

第三讲 人工神经元的学习

本讲主要内容

- 学习概论
- 生物神经元的学习
- 有监督学习实例
- 损失函数
- 感知器学习方法
- 单个神经元应用举例

一、学习概论

为什么要学习

上一讲中的阈值神经元，需要手工设计参数

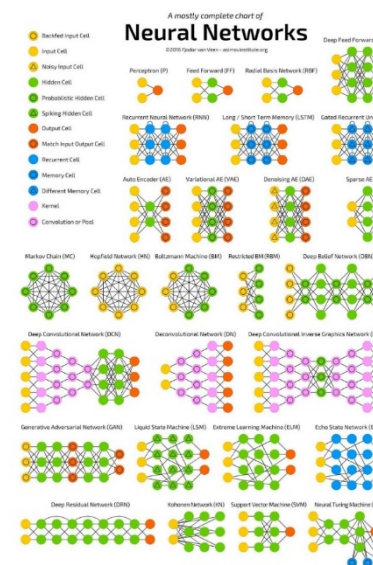
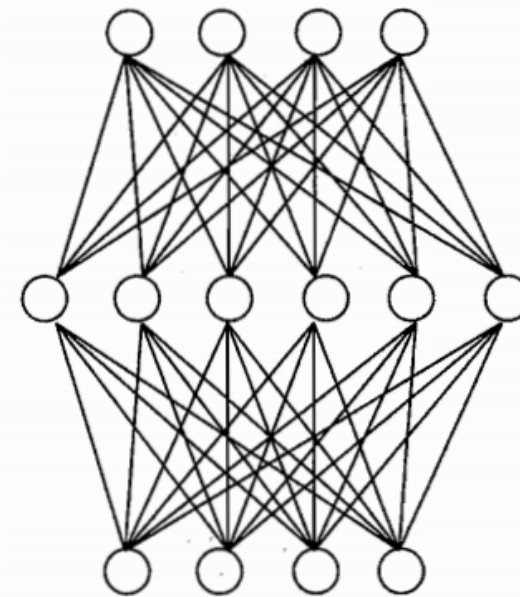
- AND(与) $x_1 \text{ AND } x_2 \rightarrow y$
 - $W = 1, \text{ threshold} = 2$
- OR (或) $x_1 \text{ OR } x_2 \rightarrow y$
 - $W = 2, \text{ threshold} = 2$
- AND NOT (与非) $x_1 \text{ AND NOT } x_2 \rightarrow y$
 - $W = 2, p = 1, \text{ threshold} = 2$

问题

- 复杂问题无法手工设计参数和阈值；
- 如果神经元个数很多，不再适合手工设计各个神经元的参数。



解决方法：可以让神经元自动化
“学习”到参数，无需手工设置；



- 思考：基于对人工神经元的学习，神经元需要自动学习的参数有哪些？

神经网络的学习

- 神经网络最具有吸引力的特点是它的学习能力。
- 1962年，Rosenblatt给出了神经网络著名的学习定理：神经网络可以学会它可以表达的任何东西。
- 神经网络的表达能力大大地限制了它的学习能力。
- 神经网络的学习过程就是对它的训练过程

二、生物神经元怎么学习

学习：是基于经验而使行为或行为潜能发生 相对一致变化的过程

- 基于经验的过程：学习只有通过经验才能发生
 - 行为的哪些方面能通过经验而改变？改变如何发生？
- 行为或行为潜能的变化：学习从你行为表现的进步中显而易见
- 相对一致的变化：行为的变化必须在不同场合表现出相对一致性
- 习惯化和敏感化
 - 习惯化：反应随时间逐渐变弱
 - 敏感化：对重复呈现刺激的反应会变得更强烈
 - 在刺激很强烈或令人不适时更可能发生

学习的两种具体类型

- 经典条件作用
 - 由一个刺激或事件预示另一个刺激或事件的到来
 - 巴浦洛夫的狗——巴浦洛夫条件作用
 - 刺激泛化：反应自动扩展到与刺激相似的新刺激上
- 操作性条件作用
 - 操作：自发产生的行为，可按照他作用于环境并使环境发生了可观察的结果来描述其特点
 - 效果律：带来满意结果的反应出现的概率会越来越大，带来不满意结果的反应出现的概率越来越小
 - 桑代克的迷笼（猫）——指向成功的特定冲动则因愉快的结果而保留下来

本能漂移

- 人和所有动物的学习都可以用单一、普遍的联想主义原则来说明
 - 动物为适应生存需要而进化
 - 学习的生物制约性
- 凯勒.布里兰、马瑞恩.布里兰的动物表演训练实验
 - 运用任何反应类型或奖赏在实验室研究中所得出的一般性原则，都能直接用于实验室以为来控制动物的各种行为
 - 随着时间的推移——习得的行为也会向着本能行为漂移

味觉-厌恶学习

- 提供的食物有种新的味道，让老鼠生病了，老鼠将再不会食用该味道的食物
- 无需反复训练，只要经历一次，味觉厌恶永久保持——one-shot
- 某些物种已经得到进化，使得该物种的成员只需要少于正常的学习经验就能获得条件性反应

观察学习

- 通过替代强化和替代惩罚进行学习的能力
 - 依据他人经验来改变自己的行为
- 观察学习：个体仅仅是在观察到他人的行为被强化或被惩罚后，就会在后来或者做出类似行为，或者抑制该行为
 - 从观察中学习：不必经历冗长的尝试-错误过程，可获得大量的、完整的行为模式
 - 观察学习并非人类特有
 - 榜样的力量：孩子为什么不应该看暴力动画片
- 观察榜样的行为影响力的四个过程
 - 注意、记忆、再现、动机

- 思考：基于生物神经元的学习，对于人工神经元的学习有哪些启发？

人工神经元学习类型

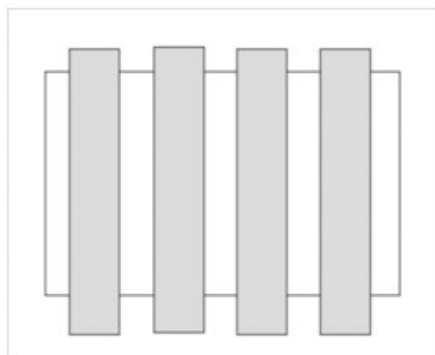
- 无监督学习
- 有监督学习

无监督学习（竞争学习）

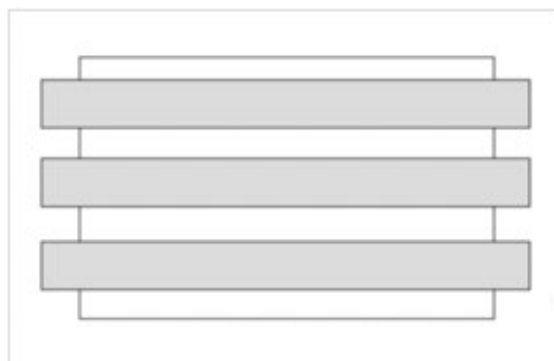
- 是在没有外界作用时对自己进行正确响应的一种网络学习。
- 抽取样本集合中蕴含的统计特性，并以神经元之间的联接权的形式存于网络中。
- 训练样本 $\{X^n\}$

无监督学习

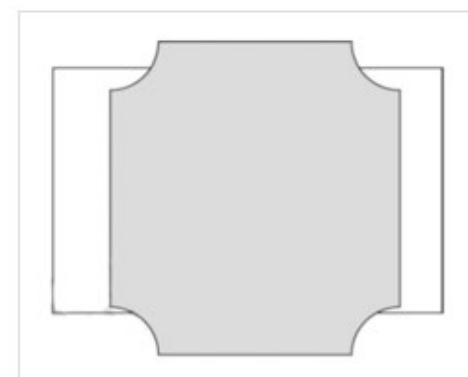
- 基本思想:
- 对给定数据进行某种“压缩”，找到数据的潜在结构



纵向结构
将相似的样本聚到同类



横向结构
高维到低维空间转换



纵向横向结构
数据由含有隐式结构的
概率模型生成

有监督学习

- 对每一个指导学习过程的输入模式使用了外部作用（实际输出）
- 输入向量与其对应的输出向量构成一个“训练对”。
- 训练样本 $\{X^n, Y\}$

有监督学习

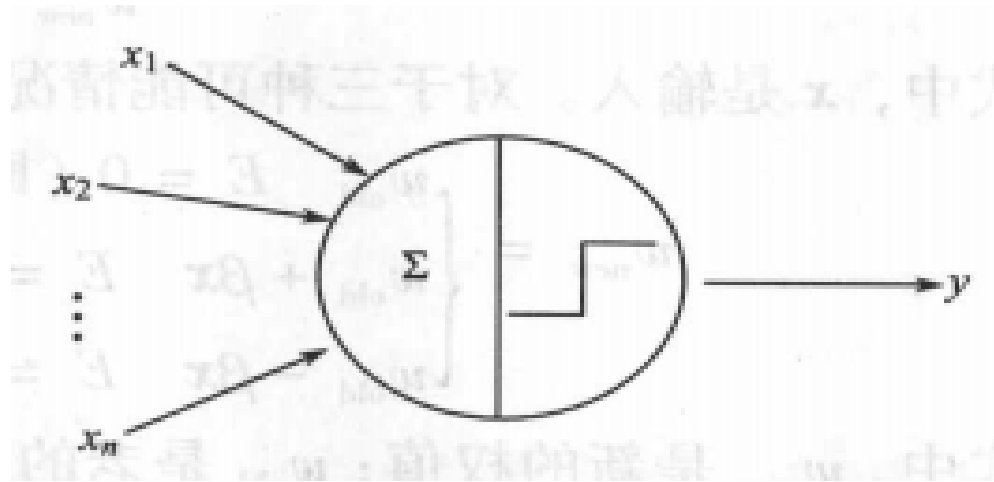
- 监督学习的训练算法的主要步骤包括：
 - 1) 从样本集合中取一个样本 $(\mathbf{X}_i, \mathbf{Y}_i)$
 - 2) 计算出网络的实际输出 \mathbf{O}
 - 3) 计算误差 $\mathbf{D} = \mathbf{Y}_i - \mathbf{O}$
 - 4) 根据 \mathbf{D} 调整权值矩阵 \mathbf{W} 和偏置矩阵 \mathbf{B}
 - 5) 对每个样本重复上述过程，直到对整个样本集来说，所有样本预测正确或误差不超过规定范围。

思考：对比有监督学习和无监督学习，以及它们的适用场景

三、有监督学习实例

作为分类器的有监督学习的感知器

- 对一个没有反馈或竞争的简单响应层神经元进行分析；
- 一个感知器网络从样本数据学习，权值在学习过程中变化；



一个单一神经元的感知器模型

权值变换的方法

- 随机学习
- Hebb学习
- 梯度下降法学习

随机学习

- 思路：随机更新权矩阵 \mathbf{W} 和偏置矩阵 \mathbf{B} 。

那么：

- 在什么范围内给权矩阵 \mathbf{W} 和偏置矩阵 \mathbf{B} 赋值？
- 怎么判断随机给出的 \mathbf{W} 和 \mathbf{B} 是合适的？
- 当判断 \mathbf{W} 和 \mathbf{B} 不合适的时候，如何进行调整？

随机学习

- 解决思路：先给出随机赋值，然后慢慢对赋值进行修正
 - 在什么范围内给权矩阵 \mathbf{W} 和偏置矩阵 \mathbf{B} 赋值？
 - 答：在一个小范围 $(-\delta, +\delta)$ 内给 \mathbf{W} 和 \mathbf{B} 赋予初始值
 - 怎么判断随机给出的 \mathbf{W} 和 \mathbf{B} 是合适的？
 - 答：利用 \mathbf{W} 和 \mathbf{B} 算出输出 \mathbf{O} ，考虑它与真实值 \mathbf{Y} 的误差 $|\mathbf{O} - \mathbf{Y}| < \epsilon$ 是否成立
 - 当判断 \mathbf{W} 和 \mathbf{B} 不合适的时候，如何进行调整？
 - 答：在 \mathbf{W} 和 \mathbf{B} 上加上一个很小的 $(-\xi, +\xi)$ 范围内的随机数，再重新判断

随机学习

- 总结:

尽管随机学习效率很低，但其思想简单，实现容易，且具有能找到全局最优解等特点，在对其搜索方法进行改进后，也能够具有一定的实际应用意义

Hebbian学习

- 一个网络里的信息被储存在神经元之间的**权值**中；
- 假定两个神经元之间的**权值变换**是与它们神经元的输出成比例的；
- 假定随着通过重复和激励一组弱连接神经元而发生学习时，它们之间权值的强度和模式经历逐步增加改变，最终导致神经元形成强连接集合形式。

学习方法：神经元通过调整它们的权值进行学习。

Hebbian学习

- 假设两个神经元有 x 和 y 的输出，如果 x 激励 y ，它们之间的连接强度就会增加。两个神经元之间的权值改变 Δw 与 x 和 y 成比例。

$$\Delta w = \beta x \cdot y$$

- 权值的新值是

$$w_{\text{new}} = w_{\text{old}} + \Delta w = w_{\text{old}} + \beta x \cdot y$$

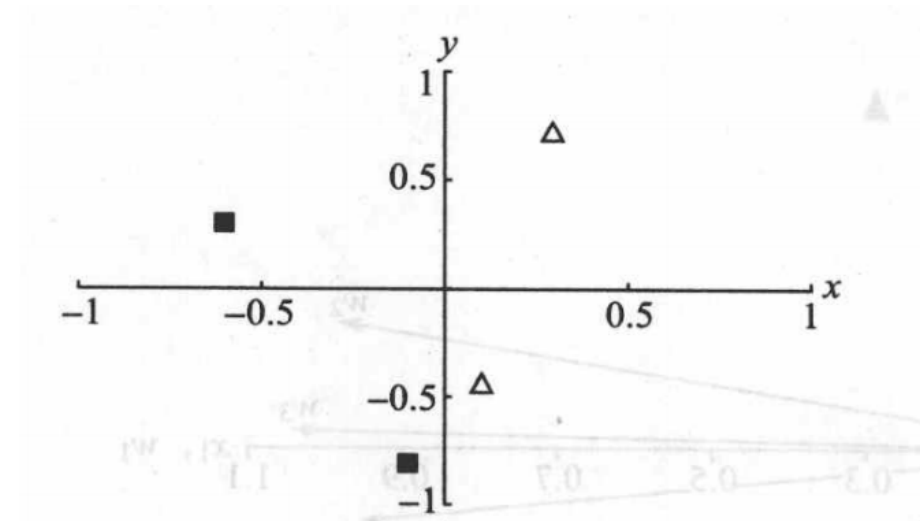
- 比例系数 β 叫做“学习率”，决定学习发生的速度。 β 越大，权值改变得越快。

作为分类器的有监督学习的感知器

训练集合:

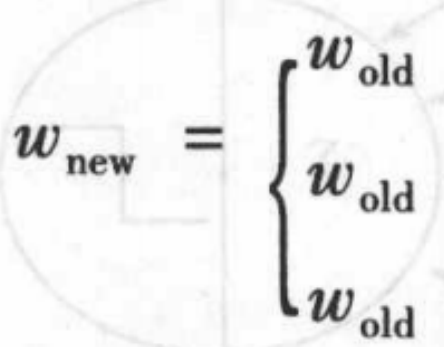
学习过程需要一系列的输入和相关输出，统称为训练集合

X ₁	X ₂	Y
0.3	0.7	1
-0.6	0.3	0
-0.1	-0.8	0
0.1	-0.45	1



作为分类器的有监督学习的感知器

- 网络输入: $u = w_1x_1 + w_2x_2$
- 阈值函数: $y = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0 \end{cases}$ 产生一个输出 y
- 误差: $\text{Error} = E = t - y$ (t 是目标输出)
- 权值的更新: $w_{\text{new}} = w_{\text{old}} + \beta x E$


$$w_{\text{new}} = \begin{cases} w_{\text{old}} & E = 0 \text{ (即 } t = y \text{)} \\ w_{\text{old}} + \beta x & E = 1 \text{ (即 } t = 1, y = 0 \text{)} \text{ (规则 1)} \\ w_{\text{old}} - \beta x & E = -1 \text{ (即 } t = 0, y = 1 \text{)} \text{ (规则 2)} \end{cases}$$

作为分类器的有监督学习的感知器

- 假定 $\beta=0.5$
- 权值随机初始化值: $w_1=0.8, w_2=-0.5$
- 第一轮:
 - $w^0 = \{0.8, -0.5\}$
 - $u = w_1x_1 + w_2x_2 = (0.8)(0.3) + (-0.5)(0.7) = -0.11$
 - $u < 0 \rightarrow y=0$
 - 分类错误, $E=t-y=1$, 利用规则1进行调整:
 - 增量:
$$\Delta w_1^1 = \beta x_1 = (0.5)(0.3) = 0.15$$
$$\Delta w_2^1 = \beta x_2 = (0.5)(0.7) = 0.35$$
 - 新的权值:
$$w_1^1 = w_1^0 + \Delta w_1^1 = 0.8 + 0.15 = 0.95$$
$$w_2^1 = w_2^0 + \Delta w_2^1 = -0.5 + 0.35 = -0.15$$

作为分类器的有监督学习的感知器

- 第二轮：重新应用第一个样本
- $u = (0.3)(0.95) + (0.7)(-0.15) = 0.18 > 0$
- $y = 1 \rightarrow$ 分类正确
- 应用第二个样本
- $U = (-0.6)(0.95) + (0.3)(-0.15) = -0.615$
- $U < 0 \rightarrow y = 0$
- 分类正确，权值不调整

作为分类器的有监督学习的感知器

- 应用第三个样本:
- $U = (-0.1)(0.95) + (-0.8)(-0.15) = 0.025$
- $U > 0 \rightarrow y = 1$
- 分类错误, $E = t - y = -1$, 利用规则2进行调整
- 增量:

$$\Delta w_1^2 = -\beta x_1 = -(0.5)(-0.1) = 0.05$$

$$\Delta w_2^2 = -\beta x_2 = -(0.5)(-0.8) = 0.4$$

- 新的权值:

$$w_1^2 = w_1^1 + \Delta w_1^2 = 0.95 + 0.05 = 1.0$$

$$w_2^2 = w_2^1 + \Delta w_2^2 = -0.15 + 0.4 = 0.25$$

作为分类器的有监督学习的感知器

- 第三轮:
- 重新应用前三个样本，分类正确。
- 应用第四个样本
- $U = (0.1)(1.0) + (-0.45)(0.25) = -0.0125$
- $U < 0 \rightarrow$ 分类错误 利用规则1进行调整

- 增量:

$$\begin{aligned}\Delta w_1^3 &= \beta x_1 = (0.5)(0.1) = 0.05 \\ \Delta w_2^3 &= \beta x_2 = (0.5)(-0.45) = -0.225\end{aligned}$$

- 新的权值:

$$\begin{aligned}w_1^3 &= w_1^2 + \Delta w_1^3 = 1.0 + 0.05 = 1.05 \\ w_2^3 &= w_2^2 + \Delta w_2^3 = 0.25 - 0.225 = 0.025\end{aligned}$$

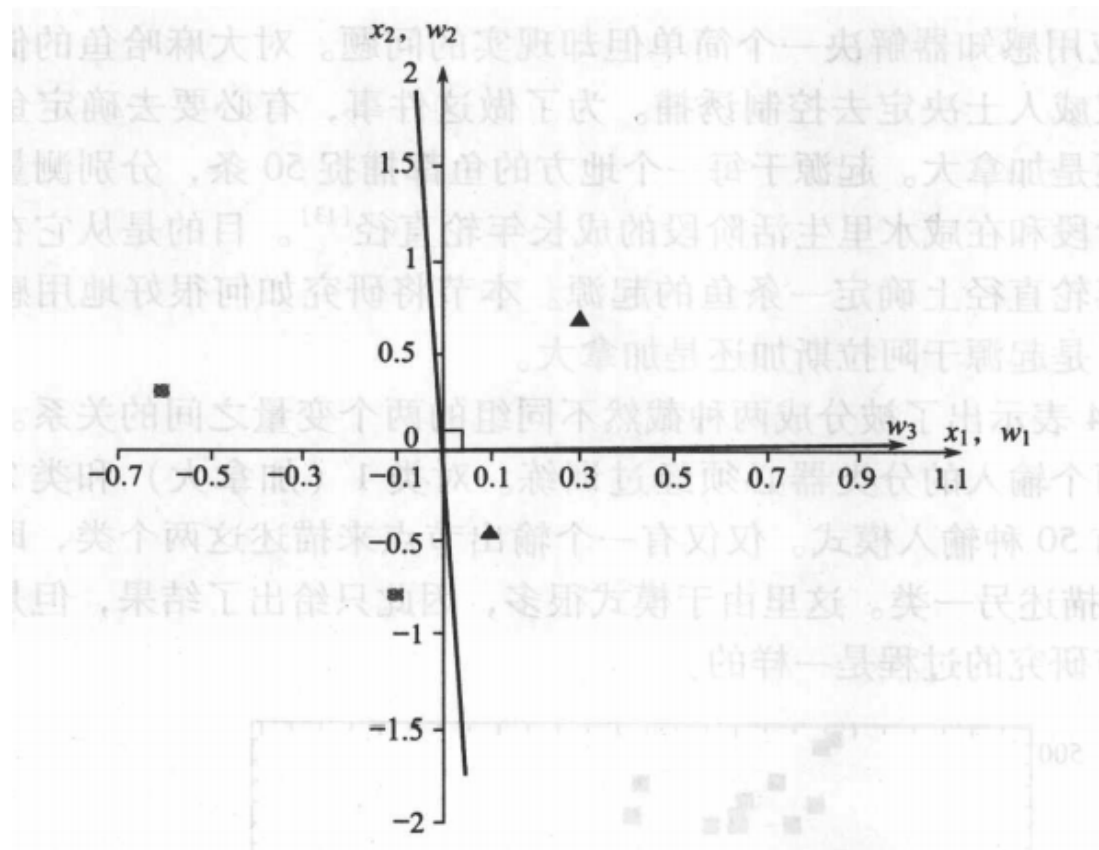
作为分类器的有监督学习的感知器

- 第四轮:
- 将权值 $W=\{1.05, 0.025\}$ 应用到所有四个样本上, 全部正确分类

x_1, x_2	t	u	y	分类精确度
0.3, 0.7	1	$0.3 \times 1.05 + 0.7 \times 0.025 = 0.3325 > 0$	1	正确
-0.6, 0.3	0	$-0.6 \times 1.05 + 0.3 \times 0.025 = -0.6225 < 0$	0	正确
-0.1, -0.8	0	$-0.1 \times 1.05 + (-0.8) \times 0.025 = -0.125 < 0$	0	正确
0.1, -0.45	1	$0.1 \times 1.05 + (-0.45) \times 0.025 = 0.09375 > 0$	1	正确

作为分类器的有监督学习的感知器

- 添加到数据中的权值向量和感知器分类边界



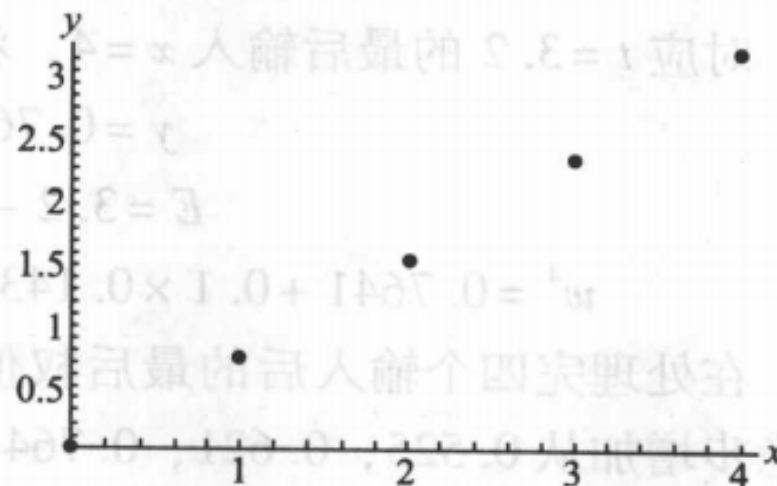
作为分类器的有监督学习的感知器

- 通过上述过程，感知器通过调整权值学习到了最终的权重参数；
- 所有输入数据的一次显示或迭代叫做一次训练时间；
- 只有感知器对所有数据都进行了正确的分类权值才能移动；
- 如果人工设计参数将是十分麻烦的，尤其是参数量和样本数量庞大的场景下。

实例2：线性神经元预报器

- 在一个线性神经元中，输出是连续的，可以是很多值。
- 输出是加权输入的线性求和，所以神经元中的激活函数是线性的。
- 训练集合：

x	t
0	0
1.0	0.75
2.0	1.53
3.0	2.34
4.0	3.2



思考：对于这个问题，如何设计神经元参数和模型

实例2：线性神经元预报器

- 为了简化，因为期望结果是经过原点的函数→忽略偏差。
- 因此神经元模型只有一个参数：

$$y=wx$$

- 误差： $E=t-y=t-wx$
- 权值增量： $\Delta w = \beta Ex$
- $w_{new}=w_{old} + \Delta w$

实例2：线性神经元预报器

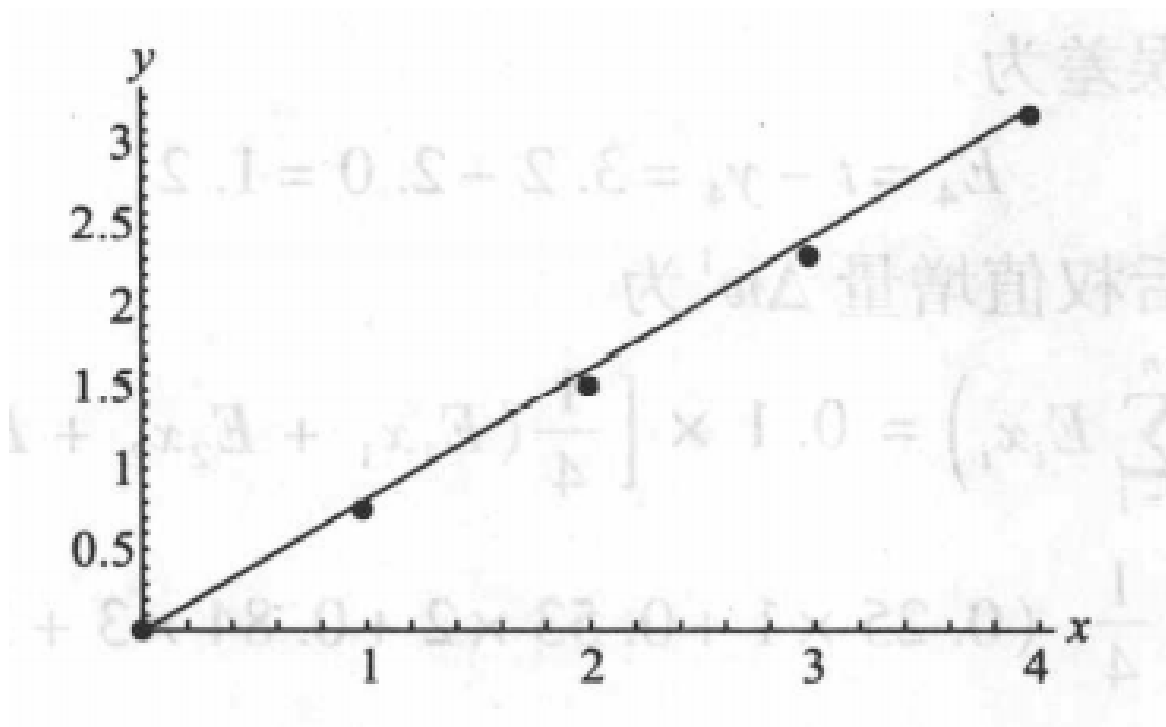
- 初始化：
 - $W=0.5 \quad \beta=0.1$
- 对第一个输入 $x=1$ 的响应 $y = 0.5 * 1 = 0.5$
 - $E = t - y = 0.75 - 0.5 = 0.25$
 - $w^1 = w + \beta E x = 0.5 + 0.1 * 0.25 * 1 = 0.5 + 0.025 = 0.525$
- 对第二个输入 $x = 2, y = 0.525 * 2 = 1.05$
 - $E = t - y = 1.53 - 1.05 = 0.48$
 - $w^2 = 0.525 + 0.1 * 0.48 * 2 = 0.525 + 0.096 = 0.621$

实例2：线性神经元预报器

- 对第三个输入 $x = 3$, $y = 0.621 * 3 = 1.863$
 - $E = t - y = 2.34 - 1.863 = 0.477$
 - $w^3 = 0.621 + 0.1 * 0.477 * 3 = 0.621 + 0.1431 = 0.7641$
- 对第四个输入 $x = 4$, $y = 0.7641 * 4 = 3.0564$
 - $E = t - y = 3.2 - 3.0564 = 0.1436$
 - $w^4 = 0.7641 + 0.1 * 0.1436 * 4 = 0.7641 + 0.05744 = 0.8215$
- 最终, $Y = 0.821 * x$

实例2：线性神经元预报器

- 结果：



思考

- 先前的学习示例中，对每个样本进行单独处理

当样本数目非常多的时候，有没有什么方法对他们进行统一的处理？

四、损失函数

上述问题的解决过程

- 误差： $E = t - y = t - wx$
- 权值增量： $\Delta w = \beta Ex$
- $w_{new} = w_{old} + \Delta w$
- 根据计算误差，更新权重
- 为何这么做，且为何有效？

平方差损失函数

- 平方差损失函数: $L(t, y) = \frac{1}{2} (y - t)^2$
- 其中 $y = w^T x$, 这里将偏置 b 纳入 w ,
- $w = (b, w_1, w_2, \dots, w_n)^T, x = (1, x_1, x_2, \dots, x_n)^T$.

利用平方差损失函数进行学习

- 平方差损失函数: $L(t, y) = \frac{1}{2}(y - t)^2$
- 其中 $y = w^T x$, 这里将偏置 b 纳入 w ,
- $w = (b, w_1, w_2, \dots, w_n)^T, x = (1, x_1, x_2, \dots, x_n)^T$.
- 对于训练数据集 D , 学习 w 权重的过程, 是最小化平方差损失函数 L 。
- 利用最小二乘法在数据集 D 上求解 w 。
- 之前学习方法则尝试利用一个样本来更新 w 权重。
- 其本质是利用单个样本在平方差损失函数上的梯度信息, 进行更新。

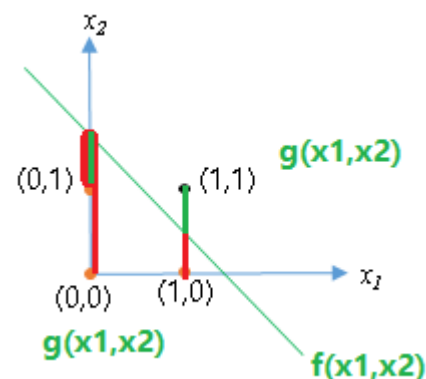
五、感知器学习方法

感知机学习算法

- 考虑最简单的感知机结构 $f(x) = \text{sign}(w * x + b)$
 - 输入 $x \in X$ 表示实例的特征向量，对应于输入空间（特征空间）的点，输出 $y \in Y$ 表示实例的类别，由输入空间到输出空间的函数：
 - 符号函数：
$$\text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$
-

感知机学习算法

- 我们给定输入维度为2（二维平面），希望用单个神经元表示逻辑门中的与门
- 与门的表示：
 $g(x_1, x_2): g(1,0) = g(0,0) = g(0,1) = 0, g(1,1) = 1$



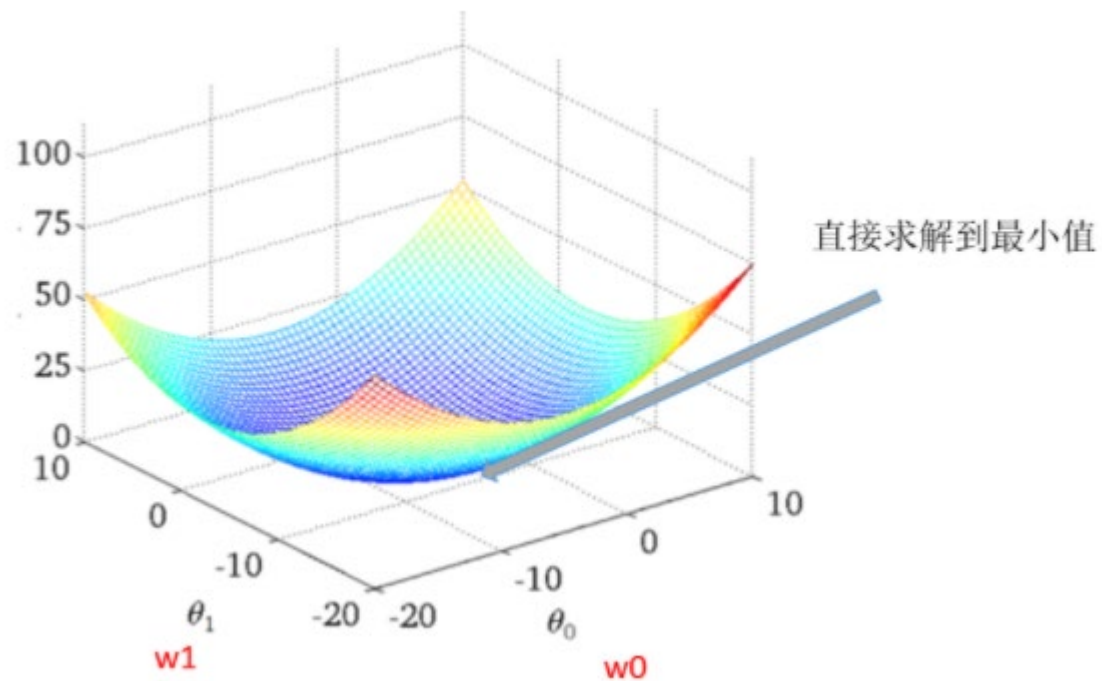
感知机学习算法

- 优化函数表示为

$$\begin{aligned} l(w_1, w_2, b) &= (f(x_1, x_2) - g(x_1, x_2))^2 \\ &= (\text{sign}(w_1 * x_1 + w_2 * x_2 + b) - g(x_1, x_2))^2 \end{aligned}$$

感知机学习算法

- 我们可视化优化函数后，我们的目的是达到这个“碗”的中心（最小值）



https://blog.csdn.net/tq_358661

感知机学习算法

- 求全局最小的常用方法：梯度下降法。
- 梯度下降的基本过程就和下山的场景很类似。
- 如何最快地下山？
 - 从最陡峭的地方向下
- 函数的梯度就确定了这个“最陡峭的方向”

梯度下降法

- 梯度的方向就是函数之变化最快的方向。
- 看待微分的意义，可以有不同的角度，最常用的两种是：

- 函数图像中，某点的切线的斜率

- 函数的变化率

几个微分的例子：

- 单变量的微分，函数只有一个变量时

$$\frac{d(x^2)}{dx} = 2x$$

$$\frac{d(-2y^5)}{dy} = -10y^4$$

$$\frac{d(5-\theta)^2}{d\theta} = -2(5-\theta)$$

梯度下降法

- 多变量的微分，当函数有多个变量的时候，即分别对每个变量进行求微分输出：

$$\frac{\partial}{\partial x}(x^2y^2) = 2xy^2$$

$$\frac{\partial}{\partial y}(-2y^5 + z^2) = -10y^4$$

$$\frac{\partial}{\partial \theta_2}(5\theta_1 + 2\theta_2 - 12\theta_3) = 2$$

$$\frac{\partial}{\partial \theta_2}(0.55 - (5\theta_1 + 2\theta_2 - 12\theta_3)) = -2$$

梯度下降法

- 梯度实际上就是多变量微分的一般化。对每个变量进行微分，然后用 $\langle \rangle$ 包括起来，表明梯度为一个向量。
- 举例说明：

$$\begin{aligned} J(\Theta) &= 0.55 - (5\theta_1 + 2\theta_2 - 12\theta_3) \\ \nabla J(\Theta) &= \left\langle \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \frac{\partial J}{\partial \theta_3} \right\rangle \\ &= \langle -5, -2, 12 \rangle \end{aligned}$$

梯度下降法

- 在单变量的函数中，梯度其实就是函数的微分，代表着函数在某个给定点的切线的斜率
- 在多变量函数中，梯度是一个向量，向量有方向，梯度的方向就指出了函数在给定点的上升最快的方向
- 梯度的方向实际就是函数在此点上升最快的方向，而我们需要朝着下降最快的方向走，自然就是负的梯度的方向，所以梯度下降法需要加上负号

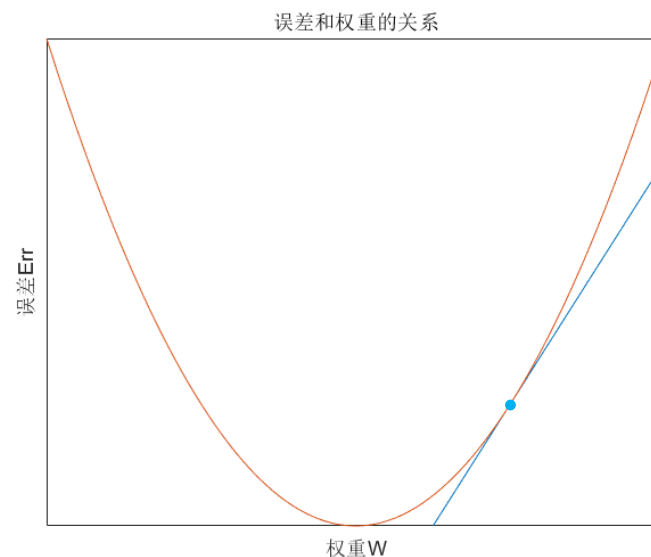
思考：如何向负梯度的方向走？怎么走？每次走多长？

梯度下降法

- 新的误差计量方式:

$$Err = \frac{1}{2}(Y_i - O)^2$$

- 梯度下降法遵循两个原则:
 - 每次更新权矩阵 W , 都应该使得误差 Err 下降, 或者说使得权矩阵 W 向最优解靠近, 如果误差无法下降了, 就终止对于权重的更新;
 - 希望能够以最快的速度找到最优解。
- 满足上述两个原则的权重调整方式: 误差 Err 沿着梯度方向下降最快, 求取误差的梯度来使得误差下降最快; 同时我不能直接下降到梯度, 应该给出学习率 α 使得梯度跨度不能太大



梯度下降法

- 我们可以推导出误差 Err 对于权矩阵 \mathbf{W} 的梯度:

$$\begin{aligned}\frac{\partial Err}{\partial \mathbf{W}} &= \frac{\partial (\frac{1}{2}(\mathbf{Y}_i - \mathbf{O})^2)}{\partial \mathbf{W}} = (\mathbf{Y}_i - \mathbf{O}) \frac{\partial (\mathbf{Y}_i - \mathbf{O})}{\partial \mathbf{W}} = (\mathbf{Y}_i - \mathbf{O}) \frac{\partial (\mathbf{Y}_i - (\mathbf{W}\mathbf{X}_i + \mathbf{B}))}{\partial \mathbf{W}} \\ &= (\mathbf{Y}_i - \mathbf{O})(-\mathbf{X}_i)\end{aligned}$$

- 权矩阵 \mathbf{W} 的更新应该沿着梯度相反的方向进行

$$\Delta \mathbf{W} = \alpha (\mathbf{Y}_i - \mathbf{O}) \mathbf{X}_i$$

- 同理，对于偏置矩阵 \mathbf{B} 也可以做同样的分析，类似的，得到:

$$\begin{aligned}\frac{\partial Err}{\partial \mathbf{B}} &= -(\mathbf{Y}_i - \mathbf{O}) \\ \Delta \mathbf{B} &= \alpha (\mathbf{Y}_i - \mathbf{O})\end{aligned}$$

更一般的写法: $\Delta W_{ij}(t) = g(o_i(t), y_j, x_j(t), W_{ij}(t))$

学习率

- 在梯度下降法更新时，我们常常在计算的负梯度前乘以一个常数 α
- α 在梯度下降算法中被称作为学习率或者步长，意味着我们可以通过 α 来控制每一步走的距离。
 - α 太小可能导致迟迟走不到最低点或者无法跳出局部极小点
 - α 太大的话会导致错过最低点，无法稳定收敛。

学习率调整相关的优化算法：学习率是神经网络优化时的重要超参数。在梯度下降方法中，学习率 η 的取值非常关键，如果过大就不会收敛，如果过小则收敛速度太慢。

学习率调整方法：

- (1) 学习率衰减
- (2) 学习率预热
- (3) 周期学习率
- (4) 自适应调整学习率：AdaGrad, RMSprop, AdaDelta

学习率衰减

- 通常在一开始我们会把学习率设置的大一些来保证收敛速度
- 当收敛到最优点附近时,应该让学习率变小一些避免震荡
- 这种学习率调整的方式可以通过学习率调整 (Learning Rate Decay) 来实现,也称为学习率退火 (Learning Rate Annealing) 。
- 假设初始的学习率为 δ_0 , 在 t 次迭代时的学习率 δ_t , 常见的衰减方式有以下几种:
 - (1) 分段常数衰减
 - (2) 逆时衰减
 - (3) 指数衰减
 - (4) 自然指数衰减

分段常数衰减

- 分段常数衰减：我们可以定义每经过 T_1, T_2, \dots, T_n 次迭代，学习率将调整为 $\delta_1, \delta_2, \dots, \delta_n$ ，其中每个 δ_i 和 T_i 根据经验设置，并且 δ_1 到 δ_n 应该是逐渐减小的。

- 逆时衰减：
$$\eta_t = \eta_0 \frac{1}{1 + \beta \times t}$$

- 指数衰减：
$$\eta_t = \eta_0 \beta^t$$

- 自然指数衰减：
$$\eta_t = \eta_0 \exp(-\beta \times t)$$

- 余弦衰减：
$$\eta_t = \frac{1}{2} \eta_0 (1 + \cos(\frac{t\pi}{T}))$$

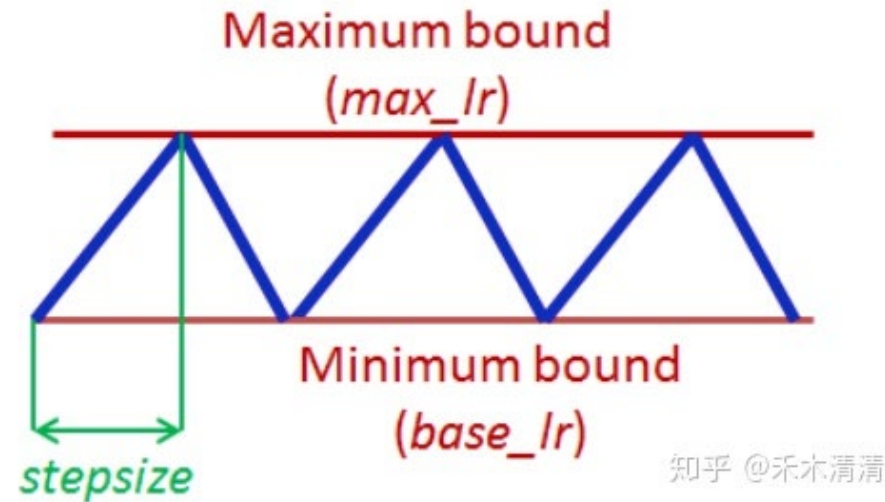
学习率预热

- 在小批量梯度下降方法中，当批量大小设置的比较大时，通常需要较大的学习率，
- 较大的学习率可能带来模型的不稳定，所以这两方面就发生了矛盾。
- 为了让初始阶段的学习更加稳定：
 - 在初始的几轮，采用较小的学习率
 - 梯度下降到一定程度之后，再恢复到初始设置的学习率
- 逐渐预热（Gradual Warmup）

$$\eta_t = \frac{t}{T} \eta_0 \quad 1 \leq t \leq T.$$

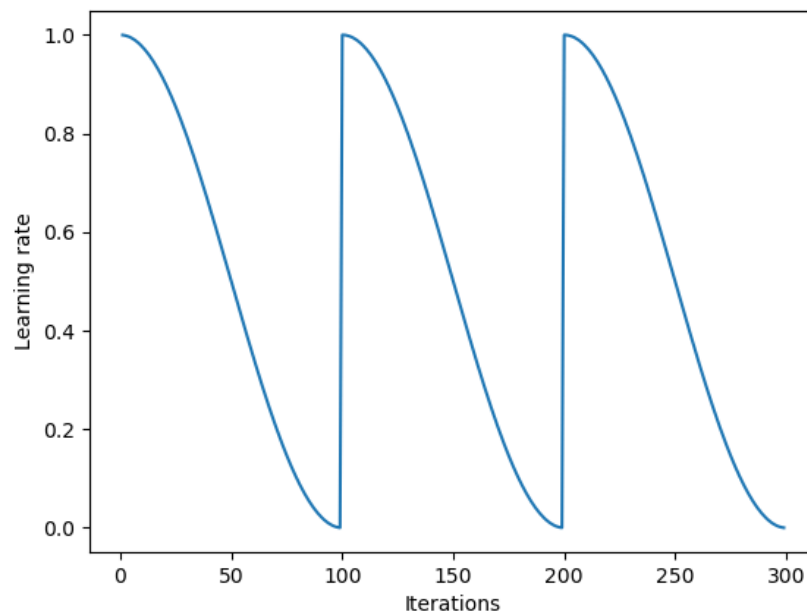
周期性学习率调整

- 循环学习率：循环学习率 (Cyclic) 是让学习率在一个区间内周期性的增大和缩小



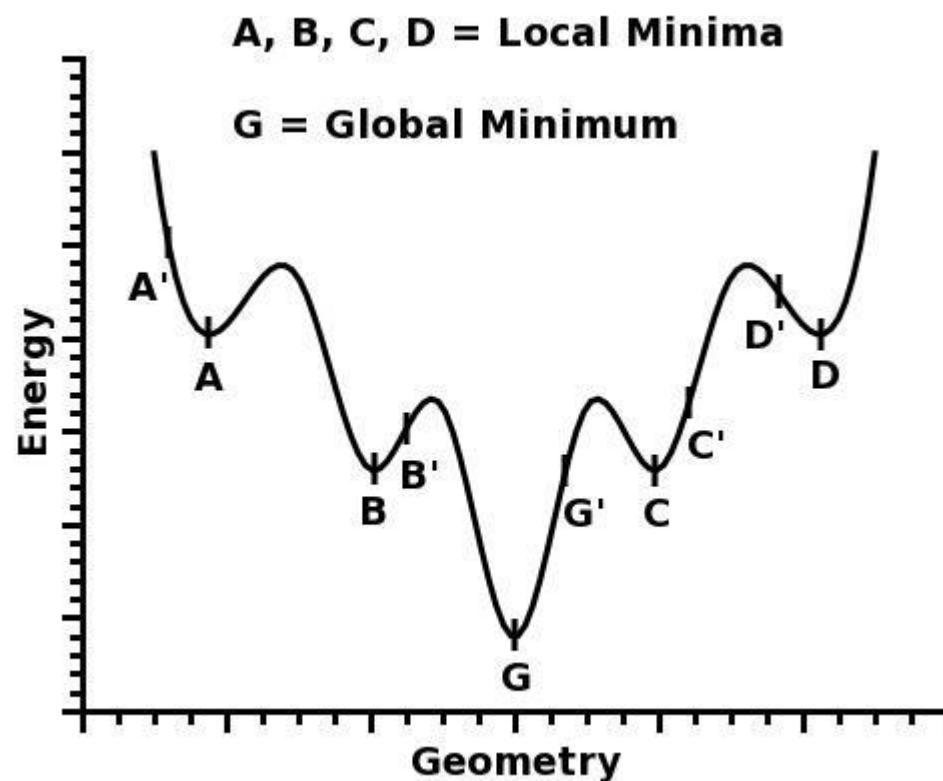
周期性学习率调整

- 带热重启的随机梯度下降：学习率会每间隔一定迭代后重新初始化为预先设定好的值，并继续衰减。



- 每100轮迭代后，学习率会重启

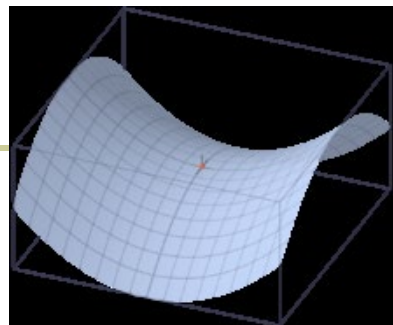
- 思考题：如果函数有多个局部极小，如何找到全局最小



- 尝试“跳出”局部最小：
- 模拟退火技术，每一步以一定概率接受比当前解更差的结果；
- 从多个初始点开始搜索极小值；

梯度下降面临的困难

- 很难选择一个合适学习率
- 对于所有的参数，均使用相同的学习率。不同参数的梯度大小有差异。
- 在非凸函数的优化过程中，我们往往希望模型能够跳过那些局部极值点，去找一个更好的极值。实际问题中，鞍点问题很难解决。



感知机学习算法

- 我们已经分析了单步内权重和偏置的更新，但也带来一些问题：
 - 如何保证迭代过程的收敛？不同初始状态是否影响收敛？
 - 线性数据集合线性不可分数据集的收敛过程是否存在不同？

感知机学习算法

- 算法的收敛性：证明经过有限次迭代可以得到一个将训练数据集完全正确划分的分离超平面及感知机模型。

- 将 b 并入权重向量 w ，记作： $\tilde{w} = (w^T, b)^T$

$$\tilde{x} = (x^T, 1)^T \quad \tilde{x} \in R^{n+1} \quad \tilde{w} \in R^{n+1} \quad \tilde{w} * \tilde{x} = w * x + b$$

- 我们给出两条优化理论的定理，条件为：

设训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ 是线性

可分的，其中 $x_i \in X = R^n$, $y_i \in Y = \{-1, 1\}$, $i = 1, 2, \dots, N$

- 我们可以得到以下两条定理：

感知机学习算法

(1) 存在满足条件 $\|\widetilde{w_{opt}}\| = 1$ 的超平面 $\widetilde{w_{opt}} * \check{x} = w_{opt} * x + b_{opt} = 0$ 且存在 $\gamma > 0$, 对所有 $i = 1, 2, \dots, N$ 则

$$y_i(\widetilde{w_{opt}} * \check{x}_i) = y_i(w_{opt} * x_i + b_{opt}) \geq \gamma$$

感知机学习算法

- 定理证明(1):
- 由于数据集线性可分,则能找到一个超平面完全划分数据。
- 设存在划分超平面: $\widetilde{w_{opt}} * \check{x} = w_{opt} * x + b_{opt} = 0$
- 使得 $\|\widetilde{w_{opt}}\| = 1$, 对于有限的点, 均有:

$$y_i(\widetilde{w_{opt}} * \check{x}_i) = y_i(w_{opt} * x_i + b_{opt}) \geq 0$$

- 存在 $\gamma = \min_i \{y_i(w_{opt} * x_i + b_{opt})\}$
- 使: $y_i(\widetilde{w_{opt}} * \check{x}_i) = y_i(w_{opt} * x_i + b_{opt}) \geq \gamma$

- 定理（2）： $R = \max_{1 \leq i \leq N} \|\tilde{x}_i\|$ ，算法在训练集的误分类次数 k 满足不等式 $k \leq \left(\frac{R}{\gamma}\right)^2$

感知机学习算法

- $R = \max_{1 \leq i \leq N} \|\tilde{x}_i\|$ 算法在训练集的误分类次数 k 满足不等式, $k \leq \left(\frac{R}{\gamma}\right)^2$

- 证明: 令 $\widetilde{w_{k-1}}$ 是第 k 个误分类实例之前的扩充权值向量, 即:

$$\widetilde{w_{k-1}} = (w_{k-1}^T, b_{k-1})^T$$

- 第 k 个误分类实例的条件是: $y_i(\widetilde{w_{k-1}} * \tilde{x}_i) = y_i(w_{k-1} * x_i + b_{k-1}) \leq 0$

- 则 w 和 b 的更新: $w_k \leftarrow w_{k-1} + \eta y_i x_i$ 即: $\widetilde{w_k} = \widetilde{w_{k-1}} + \eta y_i \tilde{x}_i$
 $b_k \leftarrow b_{k-1} + \eta y_i$

感知机学习算法

- $R = \max_{1 \leq i \leq N} \|\tilde{x}_i\|$ 算法在训练集的误分类次数 k 满足不等式, $k \leq \left(\frac{R}{\gamma}\right)^2$
- 推导两个不等式:
- (1) $\widetilde{w}_k * \overline{w_{opt}} \geq k\eta\gamma$
- 由:
$$\begin{aligned}\widetilde{w}_k * \overline{w_{opt}} &= \overline{w_{k-1}} * \overline{w_{opt}} + \eta y_i \overline{w_{opt}} * \tilde{x}_i \\ &\geq \overline{w_{k-1}} * \overline{w_{opt}} + \eta\gamma\end{aligned}$$
- 得:
$$\widetilde{w}_k * \overline{w_{opt}} \geq \overline{w_{k-1}} * \overline{w_{opt}} + \eta\gamma \geq \overline{w_{k-2}} * \overline{w_{opt}} + 2\eta\gamma \geq \dots \geq k\eta\gamma$$

感知机学习算法

- $R = \max_{1 \leq i \leq N} \|\tilde{x}_i\|$ 算法在训练集的误分类次数 k 满足不等式, $k \leq \left(\frac{R}{\gamma}\right)^2$

(2) $\|\widetilde{w}_k\|^2 \leq k\eta^2 R^2$

• 则:

$$\begin{aligned}\|\widetilde{w}_k\|^2 &= \|\widetilde{w}_{k-1}\|^2 + 2\eta y_i \widetilde{w}_{k-1} * \tilde{x}_i + \eta^2 \|\tilde{x}_i\|^2 \\ &\leq \|\widetilde{w}_{k-1}\|^2 + \eta^2 \|\tilde{x}_i\|^2 \\ &\leq \|\widetilde{w}_{k-1}\|^2 + \eta^2 R^2 \\ &\leq \|\widetilde{w}_{k-2}\|^2 + 2\eta^2 R^2 \\ &\leq \dots \\ &\leq k\eta^2 R^2\end{aligned}$$

感知机学习算法

- $R = \max_{1 \leq i \leq N} \|\tilde{x}_i\|$ 算法在训练集的误分类次数 k 满足不等式, $k \leq \left(\frac{R}{\gamma}\right)^2$

结合两个不等式: $k\eta\gamma \leq \widetilde{w}_k * \overline{w_{opt}} \leq \|\widetilde{w}_k\| \|\overline{w_{opt}}\| \leq \sqrt{k}\eta R$
 $k^2\gamma^2 \leq kR^2$

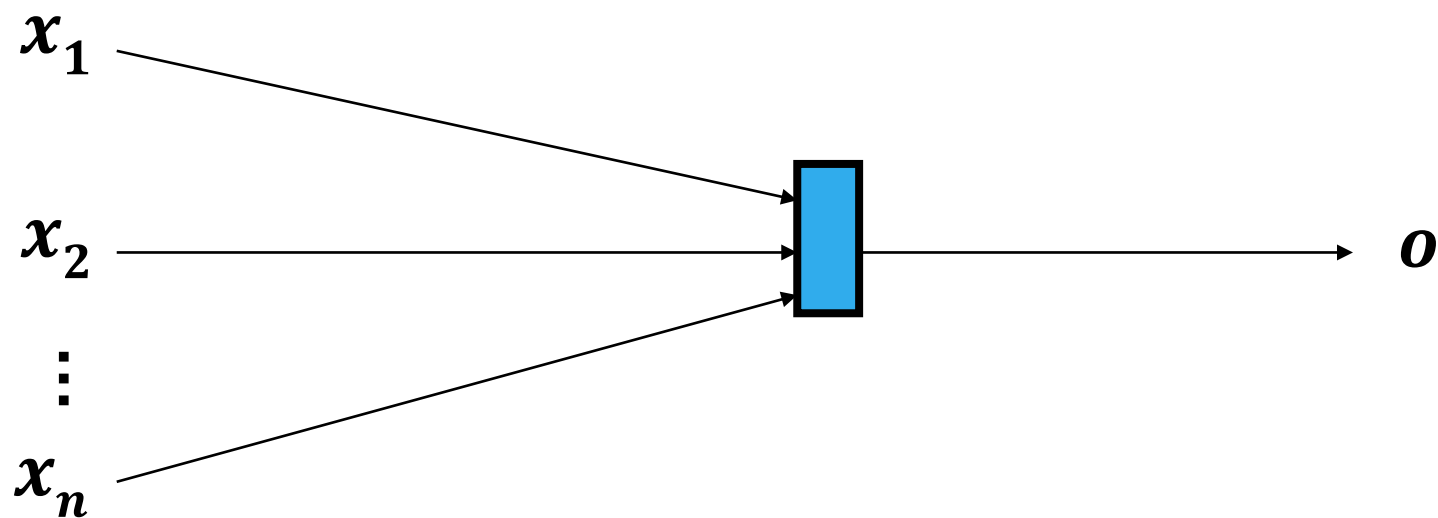
得: $k \leq \left(\frac{R}{\gamma}\right)^2$

感知机学习算法

- 两条定理表明：
 - 误分类的次数 k 是有上界的，当训练数据集线性可分时，sign感知机学习算法迭代是收敛的。
- 感知机算法存在许多解，既依赖于初值，也依赖迭代过程中误分类点的选择顺序。
- 为得到唯一分离超平面，需要增加约束。
- 线性不可分数据集，迭代震荡。

六、ADALINE: Adaptive Linear Element

ADALINE的结构



- 与感知器网络结构非常相似
- 激活函数不同：线性函数

$$v = \sum_{i=1}^N x_i \omega_i + b \quad y = \text{purelin}(v) \quad y = \text{purelin}\left(\sum_{i=1}^N x_i \omega_i + b\right)$$

ADALINE 的学习：LMS

- Least-Mean-Square (LMS) learning algorithm
 - The idea: 试图最小化均方误差 (square error), 其为权重的函数

$$E(w(n)) = \frac{1}{2} e^2(n)$$

$$e(n) = d(n) - \sum_{j=0}^m x_j(n) w_j(n)$$

- 可以通过最速下降法来寻找误差函数的最小点 (Steepest descent method)

最速下降法

(Steepest Descent Method)

- 从任意一点开始
- 寻找使得E下降最快的方向

$$-(\text{gradient of } E(\mathbf{w})) = -\left[\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_m}\right]$$

- 在该方向上前进一小步

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(\text{gradient of } E(n))$$

Least-Mean-Square algorithm (Widrow-Hoff algorithm)

- 求得gradient(E)的近似值

$$\begin{aligned}\frac{\partial E(\mathbf{w}(n))}{\partial \mathbf{w}(n)} &= \mathbf{e}(n) \frac{\partial \mathbf{e}(n)}{\partial \mathbf{w}(n)} \\ &= \mathbf{e}(n)[- \bar{\mathbf{x}}(n)]\end{aligned}$$

- 权重的更新策略为：

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta \bar{\mathbf{x}}(n) \mathbf{e}(n)$$

LMS 算法总结

训练样本: 输入信号向量 $\mathbf{x}(n)$

期望的响应 $d(n)$

用户定义的参数 $\eta > 0$

初始化 $\text{set } \hat{\mathbf{w}}(1) = \mathbf{0}$

计算 $\text{for } n = 1, 2, \dots \text{ compute}$
$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$
$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n)e(n)$$

LMS的学习率

- 学习率越小，算法的运行时间就越长，算法也就记忆了更多过去的数据
- 学习率的倒数反映了LMS算法的记忆容量大小
- 只要学习率满足下式，LMS算法就是按方差收敛的：

$$0 < \eta < \frac{2}{\lambda_{\max}}$$

λ_{\max} ：输入向量自相关矩阵的最大特征值

一般不可知，用矩阵的迹（主对角线元素之和）代替

LMS的学习率

- 学习率随着学习的进行逐渐下降比始终不变更加合理

$$\eta = \frac{\eta_0}{n}$$

反比例函数

$$\eta = c^n \eta_0$$

指数式下降

$$\eta = \frac{\eta_0}{1 + \left(\frac{n}{\tau}\right)}$$

搜索—收敛方案

线性神经元的功能

- 作为分类器
 - 可以被训练解决任何包含两类的线性可分的分类问题
 - 功能上等效于线性判别函数分类器
 - 可以比感知器更快地找到最优的权值
 - 和感知器找到的分界线有不同的斜率
- 作为预报器
 - 线性神经元有将输入到输出进行连续映射的能力
 - 可以逼近一个线性函数
 - 功能上类似于统计学中的线性回归
 - 寻找使预报误差最小的线性函数
 - 多输入线性神经元
 - 功能上与统计学中的多重线性回归等效

ADALINE和感知器的比较

- 神经元模型不同
 - 感知器：非线性模型，只能输出两种可能的值，其激活函数是阈值函数
 - ADALINE：线性模型，输出可以取任意值，其激活函数是线性函数
- 功能不同
 - Perceptron:感知器只能做简单的分类
 - LMS：还可以实现拟合或逼近
- 分类性能不同
 - LMS算法得到的分类边界往往处于两类模式的正中间
 - 感知器学习算法在刚刚能正确分类的位置就停下来
 - 使分类边界离一些模式距离过近，使系统对误差更敏感

MADALINE

- 若网络中包含多个神经元节点，就能形成多个输出，这种线性神经网络叫Madaline网络。
- Madaline可以用一种间接的方式解决线性不可分的问题
 - 用多个线性函数对区域进行划分
 - 然后对各个神经元的输出做逻辑运算
- 线性神经网络解决线性不可分问题（另一思路）
 - 对神经元添加非线性输入，从而引入非线性成分
 - 会使等效的输入维度变大

六、单个神经元应用举例

实例：回归预测股票价格

- 股票观测：
 - 通过前两天的股票价格预测后一天的股票价格

日期	1	2
股价	1	0.5

- 比较随机学习、梯度下降方法的优劣

随机学习：实例

- 股票数据集中有两个数据(1,1), (2,0.5)
- 选取 $\delta = 0.1, \xi = 0.01, \epsilon = 0.00001$
- 进行10次随机学习方法，结果如下：

轮次	a	b	误差	迭代次数
1	-0.5	1.50	0.000009153	1420451
2	-0.5	1.50	0.000009326	532145
3	-0.5	1.50	0.000008633	2627297
4	-0.5	1.50	0.000008081	5840588
5	-0.5	1.50	0.000004571	2289921
6	-0.5	1.50	0.000005472	1724894
7	-0.5	1.50	0.000001017	281716
8	-0.5	1.50	0.000007401	125548
9	-0.5	1.50	0.000005113	966248
10	-0.5	1.50	0.000000824	311006

- 结论：
 - 神经元可以通过自主学习获得合适的权重和偏置值
 - 随机学习是一种低效的学习方法
 - 其思想简单，实现容易，且具有能找到全局最优等特点，对搜索方法进行改进后有一定实用价值

梯度下降法：实例

- 执行梯度下降法10次，每次选择不同的学习率，结果如下表所示：

轮次	α	a	b	误差	迭代次数
1	0.10	-0.50	1.50	0.000000975	434
2	0.20	-0.50	1.50	0.000000976	217
3	0.30	溢出	溢出	溢出	12962
4	0.40	溢出	溢出	溢出	1280
5	0.50	溢出	溢出	溢出	801
6	0.60	溢出	溢出	溢出	626
7	0.70	溢出	溢出	溢出	533
8	0.80	溢出	溢出	溢出	474
9	0.90	溢出	溢出	溢出	433
10	1.00	溢出	溢出	溢出	402

- 结论：

- 学习率较小时，梯度下降法能收敛到期望精度下的解，证明了方法的有效性
- 学习率较大时，算法溢出了，而且随着学习率的增大，溢出所花费的迭代步数越少，这是由于线性激活函数对于输出结果不加控制造成的
- 前面采用最大迭代次数来终止算法，可以误差不变的原则提前结束程序的执行

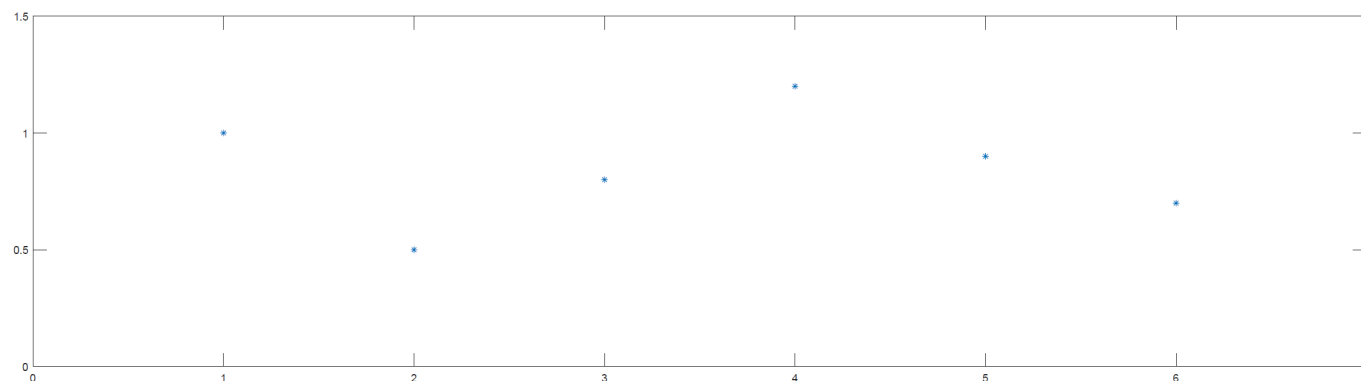
回顾上一讲股票预测的例子
我们用最小二乘法解决

单个神经元的功能： 回归预测股票价格

- 我们观测股票的仅六天的历史价格数据，形成如下表格：

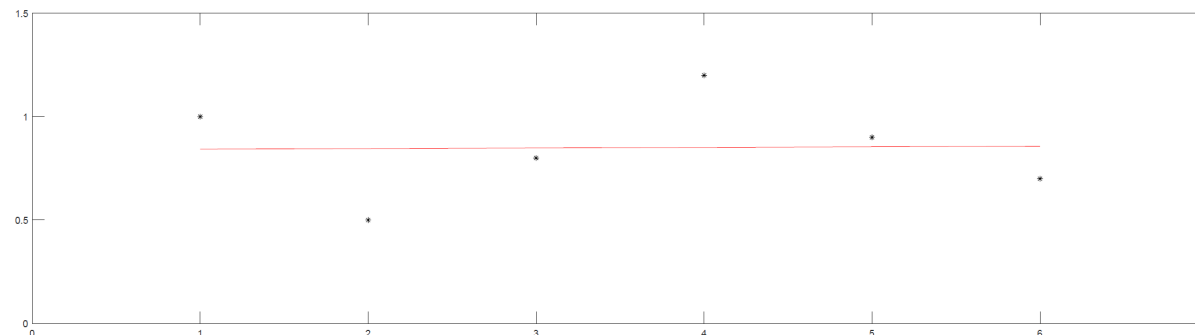
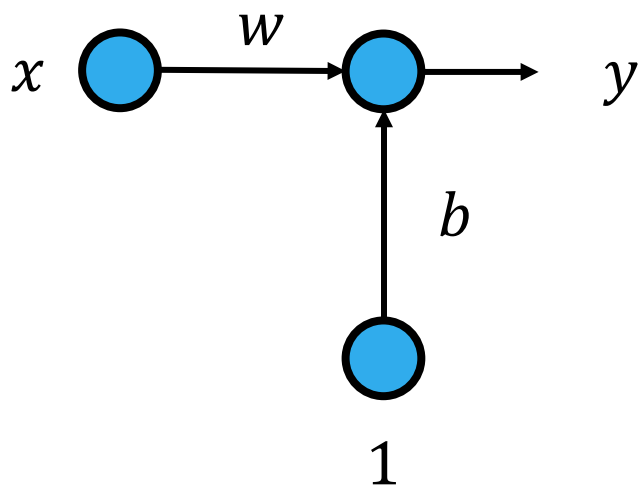
日期	1	2	3	4	5	6
股价	1	0.5	0.8	1.2	0.9	0.7

- 画在坐标系中可得到



单个神经元的功能：回归预测股票价格

- 假设股票波动只与当前日期有关，使用单输入的神经元进行线性函数 $y = wx + b$ 的拟合



- $w = 0.0029, b = 0.84$
- 可以对股票的 future 走势进行预测

单个神经元的功能： 回归预测股票价格 (*作业)

- 假如我们获得了股票20天的波动数据
- 仍采用单输入的神经元进行拟合，模型 $y = wx + b$

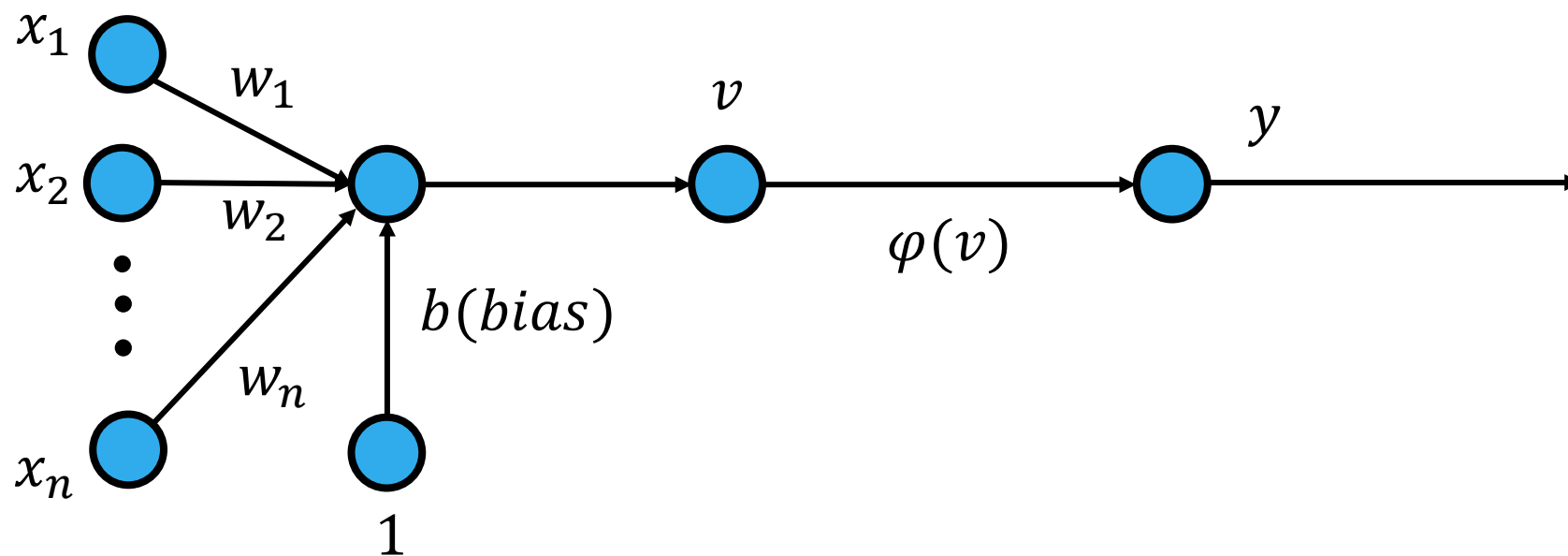
日期	1	2	3	4	5
股价	55.22	56.34	55.52	55.53	56.94
日期	6	7	8	9	10
股价	58.88	58.18	57.09	58.38	38.54
日期	11	12	13	14	15
股价	57.72	58.02	57.81	58.71	60.84
日期	16	17	18	19	20
股价	61.08	61.74	62.16	60.80	60.87

- 线性拟合的结果是 $y = 0.3249x + 55.1073$

单个神经元的功能：回归预测股票价格

- 若假设股票波动不仅与当前日期有关，还与多个前面多个日期相关
- 采用多输入的神经元进行拟合，预测模型为

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$



单个神经元的功能： 回归预测股票价格

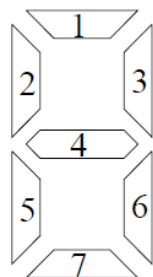
- 采用多元线性回归可以求得：

$$w_1 = -0.2064, w_2 = 0.5659, w_3 = -0.5767, w_4 = 1.0369, b = 10.9260$$

- 检测该模型的预测结果，第5天56.9746（预测值）-56.94（准确值），第10天59.3102-58.54，第13天58.8806-57.81，第20天60.4564-60.87
- 相比单元预测模型的预测结果为56.7318、58.3563、59.3310、61.6053，可以看出多元线性模型比单元线性模型的预测更为准确

作业

- 设计一个多输入单输出的神经元用于进行股票价格(*作业)的预测，通过股票的历史数据来训练该神经元，希望预测结果尽量接近真实值。
- “损坏的”LED灯问题



$$\mathbf{c}(2) = \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} \quad \mathbf{c}(3) = \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} \quad \mathbf{c}(8) = \begin{array}{|c|} \hline \text{---} \\ \hline \end{array}$$

考虑一个由7个LED灯组成的数字显示器，每个LED灯的亮暗状态分别标记为“+1”和“-1”，这7个LED灯的状态共同组成一个向量 \mathbf{x} 。显示器上显示的数字标记为 s 。例如当 $s = 2$ 时，第 j 个LED灯($j = 1, \dots, 7$)显示为 $c_j(2)$ （即正确显示）的概率为 $1 - f$ ，或者翻转显示（错误显示）的概率为 f 。假定显示的数字 s 只为2或者3，当给定一个显示状态 \mathbf{x} 的时候，显示数字 s 为2或者3的概率为多少？例如 $P(s = 2|\mathbf{x})$ 可以写成如下形式：

$$P(s = 2|\mathbf{x}) = \frac{1}{1 + \exp(-w^T \mathbf{x} + b)}$$

这里 $f = 0.1$