1. **参数初始化**

目的：为了让神经网络在训练过程中学习到有用的信息，这意味着参数梯度不应该为0

初始化必要条件一：各层激活值不会出现饱和现象；

初始化必要条件二：各层激活值不为0

方法：

1. 随机生成小的随机数

将参数初始化为小的随机数。其中randn从均值为0，标准差是1的高斯分布中取样，这样，参数的每个维度来自一个多维的高斯分布。（不能太小，太小会导致梯度弥散）

1. 标准初始化

权重参数初始化从区间均匀随机取值。

1. Xavier初始化

优秀的初始化应该使得各层的激活值和状态梯度的方差在传播过程中的方差保持一致。不然更新后的激活值方差发生改变，造成数据的不稳定。

由于ReLU无法控制数据幅度，所以可以将其与Xavier初始化搭配使用。

1. 偏置初始化

通常偏置项初始化为0，或比较小的数，如：0.01

1. **激活函数**
2. ReLU函数

优点：

ReLU本质上是分段线性模型，前向计算非常简单，无需指数之类操作；

ReLU的偏导也很简单，反向传播梯度，无需指数或者除法之类操作；ReLU不容易发生梯度发散问题，Tanh和Logistic激活函数在两端的时候导数容易趋近于零，多级连乘后梯度更加约等于0；

ReLU关闭了左边，从而会使得很多的隐层输出为0，即网络变得稀疏，起到了类似L1的正则化作用，可以在一定程度上缓解过拟合。

缺点：

比如左边全部关了很容易导致某些隐藏节点永无翻身之日，所以后来又出现pReLU、random ReLU等改进；

而且ReLU会很容易改变数据的分布，因此ReLU后加Batch Normalization也是常用的改进的方法。

2、Sigmoid函数

优点：

输出范围有限，所以数据在传递的过程中不容易发散；

缺点：

饱和的时候梯度太小。

为了防止饱和，必须对于权重矩阵的初始化特别留意。

1. Tanh函数

优点：

输出是零中心的

缺点：

饱和的时候梯度太小。

为了防止饱和，必须对于权重矩阵的初始化特别留意。

5. 激活函数的选择

1、sigmoid在压缩数据幅度方面有优势，对于深度网络，使用sigmoid可以保证数据幅度不会有问题，这样数据幅度稳住了就不会出现太大的失误。

2、但是sigmoid存在梯度消失的问题，在反向传播上有劣势，所以优化的过程中存在不足。

relu不会对数据做幅度压缩，所以如果数据的幅度不断扩张，那么模型的层数越深，幅度的扩张也会越厉害，最终会影响模型的表现。

3、但是relu在反向传导方面可以很好地将“原汁原味”的梯度传到后面，这样在学习的过程中可以更好地发挥出来。

1. **学习率的设置**

太大的学习速率导致学习的不稳定，太小值又导致极长的训练时间。

1、固定学习率的设置：

经验选择：一般情况下倾向于选取较小的学习速率以保证系统的稳定性，学习速率的选取范围在0.01~0.8之间。

对于不同大小的数据集，调节不同的学习率。根据我们选择的成本函数F()不同，问题会有区别。当平方误差和（Sum of Squared Errors）作为成本函数时， ∂F(ωj) / ∂ωj 会随着训练集数据的增多变得越来越大，因此学习率需要被设定在相应更小的值上。解决此类问题的一个方法是将学习率λ 乘上1/N，N是训练集中数据量。这样每步更新的公式变成下面的形式：ωj = ωj - (λ/N) \* ∂F(ωj) / ∂ωj

解析：固定学习率是根据cost曲线的走向来不断调整学习率，最终获得比较好的初始化。其往往搭配Early Stopping来结束训练。

2、可变的学习率：

在每次迭代中调节不同的学习率。在每次迭代中去调整学习率的值是另一种很好的学习率自适应方法。此类方法的基本思路是当你离最优值越远，你需要朝最优值移动的就越多，即学习率就应该越大；反之亦反。例如：如果相对于上一次迭代，错误率减少了，就可以增大学习率，以5%的幅度；如果相对于上一次迭代，错误率增大了（意味着跳过了最优值），那么应该重新设置上一轮迭代ωj 的值，并且减少学习率到之前的50%。

当validation accuracy满足early stopping时，但是我们可以不stop，而是让learning rate减半之后让程序继续跑。下一次validation accuracy又满足no-improvement-in-n规则时，我们同样再将learning rate减半。继续这个过程，直到learning rate变为原来的1/1024再终止程序。（1/1024还是1/512还是其他可以根据实际确定）。