MACHINE LEARNING CS60050

Instructor: Dr. Aritra Hazra
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur
Scribed by: Deepak Mewada ; 20CS91P02

Lecture Scribe for Class on 5th February 2021

1 Previously

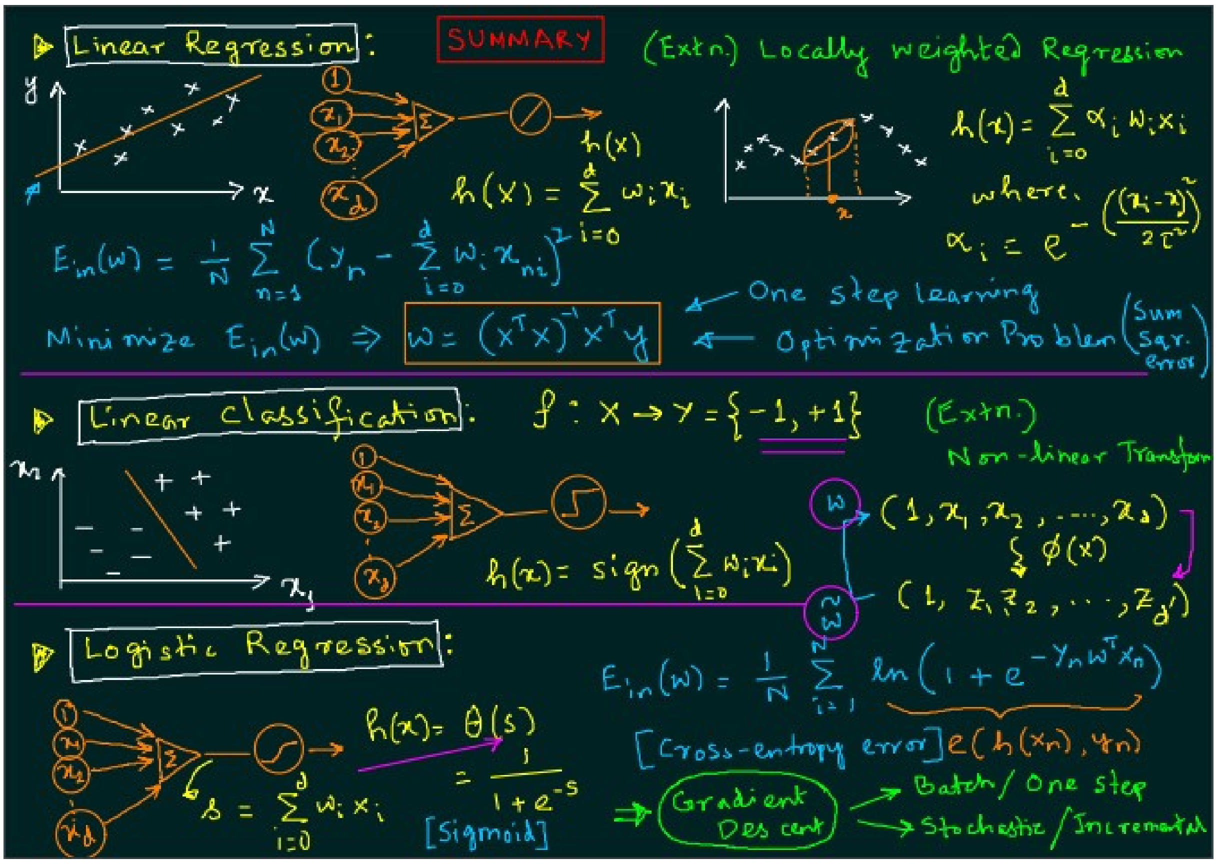

Linear Regression:
SUMMARY
(Extn.) Locally weighted Regression
$h(x) = \sum_{i=0}^{d} w_i x_i$
$h(x) = \sum_{i=0}^{d} \alpha_i w_i x_i$
where, $\alpha_i = e^{-\left(\frac{(x_i - x)^2}{2\tau^2}\right)}$
$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \sum_{i=0}^{d} w_i x_{ni}\right)^2$
Minimize $E_{in}(w) \Rightarrow \boxed{w = (X^T X)^{-1} X^T y}$
One step learning
Optimization Problem (Sum Sqr. error)
Linear classification:
$f : X \rightarrow Y = \{-1, +1\}$
(Extn.) Non-linear Transform
$h(x) = \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right)$
w $(1, x_1, x_2, \ldots, x_d)$
$\phi(x)$
$\tilde{w}$ $(1, z_1, z_2, \ldots, z_{d'})$
Logistic Regression:
$E_{in}(w) = \frac{1}{N} \sum_{i=1}^{N} \ln\left(1 + e^{-y_n w^T x_n}\right)$
[Cross-entropy error] $e(h(x_n), y_n)$
$h(x) = \theta(s) = \frac{1}{1+e^{-s}}$
[Sigmoid]
$s = \sum_{i=0}^{d} w_i x_i$
Gradient Descent
Batch / One step
Stochastic / Incremental

Figure 1: Summary? of Previous Class

We have covered certain weapons of machine learning in our previous classes. Which are as follows:

1.1 Linear Regression

Linear Regression fits a linear model with coefficients $w = (w_1, \dots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

$$\text{h(x)} = \sum_{i=0}^{d}(w_i x_i)$$

The in-sample error, or the error while training is calculated as

$$E_{in}(w) = \frac{1}{N}(y_n - \sum_{i=0}^{d}(w_i x_n i))^2$$

On minimizing E_{in} to find the value of W. We find $\boxed{\text{W} = [(\text{X}^T X)^{-1} X^T]Y}$
So It's just a One-step learning.

1.2 Linear Classification

The prime objective of Linear Classification problems is to create a hyper-plane differentiating the two classes.The sum $W^T X$ is passed by a hard threshold function, And the result thus obtained classify our data.

1.3 Logistic regression

In Logistic regression instead of the threshold being too hard we introduced a soft threshold function i.e Sigmoid/Tanh. We can find the best hypothesis by converting this problem into an optimization problem w.r.t sample set. Now the error will be Cross-Entropy error in ML terms. Which is not One step learn-able. We can minimize the error by Gradient Descent(Batch, Stochastic).

So all these concepts discussed are very much of our ingredients towards moving into Artificial Neural Network formulation.

2 Biological Neuronal Network- Inspiration for ANN

When we talk about Artificial Neural Network, it draws the analogy from biology. Neurobiology says that we have dendrites that usually sense some information, processes through its nucleus and sends back to its axon terminals, and provides it back.

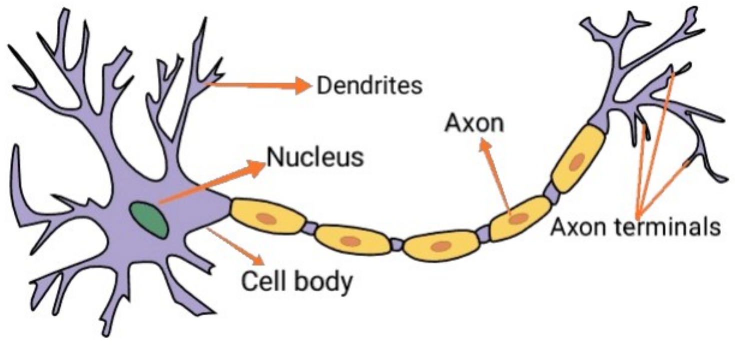
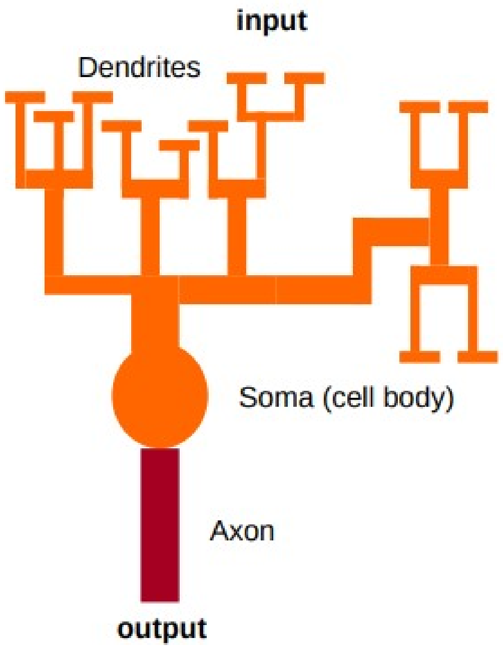


Figure 2: Biological Neuron

Essentially almost 10^{11} number of neurons are present in the human body. Each neuron is connected to 10^4 other neurons on average. People have experimented that to recognie someone it usually takes 10^{-1} sec. It has also been experimented in biology that 10^{-3} is the switching time of the neurons. So more than 100 neurons are fired/used for recognition.

Inside the human body biological neural network looks like as given in figure 3.b.



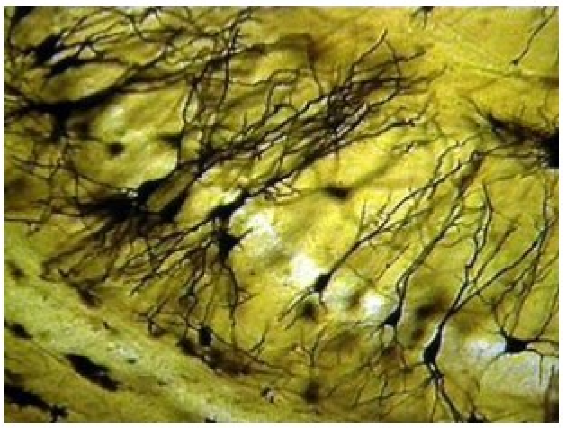


Figure 3: Biological Inspiration

Based on these biological concepts and the way input and output are processed, people thought about that why not we resemble such kind of concept in terms of mathematical nodes which will again be connected to some other just like a biological structure and process something. It will have some inputs which are values of attributes and have some output. In mathematical jugglery, these nodes could be a sum followed by a threshold. So when we consider node after node essentially in the input space that is given, one neuron will provide us one separation going with that with the other set of input from other neurons will give us other separation. Another neuron will give us another separation. So we can extract a set of the line to separate regions.

3 Linear Classification: Perceptron(linear discriminator)

As we know in linear classification problem we try to find out unknown f from attribute set X which produces value Y such that $f : X -> Y$, Where Y could be +1 or -1(assume here 2 class classification). If it is very well separated by a single line therefore linear separator or Perceptron is the exact model that could be used. If it is not separable by linear line then either we will use some other ML algorithm or we may use this similar kind of linear classification multiple of them in the staggered fashion to form an Artificial Neural Network.

Doubt: Why only Perceptron is used as a basic model?
Perceptron is foundational model to separate two classes and using it multiple times we can just segregate any pores inside the data set.

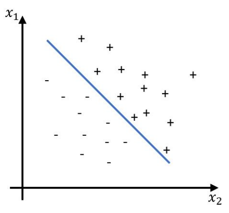


Figure 4: A linear classifier aims to find a single line that differentiates 2 classes.

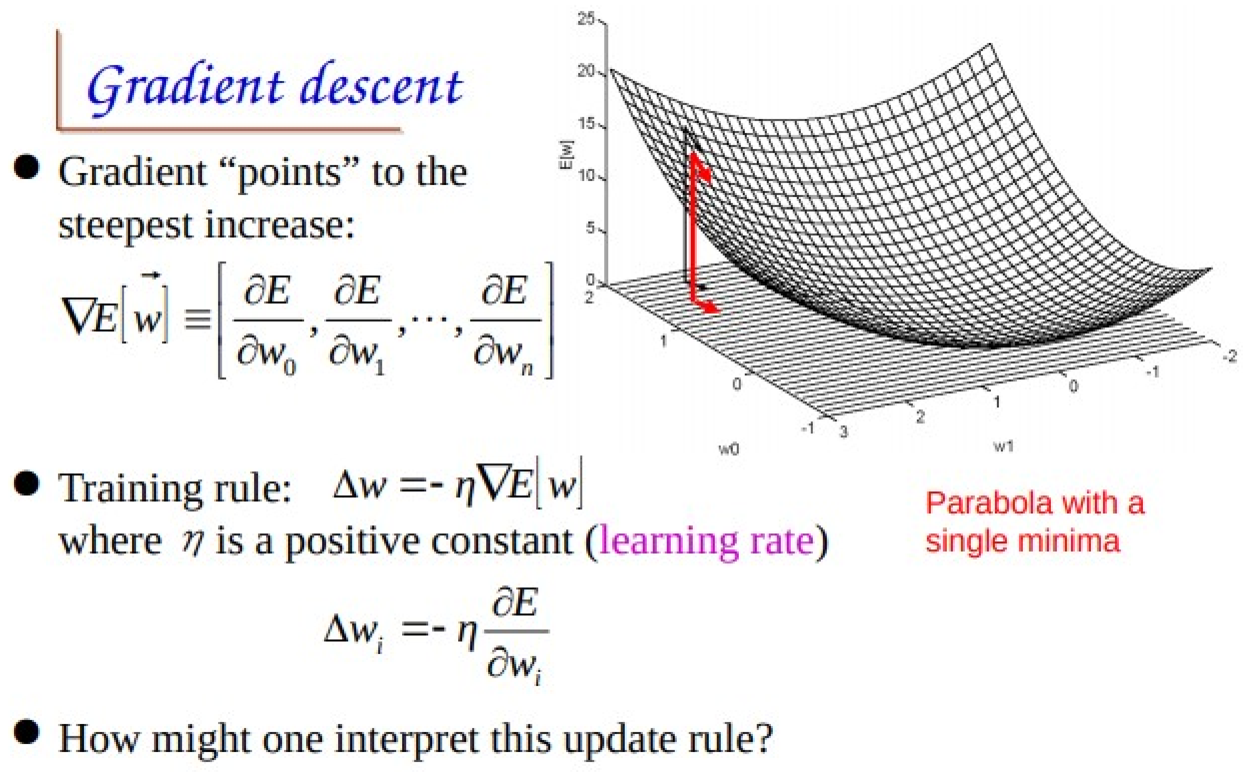
As we know that linear predictor is just the threshold of sum.

$$\text{h(x)} = sign(\sum_{i=1}^{d} w_i x_i)$$

Now we resort to our same optimization problem where we try to minimize our in-sample error. Here in-sample error will be

$$E_{in}(w) = \frac{1}{2}\sum_{n=1}^{N}[y_n - sign(\sum_{i=1}^{d} w_i x_{ni})]^2$$

So, now the problem boils down to reduce the above error in every iteration. If the objective function is smooth, we can gradually change the feature weights after every iteration based on our E_{in}.



Gradient descent

- Gradient "points" to the steepest increase:

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

- Training rule: $\Delta w = -\eta \nabla E[w]$
where η is a positive constant (learning rate)

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- How might one interpret this update rule?

Parabola with a single minima

E[w]
25
20
15
10
5
0
2
1
0
-1
3
2
1
0
-1
-2
w0
w1

Figure 5: Gradient Descent

So our weigh update rule can be written as

$$w_{t+1} = w_t + \Delta w_i$$

Where

$$\Delta w_i = -\eta \frac{\partial E_{in}}{\partial w_i}$$

Now to find Δw_i, On differentiating $E_{in}(w)$ w.r.t w_k we get

$$\frac{\partial E_{in}}{\partial w_k} = \sum_{n=1}^{N} (y_n - sign(\sum_{i=1}^{d} w_i x_{ni}))(-sign(x_{nk}))$$

Now if we say y_n is our target output 't' and $sign(\sum_{i=1}^{d} w_i x_{ni})$ is our observed output 'o' then above equation can be written as

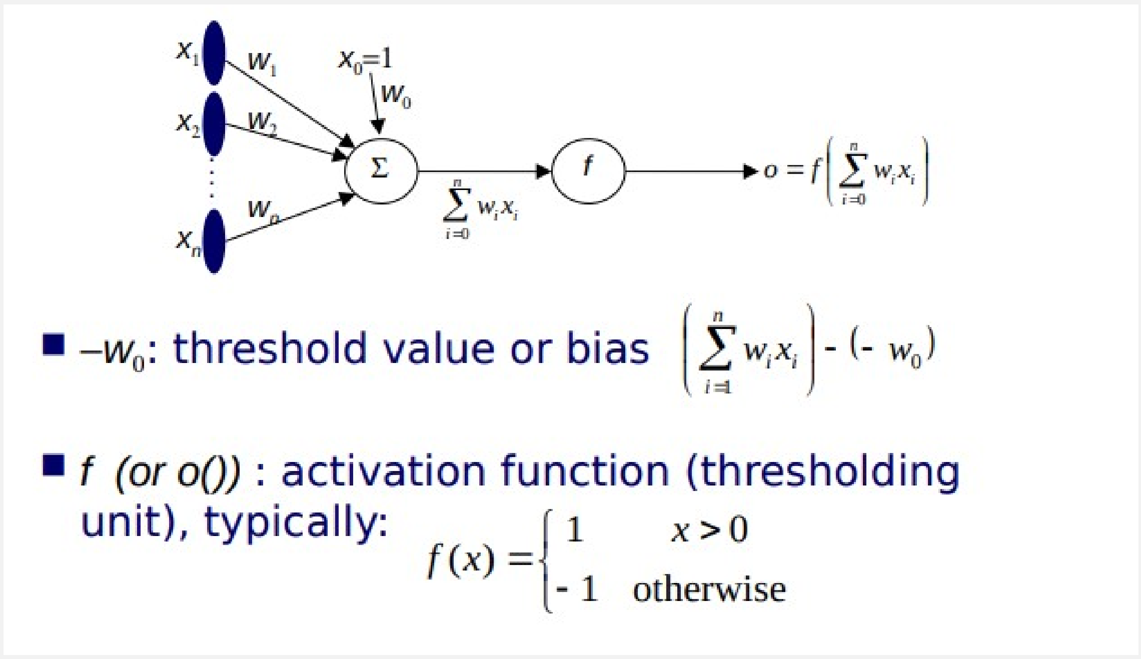
$$\frac{\partial E_{in}}{\partial w_k} = \sum_{n=1}^{N} (t - o)(-x_{nk})$$

So now our weight update rule equation will be

$$\boxed{w_{t+1} = w_t + \eta \sum_{n=1}^{N} (t - o)(x_{nk})}$$

This is the how we will update the weights so that we minimize the error.
This we are doing it together(summation) so it's a **Batch Gradient Descent**.

Note : Since here while updating weight we need to differentiate the threshold function so the threshold function should be something that is differentiable. But in the case of the linear classifier (discussed above), the threshold function (sign) is not differentiable. So now we will move to the concept of **Perceptron** where this threshold function is differentiable. Instead of the sign function, we will have a differentiable θ(or f) function in Perceptron.



x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
f
$o = f\left(\sum_{i=0}^{n} w_i x_i\right)$
■ $-w_0$: threshold value or bias $\left(\sum_{i=1}^{n} w_i x_i\right) - (-w_0)$
■ f (or o()) : activation function (thresholding unit), typically:
$f(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{otherwise} \end{cases}$

Figure 6: Perceptron

4 Artificial Neural Node

An Artificial Neural node have a summing unit followed by a soft threshold unit to get our hypothesis of x i.e $h(x)$. Which is nothing but $\theta(S)$. Here either think $\theta(S)$ as $\tanh(s)$ or $\frac{1}{1+\exp^{-s}}$ or any other soft threshold function.

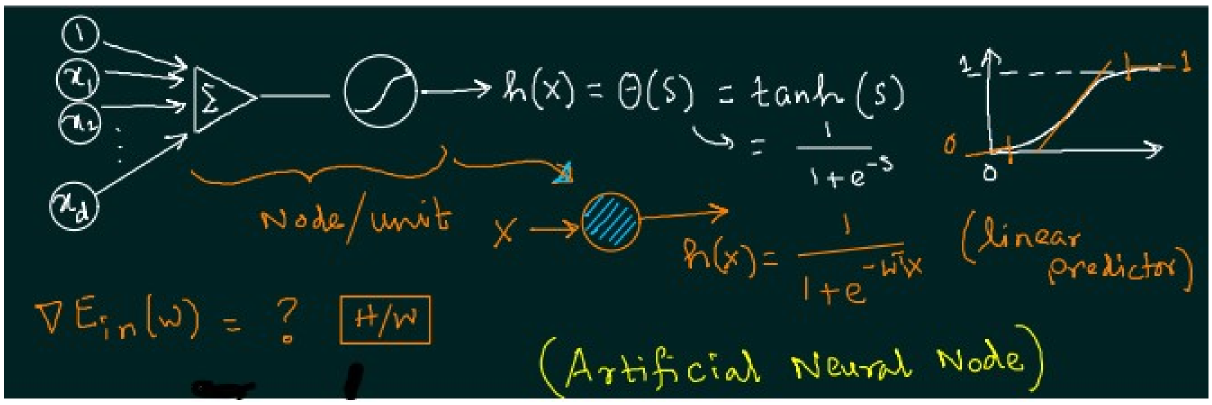


Figure 7: Artificial Neural Node

Exercise : Find $\Delta E_{in}(w)$ for the Neural Node given in figure 7?.

Let us see some examples of Artificial Neural Node. Here we will try to learn few Boolean Gates with the help of Perceptron.

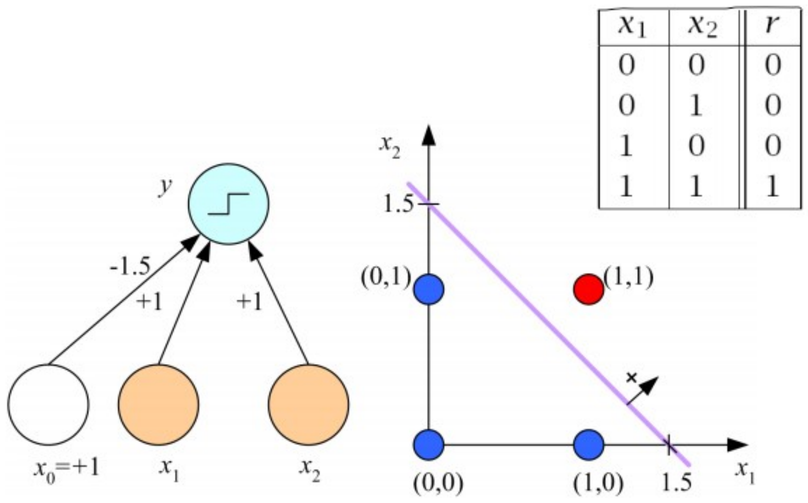
4.1 Learning Boolean AND

Boolean 'AND' can be realized by using a single-layer Perceptron.

For the input x_1 and x_2 and weights w_0 , w_1 and w_2 the hypothesis can be written as

$$h_1 = (w_0 + w_1.x_1 + w_2.x_2)$$

We can realize the Boolean 'AND' gate with the help of Perceptron just by learning appropriate weight parameters.



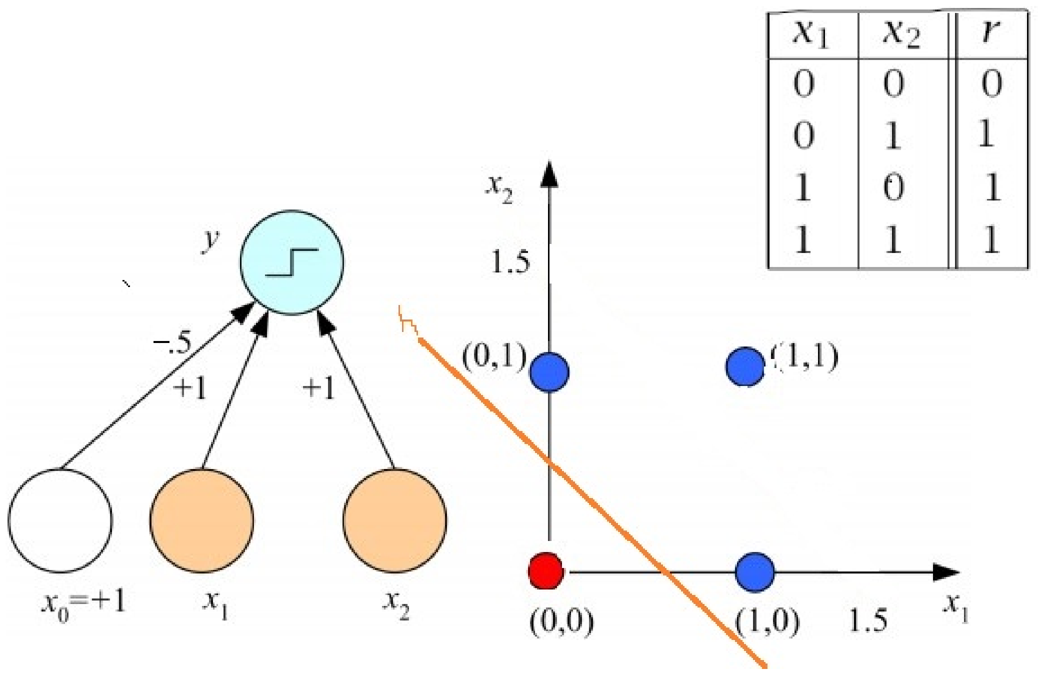
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

Figure 8: Realization of Boolean AND by single-layer Perceptron

Here to realize AND the possible values of weight parameters can be $w_1 = 1 = w_2$ and $w_0 = -1.5$

4.2 Learning Boolean OR

Just like AND Boolean 'OR' can also be realized by single-layer Perceptron. Unlike AND, if we set w_0 to -0.5 in the above Neural node then the same Neural Node will behave as an OR gate.



x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	1

Figure 9: Realization of Boolean OR by single-layer Perceptron

So we can see from the above figure that the same separator line by just dragging it toward origin our Perceptron will behave as 'OR' gate. We can drag or move the separator line by just changing w_0.

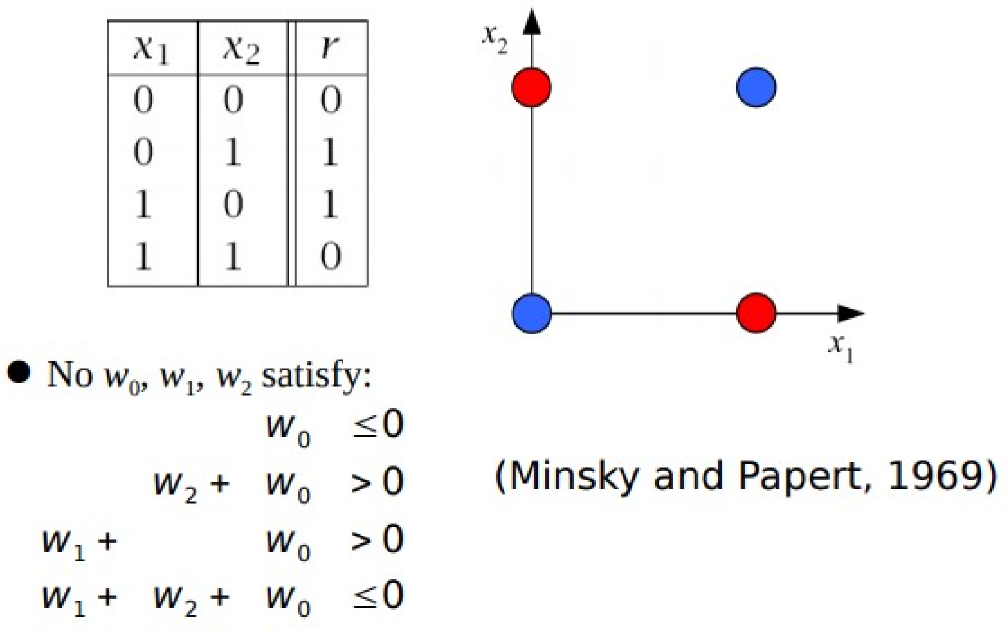
Note Here we put values of weight intuitively. But effectively we can learn these weights by Gradient Descent.

Limitation of linear predictor : If the data is not linearly separable then there is a chance of failure.

Now let us consider an example to handle the situation where linear predictor might fail.

4.3 Learning Boolean XOR

Let's consider this example of Boolean XOR.



x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

- No w_0, w_1, w_2 satisfy:

$$\begin{aligned} w_0 &\le 0 \\ w_2 + w_0 &> 0 \\ w_1 + w_0 &> 0 \\ w_1 + w_2 + w_0 &\le 0 \end{aligned}$$

Figure 10: Learning Boolean XOR

As evident from the above figure the red and blue are not linearly separable anymore. Because there is no single line that can separate both of them.

We try to combine one or more hypotheses(or try to combine one or more linear separators). We have h_1 and h_2.

So effectively the combined hypothesis can be written as

$$\boxed{h = h_1.h_2 + \bar{h_1}.\bar{h_2}}$$

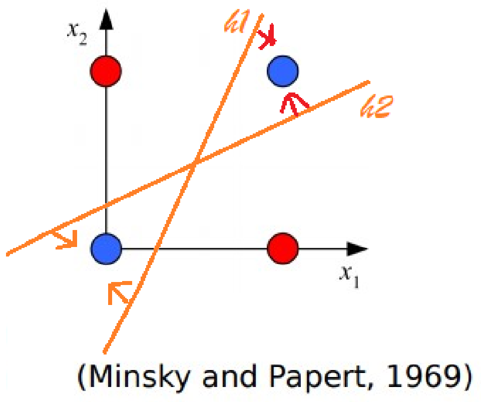


Figure 11: Multiple Linear Separator(h_1, h_2) to classify Red and Blue dots

We already have a single node Perceptron for each of h_1 and h_2. We know the Perceptron for 'AND' and we know the Perceptron of OR.
Let Us construct classifier h, which is not linear anymore but a combination of linear classifier one after another. So first merge $h_1.h_2$(AND) and $\bar{h_1}.\bar{h_2}$(AND) by an OR. And $\bar{h_1}$ can be obtained by multiplying -1 to weights of h_1.

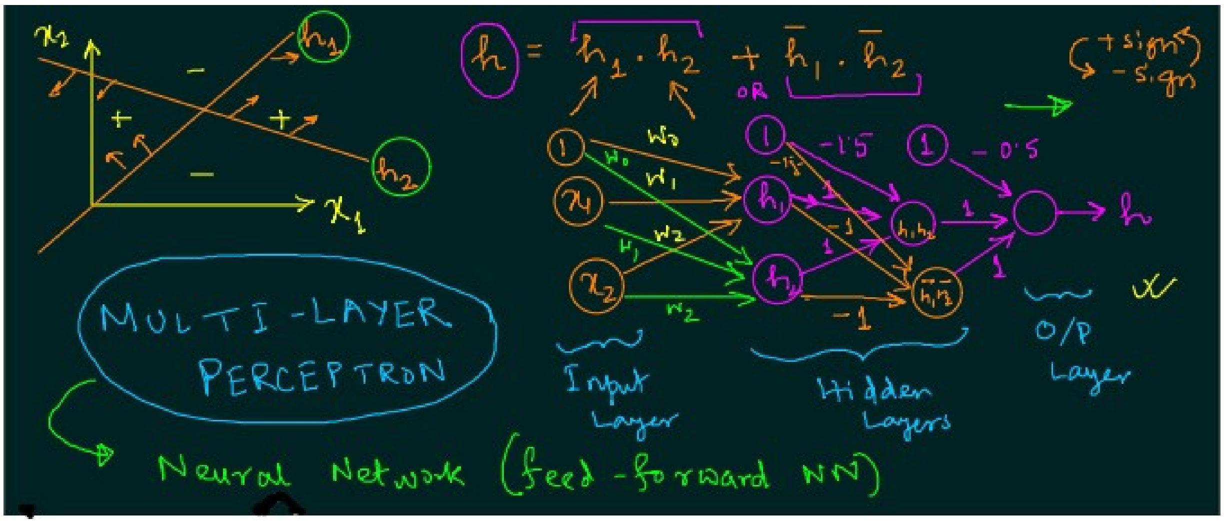


Figure 12: Realization of Boolean XOR by Multi-Layer Perceptron

As we can see from figure 12, the network has an Input layer, Hidden layers, and output layer. This is called Multi-layer Perceptron. But essentially we know it as a Neural Network and this variant is called Feed-forward Neural Network. Feed-forward because it is always feeding in the forward direction.

So this is all about Deep Learning Fundamental and all.
Now we have a problem to think.

Think!

Q : How to categorize +'s from -'s using Neural Network as given in figure 13?
Hint: Let's say it's the perfect circle as the target function(f) output . Which is not known to us. We need to approximate this 'f' (circular function) with our hypothesis 'h'.

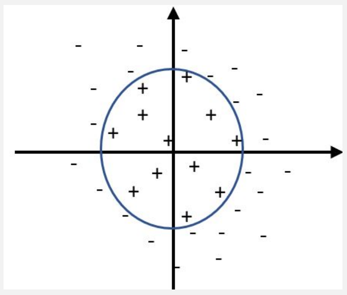


Figure 13: Non-linear distribution

Note 1. This scribe is based on lecture taught by Prof. Aritra Hazra on 05-feb-2021 in Machine Learning(CS60050) course.
2 All the figures in this document are taken from either handout-07a or Slide-07 uploaded on Machine Learning(CS60050) course website.