

第三章：单层感知机

南京大学人工智能学院

申富饶

目录

CONTENTS

01. 回顾：单个神经元

02. 神经元的连接

03. 单层感知机

04. 单层感知机的应用

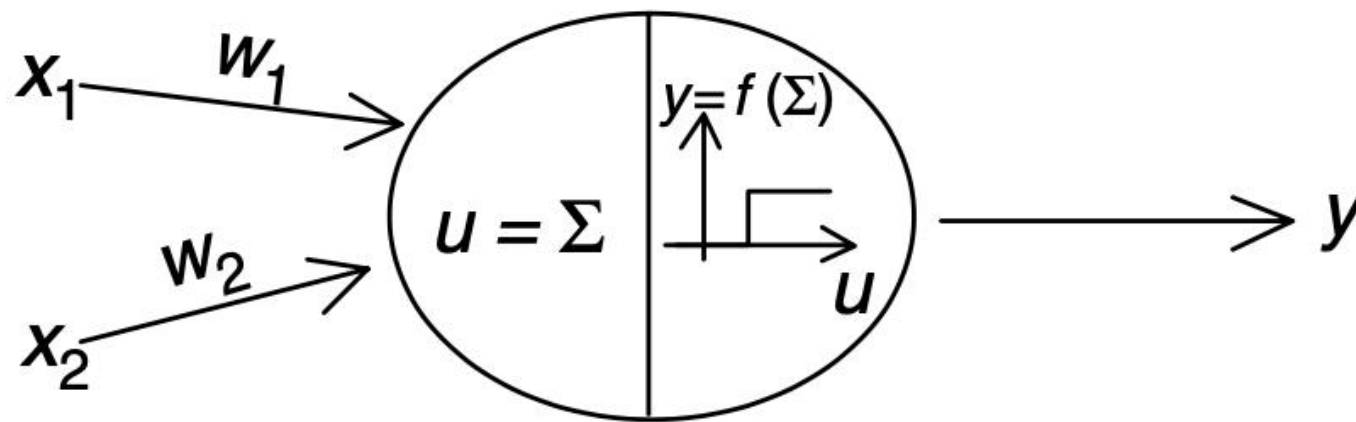
01

回顾：单个神经元

重点：单个神经元可以模拟的函数是有限的，因此可以解决的问题也是有限的

单个神经元结构：多输入——单输出

- 本章节中提到的神经元为MP神经元，从上一章的内容可以知道，单个神经元只能解决多输入-单输出的问题。



二分类问题：多输入——单输出

- 示例：利用身高（height: cm）和体重（weight: kg）去预测一个人是否存在危害身体健康的肥胖症状（obesity）（假设该问题线性可分）
- 身高： X_1
- 体重： X_2 ;
- 肥胖： Y (0 : 否; 1 :是)
- 样本 (X, Y) , $X = (X_1, X_2)$, $X_1, X_2 \in \{0, 200\}$, $Y \in \{0, 1\}$

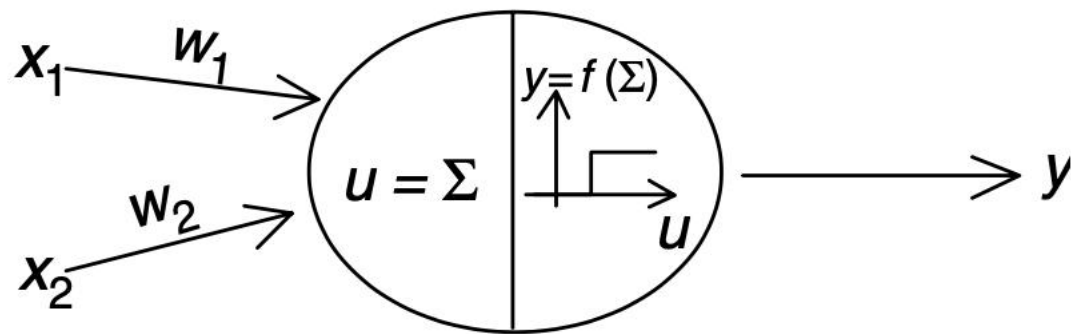
二分类问题：多输入——单输出

- 示例：利用身高（height: cm）和体重（weight: kg）去预测一个人是否存在危害身体健康的肥胖症状（假设该问题线性可分）

- 可以利用单个MP神经元来解决这个分类问题：

- $U = w_0 + w_1X_1 + w_2X_2$

- $f(U) = \begin{cases} 0, & U < 0 \\ 1, & U \geq 0 \end{cases}$



多分类问题：多输入——多输出

- 示例：Iris数据集包含150个数据样本，分为3类，每类50个数据，每个数据包含4个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度4个属性预测鸢尾花卉属于（山鸢尾、杂色鸢尾、维吉尼亚鸢尾）三个种类中的哪一类。
- 输入：花萼长度 x_1 ；花萼宽度 x_2 ；花瓣长度 x_3 ；花瓣宽度 x_4
- 输出：山鸢尾 y_1 （0-否；1-是）；杂色鸢尾 y_2 ；维吉尼亚鸢尾 y_3

思考：可否用一个输出神经元处理多分类问题？

（山鸢尾： $y = 1$ ；杂色鸢尾 $y = 2$ ；维吉尼亚鸢尾： $y = 3$ ）

总结：单个神经元

单个MP神经元的结构决定了它只能解决“**多输入——单输出**”的问题

单个神经元仅能处理**二分类问题**

由于结构限制，无论是单纯的多分类问题，还是分类和回归相结合的问题，单个神经元都无法解答

那我们如何拓展神经元的功能呢？

答：连接多个神经元

02

神经元的连接

神经元的连接概述

神经元的扩展（宽度和深度）

其他连接方式

神经元的连接概述

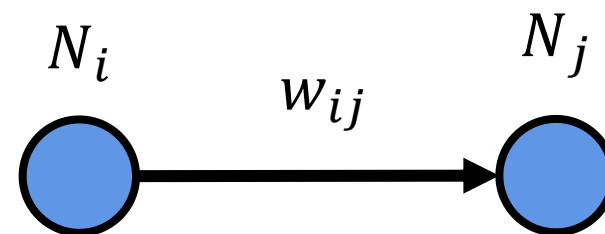
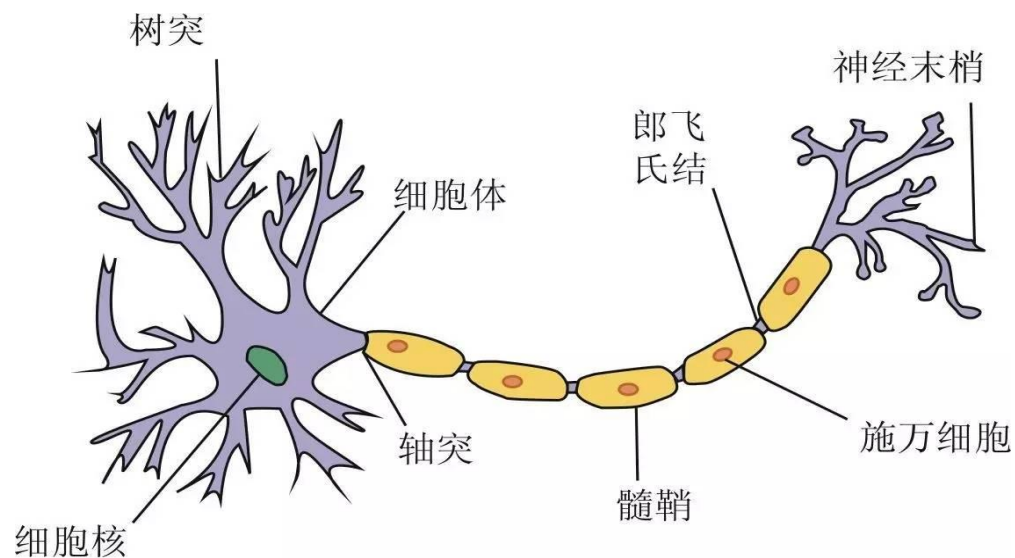
连接：神经元之间传递信息的方式

- 单个神经元
 - 计算能力有限
 - 虽然能够解决一部分问题，但是在实际生活中是远远不够的
- “人多力量大” ——使用多个神经元共同解决问题
 - 大脑皮质中成千上万的神经元组成了神经系统
- 神经元通过连接交流信息，共同完成计算任务

神经元的连接概述

连接的拓扑表示：

用正号（“+”，可省略）表示传送来的信号起刺激作用，它用于增加神经元的活跃度；
用负号（“-”）表示传送来的信号起抑制作用，它用于降低神经元的活跃度。

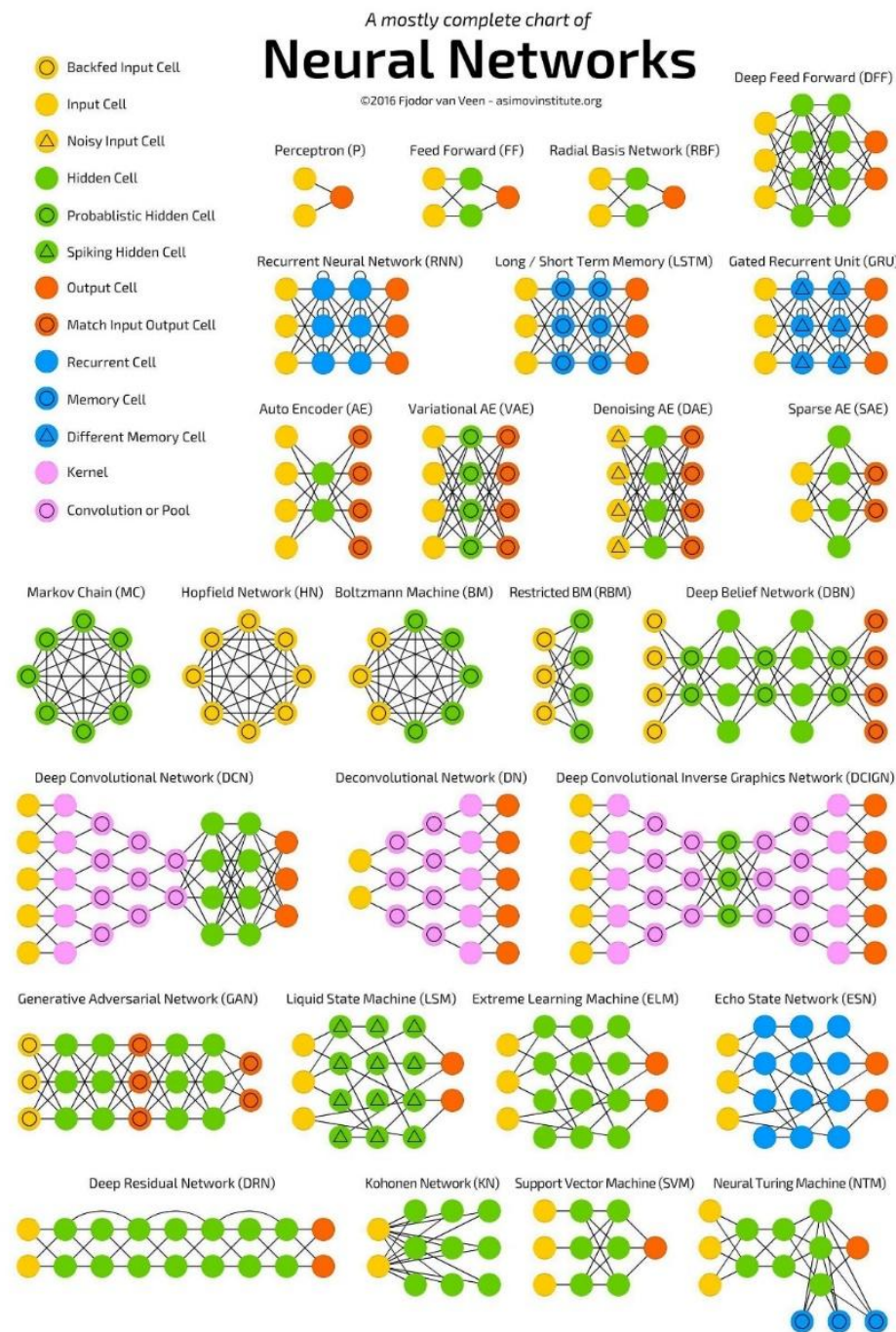


神经元的连接概述

连接方式

神经元根据不同的连接方式，组成不同功能的神经网络。

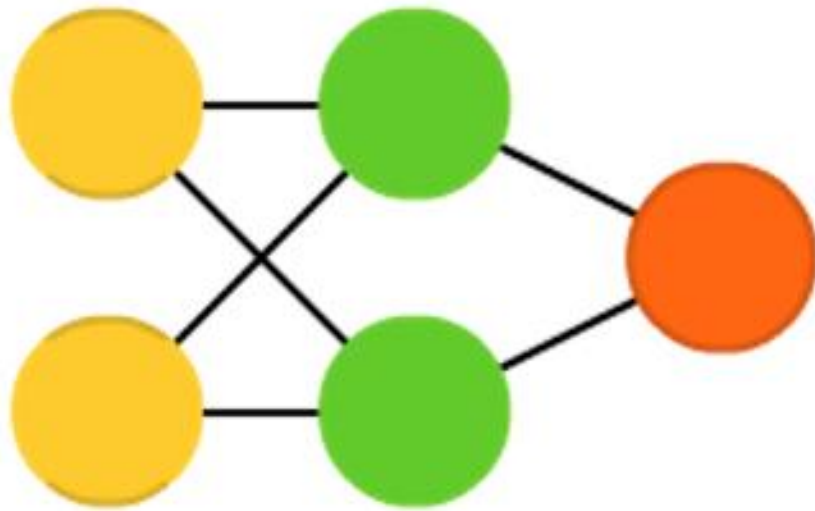
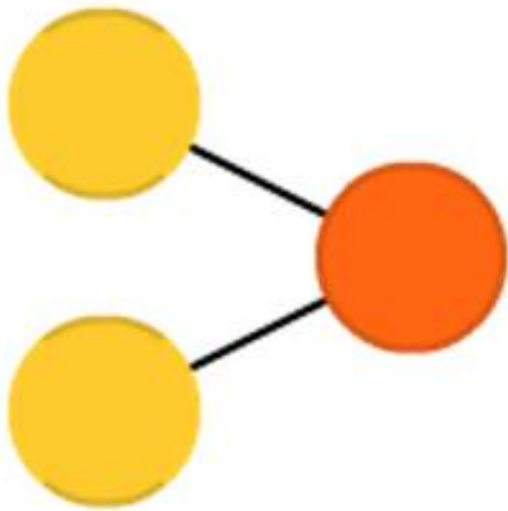
- 层级结构
- 互联网结构
- 等等



神经元的连接概述

层级结构

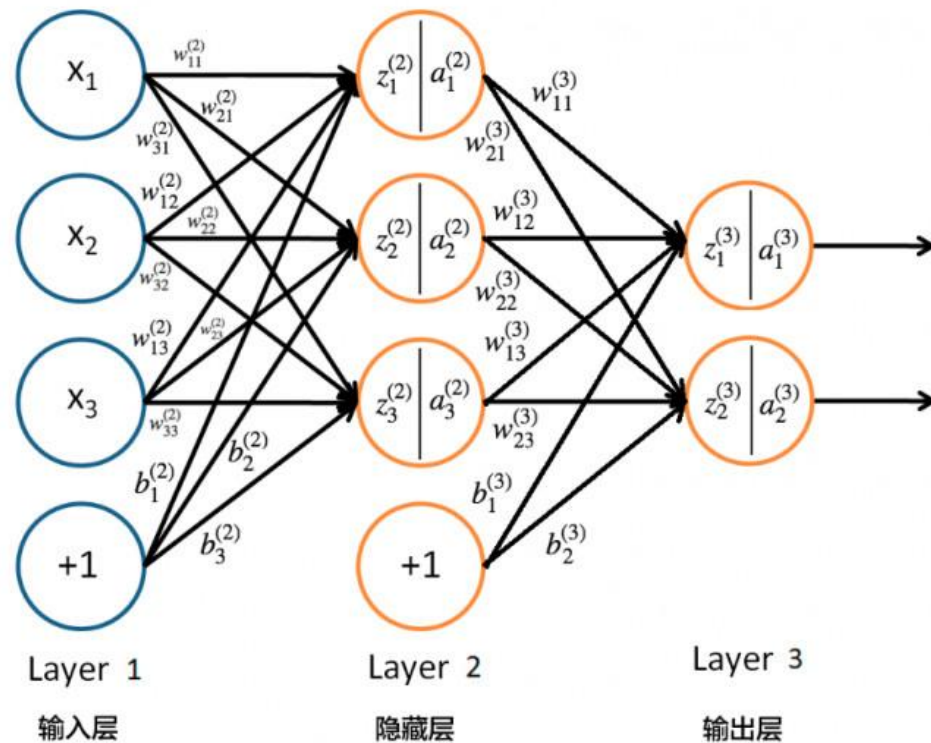
其中每层由并行的单元组成。通常同一层不具有连接、两个相邻层完全连接(每一层的每一个神经元到另一层的每个神经元)。感知机和前馈神经网络是非常典型的层级连接结构。



神经元的连接概述

层级结构

- 如果层级结构网络具有足够的隐藏神经元，则在理论上可以建模任意输入和输出之间的函数关系

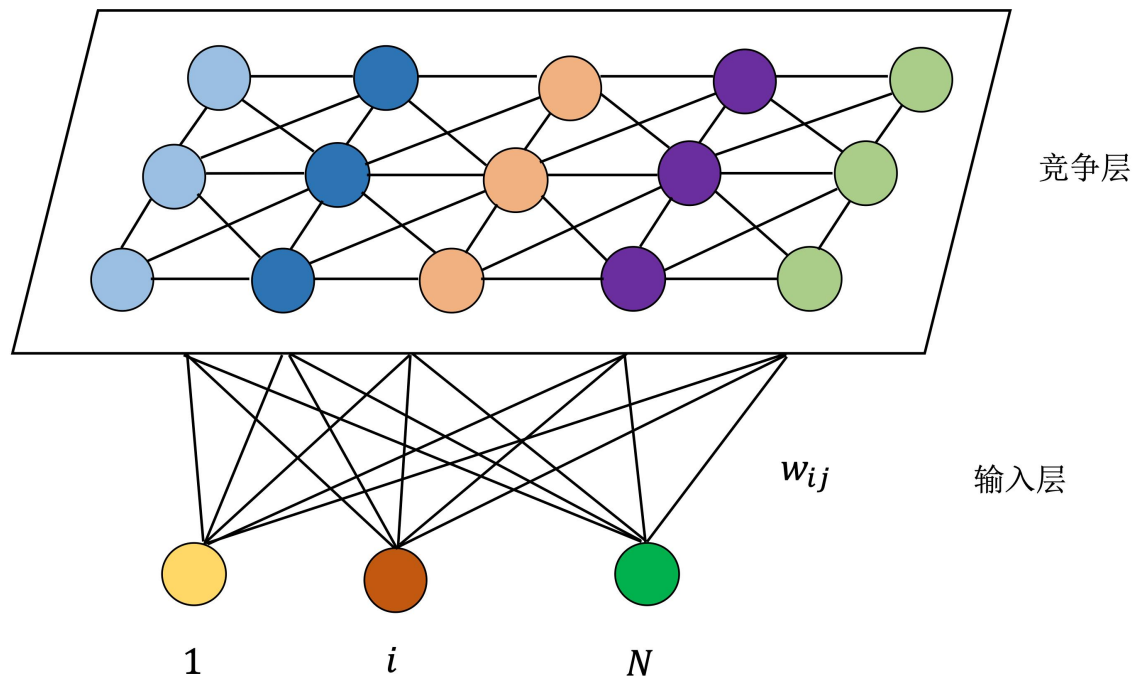


神经元的连接概述

层级结构

层（级）内联接：区域内（Intra-field）联接

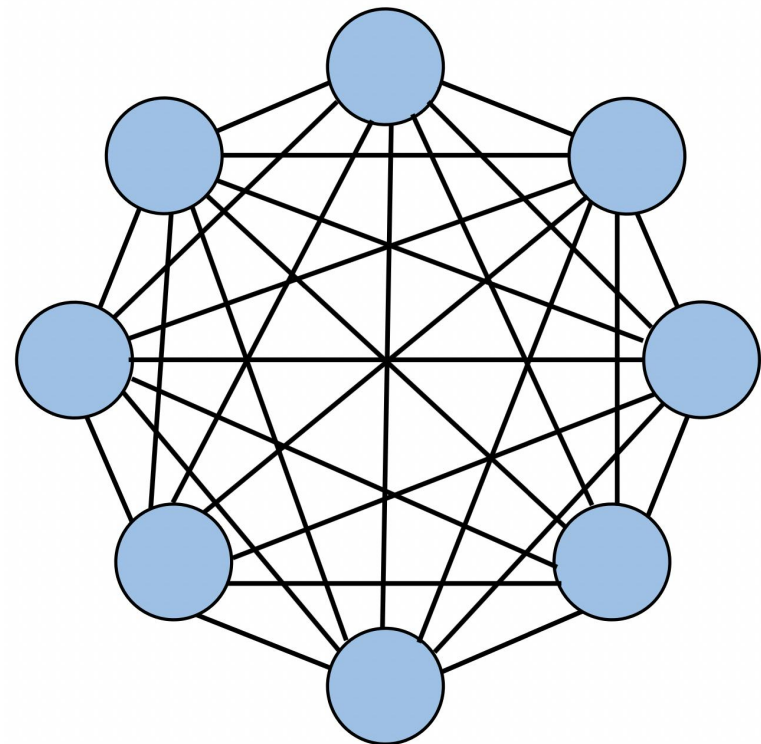
- 用来加强和完成层内神经元之间的竞争
- 实际应用范围较为有限，通常将它们与其他网络结合形成新的网络



神经元的连接概述

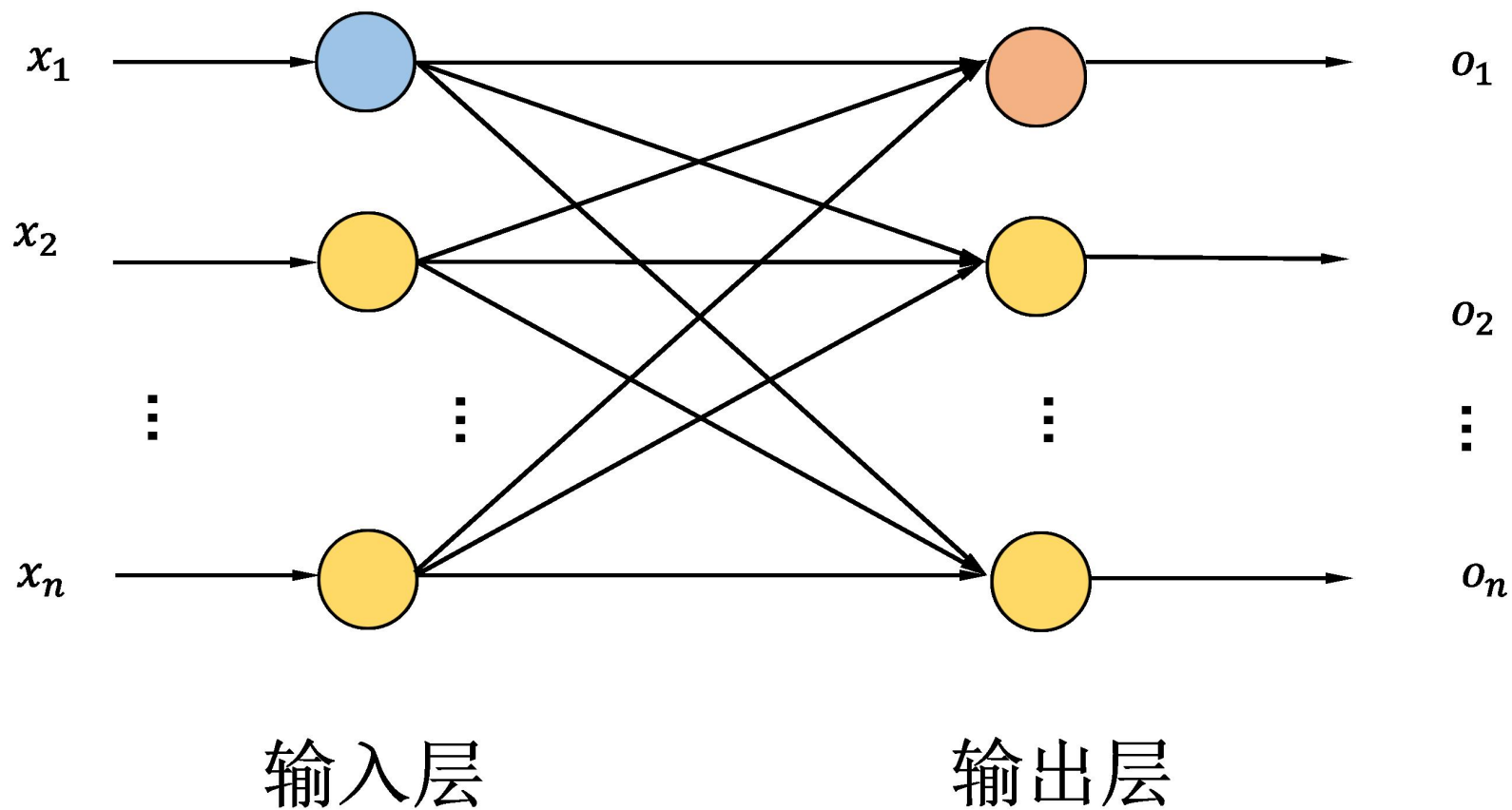
互联网型结构

- 最典型的的就是Hopfield网络(HN)和马尔可夫链。
- **Hopfield网络(HN)**的每个神经元被连接到其他神经元。每个节点在训练前输入，然后在训练期间隐藏并输出。通过将神经元的值设置为期望的模式来训练网络，此后权重不变。
- **马尔可夫链**(MC或离散时间马尔可夫链，DTMC)是玻尔兹曼机和Hopfield网络的前身。它虽然不是真正的神经网络，但其参数优化机制类似于神经网络的学习过程，奠定了BM和HNs的理论基础。



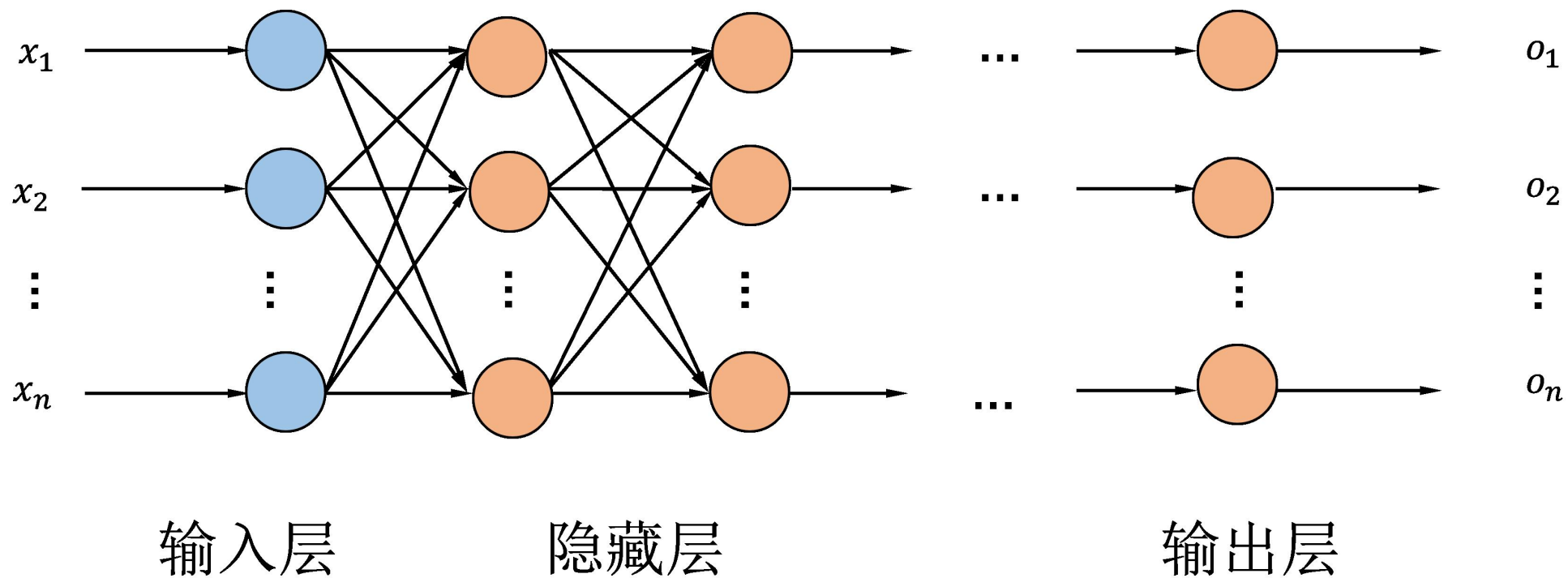
神经元的扩展：宽度扩展

多输出的单层感知机



神经元的扩展：深度扩展

多层感知机



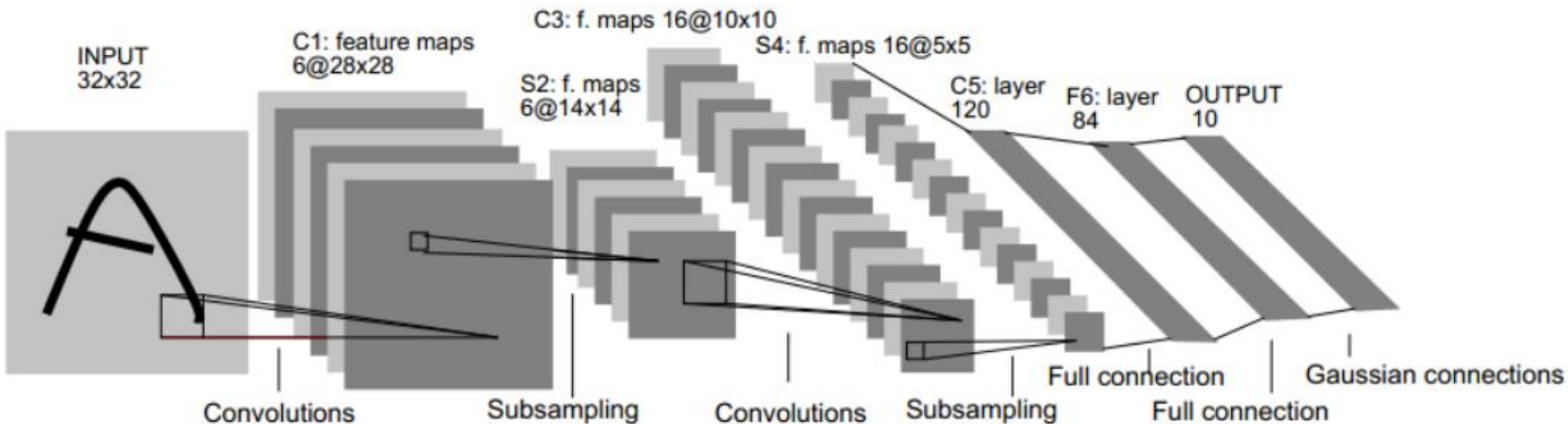
其他连接方式

- 卷积神经网络 (Convolutional Net)
- 竞争神经网络 (Competitive Net)
- 循环神经网络 (Recurrent Net)
- 等等

其他连接方式

卷积神经网络

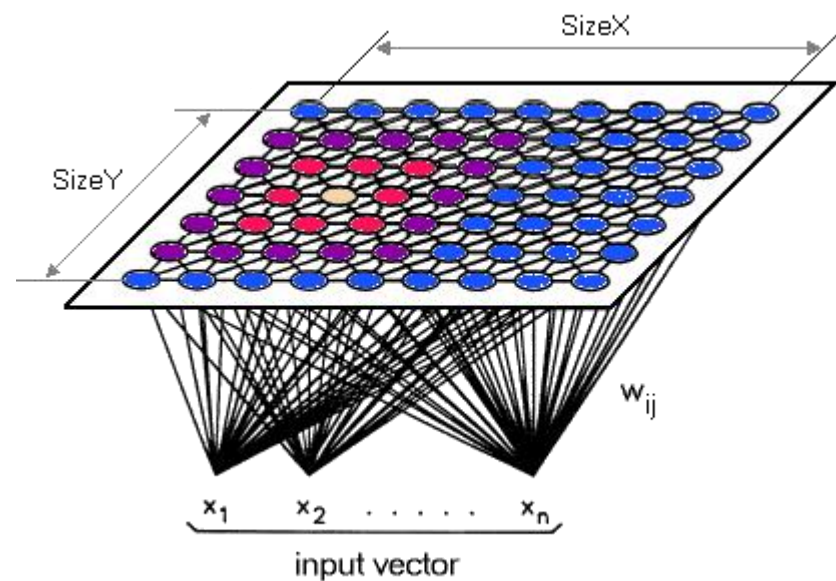
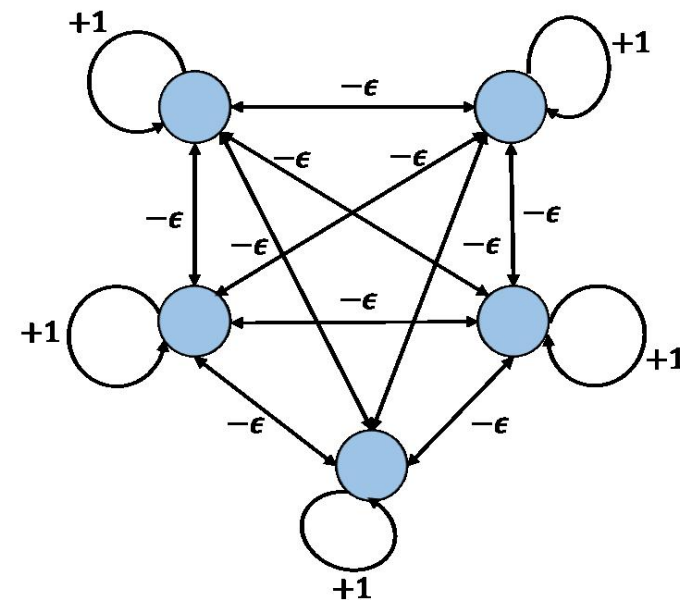
- 包含卷积计算且具有深度结构的前馈神经网络
- 主要结构：卷积层、池化层、全连接层



其他连接方式

竞争神经网络

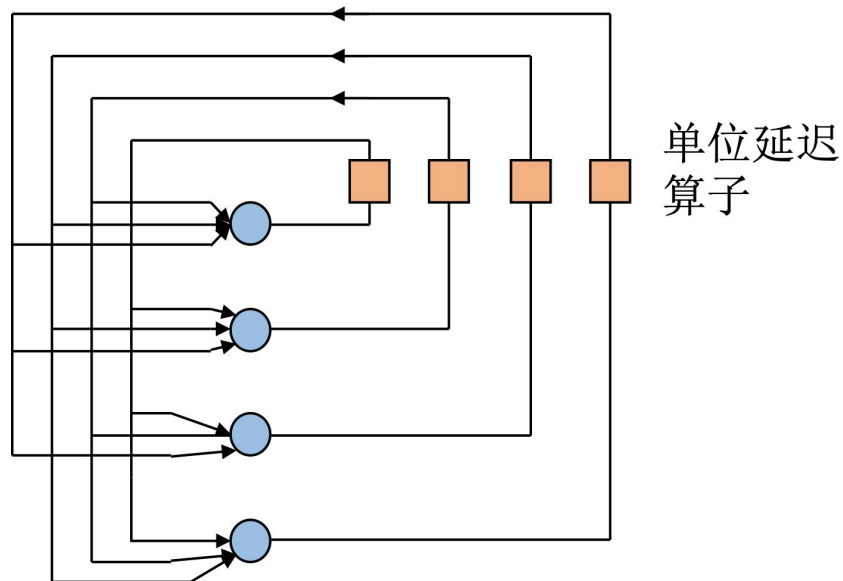
- 神经节点之间互相连接
- 很多神经网络采用竞争层作为其中一部分
- Winner-Take-All
- 例子：MAXNET、Self Organizing Map



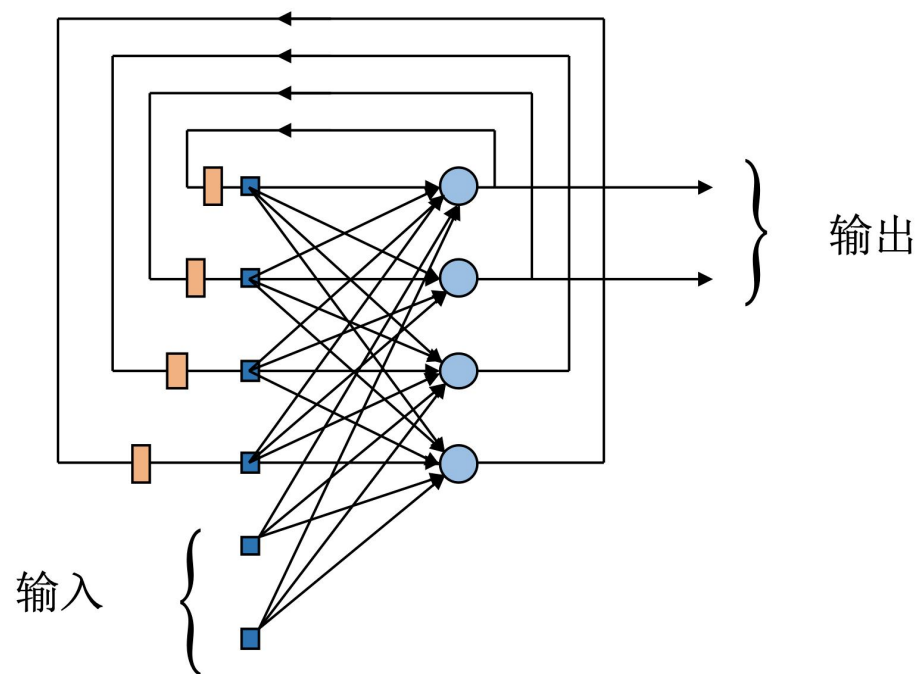
其他连接方式

循环神经网络

- 以序列数据作为输入，在序列的演进方向进行递归且所有节点按链式连接的递归神经网络



无隐藏层



有隐藏层

03

单层感知机

- 历史简述
- 结构和数学表达
- 学习算法
- 感知机收敛定理
- 示例

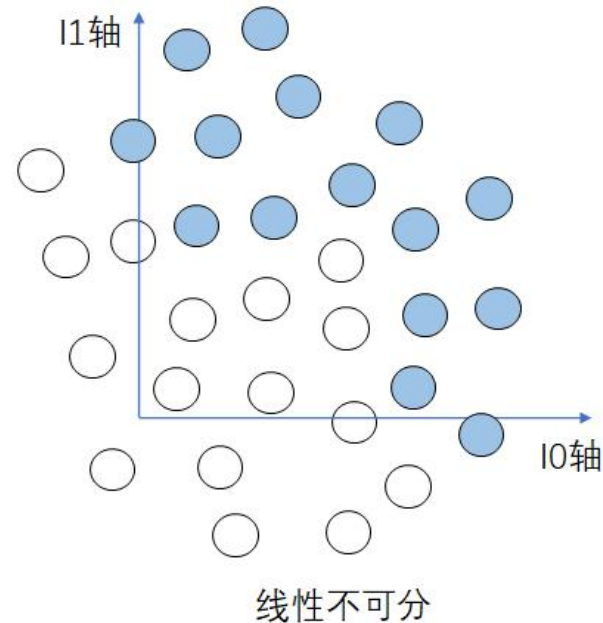
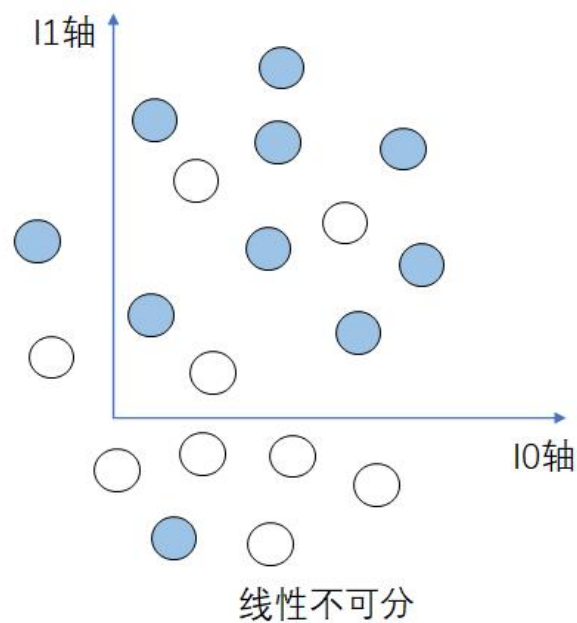
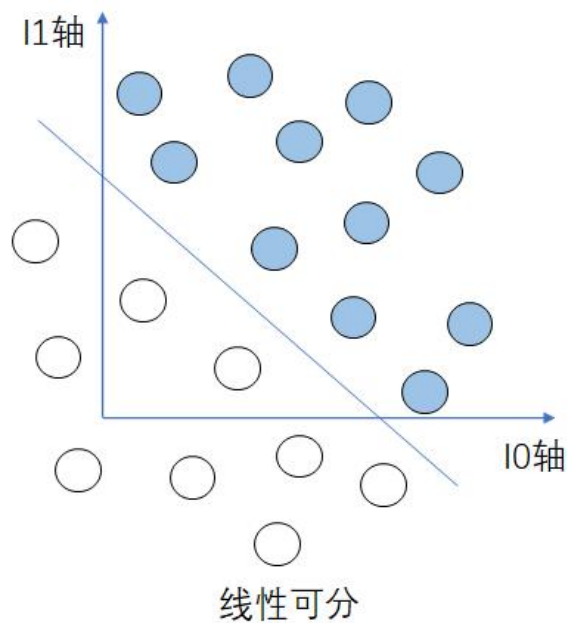
历史简介：从神经元到感知机

- 1943年, McCulloch 和 Pitts 定义了神经网络 (neural networks) 为一种计算机器 (computing machines)
- 1949年, Hebb 提出了自组织学习的第一法则
- 1957年, Frank Rosenblatt 在 Cornell Aeronautical Laboratory 发明了感知机算法

单层感知机中涉及的基础概念（二维）

线性可分 (Linearly Separable)

- 训练样本可由直线完全分开，直线可表示为 $y = wx + b$ ，其中 w 是权重， b 是偏置。
- 单层感知机的学习目标就是根据样本找到直线的权重和偏置。



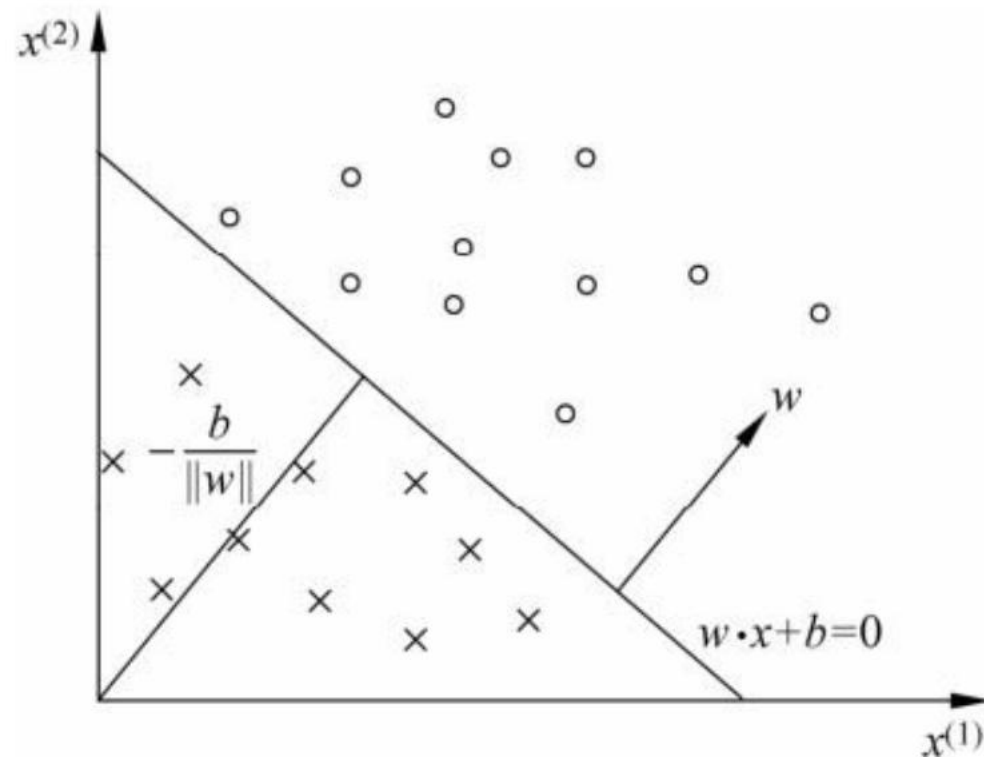
单层感知机中涉及的基础概念（二维）

几何解释

决策边界为二维空间中的直线

$$wx + b = 0$$

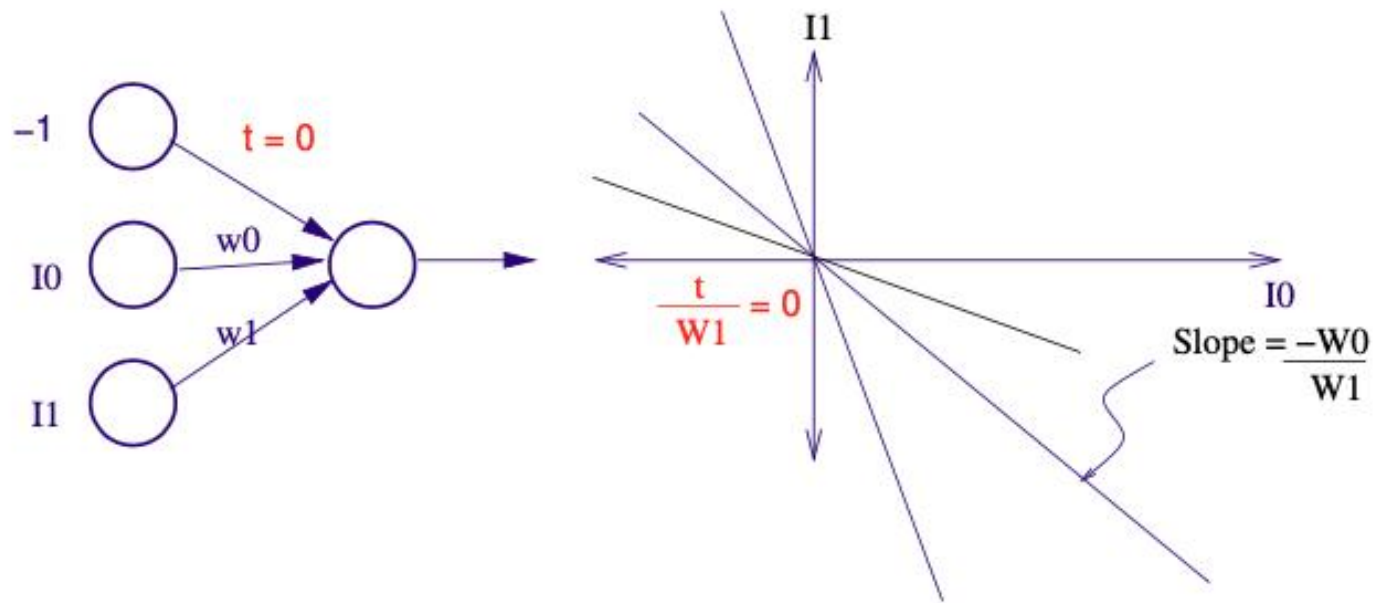
在该直线上方的点，输出为1， 下方，输出为-1



单层感知机中涉及的基础概念（二维）

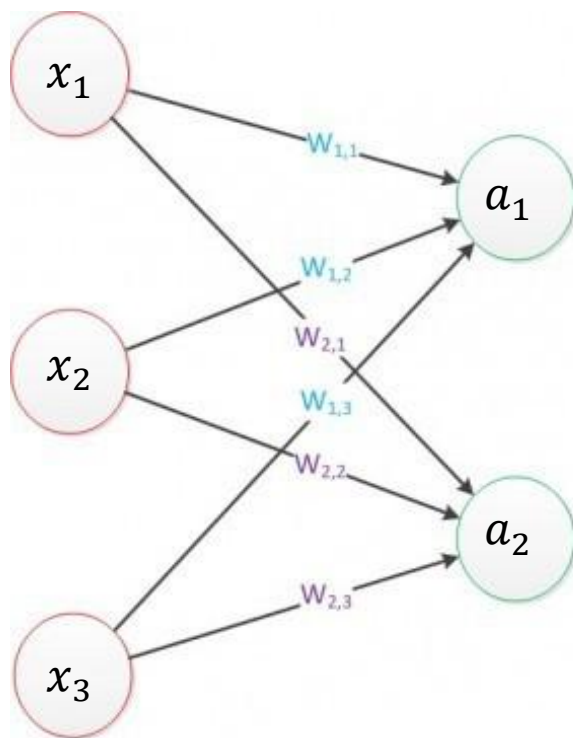
偏置 (bias) 的意义

- 没有偏置项的时候，直线必须过原点，不同的权重只调整斜率



单层感知机的数学表达：分量表达

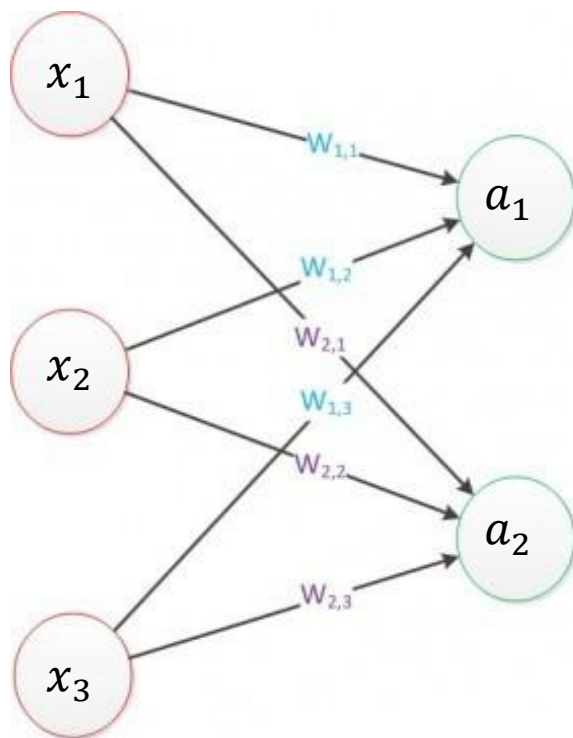
- 多输入——多输出



- a_i : 输出层神经元
- i : 该层第几个神经元
- x_j : 输入层神经元
- j : 该层第几个输入
- $w_{i,j}$: a_i 与 x_j 之间的连接权重
- $w_{i,0}$: 偏置项 (也写作 b) , $x_0 = 1$
- $g(\cdot)$: 激活函数 (阈值函数等)

单层感知机的数学表达：分量表达

- 多输入——多输出



$$z_1 = w_{1,0}x_0 + w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3$$

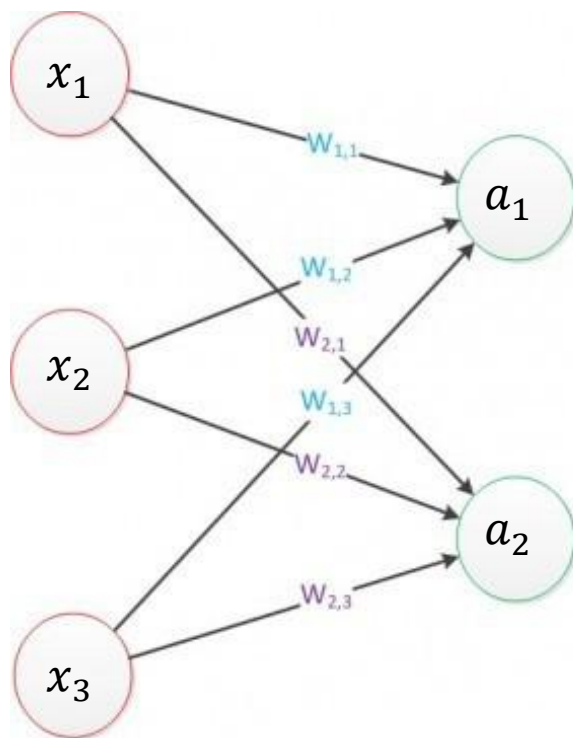
$$a_1 = g(z_1)$$

$$z_2 = w_{2,0}x_0 + w_{2,1}x_1 + w_{2,2}x_2 + w_{2,3}x_3$$

$$a_2 = g(z_2)$$

单层感知机的数学表达：张量表达

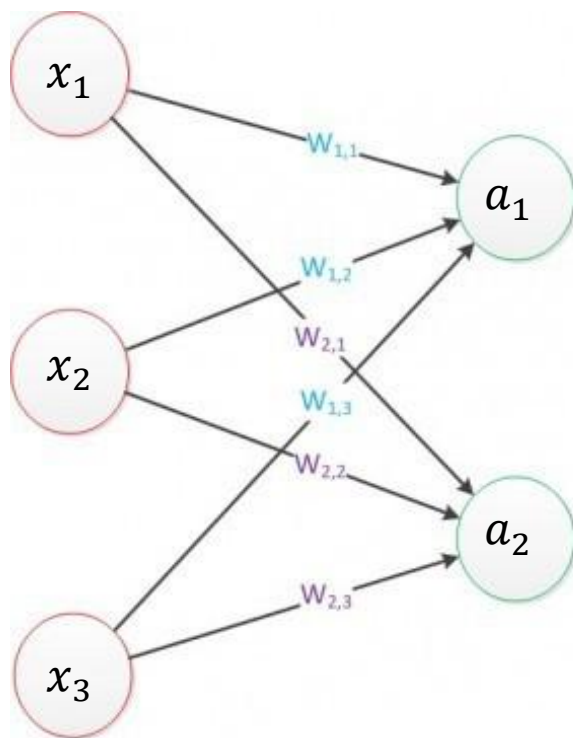
- 多输入——多输出



- a_i : 输出层神经元
- i : 该层第几个神经元
- 张量形状(2×1)
- $\mathbf{a} = [a_1, a_2]^T$

单层感知机的数学表达：张量表达

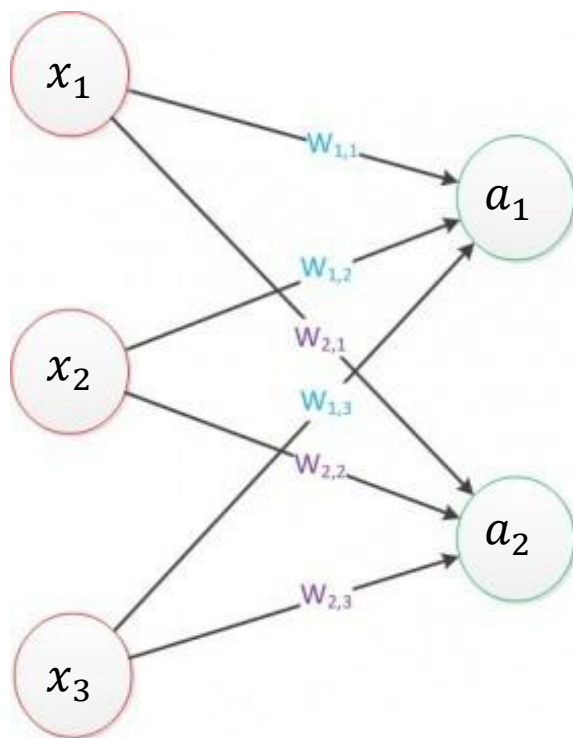
- 多输入——多输出



- x_j : 输入层
- j : 该层第几个输入
- $x_0 = 1$
- 张量 x (4×1)
- $x = [x_0, x_1, x_2, x_3]^T$

单层感知机的数学表达：张量表达

- 多输入——多输出



- 张量 W (4×2)

$$W = \begin{bmatrix} W_{1,0} & W_{2,0} \\ W_{1,1} & W_{2,1} \\ W_{1,2} & W_{2,2} \\ W_{1,3} & W_{2,3} \end{bmatrix}$$

- 张量 z (2×1) = $(2 \times 4) \cdot (4 \times 1)$

$$z = W^T x$$

- 张量 a 形状 (2×1)

$$a = g(z)$$

学习算法

Least-Mean-Square (LMS)算法

自适应滤波器

- 1960年, Widrow and Hoff 在针对单神经元的自适应滤波器(Adaptive Filter)中使用了 least-mean-square (LMS) algorithm, 也叫做 delta rule
- 在刚刚介绍的感知机中, 如果将激活函数的输出是类别, 则单个的神经元可以看作是一个独立的分类器
- 自适应滤波器是能够根据输入信号自动调整性能、进行数字信号处理的离散时间系统。从自适应滤波器的角度来说, 单个神经元是一个信号处理单元。

自适应滤波器

- 设想一个未知的动力系统中 (dynamical system) , 由 m 个输入计算1个输出
- 这个系统中的输入输出可以被这样定义:

$$\mathcal{T}: \{\mathbf{x}(i), d(i); i = 1, 2, \dots, n, \dots\}$$

- 输入: $\mathbf{x}(i) = [x_1(i), x_2(i), \dots, x_m(i)]^T$
- 理想输出: $d(i)$
- 输入的向量可以是一组在均匀时间间隔下的空间快照或者时间序列

自适应滤波器

- 自适应滤波器由两个主要流程组成：

- 滤波过程（Filter Process）：由输入生成输出

$$y(i) = \mathbf{x}^T(i) \mathbf{w}(i)$$

- 自适应过程（Adaptive Process）：自动调整权重以减小误差

$$e(i) = d(i) - y(i)$$

Unconstrained Optimization Techniques

- 如何调整 $w(i)$ 来减小 $e(i)$?
- 目标：对权重矩阵 w ，最小化代价函数 $\mathcal{E}(w)$ ：找到最优解 w^*
- 最优解的必要条件：

$$\nabla \mathcal{E}(w^*) = 0$$

- 由梯度的定义

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_m} \right]^T$$

- 得

$$\nabla \mathcal{E}(w^*) = \left[\frac{\partial \mathcal{E}}{\partial w_1}, \frac{\partial \mathcal{E}}{\partial w_2}, \dots, \frac{\partial \mathcal{E}}{\partial w_m} \right]^T$$

Steepest Descent

- 迭代更新的算法应该有以下性质

$$\mathcal{E}(\mathbf{w}(n+1)) < \mathcal{E}(\mathbf{w}(n))$$

- 定义 $\nabla \mathcal{E}(\mathbf{w})$ 为 \mathbf{g} , 则权重更新算法变为

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g}(n)$$

- 其中 η 是学习率, 则有

$$\Delta \mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) = -\eta \mathbf{g}(n)$$

Steepest Descent $\varepsilon(\mathbf{w}(n+1)) < \varepsilon(\mathbf{w}(n))$ 是否成立?

- 对 $\varepsilon(\cdot)$ 在 $\mathbf{w}(n)$ 附近做泰勒公式一阶展开得

$$\varepsilon(\mathbf{w}(n+1)) \approx \varepsilon(\mathbf{w}(n)) + \mathbf{g}^T(n) \Delta \mathbf{w}(n)$$

- 又有 $\Delta \mathbf{w}(n) = -\eta \mathbf{g}(n)$, 得

$$\varepsilon(\mathbf{w}(n+1)) \approx \varepsilon(\mathbf{w}(n)) - \eta \mathbf{g}^T(n) \mathbf{g}(n) = \varepsilon(\mathbf{w}(n)) - \underbrace{\eta \|\mathbf{g}(n)\|^2}_{\text{positive}}$$

- 得证, 在学习率相对较小的情况下, 有

$$\varepsilon(\mathbf{w}(n+1)) < \varepsilon(\mathbf{w}(n))$$

- Taylor Series:

$$f(x) = f(a) + f'(x-a) + \frac{f''(a)(x-a)^2}{2!} + \dots$$

Least-Mean-Square (LMS) Algorithm

- 由瞬时值 (instantaneous values) 定义代价函数

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

- 对 \mathbf{w} 求微分

$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = e(\mathbf{w}) \frac{\partial e(\mathbf{w})}{\partial \mathbf{w}}$$

- 代入 $e(\mathbf{w}) = d - \mathbf{x}^T \mathbf{w}$

$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{x}, \quad \frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{x}e(\mathbf{w})$$

Least-Mean-Square (LMS) Algorithm

- 将以上代入steepest descent rule, 即得LMS Algorithm

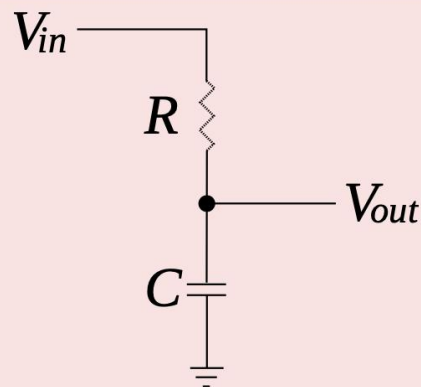
$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \eta \mathbf{x}_n e_n$$

注意： 权重更新只针对一个训练样本(\mathbf{x}_i, d_i)

LMS的特性：优势

- 简单易部署，不依赖于模型 (robust)
- LMS algorithm的本质是一种low-pass filter
 - 只通过错误信号的低频部分来抑制高频部分的影响

Low-Pass filter



LMS的特性：缺陷

- 由于只使用单个样本进行更新，则梯度方向不一定符合steepest descent
- 收敛很慢
- 对输入的相关矩阵（correlation matrix）的条件数（最大和最小的特征值之间的比值）敏感
- LMS的可收敛范围定义如下， η 是学习率， λ_{\max} 是最大的特征值

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

- 对证明过程感兴趣的同学，可以参考
- Simon Haykin - Adaptive Filter Theory (3rd Edition)

LMS用于感知机中

- 在感知机的更新中，使用LMS的代价函数 $\mathcal{E}(\mathbf{w}) = \frac{1}{2} e^2(\mathbf{w})$ ，则有

$$\Delta \mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) = -\eta \mathbf{g}(n) = \eta \left((d_j - y_j(n)) \right) \mathbf{x}(n)$$

- 提示： $\mathbf{g}(n) = \nabla \mathcal{E}(\mathbf{w}) = -\mathbf{x}(n)e(\mathbf{x}(n)) = -\mathbf{x}(n)(d - \mathbf{x}^T \mathbf{w}) = \mathbf{x}(n) \left((d_j - y_j(n)) \right)$

感知机的学习过程 (Batch Learning)

1. 设 $n = 0$ (iteration轮数)
2. 设 $d_j = \begin{cases} +1, & \text{if } x_j(n) \in \text{Class } 1 \\ -1, & \text{if } x_j(n) \in \text{Class } 2 \end{cases}$, for all $j = 1, 2, \dots, m$.
3. 初始化权重, $w^T = (w_1(n), w_2(n), \dots, w_m(n))$
4. 初始化目标值来启动循环, $y^T = \langle y_1(n), y_2(n), \dots, y_m(n) \rangle$
5. 初始化Stopping Error $\epsilon > 0$
6. 初始化学习率 η

感知机的学习过程 (Batch Learning)

While $\frac{1}{m} \sum_{j=1}^m \|d_j - y_j(n)\| > \epsilon$

For each sample (x_j, d_j) for $j = 1, 2, \dots, m$:

Calculate output $y_j = \varphi(\mathbf{w}^T(n) \cdot \mathbf{x}_j)$

Update weights $w_i(n+1) = w_i(n) + \eta \left((d_j - y_j(n)) \right) \mathbf{x}(n)$

$n = n + 1$

单层感知机的目标

将一系列样本 $\{X_1, X_2, \dots, X_m\}$, 正确的分成 C_1 和 C_2 两类

类 C_1 的输出为 $y = +1$

类 C_2 的输出为 $y = -1$

感知机的收敛定理

If we assume

Linear Separability for the classes C_1 and C_2 .

Rosenblatt - 1962

Let the subsets of training vectors C_1 and C_2 be linearly separable. Let the inputs presented to the perceptron originate from these two subsets. The perceptron converges after some n_0 iterations, in the sense that is a solution vector for

$$\mathbf{w}(n_0) = \mathbf{w}(n_0 + 1) = \mathbf{w}(n_0 + 2) = \dots \quad (35)$$

is a solution vector for $n_0 \leq n_{max}$

感知机的收敛定理: Proof 1

初始化

$$\boldsymbol{w}(0) = 0 \quad (36)$$

设在时间 $n = 1, 2, 3, \dots$ 时

$$\boldsymbol{w}^T(n) \boldsymbol{x}(n) < 0 \quad (37)$$

with $\boldsymbol{x}(n)$ belongs to class C_1 .

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS

$\boldsymbol{x}(1), \boldsymbol{x}(2), \dots$

代入感知机的纠错规则 (仅在分类错误时更新)

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{x}(n) \quad (38)$$

感知机的收敛定理: Proof 2

迭代使用correction formula得

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + \dots + \boldsymbol{x}(n) \quad (39)$$

由线性可分, 可知一定有解 \boldsymbol{w}_0 使得

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} \boldsymbol{w}_0^T \boldsymbol{x}(n) \quad (40)$$

对 (39) 两边乘 \boldsymbol{w}_0^T 得

$$\boldsymbol{w}_0^T \boldsymbol{w}(n+1) = \boldsymbol{w}_0^T \boldsymbol{x}(1) + \boldsymbol{w}_0^T \boldsymbol{x}(2) + \dots + \boldsymbol{w}_0^T \boldsymbol{x}(n) \quad (41)$$

感知机的收敛定理：Proof 3

将 (40) 带入 (41) 可得

$$\mathbf{w}_0^T \mathbf{w}(n+1) \geq n\alpha \quad (42)$$

使用 Cauchy-Schwartz 不等式得, $\|\cdot\|$ 是欧氏距离

$$\left\| \mathbf{w}_0^T \right\|^2 \|\mathbf{w}(n+1)\|^2 \geq \left[\mathbf{w}_0^T \mathbf{w}(n+1) \right]^2 \quad (43)$$

将 (42) 带入 (43) 得

$$\begin{aligned} \left\| \mathbf{w}_0^T \right\|^2 \|\mathbf{w}(n+1)\|^2 &\geq n^2 \alpha^2 \\ \|\mathbf{w}(n+1)\|^2 &\geq \frac{n^2 \alpha^2}{\left\| \mathbf{w}_0^T \right\|^2} \end{aligned}$$

感知机的收敛定理：Proof 4

对 (38) 式，两边各求欧式范数得

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \quad (44)$$

由 (37) 式得

$$\begin{aligned} \|\mathbf{w}(k+1)\|^2 &\leq \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 \\ \|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 &\leq \|\mathbf{x}(k)\|^2 \end{aligned} \quad (45)$$

感知机的收敛定理：Proof 5

对 (45) 利用裂项求和技巧可得

$$\sum_{k=0}^n \left[\|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \right] \leq \sum_{k=0}^n \|\mathbf{x}(k)\|^2 \quad (46)$$

假设

$$\begin{aligned} \mathbf{w}(0) &= \mathbf{0} \\ \mathbf{x}(0) &= \mathbf{0} \end{aligned}$$

则 (46) 可化简为

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{k=1}^n \|\mathbf{x}(k)\|^2 \quad (47)$$

感知机的收敛定理：Proof 6

定义一个正数

$$\beta = \max_{\mathbf{x}(k) \in C_1} \|\mathbf{x}(k)\|^2 \quad (48)$$

将 (48) 带入 (47) 则有

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{k=1}^n \|\mathbf{x}(k)\|^2 \leq n\beta$$

上式当存在 n_{max} 时成立

$$\frac{n_{max}^2 \alpha^2}{\|\mathbf{w}_0\|^2} = n_{max} \beta \quad (49)$$

感知机的收敛定理: Proof 7

在 (49) 中, 对 n_{max} 求解, 得

$$n_{max} = \frac{\beta \|w_0\|^2}{\alpha^2} \quad (50)$$

结论:

For $\eta(n) = 1$ for all n , $w(0) = \mathbf{0}$ and a solution vector w_0 :

权重更新在至多 n_{max} 步后停止

注意: w_0 并不是唯一解

单层感知机：示例

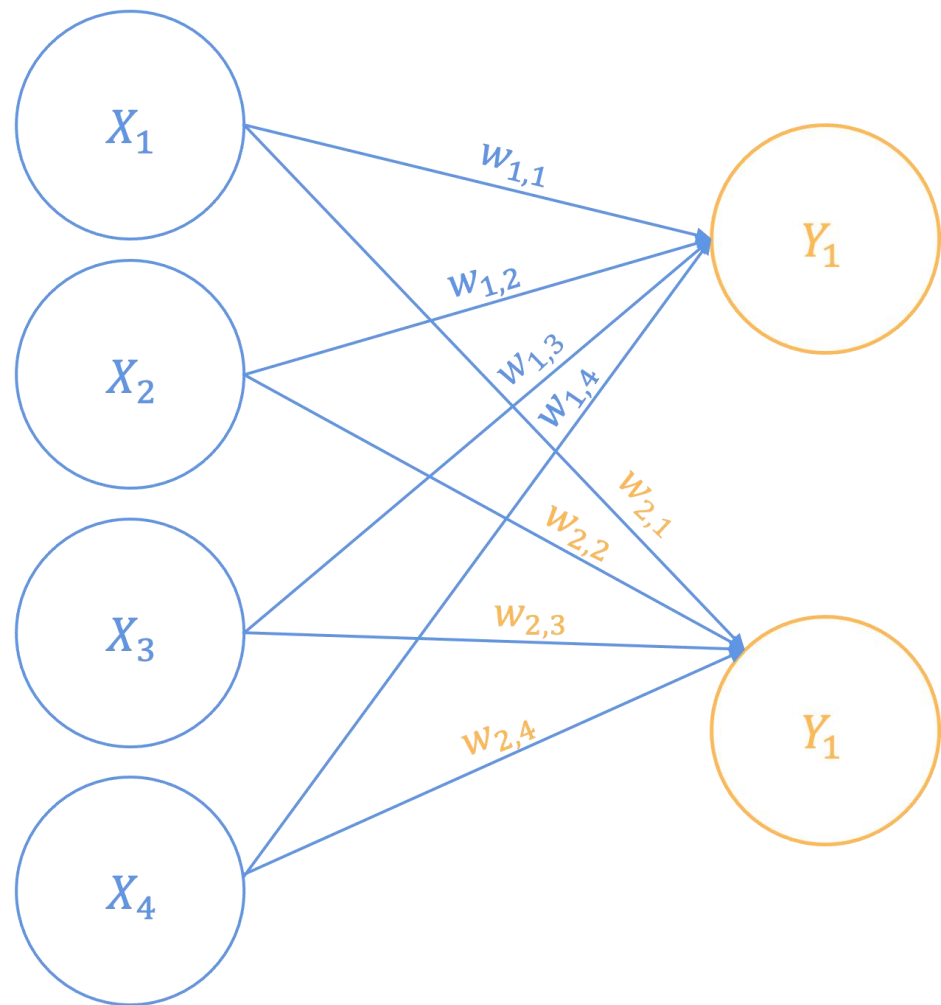
让我们回顾本章一开始提到的“多输入——多输出”的问题。

分类问题：多输入——多输出

- 示例：利用身高（height: cm）、体重（weight: kg）、年龄（age: years）、受教育水平（education level）去预测一个人是否存在危害身体健康的肥胖症状（obesity）和收入状况（income level: 1000人民币/每月）（假设该问题线性可分）
- 身高： X_1 ； 体重： X_2 ； 年龄： X_3 ；
- 受教育水平： X_4 (0:小学及以下, 1:初中, 2:高中, 3:大学及以上)
- 肥胖： Y_1 (0 :否; 1 :是)
- 收入情况： Y_2 (0: 0-10; 1 : 10-50, 2: 50-100, 3: >100)
- 样本 (X, Y) , $X = (X_1, X_2, X_3, X_4)$ 其中 $X_1, X_2, X_3 \in \{0, 200\}$, $X_4 \in \{0, 1, 2, 3\}$
- $Y = (Y_1, Y_2)$, $Y_1 \in \{0, 1\}$, $Y_2 \in \{0, 1, 2, 3\}$

单层感知机：示例

如果我们用一个4个输入神经元，2个输出神经元来解决这个问题。我们将会得到右图这样一个模型（隐去了偏置项）

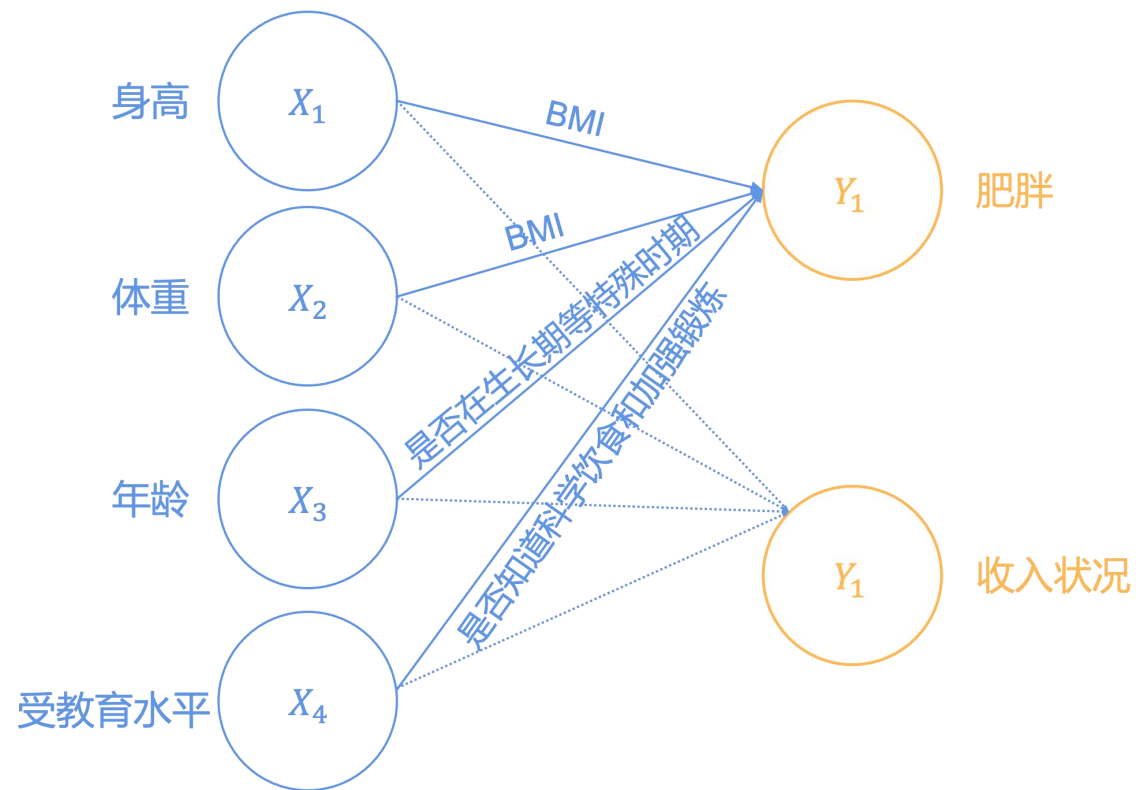


单层感知机：示例

我们现在假设这个问题是线性可分的问题，
那么也就是有最优解的。

每个权重代表什么意思呢？

首先看第一个输出神经元相关的权重分析

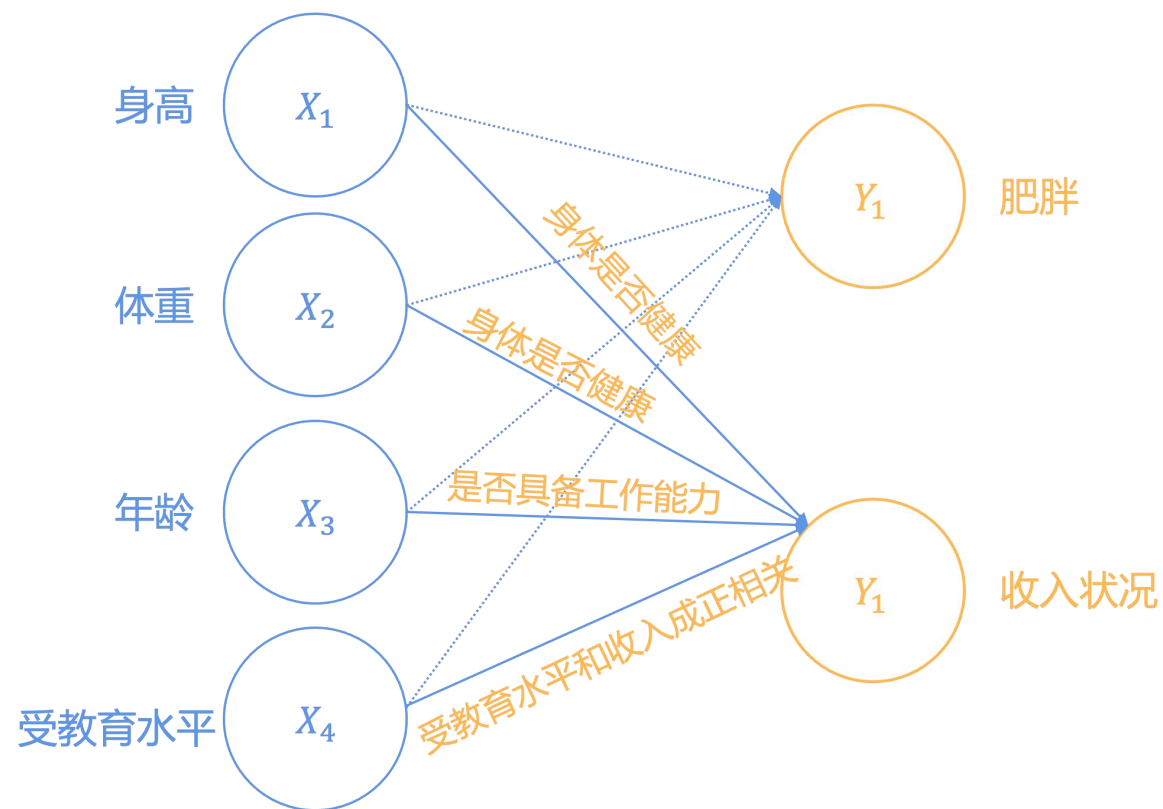


单层感知机：示例

我们现在假设这个问题是线性可分的问题，
那么也就是有最优解的。

每个权重代表什么意思呢？

再看第二个输出神经元相关的权重分析



单层感知机：示例

- 观察这些权重我们可以发现，有一些权重可能代表重复的意思。如果我们手工去设计权重，我们很可能只会建立一些有先验知识支撑的连接，而舍弃重复的，或者不重要的连接
- 感知机采用的是全连接 “fully-connected”，我们不再需要先验知识，就可以学到一个分类器，代价是，我们并不知道每个权重到底代表了什么

04

单层感知机的应用

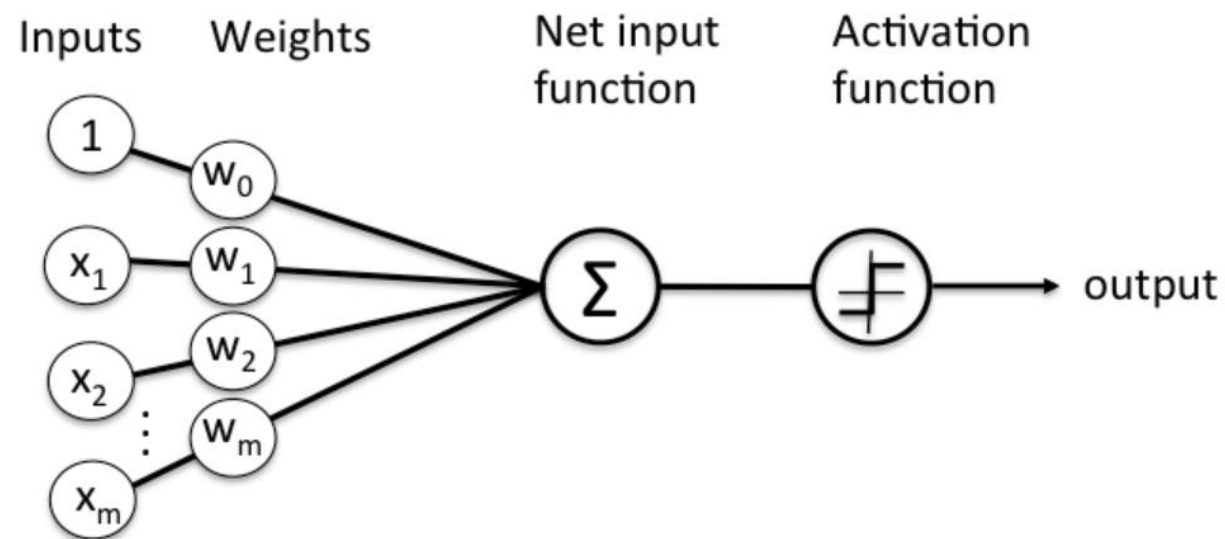
重点：单层感知机虽然只能够解决线性可分的问题（Linearly-Separable Problem），但相比单个神经元，它可以解决多输入多输出的问题。

单层感知机的应用

- 本节将用代码形式和大家展示单层感知机。虽然感知机可以解决多输入多输出的问题，但这类问题在输出高维的情况下，通常不会线性可分^{*}，因此超出了单层感知机的作用范围。大家根据课件中的例子有理论认知即可。
- 本节主要为大家介绍以下两个例子
- **示例1**: 感知机的标准形式
- **示例2**: delta rule感知机
- ^{*} 多输出的线性可分：对于每个输出，样本都可以用直线分开

示例1：hard margin

- GOAL：用花萼 (*sepal*) 和花瓣 (*petal*) 的长度去分类花的品种 (*setosa and versicolor*)
- 数据集： Iris 数据集 来自 [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)
- <https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>



感知机模型

```
import numpy as np
```

```
class Perceptron(object):
```

```
    def __init__(self, eta=0.01, epochs=50):
```

```
        self.eta = eta
```

```
        self.epochs = epochs
```

```
    def train(self, X, y):
```

```
        self.w_ = np.zeros(1 + X.shape[1])
```

3.初始化权重

```
        self.errors_ = []
```

4.初始化误差个数

```
        for _ in range(self.epochs):
```

```
            errors = 0
```

```
            for xi, target in zip(X, y):
```

5.遍历所有样本

```
                update = self.eta * (target - self.predict(xi))
```

求解 Δw

思考题—

```
                self.w_[1:] += update * xi
```

```
                self.w_[0] += update
```

更新权重

```
                errors += int(update != 0.0)
```

记录误差

```
            self.errors_.append(errors)
```

```
        return self
```

```
    def net_input(self, X):
```

1.对输入的样本求和

```
        return np.dot(X, self.w_[1:]) + self.w_[0]
```

```
    def predict(self, X):
```

2.使用阈值函数将求和结果映射到1或-1

```
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```



权重更新公式

$$\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$$

权重更新分量公式

$$\Delta w_0 = \eta (\text{target}^{(i)} - \text{output}^{(i)})$$

$$\Delta w_1 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_1^{(i)}$$

$$\Delta w_2 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_2^{(i)}$$

权重更新：正确分类

$$\Delta w_j = \eta (-1^{(i)} - -1^{(i)}) x_j^{(i)} = 0$$

$$\Delta w_j = \eta (1^{(i)} - 1^{(i)}) x_j^{(i)} = 0$$

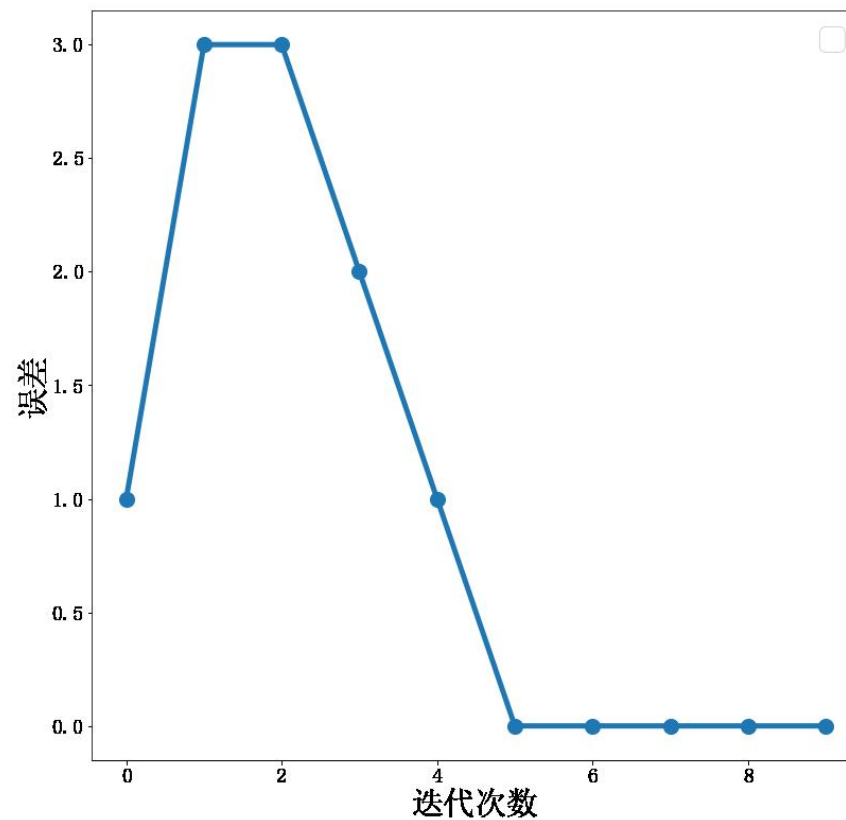
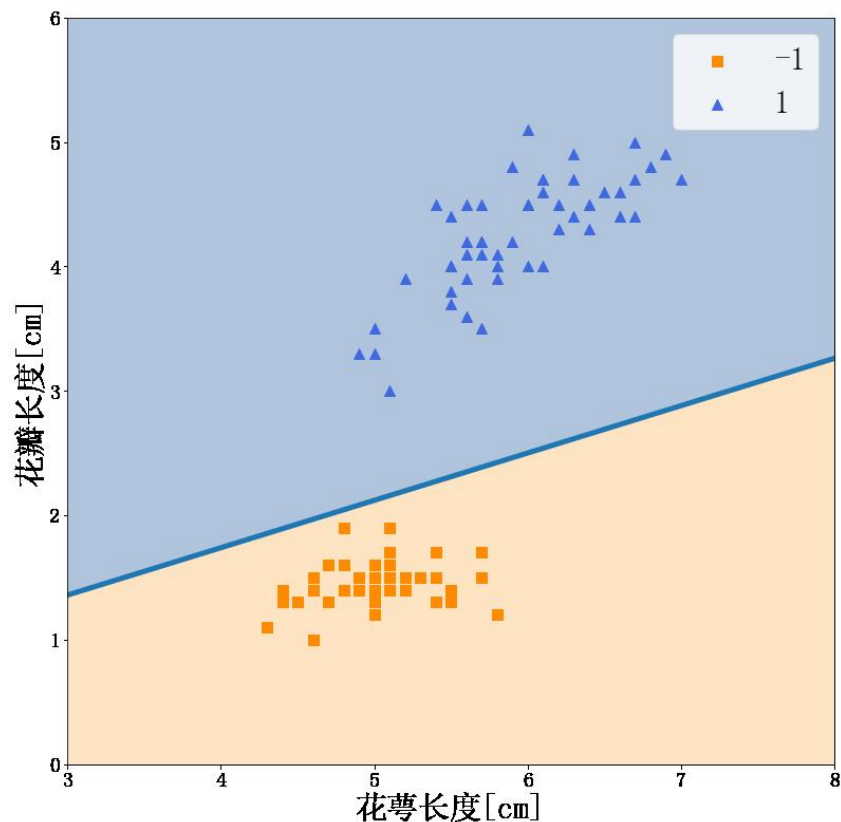
权重更新：错误分类

$$\Delta w_j = \eta (1^{(i)} - -1^{(i)}) x_j^{(i)} = \eta (2) x_j^{(i)}$$

$$\Delta w_j = \eta (-1^{(i)} - 1^{(i)}) x_j^{(i)} = \eta (-2) x_j^{(i)}$$

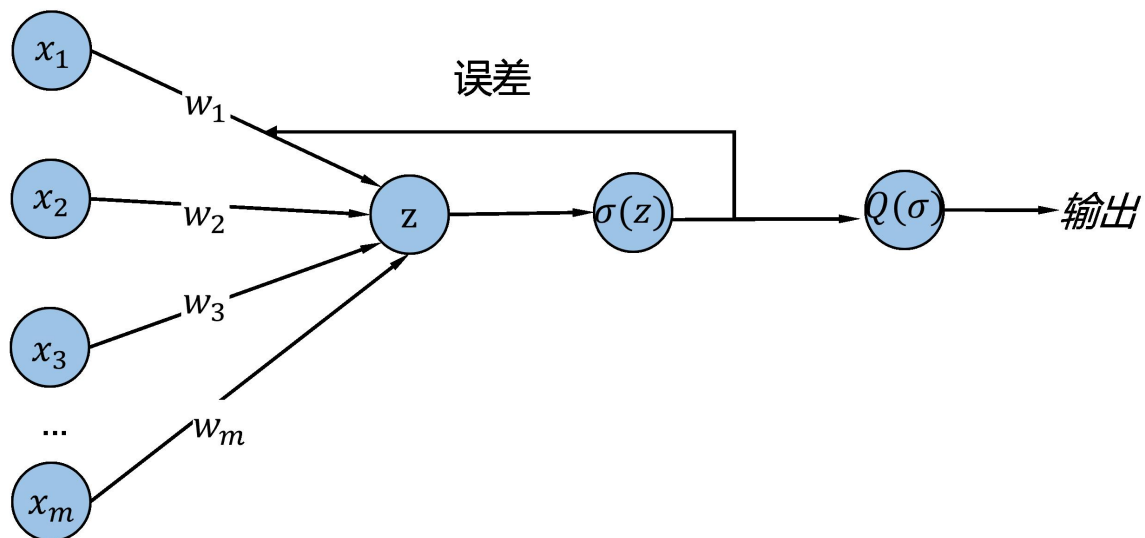
思考题：权重更新有没有更简洁写法？（提示：张量表示）

分类结果



思考题：单层感知机会存在什么问题？
提示：线性可分前提

输入 权值 加权求和 激活函数 量化器



使用delta rule的感知机

重点：使用了梯度下降（Gradient Descent）学习规则

```
import numpy as np
```

```
class AdalineGD(object):
```

```
    def __init__(self, eta=0.01, epochs=50):
```

```
        self.eta = eta
```

```
        self.epochs = epochs
```

```
    def train(self, X, y):
```

```
        self.w_ = np.zeros(1 + X.shape[1])
```

```
        self.cost_ = []
```

```
        for i in range(self.epochs):
```

```
            output = self.net_input(X)
```

```
            errors = (y - output)
```

```
            self.w_[1:] += self.eta * X.T.dot(errors)
```

```
            self.w_[0] += self.eta * errors.sum()
```

```
            cost = (errors**2).sum() / 2.0
```

```
            self.cost_.append(cost)
```

```
        return self
```

```
    def net_input(self, X):
```

```
        return np.dot(X, self.w_[1:]) + self.w_[0]
```

```
    def activation(self, X):
```

```
        return self.net_input(X)
```

```
    def predict(self, X):
```

```
        return np.where(self.activation(X) >= 0.0, 1, -1)
```

损失函数

$$J(\mathbf{w}) = \frac{1}{2} \sum_i (\text{target}^{(i)} - \text{output}^{(i)})^2 \quad \text{output}^{(i)} \in \mathbb{R}$$

计算梯度

$$\begin{aligned} \frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (t^{(i)} - o^{(i)})^2 \\ &= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)})^2 \\ &= \frac{1}{2} \sum_i 2(t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)}) \\ &= \sum_i (t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} \left(t^{(i)} - \sum_j w_j x_j^{(i)} \right) \\ &= \sum_i (t^{(i)} - o^{(i)}) (-x_j^{(i)}) \end{aligned}$$

(t = target, o = output)

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = -\eta \sum_i (t^{(i)} - o^{(i)}) (-x_j^{(i)}) = \eta \sum_i (t^{(i)} - o^{(i)}) x_j^{(i)},$$

权重更新

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}.$$

```
import numpy as np
```

```
class AdalineGD(object):
```

```
    def __init__(self, eta=0.01, epochs=50):
```

```
        self.eta = eta
```

```
        self.epochs = epochs
```

```
    def train(self, X, y):
```

```
        self.w_ = np.zeros(1 + X.shape[1])
```

```
        self.cost_ = []
```

```
        for i in range(self.epochs):
```

```
            output = self.net_input(X)
```

```
            errors = (y - output)
```

```
            self.w_[1:] += self.eta * X.T.dot(errors)
```

```
            self.w_[0] += self.eta * errors.sum()
```

```
            cost = (errors**2).sum() / 2.0
```

```
            self.cost_.append(cost)
```

```
        return self
```

```
    def net_input(self, X):
```

```
        return np.dot(X, self.w_[1:]) + self.w_[0]
```

```
    def activation(self, X):
```

```
        return self.net_input(X)
```

```
    def predict(self, X):
```

```
        return np.where(self.activation(X) >= 0.0, 1, -1)
```

看起来好像没什么差别？


```
import numpy as np
```

```
class AdalineGD(object):
```

```
    def __init__(self, eta=0.01, epochs=50):
```

```
        self.eta = eta
```

```
        self.epochs = epochs
```

```
    def train(self, X, y):
```

```
        self.w_ = np.zeros(1 + X.shape[1])
```

```
        self.cost_ = []
```

```
        for i in range(self.epochs):
```

```
            output = self.net_input(X)
```

```
            errors = (y - output)
```

```
            self.w_[1:] += self.eta * X.T.dot(errors)
```

```
            self.w_[0] += self.eta * errors.sum()
```

```
            cost = (errors**2).sum() / 2.0
```

```
            self.cost_.append(cost)
```

```
        return self
```

```
    def net_input(self, X):
```

```
        return np.dot(X, self.w_[1:]) + self.w_[0]
```

```
    def activation(self, X):
```

```
        return self.net_input(X)
```

```
    def predict(self, X):
```

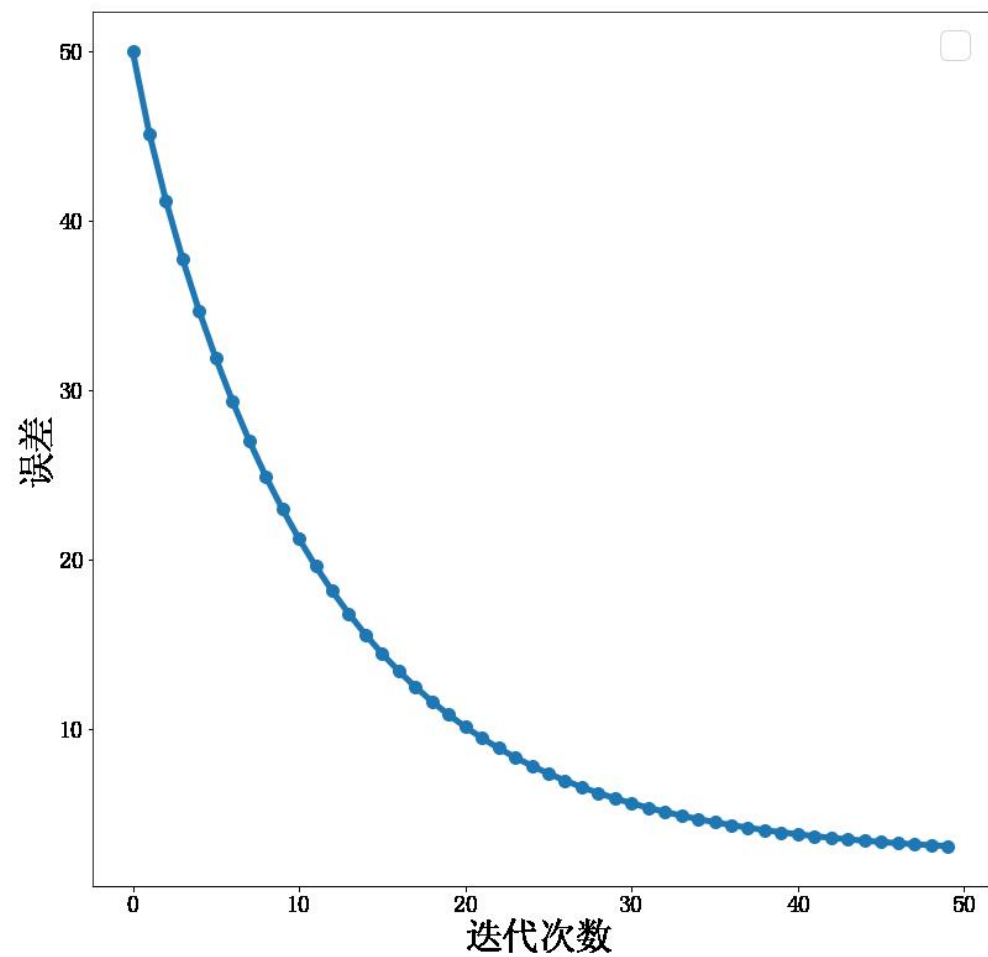
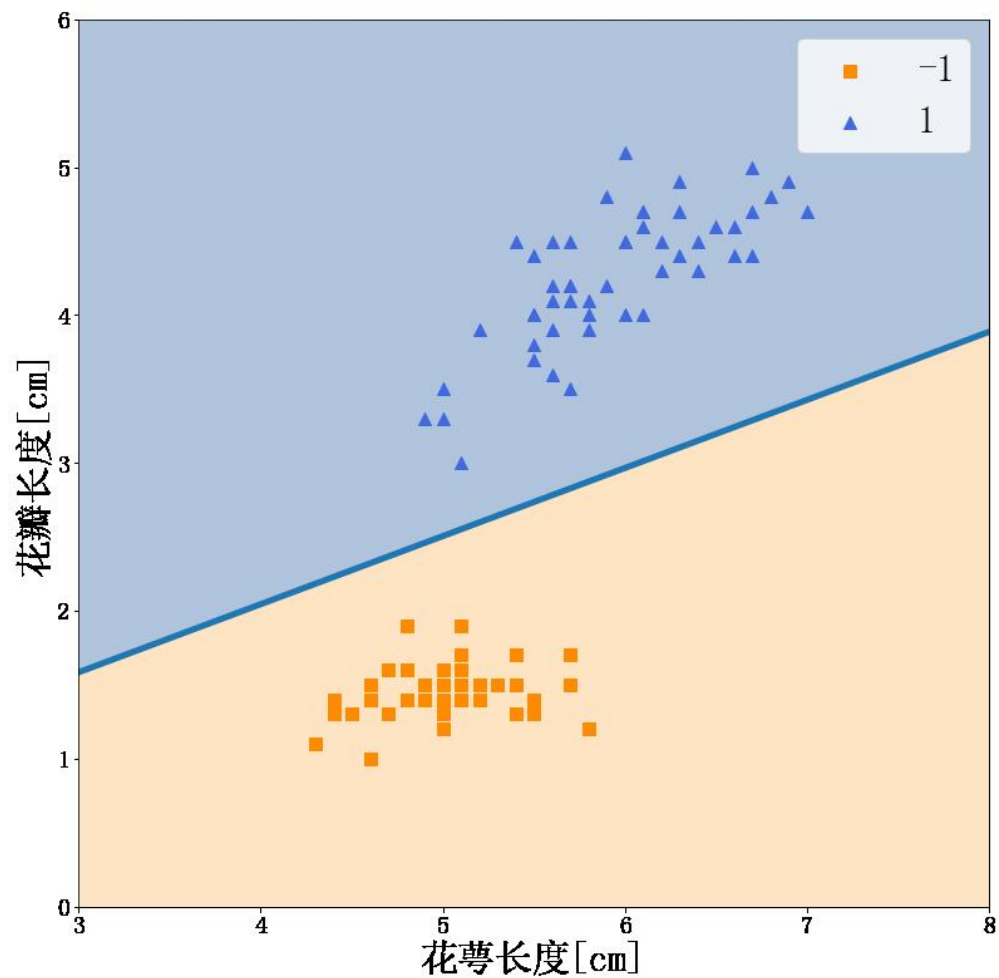
```
        return np.where(self.activation(X) >= 0.0, 1, -1)
```

差别在于两点

- 这里的“o”是一个实数，而不是之前所用的一个类别标签。
- 权重更新是直接作用于所有样本，而不同于之前的逐个样本更新，这种方法也叫做“批量梯度下降” (batch gradient descent)

这种“梯度下降”的思想，是一个重要的知识点，之后会详细讲解。

分类结果



基于单层神经网络的机器学习

多输出线性回归

SoftmaxRegression多分类

多输出线性回归

多输出线性回归的数学表达式涉及多个因变量和多个自变量，可以表示为一组线性方程，每个方程对应一个因变量。

将输入写作向量形式 x ，输出层第 i 个神经元的权重记为 w_i ，无激活函数，输出记为 o_i 。

此时的神经网络可以写成：

$$\begin{cases} o_1 = w_1^T x \\ o_2 = w_2^T x \\ \dots \\ o_m = w_m^T x \end{cases}$$

或者写成更简单的矩阵形式 $o = W^T x$

多输出线性回归

此时, 对于训练集 $\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$, 尝试习得

$$\mathbf{o}_j = \mathbf{W}^T \mathbf{x}_j$$

使得

$$\mathbf{o}_j \simeq \mathbf{y}_j, j = 1, 2, \dots, n$$

多输出线性回归

使用最小二乘法来求解，使平方误差最小化。

$$L = \sum_{i=1}^n \|\mathbf{y}_i - \mathbf{o}_i\|^2 = \|\mathbf{Y} - \mathbf{O}\|_F$$

求解的方法与单目标线性回归相同

$$\frac{\partial L}{\partial \mathbf{W}} = 2\mathbf{X}^T(\mathbf{W}^T\mathbf{X} - \mathbf{Y})$$

令其为零可求得闭式解。当 $\mathbf{X}^T\mathbf{X}$ 满秩或正定时，解得

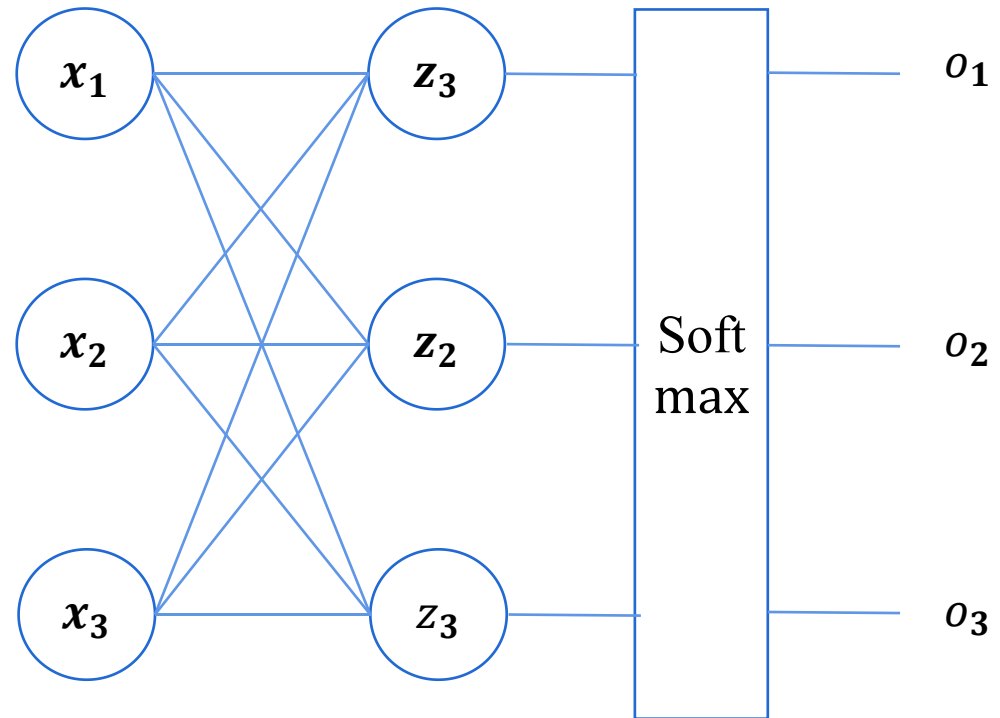
$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Softmax Regression

- Softmax函数将来自多个输出层神经元的线性输出转化为多分类概率

- $$o_{i,j} = p(y_{i,j} = 1 | x_i)$$
$$= \text{softmax}(\mathbf{w}_j^T \mathbf{x}_i)$$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{i=1}^m e^{z_i}}$$



Softmax Regression

- 同样采用极大似然估计，当有以下数据时

$$\mathbf{y}_1 = (1, 0, 0) , \quad p(\mathbf{y}_1 | \mathbf{x}_1) = o_{1,1}$$

$$\mathbf{y}_2 = (0, 0, 1) , \quad p(\mathbf{y}_2 | \mathbf{x}_2) = o_{2,3}$$

$$\mathbf{y}_3 = (0, 1, 0) , \quad p(\mathbf{y}_3 | \mathbf{x}_3) = o_{3,2}$$

计算出出现样本数据的概率

$$p(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3 | \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = o_{1,1} * o_{2,3} * o_{3,2}$$

$$p(Y|X) = \prod_{i=1}^n \prod_{j=1}^m o_{i,j}^{y_{i,j}}$$

Softmax Regression

- 从最大似然概率引出损失函数：

$$L(X, Y) = -\log(p(Y|X)) = -\sum_{i=1}^n \sum_{j=1}^m y_{i,j} \log(o_{i,j})$$

同样的求导得到：

$$\frac{\partial L}{\partial \mathbf{w}_j} = \sum_{i=1}^n (o_{i,j} - y_{i,j}) \mathbf{x}_i$$

Softmax Regression

$$L(X, Y) = - \sum_{i=1}^n \sum_{j=1}^m y_{i,j} \log \left(\frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^m \exp(\mathbf{w}_k^T \mathbf{x}_i)} \right)$$

$$= - \sum_{i=1}^n \sum_{j=1}^m y_{i,j} \mathbf{w}_j^T \mathbf{x}_i + \sum_{i=1}^n \sum_{j=1}^m y_{i,j} \log \left(\sum_{k=1}^m \exp(\mathbf{w}_k^T \mathbf{x}_i) \right)$$

$$= - \sum_{i=1}^n \sum_{j=1}^m y_{i,j} \mathbf{w}_j^T \mathbf{x}_i + \sum_{i=1}^n \log \left(\sum_{k=1}^m \exp(\mathbf{w}_k^T \mathbf{x}_i) \right)$$

$$\frac{\partial L}{\partial \mathbf{w}_j} = - \sum_{i=1}^n y_{i,j} \mathbf{x}_i + \sum_{i=1}^n \frac{\exp(\mathbf{w}_j^T \mathbf{x}_i) \mathbf{x}_i}{\sum_{k=1}^m \exp(\mathbf{w}_k^T \mathbf{x}_i)}$$

$$= - \sum_{i=1}^n y_{i,j} \mathbf{x}_i + \sum_{i=1}^n o_{i,j} \mathbf{x}_i$$

$$= \sum_{i=1}^n (o_{i,j} - y_{i,j}) \mathbf{x}_i$$

Softmax Regression

神经元模型为 $o_{i,j} = p(y_{i,j} = 1 | \mathbf{x}_i) = \text{softmax}(\mathbf{w}_j^T \mathbf{x}_i)$

- 1) 初始化权重 \mathbf{w}
- 2) 根据训练数据集求误差

$$L(X, Y) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{i,j} \log(o_{i,j})$$

- 3) 求梯度并更新权重

$$\mathbf{w}_j = \mathbf{w}_j + \eta \frac{1}{n} \sum_{i=1}^n (y_{i,j} - o_{i,j}) \mathbf{x}_i$$

- 4) 是否满足终止条件？不满足的话跳回 (2)

Q&A

Questions and Answers