CS344: Introduction to Artificial Intelligence (associated lab: CS386)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Lecture 31: Feedforward N/W; sigmoid neuron

28th March, 2011

Feedforward Network

Limitations of perceptron

- Non-linear separability is all pervading
- Single perceptron does not have enough computing power
- Eg: XOR cannot be computed by perceptron

Solutions

- Tolerate error (Ex: *pocket algorithm* used by connectionist expert systems).
  - Try to get the best possible hyperplane using only perceptrons
- Use higher dimension surfaces

  Ex: Degree - 2 surfaces like parabola
- Use layered network

Pocket Algorithm

- Algorithm evolved in 1985 – essentially uses PTA
- Basic Idea:
  - Always preserve the best weight obtained so far in the “pocket”
  - Change weights, if found better (i.e. changed weights result in reduced error).

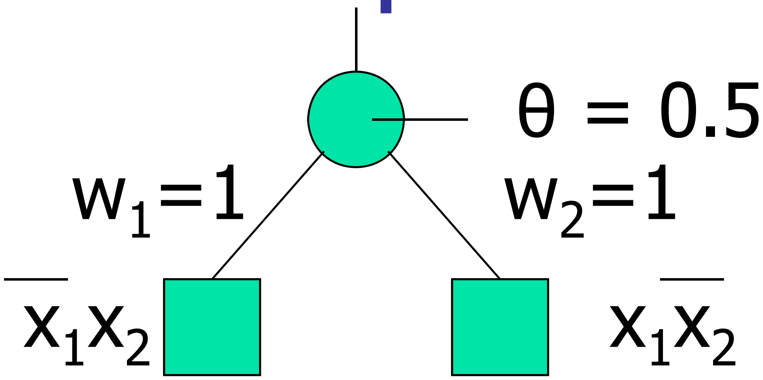
XOR using 2 layers

$$x_1 \oplus x_2 = \left(x_1 \overline{x_2}\right) \vee \left(\overline{x_1} x_2\right)$$

$$= OR(AND(x_1, NOT(x_2)), AND(NOT(x_1), x_2)))$$

- Non-LS function expressed as a linearly separable function of individual linearly separable functions.

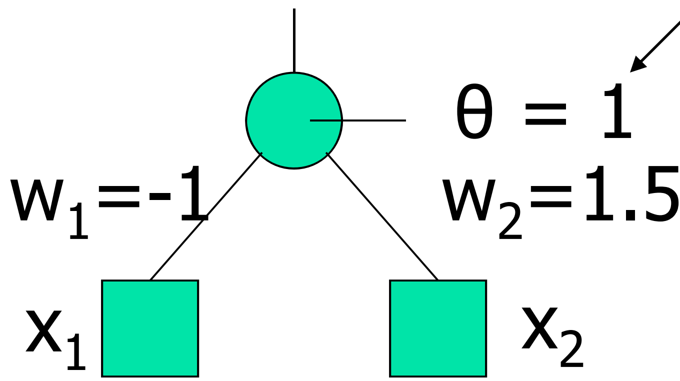
Example - XOR



← Calculation of XOR

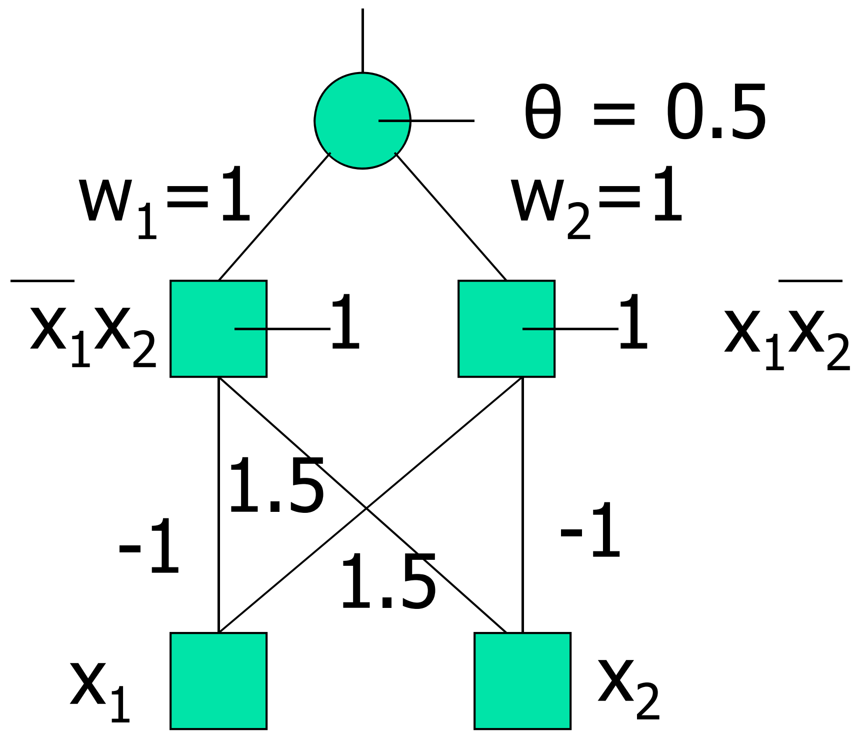
x_1	x_2	$x_1\bar{x}_2$
0	0	0
0	1	1
1	0	0
1	1	0

Calculation of $x_1\bar{x}_2$



$$0 < \Theta$$
$$w2 \geq \Theta$$
$$w1 < \Theta$$
$$w1 + w2 < \Theta$$

Example - XOR



Some Terminology

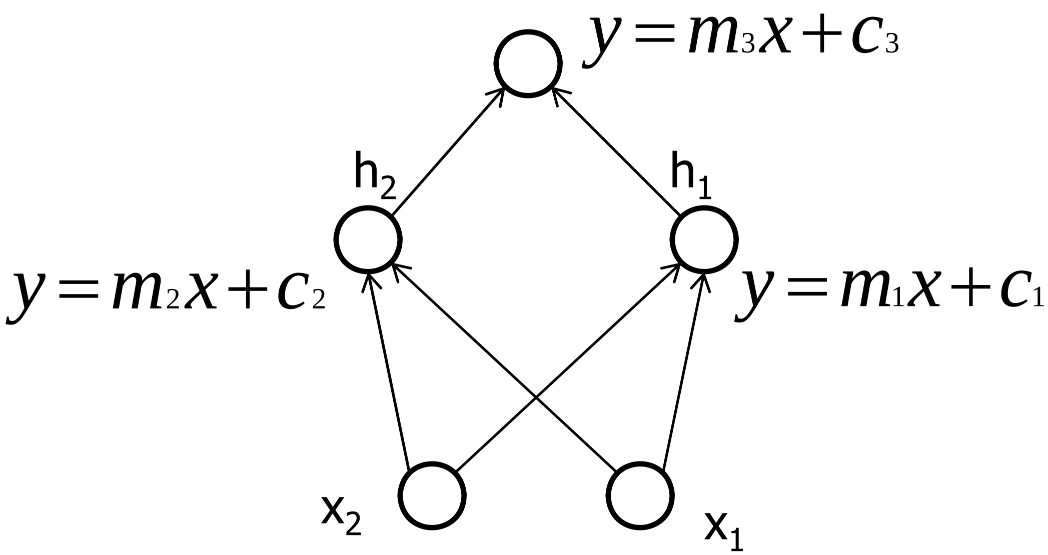
- A multilayer feedforward neural network has
  - Input layer
  - Output layer
  - Hidden layer (assists computation)

Output units and hidden units are called computation units.

Training of the MLP

- Multilayer Perceptron (MLP)
- Question:- How to find weights for the hidden layers when no target output is available?
- Credit assignment problem – to be solved by "*Gradient Descent*"

Can Linear Neurons Work?



$$h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1$$

$$h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1$$

$$Out = (w_5 h_1 + w_6 h_2) + c_3$$
$$= k_1 x_1 + k_2 x_2 + k_3$$

Note: The whole structure shown in earlier slide is reducible to a single neuron with given behavior

$$Out = k_1 x_1 + k_2 x_2 + k_3$$

Claim: A neuron with linear I-O behavior can't compute X-OR.

Proof: Considering all possible cases:

[assuming 0.1 and 0.9 as the lower and upper thresholds]

For (0,0), Zero class:

$$m(w_1.0 + w_2.0 - \theta) + c < 0.1$$
$$\Rightarrow c - m.\theta < 0.1$$

For (0,1), One class:

$$m(w_2.1 + w_1.0 - \theta) + c > 0.9$$
$$\Rightarrow m.w_1 - m.\theta + c > 0.9$$

For (1,0), One class: $m.w_1 - m.\theta + c > 0.9$

For (1,1), Zero class: $m.w_1 - m.\theta + c > 0.9$

These equations are inconsistent. Hence X-OR can't be computed.

Observations:

1. A linear neuron can't compute X-OR.
2. A multilayer FFN with linear neurons is collapsible to a single linear neuron, hence **no a additional power due to hidden layer.**
3. Non-linearity is essential for power.

Multilayer Perceptron

Training of the MLP

- Multilayer Perceptron (MLP)

- Question:- How to find weights for the hidden layers when no target output is available?

- Credit assignment problem – to be solved by "*Gradient Descent*"

Gradient Descent Technique

- Let E be the error at the output layer

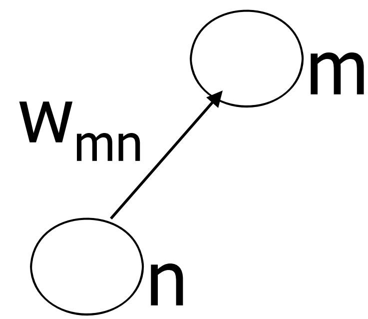
$$E = \frac{1}{2} \sum_{j=1}^{p} \sum_{i=1}^{n} (t_i - o_i)_j^2$$

- t_i = target output; o_i = observed output

- i is the index going over n neurons in the outermost layer
- j is the index going over the p patterns (1 to p)
- Ex: XOR:– p=4 and n=1

Weights in a FF NN

- w_{mn} is the weight of the connection from the n^{th} neuron to the m^{th} neuron
- E *vs* $\overline{W}$ surface is a complex surface in the space defined by the weights w_{ij}
- $-\frac{\delta E}{\delta w_{mn}}$ gives the direction in which a movement of the operating point in the w_{mn} co-ordinate space will result in maximum decrease in error



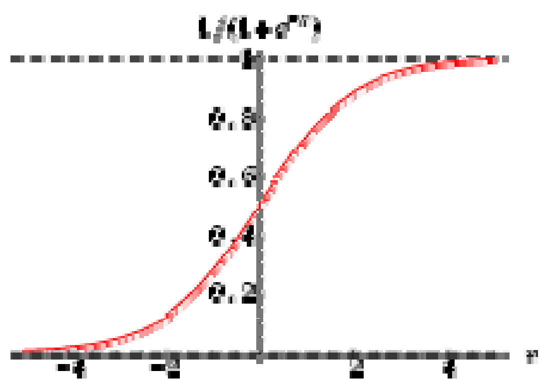
$$\Delta w_{mn} \propto -\frac{\delta E}{\delta w_{mn}}$$

Sigmoid neurons

- Gradient Descent needs a derivative computation

  - not possible in perceptron due to the discontinuous step function used!

  → Sigmoid neurons with easy-to-compute derivatives used!



$$y \rightarrow 1 \text{ as } x \rightarrow \infty$$

$$y \rightarrow 0 \text{ as } x \rightarrow -\infty$$

- Computing power comes from non-linearity of sigmoid function.

Derivative of Sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = y(1-y)$$

Training algorithm

- Initialize weights to random values.
- For input x = <$x_n, x_{n-1}, \ldots, x_0$>, modify weights as follows

  Target output = t, Observed output = o

$$\Delta w_i \propto -\frac{\delta E}{\delta w_i}$$

$$E = \frac{1}{2}(t - o)^2$$

- Iterate until E < δ (threshold)

Calculation of Δw_i

$$\frac{\delta E}{\delta w_i} = \frac{\delta E}{\delta net} \times \frac{\delta net}{\delta w_i} \left(where \ : net = \sum_{i=0}^{n-1} w_i x_i \right)$$

$$= \frac{\delta E}{\delta o} \times \frac{\delta o}{\delta net} \times \frac{\delta net}{\delta w_i}$$

$$= -(t-o)o(1-o)x_i$$

$$\Delta w_i = -\eta \frac{\delta E}{\delta w_i} (\eta = \text{learning constant}, \ \ 0 \le \eta \le 1)$$

$$\Delta w_i = \eta (t-o)o(1-o)x_i$$

Observations

Does the training technique support our intuition?

- The larger the x_i, larger is Δw_i
  - Error burden is borne by the weight values corresponding to large input values