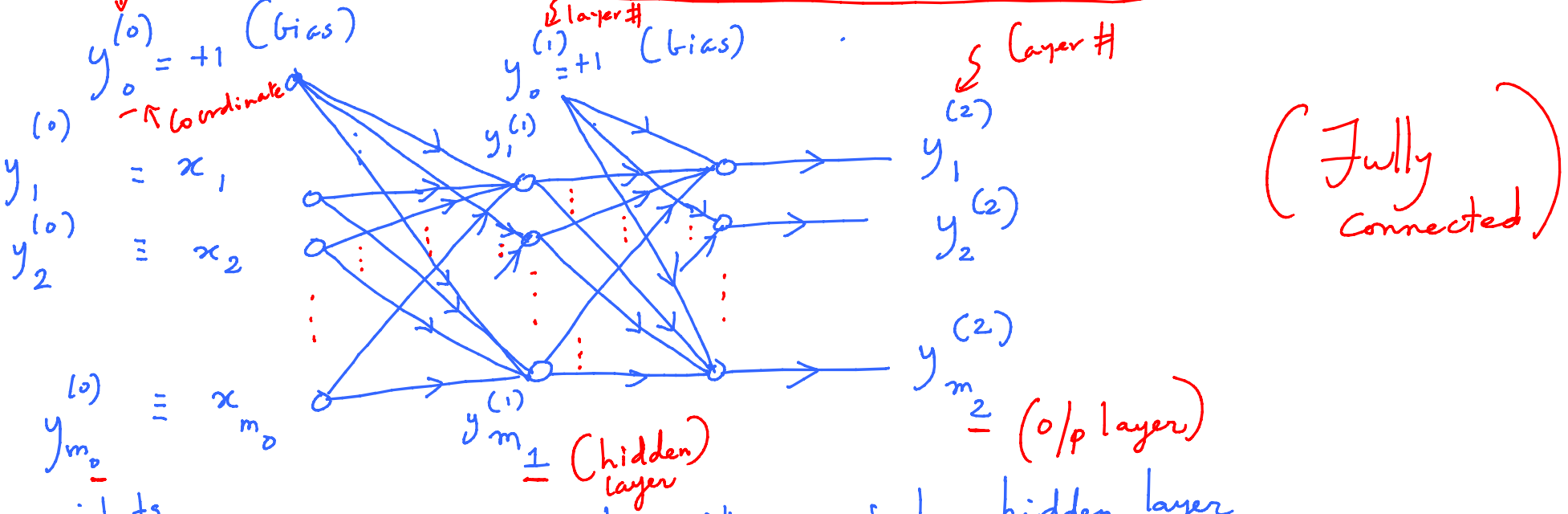
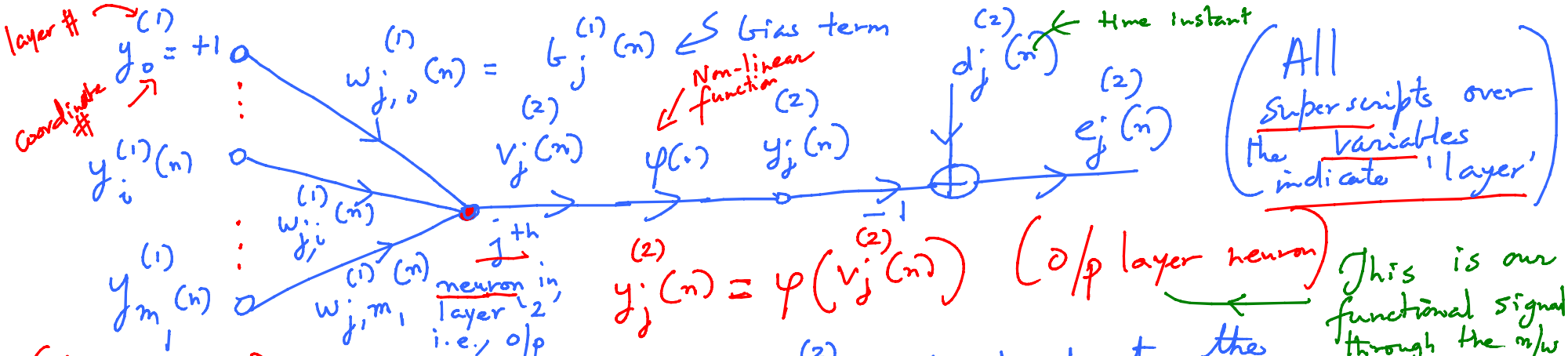
Consider a single layer of hidden neurons for simplicity



(Fully connected)

A network with a single hidden layer

$\varphi(\underline{w}^T \underline{x} + b)$ ← bias



All superscripts over the variables indicate 'layer'

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

(o/p layer neuron)

This is our functional signal through the n/w

$v_j^{(2)}(n)$: The induced local field 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_1 : # neurons in hidden layer excluding bias)

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

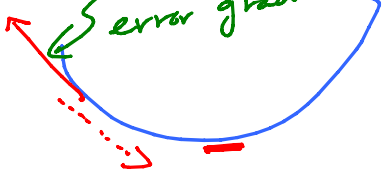
$$y_j^{(2)}(n) = \varphi\left(v_j^{(2)}(n)\right)$$

← local receptive field

We need to apply a correction $\Delta w_{ji}^{(1)}(n)$ ∝ proportional to the derivative

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

Key computation



Performance curve

Idea

$$\underbrace{\frac{\partial \mathcal{E}(n)}{\partial w_{ji}^{(1)}(n)}}_{\text{Sensitivity}} = \frac{\partial \mathcal{E}(n)}{\partial e_j^{(2)}(n)} \frac{\partial e_j^{(2)}(n)}{\partial y_j^{(2)}(n)} \frac{\partial y_j^{(2)}(n)}{\partial v_j^{(2)}(n)} \frac{\partial v_j^{(2)}(n)}{\partial w_{ji}^{(1)}(n)}$$

inst. total error over all the neurons

inst. error at neuron j | function signal | local reception field | Synaptic wt. connecting $y_i^{(1)}$ → i (hidden) | j (o/p)

I | II | III | IV

CHAIN RULE

Now, let us calculate all the partial derivatives

$$\frac{\partial \mathcal{E}(n)}{\partial e_j^{(2)}(n)} = e_j^{(2)}(n) \quad \text{(I)}$$

$$\frac{\partial e_j^{(2)}(n)}{\partial y_j^{(2)}(n)} = -1 \quad \text{(II)} \qquad \left(\because e_j^{(2)}(n) = d_j^{(2)}(n) - y_j^{(2)}(n) \right)$$

$$\frac{\partial y_j^{(2)}(n)}{\partial v_j^{(2)}(n)} = \varphi'\left(v_j^{(2)}(n)\right)$$

$$y_j^{(2)}(n) = \varphi\left(v_j^{(2)}(n)\right) \quad \text{(III)}$$

$$\frac{\partial v_j^{(2)}(n)}{\partial w_{ji}^{(1)}(n)} = y_i^{(1)}(n) \quad \text{(IV)}$$

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

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

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

('η' is the learning rate)

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

neuron 'j'

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

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

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

There are 2 cases

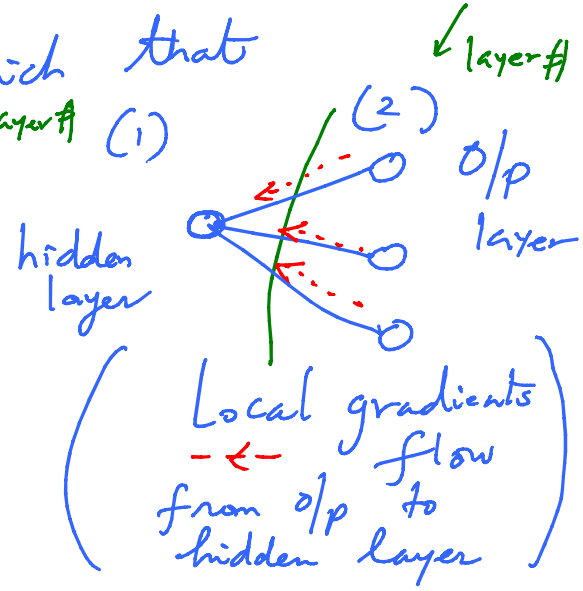
1) Neuron 'j' is an output node
2) Neuron 'j' is a hidden node.

Case 1: When neuron 'j' is located in the o/p layer, it is supplied with a desired response associated with this neuron. We can compute $e_j^{(2)}(n)$ 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'

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

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

(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

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

Observe layer # consistent with the error signal flow graph

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

Term 1 · Term 2 · 'CHAIN RULE'

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

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

$$\triangleq -\varphi'\left(v_k^{(2)}(n)\right) \quad \text{(a)}$$

$$\left\{ \frac{\partial e_k^{(2)}(n)}{\partial v_k^{(2)}(n)} = \right.$$

But,

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

Use (a) & (b) within Term 2

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

Folding all the partial derivatives together

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^{(1)}(n)} = -\sum_k \underbrace{e_k^{(2)}(n)\, \varphi_k'\left(v_k^{(2)}(n)\right)}_{\Downarrow} w_{kj}^{(1)}(n)$$

$w_{kj}^{(1)}(n)$: k^{th} o/p neuron ← j^{th} hidden neuron

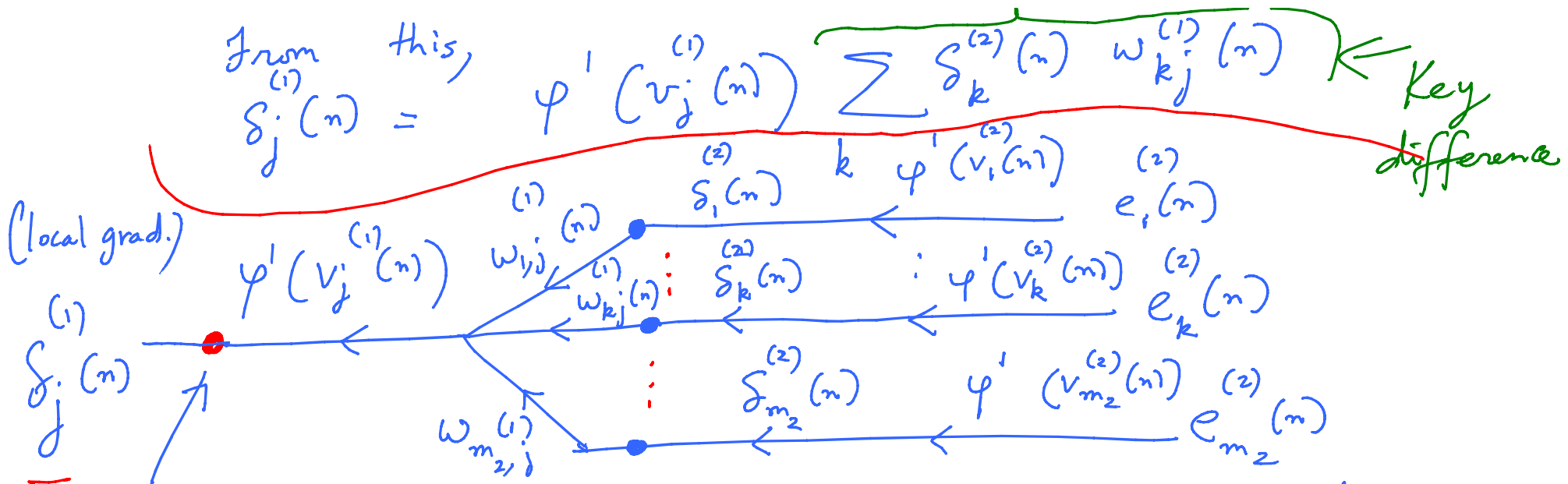
$$= -\sum_k \underbrace{\delta_k^{(2)}(n)}\, w_{kj}^{(1)}(n)$$

local gradient @ node k in the o/p layer

From this,

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

← Key difference



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

"Updation" of weights

In general,

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

↑ Const. usually

There are 2 cases

(a) Neuron is in the hidden layer (Recall: It does not see the error directly)

(b) Neuron is in the o/p layer

Going by the derivation,

For the o/p layer

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

(j: neuron in o/p; i: neuron in hidden connected to neuron j)

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

For the hidden layer

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

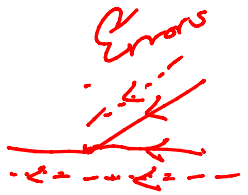
($y_0^{(0)}(n) = 1$ always, i.e., bias — NOTE)

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

1) FORWARD PASS (Function Signals)

2) BACKWARD PASS

(Local error gradients ⇒ 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

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

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

$l = 1, 2$

$y_j^{(l)}(n)$: function signal in prev layer

$v_j^{(l)}(n)$: local reception field

Observe

If neuron 'j' is in the hidden layer

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

$i = 1, \ldots, m_0$

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 'δ' for each neuron.

Activation Function (Single layer)

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

1) Logistic function

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

$v_j^{(l)}(n)$: local reception field

$a > 0$: adjustible quantity

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

$l = 1, 2$

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

$$\varphi'\left(v_j^{(l)}(n)\right) = a\, y_j^{(l)}(n)\left(1 - y_j^{(l)}(n)\right) \quad ; \; l = 1, 2 \qquad \text{(1)}$$

For a neuron 'j' located in the o/p layer

(layer # → (2); o/p)

$$y_j^{(2)}(n) = o_j(n)$$

('o' stands for output)

Plug in (1)

Local gradient

$$\delta_j^{(2)}(n) = e_j^{(2)}(n)\,\varphi'\left(v_j^{(2)}(n)\right)$$

$$= a\left[d_j^{(2)}(n) - o_j^{(2)}(n)\right] o_j(n)\left(1 - o_j(n)\right)$$

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

For an arbitrary hidden node 'j'

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

← Pay attention to the sig. flow graph

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

local gradient, in @ node 'j' the hidden layer

Home Work: Sketch $\varphi(\cdot)$, $\varphi'(\cdot)$

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

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

$$y_j^{(l)}(n) = \varphi\left(v_j^{(l)}(n)\right) = a \tanh\left(b\, v_j^{(l)}(n)\right) \qquad a, b > 0$$

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

$$= a\, b\left(1 - \tanh^2\left(b\, v_j^{(l)}(n)\right)\right) \qquad \left(x^2 - y^2 = (x+y)(x-y)\right)$$

$$= \frac{b}{a}\left(a - y_j^{(l)}(n)\right)\left(a + y_j^{(l)}(n)\right)$$

$$l = 1, 2$$

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

$$\delta_j^{(2)}(n) = e_j^{(2)}(n)\,\varphi'\left(v_j^{(2)}(n)\right)$$

$$= \frac{b}{a}\left[d_j(n) - o_j(n)\right]\left[a - o_j(n)\right]\left[a + o_j(n)\right]$$

$d_j(n) - o_j(n)$ ← directly available

With for a neuron 'j' in the hidden layer

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

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

Local gradients @ hidden nodes

Error is not directly available

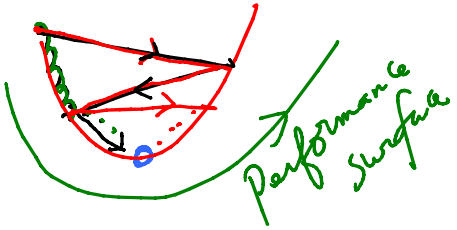
Rate of learning

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

Smaller the value of η $\Rightarrow$ Smaller changes to the synaptic weights

$\Rightarrow$ Smoother trajectory

$\Rightarrow$ Slower rate of learning



Too large η $\Rightarrow$ There can be oscillations & the n/w could be unstable

We shall introduce a momentum term

Observe the time steps

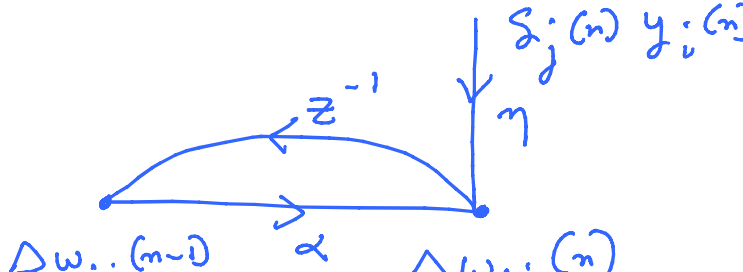
$$\Delta w_{ji}(n) = \alpha \, \Delta w_{ji}(n-1) + \eta \, \delta_j(n) \, y_i(n)$$

Std. term through grad. descent

Recursive eqn (A)

α: controls the rate of decay ($\alpha > 0$)

(1st order Aut reg. process)



Signal Flow Graph

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

$$\delta_j(t) \, y_i(t) = -\frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)}$$

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: $0 < \alpha < 1$

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

$\Longrightarrow$ Stabilizing effect.

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

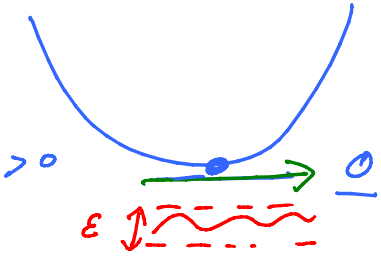
Pros: More flexibility $\Longrightarrow$ Better approx (conv.).

Cons: Difficulty to control (Non linear dynamics)

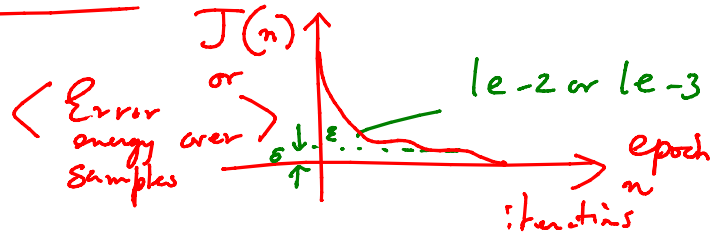
Stopping criterion

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

(a) Compute $\nabla_{\underline{w}} \mathcal{E}$ s.t. $\| \underline{w}(n+1) - \underline{w}(n) \|_2 < \varepsilon$, $\varepsilon > 0$



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



Summary of the back propagation Algo

1) **Initialization** : Assuming no apriori 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 $(\underline{x}(n), \underline{d}(n))$ ← $\underline{x}(n)$: feature vector; $\underline{d}(n)$: desired response vector

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

Compute the induced local fields and function signals of the n/w in a feed forward manner layer-by-layer

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

m_{l-1} ← # of neurons in the $(l-1)^{st}$ layer excluding 'bias'

$w_{ji}^{(l-1)}$ ← Synaptic wts.

$y_i^{(l-1)}(n)$ ← function signal of neuron 'i' in $(l-1)^{st}$ layer @ iteration 'n'

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

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

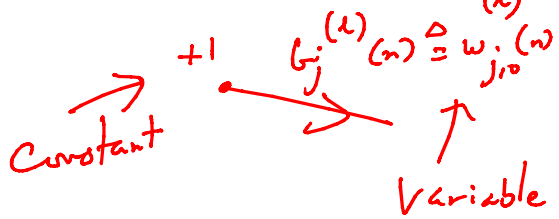
$$\begin{cases} w_{j,0}^{(l)}(n) = b_j^{(l)}(n) \leftarrow \text{Bias initialization} \\ y_0^{(l-1)} = +1 \end{cases}$$

(Hard wire '1')

For the o/p layer, $l = L$

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

o/ps



Compute the error signal

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

$o_j(n)$ ← Evaluate using the feed forward process

4) Backward computations

Key step: Compute δ_s over all neurons in the layers

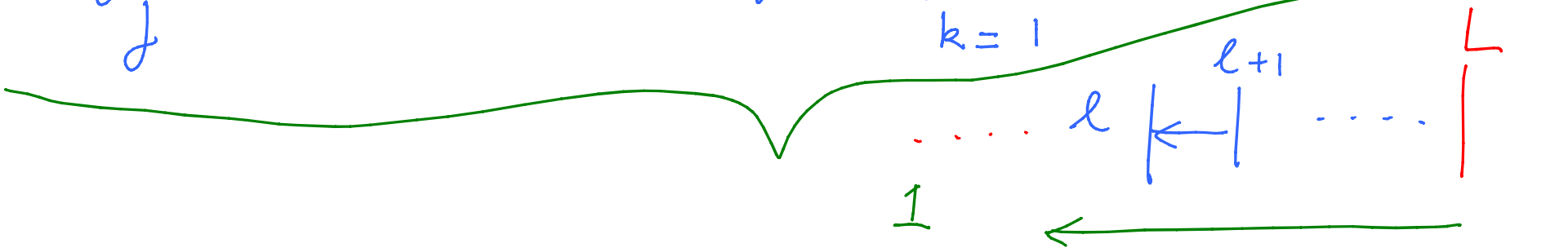
For the last layer $l = L$

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

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

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

m_{l+1} = # neurons in the $(l+1)$st layer



Adjust the synaptic wts. as per the general Δ-rule

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

momentum push

Gradient descent

α : momentum

η : learning rate

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

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 update
   - ↓ 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

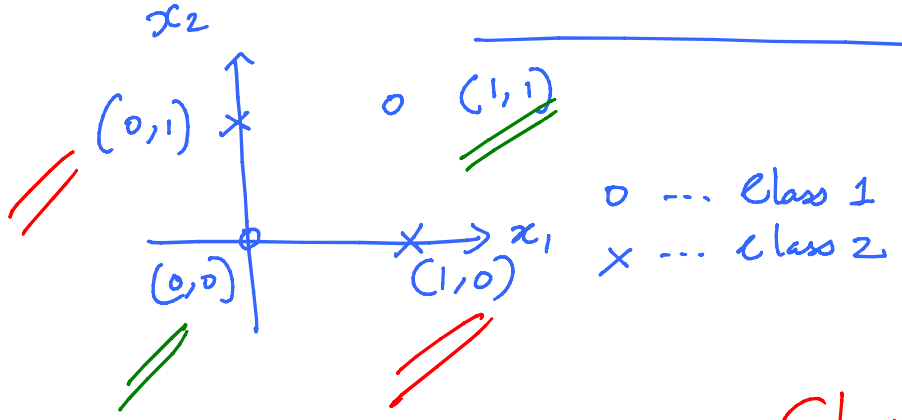
4) Choice of activation function & range
   - Preferably within the range of sig. linear region

5) Normalizing inputs

6) Wt. initialization

7) Adaptive learning rates

XOR Problem



We have four corners of a unit square

Class 1

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

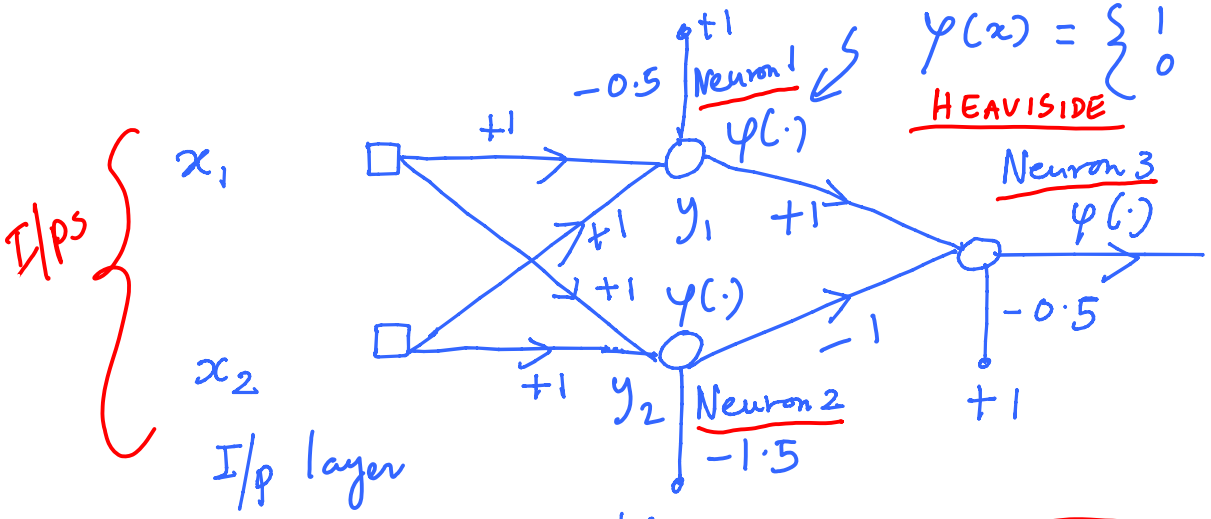
$1 \oplus 1 = 0$

Class 2

All the zeros are class '1'

All the ones are class '2'

One cannot get linear separability with just 1 hyperplane



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

HEAVISIDE

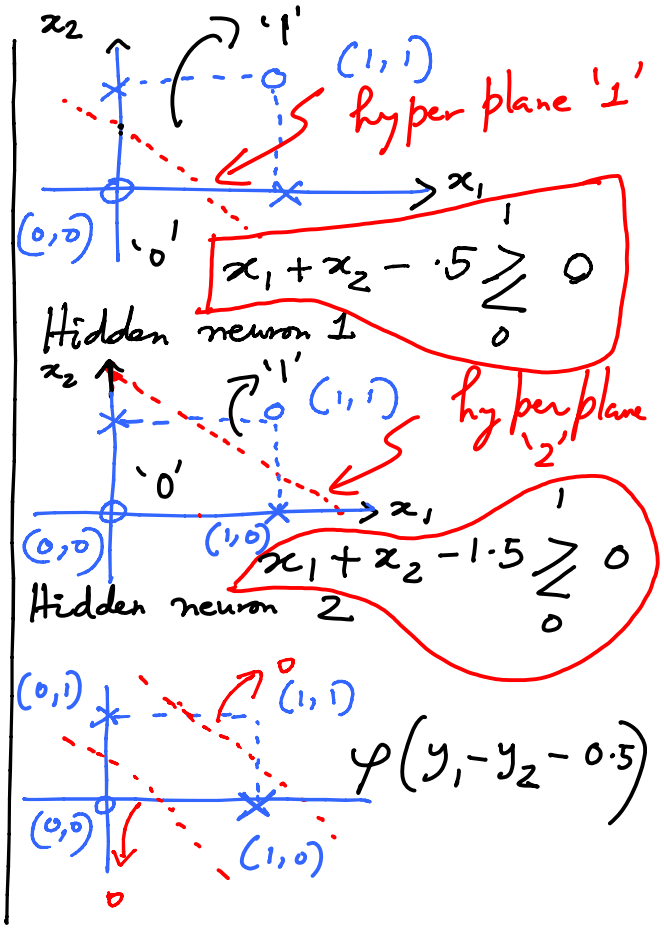
Hidden parameters

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

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

O/p parameters $w_{31} = +1 \qquad w_{32} = -1$

$b_3 = -1/2$



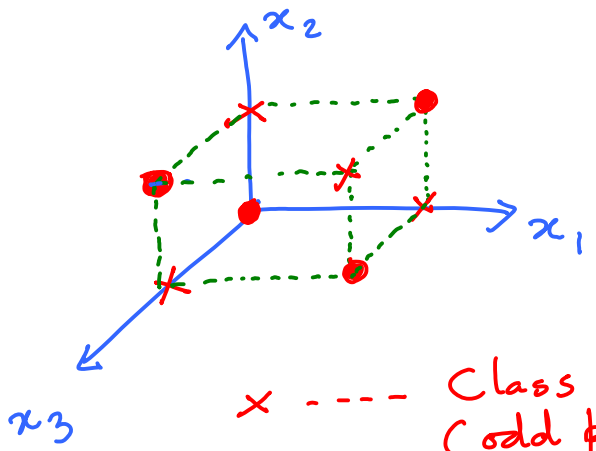
3 - Variable XOR Problem

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	1	Odd parity
	1	0	1	0		1	0	0	1	
	1	1	0	0		0	1	0	1	
	0	1	1	0		1	1	1	1	



× ---- Class 1 (odd parity)

● ---- Class 2 (even parity)

Sketch in 3D space

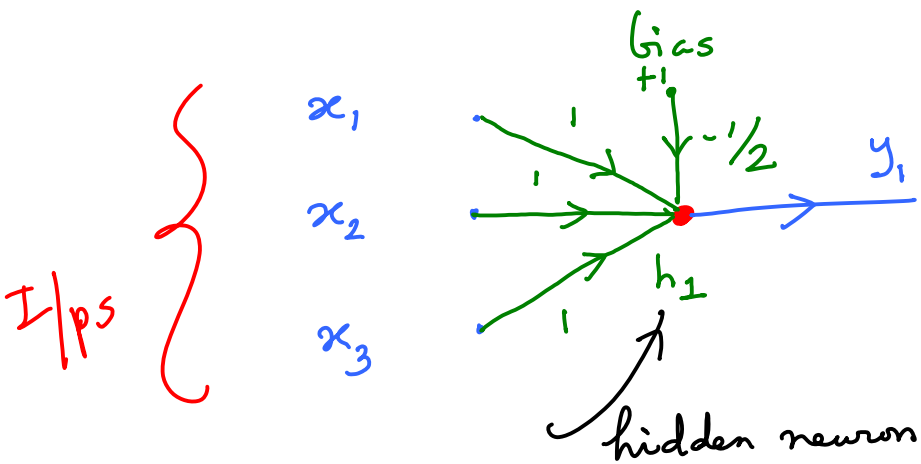
We can clearly see that the classes are not linearly separable!

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

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

$\varphi(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$ is the non-linear activation function

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



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

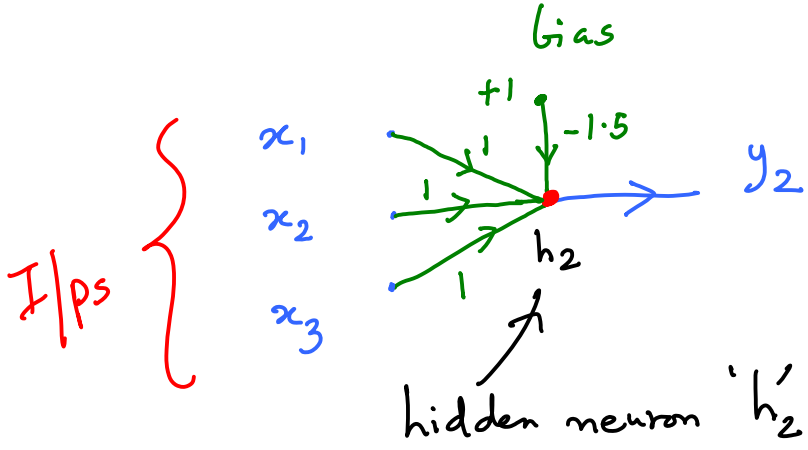
$\underline{w} = [1\ 1\ 1]$

$\underline{x}$ is any tuple in Z_2^3 (binary strings of length 3)

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

$y_1((1\ 1\ 0)) = 1$ verify for other cases

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



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

Example:

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

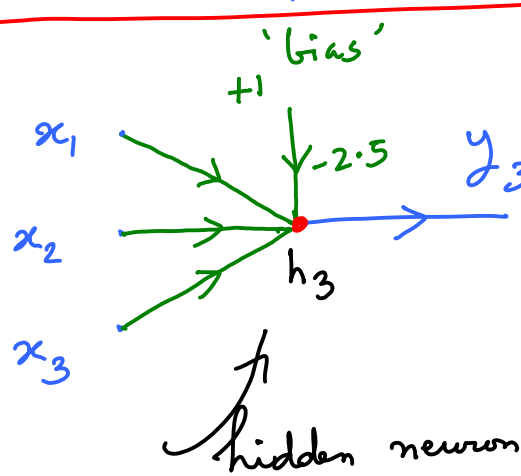
$y_2((0\ 1\ 1)) = 1$

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'



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

Example :

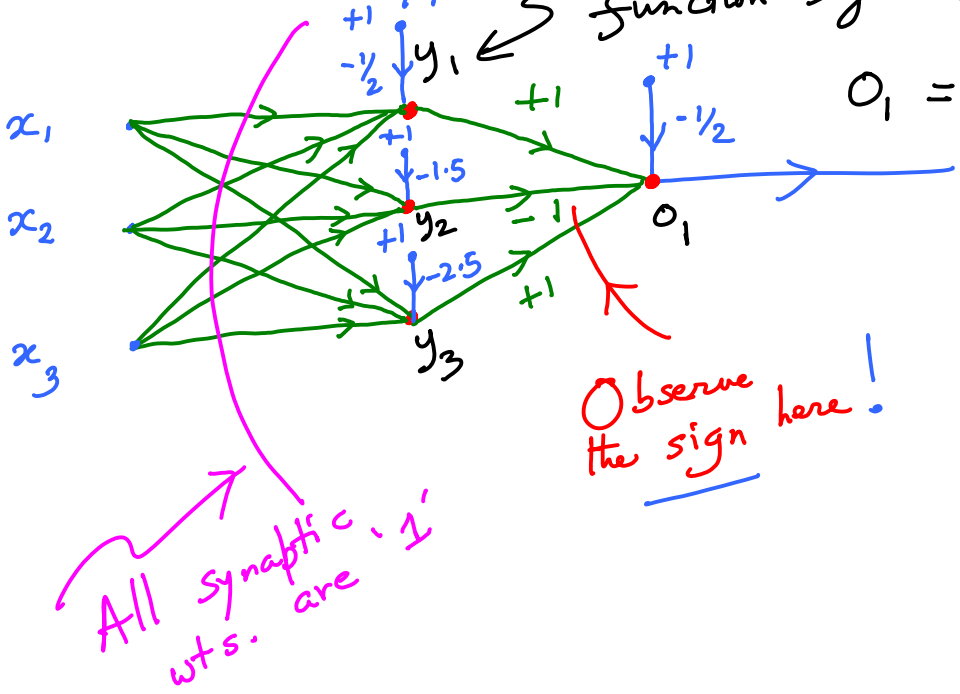
Verify for $\underline{x} = (1\ 1\ 1)$

$y_3((1\ 1\ 1)) = 1$

and $y_3 = 0$ for all $\underline{x} \in Z_2^3$

Straight forward !

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



y_i ← function signal @ hidden neuron 'i'

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

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

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

$h_1 \geq 1$ i/ps are active

$h_2 \geq 2$ i/ps are active

$h_3 = 3$ i/ps are active