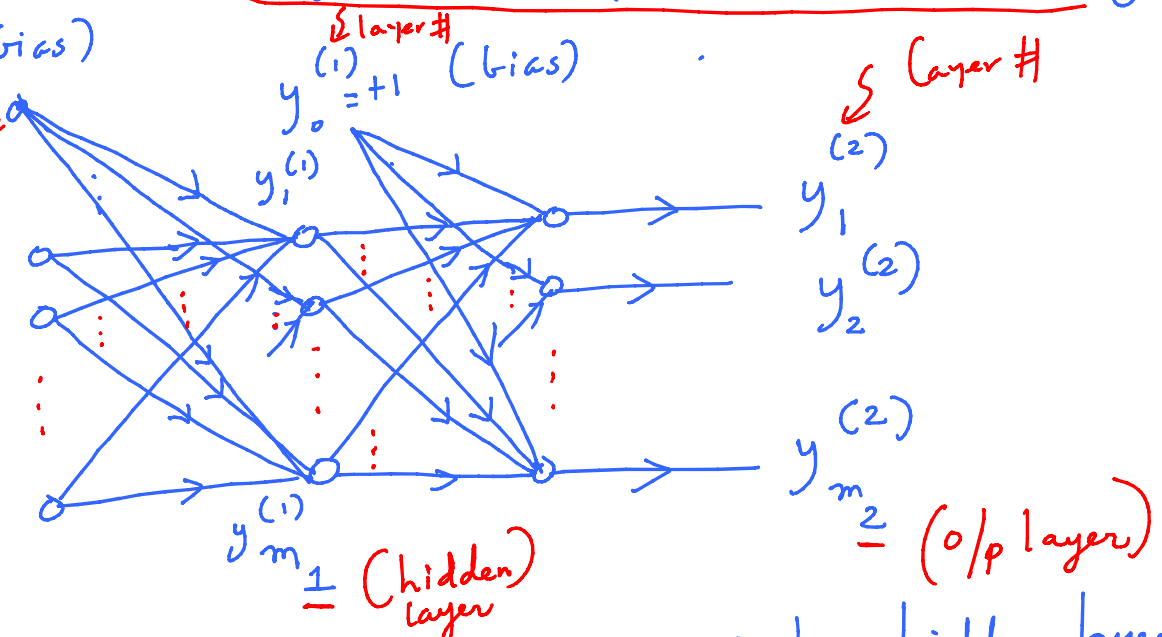


layer #  $\downarrow$  Consider a single layer of hidden neurons for simplicity

$y_0^{(0)} = +1$  (bias)  
 $\uparrow$  Coordinate

$y_1^{(0)} = x_1$   
 $y_2^{(0)} = x_2$   
 $\vdots$   
 $y_{m_0}^{(0)} = x_{m_0}$

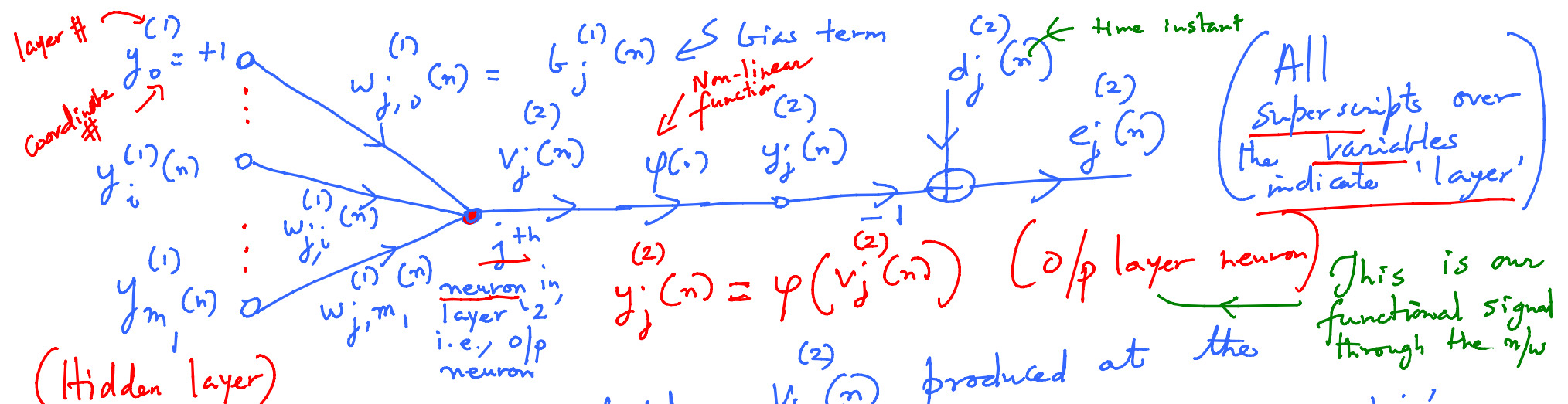


(Fully Connected)

$m_0$  inputs excluding the bias (i/p layer)

A network with a single hidden layer

$$y(\underbrace{\omega^T x}_{\uparrow} + \underbrace{b}_{\text{Bias}})$$



(Hidden layer)

The induced local field  $v_j^{(2)}(n)$  produced at the input of the activation function associated with neuron 'j' is

$$v_j^{(2)}(n) = \sum_{i=0}^{m_1} w_{ji}^{(1)}(n) y_i^{(1)}(n)$$

(m)

# neurons in hidden layer excluding bias

This is our functional signal through the m/w

Function signal  $y_j^{(2)}(n)$  at iteration 'n' for o/p neuron

$j'$

$$y_j^{(2)}(n) = \varphi(v_j^{(2)}(n))$$

local receptive field

We need to apply to the derivative

a correction

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}^{(1)}(n)}$$

$$\Delta w_{ji}^{(1)}(n)$$

proportional

error gradient

Key computation



Performance curve

Idea

$$\frac{\partial \mathcal{L}(n)}{\partial w_{ji}^{(1)}(n)}$$

Sensitivity

Now, let us calculate all the partial derivatives

inst. total error over all the neurons

$$= \frac{\partial \mathcal{L}(n)}{\partial e_j^{(2)}(n)} \frac{\partial e_j^{(2)}(n)}{\partial y_j^{(2)}(n)} \frac{\partial y_j^{(2)}(n)}{\partial v_j^{(2)}(n)} \frac{\partial v_j^{(2)}(n)}{\partial w_{ji}^{(1)}(n)}$$

I inst. error w.r.t neuron j  
 II function signal  
 III local receptive field  
 IV synaptic w.r.t. connecting  $w_{ji}^{(1)}(n)$  (o/p) hidden

CHAIN RULE

$$\frac{\partial \mathcal{L}(n)}{\partial e_j^{(2)}(n)} = e_j^{(2)}$$

$$\frac{\partial \mathcal{L}(n)}{\partial y_j^{(2)}(n)} = \psi'(v_j^{(2)})$$

$$y_j^{(2)} = \psi(v_j^{(2)})$$

$$\frac{\partial e_j^{(2)}(n)}{\partial y_j^{(2)}(n)} = -1$$

$$\frac{\partial y_j^{(2)}(n)}{\partial v_j^{(2)}(n)} = y_j^{(2)}$$

$$v_j^{(2)} = \sum_{i=0}^m w_{ji}^{(1)}(n) y_i^{(1)}(n)$$

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}^{(1)}(n)} = - e_j^{(2)}(n) \varphi'(v_j^{(2)}(n)) y_i^{(1)}(n)$$

$$\Delta w_{ji}^{(1)}(n) = - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}^{(1)}(n)}$$

( $\eta$  is the learning rate)

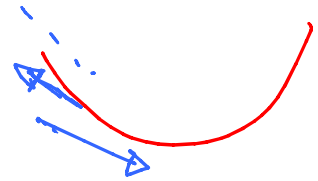
Define

$$\delta_j^{(2)}(n) \triangleq - \frac{\partial \mathcal{E}(n)}{\partial v_j^{(2)}(n)}$$

$\delta_j^{(2)}(n)$  is the local gradient

$$\delta_j^{(2)}(n) \triangleq e_j^{(2)}(n) \varphi'(v_j^{(2)}(n))$$

$$\Delta w_{ji}^{(1)}(n) \triangleq \eta \delta_j^{(2)}(n) y_i^{(1)}(n)$$



neuron  $j$

There are 2 cases

1)

Neuron 'j' is an output node

2)

Neuron 'j' is a hidden node.

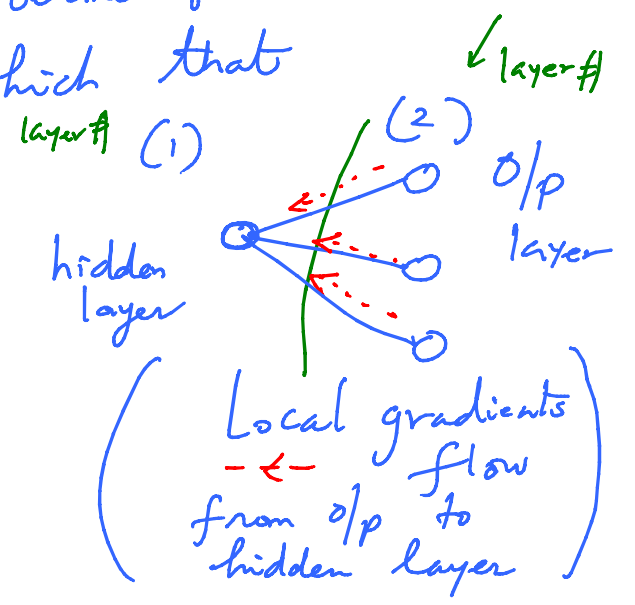
Case 1:

When neuron 'j' is located in the o/p layer,  
it is supplied with a desired response  
We can compute  $e_j^{(2)}(n)$  associated with this neuron  
As a consequence, computing  $\delta_j^{(2)}(n)$  is easy.

Case 2 : Neuron 'j' is a hidden node.

Trouble : There is no specified desired response to that neuron.

Error signal for a hidden neuron must be determined recursively by working backwards in terms of the error signals of all the neurons to which that hidden neuron is directly connected to



Suppose neuron 'j' is the hidden neuron in layer i.e., '1'

$$\delta_j^{(1)} = - \frac{\partial \mathcal{E}^{(1)}(n)}{\partial y_j^{(1)}(n)} \frac{\partial y_j^{(1)}(n)}{\partial v_j^{(1)}(n)}$$

$$= - \frac{\partial \mathcal{E}^{(1)}(n)}{\partial y_j^{(1)}(n)} \varphi' \left( v_j^{(1)}(n) \right)$$

Note the consistency of '-' sign in the defn of local gradient

To avoid confusions

$$\mathcal{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^{(2)2}(n)$$

Index 'k' refers to the o/p neuron

Index 'j' refers to the hidden neuron



$$\frac{\partial \xi(n)}{\partial y_j^{(1)}(n)} = \sum_{k \in C} e_k^{(2)}(n) \frac{\partial e_k^{(2)}(n)}{\partial y_j^{(1)}(n)}$$

Observe layer # consistent with the error signal flow graph

$$= \sum_{k \in C} \underbrace{e_k^{(2)}(n)}_{\text{Term 1}} \frac{\partial e_k^{(2)}(n)}{\partial v_k^{(2)}(n)} \underbrace{\frac{\partial v_k^{(2)}(n)}{\partial y_j^{(1)}(n)}}_{\text{Term 2}}$$

CHAIN RULE

$$e_k^{(2)}(n) \triangleq d_k^{(2)}(n) - y_k^{(2)}(n)$$

$$\triangleq d_k^{(2)}(n) - \varphi(v_k^{(2)}(n))$$

$$\left\{ \frac{\partial e_k^{(2)}(n)}{\partial v_k^{(2)}(n)} \right. \text{ But, } = -\varphi'(v_k^{(2)}(n)) \quad \left. \begin{array}{l} \text{--- (a)} \\ \text{(1)} \\ \text{--- (b)} \end{array} \right\} \text{ Use (a) \& (b) within Term 2}$$

$$v_k^{(2)}(n) = \sum_{j=0}^{m_1} w_{kj}^{(1)}(n) y_j^{(1)}(n)$$

$$\frac{\partial V_k^{(2)}}{\partial y_j^{(1)}} = w_{kj}^{(1)}$$

Folding all the partial derivatives together

$$\frac{\partial \mathcal{E}^{(n)}}{\partial y_j^{(1)}} = - \sum_k e_k^{(2)} \underbrace{\varphi_k' (V_k^{(2)})}_{\downarrow} w_{kj}^{(1)}$$

$$= - \sum_k \underbrace{\delta_k^{(2)}}_{\substack{\text{local gradient} \\ \text{@ node } k \\ \text{in the o/p layer}}} w_{kj}^{(1)}$$

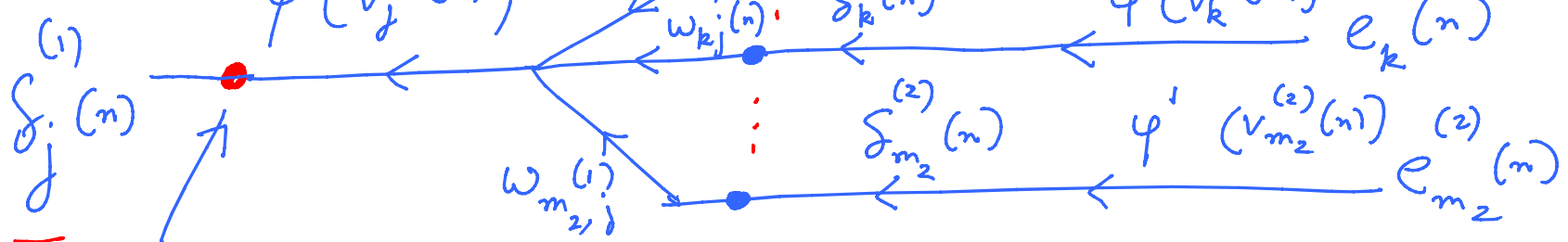
$w_{kj}^{(1)}$   
 $\uparrow$   $\leftarrow$   $j^{\text{th}}$  hidden neuron  
 $k^{\text{th}}$  o/p neuron

From this,

$$\delta_j^{(1)} = \varphi'(v_j^{(1)}) \sum_k \delta_k^{(2)} w_{kj}^{(1)}$$

← Key difference

(local grad.)



neuron 'j' in the hidden layer

Signal Flow Graph of an adjoint system pertaining to the back propagation algo

## "Update" of weights

In general,

$$\left( \begin{array}{c} \text{weight} \\ \text{correction} \end{array} \right) \Delta w_{ji}^{(n)} = \left( \begin{array}{c} \text{learning} \\ \text{rate} \end{array} \right) (\eta) \times \left( \begin{array}{c} \text{local} \\ \text{gradient} \end{array} \right) (\delta_j^{(n)}) \times \left( \begin{array}{c} \text{i/p signal} \\ \text{to neuron} \\ \text{'j'} \end{array} \right) (y_i^{(n)})$$

$\uparrow$  Const. usually

There are 2 cases

- (a) Neuron is in the hidden layer (It does not see the error directly)
- (b) Neuron is in the o/p layer

Going by the derivation,  
 For the o/p layer

Connected to neuron 'i'  
 in hidden  
 (NOTE  
 $y_0^{(1)} = 1$  always  
 i.e., bias term)


$$\Delta w_{ji}^{(1)}(n) = \eta \delta_j^{(2)}(n) y_i^{(1)}(n)$$


For the hidden layer

$$\Delta w_{ji}^{(0)}(n) = \eta \delta_j^{(1)}(n) \underbrace{y_i^{(0)}(n)}_{x_i}$$

(NOTE  
 $y_0^{(0)} = 1$   
always  
 i.e., bias)

Let us formalize the details of BPA computational steps.  
discussed so far

1) FORWARD PASS (Function Signals) 

2) BACKWARD PASS  
(Local error gradients  $\Rightarrow$  Synap. adaptation) 

# 1) Forward Pass :

In the forward pass, the synaptic weights remain unaltered through the n/w.

We mainly compute the function signals @ each neuron all the layers

$$y_j^{(l)}(n) = \varphi\left(v_j^{(l)}(n)\right)$$

$$v_j^{(l)}(n) = \sum_{i=0}^{m_{l-1}} w_{ji}^{(l)} y_i^{(l-1)}(n)$$

↑ local reception field

$l = 1, 2$

↑ function signal in prev layer

↓ observe

If neuron 'j' is in the hidden layer

(0) ↓ I/p

$$y_i^{(0)}(n) = x_i(n) \quad \forall i \neq 0$$

$i = 1, \dots, m_0$

## 2) Backward Pass :

Starts at the o/p layer by passing the error signals leftward through the n/w, layer-by-layer, by recursively computing the local gradient 'g' for each neuron.



## Activation Function

(Single layer)

$\varphi(\cdot)$  must be differentiable & can have non-linearities.

1) Logistic function

$$\varphi(v_j^{(l)}(n)) = \frac{1}{1 + \exp(-a v_j^{(l)}(n))}$$

*local reception field*

$$\varphi'(v_j^{(l)}(n)) = \frac{a \exp(-a v_j^{(l)}(n))}{[1 + \exp(-a v_j^{(l)}(n))]^2}$$

$a > 0$   
adjustable quantity

$l = 1, 2$

With  $y_j^{(l)}(n) = \varphi(v_j^{(l)}(n))$ , we can write

$$\varphi'(v_j^{(l)}(n)) = a y_j^{(l)}(n) (1 - y_j^{(l)}(n)) \quad ; \quad \text{①}$$

For a neuron 'j' located in the o/p layer  
 layer #  $\rightarrow$   $y_j^{(2)}(n) = o_j^{(n)}$  ('o' stands for output)  
 o/p

Local gradient

$$\delta_j^{(2)} = e_j^{(2)} \varphi'(v_j^{(2)}(n)) = a [d_j^{(2)} - o_j^{(n)}] o_j^{(n)} (1 - o_j^{(n)})$$

default @ layer '2' i.e., o/p layer!

local gradient @ node 'j' in the hidden layer

$$\delta_j^{(1)}(n) =$$

For an arbitrary hidden node 'j'

$$\varphi'(v_j^{(1)}(n)) \sum_{k=1}^{m_2} \delta_k^{(2)}(n) w_{kj}^{(1)}(n)$$

$$= a y_j^{(1)}(n) (1 - y_j^{(1)}(n)) \sum_{k=1}^{m_2} \delta_k^{(2)}(n) w_{kj}^{(1)}(n)$$

Pay attention to the sig. flow graph

Home Work:

Sketch  
Investigate  
 $\varphi'(\cdot)$

$\varphi(\cdot)$ ,  $\varphi'(\cdot)$   
where you see a maximum for

2) Hyperbolic tangent function  $\tanh(\cdot)$

$$y_j^{(l)} = \varphi(v_j^{(l)}) = a \tanh(b v_j^{(l)}) \quad a, b > 0$$

$$\varphi'(v_j^{(l)}) = a b \operatorname{sech}^2(b v_j^{(l)})$$

$$= a b \left(1 - \tanh^2(b v_j^{(l)})\right)$$
$$= \frac{b}{a} \left(a - y_j^{(l)}\right) \left(a + y_j^{(l)}\right)$$

$$(x^2 - y^2 = (x+y)(x-y))$$

$$l = 1, 2$$

For a neuron 'j' in the o/p layer, the local gradient

$$\delta_j^{(2)} = e_j^{(2)} \varphi' (v_j^{(2)})$$

$$= \frac{b}{a} [d_j^{(2)} - o_j^{(2)}] [a - o_j^{(2)}] [a + o_j^{(2)}]$$

← directly available

Similarly for a neuron 'j' in the hidden layer

$$\delta_j^{(1)} = \varphi' (v_j^{(1)}) \sum_{k=1}^{m_2} \delta_k^{(2)} w_{kj}^{(1)}$$

$$= \frac{b}{a} (a - y_j^{(1)}) (a + y_j^{(1)}) \sum_{k=1}^{m_2} \delta_k^{(2)} w_{kj}^{(1)}$$

← Error is not directly available

Local gradient @ hidden nodes

## Rate of learning

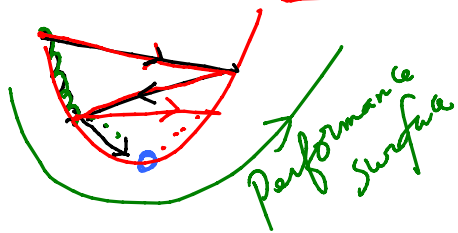
The BPA provides an approximation to the trajectory in the weight space through steepest descent

Smaller the value of  $\eta$   $\Rightarrow$

Smaller changes to the synaptic weights

Smother trajectory

Slower rate of learning



Too large  $\eta$   $\Rightarrow$

There can be oscillations & the  $n/w$  could be unstable

We shall introduce a momentum term

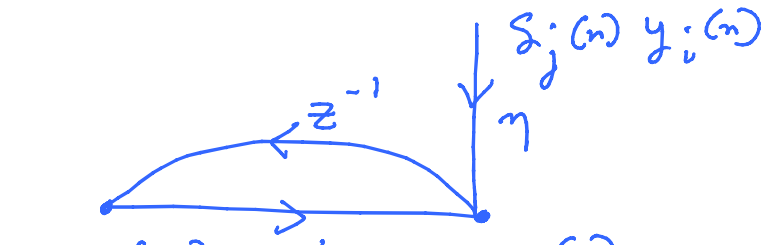
Observe the time steps

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta s_j(n) y_i(n)$$

Std. term through grad. descent  
Recursive eqn (A)

Controls the rate of decay ( $\alpha > 0$ )

(1st order auto reg. process)



Signal Flow Graph

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^n \alpha^{n-t} s_j(t) y_i(t)$$

$\frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)}$

powers of a decaying exponential

Few observations

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^n \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)}$$

1) Current adjustments  $\Delta w_{ji}(n)$  comprises of a sum  
of exponentially weighted time series.

Restrict  $\alpha$ :  $0 < \alpha < 1$

2) When  $\frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)}$  has the same algebraic sign on  
consecutive iterations,  $\Delta w_{ji}(n)$  grows  
in magnitude  $\&$  adjustments on the wts. happen  
on large scale  $\Rightarrow$  Accelerated downhill!



3) When  $\frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)}$  has opposite signs on consecutive iterations

$\Delta w_{ji}(n)$  shrinks in magnitude

$\Rightarrow$  Stabilizing effect.

NOTE:

We can have varying learning rates in a n/w for different neurons

Pros:

More flexibility  $\Rightarrow$  Better approx (conv.).

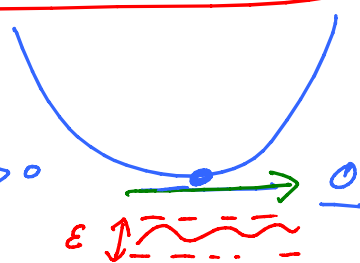
Cons:

Difficulty to control (Non linear dynamics)

## Stopping criterion

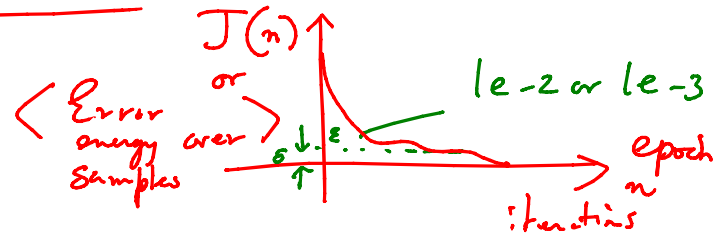
1) Stop the algo. when the Euclidean norm of the gradient vector reaches a small gradient threshold

(a) Compute  $\nabla_{\underline{w}} \mathcal{E} \quad \& \quad \|\underline{w}^{(n+1)} - \underline{w}^{(n)}\|_2 < \epsilon$



$\epsilon > 0$

2) BPA is considered to converge when the absolute rate of change in the average squared error per epoch is sufficiently small; say  $\epsilon \sim 0.01$  or  $0.001$



## Summary of the back propagation Algo

- 1) Initialization : Assuming no a priori information on the synaptic wts., pick the synap. wts. & thresholds from a uniform distribution with zero mean & variance to make the local fields lie within the linear part of the sigmoid activation fn.
- 2) Presentation of the training samples : Per epoch basis shuffle the data points & present this to the n/w.

3)

### Forward Computation

Consider the tuple  $\underline{x}(n)$  is the input

$\underline{d}(n)$  is the desired response.  
feature vector  $\underline{x}(n)$  and desired response vector  $\underline{d}(n)$

Compute the induced local fields and function signals of the  $n/w$  in a feed forward manner layer-by-layer  $\downarrow$  # of neurons in the  $(l-1)^{st}$  layer excluding 'bias'

$$v_j^{(l)}(n) = \sum_{i=1}^{m_{l-1}} w_{ji}^{(l-1)}(n) \underbrace{y_i^{(l-1)}(n)}_{\text{function signal of neuron 'i' in } (l-1)^{st} \text{ layer @ iteration 'n'}}$$

$\uparrow$   
 synaptic wts.

For the inputs,  $\sum$  I/ps.  $\forall 'j'$  in the inputs

$$y_j^{(0)}(n) = x_j^{(n)}$$

$\leftarrow$  Bias initialization

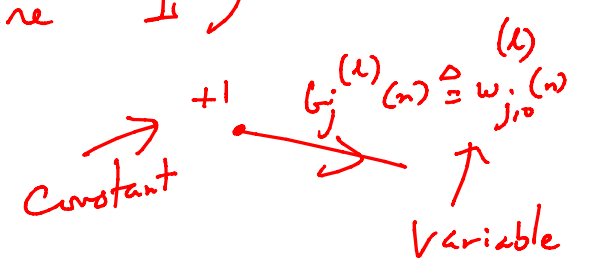
$$w_{j,0}^{(l)}(n) = b_j^{(l)}(n)$$

(Hard wire '1')

$$y_0^{(l-1)} = +1$$

$$l = L$$

For the o/p layer,



$$y_j^{(L)}(n) = o_j(n)$$

o/p

Compute the error signal

$$e_j(n) = d_j(n) - o_j(n)$$

Evaluate using  
the feed forward  
process

4)

Backward Computations

Key step: Compute  $\delta_s$  over all neurons in the layers

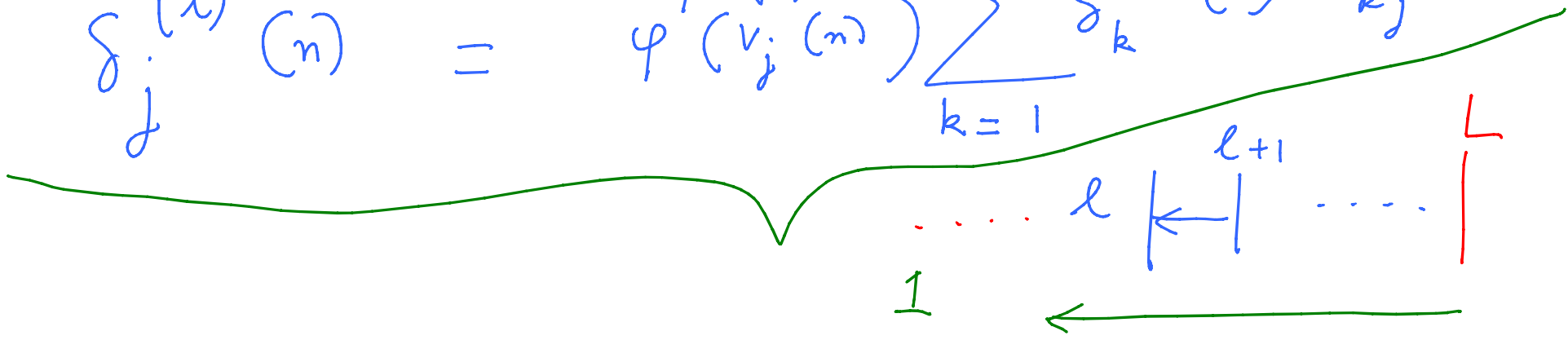
For the last layer  $l = L$

$$\delta_j^{(L)}(n) = e_j^{(L)}(n) \varphi' (v_j^{(L)}(n))$$

Over all layers  $l = 1, \dots, L-1$

# neurons in the  $(l+1)$ st layer

$$\delta_j^{(l)}(n) = \varphi' (v_j^{(l)}(n)) \sum_{k=1}^{m_{l+1}} \delta_k^{(l+1)}(n) w_{kj}^{(l)}(n)$$



Adjust the synaptic wts. as per

the general  $\Delta$ -rule

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \underbrace{\alpha \Delta w_{ji}^{(l)}(n-1)}_{\text{momentum push}} + \underbrace{\eta \delta_j^{(l)}(n) y_i^{(l-1)}(n)}_{\text{gradient descent}}$$

$\alpha$  : momentum

$\eta$  : learning rate

$$\Delta w_{ji}^{(l)}(n) = w_{ji}^{(l)}(n+1) - w_{ji}^{(l)}(n)$$



5.

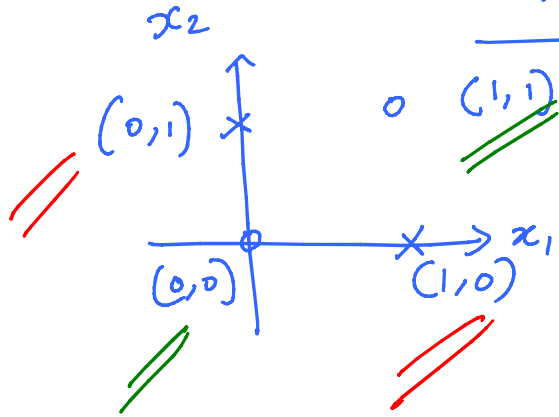
Iterations:

Loop forward and backward with  
Computations under Steps 3 & 4  
until the stopping criterion is met.

## Heuristics to make BPA work better

- 1) Stochastic vs. batch updates  
↓  
faster/works on highly redundant data
- 2) Max. information content
  - Use examples giving largest training errors
  - Have radically different data points
- 3) I/p data shuffling
- 4) Choice of activation function & range
  - Preferably within the range of sig. linear region
- 5) Normalizing inputs
- 6) Wt. initialization
- 7) Adaptive learning rates

# XOR Problem



0 ... Class 1  
x ... Class 2

We have four corners of a unit square

Class 1	$0 \oplus 0 = 0$	} Class 2
	$0 \oplus 1 = 1$	
	$1 \oplus 0 = 1$	
	$1 \oplus 1 = 0$	

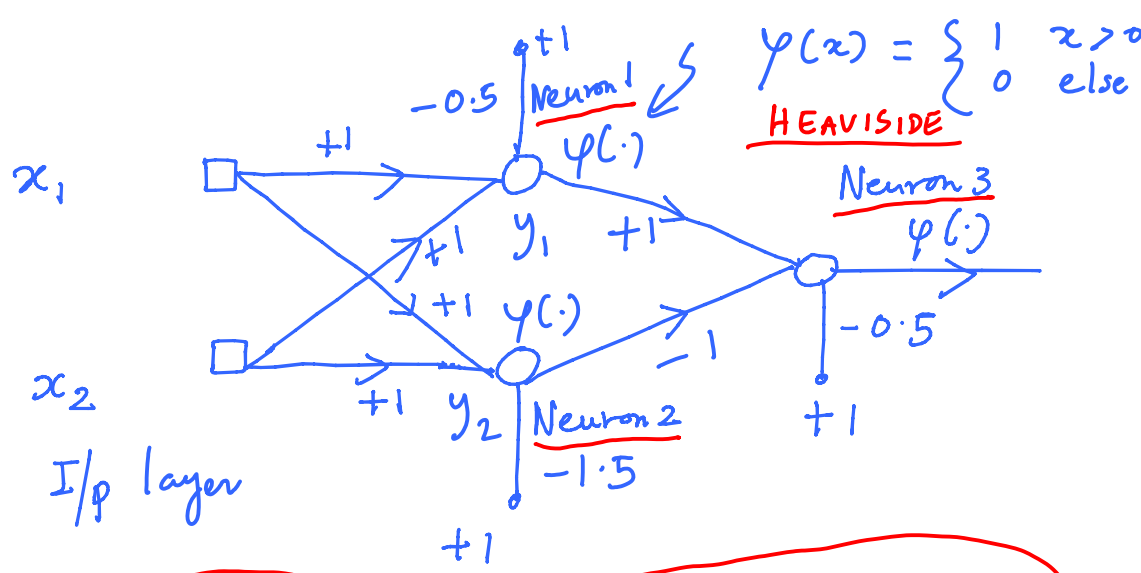
All the zeros are class '1'

All the ones are class '2'

---

One cannot get linear separability with just 1 hyperplane

I/ps



$$\varphi(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$$

HEAVISIDE

Hidden parameters

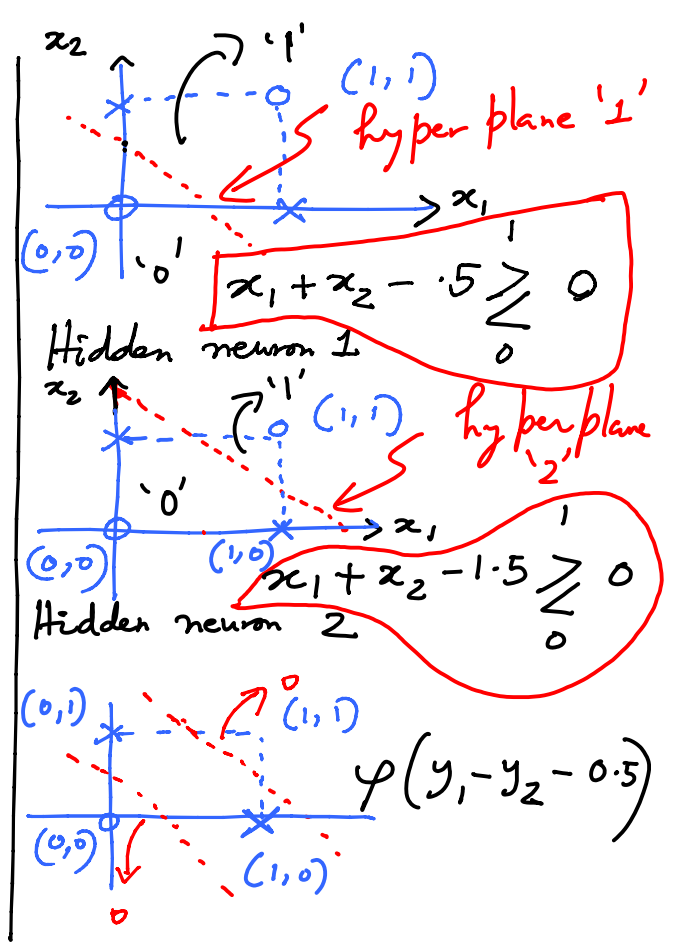
$$w_{11} = w_{12} = w_{21} = w_{22} = +1$$

$$b_1 = -1/2 \quad b_2 = -1.5$$

O/p parameters

$$w_{31} = +1 \quad w_{32} = -1$$

$$b_3 = -1/2$$



Hyper plane '1'

$$x_1 + x_2 - 0.5 \geq 0$$

Hidden neuron 1

Hyper plane '2'

$$x_1 + x_2 - 1.5 \geq 0$$

Hidden neuron 2

$$\varphi(y_1 - y_2 - 0.5)$$

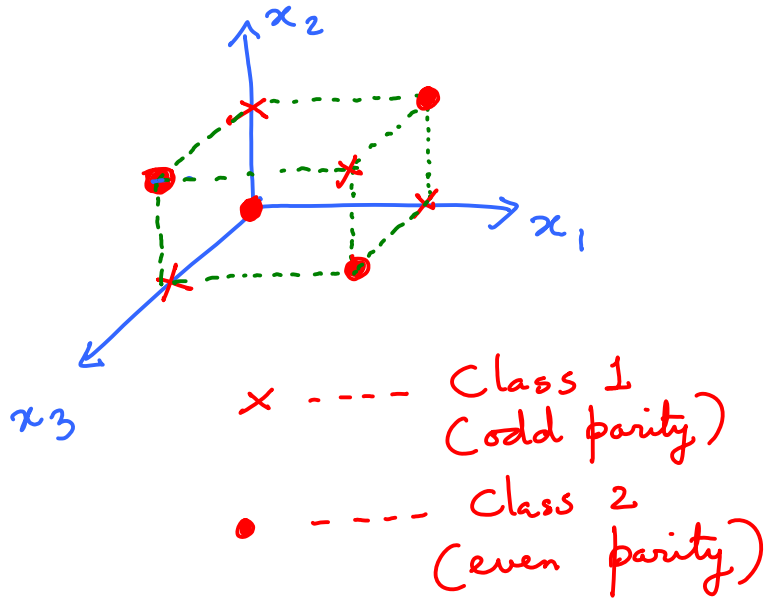
## 3 - Variable XOR Problem

Consider the 3 - variable XOR case (Boolean i/ps & o/ps)

$$y = x_1 \oplus x_2 \oplus x_3$$

We form the truth table

	$x_1$	$x_2$	$x_3$	$y$		$x_1$	$x_2$	$x_3$	$y$	
Even parity	0	0	0	0		0	0	1	}	Odd parity
	1	0	1	0		1	0	0		
	1	1	0	0		0	1	0		
	0	1	1	0		1	1	1		



Sketch in 3D space

We can clearly see that the classes are not linearly separable!

i.e., no one single plane that can shatter the two classes

Let us develop from our intuition on the 2-variables case

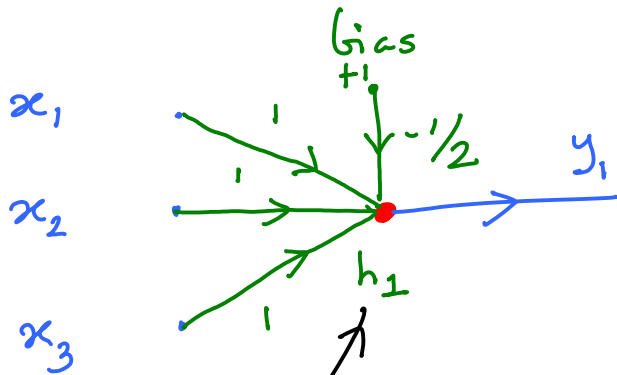
$$\varphi(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$$

1)

Let us build a neuron that responds when at least one of the inputs is a '1'

is the non-linear activation function

I/ps



$$y_1 = \varphi(\underline{w}^T \underline{x} + b_1); \quad b_1 = -1/2$$

$$\underline{w} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$\underline{x}$  is any tuple in  $\mathbb{Z}_2^3$

(Binary strings) of length 3

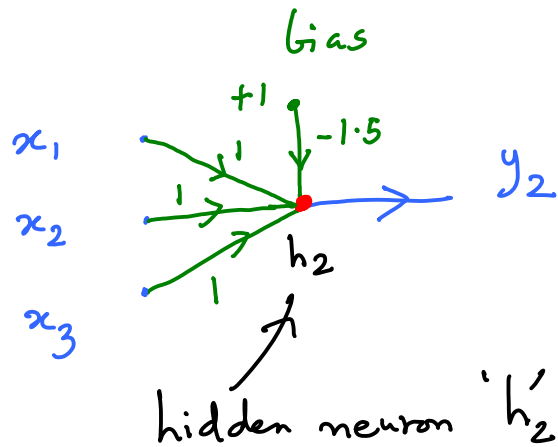
hidden neuron 'h'

$$x_1 = 1 \quad x_2 = 0 \quad x_3 = 0$$

$$y_1((1, 1, 0)) = 1 \quad \text{verify for other cases}$$

2) Let us build a neuron that activates when at least 2 inputs are '1'

I/ps



$$y_2 = \varphi(\underline{w}^T \underline{x} + b_2); \quad b_2 = -1.5$$

Example:

Plug in  $(x_1, x_2, x_3) = (0, 1, 1)$

$$y_2((0, 1, 1)) = 1$$

Plug in  $(x_1, x_2, x_3) = (0, 1, 0)$

$$y_2((0, 1, 0)) = 0$$

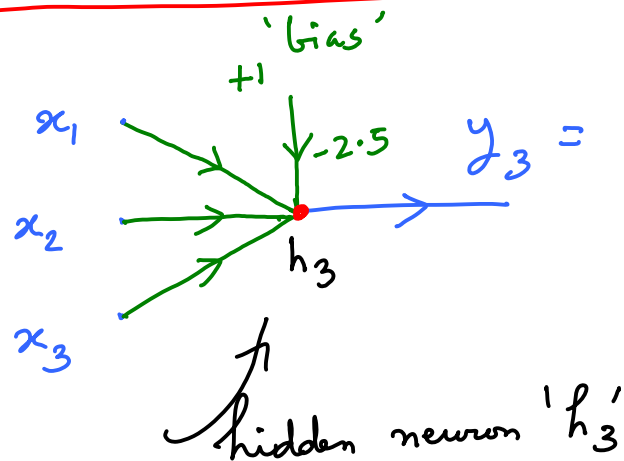
Cross check the rest!



3)

Let us build a neuron that responds when all 3 inputs are '1'

---



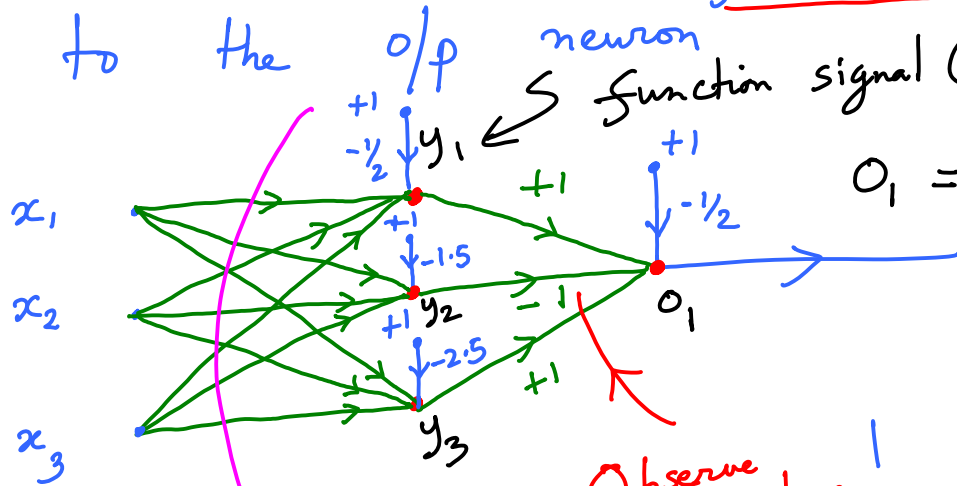
$$y_3 = \varphi(\underline{w}^T \underline{x} + b_3); \quad \underline{b_3} = -2.5$$

Example :

Verify for  $\underline{x} = (1 \ 1 \ 1)$   
 $y_3((1 \ 1 \ 1)) = 1$   
and  $y_3 = 0$  for all  
 $\underline{x} \in \mathbb{Z}_2^3$

Straight forward!

We shall build the full network with connections to the o/p neuron



function signal @ hidden neuron 'h<sub>i</sub>'

$$o_1 = \varphi \left( -y^T \underline{w}^{(1)} + b_0 \right); \underline{b}_0 = -1/2$$

$$\underline{w}^{(1)} = \begin{pmatrix} 1 & -1 & 1 \end{pmatrix}$$

$$-y = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$$

All synaptic wts. are 1

Observe the sign here!

h<sub>1</sub> ≥ 1 i/p's are active  
 h<sub>2</sub> ≥ 2 i/p's are active  
 h<sub>3</sub> = 3 i/p's are active