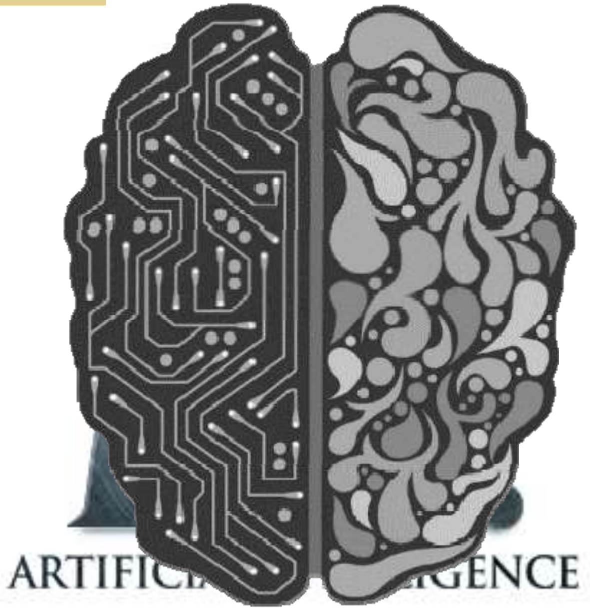
Ch.4 Computational Intelligence
第四章　智能计算 I

1. 概述
2. 神经计算
3. 进化计算
4. 模糊计算
5. 群智能



4.1 概述

- 第一个对**计算智能的定义**是由贝兹德克在92年提出的
  - **due to J.C. Bezdek who states that:**

  "…(strictly) computational systems depend on numerical data supplied by manufactured sensors and do not rely upon knowledge".

  - **Later, in 1994, Bezdek offers that CI is low-level computation in the style of the mind", whereas AI is mid-level computation in the style of the mind".**

贝兹德克
(J.C. Bezdek)

What is CI?

- 1994年James C. Bezdek提出：
  - "…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

  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。

    - 计算适应性
    - 计算容错性
    - 接近人的速度
    - 近似于人的误差率…"
- 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
  - 神经网络
  - 进化计算
  - 群智能

Some other opinions:

要定义“计算智能”并不简单。要在一个正式的定义中容纳诸如模糊集合、神经网络、进化计算、机器学习、贝叶斯推理等各自具有其既定特性的不同领域，即使不是不可能，也是非常困难的。

- **Conference: Computational Intelligence - Methods & Applications- CIMA 2005**
  - 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, [illegible] intelligence and fuzzy systems.

计算智能是关于在复杂和变化的环境中如何实现智能行为的自适应机制研究。因此，计算智能综合了人工神经网络、进化计算、群智能和模糊系统。

AI vs. CI

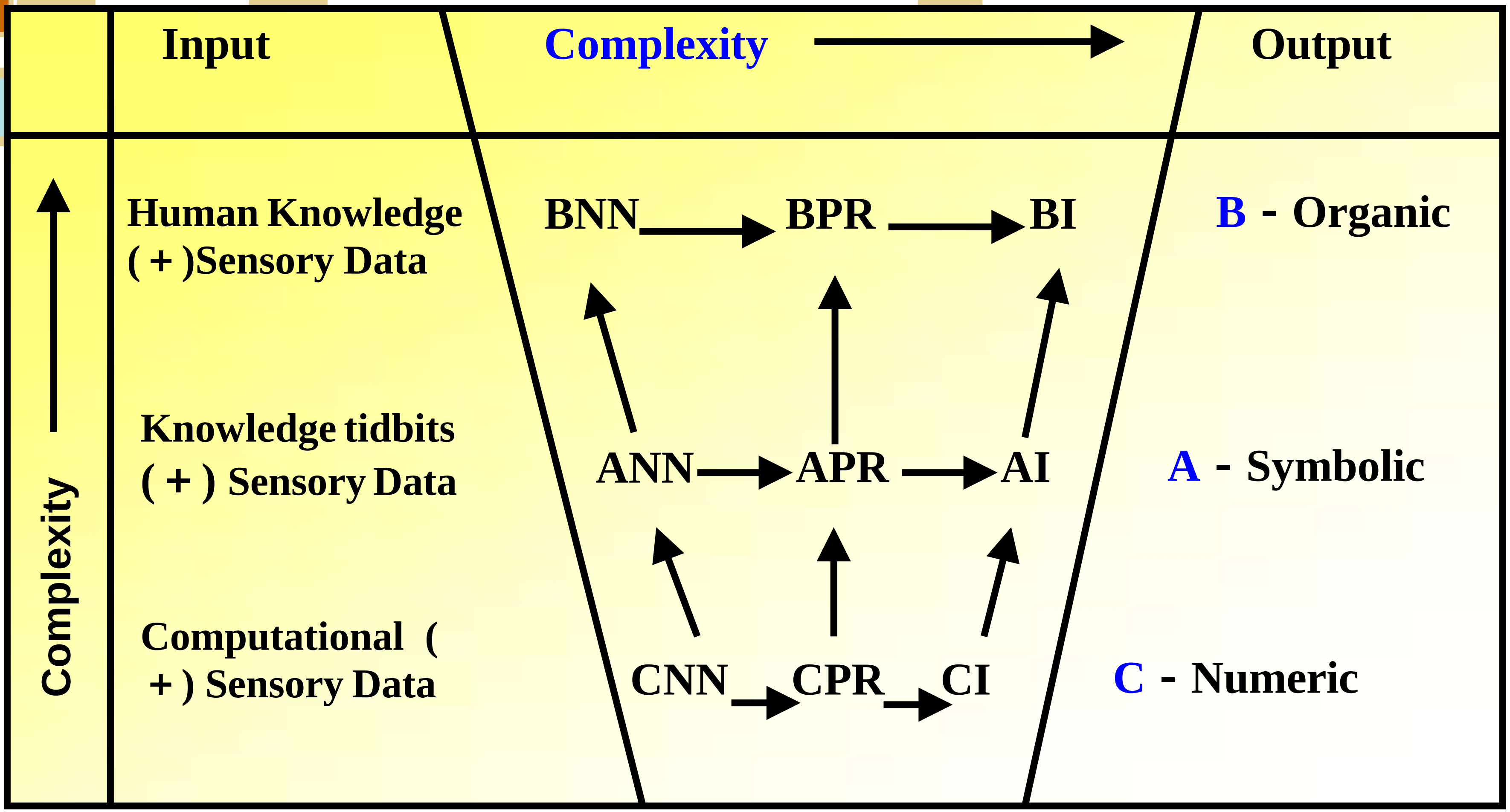
- 绝大多数的AI/CI 研究者将这两者看成不同的研究领域
  - CI forms an alternative to AI (R.C.Eberhart)
    - AI和CI追求目标一致，达到智能
    - CI是系统的核心，AI是系统的外沿部分
    - 两者是并行技术
  - AI subsumes CI (Bezdek)
    - CI是AI的一部分

艾伯哈特
(Eberhart)

AI vs. CI

- Bezdek的ABC:
  - A、B、C三者对应三种不同的系统复杂性级别
    - **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

4.2 人工神经网络

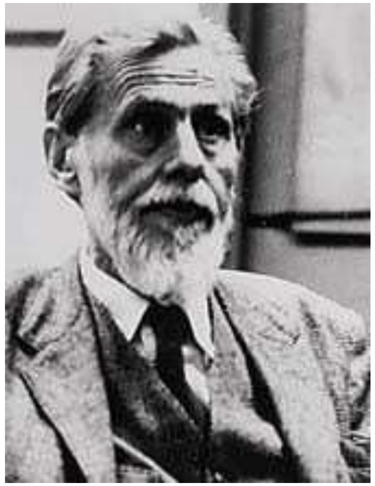
Artificial Neural Networks(ANN)

4.2.1 ANN

- As you read these words you are using a complex ***biological neural network***. You have a highly interconnected set of **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.

History of ANN Research

- McCulloch & Pitts (1943) 被公认为第一个人工神经网络的设计者(MP model)
  - 他们使用**阈值**以及用多个简单单元结合在一起以提高计算能力的思想到今天仍被广泛使用
- 试图通过用计算模型**模拟**生物神经系统的方法来理解生物神经系统的工作模式
- 高度的并行性使得其计算效率非常高
- 有助于理解神经表示的“分布式”特征

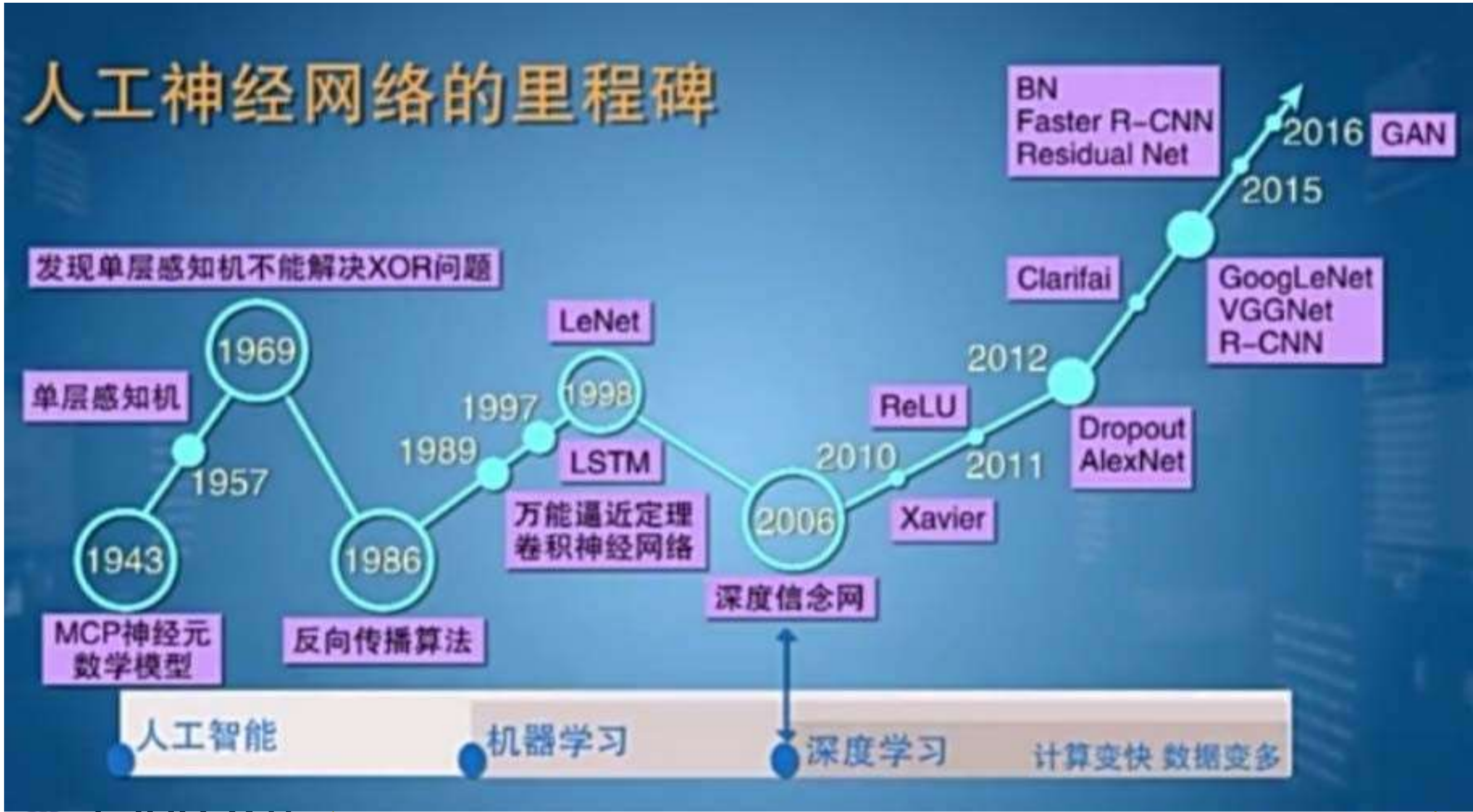


麦克洛奇(McCulloch)

皮茨 (Pitts)

History of ANN Research

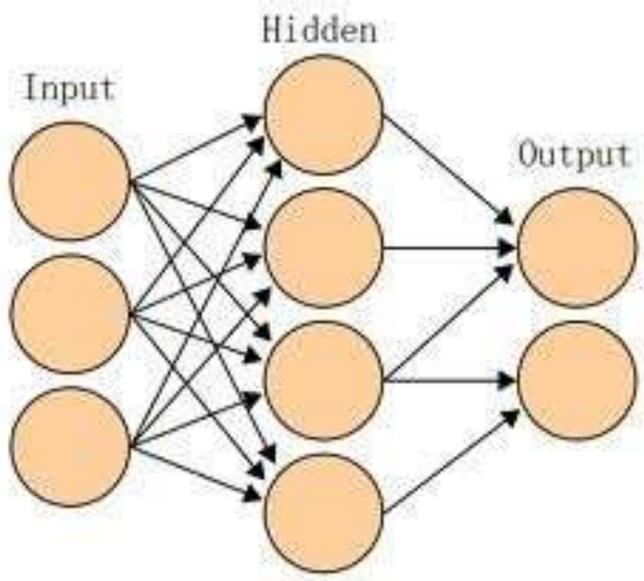
- 1949年Hebb提出了第一个神经网络的学习规则(**权值**).



- 繁荣至今（论文）

人工神经网络（ANN）的特性

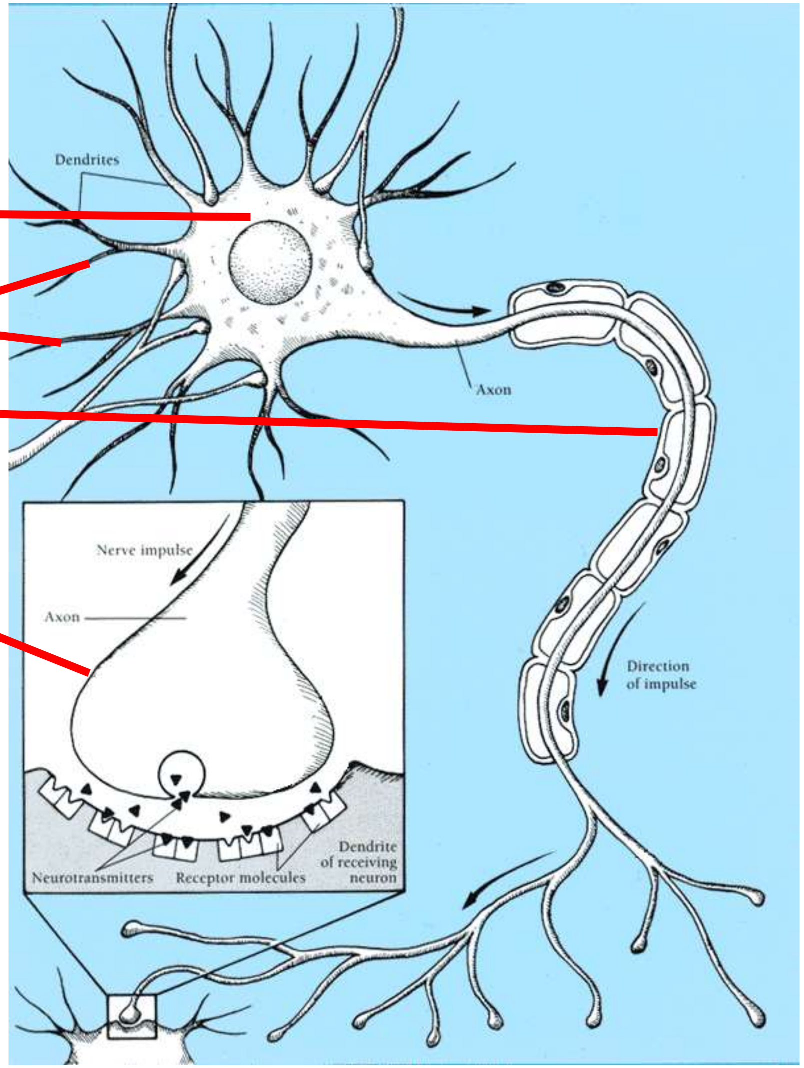
- 并行分布处理
- 非线性映射
- 通过训练进行学习
- 适应与集成
- 硬件实现



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。

生物神经元

- Cell structures
  - Cell body（细胞体）
  - Dendrites(树 突)
  - Axon(轴突)
  - Synapse(突触)
- 10^{11} - 10^{12} neurons in human brain
- Each neuron connected to 10^4 others on average



Cell Structures

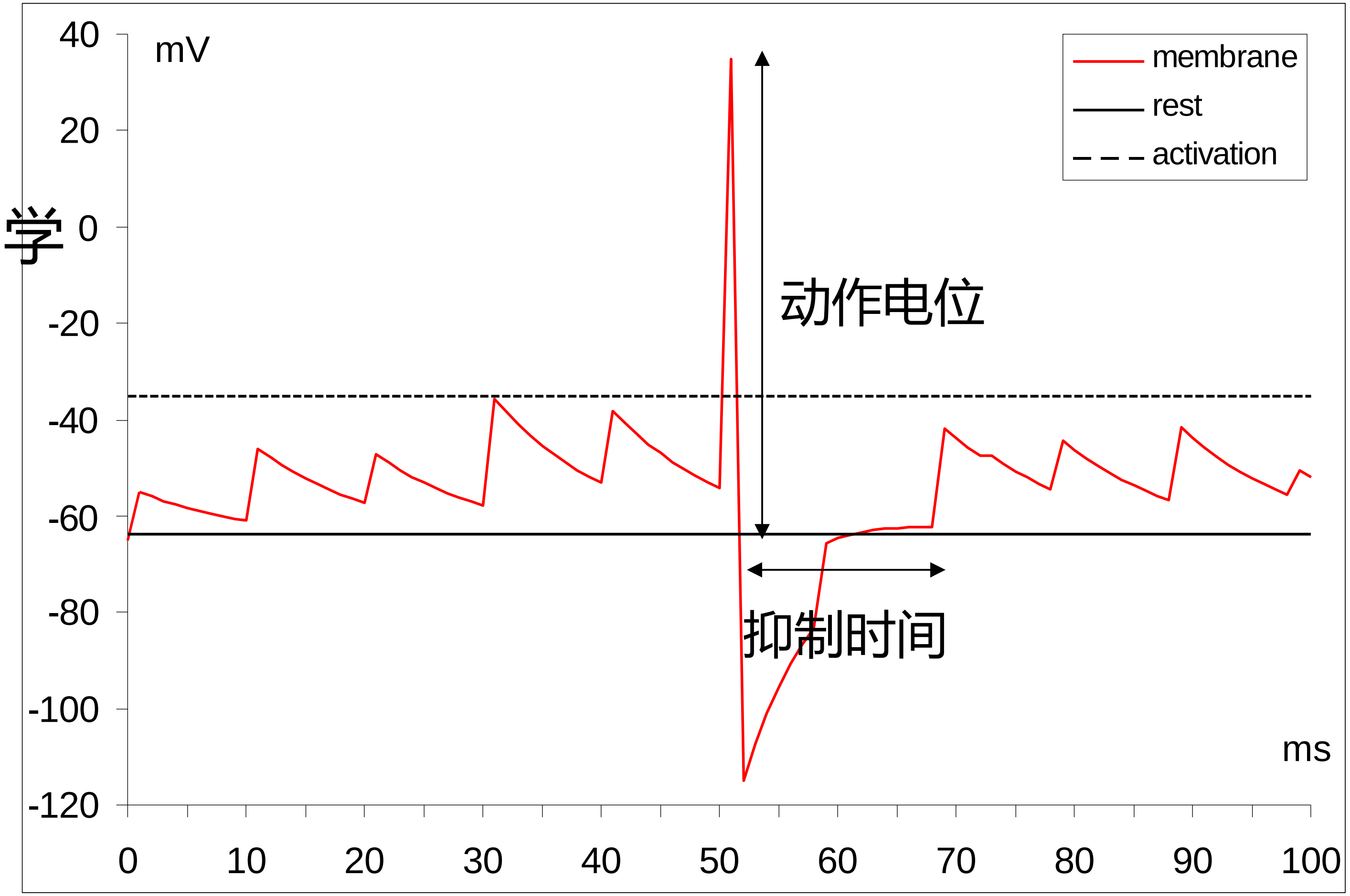
- **A Neuron:** many-inputs / one-output unit
- **Cell Body**(**细胞体**): is 5 – 10 microns in diameter, sums and thresholds these incoming signals
- **Dendrites(树 突):** carry electrical *into* the cell body
- **Axon**(**轴突**): carry the signal from the cell body *out* to other neurons
- **Synapse**(**突触**): contact between an axon of one cell and a dendrites of another cell

Real Neural Learning

- **Synapses** change size and strength with experience.
- **Hebbian learning**: When two connected neurons are firing at the same time, the strength of the synapse between them increases.
- "Neurons that fire together, wire together."

The Biological Neurons

- 神经动力学



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

Notation

- **Scalars（标量）**: small *italic* letters

  e.g., a, b, c

- **Vectors（向量）**: small **bold** nonitalic letters

  e.g., $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$

- **Matrices（矩阵）**: capital **BOLD** nonitalic letters

  e.g., $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$

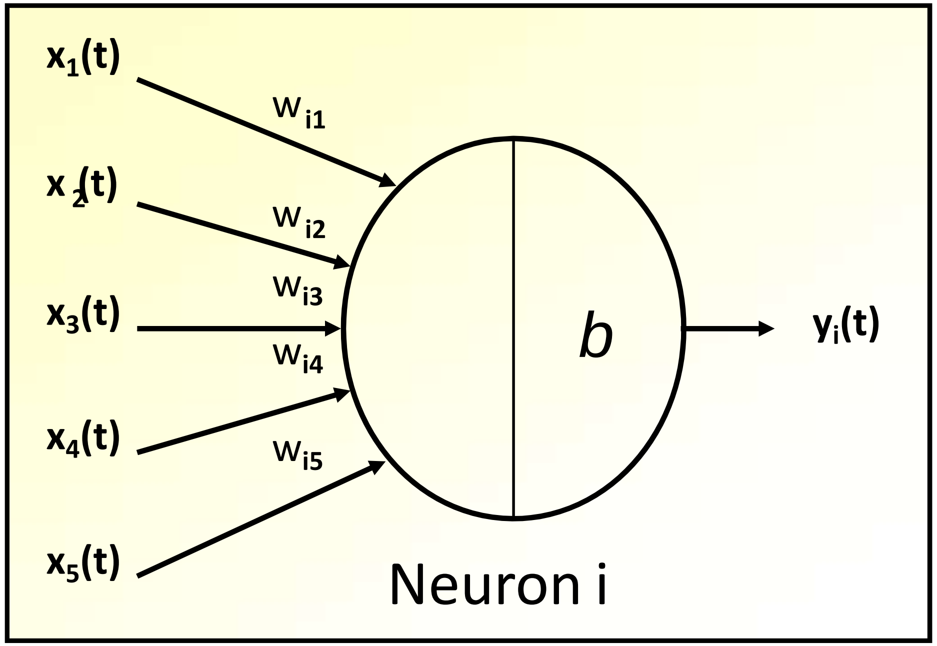
The Artificial Neuron

Stimulus

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

Response

$$y_i(t) = f\left(b + u_i(t)\right)$$



T = = *threshold* (阈值)or *bias*(偏移)

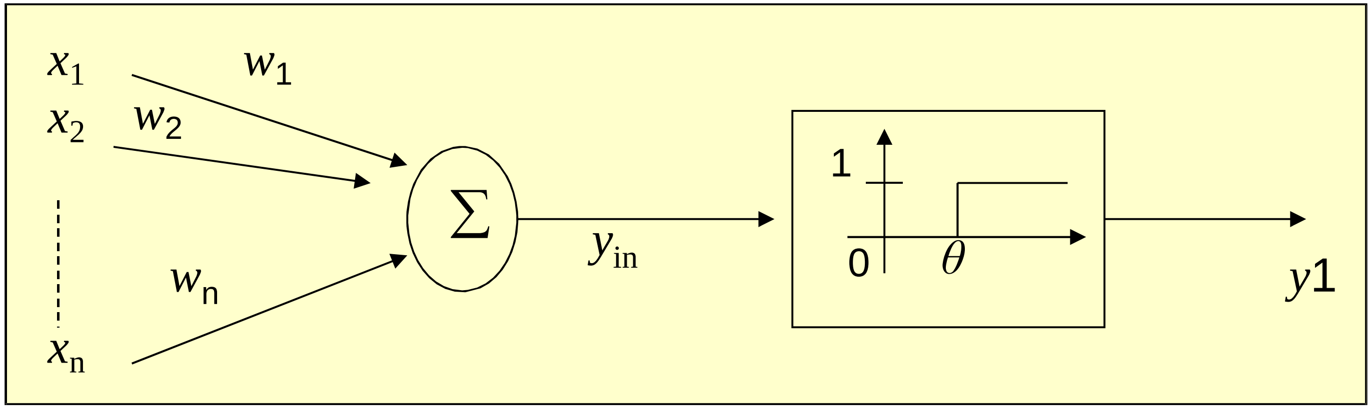
$y_i(t)$ = output of neuron *i* at time *t* w_{ij}

= 从神经元*i* 到*j*的连接权值

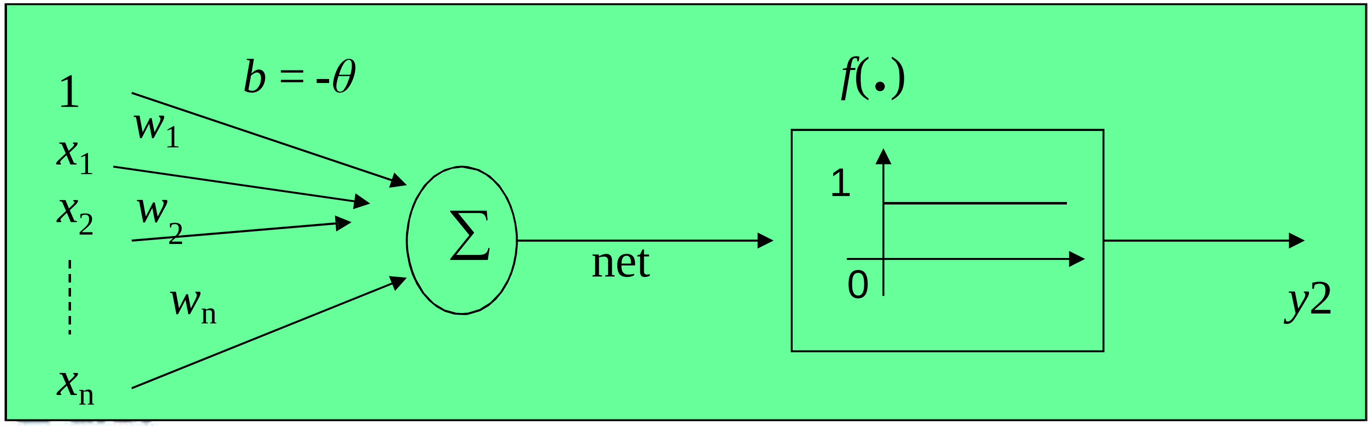
f = *transfer function*(传递函数), or *activation function*(激励函数)

Bias and Weight —(阈值与权重)

- The **bias *b*** is much like a **weight *w***, except that it has a constant input of 1. It can be **omitted** if NOT necessary.



$$\begin{cases} y_{in} = \sum_{i=1}^{n} w_i x_i \\ y1 = f(y_{in} - \theta) \end{cases}$$

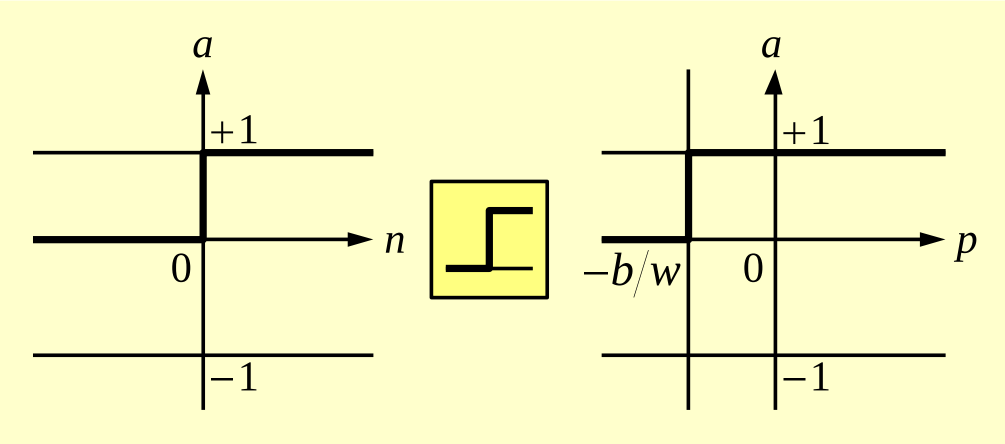


$$\begin{cases} net = b + \sum_{i=1}^{n} w_i x_i \\ y2 = f(net) \end{cases}$$

Transfer Functions传递函数

- The transfer function *f* may be a ***linear*** or ***nonlinear*** function of net input *n*
- The most commonly used func.
  - **Hard limit** transfer function
  - **Linear** transfer function
  - **Log-sigmoid** transfer function
  - **ReLU** transfer function

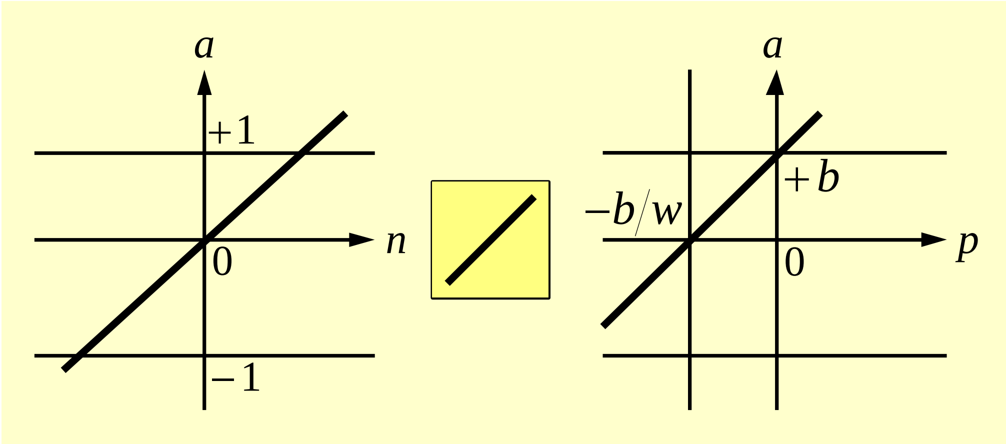
Hard Limit Transfer Func.



$a = hardlim(n)$ $a = hardlim(wp+b)$

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

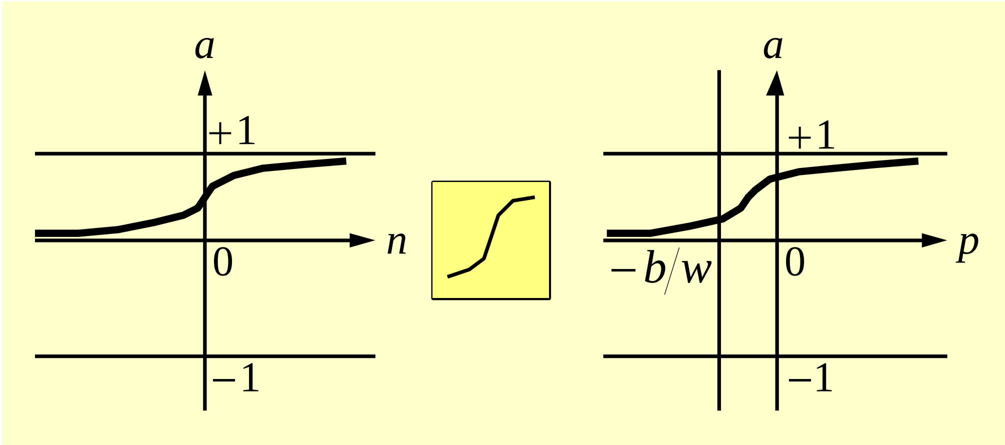
Linear Transfer Function



$a = purelin(n)$ $a = purelin(wp+b)$

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

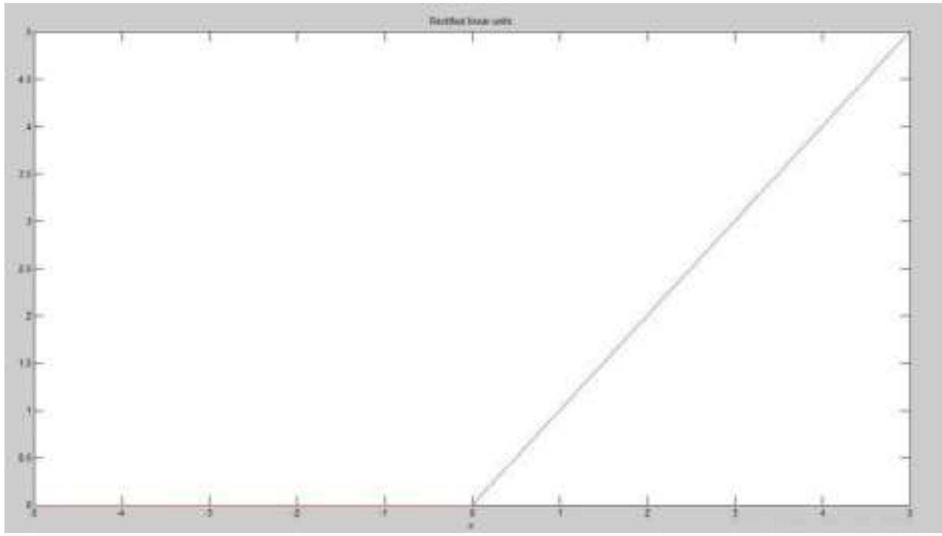
Log-Sigmoid Transfer Func.

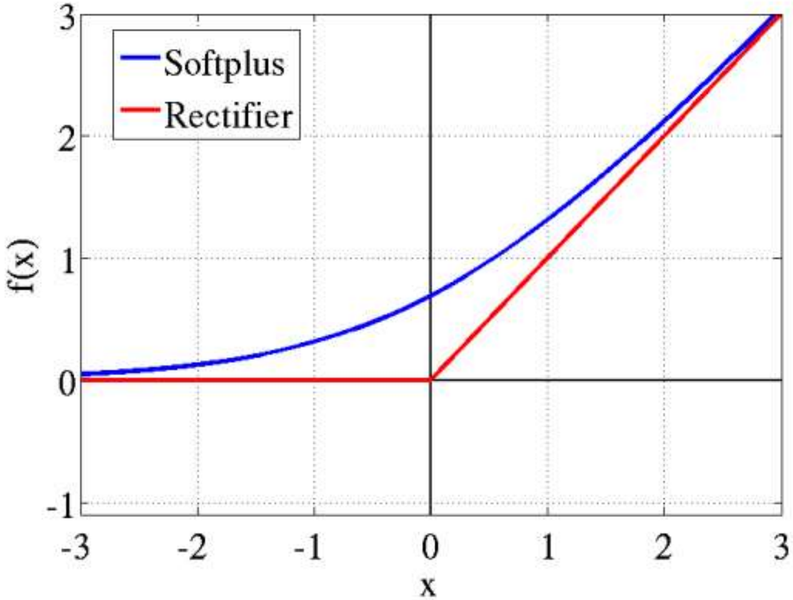


$a = logsig(n)$ $\qquad$ $a = logsig(wp+b)$

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

ReLU Transfer Func.

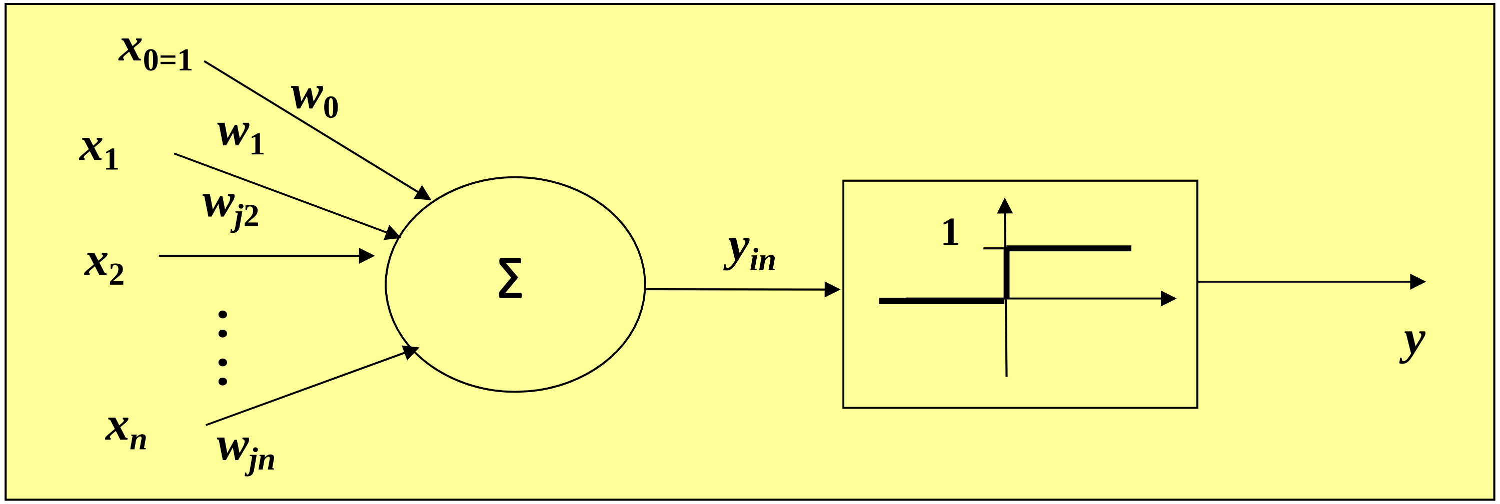




- $f(x) = 0$, **if** $x < 0$
- $f(x) = x$, **if** $x \geq 0$

感知器(Perceptron)

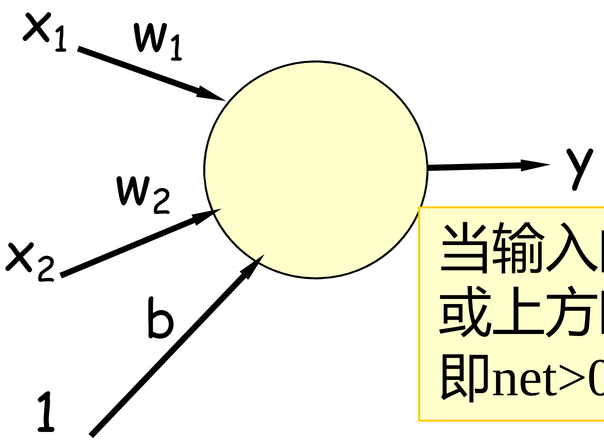
- 1958年由Rosenblatt 提出，用于将输入分为两类
- 简单的单层前馈网络
- 其神经元为一个线性阈值单元(Linear Threshold)，也称阈值逻辑单元(Threshold Logic Unit, TLU)



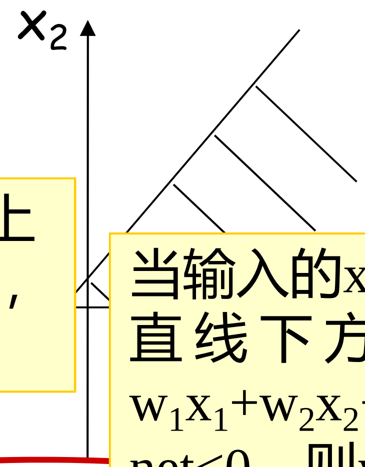
$$y = f(w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n) = \begin{cases} 1 & w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq 0 \\ 0 & ontherwise \end{cases}$$

TLU的基本功能为线性划分

- 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和纵轴，则x_1和x_2的不同值将对应该平面上的不同点



当输入的x_1和x_2位于直线上或上方时, $w_1x_1+w_2x_2+b>0$，即net>0，则y=1



当输入的x_1和x_2位于直线下方时，$w_1x_1+w_2x_2+b<0$，即net<0，则y=0

$$net = b + \sum_{i=1}^{n} w_i x_i$$

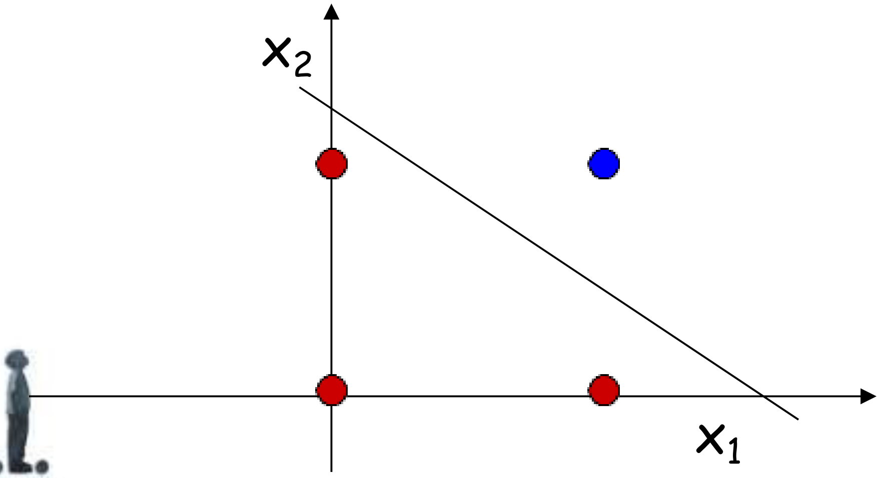
$$y = f(net) = \begin{cases} 0 \\ 1 \end{cases}$$

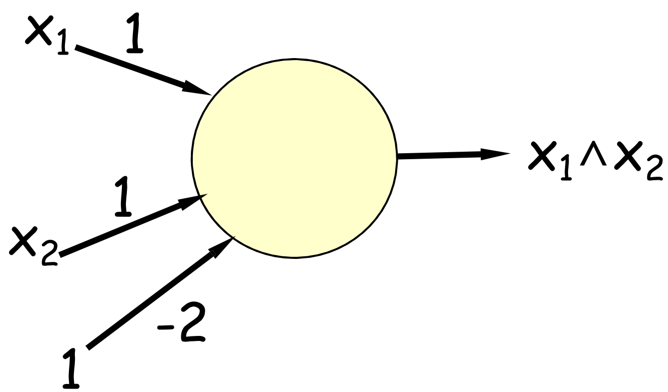
$$0 = b + \sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + b$$

Line equation

Neural Computation: And

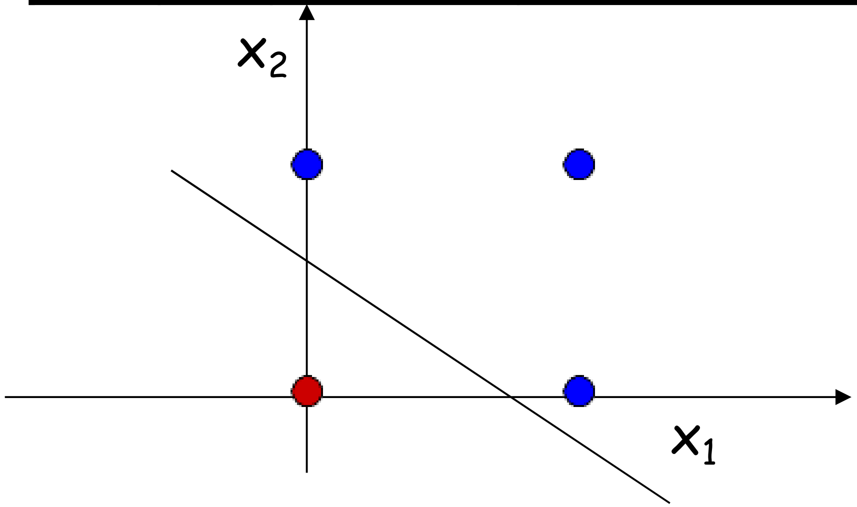
input		output	function	condition
x_1	x_2	$x_1 \wedge x_2$	$w_1 * x_1 + w_2 * x_2 + b = 0$	
0	0	0	$w_1*0 + w_2*0 + b < 0$	$b < 0$
0	1	0	$w_1*0 + w_2*1 + b < 0$	$-b > w_2$
1	0	0	$w_1*1 + w_2*0 + b < 0$	$-b > w_1$
1	1	1	$w_1*1 + w_2*1 + b \geq 0$	$-b \leq w_1 + w_2$

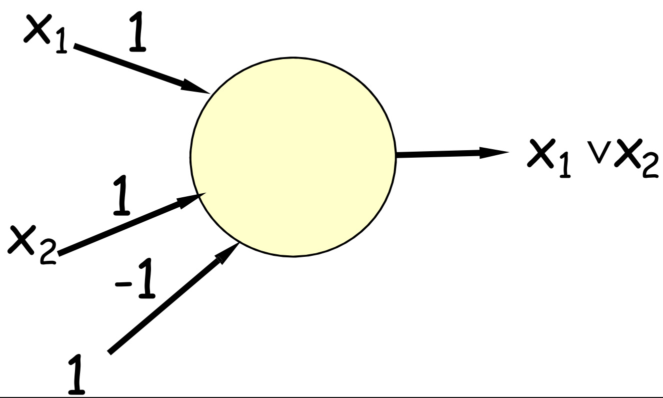




Neural Computation: Or

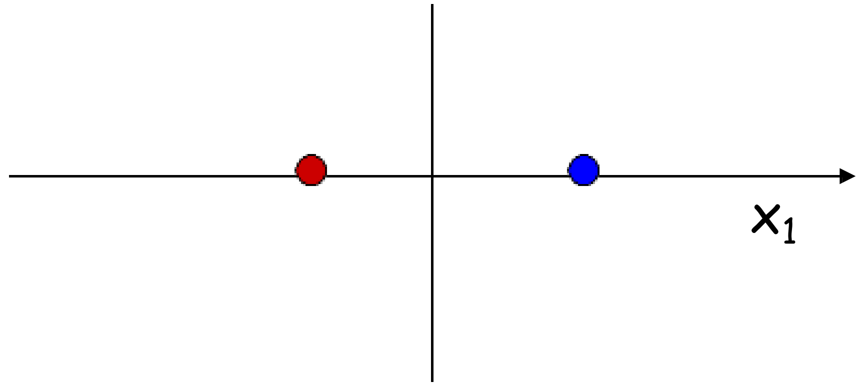
input		output	function	condition
x_1	x_2	$x_1 \vee x_2$	$w_1*x_1+ w_2* x_2+b=0$	
0	0	0	$w_1*0+ w_2*0 +b < 0$	$b < 0$
0	1	1	$w_1*0+ w_2*1 +b \geq 0$	$-b < w_2$
1	0	1	$w_1*1+ w_2*0 +b \geq 0$	$-b < w_1$
1	1	1	$w_1*1+ w_2*1+b\geq 0$	$-b\leq w_1+ w_2$

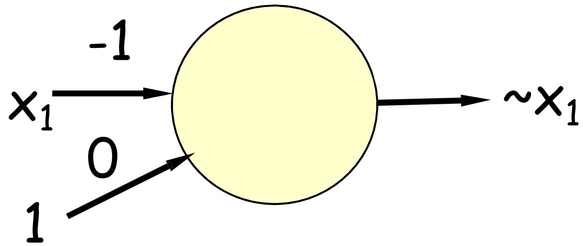




Neural Computation: Not

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

Network Architectures

- Models of neural networks are specified by the ***three* basic entities**:
  - models of the *processing elements* (neurons) （神经元模型）；
  - models of *inter-connections and structures* (network topology)（网络拓扑模型）；
  - the *learning rules* (the ways information is stored in the network)（学习规则）．
- The **weights** may be ***positive*** (excitatory) or ***negative*** (inhibitory).
- **Information** is stored in the ***connection weights***.

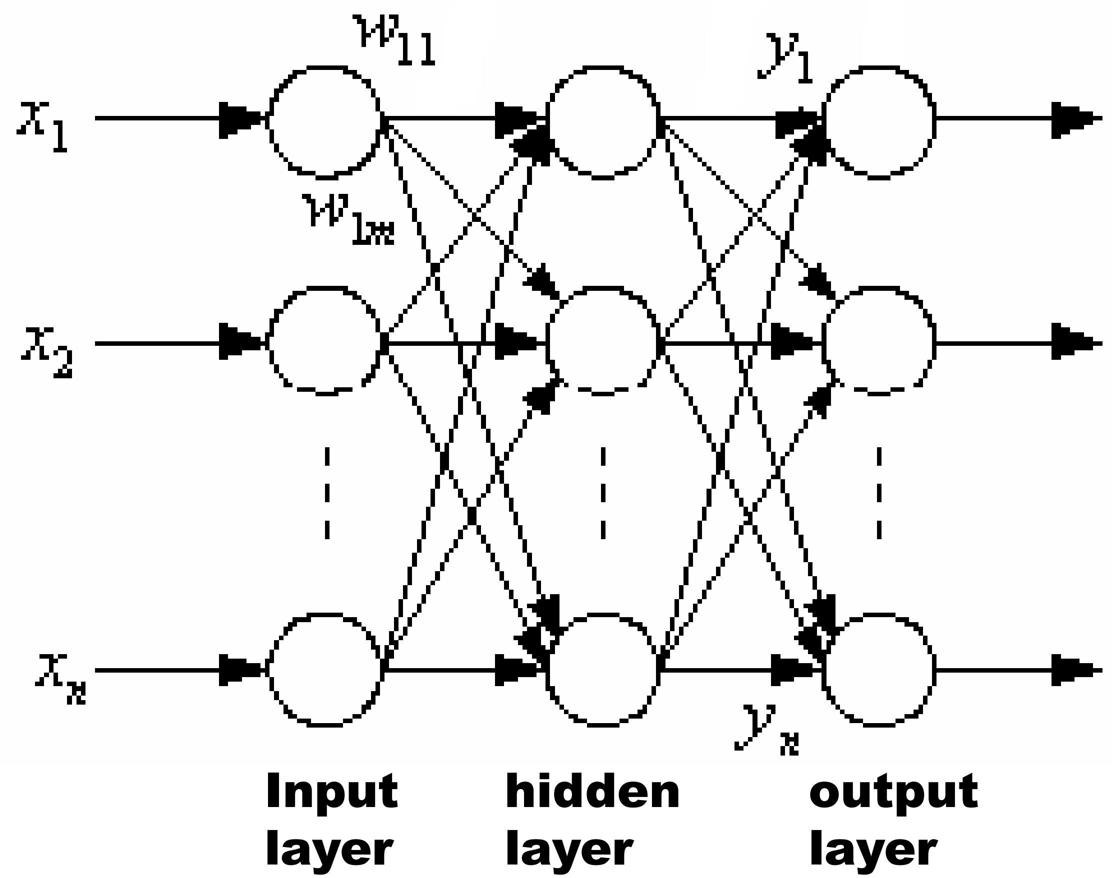
Network Structures

- The layer that receives inputs is called the ***input layer***.
- The outputs of the network are generated from the ***output layer***.
- Any layer between the input and the output layers is called a ***hidden layer***.
- There may be ***from zero to several*** **hidden layers** in a neural network.

Network Structures—Topology
神经网络的拓扑结构

- When no node output is an input to a node in the same layer or preceding layer, the network is a ***feedforward network*(前馈网络)**.
- When outputs are directed back as inputs to same- or preceding-layer nodes, the network is a ***feedback network*（反馈网络）**.
- **Feedback** networks that have **closed loops** are called ***recurrent networks*（递归网络）**.

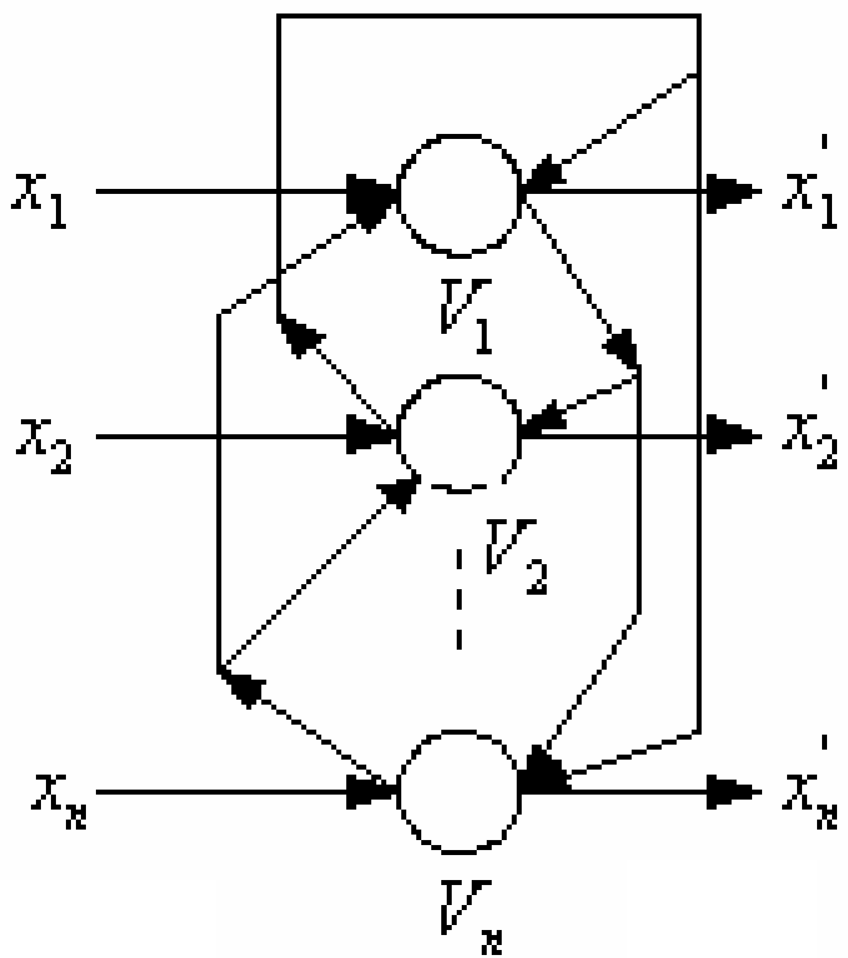
Feedforward network 前馈网络



$$W = \begin{pmatrix} w_{11} & w_{12} & w_{1m} \\ w_{21} & w_{22} & w_{2m} \\ \vdots & & \vdots \\ w_{n1} & w_{n2} & w_{nm} \end{pmatrix}$$

$$y_j = f\left(\sum_{i=1}^{n} w_{ij} x_i - \theta_j\right) \qquad j = 1,2,\ldots,m$$

Feedback network 反馈网络



How to Pick an Architecture

Problem specifications help define the network in the following ways:

1. **Number of network inputs** = number of problem inputs
2. **Number of neurons in output layer** = number of problem outputs
3. **Output layer transfer function** choice at least partly determined by problem specification of the outputs.

4.2.3 Learning of ANN

<u>Two</u> kinds of learning in neural networks:

- ***parameter learning*****（参数学习）**, which concerns the updating the **weights** and the **bias** in a neural network
- ***structure learning*****（结构学习）**, which focuses on the change in the **network structure**, including the *<u>number of nodes</u>* and their *<u>connection types</u>*.

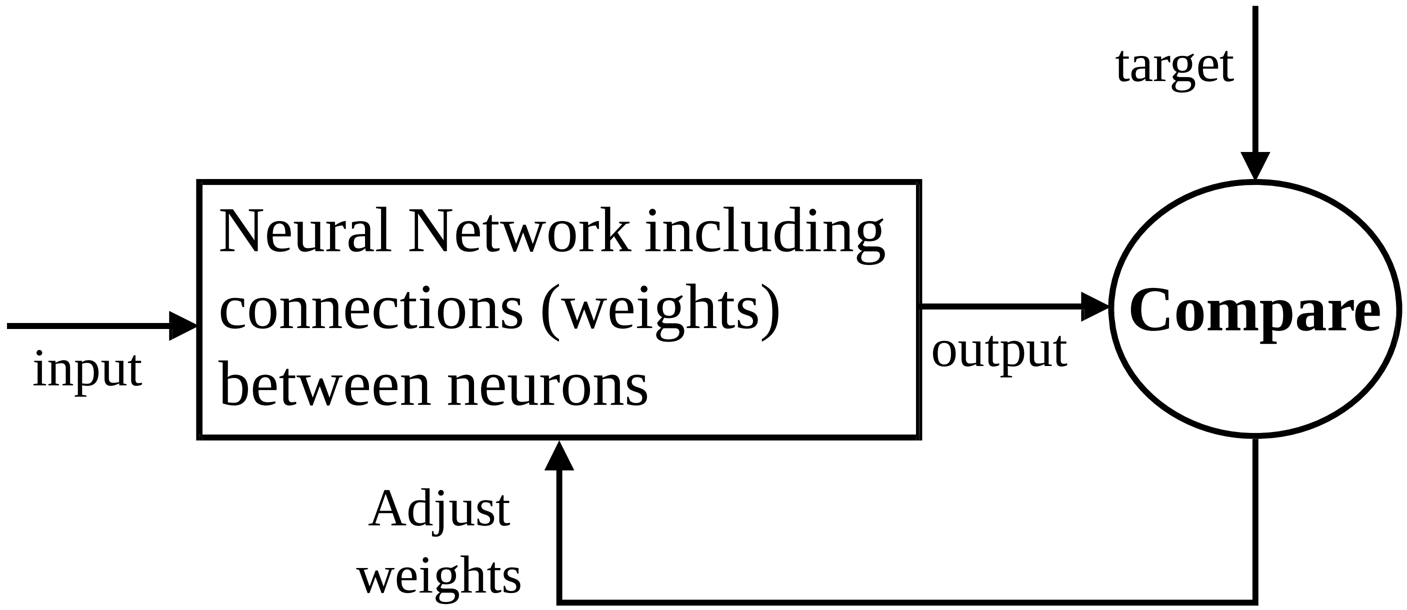
Each kind of learning can be further classified into **<u>three categories</u>**

- ***supervised learning*****（监督学习）**
- ***reinforcement learning*****（增强学习）**
- ***unsupervised learning*****（非监督学习）**.

Learning Rule
—Supervised Learning

- 又称有师学习
- 需提供训练样例(*training set*) : $\{\mathbf{p}_1,\mathbf{t}_1\}$, $\{\mathbf{p}_2,\mathbf{t}_2\}$,…, $\{\mathbf{p}_q,\mathbf{t}_q\}$，其中$\mathbf{p}_i$ 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

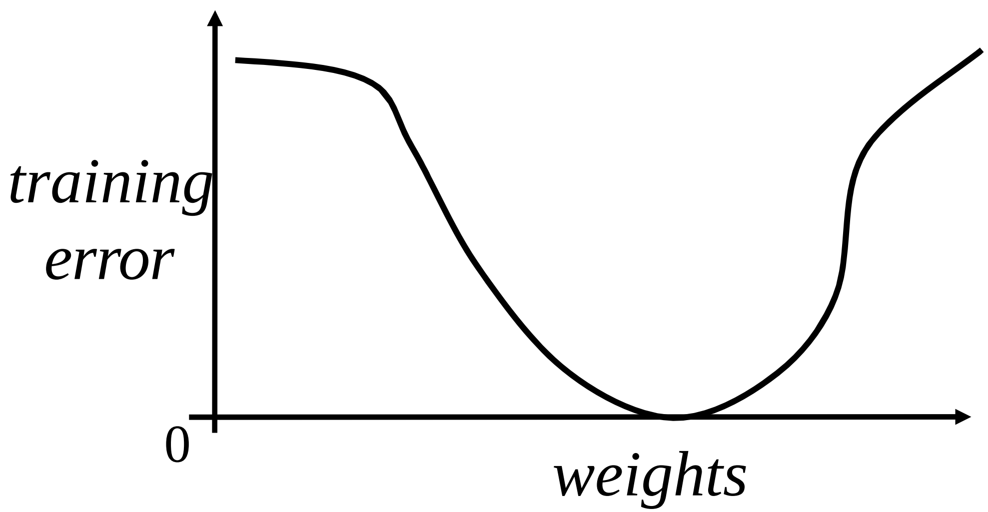
- 又称无师学习
- The **weights** and **biases** are modified in response to network inputs only. ***There are no target outputs available***.
- Most of these algorithms perform some kind of **clustering(聚类) operation**. They learn to categorize the input patterns into a **finite** number of classes.

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***.
- 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**.
- Example: Genetic Algorithm （GA）

Learning Rule Example
--Perceptron Learning

- The hypothesis space being searched is a set of weights and a threshold.
- 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



Learning Rule
— Perceptron Learning Rule

- 权值更新:

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

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

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

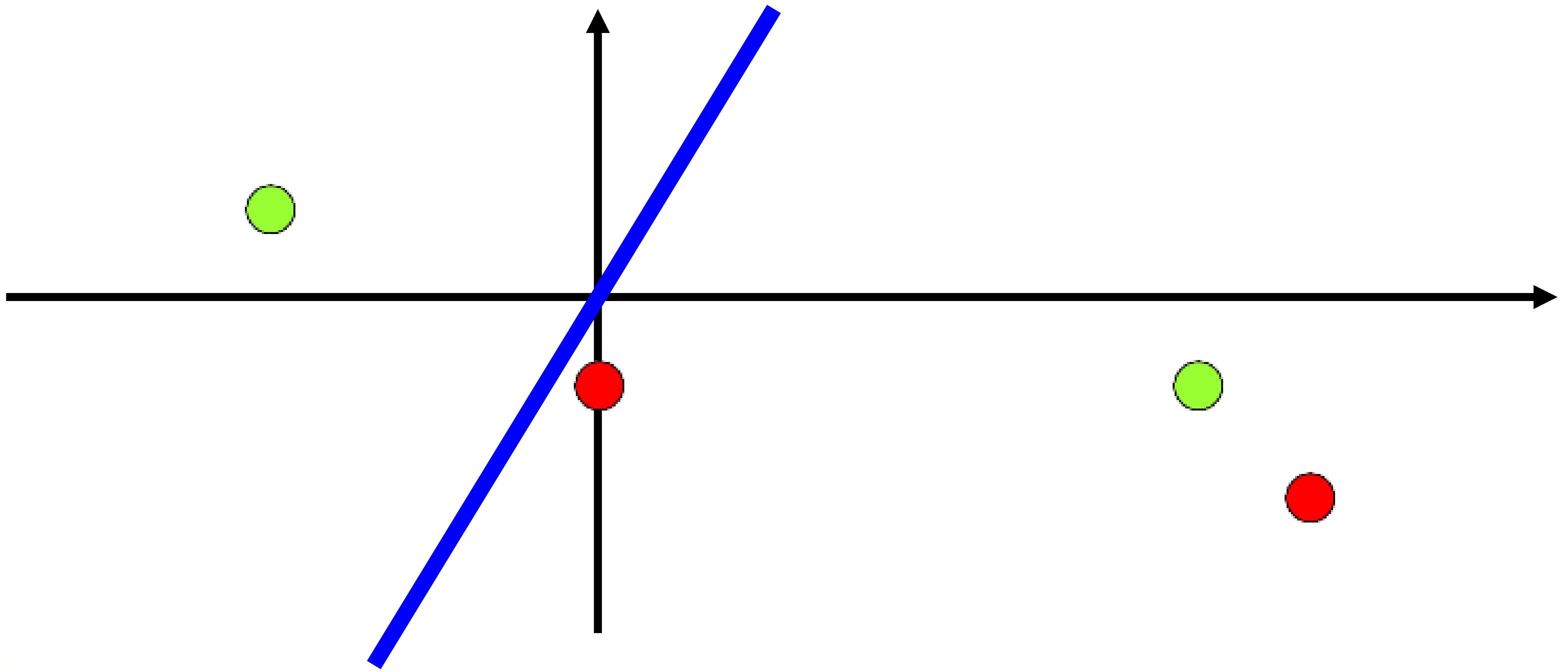
3.Update $w = w + \eta(t-o)x$（η为学习常数，t为期望输出，o为实际输出）

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

Perceptron Learning: Example

P = {(6,−1), (−3, 1)}

N= {(0,−1), (7,−2)} $\eta = 1$

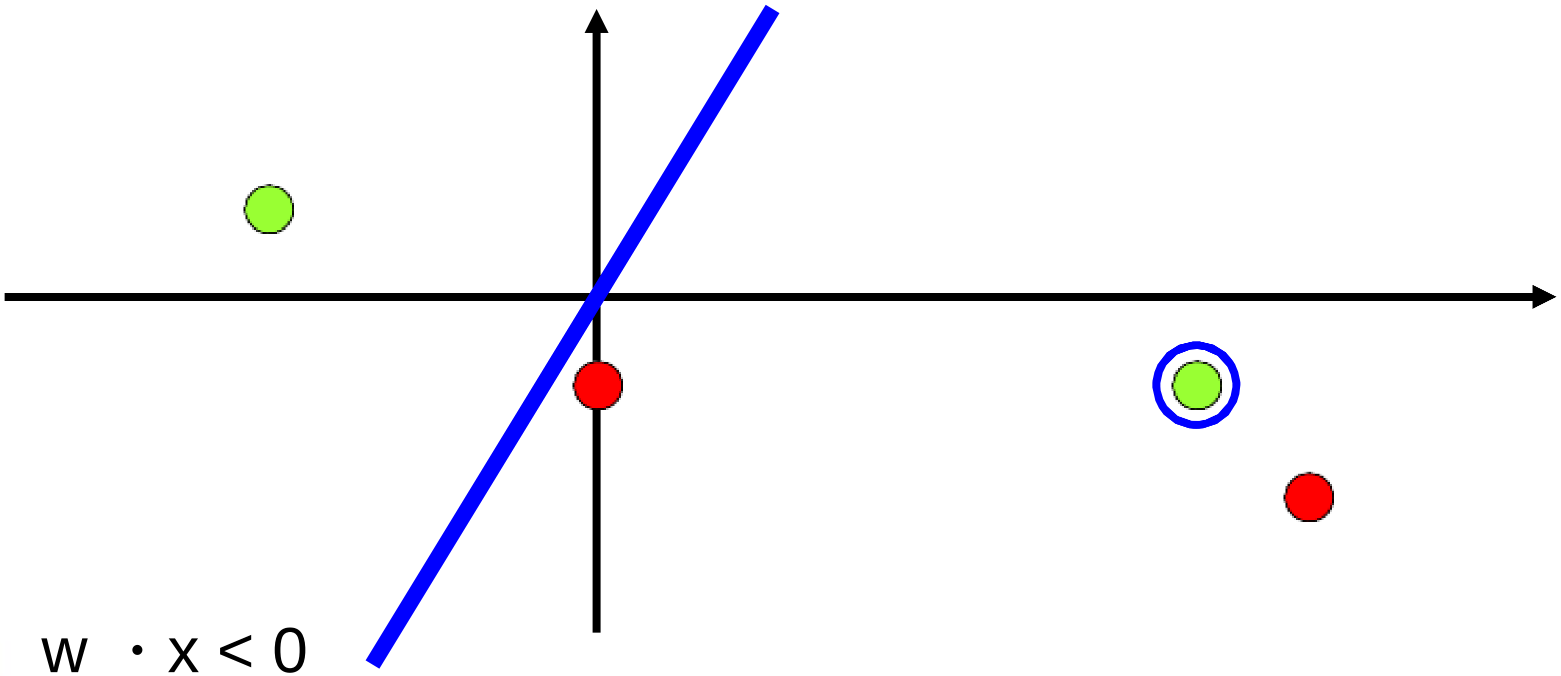


$w = (6,-1) + (-3, 1) - (0,-1) - (7,-2) = (-4, 3)$

Perceptron Learning: Example

P = {(6,−1), (−3, 1)}

N= {(0,−1), (7,−2)} $\eta = 1$

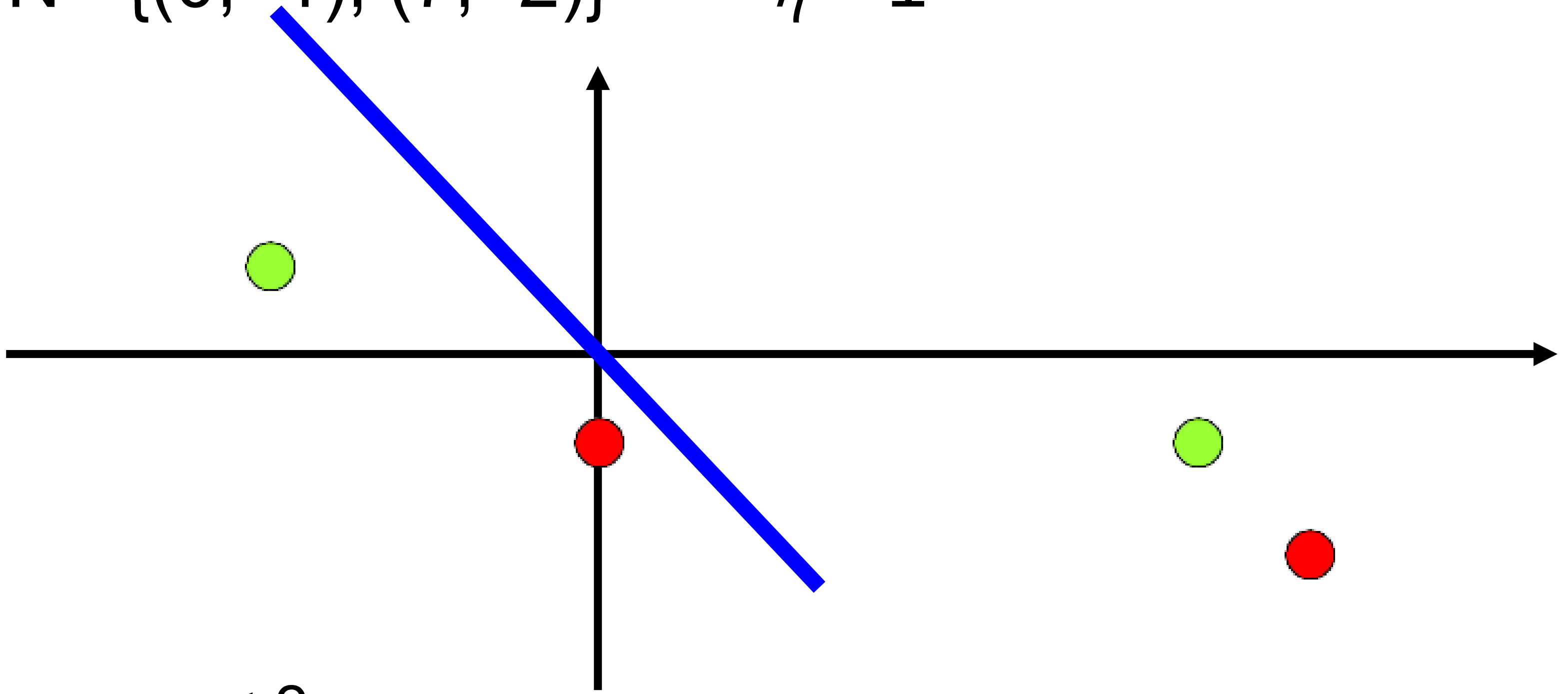


w · x < 0

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

Perceptron Learning: Example

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

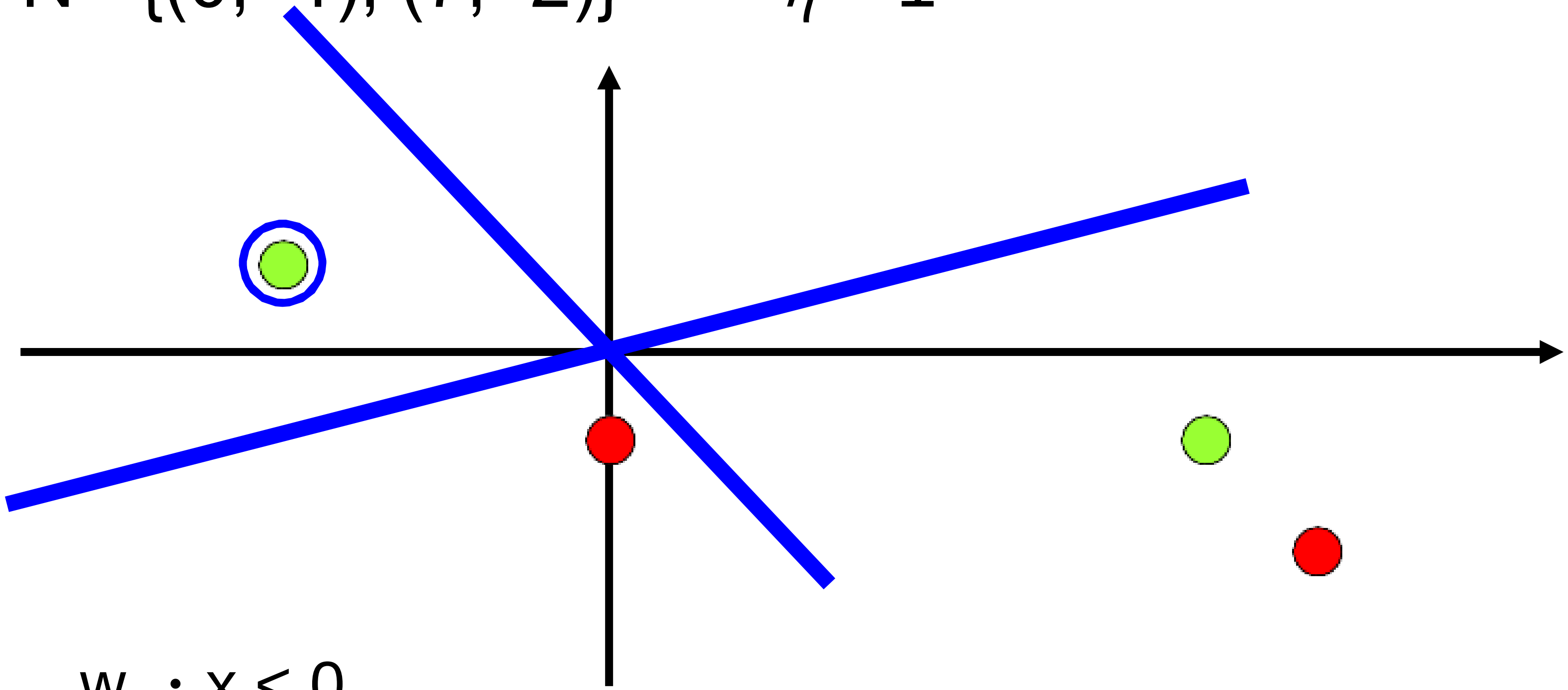


w ・x < 0
w = w + (1-0)x =(-4, 3)+(6, -1)=(2, 2)

Perceptron Learning: Example

$P = \{(6,-1), (-3, 1)\}$

$N = \{(0,-1), (7,-2)\}$ $\eta = 1$

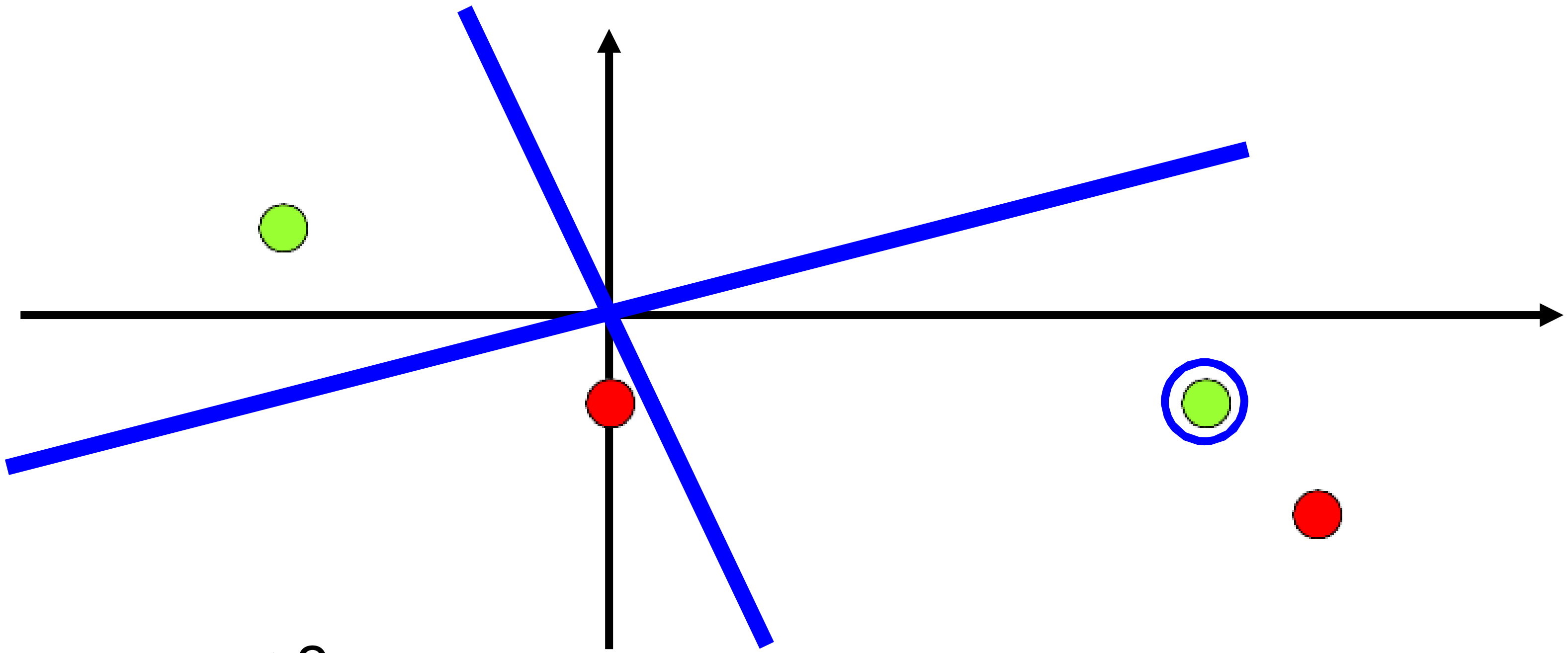


$w \cdot x < 0$

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

Perceptron Learning: Example

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



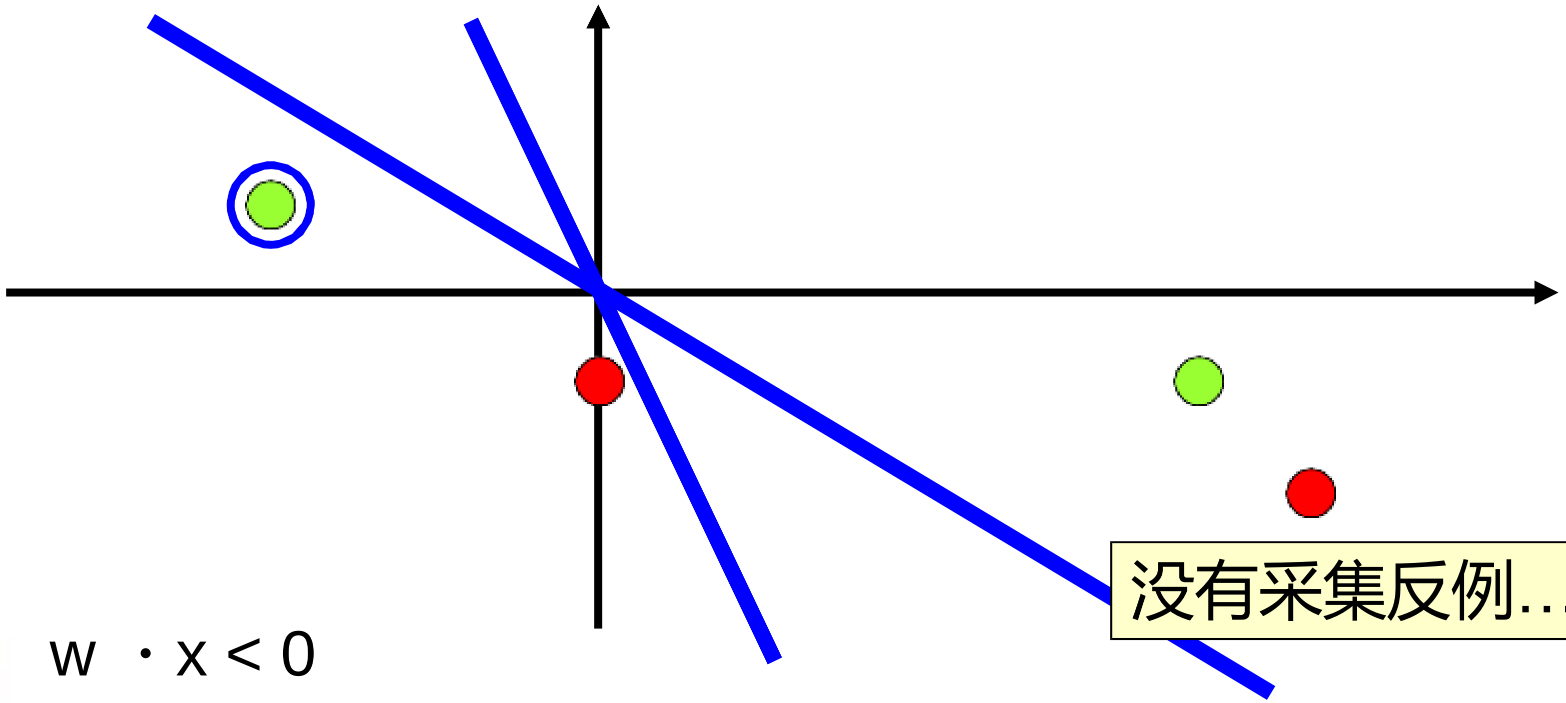
w ・x < 0
w = w + (1-0)x =(-1, 3)+(6, -1)=(5, 2)

Perceptron Learning: Example

$P = \{(6,-1), (-3, 1)\}$

$N = \{(0,-1), (7,-2)\}$ $\qquad \eta = 1$

其他神经网络学习方法将在机器学习中具体介绍…



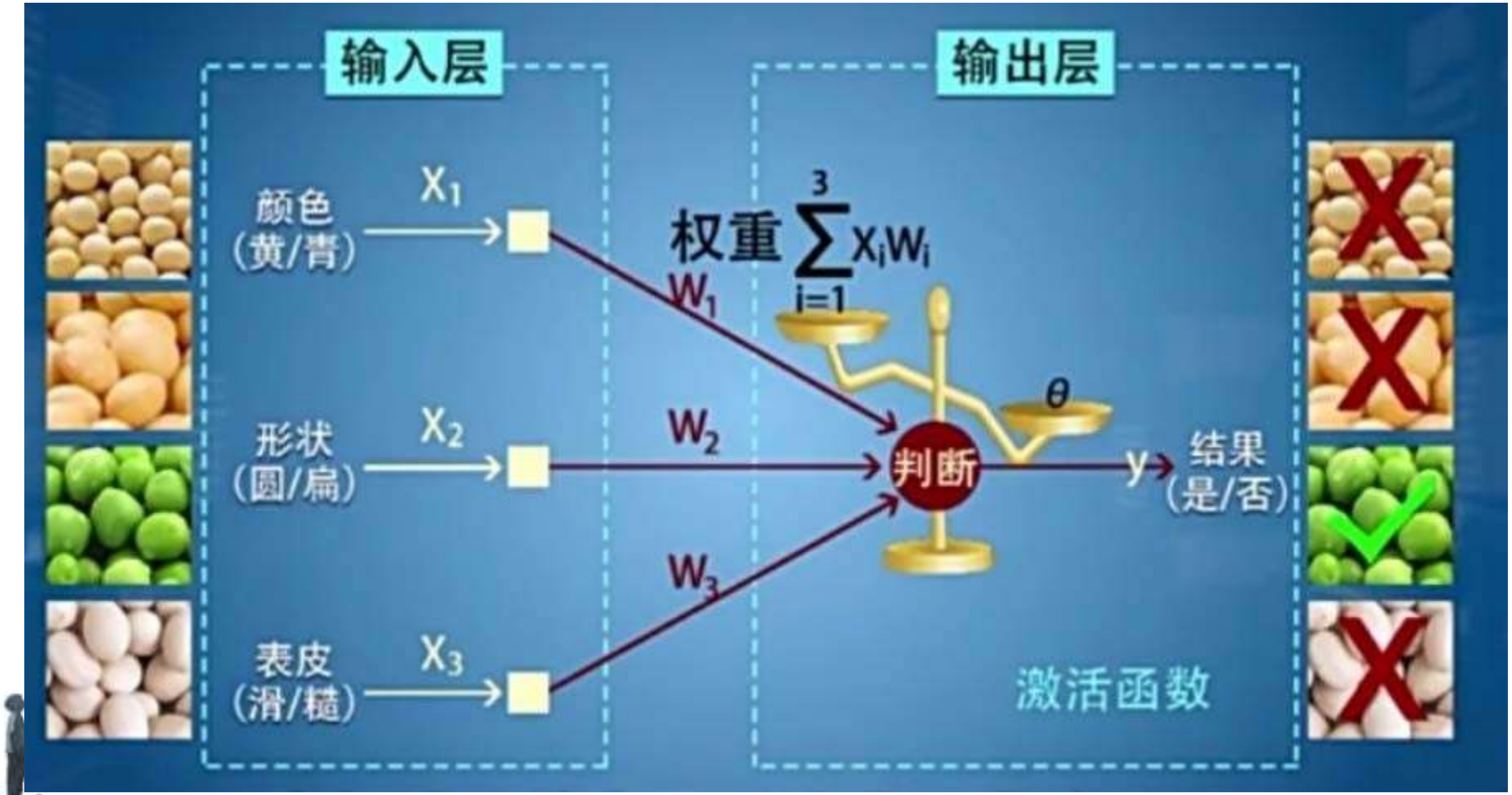
没有采集反例…

$w \cdot x < 0$

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

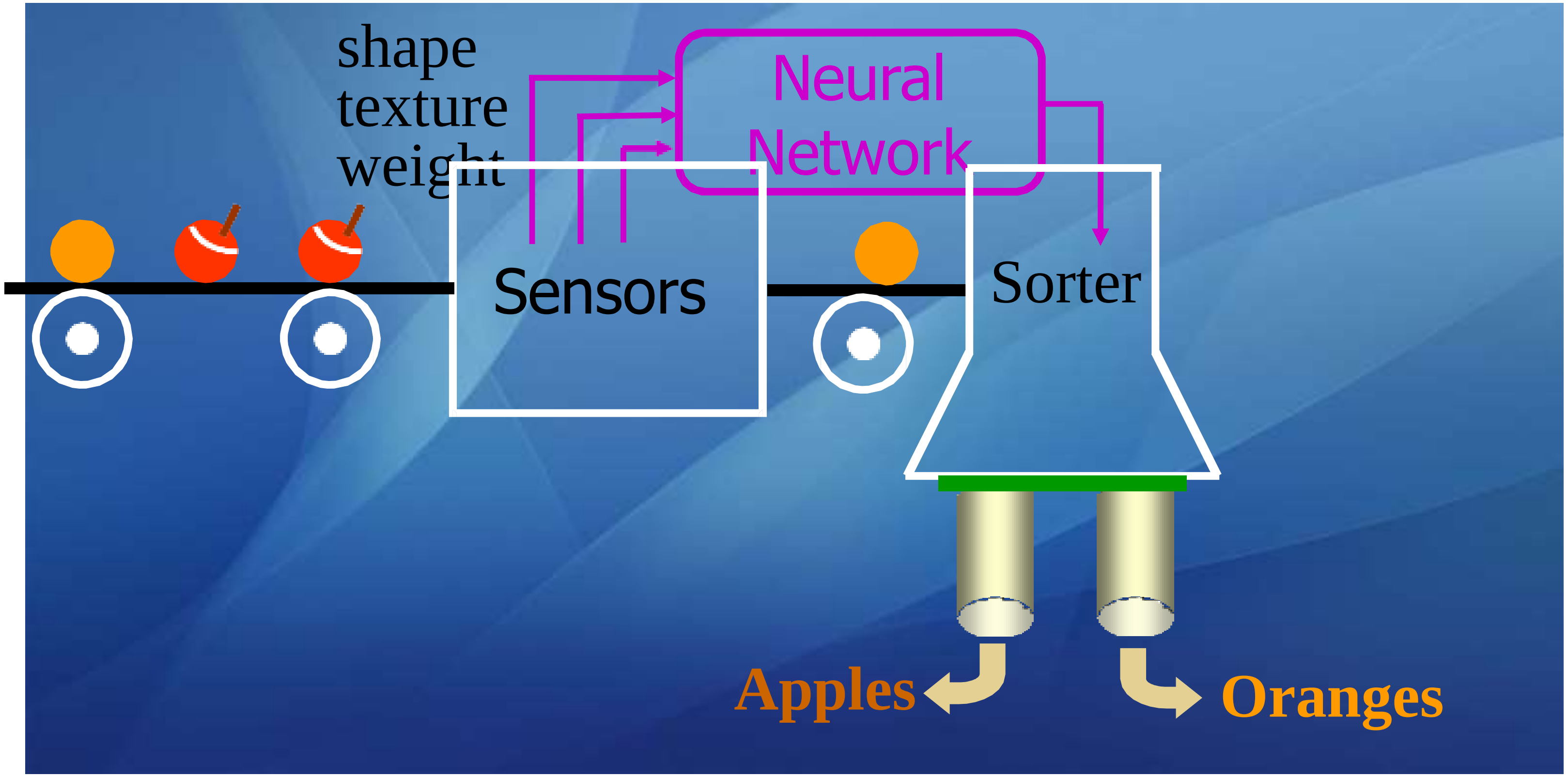
ANN application—Example

- Pattern Recognition
  - Problem Statement 1



ANN application—Example 1

- Pattern Recognition
  - Problem Statement



ANN application—Example 1

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

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

ANN application—Example 1

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

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

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

ANN application—Example 1

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

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

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

ANN application—Example 1

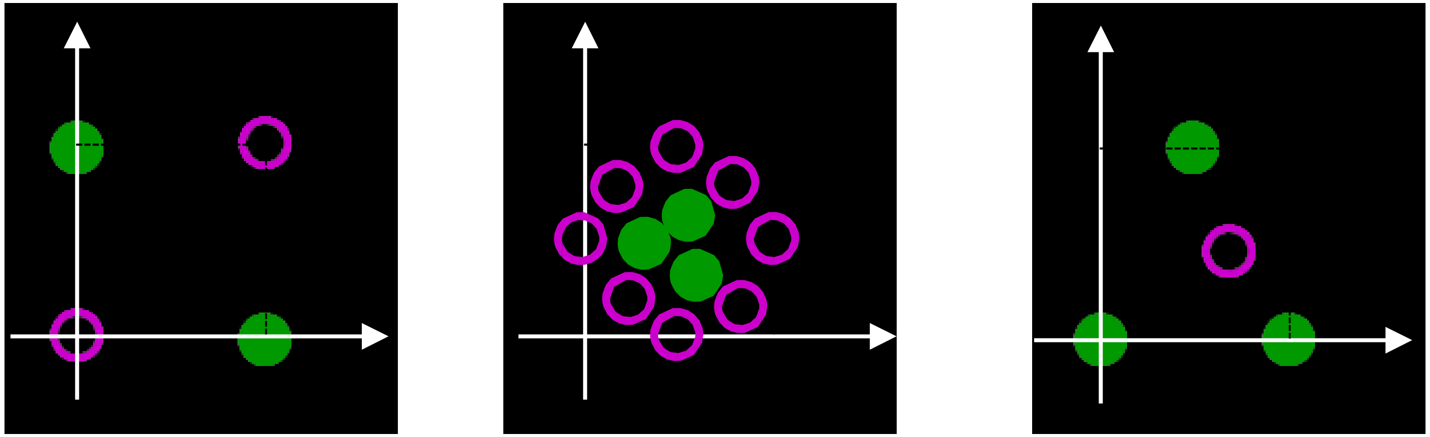
- What happens if we put a **not-so-perfect orange** into the classifier? That is to say, an orange with an **elliptical shape** is pass through the sensor.

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

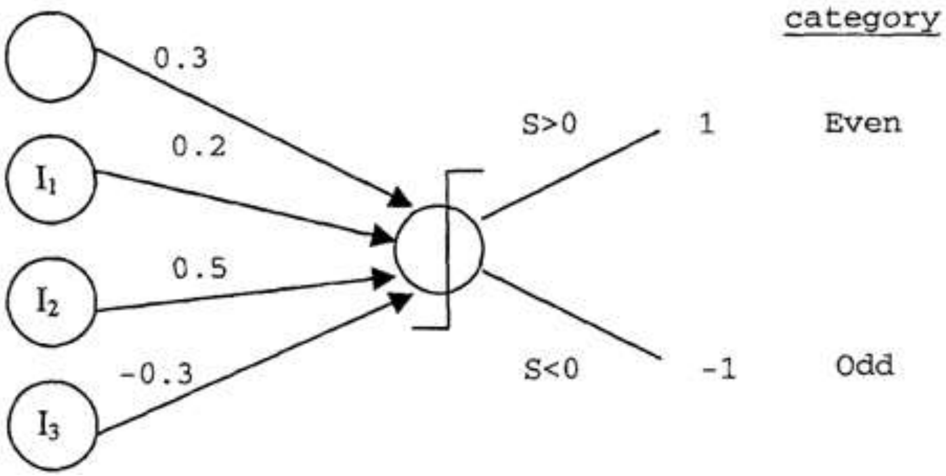
$$a = \text{hardlims}\left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + 0 \right) = -1 \Rightarrow \textbf{orange}$$

Limitations

- The perceptron can be used to classify input vectors that can be separated by a linear boundary, like **AND gate** example.
  $\Rightarrow$ **linearly separable** (**AND**, **OR** and **NOT** gates)
- **Not** linearly separable, e.g., **XOR gate**



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？