Summary of the Back Propogation Algorithm.

→ Note: Using sequential, or stochastic mode
In this mode, weight updation is to be performed after the presentation of each training example. An Epoch is complete after one complete presentation of the entire training set during the learning process.

In this sense, sequential mode operates by adjusting ntwk weights after the presentation of each training example within the epoch.

Advantages of this mode:
→ The algorithm is simple to implement
→ It provides effective soln's to large & difficult problems.

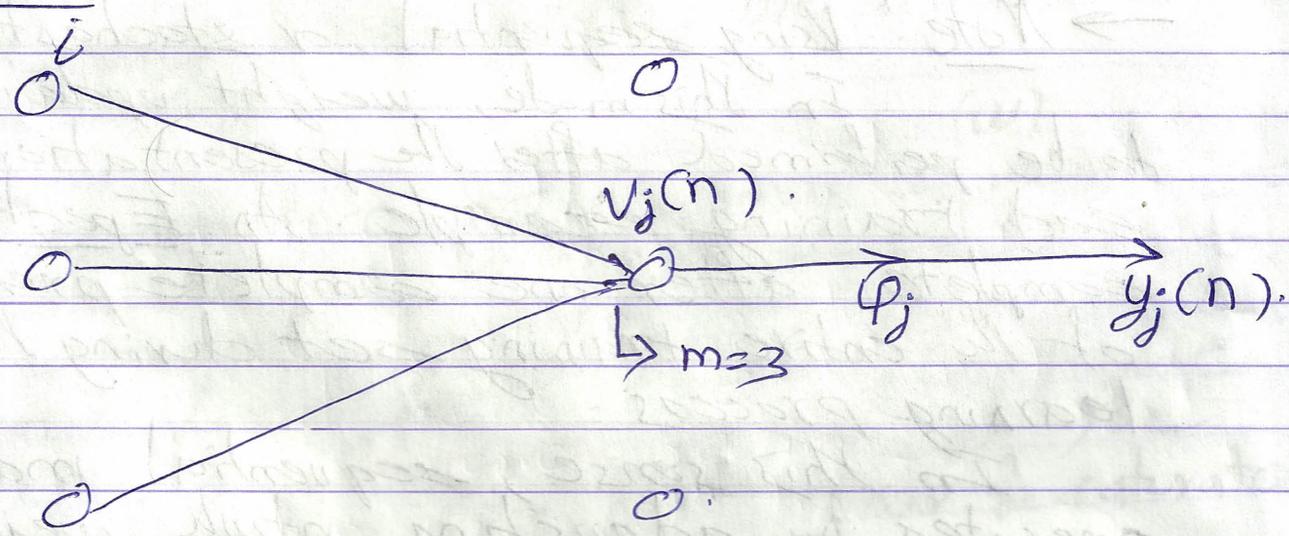
Algorithm:

I] Initialization.

• Assuming no prior information is available, pick the synaptic weights & thresholds from a uniform distribution whose mean is zero & whose variance is chosen to make the standard deviation of the induced local fields of the neurons lie at the transition betn the linear & saturated parts of the sigmoid activation functn.

(I)

For explanation, turn over:



where: $v_j(n)$: Sum of all i/ps at neuron n in j^{th} layer.

m: No. of i/ps to a given neuron

φ_j: Activation funct^n

y_j: o/p of neuron j

The induced Local field $v_j(n)$ produced at the ~~o/p~~ i/p of the activation funct^n of neuron j is:

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

$w_{ji}(n)$: weight of connection from i^{th} to j^{th} neuron layers, to n^{th} neuron in j^{th} layer, i.e., current neuron.

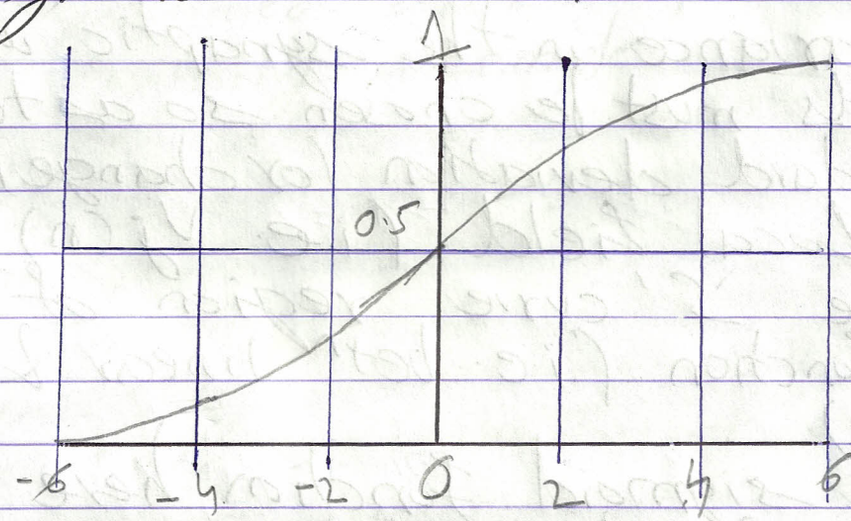
&

m is the total number of i/ps (here, 3).

→ The synaptic weight w_{j0} equals the bias b_j applied to neuron j. Hence, funct^n s/g appearing at the o/p of neuron j at iteration n is $y_j(n)$

Now, to understand the initialization statement:

a] Sigmoid Function:



As seen above, a sigmoid function is a mathematical function having a 'S' shaped curve (sigmoid curve).

Often, the sigmoid function refers to the special case of the logistic function (shown above) & defn by:

$$S(I) = \frac{1}{1+e^{-I}} \quad \text{— (a)}$$

Now, refering to statement (I), we have:

→ synaptic weights: $w \rightarrow w_{ji}$

→ thresholds

→ Induced Local Field: $V_j(n)$

↳ this is produced at the i/p of the activation funct^n, i.e. φ_j, to produce y_j

$$\therefore y_j(n) = \varphi_j(v_j(n))$$

Considering these points, we understand the following:

The variance in the synaptic weights (w) & thresholds must be chosen so as to make the standard deviation (or change) in the induced local field (i.e. $v_j(n)$) lie within the 'S' curve region of the sigmoid function (i.e. betⁿ linear & saturated regions)

The sigmoid function here is the activation function (φ_j)

∴ φ_j must lie within the 'S' shaped curve.

Conclusion: Synaptic weights & threshold values must be chosen so as to ensure v_j (the locally induced field) when applied to φ_j must lie between, or within the 'S' shaped curve.

II] Presentations of Training Examples:

→ Present the network with an epoch of training examples. For each example set perform the sequence of forward & backward computations described below in points II & IV.

→ Present the examples in an epoch in a random order each epoch!

III] Forward Computation:

⇒ Let a training example in the epoch be denoted by

$$(x(n), d(n))$$

where:

$x(n)$ → i/p vector applied to i/p nodes

$d(n)$ → desired o/p presented to the o/p layer.

→ Compute the induced local fields & funct^n s/grs by proceeding forward through the nlw layer-by-layer

⇒ Now, recall eq^n (4):

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

→ Thus the induced local field for a neuron (v_j) j in the layer L is:

$$v_j^{(L)}(n) = \sum_{i=0}^{m_0} w_{ji}^{(L)}(n)\, y_i^{(L-1)}(n)$$

where,

$v_j^{(L)}(n)$ → Induced local field of j^{th}, i.e., current neuron in layer L.

m → Total no. of i/ps.

$w_{ji}^{(L)}(n)$ → weight of connection from i^{th} to j^{th} neuron, i.e., from neuron to previous layer to that in current layer

& $y_i^{(L-1)}(n)$ → o/p functn signal of i^{th} neuron in the previous layer, i.e. $(L-1)$ layer.

Note: For $i = 0$, i.e. for i/p layer:

$$y_i^{(L-1)}(n) = y_0^{(L-1)}(n) = +1$$

&

$$w_{ji}^{(L)}(n) = w_{j0}^{(L)}(n) = b_j^{(L)}(n)$$

↳ i.e., bias appld to current neuron.

→ Assuming the use of a sigmoid functn, the o/p of neuron j in layer L is:

$$y_j^{(L)} = \varphi_j(v_j(n))$$

→ If neuron j is in the first hidden layer, i.e. $L = 1$, set:

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

→ i.e., o/p funct^n s/g of pr neuron in previous (in this case input) Layer is the same as the i/p vector $x(n)$.

→ Now, if neuron is in the o/p layer, i.e. $L = L$ ($L \rightarrow$ depth of the n/w)

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

→ Now, finally, compute the error s/g:

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

IV] Backward computation.

→ Involves computing the local gradients, i.e., δ's of the n/w.

→ Recall eq^n's (14) & the final BPN eq^n (25):

$$\delta_j(n) = e_j(n)\, \varphi_j'(v_j(n))$$

→ For a neuron j in o/p layer L:

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

△

→ For a neuron j in hidden layer $\neq L$.

$$\delta_j^{(L)}(n) = \varphi_j'\left(v_j^{(L)}(n)\right) \sum_k \delta_k^{(L+1)}(n)\, w_{kj}^{(L+1)}(n).$$

$k \Rightarrow$ neuron k in the next $(L+1)$ layer.

⟹ Readjust synaptic weights (w) of the n/w in layer L according to the generalised delta rule:

$$w_{ji}^{(L)}(n+1) = w_{ji}^{(L)}(n) + \alpha\left[w_{ji}^{(L)}(n-1)\right] + \eta\, \delta_j^{(L)}(n)\, y_i^{(L-1)}(n).$$

where:

$\eta \rightarrow$ learning rate parameter.

$\alpha \rightarrow$ Momentum constant.

↳ (Explanations to follow) ↲

Hence, the functional signal, $y_j(n)$ appearing at the o/p of neuron j at iteration n is:

$$y_j(n) = \varphi_j(v_j(n)).$$

⇒ Activation function:-

→ The computation of the δ (Eq (25)) for each neuron of the multilayer perceptron ntw requires knowledge of the derivative of the activation functn $\varphi(\cdot)$ associated with that neuron.

→ For this derivative to exist, $\varphi(\cdot)$ must be continuous.

⇒ In basic terms, differentiability is the only requirement that an activation function must satisfy.

→ Logistic Function.

Refering to eqn (4) of the sigmoid function the logistic function is defined as:

$$S = \frac{1}{1+e^{-v}}.$$

For function $\varphi_j(v_j(n))$, i.e, the activation function over the locally induced field, we have

... P.T.O.

$$\varphi_j(v_j(n)) = \frac{1}{1+\exp(-a v_j(n))} \quad a>0 \;\&\; -\infty < v_j(n) < \infty \qquad \text{(6)}$$

According to this non-linearity, the amplitude (value) of the o/p lies inside the range of $0 \leq y_j \leq 1$

Differentiating eq^n (6) w.r.t. $v_j(n)$, we get:

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

↳ [explanation: $\frac{d}{dx}\frac{1}{u} = \frac{1}{u^2}\cdot\frac{du}{dx}$ & $\frac{d}{dx}e^u = e^u \cdot \frac{du}{dx}$]

→ with $y_j(n) = \varphi_j(v_j(n))$, we may eliminate the exponential term $\exp(-a v_j(n))$. & so express the derivative as

$$\varphi_j'(v_j(n)) = a y_j(n)[1 - y_j(n)] \qquad \text{(7)}$$

↳ (?)

Now, remember eq^n (14), i.e.:

$$\delta_j(n) = e_j(n)\,\varphi_j'(v_j(n))$$

Now, consider a neuron j located in the o/p layer, i.e.,

$$y_j(n) = o_j(n)$$

Hence, the local gradient for this neuron will be:

$$\delta_j(n) = e_j(n)\, \varphi_j'(v_j(n))$$

Substituting eq^n's (2) & eq^n (1) up there:

$$\delta_j(n) = a\, y_j(n)[1 - y_j(n)][d_j(n) - y_j(n)]$$

As neuron in the o/p layer, $y_j(n) = o$.

$$\therefore \delta_j(n) = a[d_j(n) - o_j(n)]\, o_j(n)[1 - o_j(n)]$$

$d_j \rightarrow$ Desired Response.

On the other hand, for an arbitrary neuron j,

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

i.e. eq^n (2), or main BPN eq^n.

... P.T.O.

& now from eq" (2).

$$\delta_j(n) = a y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n)$$

... neuron j is hidden.

→ Note from eq" (6) that the derivative $\varphi'_j(v_j(n))$ attains:

↳ Maximum value at: $y_j(n) = 0.5$

Minimum value (0) at: $y_j(n) = 0$ or $y_j(n) = 1.0$

→ The amount of change in a synaptic weight of the network is proportional to the derivative $\varphi_j'(v_j(n))$.

⇒ It follows that for a sigmoid activation function the synaptic weights are changed most for the neurons in the network where the function signals are in their midrange

⇒ It has been said that it is this feature of back-propagation learning that contributes to its stability as a learning algorithm.

V.] Iteration:

→ Iterate the forward & backward computations by presenting new epochs of training examples to the nlw until the stopping criterion is met.

Notes:

↳ As stated earlier, randomize the presentation of training examples

& ↳ Momentum & learning-rate parameters are typically adjusted & usually decreased as the no: of iterations increase.

⇒ Delta Rule:

In machine learning, the delta rule is a gradient descent learning rule for updating the weights of the i/ps to artificial neurons.

η → Learning Parameter.

↳ The smaller it is, the smaller the changes to the synaptic weights will be betn iterations. This improved learning is obtained at the cost of a slower learning rate.

If η is too large, learning speeds up but large changes to the synaptic weights make the network unstable.

⇒ A simple method of increasing the rate of learning yet avoiding the danger of instability is to modify the delta rule of eqn (13) & (15) by including a momentum term.

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

$\alpha \rightarrow$ Momentum, usually +ve.
↳ Controls the feedback loop acting around $\Delta w_{ji}(n)$.

Eq (15)

$$\Delta w_{ji}(n) = \eta \delta(n) y_i(n)$$

may be viewed as a special case of the delta rule, with $\alpha = 0$.

(d) is called the generalised delta rule.

⇒ Two passes of Computation.

→ In the forward pass, synaptic weights remain unaltered.

→ Functn signals (y) are computed for each neuron.

$$y_j(n) = \varphi(v_j(n))$$

$v_j(n)$ → Induced local field

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

→ For neuron j in first hidden layer.

$$y_i(n) = x_i(n)$$

$x_i(n)$ → i/p vectors i^{th} element.

→ For neuron in the o/p layer:

$$y_i(n) = o_j(n).$$

$o_j(n)$ → j^{th} element of o/p vector.

→ Compare $o_j(n)$ with $d_j(n)$

$d_j(n)$ → desired ~~respo.~~ response

→ By comparing $o_j(n)$ with $d_j(n)$, obtain error $e_j(n)$

→ Thus, forward phase begins at first hidden layer & terminates at o/p layer
First hidden layer → Present i/p vector
o/p layer → Compute error for each neuron of this layer.

⇒ Backward Pass starts at o/p layer

→ Pass error s/g leftward through the n/w, layer by layer.

→ Recursively compute δ (local gradient) for each neuron.

$$\delta_j(n) = e_j(n)\,\varphi_j'(v_j(n)).$$

→ Using $\delta_j(n)$, compute change of synaptic weights in accordance with delta rule.

$$\Delta w_{ji}(n) = \eta\,\delta_j(n)\,y_i(n)$$
... when $\alpha = 0$

&

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

P.T.O. ...

→ For o/p layer neurons:

$$\delta_j(n) = e_j(n)\, \varphi_j'(v_j(n)).$$

$$\Delta w_{ji}(n) = \eta\, \delta_j(n)\, y_i(n).$$

→ For neurons in penultimate layer:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n).$$

& accordingly calculate $\Delta w_{ji}(n)$

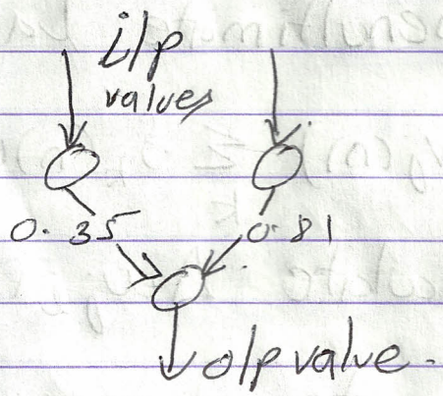
That is, we compute changes to the weights of connections feeding into current neuron.

→ Recursively compute layer by layer, propagating changes to all synaptic weights in the n/w.

Forward Propagation.

→ Set all weights to random values ranging from −1.0 to +1.0

Example: Consider a 2-layered perceptron.



i/p's: 0 1 → 0 (Targets)
1 1 → 1

Randomly set weights: 0.35 & 0.81.
η → Learning Rate → 0.25

a) o/p of i/p layer is same as i/p values

b) Activate neurons in 2nd layer. In this case, the o/p layer.

1 neuron: 2 i/ps.

1) For i/p {0 1}

i/p 1 = 0.35 × 0 = 0
i/p 2 = 1 × 0.81 = 0.81.
Adding = 0 + 0.81 = 0.81.

target = 0
subtracting from target = −0.81

Value for changing wt. 1 = 0.25 x 0 x (-0.81) = 0

Value for changing wt. 2 = 0.25 x 1 x (-0.81) = -0.2025

change in wt 1 → 0 + 0.35 = 0.35 (no change).

change in wt 2 → 0.81 + (-0.2025) = 0.6075

2]

Now for i/p {1 1}

i/p 1 = 0.35 x 1 = 0.35

i/p 2 = 0.6075 x 1 = 0.6075.

Adding = 0.9575.

Target = 1.

subtracting from target = 1 - 0.9575 = 0.0425.

Value for changing wt. 1 = 0.25 x 1 x 0.0425 = 0.010625.

Value for changing wt. 2 = 0.25 x 1 x 0.0425 = 0.010625.

change in wt 1 → 0.35 + 0.010625 = 0.360625

change in wt 2 → 0.6075 + 0.010625 = 0.618125.

#3] Compute net error.

$(-0.81)^2 + (0.0425)^2 = 0.65740625$.

Continue until $O(\approx)$.