PERCEPTRON ALGORITHM:

1. For each node in the output layer:
   - → Calculate the error which can only takes the values 1 and -1
   - → If the error is 0, the goal has been achcieved. Otherwise, we adjust the weights.
   - → Donot alter weights from inactivated input nodes.
   - → Decrease the weight if the error was 1, increase ~~it~~ the weight if the error was -1.

⇒ RULES :-

1. Weight change = some small constant * (target activation − spontaneous output Activation) * input activation.

- If we speak of error instead of the "Target activation of minus the spontaneous output activation", then.

Weight change = Some small constant * error * input activation.

PERCEPTRON ALGORITHM in pseudo-code

Start with random initial weights
[e.g: uniform random in [-0.3, 03])

```
Do
{
  For all patterns p
  {
    For all output Nodes j
    {
      CalculateActivation(j)

      Error_j = TargetValue_j_for_Pattern_p -
                Activation_j
      For all input Nodes i To output Node j
      {
        DeltaWeight = LearningConstant * Error_j *
                      Activation_i

        Weight = Weight + DeltaWeight.
      }
    }
  }
} Until "Error is sufficiently small" or "Time-out"
```

PERCEPTRON LEARNING RULE

- How do we find the weights using a learning procedure?

1. Choose initial weights randomly.
2. Present a randomly choosen pattern x.
3. Update weights using Delta rule:

$$w_{ij}(t+1) = w_{ij}(t) + err_i * x_j$$

where $err_i = (target_i - output_i)$.

4. Repeat step 2 and step 3 until the stopping criteria (convergence, maximum no. of iterations) is reached.

Perceptron Convergence Theorem :-

→ If a pattern set can be expanded by a two layer perceptron, then the perceptron learning rule will always be able to find some correct weights.

PERCEPTRON LIMITATION:

A single layer perceptron can only learn LINEALY SEPERABLE problems.

Boolean AND function is linearly seperable, whereas BOOLEAN XOR function (and the parity problem in general) is not.

TRAINING SET:

$$\left\{ p_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = [1] \right\} \left\{ p_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = [0] \right\}$$

Random Initial weights:

$$W = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \qquad b = 0.5$$

First Iteration:

$$a = hardlim(Wp_1 + b) = \cancel{hardlim([0.5 \ -1, -} \; hardlim\left(\begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5 \right)$$

$$W^{new} = W^{old} + ep^T = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} + (1) \begin{bmatrix} -1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (1) = 1.5$$