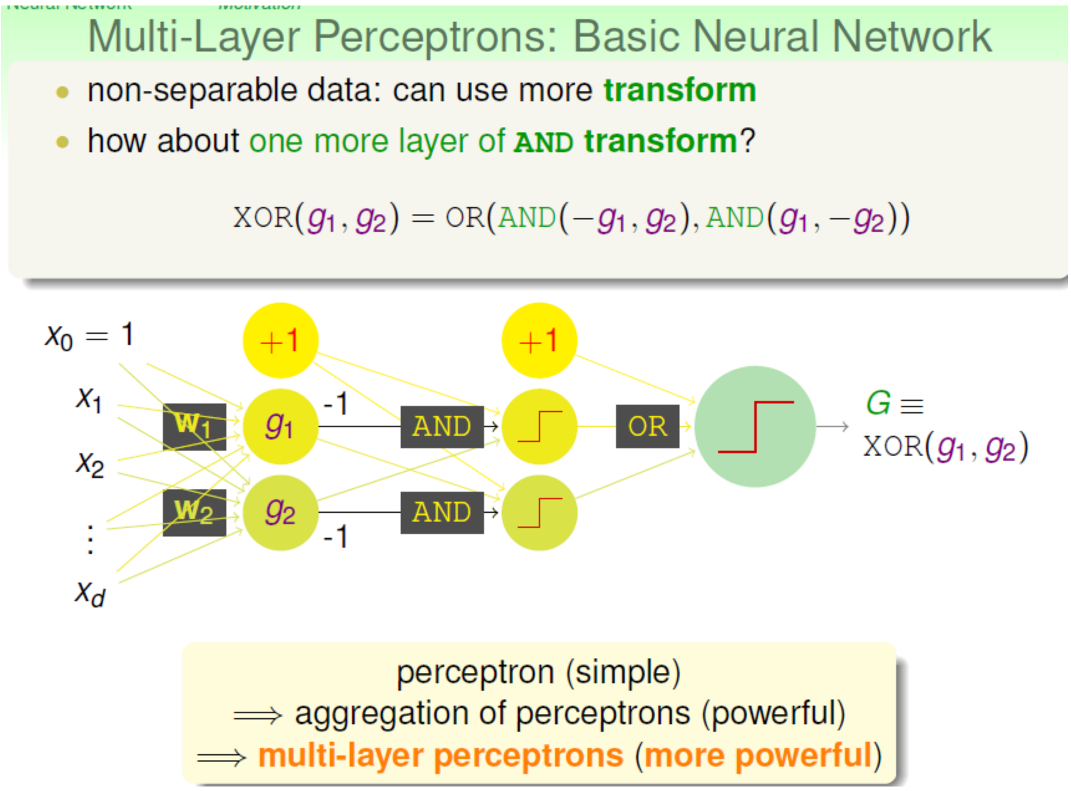
Neural Network

从这个笔记开始我们开始接触extraction model，顾名思义就是我们希望对于给定的数据本身通过一些无监督的方式进行一些特征提取。

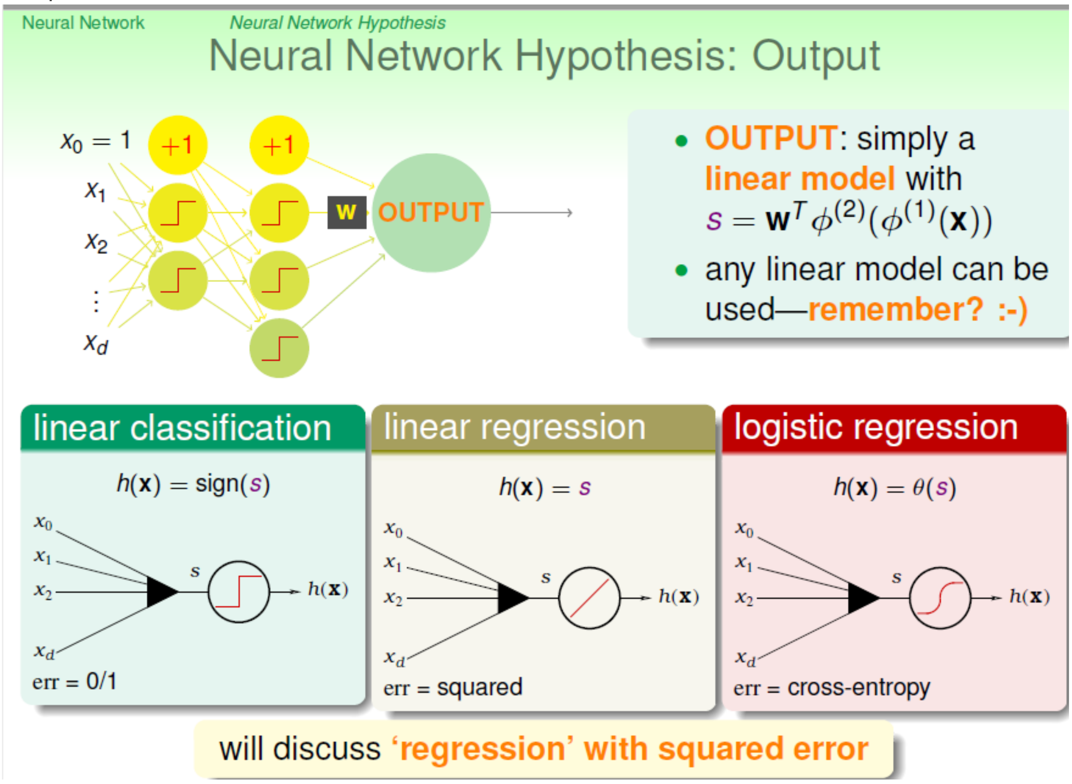
一、Motivation

1. 感知机模型：perceptrons' aggregation
2. 多层感知机模型：基本NN

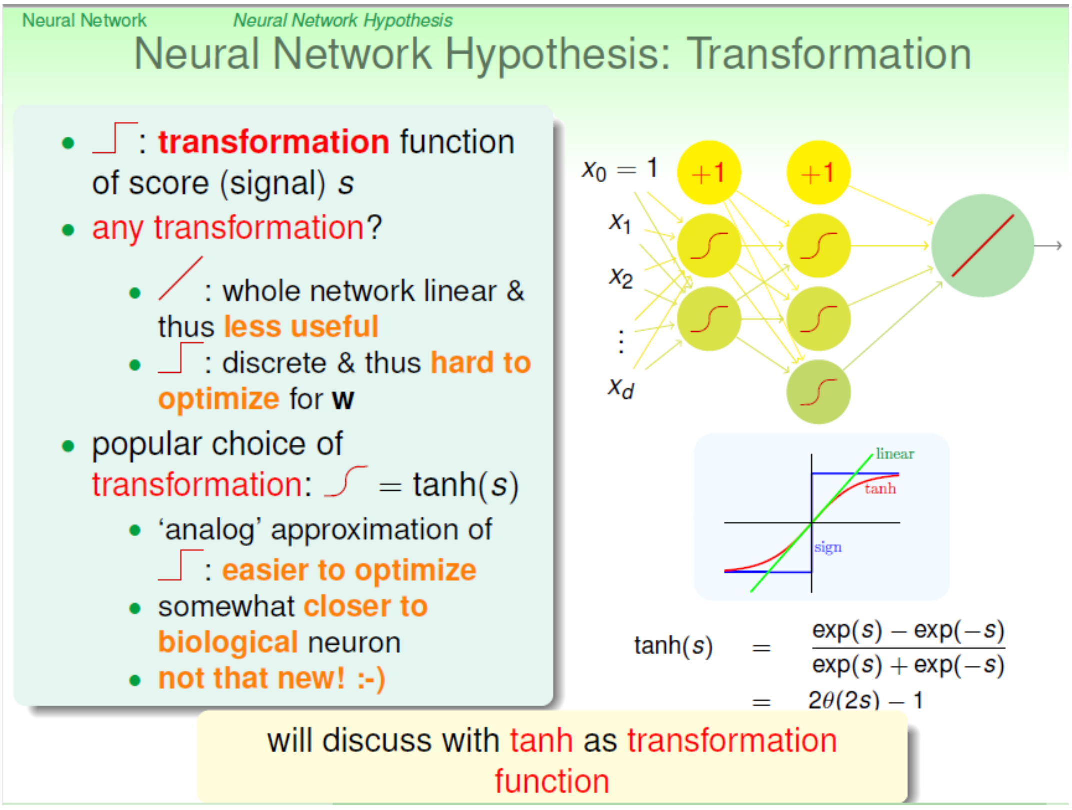


二、Neural Network Hypothesis

1. Output view:



2. Transformation view:



其中，tanh是比较常用的transformation function

3. 我们对于每一层进行一个总结，第l层的权重可以写作

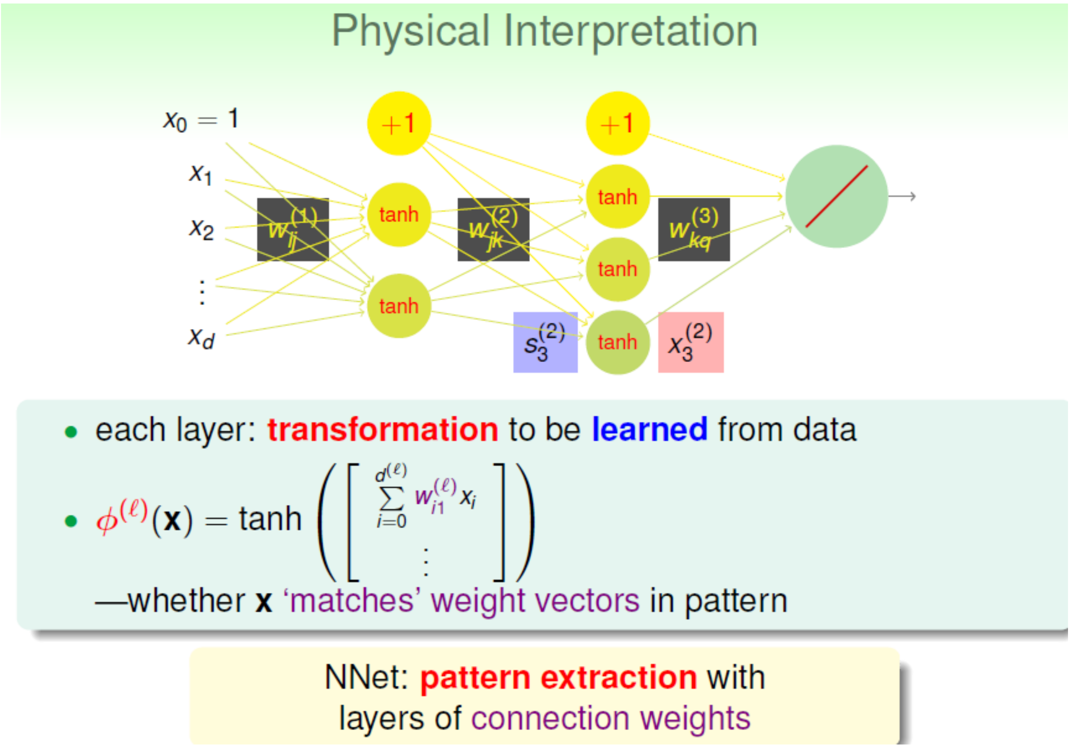
$$w_{ij}^{(l)}: \begin{cases} 1 \le l \le L \\ 0 \le i \le d^{(l-1)} \\ 1 \le j \le d^{(l)} \end{cases}$$

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

$$transformed \ \ x_j^{(l)} \begin{cases} tanh(s_j^{(l)}) & if\ l < L \\ s_j^{(l)} & if\ l = L \end{cases}$$

每一层的权重分为两个部分，i代表输入层，j代表输出层。对于这一层的输出就是输入层权重与输入的线性组合，再通过转换函数就可以得到下一层的输入。这里最后一层就不需要non-linear transformation了。

4. 一些解释：每一层相当于是对于学到特征的一种转换，类似于模式提取



三、Neural Network Learning

1. 学习weights的方法：考虑让E_{in}最小的所有w
2. 反向传播算法：

   首先考虑如果求梯度。。。

Computing $\frac{\partial e_n}{\partial w_{i1}^{(L)}}$ (Output Layer)

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

specially

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

generally

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{ij}^{(\ell)}} \\ = & \frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} \\ = & \delta_j^{(\ell)} \cdot \left(x_i^{(\ell-1)}\right) \end{aligned}$$

$\delta_1^{(L)} = -2\left(y_n - s_1^{(L)}\right)$, how about **others**?

Computing $\delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}}$

$$s_j^{(\ell)} \xrightarrow{\tanh} x_j^{(\ell)} \xrightarrow{w_{jk}^{(\ell+1)}} \begin{bmatrix} s_1^{(\ell+1)} \\ \vdots \\ s_k^{(\ell+1)} \\ \vdots \end{bmatrix} \Longrightarrow \cdots \Longrightarrow e_n$$

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

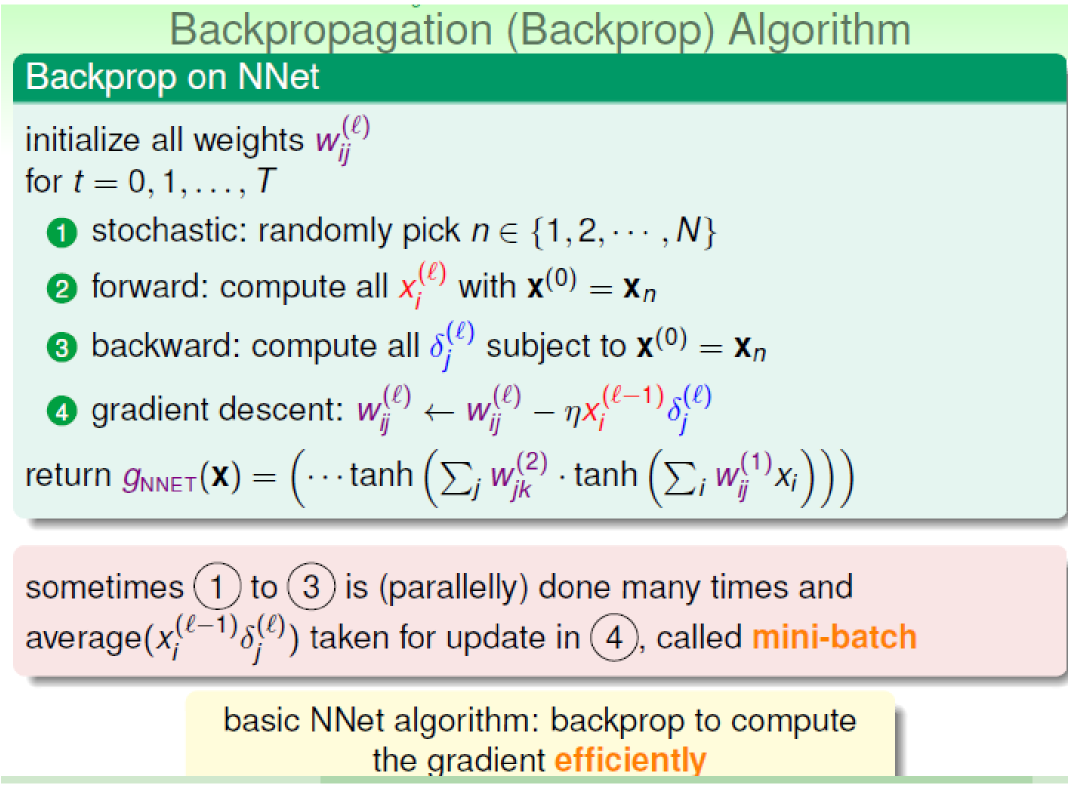
$\delta_j^{(\ell)}$ can be computed **backwards** from $\delta_k^{(\ell+1)}$

$\delta_j^{(l)}$: *downstream gradient*

$\delta_k^{(l+1)}$: *upstream gradient*

因此我们只需要计算transform层对于本层输入的计算，再累和即可。这就是反向传播的本质（源自 cs231n）

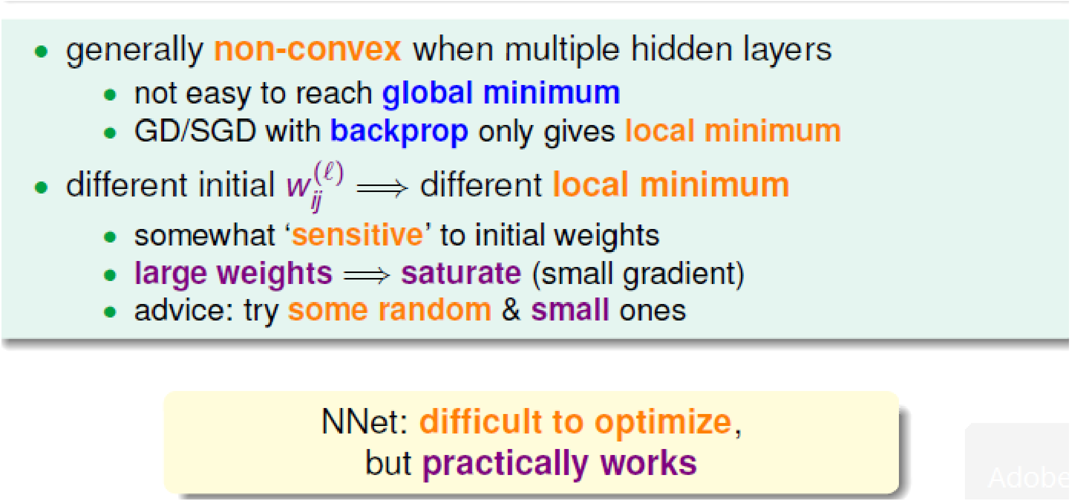
3. 反向传播算法的流程：



- 随机选择一个n
- 前向传播
- 反向传播
- 梯度下降 SGD

四、Optimization and Regularization

1. 最优化：往往很难达到global minimum，通过GD/SGD的方法一般只能给到局部最优解，其次，对于权重的初始化很有讲究，对于比较大的权重会又saturate得现象，也就是说梯度区域平缓，很难optimize，因此一般采取较小权重进行初始化，经常用的有高斯初始化（另外有何凯明得初始化等等，Andrew Ng介绍过一些，往往都是在特定的XXNet使用。不赘述。）总而言之，神经网络很难优化，但是效果不错



这里有一点十分重要，就是不能让权重都初始化相等或者都是0之类的sb选择，理由十分简单：当你forward pass再backward pass的时候，来自于初始权重的upstream实际上时一样的，所以你对于整个layer的影响都是等效的，这就失去了NN的优势（作业3里面有详细的推导）

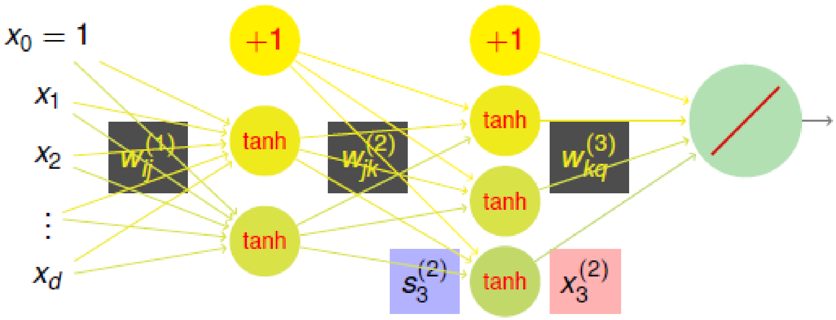
2. VC-dim的解释：

$$d_{VC} = O(VD), \quad V = \#of\ neurons, \quad D = \#of\ weights$$

- 优点：可以模拟一切模型，只要V足够大，足够deep
- 缺点：VC-bound的约束力差于是就容易过拟合

VC Dimension of Neural Network Model

roughly, with tanh**-like transfer functions**:
$d_{VC} = O(VD)$ where $V = \#$ of neurons, $D = \#$ of weights



- pros: can **approximate 'anything'** if enough neurons (V large)
- cons: can **overfit** if too many neurons

3. Regularization：

我们希望D足够小，这样的话我们的VC-dim就会小不少，因而我们希望weight matrix是一个sparse matrix，L1 Regularization 是一个很好的选择，L1正则化对于稀疏性会有很好的效果 在这里我们选择weight-elimination regularizer

Regularization for Neural Network

basic choice:

old friend weight-decay (L2) regularizer $\Omega(\mathbf{w}) = \sum \left(w_{ij}^{(\ell)}\right)^2$

- 'shrink' weights:
  large weight $\rightarrow$ large shrink; small weight $\rightarrow$ small shrink
- want $w_{ij}^{(\ell)} = 0$ (sparse) to effectively **decrease** d_{VC}
  - L1 regularizer: $\sum \left|w_{ij}^{(\ell)}\right|$, but **not differentiable**
  - weight-elimination ('scaled' L2) regularizer:
    large weight $\rightarrow$ median shrink; small weight $\rightarrow$ median shrink

weight-elimination regularizer: $\sum \dfrac{\left(w_{ij}^{(\ell)}\right)^2}{1+\left(w_{ij}^{(\ell)}\right)^2}$

多余的一项梯度：$2w_{ij}^{(\ell)} / \left(1 + \left(w_{ij}^{(\ell)}\right)^2\right)^2$

其实还有另一种选择，就是Early Stopping因为我们知道d_{VC}适中的时候其实E_{out}是最小的。至于如何选择stop的位置就需要通过validation了。