第五章 计算智能

CI vs. AI

What is CI?

- The first published definition
  - In 1994, J.C. Bezdek states that:

    "*…a system is computational intelligence when it deal only with the numerical (low-level) data, has a pattern recognition component, and does not use knowledge in the AI sense, and additionally when it exhibit …*"
- Although used fairly widespread, there is no commonly accepted definition

贝兹德克
(J.C. Bezdek)

What is CI?

- 计算智能系统
  - 一个计算智能系统应当只涉及数值(低层)数据，含有模式识别部分，不应用人工智能意义上的知识，而且能够呈现出：
    - Computational adaptivity (计算适应性)
    - Computational fault tolerance (计算容错性)
    - Speed approaching human-like turnaround (接近人的速度），and
    - Error rates that approximate human performance （近似于人的误差率）…”
- 人工智能系统
  - 当一个智能计算系统以非数值方式加上知识，即成为人工智能系统。

What is CI?

- 计算智能是信息科学与生命科学相互交叉的前沿领域，是现代科学技术发展的一个重要体现
- 典型方法
  - 模糊逻辑 (Fuzzy Logic)
  - 神经计算 (Neurocomputing)
  - 进化计算 (Evolutionary Algorithms)
  - 群优化 (Swarm Optimization)

- Some other opinions:
  - **Conference: Computational Intelligence - Methods & Applications- CIMA2005**
    - Defining "**Computational Intelligence**" is not straightforward. It is difficult, if not impossible, to accommodate in a formal definition disparate areas with their own established individualities such as fuzzy sets, neural networks, evolutionary computation, machine learning, Bayesian reasoning, etc.
  - **Book: "Computational Intelligence: An Introduction", Andries P. Engelbrecht, Wiley 2002**
    - Computational intelligence is the study of adaptive mechanisms to enable or facilitate intelligent behavior in complex and changing environments. As such, computational intelligence combines artificial neural networks, evolutionary computing, swarm intelligence and fuzzy systems.

AI vs. CI

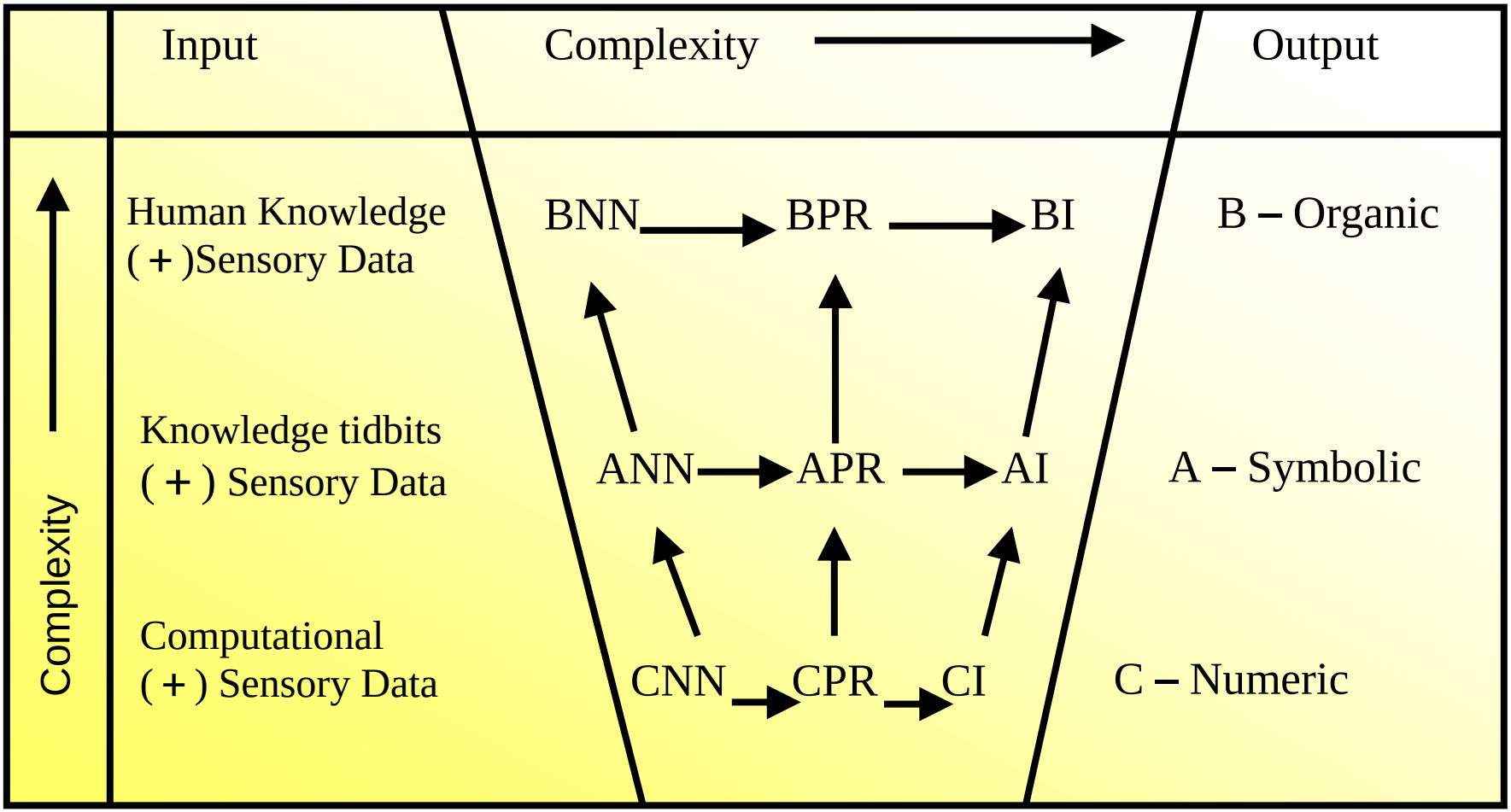
- **The huge majority of AI/CI researchers concerned with the subject sees them as different areas**
  - CI forms an alternative to AI (R. C. Eberhart)
    - **seeking similar goals**
    - **CI is buried deeply within the core of the system (either carbon-based or silicon-based)**
    - **AI is at the outer level**
  - AI subsumes CI (Bezdek)

艾伯哈特 (Eberhart)

AI vs. CI

- Due to Bezdek
  - Three levels of system complexity
    - A: Artificial or Symbolic
    - B: Biological or Organic
    - C: Computational or Numeric system
- 计算智能是一种智力方式的低层认知，它与人工智能的区别只是认知层次从中层下降至低层而已。中层系统含有知识，低层系统则没有。

Commuting through ABC



Relationships among components of intelligent system (after Bezdek 1994)

NN: Neural Network **PR: Pattern Recognition**

神经计算
Neurocomputing

Outline

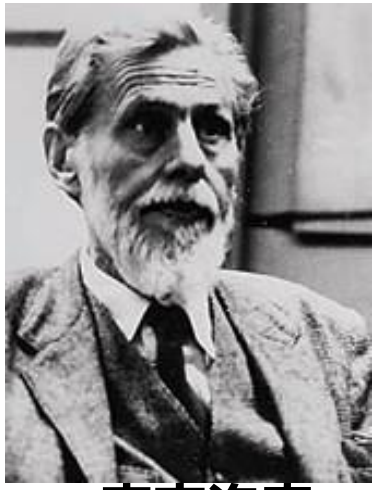
- 概述
- 生物神经网络
- 神经元模型
- 线性阈值单元TLU
- 人工神经网络结构
- 人工神经网络学习

Introduction of ANN (Artificial Neural Networks)

- As you read these words you are using a complex ***biological neural network***. You have a highly interconnected set of $\mathbf{10^{11}}$ neurons to facilitate your ***reading***, ***breathing***, ***motion*** and ***thinking***.
- In the ***artificial neural network***, the neurons are ***not biological***. They are extremely simple abstractions of biological neurons, realized as elements in a ***program*** or perhaps as ***circuits*** made of silicon.

Introduction— History

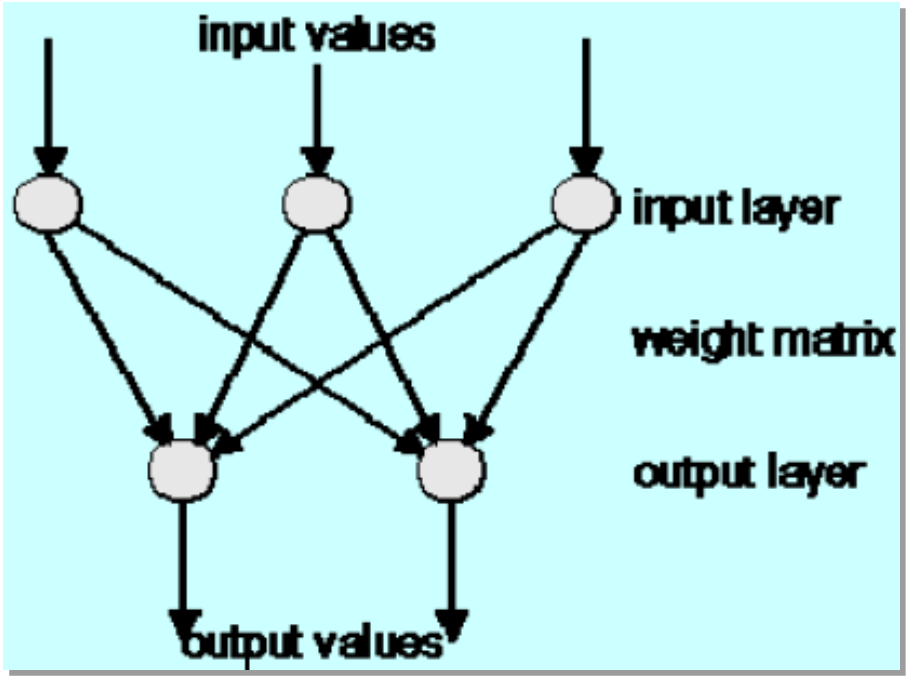
- McCulloch & Pitts (1943) are generally recognized as the designers of the first neural network
  - Their ideas such as threshold and many simple units combining to give increased computational power are still in use today
- In the 50's and 60's, many researchers worked on the perceptron
- In 1969, Minsky and Papert showed that perceptrons were limited so neural network research died down for about 15 years
- In the mid 80's interest revived (Parket and LeCun)



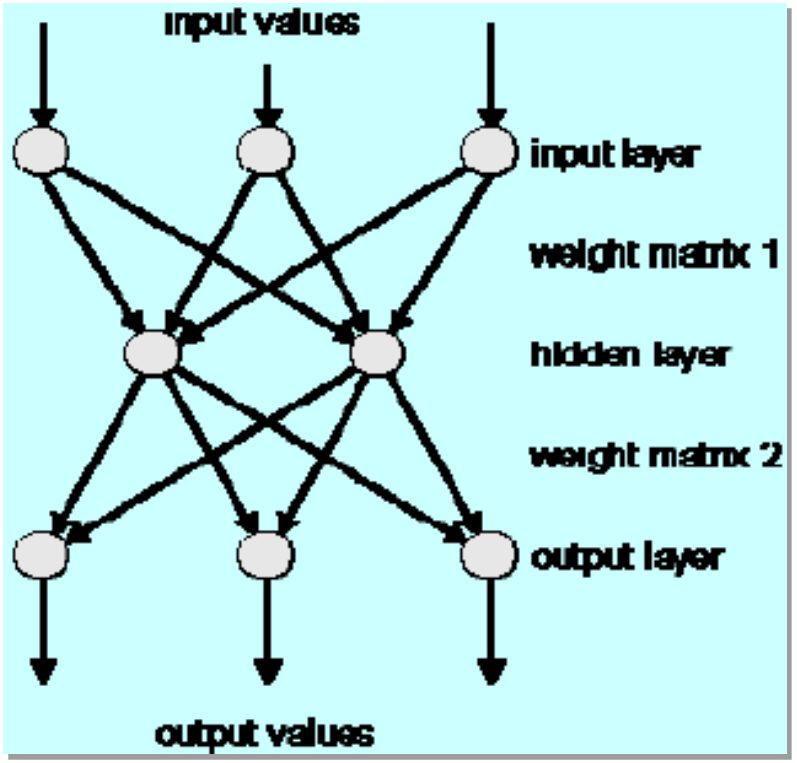
麦克洛奇 (McCulloch)

皮茨 (Pitts)

Introduction— Some Examples of ANN

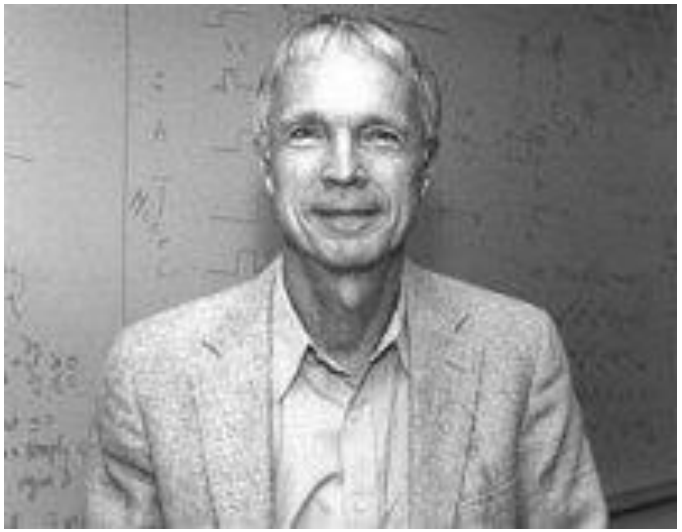


One Layer Perceptron

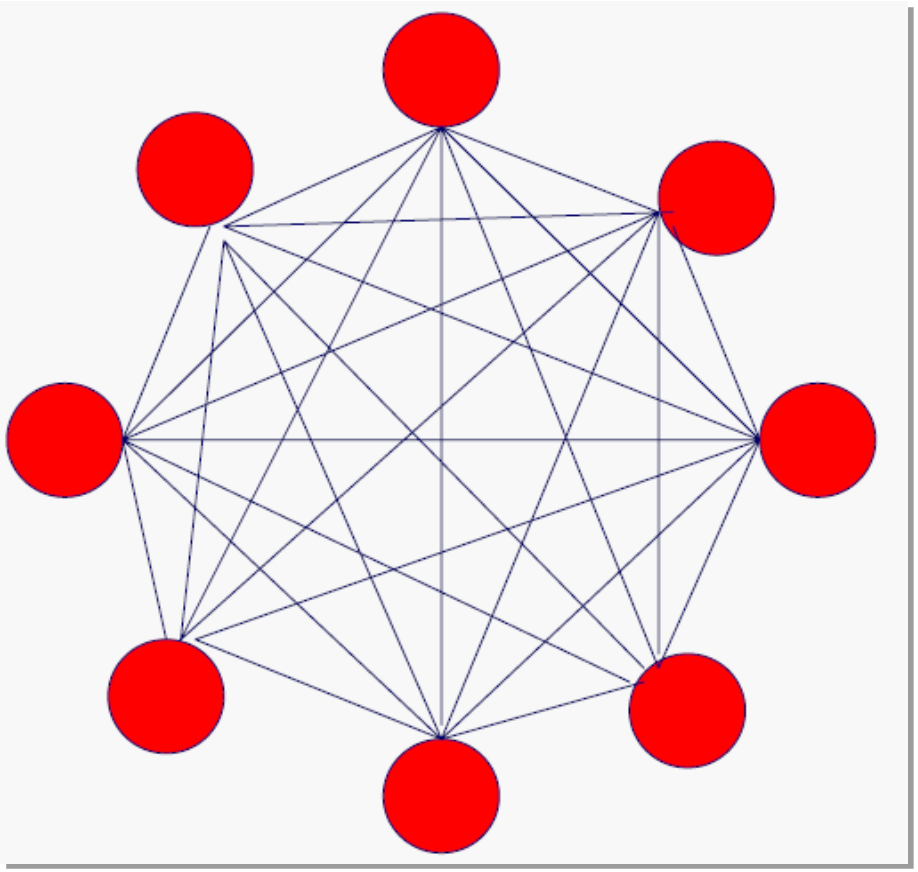


Two Layer Perceptron

Introduction— Some Examples of ANN

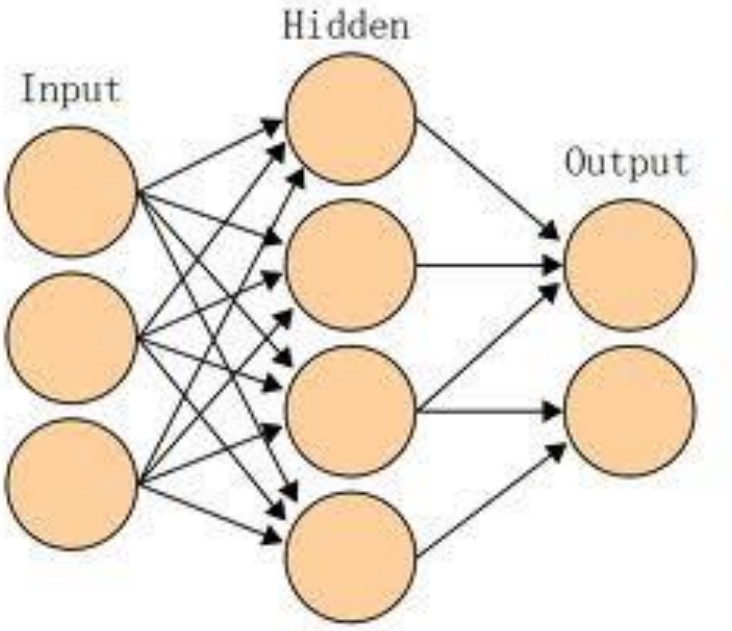


Hopfield



Hopfield Network

Introduction – ANN的特性



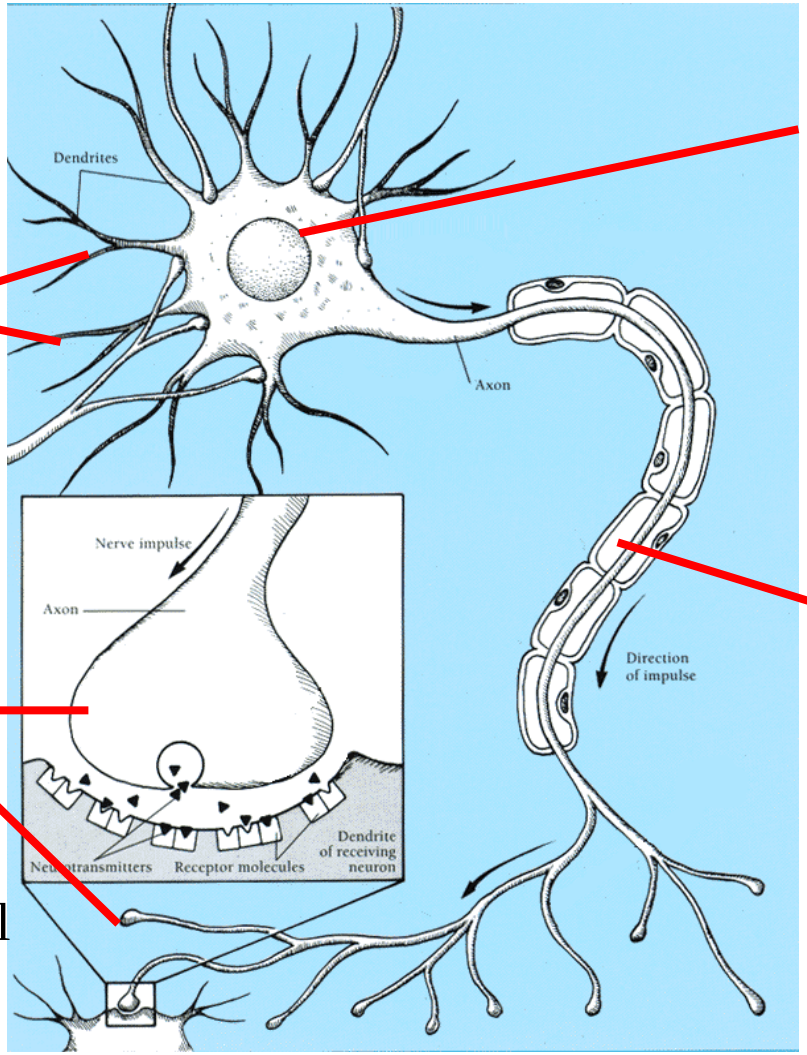
- 并行分布处理
- 非线性映射
- 通过训练进行学习
- 适应与集成
- 硬件实现

这些特性使得人工神经网络具有应用于各种智能系统的巨大潜力。

Outline

- 概述
- *生物神经网络*
- 神经元模型
- 线性阈值单元TLU
- 人工神经网络结构
- 人工神经网络学习

The Biological Neurons (生物神经网络)



Cell Body(细胞体)

•5–10 microns in diameter sums and thresholds these incoming signals

Dendrites(树突)

•carry electrical *into* the cell body

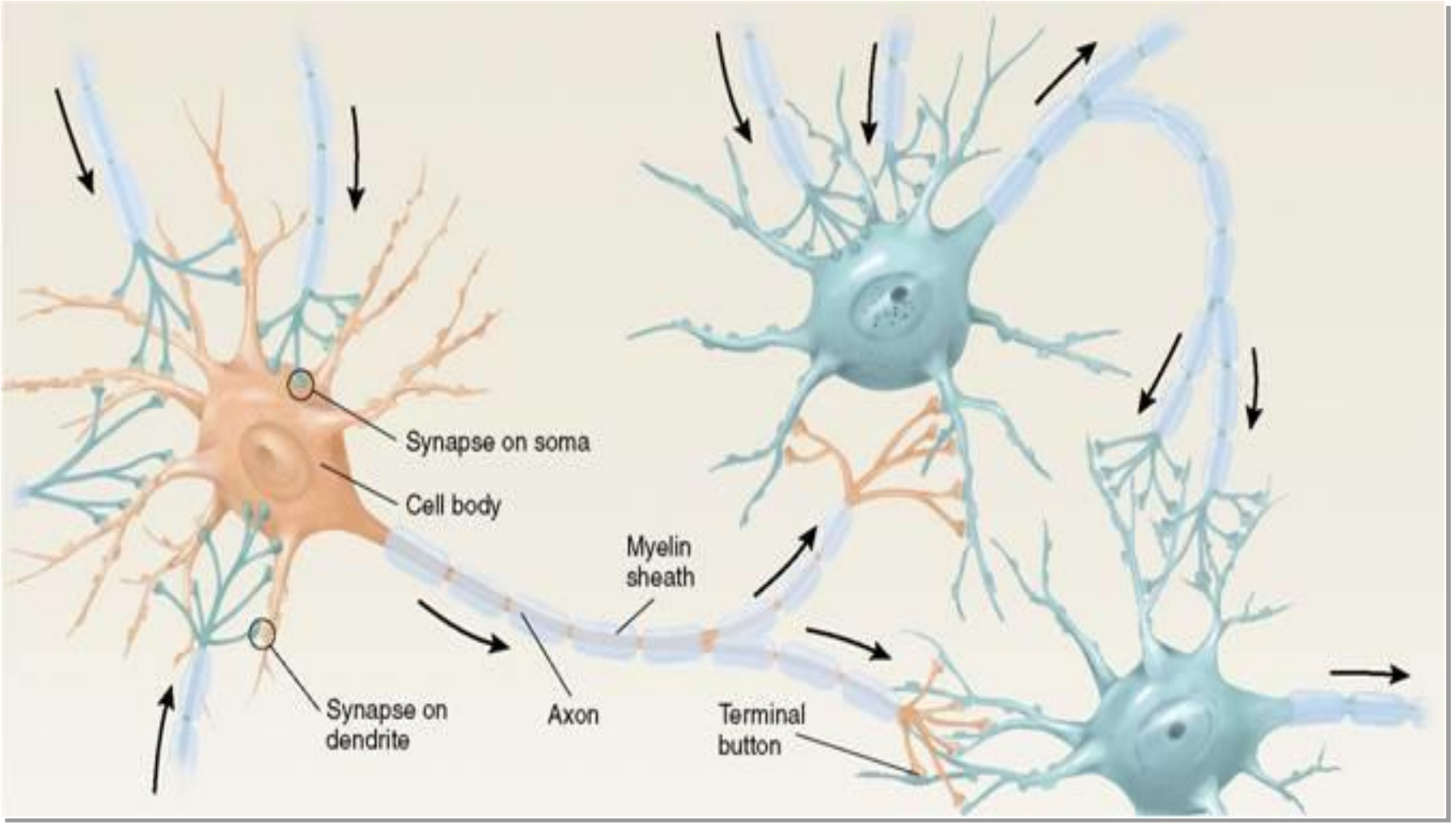
Axon(轴突)

•carry the signal from the cell body *out* to other neurons

Synapse(突触)

•contact between an axon of one cell and a dendrites of another cell

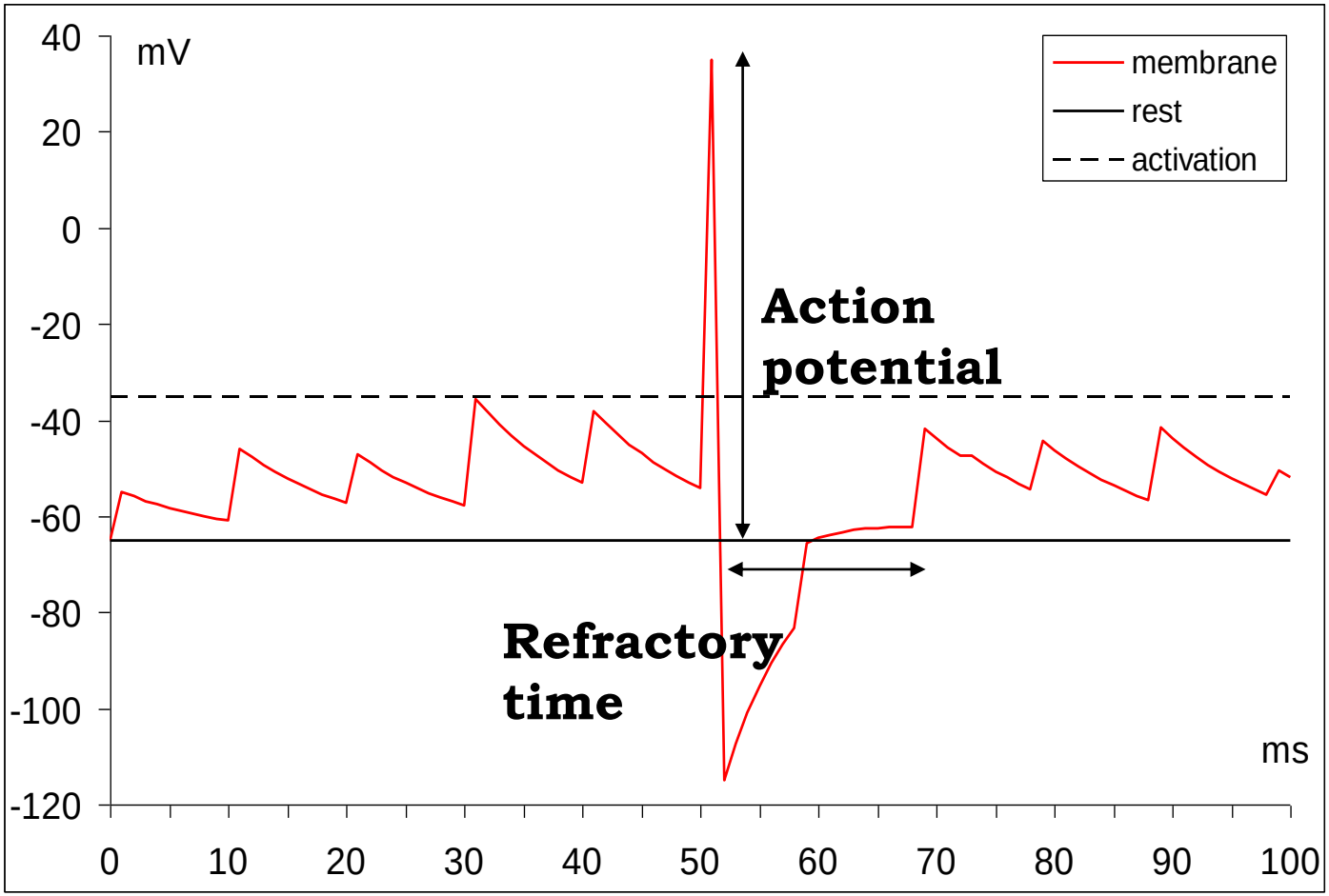
A Neuron (单神经元) : many-inputs/one-output unit



- 学习方法
  - Synapses change size and strength with experience.
  - Hebbian learning: “Neurons that fire together, wire together.”

The Biological Neurons

- **Neuron Dynamics**



Action potential ≈ 100mV **Activation threshold ≈ 20-30mV**
Rest potential ≈ -65mV **Spike time ≈ 1-2ms**
Refractory time ≈ 10-20ms

Neural Networks

- **Neural Networks**
  - a promising new generation of **information processing systems**, *usually operate in parallel*, that demonstrate the ability to **learn（学习）**, **recall（记忆）**, and **generalize（概括，归纳，推广，泛化）** from training patterns or data.

Outline

- 概述
- 生物神经网络
- *人工神经元模型*
- 线性阈值单元TLU
- 人工神经网络结构
- 人工神经网络学习

Notation

- **Scalars（标量）**: small *italic* letters
  e.g., ***a***, ***b***, ***c***
- **Vectors（向量）**: small **bold** nonitalic letters
  e.g., **a**, **b**, **c**
- **Matrices（矩阵）**: capital **BOLD** nonitalic letters
  e.g., **A**, **B**, **C**

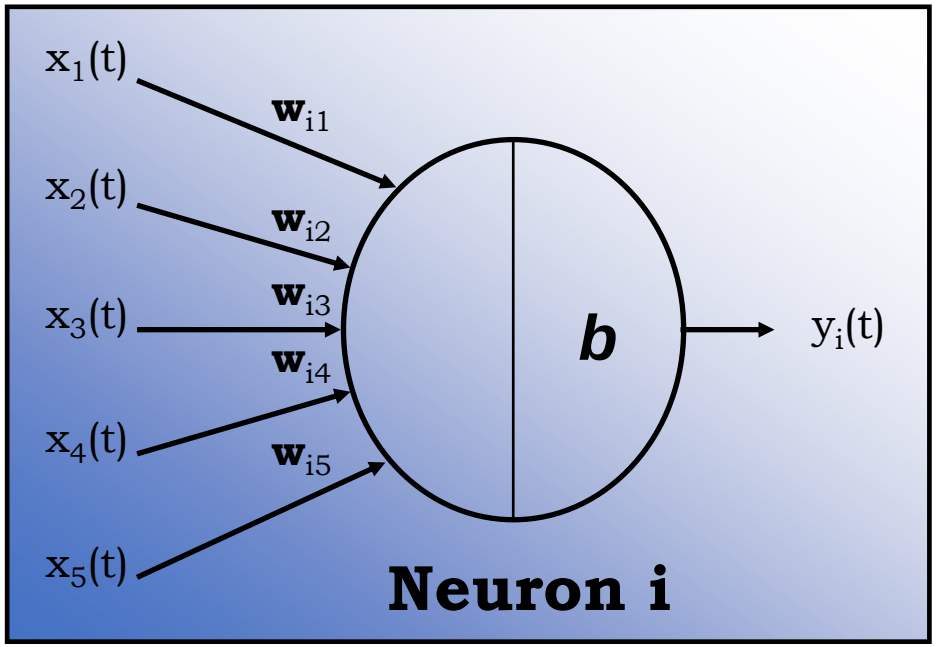
The Artificial Neuron

Stimulus

$$u_i(t) = \sum_j w_{ij} \cdot x_j(t)$$

Response

$$y_i(t) = f(b + u_i(t))$$



b* = *threshold* or *bias
$y_i(t)$* = output of neuron *i* at time *t
w_{ij}* = connection strength between neuron *i* and neuron *j*, *weight
f* = *transfer function*, or *activation function

Bias and Weight（阈值与权重）

- 可将阈值b看作权重，则其输入为常量1

$$\begin{cases} u_i = \sum_{j=1}^{n} w_{ij} x_j \\ y_i = f(u_i + b) \end{cases} \Rightarrow \begin{cases} u_i = b + \sum_{j=1}^{n} w_{ij} x_j = \sum_{j=0}^{n} w_{ij} x_j \\ y_i = f(u_i) \end{cases}$$

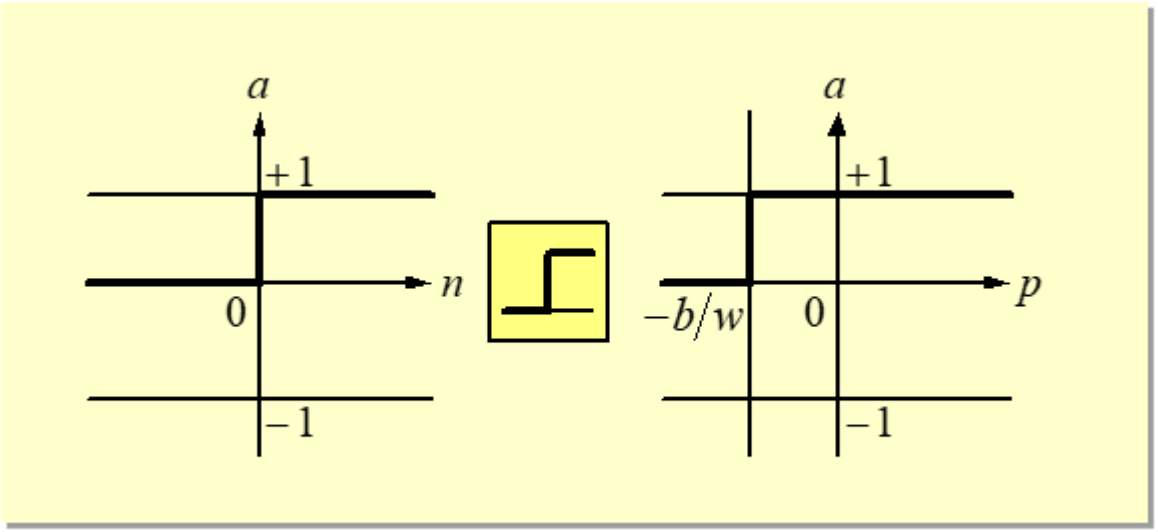
其中，$b = x_0, w_{i0} = 1$

- 若不需要时，阈值b可省略
- 阈值b和权重w是**可调节**的标量
- 根据某些**学习规则**来调节它们，使得神经元的输入输出关系满足某些特定的目标。

Transfer Functions（传递函数）

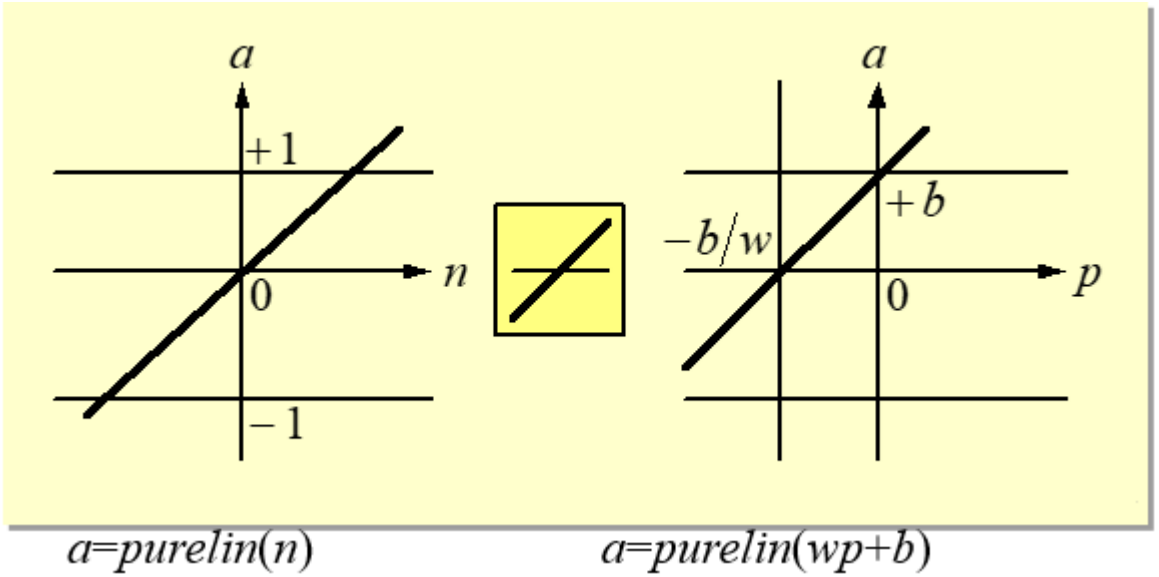
- The transfer function *f* may be a ***linear*** (**线性**)or ***nonlinear (非线性)*** function of net input *n*
- Three of the most commonly used functions.
  - **Hard limit** transfer function (二值函数)
  - **Linear** transfer function (线性函数)
  - **Log-sigmoid** transfer function (双曲正切)

Hard limit transfer function



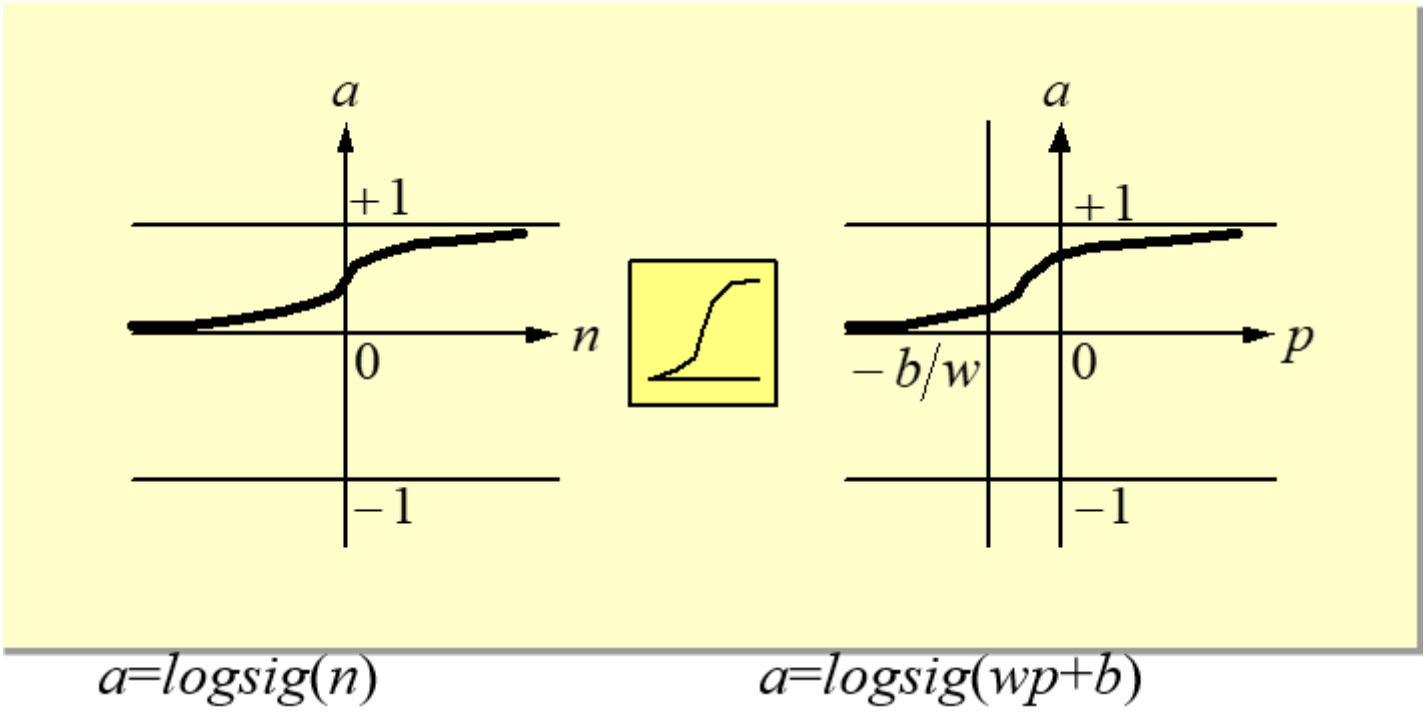
- $a = 0$, if $n < 0$
- $a = 1$, if $n \geq 0$
- **MATLAB function: *hardlim***

Linear transfer function



- ***a = n***
- **MATLAB function: *purelin***

Log-sigmoid transfer function



- $a = 1/[1 + exp(-n)]$
- MATLAB function: *logsig*

Outline

- 概述
- 生物神经网络
- 人工神经元模型
- *线性阈值单元TLU*
- 人工神经网络结构
- 人工神经网络学习

Linear Threshold Unit /Threshold Logic Unit, TLU （线性阈值单元）

- TLU的基本形式

$$y = f\left(\sum_{j=1}^{d} w_j x_j + w_0\right) = f(\mathbf{w}^T\mathbf{x}) = \begin{cases} 0 & \mathbf{w}^T\mathbf{x} < 0 \\ 1 & \mathbf{w}^T\mathbf{x} >= 0 \end{cases}$$

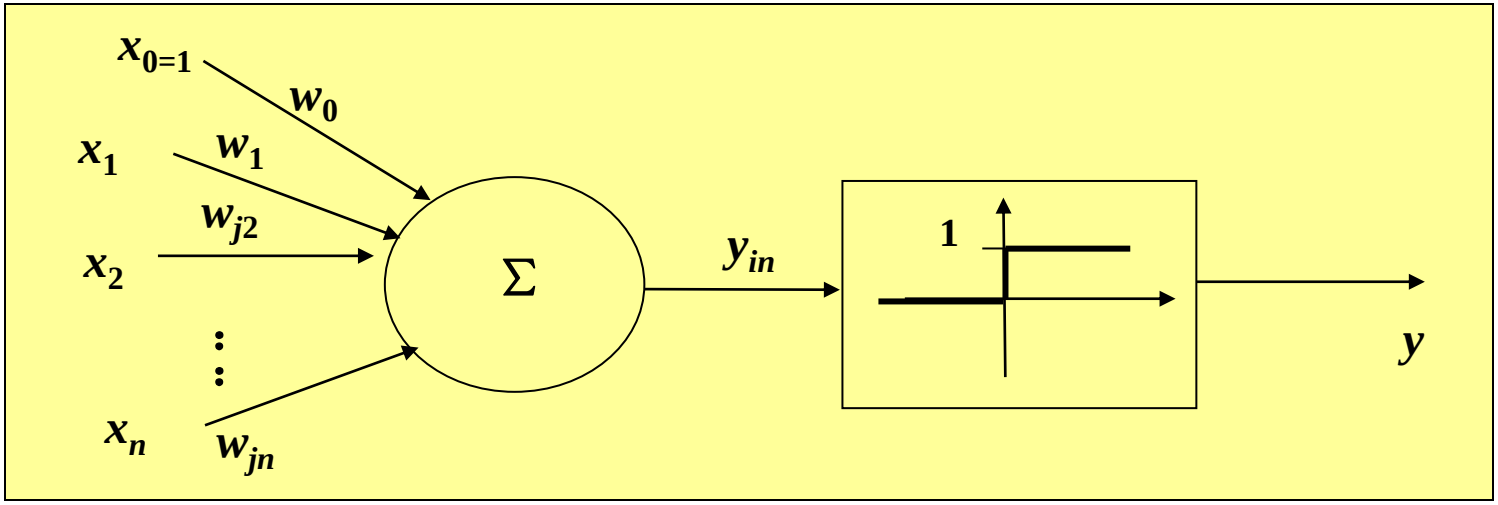
$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$$

$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$

- TLU的基本功能——线性划分

线性阈值单元(TLU)和感知器(Perceptron)

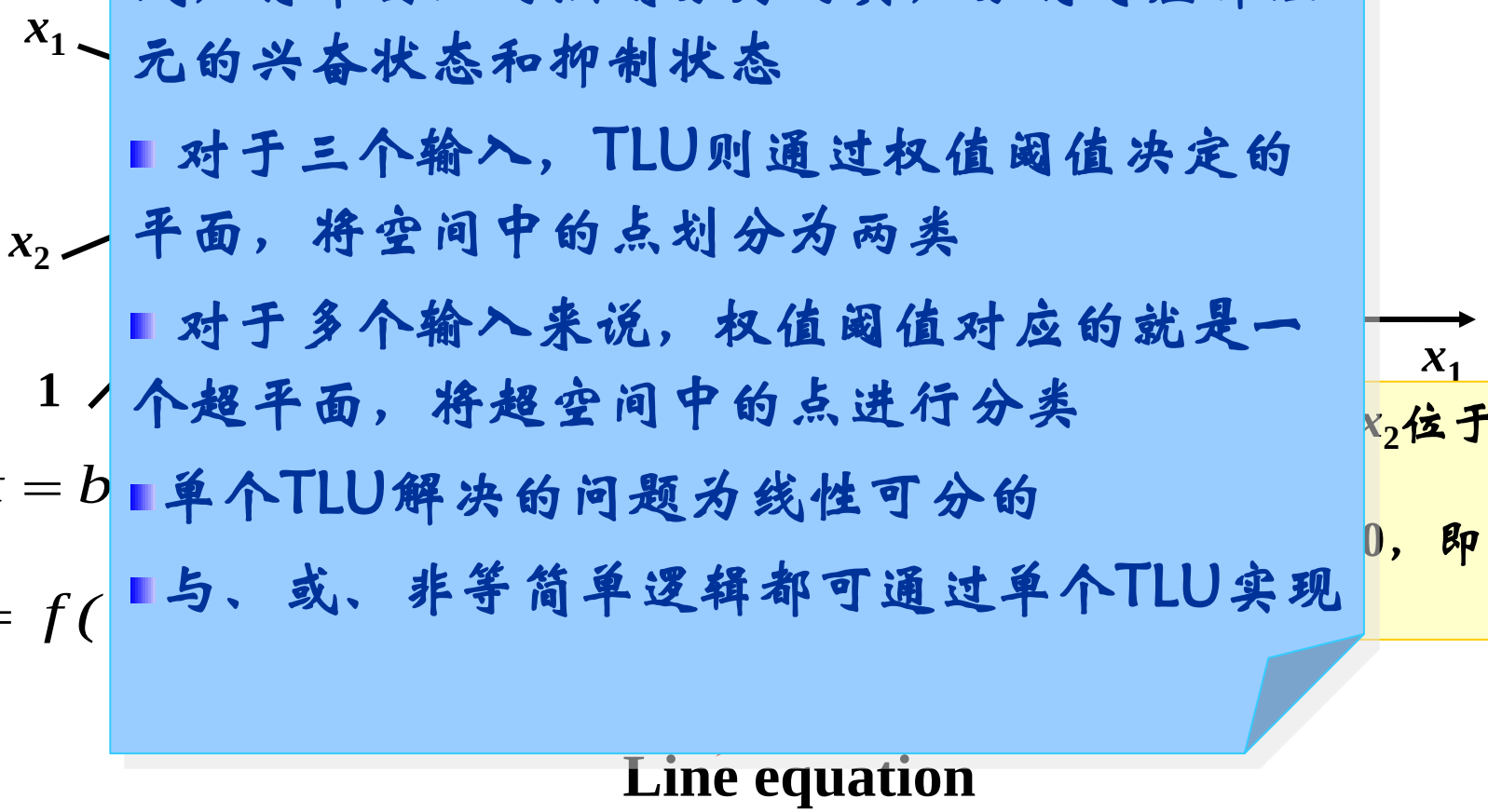
- 1958年由Rosenblatt 提出，用于将输入分为两类
- 简单的单层前馈网络
- 其神经元为一个线性阈值单元(Linear Threshold)，也称阈值逻辑单元(Threshold Logic Unit, TLU)



$$y = f(w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n) = \begin{cases} 1 & w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq 0 \\ 0 & ontherwise \end{cases}$$

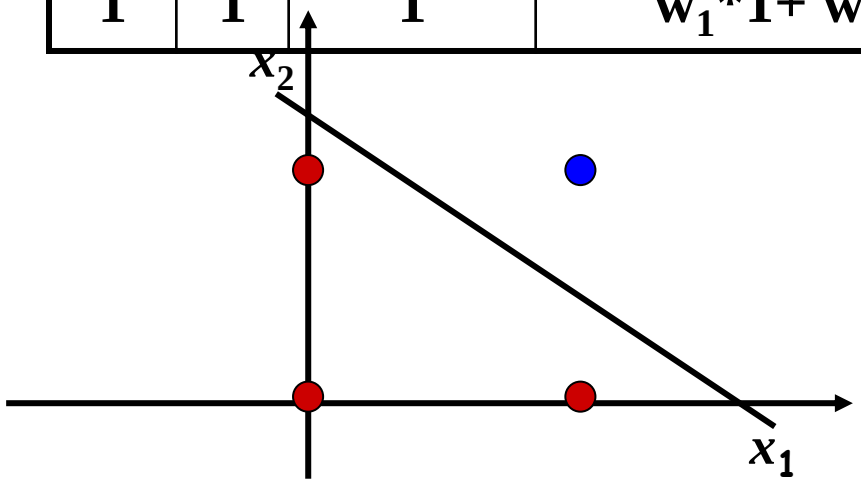
- 当有两个输入x_1和x_2时，若将x_1和x_2分别看成平面上的横轴

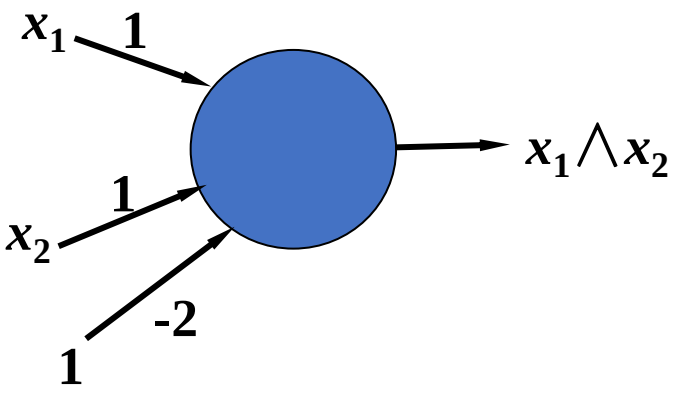
- 对于两个输入，TLU通过权值阈值决定的直线，将平面上的点划分为两类，分别对应神经元的兴奋状态和抑制状态
- 对于三个输入，TLU则通过权值阈值决定的平面，将空间中的点划分为两类
- 对于多个输入来说，权值阈值对应的就是一个超平面，将超空间中的点进行分类
- 单个TLU解决的问题为线性可分的
- 与、或、非等简单逻辑都可通过单个TLU实现



TLU 实现 AND

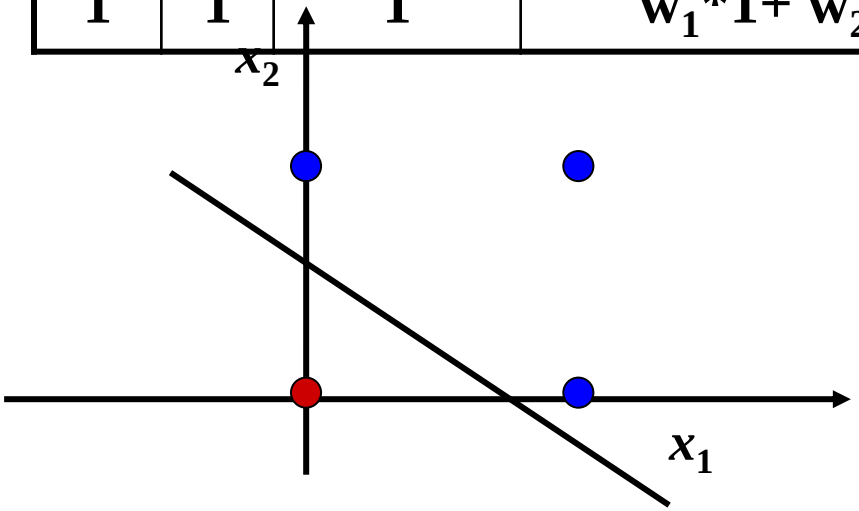
input		output	function	condition
x_1	x_2	$x_1 \wedge x_2$	$w_1*x_1+ w_2* x_2+b=0$	
0	0	0	$w_1*0+ w_2*0 +b<0$	$b < 0$
0	1	0	$w_1*0+ w_2*1 +b<0$	$-b > w_2$
1	0	0	$w_1*1+ w_2*0 +b<0$	$-b > w_1$
1	1	1	$w_1*1+ w_2*1+b\geq 0$	$-b \leq w_1+ w_2$

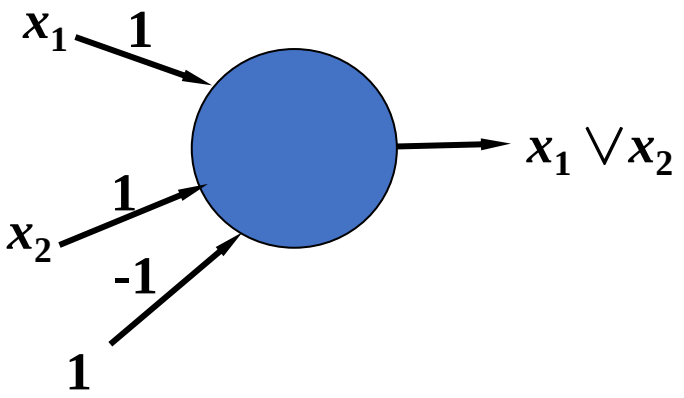




TLU 实现 OR

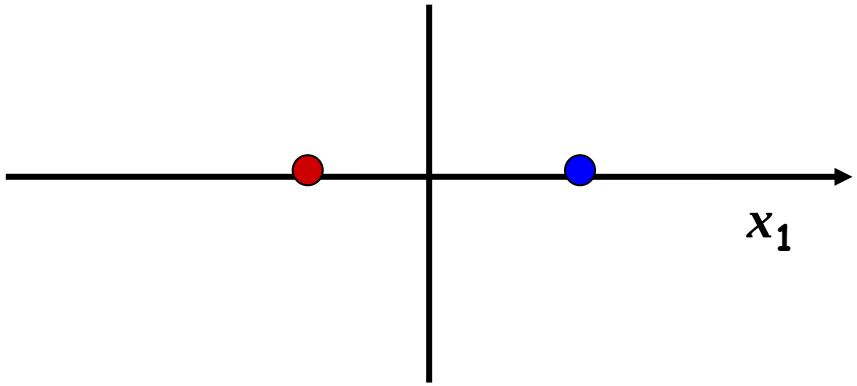
input		output	function	condition
x_1	x_2	$x_1 \vee x_2$	$w_1*x_1+ w_2* x_2+b=0$	
0	0	0	$w_1*0+ w_2*0 +b<0$	$b < 0$
0	1	1	$w_1*0+ w_2*1 +b \geq 0$	$-b < w_2$
1	0	1	$w_1*1+ w_2*0 +b \geq 0$	$-b < w_1$
1	1	1	$w_1*1+ w_2*1+b\geq 0$	$-b \leq w_1+ w_2$

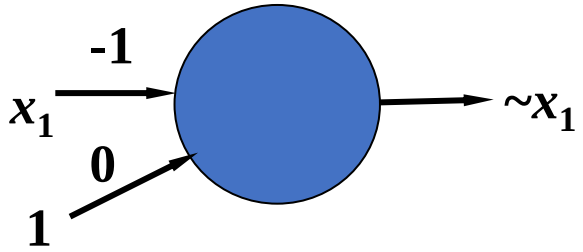




TLU 实现 NOT

input	output	function	condition
x_1	$\sim x_1$	$w_1 * x_1 + b = 0$	
0	1	$w_1 * 0 + b \geq 0$	$b \geq 0$
1	0	$w_1 * 1 + b < 0$	$-b > w_1$





NOT: Let threshold be 0, single input with a negative weight

Perceptron—Limits

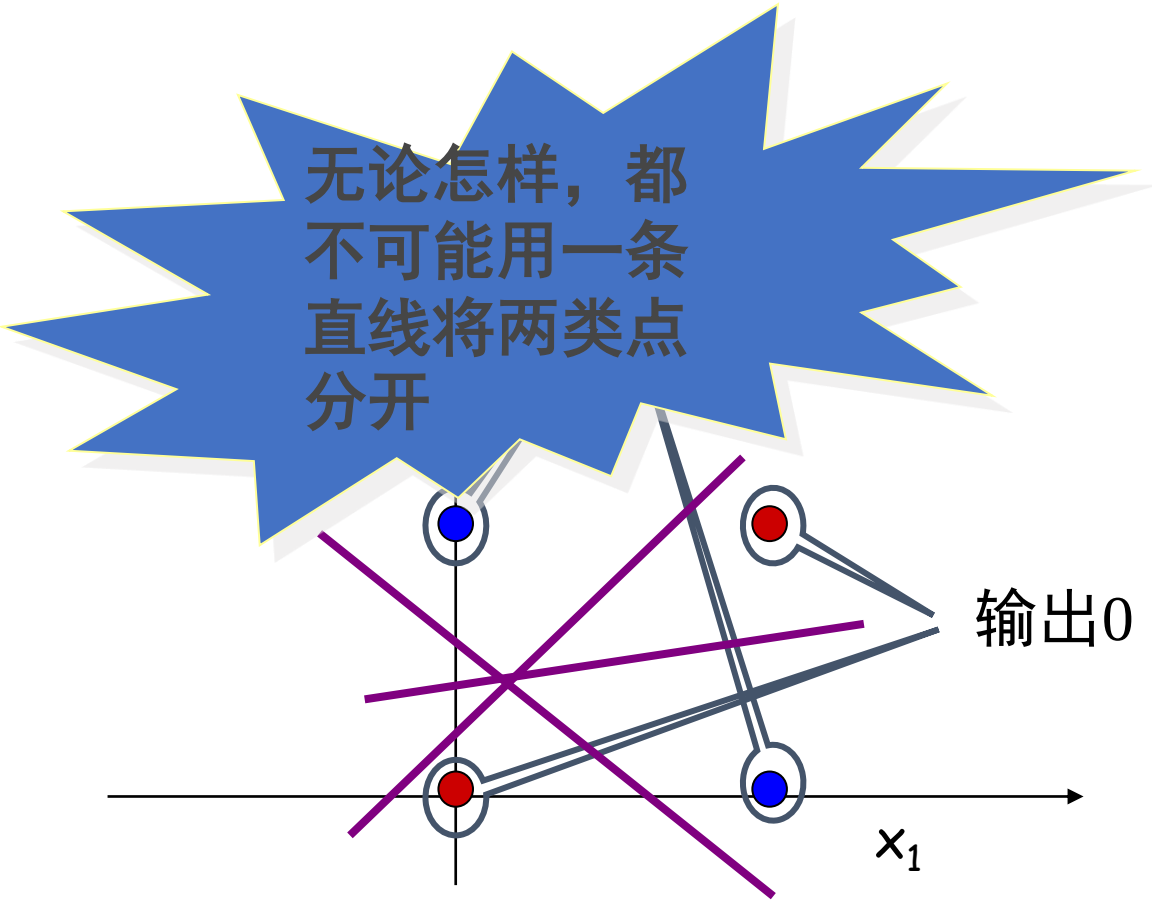
- 1969年，Minsky和Papert合作发表了颇有影响的Perceptron一书，分析了很多感知器无法解决的问题，得出了消极悲观的论点
- 指出感知器无法解决线性不可分问题

Perceptron—Limits

- 例：异或问题

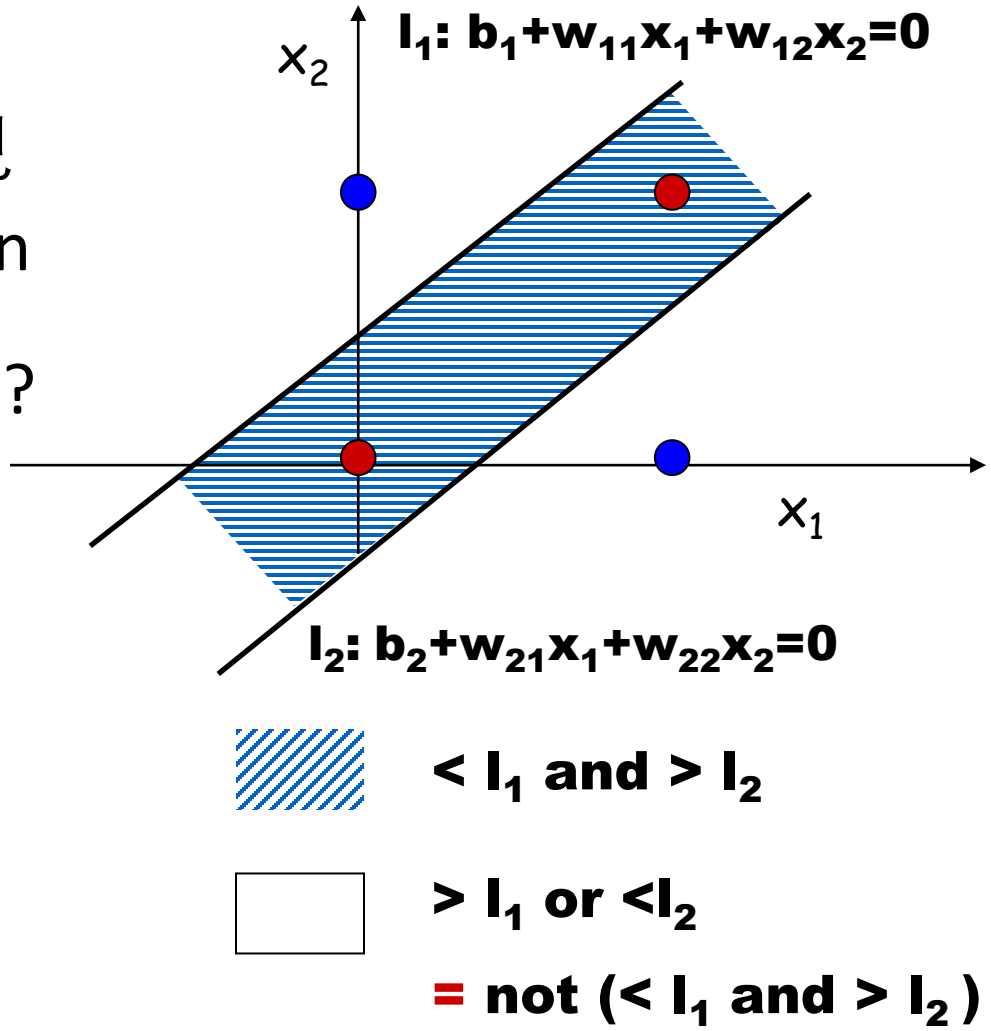
input		output
x_1	x_2	x_1 xor x_2
0	0	0
0	1	1
1	0	1
1	1	0

异或逻辑真值表

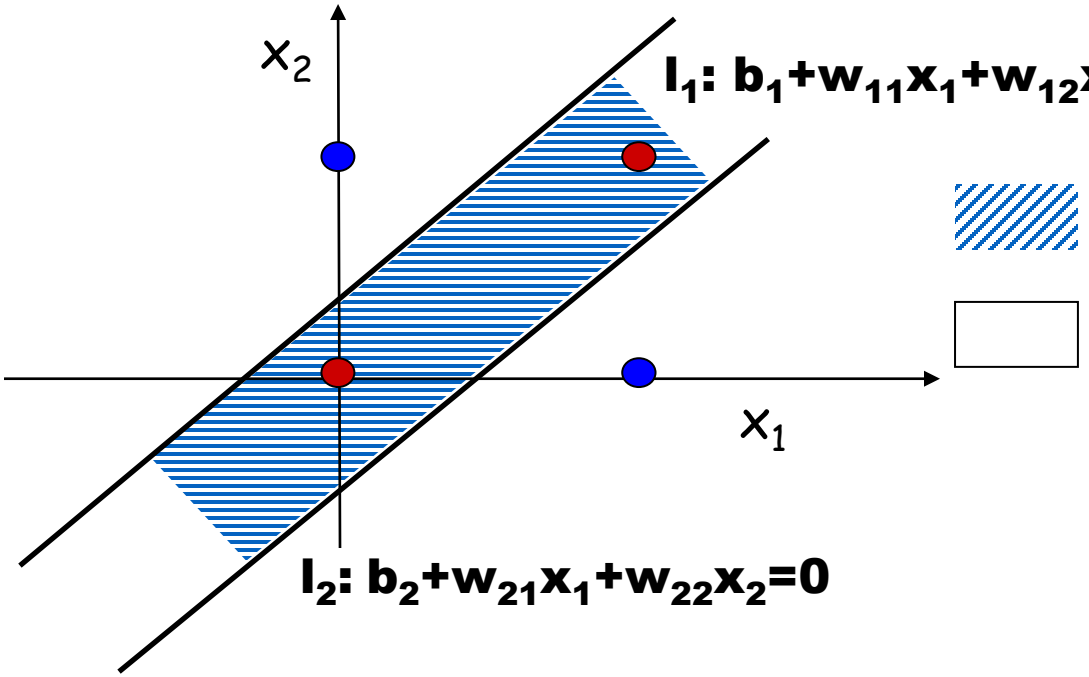


Perceptron—XOR

- 异或不是线性可分逻辑，无法通过Perceptron实现
- What if perceptron is not sufficient to solve the problem?
  - 通过多个神经元互联构成网络，形成多条直线对平面上的点进行分类
  - Multilayer Perceptrons



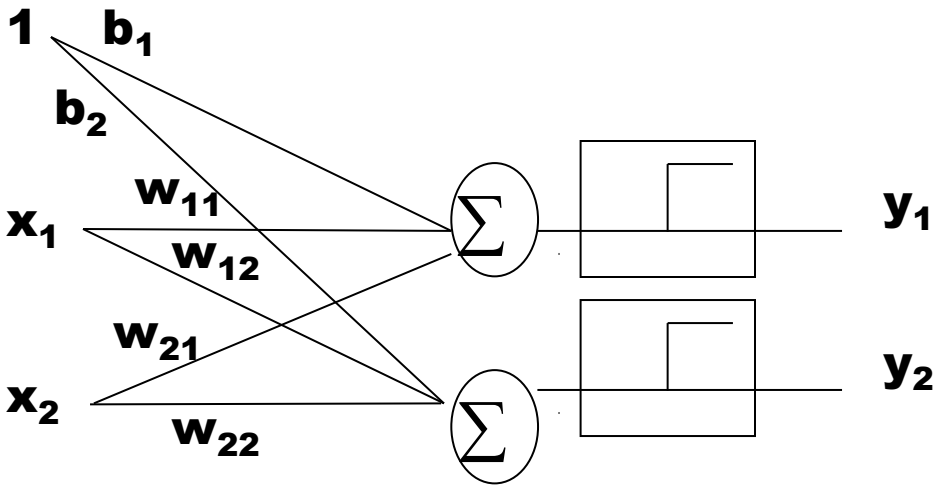
Perceptron—XOR: The first layer design



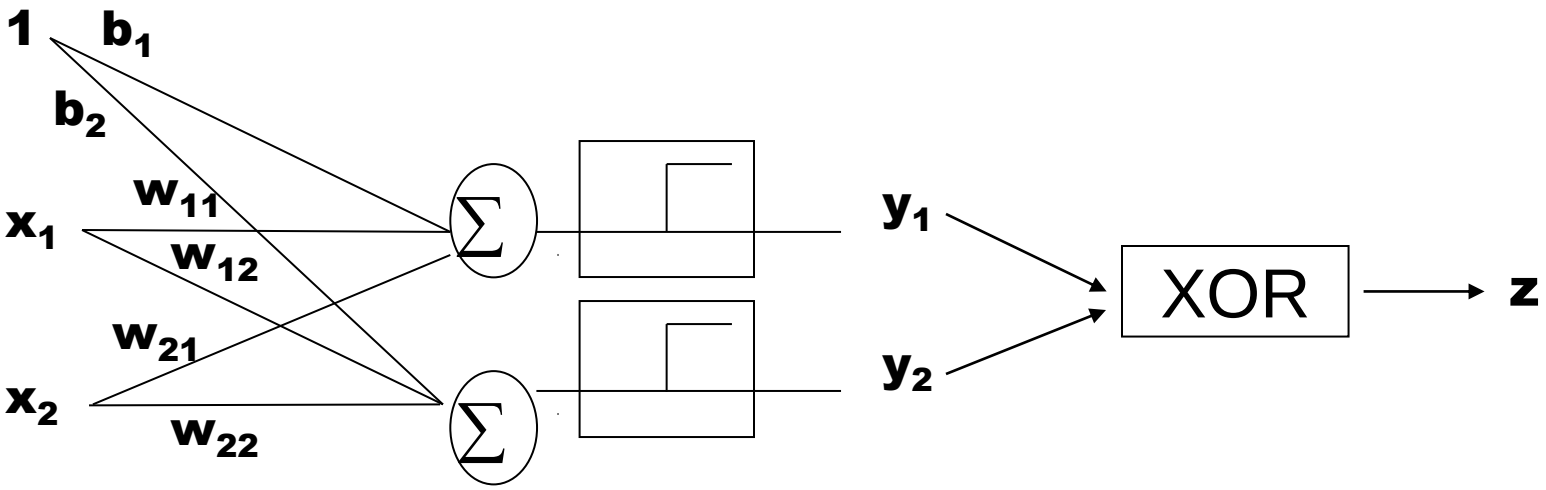
$< l_1$ and $> l_2$

$> l_1$ or $< l_2$

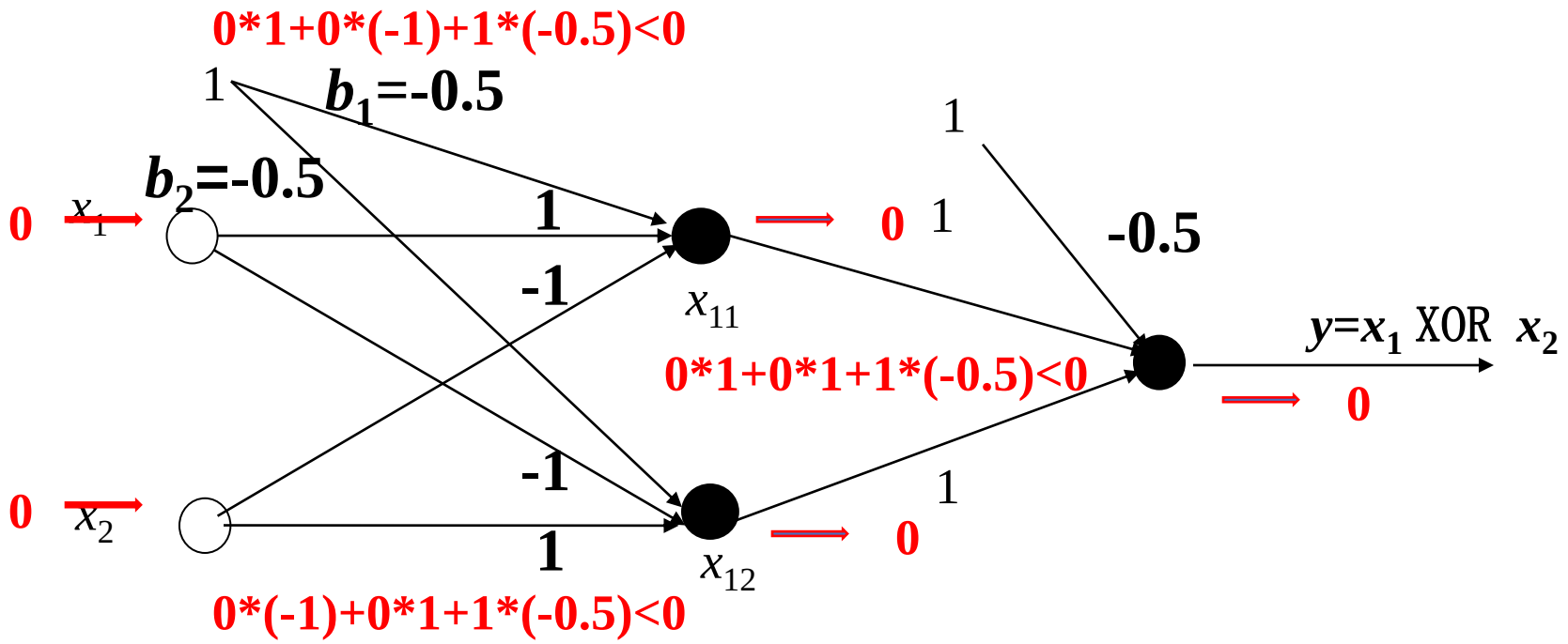
= not ($< l_1$ and $> l_2$)



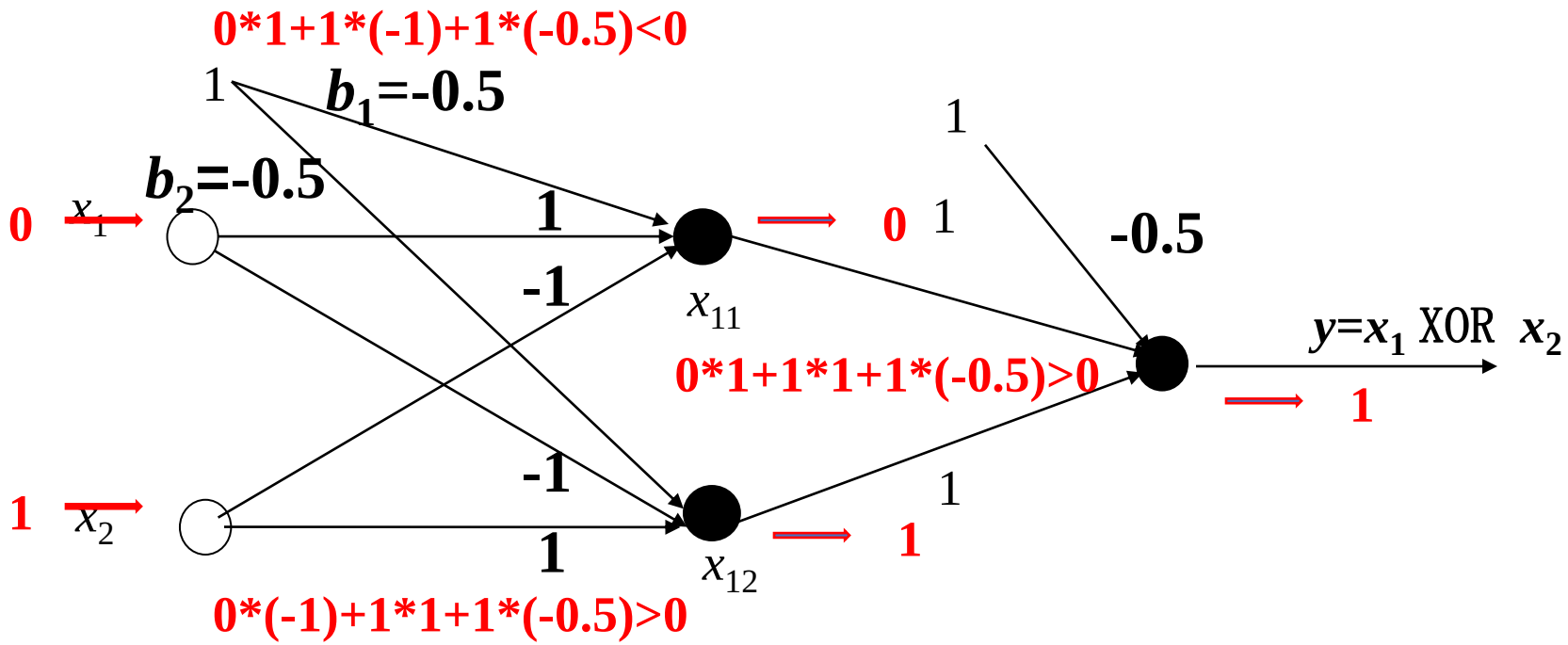
Perceptron—XOR: The second layer design



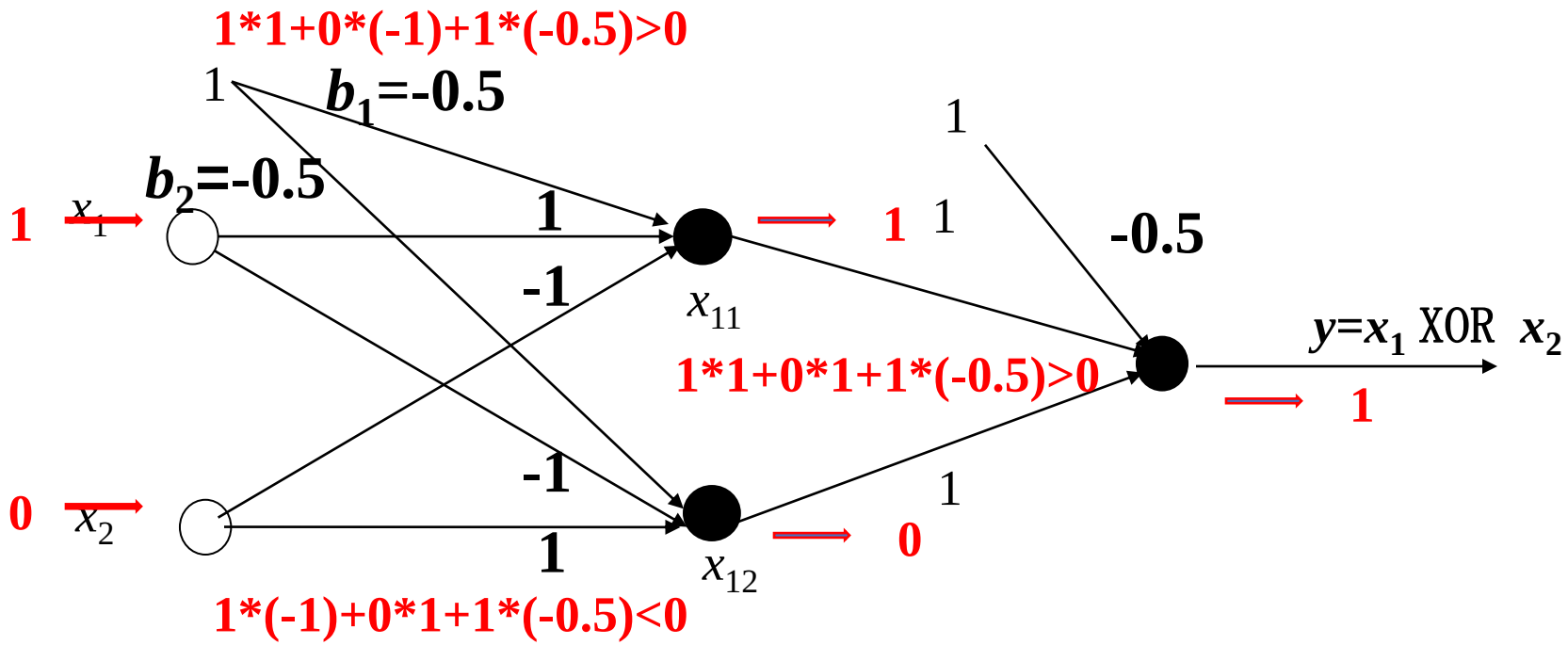
Multiple Layer Perceptrons Example： XOR



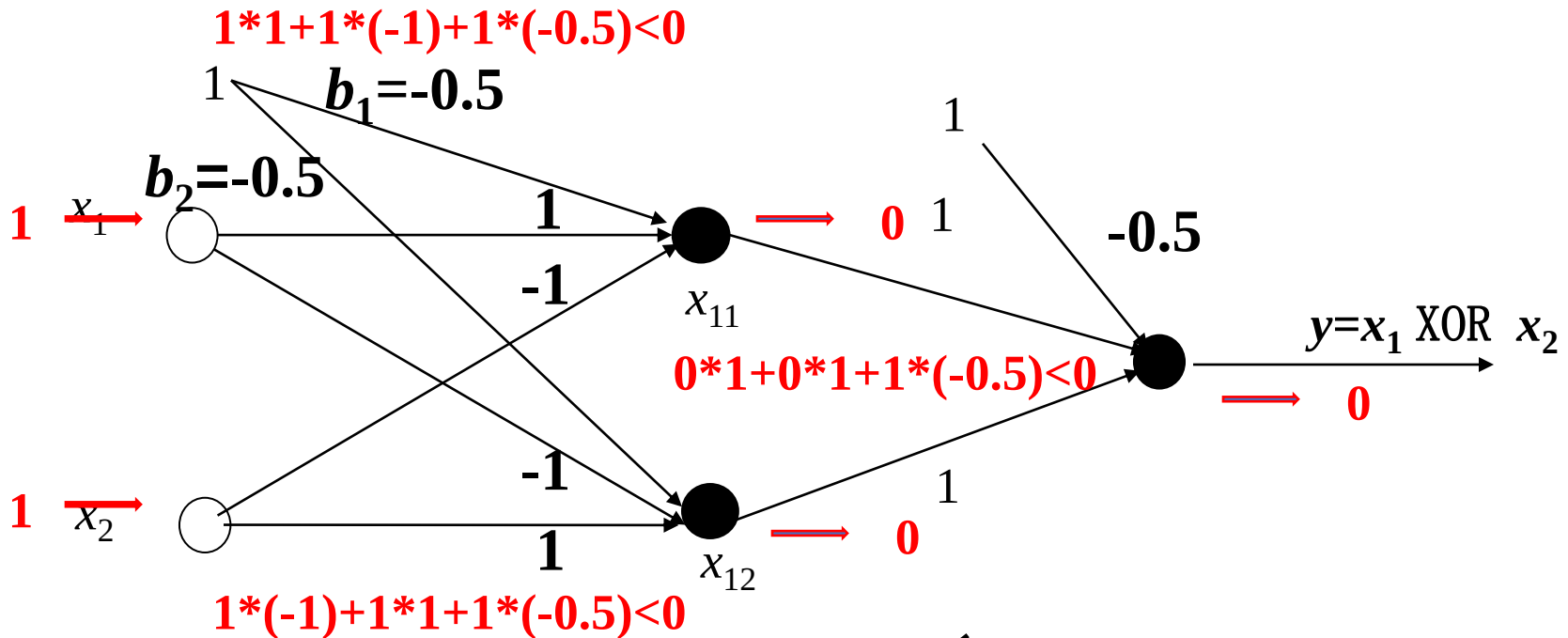
Multiple Layer Perceptrons Example： XOR

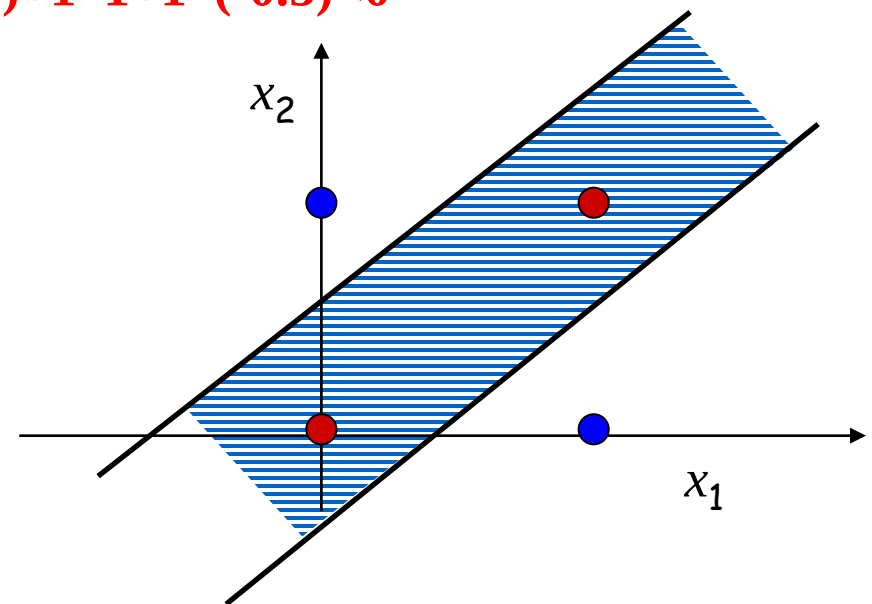


Multiple Layer Perceptrons Example： XOR

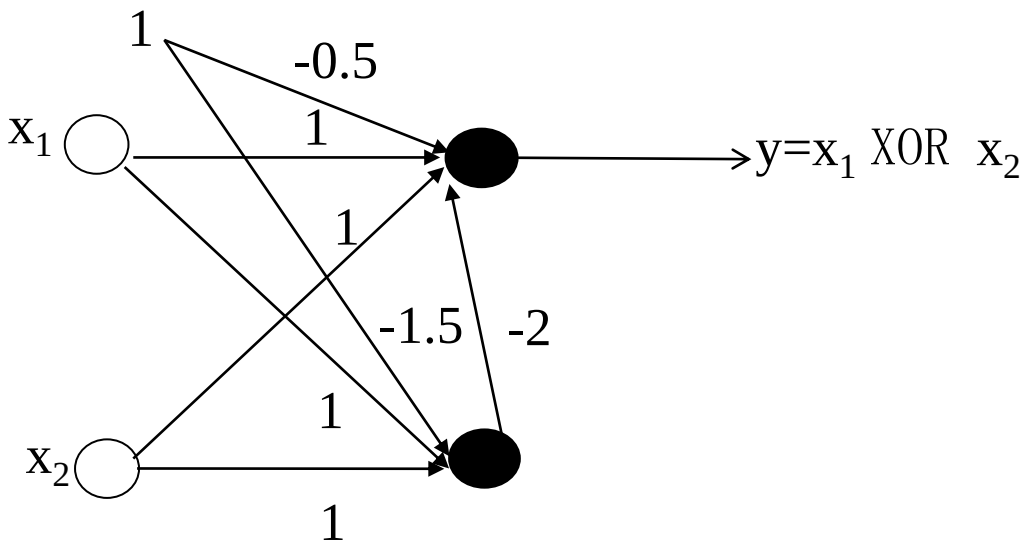


Multiple Layer Perceptrons Example: XOR



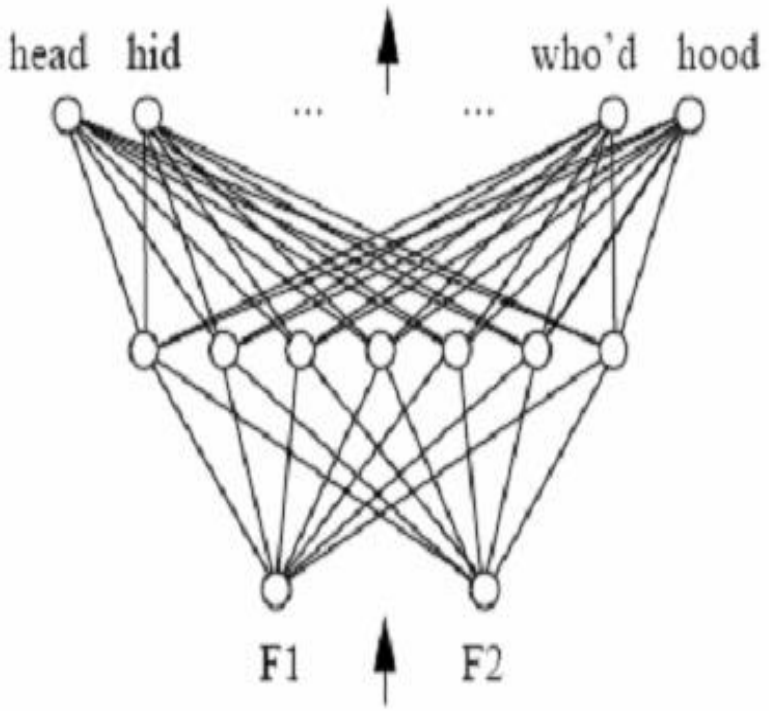


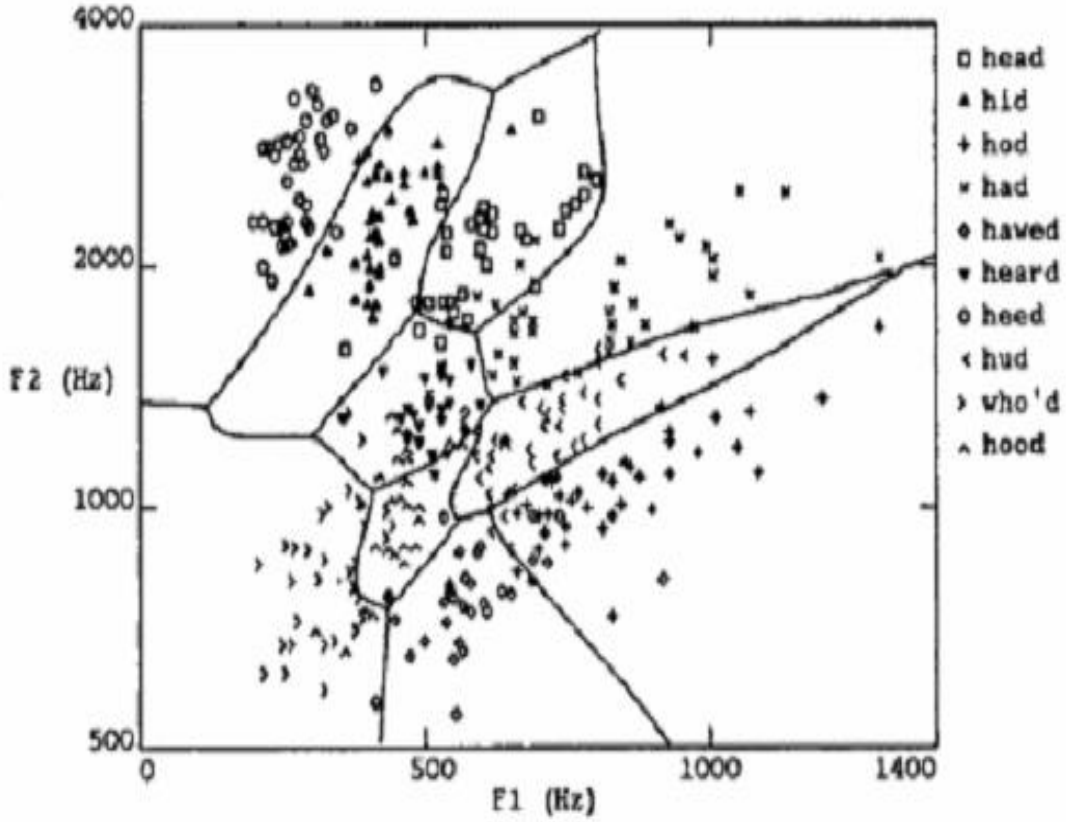
XOR也可用递归网络实现



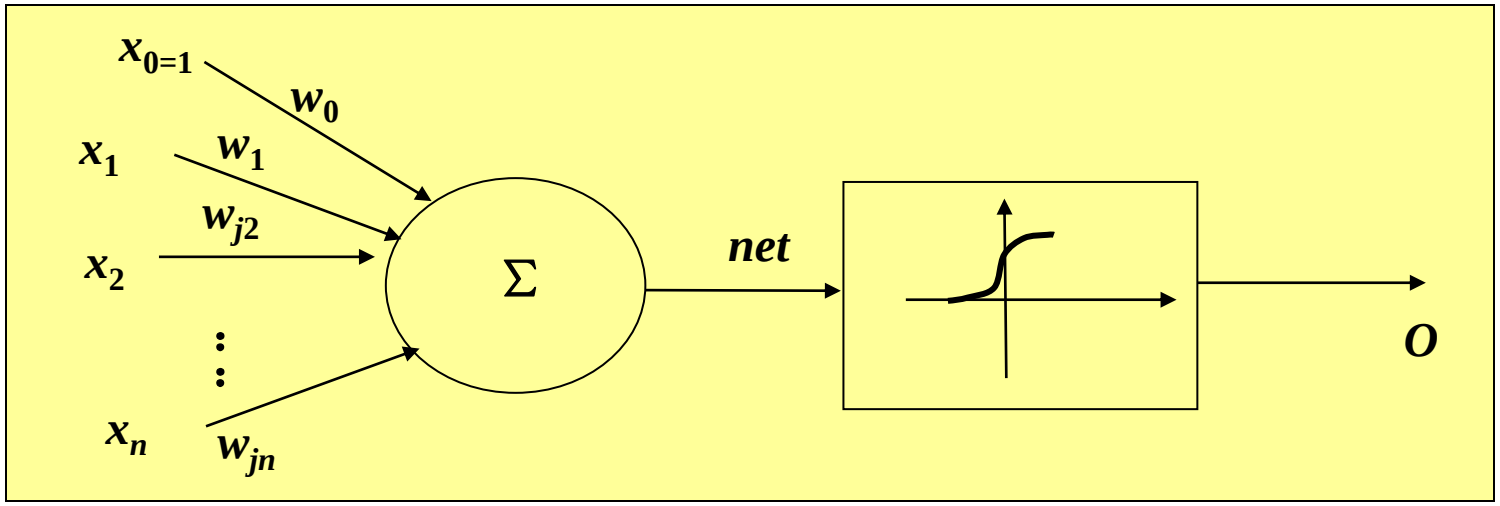
Sigmoid unit

- 当问题更加复杂，通过直线已经无法对空间进行有效划分时，则Sigmoid函数对应一个S型曲线，多个曲线相交可将空间进行更复杂的划分，完成所需任务

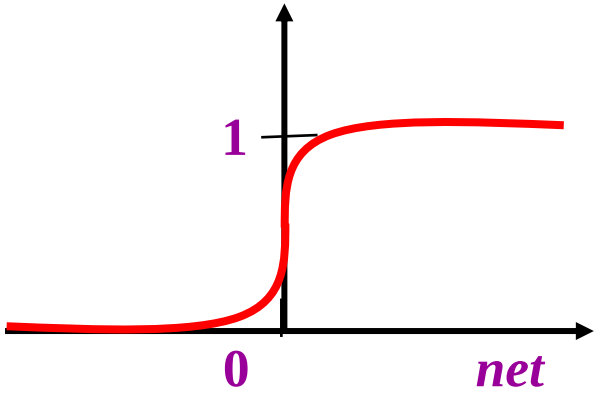




Sigmoid unit



$$o = \frac{1}{1+e^{-net}}$$



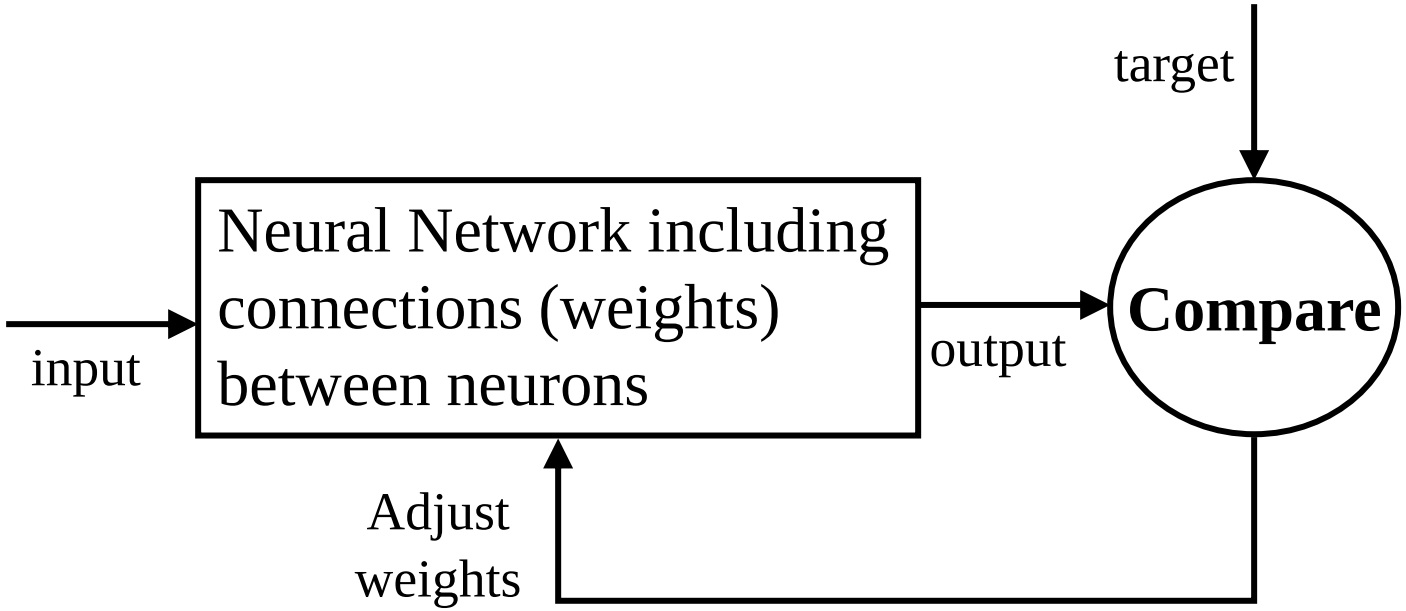
Learning Rule

- Learning rule: a **procedure (training algorithm)** for **modifying** the **weights** (权重)and the **biases** (偏置)of a network. (训练就是不断调整权重和偏置)
- The purpose of the learning rule is to **train** the network to **perform** some task. (目的是训练网络解决问题)
- Each kind of learning can be further classified into **three categories**:
  - ***supervised learning（监督学习）***
  - ***unsupervised learning（非监督学习）***
  - ***reinforcement learning（增强学习）***

Learning Rule —Supervised Learning

- 又称有师学习
- 需提供训练样例(***training set***) : $\{\mathbf{p}_1,\mathbf{t}_1\}$, $\{\mathbf{p}_2,\mathbf{t}_2\}$,…, $\{\mathbf{p}_q,\mathbf{t}_q\}$ $\mathbf{p}_i$ 是输入， $\mathbf{t}_q$ 是期望输出.
- 学习算法每次比较网络对应每个输入的实际输出和期望输出.
- 利用比较误差来调整网络权值.

Learning Rule — Supervised Learning



Learning Rule — Unsupervised Learning

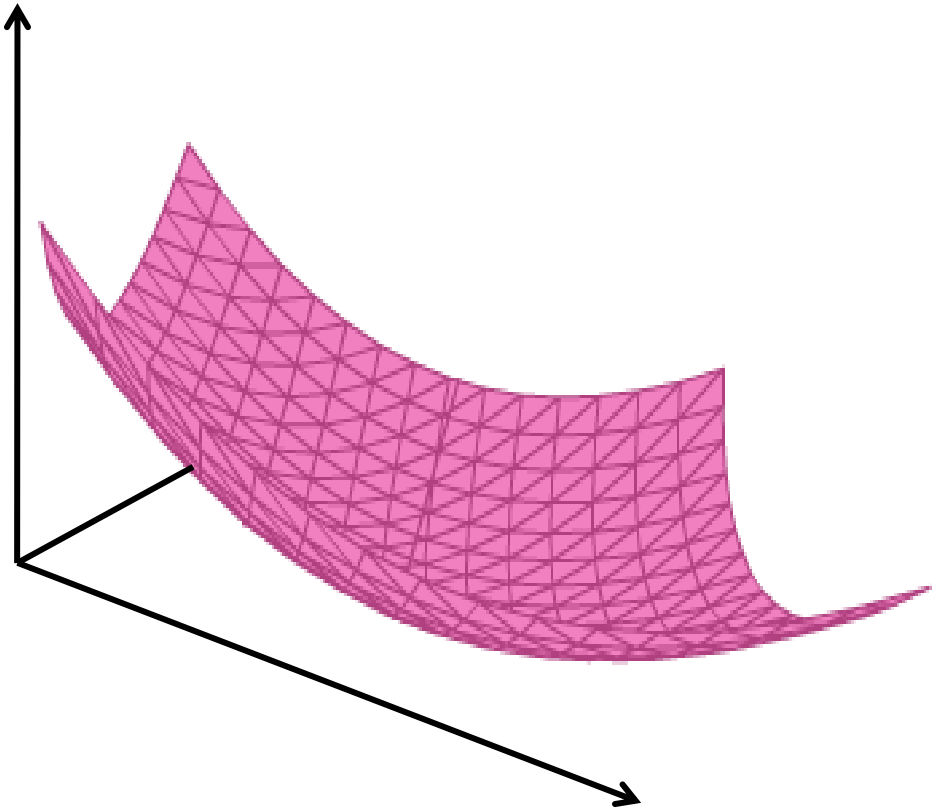
- 又称无师学习
- The **weights** and **biases** are modified in response to network inputs only. ***There are no target outputs available***. (没有目标地分类)
- Most of these algorithms perform some kind of **clustering(聚类) operation**. They learn to categorize the input patterns into a **finite** number of classes. (利用一些规则进行聚类,例如样本之间的相似度------欧氏距离)

Learning Rule — Reinforcement Learning

- The learning rule is **similar to supervised learning**, except that, instead of being provided with the correct output for each network input, ***the algorithm is only given a grade***.(和监督学习一样有指导,但只给一个程度,等级)
- The **grade (score)** is a measure of the **network performance** over some sequence of inputs.()
- It appears to be most suited to **control system applications**.(适用于控制系统)
- Example: Genetic Algorithm（GA）(典型算法:进化计算,遗传算法)

Learning Rule —Perceptron

- 类似于爬山
- 搜索空间是有**权值和阈值**组成的集合
- 学习的**目的**是使在训练集上的**误差最小**



Perceptron Learning Rule

- 权值的调整采用有师学习的方式进行
- Update weights by:

$$w_{ji} = w_{ji} + \eta(t_j - o_j)o_i$$

η ——学习常数（learning rate）

t_j ——j的期望输出

O_j ——j的实际输出

O_i ——i的实际输出（ j的输入）

- Equivalent to rules:
  - If output is correct do nothing.
  - If output is high, lower weights on active inputs
  - If output is low, increase weights on active inputs

Learning Rule — Perceptron Learning Algorithm

- 感知器学习算法步骤

  输入：给定正例集合P和反例集合N，对所有$x \in P$，$f(x) = 1$，所有$x \in N$，$f(x) = 0$

  输出：$w \in R^{n+1}$

1.Initialize weights to

$$w = \sum_{x \in P} x - \sum_{x \in N} x$$

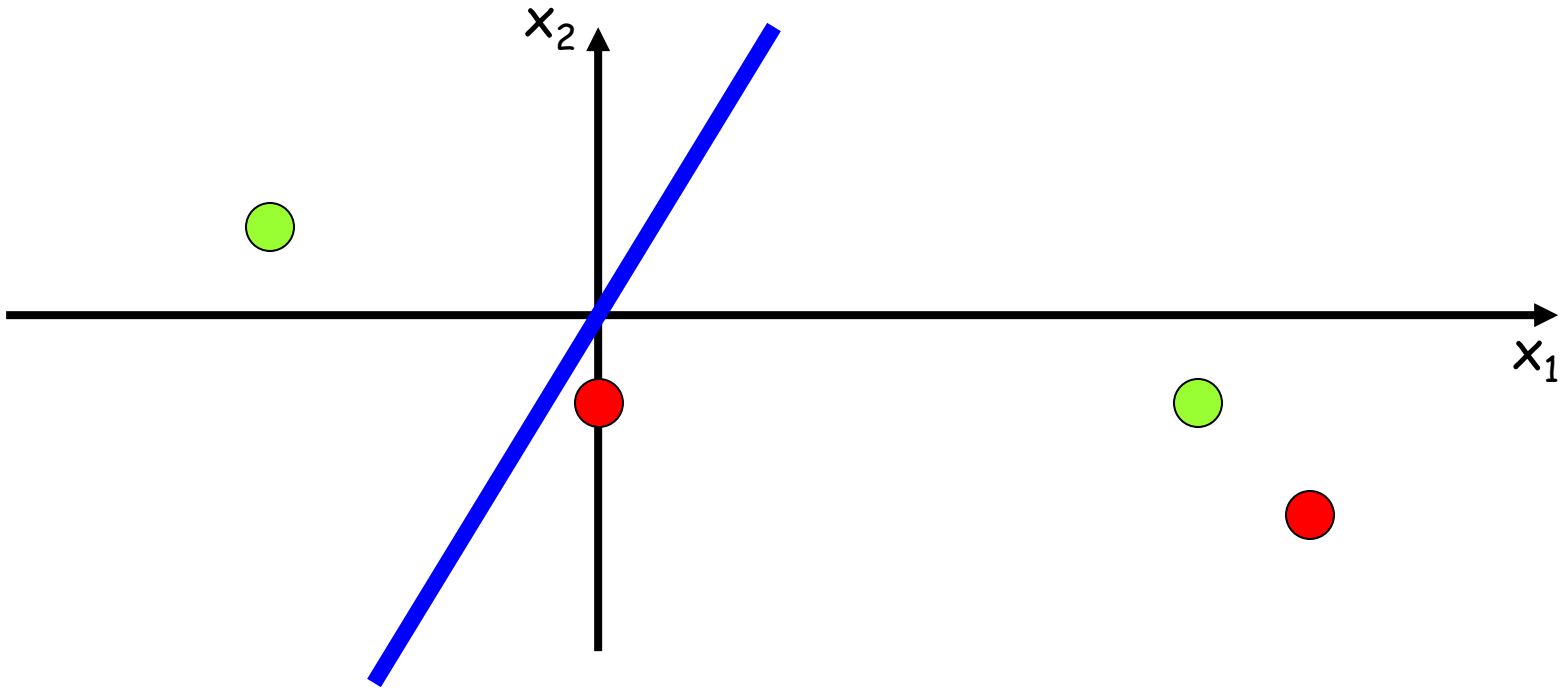
2.Choose $x \in P \cup N$ randomly

3.Update $w = w + \eta(t-o)x$（η为学习常数，t为期望输出，o为实际输出）

4.Goto 2 until outputs of all training examples are correct

Perceptron learning: Example

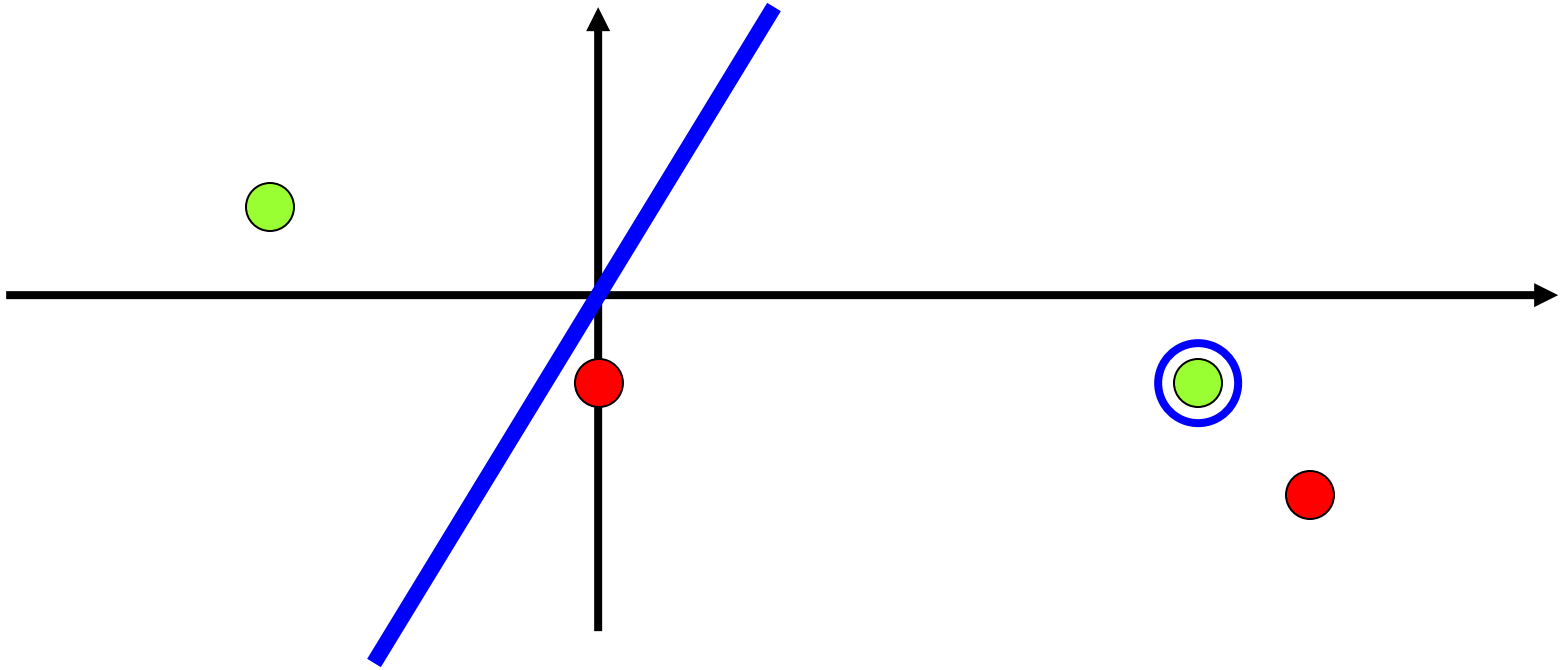
P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$



w = (6,−1) + (−3, 1) − (0,−1) − (7,−2) = (−4, 3)

Perceptron learning: Example

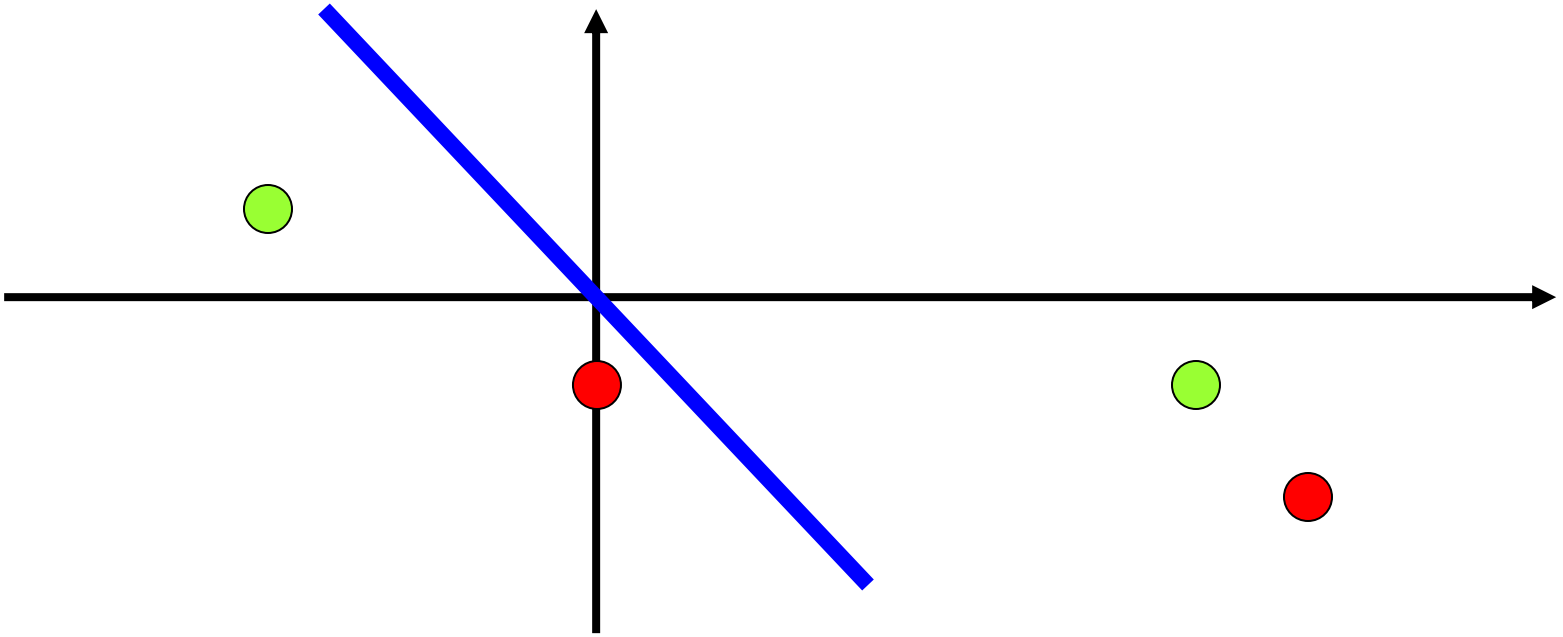
P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$



w ・ x < 0
w = w + (1-0)x =(-4, 3)+(6, -1)=(2, 2)

Perceptron learning: Example

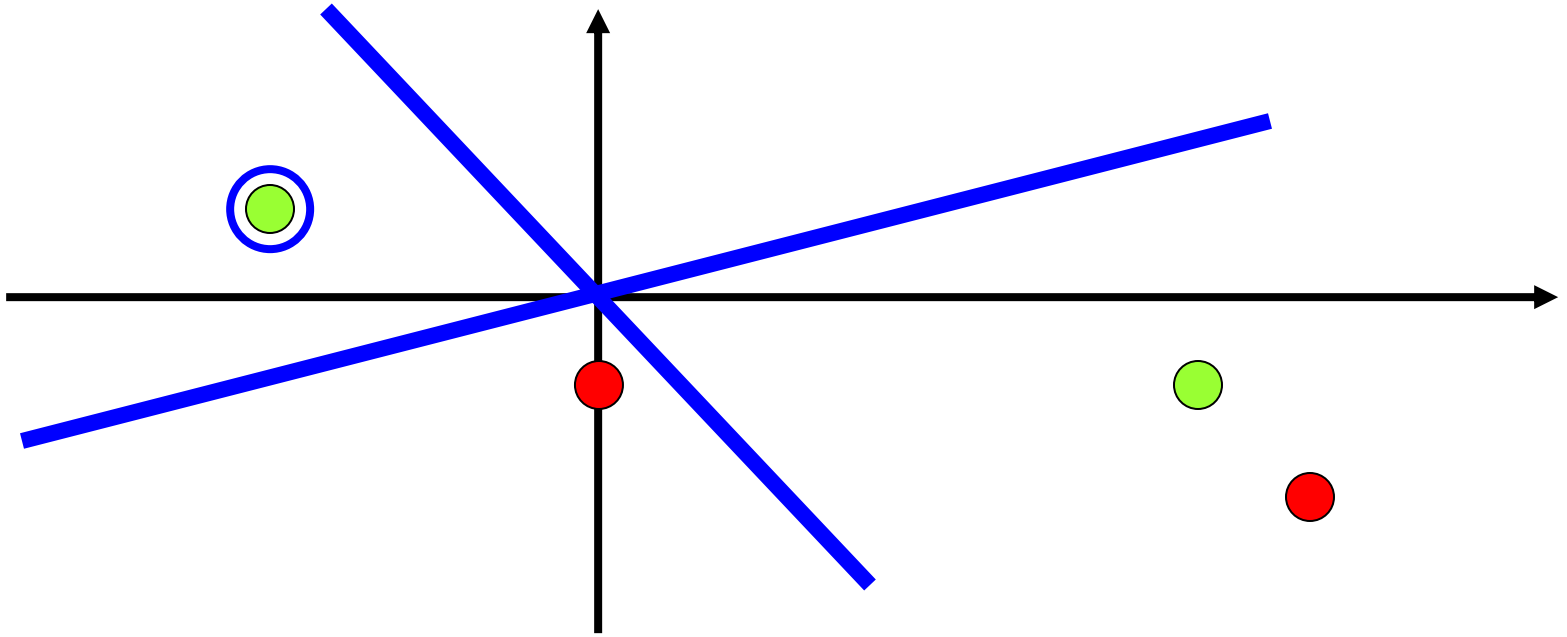
P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$



w ・ x < 0
w = w + (1-0)x =(-4, 3)+(6, -1)=(2, 2)

Perceptron learning: Example

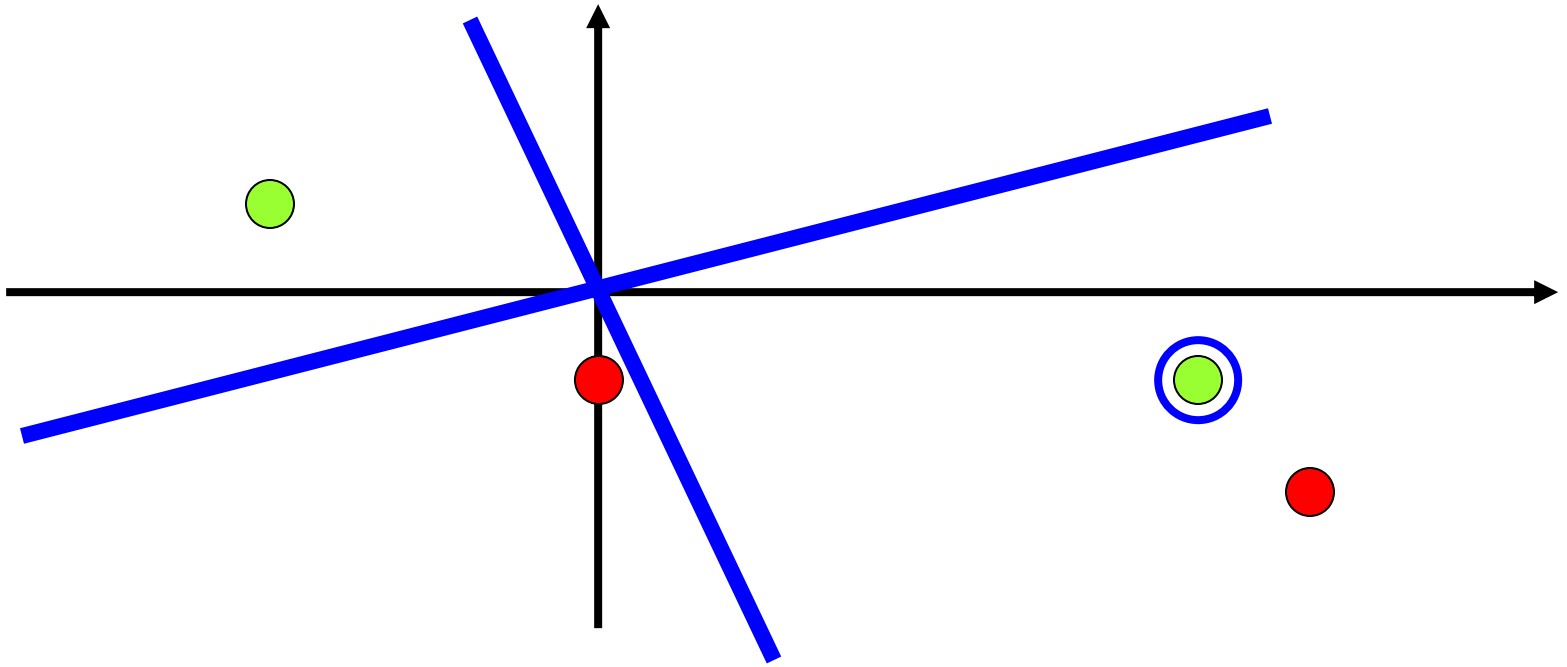
P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$



w ・ x < 0
w = w + (1-0)x =(2,3)+(6, −1)=(8, 2)
w = w + (1-0)x =(2,3)+(−3, 1)=(−1, 4)

Perceptron learning: Example

P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$

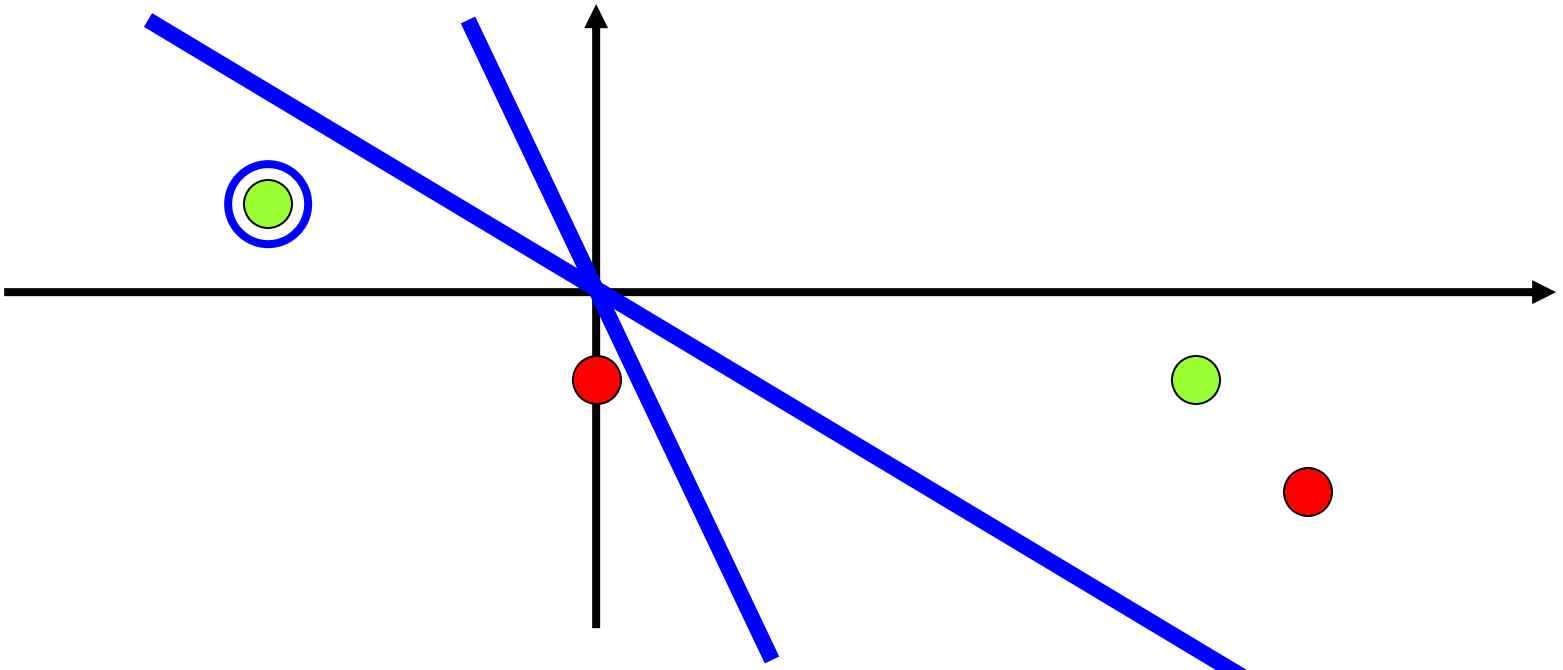


w ・ x < 0
w = w + (1-0)x =(1,2)+(6, −1)=(7, 1)

Perceptron learning: Example

P = {(6,−1), (−3, 1)}

N= {(0,−1), (7,−2)} $\eta = 1$

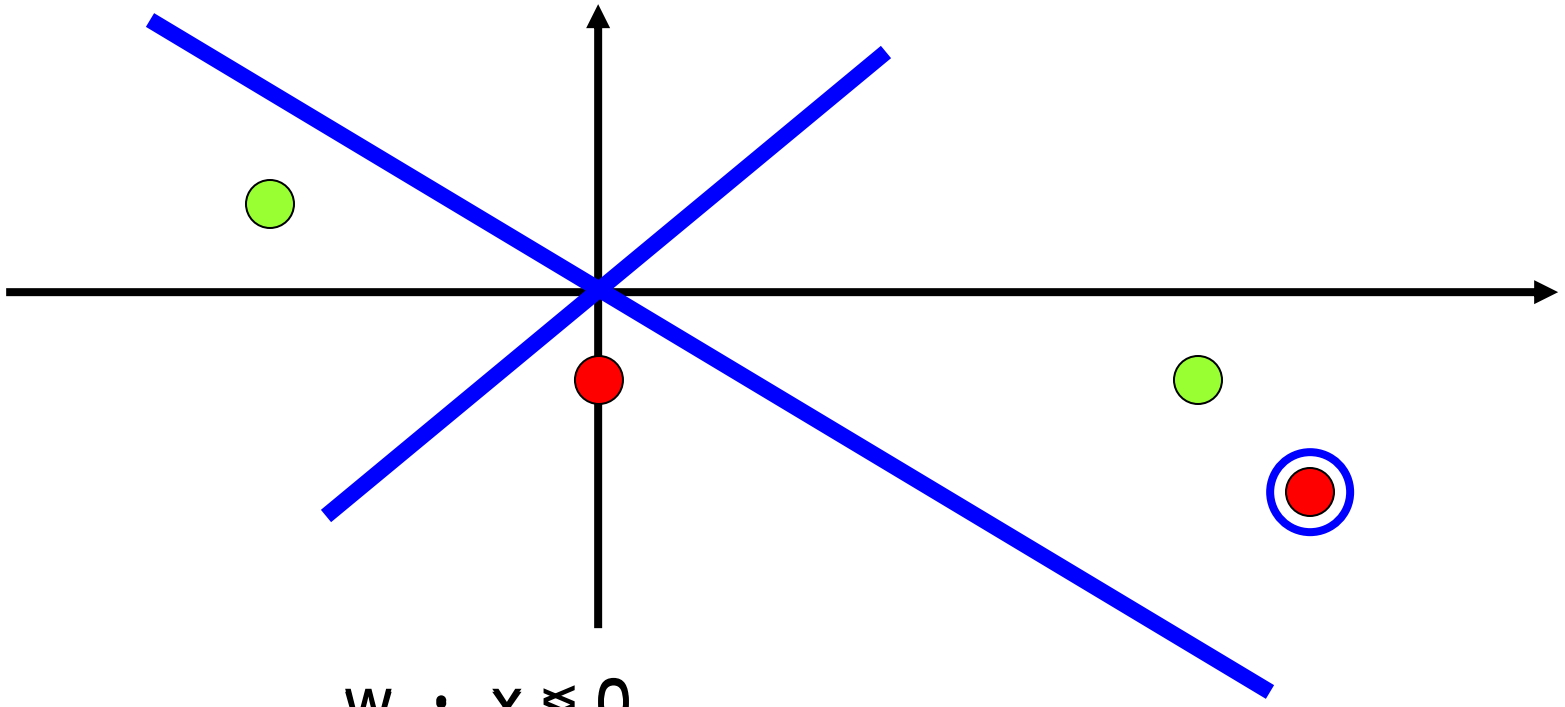


w · x < 0

w = w + (1-0)x = (5,2) + (6, −1) = (11, 1)

Perceptron learning: Example

P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$

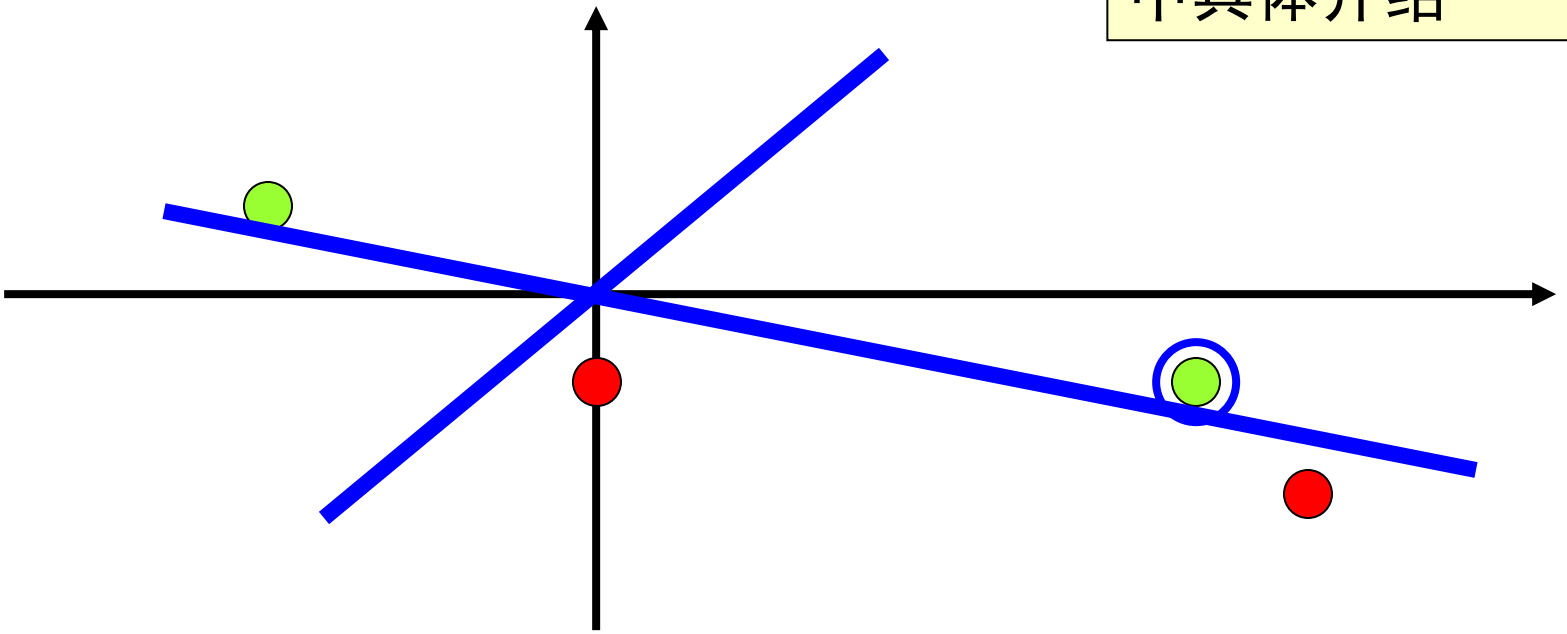


w · x ≤ 0
w = w + (1-1)x =(2, 3)+(7,-2)=(9, 1)

Perceptron learning: Example

P = {(6,−1), (−3, 1)}
N= {(0,−1), (7,−2)} $\eta = 1$

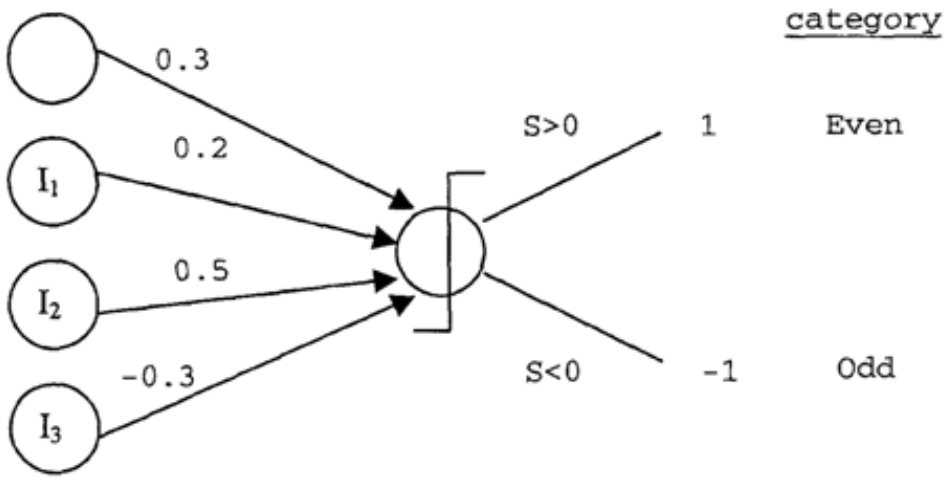
其他神经网络学习方法将在机器学习中具体介绍



$w \cdot x \leq 0$

$w = w + (0-1)x = (5,3) + (7,2) = (-2,1)$

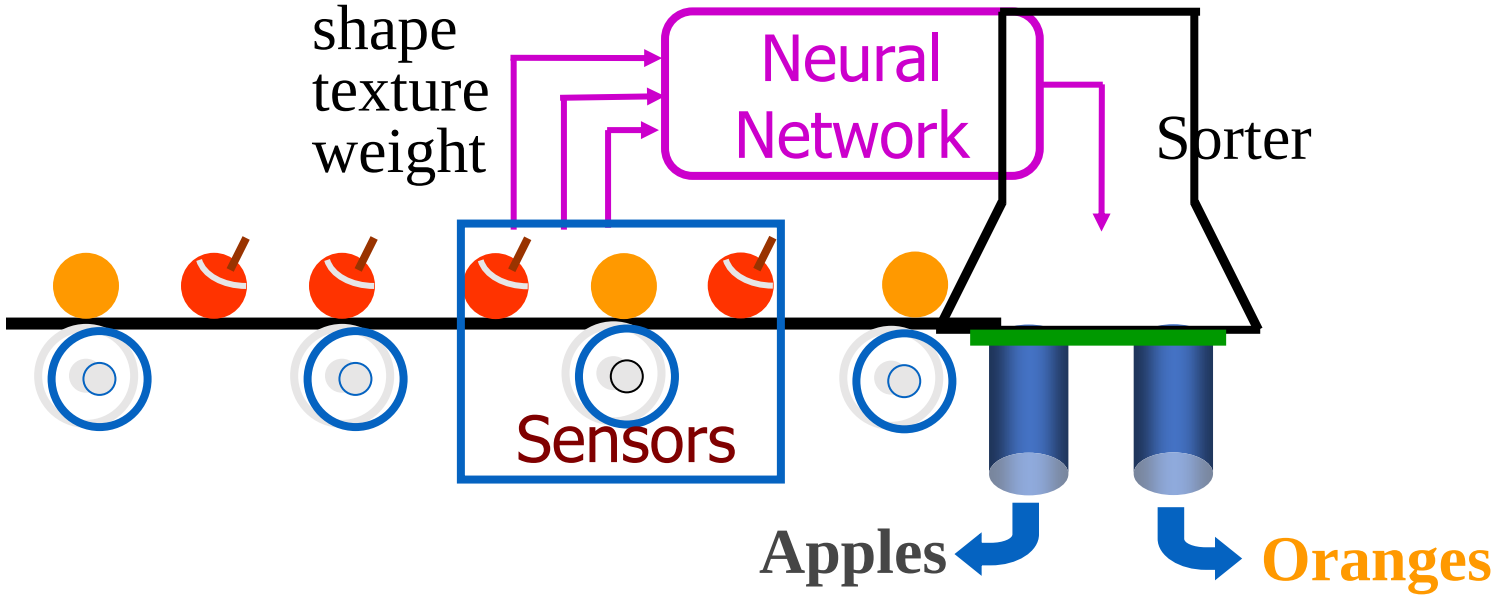
Question



- 训练一个感知器用以判断输入的三个整数之积是偶数还是奇数。该感知器有三个输入I_1~I_3分别对应输入三个整数，若是偶数，则输入值为+1，否则为-1
  - 为什么需要第4个输入端？其输入值应该设为多少？
  - 对于2*3*4，该感知判断其结果是奇数还是偶数？

ANN application—Example

- Pattern Recognition
  - Problem Statement



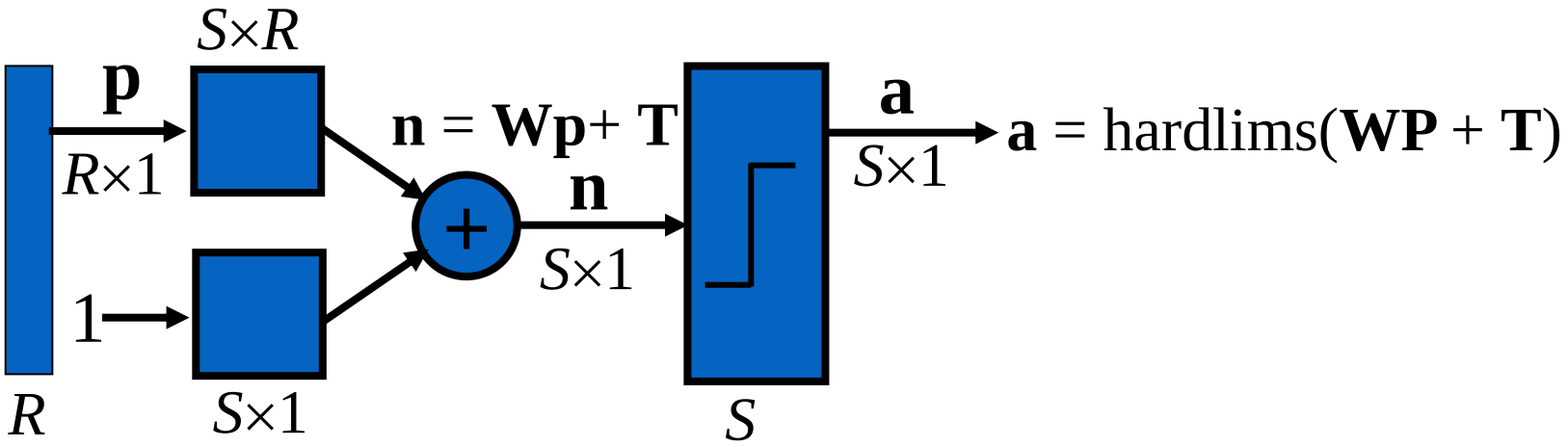
ANN application—Example

- Shape sensor: 1 ——round, –1 ——elliptical.
- Texture sensor: 1 —— smooth, –1 —— rough.
- Weight sensor: 1—— > 1 pound, –1 —— < 1 pound.

$$\mathbf{p} = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix} \Rightarrow \mathbf{p}(apple) = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \ \mathbf{p}(orange) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

ANN application—Example

- Perceptron



Symmetrical Hard Limit

- a = -1, if $n < 0$
- a = +1, if $n \geq 0$
- MATLAB function: *hardlims*

ANN application—Example

$$\mathbf{p} = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix} \Rightarrow \text{three-dimensional input } (R = 3)$$

$$n = \mathbf{W}\mathbf{p} + T, \; a = \text{hardlims}(n)$$

Choose the **bias** T and the elements of the **weight matrix W** so that the perceptron can distinguish between **apples** and **oranges**.

ANN application—Example

$$\mathbf{p}(apple)=\begin{bmatrix}1\\1\\-1\end{bmatrix}\quad \mathbf{p}(orange)=\begin{bmatrix}1\\-1\\-1\end{bmatrix}\quad \Rightarrow \mathbf{W}=\begin{bmatrix}0 & 1 & 0\end{bmatrix},\ T=0$$

$$\textbf{Orange}: a=\text{hardlims}\left(\begin{bmatrix}0 & 1 & 0\end{bmatrix}\begin{bmatrix}1\\-1\\-1\end{bmatrix}+0\right)=-1$$

$$\textbf{Apple}: a=\text{hardlims}\left(\begin{bmatrix}0 & 1 & 0\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix}+0\right)=1$$

ANN application—Example

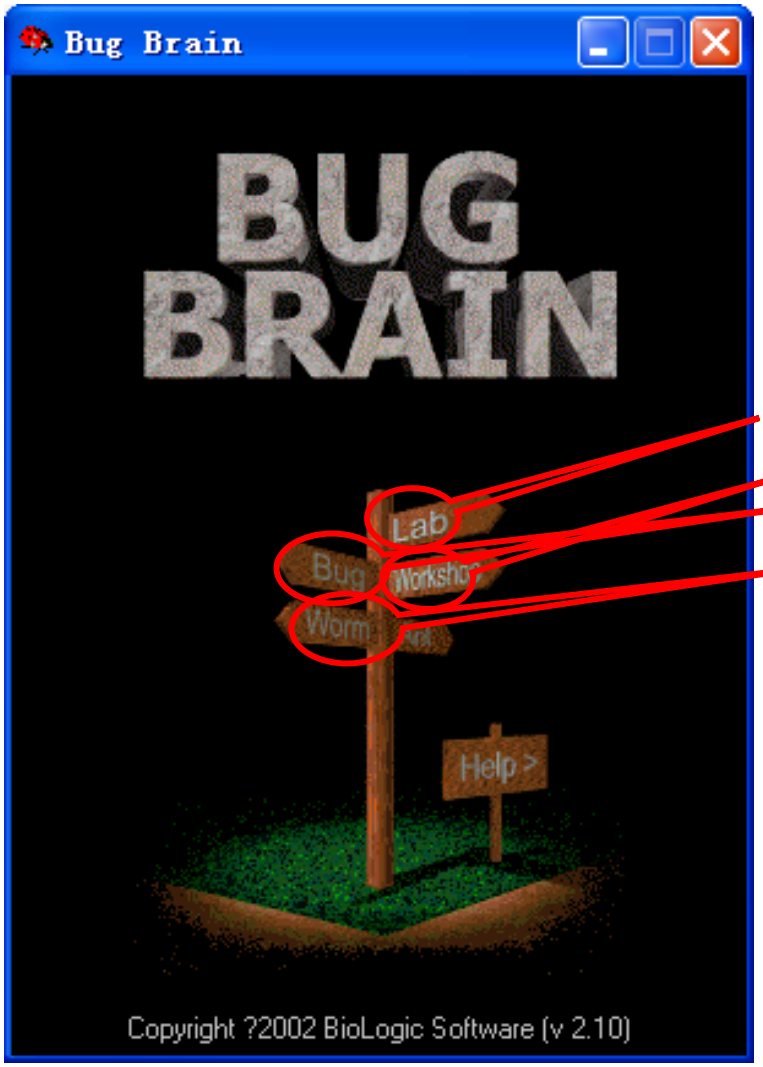
- What happens if we put a **not-so-perfect orange** into the classifier? That is to say, an orange with an **elliptical shape** is pass through the sensor.

$$\mathbf{p} = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix} \Rightarrow \quad \mathbf{p}(orange) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \Rightarrow \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$a = \text{hardlims}\left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + 0 \right) = -1 \Rightarrow \textbf{orange}$$

ANN Game—Bug Brain

- 神经网络被广泛应用于自动控制、模式识别等众多领域
- 让我们通过一个游戏，为瓢虫小姐构造一个大脑来看看神经网络的应用

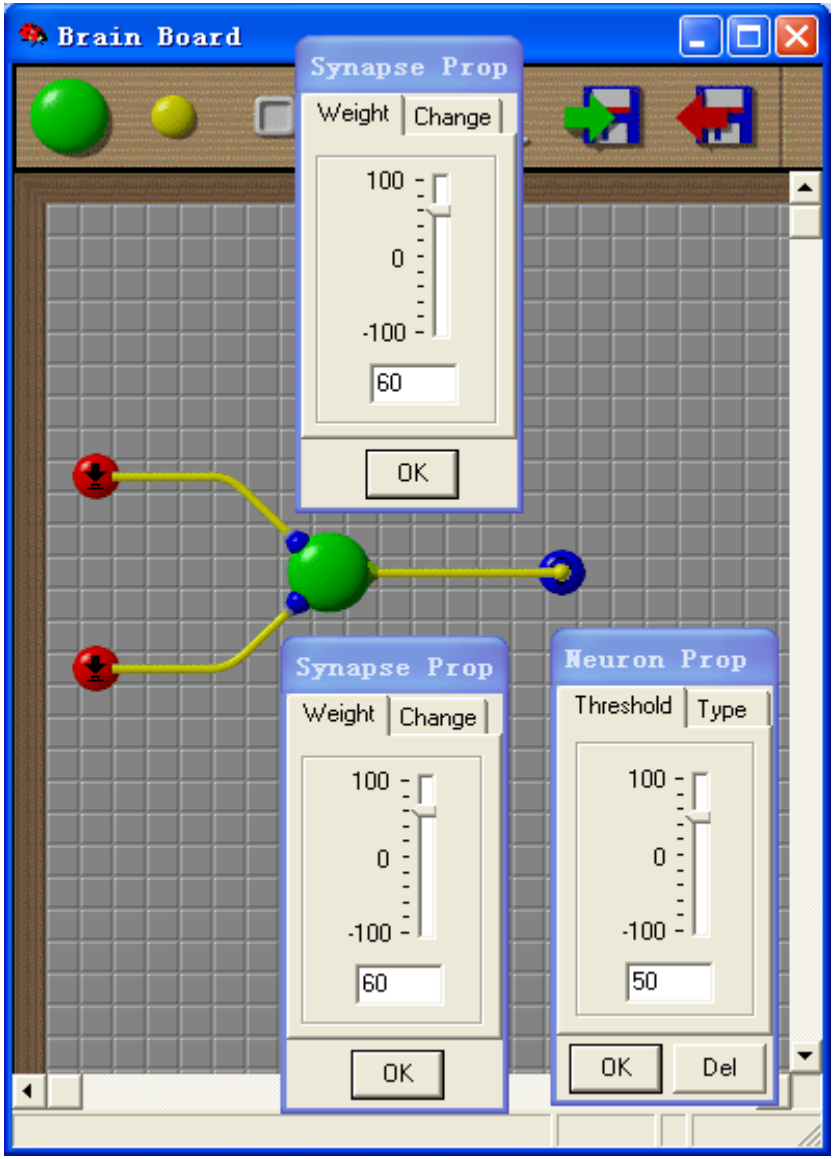


实验室
工厂
瓢虫世界
蠕虫世界

游戏可分为实验室、瓢虫世界、工厂、蠕虫世界等几关，每关难度增加

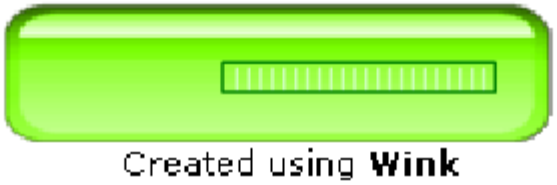
ANN Game—Bug Brain

- 实验室
  - 游戏从最简单的实验室任务开始，在这里，玩家被要求完成包括与、或、非在内的各种任务，搭建能够完成相应任务的神经网络并测试成功后才能进入下一任务
  - 实验室所有任务完成后可进入下一关
  - 例：构造完成“或”逻辑的神经网络如图，所有权值阈值都可自行设置以完成任务



ANN Game—Bug Brain

- 完成网络设计后，可进行测试，检验是否成功
- 例如：“或”任务要求在两个开关中的任何一个被按下时灯能够亮，测试我们的网络成功



ANN Game—Bug Brain

- 在进入瓢虫世界后任务变得更加实际，要求为瓢虫小姐来设计大脑，让她能够完成不同任务
- 如：在有宽度传感器（范围0~100）的情况下，如何让瓢虫小姐既能吃到蚜虫又不会掉下树干？