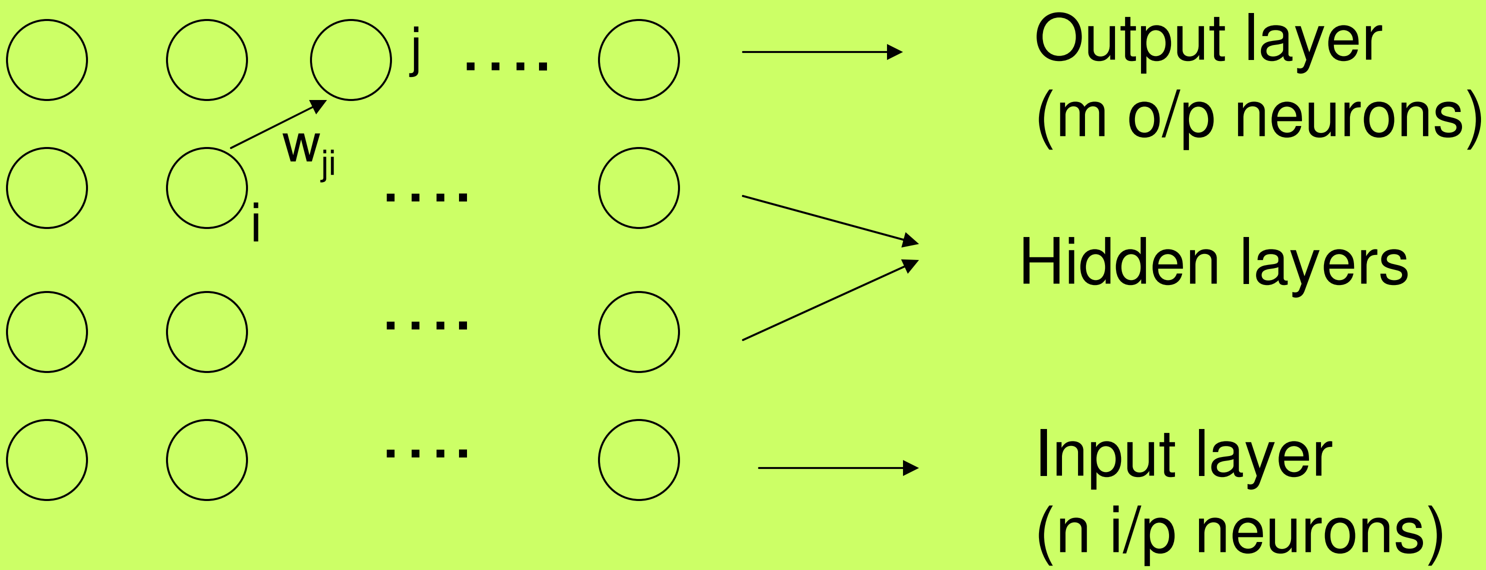
CS623: Introduction to Computing with Neural Nets *(lecture-5)*

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Backpropagation algorithm



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

Gradient Descent Equations

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

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

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

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

Backpropagation – for outermost layer

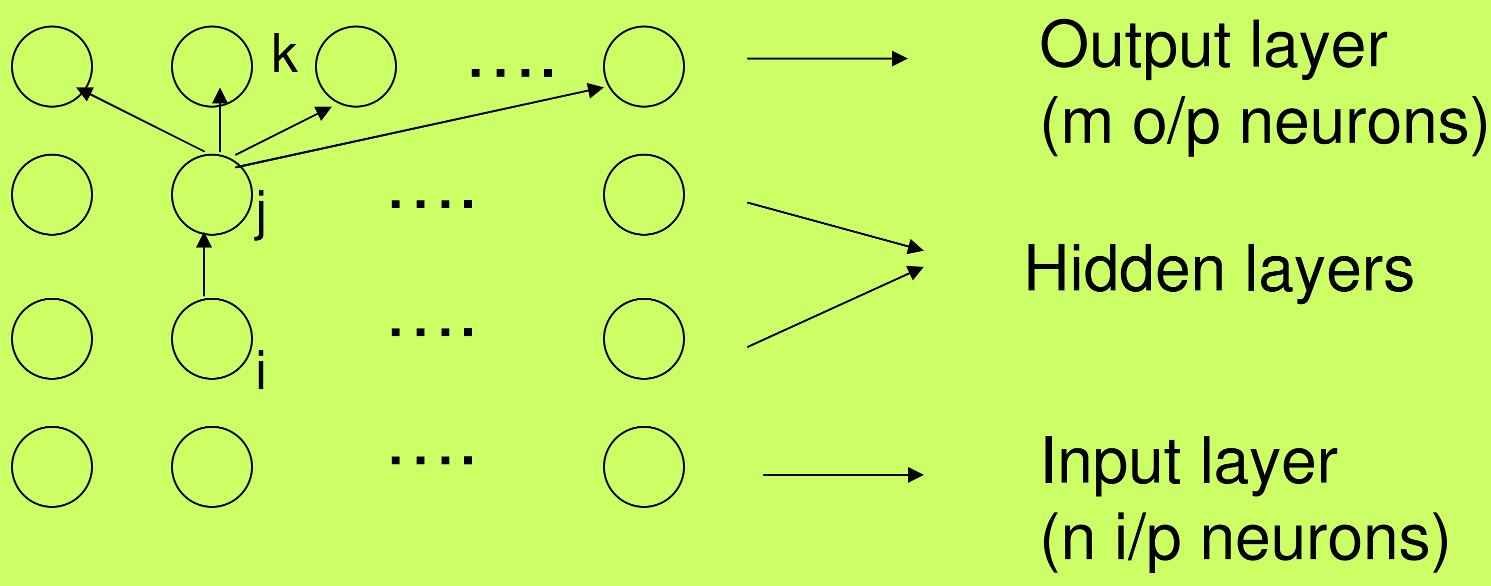
$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the j}^{th} \text{ layer})$$

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

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

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

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

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

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

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

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

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

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

General Backpropagation Rule

- General weight updating rule:

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

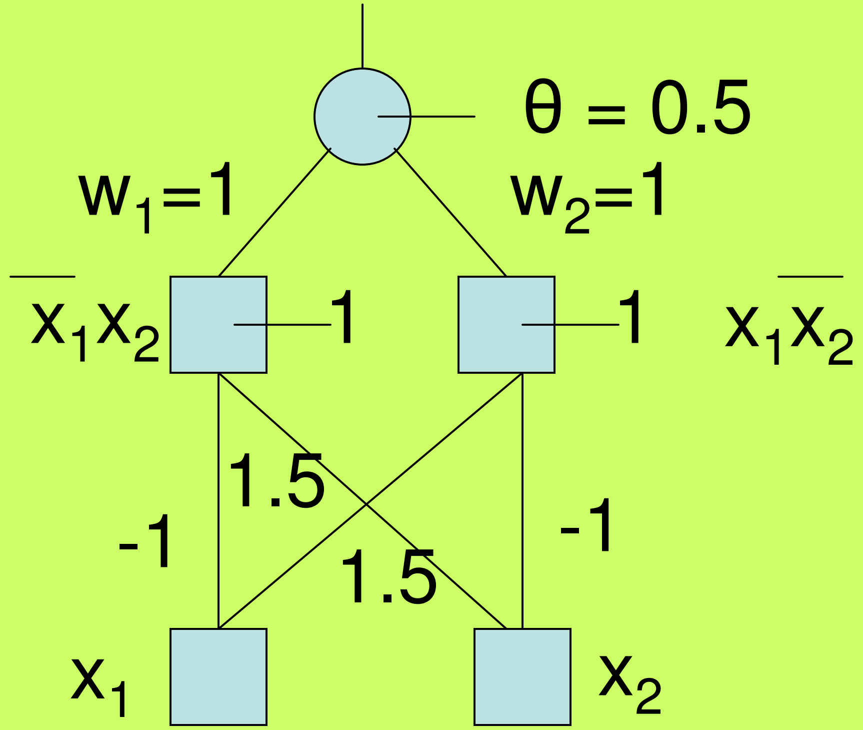
- Where

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

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

How does it work?

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

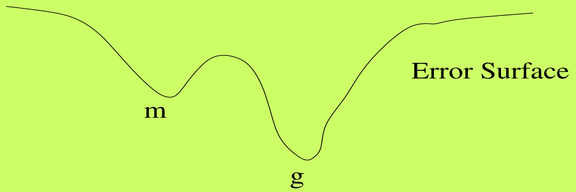


Issues in the training algorithm

- Algorithm is greedy. It always changes weight such that E reduces.
- The algorithm may get stuck up in a local minimum.

Local Minima

Due to the Greedy nature of BP, it can get stuck in local minimum *m* and will never be able to reach the global minimum *g* as the error can only decrease by weight change.



m- local minima, g- global minima

Figure- Getting Stuck in local minimum

Reasons for *no progress* in training

1. Stuck in local minimum.
2. Network paralysis. (High –ve or +ve i/p makes neurons to saturate.)
3. η (learning rate) is too small.

Diagnostics in action (1)

1) If stuck in local minimum, try the following:
   - Re-initializing the weight vector.
   - Increase the learning rate.
   - Introduce more neurons in the hidden layer.

Diagnostics in action (1) contd.

2) If it is network paralysis, then increase the number of neurons in the hidden layer.

- Problem: How to configure the hidden layer ?
- Known: One hidden layer seems to be sufficient. [Kolmogorov (1960’s)]

Diagnostics in action(2)

Kolgomorov statement:

A feedforward network with three layers (input, output and hidden) with appropriate I/O relation that can vary from neuron to neuron is sufficient to compute any function.

- More hidden layers reduce the size of individual layers.

Diagnostics in action(3)

3) Observe the outputs: If they are close to 0 or 1, try the following:

1. Scale the inputs or divide by a normalizing factor.
2. Change the shape and size of the sigmoid.

Answers to Quiz-1

- Q1: Show that of the 256 Boolean functions of 3 variables, only half are computable by a threshold perceptron
- Ans: The characteristic equation for 3 variables is

$$W_1X_1+W_2X_2+W_3X_3= \theta \qquad (E)$$

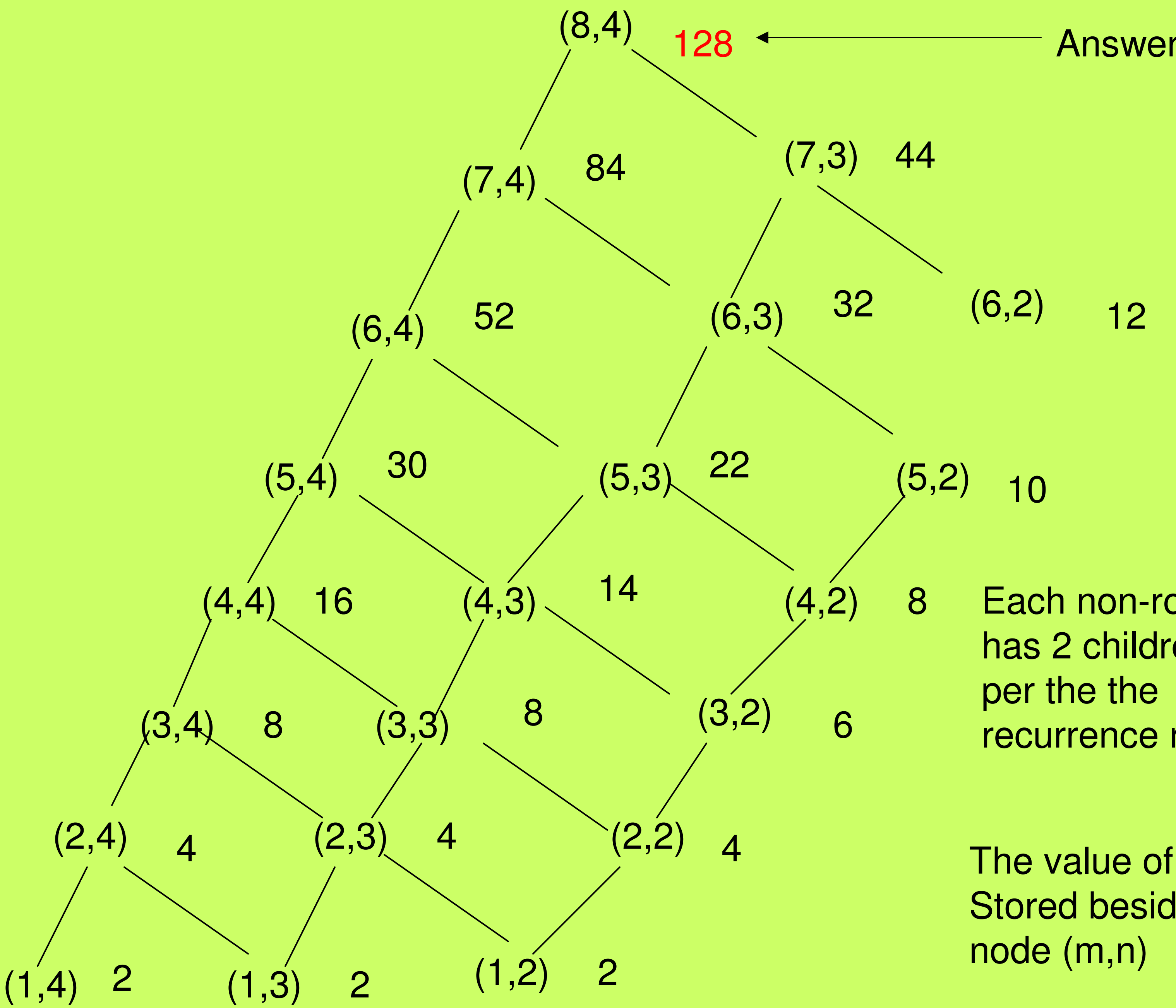
The 8 Boolean value combinations when inserted in (E) will produce 8 hyperplanes passing through the origin in the $< W_1, W_2, W_3, \theta>$ space.

Q1 *(contd)*

The maximum number of function computable by this perceptron is the number of regions produced by the intersection of these 8 planes in the 4 dimensional space

$R_{8,4}= R_{7,4} + R_{7,3}$ (1)

$R_{1,4}= 2$ and $R_{m,2}= 2m$, for m= 1,4 (boundary condition)

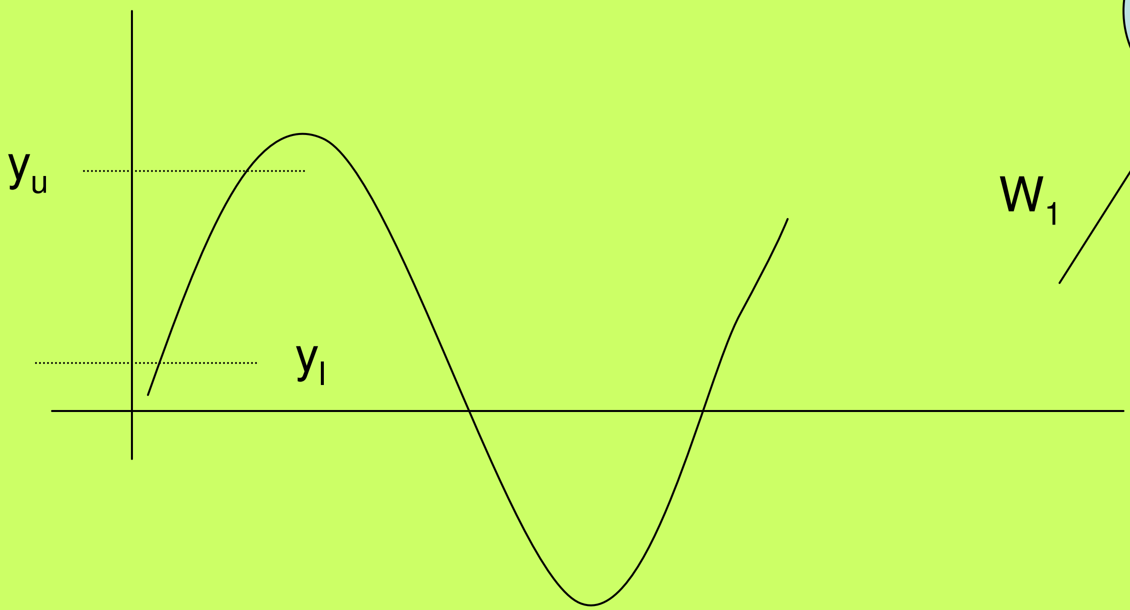


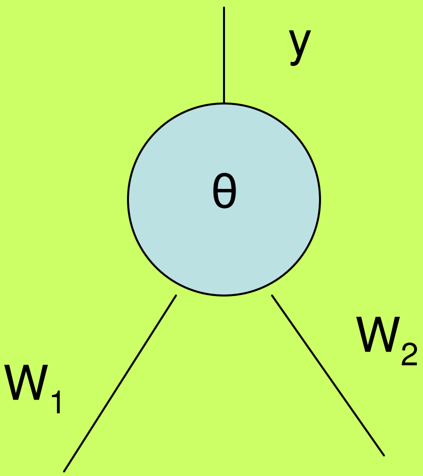
Each non-root node has 2 children as per the the recurrence relation.

The value of $R_{m,n}$ is Stored beside the node (m,n)

Answer to Quiz1 *(contd)*

- Q2. Prove if a perceptron with *sin(x)* as i-o relation can compute *X-OR*
- Ans:





Q2 *(contd)*

Input <0,0>: $y < y_l$

$$sin(\theta) < y_l \qquad (1)$$

Input <0,1>: $y > y_u$

$$sin(W_1+\theta) > y_u \qquad (2)$$

Input <1,0>: $y > y_u$

$$sin(W_2+\theta) > y_u \qquad (3)$$

Input <1,1>: $y < y_l$

$$sin(W_1+W_2+\theta) < y_l \qquad (4)$$

Q2 *(contd)*

Taking

y_l =0.1, y_u= 0.9

W_1=(Π/2)=W_2

θ= 0

We can see that the perceptron can compute X-OR

Answer to Quiz-1 *(contd)*

- Q3: If in the perceptron training algorithm, the failed vector is again chosen by the algorithm, will there be any problem?
- Ans:

In PTA,

$$W_n = W_{n-1} + X_{fail}$$

After this, X_{fail} is chosen again for testing and is added if fails again. This continues until $W_k.X_{fail} > 0$. Will this terminate?

Q3 *(contd)*

It will, because:

$W_n = W_{n-1} + X_{fail}$

$W_{n-1} = W_{n-2} + X_{fail}$

.

.

.

$W_n = W_0 + n.X_{fail}$

Therefore, $W_n.X_{fail} = W_0.X_{fail} + n.(X_{fail})^2$

-δ

Positive, growing with n.
Will overtake – δ after some iterations.
Hence "no problem" is the answer.