

# 深度学习

## Lecture 3: 深度前馈网络

王亮

智能感知与计算研究中心(CRIPAC)  
模式识别国家重点实验室(NLPR)  
中科院自动化研究所(CASIA)

# 目录

---

1 上节回顾

2 前馈网络简介

3 基本结构单元

4 网络结构设计

5 反向传播算法

# 回顾: 线性代数

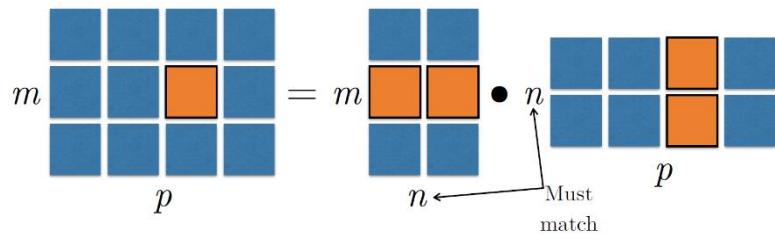
---

- 标量, 向量, 矩阵, 张量
- 矩阵转置, 矩阵点乘, 单位阵
- 方程组, 解方程组
- 矩阵求逆, 可逆性, 范数
- 特征分解, 迹

# 回顾: 线性代数

## 矩阵乘积

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$



## 解线性方程组

$$Ax = b \longrightarrow \begin{cases} (A_1, :)x = b_1 \\ (A_2, :)x = b_2 \\ \dots \\ (A_m, :)x = b_m \end{cases}$$

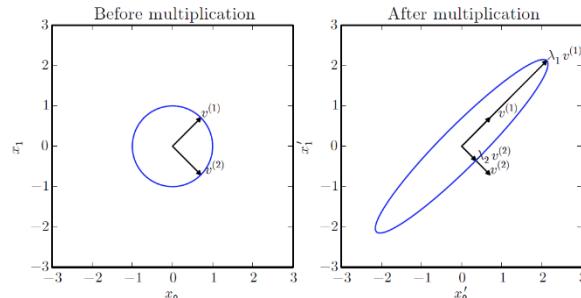
## 特征值分解

## $L^p$ 范数

$$\|x\|_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}}$$

$$A\boldsymbol{\nu} = \lambda\boldsymbol{\nu}$$

$$A = V \text{diag}(\lambda) V^{-1}$$



## 奇异值分解

$$A = UDV^T$$

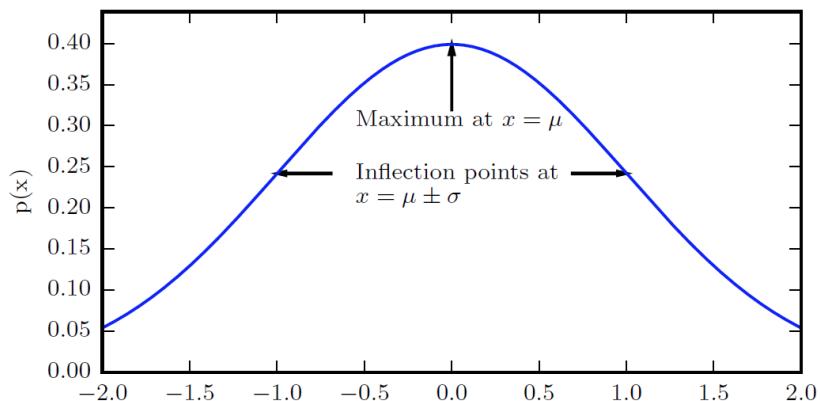
# 回顾: 概率论与信息论

---

- 概率分布函数, 概率密度函数
- 边缘概率, 条件概率
- 链式法则, 独立性假设
- 期望, 方差和协方差

# 回顾: 概率论与信息论

## 高斯分布



## 信息论

$$I(x) = -\log P(x)$$

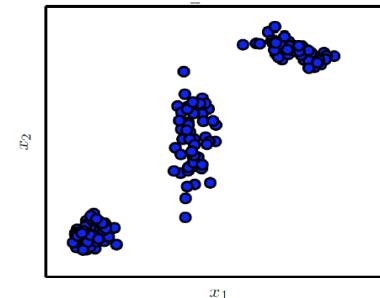
$$\mathsf{H}(x) = E_{X \sim P}[I(x)] = E_{X \sim P}[\log P(x)]$$

$$\begin{aligned} D_{KL}(P \parallel Q) &= E_{X \sim P} \left[ \log \frac{P(x)}{Q(x)} \right] \\ &= E_{X \sim P} [\log P(x) - \log Q(x)] \end{aligned}$$

## 混合分布

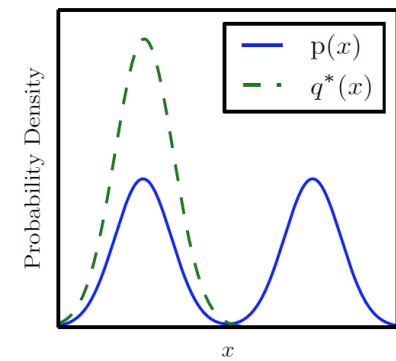
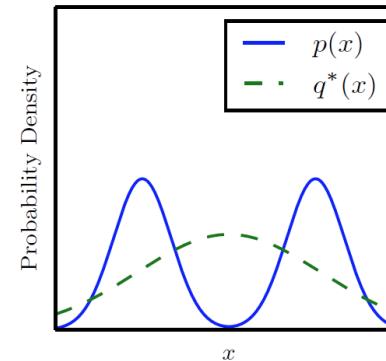
$$P(x) = \sum_i P(c = i)P(x|c = i)$$

由三个组件构成的高斯混合分布



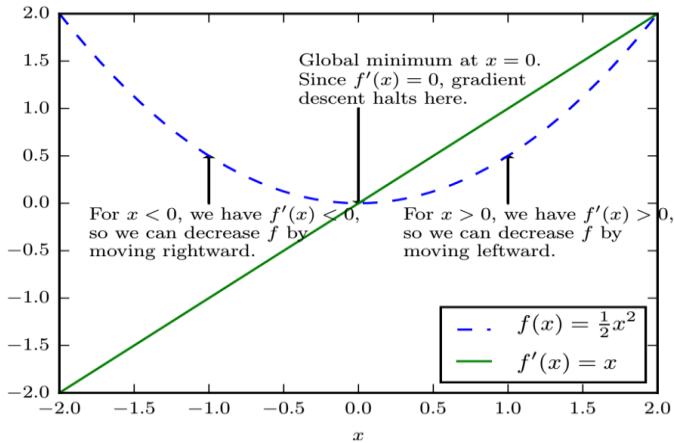
## KL散度是不对称的

$$q^* = \operatorname{argmin}_q D_{KL}(p \parallel q) \quad q^* = \operatorname{argmin}_q D_{KL}(q \parallel p)$$

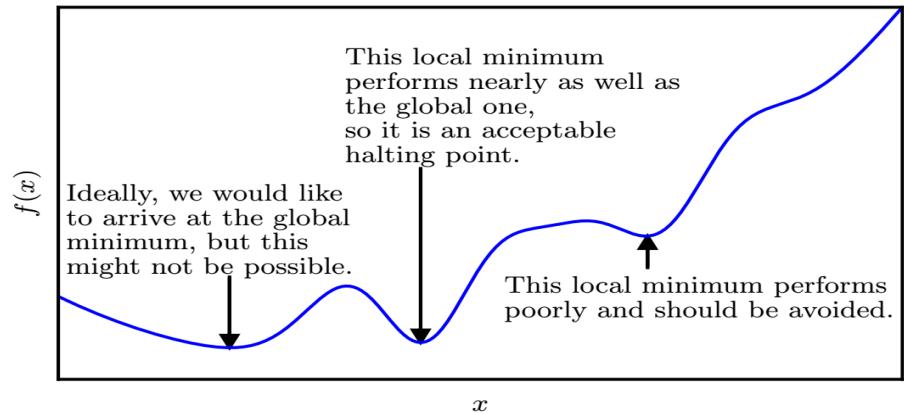


# 回顾：深度学习的数值问题

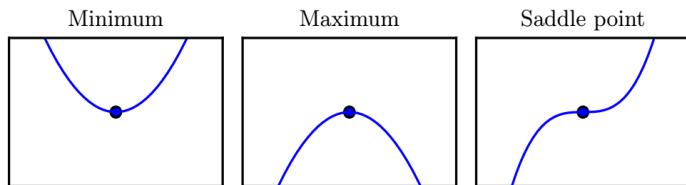
## 梯度下降



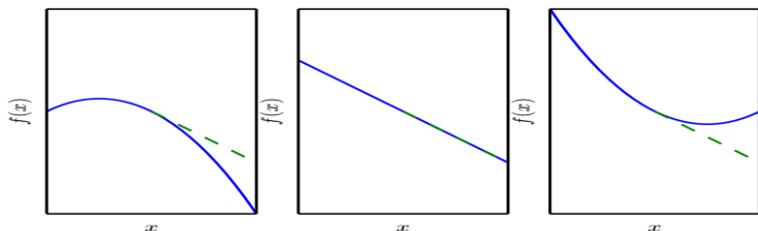
## 近似优化



## 临界点



## 曲率



## 预测最佳步长

### 使用泰勒级数

$$f(x^{(0)} - \epsilon g) \approx f(x^{(0)}) - \epsilon g^\top g + \frac{1}{2} \epsilon^2 g^\top H g.$$

$$\epsilon^* = \frac{g^\top g}{g^\top H g}.$$

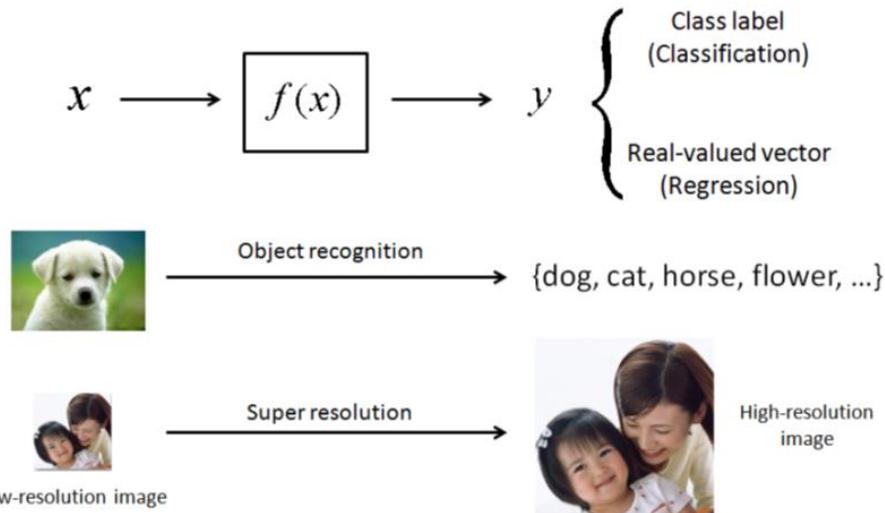
大的梯度加速收敛

如果与它们的特征向量对齐，  
大的特征值将会减速

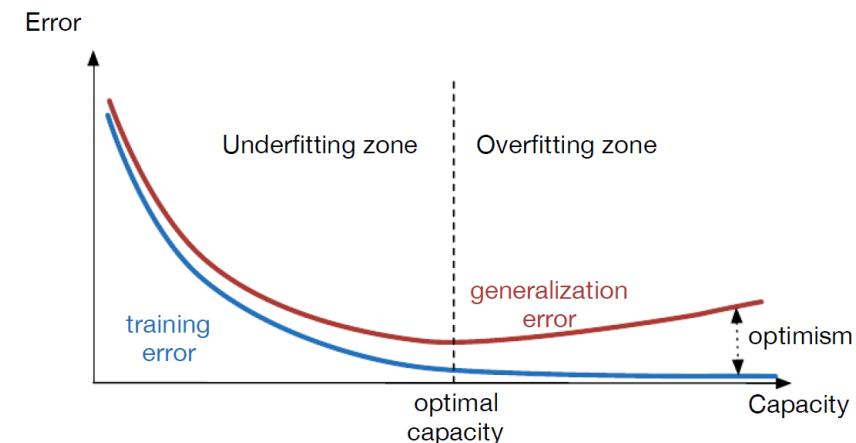
(Goodfellow 2017)

# 回顾：机器学习基础

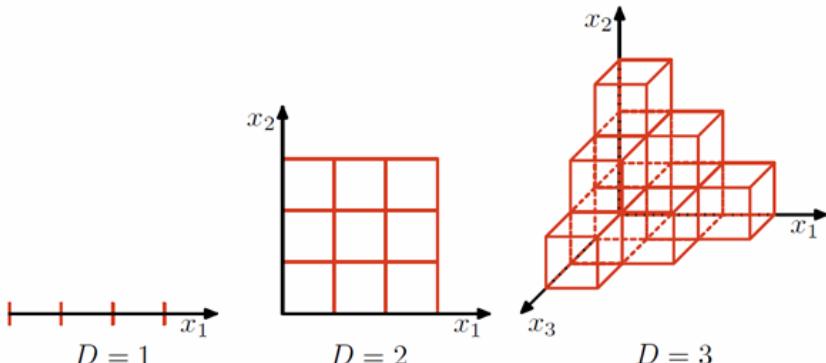
## 机器学习系统



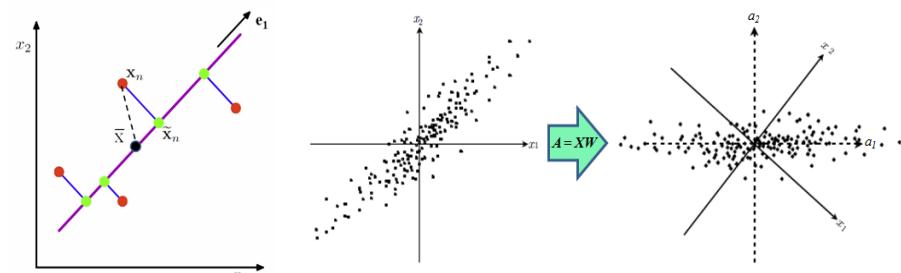
## 最佳容量



## 维度灾难



## PCA降维



# 目录

---

1 上节回顾

2 前馈网络简介

3 基本结构单元

4 网络结构设计

5 反向传播算法

# 历史背景

- 神经网络数学模型的开创性工作
  - McCulloch and Pitts 1943
  - 包括 recurrent 和 non-recurrent (带 “环” ) 的网络
  - 使用阈值函数作为非线性激活
  - 非学习方法
- 学习神经网络的早期工作
  - 始于 Rosenblatt 1958
  - 利用阈值函数作为非线性激活，避免了用链式法则计算导数，使得误差不能被传播回来指导梯度的计算
- 自1960年以来，反向传播在若干阶段中得到发展
  - 关键思想是用链式法则来计算导数
  - 出现在多个工作中，最早出自控制领域

# 历史背景

- 神经网络的标准反向传播
  - Rumelhart, Hinton, and Williams, Nature 1986. 清楚地认识到反向传播的能力，在关键任务上进行了演示，并将其广泛地应用于模式识别
  - 在1985年, Yann LeCun 独立开发了一种三层网络的学习算法，其中传播的是目标值，而不是导数。1986年，他证明了它等价于标准的反向传播；
- 证明了三层神经网络的普遍表达能力
  - Hecht-Nielsen 1989
- 卷积神经网络
  - Kunihiko Fukushima 在 1980年引入
  - LeCun, Bottou, Bengio, 和 Haffner 在1998年改进

# 历史背景

---

- 深度置信网络(DBN)
  - Hinton, Osindero, and Teh 2006
- 自动编码机
  - Hinton and Salakhutdinov 2006 (Science)
- 深度学习
  - Hinton. Learning multiple layers of representations. Trends in Cognitive Sciences, 2007
  - 无监督多层预训练+有监督 fine-tuning (BP)
- 语音识别中的大规模深度学习
  - 2009年底, Geoff Hinton 和 Li Deng 在微软雷德蒙德研究院开始研究
  - 生成式DBN的预训练没有必要
  - 大规模的训练数据+拥有上下文相关的输出层的大规模深度神经网络(DNN)取得了成功

# 历史背景

---

- 大尺度图像的无监督深度学习
  - Andrew Ng et al. 2011
  - 无监督特征学习
  - 16000 CPUs
- ImageNet图像分类中的大规模监督深度学习
  - Krizhevsky, Sutskever, and Hinton 2012
  - 卷积神经网络的监督学习
  - 没有无监督预训练

# 典型应用

---

- 人工神经网络的应用：
  - 函数近似/建模
  - 模式分类/识别(时间序列分析, 字符识别等)
  - 数据压缩
  - 安防(预防信用卡诈骗等)
  - .....

# 典型应用

---

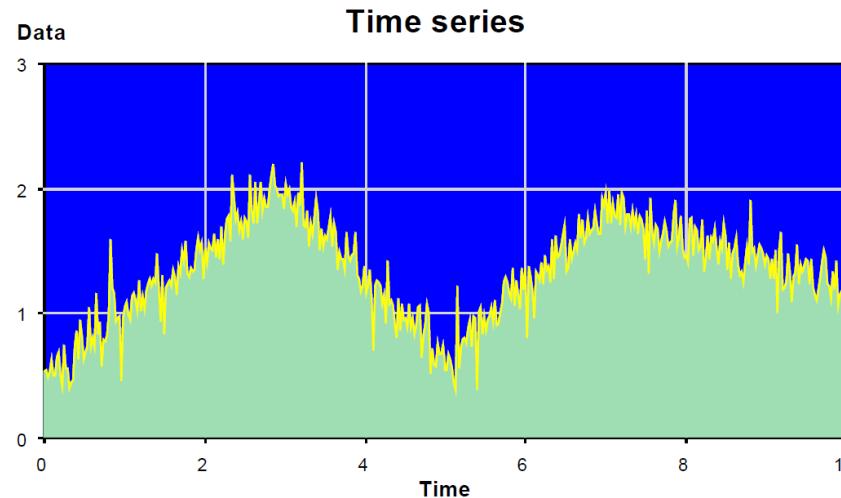
- 模式分类/识别
  - 在一些文献中，所有输入值的集合称为输入模式，输出值的集合称为输出模式
$$x = (x_1, \dots, x_m) \rightarrow y = (y_1, \dots, y_n)$$
  - 神经网络“学习”不同输入和输出模式之间的关系；
  - 因此，神经网络执行模式分类或模式识别任务(即将输入分类为输出的类别)；

# 典型应用

- **时间序列分析**

时间序列是对不同时间的某些变量(如股价、温度)的记录：

$$x(t_1), x(t_2), \dots, x(t_m)$$



- 分析的目的是学会预测未来的值

# 典型应用

## 时间序列分析：

- 我们可以用神经网络来分析时间序列：

**Input:** 过去的m个值作为输入变量：

$$x(t_1), x(t_2), \dots, x(t_m)$$

**Output:** 未来的n个值作为输出变量：

$$y(t_{m+1}), y(t_{m+2}), \dots, y(t_{m+n})$$

- 我们的目标是找到以下模型：

$$(y(t_{m+1}), y(t_{m+2}), \dots, y(t_{m+n})) = f(x(t_1), x(t_2), \dots, x(t_m))$$

- 通过训练一个具有m个输入和n个输出的时间序列数据的神经网络，来实现这样的模型

# 优势和缺陷

---

- 神经网络的优势：
  - 可以应用到很多问题，只要有数据可用
  - 可以应用于分析方法尚不存在的问题
  - 可以用来建立非线性依赖关系的模型
  - 如果有一个模式，那么神经网络应该很快就能找到它，即使数据是“有噪声的”
  - 即使输入的信息不完整也能够给出一些答案
  - 网络很容易维护

# 优势和缺陷

---

- 神经网络的局限：
  - 与其他数据驱动模型一样，如果没有或只有很少的数据可用，神经网络则不可行
  - 有许多自由参数，如隐藏节点的数量、学习率等，这些参数对最终结果有很大的影响
  - 计算精度存在局限
  - 不具备可解释性。如果有很多节点，那么就有大量的权重是无法解释的。在某些任务中，可解释性是至关重要的(如空中交通管制、医疗诊断等)

# 前馈神经网络

---

- 目标：近似一些未知的理想函数

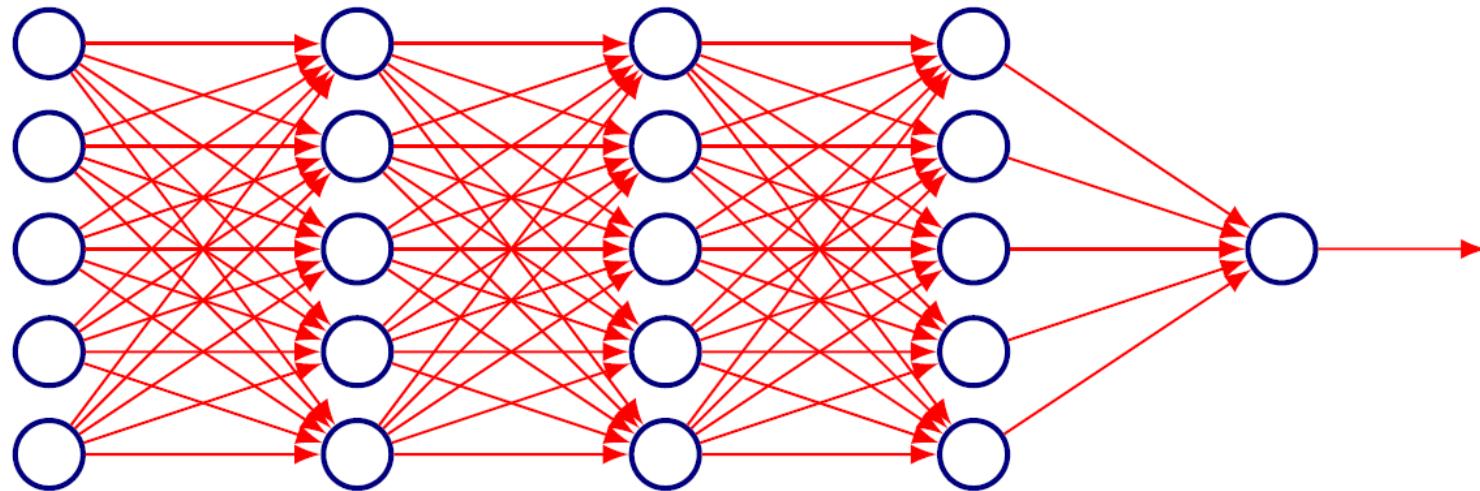
$$f^* : \mathcal{X} \rightarrow \mathcal{Y}$$

- 理想分类器： $y = f^*(\mathbf{x})$
- 前馈网络：定义映射： $y = f(\mathbf{x}, \theta)$
- 从可用的样本中学习参数 $\theta$ ，获得 $f^*$ 的一个较好的近似
- 信息流从输入开始，经过中间计算(即函数映射)，生成类别
- 没有反馈连接(循环网络!)

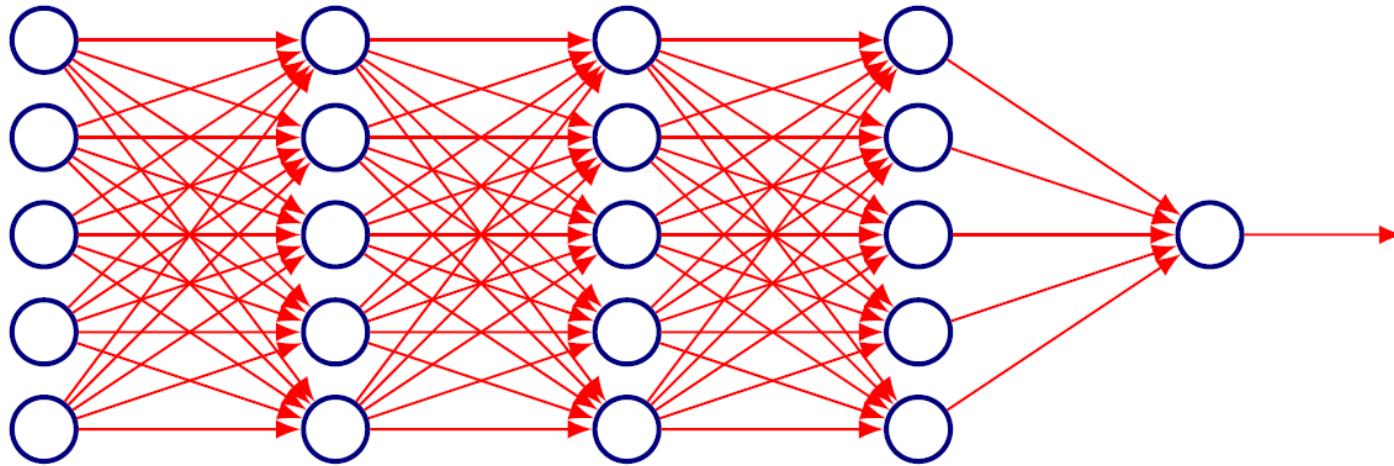
# 前馈神经网络

- 函数 $f$ 是许多不同函数的组合，例如：

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}|f^{(1)}(\mathbf{x})))$$



# 前馈神经网络



- 函数结构可以用**有向无环图**来描述(因此称为前馈网络);
- $f^{(1)}$  是第一层,  $f^{(2)}$  是第二层, 以此类推;
- 深度是函数组合链中最大的i;
- 最后一层称为输出层;

# 前馈神经网络

---

- **训练**: 优化  $\theta$  使得  $f(x; \theta)$  逼近  $f^*(x)$
- **训练数据**:  $f^*$  在不同的  $x$  实例中进行评估 (*i.e.* 期望输出)
- 只指定输出层的输出
- 中间层的输出不指定, 因此命名为隐藏层
- **神经**:  $f^{(i)}$  的选择和层次化结构组织, 受到神经科学的启发

# 回顾线性模型

---

- 优化是凸的或封闭的形式；
- 模型不能理解输入变量之间的相互作用；
- 延伸：做非线性变换  $x \rightarrow \phi(x)$ ；将线性模型应用于  $\phi(x)$
- $\phi$  给出  $x$  的特征表示
- 那么如何选择  $\phi$  ？

# 选择 $\phi$

---

- 方式1：使用通用的  $\phi$
- **优点**：足够的能力来适应训练数据；
- **缺点**：对于变化的  $f^*$  泛化能力差；
- 优先使用：局部光滑的函数。

# 选择 $\phi$

- 方式2：手工设计  $\phi$
- 依然保持凸性！

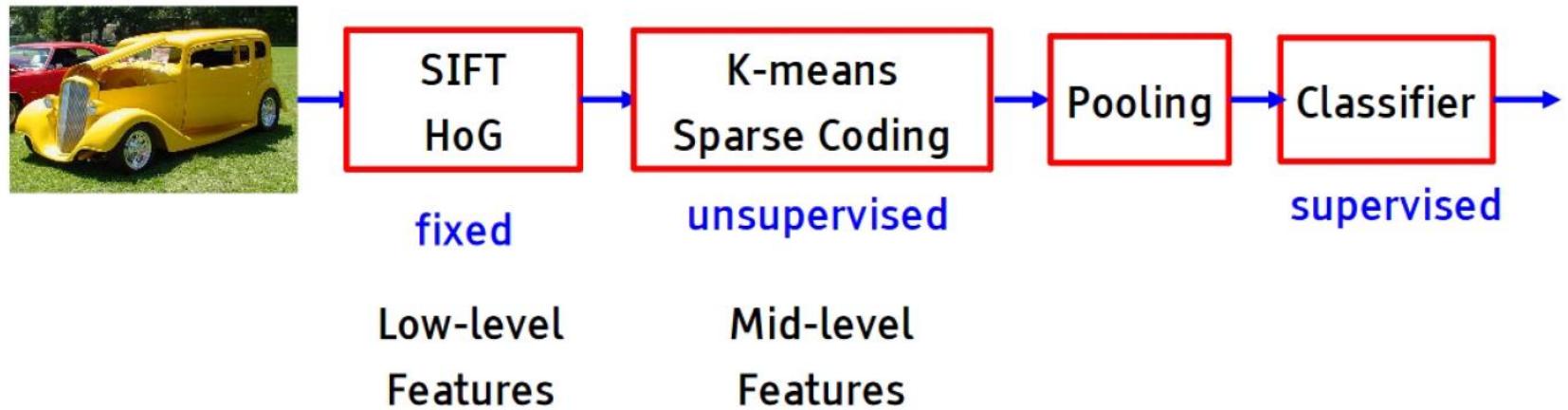
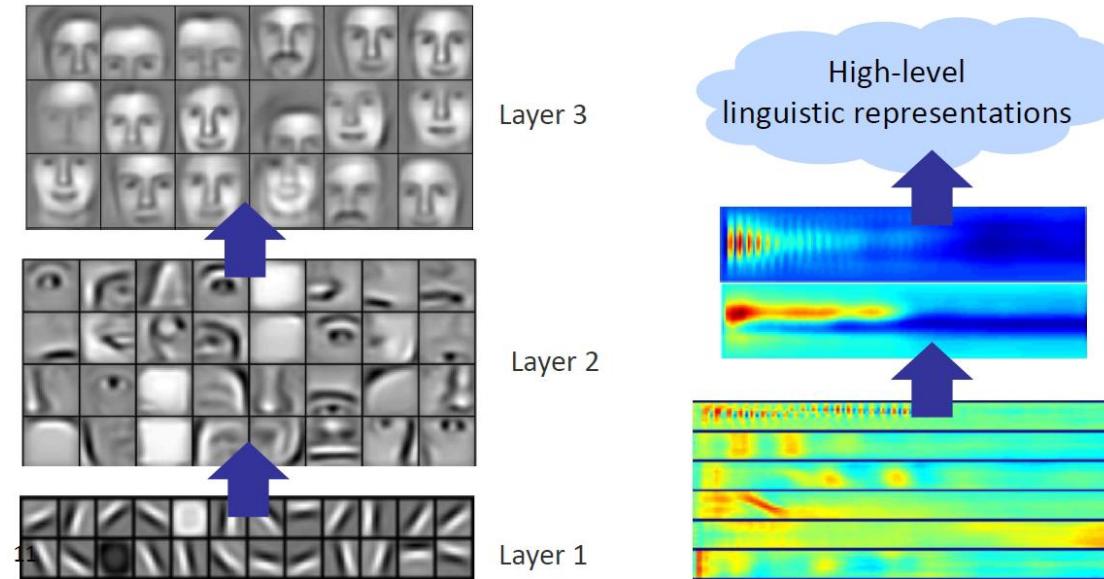


Illustration: Yann LeCun

# 选择 $\phi$

- 方式3：从数据总学习  $\phi$
- 放弃凸性
- 结合了前两个方式的优点：  $\phi$  可以是高度通用的，且手工设计同样可以融入结构中



# 设计决策

---

- 需要选择优化器、损失函数和输出形式
- 选择激活函数
- 结构设计（网络层数等）



# XOR Example

# XOR

*Exclusive-OR gate*



| A | B | Output |
|---|---|--------|
| 0 | 0 | 0      |
| 0 | 1 | 1      |
| 1 | 0 | 1      |
| 1 | 1 | 0      |

- 用XOR 作为我们想要学习的目标函数 $f^*(x)$
- 我们会调整 $f(x; \theta)$  中的参数 $\theta$  来尝试表达  $f^*$
- 可用的数据：

$$(X, Y) = \{([0, 0]^T, 0), ([0, 1]^T, 1), ([1, 0]^T, 1), ([1, 1]^T, 0)\}$$

# XOR

---

- 我们的数据

$$(X, Y) = \{([0, 0]^T, 0), ([0, 1]^T, 1), ([1, 0]^T, 1), ([1, 1]^T, 0)\}$$

- 不考虑通用性，只拟合这组数据
- 为了简单起见，考虑平方损失函数

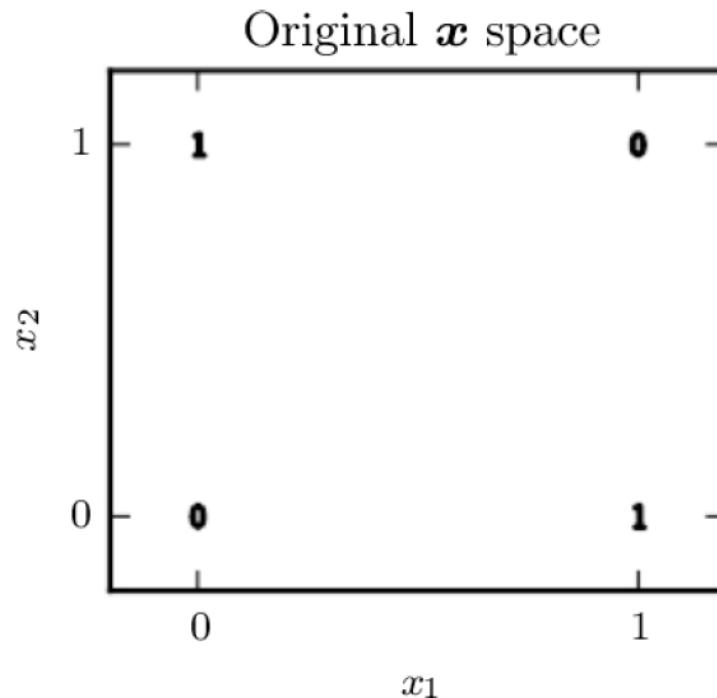
$$J(\theta) = \frac{1}{4} \sum_{x \in X} (f^*(\mathbf{x}) - f(\mathbf{x}; \theta))^2$$

- 为  $f(\mathbf{x}; \theta)$  选择一种形式：考虑线性模型，其中参数  $\theta$  为  $\mathbf{w}$  和  $b$ ：

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^T \mathbf{w} + b$$

# 线性模型

- 回忆先前章节: 可解得  $w = 0, b = \frac{1}{2}$
- 线性模型不能表示XOR, 输出值恒为0.5

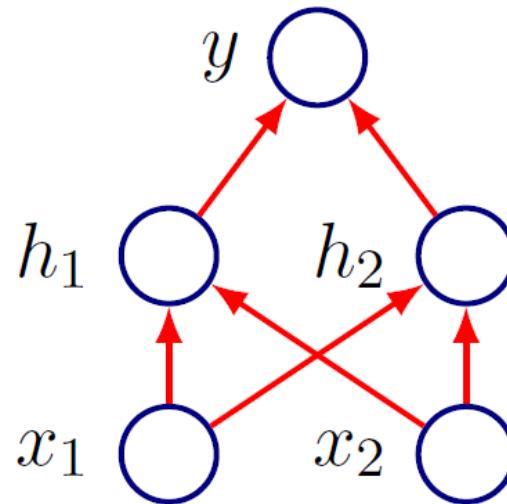


# XOR

---

- 怎么处理XOR的问题?
- 学习一个不同的特征空间，而在该新的特征空间中线性模型能够起效

# XOR



- 定义一个含有隐藏单元的前馈网络，隐藏单元的值由  $f^{(1)}(x; W, c)$  计算得到
- 使用隐藏单元的值作为第二层的输入，即计算输出
$$y = f^{(2)}(h; w, b)$$
- 完整的模型： $f(x; W, c, w, b) = f^{(2)}(f^{(1)}(x))$
- $f^{(1)}$ 是什么形式？可以是线性的吗？

# XOR

---

- 考虑非线性激活函数  $g(z) = \max\{0, z\}$
- 完整的网络模型：

$$f(\mathbf{x}; W, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\{0, W^T \mathbf{x} + \mathbf{c}\} + b$$

- 注意：上述激活是element-wise的

# A Solution

---

- 取

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

- 输入矩阵：

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

# A Solution

---

- 计算第一层输出，首先计算  $XW$

$$XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

- 接着是  $XW + c$

$$XW + c = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

- 注：该示例中存在维度不匹配，这里按照能够运算的方式进行

# A Solution

---

- 下一步:调整输出

$$\max\{0, XW + \mathbf{c}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

- 最后计算

$$\mathbf{w}^T \max\{0, XW + \mathbf{c}\} + b$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

# A Solution

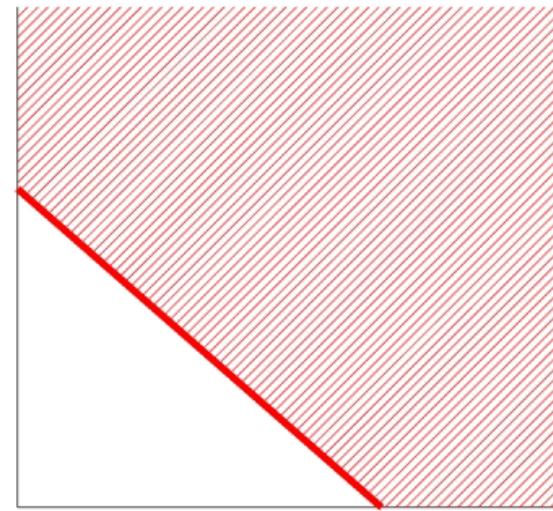
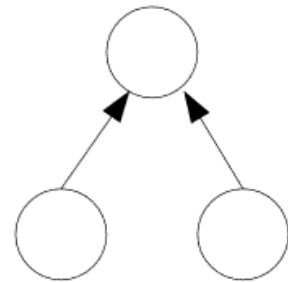
---

- 能够正确分类输入集合中的每个例子
- 这是手工构造的示例，因此看起来简单直接
- 对于更复杂的函数，网络结构则会更加复杂（例如层数加深），我们将使用基于梯度的学习

# 更进一步

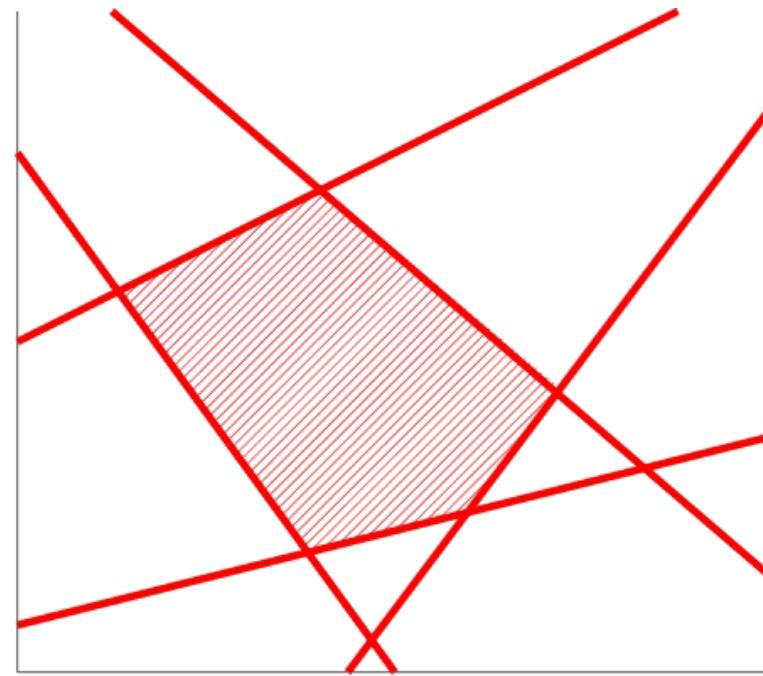
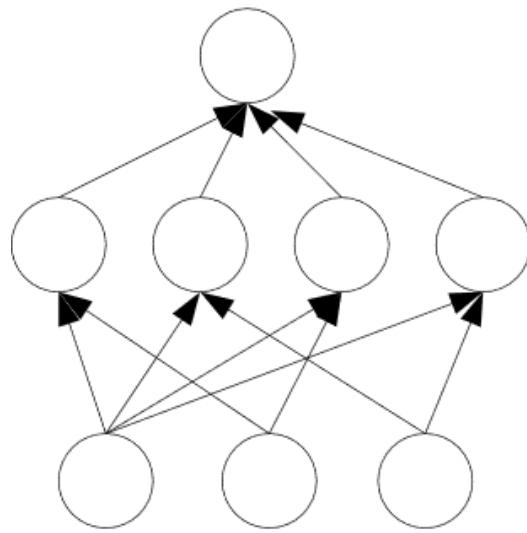
---

1 layer of  
trainable  
weights



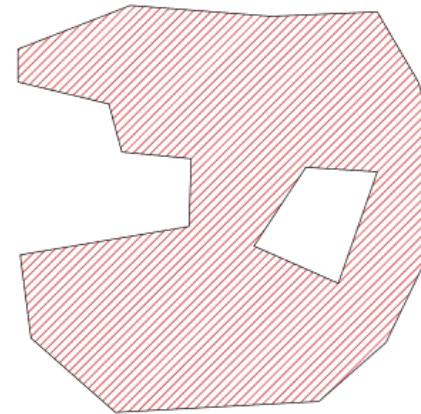
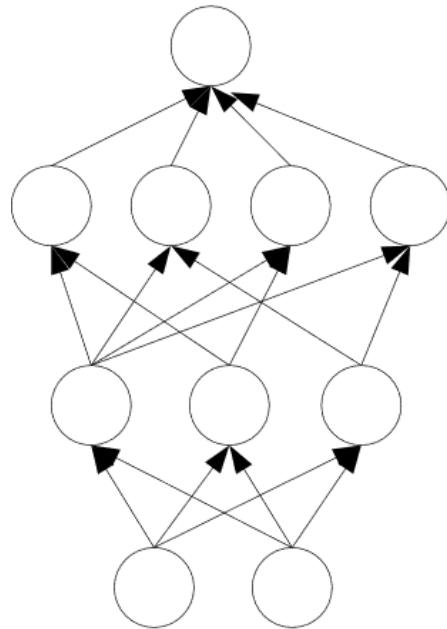
separating hyperplane

# 更进一步



convex polygon region

# 更进一步



composition of polygons:  
convex regions

# 总结

---

- 设计和训练一个神经网络与训练任何其他具有梯度下降的机器学习模型没有太大区别
- 最显著的差异：许多损失函数变成非凸函数
- 与凸优化不同，收敛性并不能够保证
- 应用梯度下降：需要指定损失函数，和输出表达

# 目录

---

1 上节回顾

2 前馈网络简介

3 基本结构单元

4 网络结构设计

5 反向传播算法

# 损失函数

# 损失函数

- 与前面的参数化模型相似的选择: 定义一个分布  $p(\mathbf{y}|\mathbf{x}; \theta)$  , 并运用极大似然原理
- 我们可以利用训练数据和模型预测之间的交叉熵值作为损失函数:

$$J(\theta) = \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{data}} \log p_{model}(\mathbf{y}|\mathbf{x})$$

- 根据  $\log p_{model}$  的特定形式而发生变化
- 例如: 如果有  $p_{model}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; f(\mathbf{x}; \theta), I)$  , 则我们可以得到损失函数:

$$J(\theta) = \frac{1}{2} \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{data}} \|\mathbf{y} - f(\mathbf{x}; \theta)\|^2 + \text{Constant}$$

# 损失函数

---

- 优点：根据指定 $p(y|x)$ ，而后可以自动获得对应的损失函数 $\log p(y|x)$
- 输出单元的选择对于损失函数的选择非常重要



# 输出单元

# 线性单元

---

- 给定特征 $h$ , 一层线性输出单元:

$$\hat{y} = W^T h + b$$

- 常用于产生条件高斯分布的均值:

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; \hat{\mathbf{y}}, I)$$

- 最大化对数似然  $\Rightarrow$  最小化平方误差

# Sigmoid

- 任务:预测一个二进制变量
- 使用sigmoid单元:

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{h} + b)$$

- 损失函数:

$$J(\theta) = -\log p(y|\mathbf{x}) = -\log \sigma((2y - 1)(\mathbf{w}^T \mathbf{h} + b))$$

- **好处:** 只有当模型有正确答案时才会饱和, 即当 $y=1$ 和 $(\mathbf{w}^T \mathbf{h} + b)$ 为较大的正值, 反之亦然
- 当 $(\mathbf{w}^T \mathbf{h} + b)$ 符号错误时, 会返回一个合适的梯度值

# Softmax

---

- 生成向量  $\hat{\mathbf{y}}$  , 其中  $\hat{y}_i = p(y = i | \mathbf{x})$
- 线性层首先产生unnormalized log probabilities:

$$\mathbf{z} = W^T \mathbf{h} + \mathbf{b}$$

- Softmax:

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- 由于我们希望最小化  $p(y = i; \mathbf{z})$  , 因此计算Softmax的对数形式:

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j)$$

# 优点

---

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j)$$

- $z_i$  项不会饱和，使得学习更加容易
- 最大化对数似然会使得  $z_i$  增大，同时促进所有其他  $z$  减小 (Softmax 鼓励竞争)
- 对数似然损失函数 ( $\sim z_i - \max_j z_j$ ) 给最严重的的错误预测最大的惩罚
- 如果模型已经能够得到正确答案，则  $\log \sum_j \exp(z_j) \approx \max_j z_j$  和  $z_i$  大致抵消
- 错分的样本主导学习的进程



# 隐藏单元

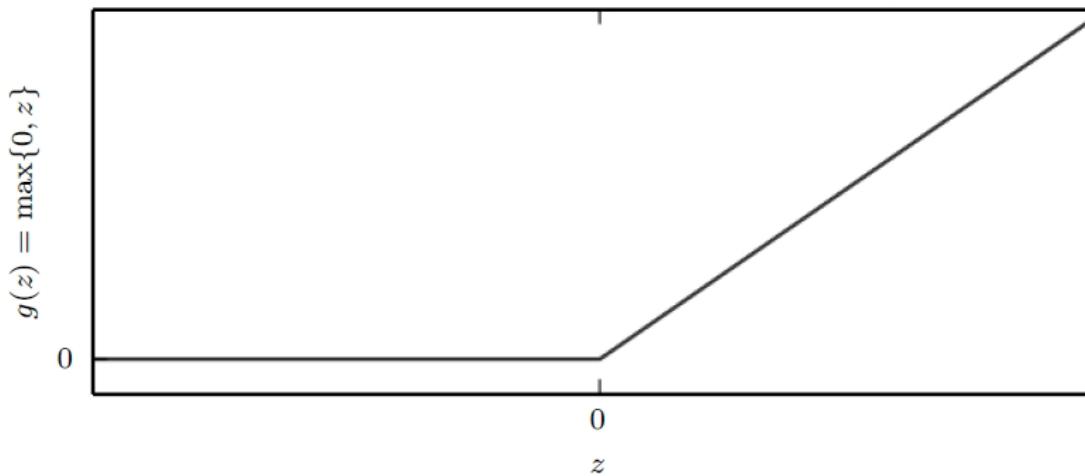
# 隐藏单元

---

- 输入  $x \rightarrow$  计算仿射变换  $z = W^T x + b \rightarrow$  应用 element-wise 非线性函数  $g(z) \rightarrow$  得到输出  $g(z)$
- 如何选择  $g$  ?
- 隐藏单元的设计是一个活跃的研究领域

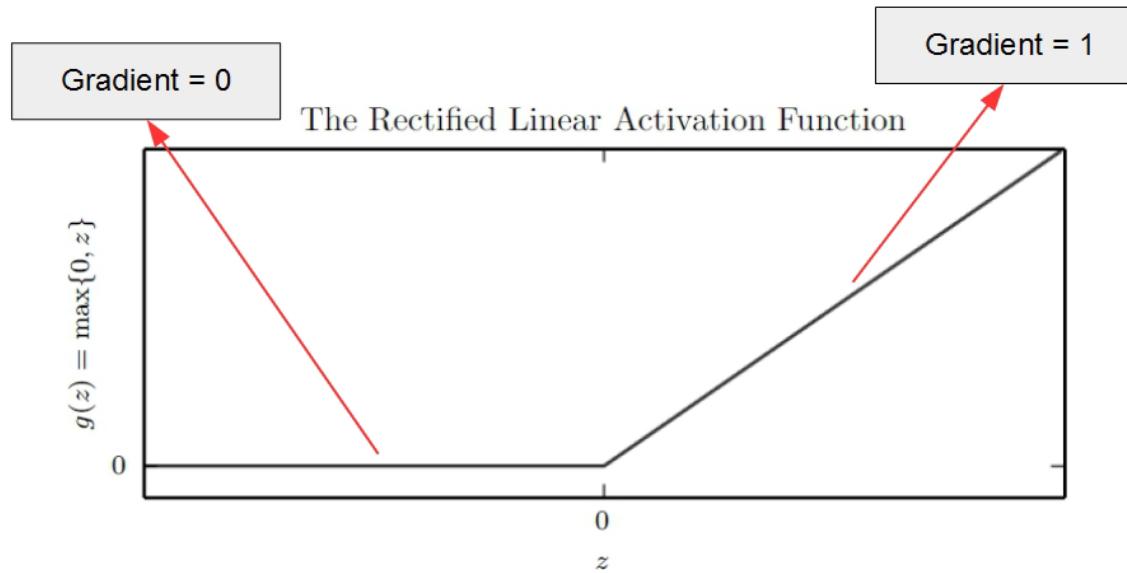
# Rectified Linear Units (ReLU)

The Rectified Linear Activation Function



- 激活函数:  $g(z) = \max\{0, z\}$  , 其中  $z \in \mathbb{R}$
- 在仿射变换之上  $\max\{0, W\mathbf{x} + \mathbf{b}\}$
- 两层网络: 第一层  $\max\{0, W_1^T \mathbf{x} + \mathbf{b}_1\}$
- 第二层  $W_2^T \max\{0, W_1^T \mathbf{x} + \mathbf{b}_1\} + \mathbf{b}_2$

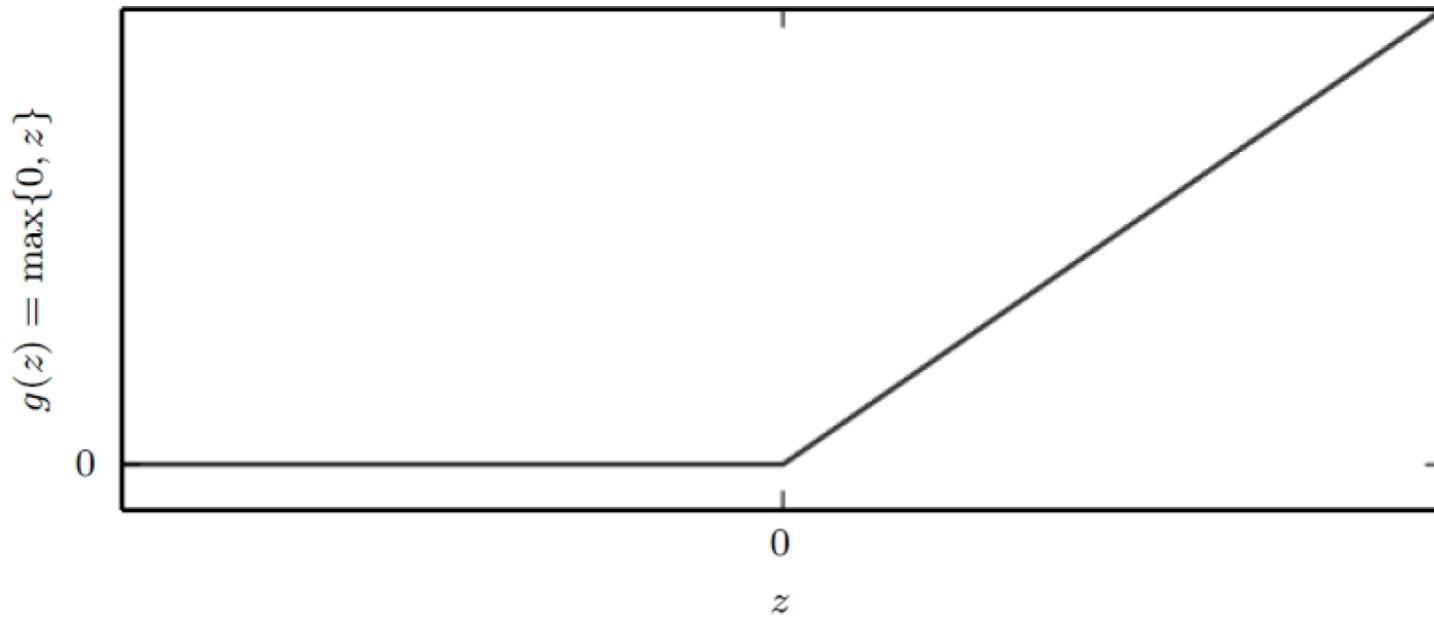
# ReLU



- 类似于线性单元，容易优化！
- 能够提供较大的且较为恒定的梯度
- **Good practice:** 将b初始化为一个小正值(例如0.1)
- 确保初始时单元是活跃的，使得大多数输入及其后续计算变量可以通过

# ReLU

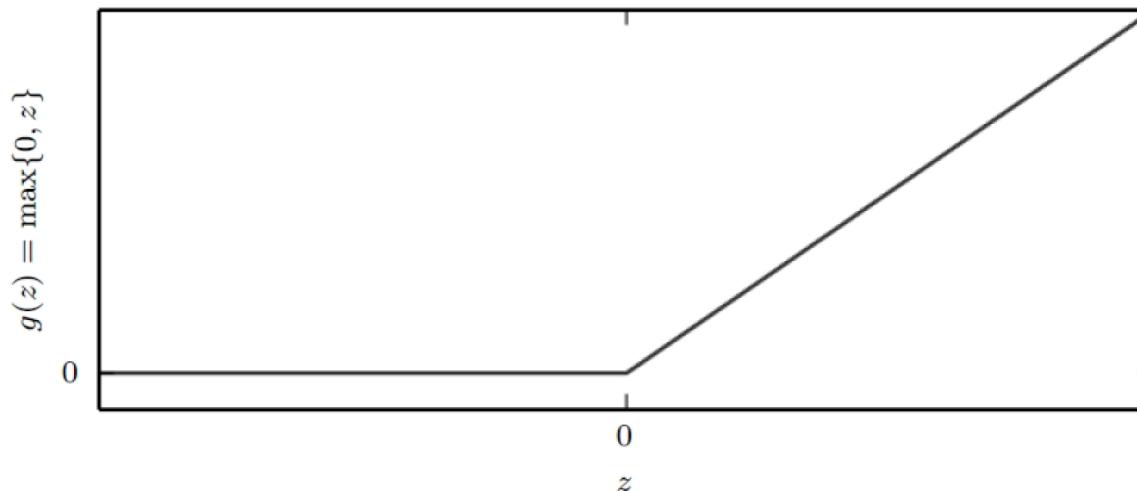
The Rectified Linear Activation Function



- 并不是处处可微
- 在  $z = 0$  处返回单侧导数
- 无论如何基于梯度的优化都会受到数值误差的影响

# ReLU

The Rectified Linear Activation Function



- 优势:
  - 提供数值较大且恒定的梯度(不会饱和)
  - 迅速收敛, 收敛速度比sigmoid或tanh快得多
- 劣势:
  - 非零中心输出
  - Units “die”: 即不活跃的单元永远不会被更新

# Generalized ReLU

---

- 在  $z_i < 0$  得到一个非零斜率
- $g(z, a)_i = \max\{0, z_i\} + a_i \min\{0, z_i\}$ 
  - **Absolute value rectification:** (*Jarret et al., 2009*) 取  $a_i = 1$ ，得到  $g(z) = |z|$
  - **Leaky ReLU:** (*Maas et al., 2013*) 固定  $a_i$  为一个很小的值 e.g., 0.01
  - **Parametric ReLU:** (*He et al., 2015*) 学习  $a_i$
  - **Randomized ReLU:** (*Xu et al., 2015*)：训练时在固定的区间采样  $a_i$ ，测试的时候则固定
  - ....

# Generalized ReLU

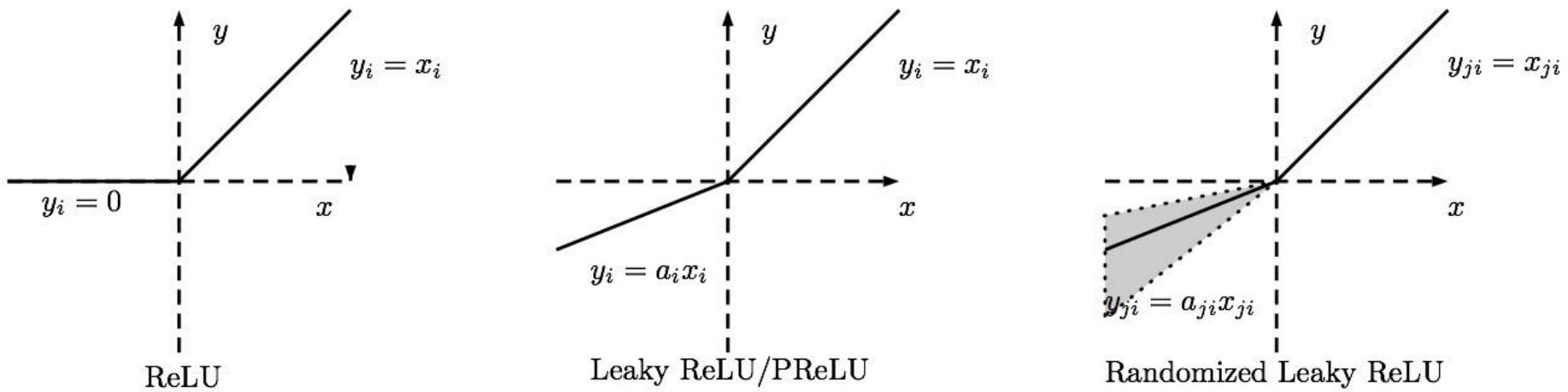


Figure: Xu et al. "Empirical Evaluation of Rectified Activations in Convolutional Network"

# Exponential Linear Units (ELUs)

$$g(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(\exp z - 1) & \text{if } z \leq 0 \end{cases}$$

- ReLU的所有优势 + 没有units die
- 问题：需要进行指数运算

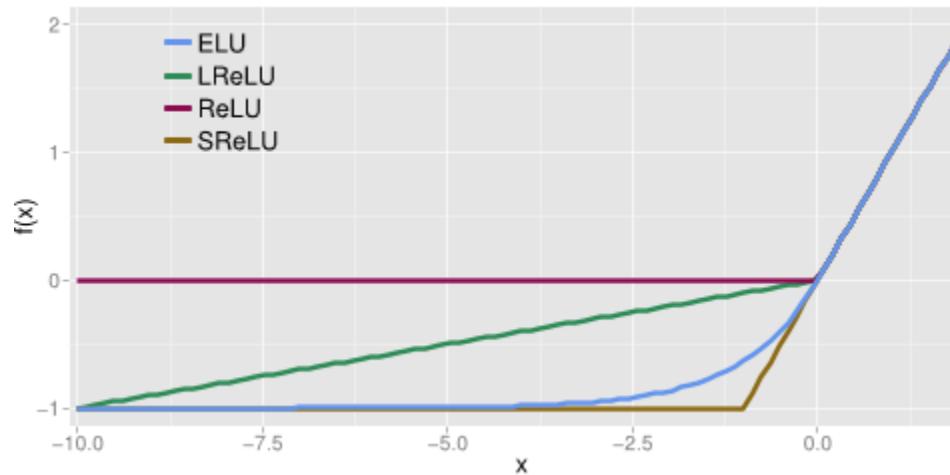
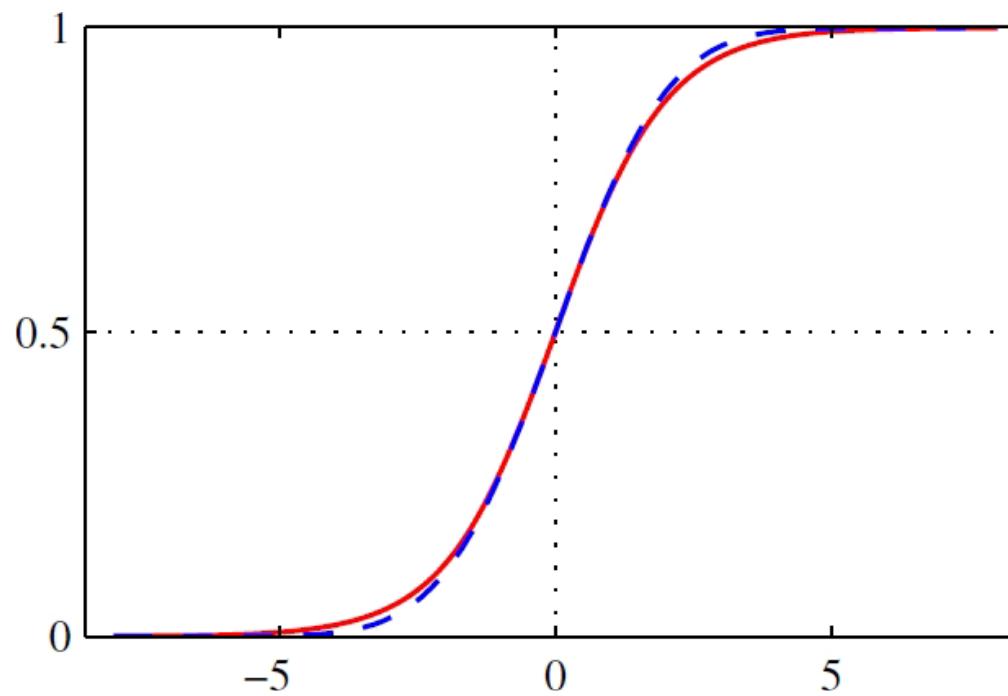


Figure: Clevert et al. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)", 2016

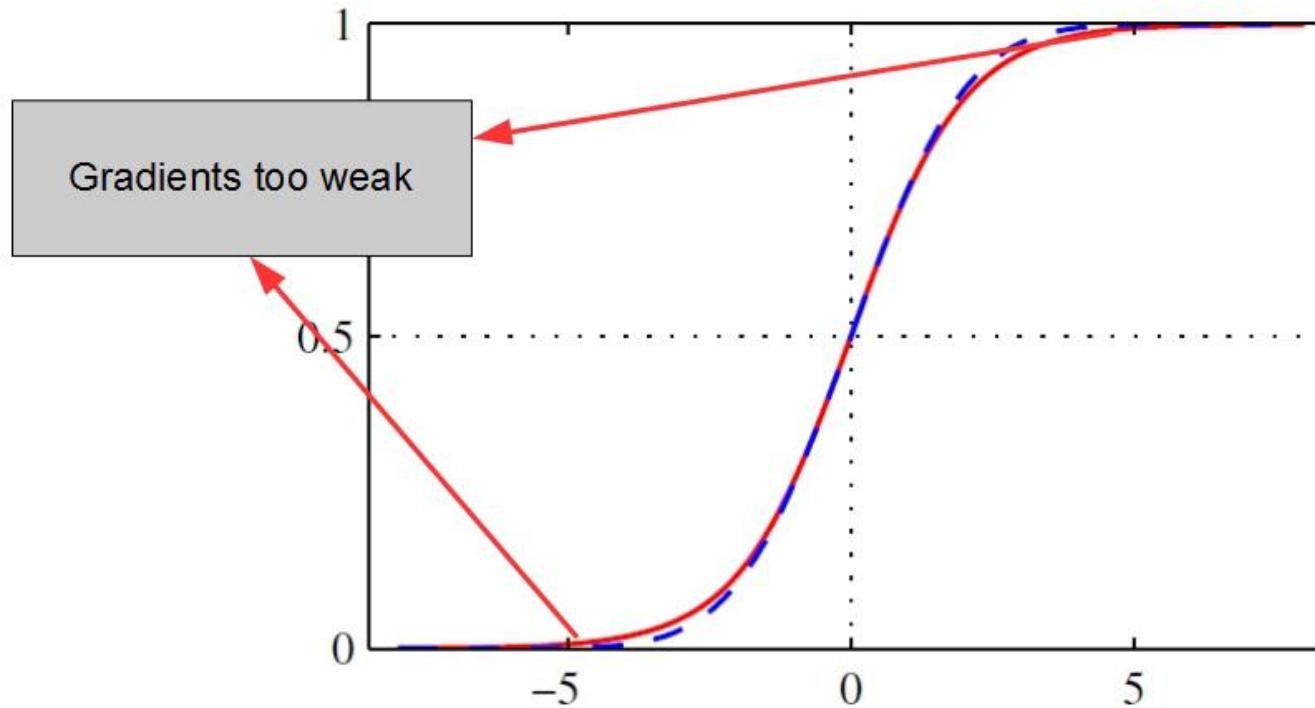
# Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- 将输出移动到范围[0,1]



# Sigmoid



- 问题：大部分领域饱和，只有 $z$ 接近零时非常敏感
- 饱和问题使得基于梯度的学习变得困难

# Tanh

---

- 与sigmoid相关:  $g(z) = \tanh(z) = 2\sigma(2z) - 1$
- 优势: 输出范围[-1,1], 输出以0为中心
- 缺陷: 同样存在饱和问题
- 比sigmoid好, 因为当激活值很小的时候,  $\hat{y} = w^T \tanh(U^T \tanh(V^T x))$  近似  $\hat{y} = w^T U^T V^T x$

# 总结

---

- 前馈网络中尽量不要使用Sigmoid
- 必须使用Sigmoid时，用Tanh代替
- 默认采用ReLU，但要谨慎选择学习率
- 尝试其他的ReLU方式等，也许会获得性能提升

# 目录

---

1 上节回顾

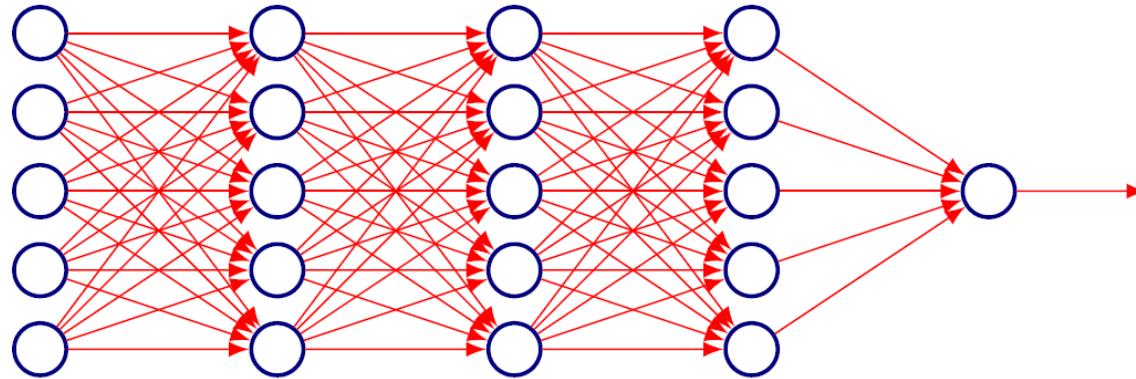
2 前馈网络简介

3 基本结构单元

4 网络结构设计

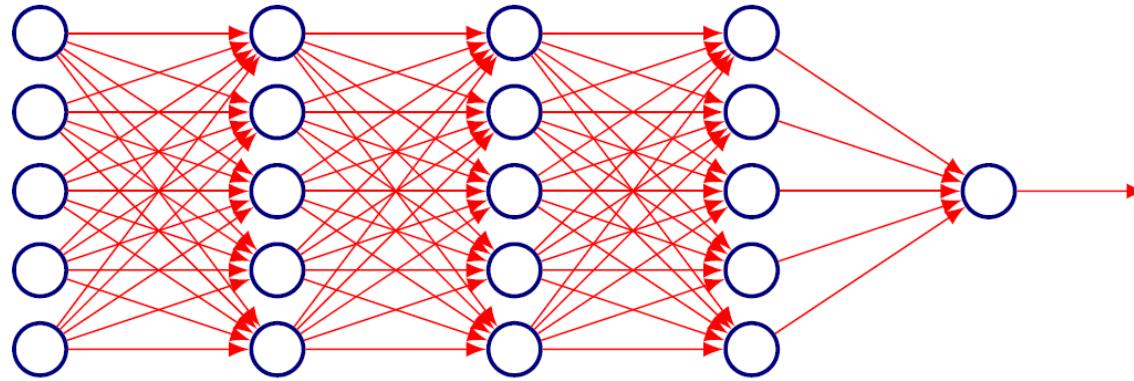
5 反向传播算法

# 结构设计



- 第一层:  $h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)})$
- 第二层:  $h^{(2)} = g^{(2)}(W^{(2)T}h^{(1)} + b^{(2)})$
- 怎么选定深度以及宽度?
- 理论上有多少层就足够?

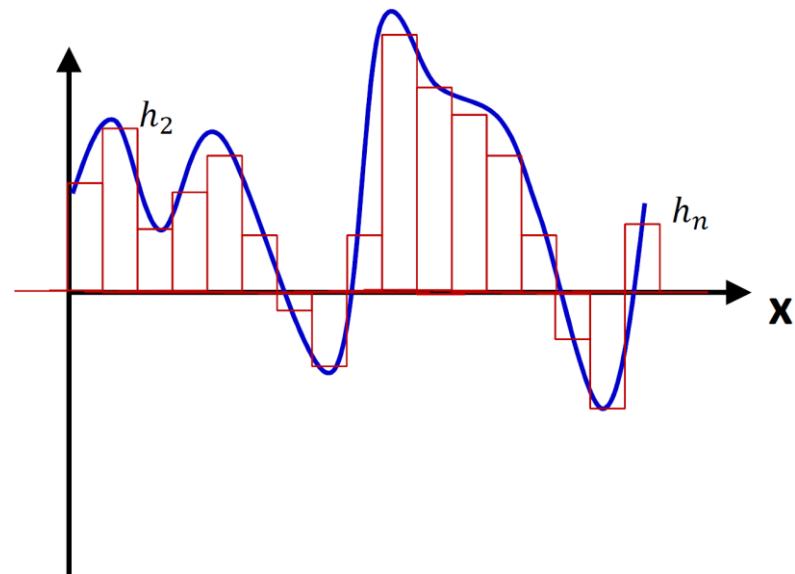
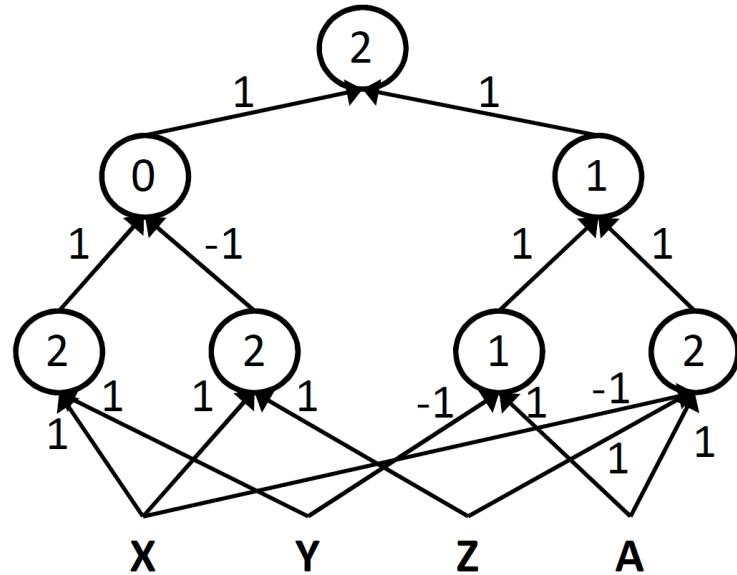
# 深度和宽度



- 神经网络中函数链的长度称为深度
- 神经网络隐含层的维数称为其宽度

# MLP构造函数

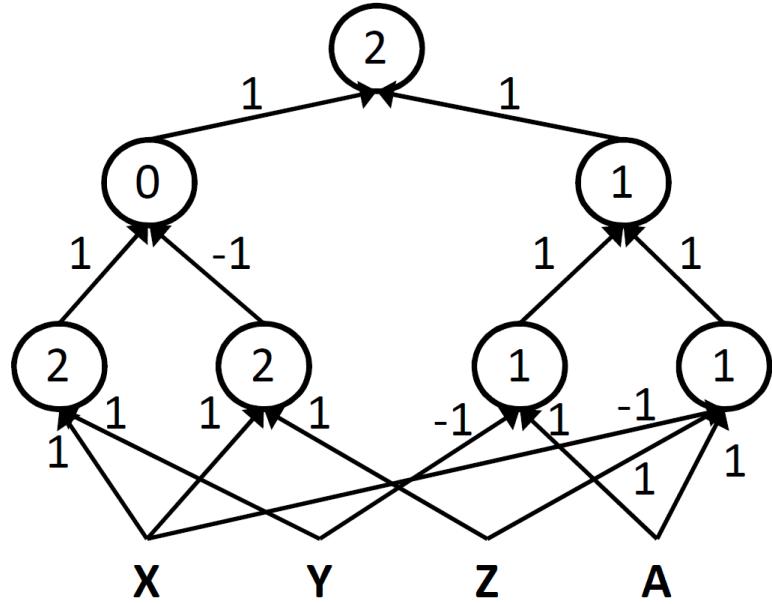
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | \bar{(X \& Z)})$$



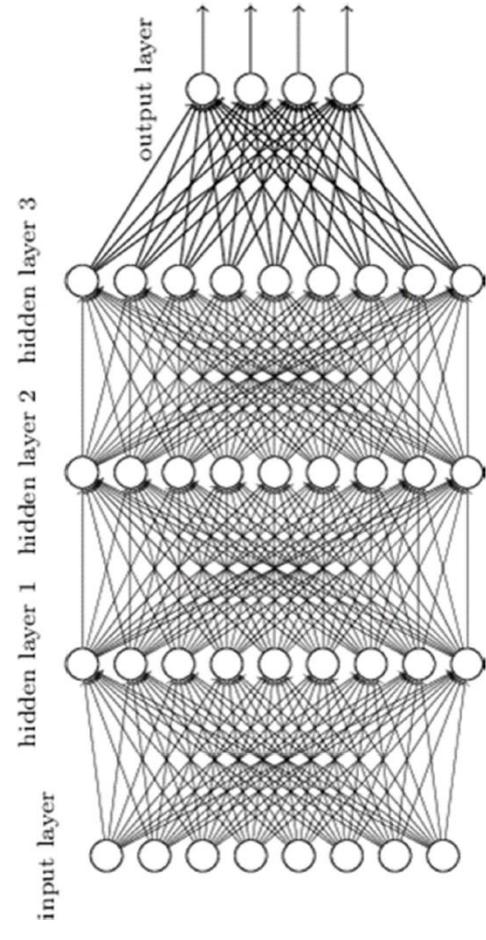
- MLPs可以构造任何布尔函数
- MLPs可以构造任何分类器
- MLPs可以构造通用逼近器

# MLP构造布尔函数

$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& Z))$$



Deep neural network



- MLPs是通用布尔函数
  - 任意数量的输入和输出的任何函数
- 需要多少“层”呢？

# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

- 布尔函数就是一个真值表

# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$

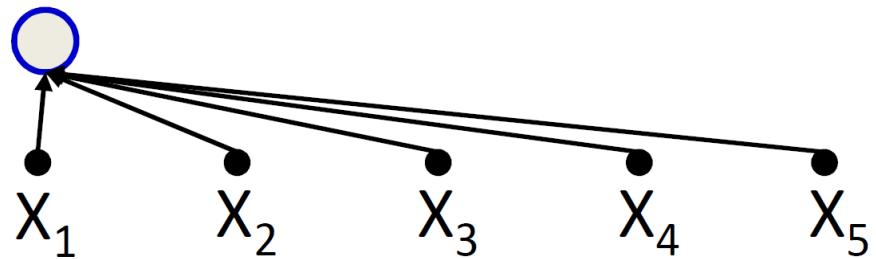
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 X_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



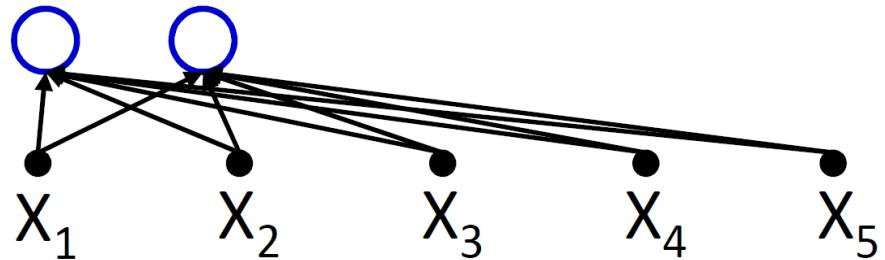
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \textcircled{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 X_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



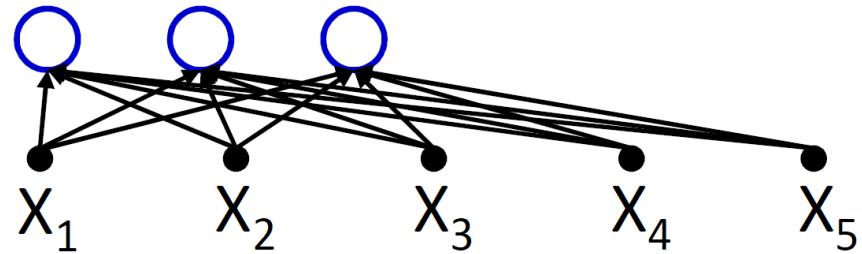
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 X_3 X_4 X_5$$



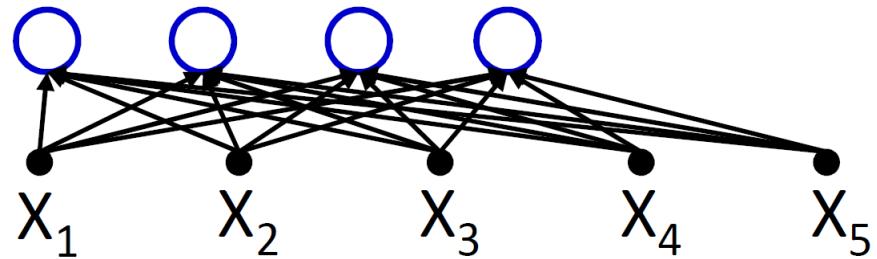
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 +$$
  
$$\textcircled{X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5} + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



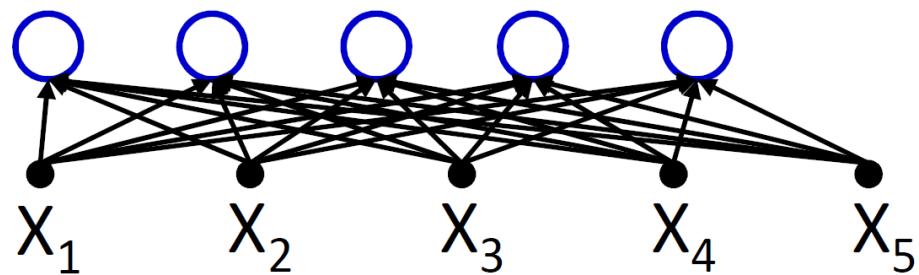
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + \text{X}_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



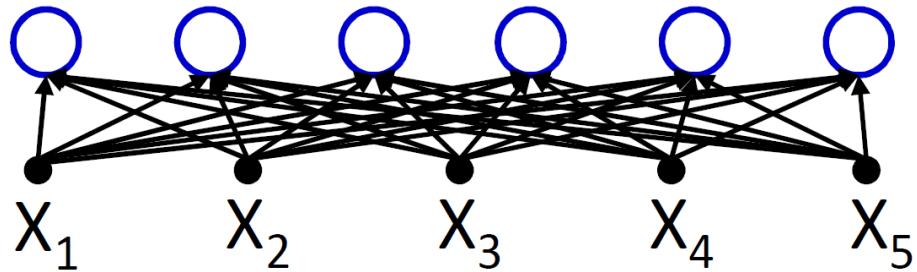
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + \text{X}_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



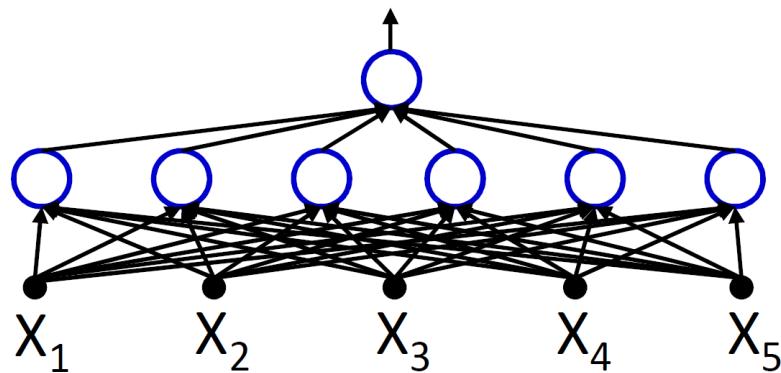
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



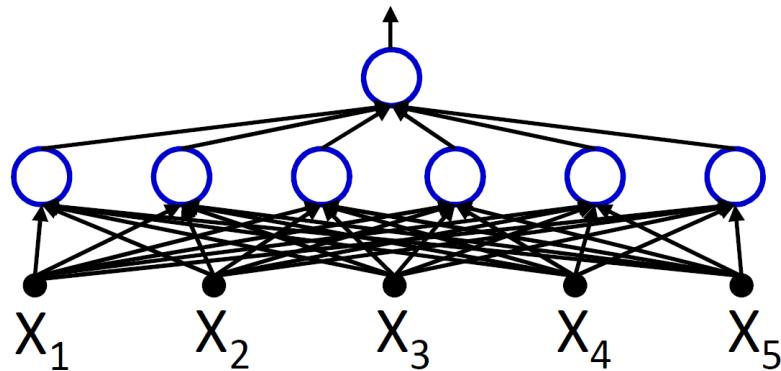
# 布尔MLP有多少层?

Truth Table

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $Y$ |
|-------|-------|-------|-------|-------|-----|
| 0     | 0     | 1     | 1     | 0     | 1   |
| 0     | 1     | 0     | 1     | 1     | 1   |
| 0     | 1     | 1     | 0     | 0     | 1   |
| 1     | 0     | 0     | 0     | 1     | 1   |
| 1     | 0     | 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 1     | 1   |

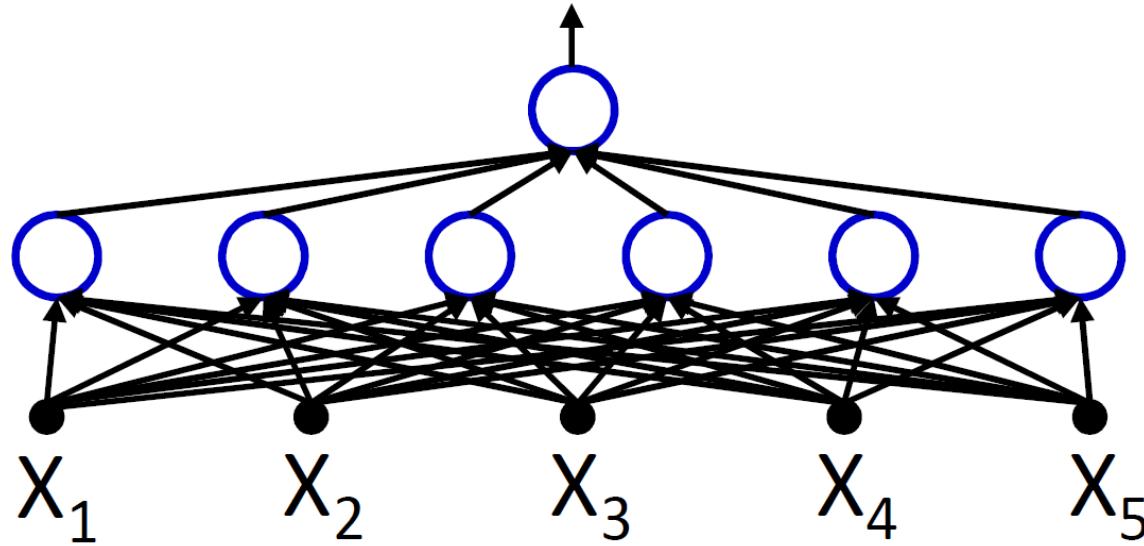
真值表显示了输出为1的所有输入组合

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



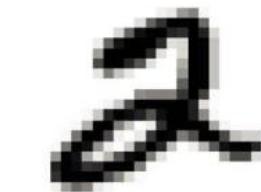
- 任何真值表都可以用这种方式表示!
- 一层的MLP能够表示任何布尔函数

# 网络参数规模

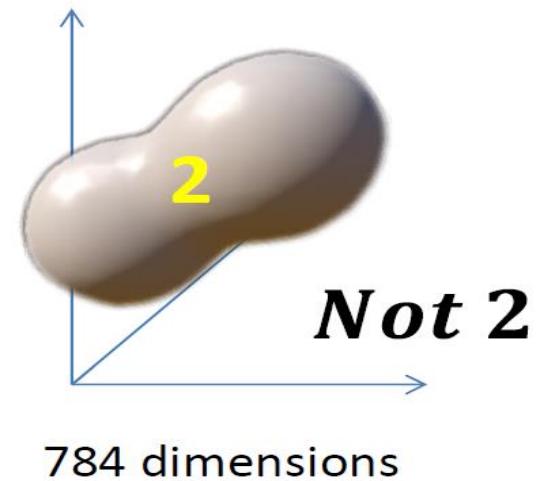
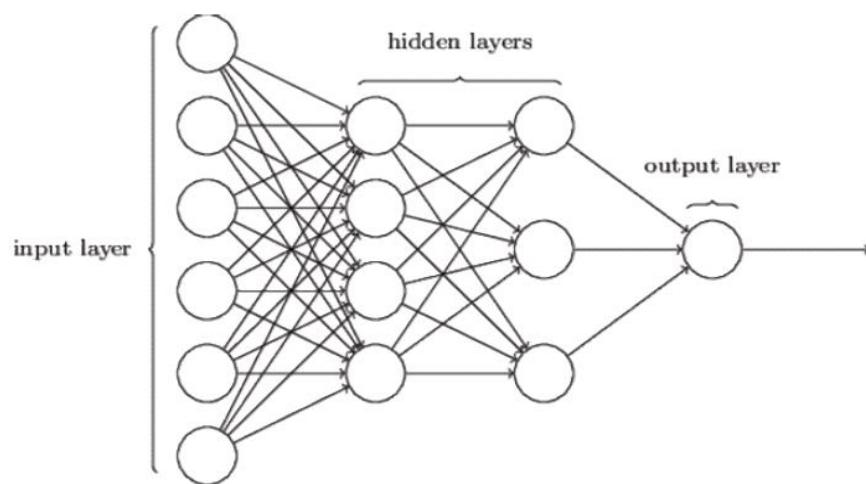


- 网络中参数的数量是连接的数量，本例中为30个
- 参数规模是软件和硬件实现中非常重要的值

# MLP构成通用分类器

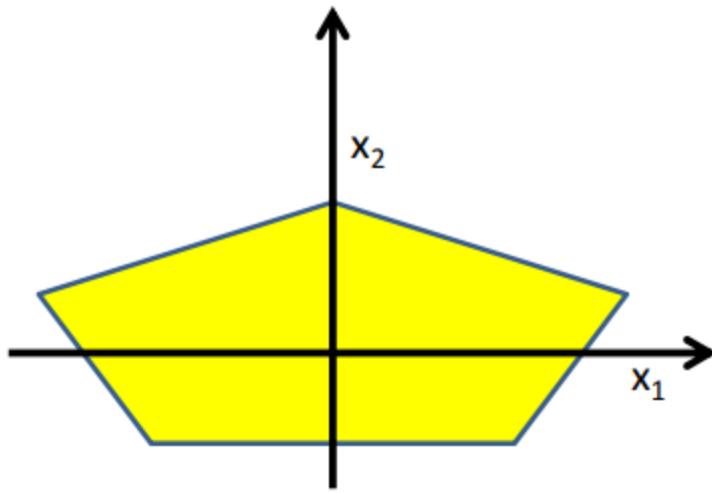


784 dimensions  
(MNIST)



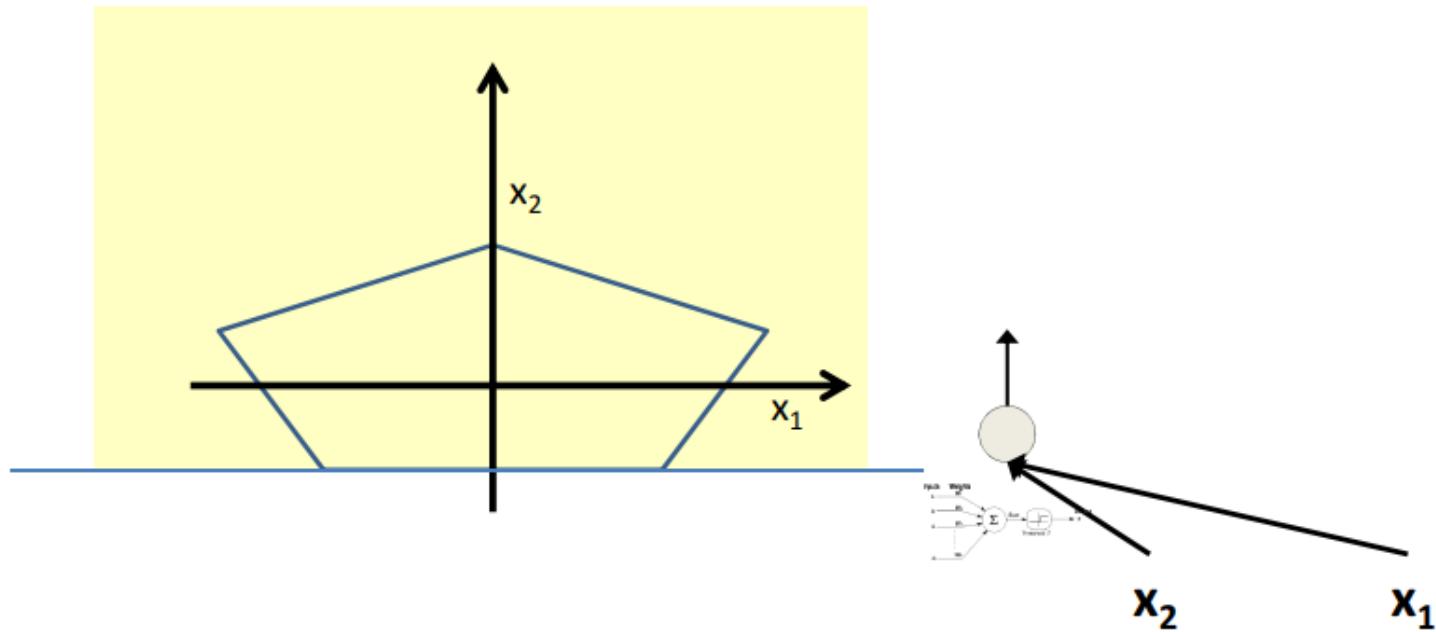
- MLP是实际输入的函数
- MLP是一个在实数空间中寻找复杂“决策边界”的函数

# MLP构成通用分类器

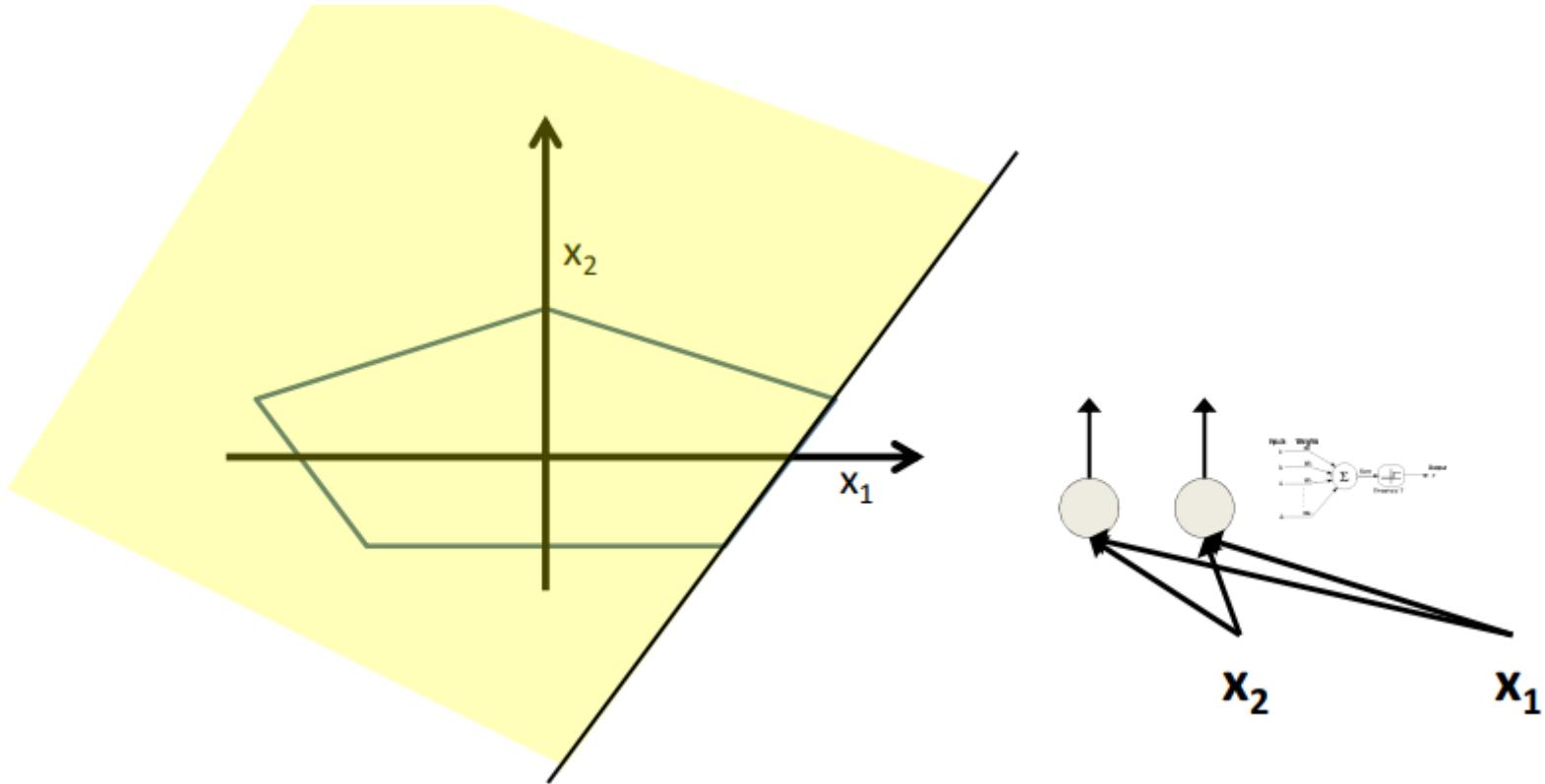


- MLPs 可以拟合任何分类边界
- 单层MLP可以对任何分类边界进行建模
- **MLP是通用分类器**

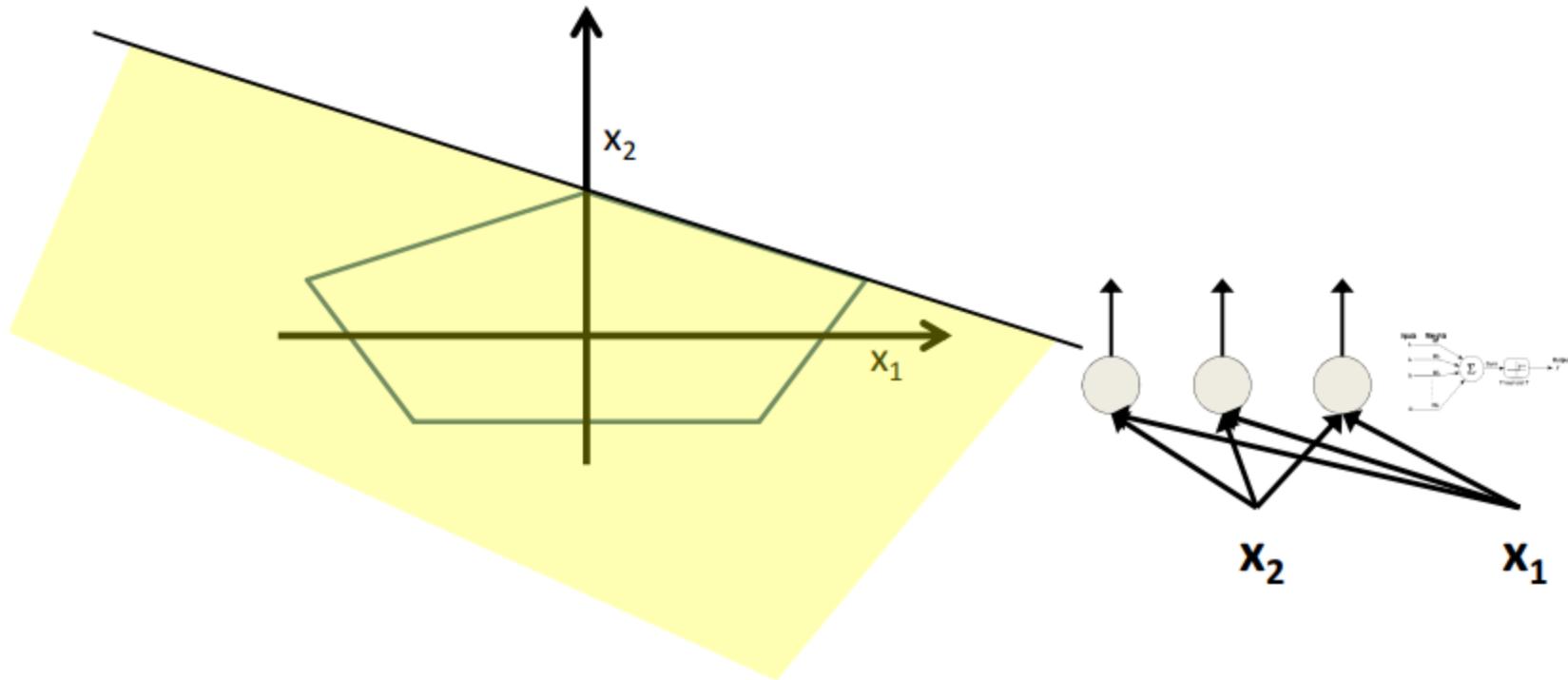
# MLP构成通用分类器



# MLP构成通用分类器



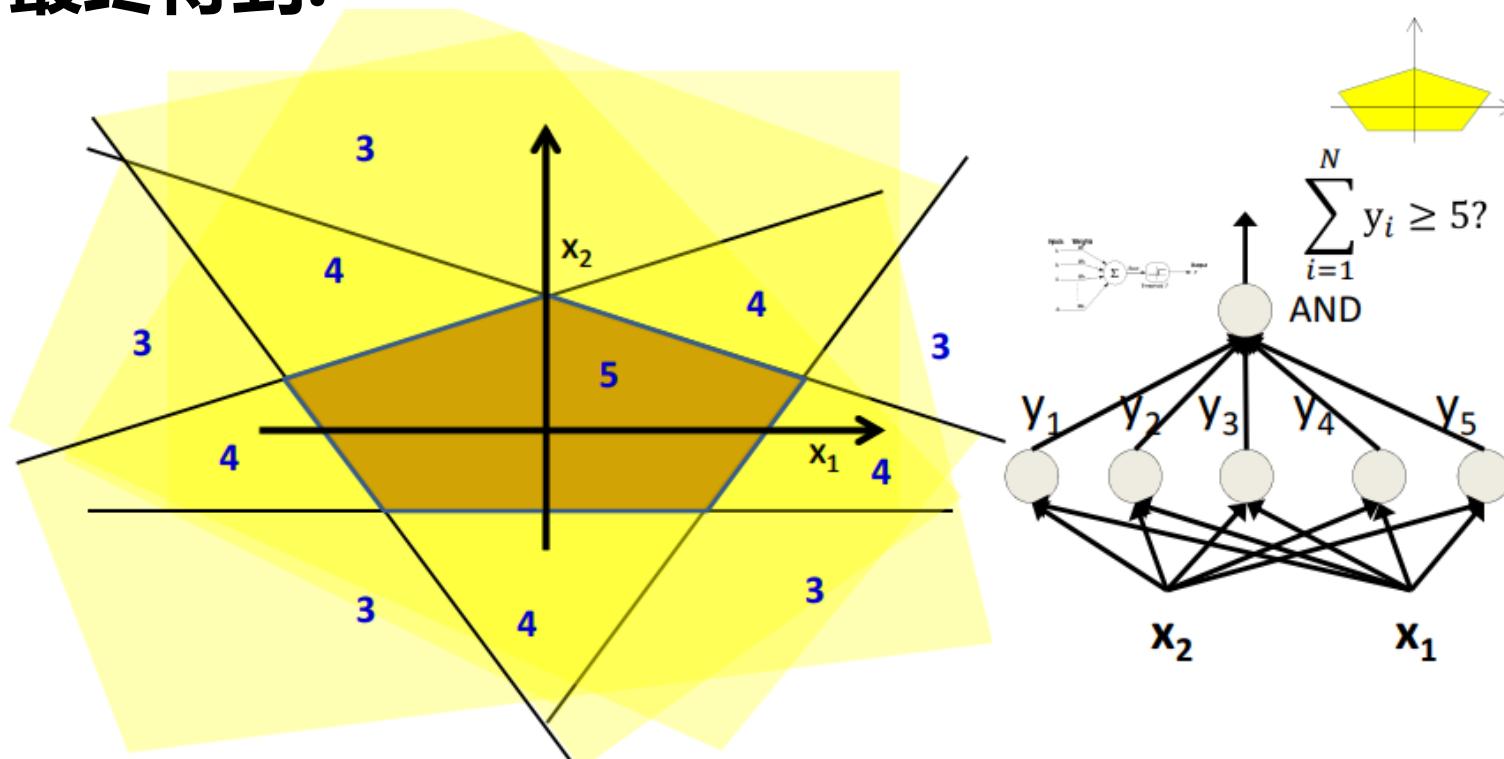
# MLP构成通用分类器



以此类推.....

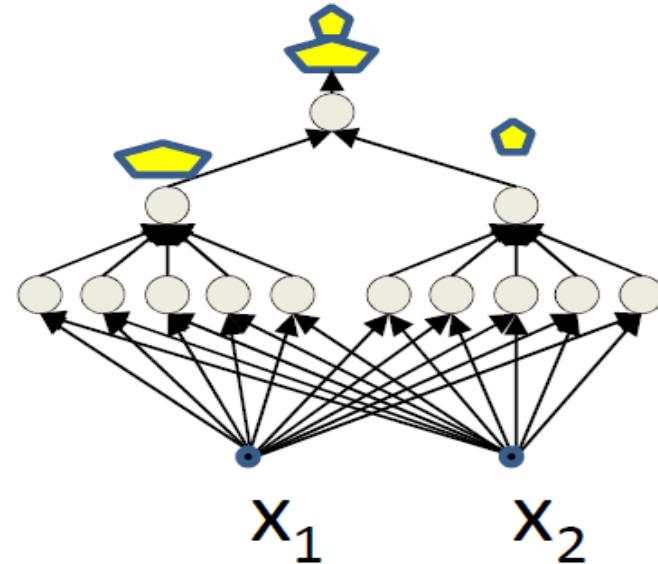
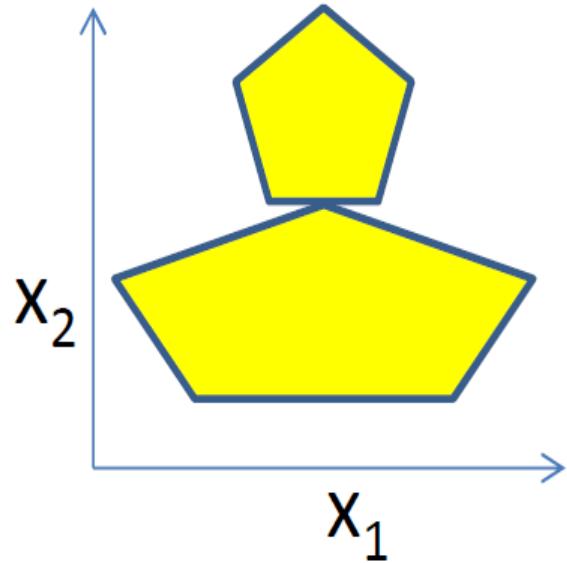
# MLP构成通用分类器

最终得到：



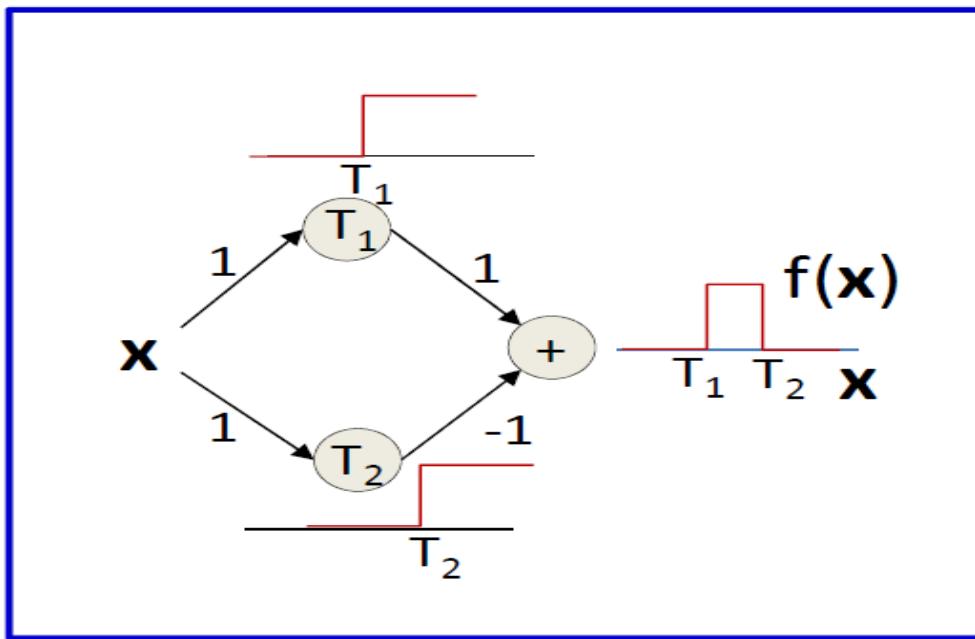
- 类似的可以推广到高维空间
- 理论上平面内的任何分界面都可以类似的利用MLP进行拟合
- 只用了一层隐藏单元，理论最优深度可以为一层
- **MLP是通用分类器**

# MLP构成通用分类器



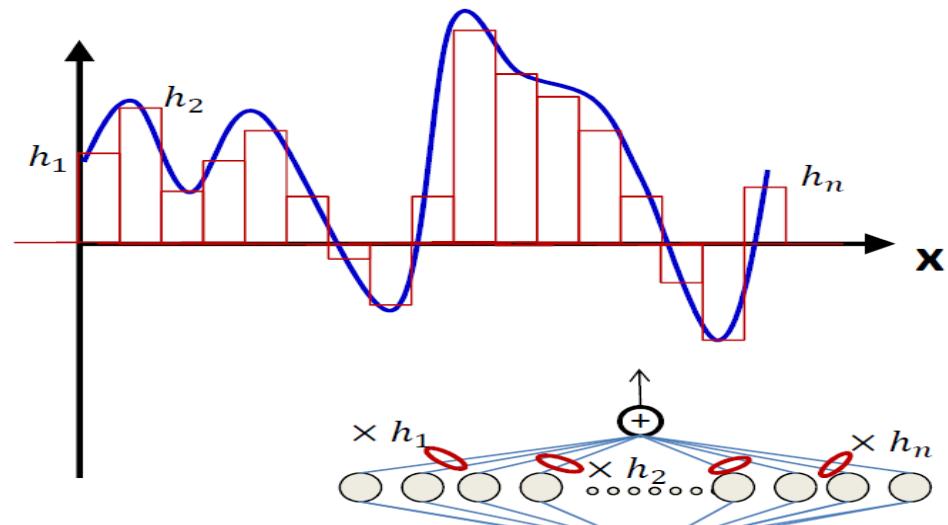
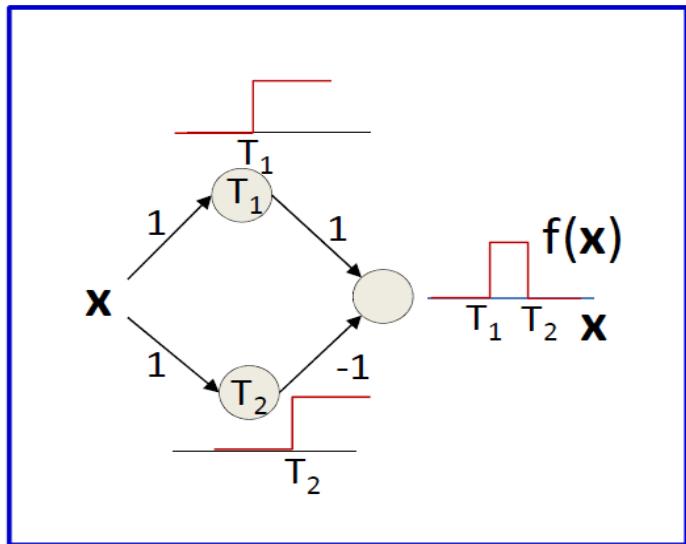
- 更深的网络需要的更少的神经元
- 深度与宽度的折衷

# MLP用来连续回归



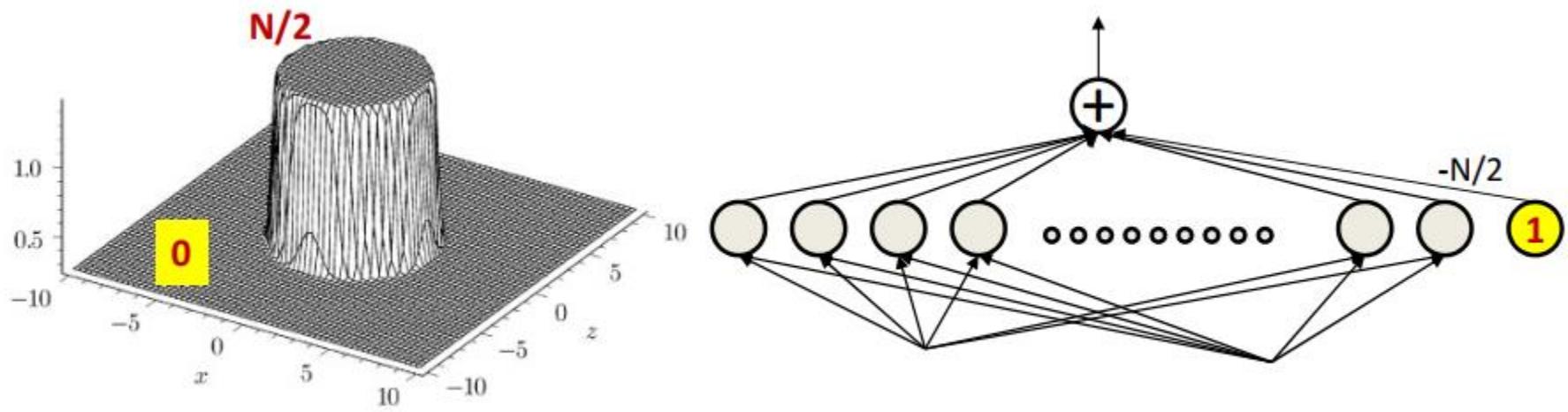
- 一个简单的3单元MLP加上一个“求和”输出单元，就可以在一个输入上产生一个“平方脉冲”
  - 只有当输入位于 $T_1$  和  $T_2$ 之间时，输出才为1
  - $T_1$  和  $T_2$ 可以任意指定

# MLP用来连续回归



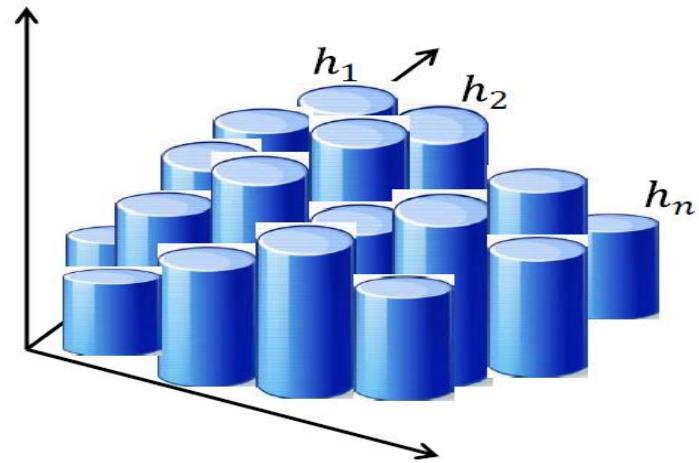
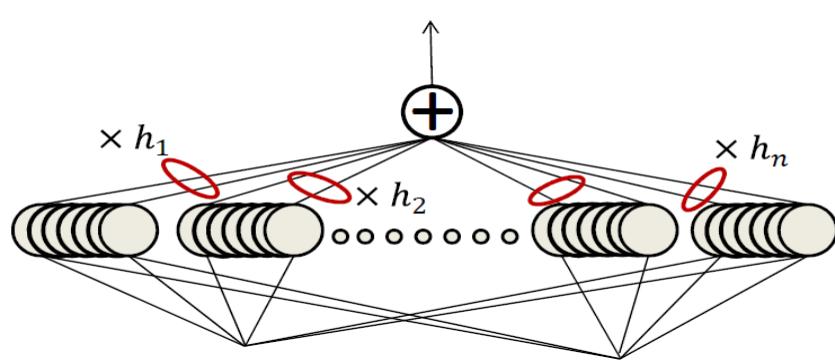
- 一个简单的3单元MLP可以在输入端产生一个“平方脉冲”
- 具有多个单元的MLP可以对输入上的任意函数进行建模(任意精度, 只需要使单个脉冲变窄)
- 单层MLP可以对单个输入的任意函数建模

# MLP构造连续值函数



- MLP可以组成任意维数的任意函数!
  - 只需要一层, 由缩放和平移的圆柱体的和构成
  - 通过使得圆柱体变“瘦”, 实现任意精度

# MLPs构造连续值函数



- 利用MLP簇构成高维空间的任意函数；
- MLP是一个通用逼近器！
- 到目前为止只解释了输出单元只进行求和操作的情况，即没有“激活”；

# 更一般的

---

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output with some squashing non-linearity in hidden units can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- 说明不管我们要学习什么函数，一个规模足够的MLP可以表示这个函数
- 但不能保证我们的训练算法能够学习到这个函数
- 没有给出关于网络将有多大规模的说明(最坏情况下是指数大小)
- 之前展示了一些有启发性的结果

# 深度的优势

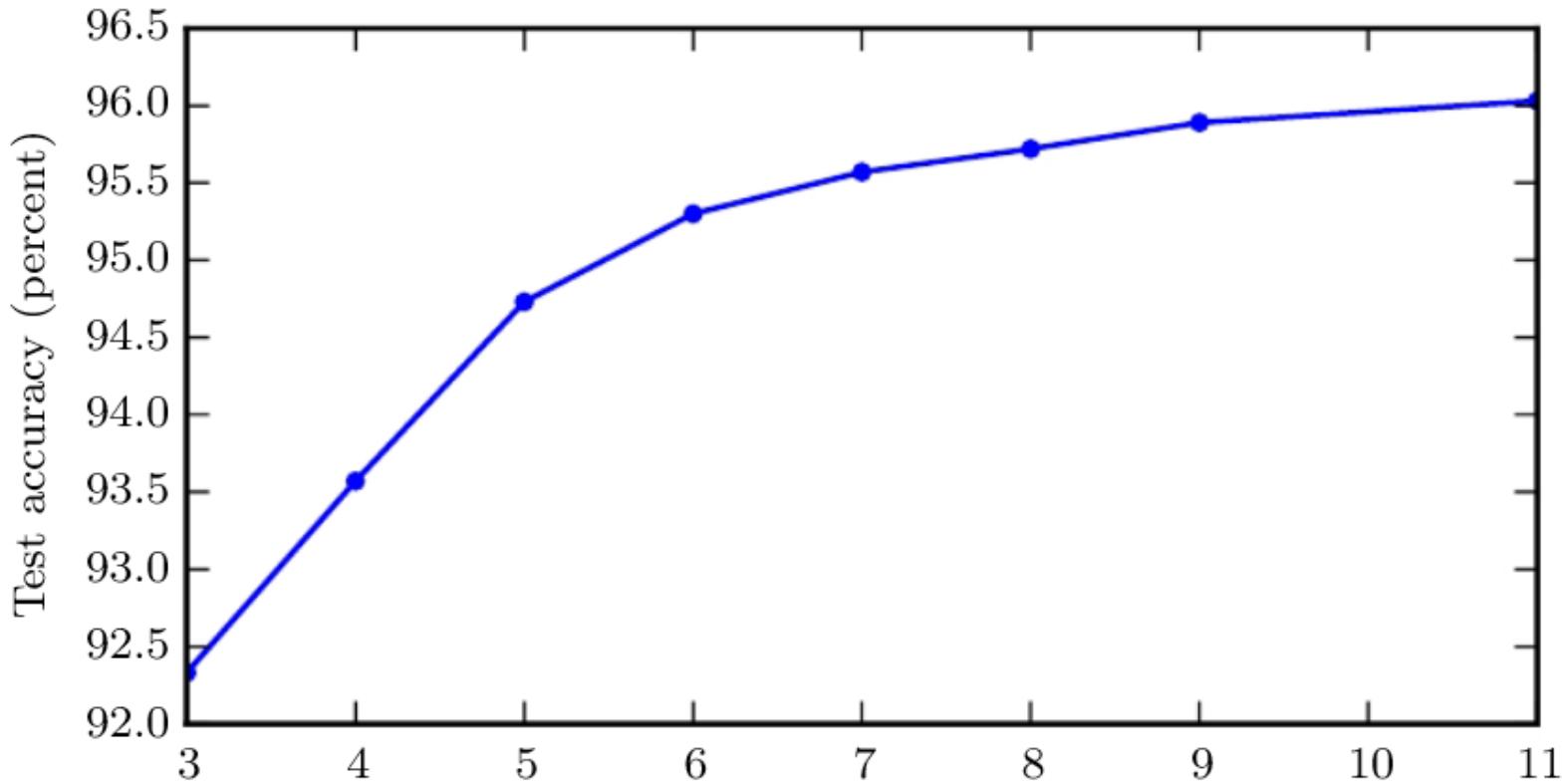
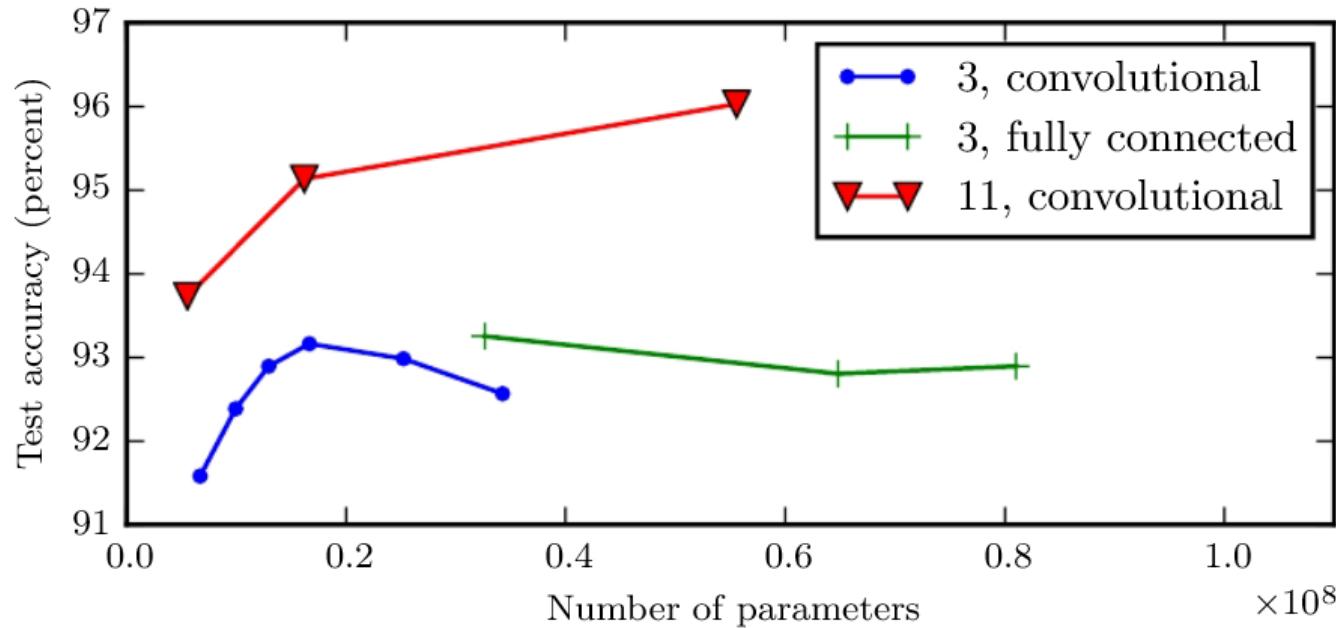


Figure: Goodfellow et al., 2014

# 深度的优势



- 对照实验表明，其他因素对模型尺寸的增加不会产生同样的效果

Figure: Goodfellow et al., 2014

# 目录

---

1 上节回顾

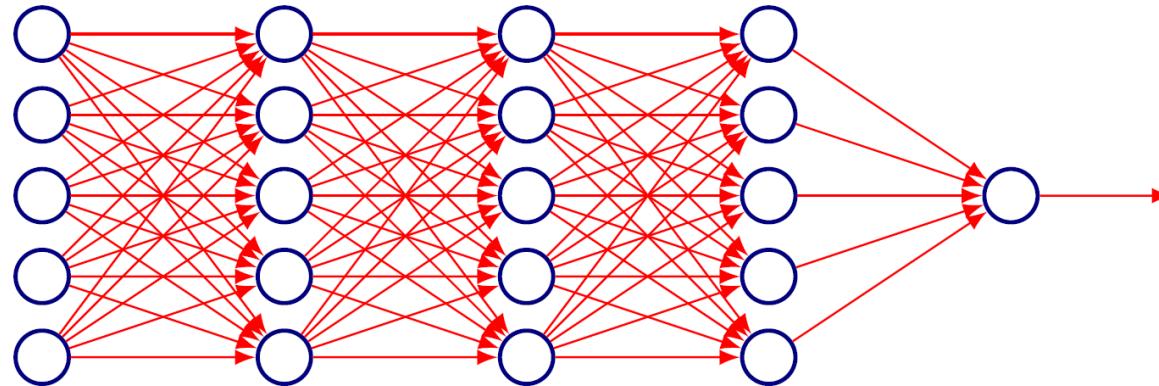
2 前馈网络简介

3 基本结构单元

4 网络结构设计

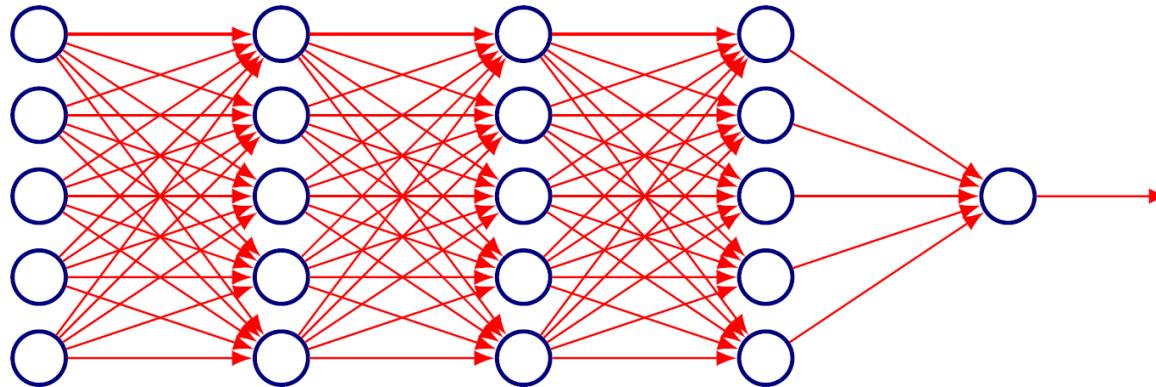
5 反向传播算法

# 如何学习权值?



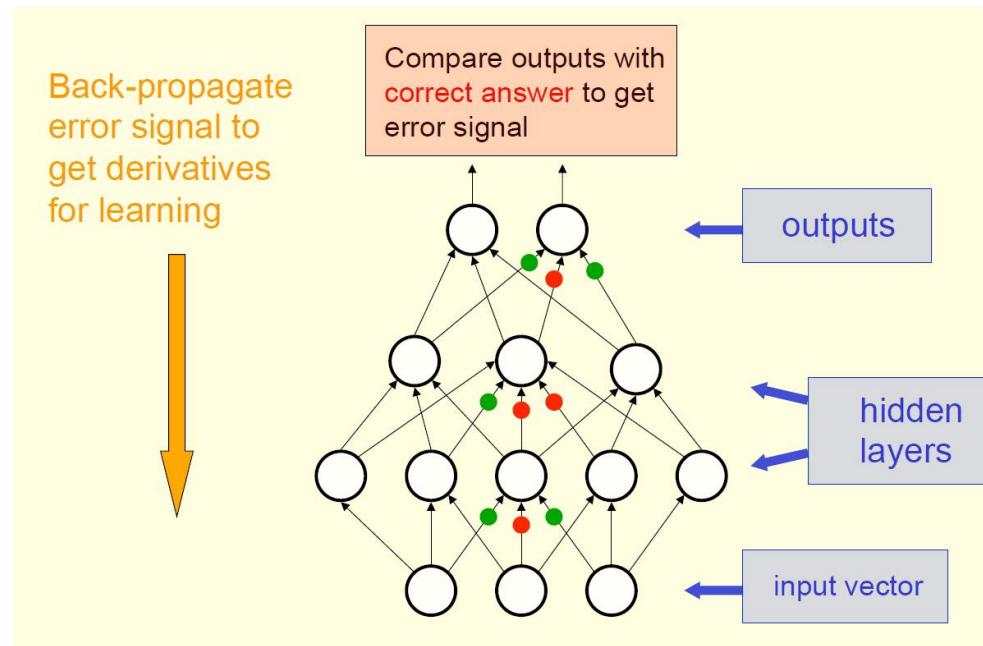
- **初步想法**: 随机扰动一个权重, 看看它是否提高了性能, 而后保存更改
- **非常低效**: 对于一个权重的改变, 需要在样本集上进行多次传递

# 如何学习权值?



- **其他想法：**同时扰动所有的权重，并将性能的提高与权重的变化联系起来
- 非常难以实现
- 所以：只扰动激活值（因为他们数量较少），但同样低效

# 反向传播



- **前向传播:** 接受输入 $x$ , 通过中间阶段, 获得输出 $\hat{y}$
- **训练阶段:** 利用 $\hat{y}$ 计算标量损失 $J(\theta)$
- 反向传播允许信息从损失函数反向流动来计算梯度

Figure: G. E. Hinton

# 反向传播

---

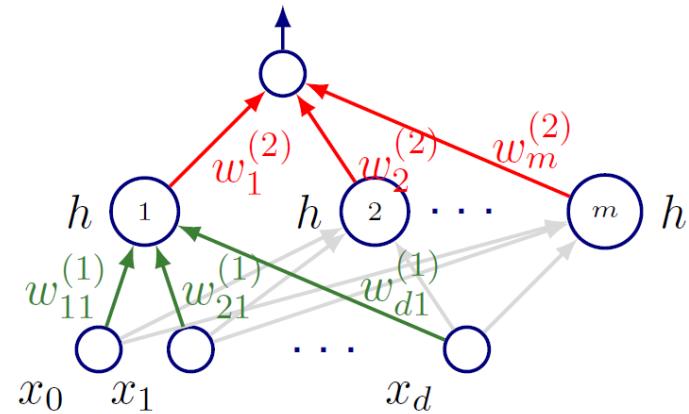
- 从训练数据来看，我们不知道隐藏的单元的效果
- 但是，当我们改变一个隐藏的激活时，我们可以得到误差传播的速度
- 使用误差导数，也称为hidden activities
- 每个隐藏的单元可以影响许多输出单元
- 单独的误差影响->合并这些影响
- 可以有效地计算隐藏单元的误差导数(一旦我们有了隐藏激活的误差导数，就很容易得到权重的误差导数)

# 反向传播：示例

输入到隐藏单元:  $a_j = \sum_{i=0}^d w_{ij}^{(1)} x_i$ ,

隐单元输出:  $z_j = \tanh(a_j)$ ,

网络输出:  $\hat{y} = a = \sum_{j=0}^m w_j^{(2)} z_j$ .



- 首先给出激活函数导数:

$$h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}},$$

$$h'(a) = 1 - h(a)^2.$$

# 反向传播：示例

$$a_j = \sum_{i=0}^d w_{ij}^{(1)} x_i, \quad z_j = \tanh(a_j), \quad \hat{y} = a = \sum_{j=0}^m w_j^{(2)} z_j, \quad h'(a) = 1 - h(a)^2.$$

- 基于输入  $x$  的损失:  $L = \frac{1}{2}(y - \hat{y})^2$ .
- 输出单元:  $\delta = \frac{\partial L}{\partial a} = \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$
- 对  $a_j$  求导:  $\delta_j = \frac{\partial L}{\partial a_j} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_j} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial a_j}$ :

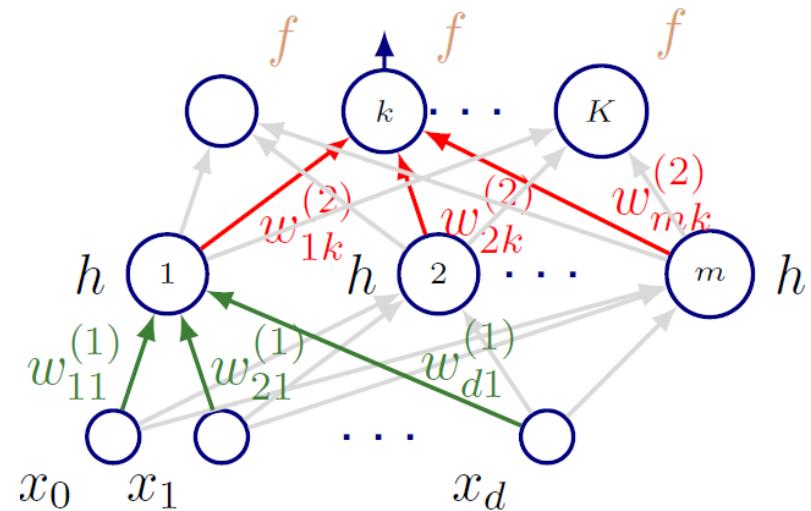
$$\frac{\partial L}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}^{(1)}} = \delta_j \cdot x_i; \quad \frac{\partial L}{\partial w_j^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} = (y - \hat{y}) \cdot z_j = \delta \cdot z_j$$

- 更新权重:  $\omega_j^{(2)} \leftarrow \omega_j^{(2)} - \eta \delta z_j$  以及  $\omega_{ij}^{(1)} \leftarrow \omega_{ij}^{(1)} - \eta \delta_j x_i$ .
- $\eta$  被称为 weight decay

# 多维输出

- 在样本(x, y)上的损失:

$$\frac{1}{2} \sum_{k=1}^K (y_k - \hat{y}_k)^2$$



- 对于每个输出单元  $\delta_k = \hat{y}_k - y_k$ ;
- 对于隐单元  $j$ ,

$$\delta_j = (1 - z_j^2) \sum_{k=1}^K w_{jk}^{(2)} \delta_k$$

# 反向传播：实践

- 由上面的推导过程可以看到，反向传播就是不断的利用求导的链式法则进行展开的过程；
- 这样的过程并不复杂，但是在实际网络规模很大的情况下非常繁琐，需要细心操作；
- 常用的深度学习框架（例如Pytorch, Tensorflow等）中均不需要我们手动编码进行反向传播；
- 只要我们将前向传播的Tensor流动路径定义清楚，框架会自动帮我们计算梯度并反传更新权值；
- 我们只需要关心损失函数的定义，网络框架的搭建等等更加宏观的内容；
- 需要保证在Tensor流图中的每一个Tensor均可以进行反向传播（例如Pytorch中需要关注`requires_grad`是否为True）；

# 随机梯度下降

- 给定n个训练样本，我们的目标函数可以表示为

$$J(\mathbf{w}) = \sum_{p=1}^n J_p(\mathbf{w})$$

- Batch梯度下降

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{p=1}^n \nabla J_p(\mathbf{w})$$

- 在某些情况下，计算梯度的代价可能会很大。
- 随机梯度下降法在每一步都对求和函数的子集进行采样进行梯度计算，这在大规模机器学习问题中非常有效。
- 在随机梯度下降法中， $J(\mathbf{w})$ 的**真实梯度近似为单个样本(或小批量样本)的梯度**：

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J_p(\mathbf{w})$$

# 随机反向传播

## Algorithm 1 (Stochastic backpropagation)

```
1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta, \eta, m \leftarrow 0$ 
2   do  $m \leftarrow m + 1$ 
3      $\mathbf{x}^m \leftarrow$  randomly chosen pattern
4      $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$ 
5   until  $\nabla J(\mathbf{w}) < \theta$ 
6   return  $\mathbf{w}$ 
7 end
```

(Duda et al. Pattern Classification 2000)

- 在随机方式中，权值更新可以减少某单个模式引起的损失，但可能增加整个训练集的损失。

# Mini-batch 随机梯度下降

---

- 把训练集合切分成多个mini-batches.
  - 在每个epoch中，随机排列mini-batches，并按顺序取一个小批来近似梯度
  - 一个epoch对应于训练集中所有模式的单一表达
- 每次迭代估计的梯度更可靠
- 从较小的batch size开始，随着训练的进行逐渐增加

# Batch反向传播

## Algorithm 2 (Batch backpropagation)

```
1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta, \eta, r \leftarrow 0$ 
2   do  $r \leftarrow r + 1$  (increment epoch)
3      $m \leftarrow 0; \Delta w_{ij} \leftarrow 0; \Delta w_{jk} \leftarrow 0$ 
4     do  $m \leftarrow m + 1$ 
5        $\mathbf{x}^m \leftarrow$  select pattern
6        $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i; \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$ 
7     until  $m = n$ 
8      $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}; w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$ 
9   until  $\nabla J(\mathbf{w}) < \theta$ 
10 return  $\mathbf{w}$ 
11 end
```

# 总结

---

- 随机学习
  - 梯度的估计是有噪声的，在每次迭代中权值可能不会沿着梯度精确地向下移动
  - 比批量学习更快，特别是当训练数据有冗余的时候
  - 噪声通常会产生更好的结果
  - 权值是波动的，它可能不会最终收敛到局部极小值
- 批学习
  - 收敛条件很好理解
  - 一些加速技术只适用于批量学习
  - 权值变化规律和收敛速度的理论分析相对简单

# 下节内容

---

- 卷积神经网络

# Thank You !

