量初始化为：w = np.random.randn(n) / sqrt(n)。其中n是输入数据的数量。这样就保证了网络中所有神经元起始时有近似同样的输出分布。实践经验证明，这样做可以提高收敛的速度。

上述结论的推导过程如下：假设权重w和输入x之间的内积为s=\sum^n_iw_ix_i，这是还没有进行非线性激活函数运算之前的原始数值。我们可以检查s的方差：

\displaystyle Var(s)=Var(\sum^n_iw_ix_i)  
\displaystyle =\sum^n_iVar(w_ix_i)  
\displaystyle =\sum^n_i[E(w_i)]^2Var(x_i)+E[(x_i)]^2Var(w_i)+Var(xIi)Var(w_i)  
\displaystyle =\sum^n_iVar(x_i)Var(w_i)  
\displaystyle =(nVar(w))Var(x)

在前两步，使用了[方差的性质](https://link.zhihu.com/?target=http%3A//en.wikipedia.org/wiki/Variance" \t "_blank)。在第三步，因为假设输入和权重的平均值都是0，所以E[x_i]=E[w_i]=0。注意这并不是一般化情况，比如在ReLU单元中均值就为正。在最后一步，我们假设所有的w_i,x_i都服从同样的分布。从这个推导过程我们可以看见，如果想要s有和输入x一样的方差，那么在初始化的时候必须保证每个权重w的方差是1/n。又因为对于一个随机变量X和标量a，有Var(aX)=a^2Var(X)，这就说明可以基于一个标准高斯分布，然后乘以a=\sqrt{1/n}，使其方差为1/n，于是得出：w = np.random.randn(n) / sqrt(n)。

Glorot等在论文[Understanding the difficulty of training deep feedforward neural networks](https://link.zhihu.com/?target=http%3A//jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf" \t "_blank)中作出了类似的分析。在论文中，作者推荐初始化公式为 \( \text{Var}(w) = 2/(n_{in} + n_{out}) \) ，其中\(n_{in}, n_{out}\)是在前一层和后一层中单元的个数。这是基于妥协和对反向传播中梯度的分析得出的结论。该主题下最新的一篇论文是：[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification](https://link.zhihu.com/?target=http%3A//arxiv-web3.library.cornell.edu/abs/1502.01852)，作者是He等人。文中给出了一种针对ReLU神经元的特殊初始化，并给出结论：网络中神经元的方差应该是2.0/n。代码为w = np.random.randn(n) \* sqrt(2.0/n)。这个形式是神经网络算法使用ReLU神经元时的当前最佳推荐。

**稀疏初始化（Sparse initialization）。**另一个处理非标定方差的方法是将所有权重矩阵设为0，但是为了打破对称性，每个神经元都同下一层固定数目的神经元随机连接（其权重数值由一个小的高斯分布生成）。一个比较典型的连接数目是10个。

**偏置（biases）的初始化。**通常将偏置初始化为0，这是因为随机小数值权重矩阵已经打破了对称性。对于ReLU非线性激活函数，有研究人员喜欢使用如0.01这样的小数值常量作为所有偏置的初始值，这是因为他们认为这样做能让所有的ReLU单元一开始就激活，这样就能保存并传播一些梯度。然而，这样做是不是总是能提高算法性能并不清楚（有时候实验结果反而显示性能更差），所以通常还是使用0来初始化偏置参数。

实践。当前的推荐是使用ReLU激活函数，并且使用w = np.random.randn(n) \* sqrt(2.0/n)来进行权重初始化，关于这一点，[这篇文章](https://link.zhihu.com/?target=http%3A//arxiv-web3.library.cornell.edu/abs/1502.01852" \t "_blank)有讨论。

**批量归一化（Batch Normalization）。**[批量归一化](https://link.zhihu.com/?target=http%3A//arxiv.org/abs/1502.03167" \t "_blank)是loffe和Szegedy最近才提出的方法，该方法减轻了如何合理初始化神经网络这个棘手问题带来的头痛：），其做法是让激活数据在训练开始前通过一个网络，网络处理数据使其服从标准高斯分布。因为归一化是一个简单可求导的操作，所以上述思路是可行的。在实现层面，应用这个技巧通常意味着全连接层（或者是卷积层，后续会讲）与激活函数之间添加一个BatchNorm层。对于这个技巧本节不会展开讲，因为上面的参考文献中已经讲得很清楚了，需要知道的是在神经网络中使用批量归一化已经变得非常常见。在实践中，使用了批量归一化的网络对于不好的初始值有更强的鲁棒性。最后一句话总结：批量归一化可以理解为在网络的每一层之前都做预处理，只是这种操作以另一种方式与网络集成在了一起。