**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**BITS C464 – MACHINE LEARNING**

**I Semester 2014-2015**

**WORKSHEET #4**

**Classification using Perceptron Model**

**OBJECTIVE:-**

* **Single Layer Perceptron model**
* **Learning using Gradient Descent**
* **Parameter estimation using Delta Rule**
* **Non Linear Separable Problems**

In machine learning, the **perceptron** is an algorithm for supervised classification of an input into one of several possible non-binary outputs. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

The perceptron is an algorithm for learning a binary classifier: a function that maps its input x (a real-valued vector) to an output value f(x) (a single binary value):


f(x) = \begin{cases}1 & \text{if }w \cdot x + b > 0\\0 & \text{otherwise}\end{cases}


where w is a vector of real-valued weights, w \cdot x is the dot product (which here computes a weighted sum), and b is the 'bias', a constant term that does not depend on any input value.



Geometric interpretation of a perceptron:

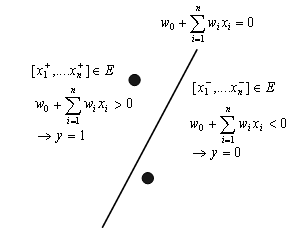
• input patterns (x1,... ,xn) are points in n-dimensional space

• points with w0 + <w,xi> = 0 are on a hyperplane defined by w0 and w

• points with w0+ <w,xi> >0 are above the hyperplane

• points with w0+ <w,xi> <0 are below the hyperplane

• perceptrons partition the input space into two halfspaces along a hyperplane



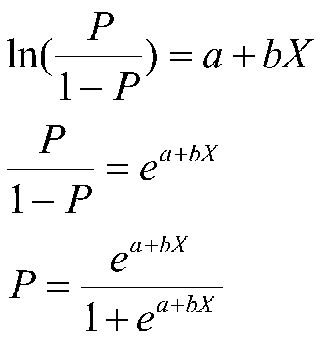
Classification is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. Binary classification problem in which y can take on only two values, 0 and 1. For instance, if we are trying to build a spam classifier for email, then x(i) may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given x(i), the corresponding y(i) is also called the label for the training example.

A single perceptron can learn only examples that are called “linearly separable”. These are

examples that can be perfectly separated by a hyperplane.

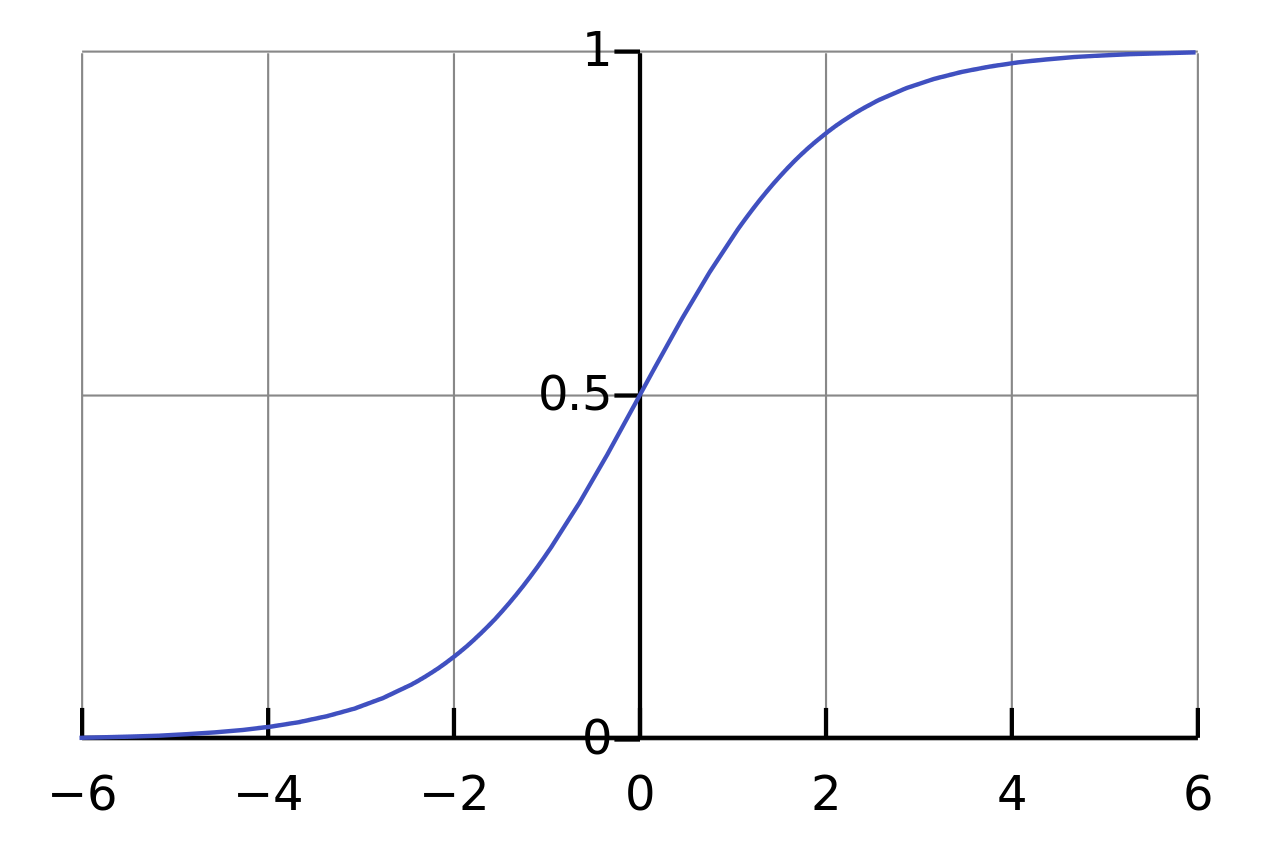
Single Perceptrons can learn many boolean functions: AND, OR, NAND, NOR, but not XOR

**Logistic regression:**



Here P is the probabilty distribution function which we are trying to learn. P

is called the logistic function or the sigmoid function. Here is a plot showing P:



Notice that p(x) tends towards 1 as x → ∞, and p(x) tends towards 0 as x → −∞. Moreover, p(x), and hence also h(x), is always bounded between 0 and 1.

the definition of g to be the threshold function:

g(z) = 1 if z ≥ 0

0 if z < 0

If we then let hθ(x) = g(θ’ x)

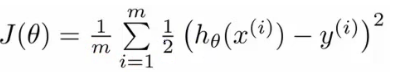
**Learning using Gradient Descent:**

Gradient descent is a simple, principled optimization algorithm used to choose parameter values in a variety of discriminative and generative predictive models. The algorithm works by efficiently searching the parameter space according to the following rule:

 \mathbf{b} = \mathbf{a}-\gamma\nabla F(\mathbf{a})

Here b is the parameter we are trying to optimise.

The cost function we are trying to optimise is:

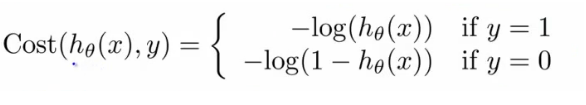


In general terms we can write it as:

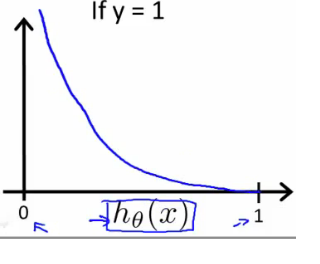


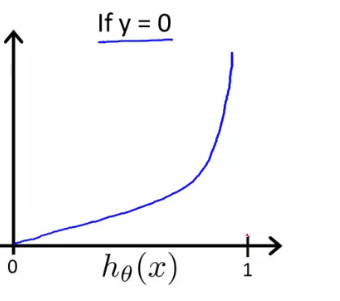
where m is the total number of training examples, hθ(xi) is the hypothesis function, where cost is penalty for wrong classification.

But it's gradient can be non-convex so we can optimise some other equivalent function

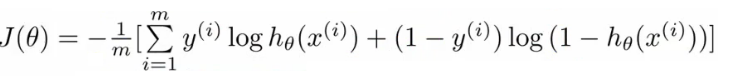


Graphically we can viewed as:



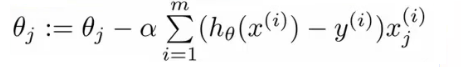


It shows convex behaviour, so can be optimised instead.



Differentiating it and substituting its gradient in the gradient descent rule we get:

The gradient descent algorithm can now be expressed as

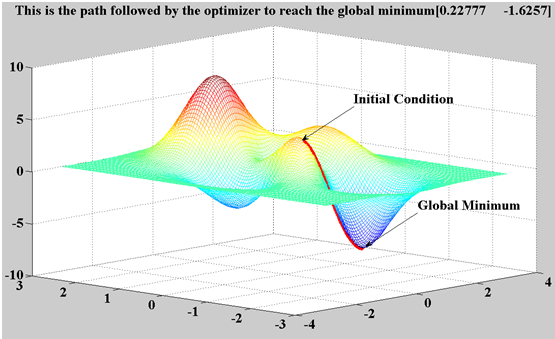


After each iteration the algorithm updates the parameters θ by iterating over all m examples.

Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point.

**NOTE:**

Gradient descent converges to global minima in case of convex functions only. In case of non-convex function it may or may not converge to global minimum. It can be trapped to local minima also(Depending on starting position)



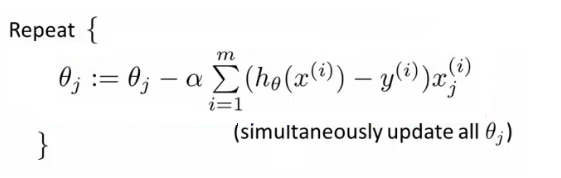
**Delta Rule:**

In machine learning, the **delta rule** is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in single-layer neural network. It is a special case of the more general backpropagation algorithm. For a neuron j \,with activation function g(x) \,, the delta rule for j \,'s i \,th weight w_{ji} \,is given by

\Delta w_{ji}=\alpha(t_j-y_j) g'(h_j) x_i  \,,

where

|  |  |
| --- | --- |
|  | \alpha \,is a small constant called *learning rate* |
|  | g(x) \,is the neuron's activation function |
|  | t_j \,is the target output |
|  | h_j \,is the weighted sum of the neuron's inputs |
|  | y_j \,is the actual output |
|  | x_i \,is the i \,th input.  The perceptron learning rule can be summarised as following by considering the delta rule for weight updation. |

****

Here alpha is the learning rate. And convergence condition is that when delta-weight is less than some epsilon.

**NOTE:**

Large value of alpha means that it will take large steps in each iteration and there is a chance it overshoots the optimum point.

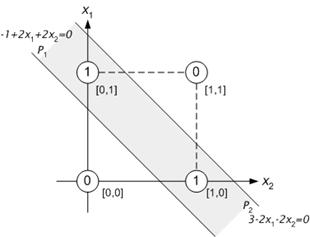
Small alpha means it will take many iterations to converge.

So a trick to take alpha is check it against the error function. If the error decreases after one iteration then increase alpha by 20% and if error increases after some iteration then decrease alpha byt 50%

**Non Linear separable Data:**

Remember that it is not possible to find weights which enable Single Layer Perceptrons

to deal with non-linearly separable problems like XOR:



OR

AND

XOR

However, Multi-Layer Perceptrons (MLPs) are able to cope with non-linearly separable

problems.

Try to stack two layers of perceptron and then try to analyse how they work and then look for algorithms for multilayer perceptron.

**Exercise:**

1. Learn various Boolean functions (AND, OR, NAND, NOR) using single layer perceptron model.
2. Try non-linearly seperable functions (XOR) using multi layer perceptron model.