Perceptron

- This simple model calculates the weighted sum of the input feature vector and passes the weighted sum through a hard thresholding function, outputs either a +1 or a -1
- This model can solve **linearly separable** problems.
- When a problem is **linearly non-separable**, the Perceptron algorithm will not converge.

Perceptron (Training) Algorithm

Let {X(k), d(k)}, k =1, 2, …K, are the K training samples, where X(k) = (x_1(k), x_2(k), …, x_N(k)) is kth N-dimensional feature vector, d(k) = +1 or d(k) = -1 is the desired output of X(k), then Perceptron training algorithm can be described in the following pseudo code

```
Initialization
Define w_i, i = 0, 1, 2, … N,  and set w_i to small random values, e.g., in the range [-1, 1]
Set x_0(k) = 1, for all k =1, 2, … K
Set training rate t_r to a value in [0, 1]
Set STOP_EPOCH = 100 // Training stops after STOP_EPOCH epochs (this value is set empirically)
Define CORRECT // This records the number of training samples correctly trained
Set epoch = 0

do
{
    epoch++
    CORRECT = 0
    for k = 1 to k = K
    {
    R(k)=0
        for i = 0 to i = N
        {
            R(k)+= w_i*x_i(k)
        }

        if (R (k) > 0) o(k) = 1 else o(k) = -1

        if (o(k) = = d(k)) CORECT++
        else
        {
            for i = 0 to N
            {
                w_i = w_i + tr*(d(k) – o(k))*x_i(k)
            }
        }
    }
}
while (CORRECT < K) //training stops when all training samples are correctly learned
or
while (epoch < STOP_EPOCH) //training stops after a pre-set number of iterations
```

ADLINE and Gradient Descent Learning (Delta Rule)

- This model is similar to Perceptron, except that it directly outputs the weighted sum of the inputs.
- There are several key concepts
  - Error function or cost function – this is defined as the squared difference between the actual output and the desired output summed over all training samples
  - Gradient descent training – training flows gradient descent or steepest descent rule where we first calculate the gradient of the error function and then move the weights along the opposition direction of the gradient.

Training ADLINE with Gradient Descent Rule

Let {X(k), d(k)}, k =1, 2, …K, are the K training samples, where X(k) = (x_1(k), x_2(k), …, x_N(k)) is kth N-dimensional feature vector, d(k) is the desired output of X(k), then ADLINE training with gradient descent rule can be described in the following pseudo code

Initialization

Define w_i, i = 0, 1, 2, … N, and set w_i to small random values, e.g., in the range [-1, 1]
Set x_0(k) = 1, for all k =1, 2, … K
Set training rate t_r to a value in [0, 1]
Set STOP_EPOCH = 100 // Training stops after STOP_EPOCH epochs (this value is set empirically)
Define ERROR = STOP_ERROR // This defines the value of the error function, when it is below a pre-defined value STOP_ERROR, training stops
Set epoch = 0

```
do
{
        epoch++
        ERROR = 0
                for i = 0 to N
                        {
                        Delta[i] = 0 //This Delta will be used in Batch Mode Learning
                        }

        for k = 1 to k = K
        {
        o(k) = 0
                for i = 0 to i = N
                {
                        o(k)+= w_i*x_i(k)
                }
                ERROR+= (o(k)-d(k))^2
        //If used online learning, then update the weights using the following for-loop
                for i = 0 to N
                {
                        w_i = w_i + tr*(d(k) – o(k))*x_i(k)
                }
        //If used batch mode learning, then cumulates the error signals using the following for-loop
                for i = 0 to N
                {
                        Delta[i]+= (d(k) – o(k))*x_i(k)
                }

        } //end of k for-loop

        //Update the weights in batch mode
        for i = 0 to N
                {
                        w_i = w_i + tr*Delta[i]
                }
}//end of do loop
while (ERROR < STOP_ERROR) //training stops when overall error is smaller than a preset value
or
while (epoch < STOP_EPOCH) //training stops after a pre-set number of iterations
```