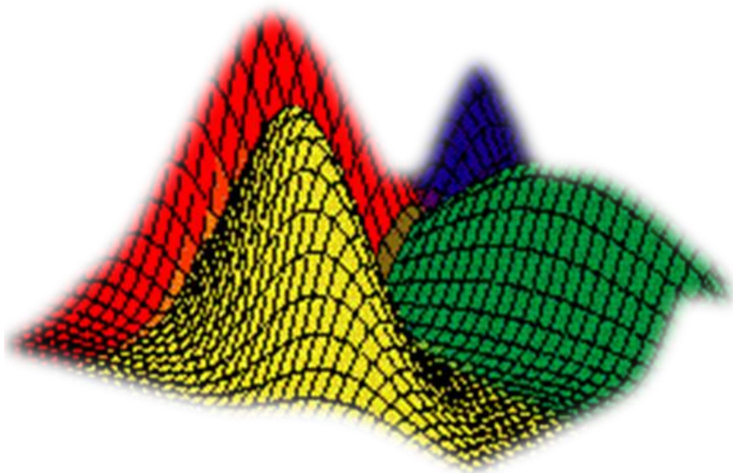


CSE 473
Pattern Recognition

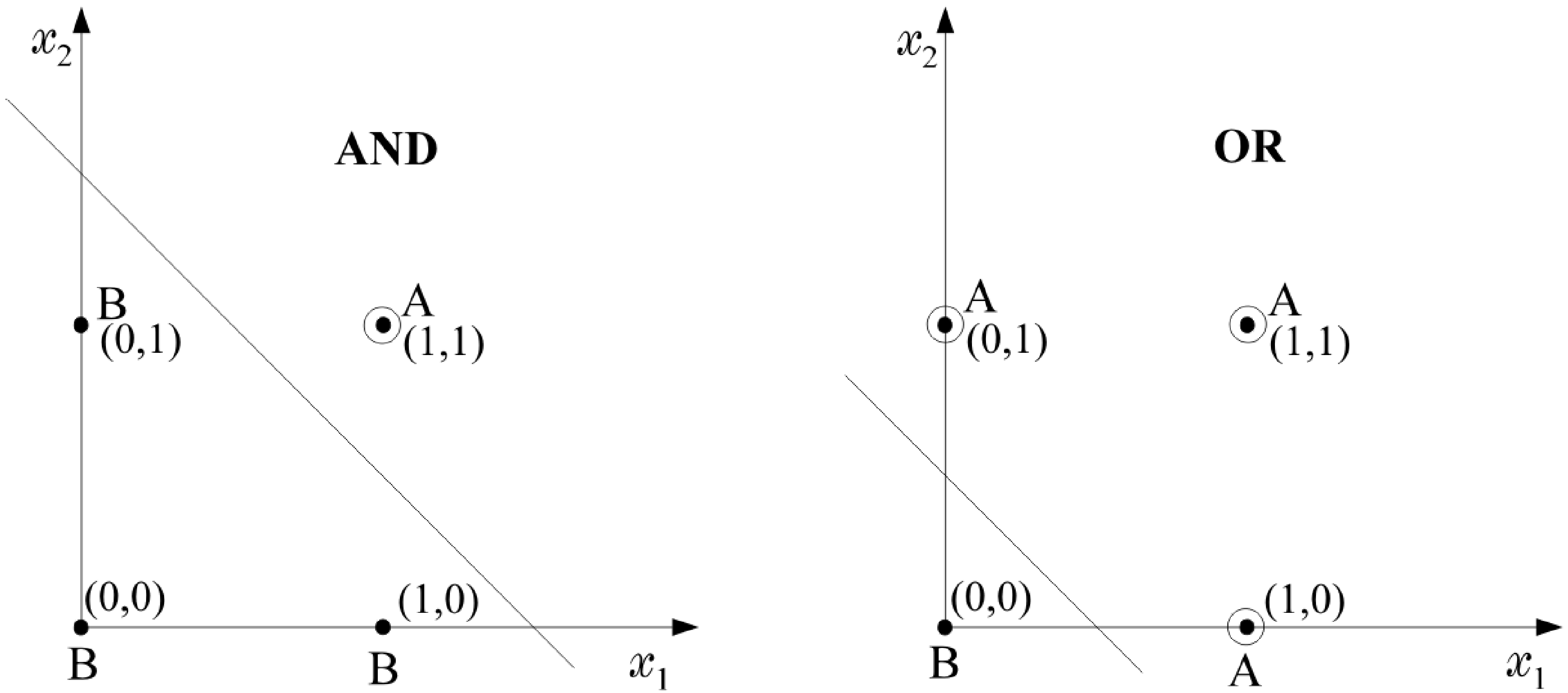
Non-Linear Classifier

Review of Perceptron's Capability

Recall the AND or OR functions

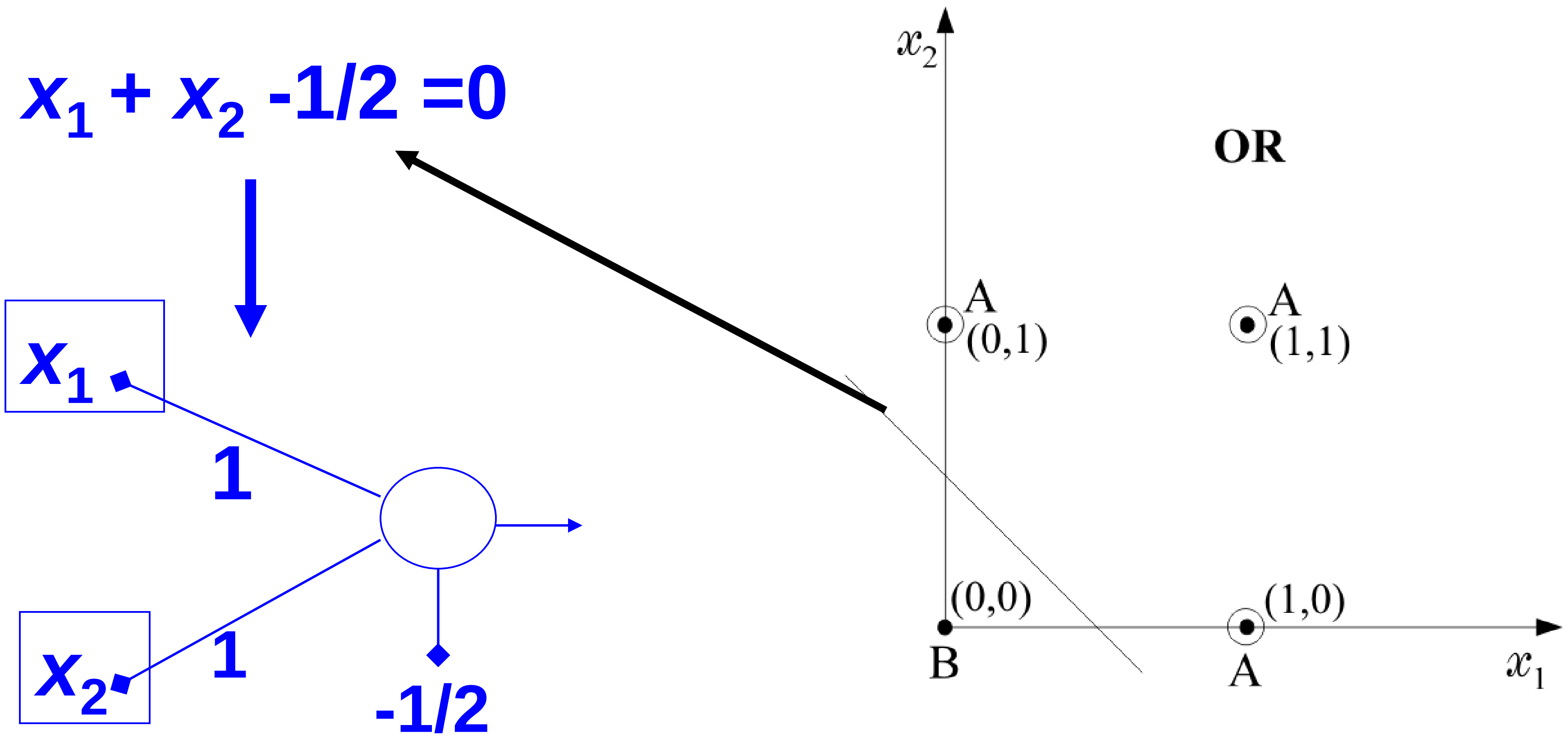
x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A

Review of Perceptron's Capability

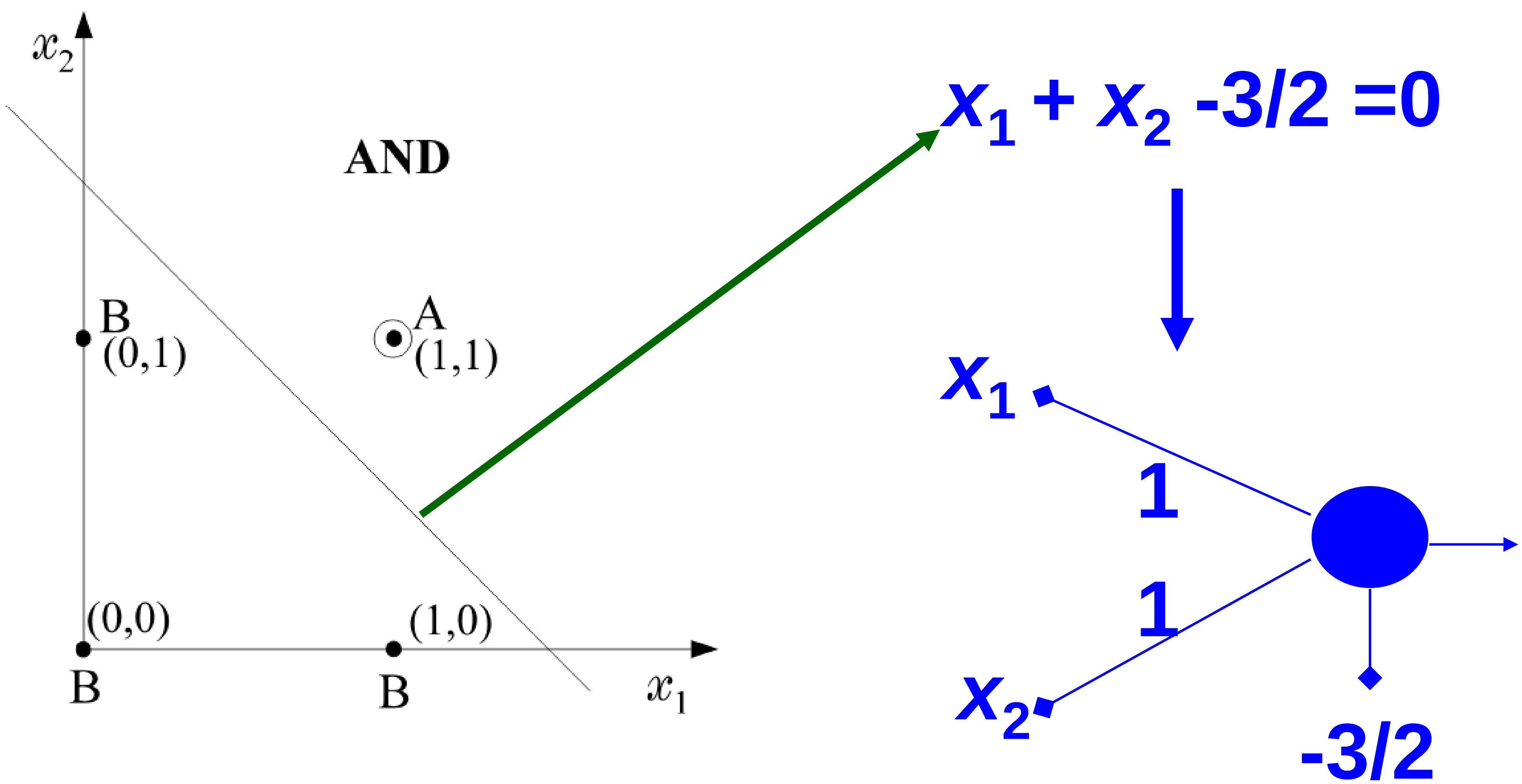


Review of Perceptron’s Capability

$$x_1 + x_2 - 1/2 = 0$$



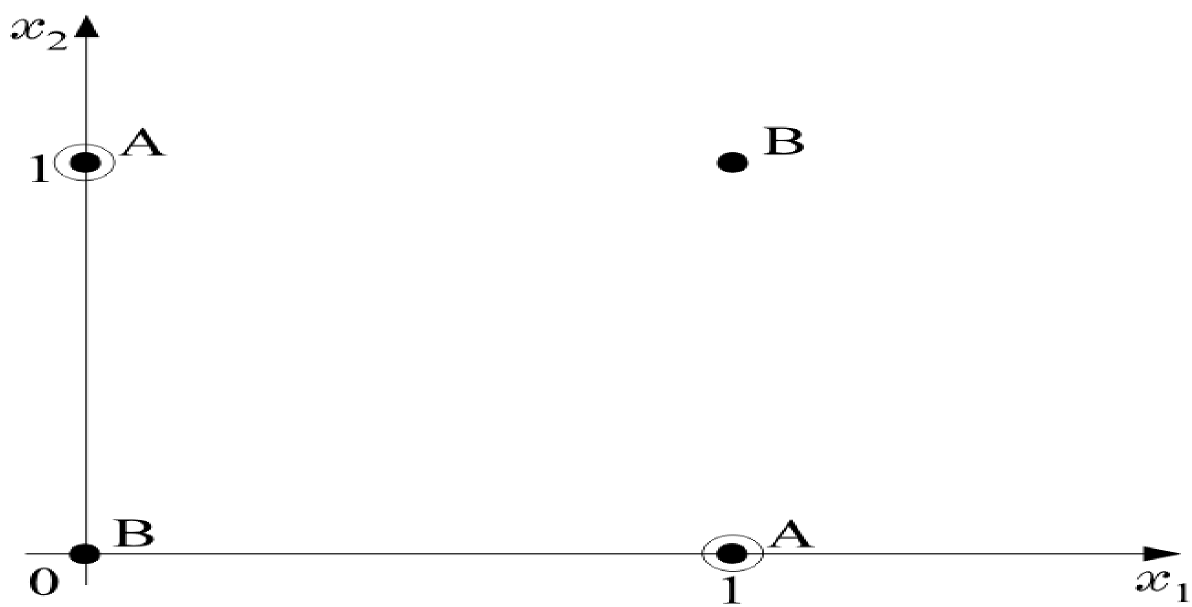
Review of Perceptron's Capability



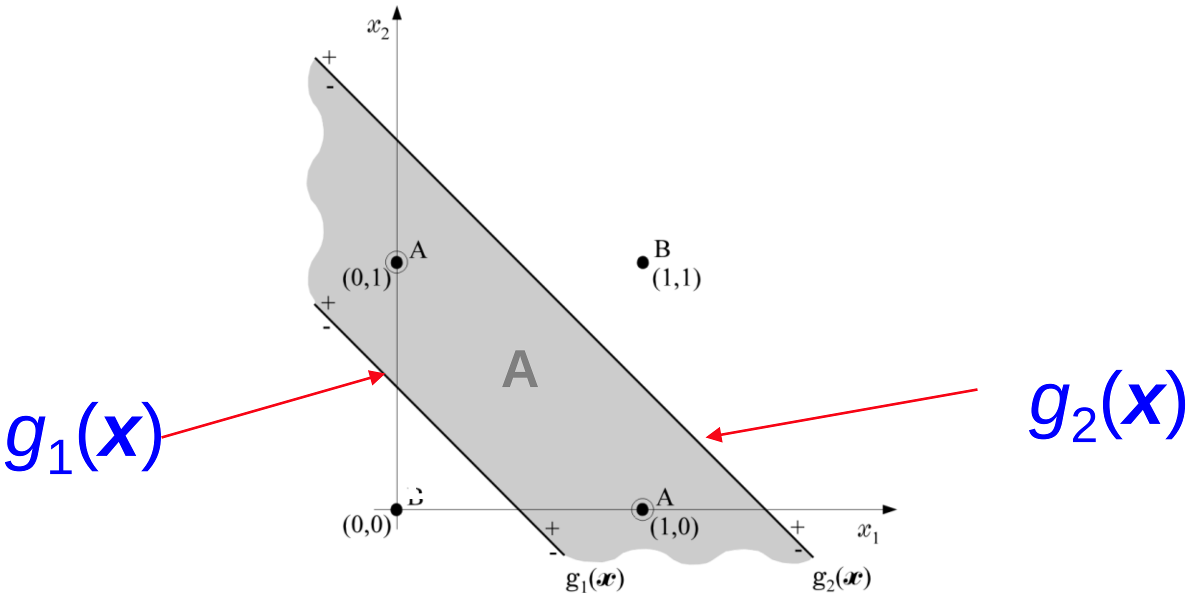
Review of Perceptron's Capability

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

Now recall the XOR function



Review of Perceptron's Capability



- Each of them is realized by a perceptron.

$$y_i = f(g_i(\underline{x})) = \begin{cases} 0 \\ 1 \end{cases} \quad i = 1, 2$$

- Find the position of $\underline{x}$ *w.r.t.* both lines, based on the values of y_1, y_2.

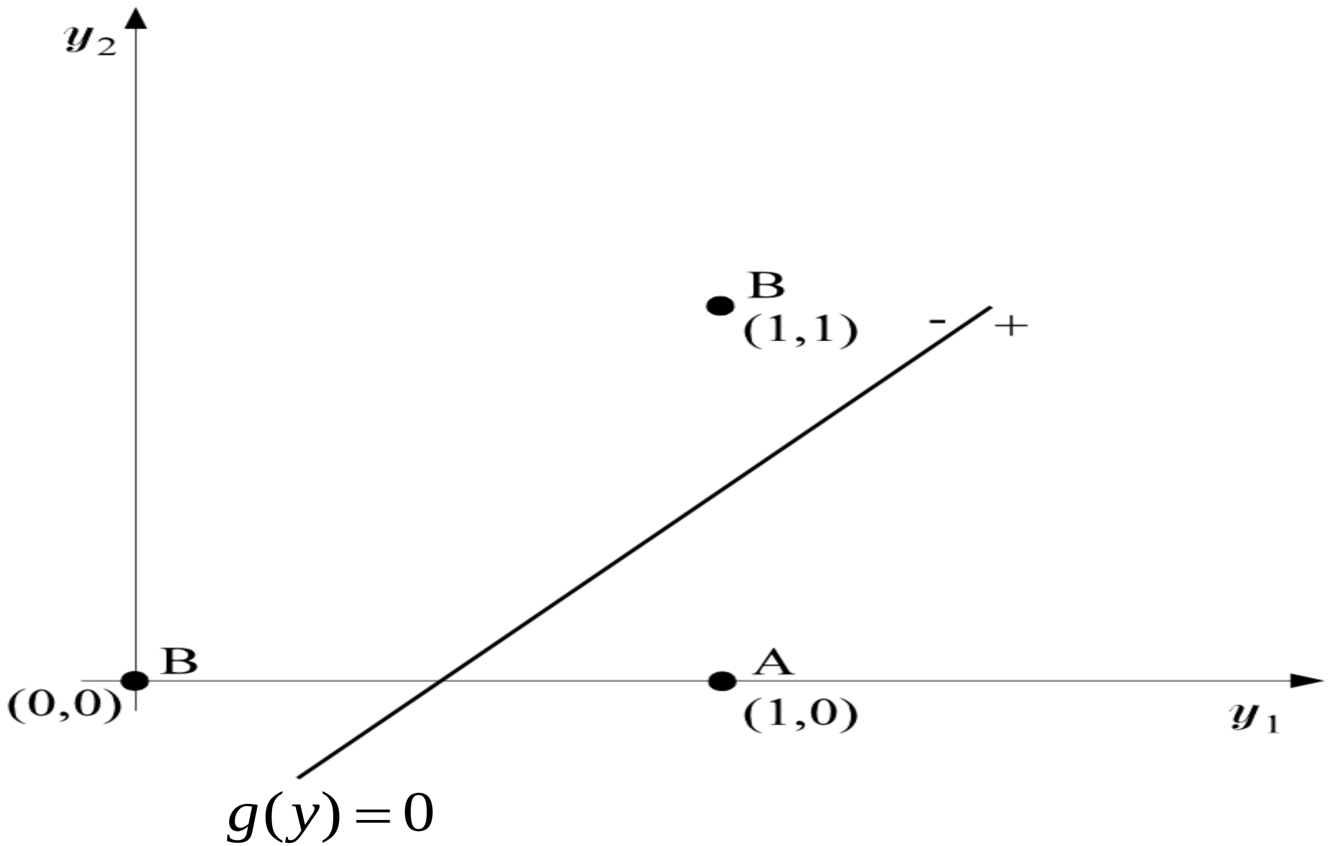
Review of Perceptron's Capability

1st phase				2nd phase
x_1	x_2	y_1	y_2	
0	0	0(-)	0(-)	B(0)
0	1	1(+)	0(-)	A(1)
1	0	1(+)	0(-)	A(1)
1	1	1(+)	1(+)	B(0)

- Equivalently: The computations of the first phase perform a mapping $\underline{x} \rightarrow \underline{y} = [y_1, y_2]^T$

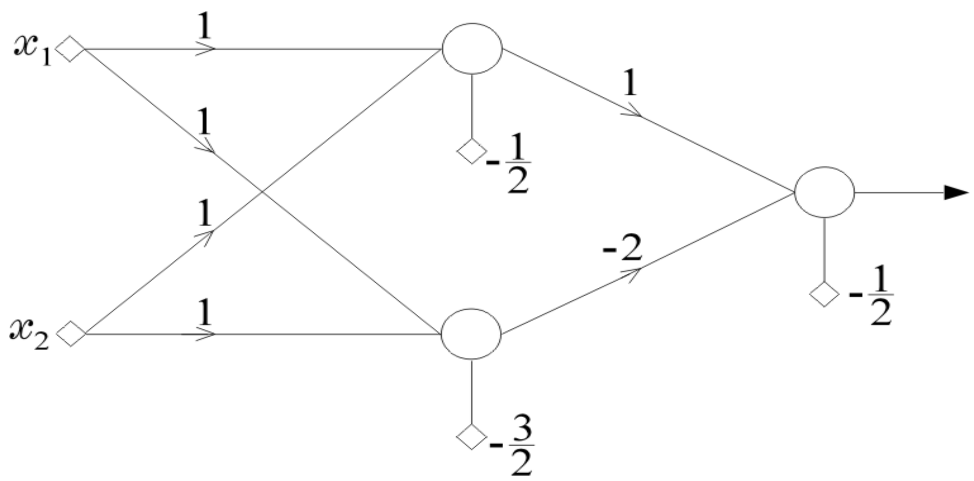
Review of Perceptron's Capability

The decision is now performed on the transformed $\underline{y}$ data.



This can be performed via a second line, which can also be realized by a perceptron.

Two Layer Perceptron



hidden layer

output layer

nodes realizes hyper planes:

$$g_1(\underline{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\underline{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

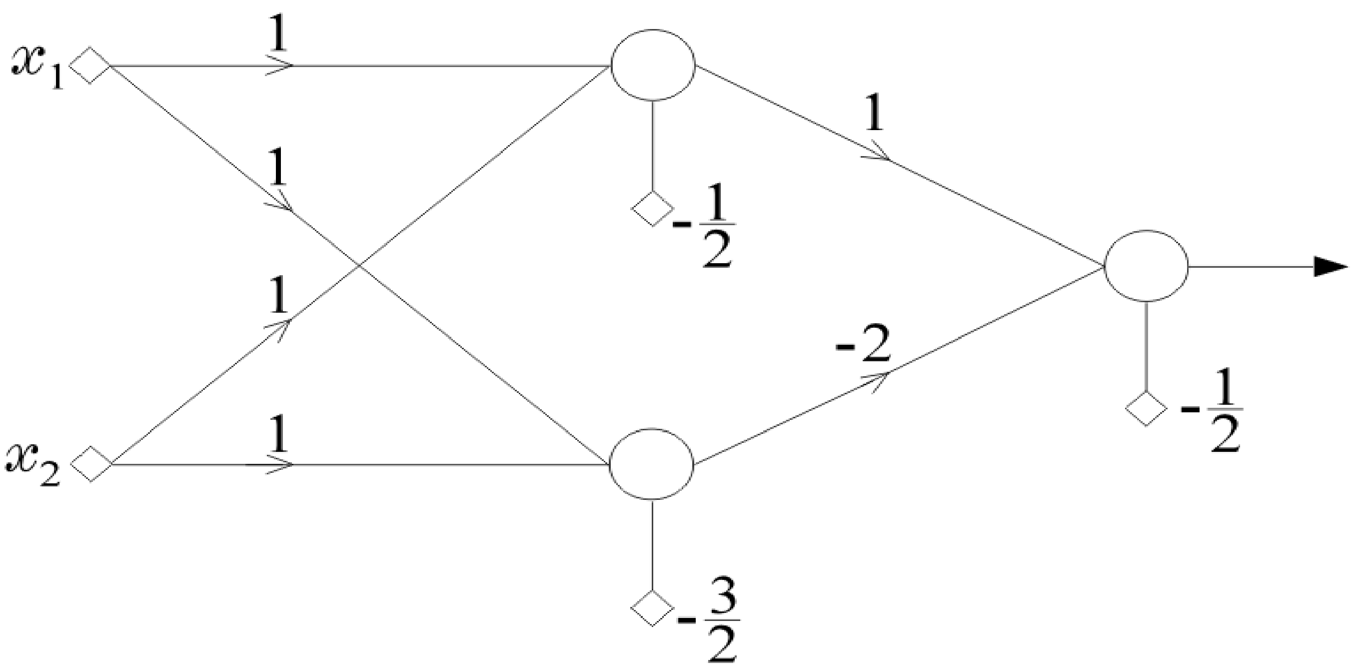
$$g(\underline{y}) = y_1 - 2y_2 - \frac{1}{2} = 0$$

Activation function:

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

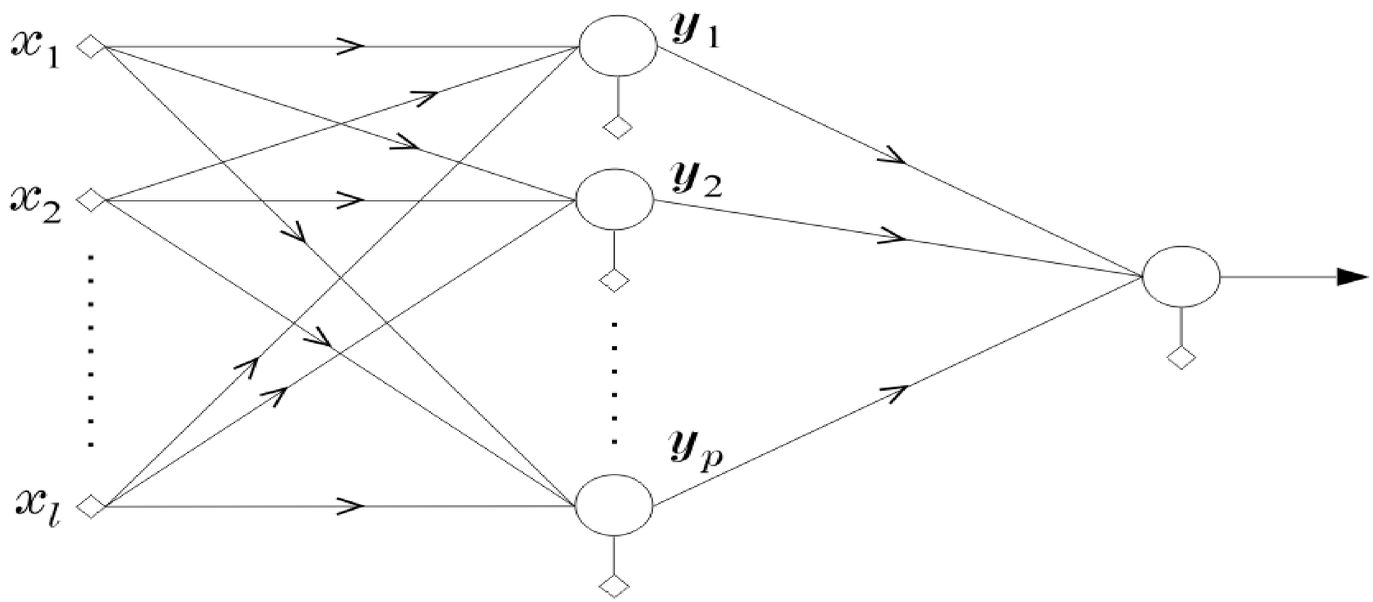
Classification Capabilities of Two Layer Perceptron

- The mapping performed by the first layer neurons is onto the vertices of the unit side square, e.g., (0, 0), (0, 1), (1, 0), (1, 1).



Classification Capabilities of Two Layer Perceptron

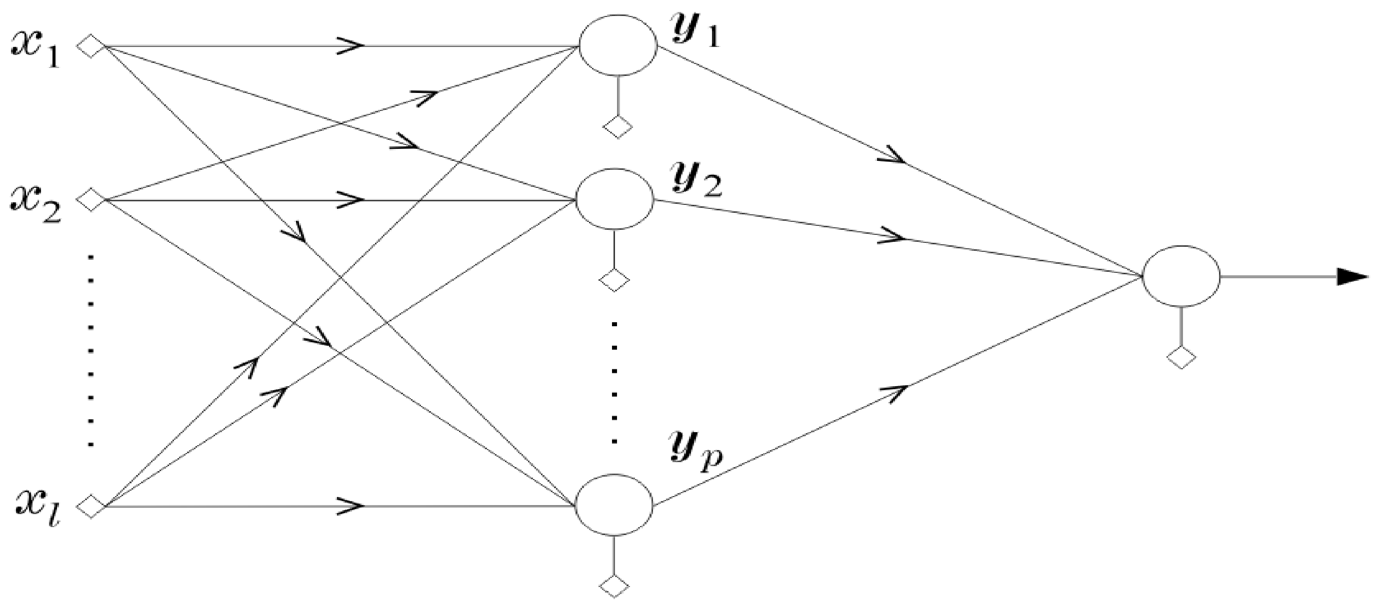
- Consider a more general case,



$$\underline{x} \in R^l$$

$$\underline{x} \rightarrow \underline{y} = [y_1, \ldots y_p]^T,\ y_i \in \{0,1\}\ \ i = 1, 2, \ldots p$$

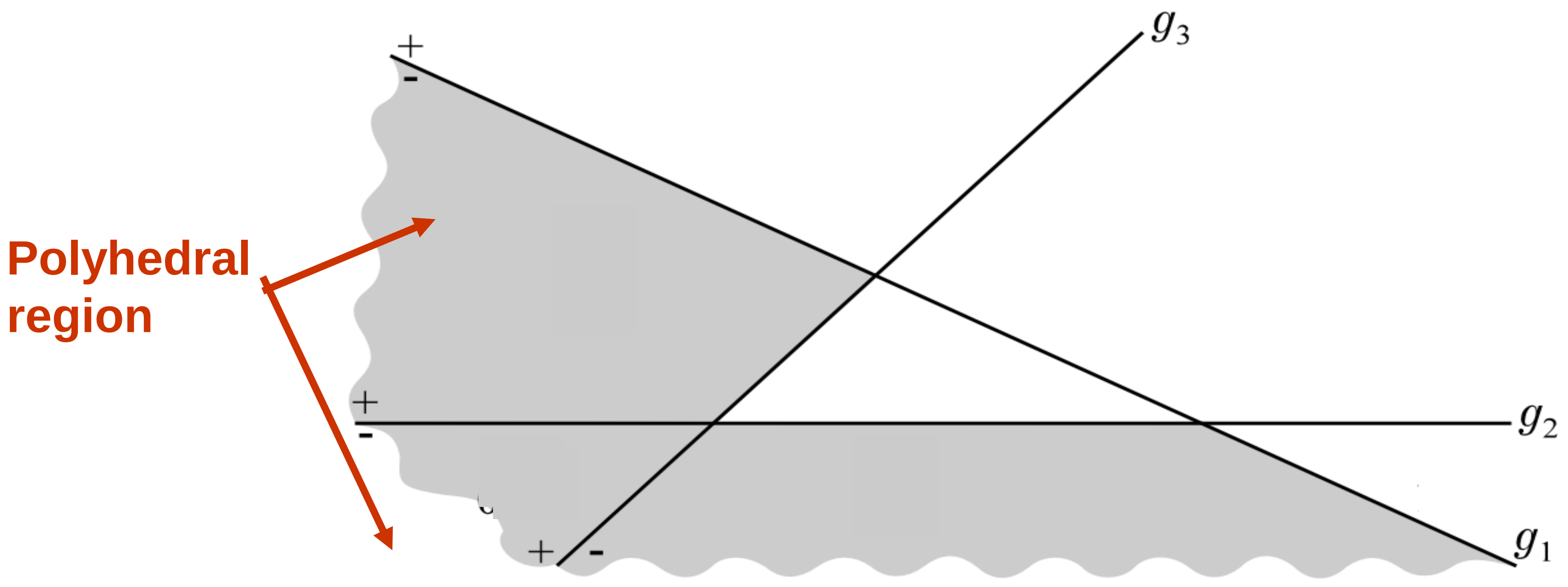
Classification Capabilities of Two Layer Perceptron



- maps a vector onto the vertices of the unit side hypercube, *Hp*
- mapping is through *p* neurons each realizing a hyper plane.
- The output of each of these neurons is 0 or 1

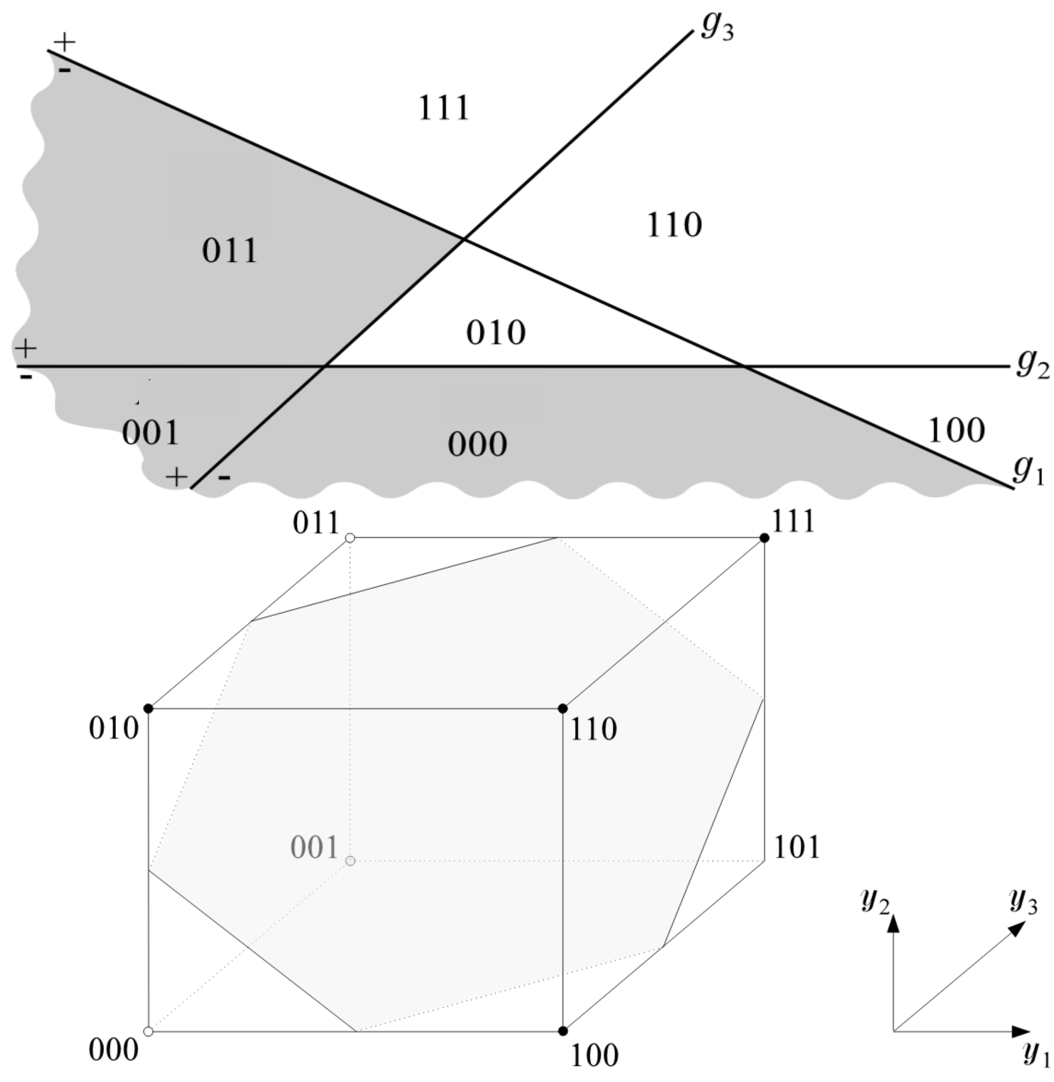
Classification Capabilities of Two Layer Perceptron

- Intersections of hyperplanes form regions.



Classification Capabilities of Two Layer Perceptron

- Intersections of hyperplanes form regions.
- Each region corresponds to a vertex of the H_p unit hypercube.



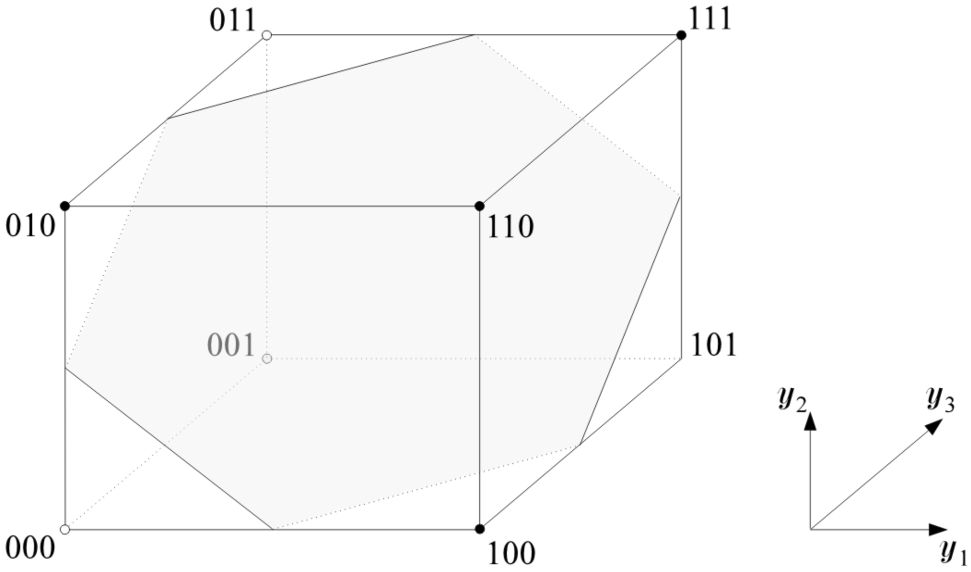
Classification Capabilities of Two Layer Perceptron

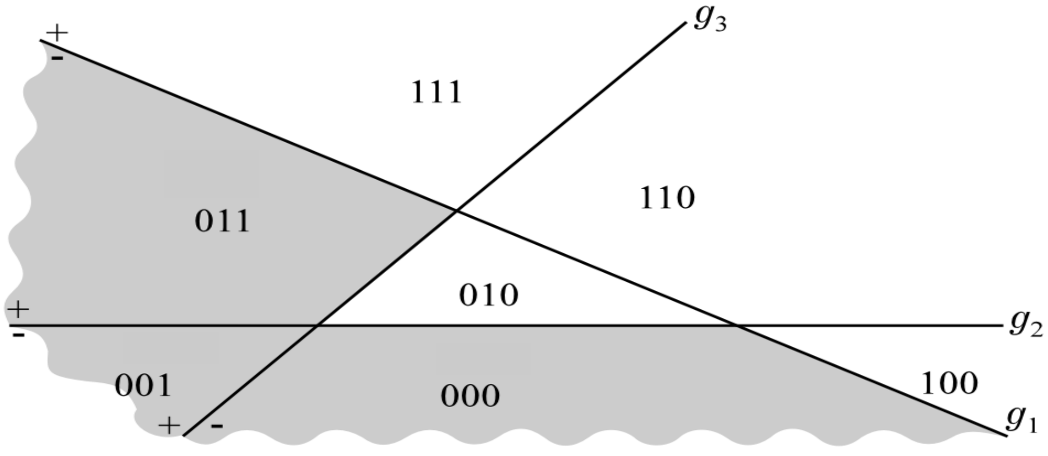
- For example, the 001 vertex corresponds to the region which is located

  to the (-) side of $g_1(\underline{x})=0$

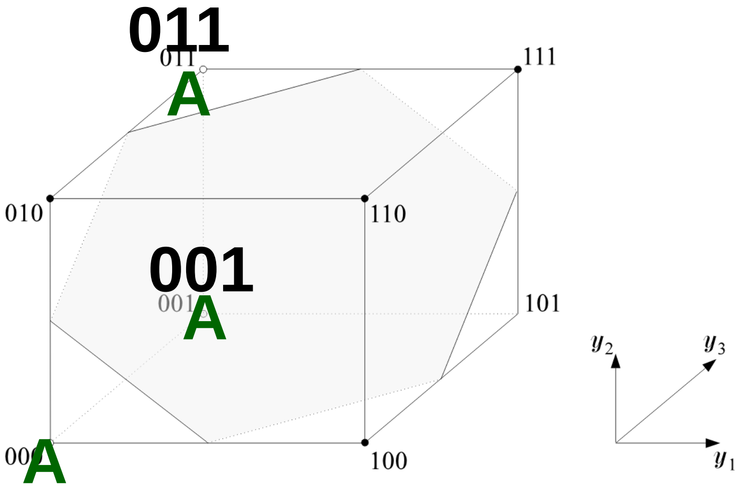
  to the (-) side of $g_2(\underline{x})=0$

  to the (+) side of $g_3(\underline{x})=0$

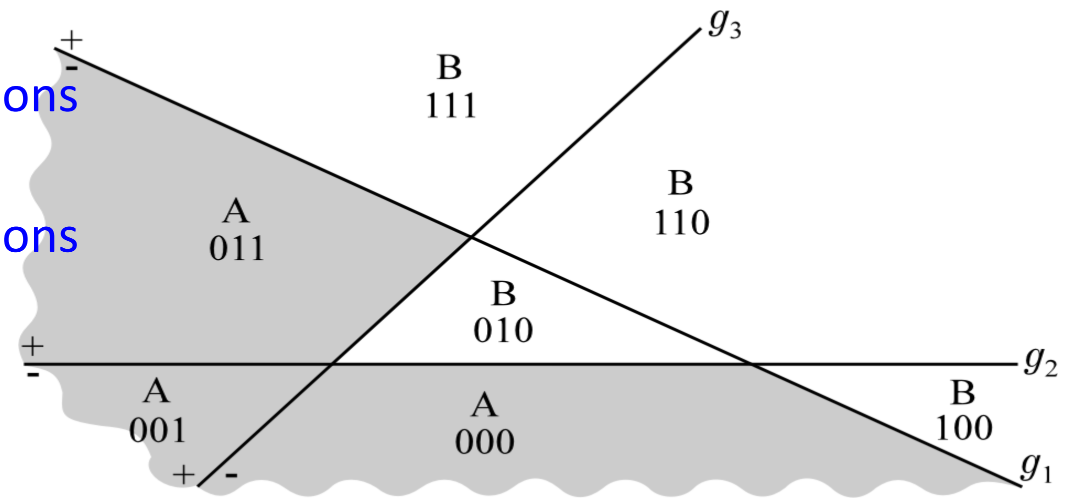




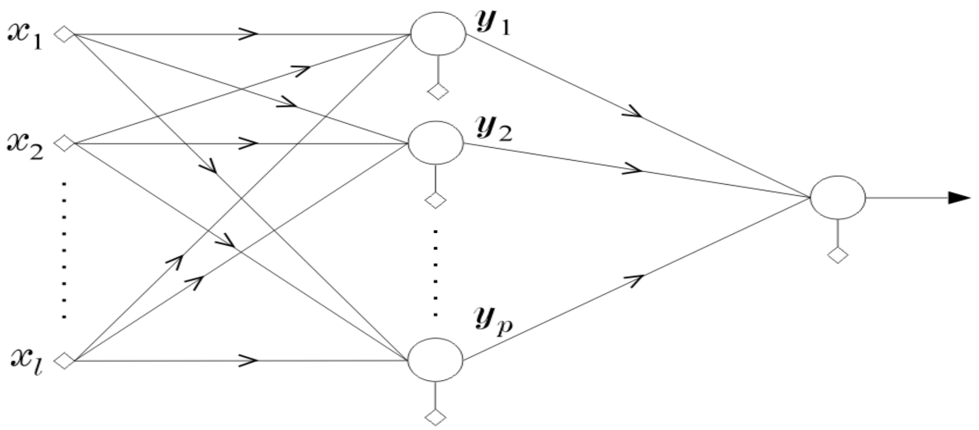
Classification Capabilities of Two Layer Perceptron

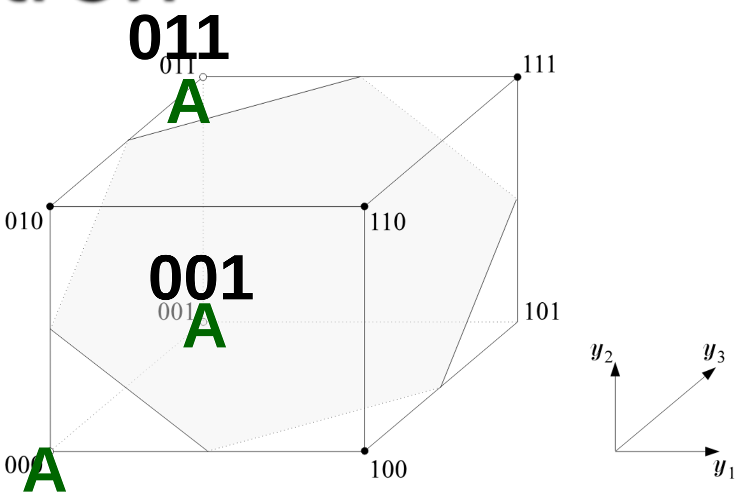


- A two-class problem
  - **Class A patterns** from regions marked as A
  - **Class B patterns** from regions marked as B

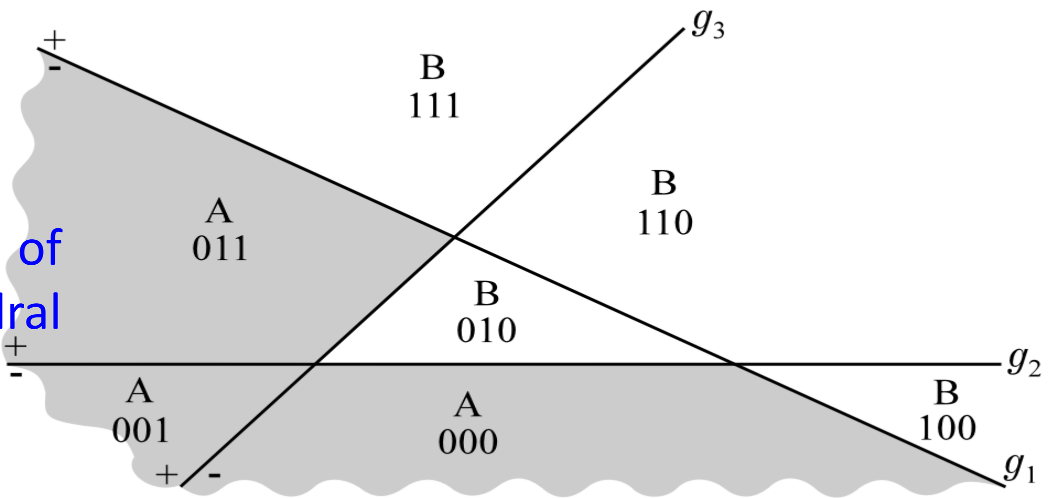


Classification Capabilities of Two Layer Perceptron

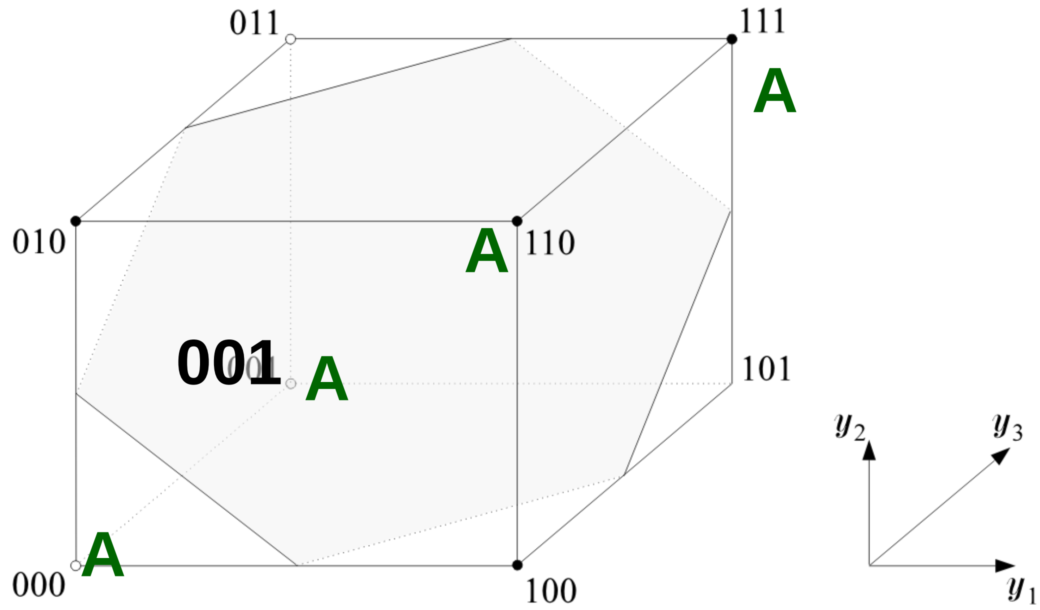




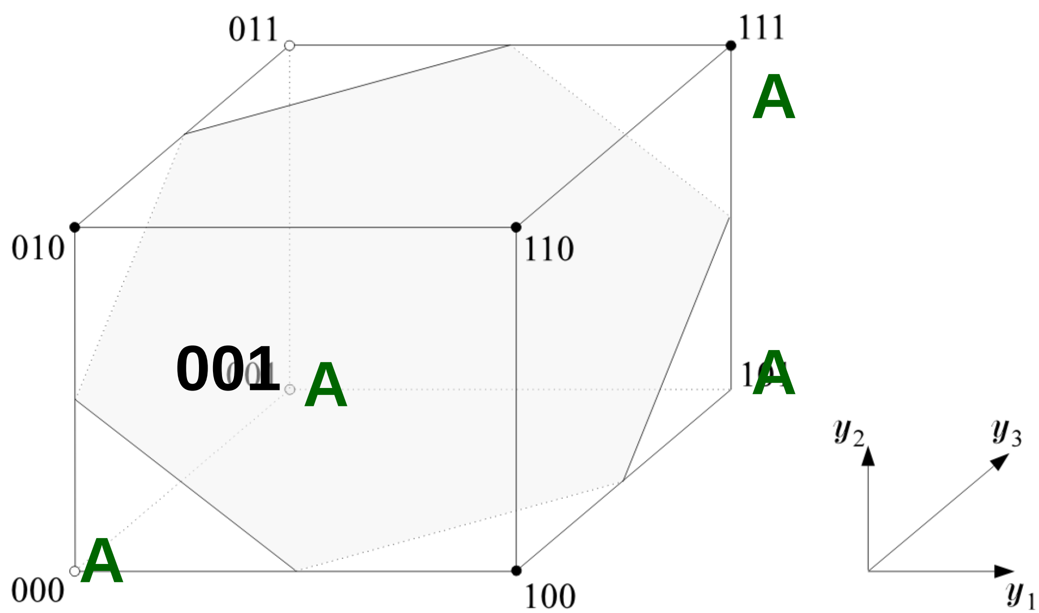
- The output neuron
  - realizes another hyperplane
  - separates the hypercube.
  - can classify vectors consisting of some unions of polyhedral regions.



Classification Capabilities of Two Layer Perceptron



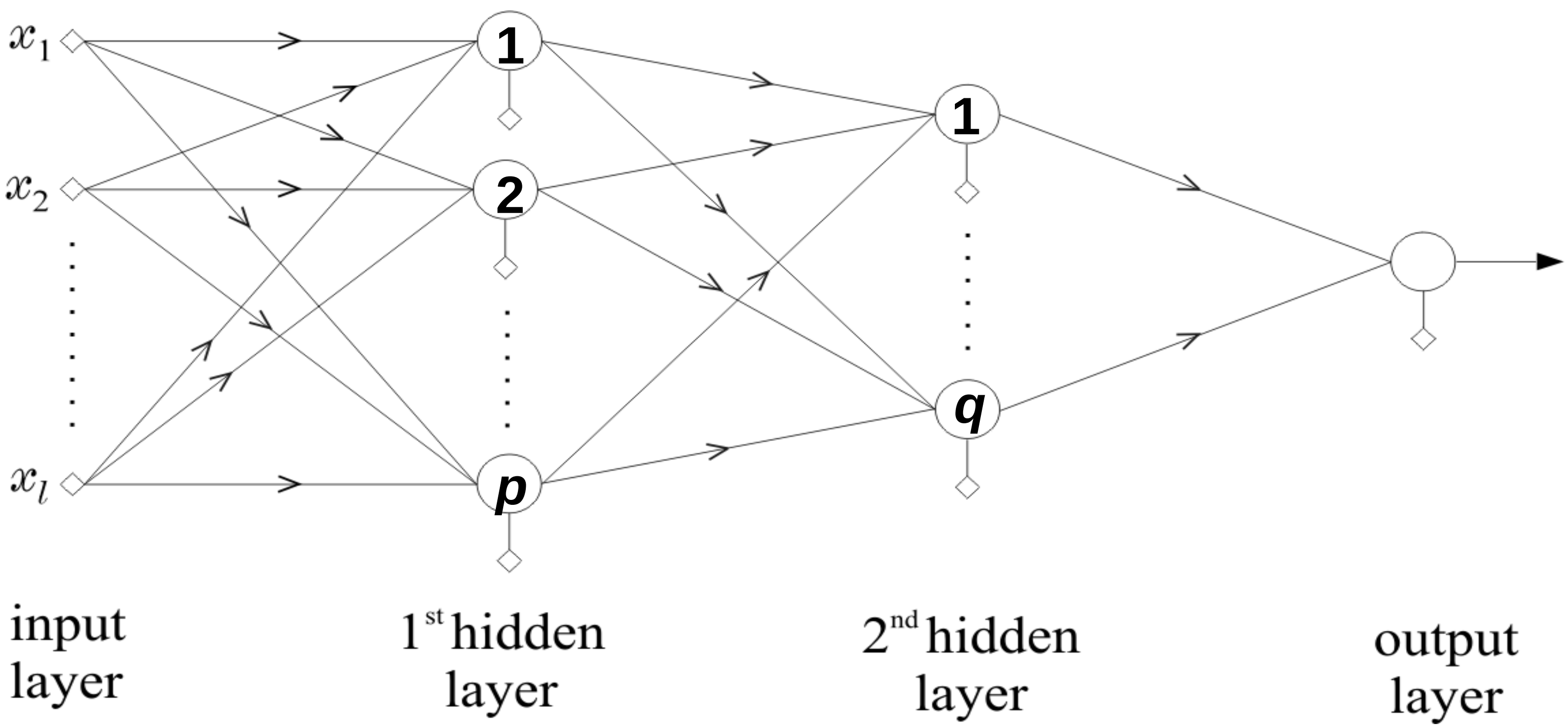
Classification Capabilities of Two Layer Perceptron



- The output neuron, *i.e.*, a 2 layer perceptron
  - cannot classify vectors consisting of arbitrary unions of polyhedral regions.

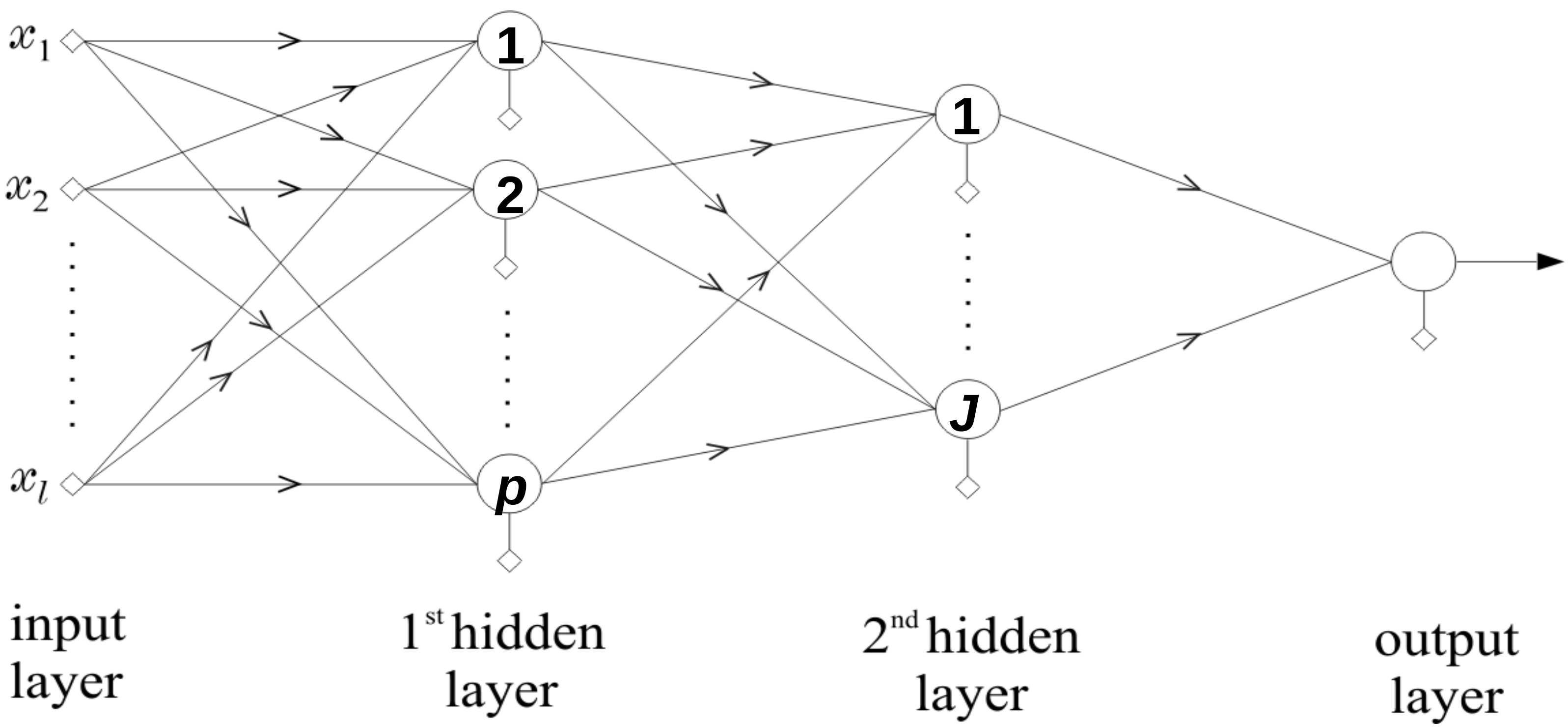
Solution: Three Layer Perceptron

- capable to classify vectors consisting of **ANY** union of polyhedral regions.
  - The idea is similar to the XOR problem.
  - Realizes more than one planes in the $\underline{y} \in R^p$ space.



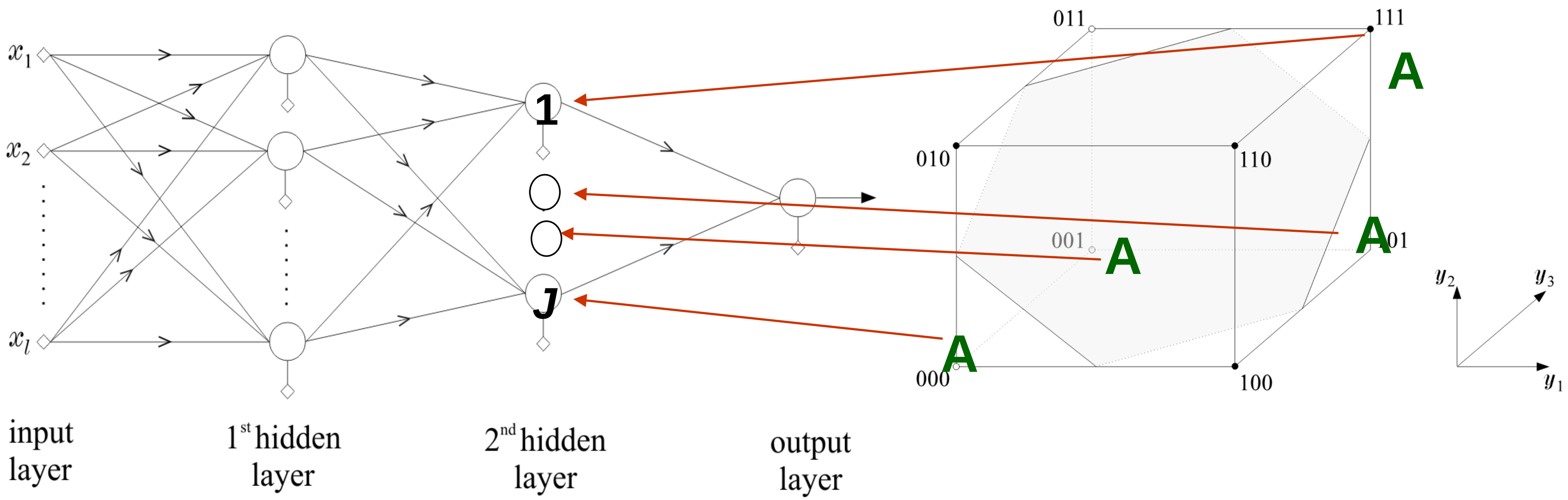
How does The Three Layer Perceptron Do It?

- Let, any *J* polyhedral regions constitutes vectors of class ***A***.
- Learn a neuron in the 2nd hidden layer for each of *J* regions



How does The Three Layer Perceptron Do It?

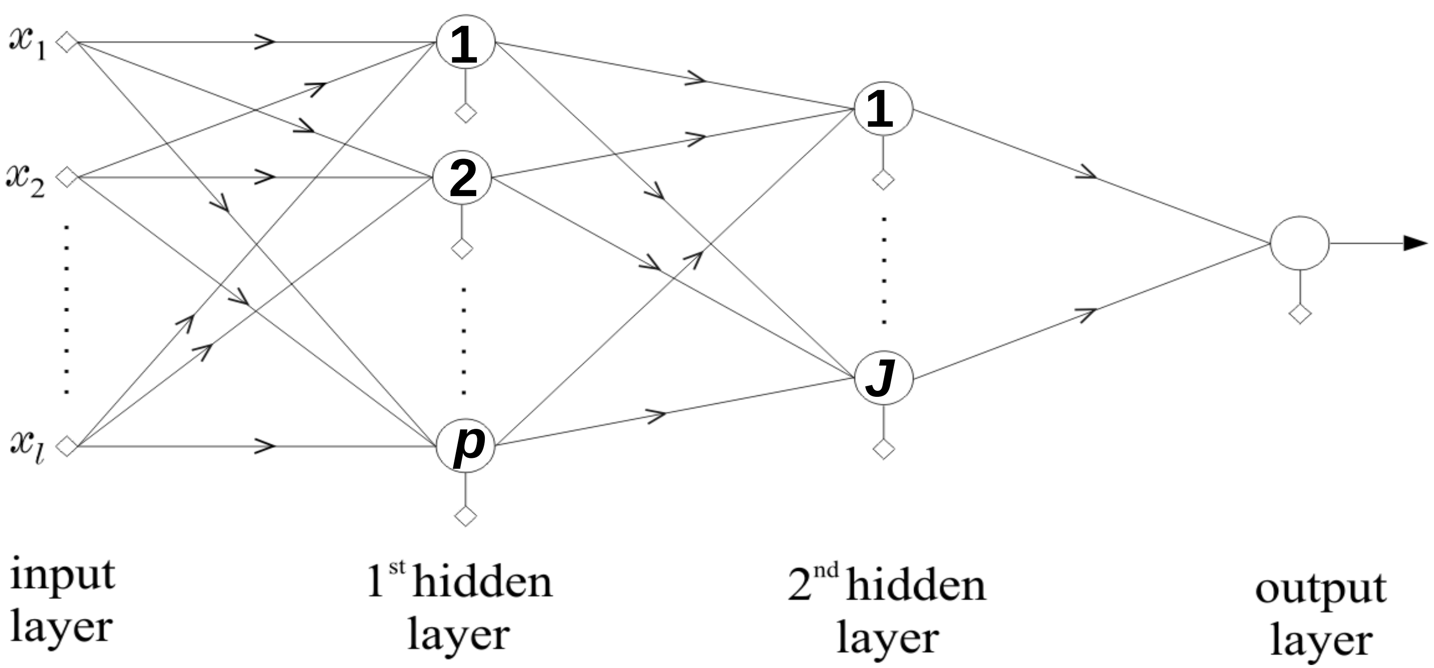
- Learn a neuron in the 2nd hidden layer for each of *J* regions



How does The Three Layer Perceptron Do It?

- For training vectors of a particular region of class ***A***, only one of the 2nd-layer neuron produces 1, the rest of neurons produce 0.

- Now realize the output neuron as an OR gate.

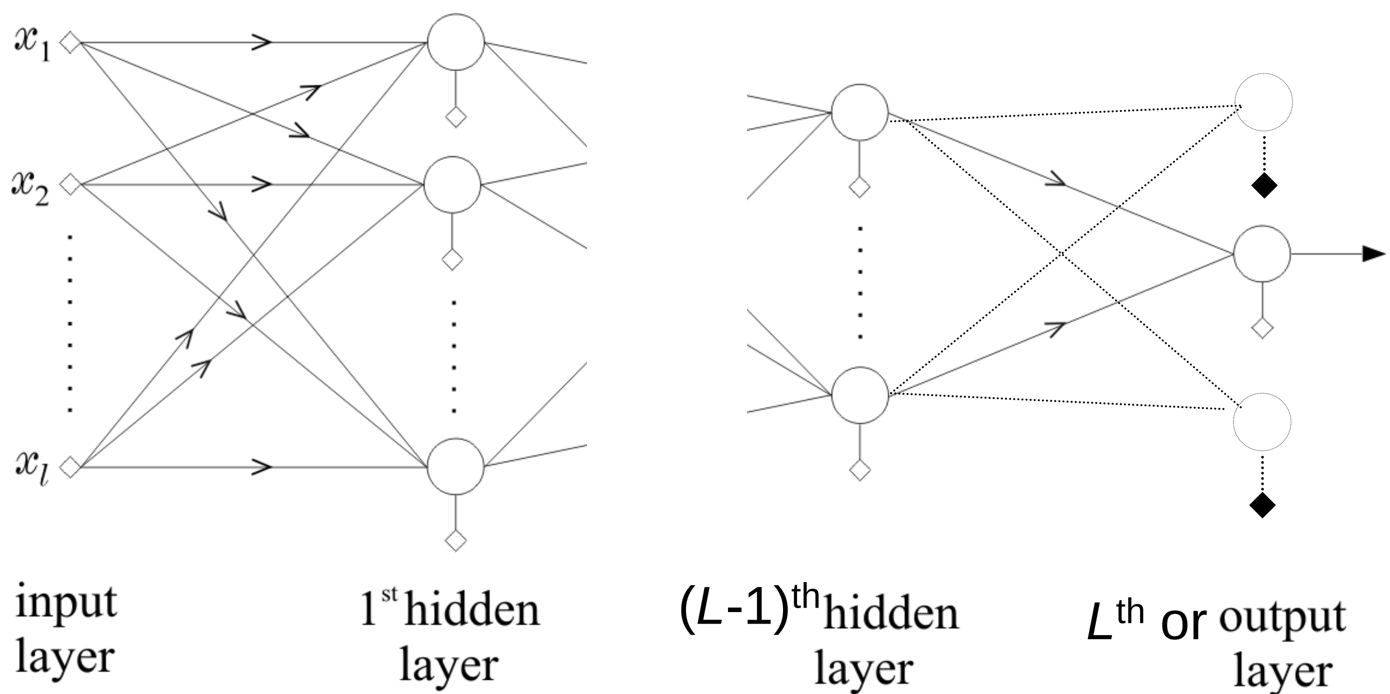


Training of a Multi Layer Perceptron (MLP)

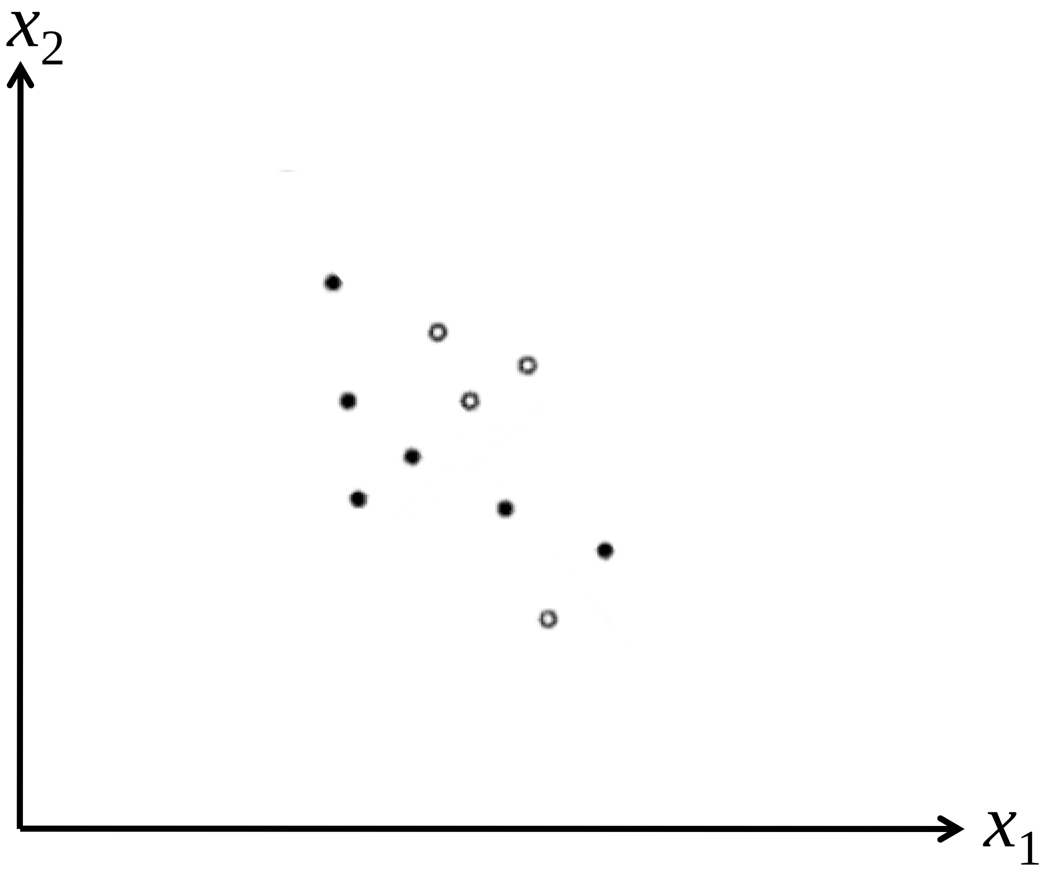
- use rationale and develop a structure that classifies **correctly all** the training patterns.

  OR

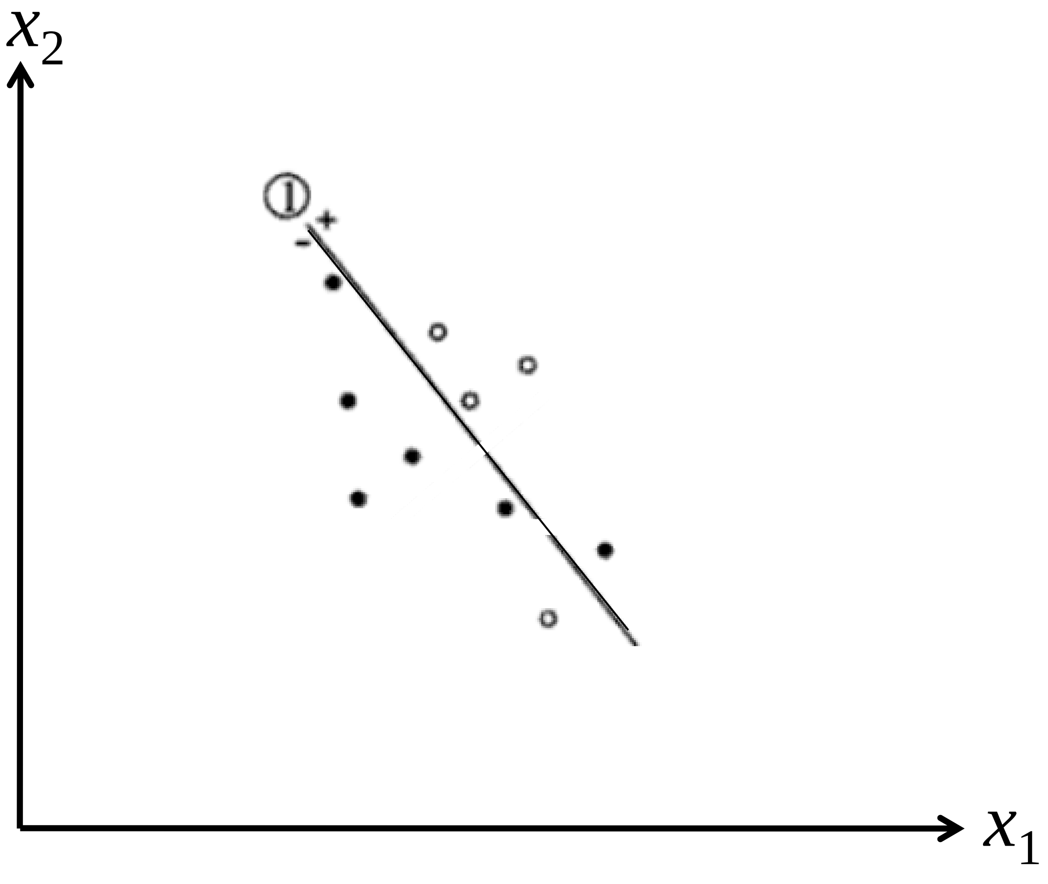
- choose a structure and compute the synaptic weights to optimize a cost function.

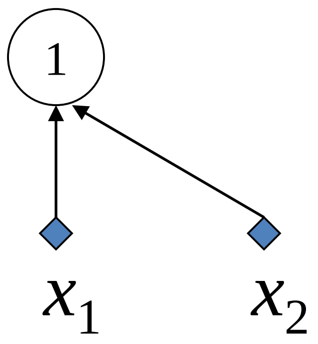


Algorithm Based on Exact Classification of Training Examples

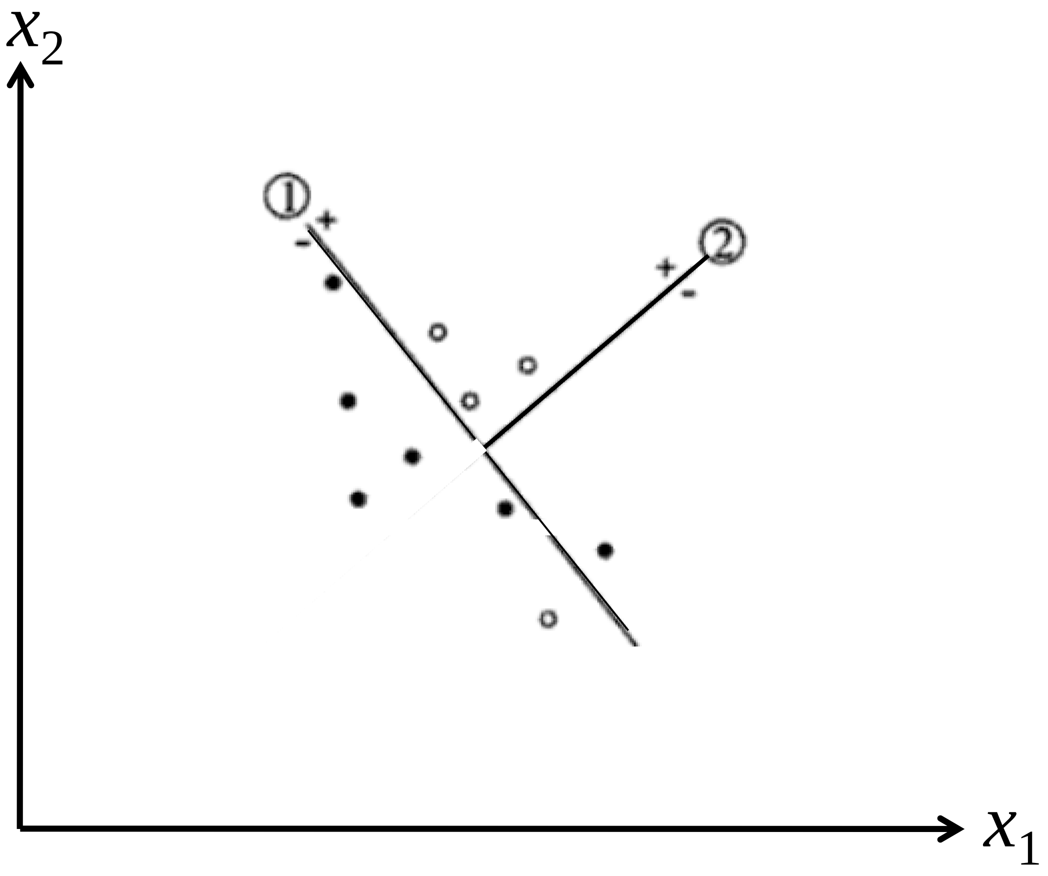


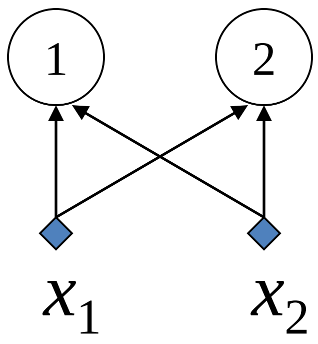
Algorithm Based on Exact Classification of Training Examples



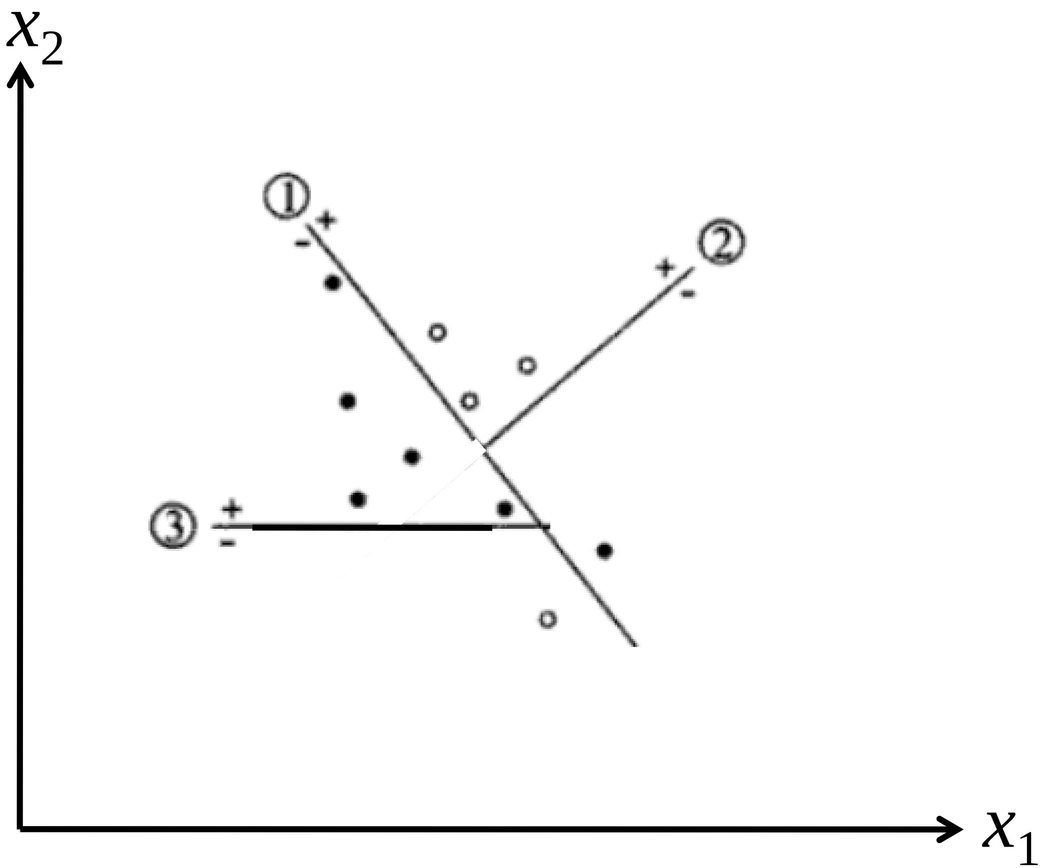


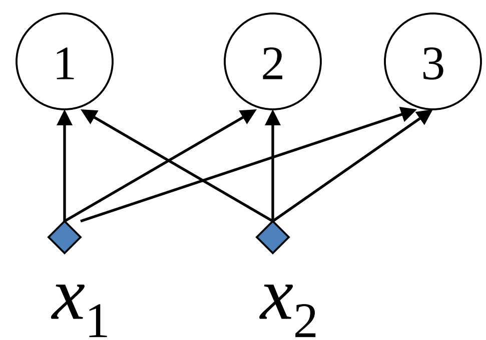
Algorithm Based on Exact Classification of Training Examples



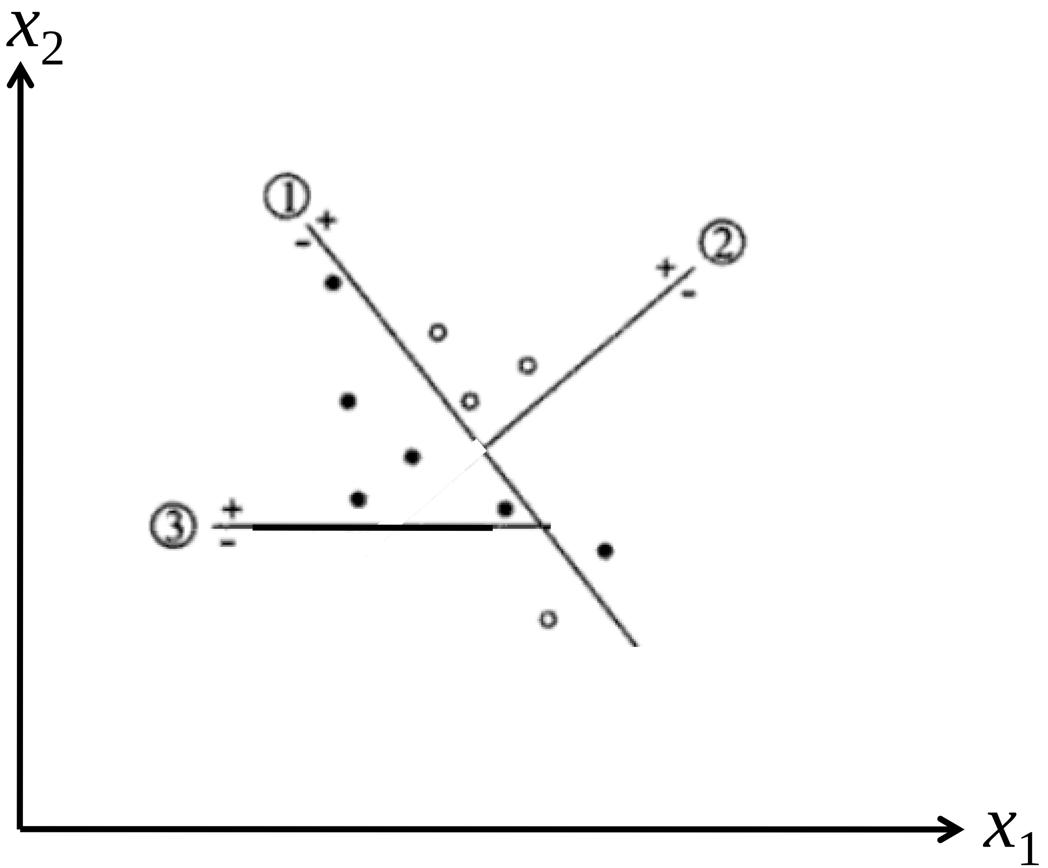


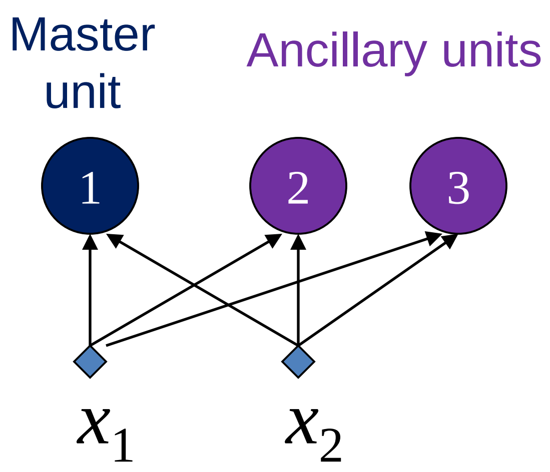
Algorithm Based on Exact Classification of Training Examples



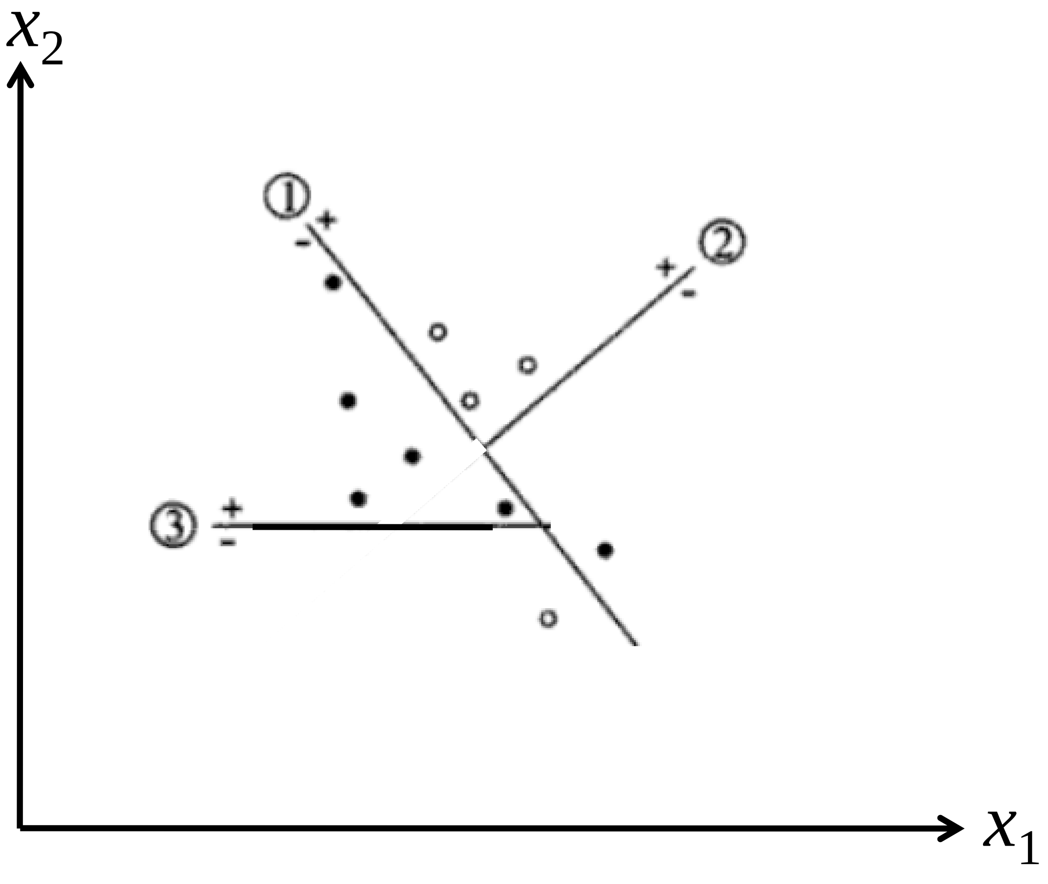


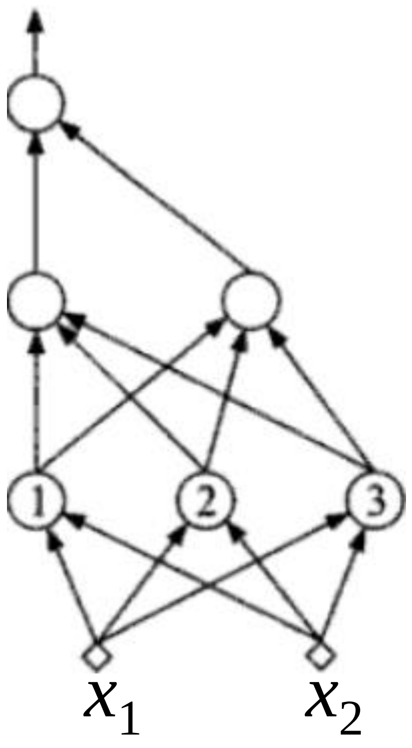
Algorithm Based on Exact Classification of Training Examples





Algorithm Based on Exact Classification of Training Examples

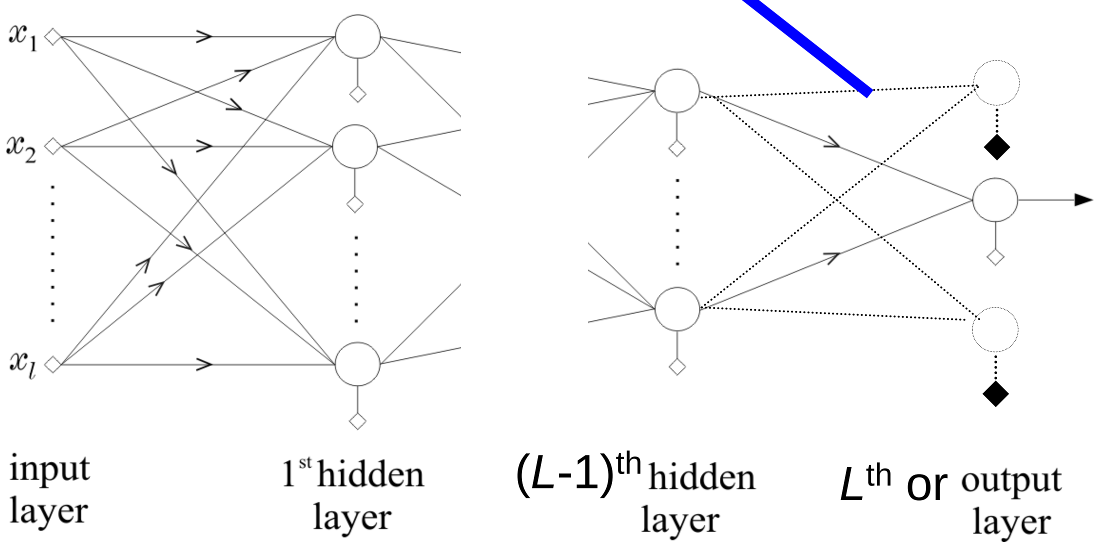




Iterative update of Synaptic weights: The Backpropagation Algorithm

- computes the weights iteratively, subject to a cost function is optimized

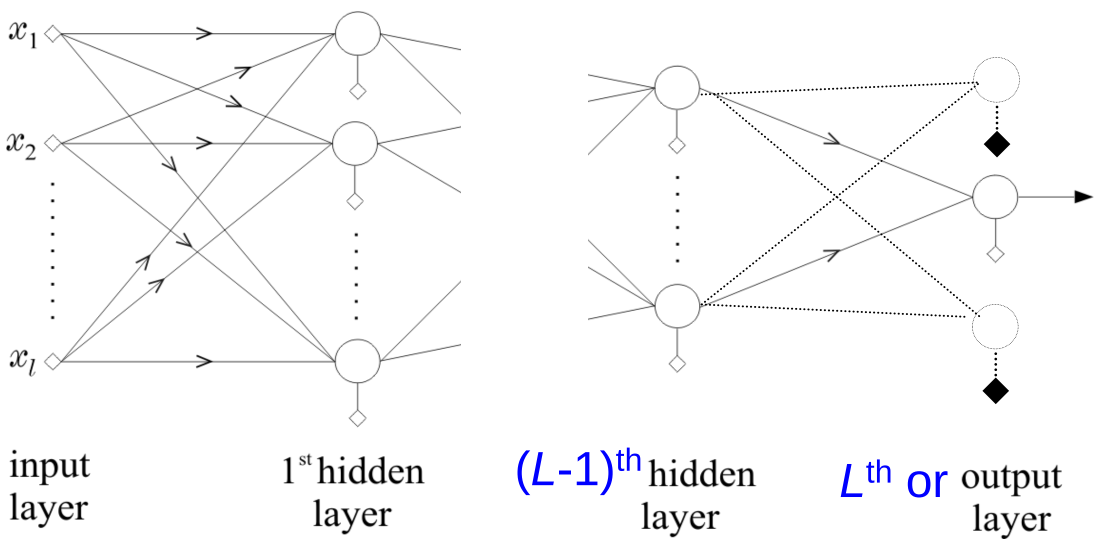
$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r$$



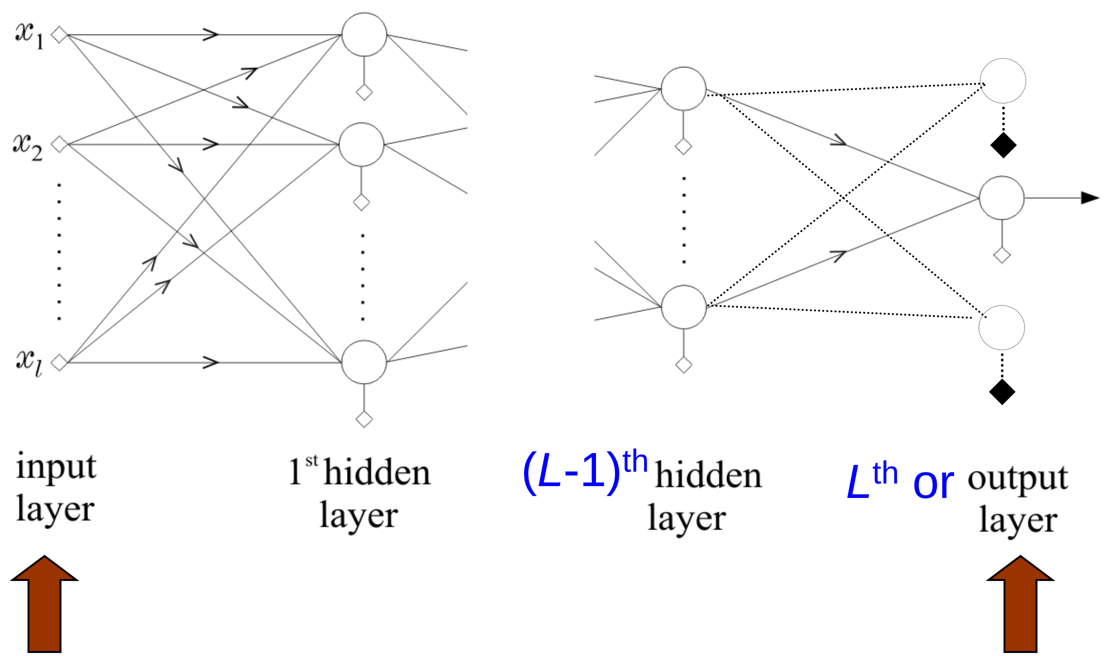
Iterative update of Synaptic weights: The Backpropagation Algorithm

Assume:

- Multiple layers
- more than one neurons in each layer
- any number of classes



Iterative update of Synaptic weights: The Backpropagation Algorithm

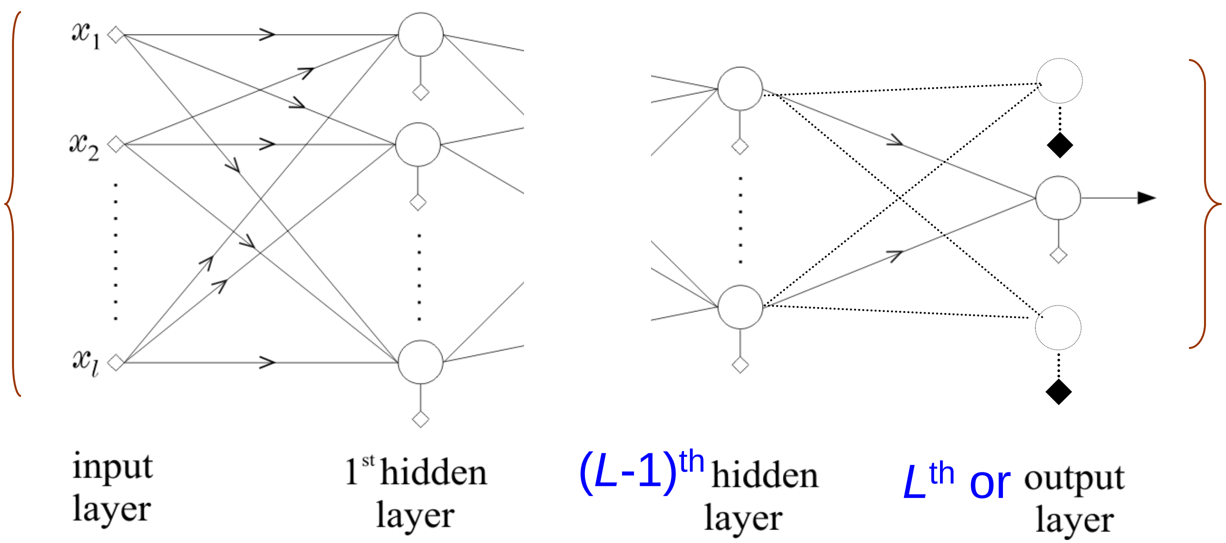


Inputs vectors applied here

Estimated Outputs, one for each class

Iterative update of Synaptic weights: The Backpropagation Algorithm

means a
single pattern



assumes a
single neuron
will be activated

Inputs vectors
applied here

Estimated Outputs,
one for each class

Iterative update of Synaptic weights: The Backpropagation Algorithm

Let:

- 4 classes

Training Sample#	Class#
Sample#1	1
Sample#2	3
Sample#3	2
Sample#4	4
Sample#5	2

Iterative update of Synaptic weights: The Backpropagation Algorithm

Let:

- 4 classes

Training Sample#	Class#	Class Vector
Sample#1	1	1 0 0 0
Sample#2	3	0 0 1 0
Sample#3	2	0 1 0 0
Sample#4	4	0 0 0 1
Sample#5	2	0 1 0 0

The Backpropagation Algorithm

- Recall the perceptron algorithm:
  - We update with this $\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta \underline{w}$

The Backpropagation Algorithm

- Recall the perceptron algorithm:
  - We update with this $\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta \underline{w}$

- Backpropagation updates multiple nodes for a number of layers:

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

The Backpropagation Algorithm

- Recall the perceptron algorithm:
  - We update with this $\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta \underline{w}$
- Backpropagation updates multiple nodes for a number of layers:

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

r-th layer

j-th neuron

The Backpropagation Algorithm

- Another difference is the activation function:
- Perceptron algorithm uses unit activation function:

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

  - This function is not differentiable at x=0.

The Backpropagation Algorithm

- Another difference is the activation function:
- Perceptron algorithm uses unit activation function:

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

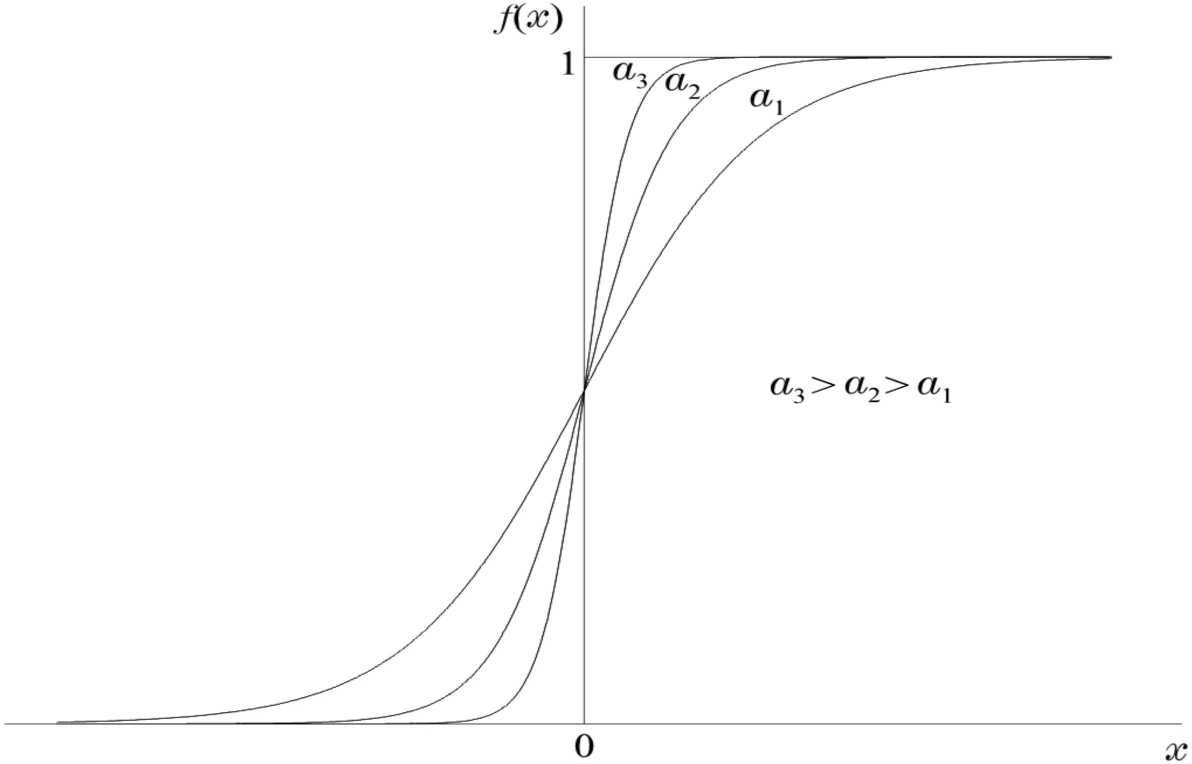
  - This function is not differentiable at x=0.

- Backpropagation uses logistic function:

$$f(x) = \frac{1}{1 + \exp(-ax)}$$

Logistic function

The Logistic function



$$f(x) = \frac{1}{1 + \exp(-ax)}$$

The Backpropagation Algorithm

- Similar to perceptron algorithm: *Backpropagation also iteratively updates weights*

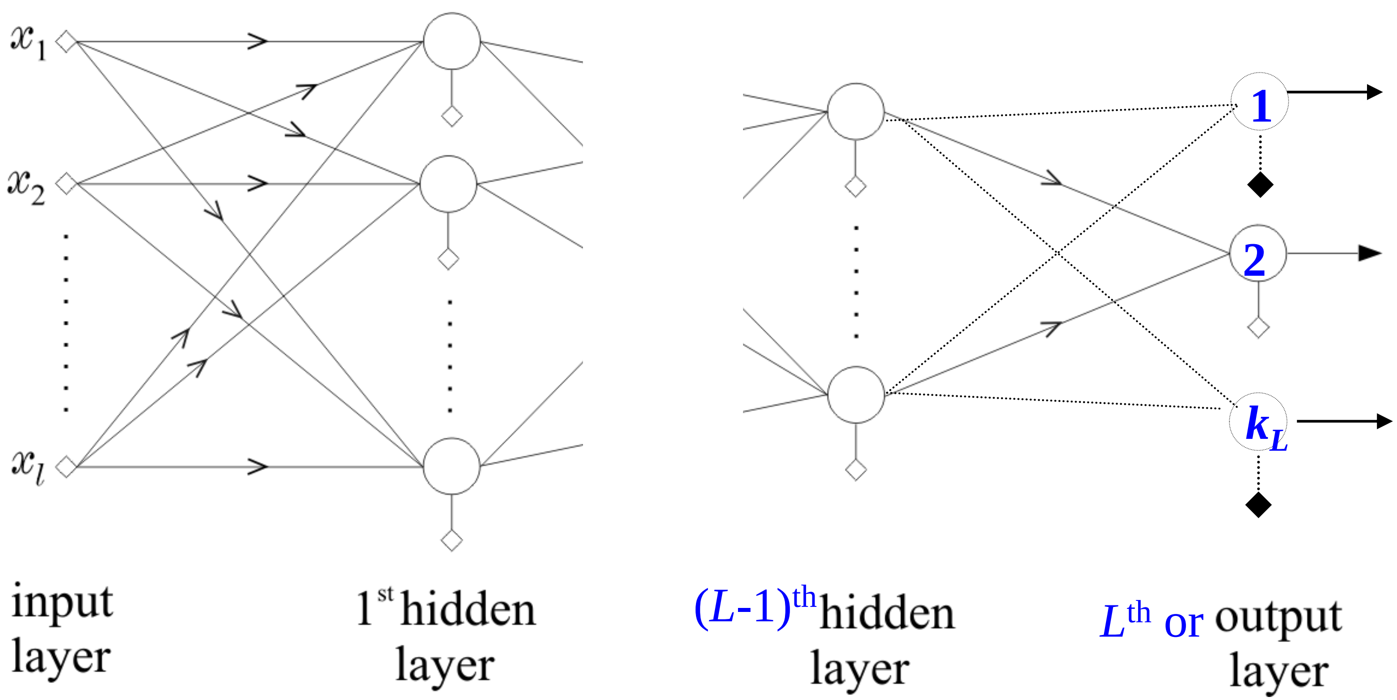
$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta\boldsymbol{w}_j^r$$

where, $$\Delta\boldsymbol{w}_j^r = -\mu\frac{\partial J}{\partial \boldsymbol{w}_j^r}$$

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

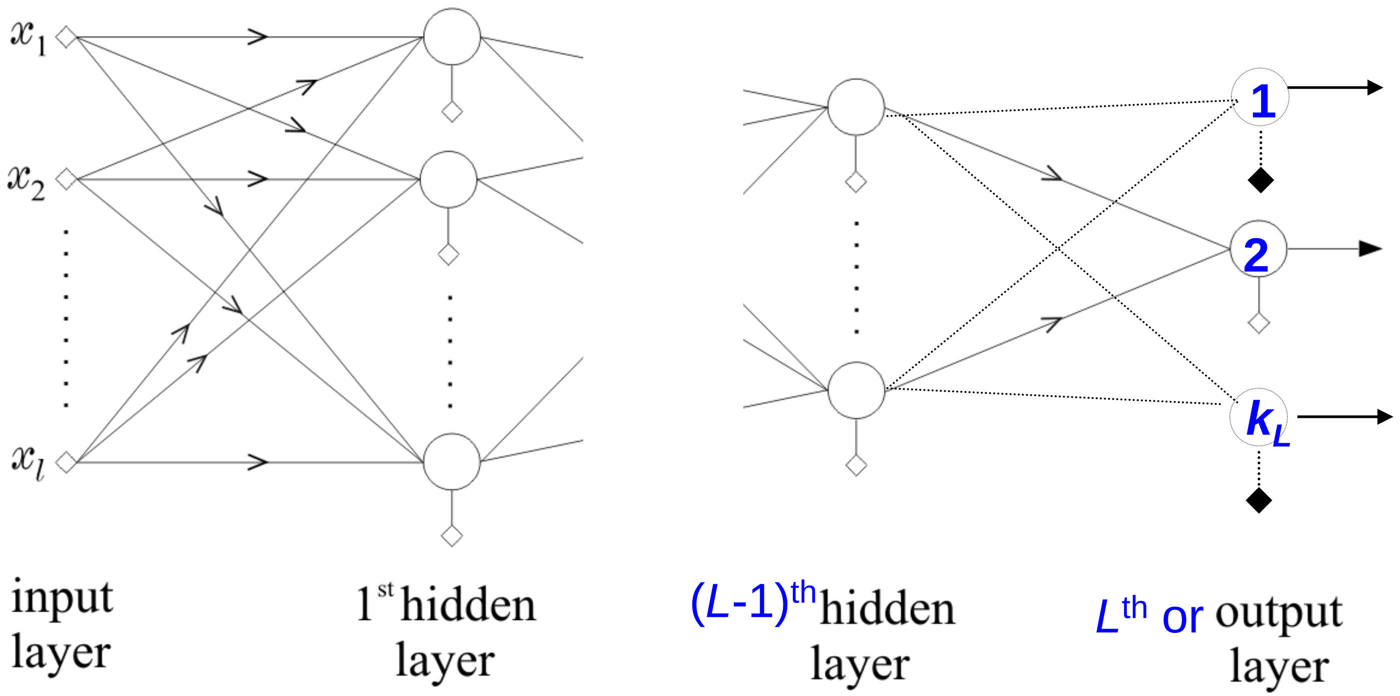
Define the Terms

- L layers of neurons
- k_r neurons in r^{th} layer
- k_0 nodes in the input layer = input feature dimension = l
- k_L nodes in the output layer = output class dimension



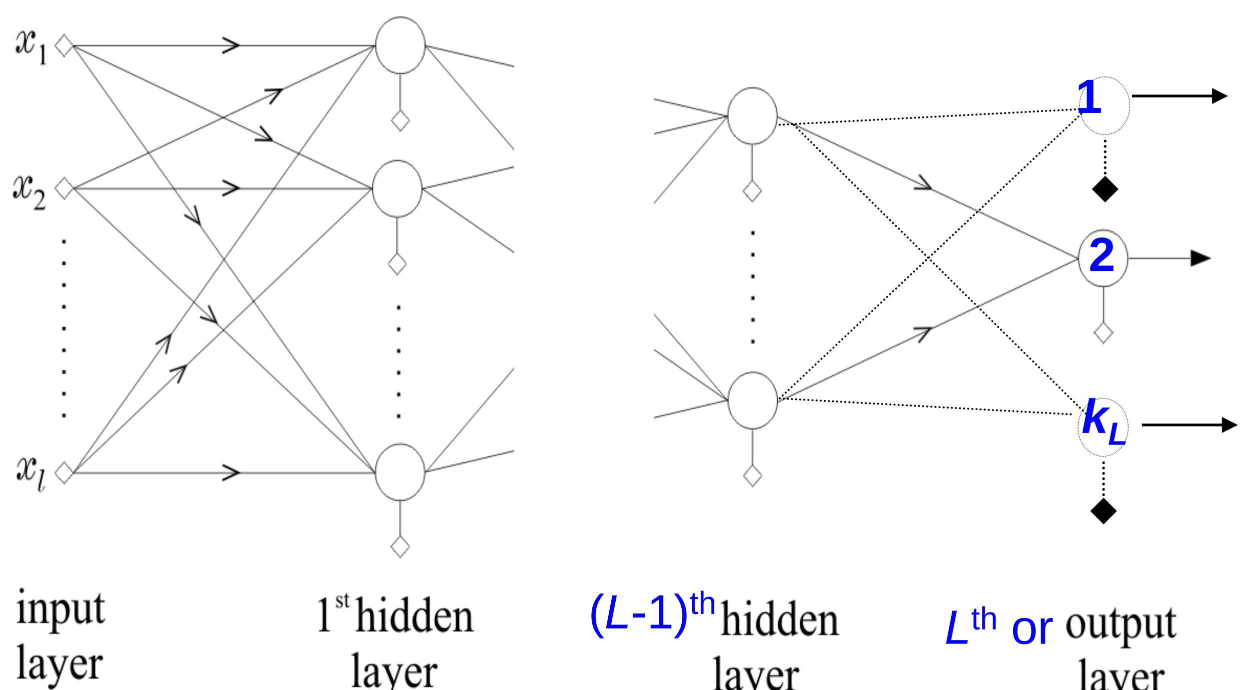
Define the Terms

- *Remember*: The number of classes is more than 2, it is $K_{L.}$
- Class value of a sample is no longer a single variable, rather it is a vector of k_L dimension.



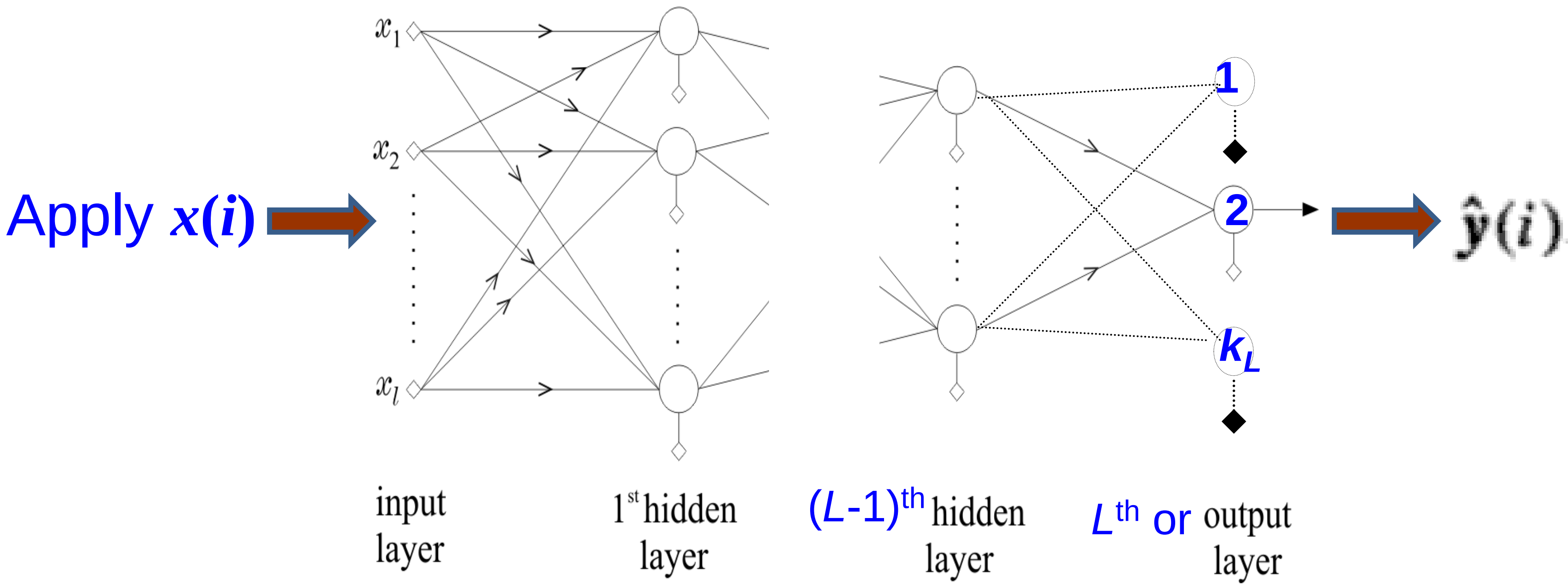
Define the Terms

- N training samples: $(\boldsymbol{x}(i), \boldsymbol{y}(i))$, for $i = 1, 2, 3, \ldots, N$
- Features of ith training sample: $\boldsymbol{x}(i) = [x_1(i), \ldots, x_{k_0}(i)]^T$
- Class of ith training sample: $\boldsymbol{y}(i) = [y_1(i), \ldots, y_{k_L}(i)]^T$



Define the Terms

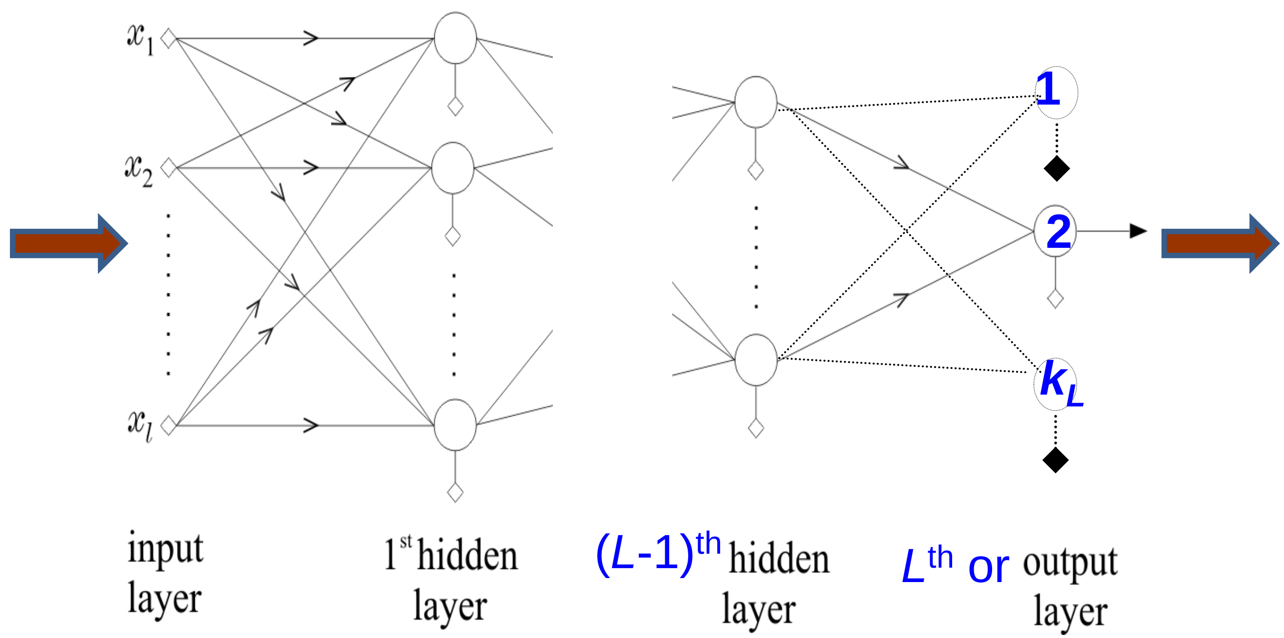
- During training, apply i^{th} training vector $\boldsymbol{x}(i)$, and output is $\hat{\mathbf{y}}(i)$, instead of $\boldsymbol{y}(i)$)



Define the Terms

- During training, apply i^{th} training vector $\boldsymbol{x}(i)$, and output is $\hat{\mathbf{y}}(i)$, instead of $\boldsymbol{y}(i)$)

Apply $\boldsymbol{x}(\boldsymbol{i})$

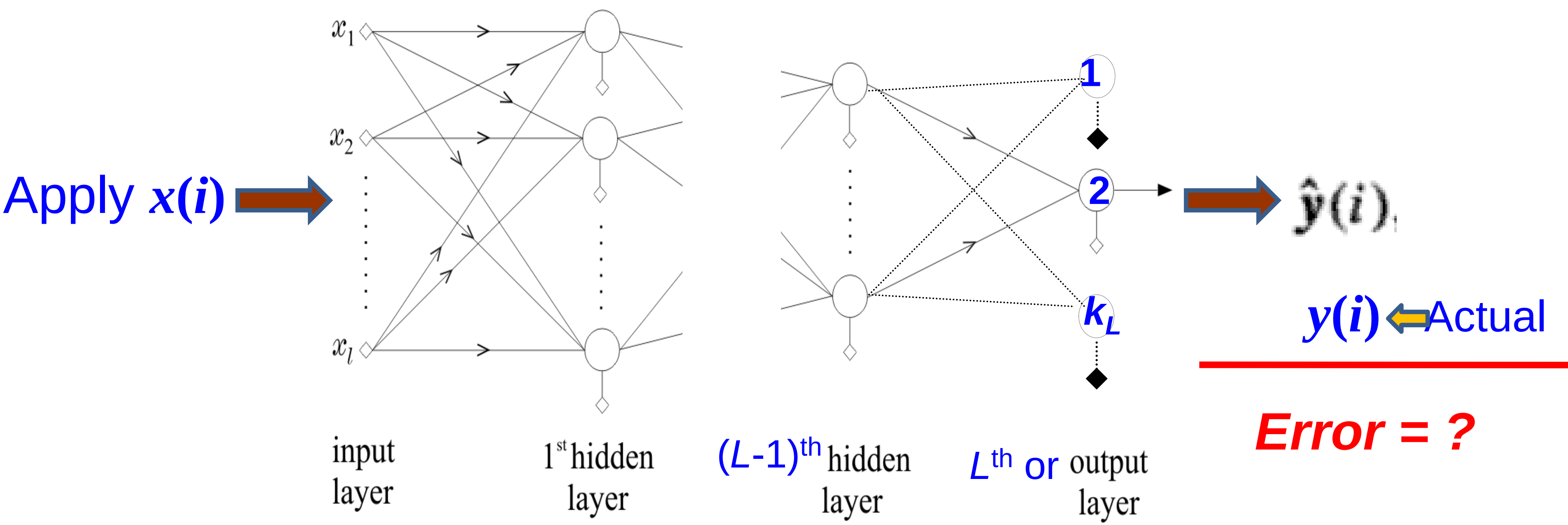


$\hat{\mathbf{y}}(i)$

$\boldsymbol{y}(\boldsymbol{i})$ Actual

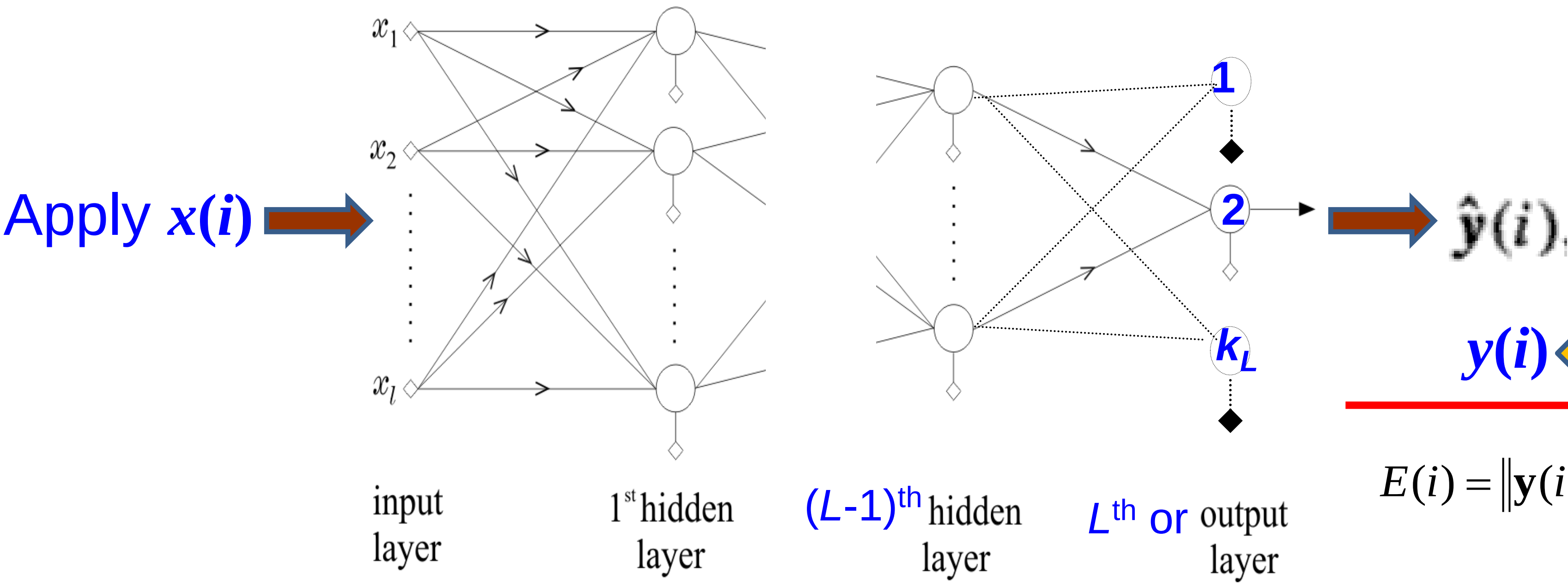
Define the Terms

- During training, apply i^{th} training vector $\boldsymbol{x}(i)$, and output is $\hat{\mathbf{y}}(i)$, instead of $\boldsymbol{y}(i)$)



Define the Terms

- During training, apply i^{th} training vector $\boldsymbol{x}(i)$, and output is $\hat{\mathbf{y}}(i)$, instead of $\boldsymbol{y}(i)$)



$\boldsymbol{y(i)}$ ⇐ Actual

$$E(i) = \left\| \mathbf{y}(i) - \hat{\mathbf{y}}(i) \right\|^2$$

Define the Terms

- During training, apply i^{th} training vector $\boldsymbol{x}(i)$, and output is $\hat{\mathbf{y}}(i)$, instead of $\boldsymbol{y}(i)$)

- Error for i^{th} vector:

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

- Total Error:

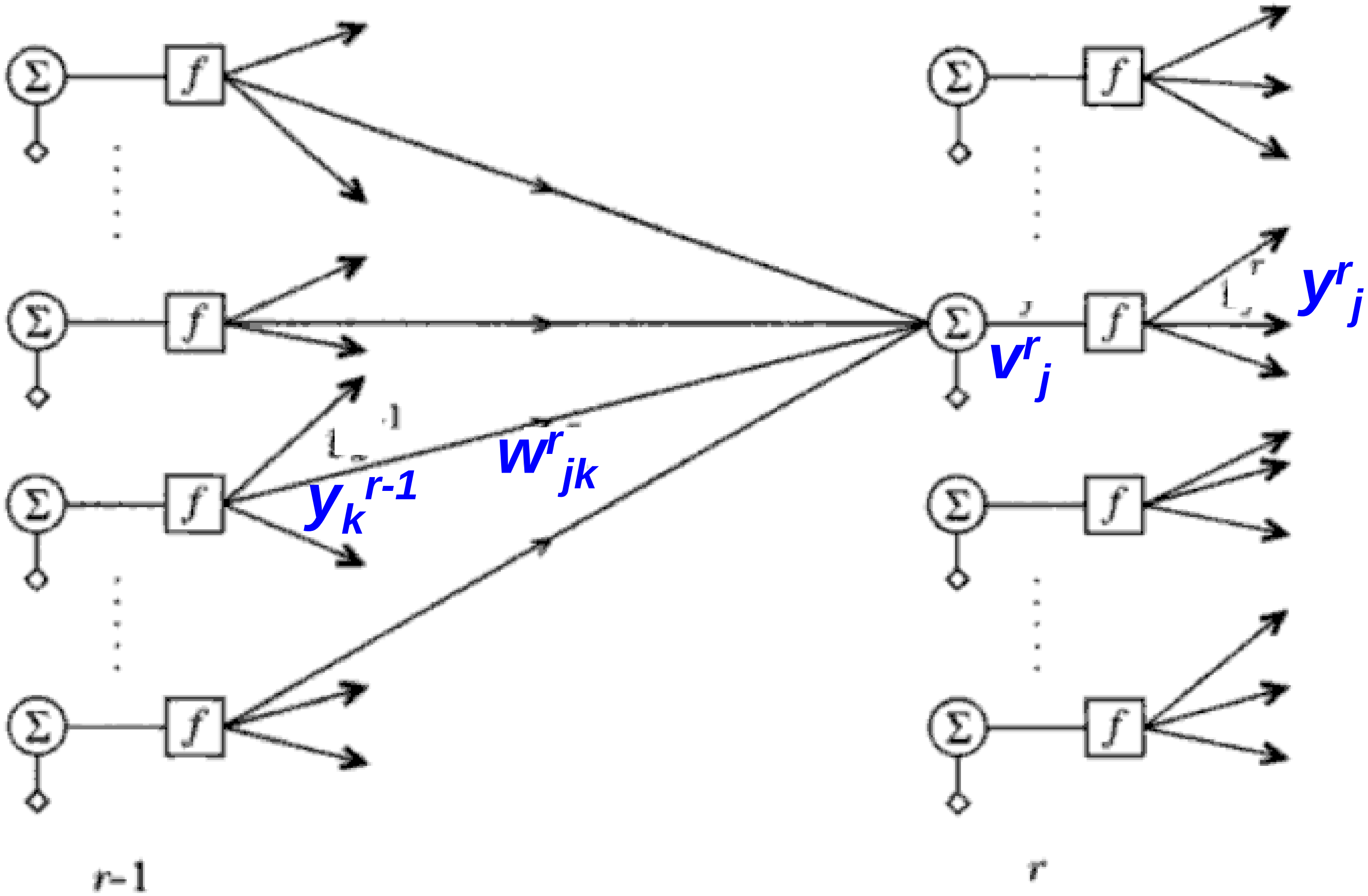
$$J = \sum_{i=1}^{N} \mathcal{E}(i)$$

Define the Terms

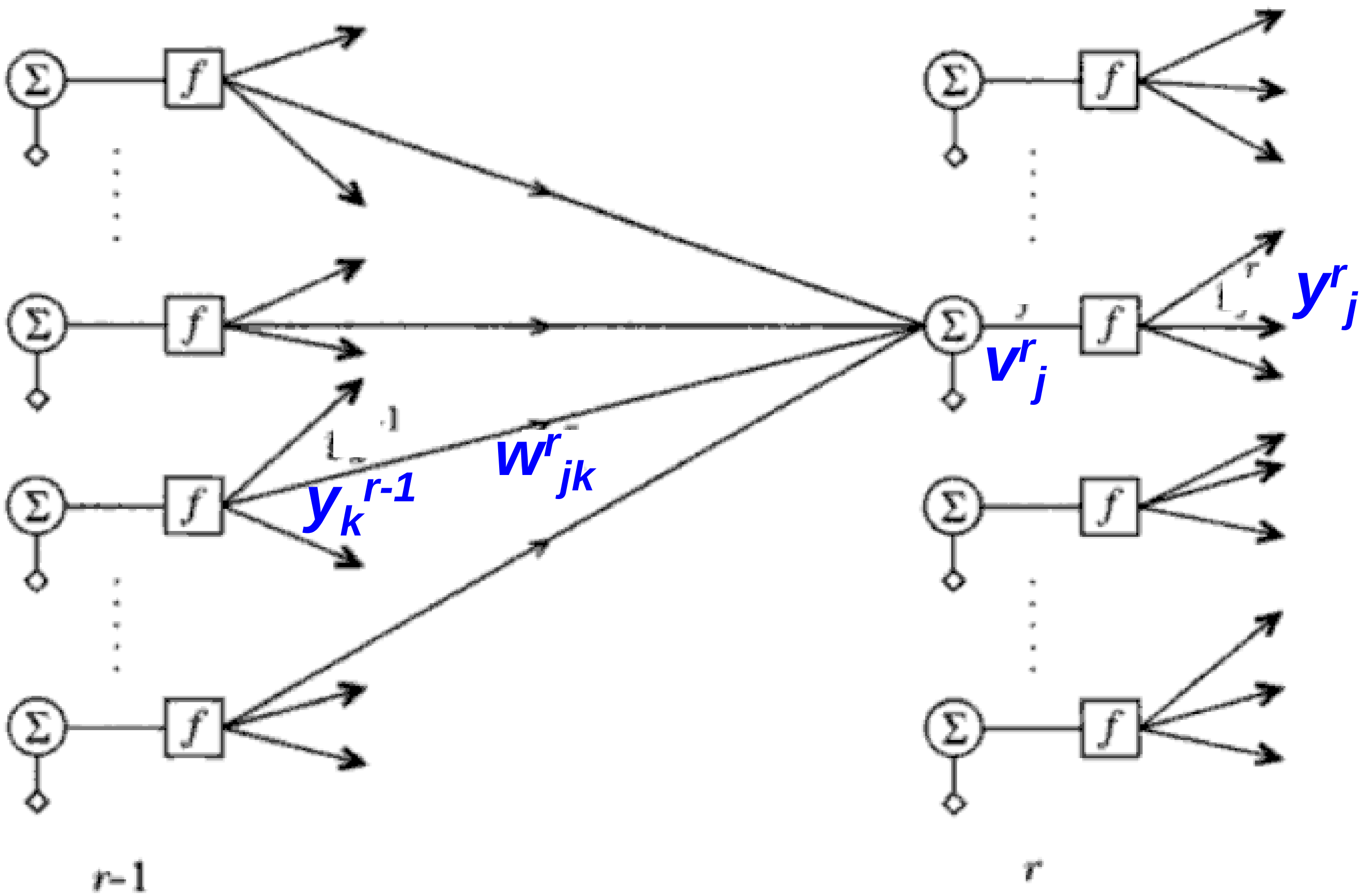
- We need to calculate $\Delta \boldsymbol{w}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{w}_j^r} = -\mu \sum_{i=1}^{N} \frac{\partial E(i)}{\partial \mathbf{w}_j^r(i)}$

Define the Terms

- We need to calculate $\Delta \boldsymbol{w}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{w}_j^r} = -\mu \sum_{i=1}^{N} \frac{\partial E(i)}{\partial \mathbf{w}_j^r(i)}$
- J depends on $\boldsymbol{w}^r_j$ and passes through v^r_j



Define the Terms



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

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial \upsilon_j^r(i)} \frac{\partial \upsilon_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial \upsilon_j^r(i)} \boxed{\frac{\partial \upsilon_j^r(i)}{\partial \boldsymbol{w}_j^r}}$$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial \upsilon_j^r(i)} \frac{\partial \upsilon_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

Recall, $\upsilon_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{jo}^r \equiv \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i)$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial \upsilon_j^r(i)} \frac{\partial \upsilon_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

Recall, $\upsilon_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{jo}^r \equiv \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i)$

Therefore, $\frac{\partial}{\partial \boldsymbol{w}_j^r} \upsilon_j^r(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} \upsilon_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} \upsilon_j^r(i) \end{bmatrix}$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}^r_j} = \frac{\partial \mathcal{E}(i)}{\partial v^r_j(i)} \frac{\partial v^r_j(i)}{\partial \boldsymbol{w}^r_j}$$

Recall, $v^r_j(i) = \sum_{k=1}^{k_{r-1}} w^r_{jk} y^{r-1}_k(i) + w^r_{jo} \equiv \sum_{k=0}^{k_{r-1}} w^r_{jk} y^{r-1}_k(i)$

Therefore, $\frac{\partial}{\partial \boldsymbol{w}^r_j} v^r_j(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w^r_{j0}} v^r_j(i) \\ \vdots \\ \frac{\partial}{\partial w^r_{jk_{r-1}}} v^r_j(i) \end{bmatrix} = \begin{bmatrix} +1 \\ y^{r-1}_1(i) \\ \vdots \\ y^{r-1}_{k_{r-1}}(i) \end{bmatrix}$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

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

Therefore, $\frac{\partial}{\partial \boldsymbol{w}_j^r} v_j^r(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} v_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} v_j^r(i) \end{bmatrix} = \boldsymbol{y}^{r-1}(i)$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \equiv \delta_j^r(i)$$

$$\frac{\partial}{\partial \boldsymbol{w}_j^r} v_j^r(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} v_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} v_j^r(i) \end{bmatrix} = \boldsymbol{y}^{r-1}(i)$$

Define the Terms

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial \upsilon_j^r(i)} \frac{\partial \upsilon_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

$$\frac{\partial \mathcal{E}(i)}{\partial \upsilon_j^r(i)} \equiv \delta_j^r(i)$$

$$\frac{\partial}{\partial \boldsymbol{w}_j^r} \upsilon_j^r(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} \upsilon_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} \upsilon_j^r(i) \end{bmatrix} = \boldsymbol{y}^{r-1}(i)$$

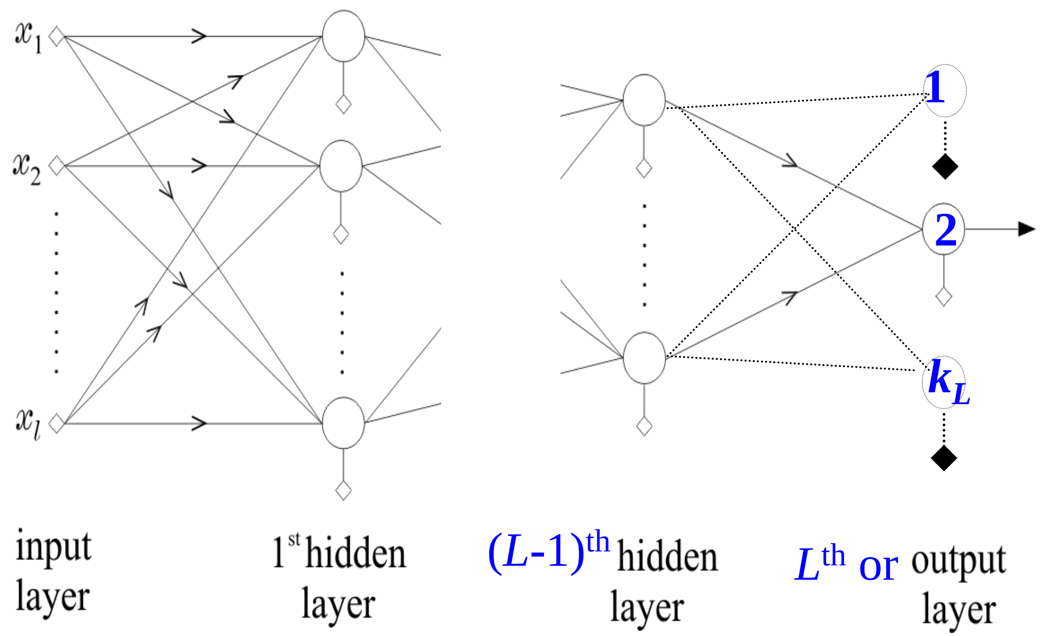
$$\Delta \mathbf{w}_j^r = -\mu \sum_{i=1}^{N} \frac{\partial E(i)}{\partial \mathbf{w}_j^r(i)} \Rightarrow \boxed{\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i)}$$

Define the Terms

$$\boxed{\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i)}$$

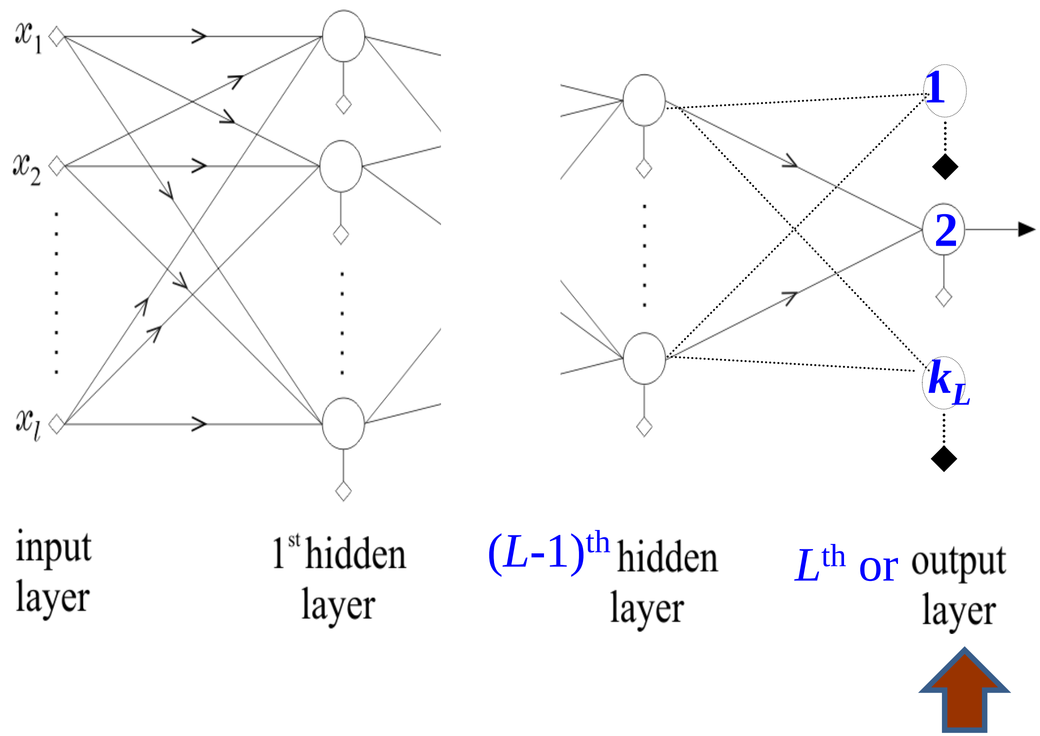
Define the Terms

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i)\, \boldsymbol{y}^{r-1}(i)$$



Define the Terms

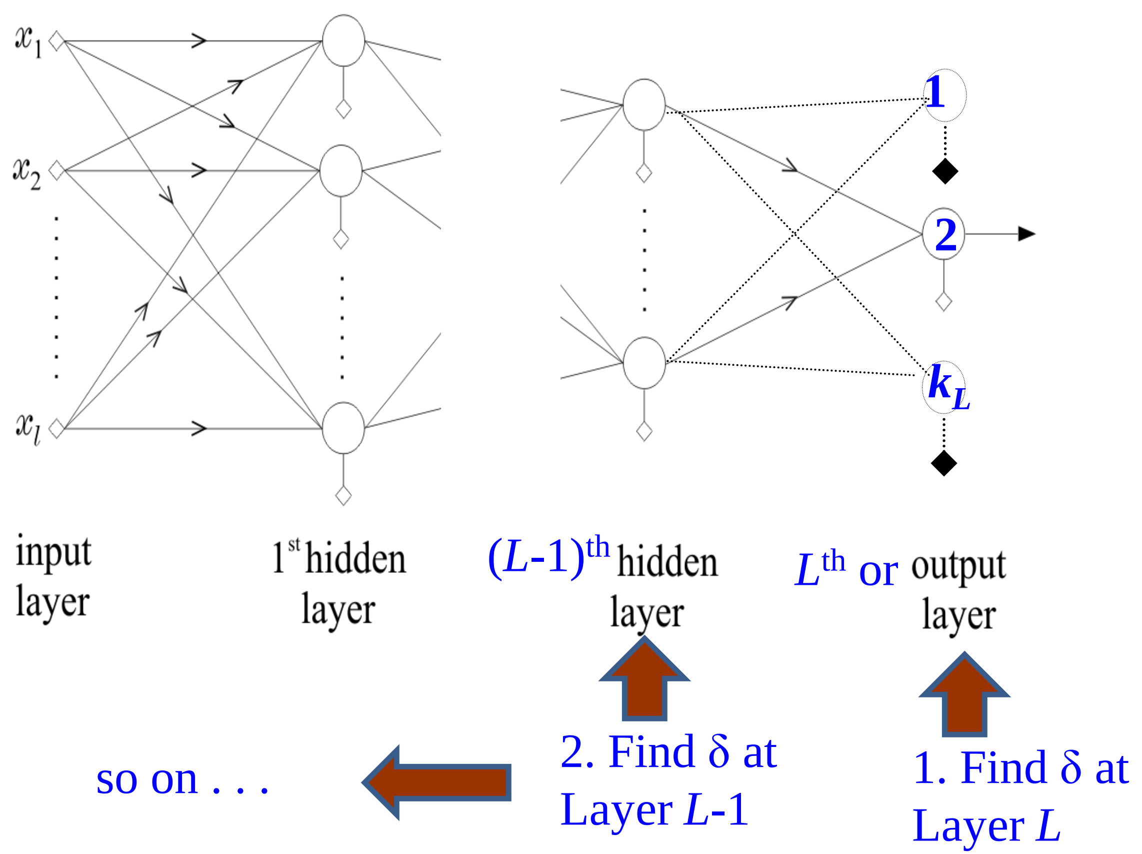
$$\boxed{\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i)}$$



1. Find δ at Layer *L*

Define the Terms

$$\boxed{\Delta w_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i)\, y^{r-1}(i)}$$



Define the Terms

- Calculate $\delta_j^r(i) = \dfrac{\partial \mathrm{E}(i)}{\partial v_j^r(i)}$

Define the Terms

- Calculate $\delta_j^r(i) = \dfrac{\partial \mathrm{E}(i)}{\partial v_j^r(i)}$
- For $r = L$

$$\delta_j^L(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^L(i)}$$

Define the Terms

- Calculate $\delta_j^r(i) = \dfrac{\partial \mathrm{E}(i)}{\partial v_j^r(i)}$
- For $r = L$

$$\delta_j^L(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^L(i)}$$

$$E(i) = \frac{1}{2}\sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L} (f(v_m^L(i)) - y_m(i))^2$$

Define the Terms

- Calculate $\delta_j^r(i) = \dfrac{\partial \mathrm{E}(i)}{\partial v_j^r(i)}$
- For $r = L$

$$\delta_j^L(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^L(i)}$$

$$E(i) = \frac{1}{2}\sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L}(f(v_m^L(i)) - y_m(i))^2$$

$$\delta_j^L(i) = e_j(i) f'(v_j^L(i))$$

Define the Terms

- For $r < L$
- Calculate $\delta_j^{r-1}(i)$ from $\delta_j^r(i)$

Define the Terms

- For $r < L$
- Calculate $\delta_j^{r-1}(i)$ from $\delta_j^{r}(i)$

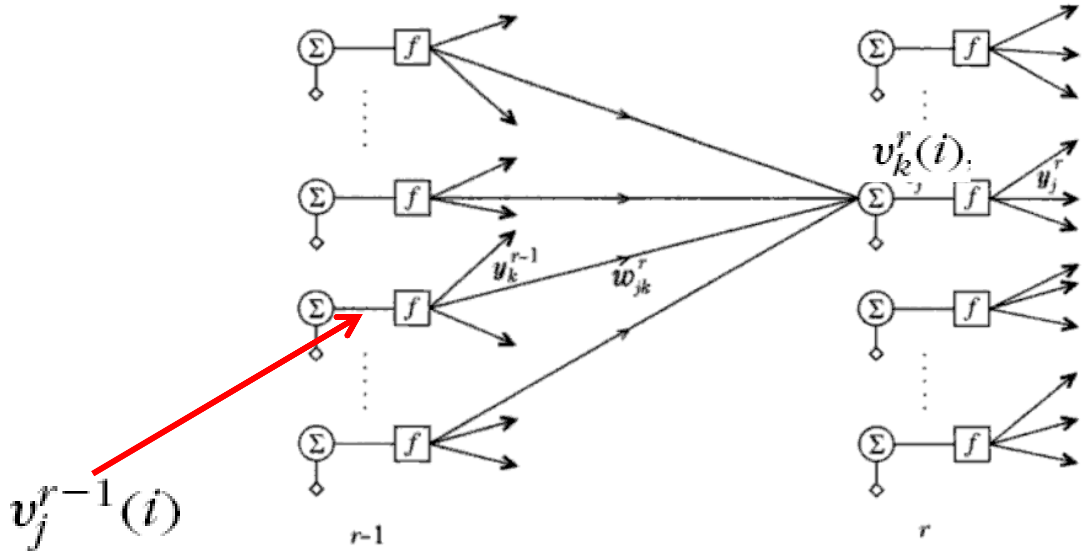
- We know,

$$\delta_j^{r-1}(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)}$$

Define the Terms

- We need to calculate, $$\delta_j^{r-1}(i) = \frac{\partial E(i)}{\partial v_j^{r-1}(i)}$$

- However, $v_j^{r-1}(i)$ influences all $v_k^r(i)$, for $k = 1, 2, 3, \ldots, k_r$



Define the Terms

- We need to calculate,

$$\delta_j^{r-1}(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)}$$

- However, $v_j^{r-1}(i)$ influences all $v_k^r(i)$, for k = 1, 2, 3, …, k_r

- Therefore,

$$\frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial \mathrm{E}(i)}{\partial v_k^r(i)} \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

Define the Terms

- For *r* < *L*
- Calculate

$$\delta_j^{r-1}(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)}$$

$$\frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial \mathrm{E}(i)}{\partial v_k^r(i)} \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

Define the Terms

- For $r < L$
- Calculate

$$\delta_j^{r-1}(i) = \frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)}$$

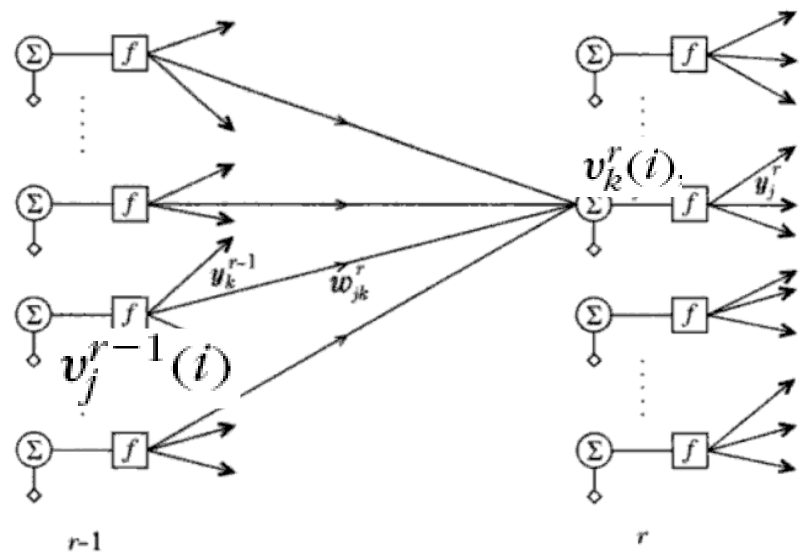
$$\frac{\partial \mathrm{E}(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial \mathrm{E}(i)}{\partial v_k^{r}(i)} \frac{\partial v_k^{r}(i)}{\partial v_j^{r-1}(i)}$$

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^{r}(i) \frac{\partial v_k^{r}(i)}{\partial v_j^{r-1}(i)}$$

Define the Terms

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \frac{\partial\left[\sum_{m=0}^{k_{r-1}} w_{km}^r y_m^{r-1}(i)\right]}{\partial v_j^{r-1}(i)}$$



where, $y_m^{r-1}(i) = f\left(v_m^{r-1}(i)\right)$

Define the Terms

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \frac{\partial \left[\sum_{m=0}^{k_{r-1}} w_{km}^r y_m^{r-1}(i) \right]}{\partial v_j^{r-1}(i)}$$ **where,** $y_m^{r-1}(i) = f(v_m^{r-1}(i))$

then,

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i))$$

Define the Terms

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i))$$

Define the Terms

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i))$$

Define the Terms

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$$

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i))$$

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r\right] f'(v_j^{r-1}(i))$$

Define the Terms

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \right] f'(v_j^{r-1}(i))$$

$$\delta_j^{r-1}(i) = e_j^{r-1}(i) f'(v_j^{r-1}(i))$$

where, $e_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r$

Define the Terms

- Only remaining is the derivative of the logistic function:

$$f'(x) = \alpha f(x)(1 - f(x))$$

The Algorithm

- Initialization:
  - Start with small random weights

- Forward Computations: $v_j^r(i)$, $y_j^r(i) = f(v_j^r(i))$,

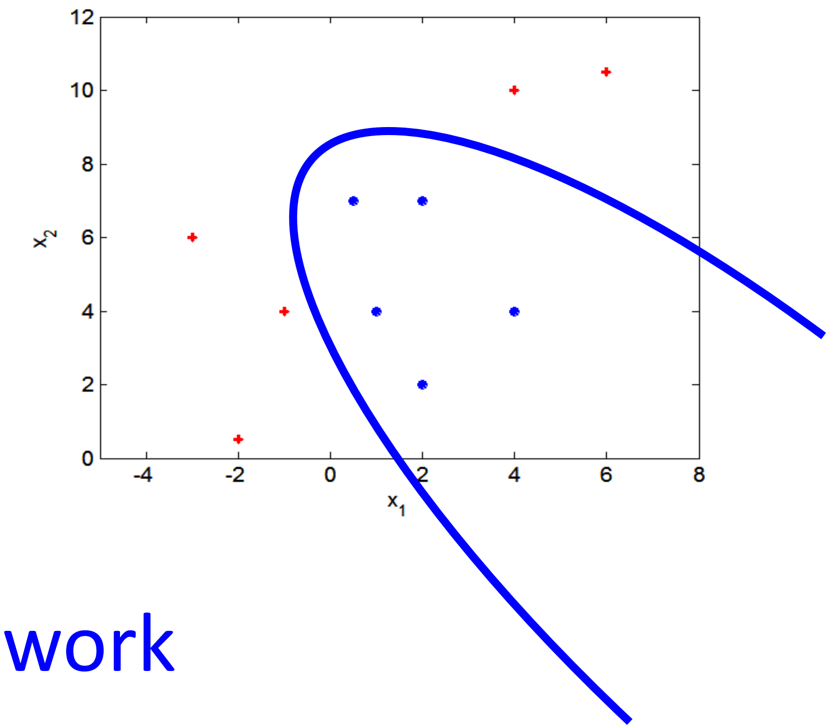
- Backward Computation: $\delta_j^L(i)$ and $\delta_j^{r-1}(i)$

- Update weight:

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

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i)$$

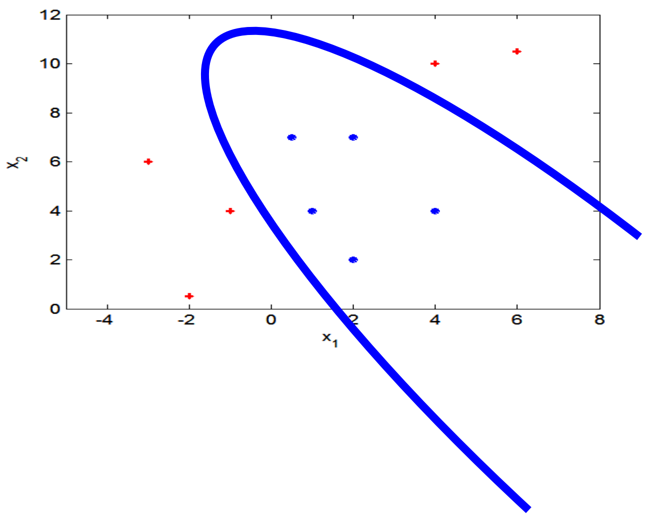
Some Nonlinear Classifiers

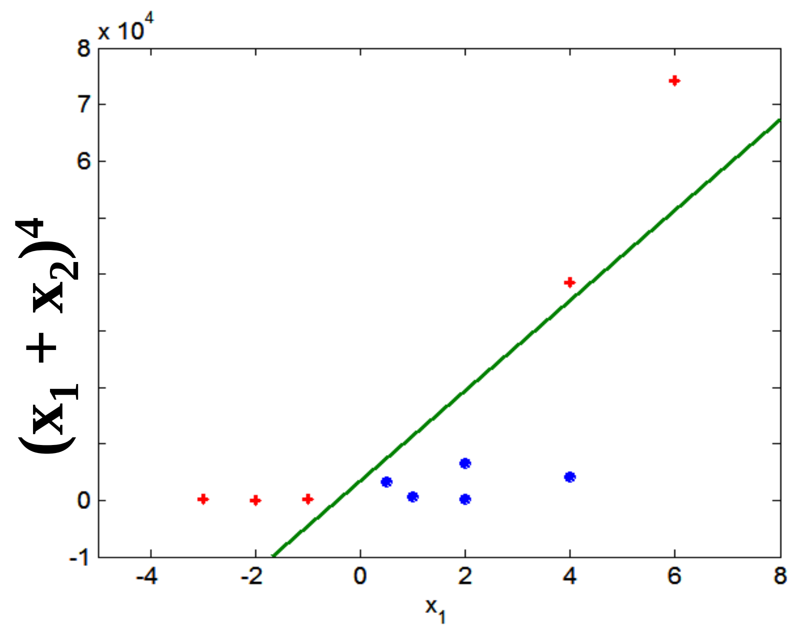


- Neural Network
- Decision Tree
- Non-linear SVM

Nonlinear SVM

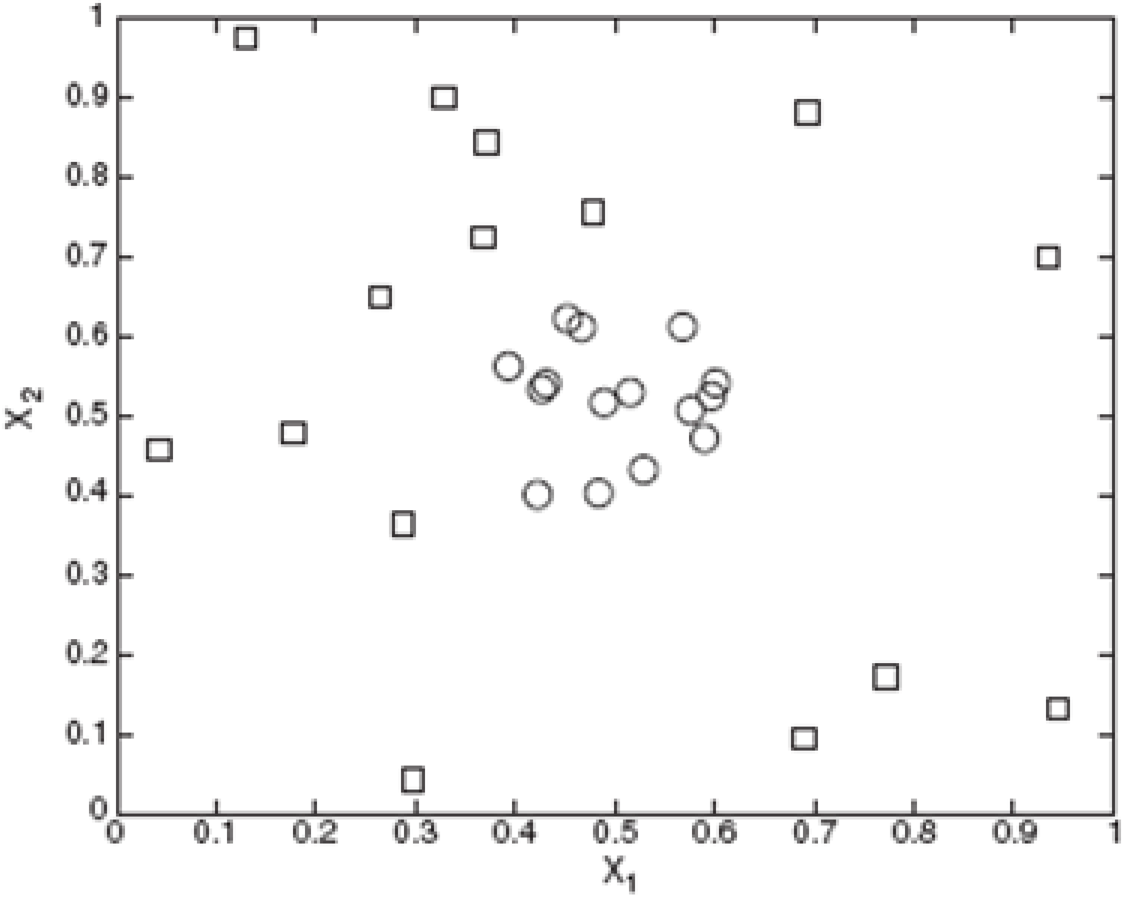
- Transform data into higher dimensional space





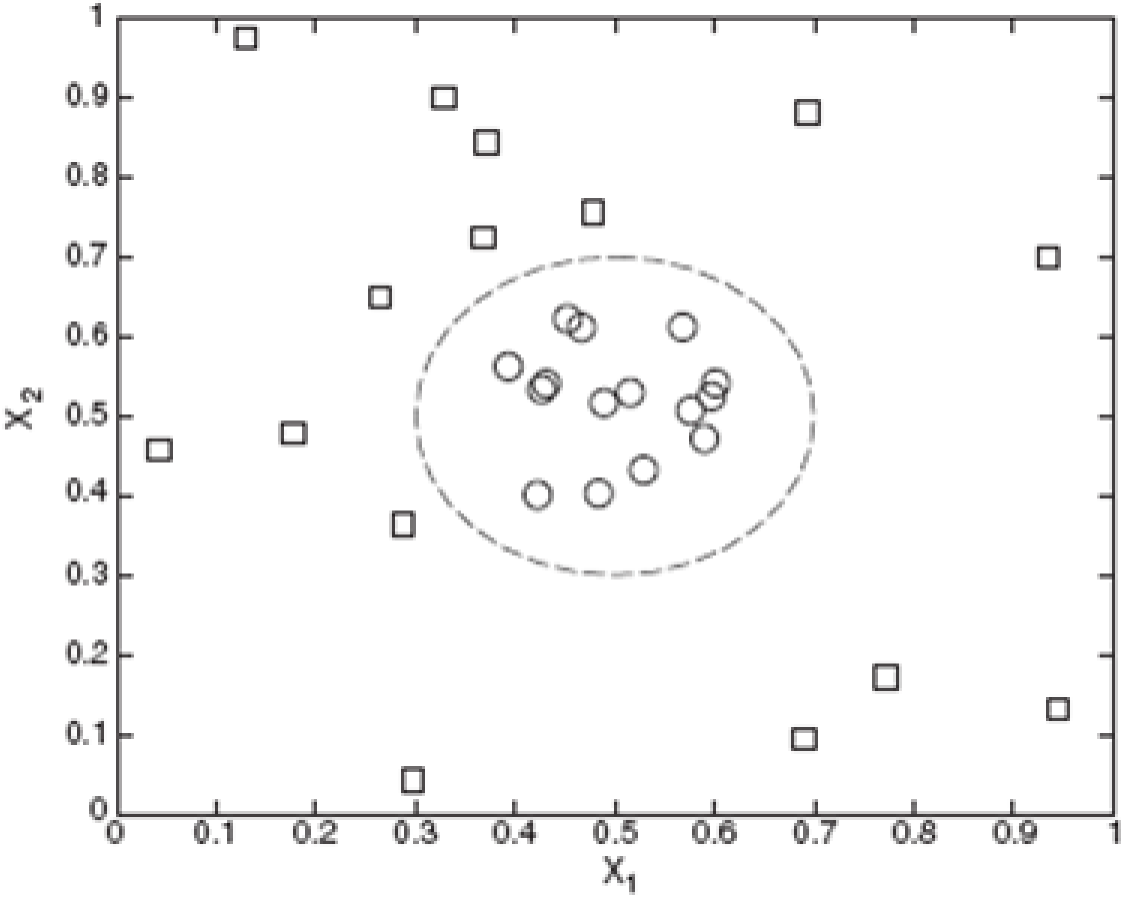
Nonlinear SVM

- Another Example



Nonlinear SVM

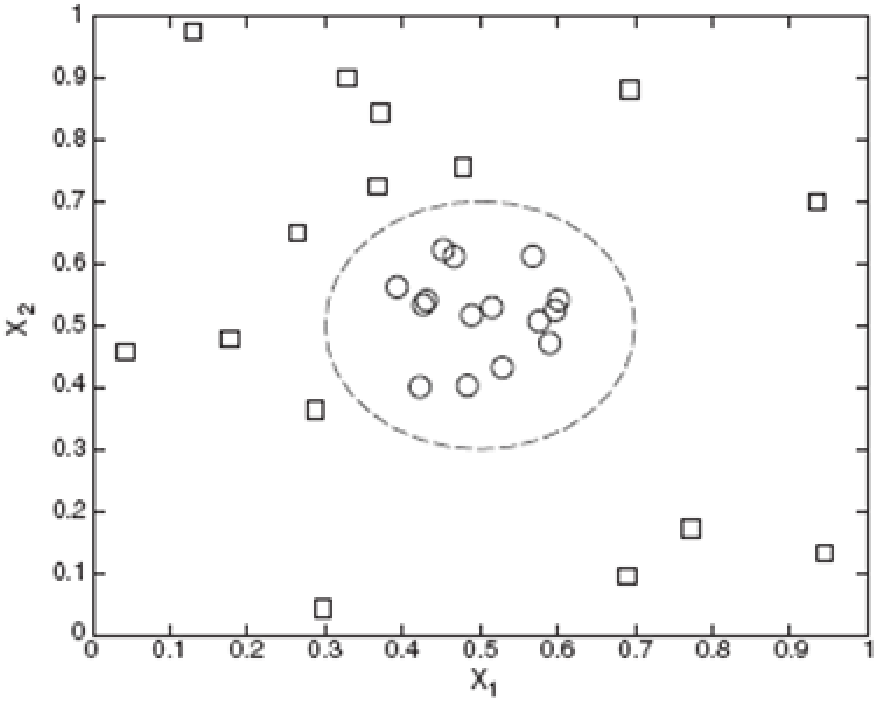
- Another Example



Nonlinear SVM

- Decision boundary:

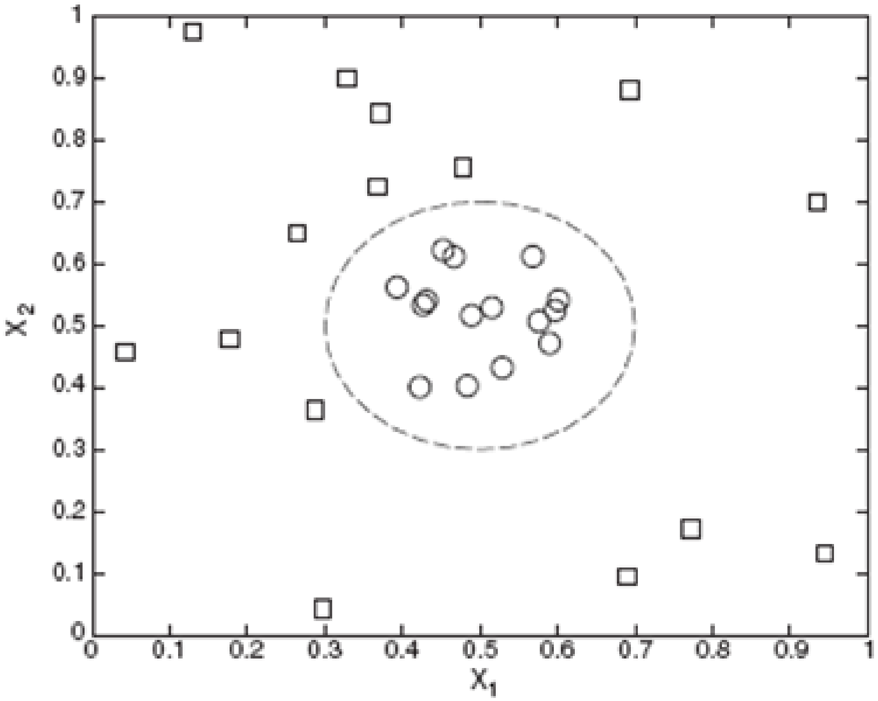
$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$



Nonlinear SVM

- Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$



- 2 classes are defined as

$$y(x_1, x_2) = \begin{cases} 1, & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1, & \text{otherwise} \end{cases}$$

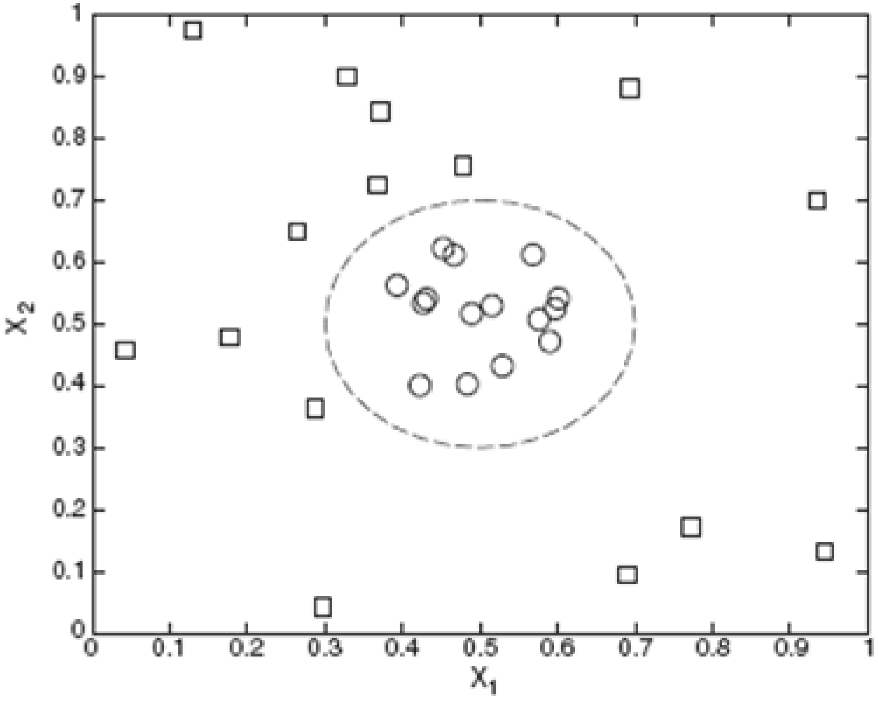
Nonlinear SVM

- Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

can be written as

$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46$$



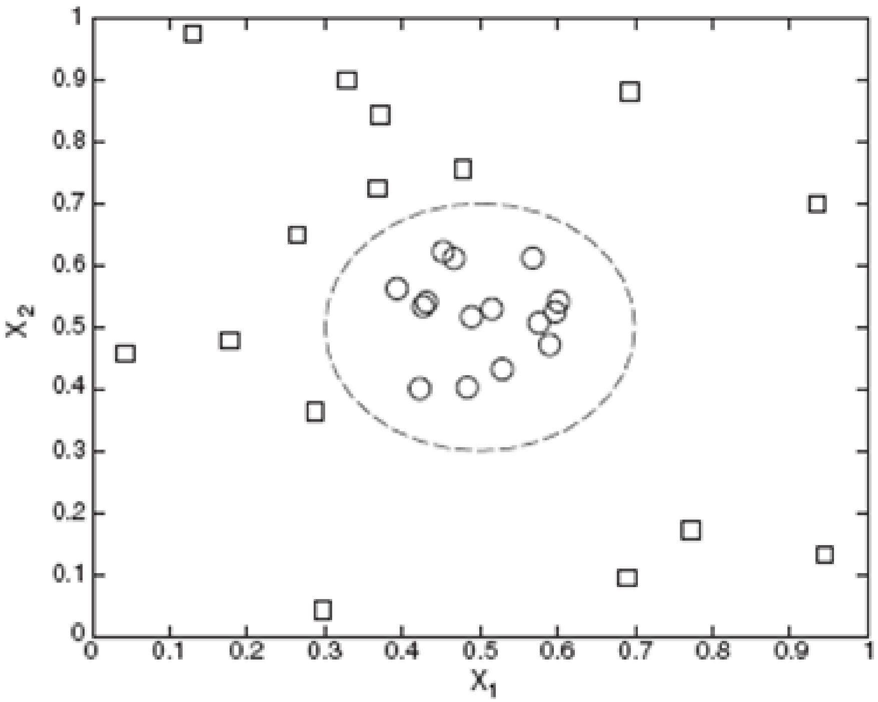
Nonlinear SVM

- Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

can be written as

$$\underbrace{x_1^2 - x_1}_{y_1} + \underbrace{x_2^2 - x_2}_{y_2} = -0.46$$



Nonlinear SVM

- Decision boundary:

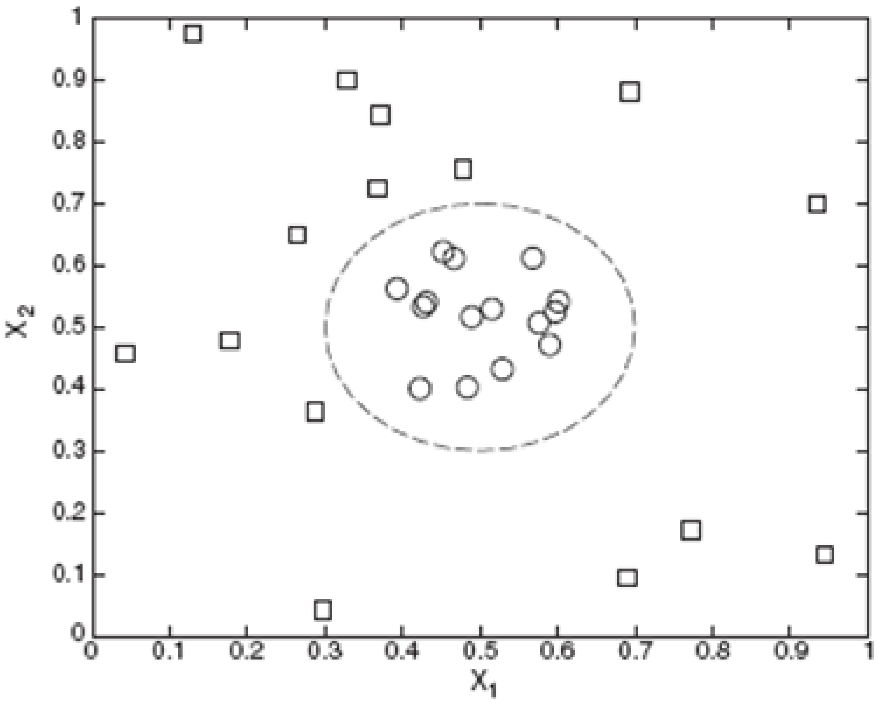
$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

can be written as

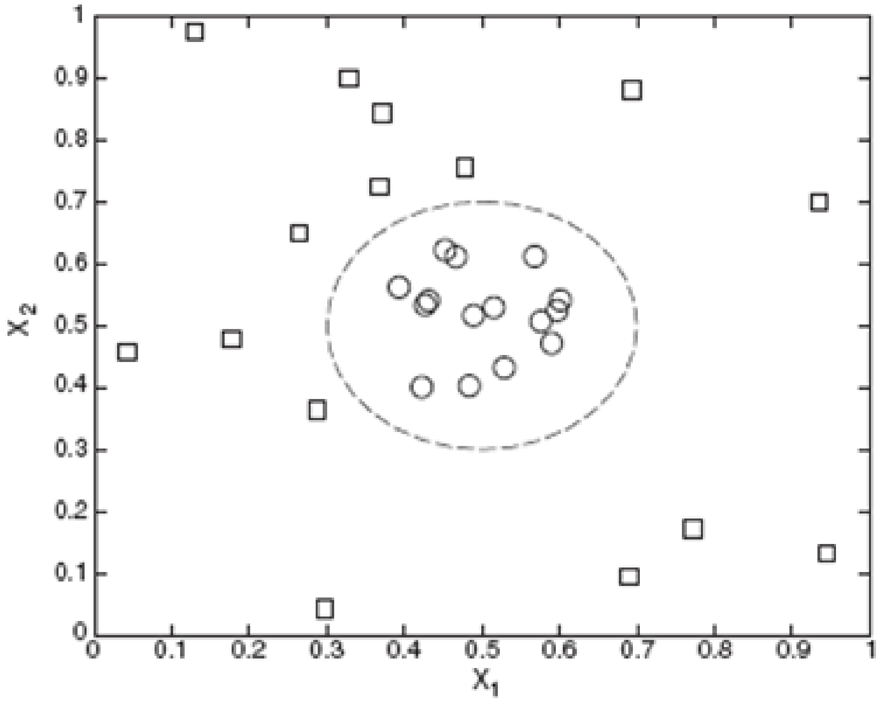
$$\underbrace{x_1^2 - x_1}_{y_1} + \underbrace{x_2^2 - x_2}_{y_2} = -0.46$$

$$\Downarrow$$

$$y_1 + y_2 = -0.46$$

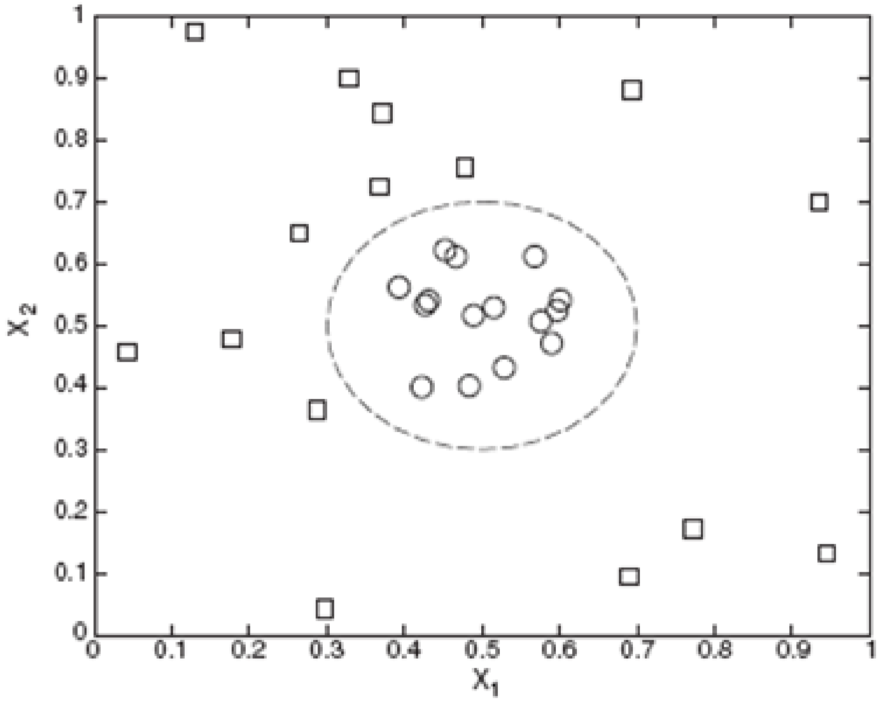


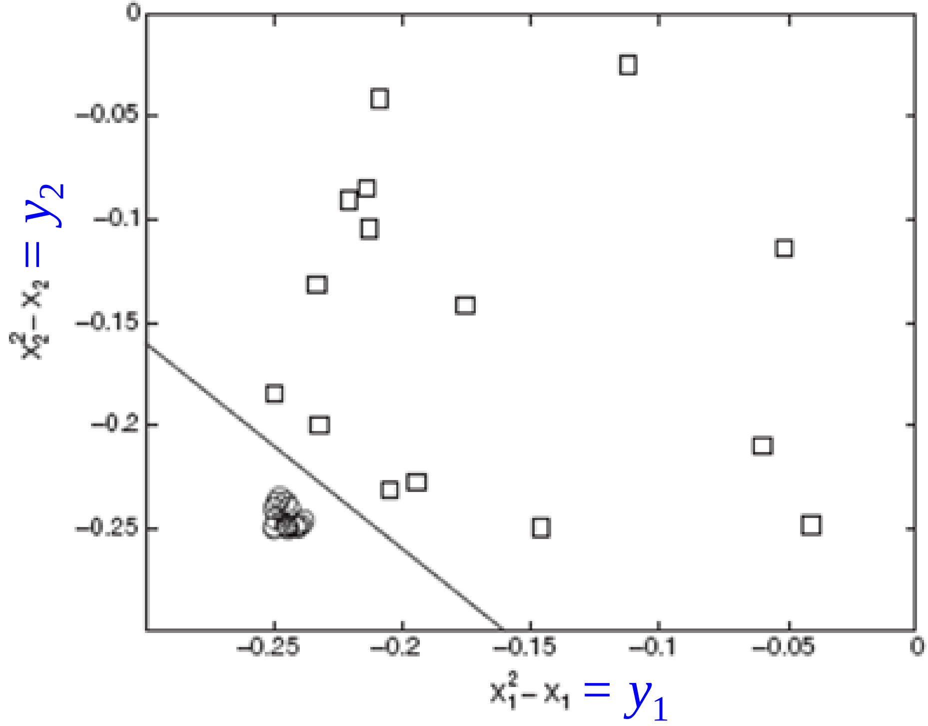
Nonlinear SVM



$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

Nonlinear SVM





$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46$$

OR, $$y_1 + y_2 = -0.46$$

Nonlinear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

Nonlinear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

OR,

$$\Phi : (x_1, x_2) \rightarrow (y_1, y_2)$$

Nonlinear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

OR,

$$\Phi : (x_1, x_2) \rightarrow (y_1, y_2)$$

OR,

$$\Phi : \mathbf{x} \rightarrow \mathbf{y}$$

Nonlinear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

- OR, more generally:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, 1)$$

Nonlinear SVM

- With the transform

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, 1)$$

The equation of the classifier will be of the form:

$$w_5 x_1^2 + w_4 x_2^2 + w_3 \sqrt{2} x_1 + w_2 \sqrt{2} x_2 + w_1 \sqrt{2} x_1 x_2 + w_0 1 = 0$$

Nonlinear SVM

- With the transform

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1)$$

The equation of the classifier will be of the form:

$$w_5 x_1^2 + w_4 x_2^2 + w_3\sqrt{2}x_1 + w_2\sqrt{2}x_2 + w_1\sqrt{2}x_1x_2 + w_0 1 = 0$$

$$w_5 y_5 + w_4 y_4 + w_3 y_3 \qquad + w_2 y_2 \qquad + w_1 y_1 \qquad + w_0 1 = 0$$

Nonlinear SVM

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1) \rightarrow (y_1, y_2, \cdots, y_5)$$

$$w_5 x_1^2 + w_4 x_2^2 + w_3\sqrt{2}x_1 + w_2\sqrt{2}x_2 w_1\sqrt{2}x_1x_2 + w_0 1)$$

- The main idea: *linearly separability* increases as the feature dimension increases

Formulation of a Nonlinear SVM

- With the new feature vectors $\Phi(\vec{x})$, replace all **x** with $\Phi(\vec{x})$ in linear SVM:
  - Minimize $L(w) = \frac{\| \vec{w} \|^2}{2}$
  - Subject to $y_i(\vec{\mathrm{w}} \bullet \Phi(\vec{\mathrm{x}}_{\mathrm{i}}) + \mathrm{b}) \geq 1$
  - The Dual function is:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

Nonlinear SVM

- Once we get the solution of λ's, find w and b using following equations:

$$\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\vec{x}_i)$$

$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i) + b) - 1 \} = 0$$

Nonlinear SVM

- The new object **z** is classified as:

$$
\begin{aligned}
f(\vec{z}) &= sign(\vec{w} \cdot \Phi(\vec{z}) + b) \\
&= sign(\sum_i \lambda_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{z}) + b)
\end{aligned}
$$

Issues in the Design Nonlinear SVM

- The mapping function is often unclear
- The dimensionality increases, leading to high computation

Issues in the Design Nonlinear SVM

- The mapping function is often unclear
- The dimensionality increases, leading to high computation

Solution?

Issues in the Design Nonlinear SVM

- Note the equations:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i) + b) - 1 \} = 0$$

$$f(\vec{z}) = sign(\sum_i \lambda_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{z}) + b)$$

Issues in the Design Nonlinear SVM

- Note the equations:

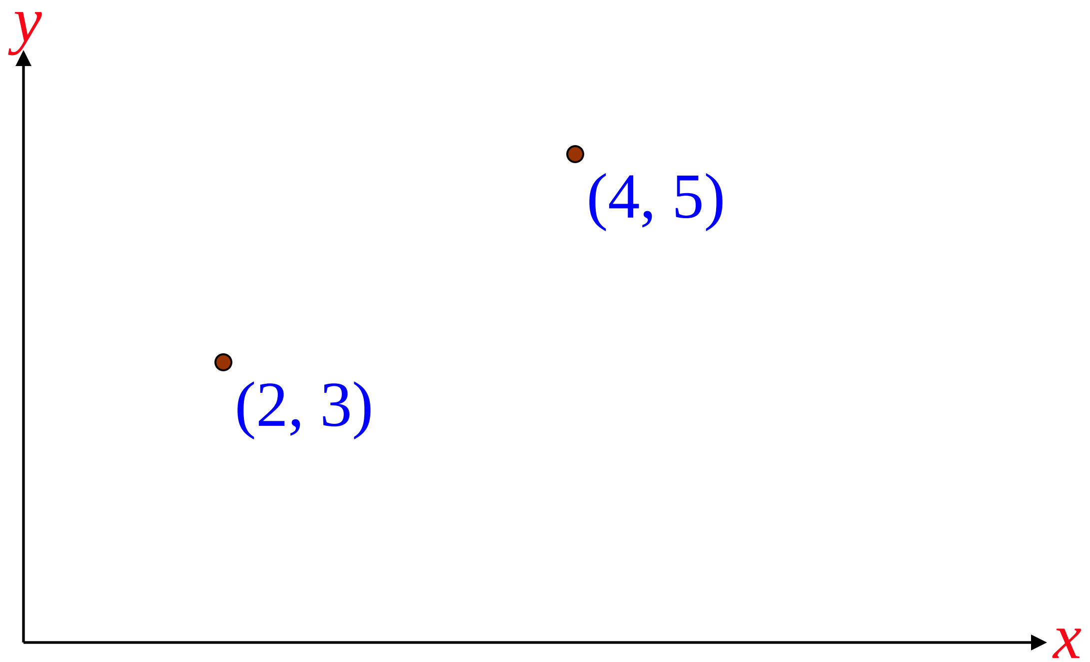
$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i) + b) - 1 \} = 0$$

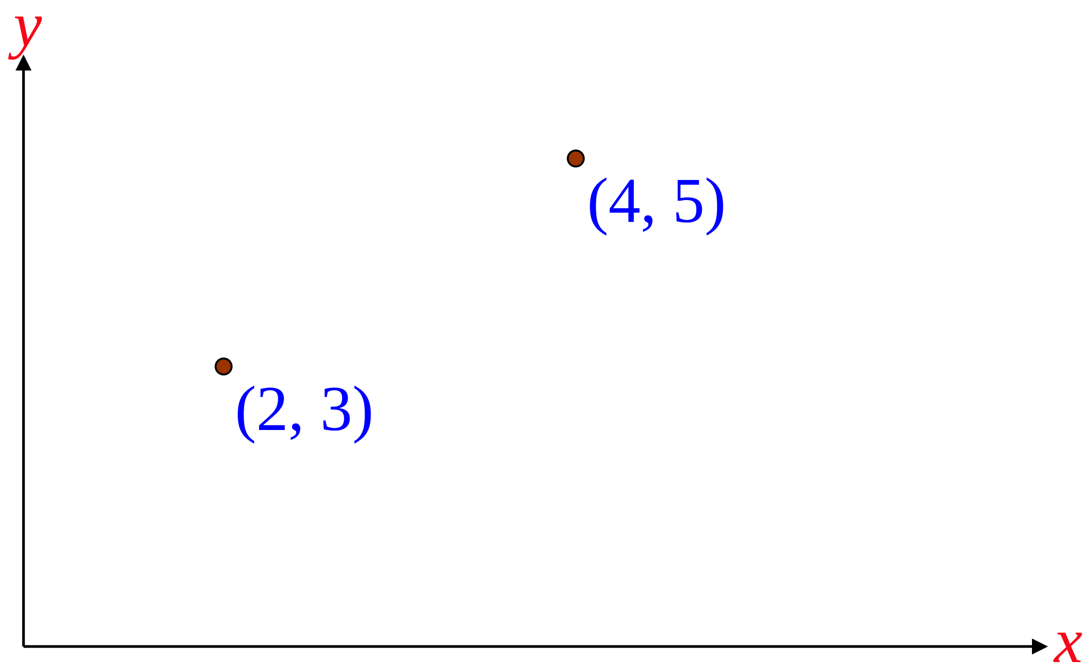
$$f(\vec{z}) = sign(\sum_i \lambda_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{z}) + b)$$

- The dot product $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ is a similarity measurement

Similarity/Distance Measurement



Similarity/Distance Measurement



$$\text{distance} = \left[(2-4)^2 + (3-5)^2\right]^{1/2}$$

$$\text{Cosine Simmilarit y} = \frac{2.4 + 3.5}{\sqrt{(2^2 + 3^2)}\sqrt{(4^2 + 5^2)}}$$

Issues in the Design Nonlinear SVM

- We can calculate the term $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ in the original space using Kernel trick

Issues in the Design Nonlinear SVM

- We can calculate the term $\Phi(\vec{x}_j)\cdot\Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u})\cdot\Phi(\vec{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1)$$

$$\cdot(v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1)$$

$\cdot$

$\cdot$

$$= (\vec{u}\cdot\vec{v}+1)^2$$

Issues in the Design Nonlinear SVM

- Kernel trick

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (\vec{u} \cdot \vec{v} + 1)^2$$

Issues in the Design Nonlinear SVM

- Kernel trick

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (\vec{u} \cdot \vec{v} + 1)^2$$

$$K(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v} + 1)^2$$

Issues in the Design Nonlinear SVM

- Some Kernel Functions are:

$$K(\vec{u},\vec{v}) = (\vec{u} \cdot \vec{v} + 1)^p$$

$$K(\vec{u},\vec{v}) = e^{-\|\vec{u}-\vec{v}\|^2/(2\sigma^2)}$$