

# 人工智能之深度学习

## 深度学习概述

主讲人：刘老师(GerryLiu)

# 课程要求

- 课上课下 “九字” 真言
  - 认真听, **善摘录, 勤思考**
  - **多温故, 乐实践**, 再发散
- 四不原则
  - **不懒散惰性, 不迟到早退**
  - **不请假旷课, 不拖延作业**
- 一点注意事项
  - 违反 “四不原则” , 不推荐就业

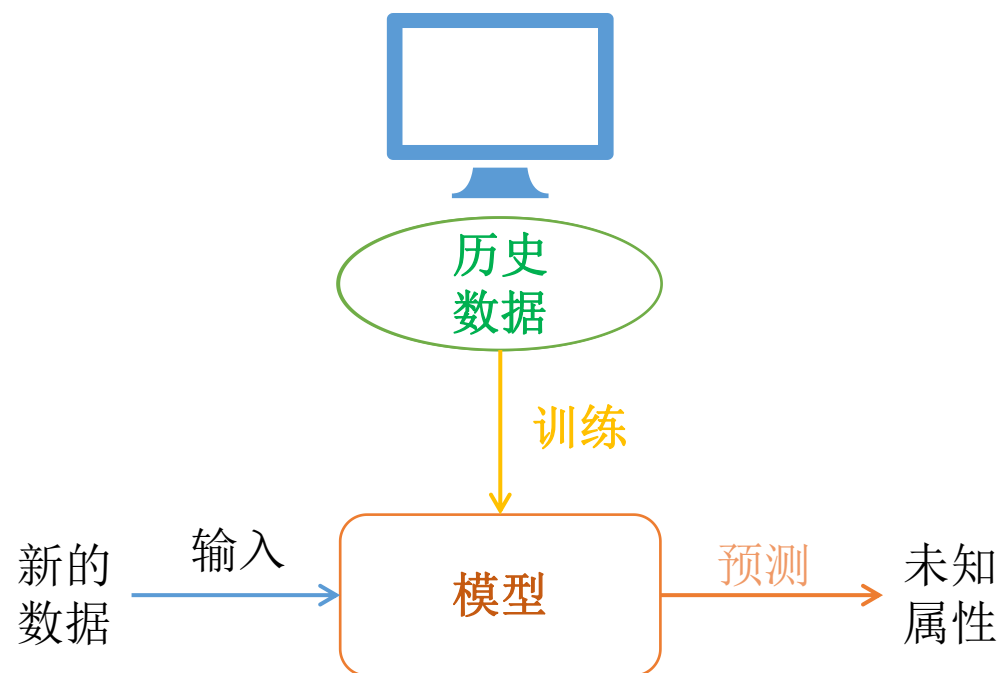
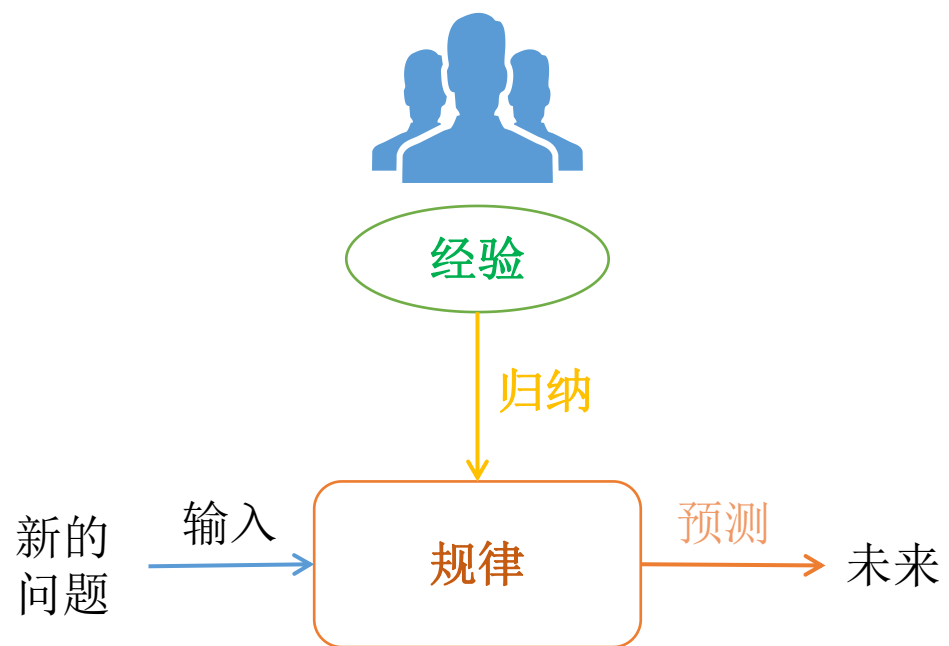
## 课程内容

- 深度学习的应用场景
- 神经网络的起源
- 神经网络的基本结构
- BP神经网络
- RBF神经网络

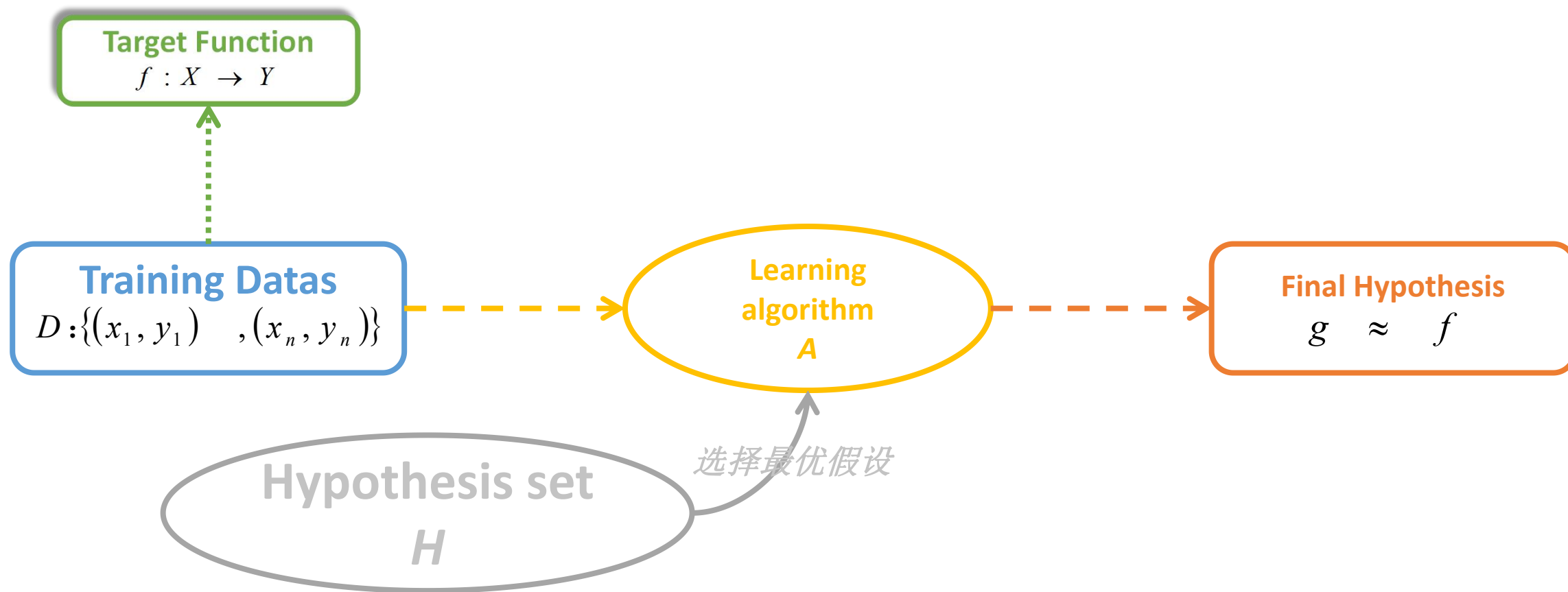
# 深度学习的应用

- 图像应用
  - 大规模(大数据量)图片识别(聚类/分类), 如人脸识别, 车牌识别, OCR等
  - 以图搜图, 图像分割
  - 目标检测, 如自动驾驶的行人检测, 安防系统的异常人群检测
- 自然语言处理
  - 语音识别, 语音合成自动分词, 句法分析, 语法纠错, 关键词提取, **文本分类/聚类**, 文本自动摘要, **信息检索 (ES,Solr)**
  - **知识图谱**, 机器翻译, **人机对话**, **机器写作**
  - 推荐系统, 高考机器人
  - 信息抽取, 网络爬虫, **情感分析**, 问答系统
- 数据挖掘, 风控系统, 推荐系统, 广告系统等

# 机器学习回顾



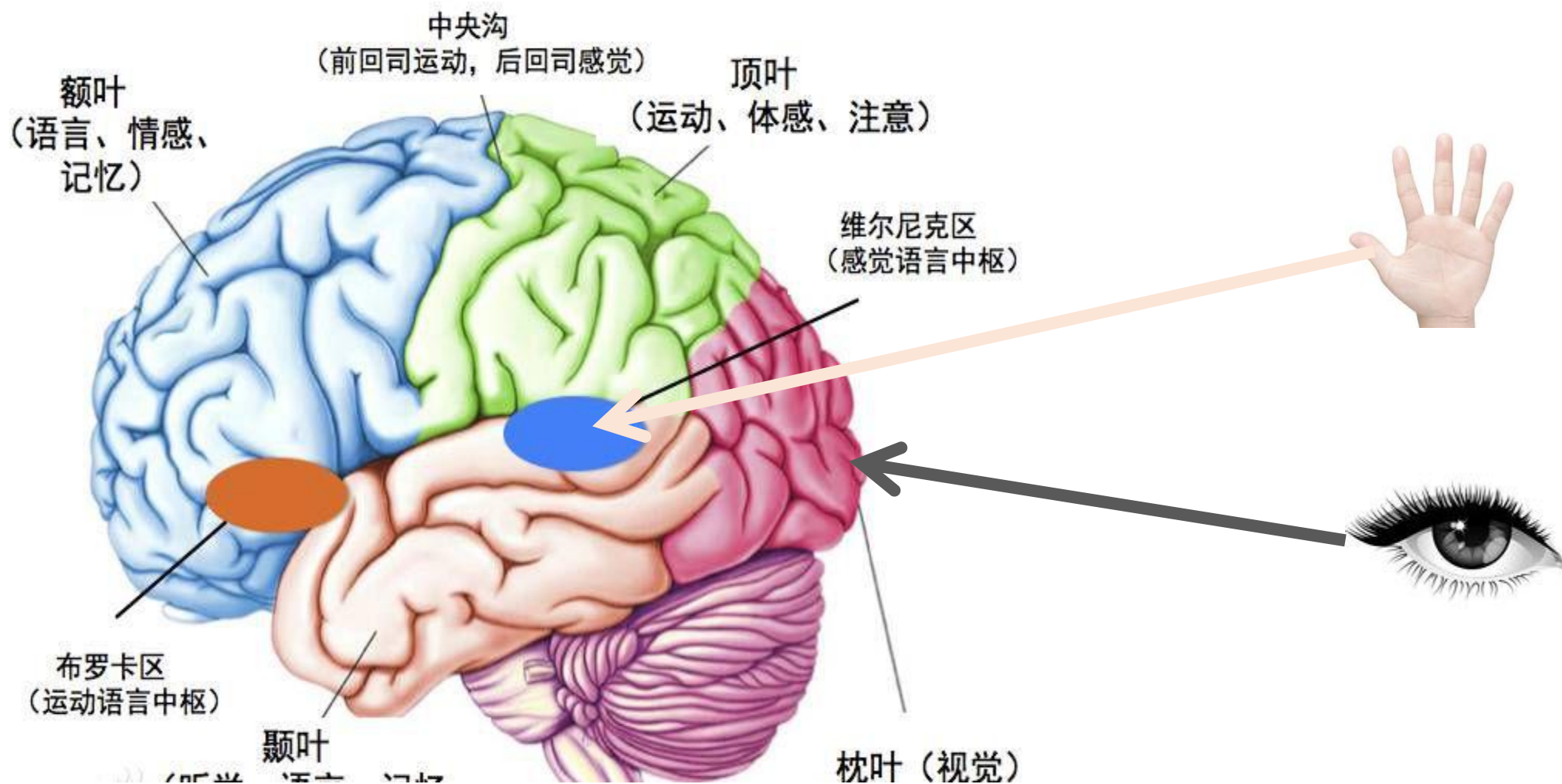
# 机器学习回顾



- 机器学习

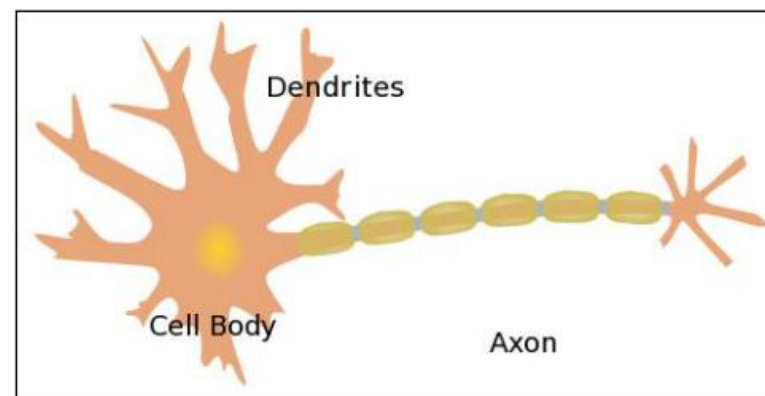
从数据中获得一个假设的函数 $g$ ，使其非常接近目标函数 $f$ 的效果。

# 神经网络来源之人的思考



## 神经网络来源之人的思考

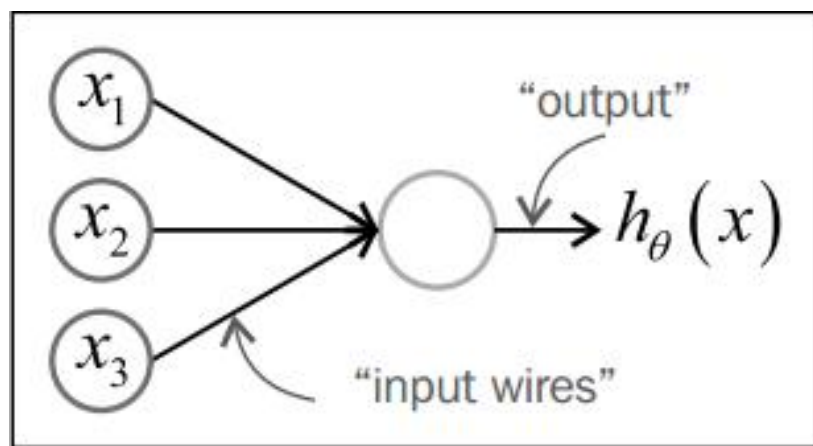
- 大脑是由处理信息的**神经元细胞**和连接神经元的细胞进行信息传递的**突触**构成的。**树突**(Dendrites)从一个神经元接受电信号，信号在**细胞核**(Cell Body)处理后，然后通过**轴突**(Axon)将处理的信号传递给下一个神经元。
- 一个神经元可以看作是将一个或多个输入处理成一个输出的计算单元。
- 通过多个神经元的传递，最终大脑会得到这个信息，并可以对这个信息给出一个合适的反馈。





## 感知器模型

- 感知器是一种模拟人的神经元的一种算法模型，是一种研究单个训练样本的二元分类器，是SVM和人工神经网络(ANN, Artificial Neural Networks)的基础。
- 一个感知器接受几个二进制的输入，并产生一个二进制的输出，通常的表达方式如下：



$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq threshold \\ 1, & \text{if } \sum_j w_j x_j > threshold \end{cases}$$

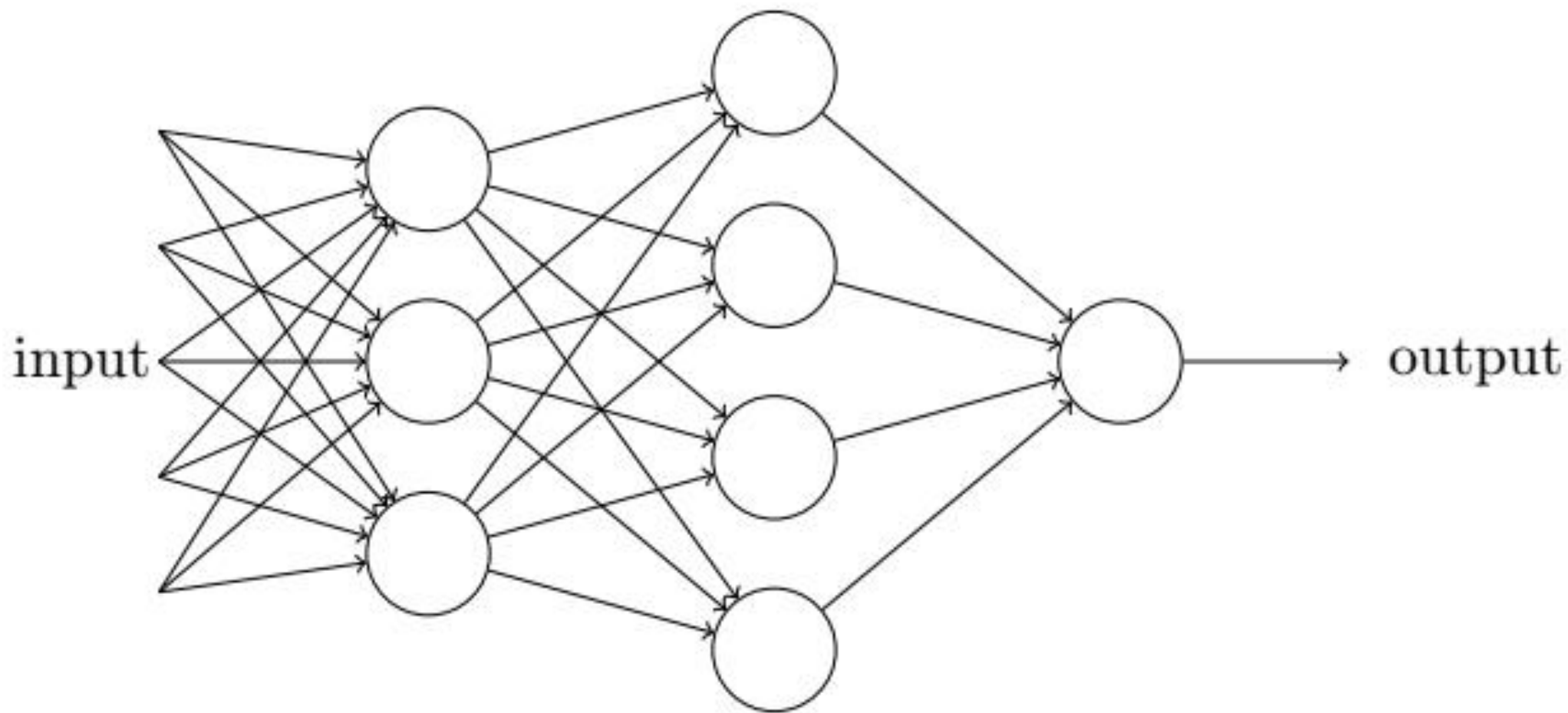
$$output = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

## 感知器模型案例直观理解

- 感知器可以看作是根据权重来做出决定的一个设备/单元，只要我们可以给定一个比较适合权重以及阈值，那么感知器应该是能够对数据进行判断的/分类预测的。假定你现在在考虑是否换工作，也许你会考虑一下三个方面的因素：
  - 新工作的待遇会提高吗？
  - 你家庭经济压力大吗？
  - 新工作稳定吗？

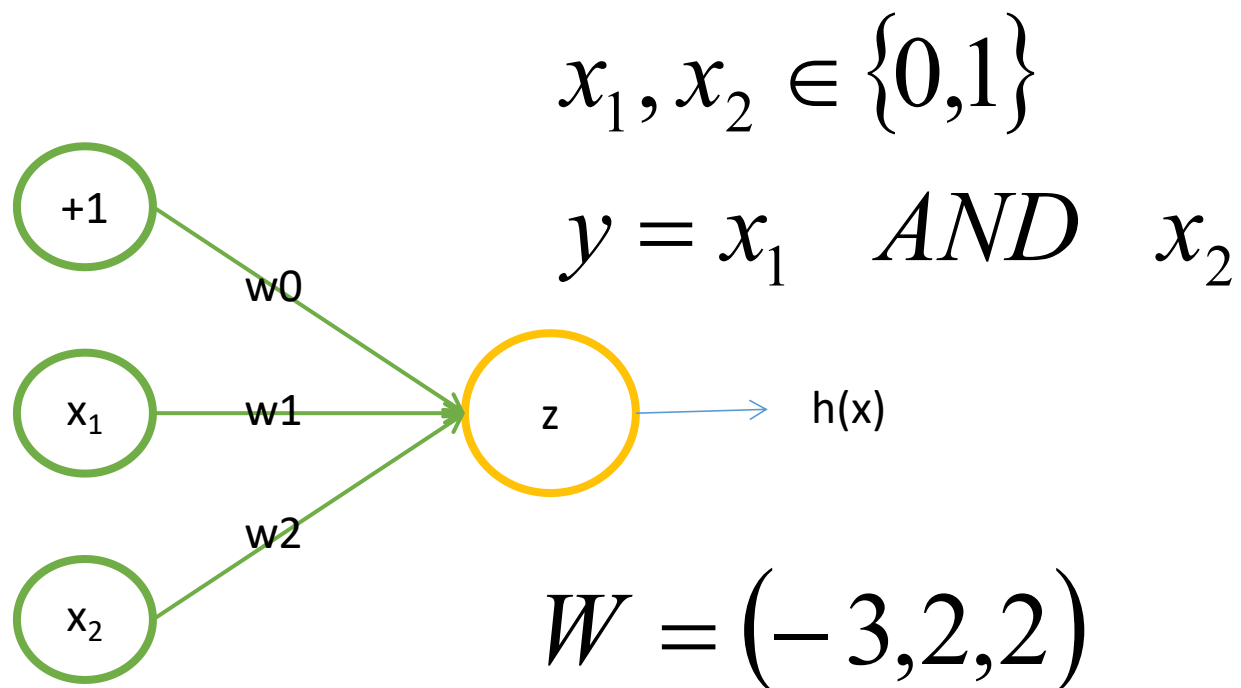
## 多层感知器(人工神经网络)

- 将多个感知器进行组合，我们就可以得到一个多层感知器的网络结构，网络中的每一个节点我们叫做神经元。

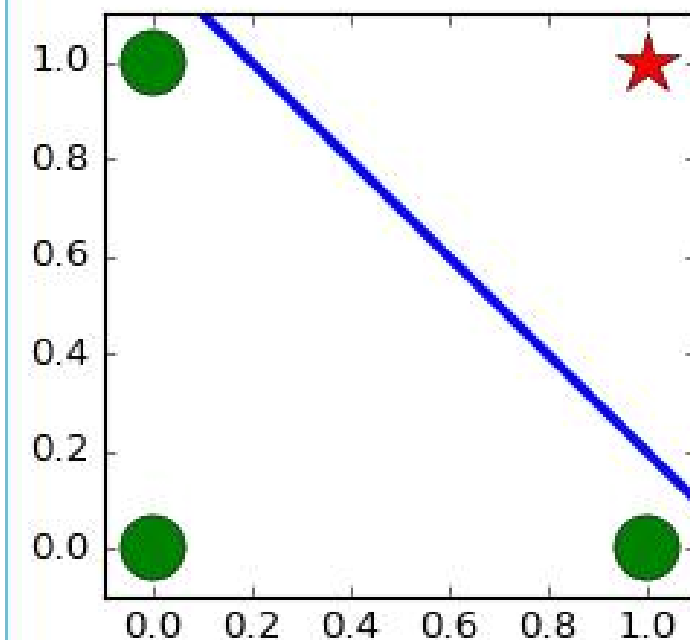


## 感知器神经元直观理解之逻辑与

- 单个神经元完成**逻辑与**功能



$$h_w(z) = h(-3 + 2 \cdot x_1 + 2 \cdot x_2) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$



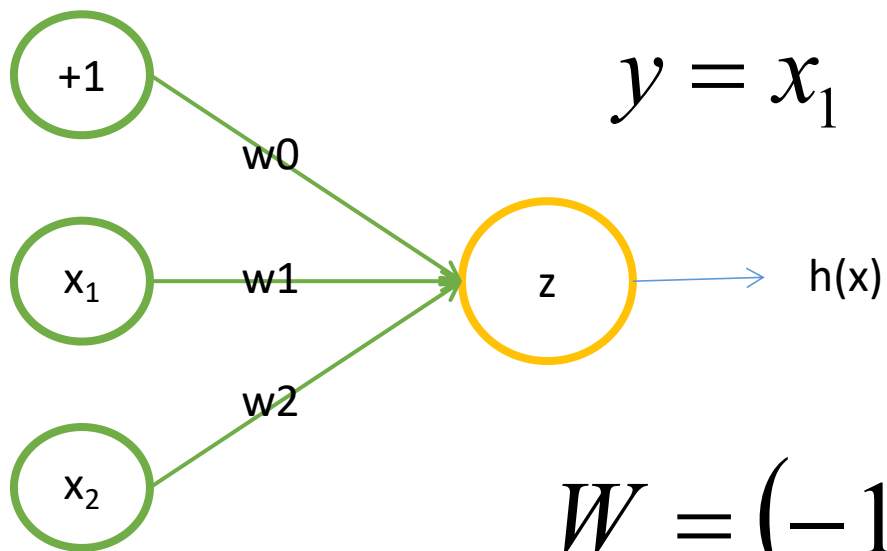
$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	0
0	1	0
1	0	0
1	1	1

## 感知器神经元直观理解之逻辑或

- 单个神经元完成**逻辑或**功能

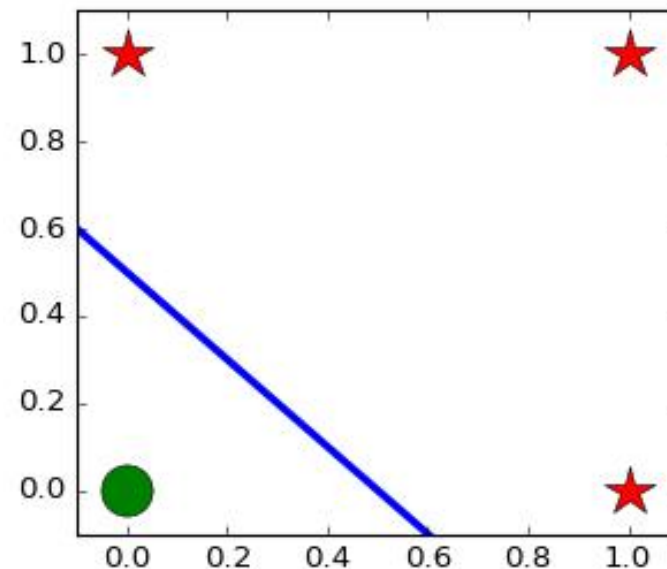
$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ OR } x_2$$



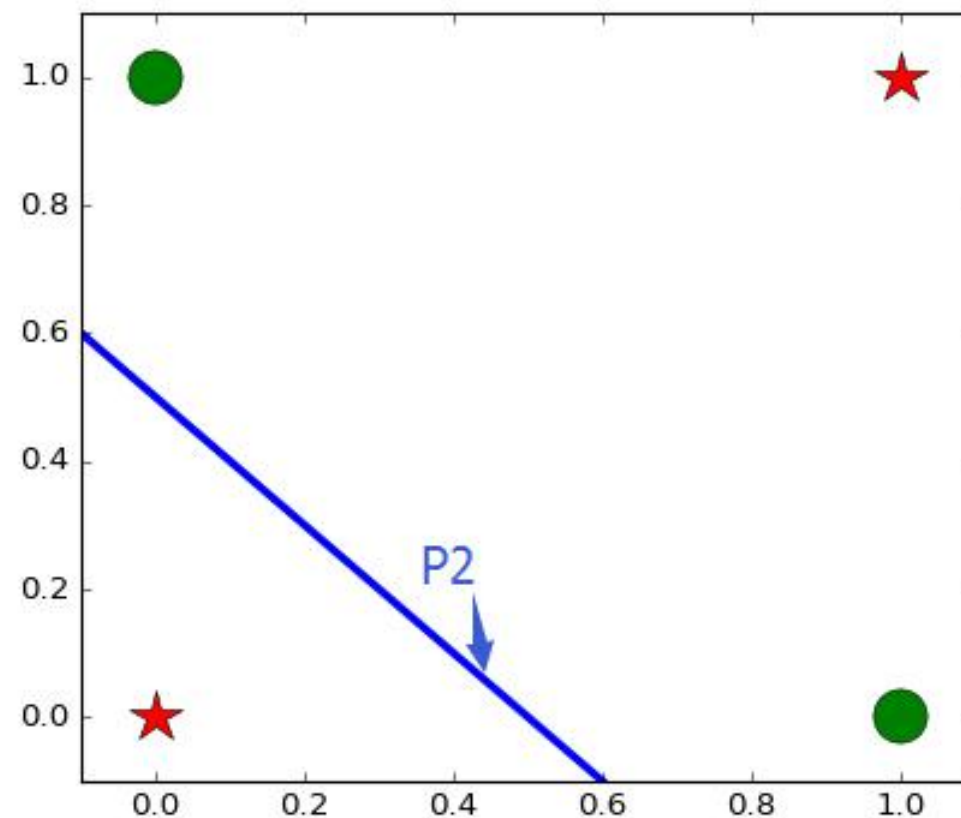
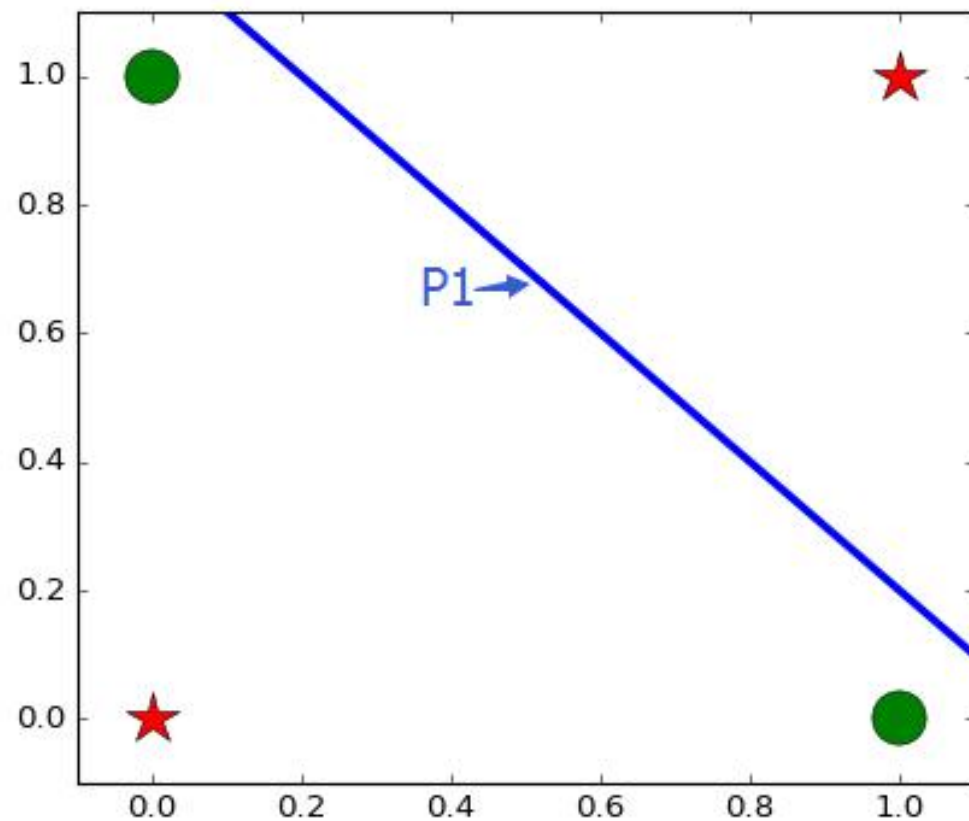
$$W = (-1, 2, 2)$$

$$h_w(z) = h(-1 + 2 \cdot x_1 + 2 \cdot x_2) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

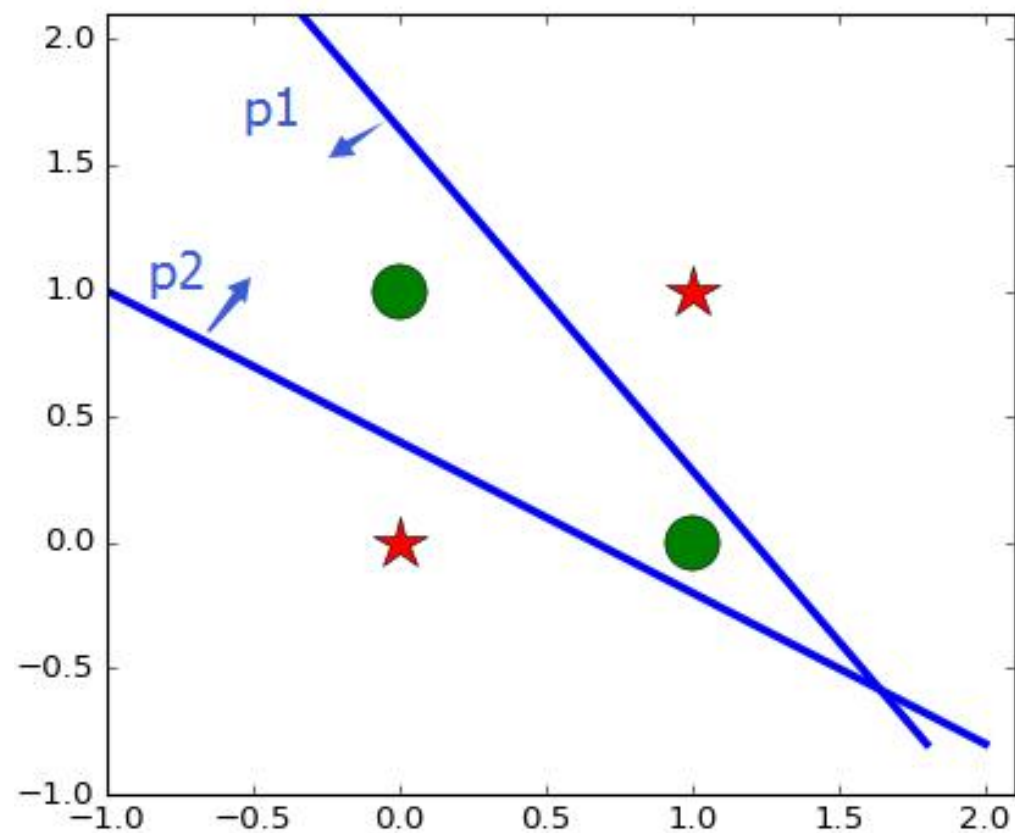
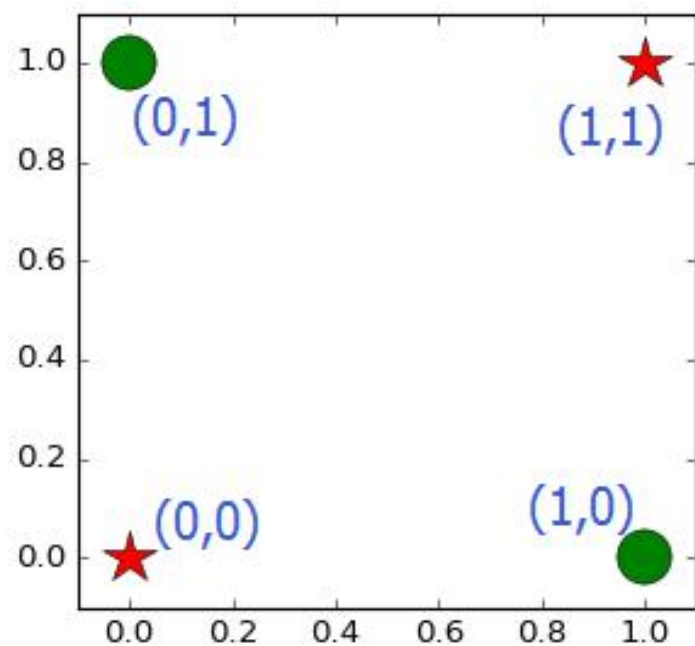


$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	0
0	1	1
1	0	1
1	1	1

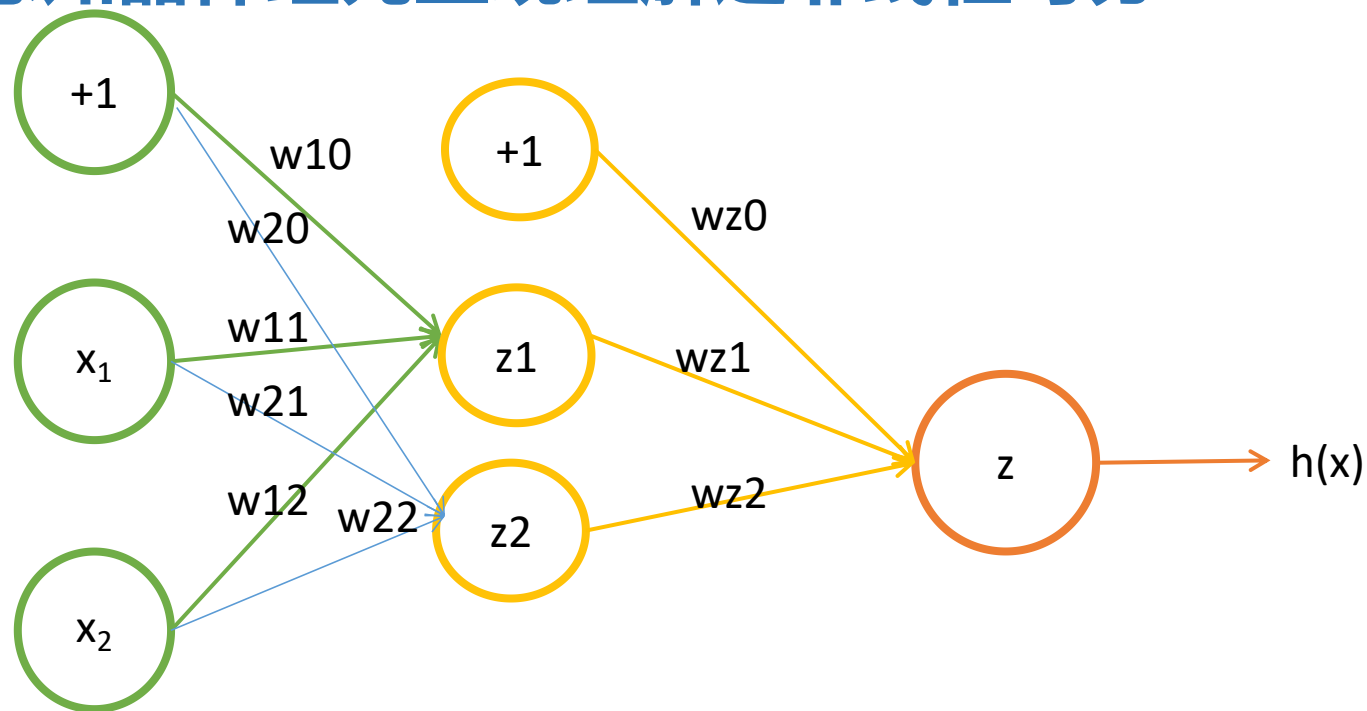
## 感知器神经元直观理解之非线性可分



## 感知器神经元直观理解之非线性可分



## 感知器神经元直观理解之非线性可分



$x_1$	$x_2$	$z_1$	$z_2$	$h(x)$
0	0	0	0	1
0	1	0	1	0
1	0	0	1	0
1	1	1	1	1

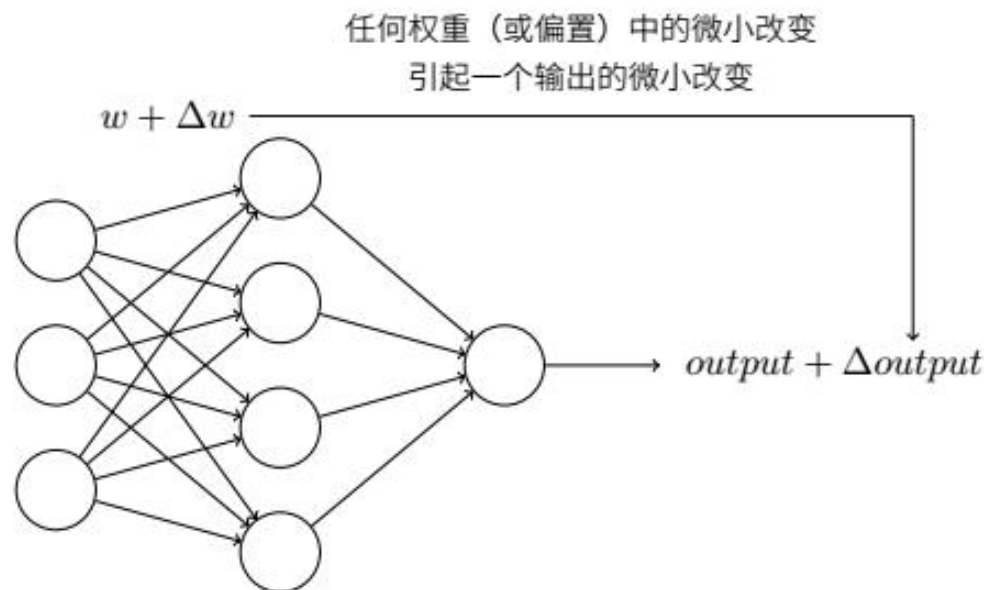
$$W_1 = (-3, 2, 2) \quad W_2 = (-1, 2, 2) \quad W_z = (1, 3, -3)$$

$$h_w(z) = h(Wx) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$



## 感知器网络理解以及S型神经元

- 其实只要将网络中的权重或者偏置项稍微的做一点小的改动，都会导致最终的输出发生一定的变化。但是在感知器神经网络中，单个感知器上的权重或者偏置项发现一点小的变化，最终的输出要不不变，要不完全翻转(因为只有两种取值0和1)，这种翻转会导致接下来的感知器可能发生复杂的完全没法控制的变化，这样会导致我们的网络很难得到最终的逼近结果。



## 感知器网络理解以及S型神经元

- 针对感知器网络的这种很难学习的问题，引入S型神经元来代替感知器，从而解决这个问题。
- 从感知器模型中，我们可以将单个神经元的计算过程看成下列两个步骤：
  - 先计算权重 $w$ 和输入值 $x$ 以及偏置项 $b$ 之间的线性结果值 $z$ ： $z = xw + b$
  - 然后对结果值 $z$ 进行一个数据的sign函数(变种)转换，得到一个离散的0/1值： $y = \text{int}((\text{sign}(z) + 1)/2)$
- 在S型神经元中，和感知器神经元的区别在于：
  - 对于结果值 $z$ 的转换，采用的不是sign函数进行转换，是采用平滑类型的函数进行转换，让输出的结果值 $y$ 最终是一个连续的，S型神经元转指使用的是sigmoid函数。

## 神经网络来源之“神经元”

- 输入：特征属性 $x_1$ 、 $x_2$ 、 $x_3$ 和截距+1

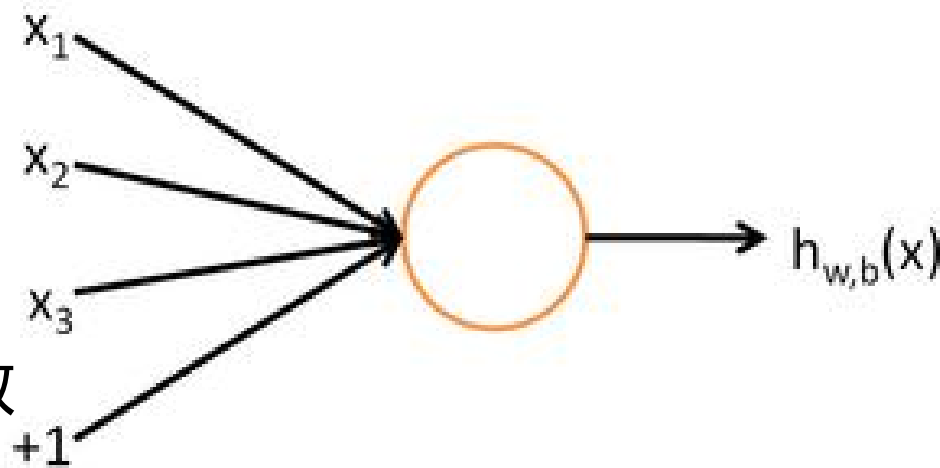
- 输出：函数 $h_{w,b}(x)$ ，其中 $w$ 权重和 $b$ 偏置项是参数

$$h_{W,b}(x) = f\left(W^T x, b\right) = f\left(\sum_{i=1}^3 W_i x_i + b\right)$$

- 注意：函数 $f$ 被称为“**激活函数**”；常用/最好出现激活函数有sigmoid(逻辑回归函数)和tanh(双曲正切函数)

$$\tanh(z) = f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}; \quad f'(z) = 1 - (f(z))^2$$

$$\text{sigmoid}(z) = f(z) = \frac{1}{1 + e^{-z}}; \quad f'(z) = f(z)(1 - f(z))$$



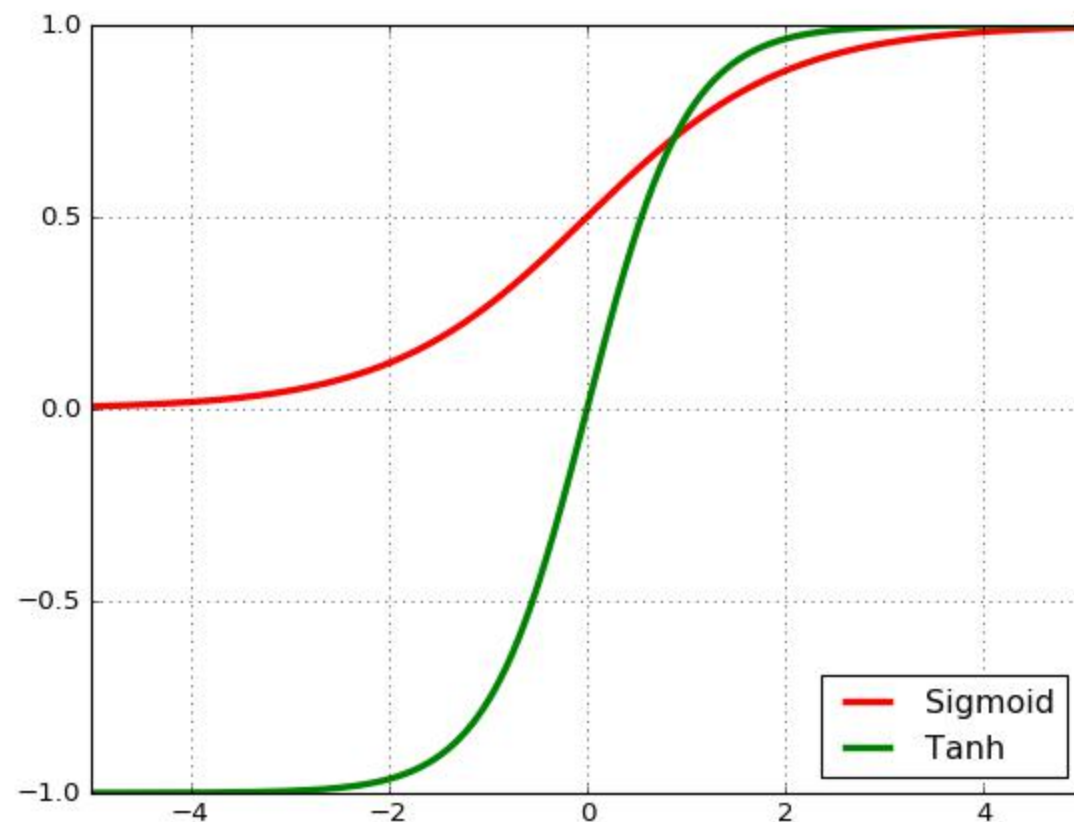
## 激活函数

- 激活函数的主要作用是**提供网络的非线性建模能力**。如果没有激活函数，那么该网络仅能够表达线性映射，此时即便有再多的隐藏层，其整个网络跟单层神经网络也是等价的。因此也可以认为，只有加入了激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。激活函数的主要特性是：**可微性、单调性、输出值的范围**；
- 常见的激活函数：Sign函数、Sigmoid函数、Tanh函数、ReLU函数、P-ReLU函数、Leaky-ReLU函数、ELU函数、Maxout函数等

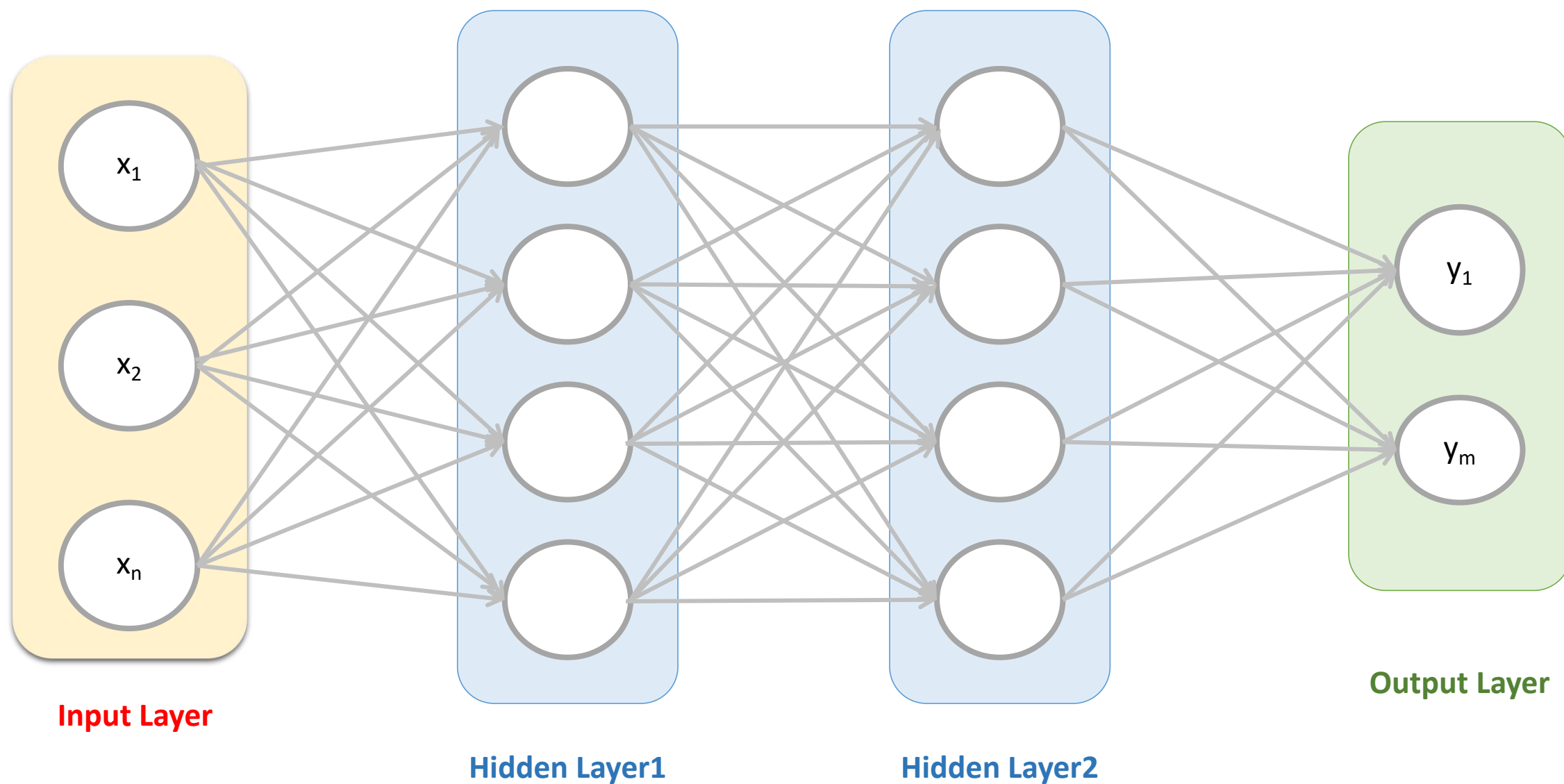
## 激活函数

$$\tanh(z) = f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\text{sigmoid}(z) = f(z) = \frac{1}{1 + e^{-z}}$$



# 神经网络之结构



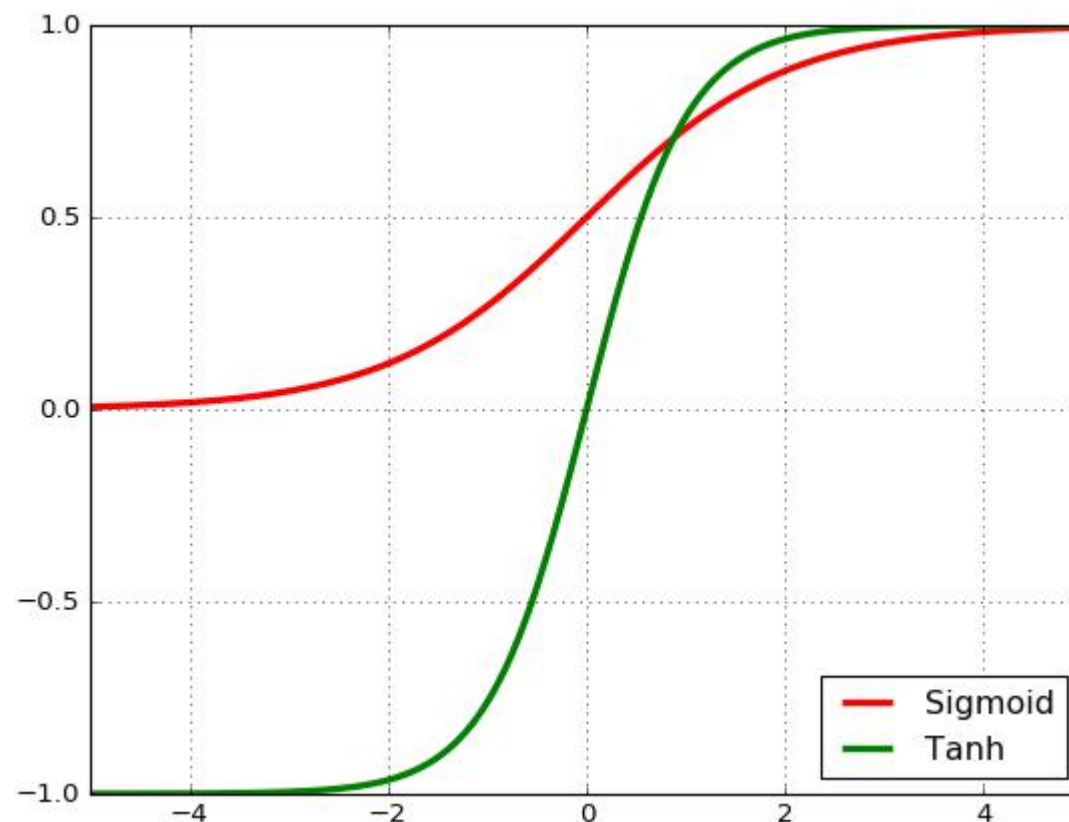
# 神经网络

- 神经网络主要由三个组成部分，第一个是**架构**（architecture）或称为拓扑结构（topology），描述神经元的层次与连接神经元的结构。第二个组成部分是神经网络使用的**激励/激活函数**。第三个组成部分是找出最优权重值的学习算法。
- 神经网络主要分为两种类型，前馈神经网络(Feedforward Neural Networks)是最常用的神经网络类型，一般定义为**有向无环图**，信号只能沿着最终输出的那个方向传播。另外一个**是**反馈神经网络(Feedback Neural Networks)，也称为递归神经网络(Recurrent Neural Networks)，也就是**网络中环**。

## 神经网络之传递函数(激活函数)

S函数:  $\text{sigmoid}(z) = f(z) = \frac{1}{1 + e^{-z}}$

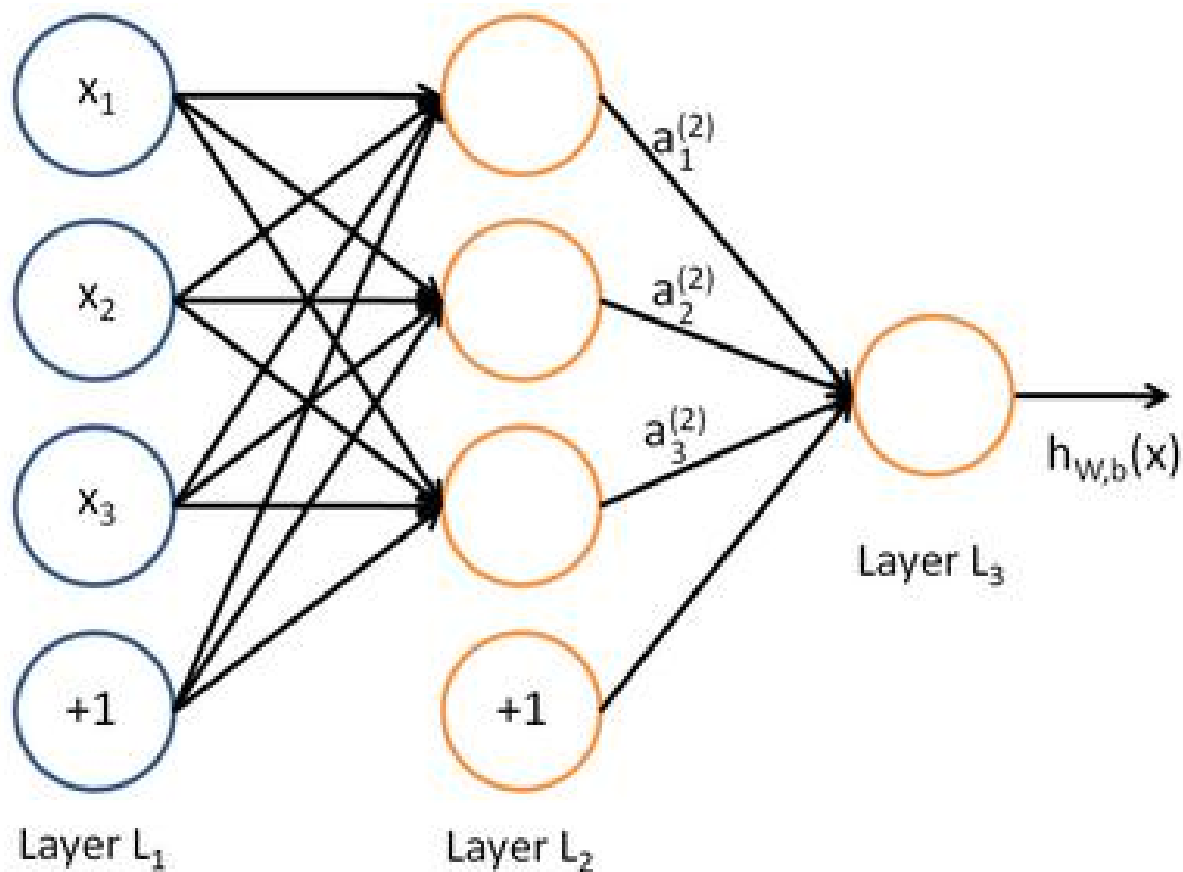
双S函数:  $\tanh(z) = f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$





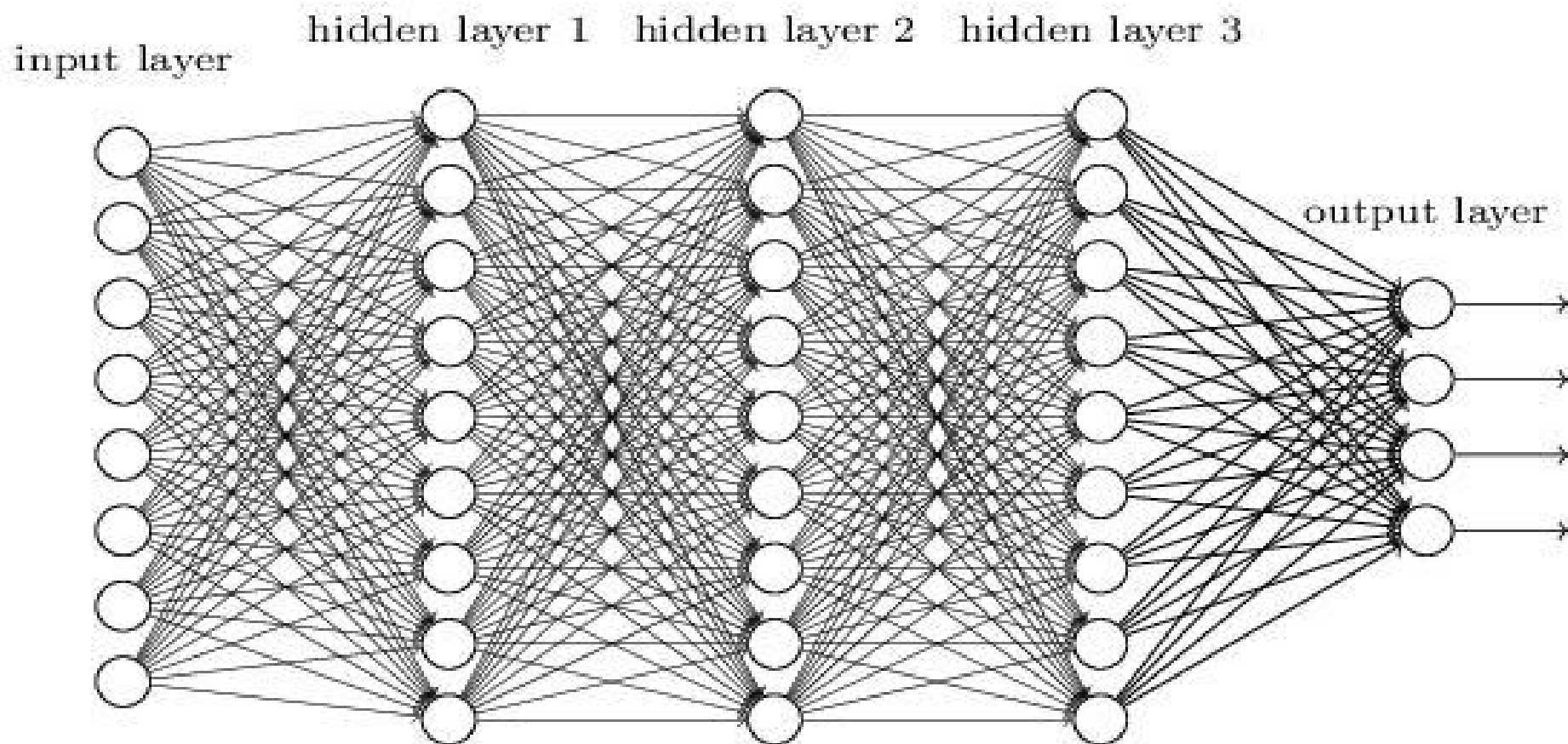
## 神经网络之浅层神经网络

- 添加少量隐层的神经网络就叫做浅层神经网络；也叫作传统神经网络，一般为2隐层的神经网络



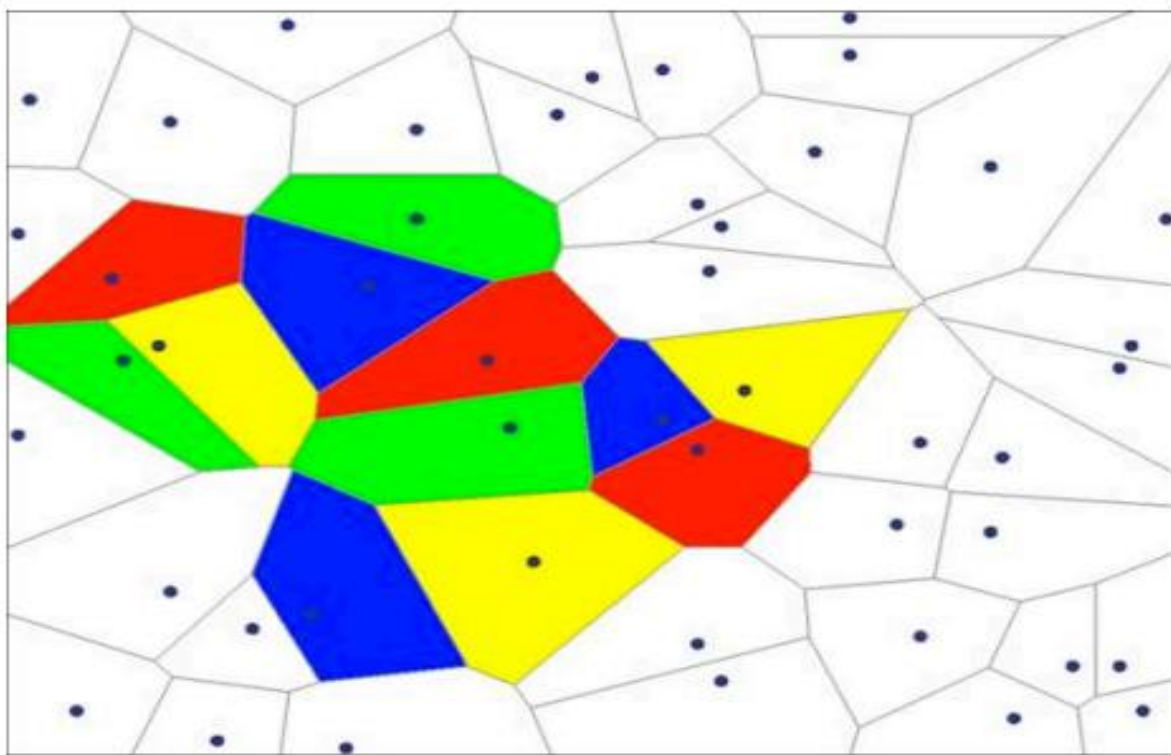
## 神经网络之深度神经网络

- 增多中间层(隐层)的神经网络就叫做深度神经网络(DNN)<全连接神经网络>； 可以认为深度学习是神经网络的一个发展

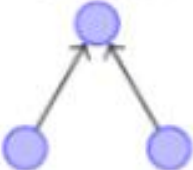

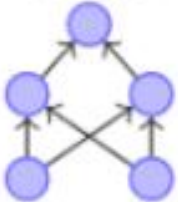




## 神经网络之非线性可分

- 对线性分类器的**与**和**或**的组合可以完成非线性可分的问题；即通过多层的神
- 经网络中加入激活函数的方式可以解决非线性可分的问题。



# 神经网络

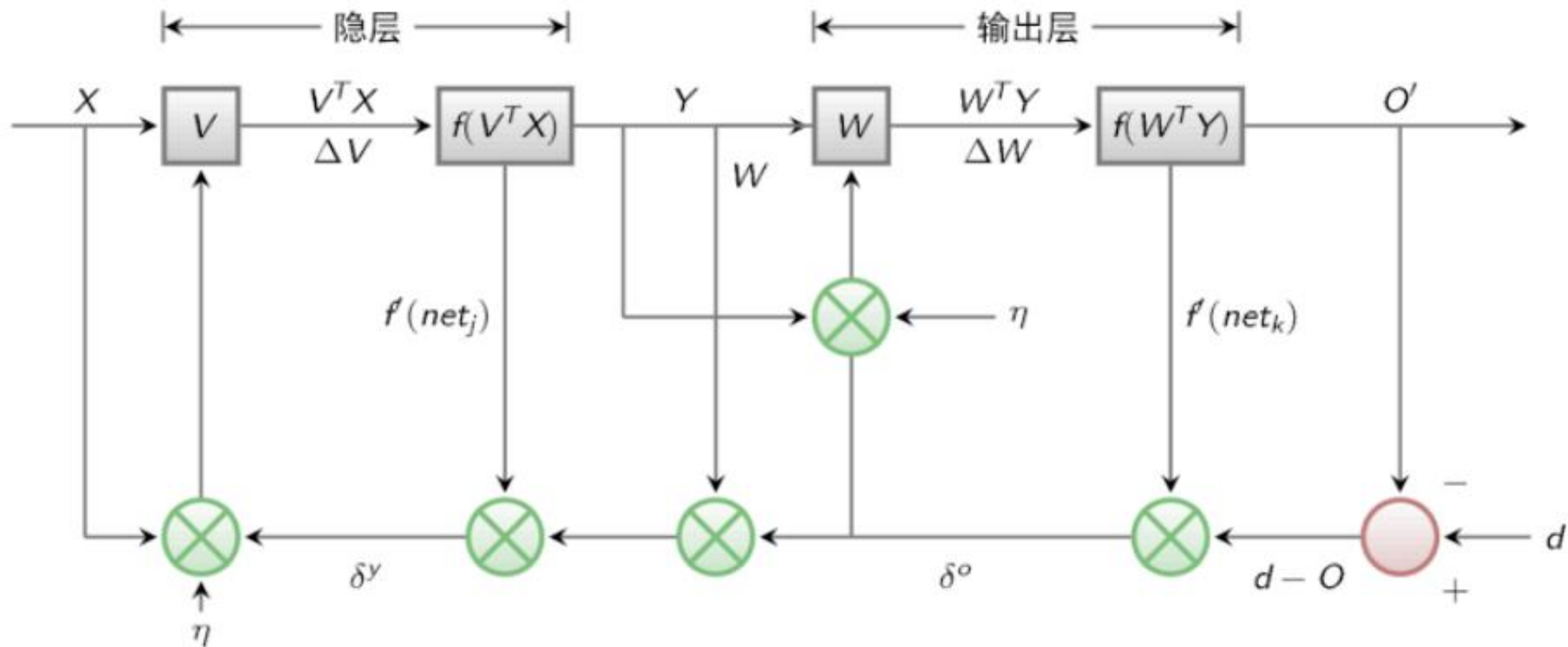
结构	决策区域类型	区域形状	异或问题
无隐层 	由一超平面分成两个		
单隐层 	开凸区域或闭凸区域		
双隐层 	任意形状（其复杂度由单元数目确定）		

## 神经网络之过拟合

- 理论上讲，单隐层的神经网络可以逼近任何连续函数（只要隐层的神经元个数足够的多<一个神经元将数据集分为两类>）
- 虽然从数学表达上来讲，效果一样，但是在网络工程效果中，多隐层的神经网络效果要比单隐层的神经网络效果好
- 对于一些分类的问题来讲，三层的神经网络效果优于两层的神经网络，但是如果把层次不断增加(4,5,6,7....)，对于最终的效果不会产生太大的变化
- 提升**隐层层数**或者**神经元个数**，神经网络的“容量”会变大，那么空间表达能力会变强（模型的预测能力、模型的学习能力），从而有可能导致**过拟合**的问题
- 对于视频/图片识别等问题，传统的神经网络(全连接神经网络)不太适合

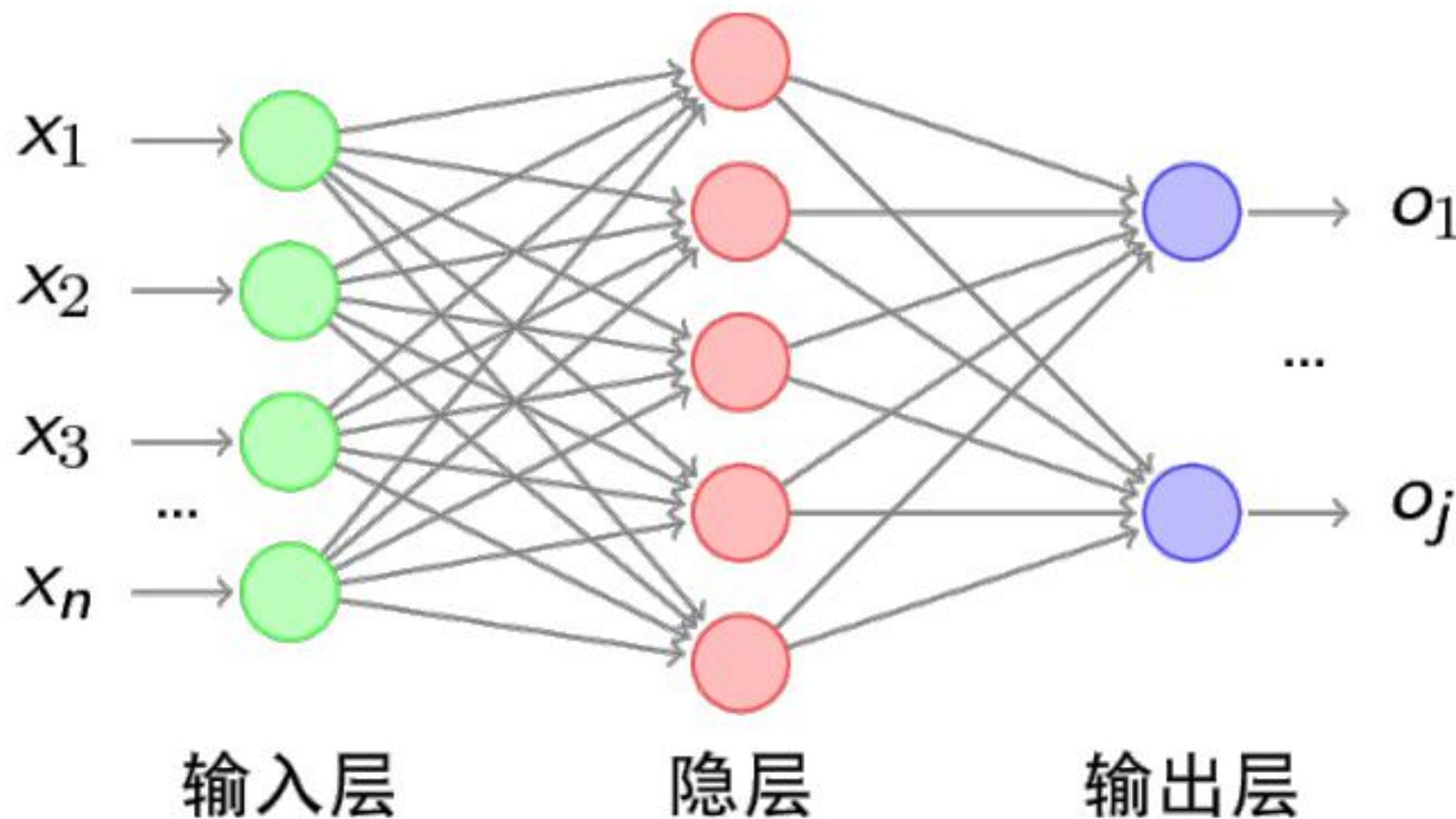
## 神经网络之BP算法

- 神经网络的一种求解 $W$ 的算法，分为信号“正向传播(FP)”求损失，“反向传播(BP)”回传误差；根据误差值修改每层的权重，继续迭代



## 神经网络之BP算法

- BP算法也叫做 $\delta$ 算法
- 以三层的S型神经元为例（假定现在隐层和输出层均存在相同类型的激活函数）



## 神经网络之BP算法

- 输出层误差

$$E = \frac{1}{2} (d - O)^2 = \frac{1}{2} \sum_{k=1} (d_k - O_k)^2$$

- 隐层的误差

$$E = \frac{1}{2} \sum_{k=1} (d_k - f(net_k))^2 = \frac{1}{2} \sum_{k=1} \left( d_k - f \left( \sum_{j=1}^m w_{jk} y_j \right) \right)^2$$

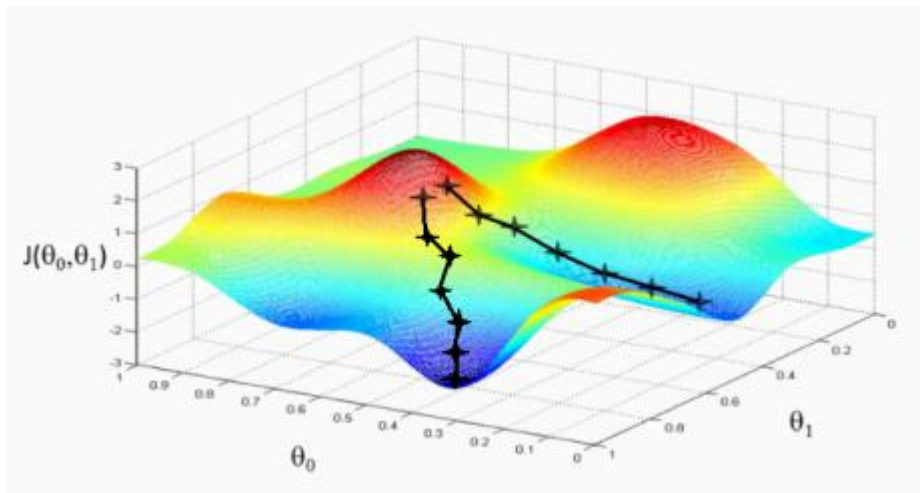
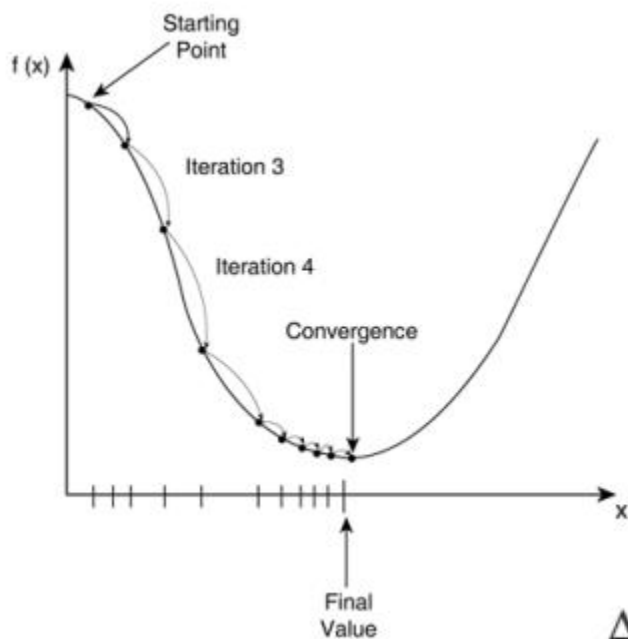
- 输入层误差

$$E = \frac{1}{2} \sum_{k=1} \left( d_k - f \left[ \sum_{j=0}^m w_{jk} f(net_j) \right] \right)^2 = \frac{1}{2} \sum_{k=1} \left( d_k - f \left[ \sum_{j=0}^m w_{jk} f \left( \sum_{i=1}^n v_{ij} x_i \right) \right] \right)^2$$



# 神经网络之SGD

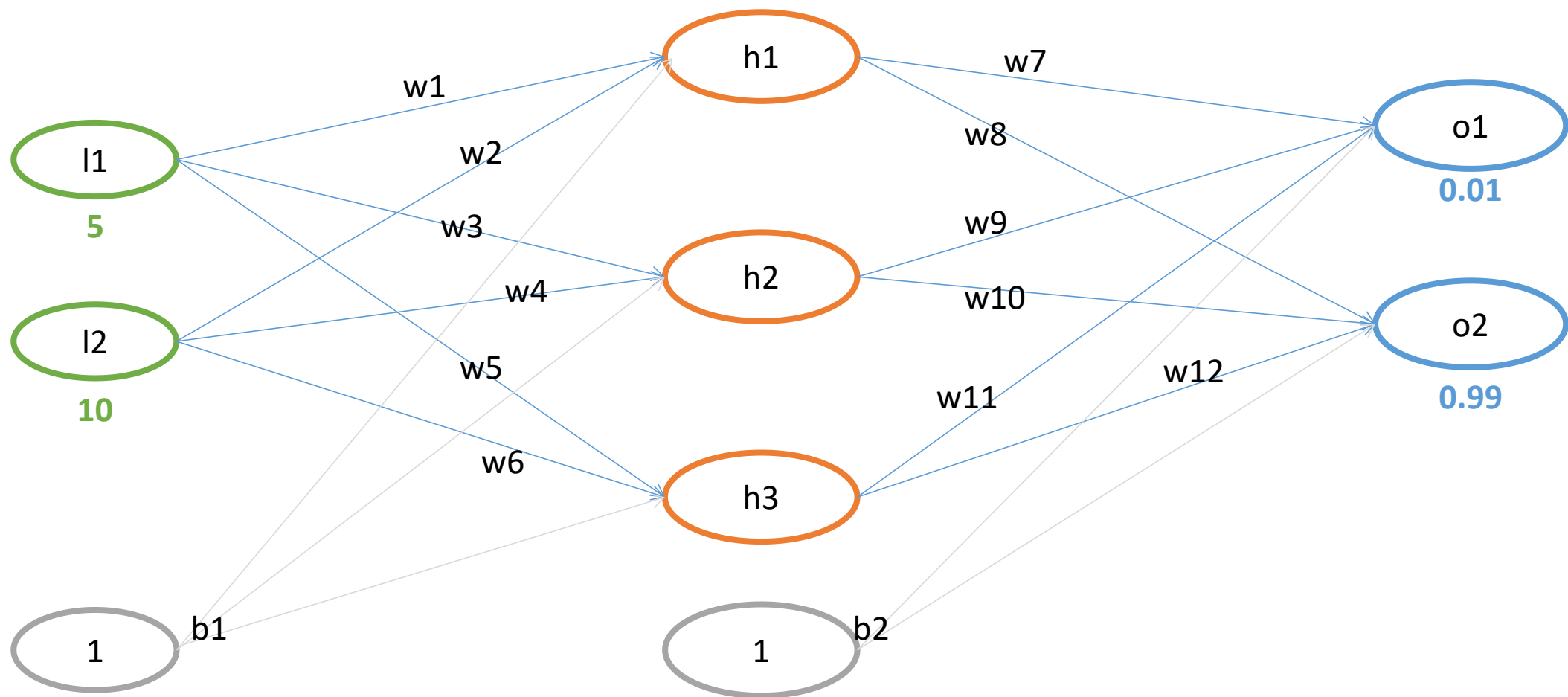
- 误差E有了，那么为了使误差越来越小，可以采用随机梯度下降的方式进行 $\omega$ 和 $v$ 的求解，即求得 $\omega$ 和 $v$ 使得误差E最小



$$\Delta \omega_{jk} = -\eta \frac{\partial E}{\partial \omega_{jk}} \quad j = 0, 1, 2, \dots, m; \quad \kappa = 1, 2, \dots, \ell$$

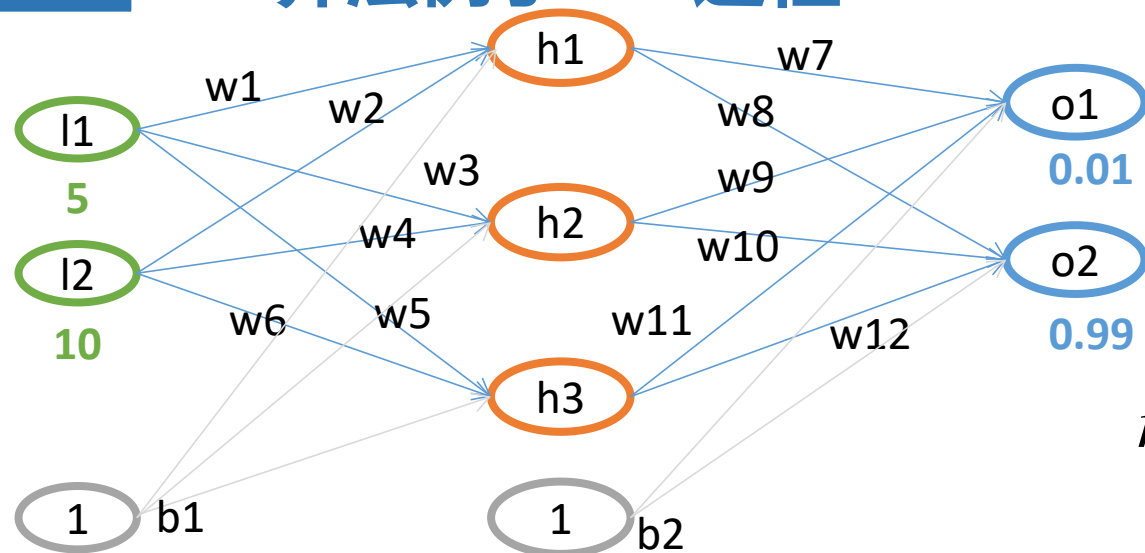
$$\Delta v_{ij} = -\eta \frac{\partial E}{\partial v_{ij}} \quad i = 0, 1, 2, \dots, n; \quad j = 1, 2, \dots, m$$

## BP算法例子



$$w = (0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65)$$
$$b = (0.35, 0.65)$$

## BP算法例子-FP过程



$$b = (0.35, 0.65)$$

$$w = \begin{pmatrix} 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, \\ 0.4, 0.45, 0.5, 0.55, 0.6, 0.65 \end{pmatrix}$$

$$net_{h1} = w_1 * l_1 + w_2 * l_2 + b_1 * 1$$

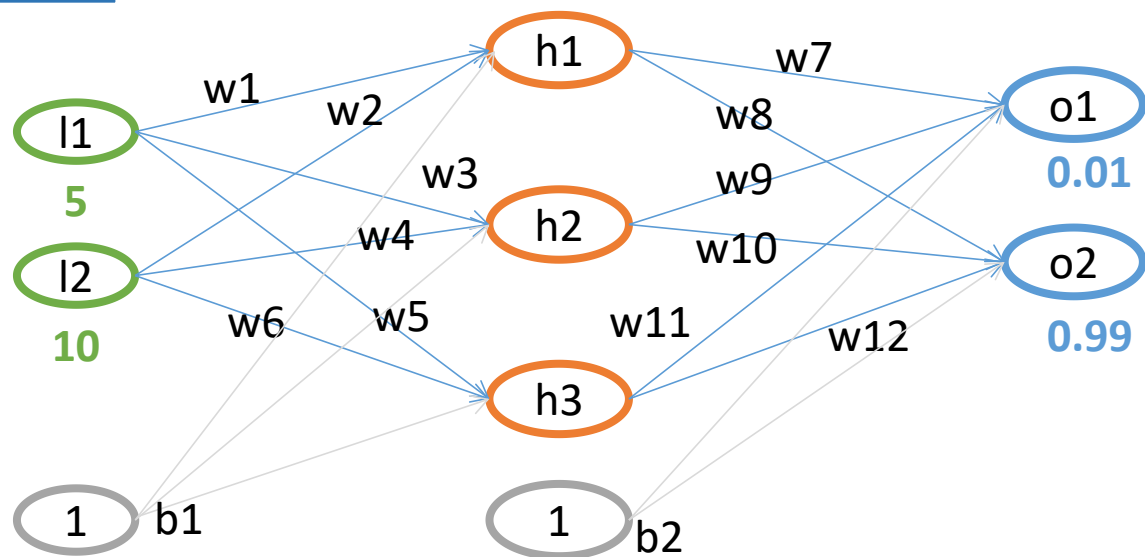
$$net_{h1} = 0.1 * 5 + 0.15 * 10 + 0.35 * 1 = 2.35$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-2.35}} = 0.912934$$

$$out_{h2} = 0.979164$$

$$out_{h3} = 0.995275$$

## BP算法例子-FP过程



$$net_{o1} = w_7 * out_{h1} + w_9 * out_{h2} + w_{11} * out_{h3} + b_2 * 1$$

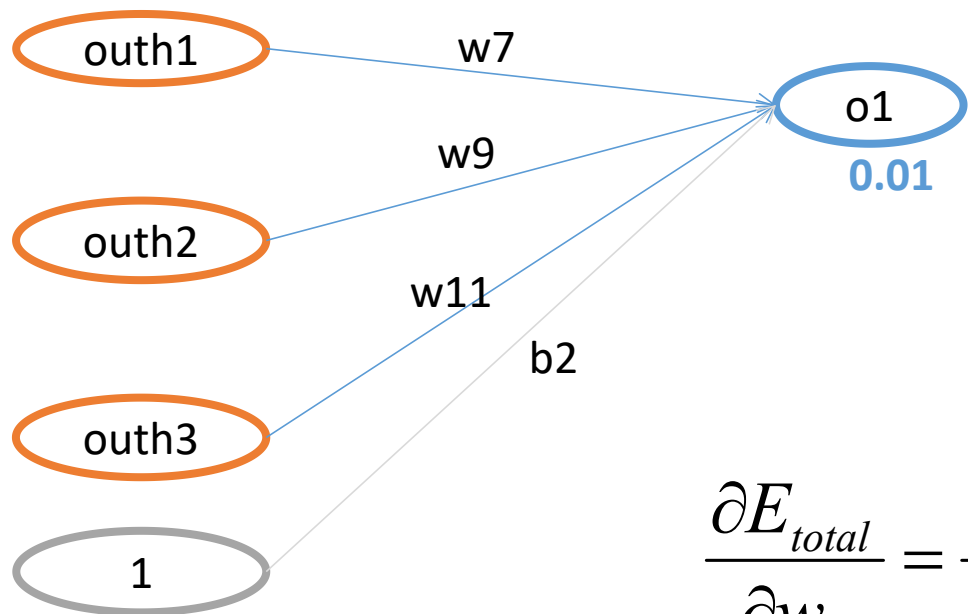
$$net_{o1} = 0.4 * 0.912934 + 0.5 * 0.979164 + 0.6 * 0.995275 = 2.1019206$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-2.1019206}} = 0.891090$$

$$out_{o2} = 0.904330$$

$$E_{total} = E_{o1} + E_{o2} = \frac{1}{2} (0.01 - 0.891090)^2 + \frac{1}{2} (0.99 - 0.904330)^2 = 0.391829$$

## BP算法例子-BP过程(W7)



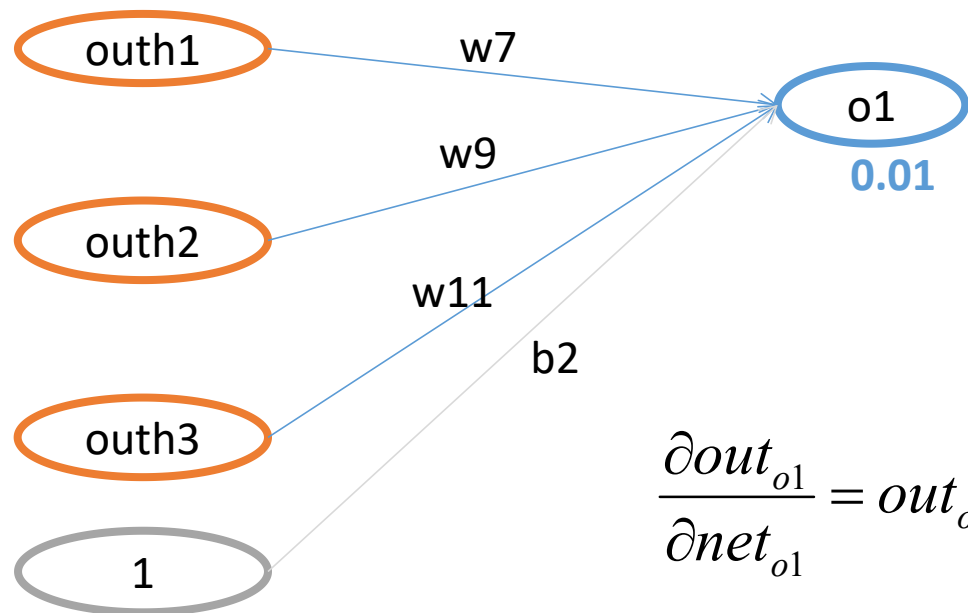
$$E_{o1} = \frac{1}{2} (\text{target}_{o1} - \text{out}_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial \text{out}_{o1}} * \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} * \frac{\partial \text{net}_{o1}}{\partial w_7}$$

$$\frac{\partial E_{total}}{\partial \text{out}_{o1}} = 2 * \frac{1}{2} (\text{target}_{o1} - \text{out}_{o1})^{2-1} * -1 + 0 = -(0.01 - 0.891090) = 0.88109$$

## BP算法例子-BP过程(W7)



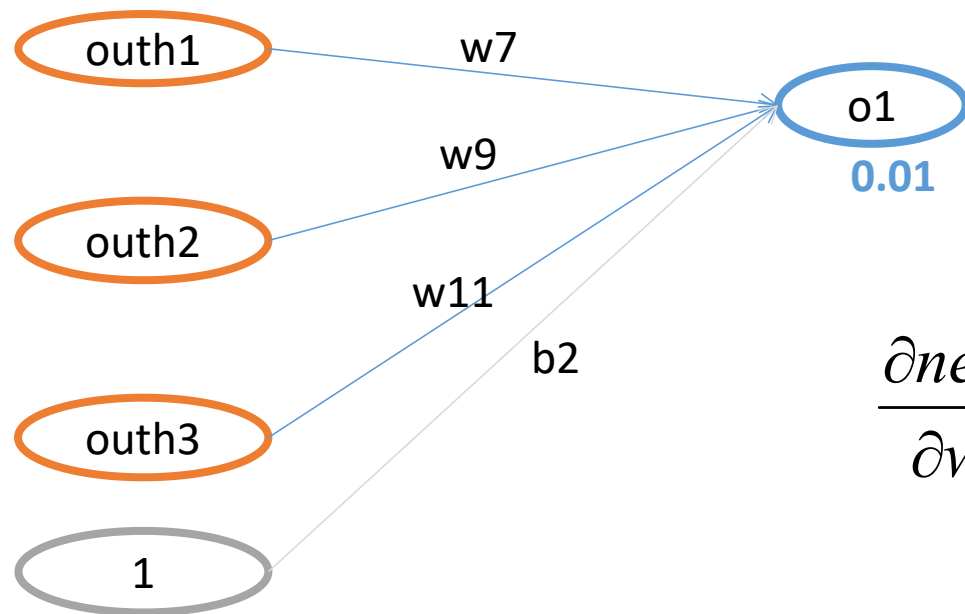
$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.891090(1 - 0.891090) = 0.097049$$

$$out'_{o1} = \frac{e^{-net}}{(1 + e^{-net})^2} = \frac{1 + e^{-net} - 1}{(1 + e^{-net})^2} = \frac{1}{1 + e^{-net}} - \frac{1}{(1 + e^{-net})^2} = out_{o1}(1 - out_{o1})$$

## BP算法例子-BP过程 (W7)

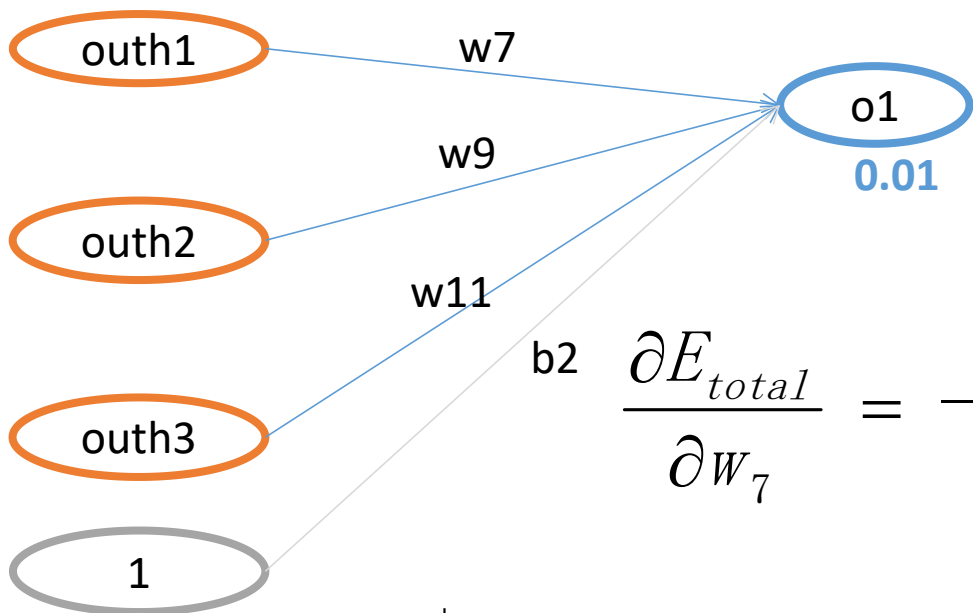
$$net_{o1} = w_7 * out_{h1} + w_9 * out_{h2} + w_{11} * out_{h3} + b_2 * 1$$



$$\frac{\partial net_{o1}}{\partial w_7} = 1 * out_{h1} * w_7^{(1-1)} + 0 + 0 + 0 = 0.912934$$

$$\frac{\partial E_{total}}{\partial w_7} = 0.88109 * 0.097049 * 0.912934 = 0.078064$$

## BP算法例子-BP过程(W7)



$$E_{o1} = \frac{1}{2} (\text{target}_{o1} - out_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_7}$$

$$\frac{\partial E_{total}}{\partial w_7} = -(\text{target} - out_{o1}) * out_{o1} * (1 - out_{o1}) * out_{h1}$$

$$w_7^+ = w_7 + \Delta w_7 = w_7 - \eta \frac{\partial E_{total}}{\partial w_7} = 0.4 - 0.5 * 0.078064 = 0.360968$$

$$w_8^+ = 0.453383$$

$$w_9^+ = 0.458137$$

$$w_{10}^+ = 0.553629$$

$$w_{11}^+ = 0.557448$$

$$w_{12}^+ = 0.653688$$



## BP算法例子-BP过程(W1)

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1} = \left( \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} \right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = -(\text{target}_{o1} - out_{o1}) * out_{o1} * (1 - out_{o1}) * w_7$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = -(0.01 - 0.891090) * 0.891090 * (1 - 0.891090) * 0.360968 = 0.030866$$

## BP算法例子-BP过程(W1)

$$\frac{\partial E_{o2}}{\partial out_{h1}} = \frac{\partial E_{o2}}{\partial out_{o2}} * \frac{\partial out_{o2}}{\partial net_{o2}} * \frac{\partial net_{o2}}{\partial out_{h1}} = -(\text{target}_{o2} - out_{o2}) * out_{o2} * (1 - out_{o2}) * w8$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.011204$$

$$w_1^+ = w_1 + \Delta w_1 = w_1 - \eta \frac{\partial E_{total}}{\partial w_1} = 0.1 - 0.5 * 0.011204 = 0.094534$$

## BP算法例子-BP过程

$$w_1^+ = 0.094534$$

$$w_2^+ = 0.139069$$

$$w_3^+ = 0.198211$$

$$w_4^+ = 0.246422$$

$$w_5^+ = 0.299497$$

$$w_6^+ = 0.348993$$

$$w_7^+ = 0.360968$$

$$w_8^+ = 0.453383$$

$$w_9^+ = 0.458137$$

$$w_{10}^+ = 0.553629$$

$$w_{11}^+ = 0.557448$$

$$w_{12}^+ = 0.653688$$

$$b_1 = 0.35$$

$$b_2 = 0.65$$

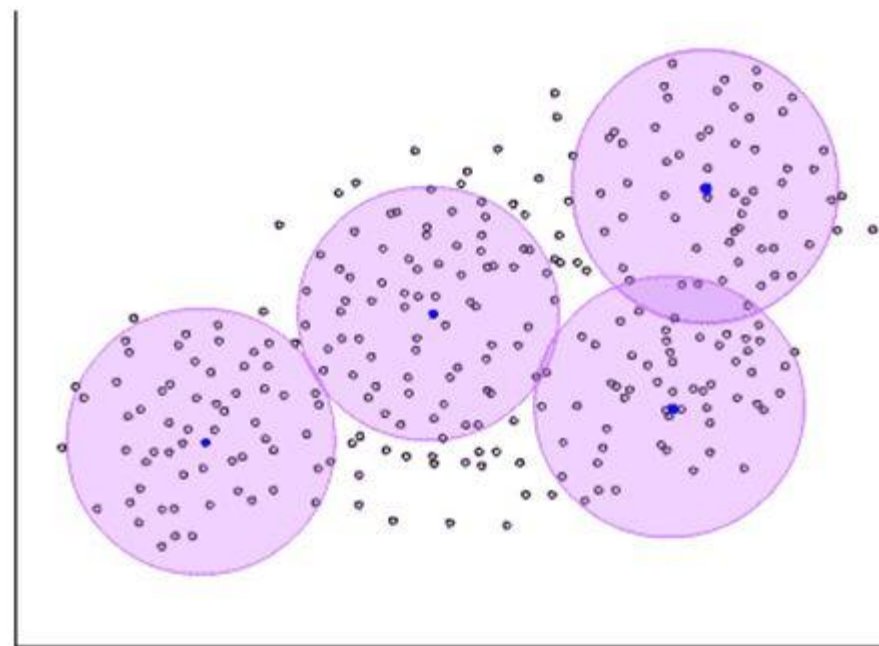
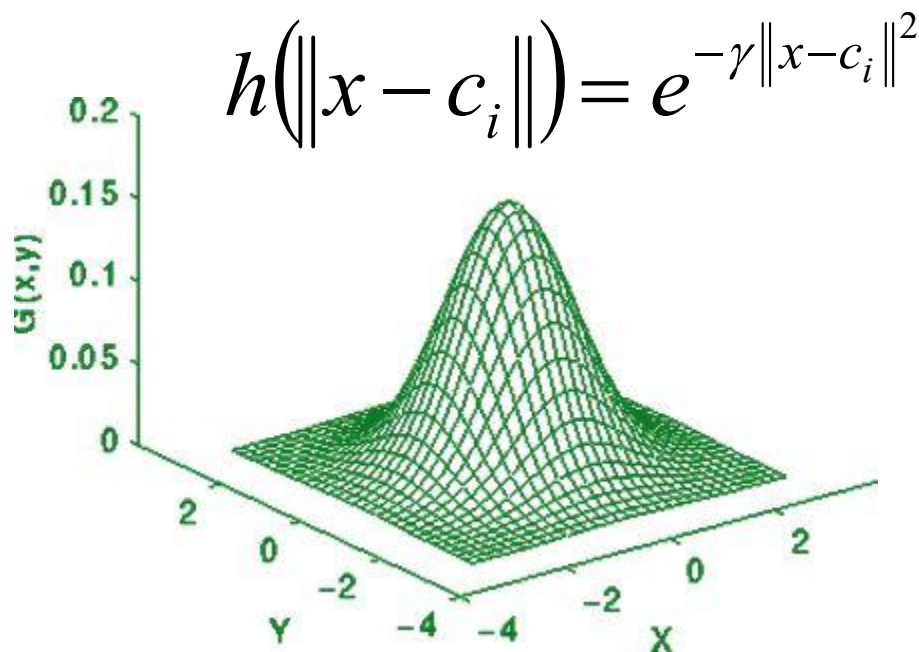
## BP算法例子-FP多次迭代效果

- 第10次迭代结果:  $O = (0.662866, 0.908195)$
- 第100次迭代结果:  $O = (0.073889, 0.945864)$
- 第1000次迭代结果:  $O = (0.022971, 0.977675)$

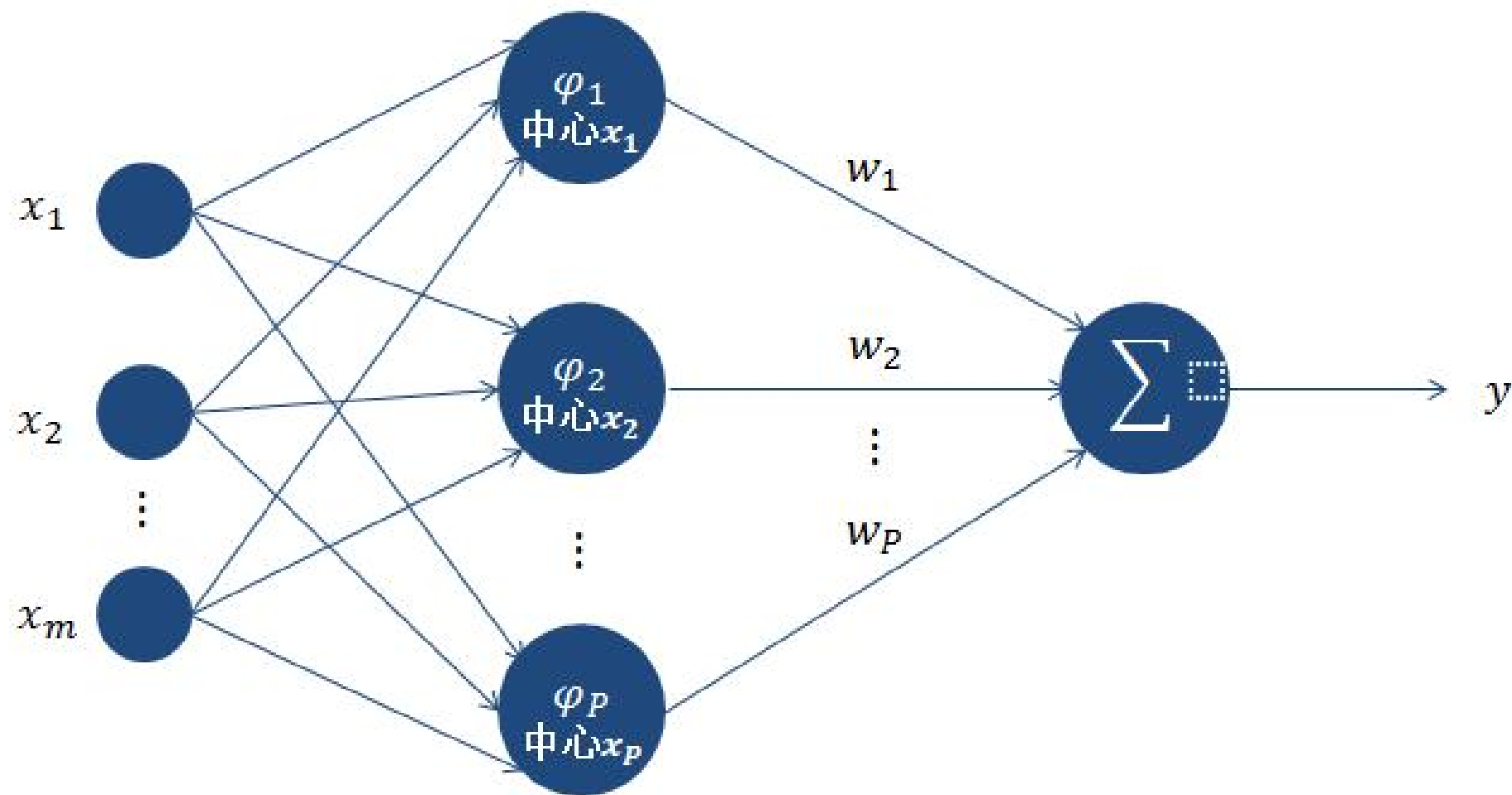
$$w^0 = \begin{pmatrix} 0.1, 0.15, 0.2, 0.25, \\ 0.3, 0.35, 0.4, 0.45, \\ 0.5, 0.55, 0.6, 0.65 \end{pmatrix} \quad w^{1000} = \begin{pmatrix} 0.214925, & 0.379850, & 0.262855, \\ 0.375711, & 0.323201, & 0.396402, \\ -1.48972, & 0.941715, & -1.50182, \\ 1.049019, & -1.42756, & 1.151881 \end{pmatrix}$$

## RBF神经网络

- RBF神经网络通常只有三层，即输入层、中间层和输出层。其中中间层主要计算输入 $x$ 和样本矢量 $c$ （记忆样本）之间的欧式距离的Radial Basis Function (RBF) 的值，输出层对其做一个线性的组合。



## RBF网络的原理



## RBF神经网络

- RBF神经网络的训练可以分为两个阶段：
  - 第一阶段为无监督学习，从样本数据中选择记忆样本/中心点；可以使用聚类算法，也可以选择随机给定的方式。
  - 第二阶段为监督学习，主要计算样本经过RBF转换后，和输出之间的关系/权重；可以使用BP算法计算、也可以使用简单的数学公式计算。

## RBF神经网络

- RBF网络能够逼近任意非线性的函数(因为使用的是一个局部的激活函数。在中心点附近有最大的反应；越接近中心点则反应最大，远离反应成指数递减；就相当于每个神经元都对应不同的感知域)。
- 可以处理系统内难以解析的规律性，具有很好的泛化能力，并且具有较快的学习速度。
- 有很快的学习收敛速度，已成功应用于非线性函数逼近、时间序列分析、数据分类、模式识别、信息处理、图像处理、系统建模、控制和故障诊断等。
- 当网络的一个或多个可调参数（权值或阈值）对任何一个输出都有影响时，这样的网络称为**全局逼近网络**。由于对于每次输入，网络上的每一个权值都要调整，从而导致全局逼近网络的学习速度很慢，比如BP网络。
- 如果对于输入空间的某个局部区域只有少数几个连接权值影响输出，则该网络称为**局部逼近网络**，比如RBF网络。



## RBF和BP神经网络

- BP神经网络(使用Sigmoid激活函数)是**全局逼近**； RBF神经网络(使用径向基函数作为激活函数)是**局部逼近**；
- 相同点：
  - RBF神经网络中对于权重的求解也可以使用BP算法求解。
- 不同点：
  - 中间神经元类型不同(RBF:径向基函数； BP:Sigmoid函数)
  - 网络层次数量不同(RBF:3层； BP:不限制)
  - 运行速度的区别(RBF:快； BP:慢)

## 神经网络之DNN问题

- 一般来讲，可以通过增加神经元和网络层次来提升神经网络的学习能力，使其得到的模型更加能够符合数据的分布场景；但是实际应用场景中，神经网络的层次一般情况不会太大，因为太深的层次有可能产生一些求解的问题
- DNN网络中，需要训练的参数数量特别大。
- 在DNN的求解中有可能存在两个问题：**梯度消失**和**梯度爆炸**；我们在求解梯度的时候会使用到链式求导法则，实际上就是一系列的连乘，如果每一层都小于1的话，则梯度越往前乘越小，导致梯度消失，而如果连乘的数字在每层都是大于1的，则梯度越往前乘越大，导致梯度爆炸。

## 神经网络案例/作业

- 使用Python实现BP神经网络实现对公路客运量即公路货运量预测案例
- Python实现RBF神经网络
- SimpleNeuralNetwork简单神经网络实现手写数字案例（TensorFlow实现）

