人工神经网络

卜晨阳

邮箱：**chenyangbu@hfut.edu.cn**

人工神经网络的应用？

- https://www.bilibili.com/video/av19184124?from=search&seid=15202434440337473404
- https://www.bilibili.com/video/av26004653?from=search&seid=18132351520209860073

本章内容

- 人工神经网络简介
- 人工神经元结构
  - 输入信号
  - 激活函数
  - 几何构型
  - 学习规则

生物神经系统

- 人脑：复杂的、非线性的、并行的“计算机”。
  - 具有模式识别、感知和电机控制人物的能力。

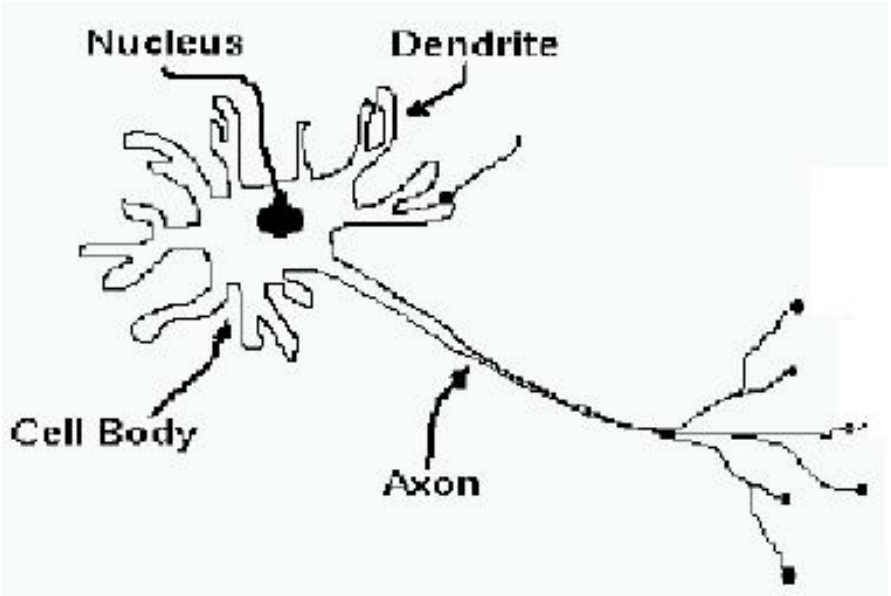
- 据估计，人类脑皮层中有100亿~5000亿数量级的神经元和60万亿的突触。这些神经元排列成1000个主模块，每个有大约500个神经网络。

- 是否有可能对人脑建模？
  - 现在还不行。当前的神经建模仅仅是针对解决特定问题所建立的小规模的人工神经网络。

生物神经系统

- 生物神经系统的基本构成单元是神经细胞，称为神经元。

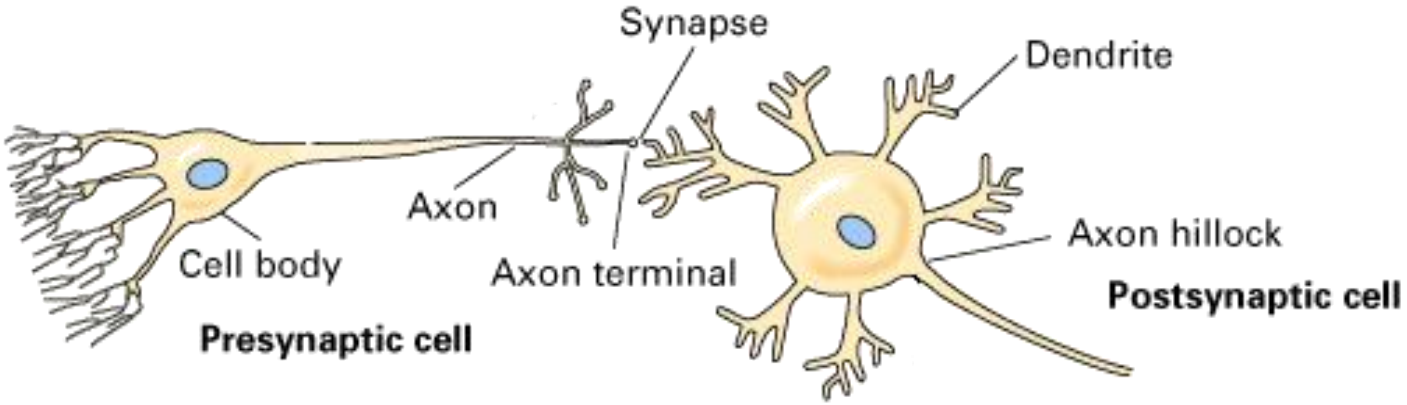
- 一个神经元由一个细胞体、大量树突和一个轴突组成。



- **Nucleus**：神经元
- **Cell Body**：细胞体
- **Dendrite**：树突
- **Axon**：轴突

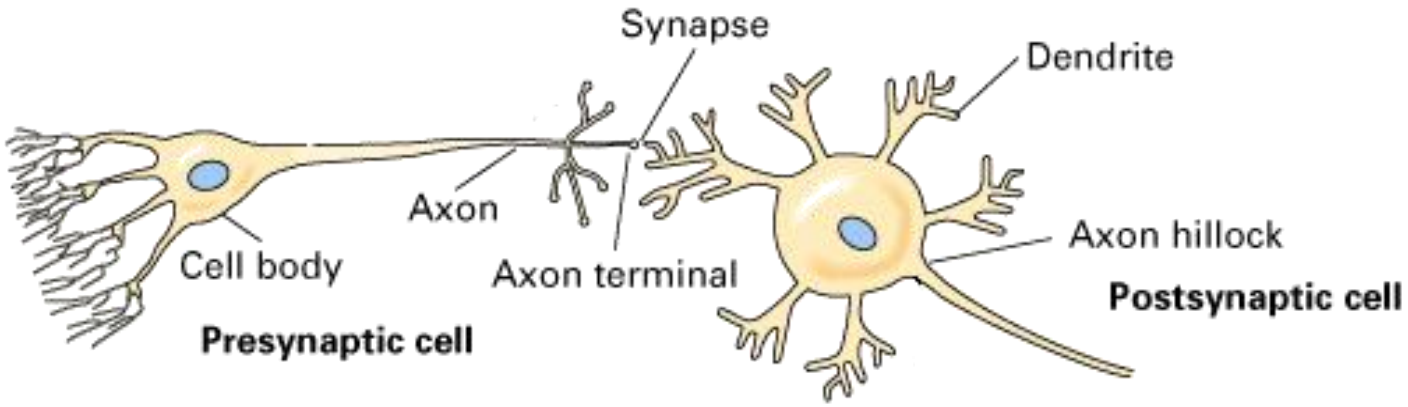
生物神经元

- 神经元之间互相广泛地连接，这种连接是由一个神经元的轴突和另一个神经元的树突相连接来实现的。



- **Synapse**：突触
- **Axon terminal**：轴突终端
- **Axon hillock** ：轴丘
- 信号输出端：轴突
- 信号输入端：树突

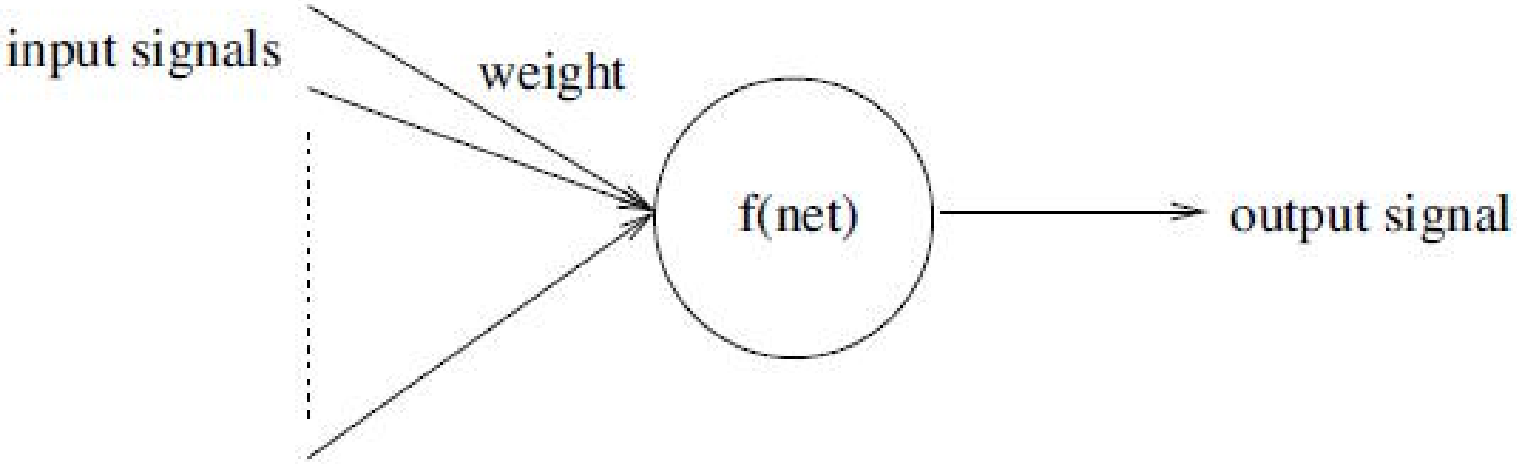
生物神经元



- 只有当细胞“激发”，信号才能被传播到神经元的轴突。
- 一个神经元既可以抑制、也可以激活一个信号。

人工神经元的结构

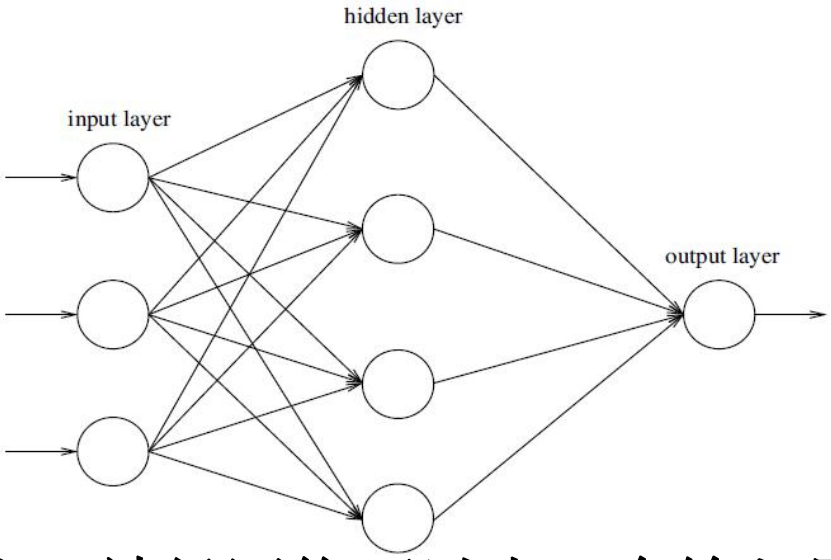
- 每一个神经元从外部环境或其他人工神经元接收信号，当它被激发后就将信号传递给所有与之相连的人工神经元。



- *f*(*net*)：激活函数。
- 对一个神经元的激发或抑制信号的强度，都是由激活函数来控制的。

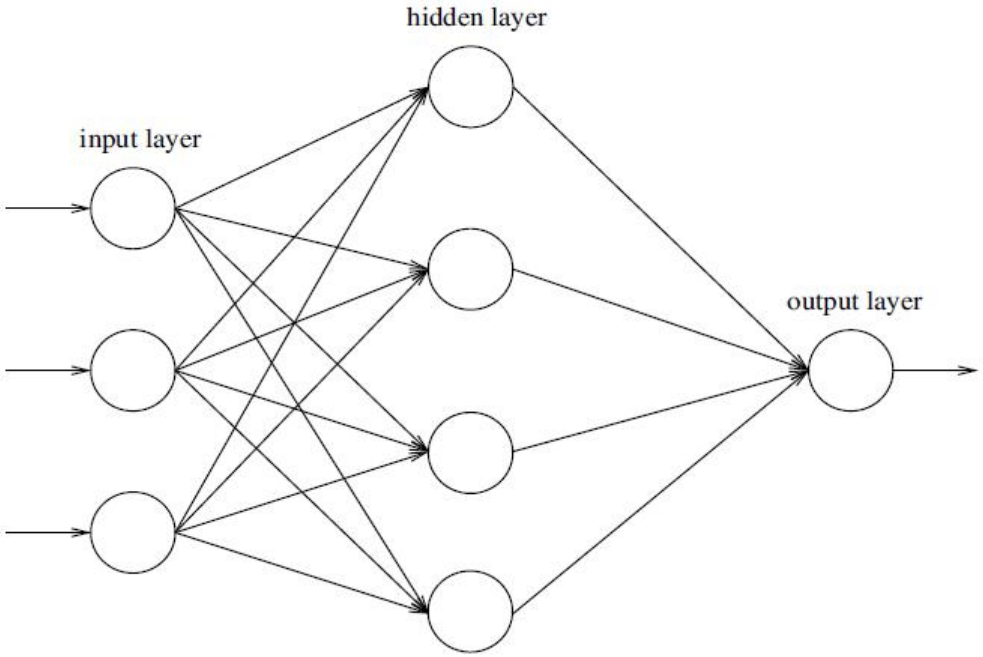
人工神经网络

- 人工神经网络（**Artificial Neural Network**）是一个由人工神经元（**Artificial Neuron**）组成的分层网络。



- 通常，一个人工神经网络可以由一个输入层、多个隐层和一个输出层组成。
  - 一个上层的神经元与下一层的神经元是彼此全连接或部分连接的。
  - 有时，也可能与前一层的反馈连接。

人工神经网络



- 本质上，神经网络是一个从$\mathbf{R^I}$到$\mathbf{R^K}$的非线性映射的实现，即：

$$f_{NN}: \ R^I \rightarrow R^K$$

  - 其中，I和K分别是输入空间和目标（期望输出）空间的维数。

人工神经网络的类型

- 单层神经网络，例如Hopfield网络。
- 多层前馈神经网络，例如标准反向传播（standard backpropagation）、函数连接（functional link）、乘积单元网络（product unit networks）。
- 时间神经网络，例如simple recurrent networks，time-delay neural networks。
- 自组织神经网络，例如Kohonen自组织特征映射和学习向量量化器。
- 监督和非监督神经网络的结合，例如某些径向基函数神经网络。

人工神经网络的应用领域

- 分类：预测输入向量的类别
- 模式匹配：产生一个与给定输入向量最关联的模式
- 模式完善：补全一个给定输入向量缺失的部分
- 优化：在一个优化问题中找出参数的最优值
- 控制：在给定一个输入向量的情况下，给出一个恰当动作的建议
- 函数拟合/时间序列建模：学习输入向量和期望输出向量之间的函数关系
- 数据挖掘：从数据中发现隐藏的模式

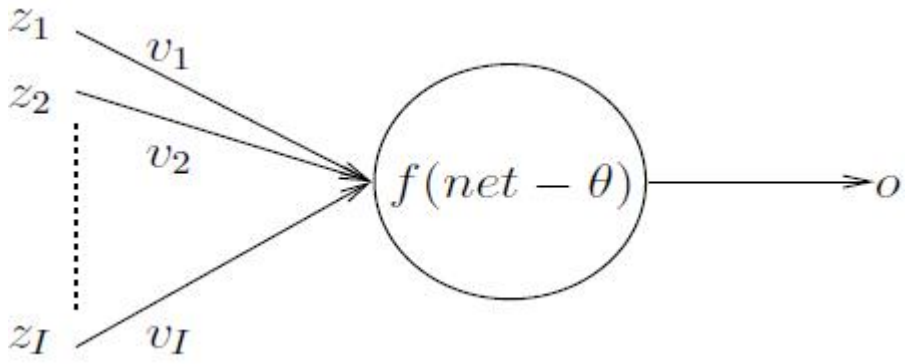
本章内容

- 人工神经网络简介
- 人工神经元结构
  - 输入信号
  - 激活函数
  - 几何构型
  - 学习规则

人工神经元的结构

- 一个人工神经元（即神经元），通常实现了从R^I到[0, 1]或[1, −1]的一个非线性映射，映射区间取决于使用的激活函数。

$$f_{AN}: \mathrm{R}^{\mathrm{I}} \to [0,1] \qquad 或 \qquad f_{AN}: \mathrm{R}^{\mathrm{I}} \to [-1,1]$$



$$z = (z_1, z_2, \cdots, z_I)$$

- 输入向量**z**来自环境或者其他神经元。每一个输入信号z_i都关联一个增强或者衰减该输入信号的权重v_i。
- 输出信号的强度也受阈值θ的影响， θ也称为偏置。

输入信号

- 一个神经元的网络输入信号通常为所有输入信号的加权和，这类神经元称为求和单元。

$$net = \sum_{i=1}^{I} z_i v_i$$

- 计算网络输入信号的另一种方法是采用所有输入信号的乘积，这类神经元称为乘积单元。

$$net = \prod_{i=1}^{I} z_i^{v_i}$$

激活函数

- 函数f_{AN}接受网络输入信号和偏置，并且决定神经元的输出（或者激活强度），这个函数称作激活函数。

- 一般而言，激活函数是单调递增映射，其中：

$$f_{AN}(-\infty)=0 \text{或者} f_{AN}(-\infty)=-1$$

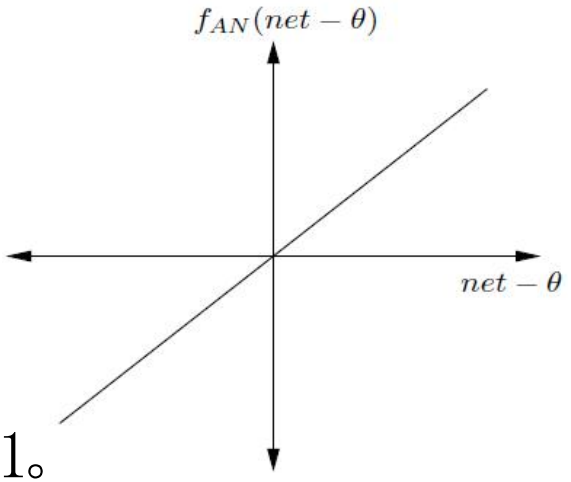
且

$$f_{AN}(\infty)=1$$

常用的激活函数

- **线性函数**

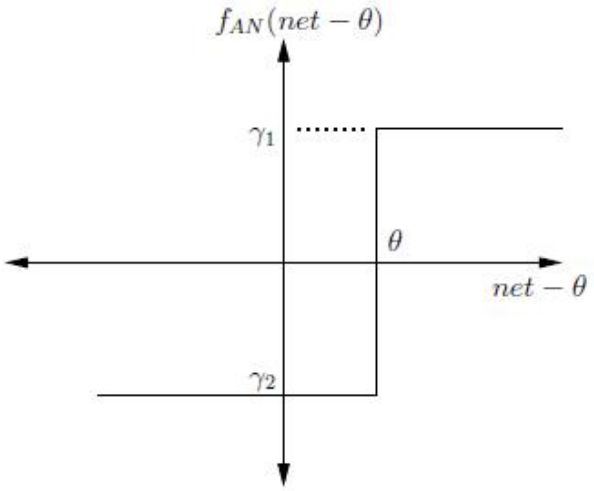
$$f_{AN}(net-\theta)=\lambda(net-\theta)$$



注意：$f_{AN}(-\infty)\neq 0$或-1且$f_{AN}(\infty)\neq 1$。

- **阶跃函数**

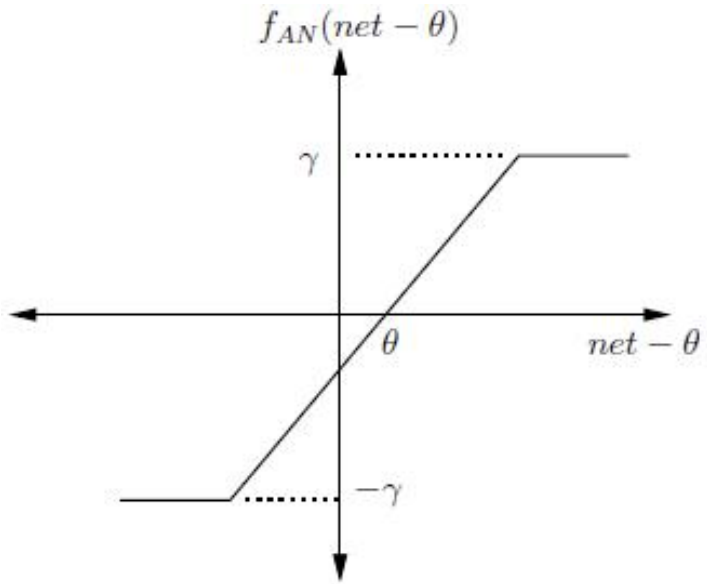
$$f_{AN}(net-\theta)=\begin{cases}\gamma_1 & \text{如果} net\geq\theta\\ \gamma_2 & \text{如果} net\leq\theta\end{cases}$$



常用的激活函数

- 斜坡函数

$$f_{AN}(net-\theta)=\begin{cases}\gamma & \text{如果} net-\theta \geq \varepsilon \\ net-\theta & \text{如果} -\varepsilon < net-\theta < \varepsilon \\ -\gamma & \text{如果} net-\theta \leq -\varepsilon\end{cases}$$

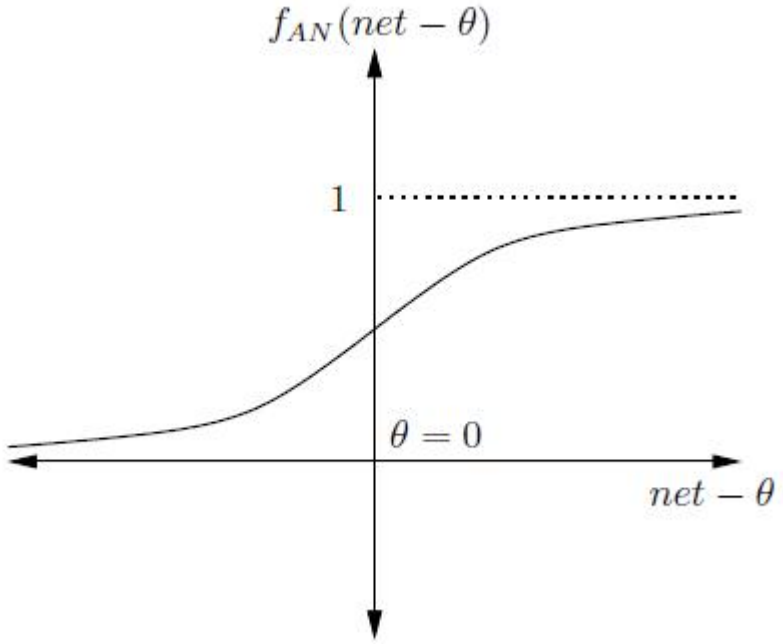


常用的激活函数

- **Sigmoid函数**

$$f_{AN}(net-\theta)=\frac{1}{1+e^{-\lambda(net-\theta)}}$$

其中，参数λ控制函数的陡度，通常λ=1。



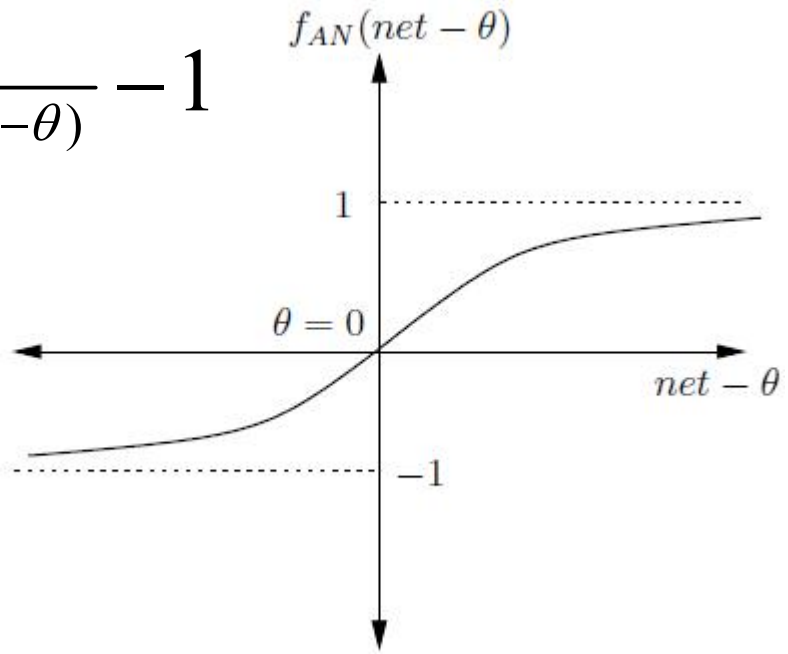
常用的激活函数

- 双正切函数

$$f_{AN}(net-\theta)=\frac{e^{\lambda(net-\theta)}-e^{-\lambda(net-\theta)}}{e^{\lambda(net-\theta)}+e^{-\lambda(net-\theta)}}$$

或近似为

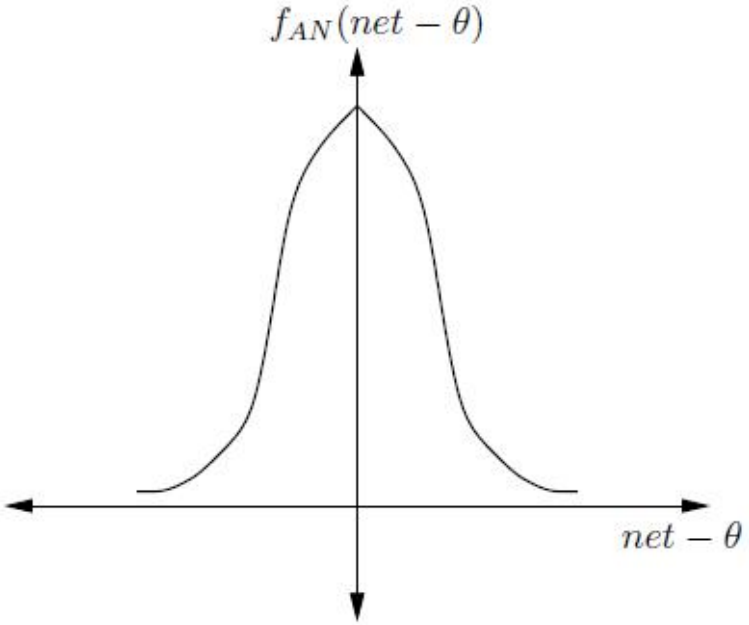
$$f_{AN}(net-\theta)=\frac{2}{1+e^{-\lambda(net-\theta)}}-1$$



常用的激活函数

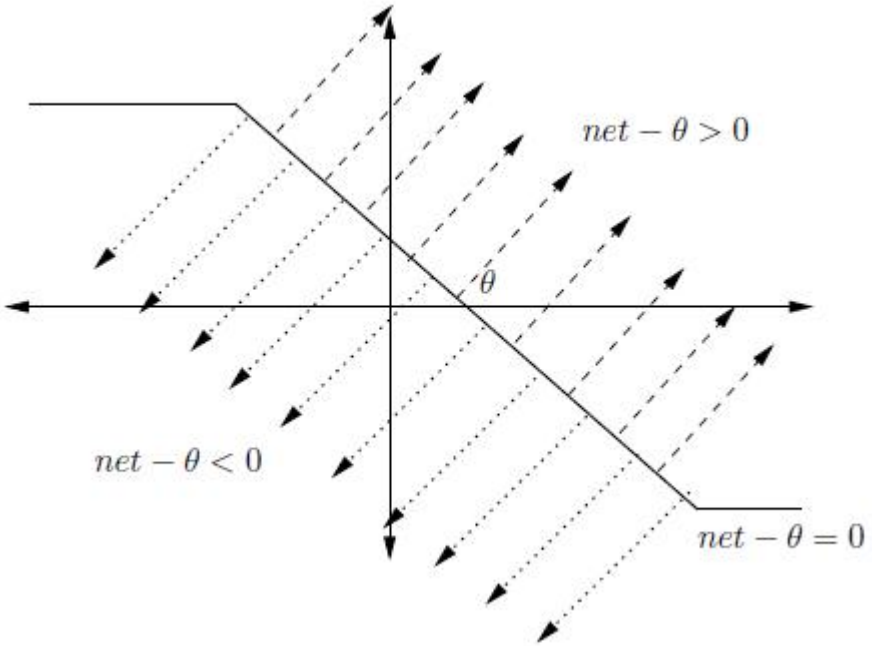
- 高斯函数
  - 其中，***net*－θ**是均值，σ是高斯分布的标准差。

$$f_{AN}(net-\theta)=e^{\frac{-(net-\theta)^2}{\sigma^2}}$$



几何构型

- 单个神经元可实现没有任何错误的线性可分函数。
  - 线性可分是指神经元可以通过一个*I*维超平面分割*I*维的输入向量空间，将超过阈值的响应和那些低于阈值的响应分割开。
- 例如，斜坡激活函数神经元的决策边界：



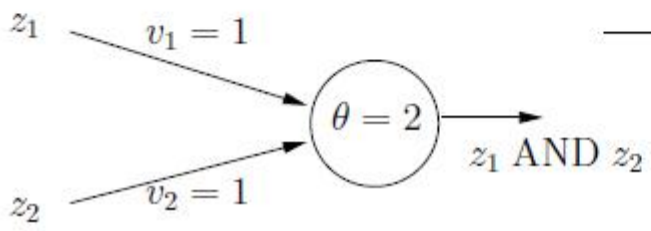
例1

- 用单个感知器实现布尔函数**AND**
  - 感知器是由美国计算机科学家罗森布拉特（**F. Roseblatt**）于**1957**年提出的，是一种最简单形式的前馈式人工神经网络，是一种线性分类器。

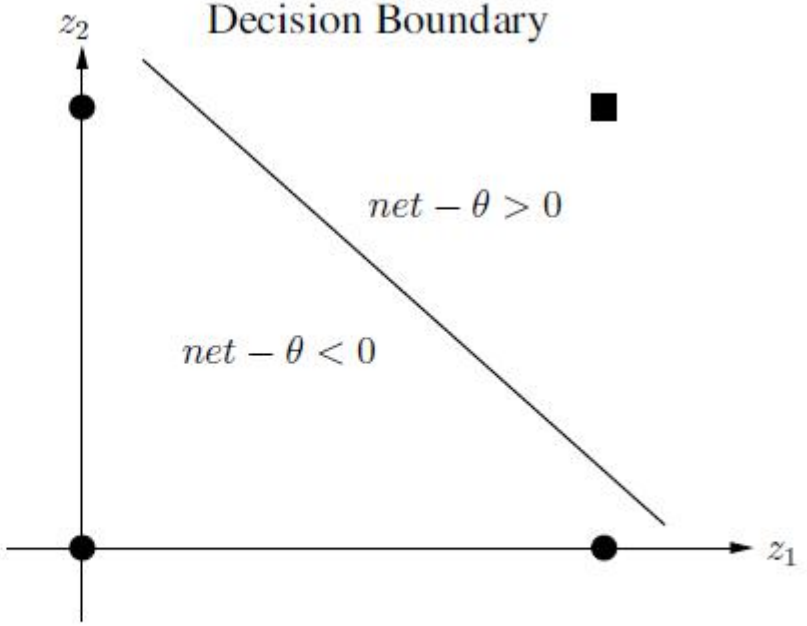
Truth Table

z_1	z_2	z_1 AND z_2
0	0	0
0	1	0
1	0	0
1	1	1

The Artificial Neuron



Decision Boundary



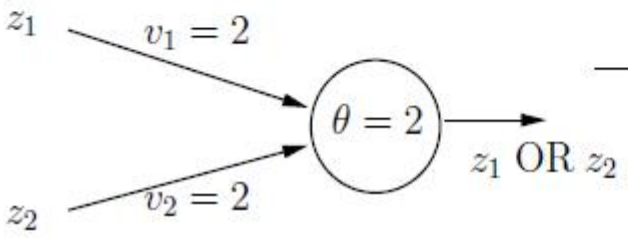
例2

- 用单个感知器实现布尔函数OR

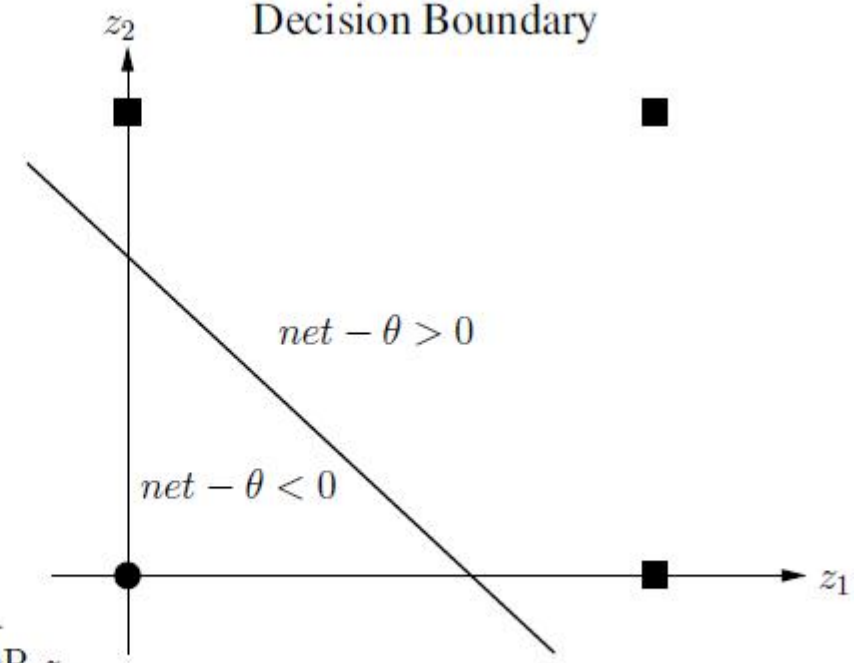
Truth Table

z_1	z_2	z_1 OR z_2
0	0	0
0	1	1
1	0	1
1	1	1

The Artificial Neuron



Decision Boundary



布尔函数AND和OR

- 布尔函数AND和OR都是线性可分函数的例子。
  - 对于这种简单函数，可以很容易手工决定偏置和权重的取值。

- 在给定输入信号以及θ的前提下，权值v_i的取值可以通过求解下式得到。

对于每个输入模式$z=(z_1,z_2,\cdots,z_I)$，应有

$$\sum_i z_i v_i-\theta>0 \text{或} \sum_i z_i v_i-\theta<0$$

思考：

- 如何实现异或函数？

Truth Table

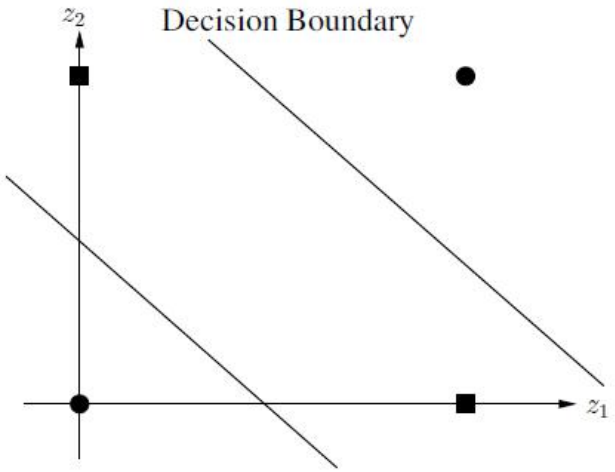
z_1	z_2	z_1 XOR z_2
0	0	0
0	1	1
1	0	1
1	1	0

布尔函数XOR

- 布尔函数**XOR**是非线性可分布尔函数。
  - 单个感知器不能实现该函数。
  - 若使用单个感知器，则所能获得的最佳正确率是**75%**。

Truth Table

z_1	z_2	z_1 XOR z_2
0	0	0
0	1	1
1	0	1
1	1	0



- 为了能够学习非线性可分的函数，需要一个由若干神经元组成的分层的 神经网络。
  - 例如，异或函数需要两个输入单元、两个隐层单元和一个输出单元。

习题

1. 下列哪一个布尔函数可以通过一个使用求和单元的单个神经元来实现？给出权值和阈值的取值来证明你的答案。

$(a)\ z_1 z_2 \bar{z}_3$

$(b)\ z_1 \bar{z}_2 + \bar{z}_1 z_2$

$(c)\ z_1 + z_2$

其中，$z_1 z_2$表示z_1和z_2取与操作，$z_1 + z_2$表示z_1和z_2取或操作，$\bar{z}_1$表示z_1取非操作。

学习

- 是否存在一个自动的方法来确定权值v_i和阈值θ的值？
  - 对于简单问题，可以很容易计算。
  - 但是，假定出来数据以外，并没有关于函数的任何先验知识，那么将如何计算v_i和θ的值呢？
    - 通过学习！

- 学习包括调整权值和阈值，直到满足要求。有三种主要的学习方法：
  - 监督学习
  - 无监督学习
  - 增强学习

学习方法

- 监督学习（Supervised Learning）
  - 提供训练集，目标是调整权值以使得神经元的真实输出和目标输出之间的误差最小化。
- 无监督学习（Unsupervised Learning）
  - 在没有外部辅助的情况下，从输入数据中发掘模式和特征。许多无监督的学习算法是对训练模式进行聚类。
- 强化学习（Reinforcement Learning）
  - 目标是对性能良好的神经元（或神经网络的一部分）给予奖励，而对性能不佳的神经元给予惩罚。

增广向量

- 人工神经元的网络输入信号（假定求和单元）：

$$net = \sum_{i=1}^{I} z_i v_i - \theta$$

$$= \sum_{i=1}^{I} z_i v_i + z_{i+1} v_{i+1}$$

$$= \sum_{i=1}^{I+1} z_i v_i$$

其中，$z_{i+1} = -1$，$v_{i+1} = \theta$。

- 输入向量被增广，从而使其包含一个成为偏置单元的额外输入单元z_{I+1}，而且权值v_{I+1}被用作阈值，目的是简化学习方程。

梯度下降学习规则

- 梯度下降法是最常用的方法。梯度下降法需要定义一个误差函数（目标函数）来衡量逼近目标时神经元的误差。

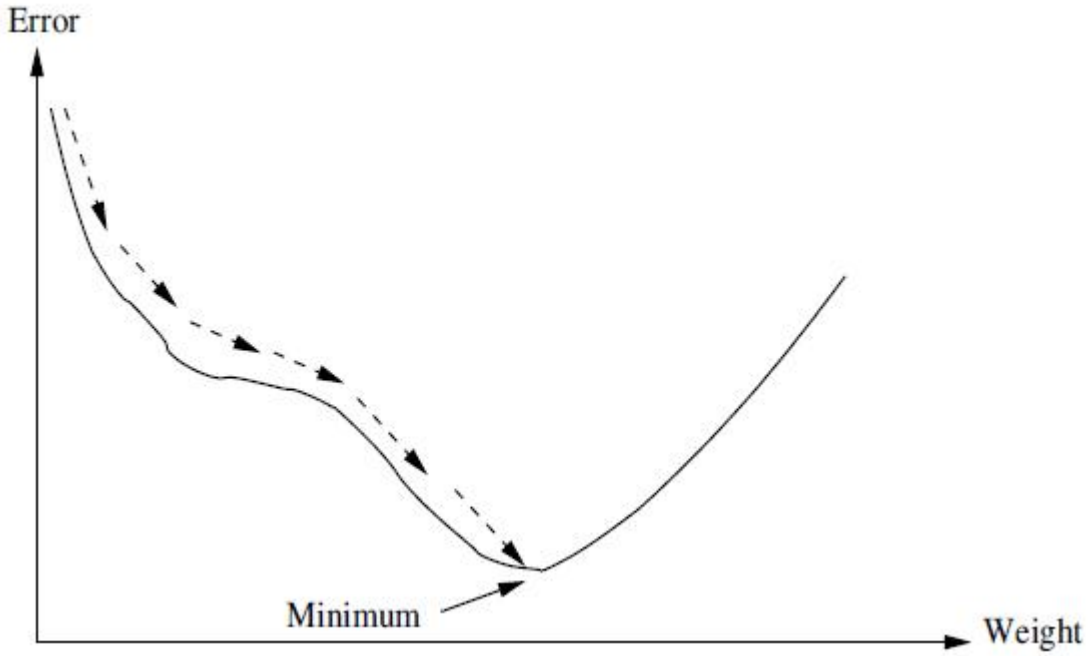
- 通常使用平方误差。

$$\varepsilon = \sum_{p=1}^{P_T} (t_p - o_p)^2$$

其中，t_p和o_p分别是第p个输入模式的目标输出和真实输出，P_T是训练集中输入—目标向量对（模式）的总数。

梯度下降学习规则

- 梯度下降的目标是找到使得ε最小的权值。
  - 这可以通过计算权值空间中ε的梯度，并沿负梯度方向移动权值向量来实现。



梯度下降学习规则

- 给定单个训练模式，一个神经元的权值更新公式为：

$$v_i(t) = v_i(t-1) + \Delta v_i(t)$$

$$\Delta v_i(t) = \eta\left(-\frac{\partial \varepsilon}{\partial v_i}\right)$$

$$\frac{\partial \varepsilon}{\partial v_i} = -2\left(t_p - o_p\right)\frac{\partial f}{\partial net_p} z_{i,p}$$

- η是学习率（即负梯度方向上所取的步长大小）。
- $z_{i,p}$是对应于模式p的第i个输入向量。
- 对于所有非连续的激活函数（例如阶跃和斜坡函数等），f相对于net_p（模式p的网络输入）的偏导数的计算存在问题。

Widrow-Hoff学习规则

假定$f = net_p$，则$\frac{\partial f}{\partial net_p} = 1$，有

$$\frac{\partial \varepsilon}{\partial v_i} = -2(t_p - o_p)z_{i,p}$$

那么权值更新公式为：

$$v_i(t) = v_i(t-1) + 2\eta(t_p - o_p)z_{i,p}$$

- **Widrow-Hoff学习规则，即最小均方（least-means-square）算法，是最早用于训练具有多个自适应线性神经元的分层神经网络算法之一。**

广义delta学习规则

- 广义delta学习规则是Widrow-Hoff学习规则的一个推广，该规则假定激活函数是可微的。

- 假定激活函数是Sigmoid函数，则

$$\frac{\partial f}{\partial net_p} = o_p(1-o_p)$$

$$\frac{\partial \varepsilon}{\partial v_i} = -2(t_p - o_p)o_p(1-o_p)z_{i,p}$$

注：$f_{AN}(net) = \dfrac{1}{1+e^{-net}}$

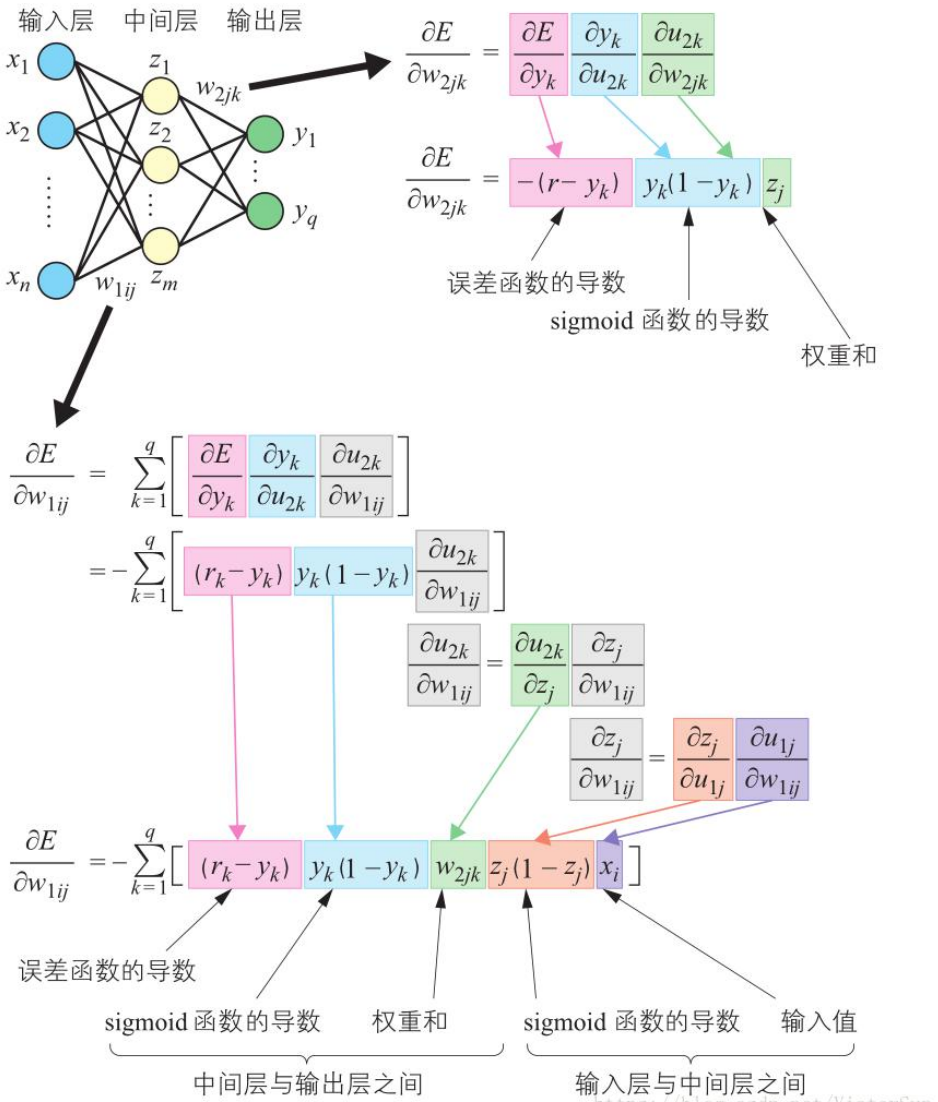
误差修正学习规则

- 误差修正学习规则使用二值激活函数，例如阶跃函数。权值仅当神经元引发错误时才被调整。

即，仅当$(t_p - o_p) = 1$或$(t_p - o_p) = -1$时，才使用下式调整。

$$v_i(t) = v_i(t-1) + 2\eta(t_p - o_p)z_{i,p}$$

偏导数计算过程



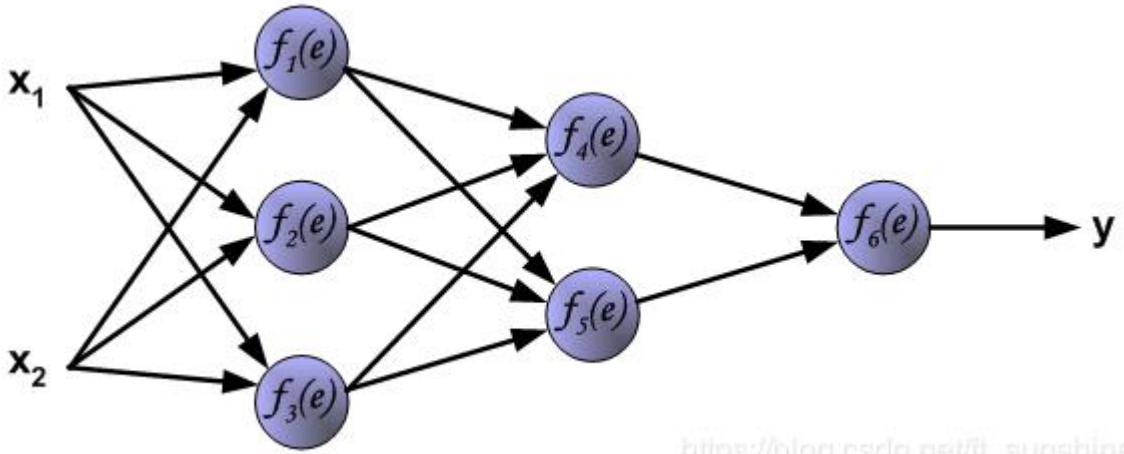
推导过程

1) $$\frac{\partial E}{\partial w_{2jk}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{2k}} \cdot \frac{\partial u_{2k}}{\partial w_{2jk}}$$
$$= -(r_k - y_k) \cdot y_k \cdot (1 - y_k) \cdot z_j$$

2) $$\frac{\partial E}{\partial w_{1ij}} = \frac{\partial E}{\partial w_{2jk}} \cdot \frac{\partial w_{2jk}}{\partial w_{1ij}}$$
$$= \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{2k}} \cdot \frac{\partial u_{2k}}{\partial w_{2jk}} \cdot \frac{\partial w_{2jk}}{\partial w_{1ij}}$$
$$= \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{2k}} \cdot \frac{\partial u_{2k}}{\partial w_{1ij}}$$
$$= \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{2k}} \cdot \frac{\partial u_{2k}}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{1ij}}$$
$$= \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{2k}} \cdot \frac{\partial u_{2k}}{\partial z_j} \cdot \frac{\partial z_j}{\partial u_{1j}} \cdot \frac{\partial u_{1j}}{\partial w_{1ij}}$$
$$= \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial u_{2k}} \cdot \frac{\partial u_{2k}}{\partial z_j} \cdot \frac{\partial f(u_{1j})}{\partial u_{1j}} \cdot \frac{\partial u_{1j}}{\partial w_{1ij}}$$
$$= -\sum_{k=1}^{q} \{(r_k - y_k) \cdot y_k \cdot (1 - y_k) \cdot w_{2jk} \cdot z_j \cdot (1 - z_j) \cdot x_i\}$$

前向传播与反向传播

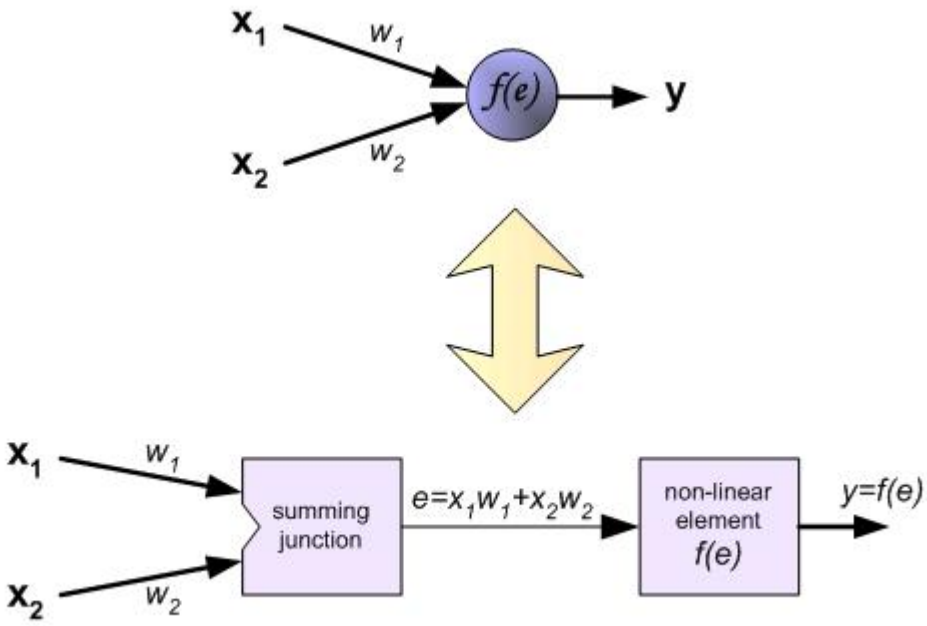
用一个简单的三层神经网络举例[1]



[1] 示例来自于网址：http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html, 文字来源于CSDN: https://blog.csdn.net/ft_sunshine/java/article/details/90221691

前向传播与反向传播

- 每个神经元由两部分组成
  - 第一部分（e）是输入值和权重系数乘积的和
  - 第二部分（f(e)）是一个激活函数（非线性函数）的输出， y=f(e)即为某个神经元的输出



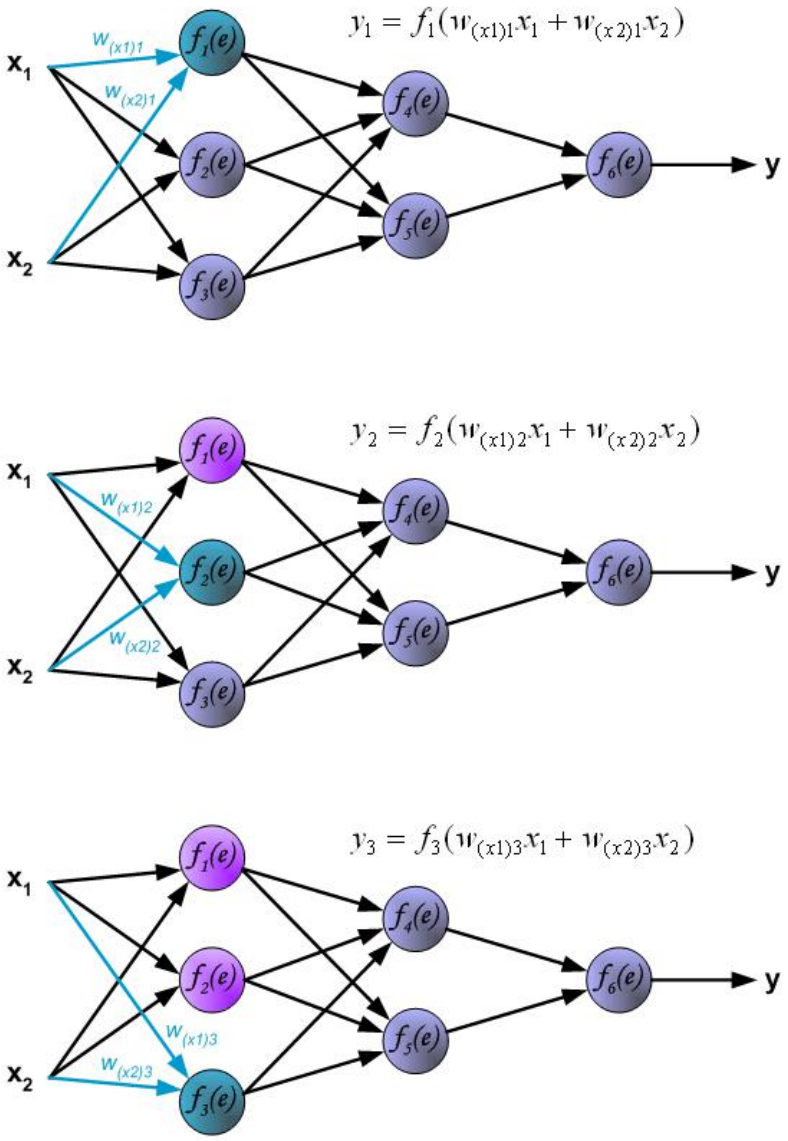
前向传播与反向传播

- 每个神经元由两部分组成
  - 第一部分（e）是输入值和权重系数乘积的和
  - 第二部分（f(e)）是一个激活函数（非线性函数）的输出， y=f(e)即为某个神经元的输出

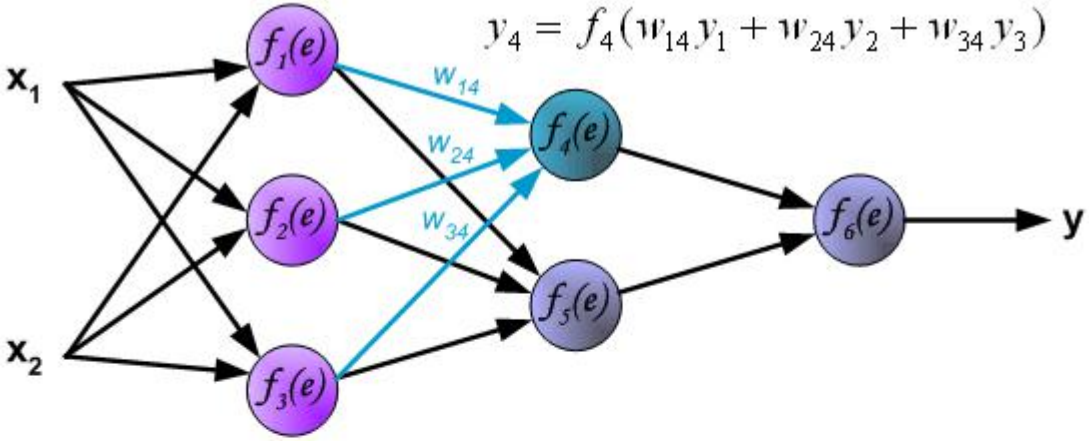
- 基本概念：
  - 正向传播：求损失
  - 反向传播：回传误差（基于链式求导法则）
  - 神经网络每层中每个神经元都可以根据误差信号修正每层的权重

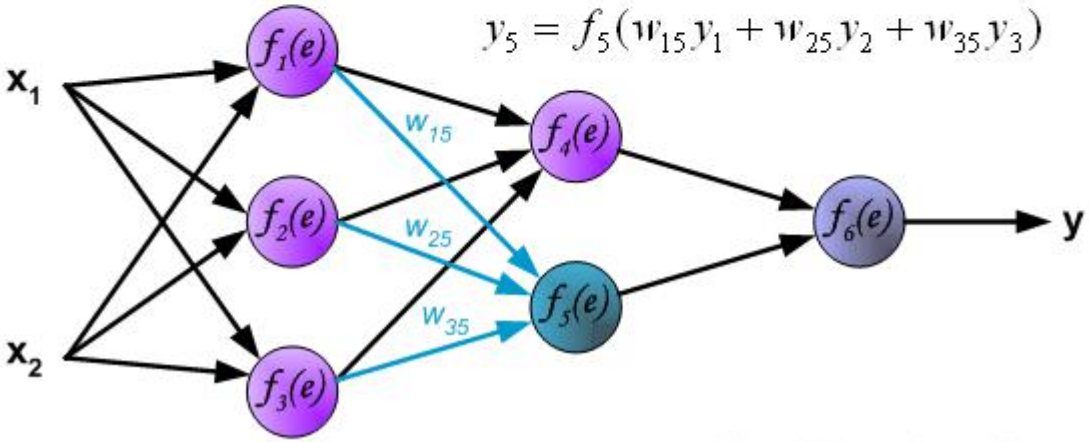
前向传播与反向传播

- 前向传播过程：



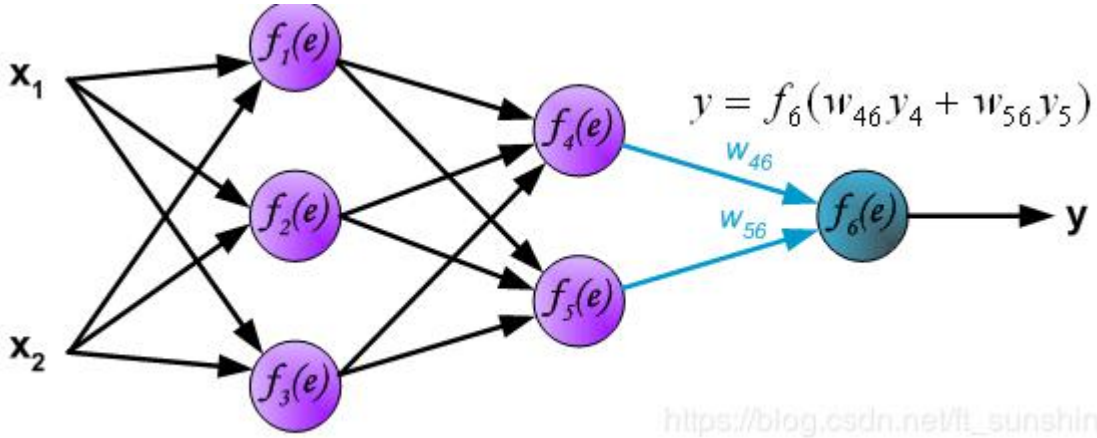
前向传播与反向传播





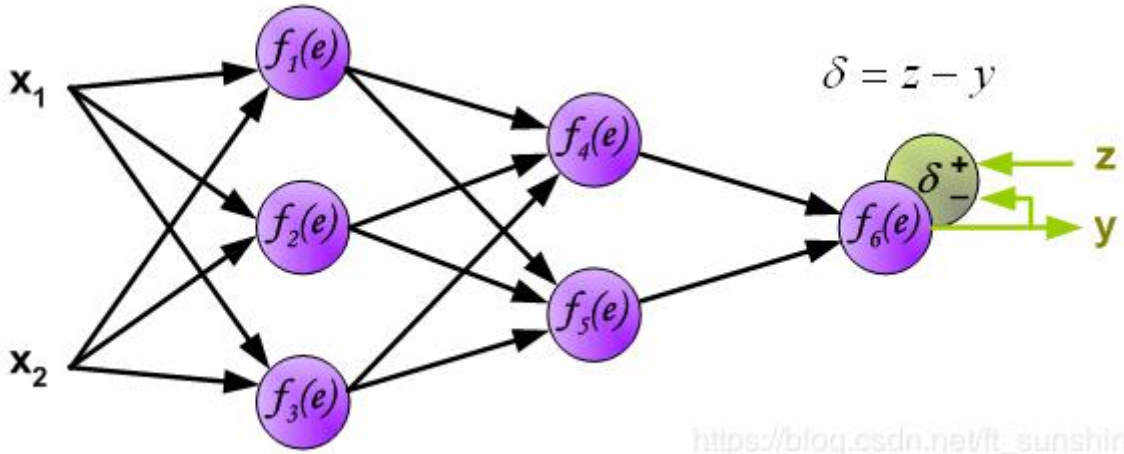
前向传播与反向传播

- 到这里为止，神经网络的前向传播已经完成，最后输出的**y**就是本次前向传播神经网络计算出来的结果（预测结果）
- 但这个预测结果不一定是正确的，要和真实的标签（**z**）相比较



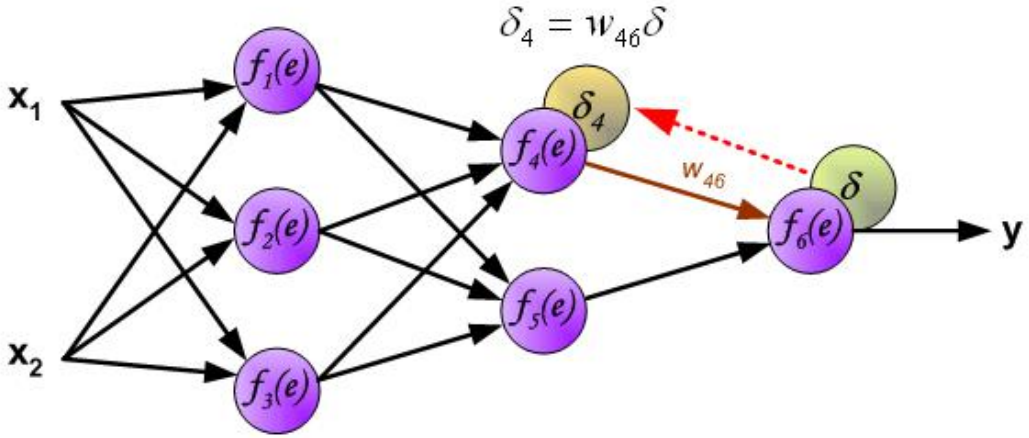
前向传播与反向传播

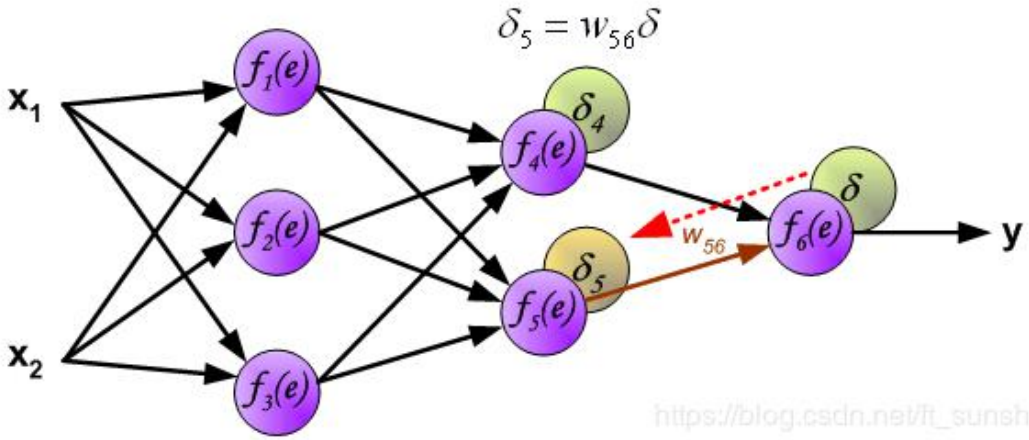
•和真实的标签（z）相比较，计算预测结果和真实标签的误差



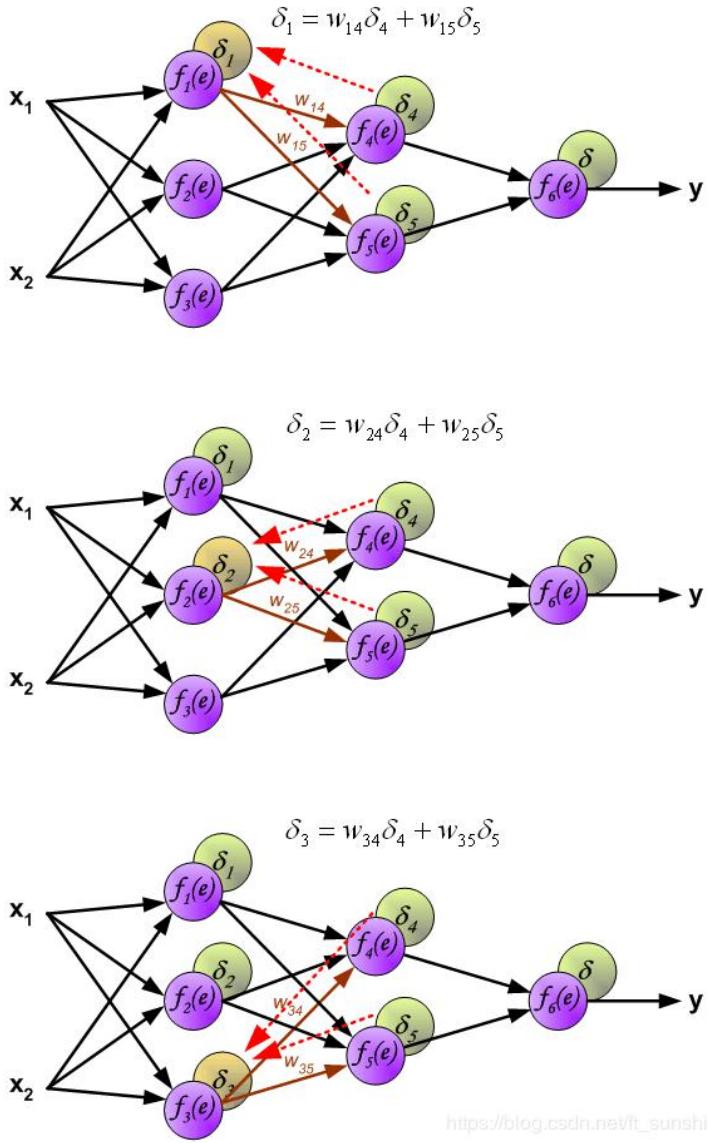
前向传播与反向传播

- 下面开始计算每个神经元的误差（δ）



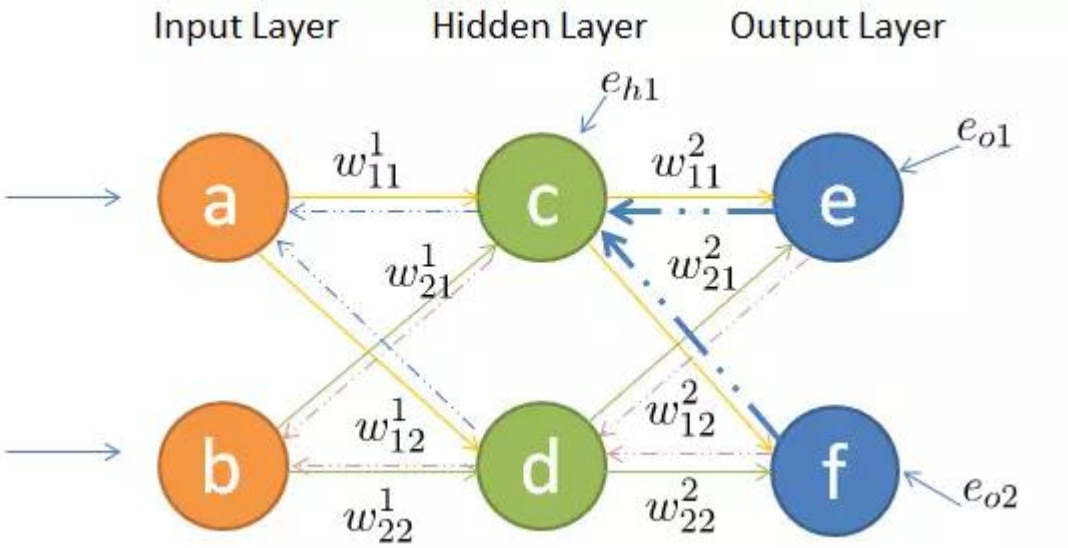


前向传播与反向传播



- 问题：
- 为什么采用与前向过程相同的权重？

反向传播的权重问题



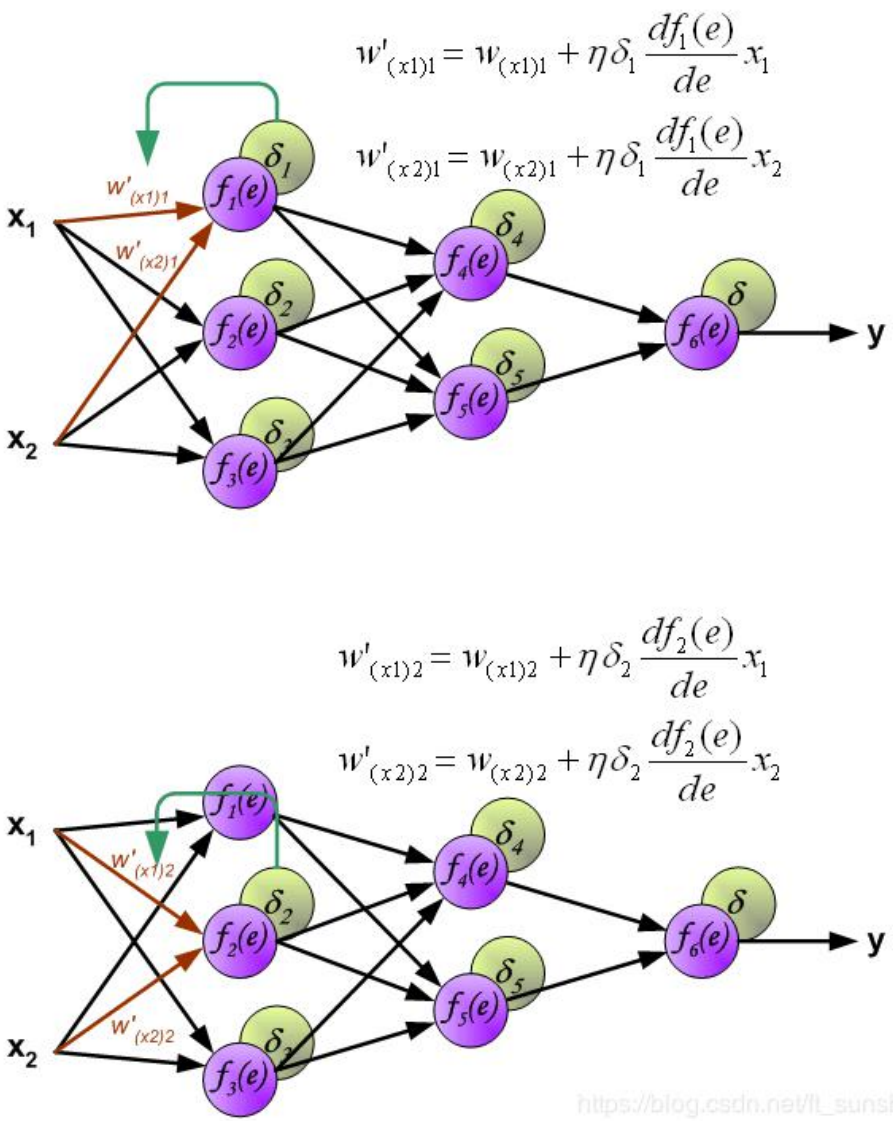
$$e_{h1} = \frac{w_{11}^2}{w_{11}^2 + w_{21}^2} \cdot e_{o1} + \frac{w_{12}^2}{w_{12}^2 + w_{22}^2} \cdot e_{o2}$$

$$e_{h2} = \frac{w_{21}^2}{w_{11}^2 + w_{21}^2} \cdot e_{o1} + \frac{w_{22}^2}{w_{12}^2 + w_{22}^2} \cdot e_{o2}$$

$$\begin{pmatrix} e_{h1} \\ e_{h2} \end{pmatrix} = \begin{pmatrix} \frac{w_{11}^2}{w_{11}^2+w_{21}^2} & \frac{w_{12}^2}{w_{12}^2+w_{22}^2} \\ \frac{w_{21}^2}{w_{11}^2+w_{21}^2} & \frac{w_{22}^2}{w_{12}^2+w_{22}^2} \end{pmatrix} \bullet \begin{pmatrix} e_{o1} \\ e_{o2} \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} e_{h1} \\ e_{h2} \end{pmatrix} = \begin{pmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{pmatrix} \bullet \begin{pmatrix} e_{o1} \\ e_{o2} \end{pmatrix}$$

前向传播与反向传播：链式求导

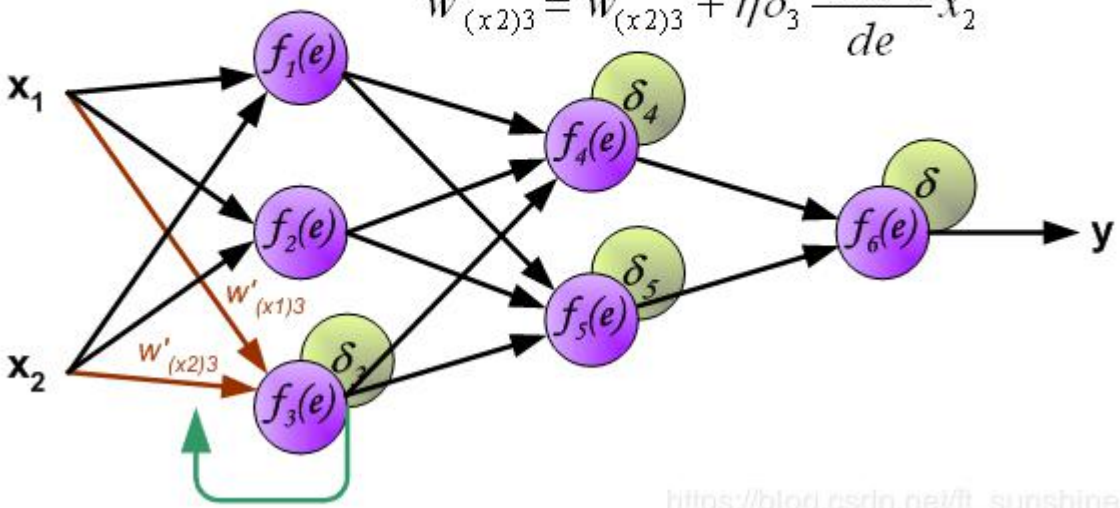
- 下面开始利用反向传播的误差，计算各个神经元（权重）的导数，开始反向传播修改权重



前向传播与反向传播

$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$

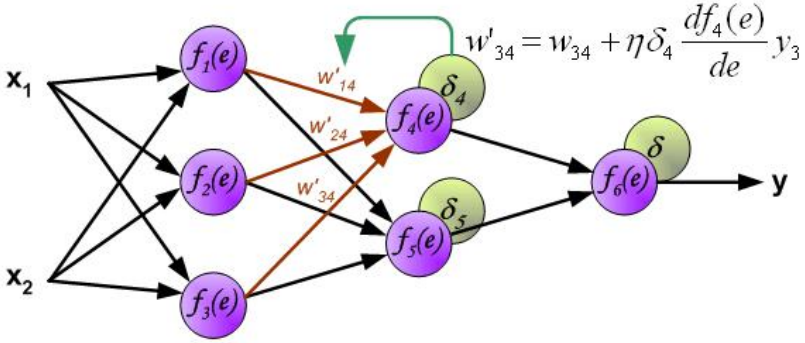


前向传播与反向传播

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

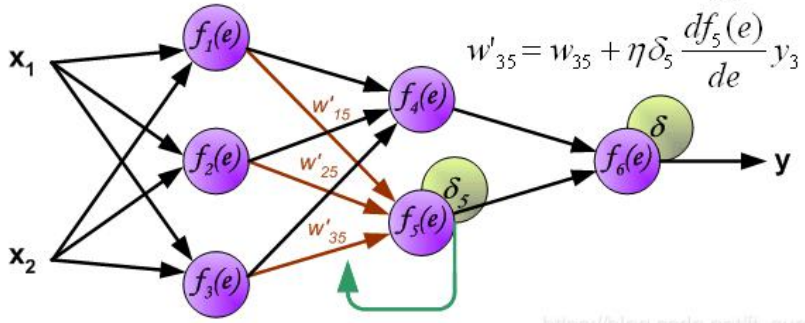
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



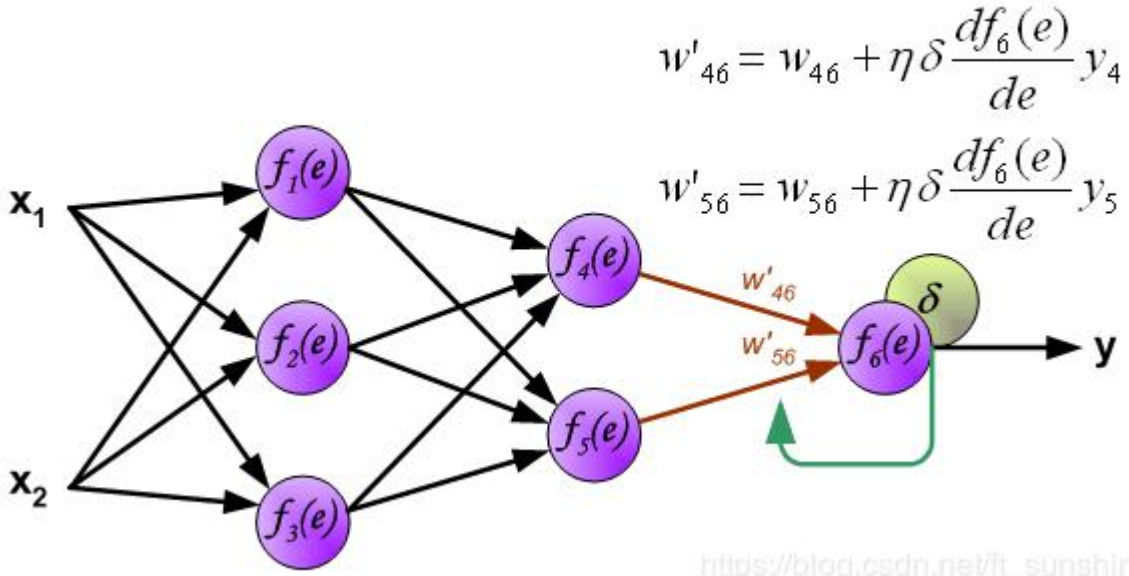
$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$

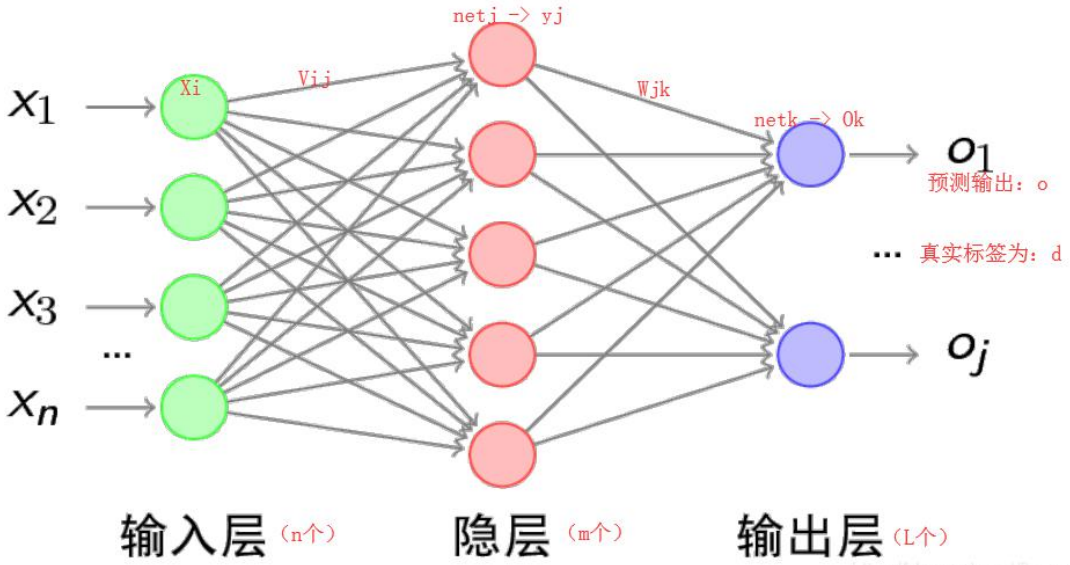


前向传播与反向传播



•到此为止，整个网络的前向，反向传播和权重更新已经完成

前向传播与反向传播



- 输出层

$$E = \frac{1}{2}(d - O)^2 \qquad = \frac{1}{2}\sum_{\kappa=1}^{\ell}(d_k - o_k)^2$$

- 误差展开至隐层

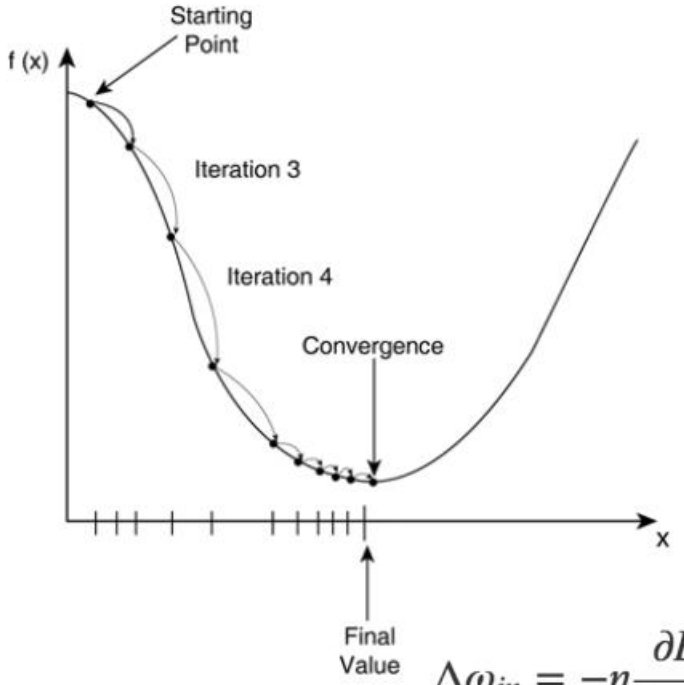
$$E = \frac{1}{2}\sum_{\kappa=1}^{\ell}[d_\kappa - f(net_\kappa)]^2 \qquad = \frac{1}{2}\sum_{\kappa=1}^{\ell}[d_\kappa - f(\sum_{j=0}^{m}\omega_{j\kappa}y_j)]^2$$

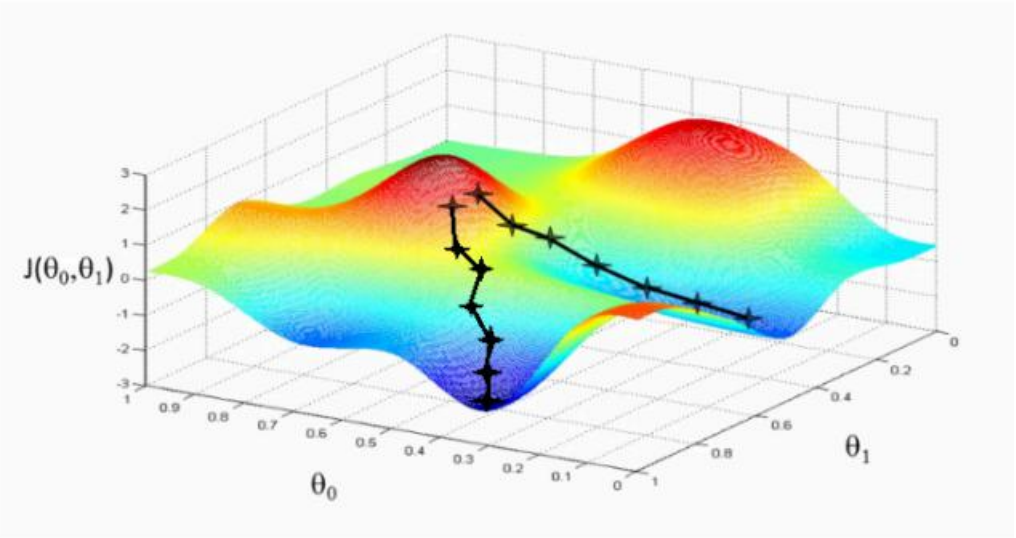
- 展开至输入层

$$E = \frac{1}{2}\sum_{\kappa=1}^{\ell} d_\kappa - f[\sum_{j=0}^{m}\omega_{j\kappa}f(net_j)]^2 \qquad = \frac{1}{2}\sum_{\kappa=1}^{\ell} d_\kappa - f[\sum_{j=0}^{m}\omega_{j\kappa}f(\sum_{i=0}^{n}v_{ij}\chi_i)]^2$$

前向传播与反向传播

- 目标：调整权重使得损失最小 ⇨ 最优化问题

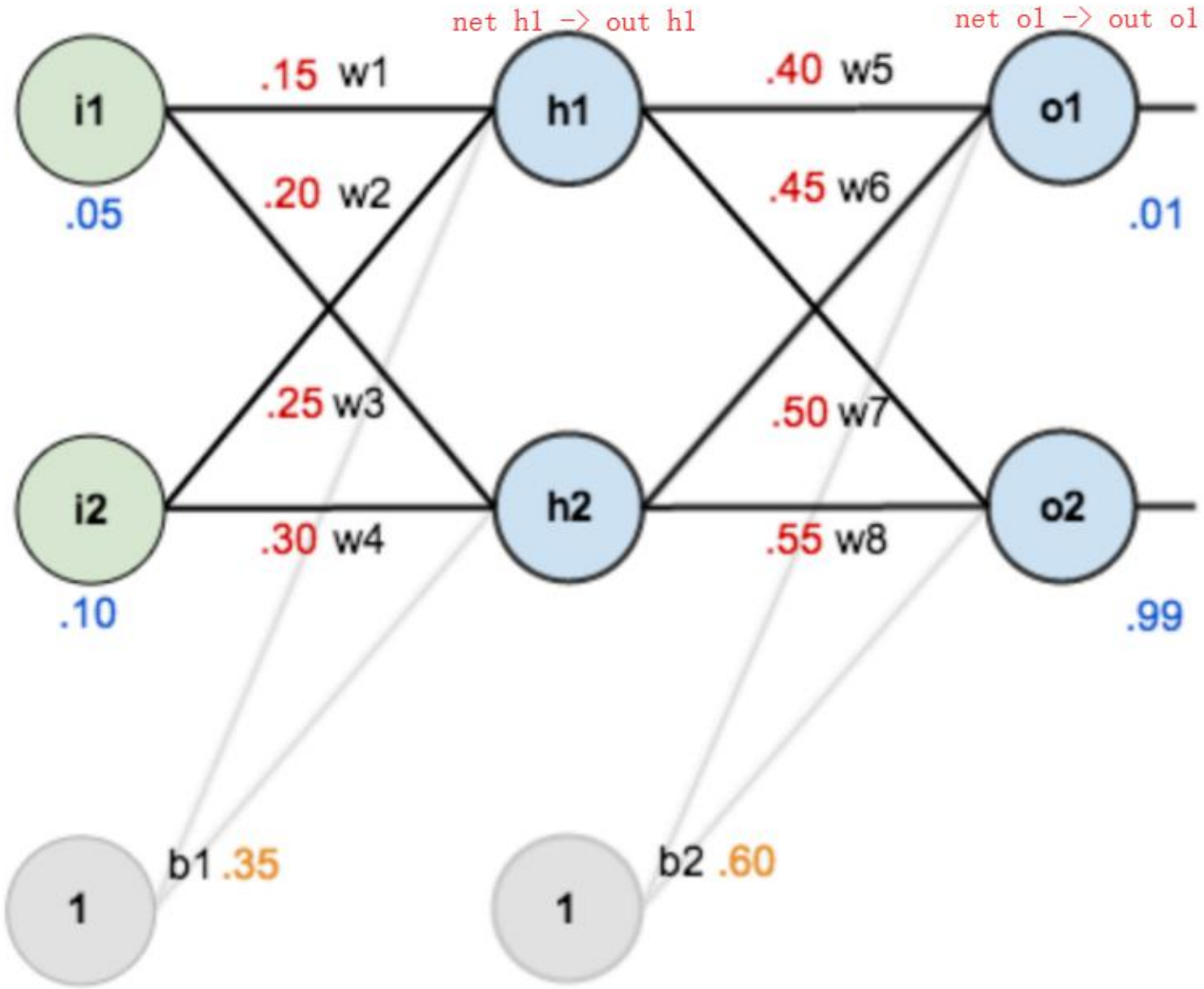




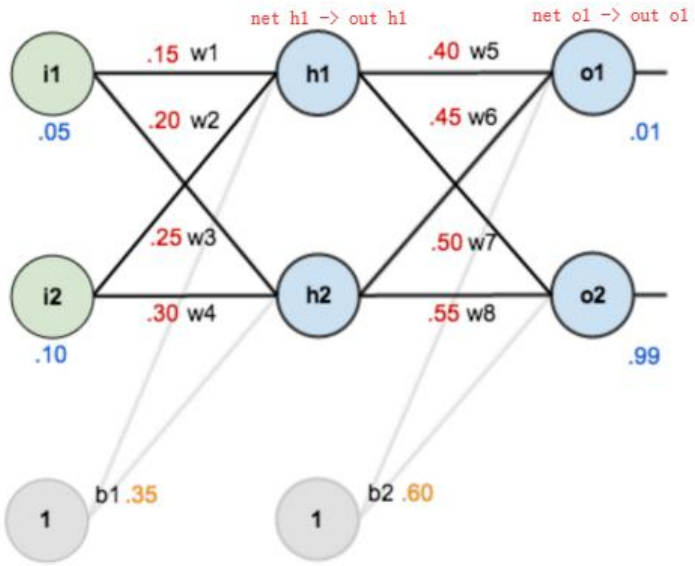
$$\Delta\omega_{j\kappa} = -\eta\frac{\partial E}{\partial\omega_{j\kappa}} j = 0, 1, 2, \ldots, m; \qquad \kappa = 1, 2, \ldots, \ell$$

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

前向传播与反向传播（举例）



前向传播与反向传播（举例）



$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

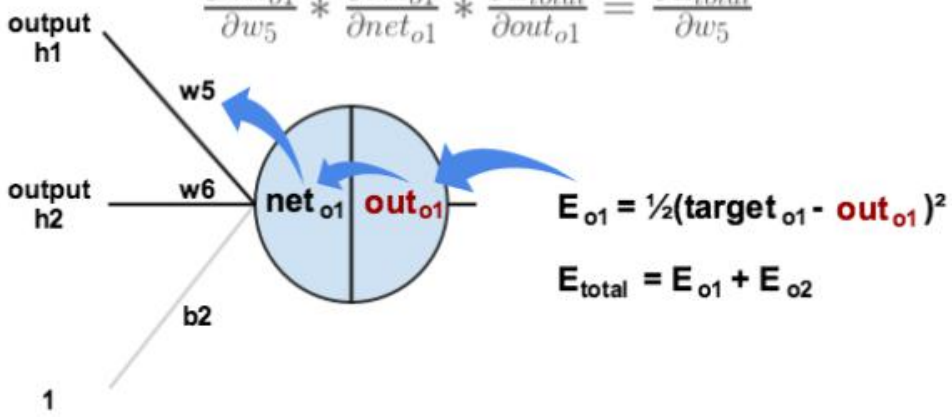
$$out_{o2} = 0.772928465$$

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

前向传播与反向传播（举例）

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

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

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

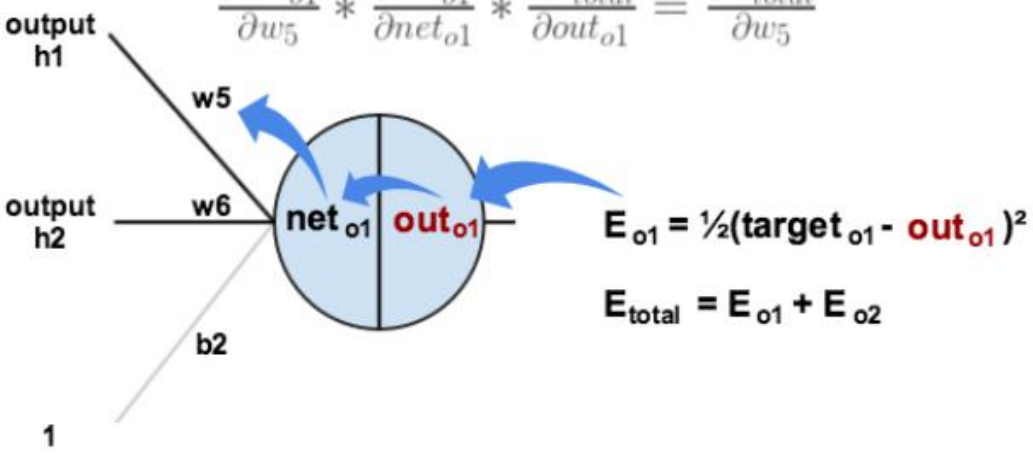
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

前向传播与反向传播（举例）

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

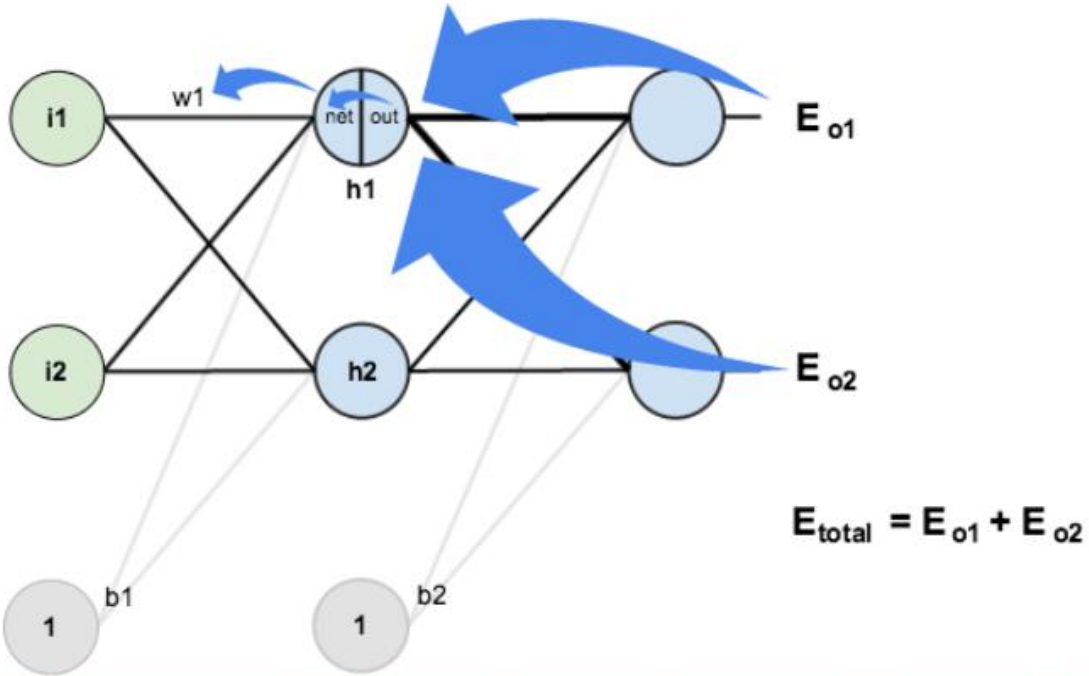
$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

前向传播与反向传播

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

前向传播与反向传播

- 当最初前馈传播时输入为**0.05**和**0.1**，网络上的误差是**0.298371109**
- 在第一轮反向传播之后，总误差现在下降到**0.291027924**
- 重复此过程**10,000**次之后，损失值为**0.000035085**。
  - 两个输出神经元输出为**0.015912196**（相对于目标为**0.01**）和**0.984065734**（相对于目标为**0.99**）

- 问题：
- 链式法则更新权重虽然简单，但过于冗长，是否可以减少不必要的重复计算？

- 问题：
- 链式法则更新权重虽然简单，但过于冗长，是否可以减少不必要的重复计算？

- 已经计算完毕的节点我们完全可以直接拿来用
- 先更新后边的权重，之后再在此基础上利用更新后边的权重产生的中间值来更新较靠前的参数

小结

- 神经元的基本结构
- 人工神经元结构
  - 输入信号
  - 激活函数
  - 几何构型
  - 学习规则