讲师：aopu老师

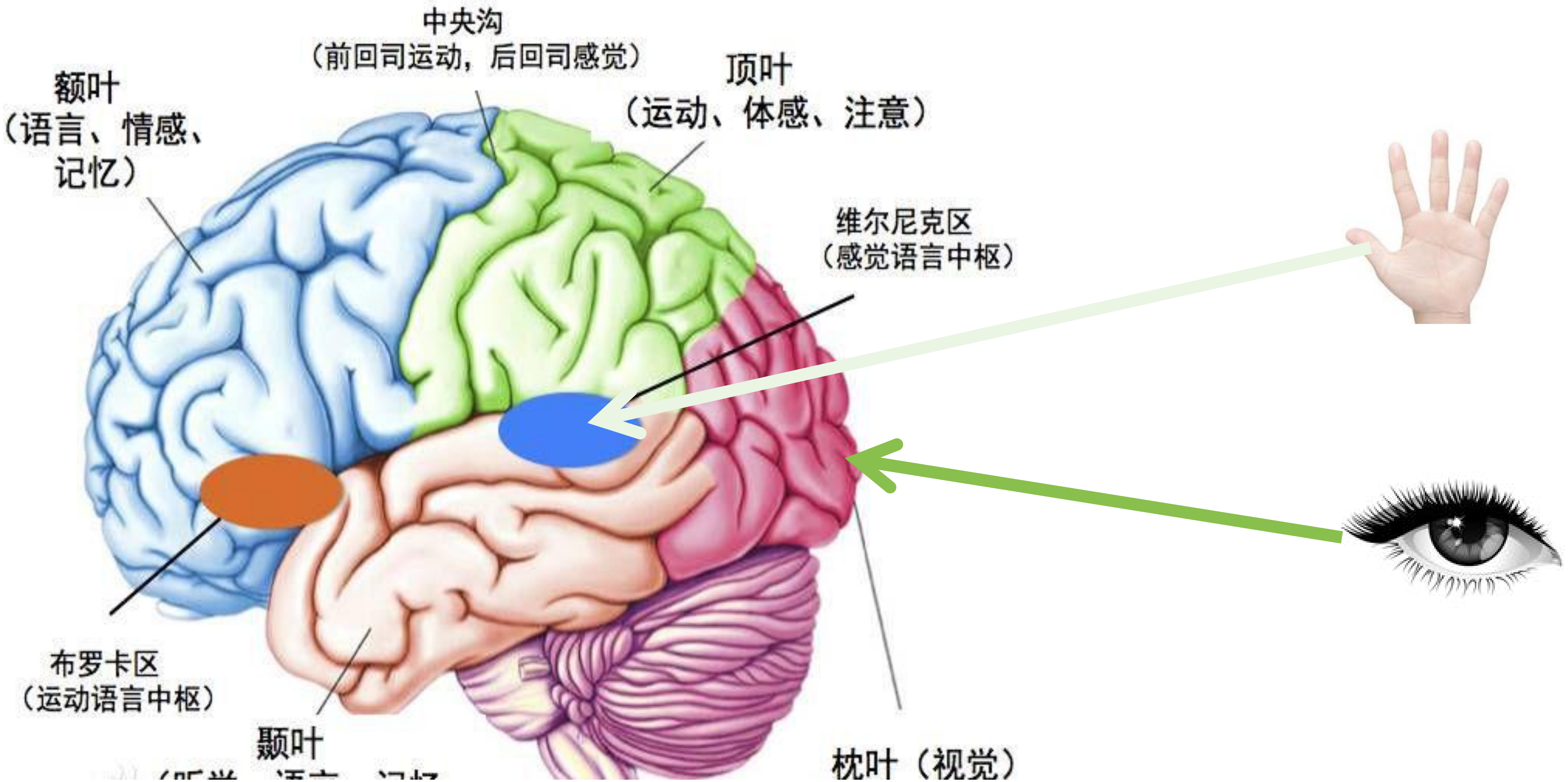
神经网络

目录

神经网络的应用

- 语音识别，声纹识别
- 图像应用
  - 大规模(大数据量)图片识别(聚类/分类)
  - 基于图片的搜索服务
  - 目标检测
- NLP(自然语言)，知识图谱
- 游戏、机器人、推荐系统等
- 数据挖掘(聚类、分类、回归等问题)

神经网络来源之人的思考

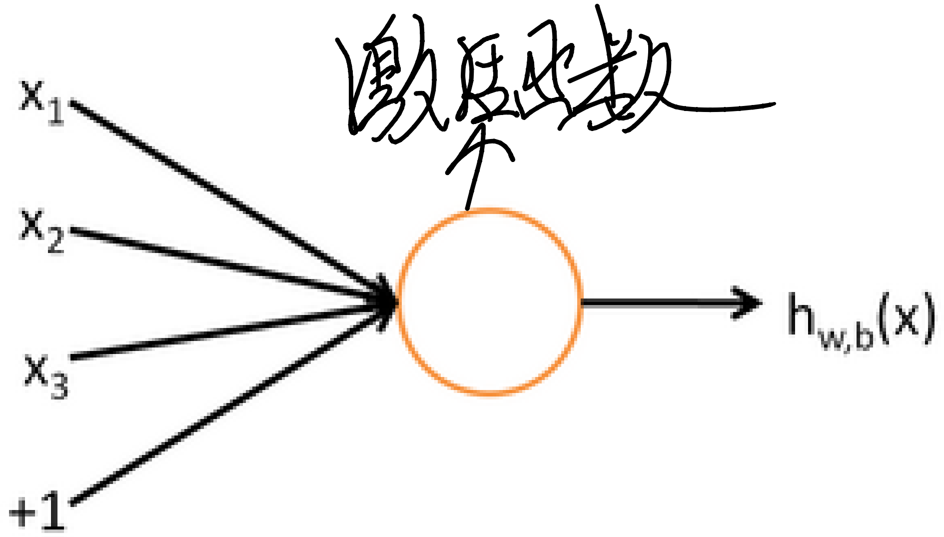


神经网络来源之“神经元”

- 输入：x_1、x_2、x_3和截距+1
- 输出：函数$h_{w,b}(x)$，其中w和b是参数

一般取0

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



- 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)

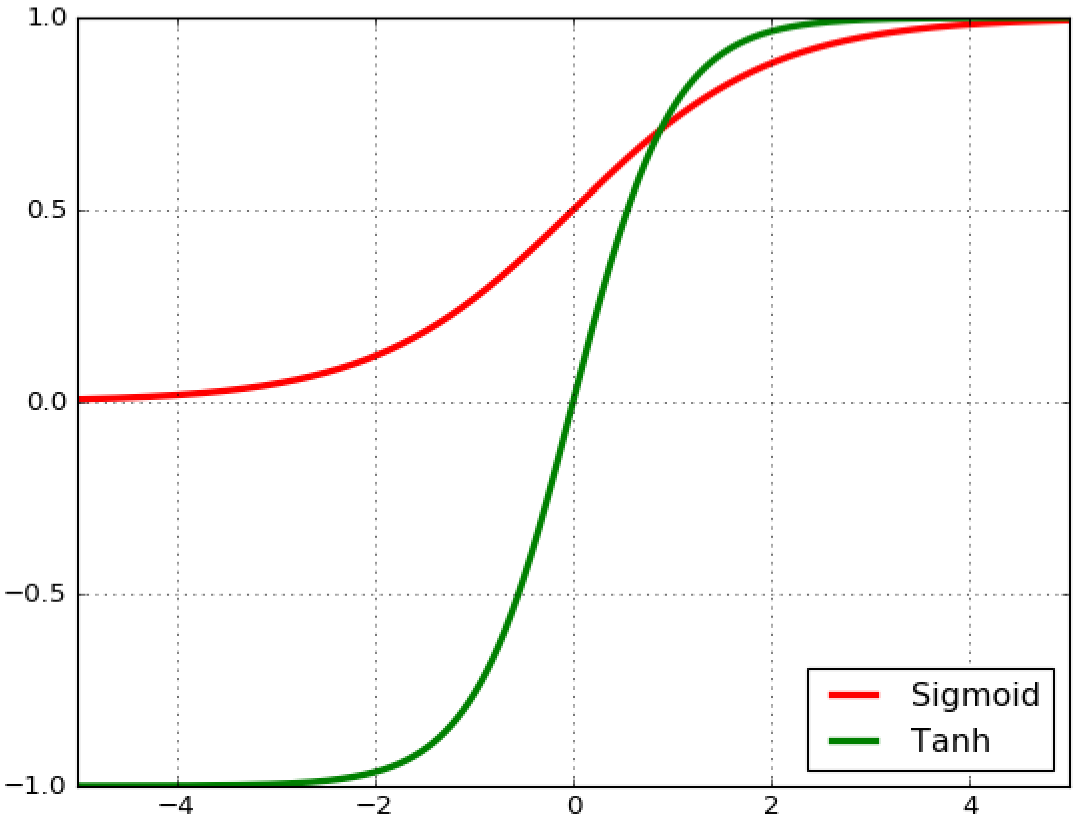
$$\tanh(z)=f(z)=\frac{e^z-e^{-z}}{e^z+e^{-z}};\quad f'(z)=1-\left(f(z)\right)^2$$

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

神经网络来源之“神经元”

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

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



神经网络之感知器

- 当激活函数的返回值是两个固定值的时候，可以称为此时的神经网络为感知器

$$f(z)=\begin{cases}0, z\le 0\\1, z>0\end{cases}\qquad f(z)=\begin{cases}-1, z\le 0\\1, z>0\end{cases}$$

- 因为感知器的返回值只有两种情况，所以感知器只能解决二类线性可分的问题，感知器比较适合应用到模式分类问题中

神经网络之线性神经网络

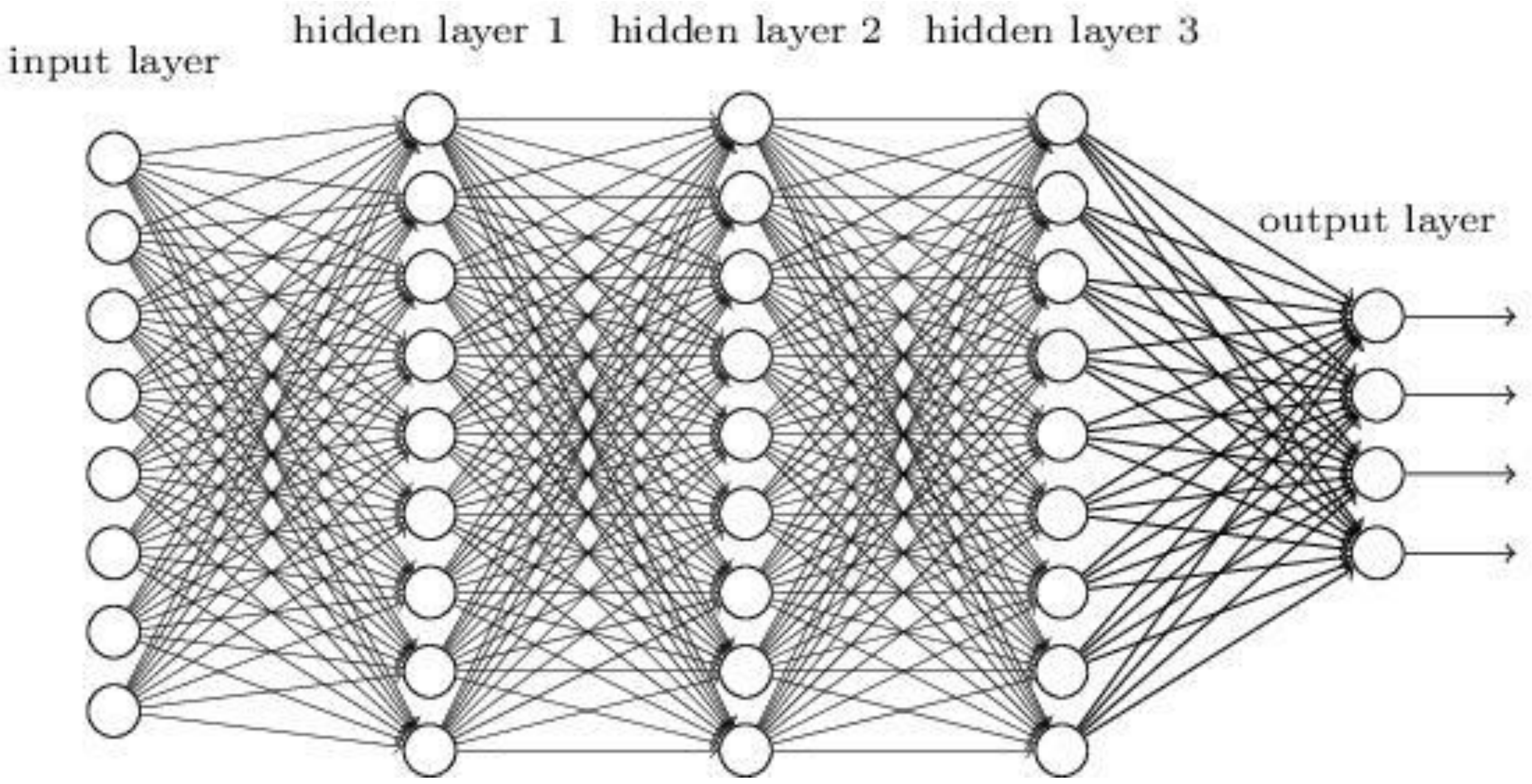
- 线性神经网络是一种简单的神经网络，可以包含多个神经元；激活函数是一个线性函数，可以返回多个值，常用的激活函数为sigmoid函数和tanh函数
- 线性神经网络和感知器一样，只适合线性可分类问题；但是效果比感知器要好，而且可以做多分类问题

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

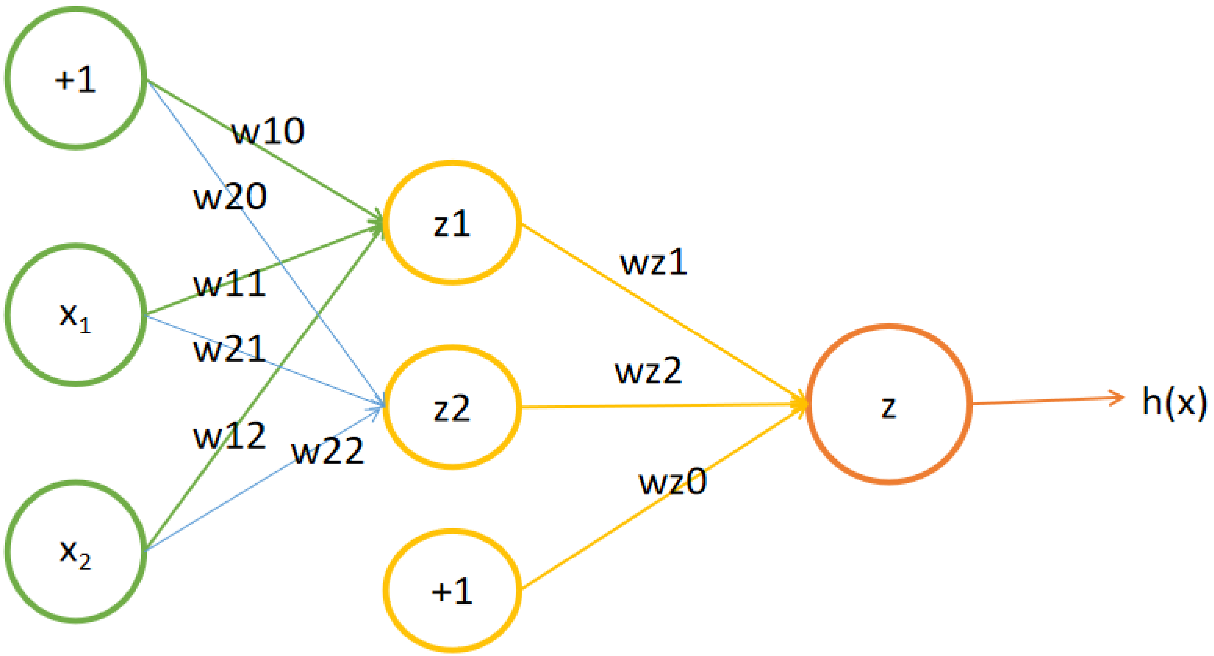
bert (12层神经网络,谷歌提出的网络模型)

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)；可以认为深度学习是神经网络的一个发展



神经网络直观理解之非线性可分

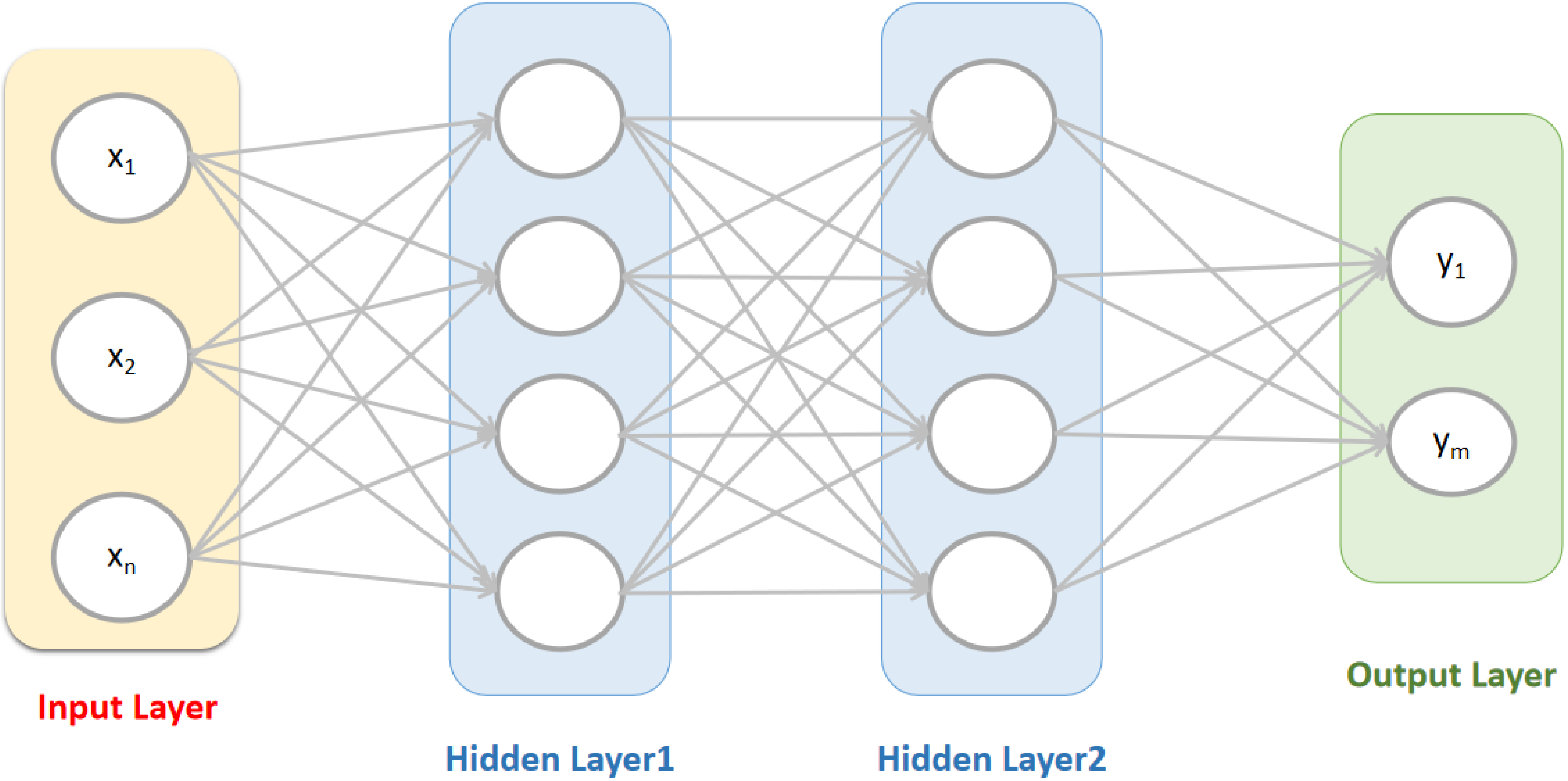


x_1	x_2	z_1	z_2	h(x)
0	0	0	0	1
0	1	0	1	0
1	0	0	1	0
1	1	1	1	1

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

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

神经网络之结构

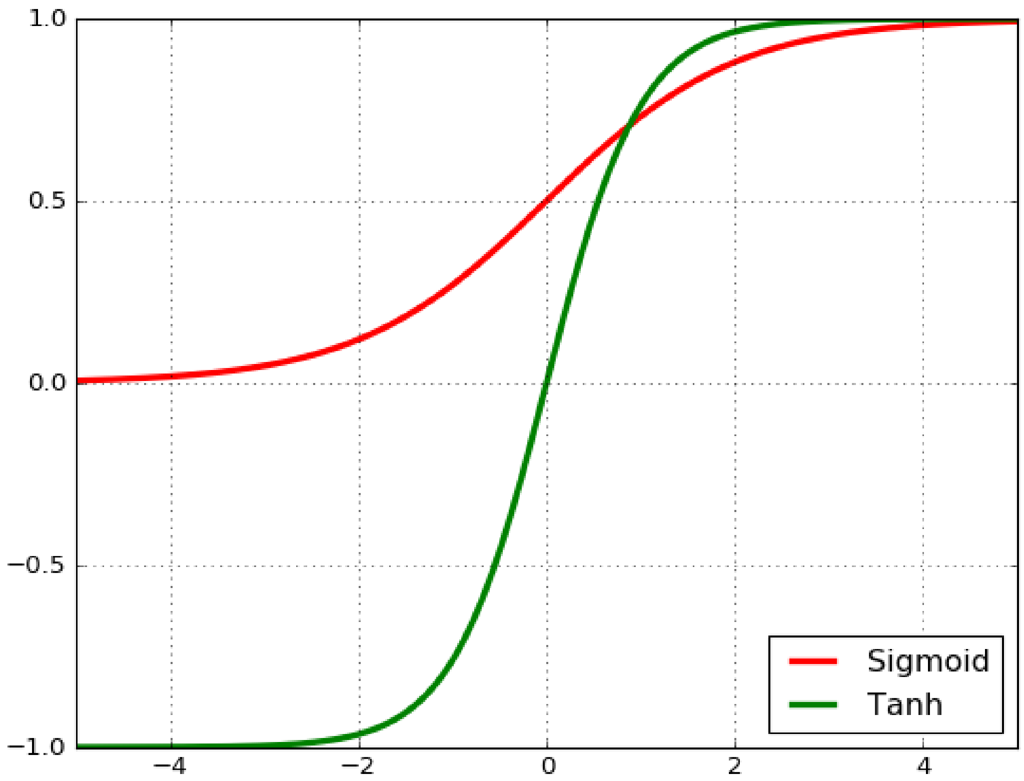


x_1
x_2
x_n
y_1
y_m
Input Layer
Hidden Layer1
Hidden Layer2
Output Layer

神经网络之传递函数(激活函数)

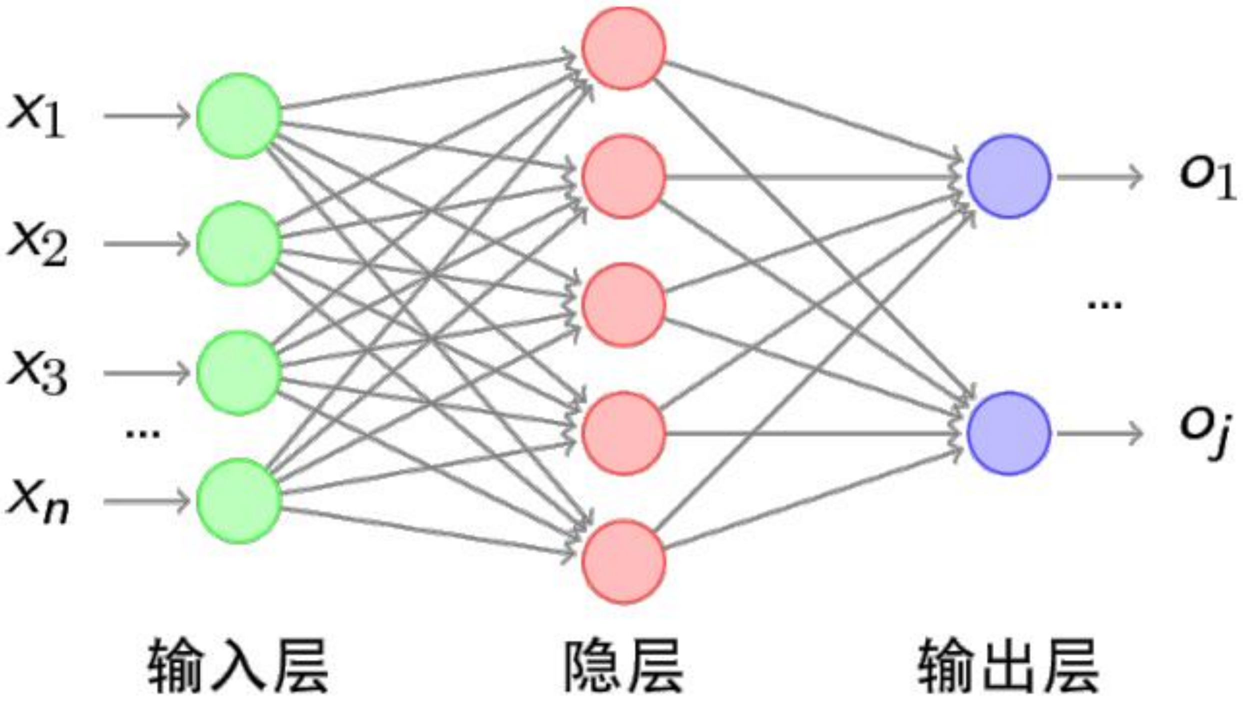
$$S函数: sigmoid(z) = f(z) = \frac{1}{1+e^{-z}}$$

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



神经网络之BP算法 反向误差传递技术

- BP算法也叫做δ算法
- 以三层的感知器为例



线性函数的函数还是线性

神经网络之BP算法

- 输出层误差

预测　实际值

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

- 隐层的误差

预测

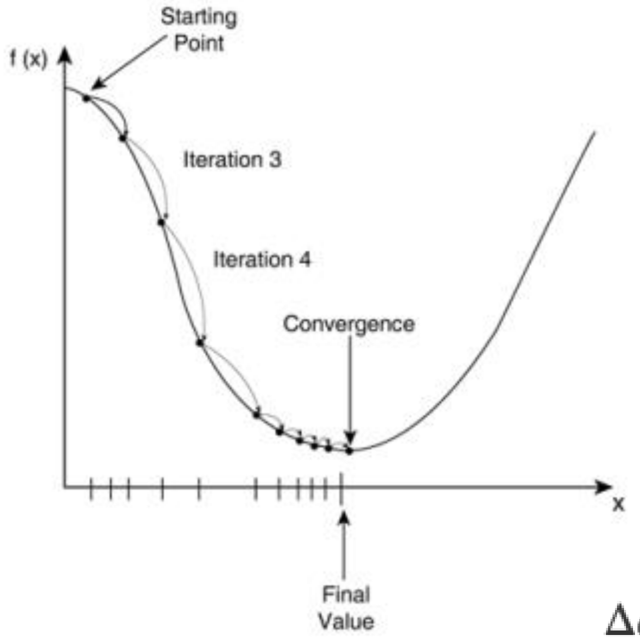
$$E = \frac{1}{2}\sum_{k=1}^{\ell}(d_k - f(net_k))^2 = \frac{1}{2}\sum_{k=1}^{\ell}\left(d_k - f\left(\sum_{j=1}^{m} w_{jk} y_j\right)\right)$$

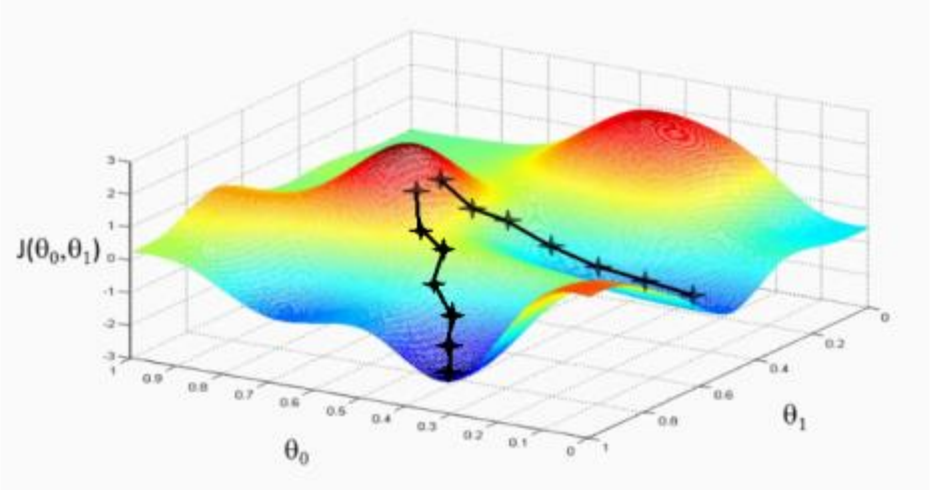
- 输入层误差

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

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ω和υ的求解，即求得ω和υ使得误差E最小



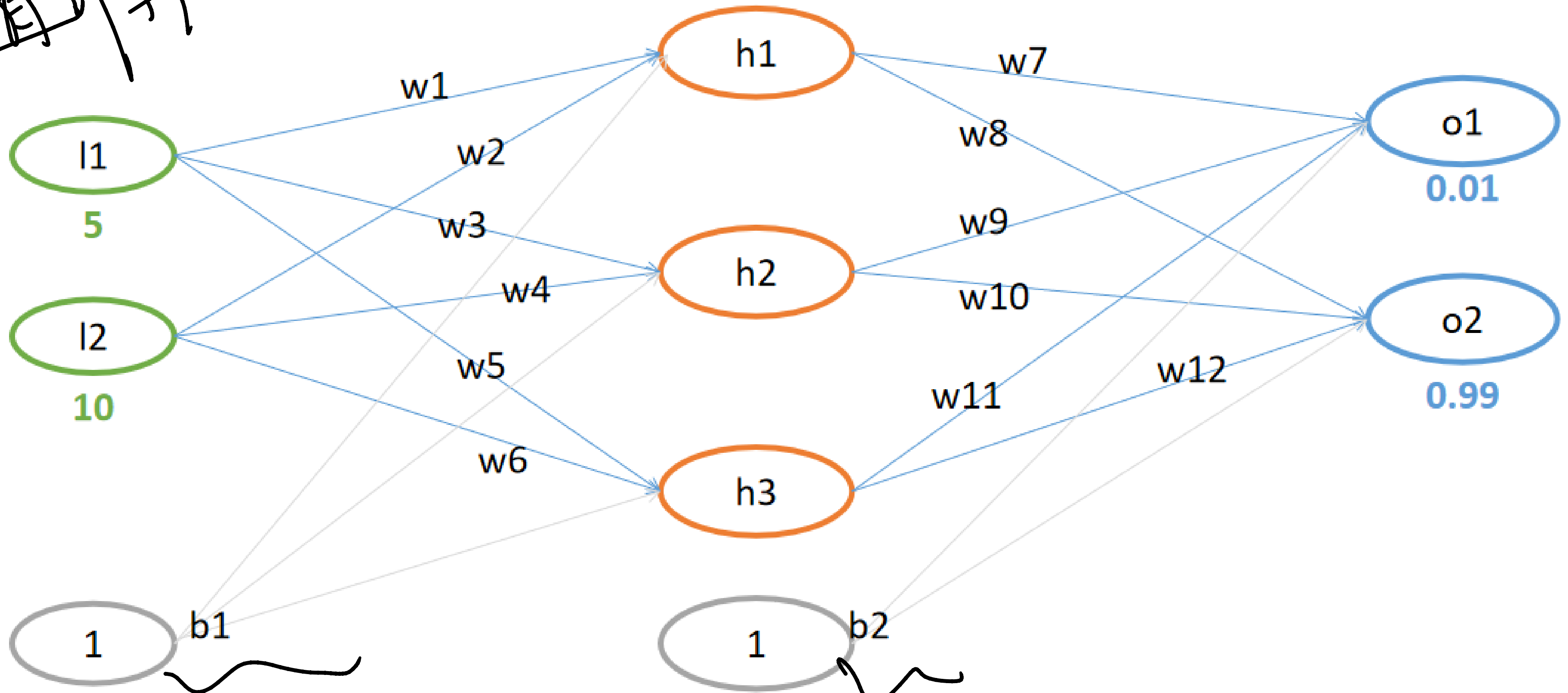


$$\Delta\omega_{j\kappa} = -\eta\frac{\partial E}{\partial\omega_{j\kappa}} j = 0, 1, 2, \ldots, m; \qquad \kappa = 1, 2, \ldots, \ell$$

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

BP算法例子

正向值计算，反向值更新参数



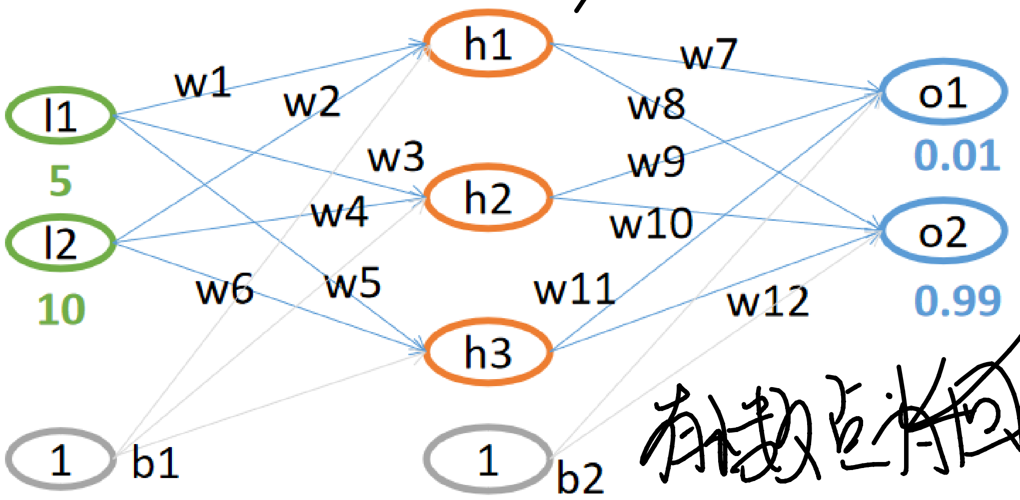
$$w = (0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65)$$

$$b = (0.35, 0.65)$$

BP算法例子-FP过程

每层共用个偏置

是偏置



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

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

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

有数点的回归问题

$$out_{h2} = 0.979164 \quad out_{h2} = 0.995275$$

$$b = (0.35, 0.65)$$

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

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

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

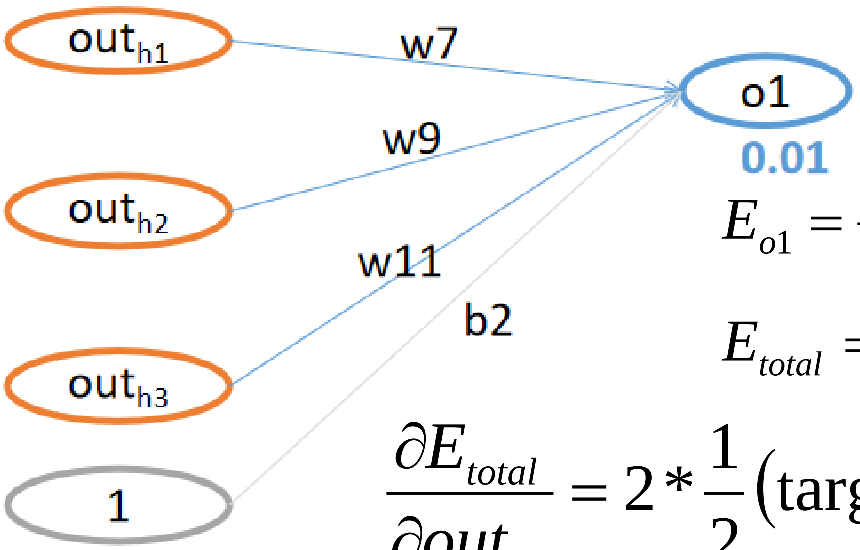
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-2.1019206}} = 0.891090 \qquad out_{o2} = 0.904330$$

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

BP算法例子-FP过程

求极值，求导，方程少，未知数多，求偏导

$$\left(\frac{1}{1+e^{-x}}\right)' = x(1-x)$$



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

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

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

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

激活函数 $out_{o1} = \dfrac{1}{1+e^{-net_{o1}}}$

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

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

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

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

BP算法例子-FP过程

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

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

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

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

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

学习率 一般取 0.001，太大了不稳定
小些较好.

更新

梯度

BP算法例子-FP过程

$w_1^{+} = 0.094534$ $w_2^{+} = 0.139069$ $w_3^{+} = 0.198211$

$w_4^{+} = 0.246422$ $w_5^{+} = 0.299497$ $w_6^{+} = 0.348993$

$w_7^{+} = 0.360968$ $w_8^{+} = 0.453383$ $w_9^{+} = 0.458137$

$w_{10}^{+} = 0.553629$ $w_{11}^{+} = 0.557448$ $w_{12}^{+} = 0.653688$

$$b_1 = 0.35 \qquad b_2 = 0.65$$

BP算法例子-FP过程

①损失函数趋向0 —— 应该停止，否则会越学越坏。

- 第10次迭代结果: $O=(0.662866,0.908195)$

②损失函数不变

- 第100次迭代结果: $O=(0.073889,0.945864)$

一直振荡

- 第1000次迭代结果: $O=(0.022971,0.977675)$

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

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
- 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. 值太大

值太小

THANKS!