NEURAL NETWORKS

Terminology

If $\mathbf{w}^T\mathbf{x} + w_0 > 0$ assign $\mathbf{x}$ to ω_1

If $\mathbf{w}^T\mathbf{x} + w_0 < 0$ assign $\mathbf{x}$ to ω_2

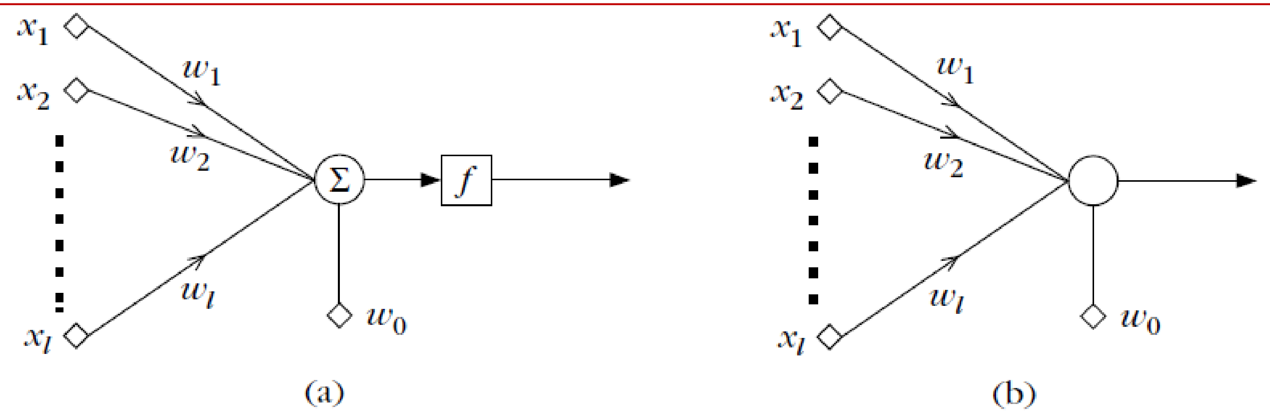


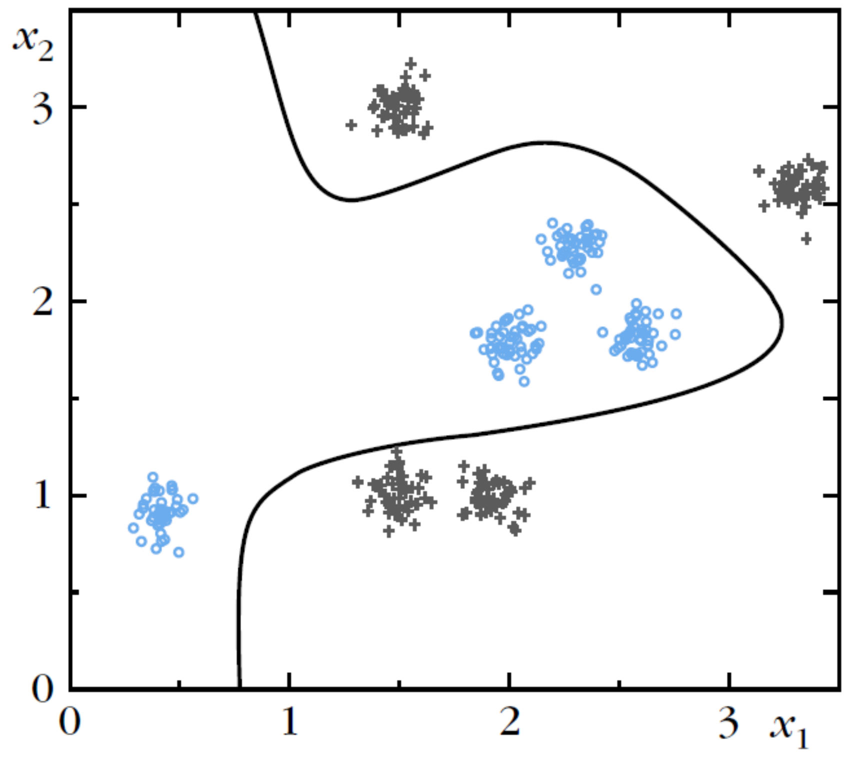
FIGURE 3.5

The basic perceptron model. (a) A linear combiner is followed by the activation function. (b) The combiner and the activation function are merged together.

- **Perceptron** or **neuron**
- **Synaptic weights** or **synapses**
- **Activation function**: *e.g.* $f(x) = \delta(x)$

Nonlinear Classifiers

We deal with problems that are not linearly separable



ONE! TWO! THREE!

One-Layer Perceptron

- XOR problem is not linearly separable

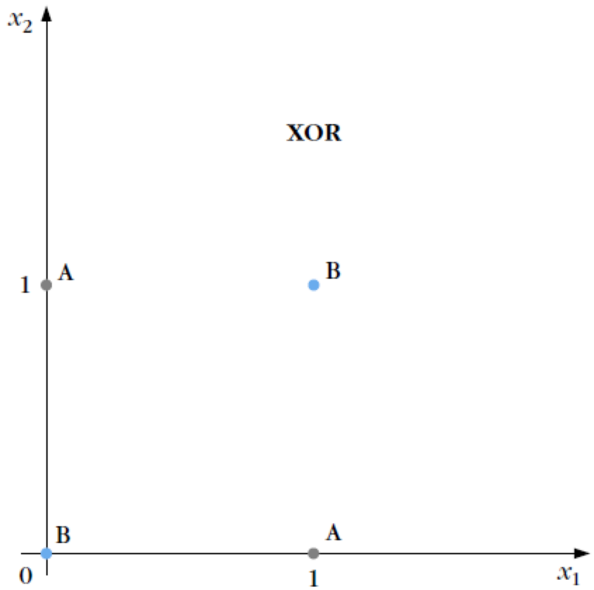
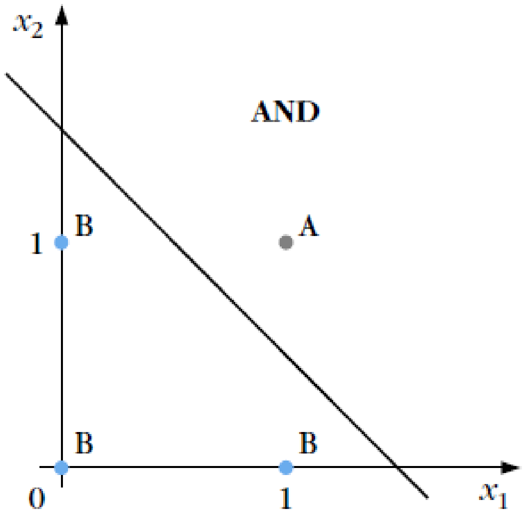


Table 4.1 Truth Table for the XOR Problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

One-Layer Perceptron

- AND and OR problems are linearly separable



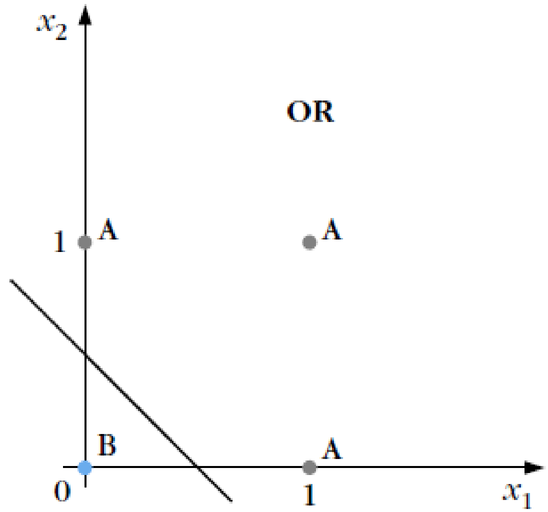
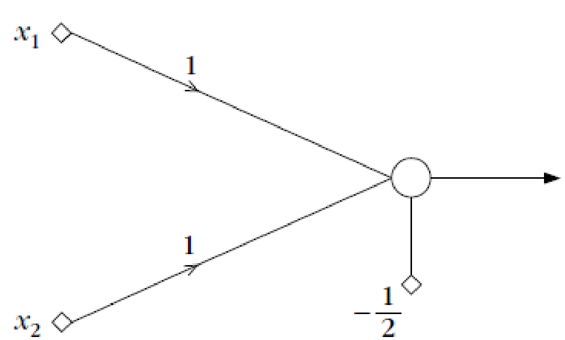


Table 4.2 Truth Table for AND and OR Problems

x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A

1-layer perceptron implementation



Two-Layer Perceptron

- XOR problem: solve it in two successive phases
  - 1st phase (or layer) uses two lines

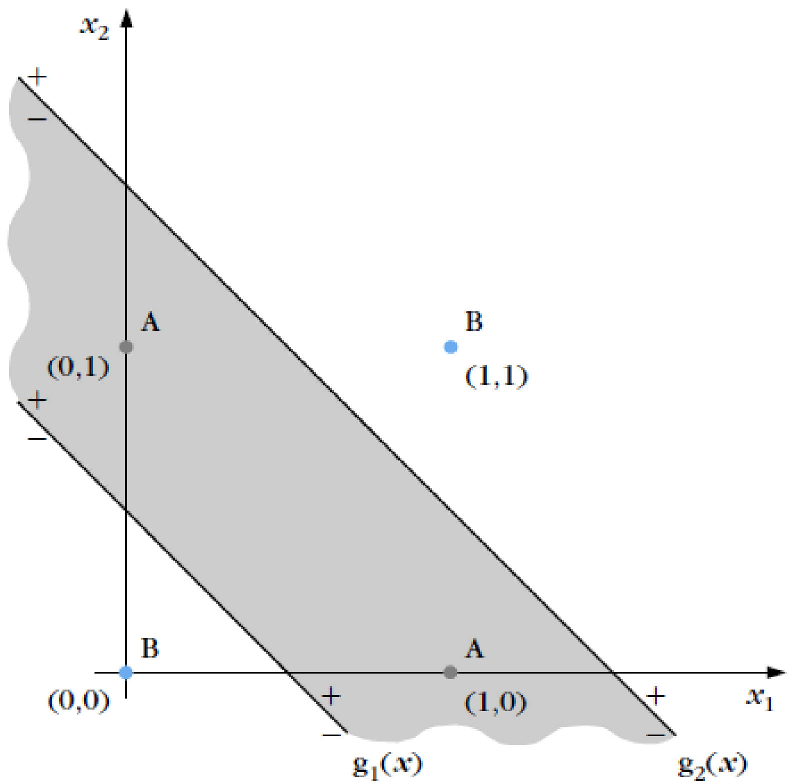


Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

Two-Layer Perceptron

- XOR problem: solve it in two successive phases
  - 2^{nd} phase

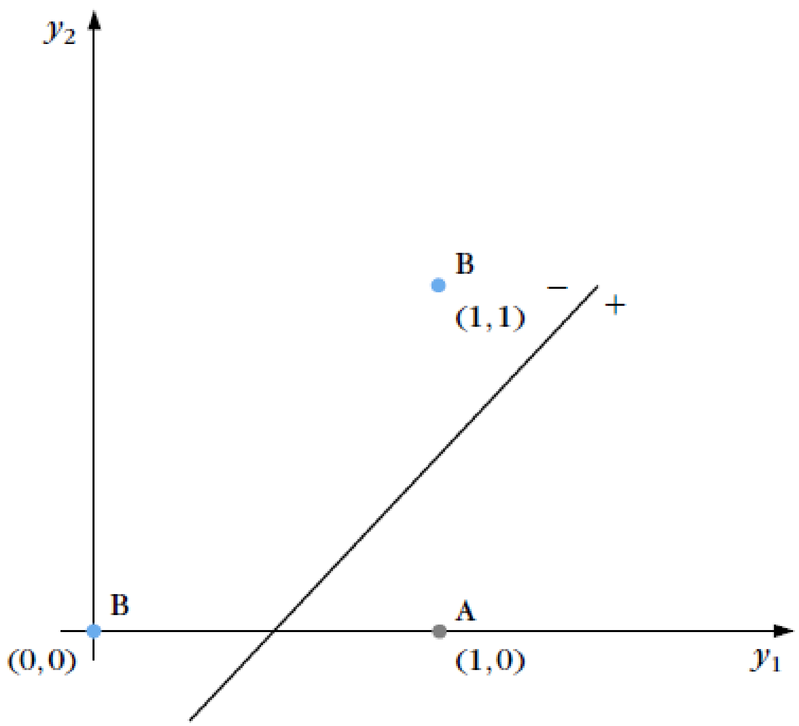
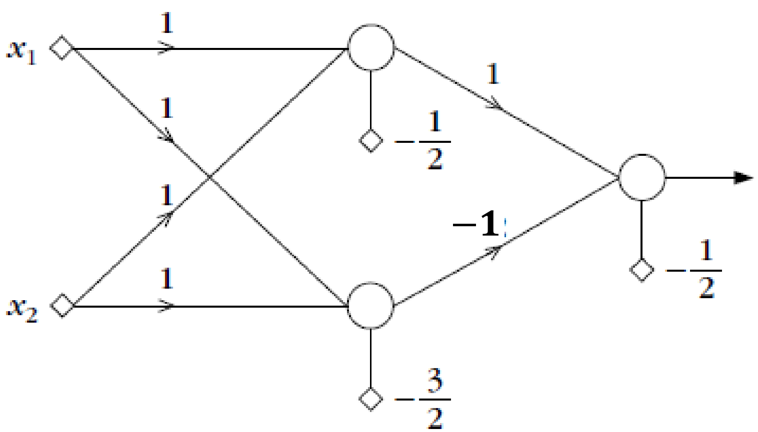


Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

Two-Layer Perceptron

- XOR problem: solve it in two successive phases
  - 2-layer perceptron (or 2-layer feedforward neural network)



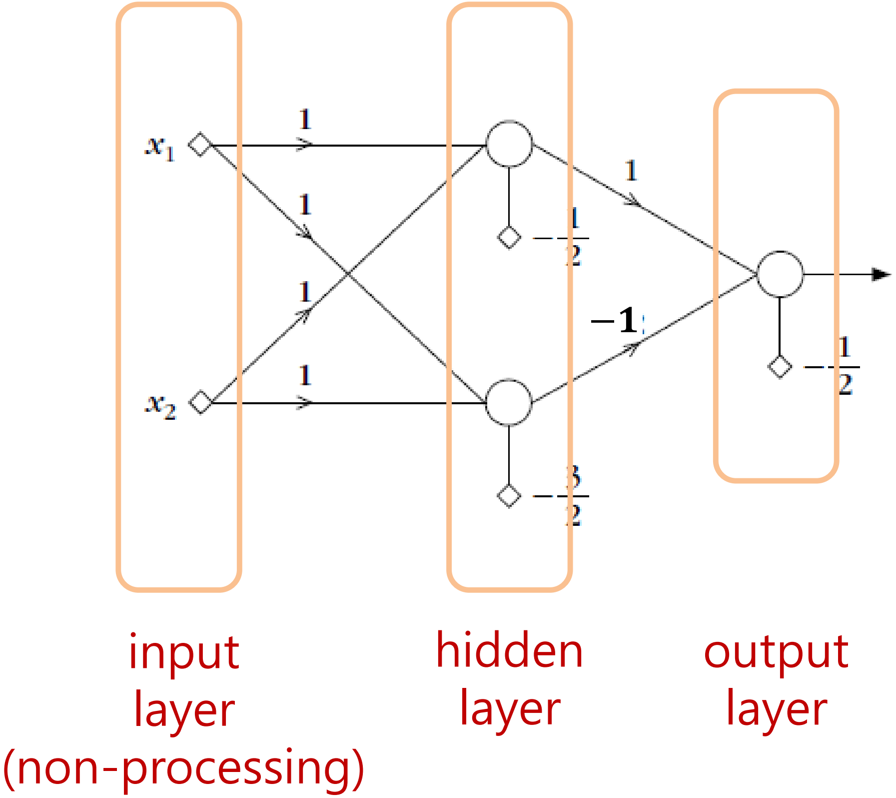
- $g_1(\mathbf{x}) = x_1 + x_2 - \frac{1}{2} = 0$
- $g_2(\mathbf{x}) = x_1 + x_2 - \frac{3}{2} = 0$
- $g(\mathbf{y}) = y_1 - y_2 - \frac{1}{2} = 0$

Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

Two-Layer Perceptron

- Terminology
  - 2-layer perceptron (or 2-layer feedforward neural network)

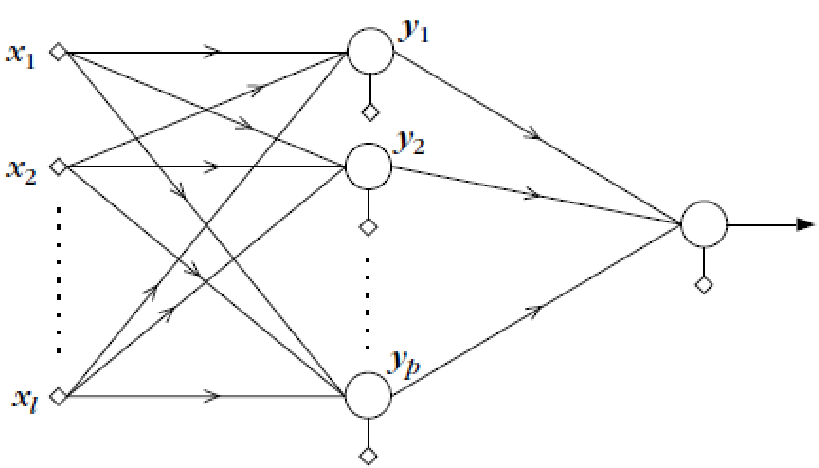


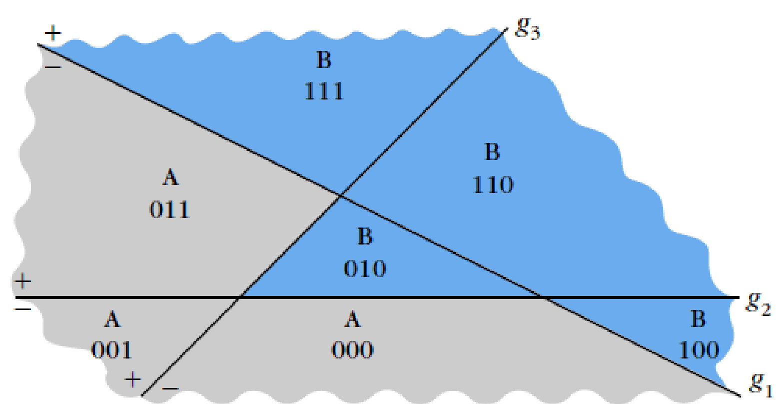
Two-Layer Perceptron

- Classification capabilities of two-layer perceptron
  - 1st layer maps input to vertices of the unit hypercube

$$H_p = \left\{ \left[y_1, \ldots, y_p \right]^T \in \mathbb{R}^p \colon \ y_i \in [0,1] \ \text{ for } \ 1 \le i \le p \right\}$$

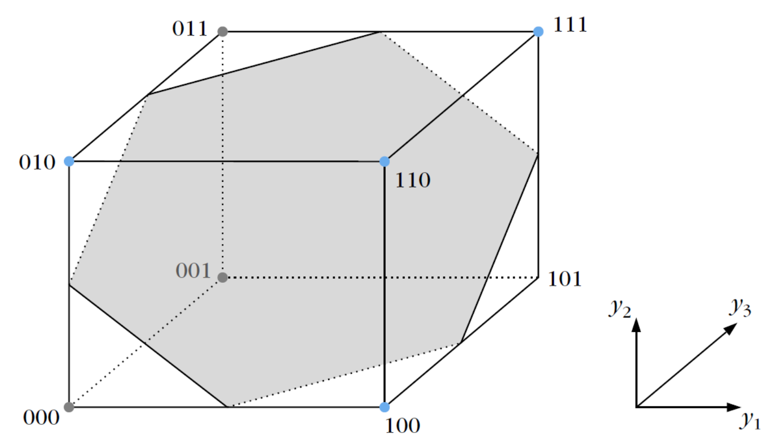
  - An output of 1st layer corresponds to a polyhedron

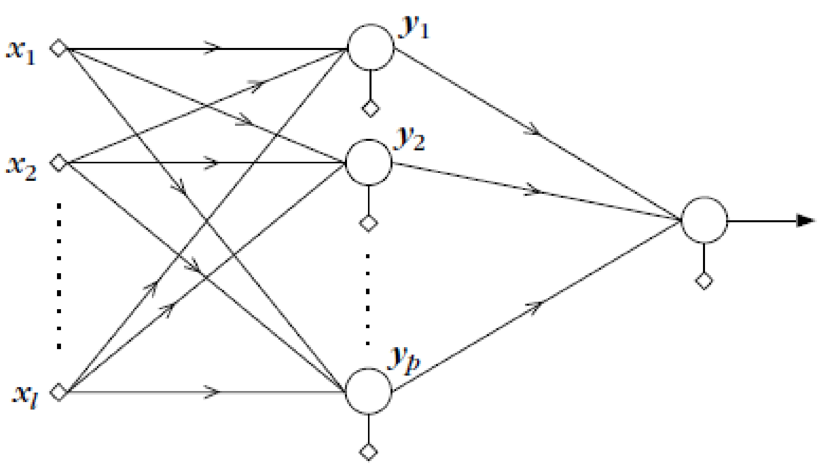


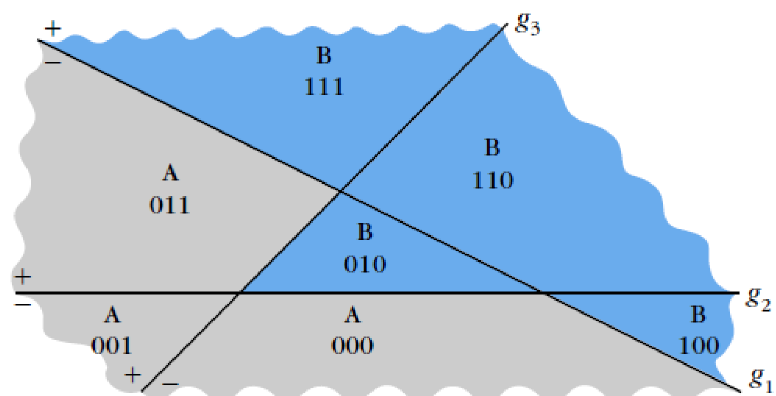


Two-Layer Perceptron

- Classification capabilities of two-layer perceptron
  - 2^{nd} layer detects a union of selected polyhedra



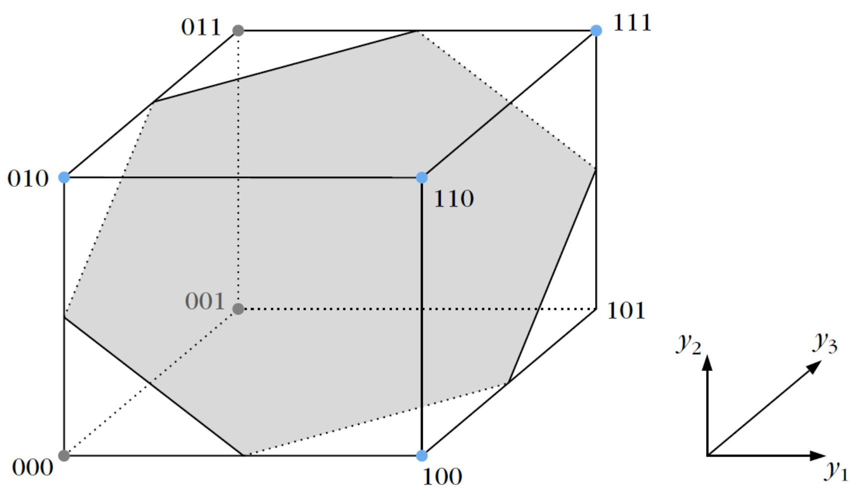




Two-Layer Perceptron

- Classification capabilities of two-layer perceptron

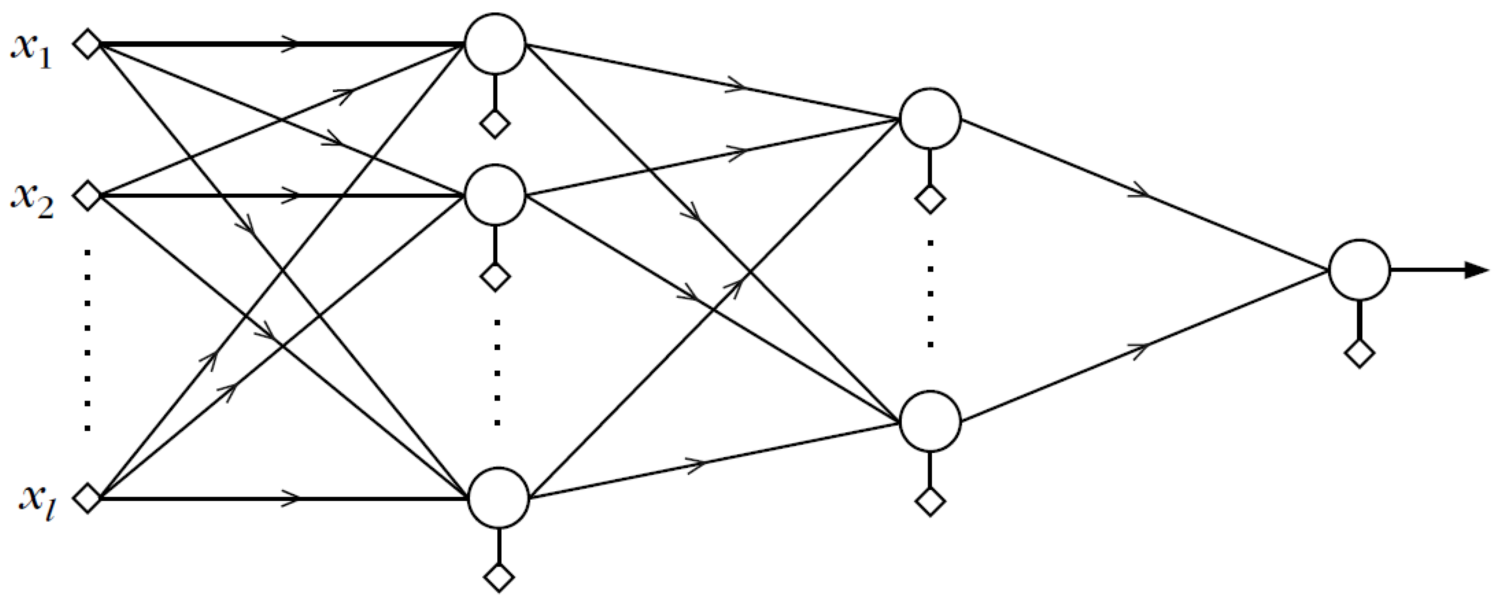
  Two-layer perceptron can detect a class, which consists of a union of polyhedral regions, but not any union of such regions



Three-Layer Perceptron

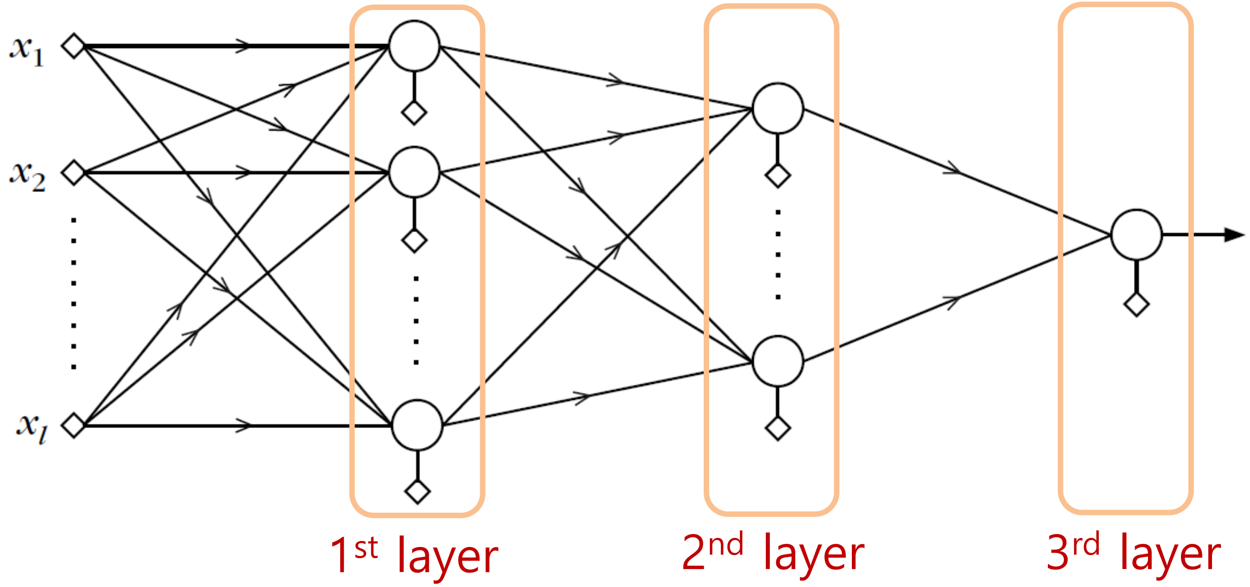
- Classification capabilities of three-layer perceptron

  Three-layer perceptron can detect a class, which consists of **any** union of polyhedral regions



Three-Layer Perceptron

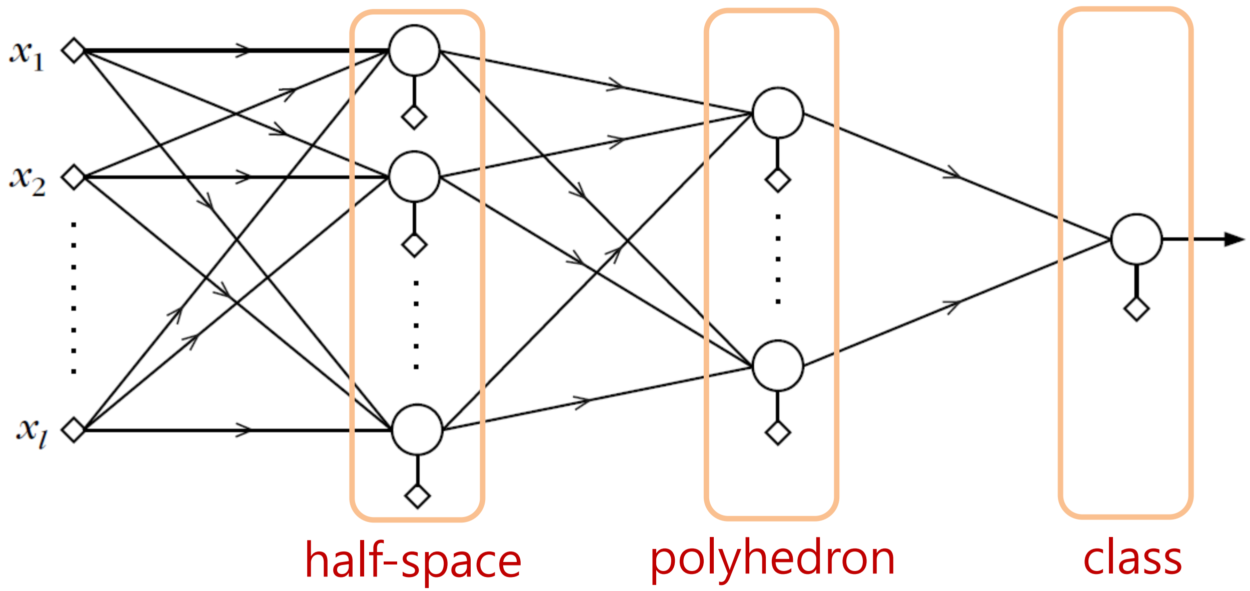
- Classification capabilities of three-layer perceptron



  - In 2nd layer, for each neuron, the synaptic weights are chosen so that the realized hyperplane leaves only one of the H_p vertices on one side and all the rest on the other
  - 3rd layer implements OR gate

Three-Layer Perceptron

- Classification capabilities of three-layer perceptron

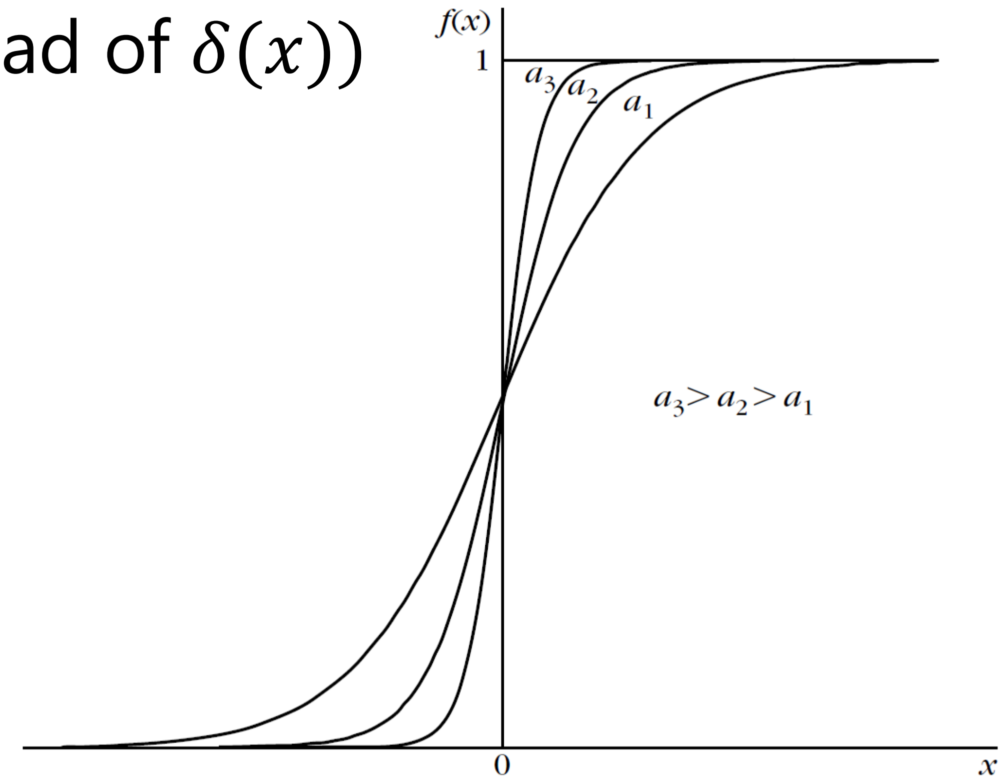


  - 1st layer detects half-spaces
  - 2nd layer detects polyhedra
  - 3rd layer detects a class, which is any union of polyhedra

BACKPROPAGATION ALGORITHM

Multilayer Perceptron Design

- Design a multilayer perceptron
  - Fix an architecture, and optimize the synaptic weights
  - To use the gradient descent scheme, we need a continuous activation function
- Logistic function (instead of $\delta(x)$)
  - $f(x) = \frac{1}{1+\exp(-ax)}$



Architecture and Formulation

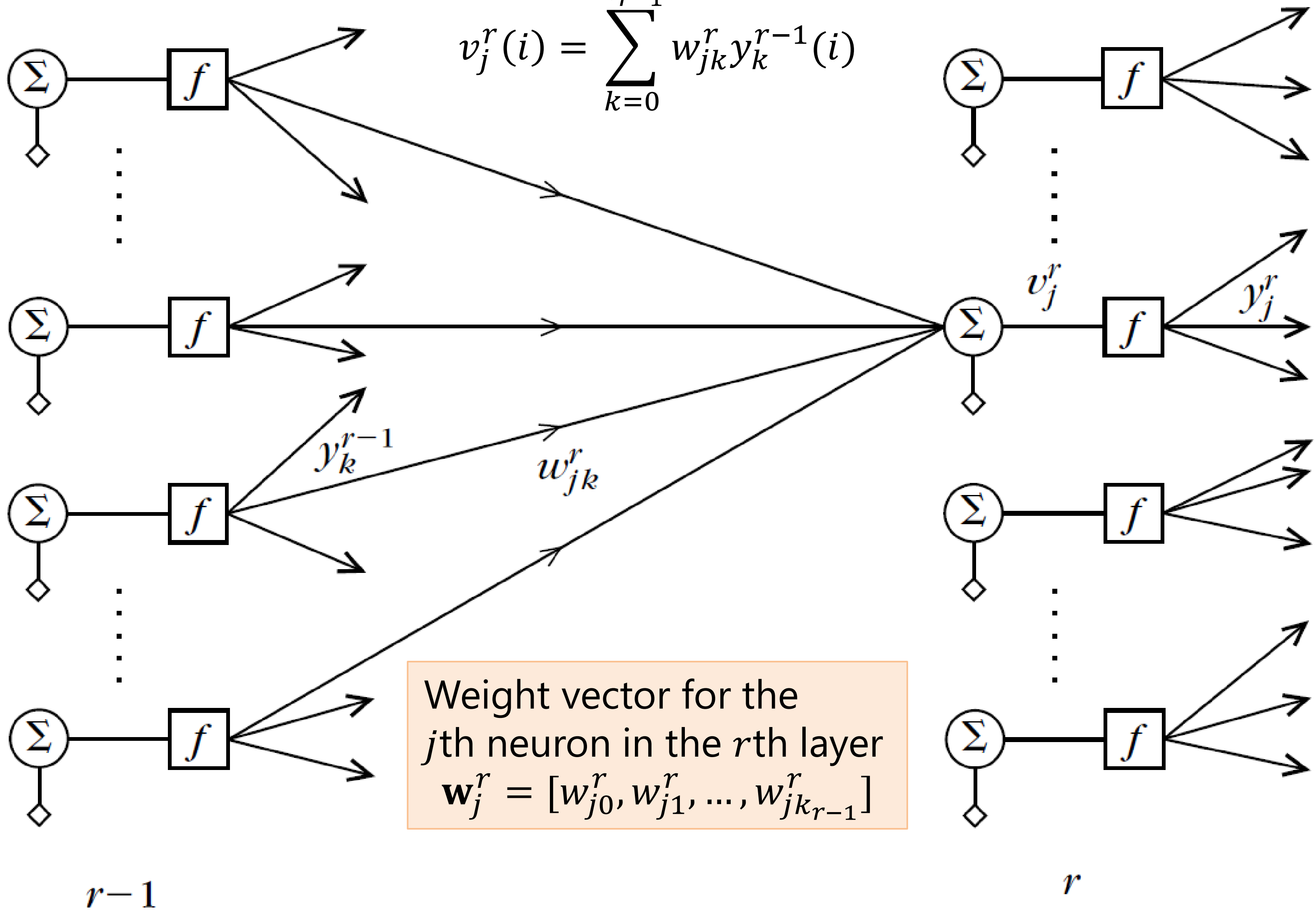
- L layers and k_r neurons in the rth layer ($r = 1, \dots, L$)
  - $k_0 = l$ nodes in the input layer
  - k_L output neurons
- N training pairs, $(\mathbf{y}(i), \mathbf{x}(i))$, $i = 1, \dots, N$, are available
  - $\mathbf{y}(i) = \left[y_1(i), \dots, y_{k_L}(i)\right]^T$
  - $\mathbf{x}(i) = \left[x_1(i), \dots, x_{k_0}(i)\right]^T$
- During training, the actual output $\hat{\mathbf{y}}(i)$ is different from the desired one $\mathbf{y}(i)$
- Compute the synaptic weights to minimize

$$J = \sum_{i=1}^{N} \mathcal{E}(i)$$

$$\mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2} \sum_{m=1}^{k_L} \left(\hat{y}_m(i) - y_m(i)\right)^2$$

Definition of Variables

$$v_j^r(i) = \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i)$$



Weight vector for the
jth neuron in the rth layer
$\mathbf{w}_j^r = [w_{j0}^r, w_{j1}^r, \dots, w_{jk_{r-1}}^r]$

Gradient Descent

$$\mathbf{w}_j^r(\text{new}) = \mathbf{w}_j^r(\text{old}) + \Delta\mathbf{w}_j^r$$

$$\Delta\mathbf{w}_j^r = -\mu \frac{\partial J}{\partial \mathbf{w}_j^r}$$

- Details for subsequent steps are omitted