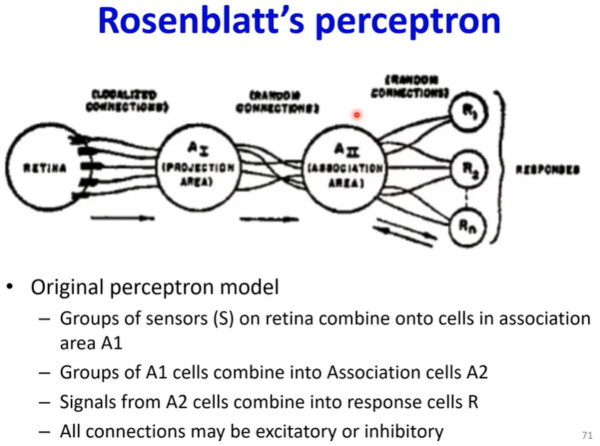
Lecture 1b - The Perceptron

Contents

2 The Perceptron

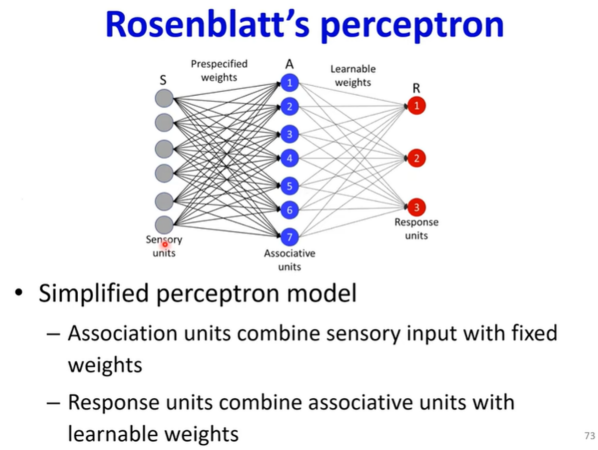
2.1 Introduction

The Perceptron was developed by Frank Rosenblatt in 1958. He was a psycologist at yale.

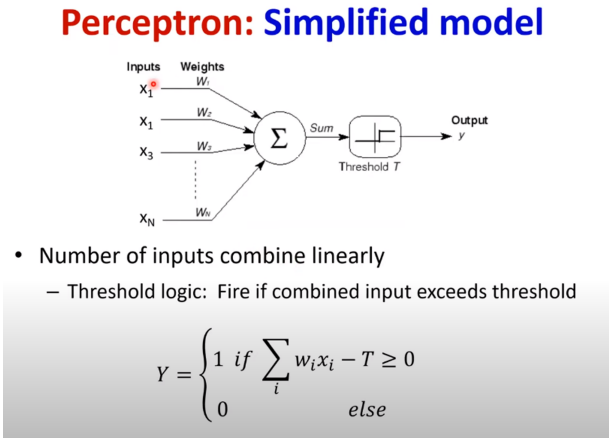


The perceptron is a type of artificial neuron, a fundamental unit in the field of neural networks. It was meant for explaining visual perception. The simplified version is shown in the 2nd image. **The term "Perceptron" is often used to refer to the individual computational unit rather than the entire model rather than the entire model.**

Sensory units: Cells in the retina that receive input from the environment. Layers: Sensory units pass signals to intermediate cells, which in turn forward them to response units. Each individual unit is shown in the 3rd image.



2.2 The Simplified Model



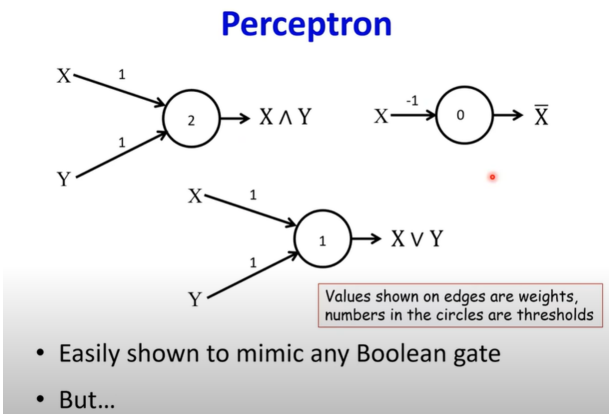
The unit gets a collection of inputs x, and each input has an associated weight w. And if the weighted sum of all of these inputs exceed some threshold
Each unit takes in inputs, each associated with a weight. If the weighted sum of the inputs exceeds a certain threshold (T), the perceptron "fires" and outputs a 1, otherwise, it outputs a 0. Weights are learned and adjusted over time.

$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i - T \geq 0 \\ 0 & \text{elsewhere} \end{cases} \tag{1}$$

He claimed that the larger version of the model could perform any Boolean logic. That isn't true, but the single version is also very powerful.

2.3 Logic Gates in the Perceptron

The individual unit can model `AND`, `OR` gates, but cannot model the `XOR` gate. However, A network of perceptrons can overcome this limitation, e.g., *multi-layered perceptrons (MLPs)* can compute `XOR` gates and other complex Boolean functions.
`Identify the types of Logic Gates in the Image Below`



2.4 Learning of Weights

We can also learn the weights (for the inputs of a single unit), which can then be able to compute any boolean function. Below is the formula. The perceptron learns by adjusting its weights based on the output. If the output is correct, no adjustment is needed. If the output is incorrect, the weights are adjusted to correct future outputs (increase or decrease depending on the error).
If the output is smaller than the desired output then the we increase the weights. The weights are updated by the product of the input and the *error* between the desired and actual outputs.

$$w = w + \eta(d(\mathrm{x}) - y(\mathrm{x})) \cdot \mathrm{x} \tag{2}$$

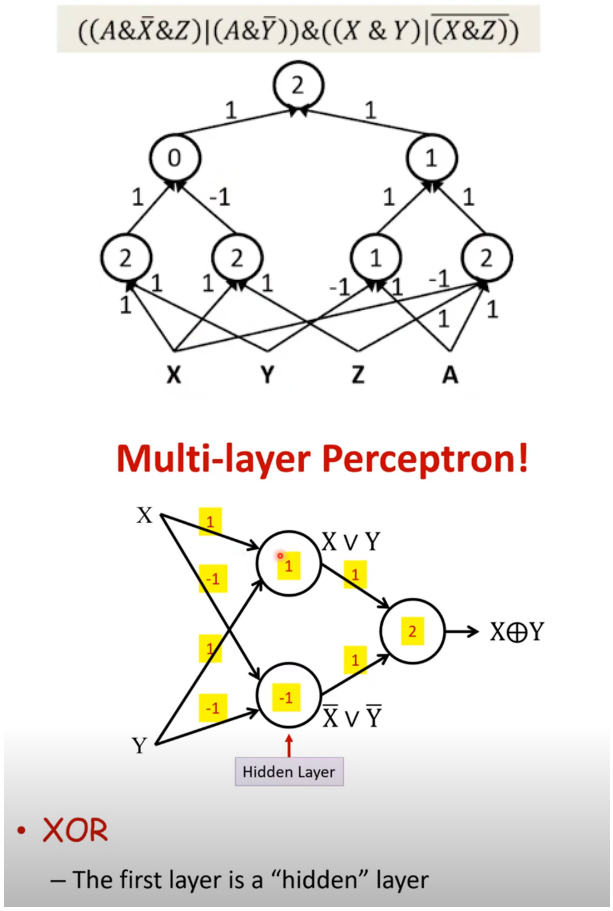
Where:

- w: weight
- x: input feature
- η: learning rate
- $d(\mathrm{x})$: desired output for input x
- $y(\mathrm{x})$: predicted output for input x

2.5 Multi-Layered Perceptrons (MLP)

> **Definition 1: Multi-Layered Perceptrons (MLP)**
>
> MLPs are universal Boolean function approximators, i.e. Given any Boolean function, an appropriately structured MLP can compute it.

An MLP can compute arbitrarily complex boolean functions. In other words if I form a sufficiently complicated network of basic perceptrons, I can compute any boolean function. This is why an MLP is called a Universal Boolean Function. It doesnt mean a simple MLP can compute any Boolean Function, rather it means that if you give me a boolean function I can `Always` design an MLP that will compute it. As it was mentioned above that a perceptron cannot model the `XOR` Gate, but using an MLP we can, as shown in the Image below: `XOR Gate can be made with another way/architecture by through an MLP - how?`

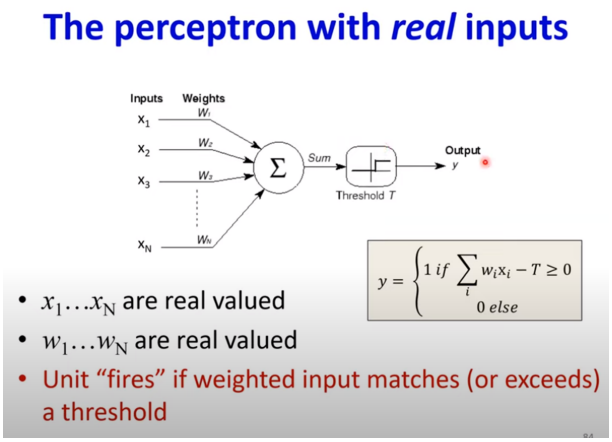


Summary 1: Story uptil now

- Neural Networks began as computational models of the brain
- Neural Networks are connectionist machines
- M&P model showed neurons as Boolean Threshold units
- M&P model models the brain as performing proposotional logic but had no learning rule.
- Hebbs learning rule showed that neurons fire together and wire together but was unstable.
- The perceptron is a variant of the M&P model with a provably convergant learning rule.
- But individual units are limited in their capacity
- MLPs can model any arbitrarily complex Boolean functions

But the issue is that the Brain is not Boolean i.e. we work with real inputs and make non-boolean inferences/predictions.

2.6 The Preceptron with Real Inputs



The same thing works with real inputs as well. Instead of having all inputs (x) boolean, all inputs and weights are `Real`. But the output would be boolean. So for now equation (1) represents the operation of the unit of a perceptron.

2.6.1 Rewriting the Perceptron equation

Below is our current equation for the operation/function of a single unit of perceptron:

$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i - T \geq 0 \\ 0 & \text{elsewhere} \end{cases} \tag{3}$$

The bias Term b

Currently the Threshold T is not incorporated into the model. To make the equation more flexible we incorporate it into the model as a *bias* term b.

Activation Function θ

- Currently our outputs are also Real. In order to make our outputs `Discrete` (which is necessary for classification) we need a function which maps this real value to a specific range (like 0 or 1 in the case of the step function)
- Currently the perceptron can only compute a linear combination of inputs i.e. it is unable to model complex `Non-Linear Decision Boundaries`.

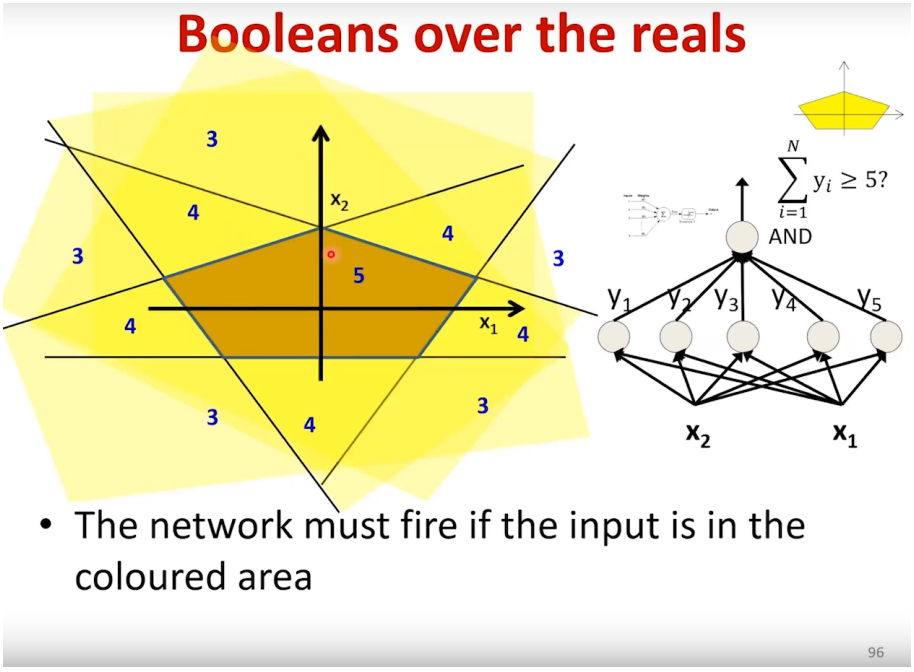
And so because of the above two reasons, we re-write equation (3) as equation (4), note that both of the equations are same:

$$y = \theta(\sum_i w_i x_i + b) \tag{4}$$

`Equation 4 is computing an Affine function of the inputs and putting that through a threshold activation`. Affine functions have an offset i.e. they have non-zero intercepts.
$\sum w_i x_i = 0$ is a linear function and the equation of the line goes through the origin. Adding a bias makes the equation it's still going to be an equation of a line/hyper-plane but the intercept is not going to be through the origin.
So we are first computing an affine function of the inputs and passing it through some activation function. Besides the `Threshold Activation`, we also have `Sigmoid` and `ReLU` activation.
`We can model any Deicision Boundary using the perceptron`

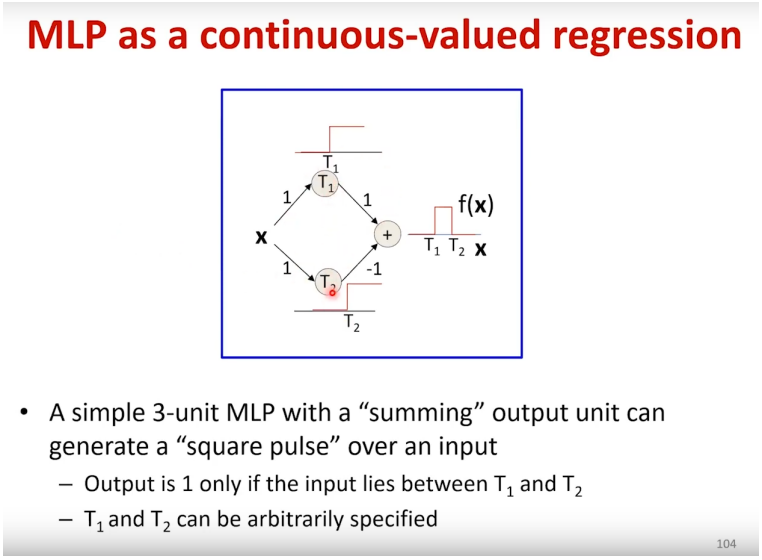
2.7 Classification using the MLP

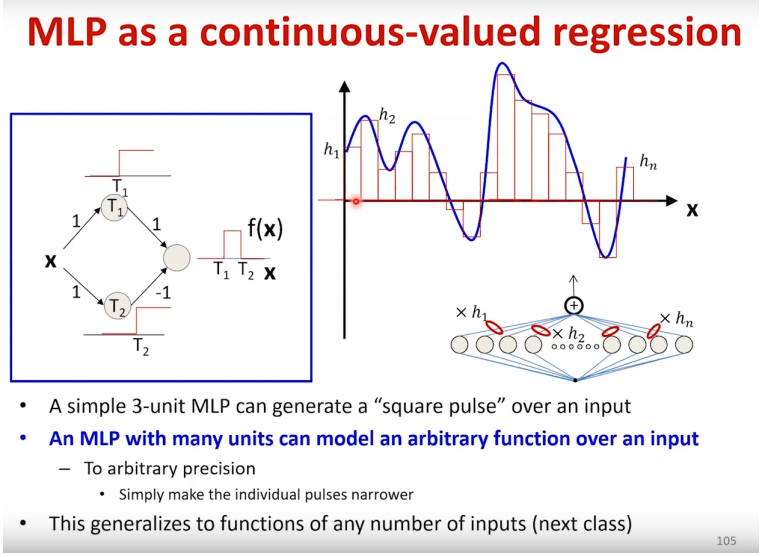
We can model any decision boundary using the perceptron. The problem of classification is about finding the decision boundary in high-dimensional spaces. Any classification boundary and I can always construct a multi-layered perceptron that captures that classification boundary Rosenblatt demonstrated that if inputs for which outputs are 1 or 0 are linearly separable, the perceptron learning rule can compute the Boolean function exactly.



`How many threshold activation perceptrons will we need in an MLP to model a hexagonal decision region (a decision boundary bounded by a six-sided Polygon) over a 2 dimensional input space?`

2.8 Regression using the MLP





2.9 Summary

Summary 2: Story uptil now

- MLPs can model any arbitrarily complex Boolean functions
- MLPs are connectionist computational models
- MLPs are classification engines
- MLP can also model continuous valued functions
- (All?) AI tasks are functions that can be modelled by the network