Lecture 12: Neural Network

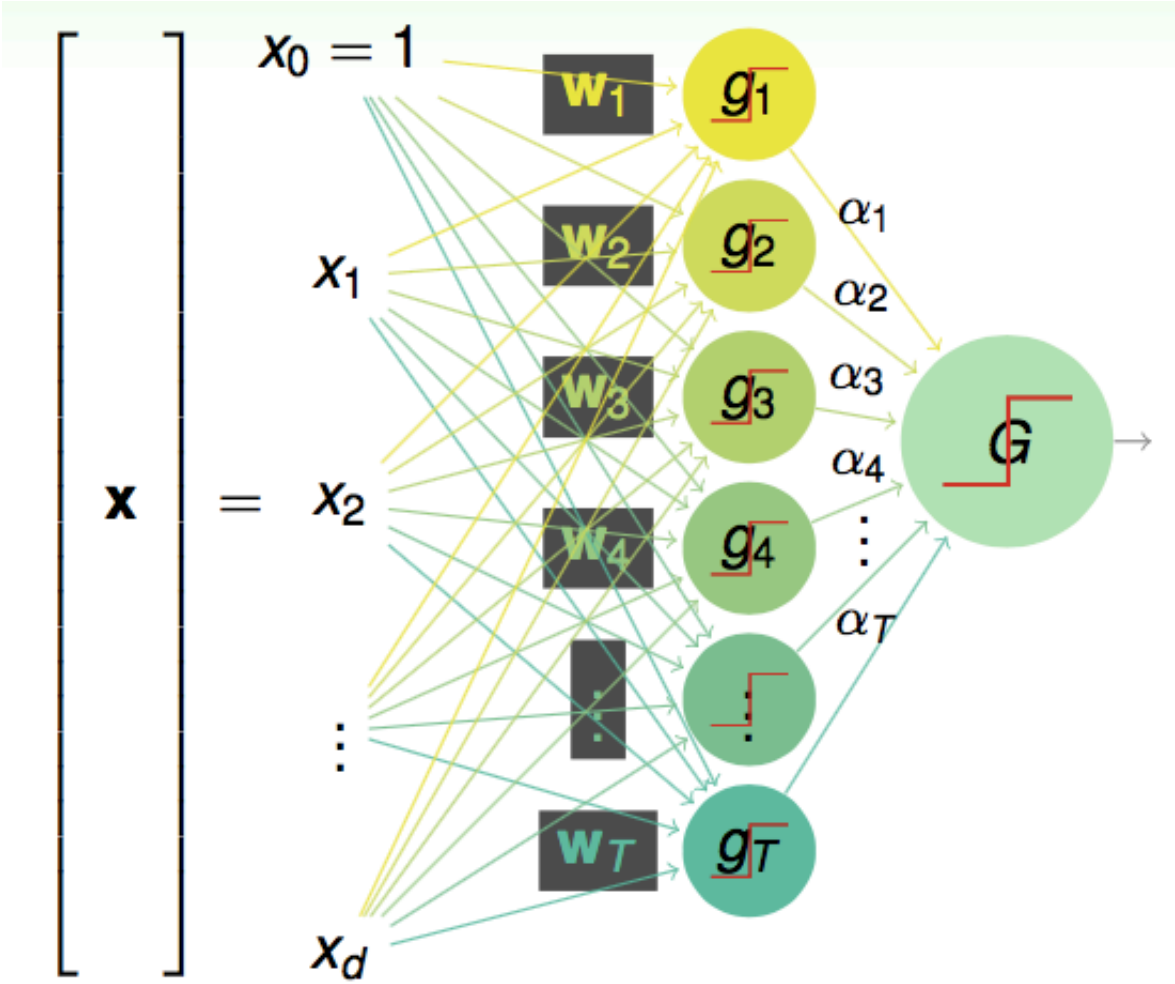
> 课件链接：Hsuan-Tien Lin - neural network
>
> **Neural Network(神经网络)**
>
> - Motivation：发展背景
> - Neural Network Hypothesis：神经网络的假说形式
> - Neural Network Learning：神经网络的学习算法
> - Optimization and Regularization：优化与正则化

1. Motivation：发展背景

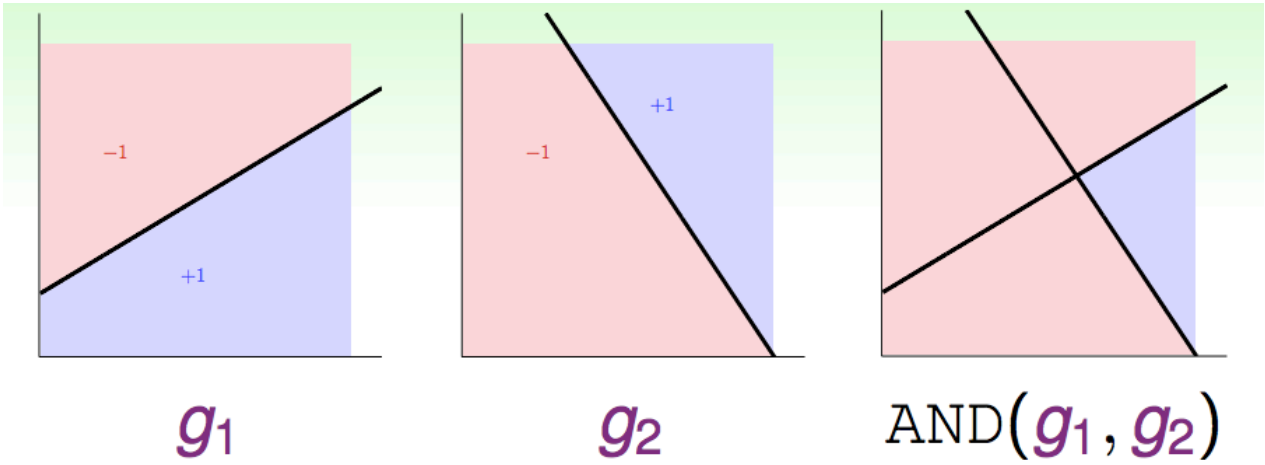
将一系列Perceptrons聚合在一起，得到的聚合模型如下：

$$\begin{aligned} G(\mathbf{x}) &= sign\left(\sum_{t=1}^{T}\alpha_t g_t(\mathbf{x})\right) \\ &= sign\left(\sum_{t=1}^{T}\alpha_t sign(\mathbf{w}_t^T\mathbf{x})\right) \end{aligned}$$

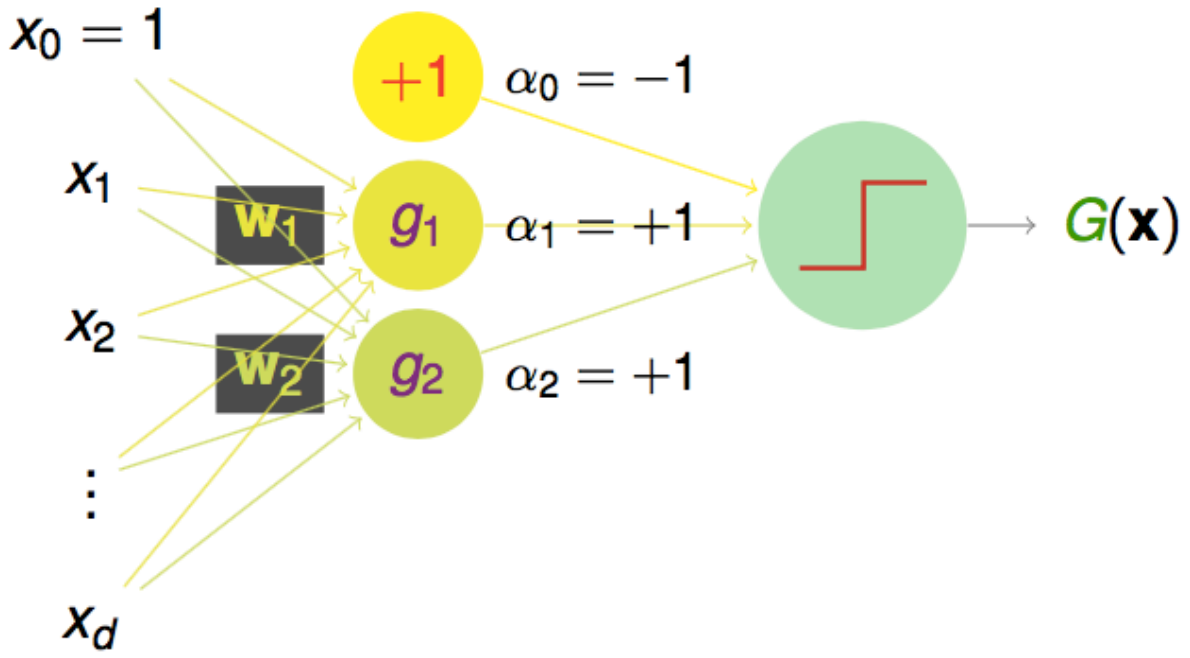
注意，这里我们有两层权重：$\mathbf{w}_t$与α_t；也有两层sign函数：g_t与G中。如下图所示：



下面介绍线性聚合感知器模型能够拟合的一些边界。首先，它能够将两个感知机以AND的逻辑运算形式聚合在一起，拟合出AND型边界，如下图所示：

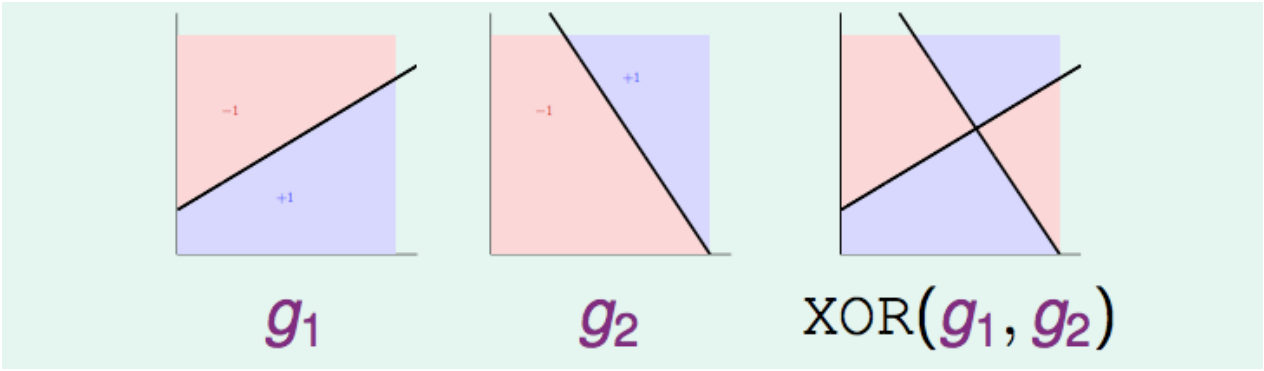


为此，一种可行的方法是：取$\alpha_0 = -1, \alpha_1 = +1, \alpha_2 = +1$：



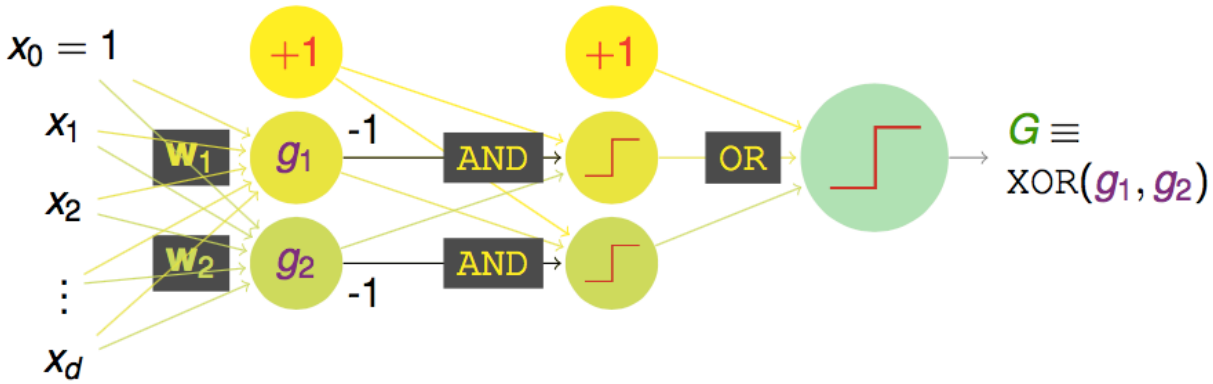
类似的，线性聚合感知器模型还能够拟合OR与NOT型边界。通常，随着聚合数量的增加，线性聚合感知器模型的能力会不断增强，其VC维会趋向无穷大。这样看来，只要将足够多的感知器聚合在一起，任何边界都可以拟合。

然而，线性聚合感知器模型却连一种比较简单的边界都无法拟合——XOR型：



为何不能产生XOR型边界？如果将g_1与g_2看做特征转换$\Phi(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}))$，转换后的资料在新的空间里仍然不是线性可分的，所以没有办法找到100%正确的分类器，因此无法拟合该边界。

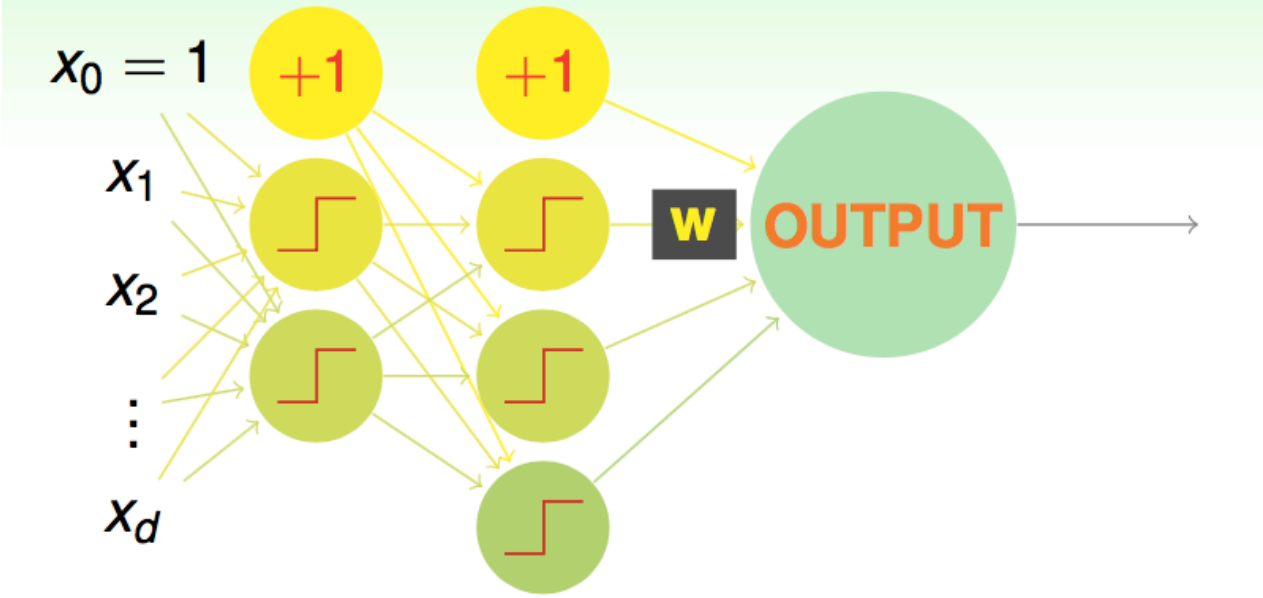
如果在新的空间里，资料仍然线性不可分，那么需要进行进一步的转换，直至将资料转到某一个线性可分的空间里——这就对应到"多层的(Multi-Layer)"感知器：



综上，我们从简单的perceptron，到稍微复杂的aggregation of perceptrons，即感知器的聚合，再到接下来更加强大的multi-layer perceptrons，即多层感知器，模型的复杂度越来越高，拟合能力越来越强。

2. Neural Network Hypothesis：神经网络的假说形式

我们可以将神经网络模型用如下的图示表示：



神经网络从输入input出发，经过一层层的运算，最后得到输出output。如果我们将除最后一层外的其他层运算看做"转换(transformation)"，那么神经网络也就是将原始资料进行一次次的特征转换，最后用一个权重向量计算一个"分数"——一个简单的linear model。因此，在最后一层计算output的时候，我们可以使用我们学过的任何linear model——linear classification、linear regression、logistic regression。本章我们使用基于平方误差的linear regression，即直接输出"分数"。

刚刚提到，我们可以将中间层看做是一次次特征转换(transformation)。每层的每个单元，都会将计算出的分数，通过某一个特定的函数(transformation function)，转化为该单元的输出。一些可能的转换函数有：

- 线性函数：不常用——如果使用线性函数，那么整个神经网络模型仍然是线性的，完全等价于一个线性模型——没有必要搭建神经网络；
- 阶梯函数：离散的，不易最优化；
- **tanh函数**：hyperbolic tangent函数，容易优化：

$$\begin{aligned} tanh(s) &= \frac{exp(s) - exp(-s)}{exp(s) + exp(-s)} \\ &= 2\theta(2s) - 1 \end{aligned}$$

小结：本章讨论的神经网络，**最后一层直接输出"分数"，转换函数选择tanh函数。**

神经网络的hypothesis

一个hypothesis，即确定了相关参数后，对于任何一个input输入x，都有一个确定的ouput输出y。因此，一个hypothesis对应一系列参数。神经网络中的参数为：

$$w_{ij}^{(l)}$$

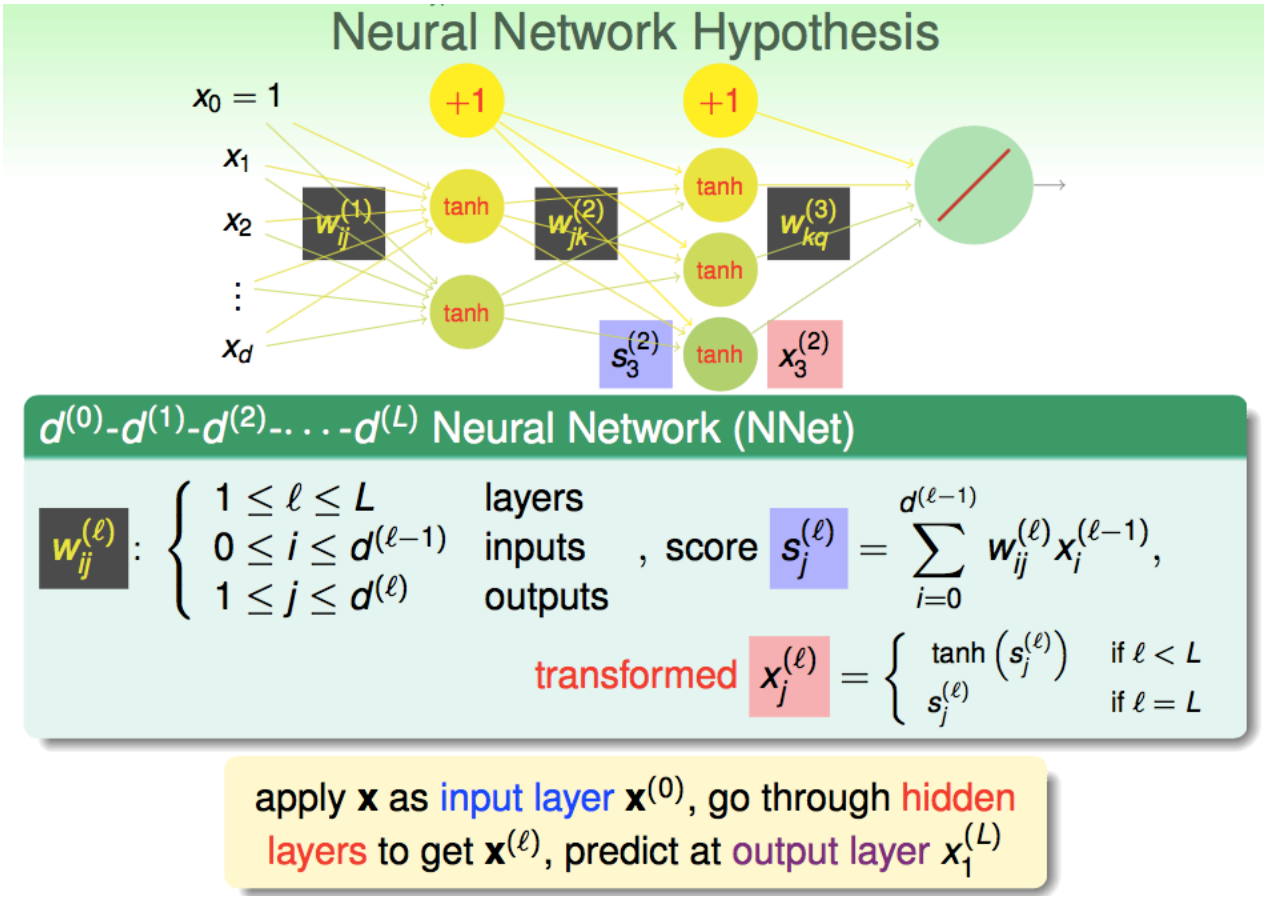
该参数表示"**第l层中连接上一层第i个单元与本层第j个单元的权重**"：

- 神经网络总层数记做L，输入层为第0层，以此类推，输出层为第L层(中间层又被称为隐藏层，hidden layer)；层数信息均标记为上标；
- 第l层的单元数量为$d^{(l)}$，注意这里没有将偏置单元+1计算在内；
- 因此，对于$w_{ij}^{(l)}$，

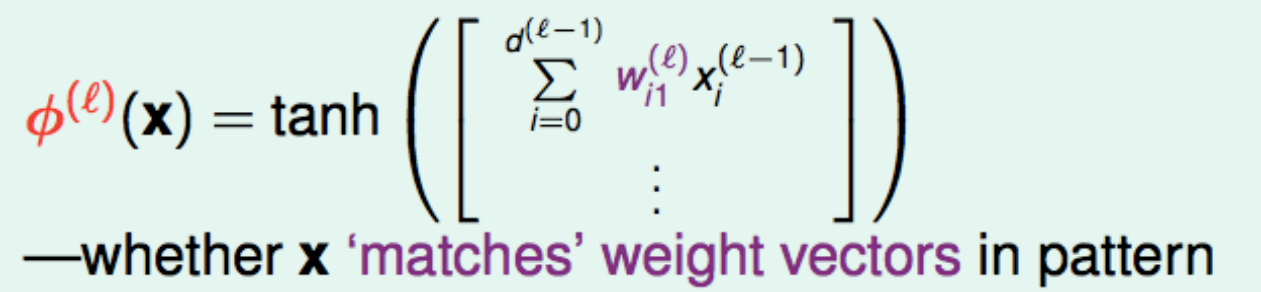
$$1 \leq l \leq L$$
$$0 \leq i \leq d^{(l-1)}$$
$$1 \leq j \leq d^{(l)}$$

- s表示某单元计算的"分数"，x表示单元的输出，即"分数"s经过转换函数后的结果。例如：$s_3^{(2)}$表示第2层第3个单元计算的分数，$x_3^{(2)}$表示第2层第3个单元计算的分数经过转换函数后的结果，即$x_3^{(2)} = tanh\ s_3^{(2)}$，

$$s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)}\, x_i^{(l-1)}$$



除输出输出层外，中间隐藏层可以看做是利用在数据中学到的权重，对数据进行一次次的特征转换。而每次特征转换的每个维度的计算，是将上一层的输出$\mathbf{x}$与某个权重向量$\mathbf{w}$进行内积——可以看做是进行某种相似性的确认：



因此，神经网络能够从资料中萃取出一些模式(pattern extraction)，这体现在它对于权重的学习上面。

最后附一道例题以检测对于神经网络结构的理解：对于3-5-1的神经网络，共有多少权重参数$w_{ij}^{(l)}$？

- 第一层有：$(3+1)\times 5=20$；
- 第二层有：$(5+1)\times 1=6$；
- 共26个。

3. Neural Network Learning：神经网络的学习算法

我们的目标是：学习所有的权重参数$\{w_{ij}^{(l)}\}$，以最小化$E_{in}\left(\{w_{ij}^{(l)}\}\right)$。我们将单一样本n的平方误差记做：

$$e_n=\left(y_n-NNet(\mathbf{x_n})\right)^2$$

如果我们能够计算：

$$\frac{\partial e_n}{\partial w_{ij}^{(l)}}$$

那么我们就可以用SGD进行梯度下降，以获得最佳的权重参数。因此，学习问题的核心转化成如何高效地计算样本误差对于每一个权重的偏微分。

我们先以输出层为例，计算$\frac{\partial e_n}{\partial w_{i1}^{(L)}}$：

$$e_n=(y_n-s_1^{(L)})^2=\left(y_n-\sum_{i=0}^{d^{(L-1)}} w_{i1}^{(L)}x_i^{(L-1)}\right)^2$$

因此，不难计算：

$$\begin{aligned}\frac{\partial e_n}{\partial w_{i1}^{(L)}}&=\frac{\partial e_n}{\partial s_1^{(L)}}\cdot\frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}}\\&=-2(y_n-s_1^{(L)})\cdot(x_i^{(L-1)})\end{aligned}$$

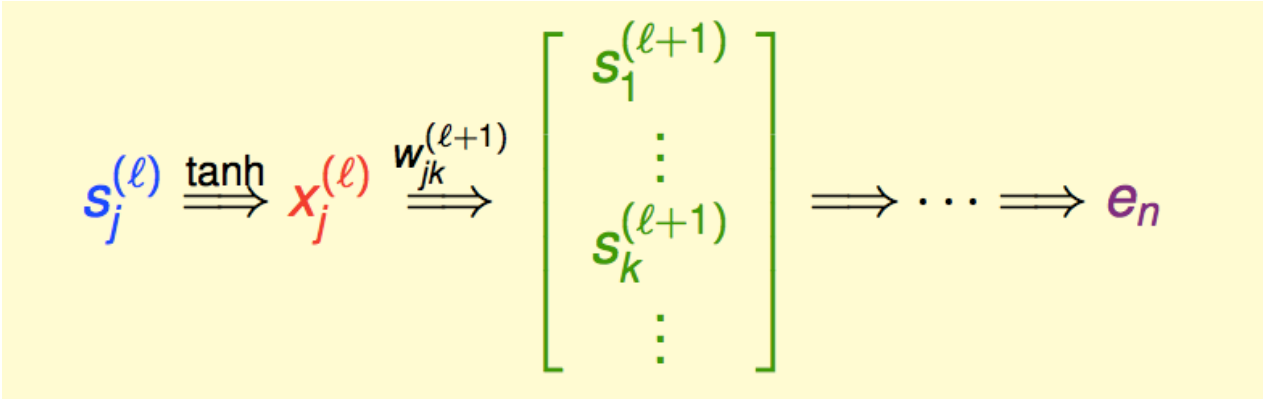
这里，我们找到了$s_1^{(L)}$作为"中间桥梁"，即**权重对应的"分数"**。**推广到一般**，有：

$$\frac{\partial e_n}{\partial w_{ij}^{(l)}} = \frac{\partial e_n}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$
$$= \delta_j^{(l)} \cdot (x_i^{(l-1)})$$

上式中我们将误差对于某个"分数"的偏微分记做$\delta_j^{(l)}$。通过对于输出层权重的计算，我们知道了：

$$\delta_1^{(L)} = -2(y_n - s_1^{(L)})$$

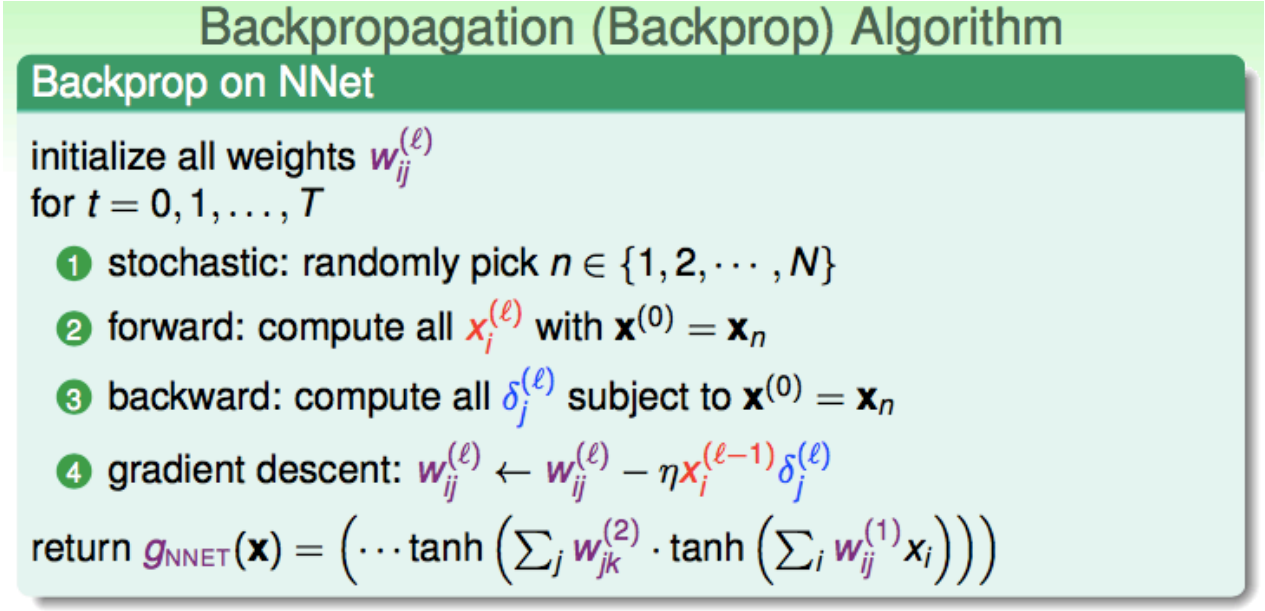
对于其他的$\delta_j^{(l)}$，还需要进一步的分析。下图表现了某个"分数"与误差间的关系：



易知：

$$\delta_j^{(l)} = \frac{\partial e_n}{\partial s_j^{(l)}}$$
$$= \sum_{k=1}^{d^{(l+1)}} \frac{\partial e_n}{\partial s_k^{(l+1)}} \cdot \frac{\partial s_k^{(l+1)}}{\partial x_j^{(l)}} \cdot \frac{\partial x_j^{(l)}}{\partial s_j^{(l)}}$$
$$= \sum_k \left(\delta_k^{(l+1)}\right) \cdot \left(w_{jk}^{(l+1)}\right) \cdot \left(tanh'(s_j^{(l)})\right)$$

可见，误差对于某个"分数"的偏微分，可以通过下一层的误差对"分数"的偏微分求出来——$\delta_j^{(l)}$ can be computed backwards from $\delta_k^{(l+1)}$。这样，我们就可以根据$\delta^{(L)}$，求出所有的$\delta^{(L-1)}$，再求出$\delta^{(L-2)}$ ……一直求到$\delta^{(1)}$。这样，最开始的误差对于某个权重参数的偏微分也就迎刃而解了。因此，有如下的**反向传播算法(Backpropagation)**，用以进行参数更新：



有时，第1步到第3步常常并行执行多次，然后取$x_i^{(l-1)}\delta_j^{(l)}$的平均值来做第4部的更新，这样的方法叫做mini-batch——既不是只用一个点更新(SGD)，也不是用所有样本点更新。

4. Optimization and Regularization：优化与正则化

神经网络的训练实质是最小化下面的E_{in}：

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \text{err}\left(\left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_{n,i}\right)\right)\right), y_n\right)$$

该代价函数不是凸函数，因此是**非凸函数的最优化问题**，很难找到全局最优解。每次迭代过后得到的可能只是局部最优解。并且，**不同的初始值，经过迭代后可能得到不同的局部最优解**。那么如果选择初始值？建议：

- 大的初始权重不好。因为w比较大时，分数也会比较大；而分数要经过tanh函数；tanh函数在自变量比较大的地方，梯度很小；因此，w比较大时，w即使变化很多，但是因为经过了tanh函数，其导致的代价函数的变化也会大大折扣，具体表现为梯度很小，也就导致每次更新迈的步子很小——卡在初始值附近；
- 因此，初始权重应该小一点，且最好随机。

神经网络模型的VC维

当使用tanh作为转换函数时，神经网络的VC维大概是：

$$d_{VC} = O(VD)$$

其中，V是神经元的数量，D是权重参数的数量。当V很大时候(D也会很大)，神经网络几乎可以拟合任何边界，但同样容易**过拟合**。

神经网络的正则化

一个基础的思路是，加入weight-decay(L2)正则项：

$$\Omega(\mathbf{w}) = \sum \left(w_{ij}^{(l)} \right)^2$$

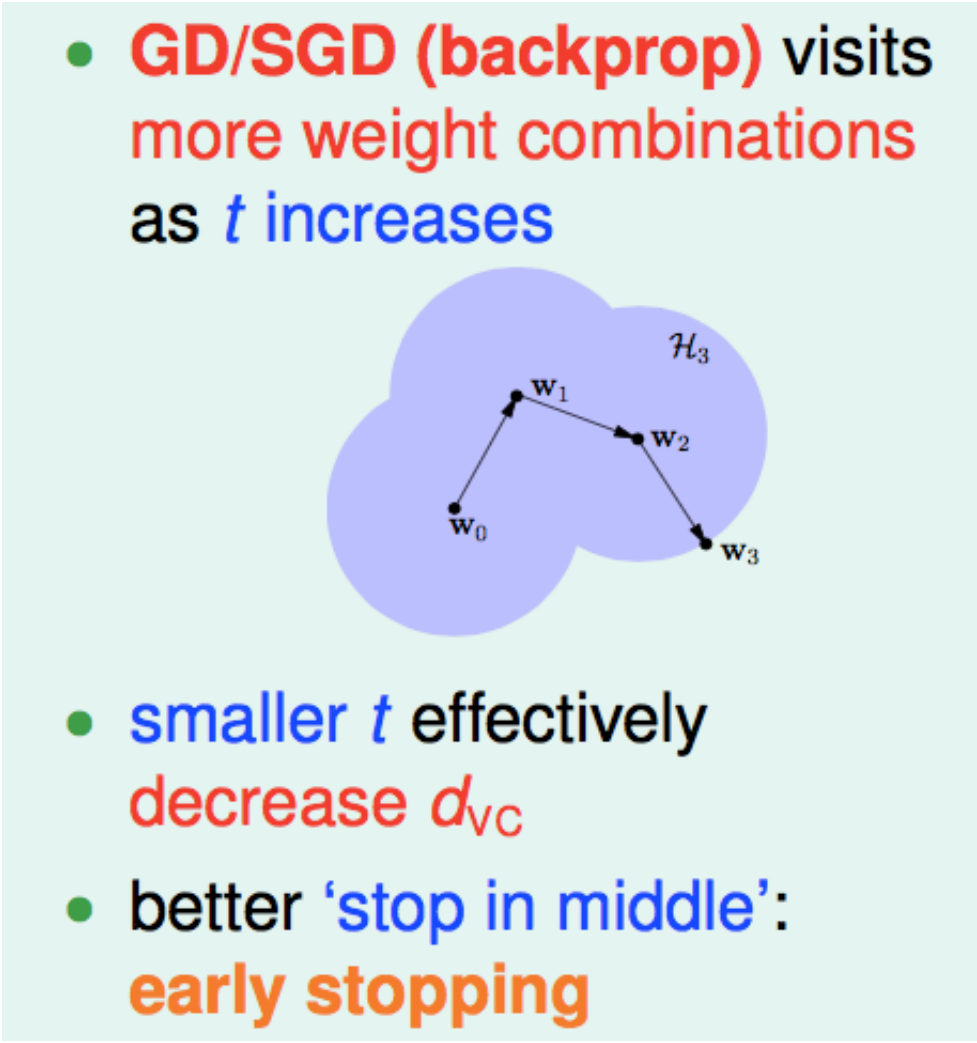
但L2正则化项只是在对所有的w进行几乎等比例的缩小动作(shrink)——原来比较大的w，缩小的多一点；原来比较小的w，缩小的少一点。这样得到的正则化后的权重依然是很dense的。然而，我们希望多一点sparse，即w=0的正则化效果。因此可以考虑使用L1正则项。然而L1正则项无法微分，因此我们使用所谓**weight-elimination(scaled L2)正则项**：

$$\sum \frac{(w_{ij}^{(l)})^2}{1 + (w_{ij}^{(l)})^2}$$

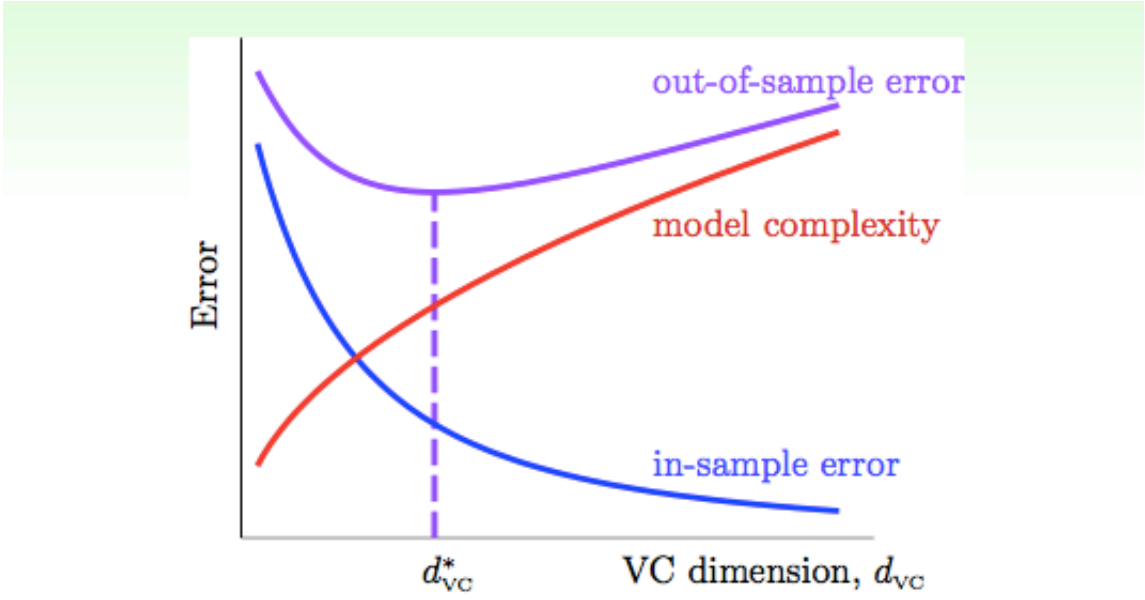
该正则项能够使所有w进行同等尺度的缩小动作，而非同等比例的缩小——大的w，中等缩小；小的w，也是中等缩小——这样可能将原本比较小的w"逼"至0。

另一种神经网络中使用的正则化方法是**早停法(Early Stopping)**。

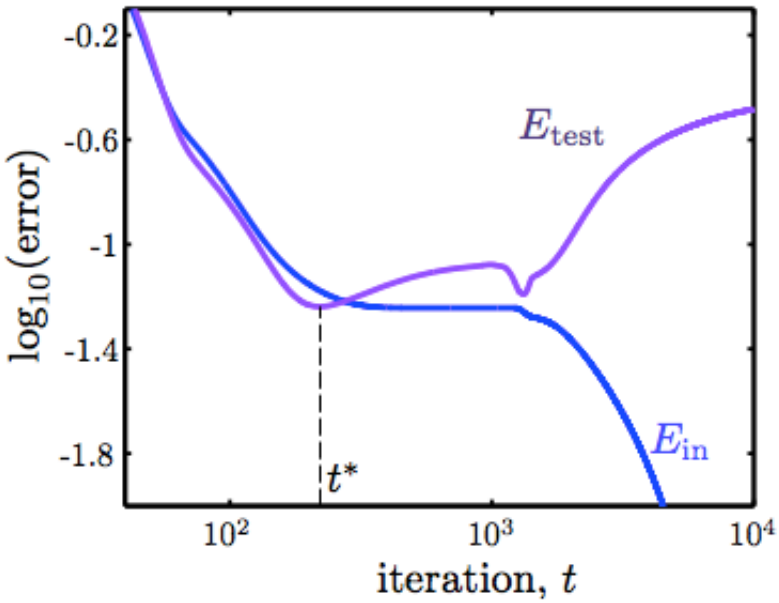
梯度下降算法实质在每一步考虑方圆某个范围内的w。因此，迭代的步数越多，算法看过的w也就越多；迭代的步数越少，算法看过的w也就越小。所以，因为梯度下降这样的机制，虽然有无限多个w可供选择，但算法每一次仅在有限的范围内进行选择——**有效VC维随着迭代次数的增大而增大**。



因此，为了避免过拟合，可以考虑在"中途"停止迭代，如下图所示：



(d^*_{VC} in middle, remember? :-))



具体的t^*可以通过validation的方法得到。

5. Summary

- 神经网络NN的前身是aggregation of perceptrons，它能够解决AND、OR、NOT的边界拟合问题，但无法解决XOR的边界拟合问题。为了解决XOR问题，需要更多的转换，也就是需要叠加更多的layers，构建multi-layer perceptrons——神经网络的雏形。
- 可以将神经网络看做**"一系列特征转换+最后一层的线性模型"**：本章选择tanh作为转换函数，线性模型选择基于平方误差的线性回归模型。
- 神经网络的hypothesis由所有权重参数$w_{ij}^{(l)}$决定；需要特别注意i、j、l的取值范围，并理解各自的含义。
- 神经网络的训练使用梯度下降法，因此需要计算误差对于每个权重的偏导数$\frac{\partial e_n}{\partial w_{ij}^{(l)}}$。我们选择该权

重所对应的"分数"为"中间变量"，将$\frac{\partial e_n}{\partial w_{ij}^{(l)}}$写作$\delta_j^{(l)}$与$x_i^{(l-1)}$的乘积。而前者可以通过反向递推得到。因此，有**反向传播算法**。

- 神经网络训练时，初始权重应该设置地随机一点、小一点，避免训练初期就卡在某一位置。
- 因为神经网络的拟合能力很强，因此需要使用正则化以避免过拟合。常用的正则化方法有：①增加weight-elimination(scaled L2)正则项；②早停法(Early Stopping)。