Machine Learning Lect 5 48

Multi Layer Neural Network

Dr. Purana Mukherjee

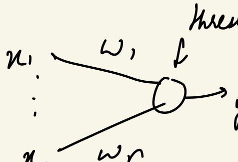
Multilayer Neural N/w

Single layer NN can represent the linear functions but in order to represent non-linear functions we require multilayer NN which can be achieved by stacking perceptrons into different architectures

Feed forward NN.

Limitations in Perceptron:

i) They exhibit the monotonic behaviour

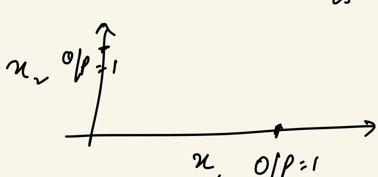


ii) if link has +ve weight activation can only increase as input values increases. It cannot represent functions where input interactions can cancel each other.

iii) Logic gates like AND can be represented as it is monotonic function O/P is 1 when $x_1 = 1, x_2 = 1$ but XOR cannot be represented by SLP as it is not a monotonic property

Can't represent functions where input interactions can cancel each other

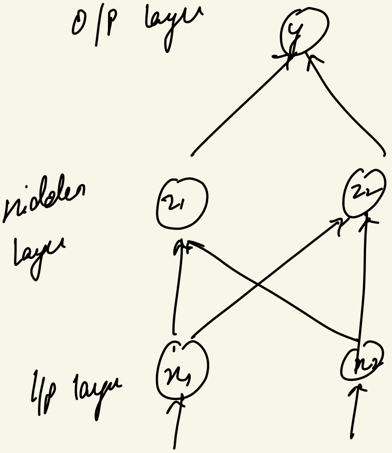
as

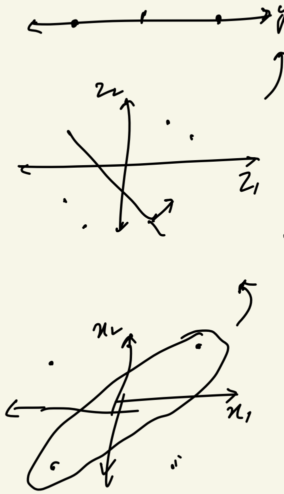


Soln: Multi Layer NN.

iv) can represent only linear separable functions

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0





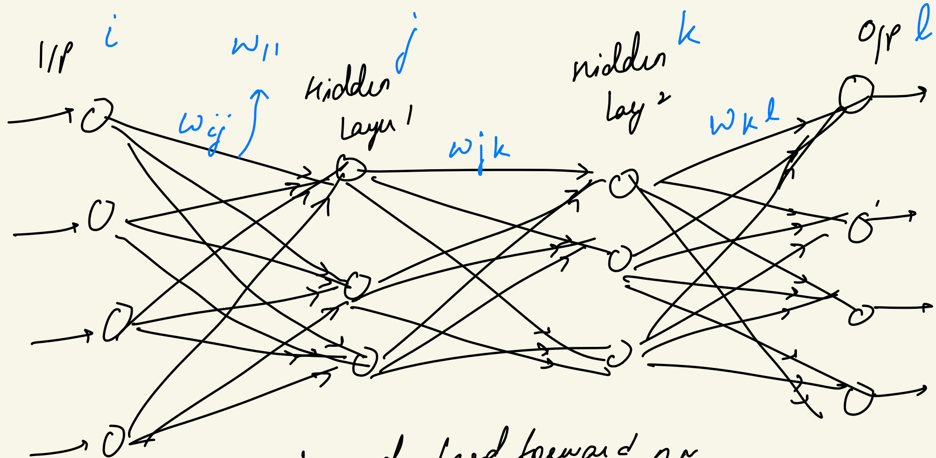
Each z_1 & z_2 is a linear fn. of x_1 & x_2 & using non-linear activation we can represent XOR fn.

Why hidden units: because in training we can't observe them & using them we can represent many non-linear functions

Power / Expressiveness of MLNN

- can represent interactions among inputs
- Two layer inputs can represent any Boolean function & continuous functions (within a tolerance) as long as the number of hidden units is sufficient & appropriate activation functions used
- Learning algorithm exists, but weaker guarantees than perceptron learning algorithms.

Three layer NN can represent all computable functions as they have good representational capability.



fully connected

Layered feed forward nn

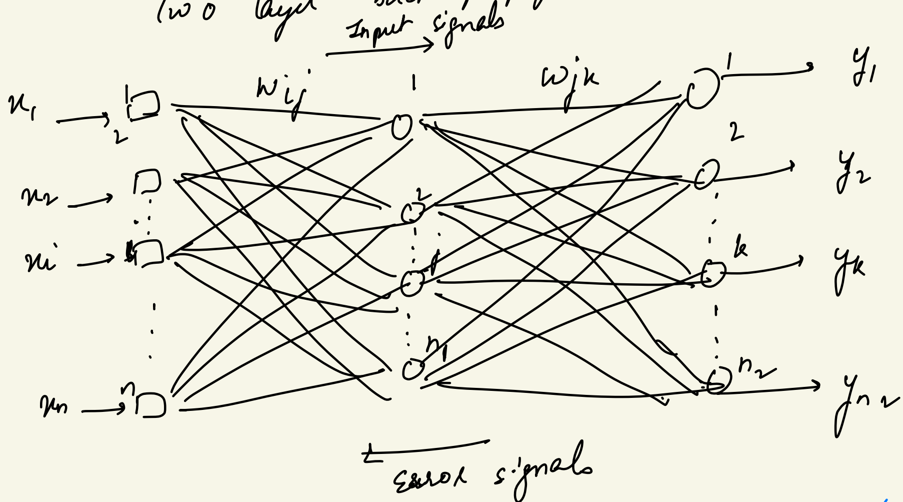
⇓ organized the neurons into layers

layer i → layer i+1

⇓ as there is no back links

4, 3, 3, 4

Two layer back propagation NN.



$\Delta W =$

Perception training rule : if there is change in o/p (error) you change the weights

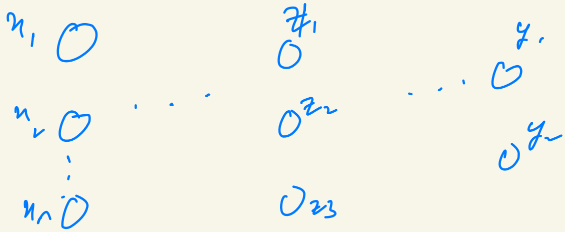
i.e difference b/w ideal & obtained o/p

It can be achieved in SLP

But there is problem in MLP ??

We do not know the ideal output in hidden units to compute the change in weights (w_{ij}')

Solution: Error observed at O/P layer is propagated backwards.



Error at y_1 is backpropagated to z_1, z_2 & z_3

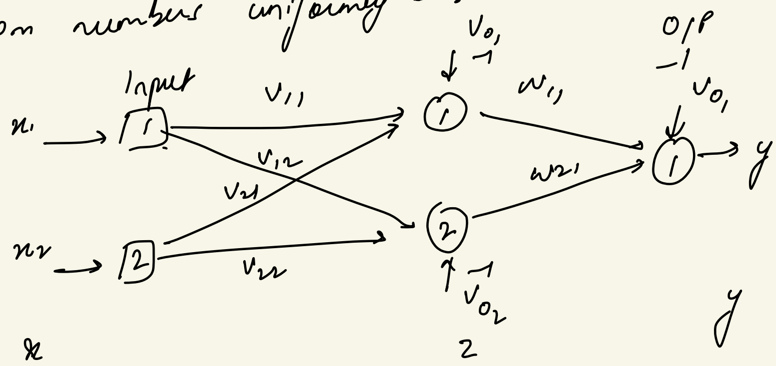
Error at y_2 is also backpropagated to z_1, z_2 & z_3

& thus we get the notional error & then we can update the weights

Backpropagation training algo.

Step 1: Initialization

Set all weights & threshold levels of the network to random numbers uniformly distributed inside a small range.



Step 2 Forward Computing

- Apply an input vector $\bar{x}$ to input units
- Compute the activation / output at the hidden layer

$\Sigma \mid \phi$ $\quad z_j = \phi\left(\sum_i v_{ij}\, x_i\right)$

- Compute the output vector y at output layer

$$y_k = \phi\left(\sum_j w_{jk}\, z_j\right)$$

$\bar{y}$ is the result of computation

Training examples

$\bar{x}_1$	$\bar{y}_1$	$\hat{y}_1$
$\bar{x}_2$	$\bar{y}_2$	$\hat{y}_2$
$\vdots$	$\vdots$	
$\bar{x}_m$	$\bar{y}_m$	$\hat{y}_m$

Error $\Rightarrow$ based on this change weights

Step 3 Learning for BP Nets

- Update of weights in W (between o/p & hidden layers)
  - delta rule
- Not applicable to updating V (b/w input & hidden)
  - don't know the target values of hidden units $z_1, z_2 \ldots z_p$

Solution: Propagate errors at o/p units to hidden units to drive the update of weights in V (again by delta rule) (error backpropagation learning)

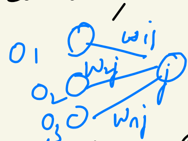
- Error backpropagation can be continued downward if net has more weight than one hidden layer
- How to compute errors on hidden units?

For 1 output neuron, error fn. is

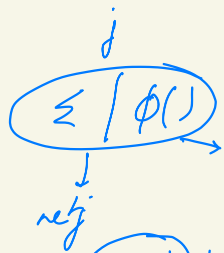
$$E = \frac{1}{2}\sum_{d}(y-\hat{y})^2$$

For each unit j, the output o_j is defined as,

$$o_j = \phi(net_j) = \phi\left(\sum_{k=1}^{n} w_{kj}\, o_k\right)$$



The input net_j to a neuron is the weighted sum of outputs o_k of previous n neurons.



Find derivative of the error:

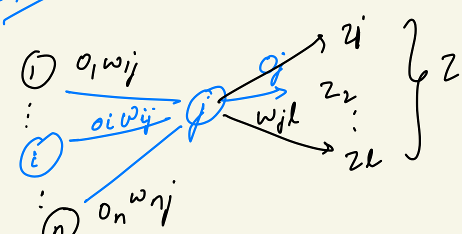
Backpropagation training rule →

$$\boxed{\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial net_j}\cdot\frac{\partial net_j}{\partial w_{ij}}} \quad — (*)$$

$\rightarrow w_1 o_1 + w_2 o_2 + \dots + w_{ij}\, o_i + \dots w_{kj}\, o_k$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj}\, o_k\right)$$

$$= o_i \quad — (1)$$

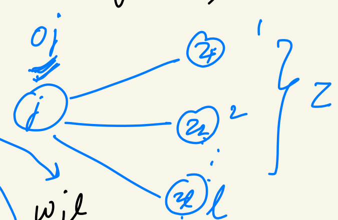


$\sigma'(z) = \sigma(z)(1-\sigma(z))$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\phi(net_j) = \phi(net_j)(1-\phi(net_j)) \quad — (2)$$

$\sigma(z)$

Depends on form of activation fn. (Let's assume sigmoid)



$\sum w_{jl}\, o_j$

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(1,2,\dots l)}{\partial o_j}$$

$net_l = \sum w_{jl} \cdot o_j = w_1 o_1 + w_2 o_2 + \dots$

$w_{jl}\, o_j$

$$= \sum_l \left(\frac{\partial E}{\partial net_l}\cdot\frac{\partial net_l}{\partial o_j}\right) \qquad \frac{\partial E}{\partial net_l} = \frac{\partial E}{\partial o_l}\cdot\frac{\partial o_l}{\partial net_l} \qquad \frac{\partial net_l}{\partial o_j} = w_{jl} \quad — (3)$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j \cdot o_i$$ Errors computed at 2^{th} layer which is backpropagated at j

$$\delta_j = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \begin{cases} (o_j - t_j)\, o_j\, (1-o_j) & \Rightarrow \text{if it is an o/p unit} \\ \sum_l (\delta_l w_{jl})\, o_j\, (1-o_j) & \Rightarrow \text{if there is some intermediate unit} \end{cases}$$

3. ✓ Perceptron training rule
✓ Gradient training rule
✓ Backpropagation training rule

NN ✓