# CS772: Deep Learning for Natural Language Processing (DL-NLP)

## Backpropagation

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

*Week 4 of 18th Aug, 2025*

# 1-slide recap, Lecture 2

- Perceptron: what it is, linear inequality based solution, linear separability, training algo, convergence
- Sigmoid: nature of the function, derivative, probability computer
- Softmax: generalizes sigmoid, derivatives, gives a probability distribution
- POS assignment: compare and contrast HMM, EnCo-DeCo and LLMs

# Finding weight change rule

# Foundation: **Gradient descent**

Change is weight $\Delta w_{ji} = -\eta \delta E / \delta w_{ji}$

$\eta$= learning rate,
E=loss, $w_{ji}$= weight of connection from the $i^{th}$ neuron to $j^{th}$

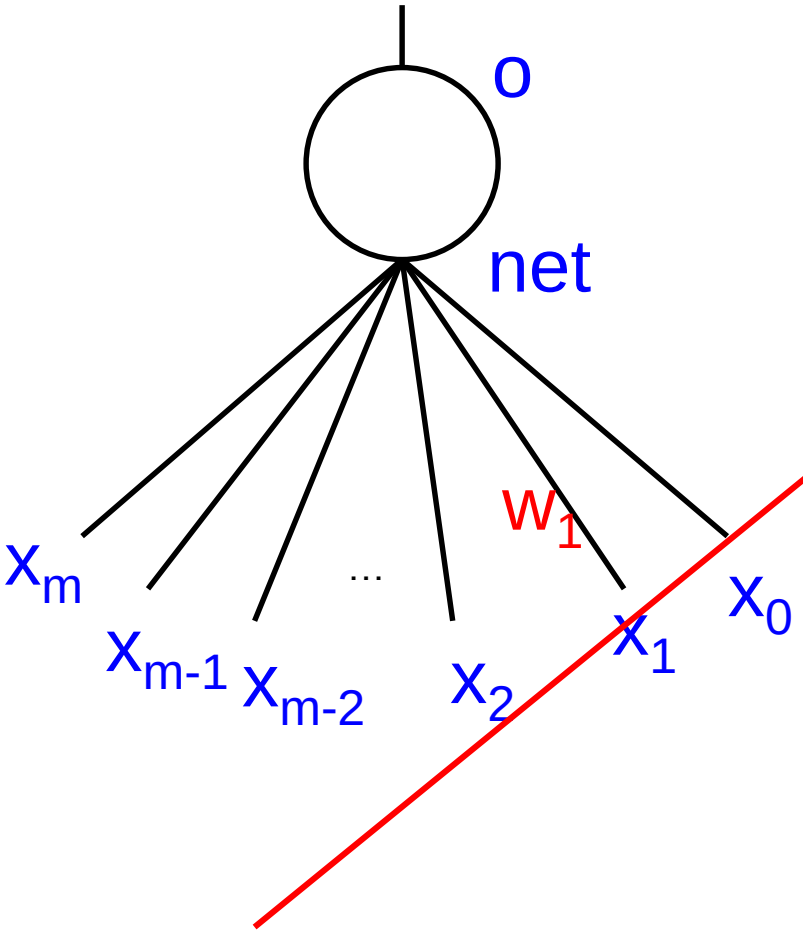
At A, $\delta E / \delta w_{ji}$ is negative, so $\Delta w_{ji}$ is positive.

At B, $\delta E / \delta w_{ji}$ is positive,

so $\Delta w_{ji}$ is negative.

E always decreases. Greedy algo.

# Weight change: Single sigmoid neuron and *cross entropy* loss suffix *i*




$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial o} . \frac{\partial o}{\partial net} . \frac{\partial net}{\partial w_1}$$

$$E = -t \log o - (1-t) \log(1-o)$$

$$\Rightarrow \frac{\partial E}{\partial o} = -\frac{t}{o} + \frac{1-t}{1-o} = -\frac{t-o}{o(1-o)}$$

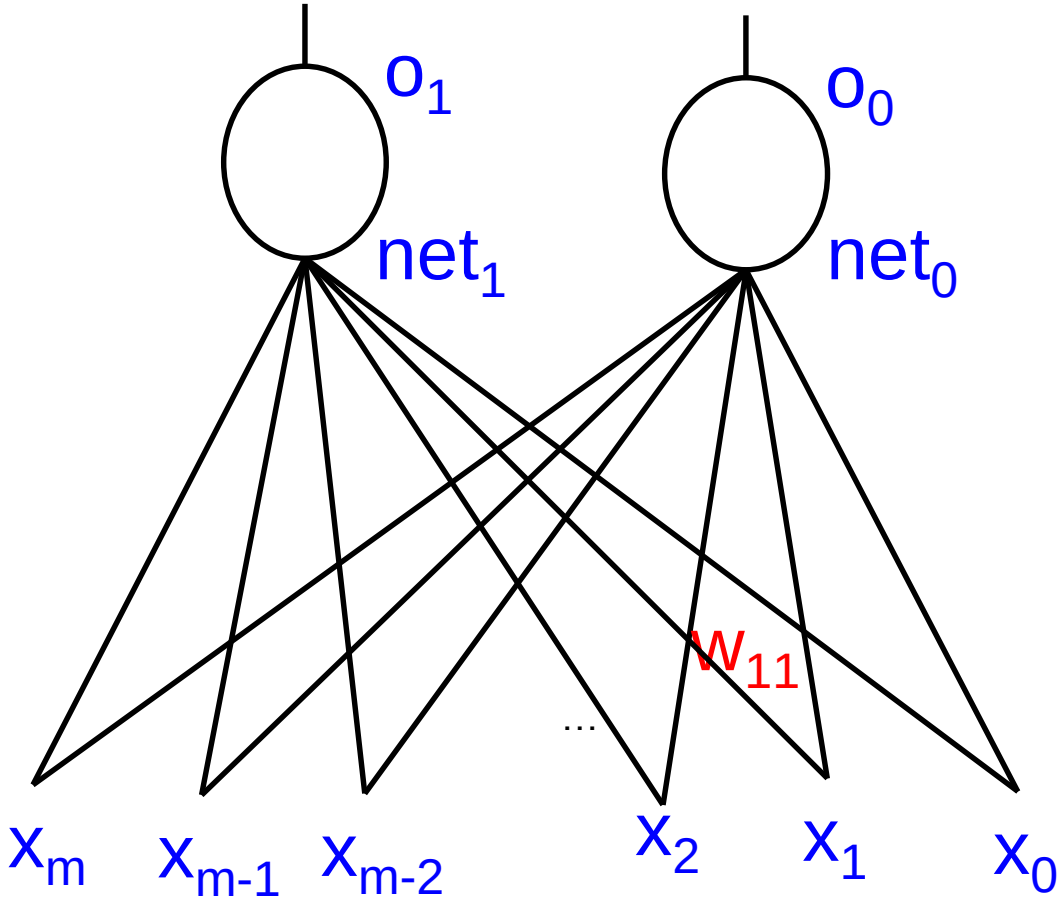
$$o = \frac{1}{1+e^{-net}} (sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o)$$

$$net = \sum_{j=0}^{m} w_j x_j \Rightarrow \frac{\partial net}{\partial w_1} = x_1$$

$$\Rightarrow \Delta w_1 = -\eta \frac{\partial E}{\partial w_1} = \eta (t-o) x_1$$

$$\Delta w_1 = \eta (t-o) x_1$$

# Multiple neurons in the output layer: softmax+*cross entropy* loss (1/2): illustrated with 2 neurons and single training data point




$$O =< o_1, o_0 >$$

$$NET =< net_1, net_0 >$$

$$o_1 = \frac{e^{net_1}}{e^{net_1} + e^{net_0}},\ o_0 = \frac{e^{net_0}}{e^{net_1} + e^{net_0}}$$

$$\frac{\partial O}{\partial NET} = \begin{bmatrix} \frac{\partial o_0}{\partial net_0} & \frac{\partial o_1}{\partial net_0} \\ \frac{\partial o_0}{\partial net_1} & \frac{\partial o_1}{\partial net_1} \end{bmatrix}$$

$$= \begin{bmatrix} o_0(1-o_0) & -o_0 o_1 \\ -o_1 o_0 & o_1(1-o_1) \end{bmatrix}$$

# Softmax and Cross Entropy (2/2)

$$E = -t_1 \log o_1 - t_0 \log o_0$$

$$o_1 = \frac{e^{net_1}}{e^{net_1} + e^{net_0}},\ o_0 = \frac{e^{net_0}}{e^{net_1} + e^{net_0}}$$

$$\frac{\partial E}{\partial w_{11}} = -\frac{t_1}{o_1}\frac{\partial o_1}{\partial w_{11}} - \frac{t_0}{o_0}\frac{\partial o_0}{\partial w_{11}}$$

$$\frac{\partial o_1}{\partial w_{11}} = \frac{\partial o_1}{\partial net_1}.\frac{\partial net_1}{\partial w_{11}} + \frac{\partial o_1}{\partial net_0}.\frac{\partial net_0}{\partial w_{11}} = o_1(1-o_1)x_1 + 0$$

$$\frac{\partial o_0}{\partial w_{11}} = \frac{\partial o_0}{\partial net_1}.\frac{\partial net_1}{\partial w_{11}} + \frac{\partial o_0}{\partial net_0}.\frac{\partial net_0}{\partial w_{11}} = -o_1 o_0 x_1 + 0$$

$$\Rightarrow \frac{\partial E}{\partial w_{11}} = -t_1(1-o_1)x_1 + t_0 o_1 x_1 = -t_1(1-o_1)x_1 + (1-t_1)o_1 x_1$$

$$= [-t_1 + t_1 o_1 + o_1 - t_1 o_1]x_1 = -(t_1 - o_1)x_1$$

$$\Delta w_{11} = -\eta \frac{\partial E}{\partial w_{11}} = \eta(t_1 - o_1)x_1$$

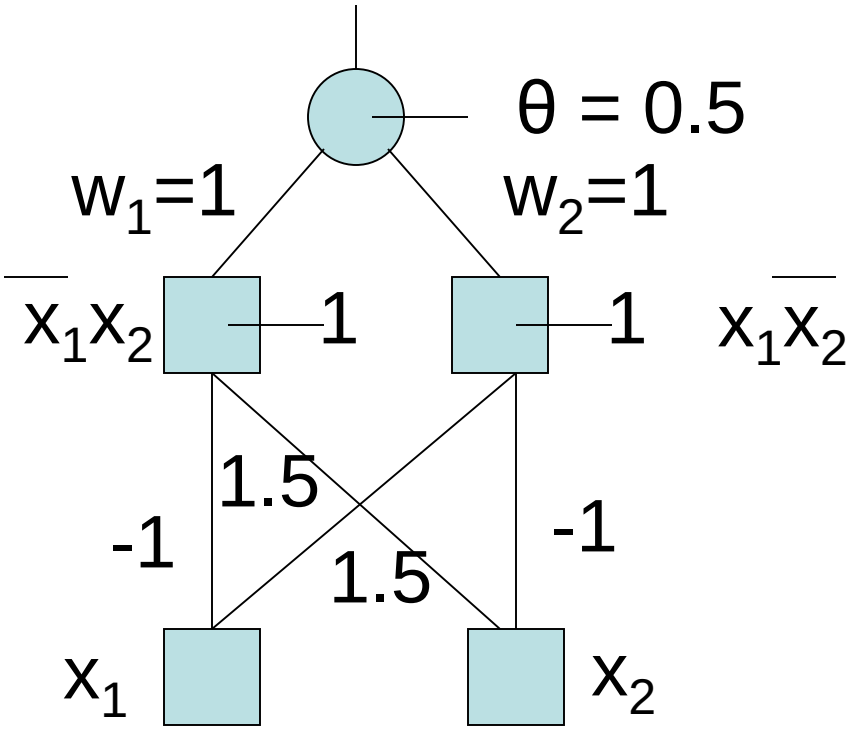
# Can be generalized

- When E is Cross Entropy Loss
- The change in any weight is

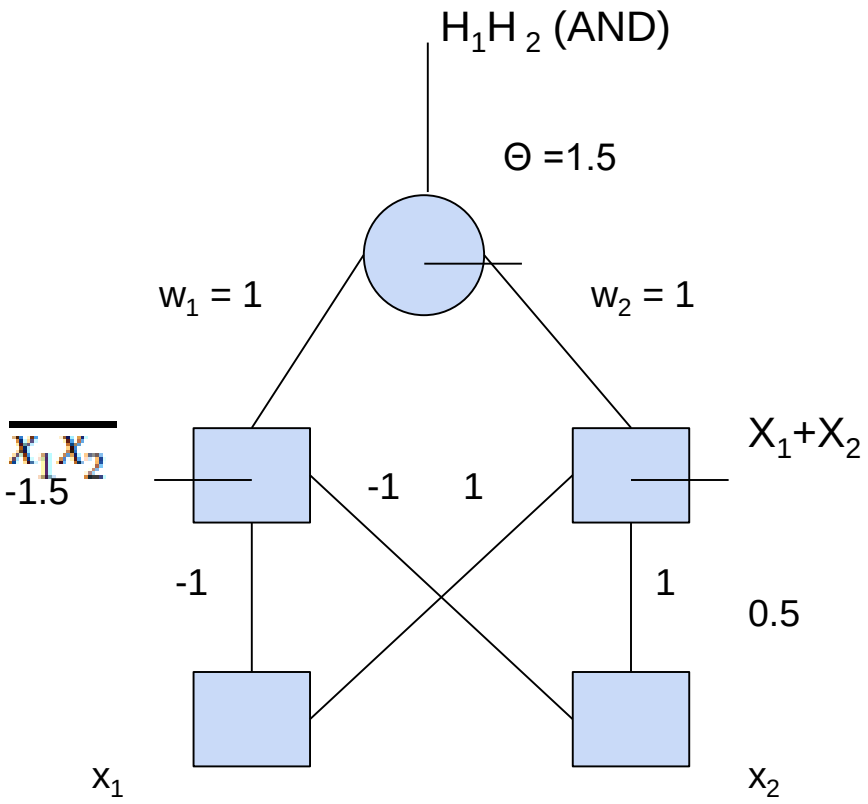
*learning rate * diff between target and observed outputs * input at the connection*

# Feedforward Network and Backpropagation

# Example - XOR



# Alternative network for XOR

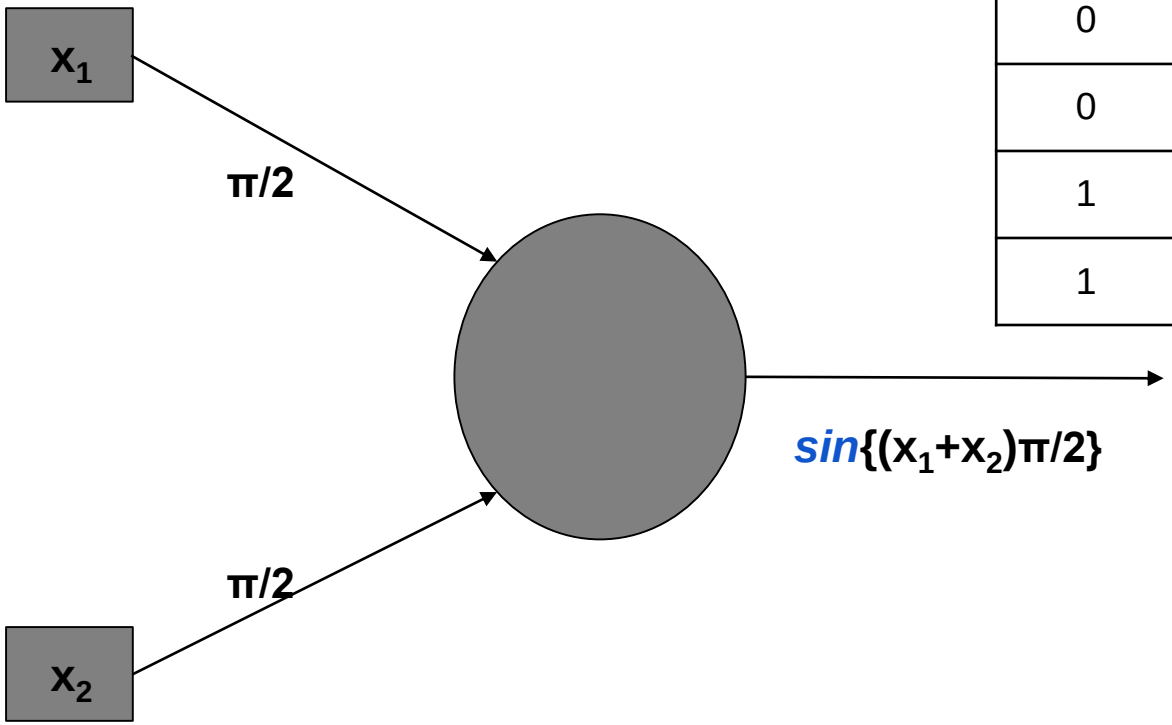



- XOR: not possible using a single perceptron
- Hidden layer gives more computational capability
- Deep neural network: With multiple hidden layers
- Kolmogorov's theorem of equivalence proves equivalence of multiple layer neural network to a single layer neural network, and each neuron have to correspond to an appropriate functions.

# Compositionality

- XOR being computed as OR($X_1$'$X_2$, $X_1X_2$') or as AND(($X_1$'+$X_2$'),(X1+X2)) is an example of a nonlinearly separable function computed as composition of linearly separable functions)
- In general not possible for most practical situations like weather prediction, stock market prediction etc.

# XOR neuron with sin()




| $x_1$ | $x_2$ | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Question

- Since SINE can compute XOR, why do not we use sine neuron for practical applications?

# Exercise: Back-propagation

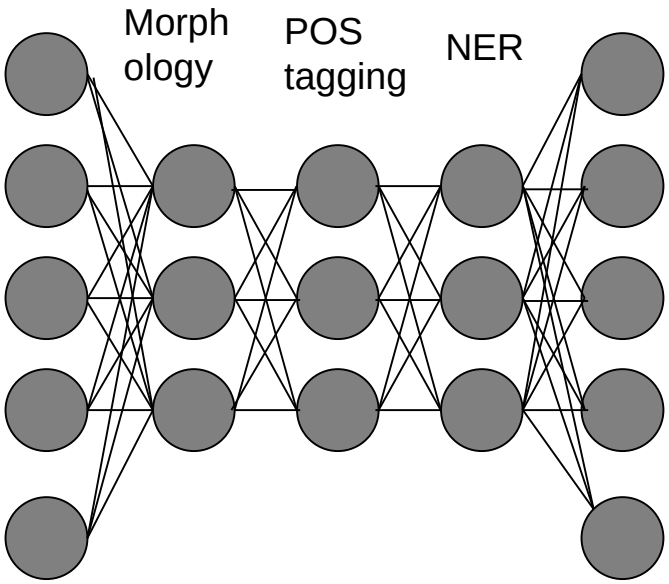
- Implement back-propagation for XOR network

- Observe
  - Check if it converges (error falls below a limit)

  - What is being done at the hidden layer

# What a neural network can represent in NLP: Indicative diagram

- Each layer of the neural network possibly represents different NLP stages!!



# Batch learning versus Incremental learning

- Batch learning is updating the parameters after ONE PASS over the whole dataset
- Incremental learning updates parameters after seeing each PATTERN (input-ouput pair)
- An epoch is ONE PASS over the entire dataset
  - Take XOR: data set is $V_1$=(<0,0>, 0), $V_2$=(<0,1>, 1), $V_3$=(<1,0>, 1), $V_4$=(<1,1>, 0)
  - If the weight values are changed after each of Vi, then this is incremental learning
  - If the weight values are changed after one pass over all $V_i$s, then it is bathc learning

# Can we use PTA for training FFN?

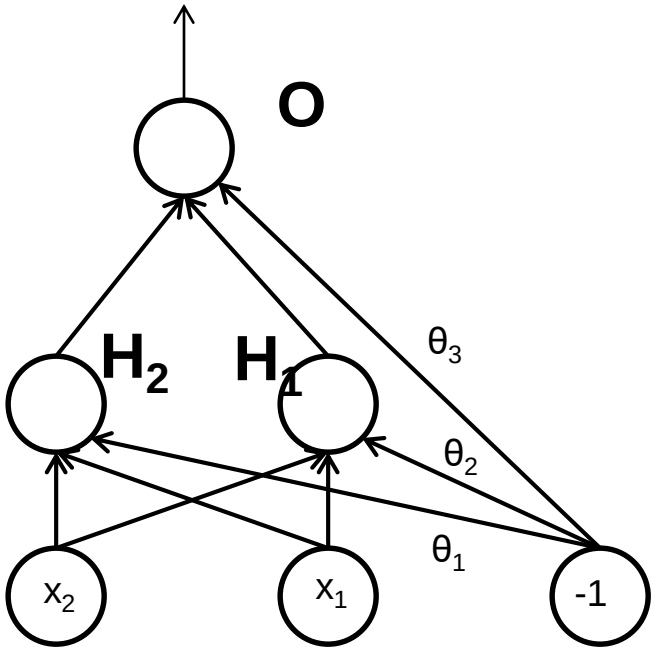
| | |
|---|---|
| 0, 0 | 0 |
| 0, 1 | 1 |
| 1, 0 | 1 |
| 1, 1 | 0 |

⇒

| | |
|---|---|
| -1, 0, 0 | 0 |
| -1, 0, 1 | 1 |
| -1, 1, 0 | 1 |
| -1, 1, 1 | 0 |

⇒

| | |
|---|---|
| 1, 0, 0 | 0 |
| -1, 0, 1 | 1 |
| -1, 1, 0 | 1 |
| 1, -1, -1 | 0 |




No, else the individual neurons are solving XOR, which is impossible.

Also, for the hidden layer neurons we do nothave the i/o behaviour.

Note: This n/w is NOT a pure FFNN; there is jumping of lair.

# Gradient Descent Technique

- Let E be the error at the output layer
- *i* goes over *N* neurons in the o/p layer, *j* goes over *P* patterns

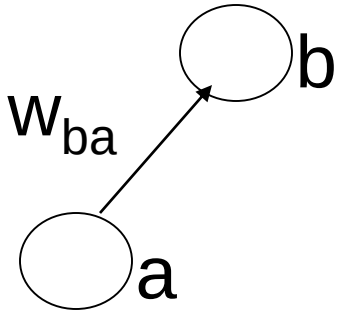
$$E = \frac{1}{2}\sum_{j=1}^{P}\sum_{i=1}^{N}(t_i - o_i)_j^2$$

- $t_i$ = target output; $o_i$ = observed output

- E.g.: XOR:– *P=4 and N=1*

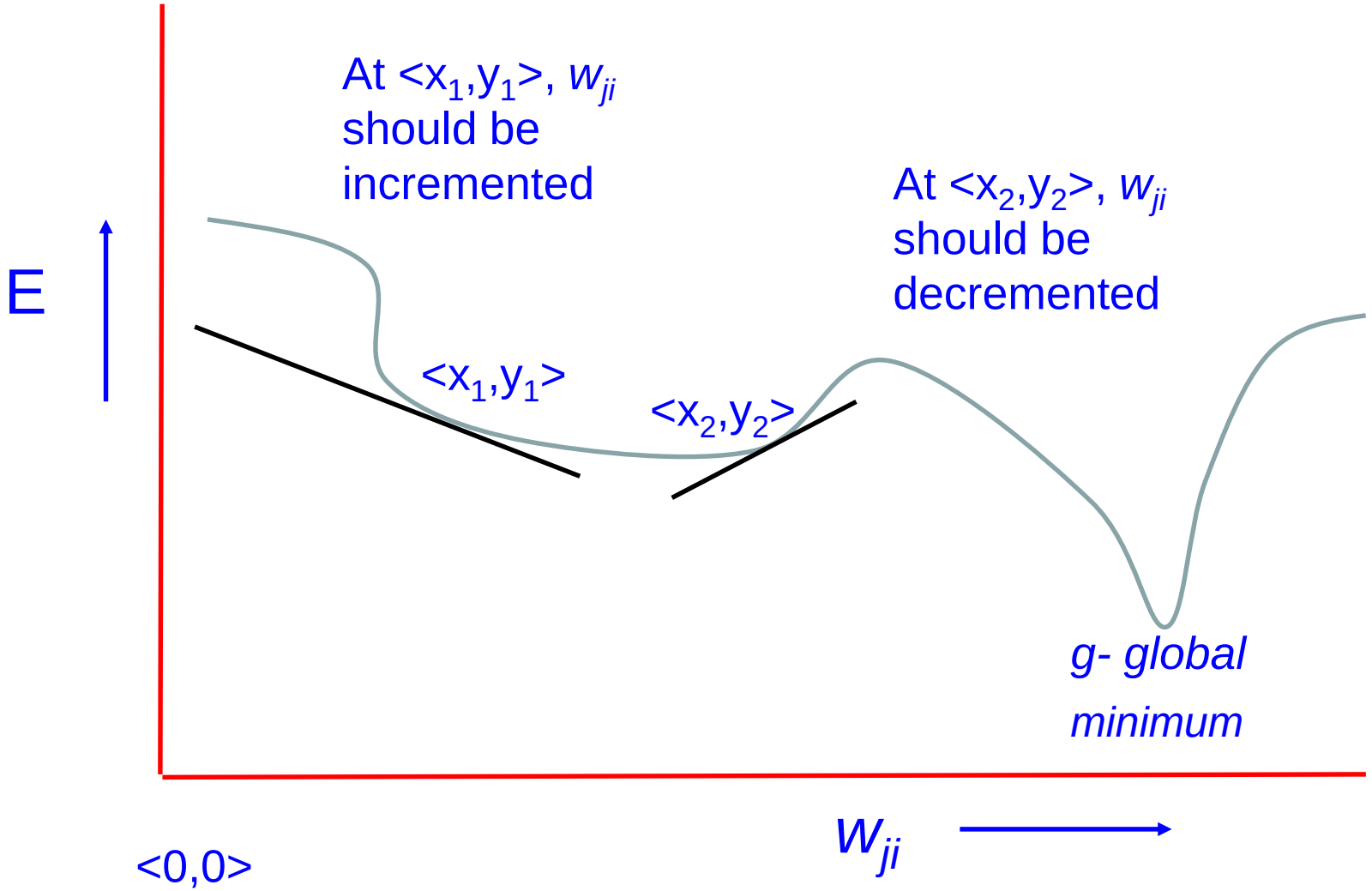
# Weights in a FF NN

- $w_{ba}$ is the weight of the connection from the $a^{th}$ neuron to the $b^{th}$ neuron
- E *vs* $\overline{W}$ surface is a complex surface in the space defined by the weights $w_{ij}$
- $-\frac{\delta E}{\delta w_{ba}}$ 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




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

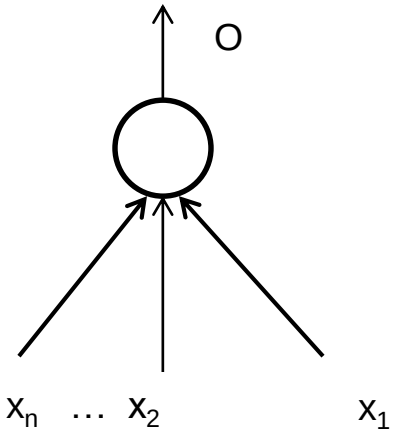
# Intuition for gradient descent



# Pertains to life!!

- Gradient descent is greedy in nature, E ALWAYS decreases
- Can get stuck in local minimum, miss global minimum
- So: “greed does not always pay”, “short term gains may not lead to long term gains”, “local optimizations need not always lead to global optimizations”

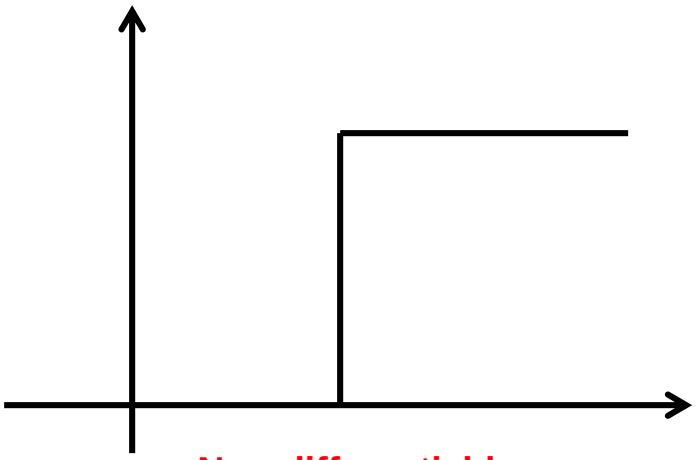
# Step function v/s Sigmoid function



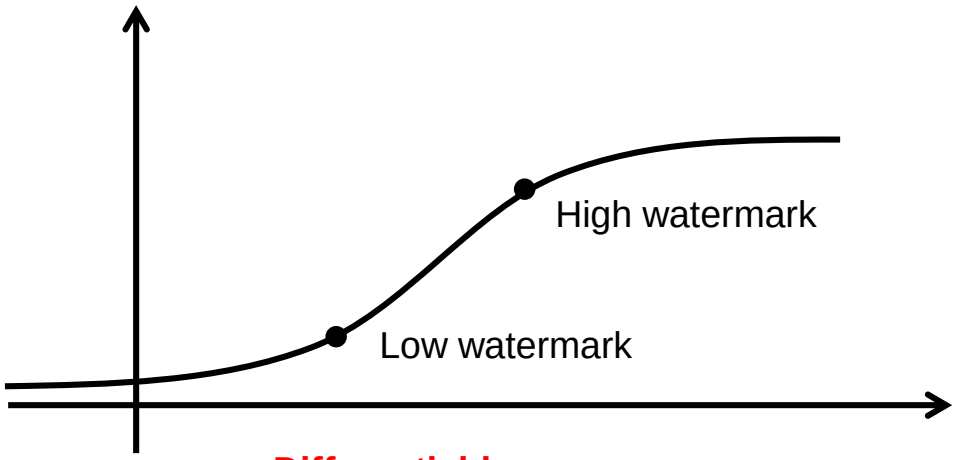

$$O = f(\sum w_i x_i)$$
$$= f(net)$$

So partial derivative of $O$ w.r.t. $net$ is

$$\frac{\delta O}{\delta net}$$

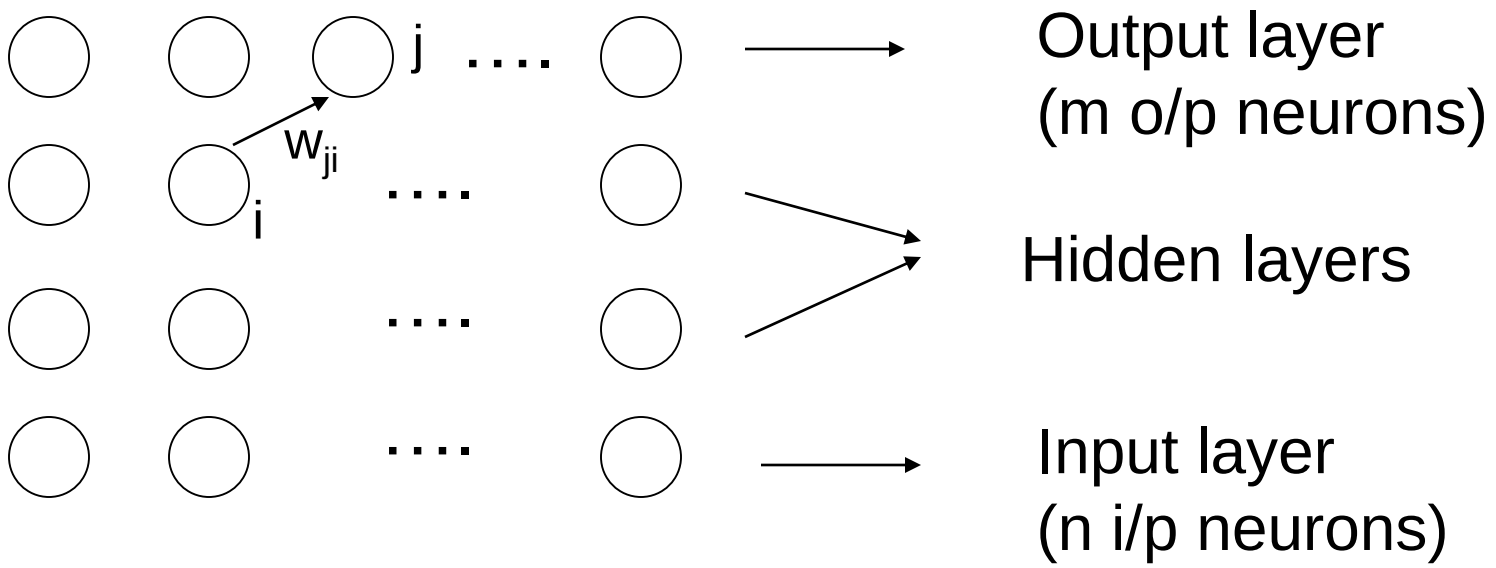


**Non-differentiable**




**Differentiable**

# Backpropagation algorithm




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

# Gradient Descent Equations

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

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

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

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

A quantity of great importance

# Backpropagation – for outermost layer

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

$$E = \frac{1}{2}\sum_{i=1}^{N}(t_j - o_j)^2$$

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

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

# Observations from $\Delta w_{ji}$

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

$\Delta w_{ji} \to 0$ if,

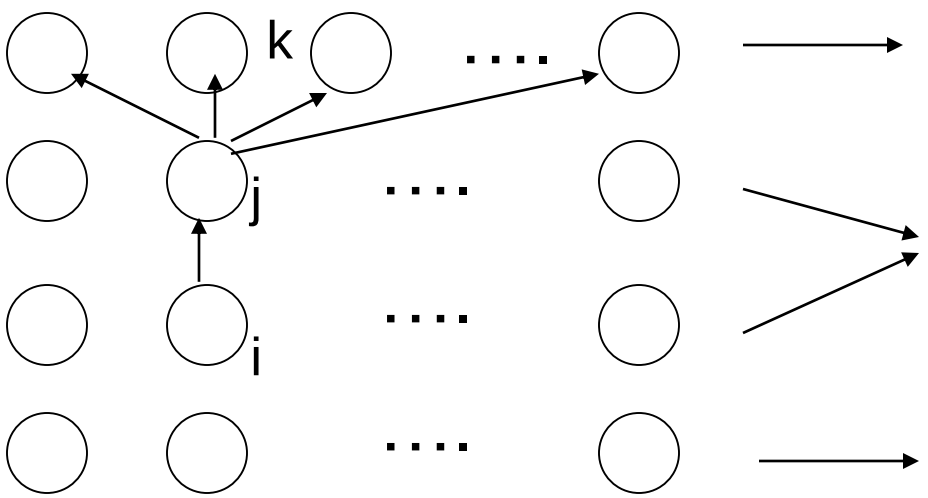
1. $o_j \to t_j$ and/or
2. $o_j \to 1$ and/or
3. $o_j \to 0$ and/or

} Saturation behaviour

4. $o_i \to 0$

} Credit/Blame assignment

# Backpropagation for hidden layers




Output layer
(m o/p neurons)

Hidden layers

Input layer
(n i/p neurons)

$\delta_k$ is propagated backwards to find value of $\delta_j$

# Backpropagation – for hidden layers

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

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

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

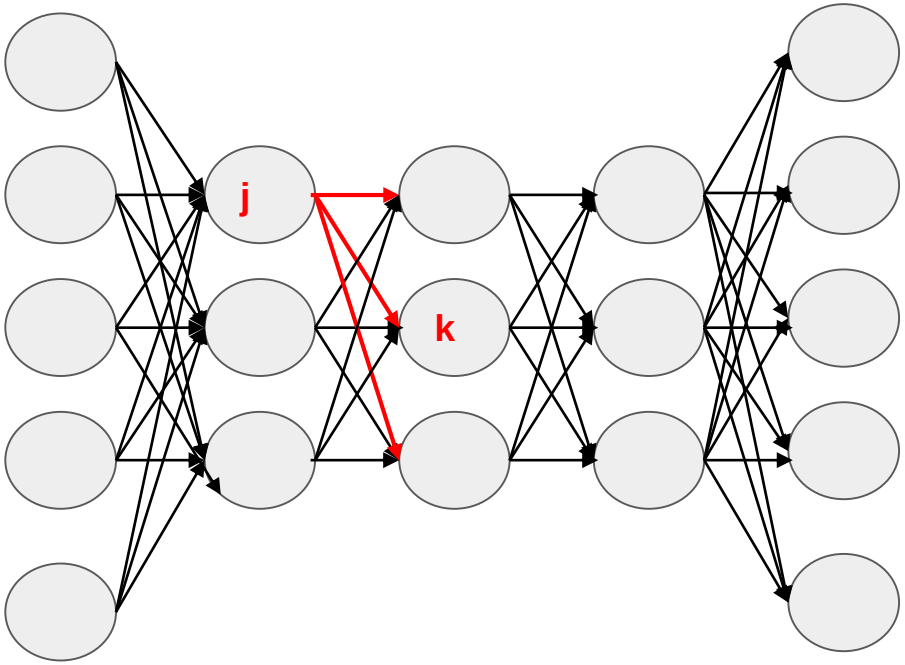
This recursion can give rise to vanishing and exploding Gradient problem

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

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

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

# Back-propagation- for hidden layers: Impact on net input on a neuron




- $O_j$ affects the net input coming to all the neurons in next layer

# General Backpropagation Rule

- General weight updating rule:

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

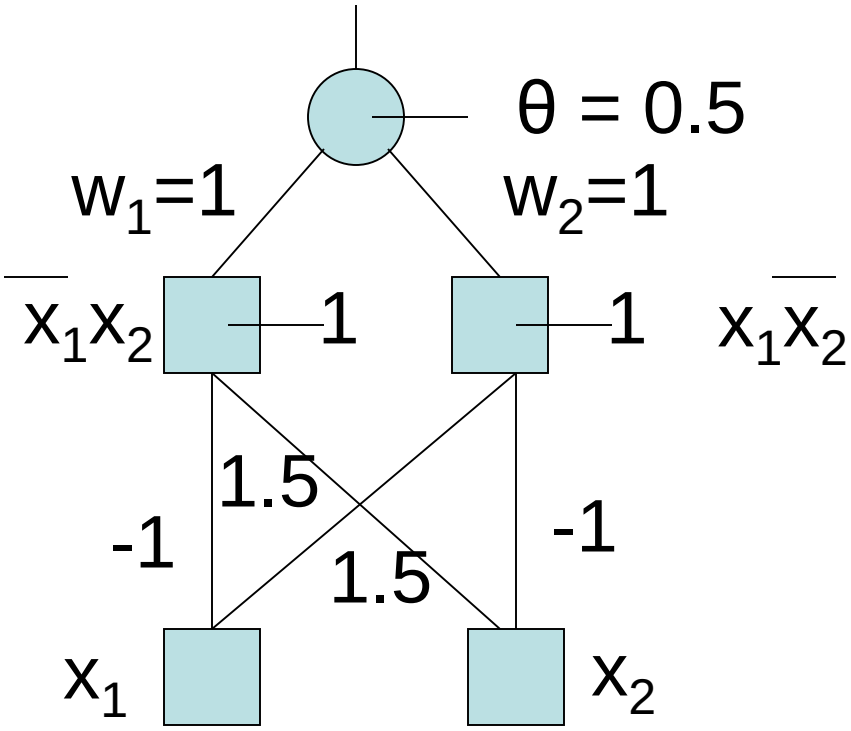
- Where

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

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

# How does it work?

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



# Optional Assignment

- Implement your OWN BP on XOR
- Observe what the hidden layer neurons compute

# An application in Medical Domain

# Expert System for Skin Diseases Diagnosis

- Bumpiness and scaliness of skin
- Mostly for symptom gathering and for developing diagnosis skills
- Not replacing doctor’s diagnosis

# Architecture of the FF NN

- 96-20-10
- 96 input neurons, 20 hidden layer neurons, 10 output neurons
- Inputs: skin disease symptoms and their parameters
  - *Location, distribution, shape, arrangement, pattern, number of lesions, presence of an active norder, amount of scale, elevation of papuls, color, altered pigmentation, itching, pustules, lymphadenopathy, palmer thickening, results of microscopic examination, presence of herald pathc, result of dermatology test called KOH*

# Output

- 10 neurons indicative of the diseases:
  - *psoriasis, pityriasis rubra pilaris, lichen planus, pityriasis rosea, tinea versicolor, dermatophytosis, cutaneous T-cell lymphoma, secondery syphilis, chronic contact dermatitis, soberrheic dermatitis*

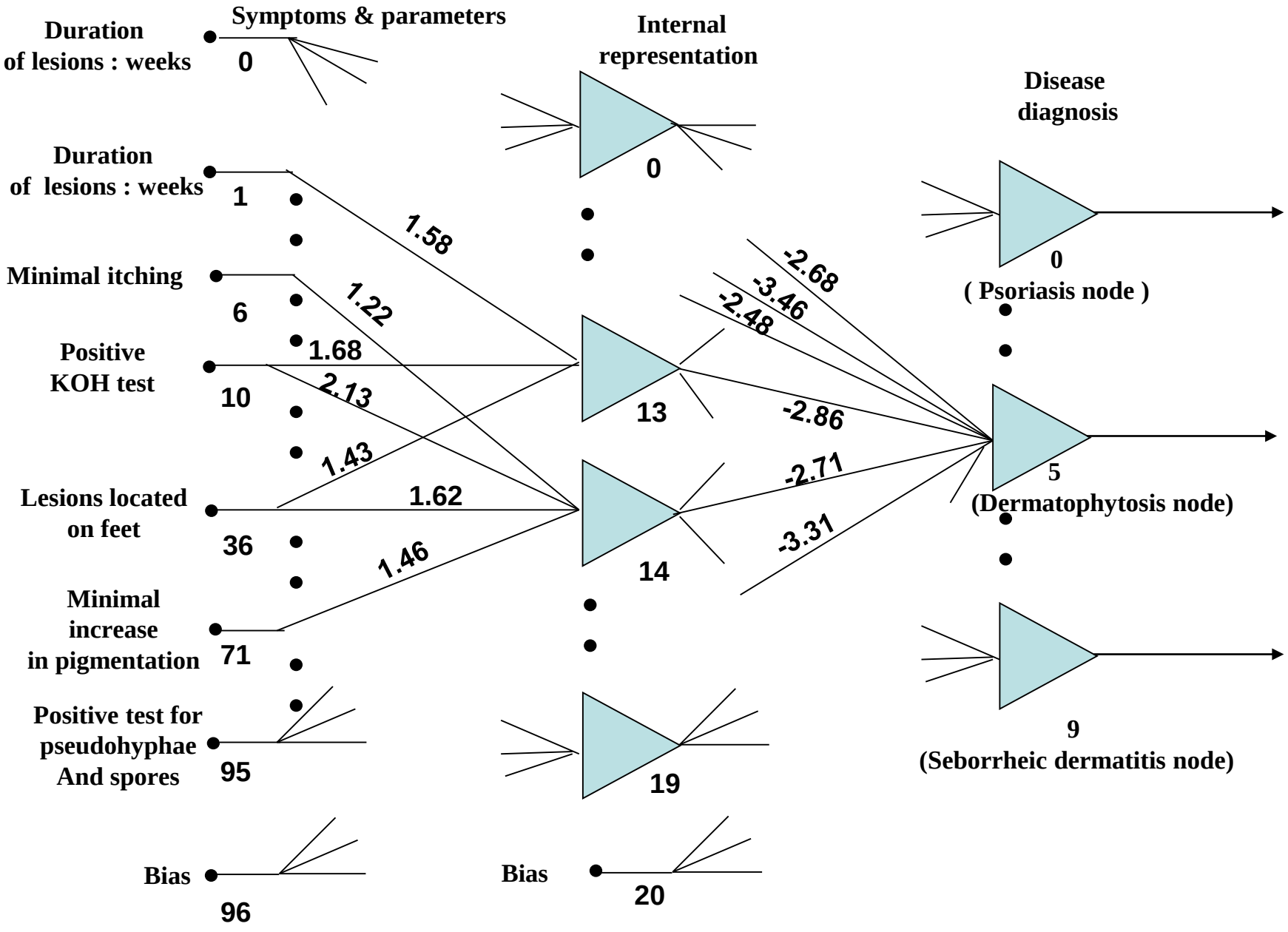



*Figure : Explanation of dermatophytosis diagnosis using the DESKNET expert system.*

# Training data

- Input specs of 10 model diseases from 250 patients
- 0.5 is some specific symptom value is not known
- Trained using standard error backpropagation algorithm

# Testing

- Previously unused symptom and disease data of 99 patients
- Result:
- Correct diagnosis achieved for 70% of papulosquamous group skin diseases
- Success rate above 80% for the remaining diseases except for psoriasis
- psoriasis diagnosed correctly only in 30% of the cases
- Psoriasis resembles other diseases within the papulosquamous group of diseases, and is somewhat difficult even for specialists to recognise.

# Explanation capability

- Rule based systems reveal the explicit path of reasoning through the textual statements
- Connectionist expert systems reach conclusions through complex, non linear and simultaneous interaction of many units
- Analysing the effect of a single input or a single group of inputs would be difficult and would yield incorrect results

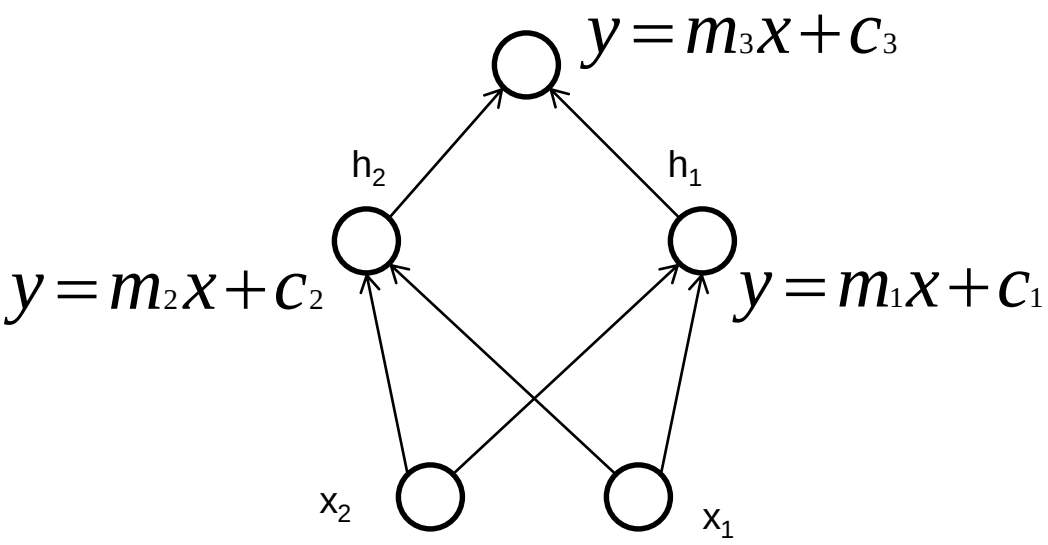
# Explanation contd.

- The hidden layer re-represents the data
- Outputs of hidden neurons are neither symtoms nor decisions

# Discussion

- Symptoms and parameters contributing to the diagnosis found from the n/w
- Standard deviation, mean and other tests of significance used to arrive at the importance of contributing parameters
- The n/w acts as apprentice to the expert

# Can Linear Neurons Work?




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

$$h_1 = m_2(w_1 x_1 + w_2 x_2) + c_2$$

$$Out = (w_5 h_1 + w_6 h_2) + c_3$$

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

**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]

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

$$\Rightarrow c - m.\theta < 0.1$$

For (0,0), Zero class:

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

$$\Rightarrow m.w_1 - m.\theta + c > 0.9$$

For (0,1), One class:

For (1,0), One class: $m.w_2 - m.\theta + c > 0.9$

For (1,1), Zero class: $m.w_1 - m_2.\theta + c < 0.1$

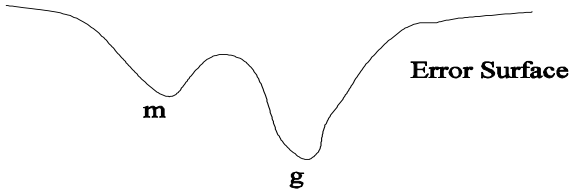
These equations are inconsistent. Hence X-OR can't be computed.

**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.

# 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

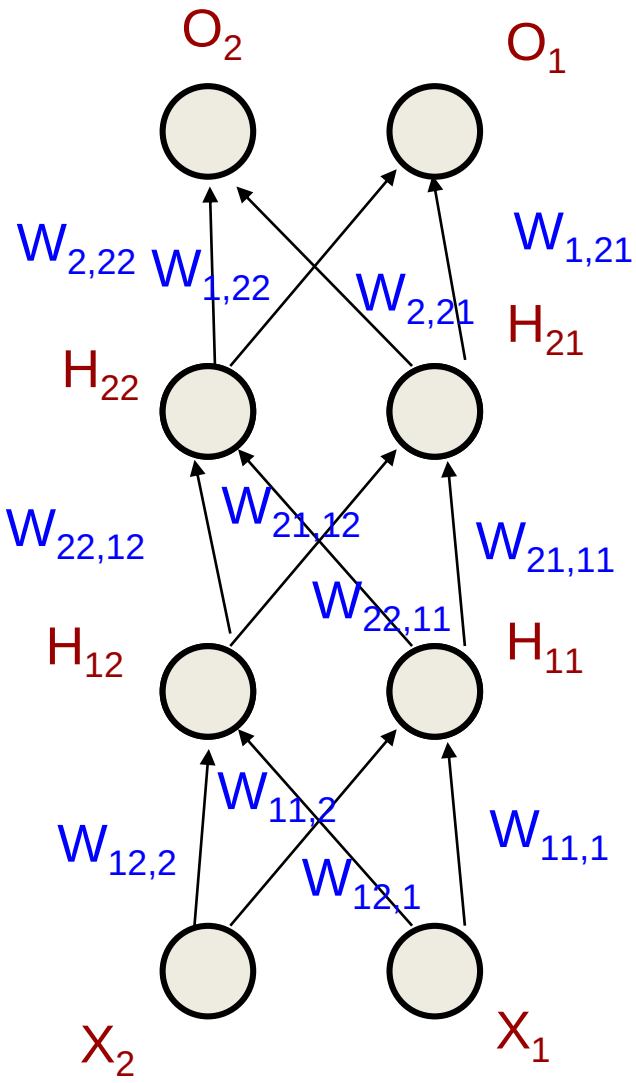
# Momentum factor

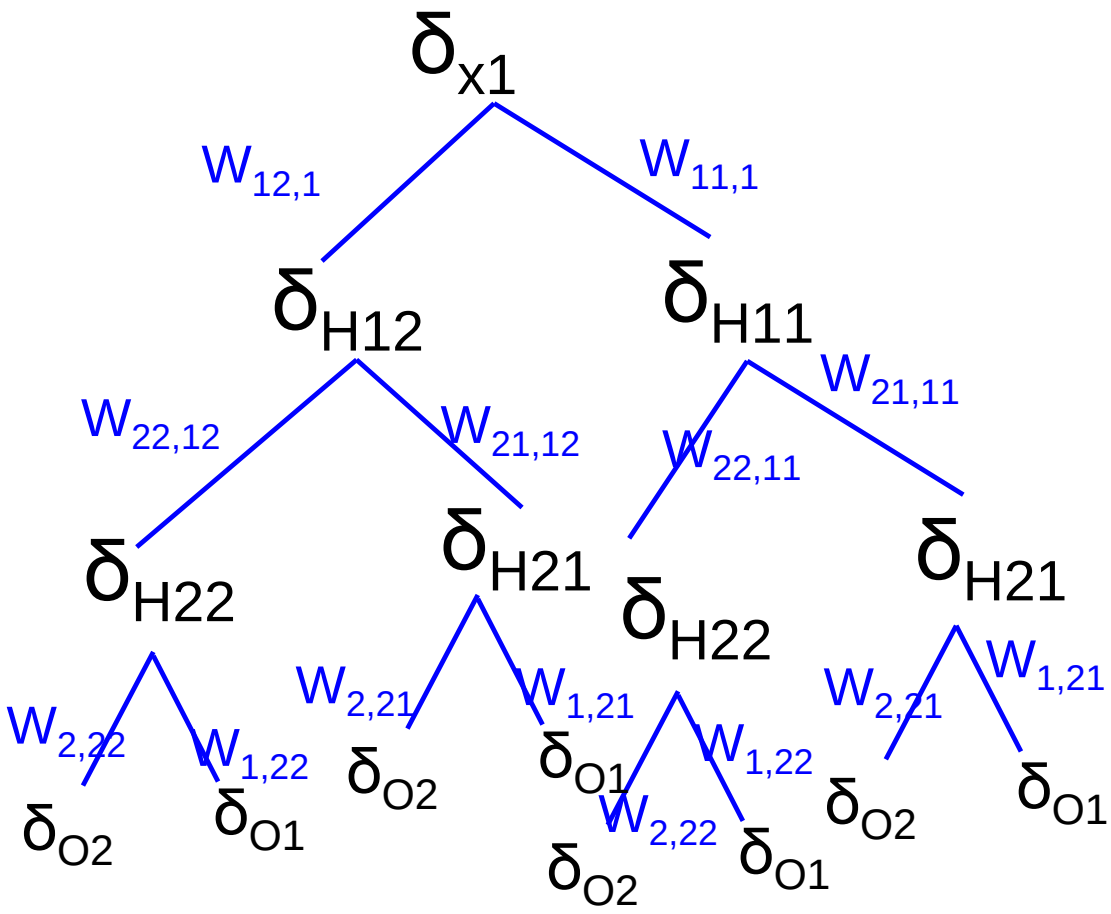
1. Introduce momentum factor.

$$(\Delta w_{ji})_{nth-iteration} = \eta \delta_j O_i + \beta (\Delta w_{ji})_{(n-1)th-iteration}$$

- Accelerates the movement out of the trough.
- Dampens oscillation inside the trough.
- Choosing $\beta$ : If $\beta$ is large, we may jump over the minimum.

# Vanishing/Exploding Gradient




$$\delta_{x1}=W_{11,1}\delta_{H11}+W_{12,1}\delta_{H12}$$

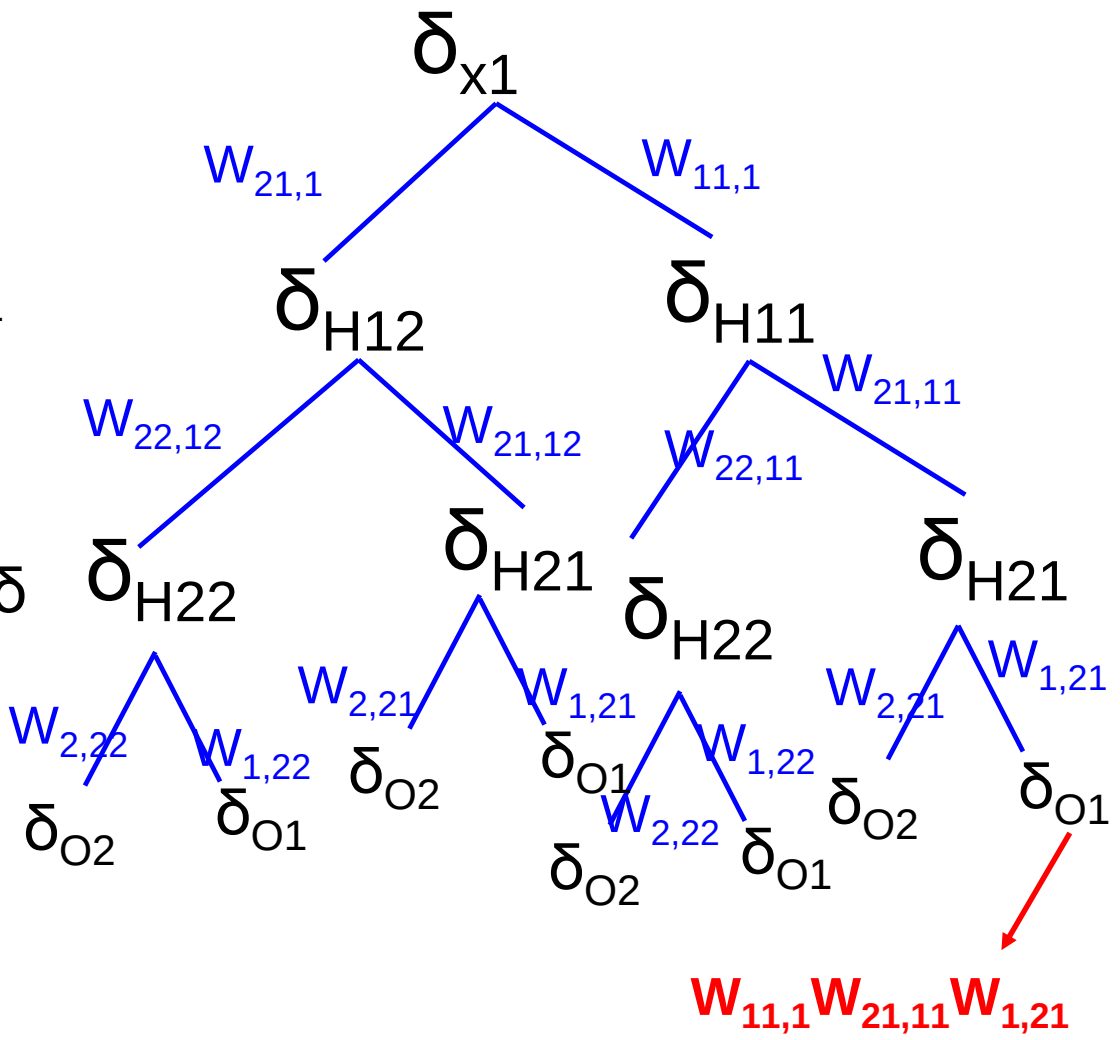
# Vanishing/Exploding Gradient

$\delta_{x1}=W_{11,1}\delta_{H11}+W_{21,1}\delta_{H12}$ [2 terms]

$=W_{11,1}(W_{21,11}\delta_{H21}+W_{22,11}\delta_{H22})+W_{21,1}(W_{21,12}\delta_{H21}+W_{22,12}\delta_{H22})$ [4 terms]

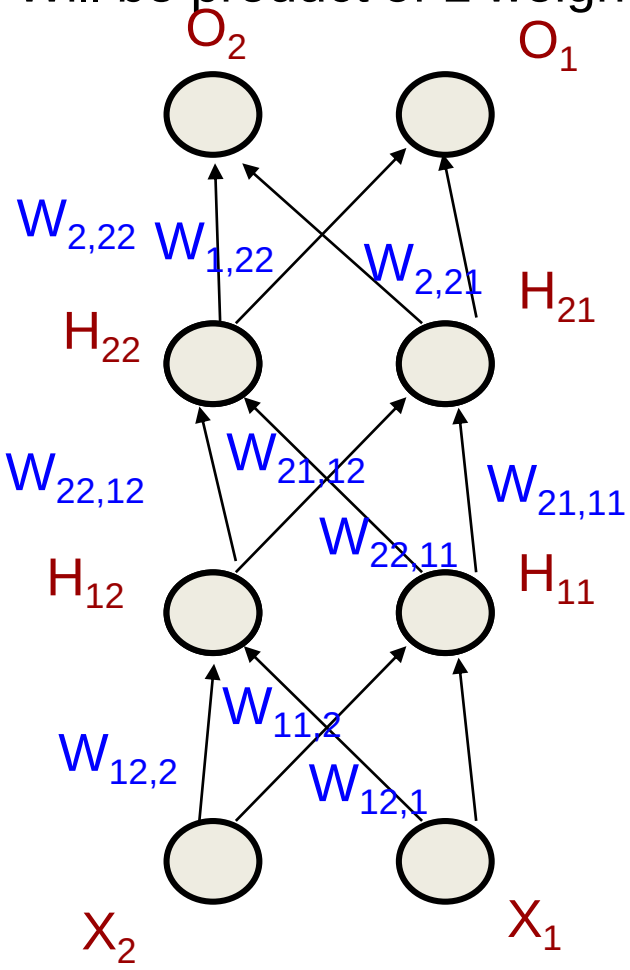
$= W_{11,1}W_{21,11}\delta_{H21}+W_{11,1}W_{22,21}\delta_{H22}+W_{21,1}W_{21,12}\delta_{H21}+W_{21,1}W_{22,12}\delta_{H22}$

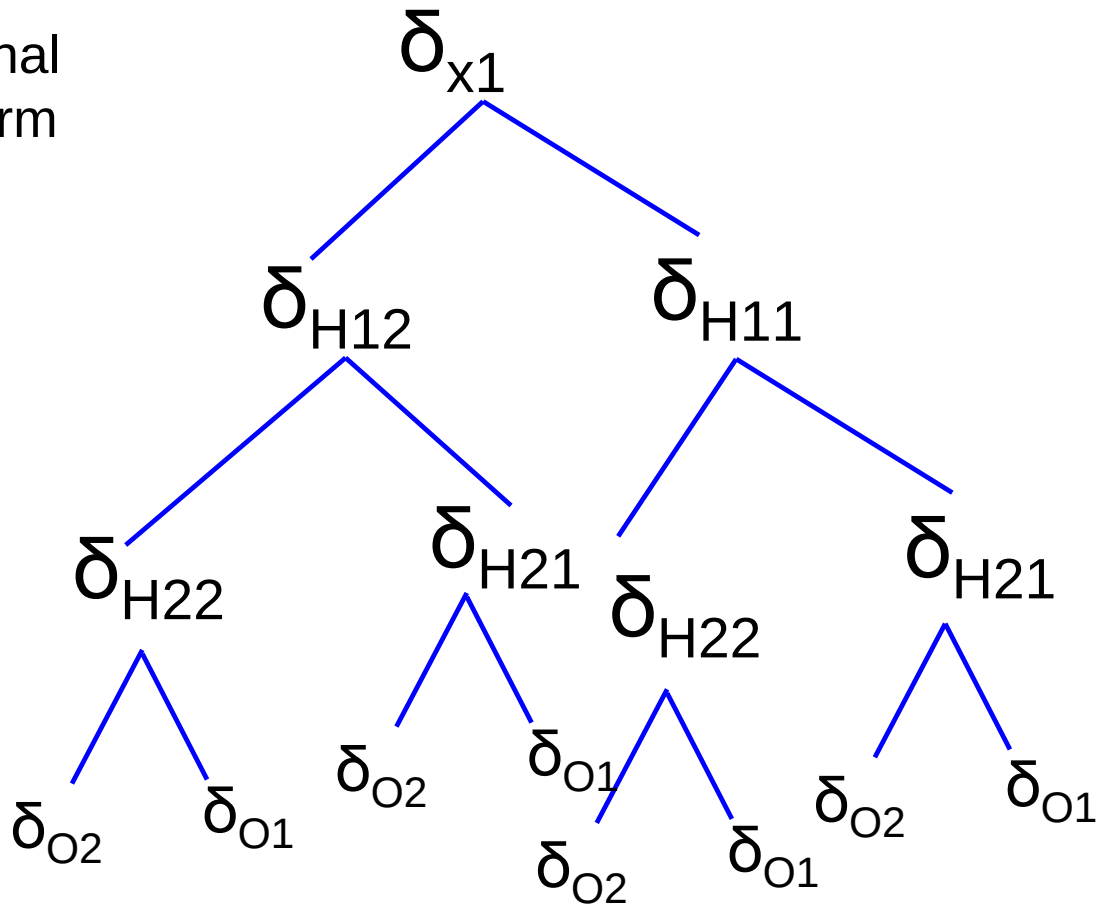
= (4 terms with $\delta_{o1}$) + (4 terms with $\delta_{o2}$; one term shown for the leftmost leaf's weight)



# Vanishing/Exploding Gradient

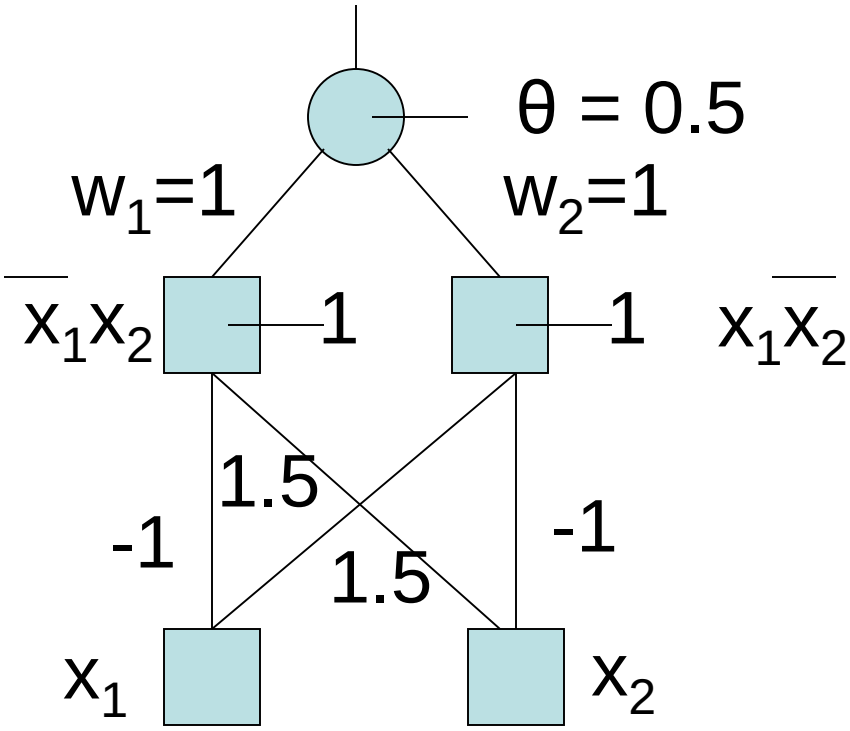
With '*B*' as branching factor and '*L*' as number of levels, There will be $B^L$ terms in the final Expansion of $\delta_{x1}$. Also each term Will be product of L weights



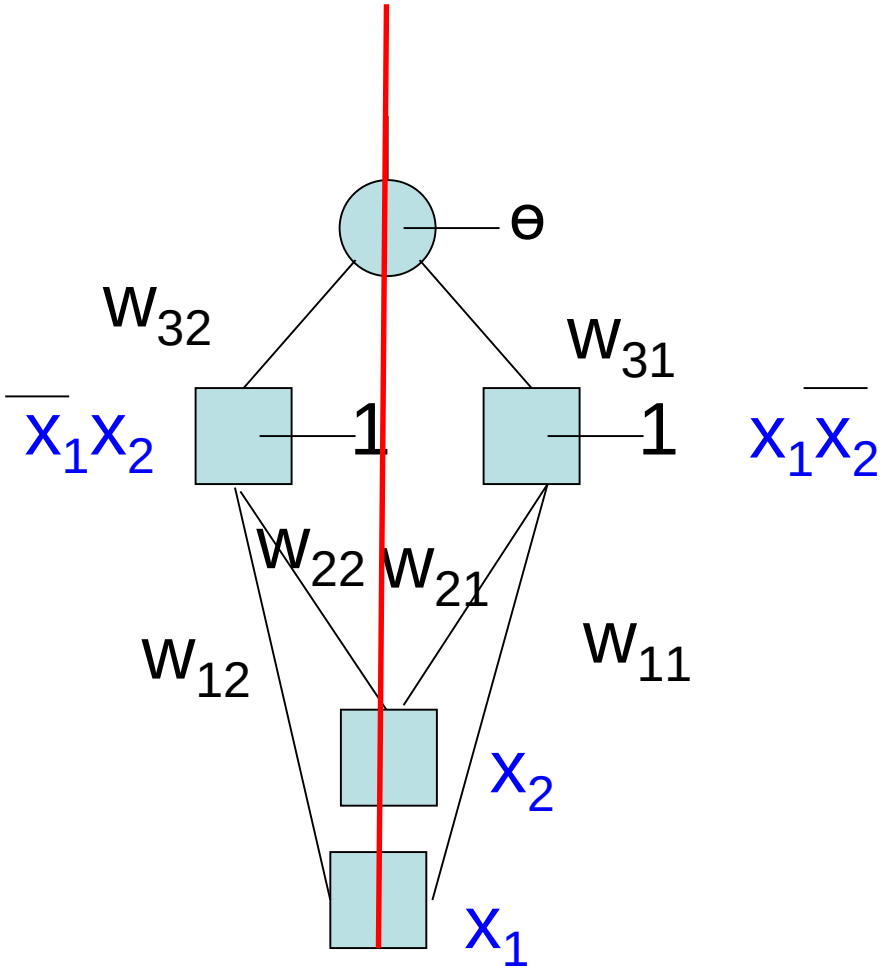
# Symmetry breaking

- If mapping demands different weights, but we start with the same weights everywhere, then BP will never converge.




XOR n/w: if we s started with identical weight everywhere, BP will not converge

# Symmetry breaking: understanding with proper diagram




***Symmetry About The red Line should Be broken***

# Your Assignment- due 30aug25

- Take POS tag data from NLTK https://www.nltk.org/
- Use HMM, EnCo-DeCo and any LLM of your choice to compare performance
- You can use pre-written/available/LLM-geneated code, but you will have to explain what the code is doing
- Later you will be asked to implement something innovative on/based-on POS tagging which will require you to code

# Assignment discussion

Template for

*You have to strictly follow this format*

# Define POS tagging

- Input (caution: you cannot give input inside the code)
- Output

# Data downloading and cleaning

- How much of data did you use
- From what source
- Did you use any cleaning? If so, what and how and why
- Did you use the POS tags as such or modify

# Our recommendation

- Use 12 universal cross lingual tags (yes 12 only)
- Run
  - *import nltk*
  - *nltk.download('brown')*
  - *from nltk.corpus import brown*
  - *print(brown.tagged_sents(tagset='universal')[0])*
- This returns Brown corpus sentences mapped to universal tags

# HMM Based POS tagging

- What did you read for this part of the assignment
- Why is Viterbi linear time?
- Is your program running?
- If yes, give the demo

# EnCo-DeCo based

- What did you read for this part of the assignment

- What algo does the decoding phase use?
- Is your program running?
- If yes, give the demo

# LLM based

- What did you read for this part of the assignment
- Which LLM did you use?
- Is your code running?
- If yes, give the demo

# Compare and contrast

- Give a tabular comparison of Precision (P), Recall (R) and F1 score
- Analyse and explain the observations

# Per POS accuracy

- For each of the 12 tags, compare and contrast as in the previous slide