第三章：单层感知机

南京大学人工智能学院

申富饶

目录

CONTENTS

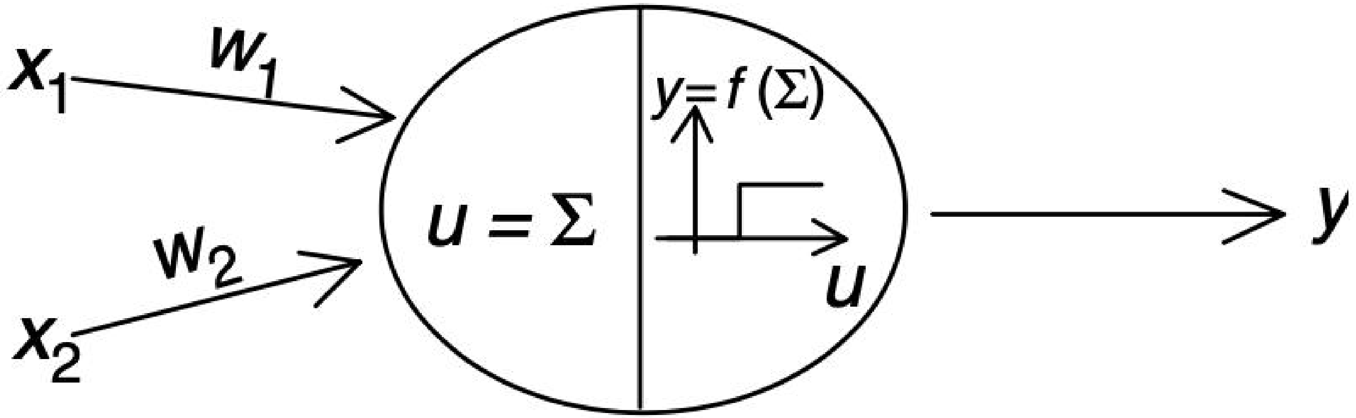
01

回顾：单个神经元

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，上一章讨论的其他类型的神经元（IC神经元等）不在本章的范围内。因此，我们默认神经元只能解决线性可分的问题。



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), $X = (X_1, X_2)$ $X_1, X_2 \in \{0, 200\}$, $Y \in \{0, 1\}$

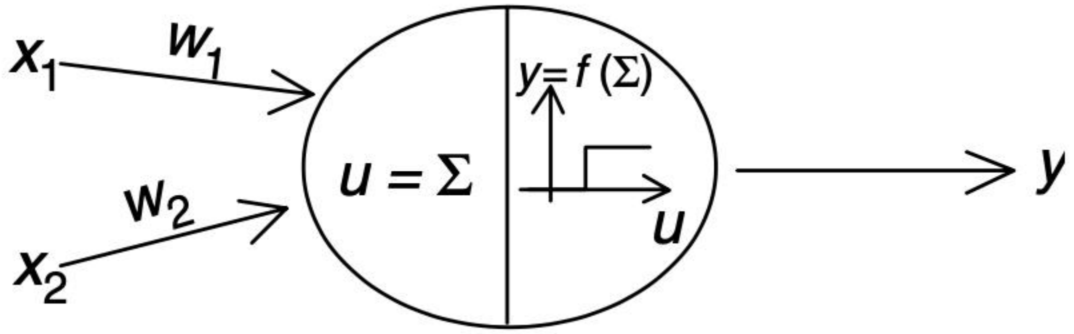
分类问题：多输入——单输出

- 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）

- 可以利用单个MP神经元来解决这个分类问题：

- $U = w_0 1 + w_1 X_1 + w_2 X_2$

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



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$X_1, X_2, X_3 \in \{0, 200\}, X_4 \in \{0, 1, 2, 3\}$
- $Y = (Y_1, Y_2), Y_1 \in \{0, 1\}, Y_2 \in \{0, 1, 2, 3\}$

分类问题：多输入——多输出

单个神经元，只能解决单输出问题。也就是说，只能预测肥胖f_1，或只能预测收入状况f_2

- $U = w_0 1 + w_1 X_1 + w_2 X_2 + w_3 X_3 + w_4 X_4$
- $f_1(U) = \begin{cases} 0, & U < 0 \\ 1, & U \geq 0 \end{cases}$
- $f_2(U) = \begin{cases} 0, & 0 \leq U < 10 \\ 1, & 10 \leq U < 50 \\ 2, & 50 \leq U < 100 \\ 3, & U \geq 100 \end{cases}$

分类+回归问题：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

- 样本(X, Y), $X = (X_1, X_2, X_3, X_4)$ 其中$X_1, X_2, X_3 \in \{0, 200\}, X_4 \in \{0, 1, 2, 3\}$
- $Y = (Y_1, Y_2), Y_1 \in \{0, 1\}, Y_2 \in \{0, 100\}$

分类+回归问题：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）

- 同样，单个神经元，只能解决单输出问题。也就是说，只能预测肥胖f_1，或只能预测收入状况f_2

- $U = w_0 1 + w_1 X_1 + w_2 X_2 + w_3 X_3 + w_4 X_4$
- $f_1(U) = \begin{cases} 0, & U < 0 \\ 1, & U \geq 0 \end{cases}$
- $f_2(U) = \begin{cases} 0, & U < 0 \\ U, & U \geq 0 \end{cases}$ (ReLu 函数)

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”的线性可分问题

由于结构限制，无论是单纯的分类问题，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

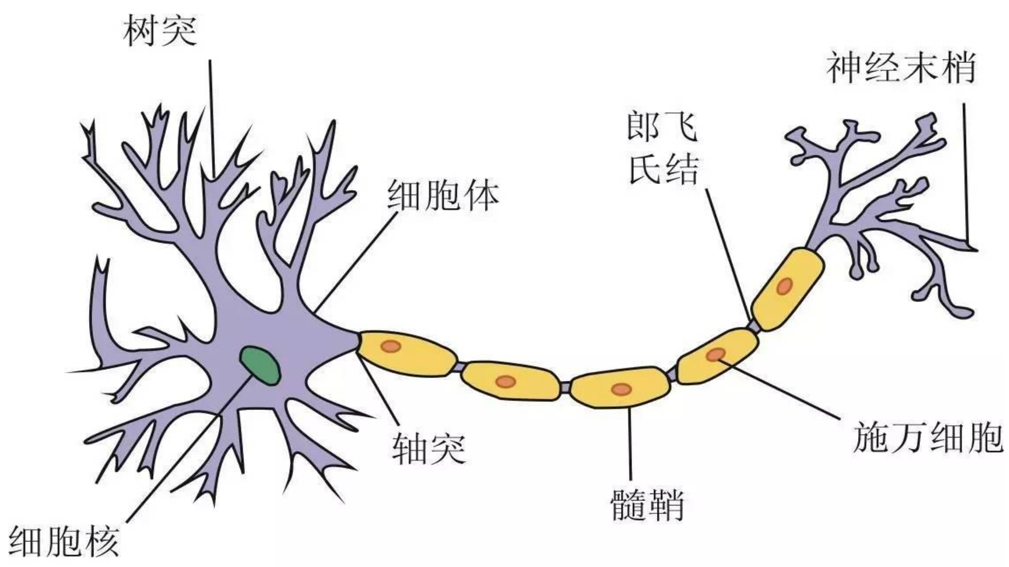
- 神经元通过连接交流信息，共同完成计算任务

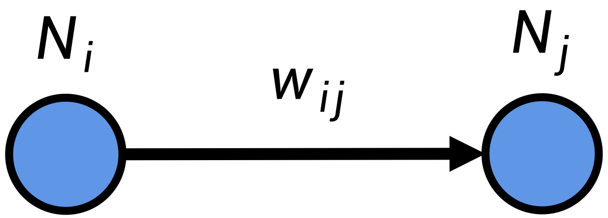
神经元的连接概述

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。



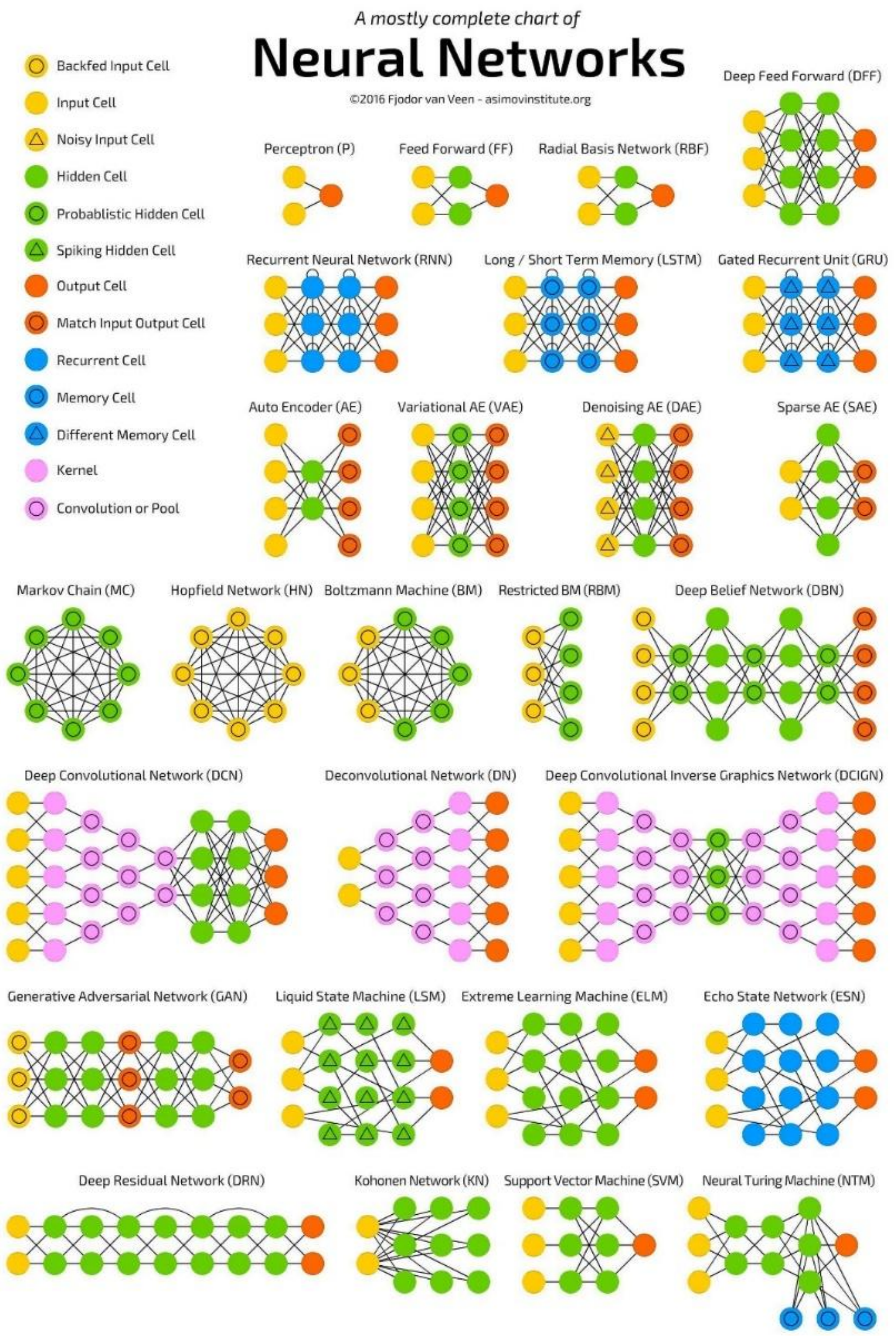


神经元的连接概述

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。

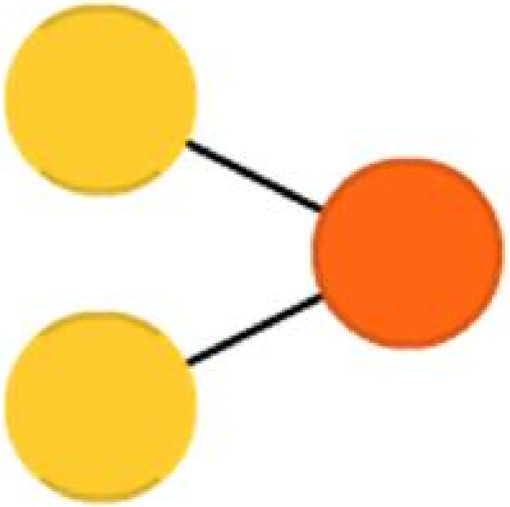
- 层级结构
- 互联网结构
- 等等

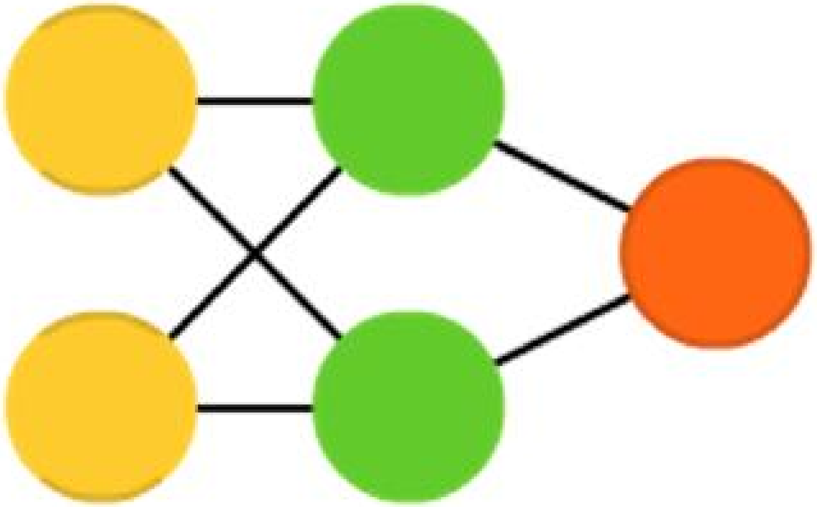


神经元的连接概述

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。

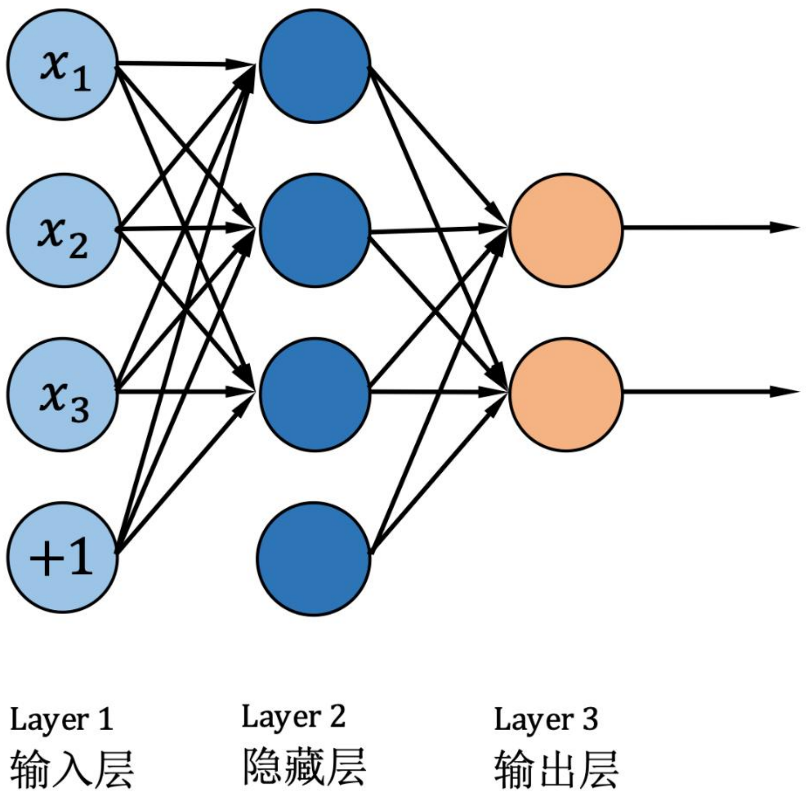




神经元的连接概述

层级结构

- 层级结构网络具有足够的隐藏神经元，理论上可以总是对输入和输出之间的关系建模

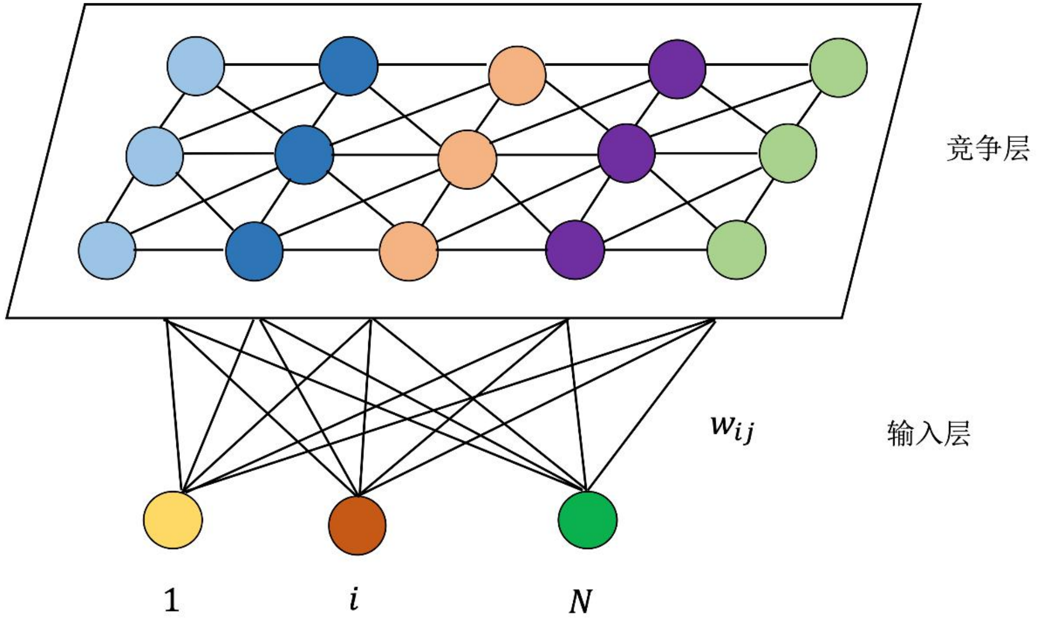


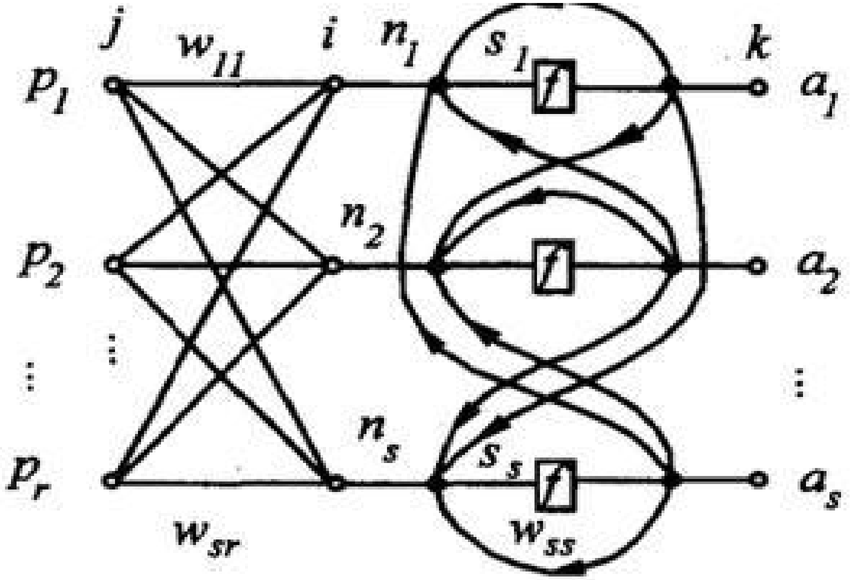
神经元的连接概述

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
- ➢ 实际应用中其应用是有限，通常将它们与其他网络结合形成新的网络

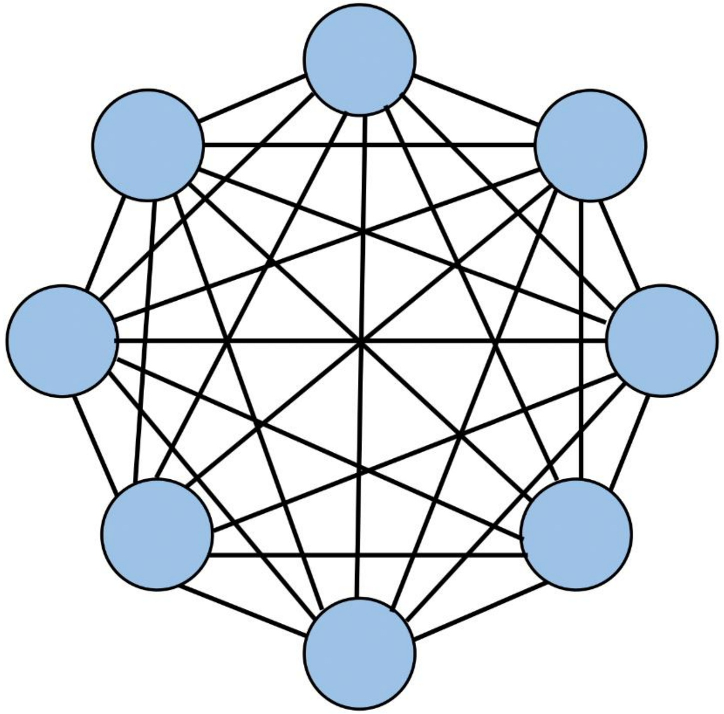




神经元的连接概述

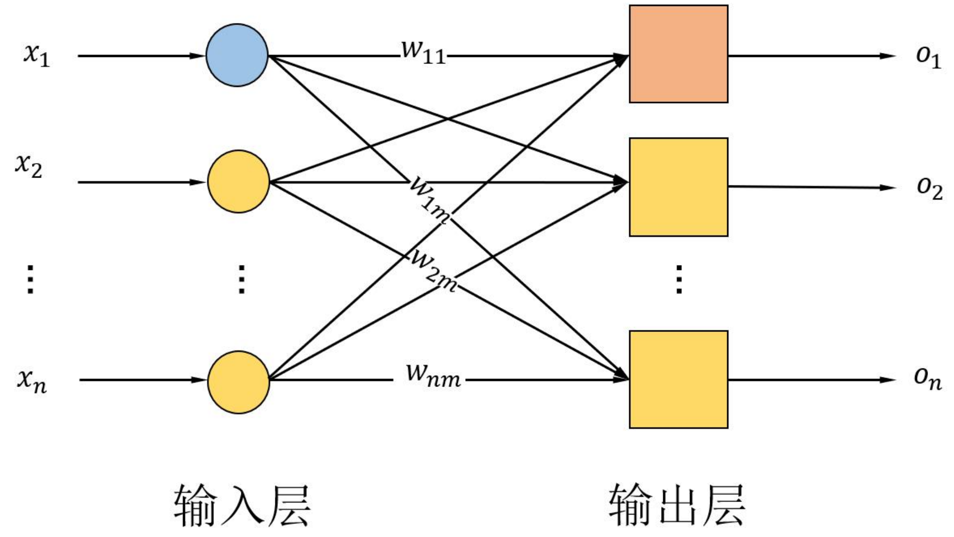
互联网型结构

- 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，但类似于神经网络，并且构成了BM和HNs的理论基础。



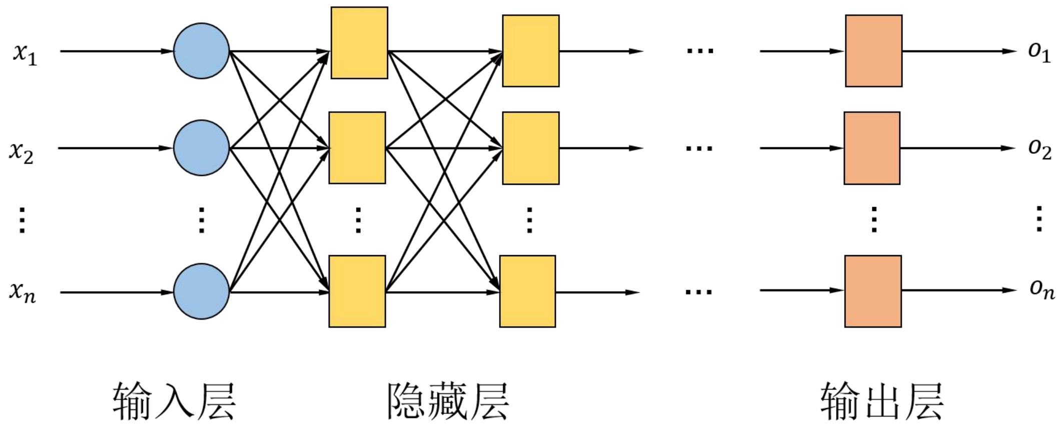
神经元的扩展

- 宽度



x_1
x_2
x_n
w_{11}
w_{1m}
w_{2m}
w_{nm}
o_1
o_2
o_n
输入层
输出层

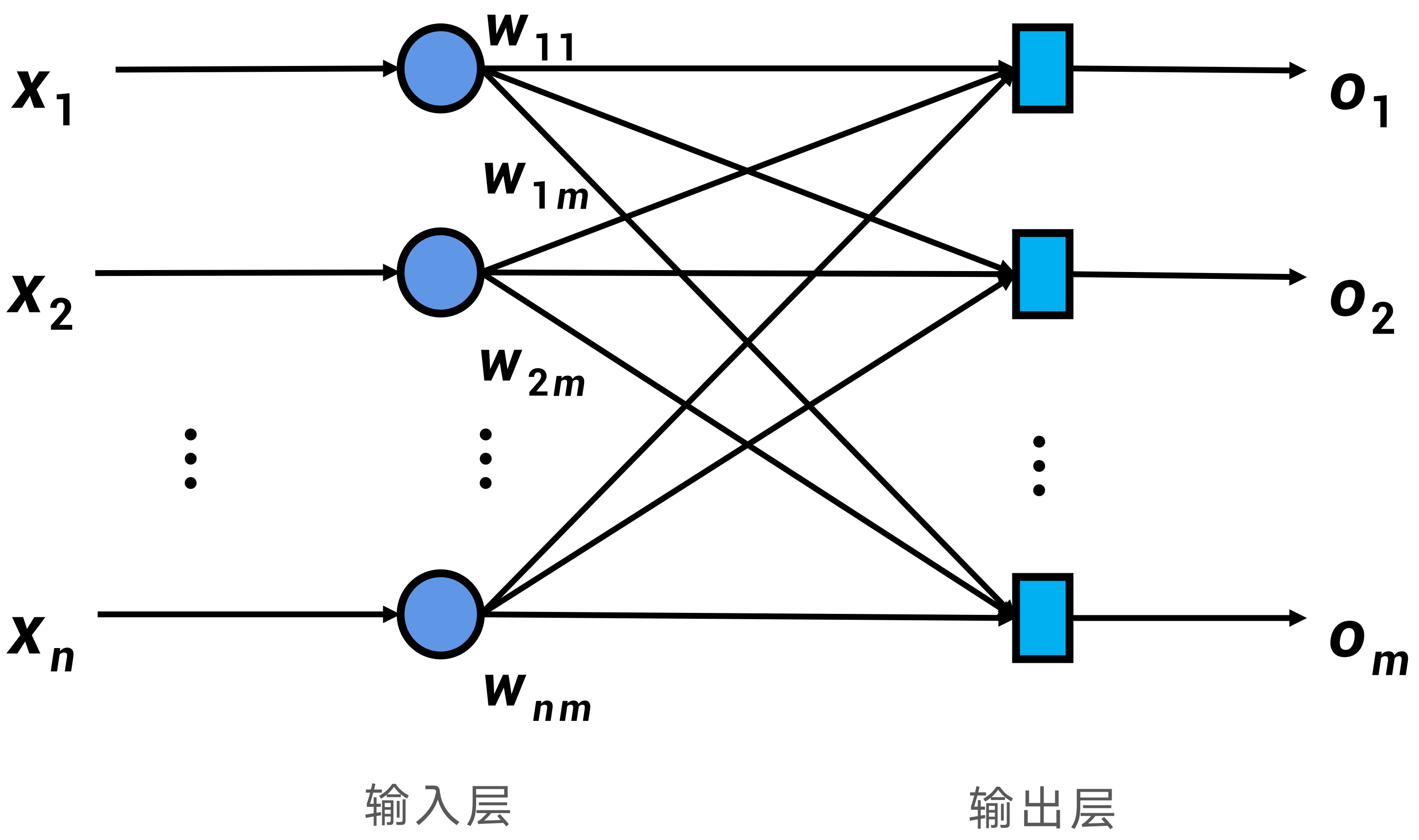
- 深度



x_1
x_2
x_n
o_1
o_2
o_n
输入层
隐藏层
输出层

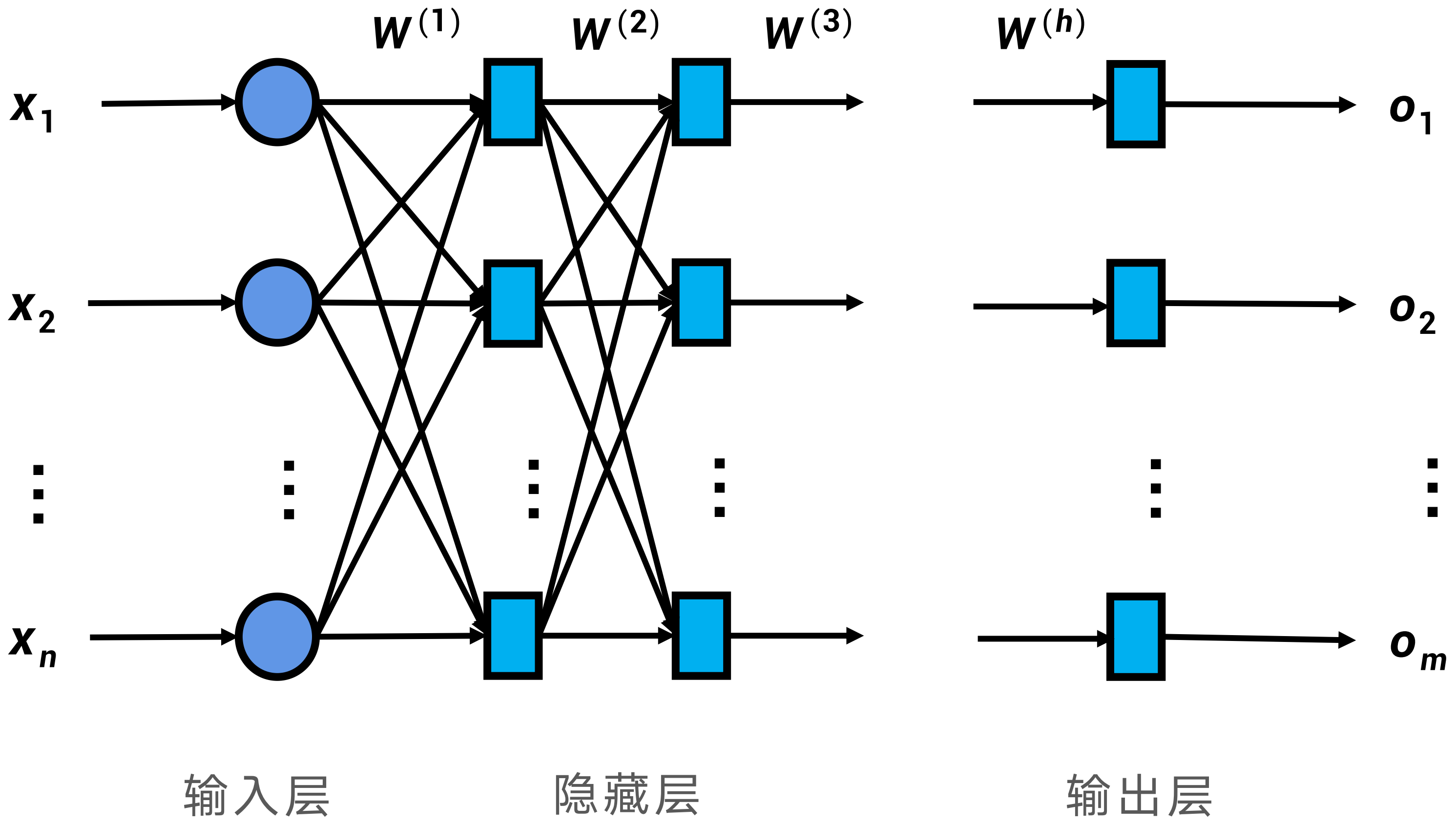
神经元的扩展：宽度扩展

多输出的单层感知机



神经元的扩展：深度扩展

多层感知机



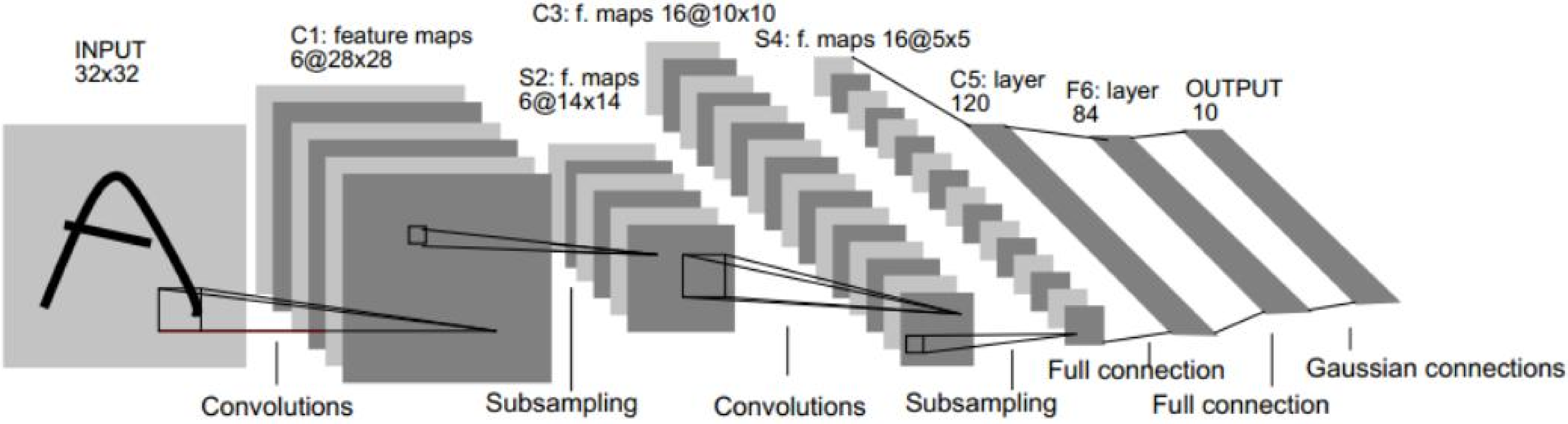
其他连接方式

- 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

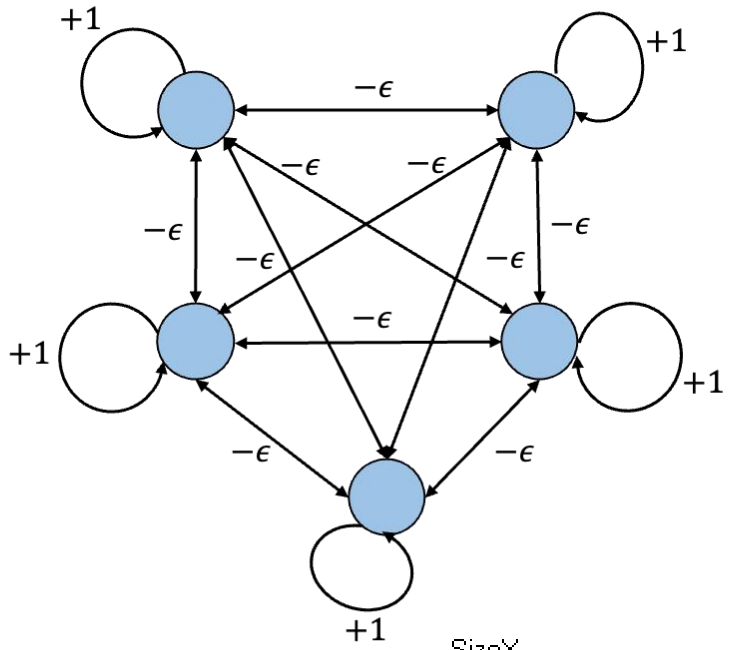
- 包含卷积计算且具有深度结构的前馈神经网络
- 主要结构：卷积层、池化层、全连接层

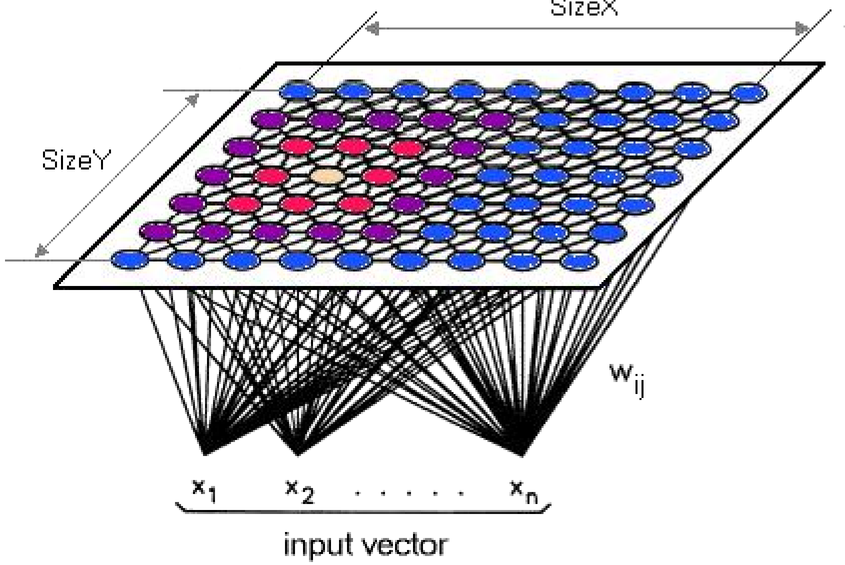


其他连接方式

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、SOM

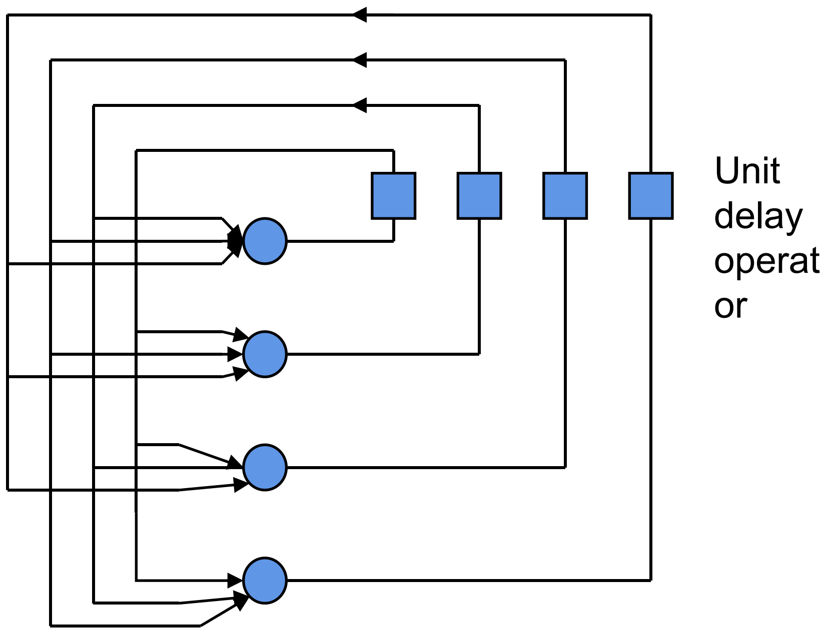




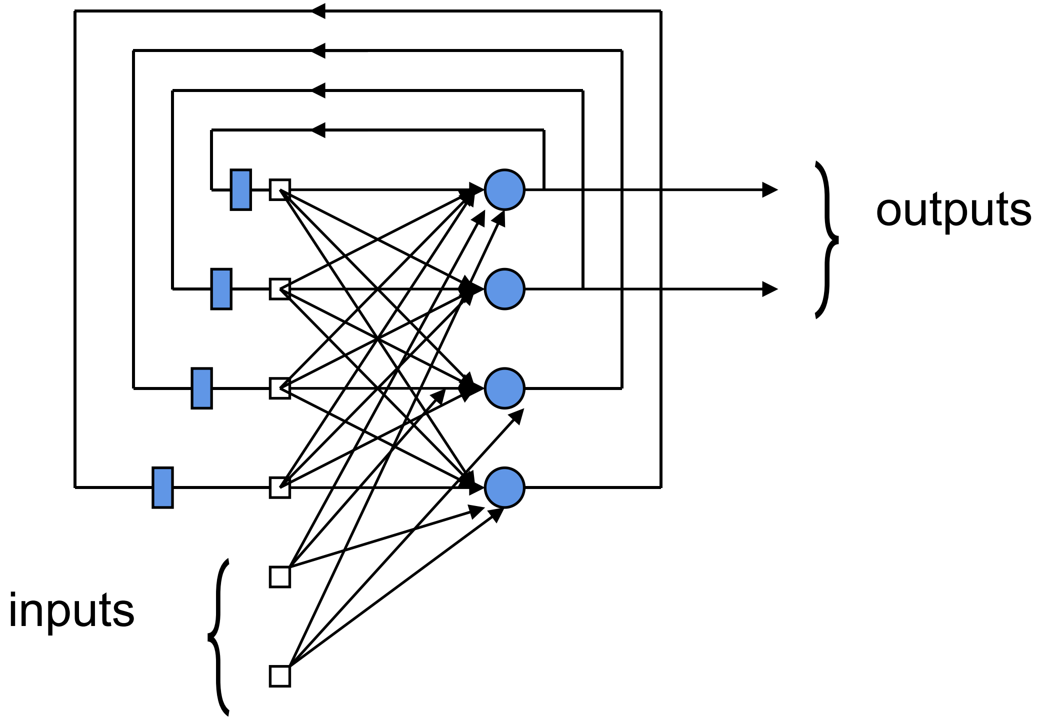
其他连接方式

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

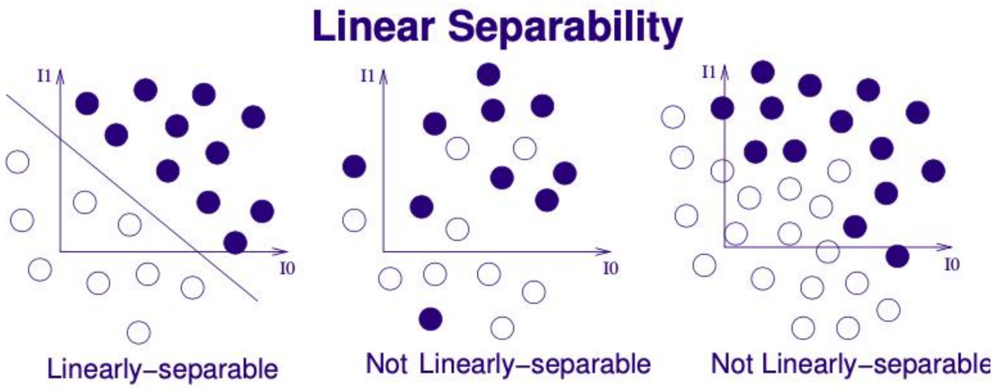
历史简介：从神经元到感知机

- 1943年，McCulloch and 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$y = wx + b$，其中是w权重， b是偏置
- 单层感知机的目标就是找到直线的权重。

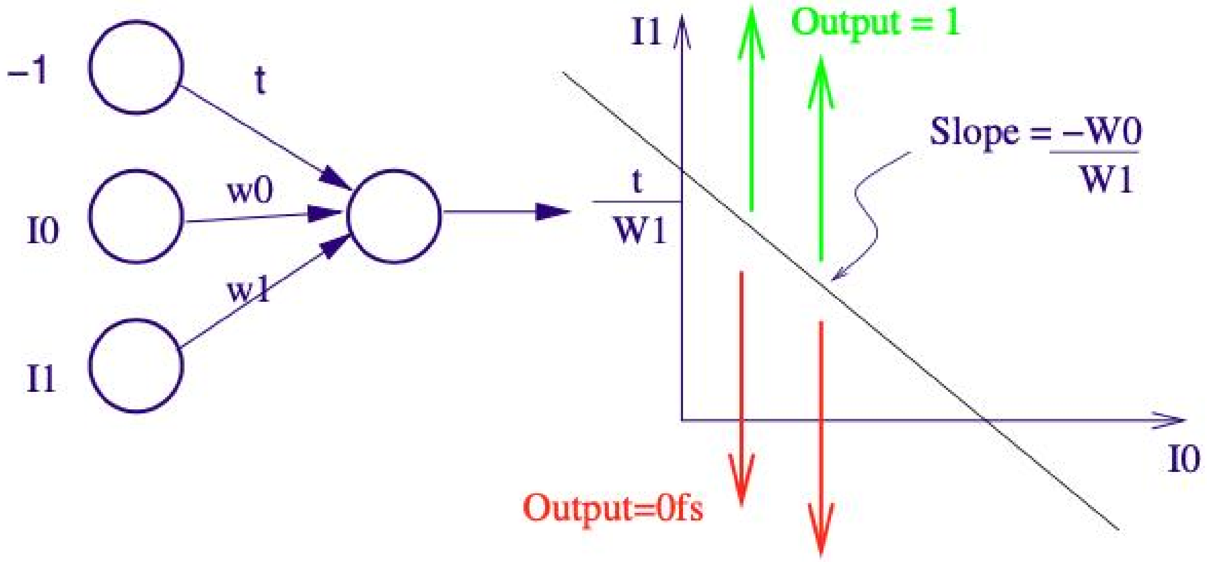


- For functions that take integer or real values as arguments and output either 0 or 1.
- Left: linearly separable (i.e., can draw a straight line between the classes).
- Right: not linearly separable (i.e., perceptrons cannot represent such a function)

单层感知机中涉及的基础概念（二维）

几何学意义

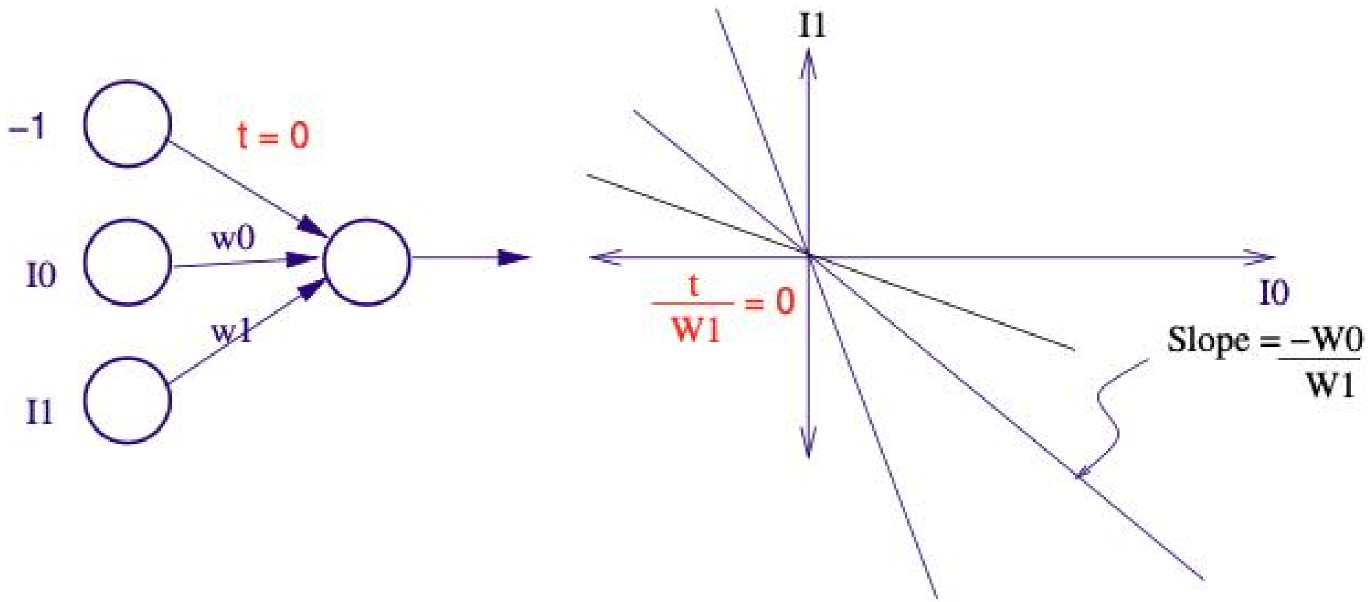
- 在该直线上方的点，输出为1， 下方，输出为0



单层感知机中涉及的基础概念（二维）

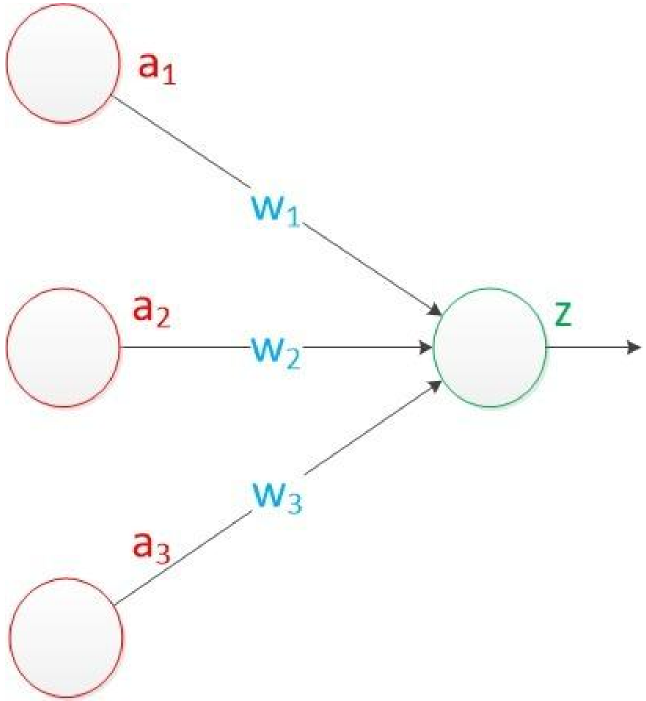
偏置（bias）的意义

- 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

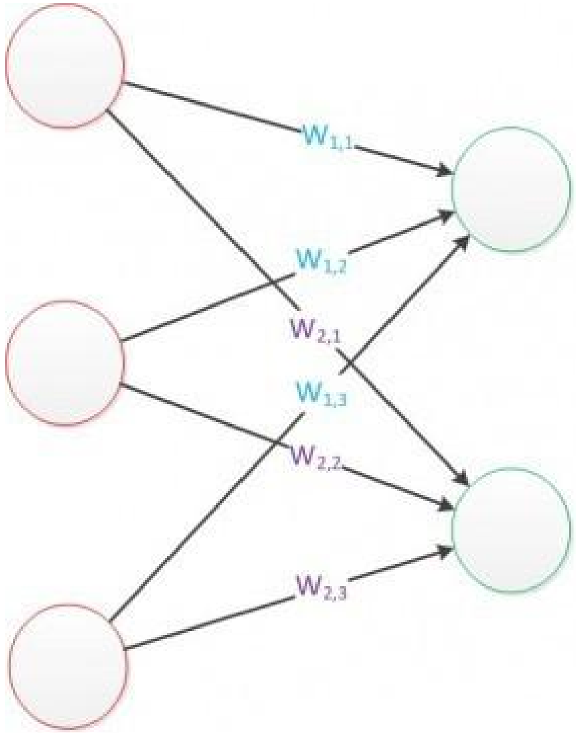


单层感知机的结构：只有输入层和输出层

- 多输入——单输出



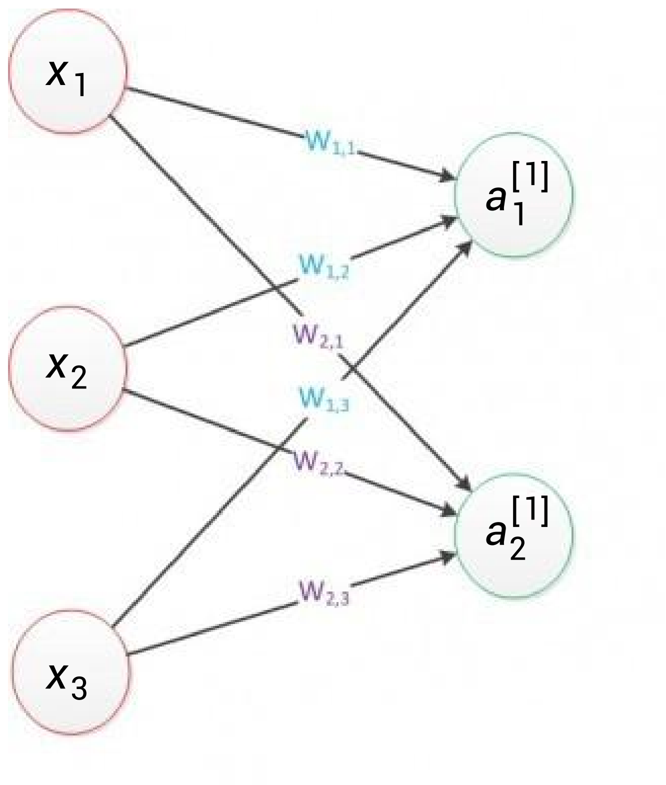
- 多输入——多输出



https://cloud.tencent.com/developer/article/1011180

单层感知机的数学表达：分量表达

- 多输入——多输出



- $a_j^{[i]}$: 输出层神经元
- i：第几层
- j：该层第几个神经元

- x_j ：输入层
- j：该层第几个输入

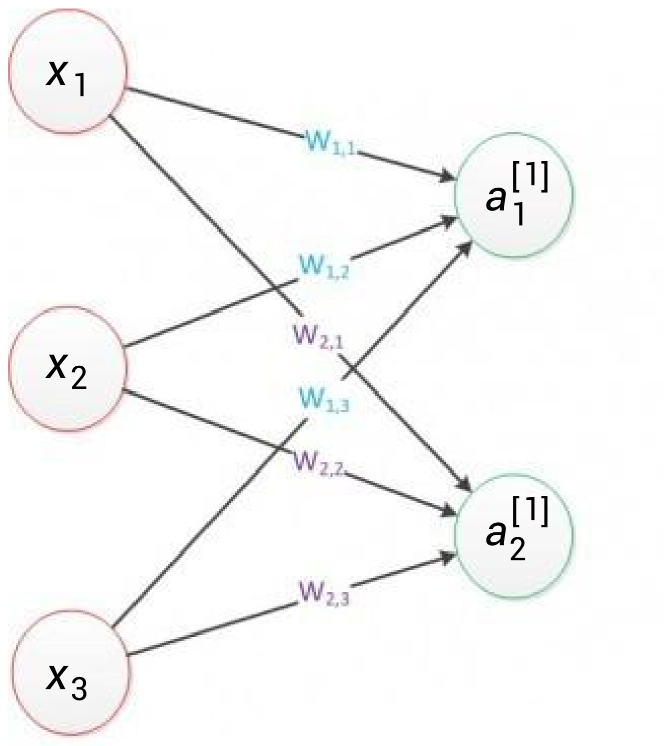
- $w_{i,0}x_0$: 偏置项（也写作 b），$x_0 = 1$

- $g(\cdot)$：激活函数（阈值函数等）

https://cloud.tencent.com/developer/article/1011180

单层感知机的数学表达：分量表达

- 多输入——多输出



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

$$a_1^{[1]} = g(z_1^{[1]})$$

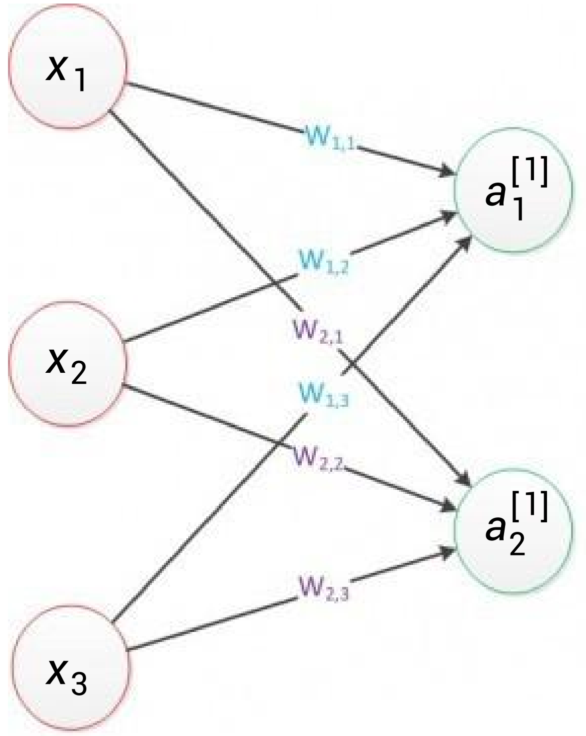
$$z_2^{[1]} = w_{2,0}x_0 + w_{2,1}x_1 + w_{2,2}x_2 + w_{2,3}x_3$$

$$a_2^{[1]} = g(z_2^{[1]})$$

https://cloud.tencent.com/developer/article/1011180

单层感知机的数学表达：向量表达

- 多输入——多输出



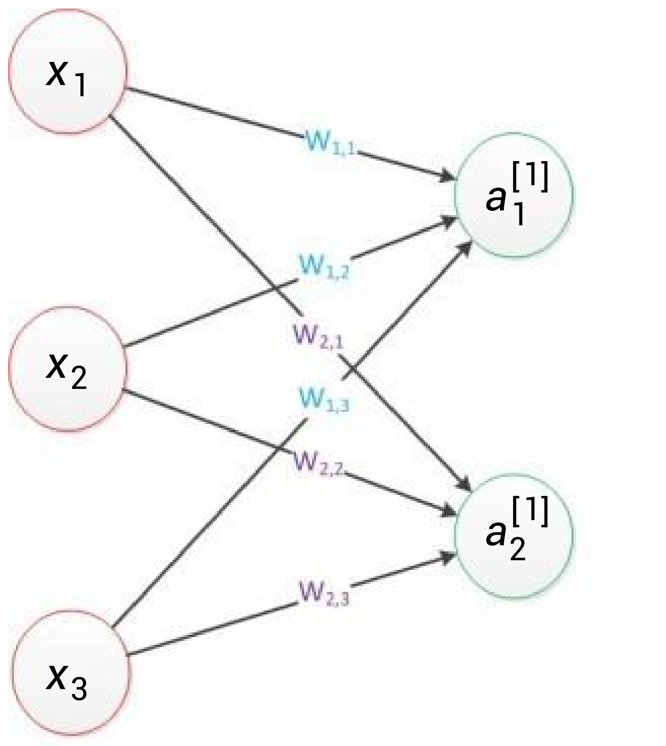
- $a_j^{[i]}$: 输出层神经元
- i： 第几层
- j： 该层第几个神经元

- 向量形状（2×1）
- $A^{[1]} = [a_1^{[1]}, a_2^{[1]}]^T$

https://cloud.tencent.com/developer/article/1011180

单层感知机的数学表达：向量表达

- 多输入——多输出



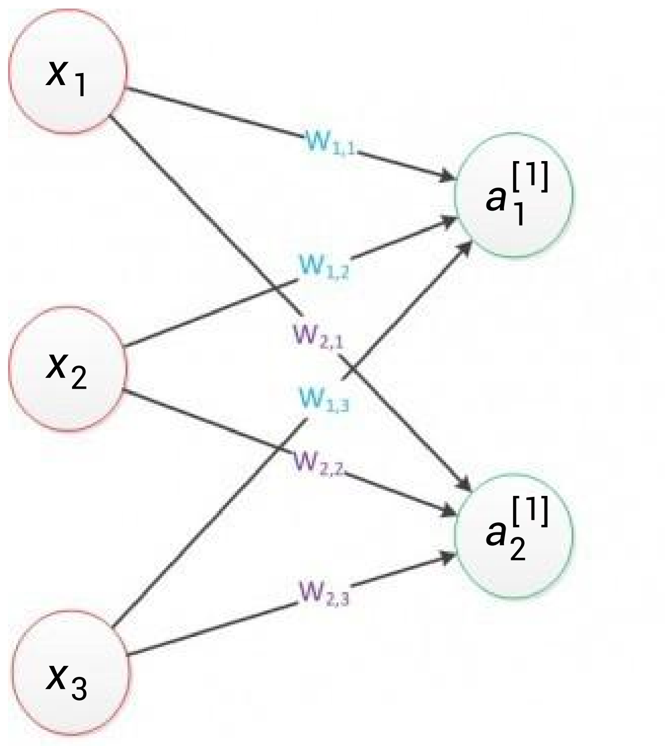
- x_j ： 输入层
- j： 该层第几个输入

- 向量形状（4 × 1）
- $X = [x_0, x_1, x_2, x_3]^T$

https://cloud.tencent.com/developer/article/1011180

单层感知机的数学表达：向量表达

- 多输入——多输出

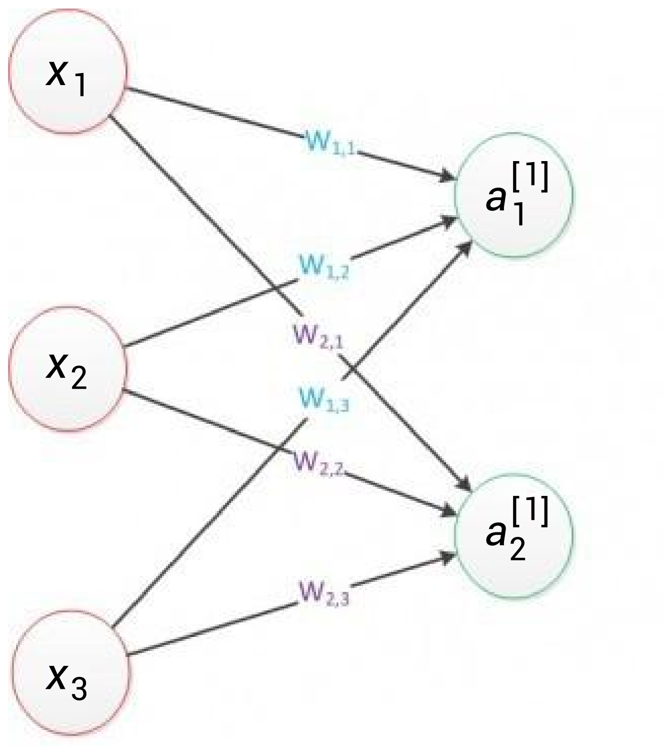


- $w_{i,0}x_0$：偏置项（也写作 b）， $x_0 = 1$
- $g(\cdot)$：针对矩阵中所有元素的激活函数（阈值函数等）

https://cloud.tencent.com/developer/article/1011180

单层感知机的数学表达：向量表达

- 多输入——多输出



- 向量形状（2 × 1）

$$Z^{[1]} = [Z_1^{[1]}, Z_2^{[1]}]^T$$

- 向量形状（4 × 2）

$$W^{[1]} = \begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} \end{bmatrix}^T$$

- 向量形状（2 × 1） = (2 × 4) · (4 × 1)

$$Z^{[1]} = {W^{[1]}}^T X$$

- 向量形状(2 × 1)

$$A^{[1]} = g(Z^{[1]})$$

https://cloud.tencent.com/developer/article/1011180

学习算法

Least-Mean-Square (LMS)算法

Adaptive Filter

- 1960年，Widrow and Hoff 在针对单神经元的 adaptive filters 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
- 从Adaptive Filter的角度来说，单个神经元是一个信号处理单元（signal processing）

https://people.engr.tamu.edu/choe/choe/courses/17spring/636/lectures/slide03.pdf

Adaptive Filtering Problem

- 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：

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

- 输入： $\boldsymbol{x}(i) = [x_1(i), x_2(i), \ldots, x_m(i)]^T$
- 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(spatial snapshot)或者时间序列(temporal sequence)

Adaptive Filtering Problem

- Adaptive Filtering由两个主要流程组成:
  - Filter Process：由输入生成输出

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

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

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

- 由梯度的定义

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

- 得

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

Steepest Descent

- 迭代更新的算法应该有以下性质

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

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

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

- 其中η是学习率，则有

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

Steepest Descent $\mathcal{E}(\boldsymbol{w}(n+1)) < \mathcal{E}(\boldsymbol{w}(n))$是否成立？

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

$$\mathcal{E}(\boldsymbol{w}(n+1)) \approx \mathcal{E}(\boldsymbol{w}(n)) + \boldsymbol{g}^T(n)\Delta\boldsymbol{w}(n)$$

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

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

- 得证，在学习率相对较小的情况下,有

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

- Taylor Series：

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

Least-Mean-Square（LMS） Algorithm

- 由瞬时值（instantaneous values）定义代价函数

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

- 对$\boldsymbol{w}$求微分

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

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

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

Least-Mean-Square（LMS） Algorithm

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

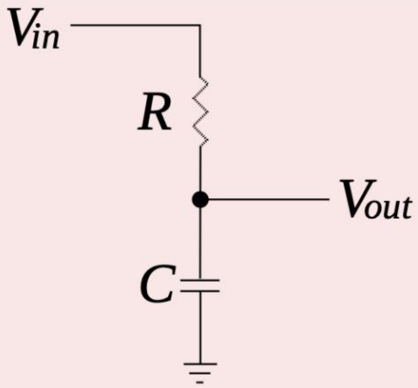
$$\hat{\boldsymbol{w}}_{n+1} = \hat{\boldsymbol{w}}_n + \eta \boldsymbol{x}_n e_n$$

注意：权重更新只针对一个训练样本$(\boldsymbol{x}_i, d_i)$

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$\mathcal{E}(\boldsymbol{w}) = \frac{1}{2}e^2(\boldsymbol{w})$
- 则有
- $\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n) = -\eta \boldsymbol{g}(n) = \eta((d_j - y_j(n)))\mathbf{x}(\mathrm{n})$
- **提示：** $\boldsymbol{g}(n) = \nabla \mathcal{E}(\boldsymbol{w}) = -\boldsymbol{x}(\boldsymbol{n})e(\boldsymbol{x}(\boldsymbol{n})) = -\boldsymbol{x}(n)(d - \boldsymbol{x}^T\boldsymbol{w}) = \mathbf{x}(\mathrm{n})((d_j - y_j(n)))$

感知机的学习过程（Batch Learning）

1. 设 $n = 0$ (iteration轮数)

2. 设$d_j = \begin{cases} +1, & if\ x_j(n) \in Class\ 1 \\ -1, & if\ x_j(n) \in Class\ 2 \end{cases}$, for all j = 1, 2, ..., m.

3. 初始化权重， $w^T = (w_1(n), w_2(n), \ldots, w_m(n))$

4. 初始化目标值来启动循环， $y^T = < y_1(n), y_2(n), \ldots, y_m(n) >$

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, …, m:

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

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

$n = n + 1$

单层感知机的目标

将一系列样本$\{X_1, X_2, \ldots, X_m\}$，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 Separabilty 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

$$\boldsymbol{w}(n_0) = \boldsymbol{w}(n_0 + 1) = \boldsymbol{w}(n_0 + 2) = ... \tag{35}$$

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

Jump To Proof 4

感知机的收敛定理：Proof 1

初始化

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

设在时间 n = 1，2，3，... 时

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

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

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS

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

代入感知机的correction formula（仅在分类错误时更新）

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

感知机的收敛定理：Proof 2

迭代使用correction formula得

$$\boldsymbol{w}(n+1)=\boldsymbol{x}(1)+\boldsymbol{x}(2)+\ldots+\boldsymbol{x}(n) \tag{39}$$

由线性可分，可知一定有解 w_n 使得

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

对（39）两边乘 $\boldsymbol{w}(0)$ 得

$$\boldsymbol{w}_{\mathbf{0}}^{\boldsymbol{T}}\boldsymbol{w}(n+1)=\boldsymbol{w}_{\mathbf{0}}^{\boldsymbol{T}}\boldsymbol{x}(1)+\boldsymbol{w}_0^T\boldsymbol{x}(2)+\ldots+\boldsymbol{w}_{\mathbf{0}}^{\boldsymbol{T}}\boldsymbol{x}(n) \tag{41}$$

感知机的收敛定理：Proof 3

将（40）带入（41）可得

$$\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{w}(n+1) \geq n\alpha \tag{42}$$

使用 Cauchy-Schwartz 不等式得，‖·‖是欧氏距离

$$\left\|\boldsymbol{w}_0^T\right\|^2 \|\boldsymbol{w}(n+1)\|^2 \geq \left[\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{w}(n+1)\right]^2 \tag{43}$$

将（42）带入（43）得

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

感知机的收敛定理：Proof 4

对（38）式，两边各求欧式范数得

$$\|\boldsymbol{w}(k+1)\|^2 = \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2 + 2\boldsymbol{w}^T(k)\,\boldsymbol{x}(k) \tag{44}$$

由（37）式得

$$\begin{aligned} \|\boldsymbol{w}(k+1)\|^2 &\le \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2 \\ \|\boldsymbol{w}(k+1)\|^2 - \|\boldsymbol{w}(k)\|^2 &\le \|\boldsymbol{x}(k)\|^2 \end{aligned} \tag{45}$$

感知机的收敛定理：Proof 5

对（45）利用裂项求和技巧可得

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

假设

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

则（46）可化简为

$$\|\boldsymbol{w}(n+1)\|^2\leq\textstyle\sum_{k=1}^{n}\|x(k)\|^2 \tag{47}$$

感知机的收敛定理：Proof 6

定义一个正数

$$\beta = \max_{\boldsymbol{x}(k)\in C_1} \left\| \boldsymbol{x}(k) \right\|^2 \tag{48}$$

将（48）带入（47）则有

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

上式当存在 n_{max} 时成立

$$\frac{n_{max}^2 \alpha^2}{\left\| \boldsymbol{w}_0 \right\|^2} = n_{max}\beta \tag{49}$$

感知机的收敛定理：Proof 7

在（49）中，对 n_{max} 求解，得

$$n_{max} = \frac{\beta \left\| \boldsymbol{w}_0 \right\|^2}{\alpha^2} \tag{50}$$

结论：

For $\eta(n) = 1$ for all n, $\boldsymbol{w}(0) = \mathbf{0}$ and a solution vector $\boldsymbol{w}_0$:

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

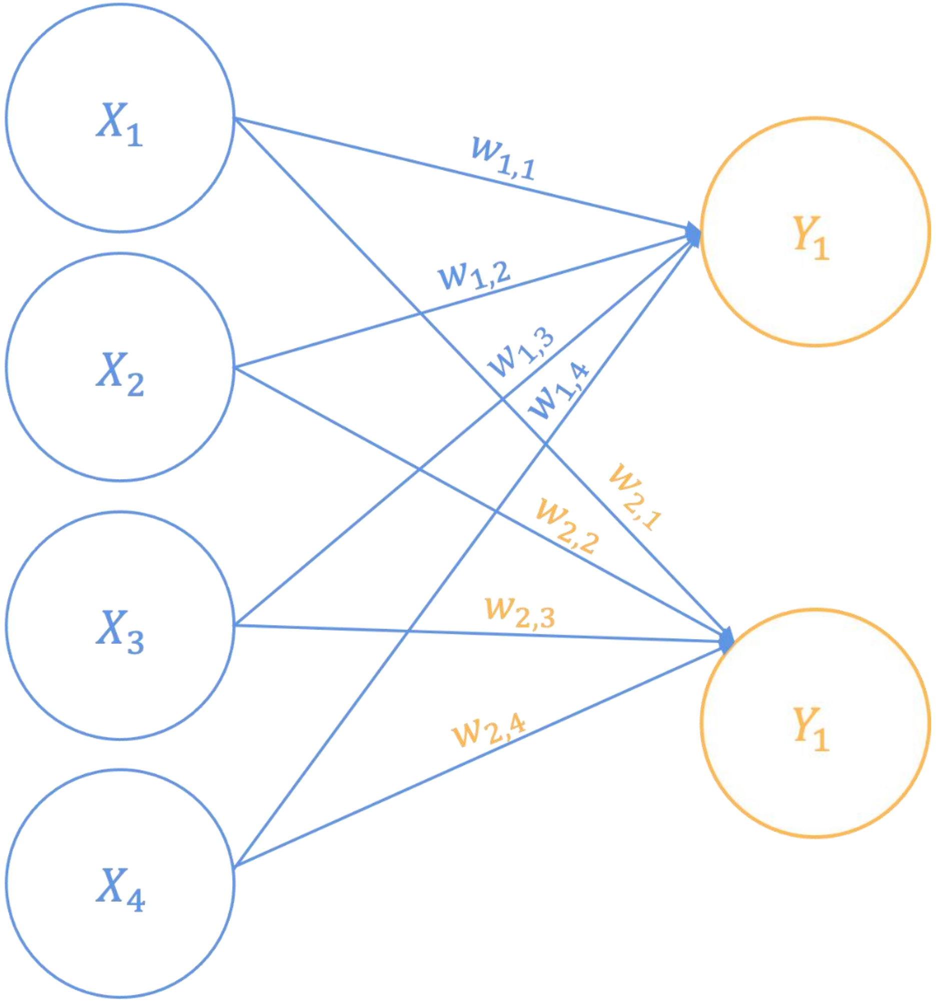
- 示例：利用身高（ height: cm ）、体重（ weight: kg ）、年龄（age: years）、受教育水平（education level）去预测一个人是否存在危害身体健康的肥胖症状（obesity）和 收入状况（income level：1000人民币/每月）（假设该问题线性可分）

- 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）

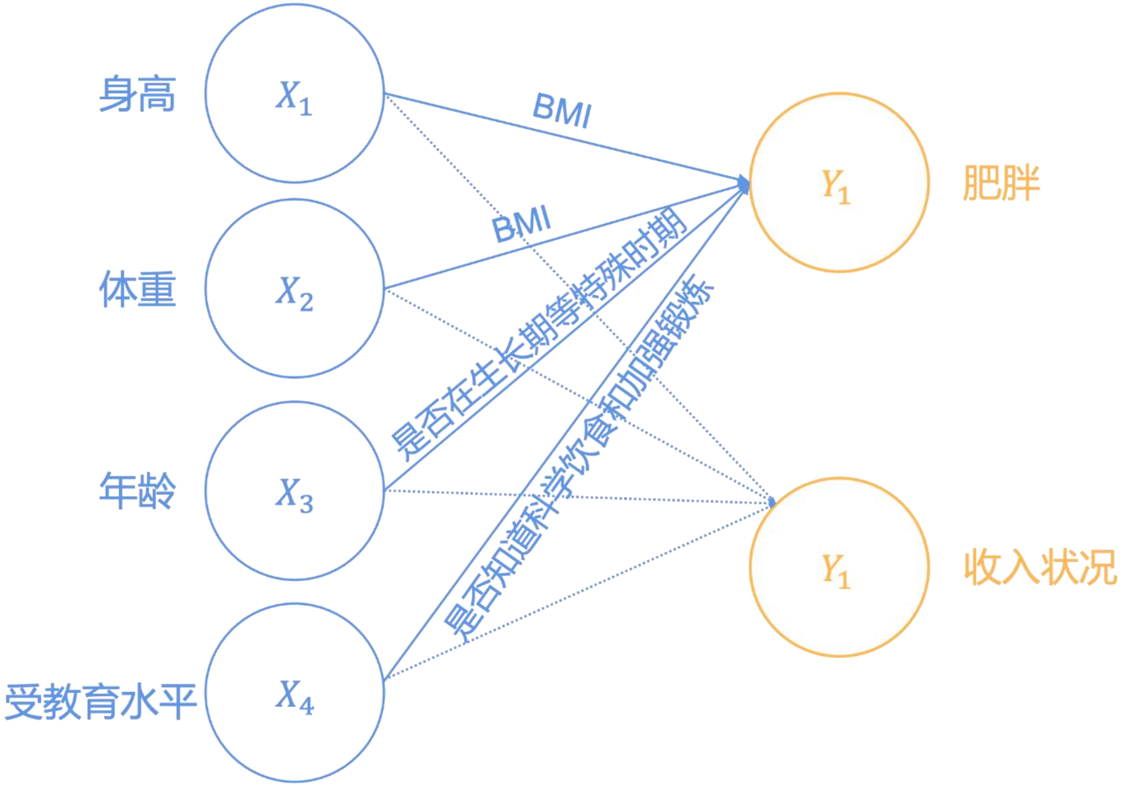


单层感知机：示例

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

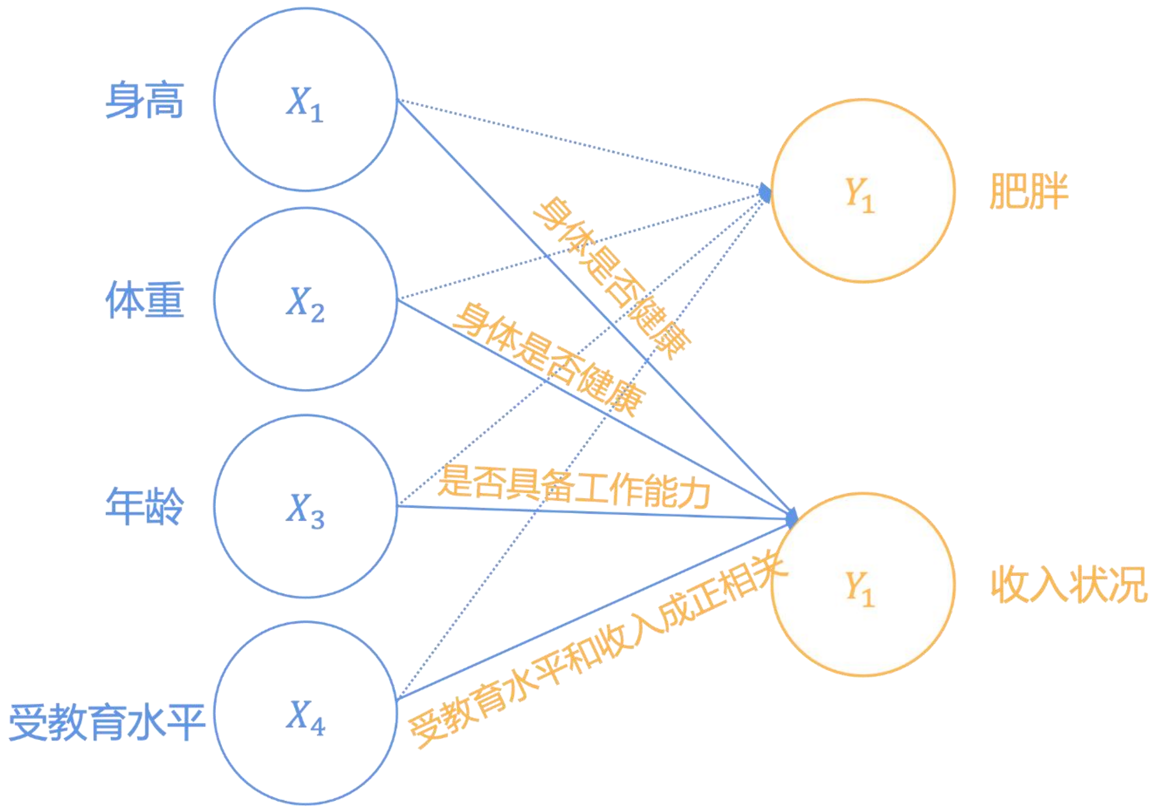


单层感知机：示例

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

- **思考题：** 请按照这个思路，去思考一下在章节开头的“分类+回归问题”中，权重分析会有什么变化吗？

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。

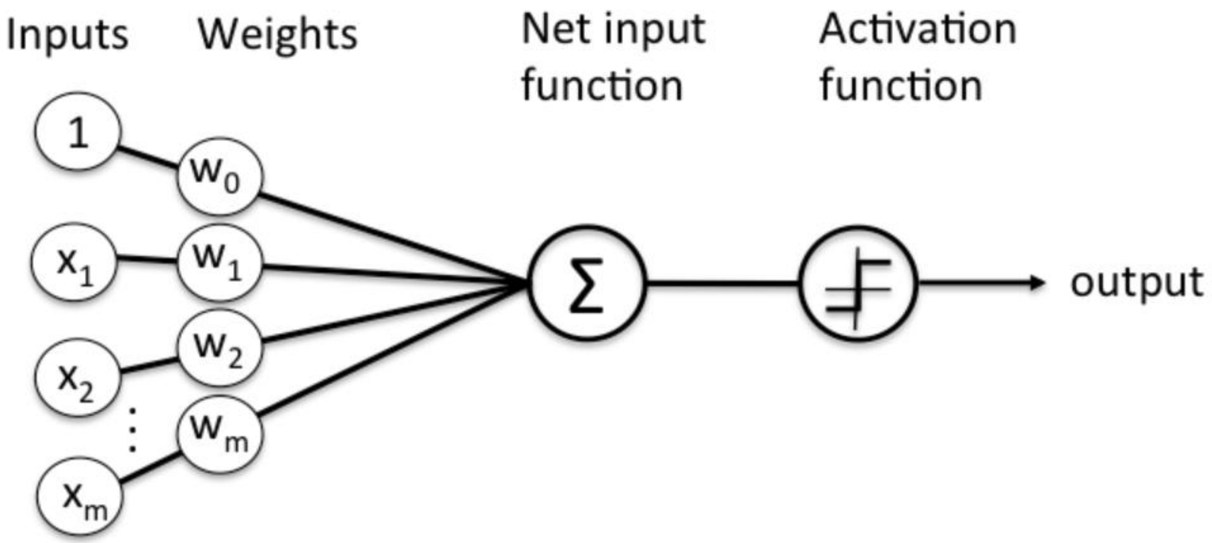
单层感知机的应用

- 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


Inputs
Weights
Net input function
Activation function
1
w_0
x_1
w_1
x_2
w_2
w_m
x_m
Σ
output

感知机模型

```python
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])
        self.errors_ = []

        for _ in range(self.epochs):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] +=  update * xi
                self.w_[0] +=  update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

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

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

3.初始化权重

4.初始化误差个数

5.遍历所有样本

求解Δ*w*

思考题

更新权重

记录误差

1.对输入的样本求和

2.使用阈值函数将求和结果映射到1或-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

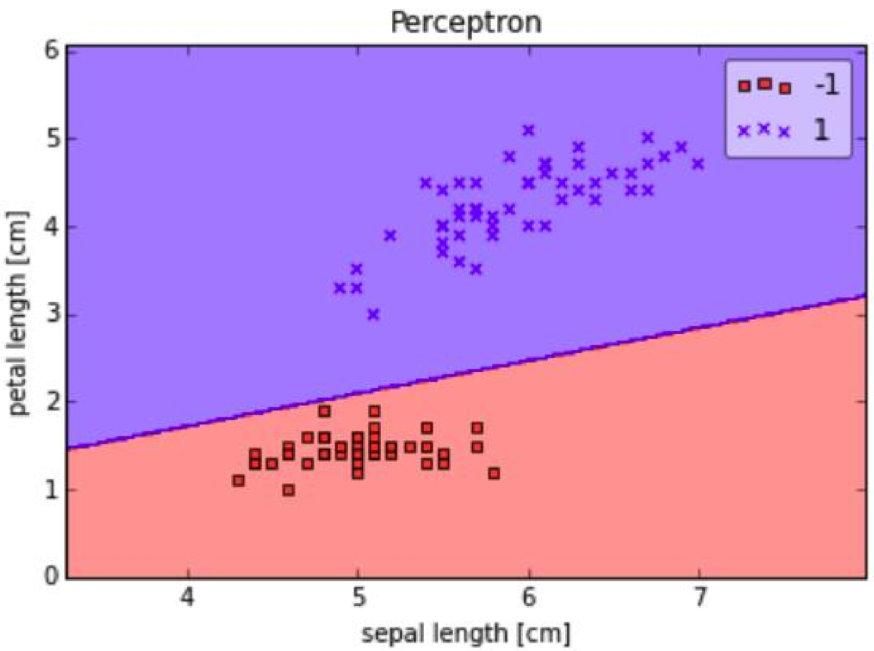
$$\Delta w_j = \eta(-1^{(i)} - -1^{(i)})\ x_j^{(i)} = 0$$
$$\Delta w_j = \eta(1^{(i)} - 1^{(i)})\ x_j^{(i)} = 0$$

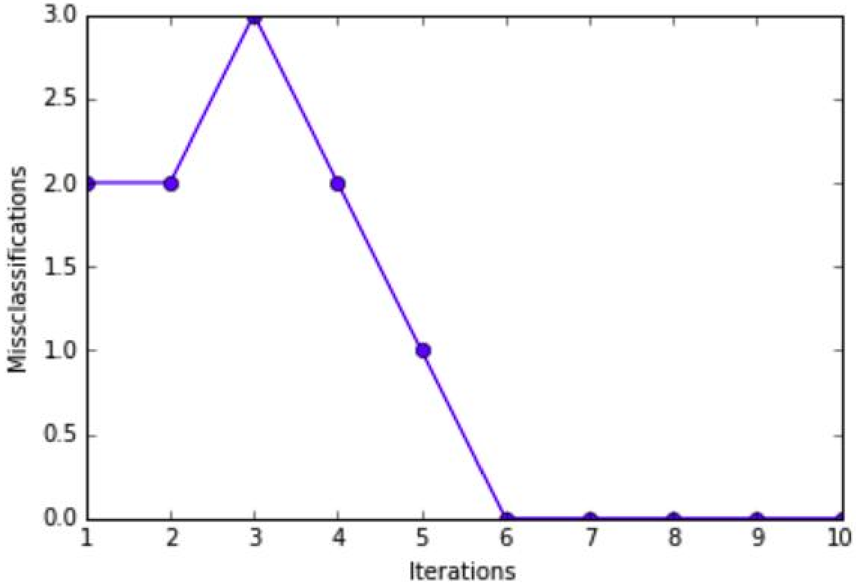
权重更新：错误分类

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

思考题：权重更新有没有更简洁写法？
（提示：感知机的向量表示）

分类结果



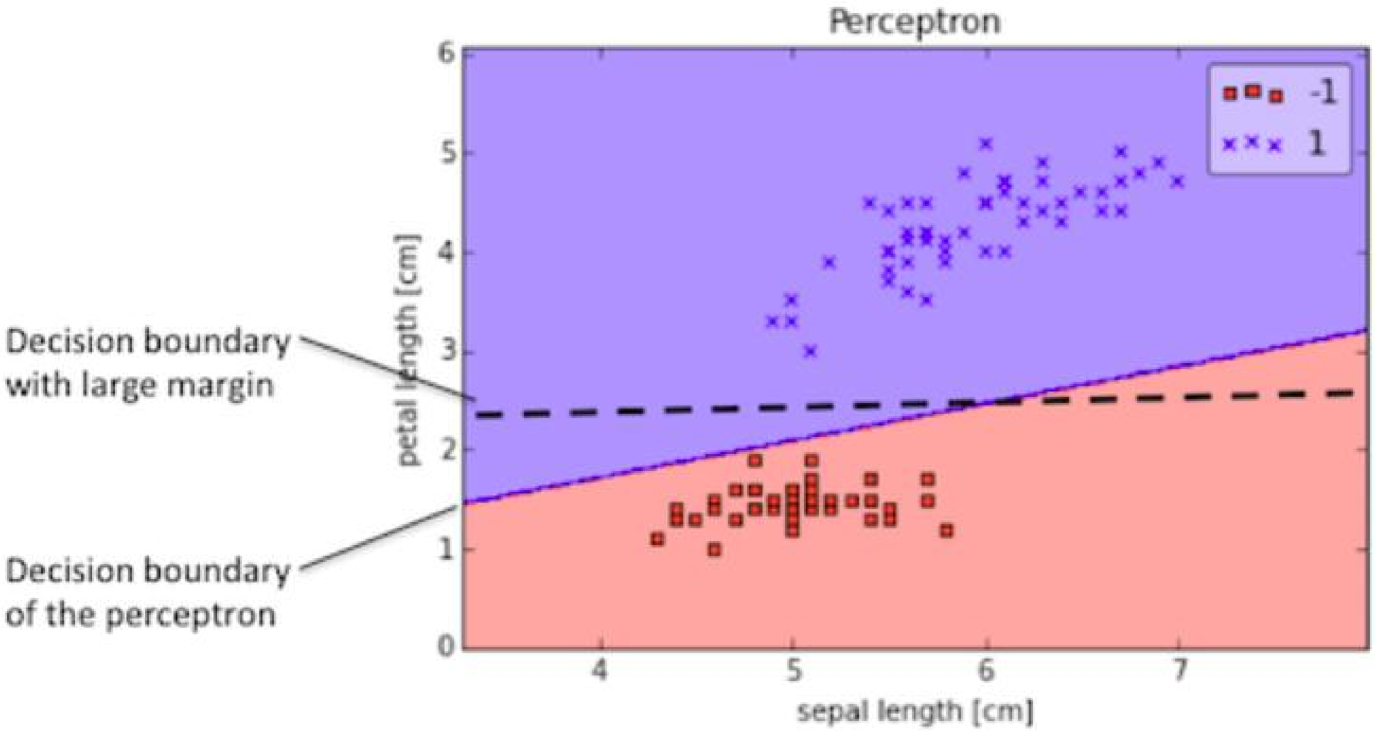


思考题：单层感知机会存在什么问题？
提示：线性可分前提

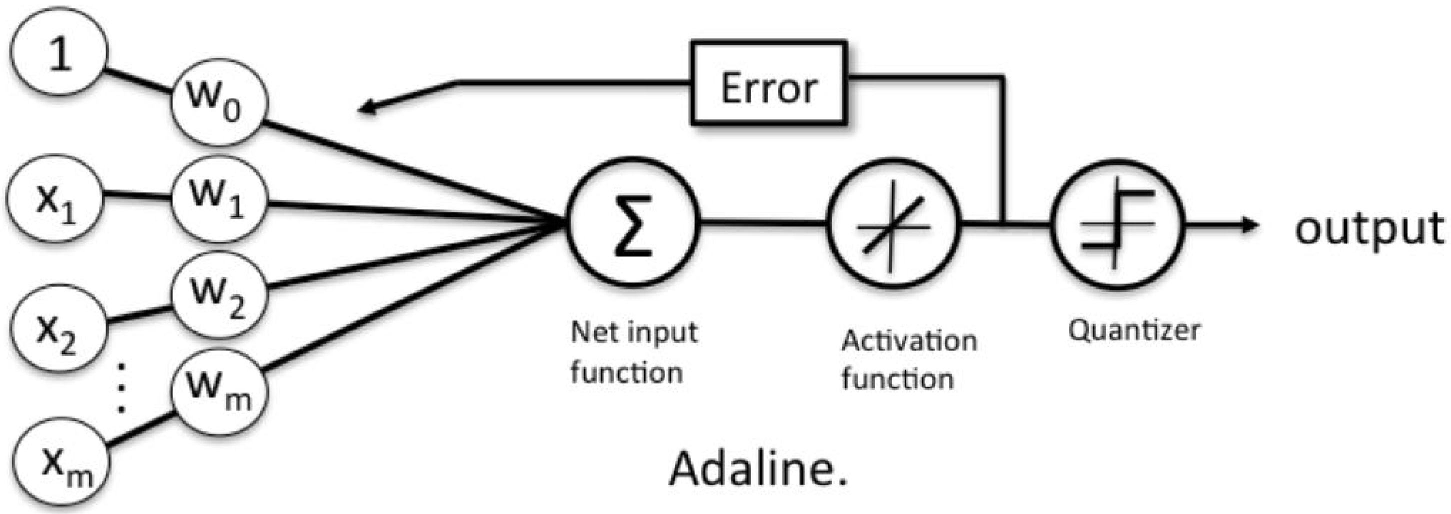
示例2:larger margin

- 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/)
- https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

当前模型在正确分类后便停止更新



Example of a large-margin decision boundary.



Adaline.

使用delta rule的感知机

重点：使用了Gradient Descent Rule

```python
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)
```

数学角度

损失函数

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

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)

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

带入更新

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

```python
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？

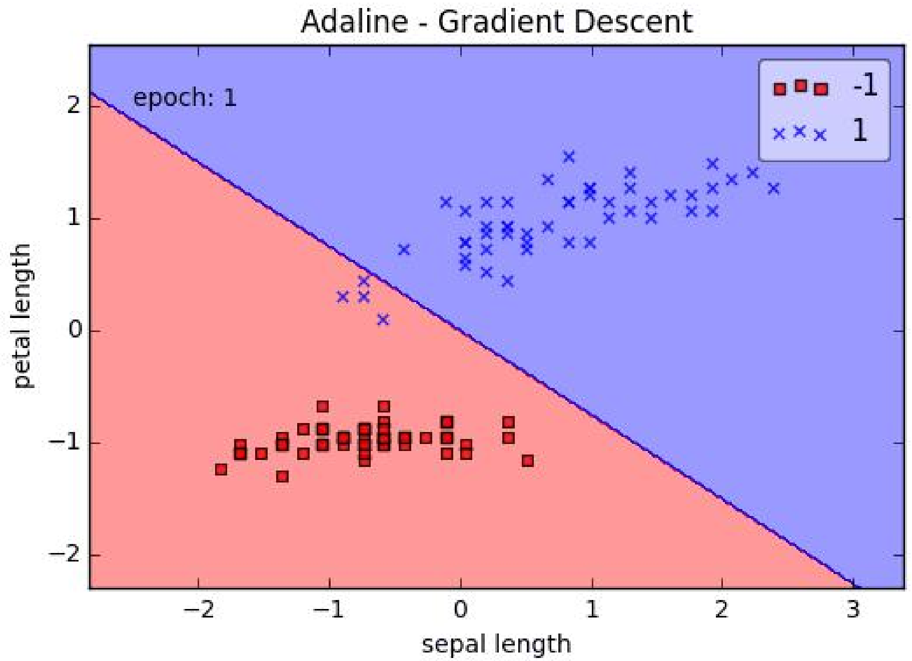
```python
import numpy as np

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)
```

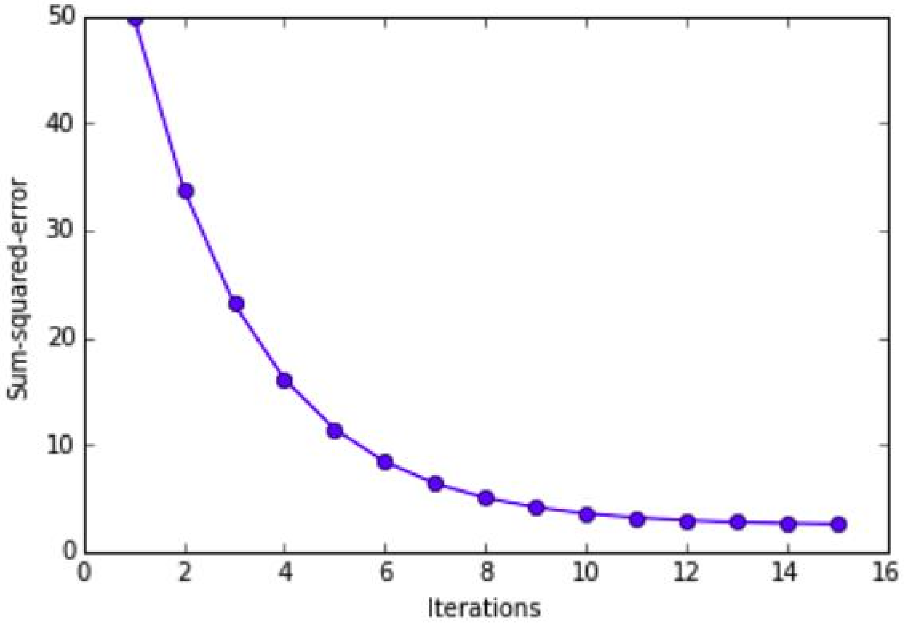
差别在于两点

- 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


Adaline - Gradient Descent
epoch: 1
-1
1
petal length
sepal length
2
1
0
−1
−2
−2
−1
0
1
2
3


Sum-squared-error
Iterations
50
40
30
20
10
0
0
2
4
6
8
10
12
14
16

思考题

- 简述神经元联接的不同方式及作用
- 简述神经元扩展的方式及使用场景
- 了解感知机权重不同的初始化方法
- 思考 LMS 算法的优劣势，以及对应的改进措施
- 参照应用中示例一与示例二的感知机模型与代码，思考对于Iris的完整数据集，输入和输出神经元的个数该如何设计

（注：对于完整的Iris数据集，输入特征有4个，花的种类有三种）

Q&A

Questions and Answers