

# Chapter 6 Deep Learning

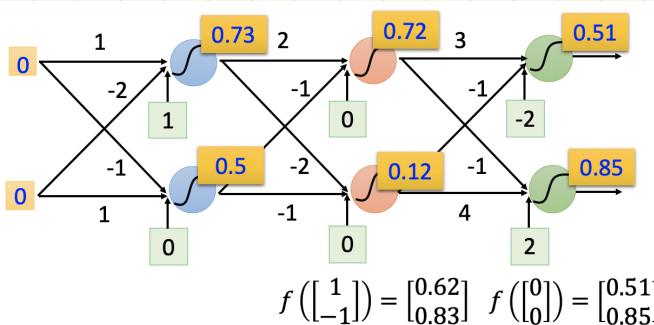
## 1. 概述

### ① Ups and downs of DL

- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?
- 2006: RBM initialization
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition
- 2015.2: Image recognition surpassing human-level performance
- 2016.3: Alpha GO beats Lee Sedol
- 2016.10: Speech recognition system as good as humans

### ② 深度学习三步走之一 (Define a set of function)

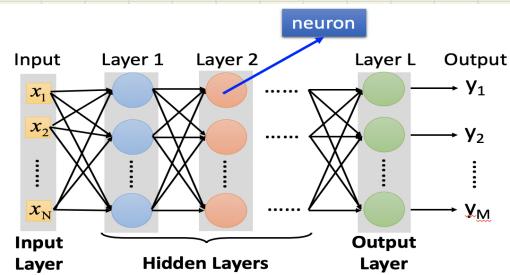
Function Set 主要由神经网络 (Neural Network) 定义，神经网络由一系列 Logistic Regression 组成，即由许多对  $w$  和  $b$  决定



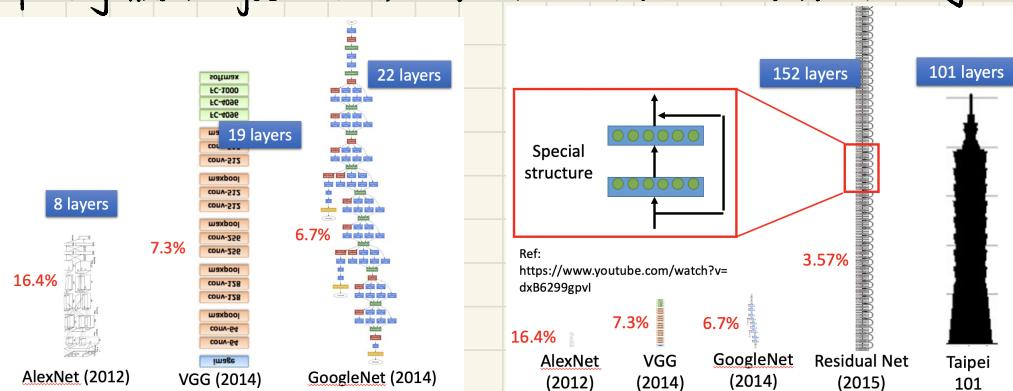
右图为 Full Connect Feedforward Net

恰是一个函数结构  $\Leftrightarrow$  定义了一个函数集合

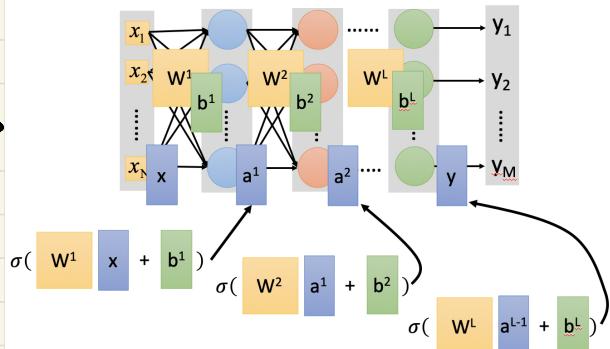
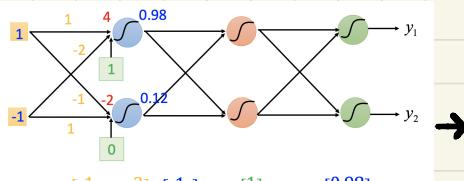
上图的全连接神经网络可以抽象为下图：



Deep = Many hidden Layers; “深度”的定义因人而异，一般使用到神经元结构的称为 Deep learning



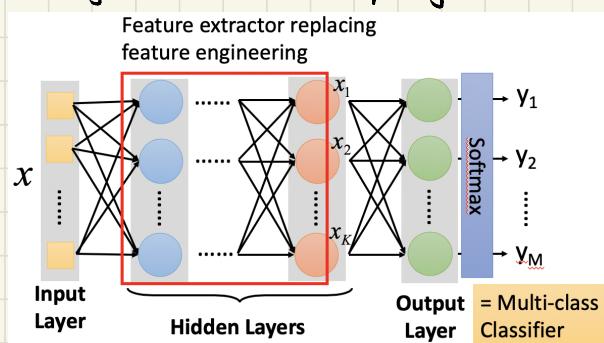
使用矩阵操作计算神经网络：



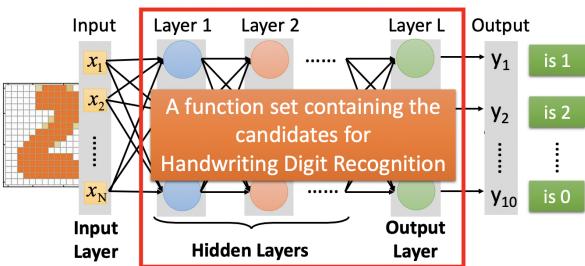
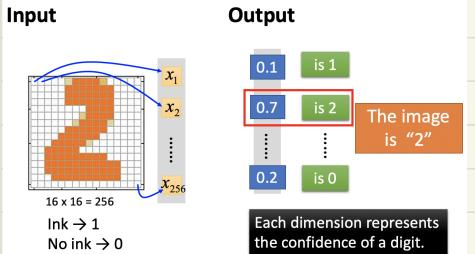
$$y = f(x) = \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

该层不断的对前一层进行嵌套运算  
可以使用GPU并行计算加速

Hidden Layer 的作用相当于特征提取，Output Layer 的作用相当于将提取起的特征用一个类别分类器进行分类



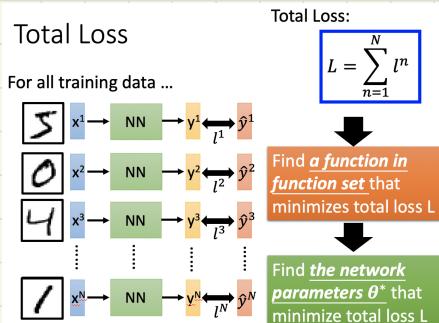
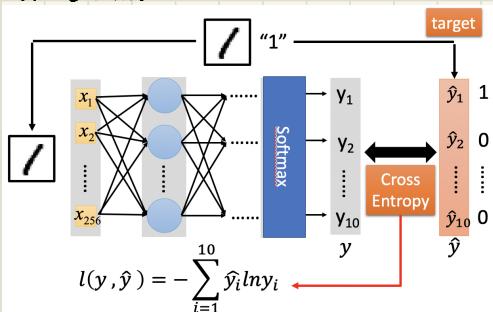
以手写数字的识别为例：



FAQ { 需要多少层？每层需要多少个神经元？由“Trial and Error” + “Intuition”决定  
网络结构可以自动化的生成？ Evolutionary Artificial Neural Network

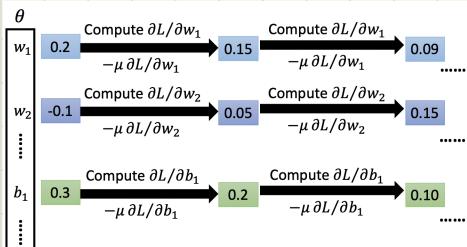
## ③深度学习三步走之二 (Goodness of Function)

定义 Loss Function:



## ④深度学习三步走之三 (Pick the best Function)

使用 Gradient Descent 计算出一组最优的参数，从而确定 Best Function



- Backpropagation: an efficient way to compute  $\partial L / \partial w$  in neural network



Caffe

Deep Learning Library produced by DSSTNE



theano



Chainer



libdnn

台大周伯威  
同學開發

## ⑤ Why "Deep"?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1 <span style="color:red">•</span>

Not surprised, more parameters, better performance

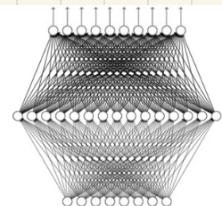
Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 

Any continuous function  $f$

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given enough hidden neurons)



Reference for the reason:  
<http://neuralnetworksanddeeplearning.com/chap4.html>

"Fat" Neural Network

## 2. Backpropagation

### ① Gradient Descent

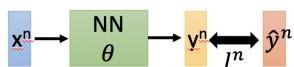
Network parameters  $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

Starting Parameters  $\theta^0 \longrightarrow \theta^1 \longrightarrow \theta^2 \longrightarrow \dots$

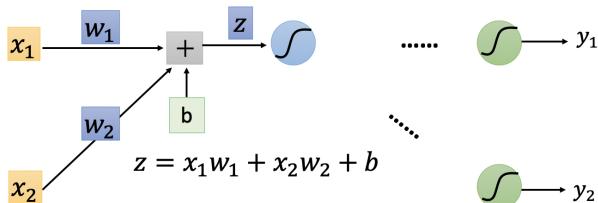
$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial w_1} \\ \frac{\partial L(\theta)}{\partial w_2} \\ \vdots \\ \frac{\partial L(\theta)}{\partial b_1} \\ \frac{\partial L(\theta)}{\partial b_2} \\ \vdots \end{bmatrix}$$

Compute  $\nabla L(\theta^0)$        $\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$   
 Compute  $\nabla L(\theta^1)$        $\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

神经网络有相当多的参数，对每一个参数求梯度进行更新是很费时费力的，所以使用反向传播算法进行更新



$$L(\theta) = \sum_{n=1}^N l^n(\theta) \quad \Rightarrow \quad \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^N \frac{\partial l^n(\theta)}{\partial w}$$



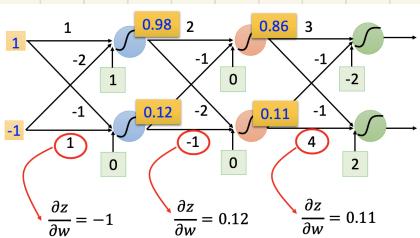
关注一个样本 data 对于 w 的偏微分，然后再求和即可

$$\frac{\partial L}{\partial w} = \left( \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w} \right) + \left( \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w} \right)$$

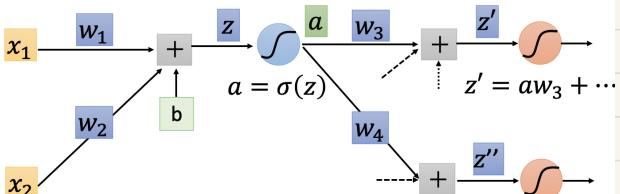
Forward Pass  
Backward Pass

## ② Forward Pass ( $\frac{\partial z}{\partial w}$ )

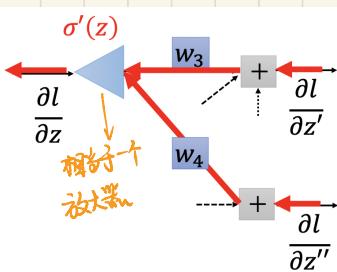
$$\left. \begin{array}{l} \frac{\partial z}{\partial w_1} = x_1 \\ \frac{\partial z}{\partial w_2} = x_2 \end{array} \right\} \text{关于 } w_i \text{ 的偏微分为其输入的 Input value (x_i)}$$



## ③ Backward Pass ( $\frac{\partial l}{\partial z}$ )



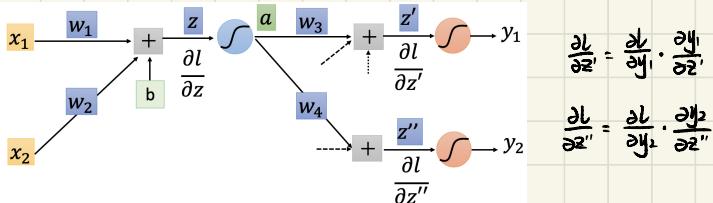
$$\begin{aligned} \frac{\partial l}{\partial z} &= \frac{\partial l}{\partial a} \cdot \frac{\partial a}{\partial z} \\ &= \left( \frac{\partial l}{\partial z'} \cdot \frac{\partial z'}{\partial a} + \frac{\partial l}{\partial z''} \cdot \frac{\partial z''}{\partial a} \right) \cdot \sigma'(z) \\ &= \left( \frac{\partial l}{\partial z'} \cdot w_3 + \frac{\partial l}{\partial z''} \cdot w_4 \right) \cdot \sigma'(z) \end{aligned}$$



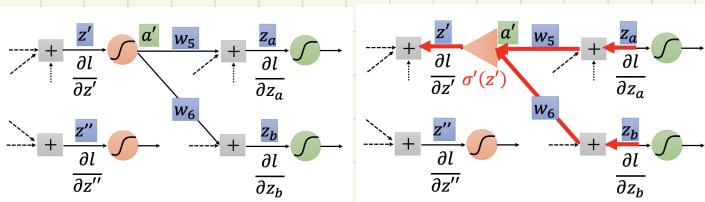
$$\frac{\partial l}{\partial z} = \sigma'(z) \left[ w_3 \frac{\partial l}{\partial z'} + w_4 \frac{\partial l}{\partial z''} \right]$$

$\sigma'(z)$  为一个常量，因为在 forward pass 中已经计算出来了

$\frac{\partial l}{\partial z}$  和  $\frac{\partial l}{\partial z''}$  的计算方法 (红色激活函数已经是 Output Layer, 蓝色激活函数为隐藏层的最后一层时)



$\frac{\partial l}{\partial z}$  和  $\frac{\partial l}{\partial z''}$  的计算方法 (红色激活函数非 Output Layer 时)

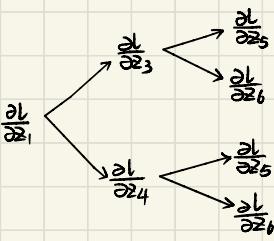
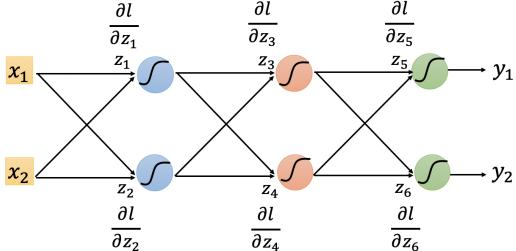


· 改做  $\frac{\partial l}{\partial z_a}$  和  $\frac{\partial l}{\partial z_b}$  时，可以套用上面的公式求解  $\frac{\partial l}{\partial z}$

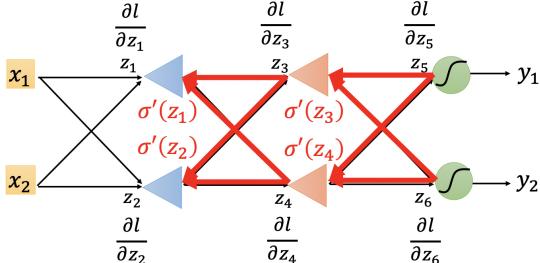
$$\frac{\partial l}{\partial z} = \sigma'(z') \cdot \left( w_5 \cdot \frac{\partial l}{\partial z_a} + w_6 \cdot \frac{\partial l}{\partial z_b} \right)$$

· 当  $\frac{\partial l}{\partial z_a}$  和  $\frac{\partial l}{\partial z_b}$  未知时，可使用上述方法，递归计算直到达到 Output layer

使用递归的方法计算偏微分时 ( $a \rightarrow b$  表示已知  $b$  才算出  $a$ )

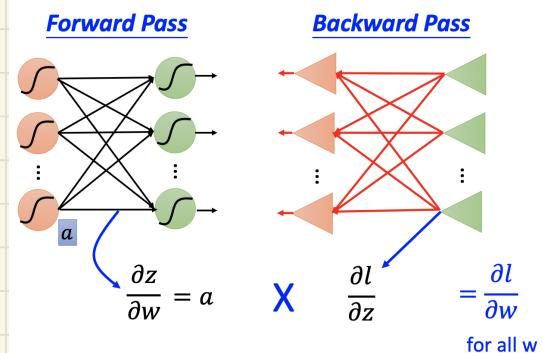


为了更有效地计算偏微分，使用动态规划的方法进行计算，即从输出层开始向前计算

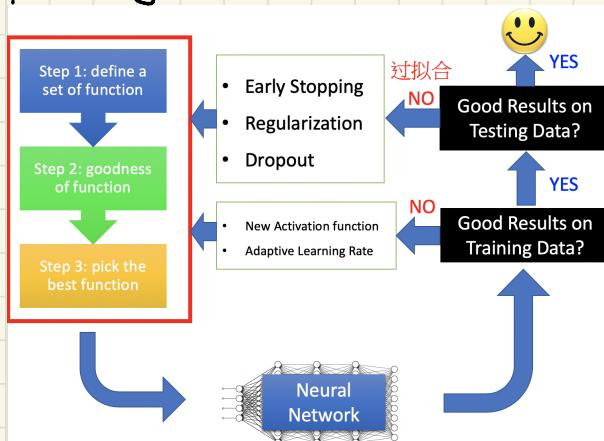


## ④ Back propagation

将 Forward Pass ( $\frac{\partial l}{\partial w}$ ) 和 Backward Pass ( $\frac{\partial l}{\partial z}$ ) 相乘即可得到关于  $w$  的梯度

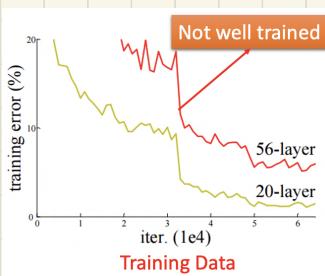
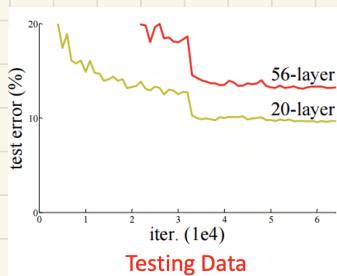


### 3. Tips for Deep Learning



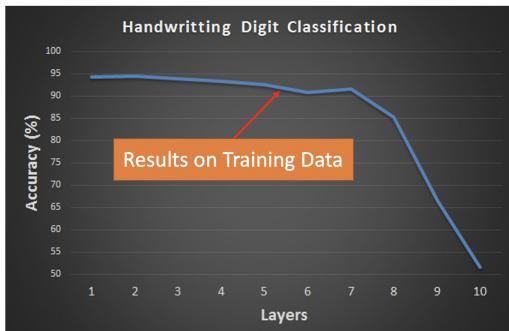
#### ① Performance & Overfitting

模型的 Performance 不够好，不一定是由于 Overfitting 造成的，需要具体分析。在 testing data 上，层数多的神经网络反而得不到好的 Performance，这有可能是 Overfitting，但也有可能是因为右图中显示在 training data 上，没有 well trained。

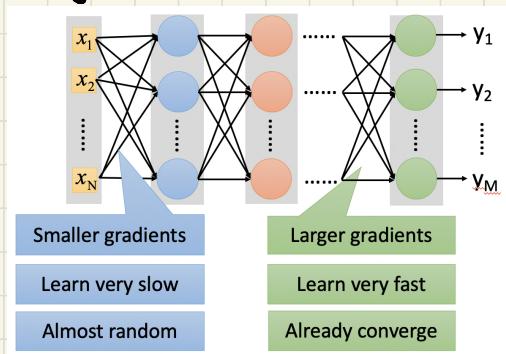


#### ② 激活函数与 Not Good Result in Training Data

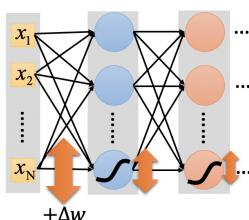
神经网络的层数越多并不意味着效果一定越来越好，可能的原因有 Vanishing Gradient 和 Maxout。



Vanishing Gradient: 在靠近 Input layer 的地方,  $\frac{\partial L}{\partial w}$  很小; 在靠近 Output layer 的地方,  $\frac{\partial L}{\partial w}$  很大

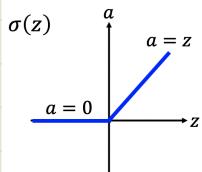


在设置同样的学习率时, 靠近输入层的地方  $w$  还接近于随机变化时, 靠近输出层的权重已经基于前者找到了一个局部最优, 使得 loss 的变化越来越小, 无法进行有效更新



- $\frac{\partial L}{\partial w}$  表示的是 loss 对于  $w$  的变化程度, 也可以通过改变  $w$ , 观察 loss 的变化量
- $\frac{\partial L}{\partial w} \approx \frac{\Delta L}{\Delta w}$
- 在 Input layer 后给  $w$  拥加一个较大的  $\Delta w$ , 因为 Sigmoid 函数的作用是将实数域压缩到 [0, 1], 所以经过多层的压缩, 较大的  $\Delta w$  带来的影响已经被压缩到微乎其微

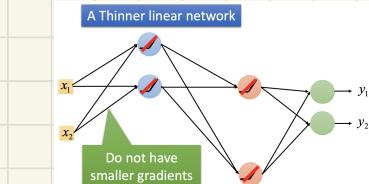
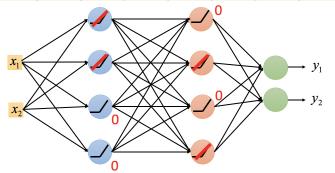
## ReLU (Rectified Linear Unit)



[Xavier Glorot, AISTATS'11]  
[Andrew L. Maas, ICML'13]  
[Kaiming He, arXiv'15]

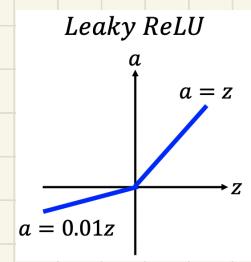
- 快速计算
- 相当于无限个不同 bias 的 Sigmoid 函数量加在一起
- 可以解决梯度消失的问题

- 当 Neuron 的输入为零时其输出也为零, 对神经网络的输出无影响; 将其从网络中剔除后, 网络相当于一个很长的 Linear Network, 此时因为 Activation Function 是线性的, 不会对输入进行压缩, 因此可以解决梯度消失问题

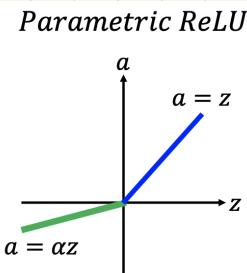


- 虽然 ReLU 在大于 0 和小于 0 的部分是线性的, 但其整体不是线性的 (不是直线), 组合多个 (线性操作 + ReLU) 就可以往复杂的划分空间 (一条曲线无限分段, 每一段都是线性的, 但组合起来也可以拟合复杂的非线性情况)。
- 在浅层的机器学习中, 以经典三层神经网络为例, 使用 ReLU 作为激活函数, 表现出的性质是线性的; 但在更多的深度神经网络中, 更多的 ReLU 组合起来可以表现出非线性效果
- 虽然 ReLU 在零点不可微, 但不影响梯度下降

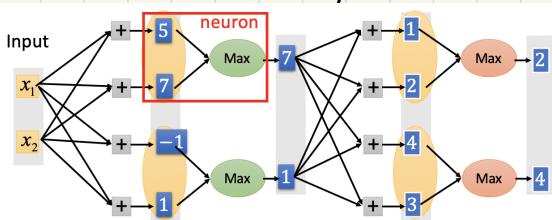
### • Leaky ReLU



### • Parametric ReLU

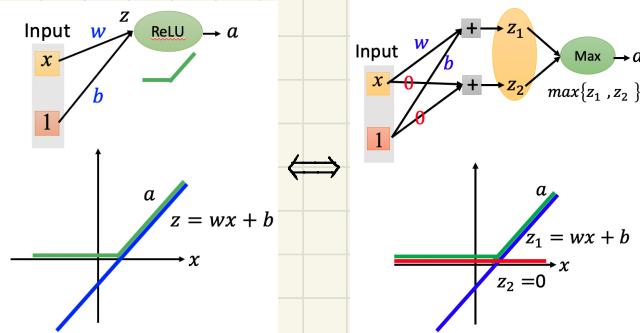


### Maxout (Learnable activation function)



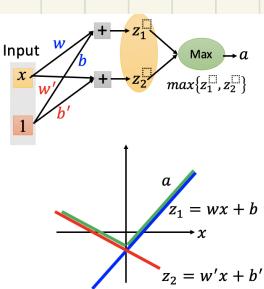
• 将多个神经元组成一个 Group (数量为数是可变的), Maxout 激活函数在 Group 中选择最大的一个值作为神经元的输出值

### • ReLU是Maxout的一种特例



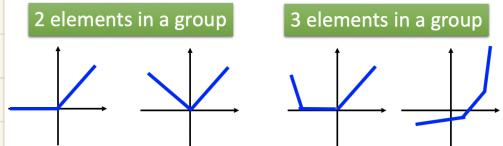
• 通过将一组参数设置为  $w, b$ , 另一组参数设置为全零, 就可以让 Maxout 达到 ReLU 的效果

### • Maxout是一种Learnable Activation Function

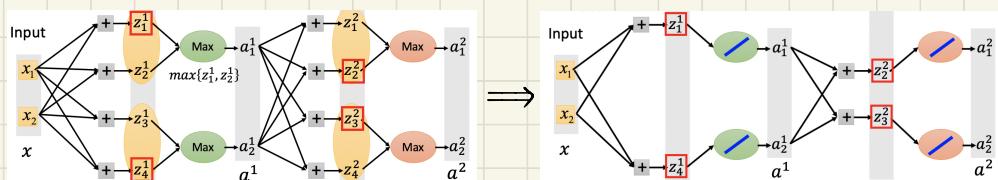


Maxout 的输出一部分取决于参数  $w$  和  $b$  的大小, 不同的  $w$  和  $b$  组合可以生成不同的激活函数

- Maxout 网络中的激活函数可以是任意 Piecewise Linear Convex Function (分段线性凸函数)
- Maxout 激活函数的形状取决于 Group 内的单元数目



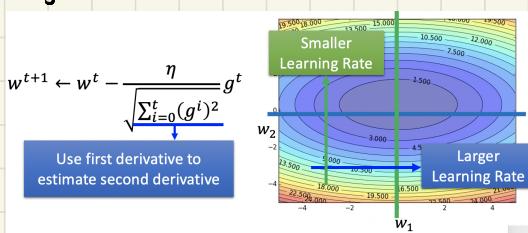
虽然 Maxout 激活函数不能微分，但对网络简化后网络变成一个深的 Linear Network，就可以求解了；一组数据只能使得部分参数得到更新，但只要经过大量样本的训练，所有参数均可以被训练到



## ② 自适应学习率与 Not Good Result in Training Data

### Gradient Descent Review:

- Adagrad



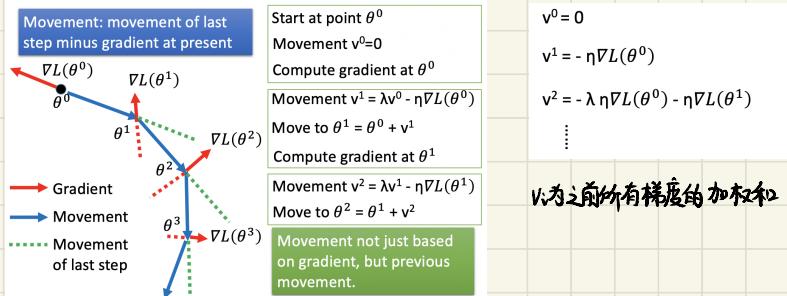
- RMSProp

$$\begin{aligned}
 w^1 &\leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 & \sigma^0 = g^0 \\
 w^2 &\leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 & \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2} \\
 w^3 &\leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 & \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2} \\
 &\vdots \\
 w^{t+1} &\leftarrow w^t - \frac{\eta}{\sigma^t} g^t & \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}
 \end{aligned}$$

Root Mean Square of the gradients with previous gradients being decayed

- 在一个神经网络中出现 local minima 的概率是比较小的。当 loss function 随着每一个维度的参数均呈下降趋势时，才会呈现“山谷”一样的 local minima，假设 loss 随一个参数量下降的概率是  $P$ ，网络共有  $N$  个参数 ( $N$  较大)，则  $P^N$  是一个很小的值，因此训练神经网络时卡在了一个梯度为零的点，则该点很有可能不是 Global Minima 或靠近全局最优的 local minimax 处。

- Momentum 可以跳出 local minimas 的情况，但不保证一定可以到达全局最优



- Adam

### RMSProp + Momentum

**Algorithm 1:** Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $\hat{\sigma}_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

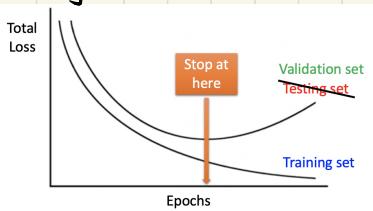
```

Require:  $\alpha > 0$ ;  $\beta_1, \beta_2 \in [0, 1]$ ; Exponential decay rates for the moment estimates
Require:  $f(\theta)$ ; Stochastic objective function with parameters  $\theta$ 
Require:  $\theta_0$ ; Initial parameter vector
 $m_0 \leftarrow 0$  (Initialize 1st moment vector)  $\rightarrow$  for momentum
 $v_0 \leftarrow 0$  (Initialize 2nd moment vector)  $\rightarrow$  for RMSprop
 $t \leftarrow 0$  (Initial timestep)
while  $\theta_t$  not converged do
     $t \leftarrow t + 1$ 
     $g_t \leftarrow \partial f_i(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )
     $g_t \leftarrow \beta_2^t \cdot m_{t-1} + (1 - \beta_2^t) \cdot g_t$  (Update biased first moment estimate)
     $v_t \leftarrow \beta_2^t \cdot v_{t-1} + (1 - \beta_2^t) \cdot g_t^2$  (Update biased second raw moment estimate)
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)
     $\hat{\theta}_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)
end while
return  $\hat{\theta}_t$  (Resulting parameters)

```

## ④ Early Stopping & Not Good Result in Testing Data

- 在 Training Set 的 Loss 达到一定程度，而 Validation Set 的 Loss 达到最小的时候停止训练模型，防止过拟合



Keras: <http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isn-t-decreasing-anymore>

## ⑤ Regularization & Not Good Result in Testing Data

- 在原有的 Loss Function 基础上重新构造新的 Loss Function，使得权重不仅可以使 Loss Function 最小化，而且权重越接近零越好

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2^2$$

original loss: square error, cross entropy

L2范式:  $\|\theta\|_2^2 = (w_1)^2 + (w_2)^2 + \dots$ , 不考虑bias (因为bias与模型无关)

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2 \quad \text{Gradient: } \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda w$$

Update:  $w^{t+1} \rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta \left( \frac{\partial L}{\partial w} + \lambda w^t \right)$

$$= (1 - \eta \lambda) w^t - \eta \frac{\partial L}{\partial w}$$

$(1 - \eta \lambda)$  是一个接近于 1 的值 (例如 0.99), 每次更新参数时都变小一点, 从而达到参数接近于零的目的  
但并不是所有参数均接近零, 后期  $(1 - \eta \lambda) w^t$  会与  $-\eta \frac{\partial L}{\partial w}$  达到平衡

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_1 \quad \|\theta\|_1 = |w_1| + |w_2| + \dots$$

$$\frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w)$$

Update:

$$\begin{aligned} w^{t+1} &\rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta \left( \frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w^t) \right) \\ &= w^t - \eta \frac{\partial L}{\partial w} - \eta \lambda \operatorname{sgn}(w^t) \end{aligned}$$

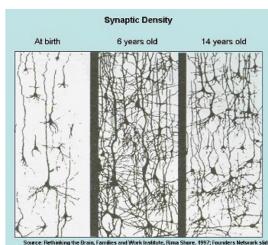
$\Rightarrow$  Weight Decay

$$\frac{\partial \|\theta\|_1}{\partial w} = \operatorname{sign}(w) = \begin{cases} 1, & \text{if } w > 0 \\ -1, & \text{if } w < 0 \end{cases}$$

w=0 处不可微可用次梯度

因此当  $w > 0$  时, 更新参数  $-1/\lambda$ ; 当  $w < 0$  时, 更新时  $+1/\lambda$ . 两种操作均使得  $w$  接近于零

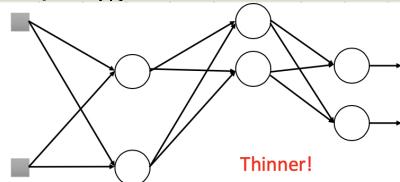
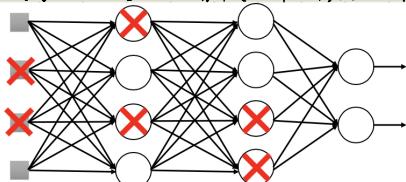
- L1 正则中  $w$  的下降速度取决于  $w$  本身,  $w$  越大下降的越快, 因此结果中所有参数都较为平均的很进零; 而 L2 正则  $w$  的变化为固定值  $\lambda$ , 所以结果中有很接近零的值, 也有离零很远的值
- 神经网络中的链接与人脑类似, 都会“用进废退”



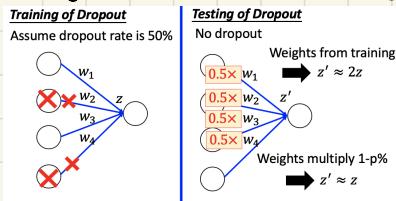
- Our brain prunes out the useless link between neurons.

## ⑥ Dropout & Not Good Result in Testing Data

- Dropout 指每次更新参数前, 采样一些神经元进行丢弃 (每个神经元被采样的概率为  $P\%$ )
- 对于每一个 mini-batch, 都需要重新采样失去一些神经元, 从而调整网络结构

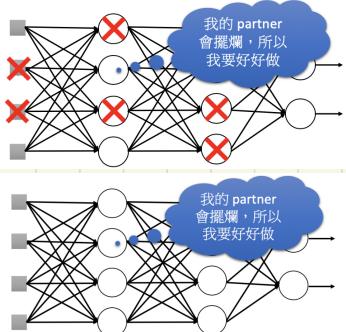


- Dropout 会使得在 training set 上的 performance 低于不使用 dropout，但在 testing data 上会更好
- 在 testing data 上不使用 dropout，但在 training data 上如果使用的 dropout rate 为 p%，测试时所有权重都要乘以  $(1 - p\%)$

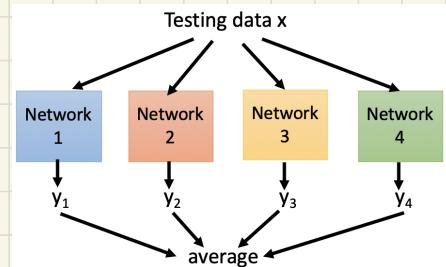
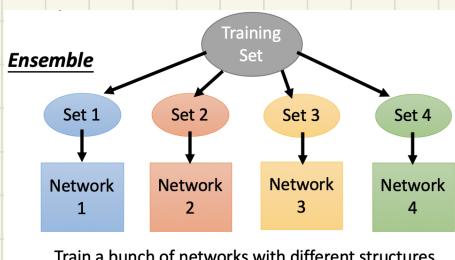


在 training data 上， $w_1$  和  $w_3$  的作用可以得到  $z$ ， $w_2$  和  $w_4$  也可以得到  $z$ 。  
那么在测试集上  $w_1, w_2, w_3, w_4$  共同作用会生成  $2z$ 。因此对权重乘以  $(1 - p\%)$  进行一下标准化。

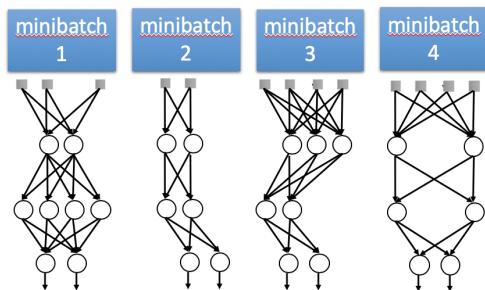
- Intuitive Reason：假设神经网络中每一个神经元都是一个学生，大家要共同完成一个大作业，当采用 Dropout 时，未被选中的同学就会认为失选的同学是偷懒的，所以我要做的更好。当每一个神经元都做到更好时，在测试集上不采用 Dropout 时，效果就会更好。



- Dropout 是一种 ensemble 方法



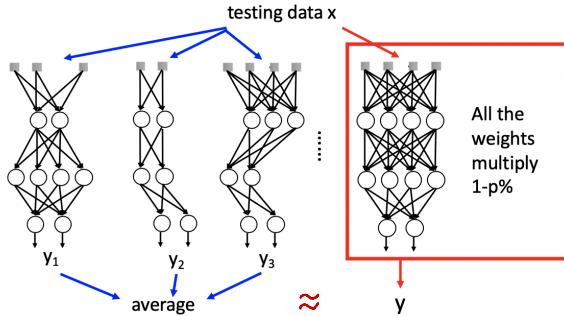
### Training of Dropout



Dropout 相当于使用不同的 minibatch 训练不同结构的网络。假设有 M 个 Neuron，则至多有  $2^M$  个神经网络。

一个 minibatch 中的数据可能不足以训练一个网络，但不同网络之间是共享参数的。

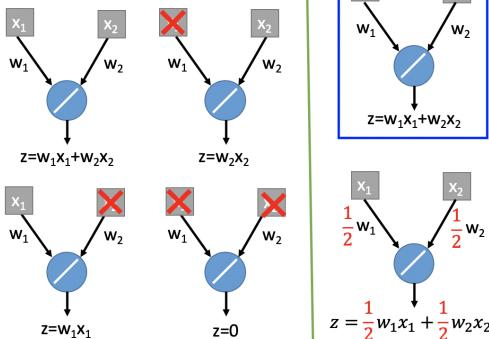
### Testing of Dropout



经过训练多个网络，将 testing data 送入网络得到输出后求均值即可得到结果 Average，但该方法过于笨重。

Dropout 提出使用  $(1-p\%)$  来随机断开网络，得到  $y$ ，经验证： $Average \approx y$

### Testing of Dropout



$$\text{左侧: } \text{Average} = \frac{1}{4} (w_1x_1 + w_2x_2 + w_2x_2 + w_1x_1 + 0) \\ = \frac{1}{2} (w_1x_1 + w_2x_2)$$

$$\text{右侧: } y = (1 - 50\%) (w_1x_1 + w_2x_2) = \frac{1}{2} (w_1x_1 + w_2x_2)$$

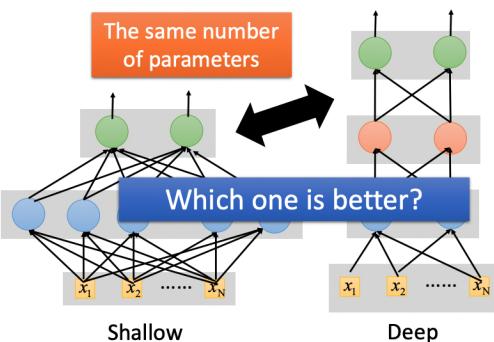
当 Network 是 Linear 时，左的结果才对；但 Dropout 的神奇之处在于即使 Network 不是 Linear 的，也可以得到不错的效果



由上可知，当 Network 是近似线性的情况下，效果会比较好（比如相对于 Sigmoid, ReLU 和 Maxout 更近似于线性）

## 4. Why Deeper?

### ① Shallow Network VS. Deep Network



Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

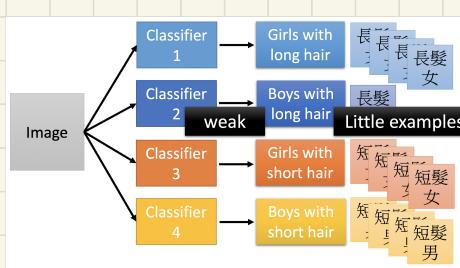
Fat + Short

Thin + Tall

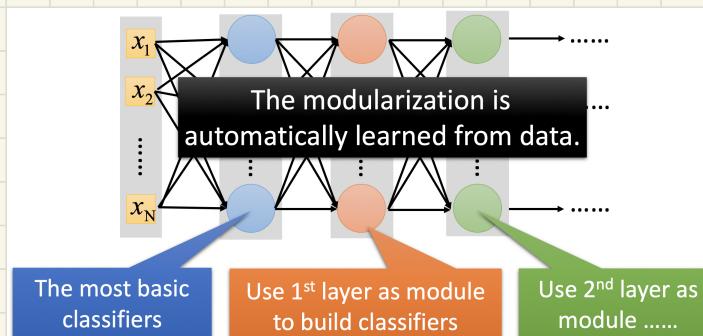
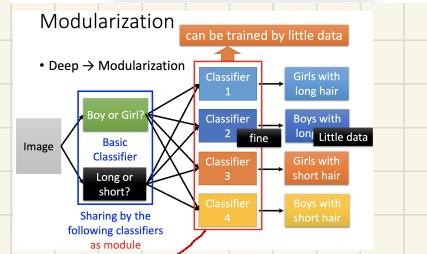
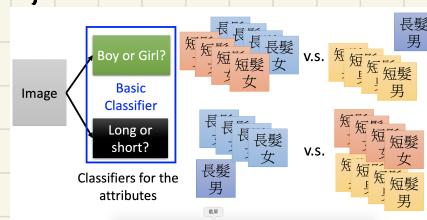
Why?

## ② Modularization (模块化)

- Deep与Modularization: 在图像分类中,因为长发男生的样本数量较少,所以对应的分类器效果一般。因此可以采用右图模块化的思想,对男女生和长短发分别进行分类,这样每一个 basic classifier都有足够的数据训练到不错的效果。



该分类器可以借用 basic classifier 的结果用较少的数据进行训练。

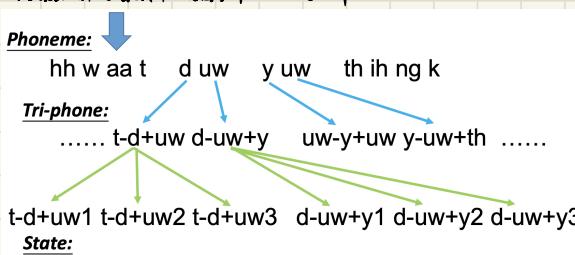


每层神经元都将上一层视为 module, 并利用其结果训练自己所在层的 classifier

Modularization都是神经网络自动学习的

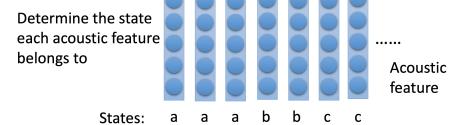
- “AI=Big Data + Deep Learning”的思想并不正确, 相反 Big Data 与 Deep Learning 的理念有些冲突的地方。比如: 现在用有全世界所有的 Data, 就不需要 Deep Learning, 只需要在所有 Data 中搜索就可以了。正是因为 Data 不够多, 才用现有的 Data 训练神经网络, 达到更广的适用范围。

## • Modularization 在语音辨识上的应用



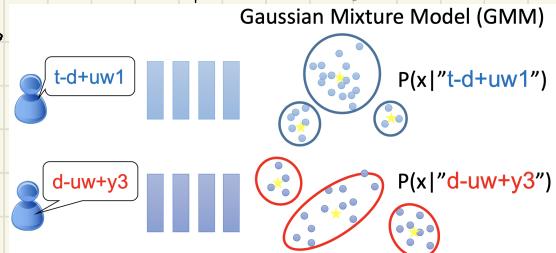
语音辨识的第一步是在输入语音上, 根据指定窗口, 将窗口内的语音转化为 acoustic feature, 然后决定 acoustic 属于哪一个 state

- Classification: input → acoustic feature, output → state

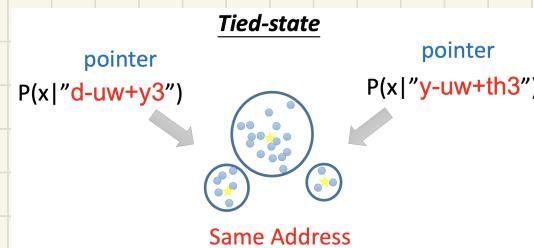


## HMM-GMM在state分类上的应用：

-↑ acoustic feature 对于每个分类都有一个 stationary distribution；因为一般语音有30个左右的 phoneme，根据上下文的内部不同，tri-phone 的数量高达  $30^3$  个，每个 tri-phone 又有3个 state，所以 state 的数量有  $3 \times 30^3$  个，每一个 state 都要用一个 Gaussian Distribution 描述，过于复杂。

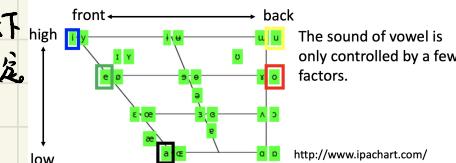


因此，引入“Tied-state”概念，即部分 state 可以采用 Gaussian Distribution；因为可以确定一些高斯分布样本，每

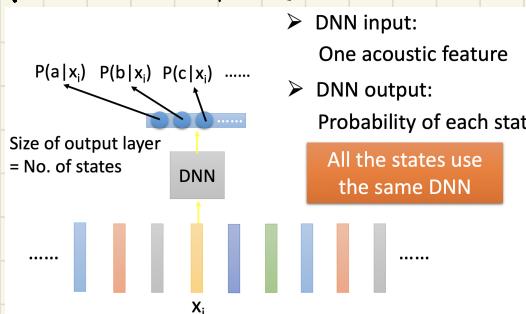


遇到一个 tri-phone 时，在高斯样本池中选取一些样本进行描述

在 HMM-GMM 中，所有 phoneme 都是被独立建模的，这是一件效率低下  
的工作。而实际上，音素受人体的舌头前后位置、上下位置以及口型决定  
相互之间是有一定联系的



使用 DNN 进行 state 分类时，Input 为一个 acoustic feature，Output 为属于每一个 state 的概率，输出层的数量等于 state 的数量；所有的 state 共用一个深度神经网络



将 DNN 中的其中一层 Hidden Layer 的输出降维至二维如图所示：

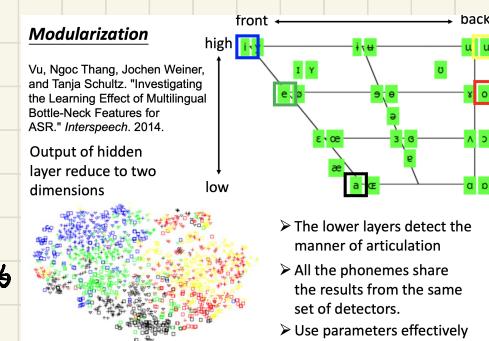
左下和右上两张图的颜色分布很相似

{较低的层检测的是发音的方式}

{所有 phoneme 到 state 的分类都属于上一条的发音方式检测的后果}

模块化可以以效率较高的方式使用参数

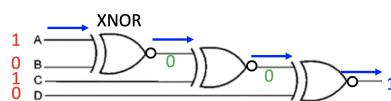
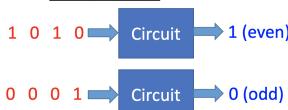
DNN 和 HMM-GMM 的参数数量相当，但 DNN 相当于用了  
很多参数的大模型，而 HMM-GMM 是用少参数的一堆小模型



尽管在神经元足够多的情况下, shallow network 可以表示任何函数, 但 Deep Learning 的结构会更高效。以逻辑电路与神经网络类比如下:

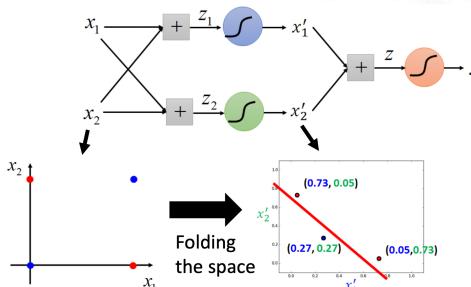
Logic Circuits	由门控电路组成 (与/或/非)	两层门控单元可以表示任何 Boolean Function	似用更多的层数, 更少的门控单元 高效的完成目标
Neural Network	由神经元组成	两层神经网络 (Input + 1 层 Hidden + Output) 可以表示任何 Continuous Function	可以用更多的层数, 更少的 Neuron 高效的完成目标

• E.g. parity check



With multiple layers, we need only  $O(d)$  gates.

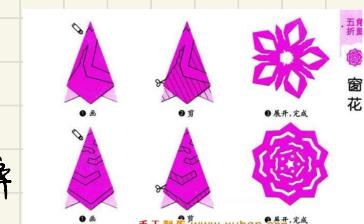
Analogy 2: 剪窗花时先折纸剪会更有效率



Analogy 3: shallow & deep 在不同数据量的效果  
在 1 hidden layer 和 3 hidden layer 的参数量大  
致相同时, 三隐层的网络可以更高效的使用数  
据

Analogy 1:

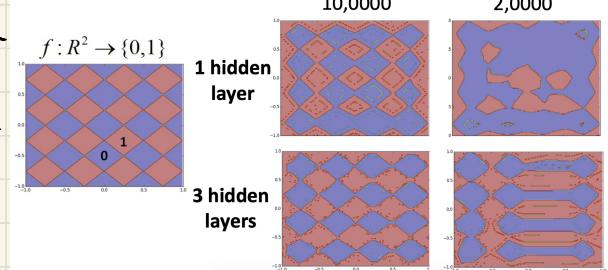
实现  $d$  个 bits 的奇偶校验, 两层电路需要  $O(2^d)$  个门控单元, 多层电路只需要使用  $O(d)$  个门控单元



Analogy 3: 两层神经网络实现 2 蓝点分类, 第一层进行 Feature

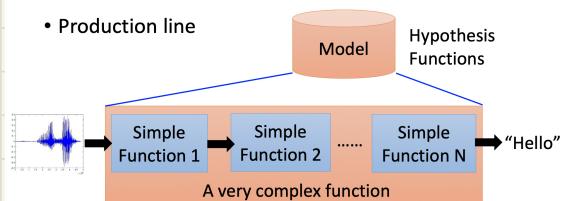
Transform (折叠空间), 第二层基于第一层的结果进行分类

Different numbers of training examples



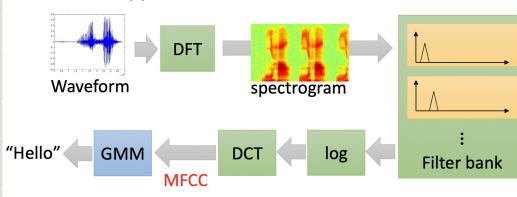
### ③ End-to-End Learning

- Deep learning 可以进行端到端的学习，对于一些复杂的任务不需要过多关注内部细节，关注输入输出即可。神经网络的每层都会自动学习到 Production Line 中对应的 Function.

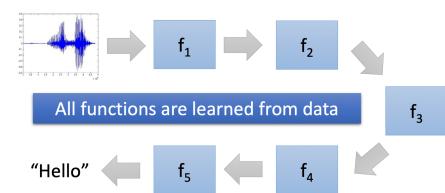


### • Shallow Approach 与 Deep learning 在 Speech Recognition 应用上的对比

#### • Shallow Approach



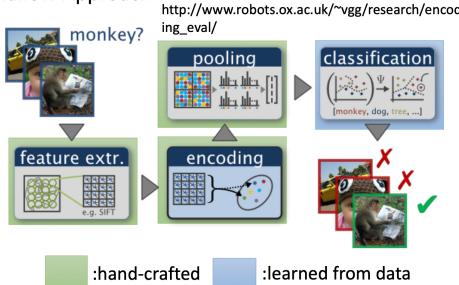
#### • Deep Learning



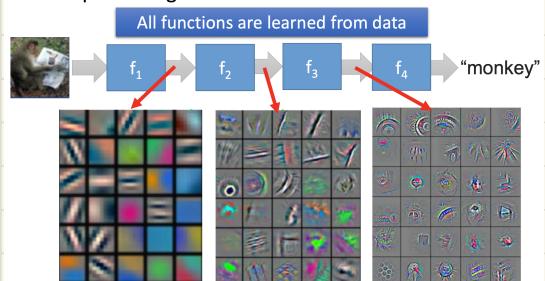
Less engineering labor, but machine learn more

### • Shallow Approach 与 Deep Learning 在 Image Recognition 应用上的对比

#### • Shallow Approach



#### • Deep Learning



### ④ Complex Task

- Deep Learning 可以完成很多复杂场景下的任务

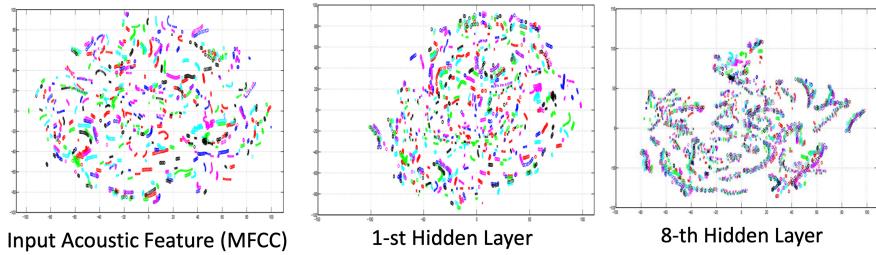
• Very similar input, different output



• Very different input, similar output



· 在 Speech Recognition 中，将 MFCC 特征降到二维如图。不同颜色表示不同人说的同一句话，在第一层图像还比较分散，在第八层，很多不同颜色被叠在一起形成线条，即经过八层的转换，不同的人说的同样的话对于网络而言是相同的。



· 在 Image Recognition 中“深度”的效果

