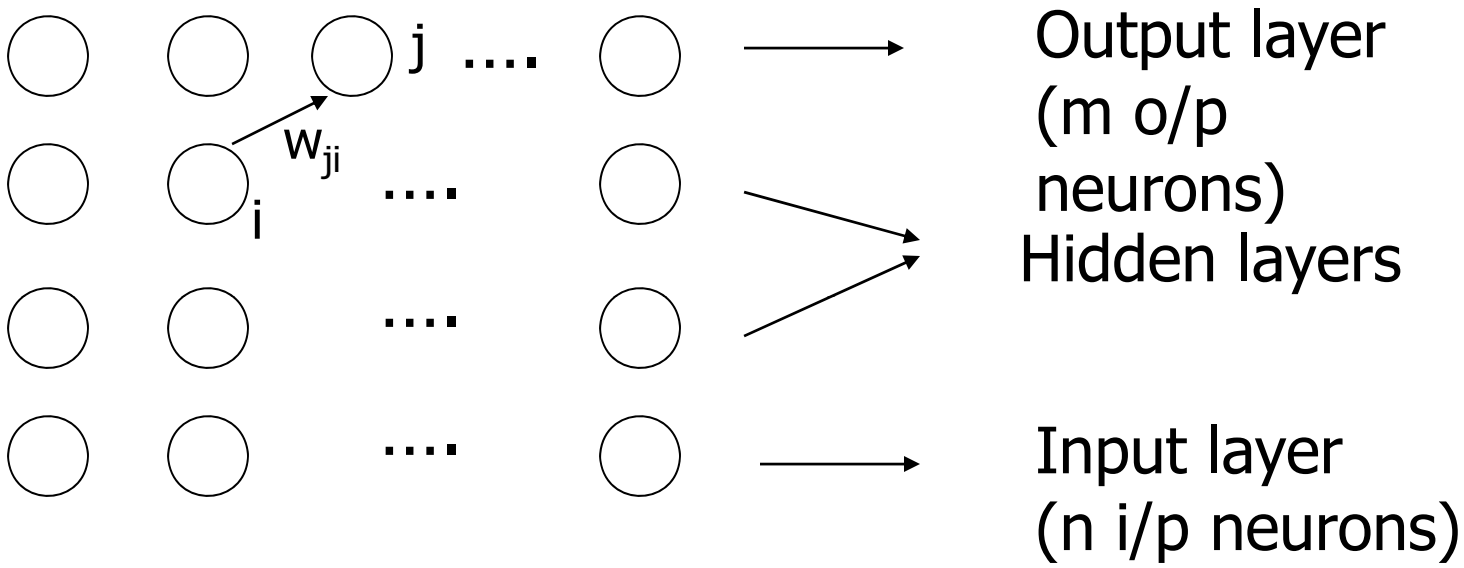
CS344: Introduction to Artificial Intelligence (associated lab: CS386)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Lecture 34: Backpropagation; need for multiple layers and non linearity

5th April, 2011

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)

Gradient Descent Equations

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} (\eta = \text{learning rate}, 0 \le \eta \le 1)$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} (net_j = \text{input at the j}^{th} \text{ layer})$$

$$\frac{\delta E}{\delta net_j} = -\delta j$$

$$\Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i$$

Backpropagation – for outermost layer

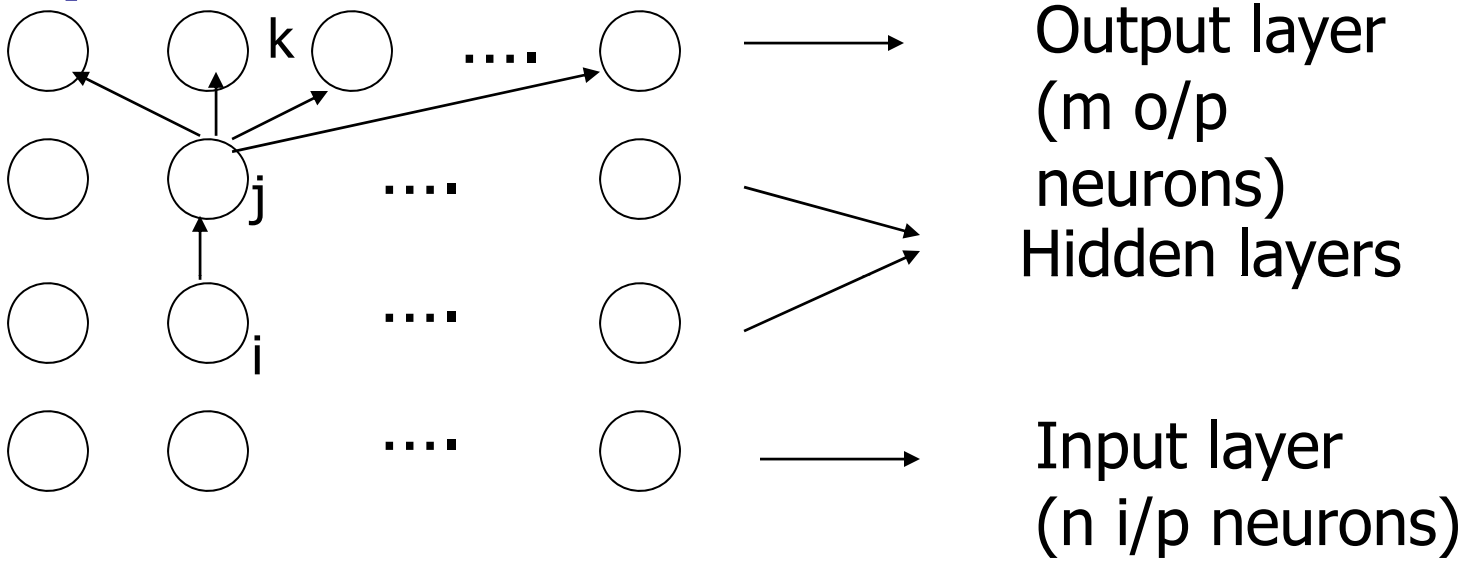
$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} \quad (net_j = \text{input at the j}^{th} \text{ layer})$$

$$E = \frac{1}{2}\sum_{p=1}^{m}(t_p - o_p)^2$$

Hence, $\delta j = -(-(t_j - o_j)o_j(1-o_j))$

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)o_i$$

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j}$$

$$= -\frac{\delta E}{\delta o_j} \times o_j (1 - o_j)$$

$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times o_j (1 - o_j)$$

$$\text{Hence, } \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times o_j (1 - o_j)$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j)$$

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \quad \text{for hidden layers}$$

Observations on weight change rules

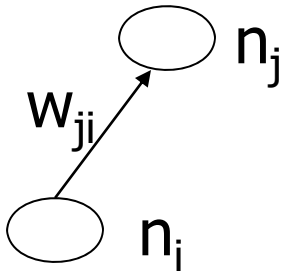
Does the training technique support our intuition?

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

Observations contd.

- Δw_i is proportional to the departure from target
- Saturation behaviour when o is 0 or 1
- If $o < t$, $\Delta w_i > 0$ and if $o > t$, $\Delta w_i < 0$ which is consistent with the Hebb's law

Hebb's law



- If n_j and n_i are both in excitatory state (+1)
  - Then the change in weight must be such that it enhances the excitation
  - The change is proportional to both the levels of excitation

$$\Delta w_{ji} \alpha \; e(n_j)\, e(n_i)$$

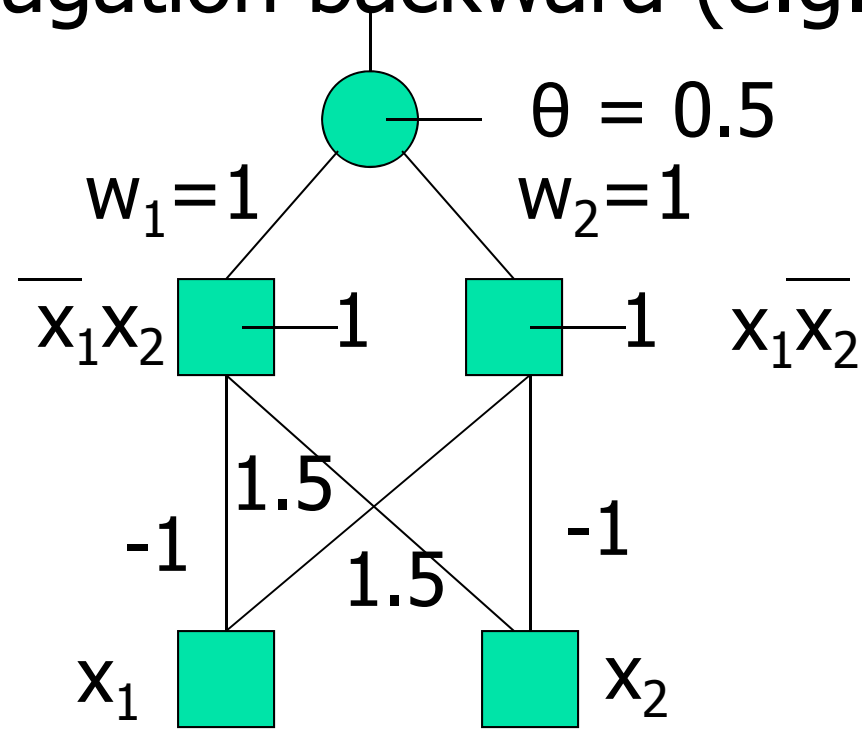
- If n_i and n_j are in a mutual state of inhibition (one is +1 and the other is -1),
  - Then the change in weight is such that the inhibition is enhanced (change in weight is negative)

Saturation behavior

- The algorithm is iterative and incremental
- If the weight values or number of input values is very large, the output will be large, then the output will be in saturation region.
- The weight values hardly change in the saturation region

How does it work?

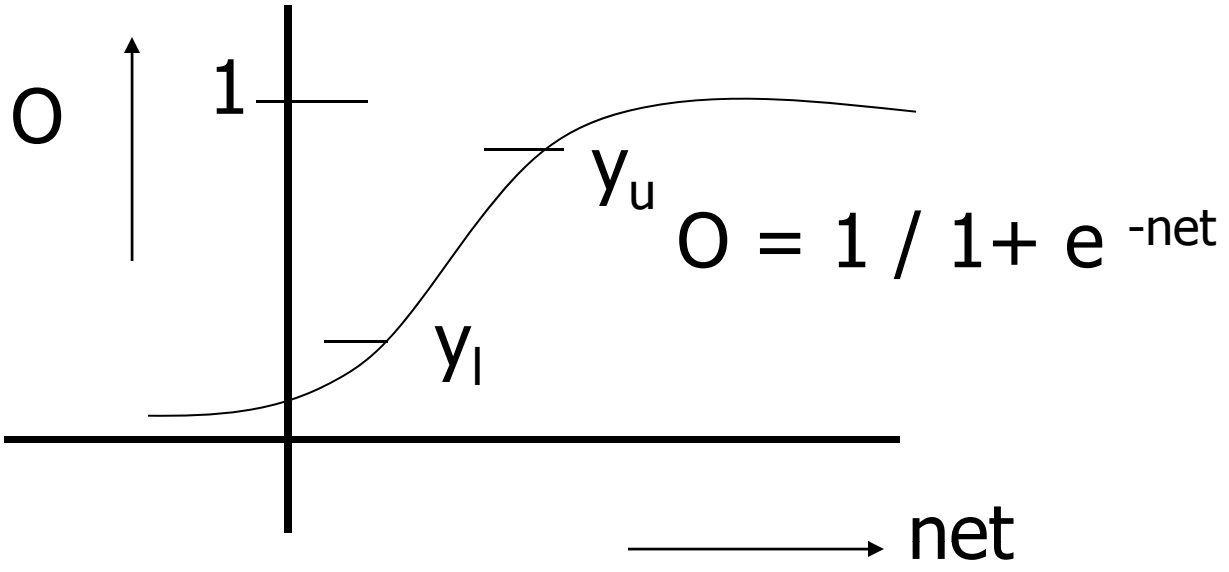
- Input propagation forward and error propagation backward (e.g. XOR)



If Sigmoid Neurons Are Used, Do We Need MLP?

Does sigmoid have the power of separating non-linearly separable data?

Can sigmoid solve the X-OR problem



$$O = 1 / 1 + e^{-net}$$

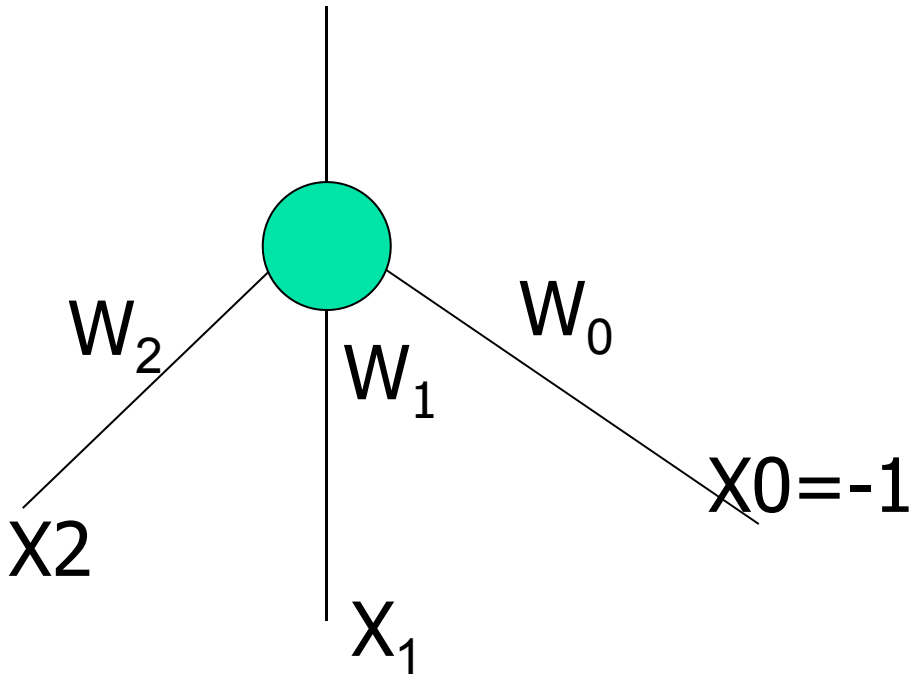
$$O = 1 \text{ if } O > y_u$$

$$O = 0 \text{ if } O < y_l$$

Typically $y_l << 0.5$, $y_u >> 0.5$

Inequalities

$$O = 1 / (1 + e^{-net})$$



<0, 0>

O = 0

i.e $0 < y_l$

$$1 / 1 + e^{(-w_1 x_1 - w_2 x_2 + w_0)} < y_l$$

i.e. $$(1 / (1 + e^{w_o})) < y_l \qquad (1)$$

<0, 1>

$O = 1$

i.e. $0 > y_u$

$$1/(1+ e^{(-w_1 x_1 - w_2 x_2 + w_0)}) > y_u$$

$$(1 / (1+ e^{-w_2 + w_0})) > y_u \qquad (2)$$

<1, 0>

O = 1

i.e. $(1/1+ e^{-w_1+w_0}) > y_u$ (3)

<1, 1>

O = 0

i.e. $1/(1+ e^{-w_1-w_2+w_0}) < y_l$ (4)

Rearranging, 1 gives

$$1/(1+ e^{w_o}) < y_l$$

$$\text{i.e.} \quad 1+ e^{w_o} > 1 / y_l$$

$$\text{i.e.} \quad W_o > \ln ((1- y_l) / y_l) \qquad (5)$$

2 Gives

$$1/1+ e^{-w_2+w_0} > y_u$$

$$\text{i.e.} \quad 1+ e^{-w_2+w_0} < 1 / y_u$$

$$\text{i.e.} \quad e^{-w_2+w_0} < 1-y_u / y_u$$

$$\text{i.e.} \quad -W_2 + W_o < \ln (1-y_u) / y_u$$

$$\text{i.e.} \quad W_2 - W_o > \ln (y_u / (1-y_u)) \qquad (6)$$

3 Gives

$$W_1 - W_o > \ln (y_u / (1- y_u)) \qquad (7)$$

4 Gives

$$-W_1 - W_2 + W_o > \ln ((1- y_l)/ y_l) \qquad (8)$$

5 + 6 + 7 + 8 Gives

$$0 > 2\ln (1- y_l)/ y_l + 2 \ln y_u / (1 - y_u)$$

$$\text{i.e. } 0 > \ln [(1- y_l)/ y_l * y_u / (1 - y_u)]$$

$$\text{i.e. } ((1- y_l) / y_l) * (y_u / (1 - y_u)) < 1$$

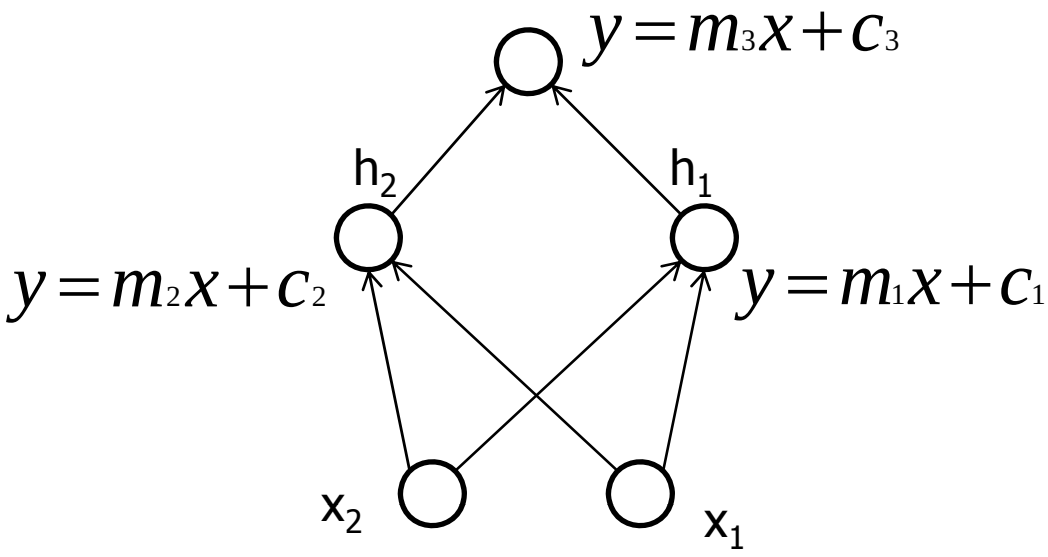
i. $[(1- y_l) / (1- y_y)] * [y_u / y_l] < 1$

ii. 2) $Y_u >> 0.5$

iii. 3) $Y_l << 0.5$

From i, ii and iii; Contradiction, hence sigmoid cannot compute X-OR

Can Linear Neurons Work?



$$h_1 = m_1(w_1x_1 + w_2x_2) + c_1$$

$$h_1 = m_1(w_1x_1 + w_2x_2) + c_1$$

$$Out = (w_5h_1 + w_6h_2) + c_3$$

$$= k_1x_1 + k_2x_2 + k_3$$

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.