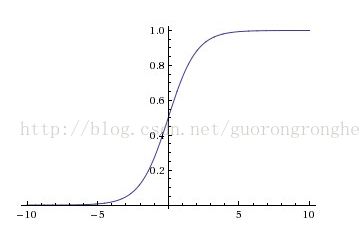
**Neural Network**

**1. 请对比下Sigmoid、Tanh、ReLU这三个激活函数。**

1. Sigmoid激活函数

公式：

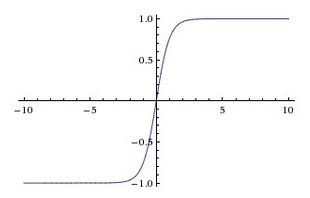
图像：



1. Tanh激活函数

公式：

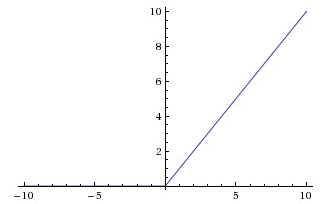
图像：



1. ReLU激活函数

公式：

图像：



**2. Sigmoid、Tanh、ReLU这三个激活函数有什么优缺点？**

|  |  |  |
| --- | --- | --- |
| 激活函数 | 优点 | 缺点 |
| Sigmoid | ①输出为0到1之间的连续实值，此输出范围和概率范围一致，因此可以用概率的方式解释输出。  ②将线性函数转变为非线性函数。 | ①容易出现梯度消失  ②函数输出并不是zero-centered  ③幂运算相对来讲比较耗时 |
| Tanh | ①对比sigmoid和tanh两者导数输出可知，tanh函数的导数比sigmoid函数导数值更大，即梯度变化更快，也就是在训练过程中收敛速度更快。  ②输出范围为-1到1之间，这样可以使得输出均值为0，这个性质可以提高BP训练的效率。  ③将线性函数转变为非线性函数。 | ①容易出现梯度消失  ②幂运算相对来讲比较耗时 |
| ReLU | ①解决了gradient vanishing问题 (在正区间)  ②计算速度非常快，只需要判断输入是否大于0  ③收敛速度远快于sigmoid和tanh | ①不是zero-centered  ②某些神经元可能永远不会被激活 |

**3. 为什么引入非线性激活函数？**

如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合。

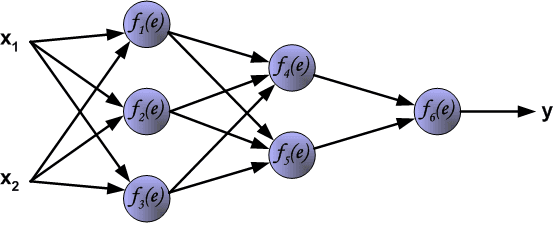
如果使用激活函数的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。

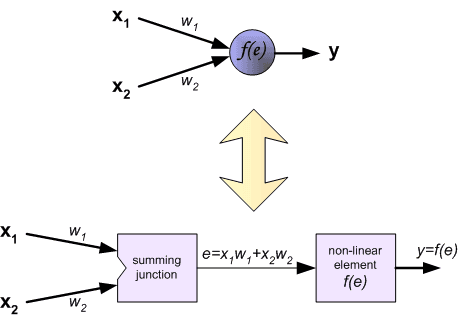
**4. 请问人工神经网络中为什么ReLU要好过于tanh和sigmoid function?**

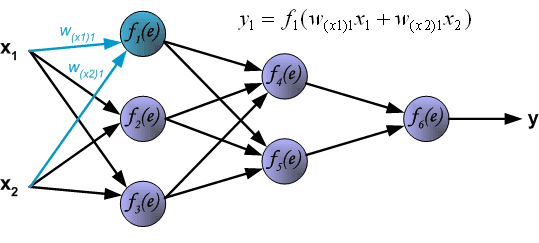
采用sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大，而采用ReLU激活函数，整个过程的计算量节省很多。

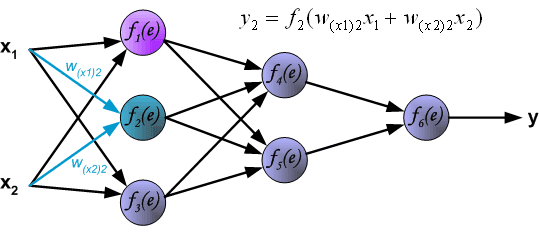
对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失），从而无法完成深层网络的训练。ReLU会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生。

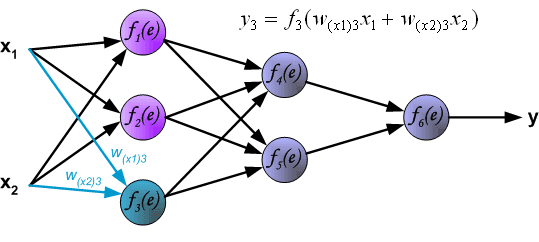
**5. 推导下反向传播Backpropagation。**

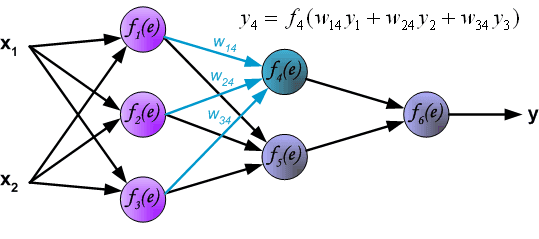


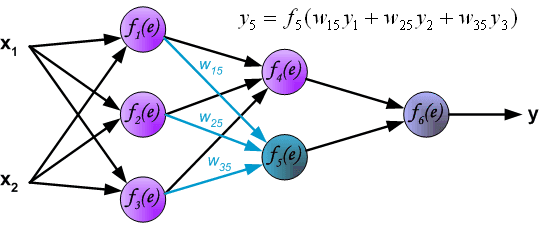


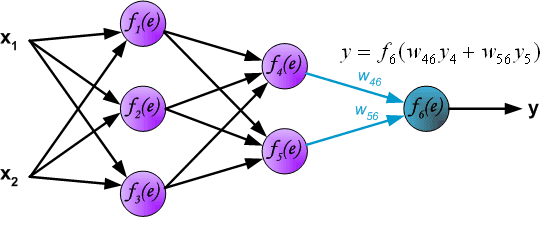


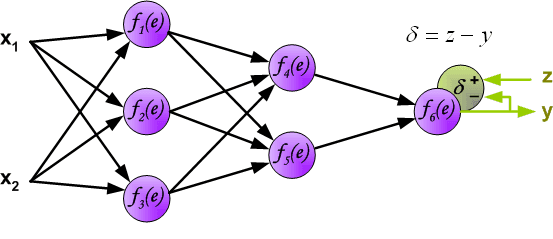


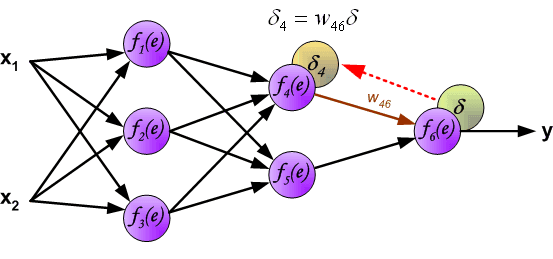


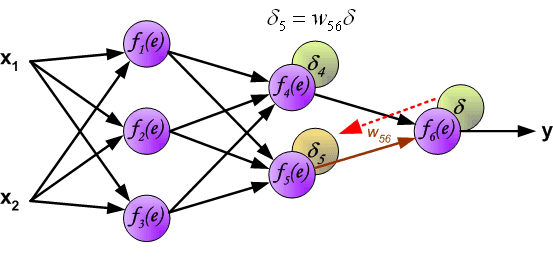


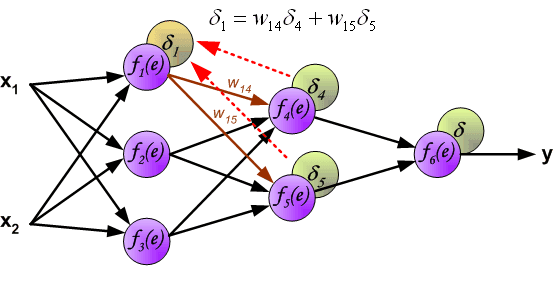


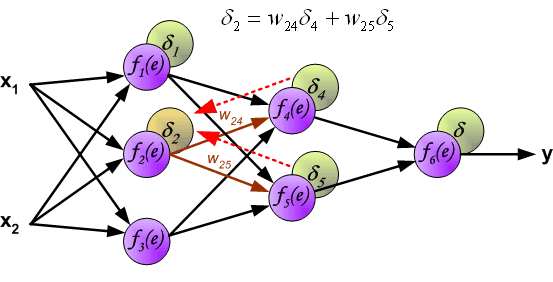


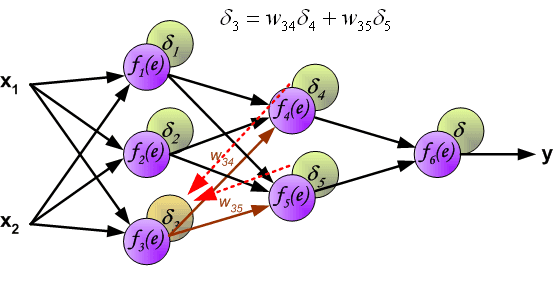


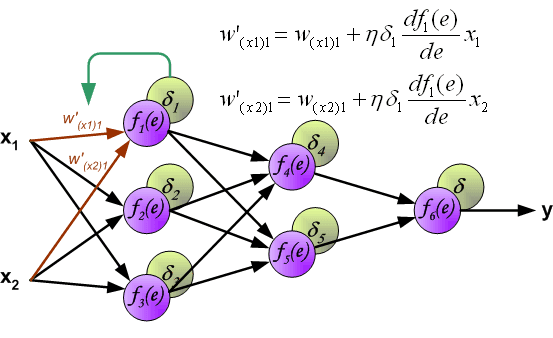


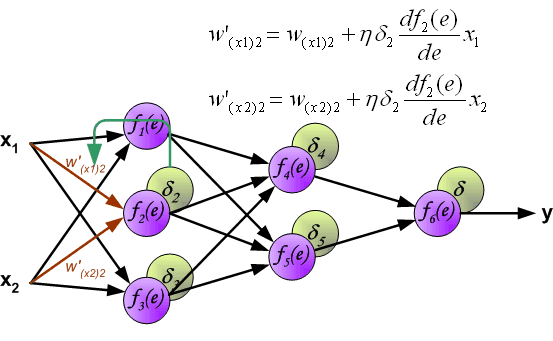


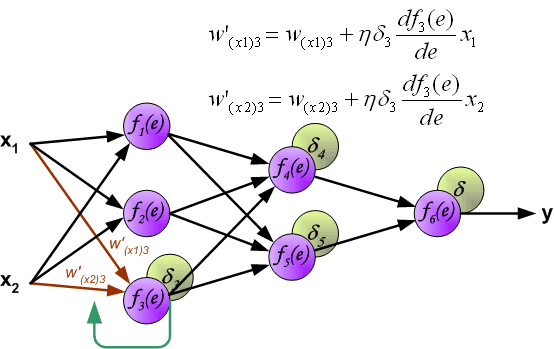


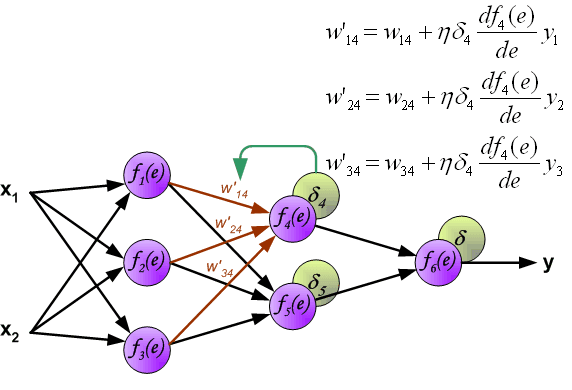


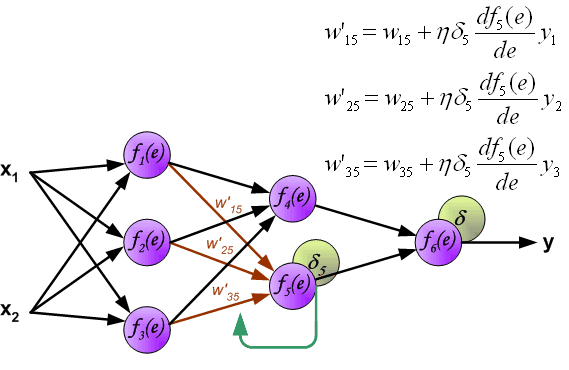


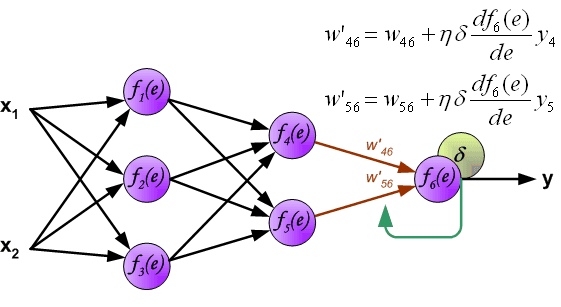












**6. 神经网络中激活函数的真正意义？一个激活函数需要具有哪些必要的属性？还有哪些属是好的属性但不必要的？**

激活函数的意义是“让神经网络具备强大的拟合能力”。激活函数需要具备的属性如下：

1. 非线性：即导数不是常数。这个条件前面很多答主都提到了，是多层神经网络的基础，保证多层网络不退化成单层线性网络。这也是激活函数的意义所在。
2. 几乎处处可微：可微性保证了在优化中梯度的可计算性。传统的激活函数如sigmoid等满足处处可微。对于分段线性函数比如ReLU，只满足几乎处处可微（即仅在有限个点处不可微）。对于SGD算法来说，由于几乎不可能收敛到梯度接近零的位置，有限的不可微点对于优化结果不会有很大影响。
3. 计算简单：正如题主所说，非线性函数有很多。极端的说，一个多层神经网络也可以作为一个非线性函数，类似于把它当做卷积操作的做法。但激活函数在神经网络前向的计算次数与神经元的个数成正比，因此简单的非线性函数自然更适合用作激活函数。这也是ReLU之流比其它使用exp等操作的激活函数更受欢迎的其中一个原因。
4. 非饱和性（saturation）：饱和指的是在某些区间梯度接近于零（即梯度消失），使得参数无法继续更新的问题。最经典的例子是Sigmoid，它的导数在x为比较大的正值和比较小的负值时都会接近于0。更极端的例子是阶跃函数，由于它在几乎所有位置的梯度都为0，因此处处饱和，无法作为激活函数。ReLU在x>0时导数恒为1，因此对于再大的正值也不会饱和。但同时对于x<0，其梯度恒为0，这时候它也会出现饱和的现象（在这种情况下通常称为dying ReLU）。Leaky ReLU和PReLU的提出正是为了解决这一问题。
5. 单调性（monotonic）：即导数符号不变。这个性质大部分激活函数都有，除了诸如sin、cos等。个人理解，单调性使得在激活函数处的梯度方向不会经常改变，从而让训练更容易收敛。
6. 输出范围有限：有限的输出范围使得网络对于一些比较大的输入也会比较稳定，这也是为什么早期的激活函数都以此类函数为主，如Sigmoid、TanH。但这导致了前面提到的梯度消失问题，而且强行让每一层的输出限制到固定范围会限制其表达能力。因此现在这类函数仅用于某些需要特定输出范围的场合，比如概率输出（此时loss函数中的log操作能够抵消其梯度消失的影响）、LSTM里的gate函数。
7. 接近恒等变换：即约等于x。这样的好处是使得输出的幅值不会随着深度的增加而发生显著的增加，从而使网络更为稳定，同时梯度也能够更容易地回传。这个与非线性是有点矛盾的，因此激活函数基本只是部分满足这个条件，比如TanH只在原点附近有线性区（在原点为0且在原点的导数为1），而ReLU只在x>0时为线性。这个性质也让初始化参数范围的推导更为简单。额外提一句，这种恒等变换的性质也被其他一些网络结构设计所借鉴，比如CNN中的ResNet和RNN中的LSTM。
8. 参数少：大部分激活函数都是没有参数的。像PReLU带单个参数会略微增加网络的大小。还有一个例外是Maxout，尽管本身没有参数，但在同样输出通道数下k路Maxout需要的输入通道数是其它函数的k倍，这意味着神经元数目也需要变为k倍；但如果不考虑维持输出通道数的情况下，该激活函数又能将参数个数减少为原来的k倍。
9. 归一化（normalization）：这个是最近才出来的概念，对应的激活函数是SELU，主要思想是使样本分布自动归一化到零均值、单位方差的分布，从而稳定训练。在这之前，这种归一化的思想也被用于网络结构的设计，比如Batch Normalization。

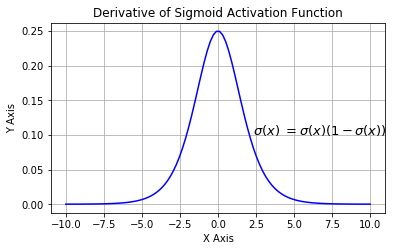
**7. 为什么网络够深(Neurons 足够多)的时候，总是可以避开较差Local Optima？**

**（目前尚未找到比较好的答案）**

理论上而言，参数越多的模型复杂度越高，“容量”也就越大，也就意味着它能完成更复杂的学习任务。多层感知机给我们带来的启示是，神经网络的层数直接决定了它对现实的刻画能力——利用每层更少的神经元拟合更加复杂的函数。但是随着神经网络层数的加深，优化函数越来越容易陷入局部最优解（即过拟合，在训练样本上有很好的拟合效果，但是在测试集上效果很差），并且这个“陷阱”越来越偏离真正的全局最优。利用有限数据训练的深层网络，性能还不如较浅层网络。同时，另一个不可忽略的问题是随着网络层数增加，“梯度消失”现象更加严重。具体来说，我们常常使用Sigmoid作为神经元的输入输出函数。对于幅度为1的信号，在BP反向传播梯度时，每传递一层，梯度衰减为原来的0.25。层数一多，梯度指数衰减后低层基本上接受不到有效的训练信号。

**8. Sigmoid激活函数为什么会出现梯度消失？它的导数在哪一点最大？**

Sigmoid函数有个特点，就是能将负无穷到正无穷的数映射到0和1之间，并且对这个函数求导的结果是。因此两个0到1之间的数相乘，得到的结果就会变得很小了。神经网络的反向传播是逐层对函数偏导相乘，因此当神经网络层数非常深的时候，最后一层产生的偏差就因为乘了很多的小于1的数而越来越小，最终就会变为0，从而导致层数比较浅的权重没有更新，这就是梯度消失。



在x=0处，其导数取得最大值。

**9. ReLU激活函数为什么能解决梯度消失问题？**

Sigmoid函数的gradient会随着增大或减小和消失;

ReLU函数不会， 。

**10. Softmax是和什么损失函数配合使用？它的公式是什么？**

通常配合多项式回归损失函数使用，公式：

**11. 防止过拟合的方法？**

1. Early stopping：一种迭代次数截断的方法来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。
2. 数据集扩增：得到更多的符合要求的数据，即和已有的数据是独立同分布的，或者近似独立同分布的。
3. 正则化：在进行目标函数或代价函数优化时，在目标函数或代价函数后面加上一个正则项，一般有L1正则与L2正则等。
4. Dropout：在训练开始时，随机得删除一些隐藏层神经元，即认为这些神经元不存在，同时保持输入层与输出层神经元的个数不变，然后按照BP学习算法对网络中的参数进行学习更新。
5. batch normalization：因为深层神经网络在做非线性变换前的激活输入值，随着网络深度加深或者在训练过程中，其分布逐渐发生偏移或者变动，之所以训练收敛慢，一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近，所以这导致后向传播时低层神经网络的梯度消失，这是训练深层神经网络收敛越来越慢的本质原因，而BN就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为0方差为1的标准正态分布，其实就是把越来越偏的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，意思是这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。一种解释是这是类似于Dropout的一种防止过拟合的正则化表达方式，所以不用Dropout也能达到相当的效果。另外调参过程也简单多了，对于初始化要求没那么高，而且可以使用大的学习率等。

**12. dropout的原理是什么？**

Dropout的原理如下图所示。

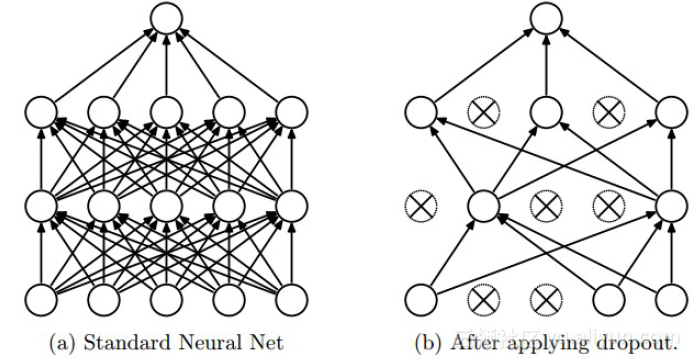
在训练模型阶段，没有dropout的网络计算公式为：

采用dropout的网络计算公式为：

上面公式中函数是为了生成概率向量，也就是随机生成一个0、1的向量。

注意： 经过上面屏蔽掉某些神经元，使其激活值为0以后，我们还需要对向量进行缩放，也就是乘以。如果你在训练的时候，经过置0后，没有对进行缩放，那么在测试的时候，就需要对权重进行缩放，操作如下：

在测试模型阶段，每一个神经单元的权重参数要乘以概率：



Dropout的作用如下：

1. 取平均的作用：先回到正常的模型（没有dropout），我们用相同的训练数据去训练5个不同的神经网络，一般会得到5个不同的结果，此时我们可以采用“取均值”或者“多数取胜的投票策略”去决定最终结果。这种“综合起来取平均”的策略通常可以有效防止过拟合问题。因为不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消。dropout掉不同的隐藏神经元就类似在训练不同的网络（随机删掉一半隐藏神经元导致网络结构已经不同)，整个dropout过程就相当于 对很多个不同的神经网络取平均。而不同的网络产生不同的过拟合，一些互为“反向”的拟合相互抵消就可以达到整体上减少过拟合。
2. 减少神经元之间复杂的共适应关系： 因为dropout程序导致两个神经元不一定每次都在一个dropout网络中出现。（这样权值的更新不再依赖于有固定关系的隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况）。 迫使网络去学习更加鲁棒的特征 （这些特征在其它的神经元的随机子集中也存在）。换句话说假如我们的神经网络是在做出某种预测，它不应该对一些特定的线索片段太过敏感，即使丢失特定的线索，它也应该可以从众多其它线索中学习一些共同的模式（鲁棒性）。（这个角度看 dropout就有点像L1，L2正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高）

**13. 神经网络缓解陷入局部最优的方法。**

启发式算法中，局部最优值的陷入无法避免。启发式，本质上是一种贪心策略，这也在客观上决定了不符合贪心规则的更好（或者最优）解会错过。

简单来说，避免陷入局部最优就是两个字：随机。

1. 冲量。人如其名，本次前进的步伐，根据上一次的步伐，适当调大，好比从高处降落的石头，会更有机率跨过一些小坑。
2. 使用随机梯度下降代替真正的梯度下降。可以这样理解，每次针对单个数据样例进行摸索前进时，本质上是在一个样例形成的误差曲面上摸索前进，而每个样例的曲面大体类似，又不尽相同，当你掉入一个坑里时，往往能被别的曲面拽出来。
3. 用不同的初始权值进行训练。这个很好理解吧，假定误差曲面是个坑坑洼洼的曲面，我们尝试第一次降落到随机的不同的点，然后再开始摸索前进，也许会有运气好的一次，能够不落在某个小坑附近。或者你也可以保留所有的网络，形成一个网络委员会，每个网络的输出都用来参考。