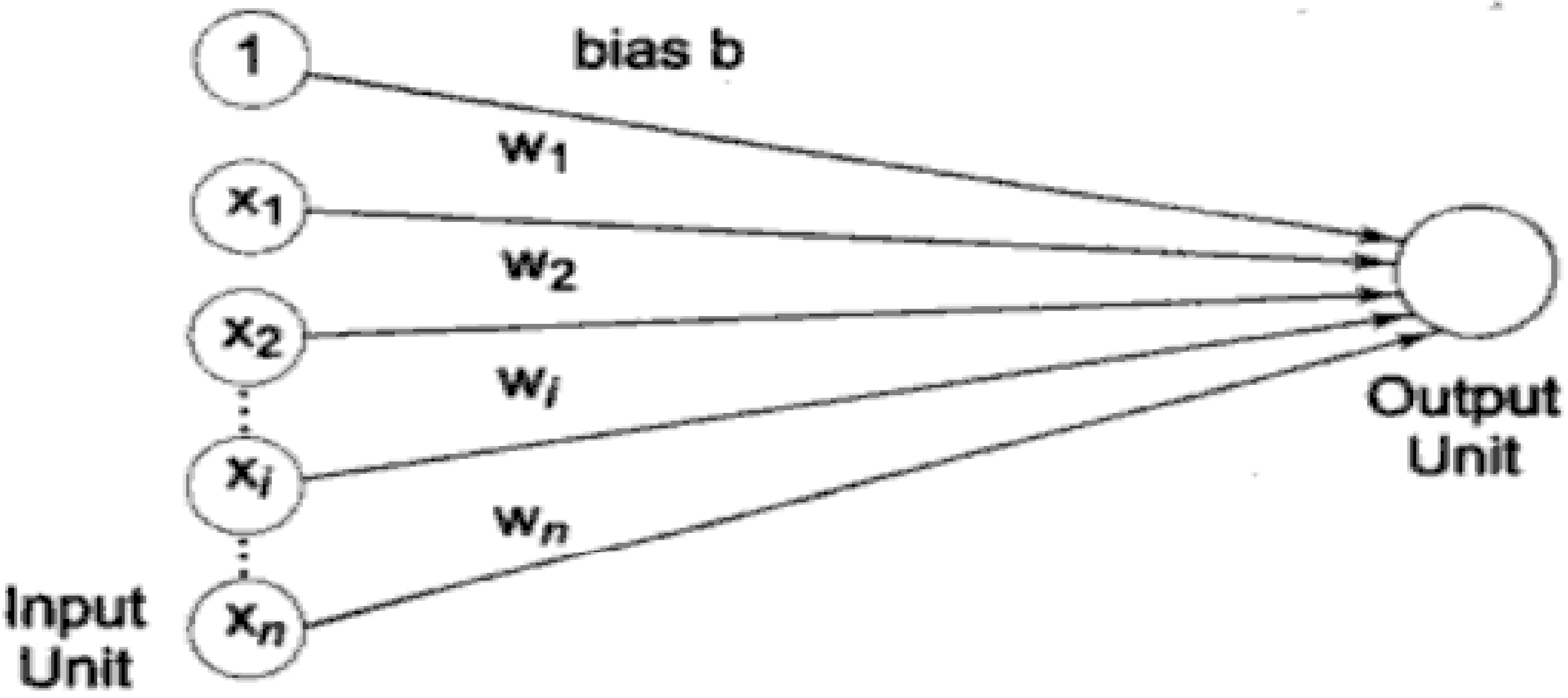
Unit – III

A Survey of Neural Network Model

Single Layer Perceptron

- Perceptron the first adaptive network architecture was invented by Frank Rosenblatt in 1957.
- It can be used for the classification of patterns that are linearly separable.
- Fundamentally, it consists of a single neuron with adjustable weights and bias.
- This algorithm is suitable for binary/bipolar input vector with bipolar target

Architecture of Single Layer Perceptron



Algorithm

- Step 1: Initialize all weights and bias to Zero and set learning rate α ((0 to 1)
  - $w_i = 0$ for i =1 to n where n is number of input neurons and b =0.
- Step 2: While stopping condition is false do step 3-7.
- Step 3: For each input training pair s:t do steps 4-6.
- Step 4: Set activation for input units with the input vectors.
  - $x_i = S_i$ (i= 1 to n)
- Step 5: Compute the output unit response

$$Y_{in} = b + \sum_i x_i w_i$$

- The activation function is used is

$$y = f(y_{in}) = \begin{cases} 1, & \text{if} \quad y_{in} > \theta \\ 0, & \text{if} \quad -\theta \le y_{-in} \le \theta \\ -1, & \text{if} \quad y_{-in} < -\theta \end{cases}$$

Algorithm…

- Step 6: The weights and bias are updated if the target is not equal to the output response.
- If $t \neq y$ and the value of x_i is not zero
  - $w_{i(new)} = w_{i(old)} + \alpha x_i t$
  - $b_{(new)} = b_{(old)} + \alpha t$
- Else
  - $w_{i(new)} = w_{i(old)}$
  - $b_{(new)} = b_{(old)}$
- Step 7 : test for stopping condition

Problems

1. Apply the Perceptron to the training the pattern that define AND function with bipolar inputs and targets.
2. Apply the Perceptron to the training patterns that define OR function input and target.
3. Classify the two dimensional input patterns (representing letters) using the Perceptron rule (The T-C Problem).
4. Classify the two dimensional input patterns (representing letters) using the Perceptron rule (The I- J Problem).
5. Apply the Perceptron to the training patterns that define XOR function input and target.

Application Procedure

- Step 1: The weights to be used here are taken from the training algorithm
- Step 2: For each input vector x to be classified do steps 3- 4.
- Step 3: Input units activations are set
- Step 4: Calculate the response of output unit,

$$y_{-in} = \sum_i x_i w_i$$

$$y = f(y_{-in}) = \begin{cases} 1, & \text{if} \quad y_{-in} > \theta \\ 0, & \text{if} \quad -\theta \le y_{-in} \le \theta \\ -1, & \text{if} \quad y_{-in} < -\theta \end{cases}$$

More Problem

- For the following noisy version of training patterns, identify the response of network by segregating it into correct, incorrect, or indefinite.
- (0 -1 -1) , (0 1 -1), (0 0 1), (0 0 -1), (0 1 0), (1 0 1),
- (1 0 -1), (1 -1 0), (1 0 0), (1 1 0), (0 -1 0), (1 1 1)
- Solution (Hints)
  - If $x_1w_1 + x_2w_2 + x_3w_3 > 0$ then the response is correct
  - If $x_1w_1 + x_2w_2 + x_3w_3 < 0$ then the response is incorrect
  - If $x_1w_1 + x_2w_2 + x_3w_3 = 0$ then the response is indefinite

More Problem..

- Using the perceptron learning rule, find the weights required to perform the following classifications. Vector (1 1 1 1), (-1 1 -1 -1) and (1 -1 -1 1) are the member of class (having target value 1); Vectors (1 1 1 -1) and (1 -1 -1 1) are not the members of class (having target value -1). Use learning rate of 1 and starting weights of zero (0). Using each of the training and vectors as input, test the response of the net.

Least Mean Square

- This rule is refer Delta Learning Rule or Widrow Hoff Rule
- The delta rule is valid only for continuous activation functions and in the supervised training mode.
- The learning signal for this rule is called delta.
- The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse.
- The delta rule can be applied for single output unit and several output unit.

Derivation of Delta Rule

- The delta rule changes the weight of the connections to minimize the difference between the net input to the output unit, $\mathbf{y_{in}}$ and the target value **t.**

- The delta rule is given by

- $\Delta w_i = \alpha(t - y_{in})\, x_i$
  - Where x is the vector of activation of input units.
  - y_{in} is the net input to output unit
  - t is the target vector
  - α is the learning rate

Derivation of Delta Rule…

- The mean square error for a particular training pattern is $E = \sum_j (t_j - y_{inj})^2$

- The gradient of E is a Vector consisting of the partial derivatives of E with respect to each of the weights. The error can be reduced rapidly by adjusting weight w_{ij}

- Taking partial differentiation of E w.r.t. w_{ij}

$$\frac{\partial E}{\partial w_{ij}} = \sum_i (t_j - y_{inj})^2$$

$$= \frac{\partial}{\partial w_{ij}} (t_j - y_{inj})^2$$

Derivation of Delta Rule…

- Since the weight w_{ij} influence the error only at output unit y_J

$$y_{in\,J} = \sum_{i=1}^{n} (t_J - y_{in\,J})^2$$

$$\frac{\partial E}{\partial w_{IJ}} = \frac{E}{\partial w_{IJ}} (t_J - y_{in\,J})^2$$

$$= 2\,(t_J - y_{in\,J})\,(-1)\,\frac{\partial y_{in\,J}}{\partial w_{IJ}}$$

$$\frac{\partial E}{\partial w_{IJ}} = -2\,(t_J - y_{in\,J})\,\frac{\partial y_{in\,J}}{\partial w_{IJ}}$$

$$\frac{\partial E}{\partial w_{IJ}} = -2\,(t_J - y_{in\,J})\,x_I$$

- Thus the error will be reduced rapidly depending upon the given learning by adjusting the weights: $\Delta w_{IJ} = \propto (t_J - y_{in\,J})\, x_I\,.$

Multilayer Perceptron

- Hidden layers of computation nodes
- input propagates in a forward direction, layer-by-layer basis
  - also called Multilayer Feedforward Network, MLP
- Error back-propagation algorithm
  - supervised learning algorithm
  - error-correction learning algorithm
  - Forward pass
    - o input vector is applied to input nodes
    - o its effects propagate through the network layer-by-layer
    - o with fixed synaptic weights
  - backward pass
    - o synaptic weights are adjusted in accordance with error signal
    - o error signal propagates backward, layer-by-layer fashion

MLP Distinctive Characteristics

$$y_i = \frac{1}{1+\exp(-v_j)}$$

- Non-linear activation function
  - differentiable
  - sigmoidal function, logistic function
  - nonlinearity prevent reduction to single-layer perceptron
- One or more layers of hidden neurons
  - progressively extracting more meaningful features from input patterns
- High degree of connectivity
- Nonlinearity and high degree of connectivity makes theoretical analysis difficult
- Learning process is hard to visualize
- BP is a landmark in NN: computationally efficient training

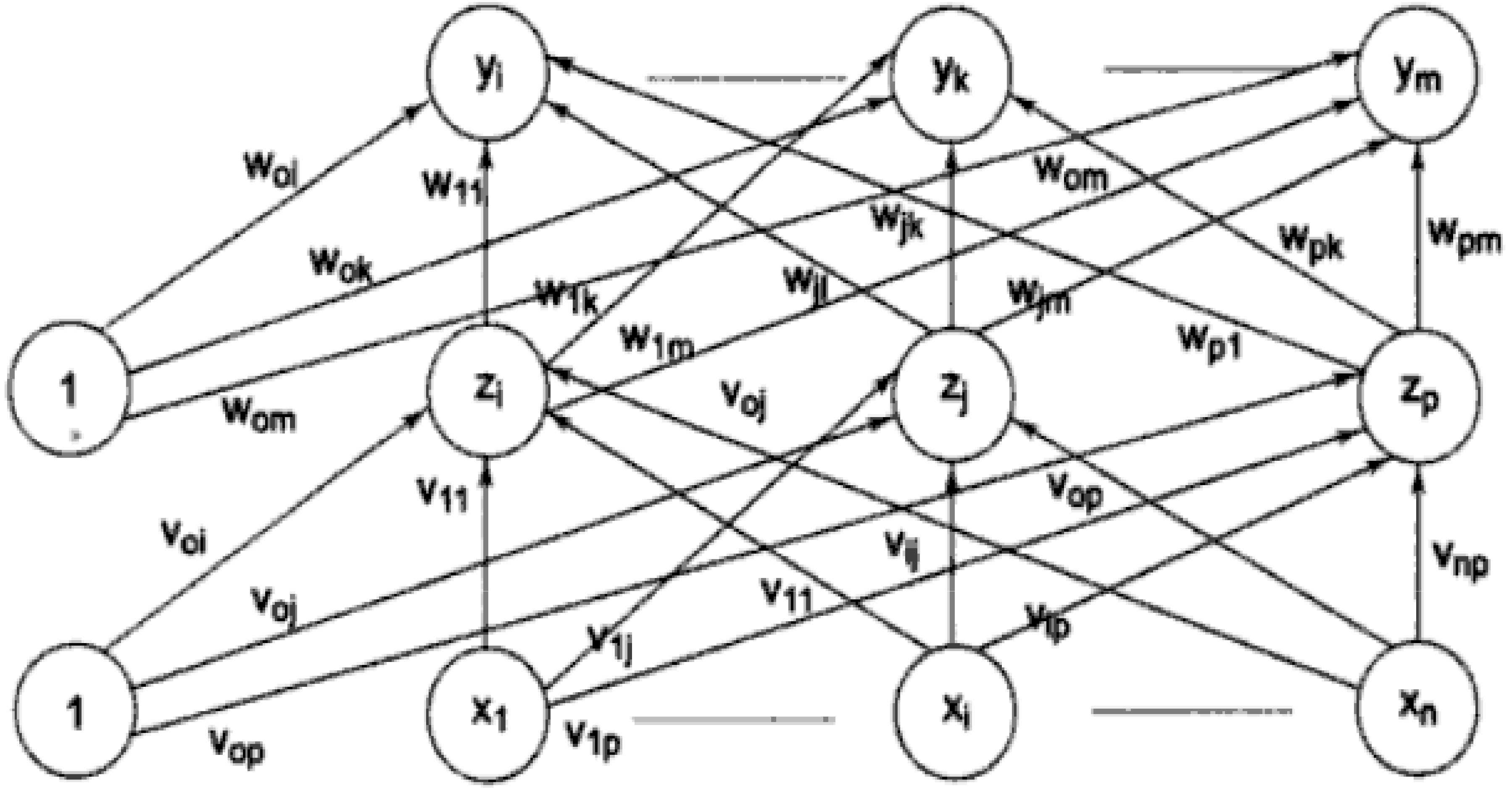
Preliminaries

- Function signal
  - input signals comes in at the input end of the network
  - propagates forward to output nodes
- Error signal
  - originates from output neuron
  - propagates backward to input nodes
- Two computations in Training
  - computation of function signal
  - computation of an estimate of gradient vector
    - o gradient of error surface with respect to the weights

Back Propagation Network (BPN)

- Back propagation is a systematic method for training multi-layer artificial neural networks.
- It has a mathematical foundation that is strong if not highly practical.
- It is multi-layer forward network using extend gradient descent based delta learning rule.
- Back propagation provides a computationally efficient method for changing the weights in feed forward network with differentiable activation function units, to learn a training set of input-output.

Architecture



Training Algorithm

- The training algorithm of back propagation involves four stages:

1. Initialization of weights
2. Feed forward
3. Back Propagation of errors
4. Update of the weights and biases

Parameters

- x: input training vector
- x = (x1, x2, …, xn)
- t: Output target vector
- t = (t1, t2, …, tm)
- α : learning rate
- δ_k : error at output unit y_k
- δ_j : error at hidden unit z_j
- y_k = out put unit k.

Algorithm

- Step 1: Initialize weight to small random values
- Step 2: While stopping condition is false, do steps 3-10.
- Step 3: for each training pair do step 4-9
- **Feed Forward**
- Step 4: Each input unit receive the input signal xi and transmit this signals to all unit in the layer above i.e. hidden units.
- Step 5: Each hidden unit (z_j : j = 1,..,p) sums it weighted input signals.

$$z_{-inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

applying activation function

$$Z_j = f(z_{inj})$$

and sends this signal to all units in the layer above i.e output units.

Algorithm..

- Step 6: Each output unit (y_k, k = 1,.., m) sum its weighted input signals.

$$y_{-ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

and applies its activation function to calculate the output signals.

$$Y_k = f(y_{-ink})$$

Algorithm…

- **Back Propagation Error**
- Step 7: Each output unit (yk, k=1, …, m) receives a target pattern corresponding to an input pattern error information term is calculated as:
  - $\delta_k = (t_k - y_k)\ f(y_{-in}k)$
- Step 8: Each hidden unit (zj, j= 1,…, n) sum its delta inputs from units in the layer above
  - $\delta_{-inj} = \sum \delta_j w_{jk}$
  - The error information term is calculated as
  - $\delta_j = \delta_{-inj}\ f(z_{-inj})$

Algorithm…

- Updating of weight and biases
- Step 9: Each output unit (yk, k = 1, …, m) updates it bias and weights (j = 0, …, p)
- The weight correction term is given by
  - $\Delta w_{jk} = \alpha \delta_k z_j$
- And the bias correction term is given by
  - $\Delta w_{0k} = \alpha \delta_k$
- Therefore
  - $w_{jk(new)} = w_{jk(old)} + \Delta w_{jk}$
  - $w_{0k(new)} = w_{0k(old)} + \Delta w_{0k}$

Algorithm…

- Each hidden unit (zj, j = 1,…, p) updates it bias and weights (I = 0,…,n)
- The weight correction term
  - $\Delta v_{ij} = \alpha \delta_j x_i$
- The bias correction term
  - $\Delta v_{0j} = \alpha \delta_j$
- Therefore
  - $v_{ij(new)} = v_{ijold)} + \Delta v_{ij}$
  - $v_{0j(new)} = v_{0j(old)} + \Delta v_{0j}$

Application Algorithm

- Step 1: Initialize weights from training program
- Step 2: For each input vector do steps 3-5.
- Step 3: For i = 1, …, n; set activation of input unit x_i:
- Step 4: For j = 1,…, p

$$Z_{-inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

- Step 5: For k = 1, …, m

$$y_{-ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

$$Y_k = f(y_{-ink})$$

Merit of Back Propagation

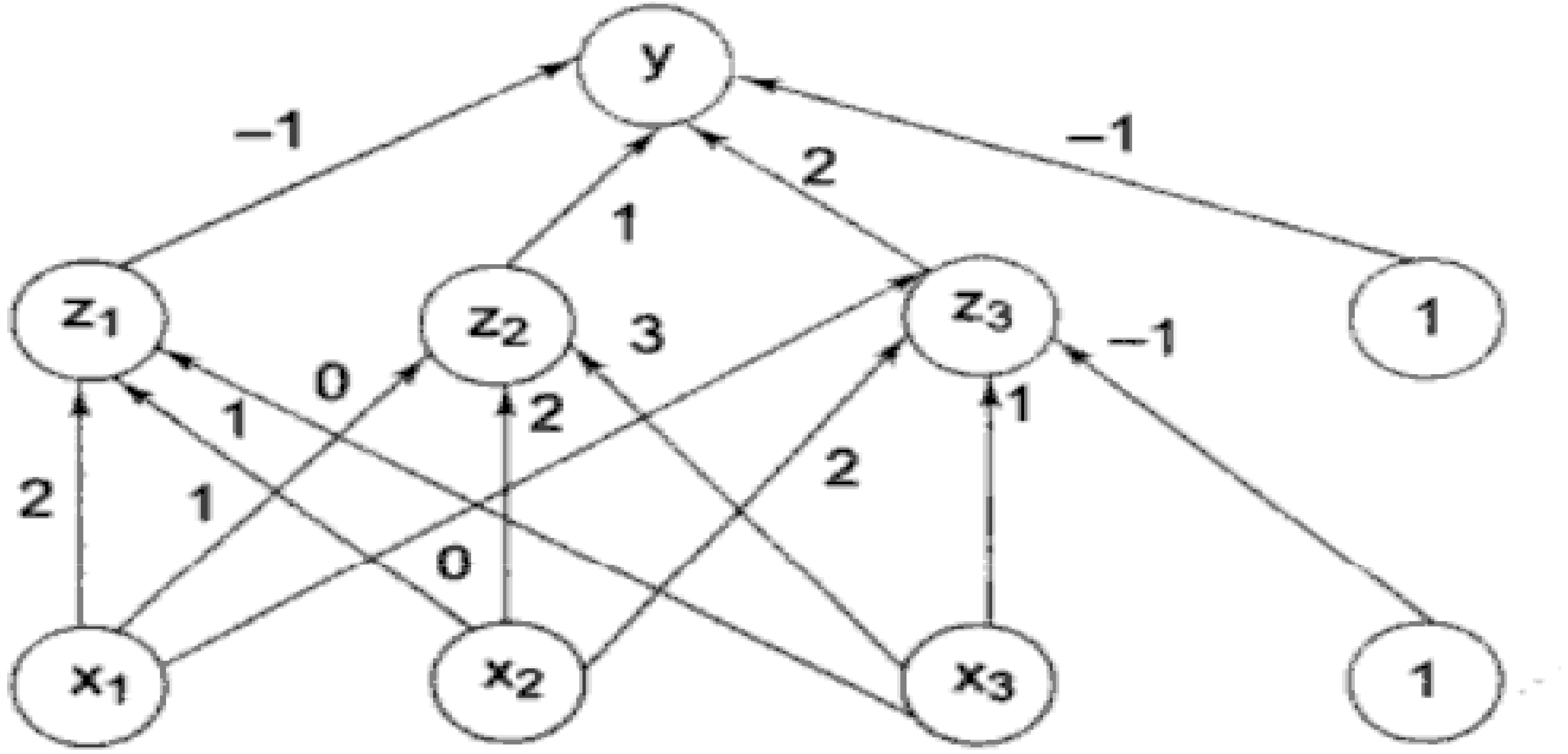
1. The mathematical formula present here, can be applied to any network and does not require any special mention of the features of function to be learnt.
2. The computing time is reduced if the weights chosen are small at the beginning.
3. The batch update of weights exist, which provides a smoothing effect on the weight correction terms.

Demerit of Back Propagation

1. The number of learning steps may be high and also the learning phase has intensive calculations.
2. The selection of the number of hidden nodes in the network is a problem. If number of hidden network is small, then the function to be learnt may not be possibly represented, as capacity of network is small. If the number of hidden neurons is increased, the number of independent variable of the error function also increases and the computing time also increases rapidly.
3. For complex problem it may require days or weeks to train the network or it may not train at all. Long training time results in non-optimum step size.
4. The network may get trapped in a local minima even though there is much deeper minimum nearby.
5. The training may sometimes cause temporal instability to the system

Problem

- Find the new weights when the network illustrated in following figure. The input pattern is [0.6 0.8 0] and target output is 0.9. Use learning rate α = 0.3 and use binary sigmoid activation function.



Solution

- Step 1: Initialize the weight and bias
  - $w = [-1\ 1\ 2]$, $w_0 = [-1]$
- $$v = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 3 & 1 \end{bmatrix} v_0 = [0 \quad 0 \quad -1]$$
- Step 3: For each training pair
  - $x = [0.6\ 0.8\ 0]$
  - $t = [0.9]$

Rest part of solution see on board

Problem

- Apply the Back Propagation Learning to the training patterns that define XOR function input and target.

Important point for the selection of parameters

- **Initial Weights**
  - It will influence whether the net reaches a global (or only a local) minima of the error and if so how rapidly it converges.
  - If the initial weight is too large the initial input signals to each hidden or output unit will fall in the saturation region where derivative of sigmoid has a very small value.
  - If initial weights are too small, the net input of a hidden or output unit will approach zero, which then causes extremely slow learning.
  - To get better result the weight (and bias) are set to random numbers between -0.5 and 0.5 or between -1 and 1.

Important point for the selection of parameters

- **Number of hidden units**
- If the activation function can vary with the function, then it can be seen that a n-input, m-output function requires at most 2n+1 hidden units.
- If more number of hidden layers are present, then the calculation for the δ are repeated.

Important point for the selection of parameters…

- **Selection of learning rate**
- A high learning rate leads to rapid learning but the weights may oscillate, while a lower learning rate leads to slower learning. Method suggested for adopting learning rate are:
- Start with a high learning rate and steadily decrease it. Changes in the weight vectors must be small in order to reduce oscillations or an divergence.
- A simple method is to increase the learning rate in order to improve performance and to decrease the learning rate in order to worsen performance.
- Another method is to double the learning rate until the error value worsen.

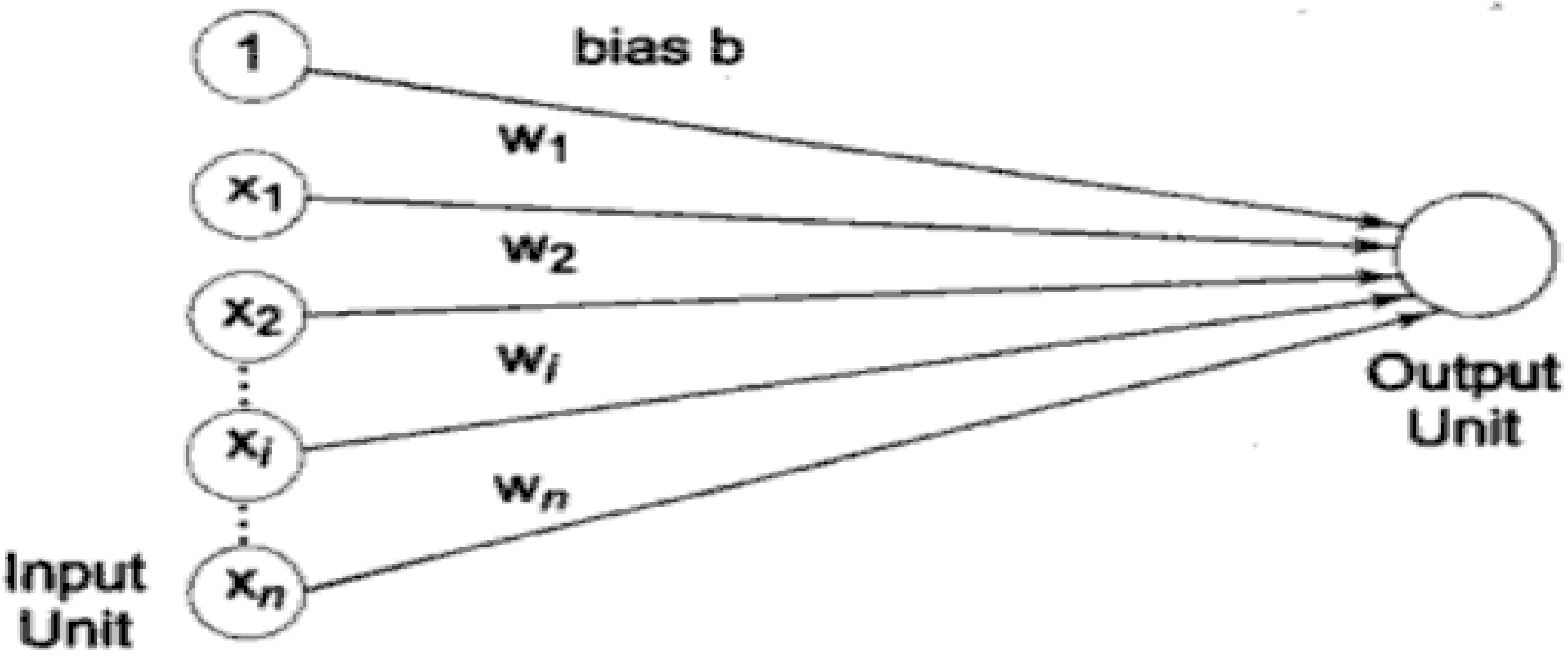
Application of Back Propagation

- Optical Character recognition
- Image Compression
- Data Compression
- Load forecasting problem in power system area
- Control problem
- Non linear simulation
- Fault detection problem etc.

Adaline

- It is developed Widrow and Hoff.
- It is used for bipolar activations for its input signals and target output.
- The weights and the bias of the Adaline are adujustable.
- The learning rule used can be called as Delta rule, Least Mean Square rule or Widrow Hoff Rule.
- The activation of this rule with single output unit, several output units.

Architecture



Training Algorithm

- Step 1: Initialize weights (not zero but small random values are used). Set learning rate α.
- Step 2: While stopping condition is false, do step 3-7.
- Step 3: For each bipolar training pair s:t perform Steps 4-6.
- Step 4: Set activations of input units $x_i = s_i$ for i = 1 to n.
- Step 5: Compute net input to output unit $y_{-in} = b + \sum x_i w_i$
- Step 6: update bias and weights, i = 1 to n.
  - $w_{i(new)} = w_{i(old)} + \alpha (t - y_{-in})x_i$
  - $b_{(new)} = b_{(old)} + \alpha (t - y_{-in})$
- Step 7: Test for stopping condition

Application Algorithm

- Step 1: Initialize weights obtained from the training algorithm.
- Step 2: For each bipolar input vector x perform Steps 3-5.
- Step 3: Set activation of input unit.
- Step 4: Calculate the net input to output unit $y_{-in} = b + \sum x_i w_i$
- Step 5: Finally apply the activation to obtain the output y.

$$y = f(y_{-in}) = \begin{cases} 1, & y_{-in} \geq 0 \\ -1 & y_{-in} < 0 \end{cases}$$