Introduction to Artificial Neural Networks

Resana Machine Learning Workshop

prepared by Hosein Hasani

Summer 1399

Outline

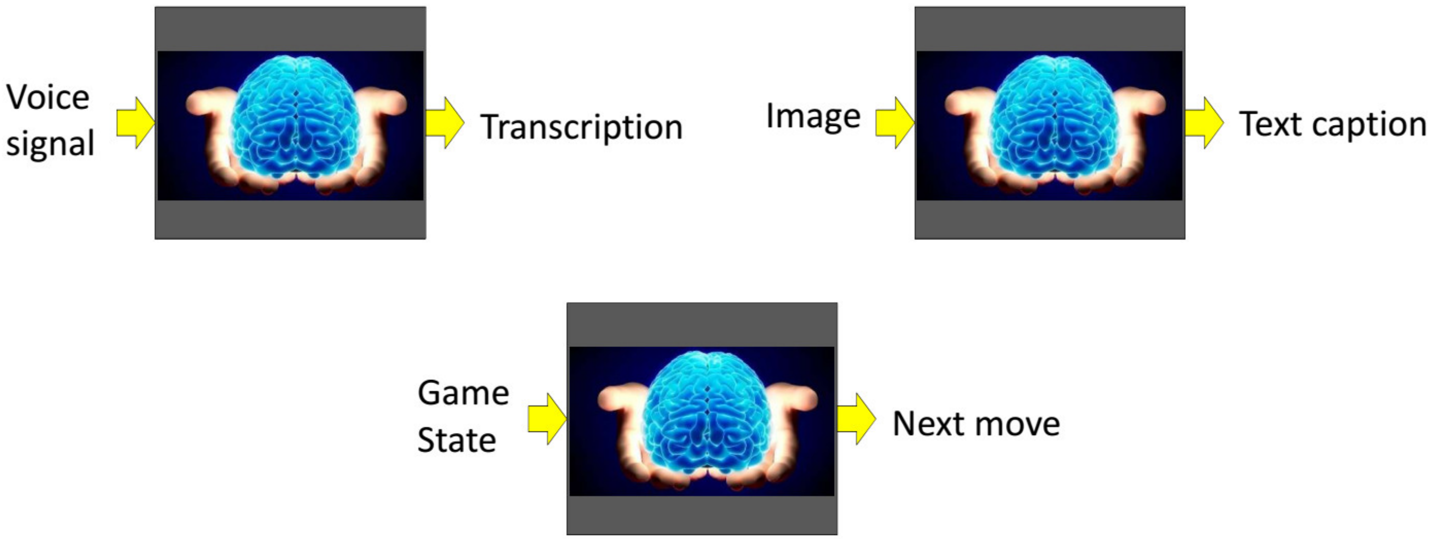
- Motivation
- Biological Background
- Threshold Logic Units
  - Geometric Interpretation
  - Limitations
  - Training
- Feed Forward Networks
  - Multilayer Perceptrons
  - Radial Basis Function Networks
- Autoencoder
- Spiking Neural Networks

Motivation: Human Perception

- Humans can:
  - Learn
  - Solve problems
  - Recognize patterns
  - Memorize
  - Percept
  - Cogitate
  - …
- In a human, All of these cognitive functions (ultimate goal of artificial intelligence) are computed by the brain.
- The structure and function of the brain has been studied for decades in the field of **neuroscience**.

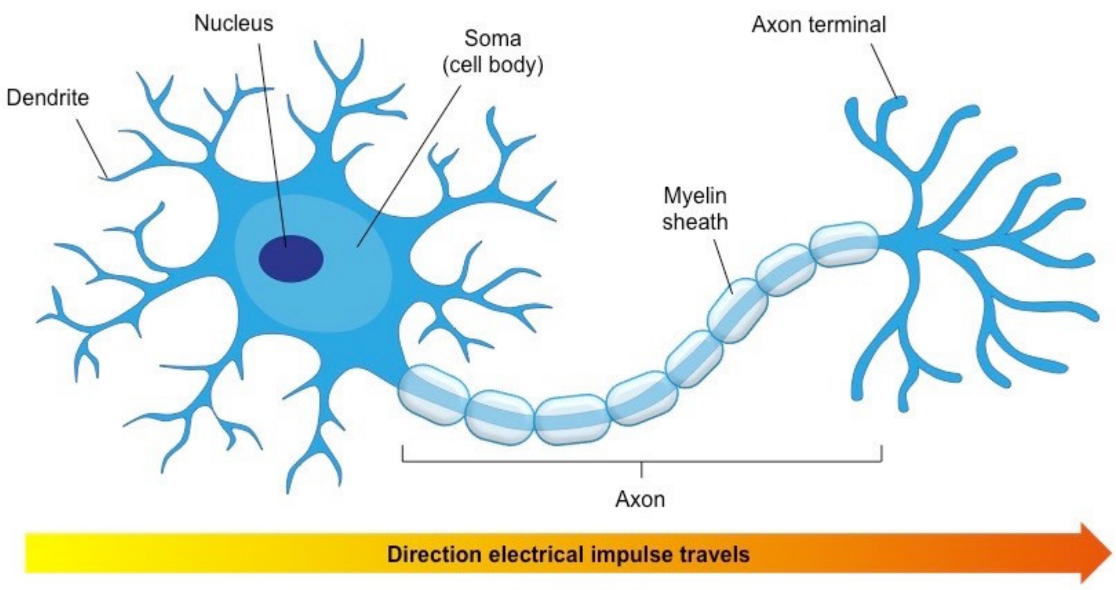
Motivation: Human Perception

- The Brain is composed of networks of neurons.
- Most of our knowledge is stored in the **connections** between the neurons.



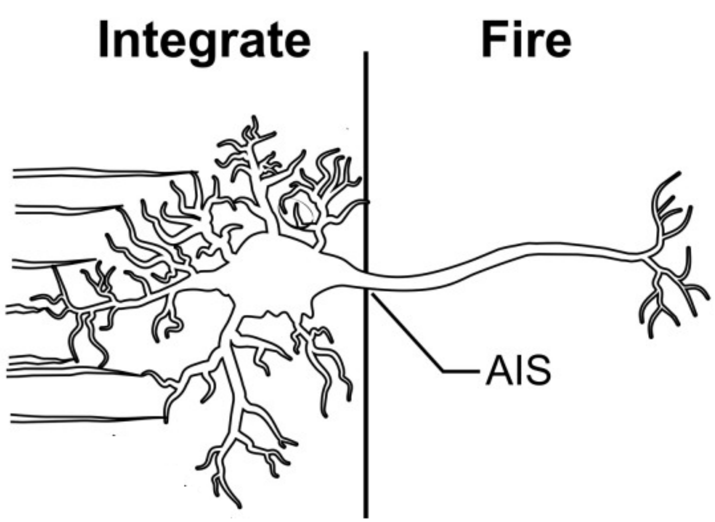
Biological Background

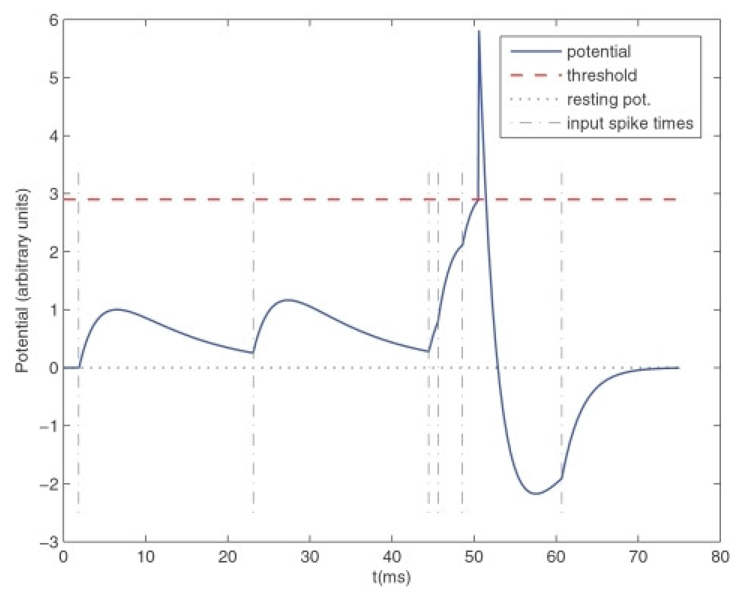
- There are about 10^{11} neurons in the brain.
- There are about 10^{15} connections (synapses) in the brain.



Biological Background

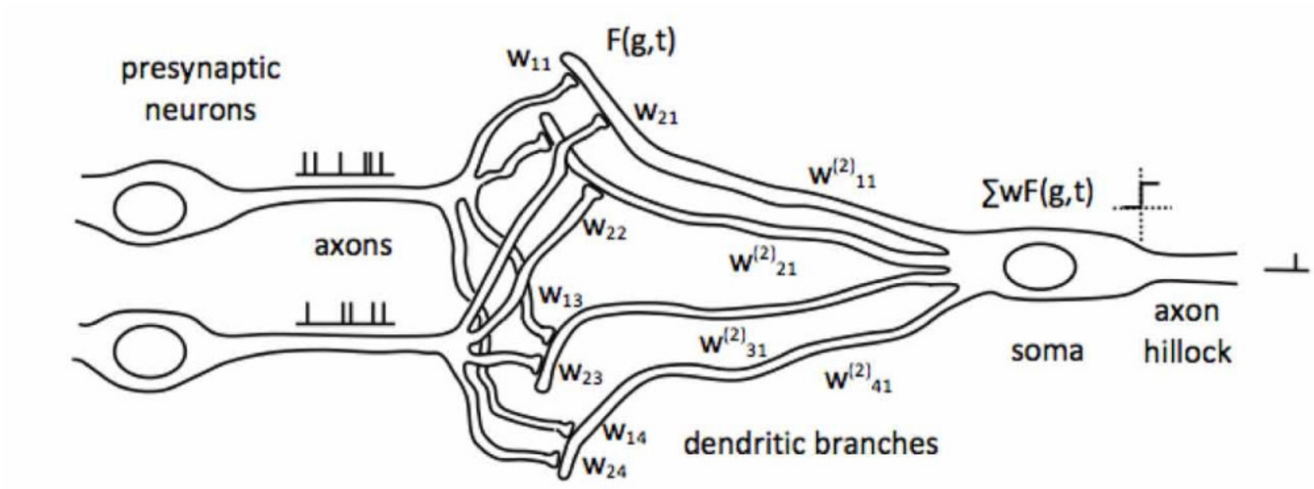
- Biological neuron: Integrate-and-fire model





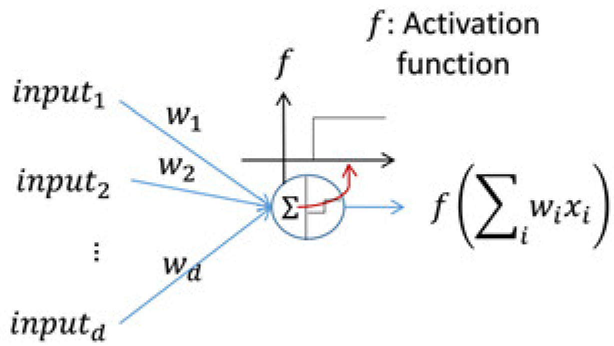
Biological Background

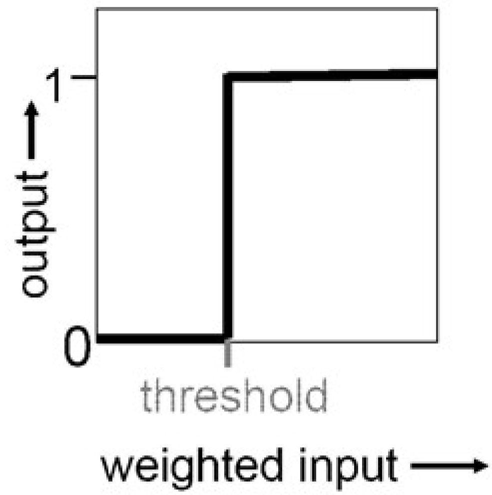
- Biological neuron: Integrate-and-fire model



Threshold Logic Units

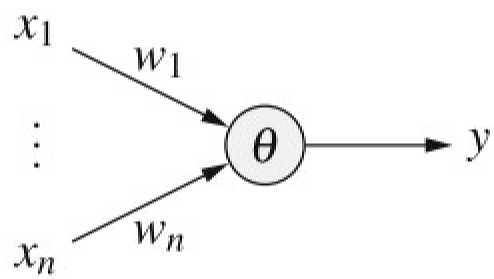
- First compute a weighted sum of the inputs.
- Then send out a spike of activity if the weighted sum exceeds a threshold.





McCulloch-Pitts, 1943

Threshold Logic Units

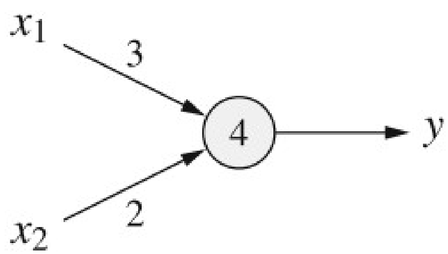


$$\mathbf{x} = (x_1, \ldots, x_n)$$

$$\mathbf{w} = (w_1, \ldots, w_n)$$

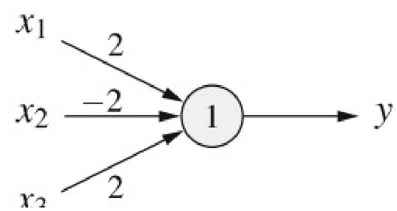
$$y = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0 \text{ otherwise.} \end{cases} \longrightarrow \mathbf{wx} \geq \theta$$

Logical Functions



x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

$x_1 \wedge x_2$



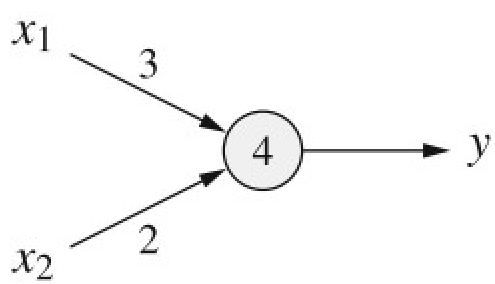
x_1	x_2	x_3	$\sum_i w_i x_i$	y
0	0	0	0	0
1	0	0	2	1
0	1	0	−2	0
1	1	0	0	0
0	0	1	2	1
1	0	1	4	1
0	1	1	0	0
1	1	1	2	1

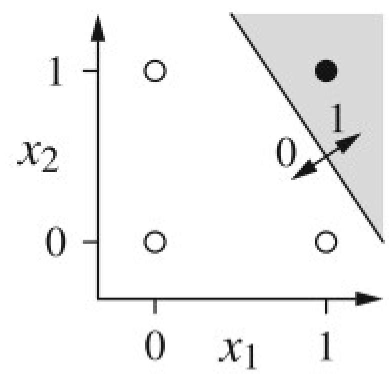
$(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$

Geometric Interpretation

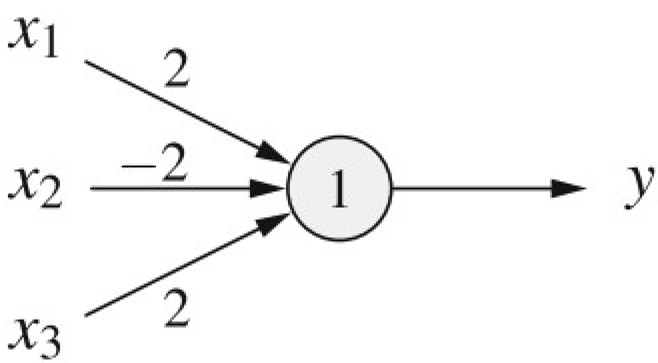
a hyperplane equation!

$$\sum_{i=1}^{n} w_i x_i = \theta \qquad \text{or} \qquad \sum_{i=1}^{n} w_i x_i - \theta = 0.$$

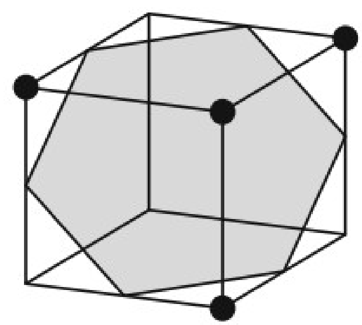




Geometric Interpretation



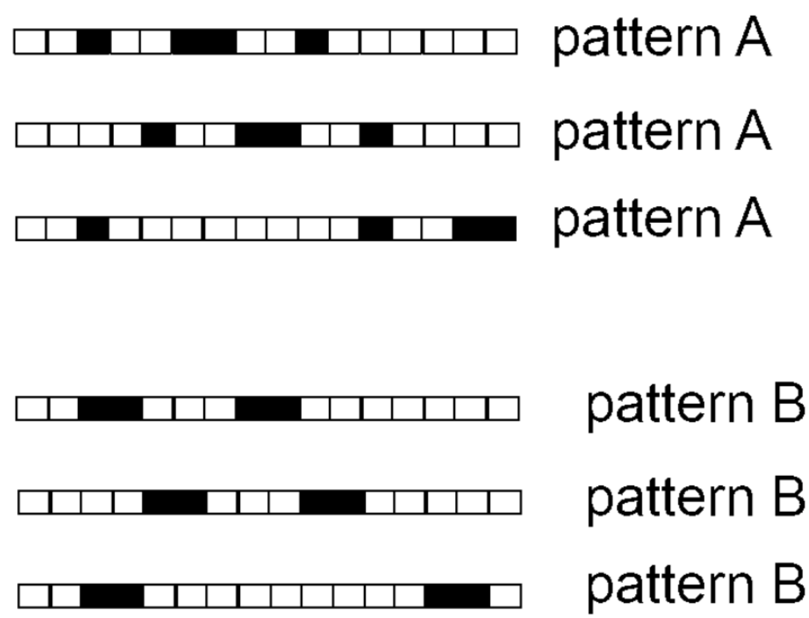
$$(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$$



equation $2x_1 - 2x_2 + 2x_3 = 1$

Limitations

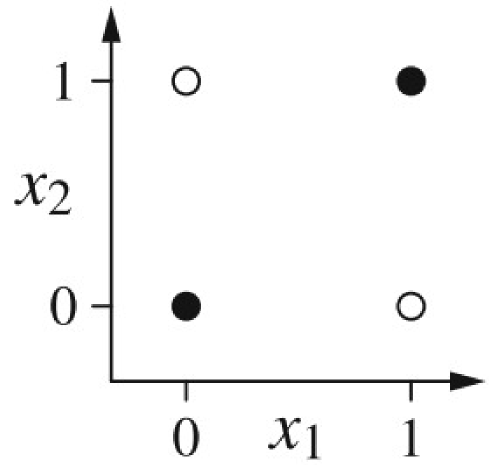
- Discriminating simple patterns under translation with wrap-around
- Binary decision unit cannot discriminate patterns with same number of "on" inputs



Limitations

- In some cases there is no separating *straight line*

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



Limitations

- In some cases there is no separating *straight line*

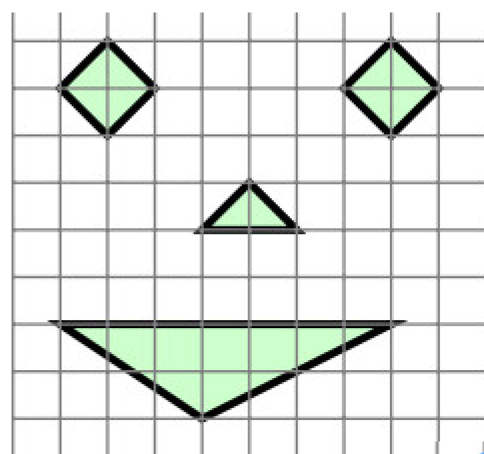
Inputs	Boolean functions	Linearly separable functions
1	$2^{2^1} = 4$	4
2	$2^{2^2} = 16$	14
3	$2^{2^3} = 256$	104
4	$2^{2^4} = 65536$	1774
5	$2^{2^5} \approx 4.3 \cdot 10^9$	94572
6	$2^{2^6} \approx 1.8 \cdot 10^{19}$	$5.0 \cdot 10^6$

Limitations

- In some cases there is no separating *straight line*

A SIMPLE SOLUTION:

Add another layer!



[shai. 2014]

Universal Logical Function

- Let $y = f(x_1, \ldots, x_n)$be a Boolean function.
- Represent the Boolean function $f(x_1, \ldots, x_n)$ in disjunctive normal form:

$$D_f = K_1 \vee \ldots \vee K_m \quad \text{, where} \qquad K_j = l_{j1} \wedge \ldots \wedge l_{jn}$$

- For first layer:

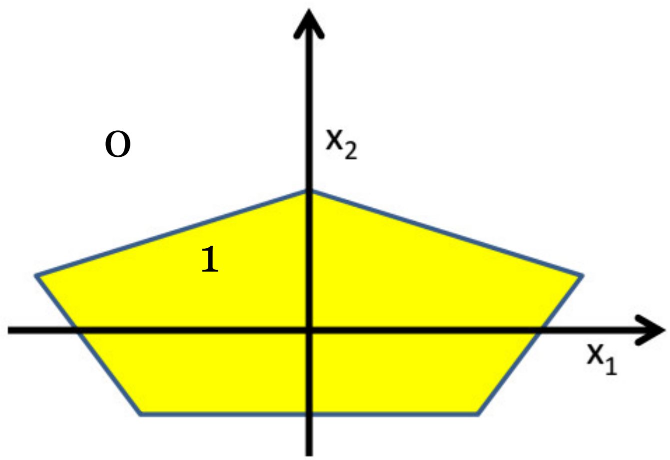
$$w_{ji} = \begin{cases} 2 \text{ if } l_{ji} = x_i \,, \\ -2 \text{ if } l_{ji} = \neg x_i \,, \end{cases} \qquad \text{and} \qquad \theta_j = n - 1 + \frac{1}{2} \sum_{i=1}^{n} w_{ji}$$

- For second layer:

$$w_{(n+1)k} = 2, \quad k = 1, \ldots, m, \qquad \text{and} \qquad \theta_{n+1} = 1.$$

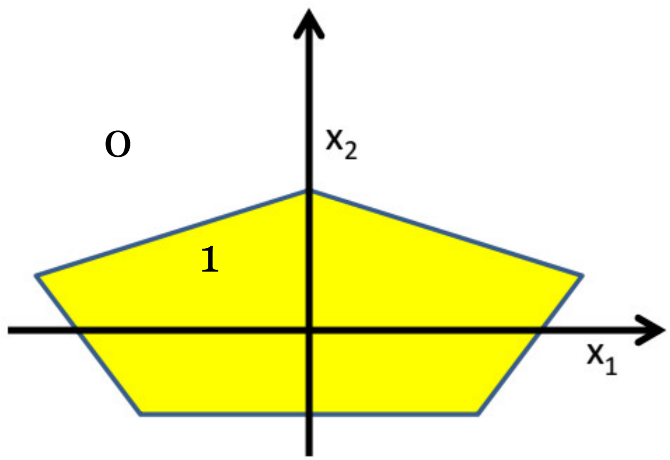
Complicated Decision Boundaries: Case Study

- Composing complicated "decision" boundaries



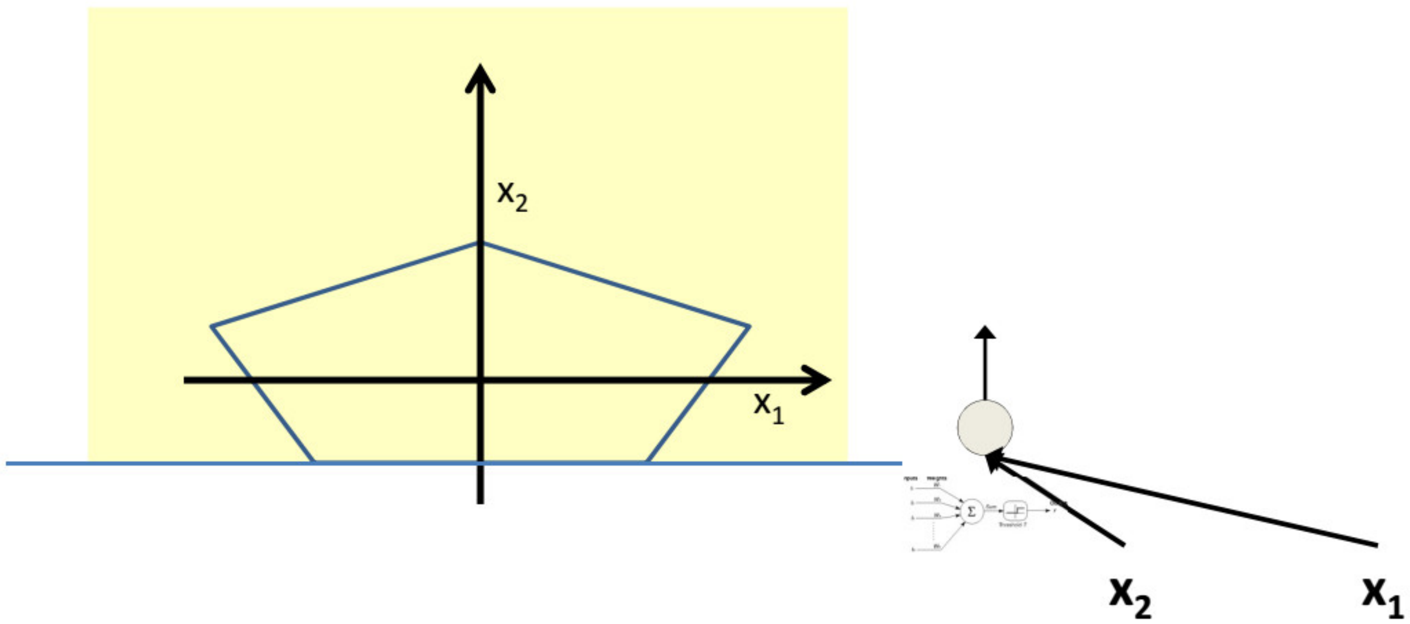
Complicated Decision Boundaries: Case Study

- Composing complicated "decision" boundaries



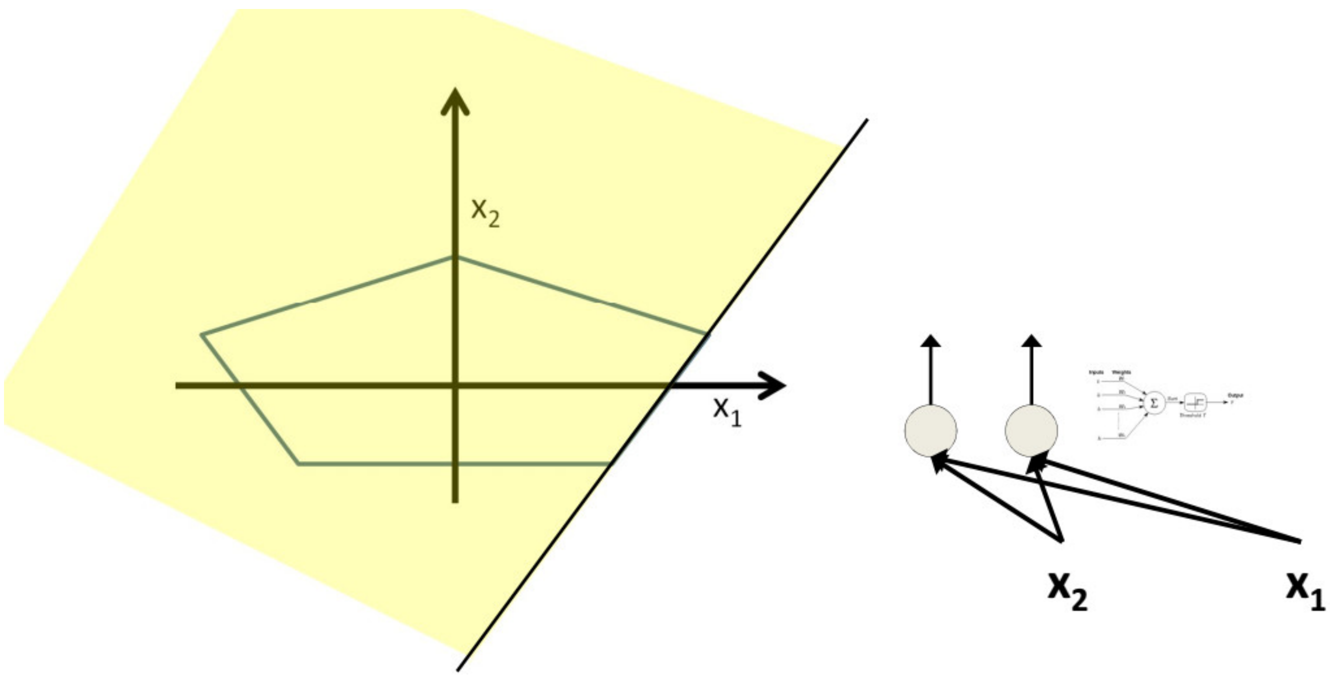
Complicated Decision Boundaries: Case Study

- Composing complicated "decision" boundaries



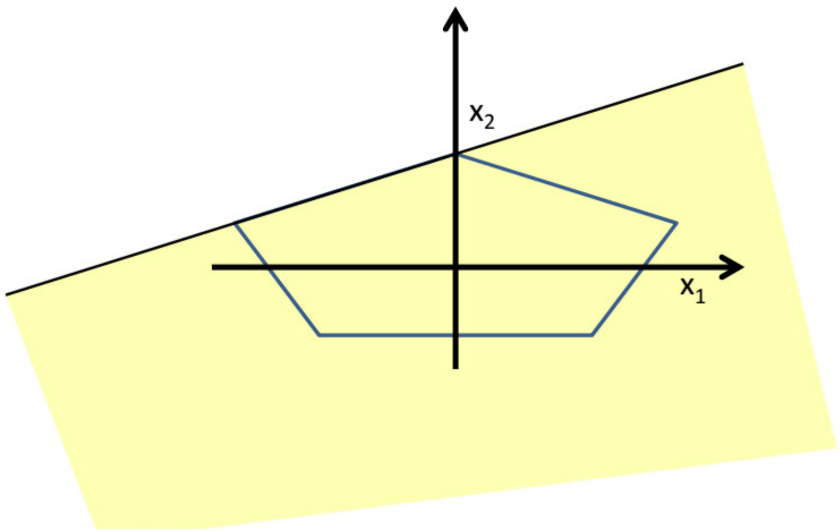
Complicated Decision Boundaries: Case Study

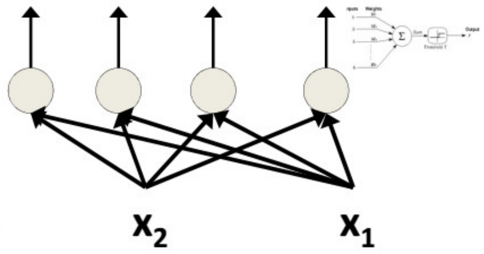
- Composing complicated “decision” boundaries



Complicated Decision Boundaries: Case Study

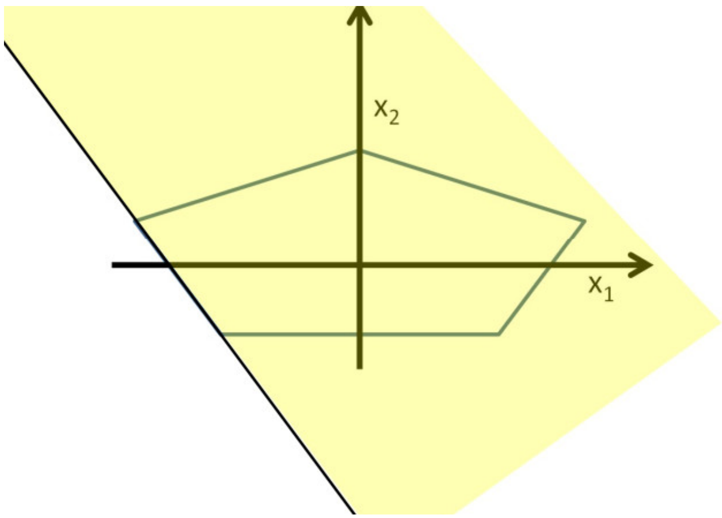
- Composing complicated "decision" boundaries

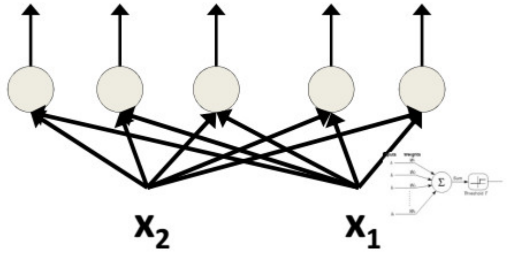




Complicated Decision Boundaries: Case Study

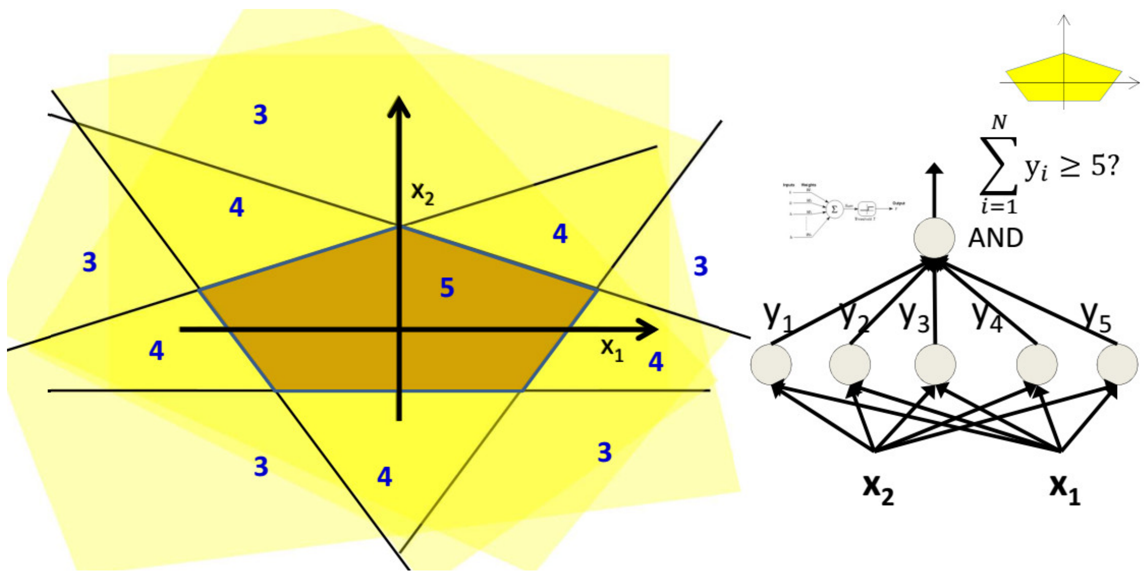
- Composing complicated "decision" boundaries



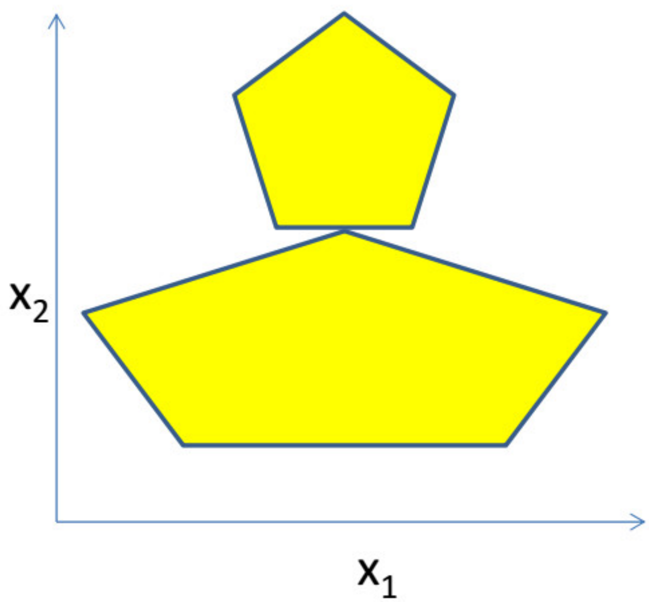


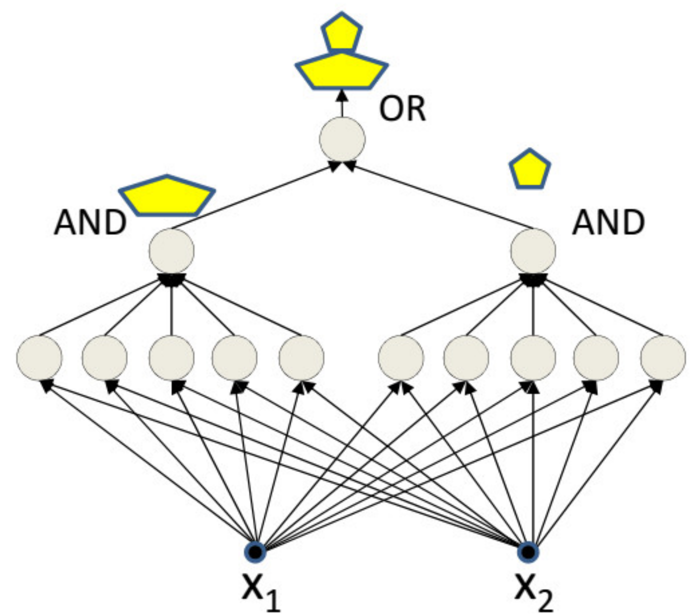
Complicated Decision Boundaries: Case Study

- Composing complicated "decision" boundaries



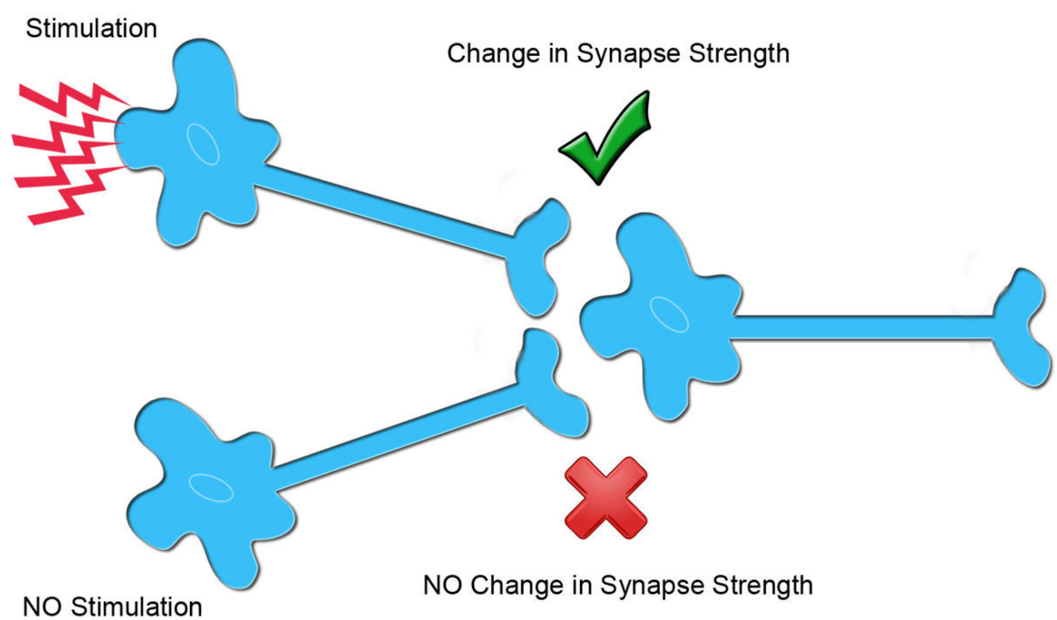
More complex decision boundaries





Learning Procedure: Synaptic Plasticity

- Synaptic plasticity is the ability of synapses to strengthen or weaken over time, in response to increases or decreases in their activity.
- One of the important foundation of learning and memory in brain.



Learning Procedure: Hebbian Learning

[Donald Hebb, 1949]

- Neurons that fire together wire together!
- Mechanism of rewiring:
  - Long-term depression
  - Long-term potentiation
- The importance role of unsupervised and semi-supervised learning in the brain!

Training Procedure

- Delta rule (Widrow and Hoff 1960):

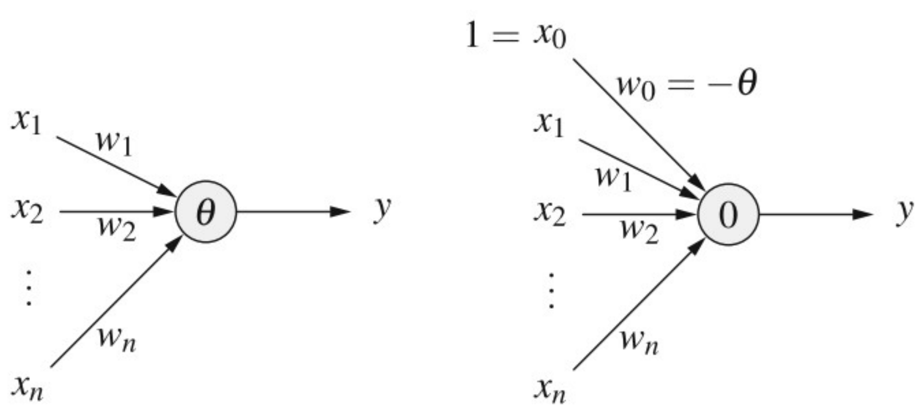
$$\theta^{(\text{new})} = \theta^{(\text{old})} + \Delta\theta \quad \text{with } \Delta\theta = -\eta(o-y),$$
$$w_i^{(\text{new})} = w_i^{(\text{old})} + \Delta w_i \ \text{ with } \Delta w_i = \quad \eta(o-y)x_i$$

- If the output unit is correct, leave its weights alone.
- If the output unit incorrectly outputs a zero, add the input vector to it.
- If the output unit incorrectly outputs a 1, subtract the input vector from it.

Training Procedure

- Delta rule (Widrow and Hoff 1960):

$$\theta^{(\text{new})} = \theta^{(\text{old})} + \Delta\theta \quad \text{with } \Delta\theta = -\eta(o-y),$$
$$w_i^{(\text{new})} = w_i^{(\text{old})} + \Delta w_i \quad \text{with } \Delta w_i = \eta(o-y)x_i$$



$$\sum_{i=1}^{n} w_i x_i \geq \theta \qquad \sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

Turning the threshold into a weight

Training Procedure

- Delta rule (Widrow and Hoff 1960):

$$\theta^{(\text{new})} = \theta^{(\text{old})} + \Delta\theta \quad \text{with } \Delta\theta = -\eta(o-y),$$
$$w_i^{(\text{new})} = w_i^{(\text{old})} + \Delta w_i \text{ with } \Delta w_i = \quad \eta(o-y)x_i$$

Problem!

❖ With binary encoding, with an input of false (0) the corresponding weight cannot be changed. (slows down training)

✓ Solution: **ADALINE** model (ADAptive LINear Element) encoding false as -1 and true as 1.

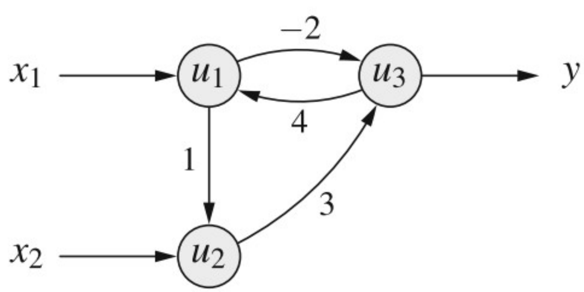
Training Procedure (Problem)

- The values of previous layers aren't available, so delta rule just can performed on one layer single neuron!

The “**dark age**” of neural network research began ...

General Neural Networks

- Represented with directed graph
- Each computational node has it’s own **successors** and **predecessors**.



$$\begin{array}{c c} \begin{matrix} u_1 & u_2 & u_3 \end{matrix} & \\ \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & 0 \\ -2 & 3 & 0 \end{pmatrix} & \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} \end{array}$$

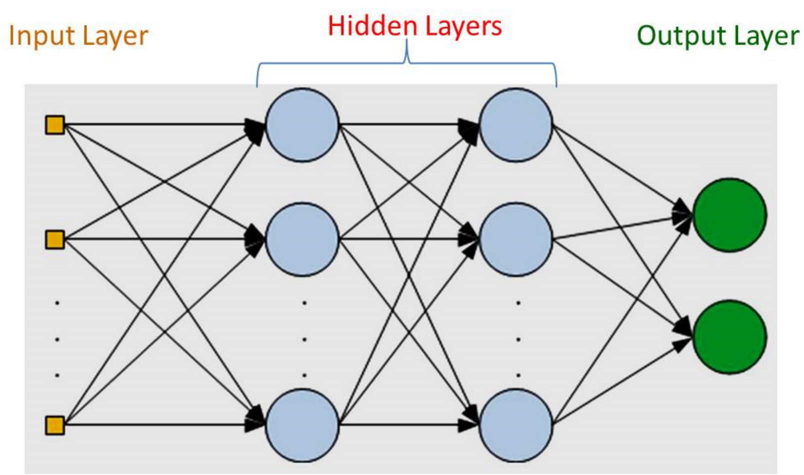
$$\begin{array}{c c} \begin{matrix} u_1 & u_2 & \dots & u_r \end{matrix} & \\ \begin{pmatrix} w_{u_1u_1} & w_{u_1u_2} & \dots & w_{u_1u_r} \\ w_{u_2u_1} & w_{u_2u_2} & & w_{u_2u_r} \\ \vdots & & & \vdots \\ w_{u_ru_1} & w_{u_ru_2} & \dots & w_{u_ru_r} \end{pmatrix} & \begin{matrix} u_1 \\ u_2 \\ \vdots \\ u_r \end{matrix} \end{array}$$

General Neural Networks

- Categorization based on feedback:
  - Multilayer Perceptrons
  - Radial Basis Function Networks
  - Self-organizing Maps
  - Hopfield Networks
  - Recurrent Networks

Feedforward Neural Networks

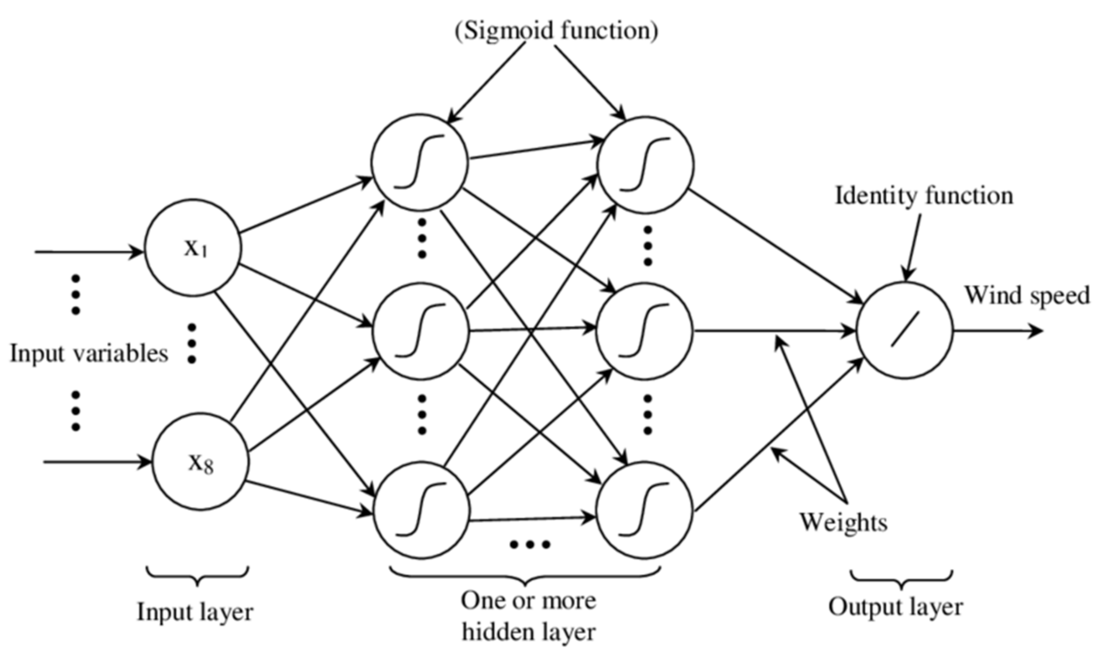
- Multilayer Perceptrons
- Radial Basis Function Networks



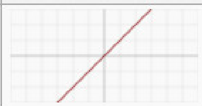
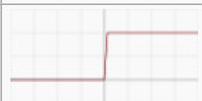
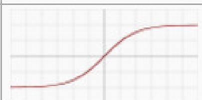
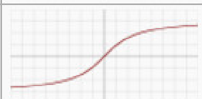
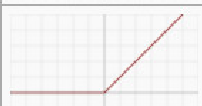
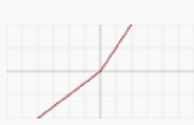
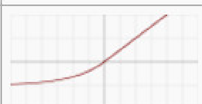
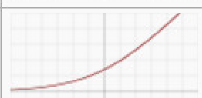
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Multilayer Perceptrons (MLP)

- An MLP consists of multiple layers of nodes with each layer fully connected to the next one.
- Able to model complex non-linear functions.
- Utilize backpropagation to train network



Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Activation Functions

- Networks without hidden units are very limited in the input-output mappings they can learn to model
- We need multiple layers of adaptive, non-linear hidden units
- What if we use linear activation functions?

Problem with linear activation functions

Consider $z = f_{act}(y) = \alpha y \ - \theta = \ \alpha(wx) \ - \theta$

for second layer:

$$z_1 = \alpha_1 y_1 - \theta_1 = \alpha_1(w_1 x_1) - \theta_1$$
$$z_2 = \alpha_2 y_2 - \theta_2 = \alpha_2(w_2 x_2) - \theta_2$$
$$x_2 = z_1$$

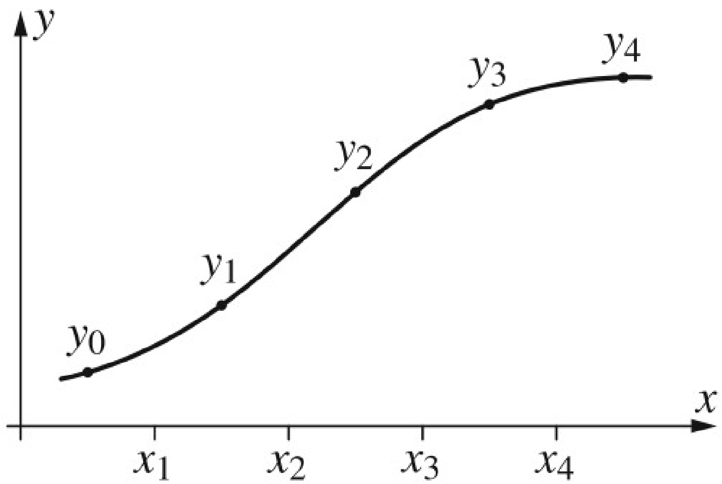
thus

$$z_2 = \alpha_2\big(w_2(\alpha_1(w_1 x_1) - \theta_1)\big) - \theta_2$$
$$= \alpha_1 \alpha_2 (w_1 w_2 x_1) - [\alpha_2 w_2 \theta_1 - \theta_2] = \alpha'(w'x) - \theta'$$

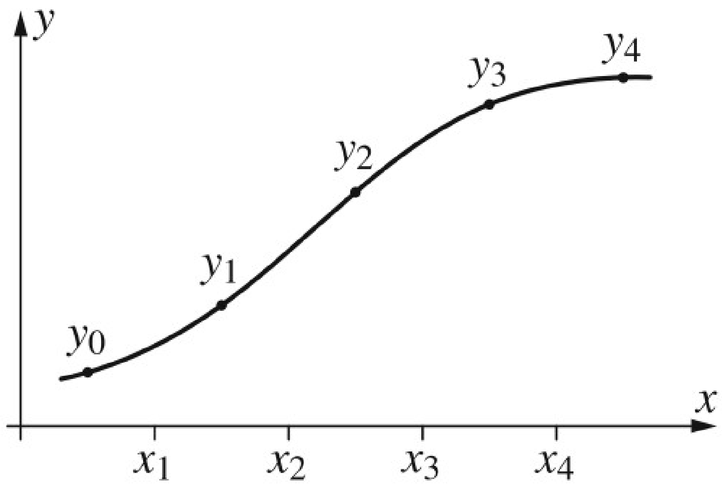
, where

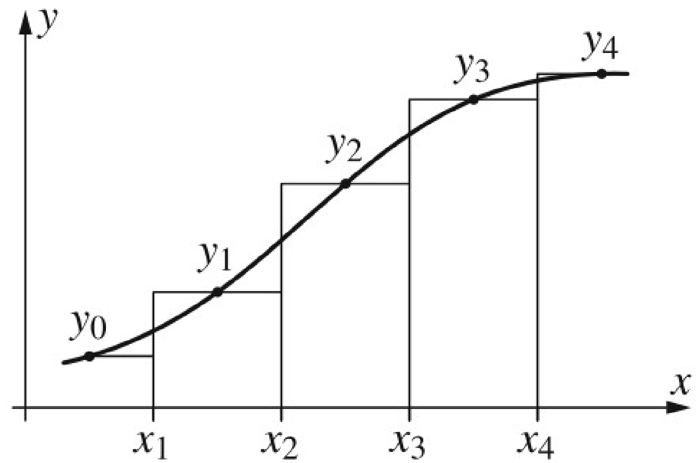
$\alpha' = \alpha_1 \alpha_2 \ , w' = \ w_1 w_2 \ , \theta' = [\alpha_2 w_2 \theta_1 - \theta_2]$

Function Approximation

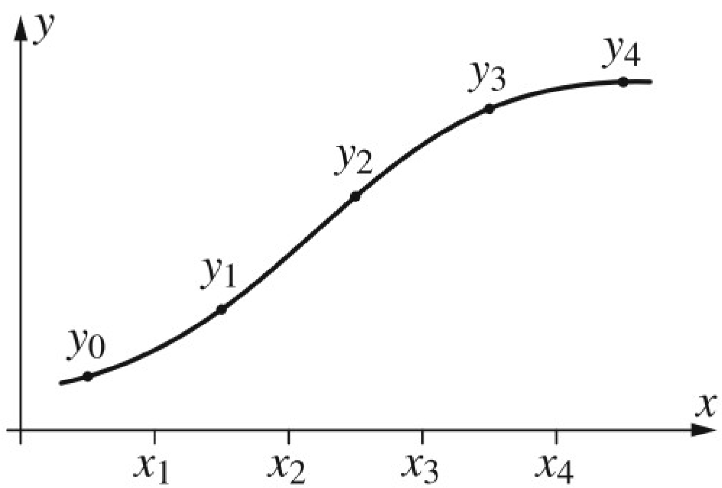


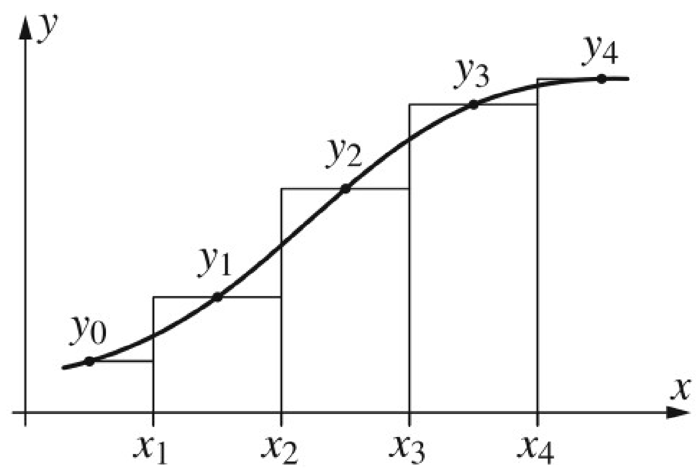
Function Approximation

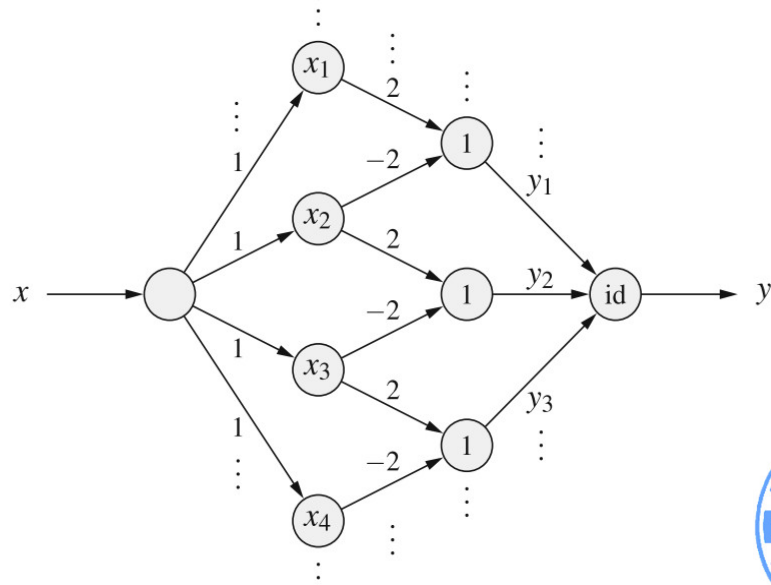




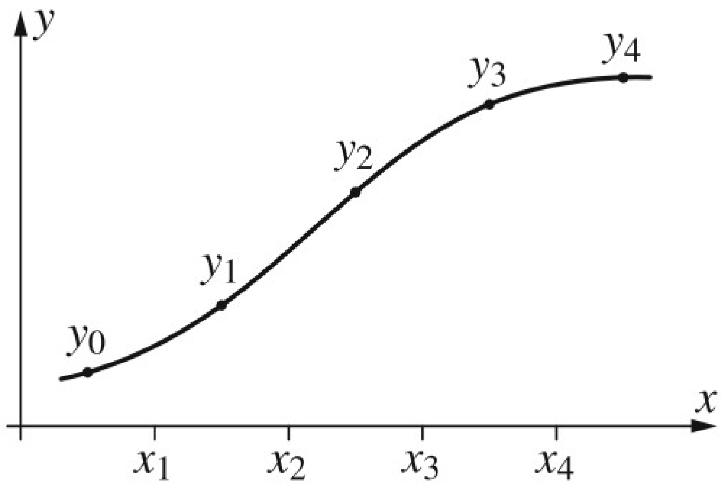
Function Approximation

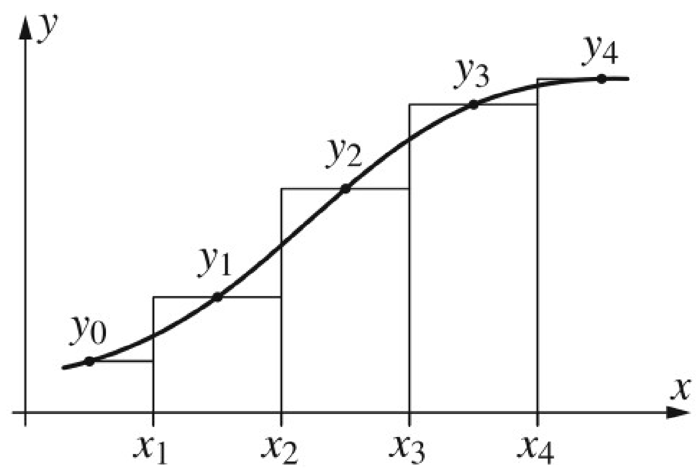


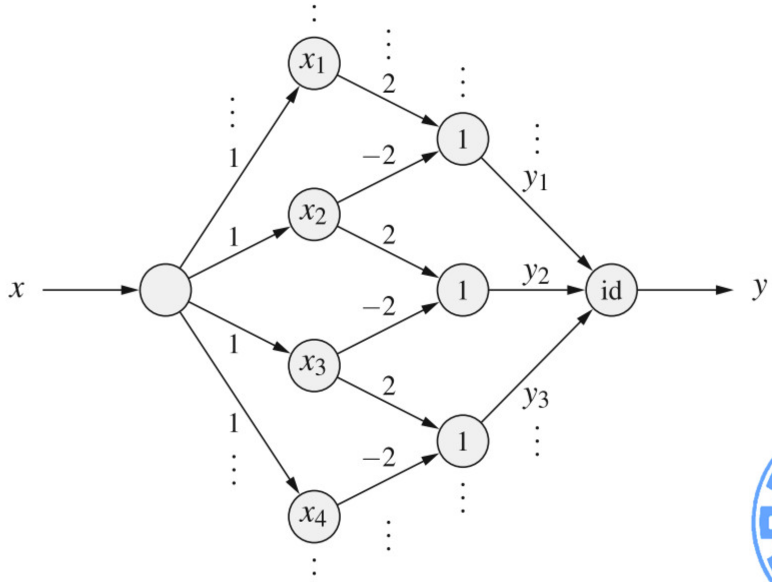




Function Approximation







Multilayer feedforward networks
are **universal approximators!**

Radial Basis Function Networks

- The network input function of each hidden neuron is a **distance function** of the input vector and the weight vector.

$$f_{\text{net}}^{(u)}(\mathbf{w}_u, \text{in}_u) = d(\mathbf{w}_u, \text{in}_u)$$

$$f : \mathbb{R}_0^+ \to [0, 1] \quad \text{with} \quad f(0) = 1 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 0.$$

Radial Basis Function Networks

- The network input function of each hidden neuron is a **distance function** of the input vector and the weight vector.

$$f_{\mathrm{net}}^{(u)}(\mathbf{w}_u, \mathrm{in}_u) = d(\mathbf{w}_u, \mathrm{in}_u)$$

- Well-known family of distance functions (*Minkowski* family):

$$d_k(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} (x_i - y_i)^k\right)^{\frac{1}{k}}$$

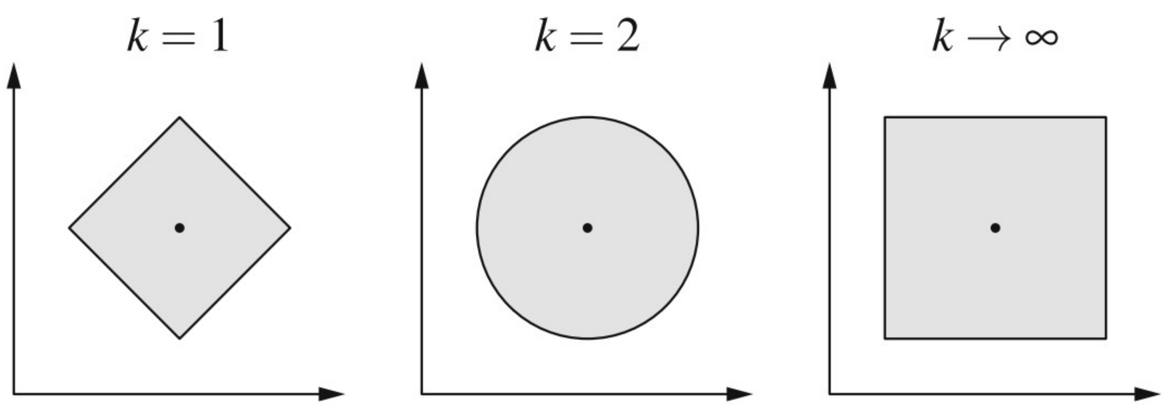
Radial Basis Function Networks

Well-known special members of this family are:

$k = 1$: Manhattan or city block distance,

$k = 2$: Euclidean distance,

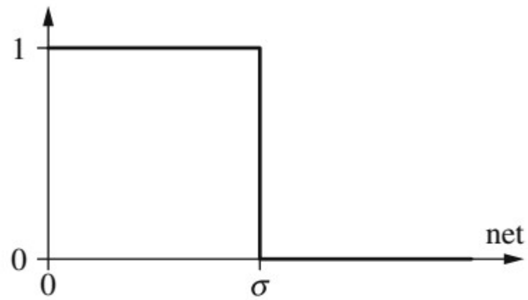
$k \to \infty$: Maximum distance, that is, $d_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1}^{n} |x_i - y_i|$.



Radial activation functions

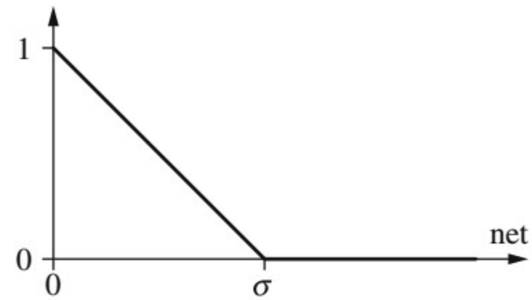
rectangular function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0 & \text{if net} > \sigma, \\ 1 & \text{otherwise.} \end{cases}$$



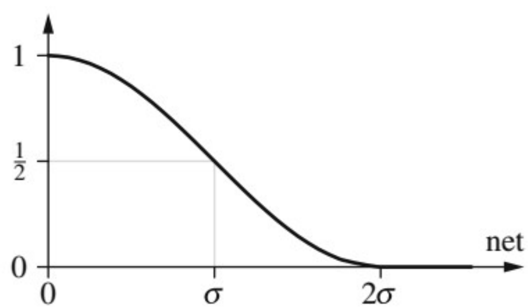
triangular function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0 & \text{if net} > \sigma, \\ 1 - \frac{\text{net}}{\sigma} & \text{otherwise.} \end{cases}$$



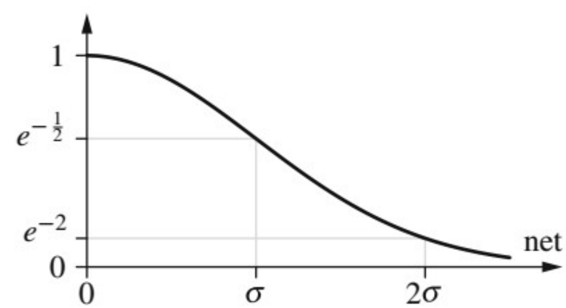
cosine down to zero:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0 & \text{if net} > 2\sigma, \\ \frac{\cos\left(\frac{\pi}{2\sigma}\text{net}\right)+1}{2} & \text{otherwise.} \end{cases}$$

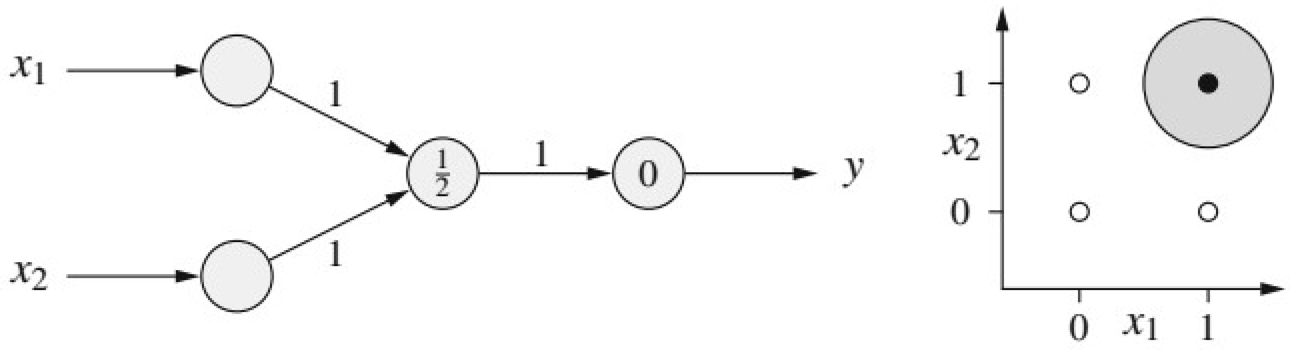


Gaussian function:

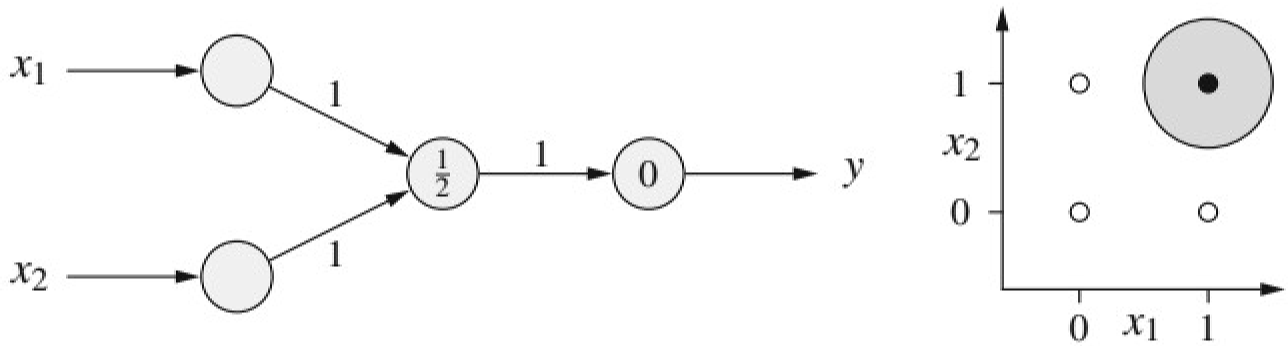
$$f_{\text{act}}(\text{net}, \sigma) = e^{-\frac{\text{net}^2}{2\sigma^2}}$$

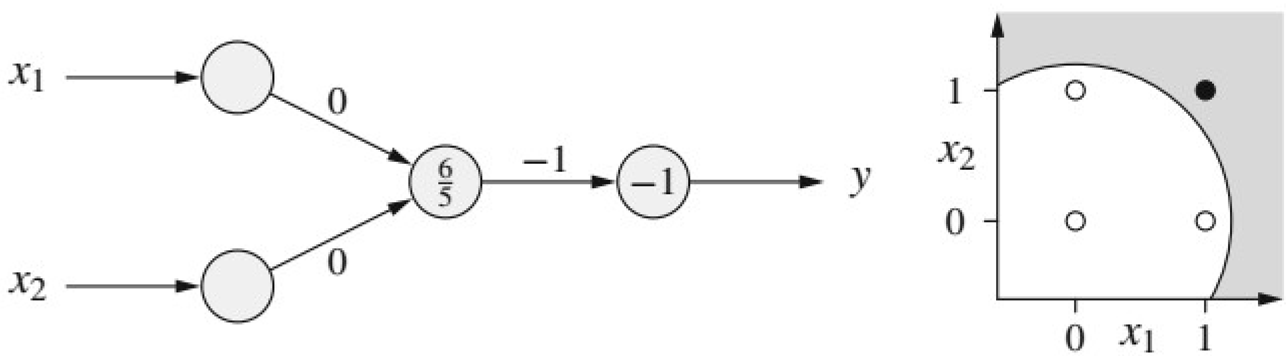


RBFNet exapmles

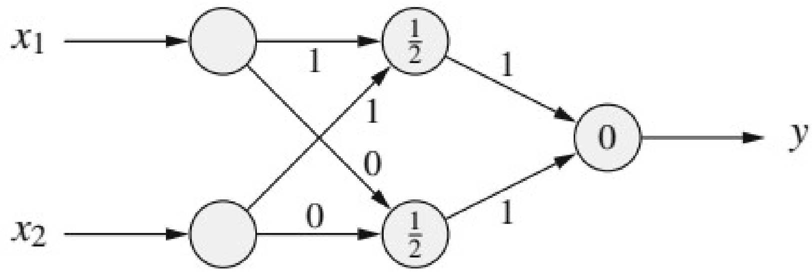


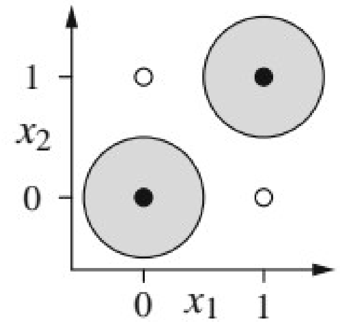
RBFNet exapmles



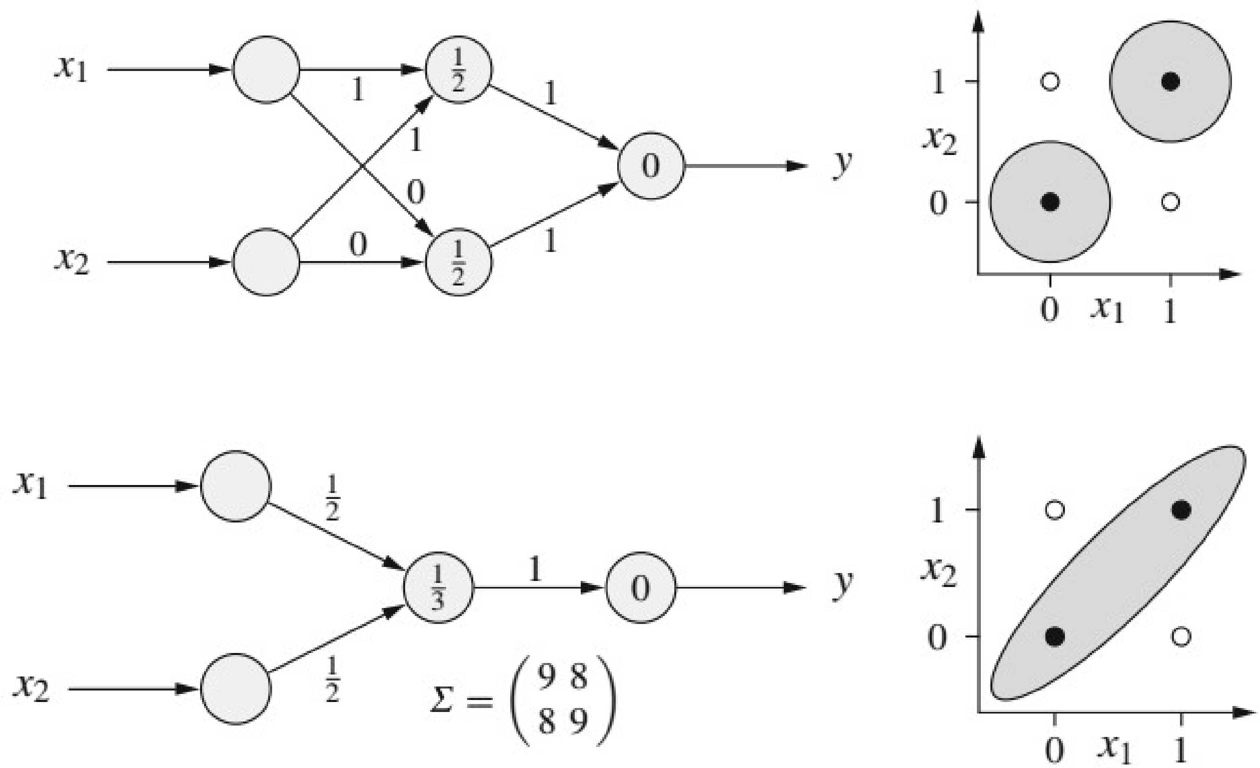


RBFNet exapmles





RBFNet exapmles

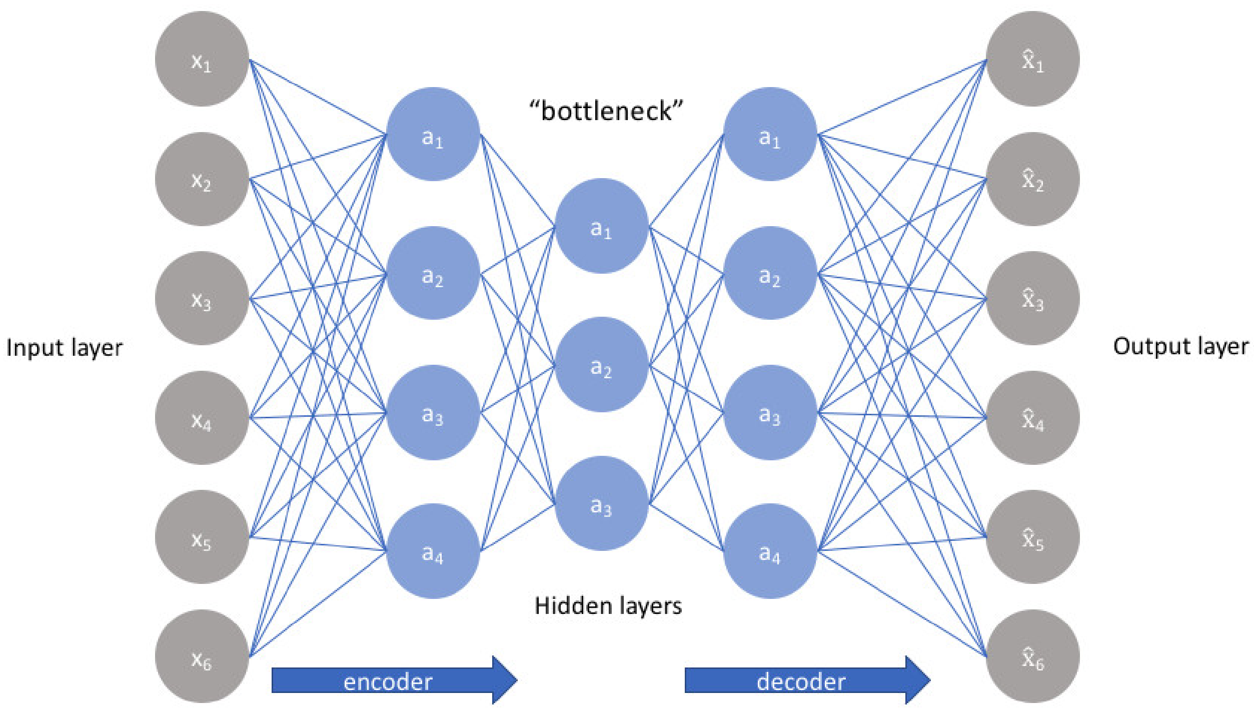


Mahalanobis distance, which is defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$$

Autoencoder

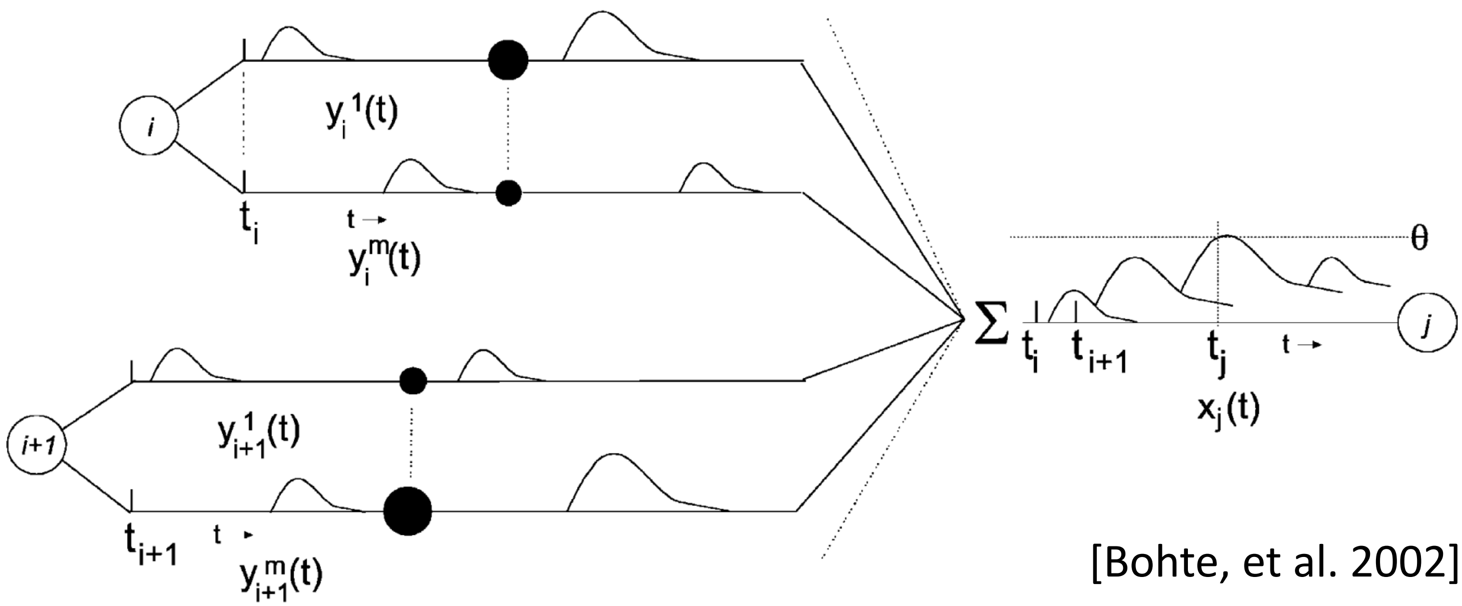
- Representation or manifold learning
- Data denoising
- Feature reduction (like PCA)



[jeremyjordan.me/autoencoders/]

Spiking Neural Networks (SNN)

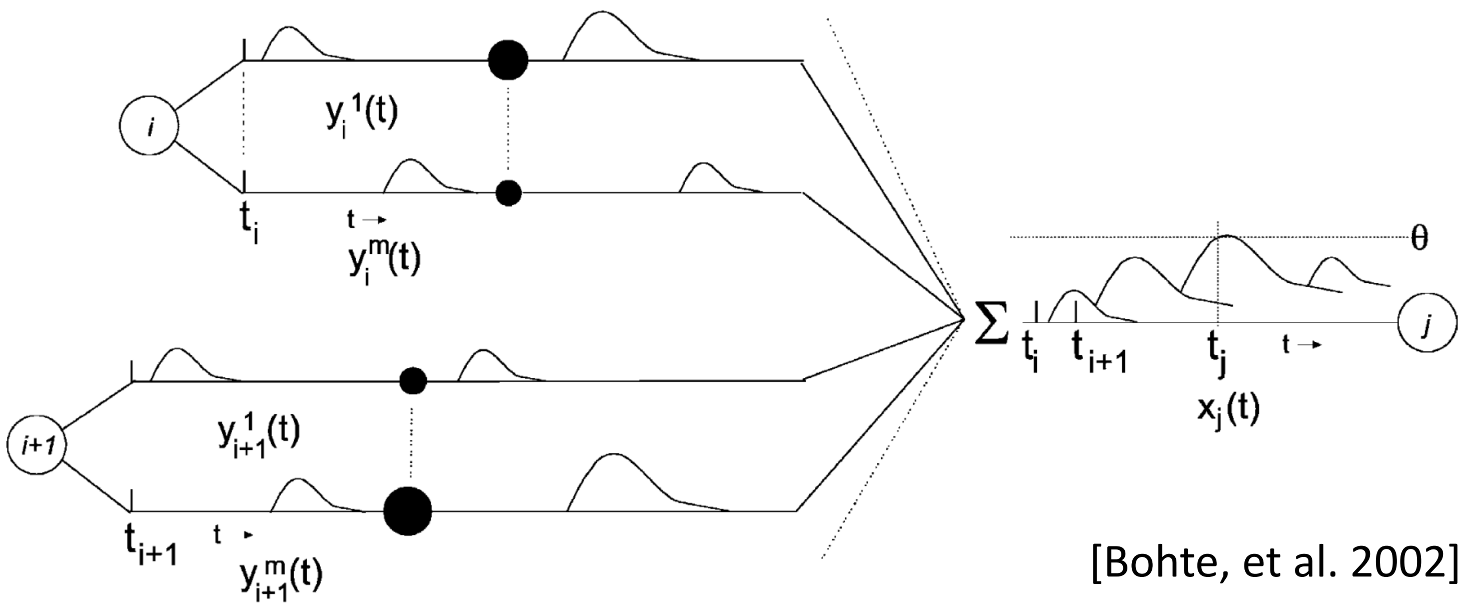
- A SNN cell will fire when the accumulated stimuli in specific time t_j reaches above a threshold value.



[Bohte, et al. 2002]

Spiking Neural Networks (SNN)

- More biologically plausible (Neural dynamics, temporal coding)
- Theoretical higher upper bounds in capacity
- More computational costs



[Bohte, et al. 2002]

Main Resource

- Kruse, Rudolf, et al. **Computational intelligence: a methodological introduction**. Springer, 2016.
- 11-785 **Introduction to Deep Learning**, CMU, spring 2019