National University of Computer and Emerging Sciences

Lab Manual 08
Al2002-Artificial Intelligence Lab

Department of Data Science
FAST- NU, Lahore, Pakistan

Single Perceptron

A perceptron is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and process elements in the training set one at a time.

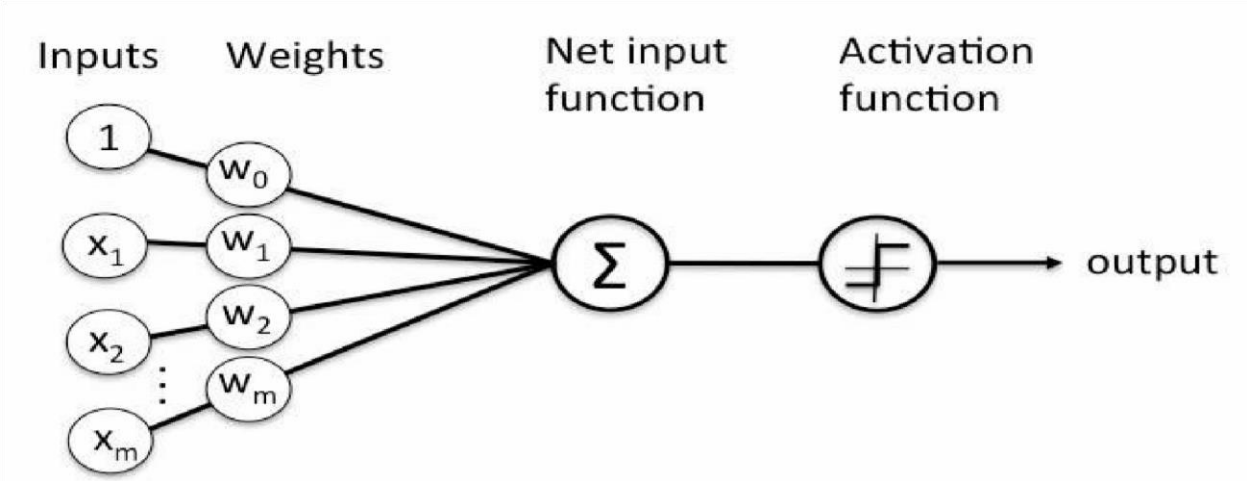

Figure 1 Single Perceptron Illustration

1.1 Types of Perceptrons

Single layer Perceptrons can learn only linearly separable patterns.

Multilayer Perceptrons or feedforward neural networks with two or more layers have the greater processing power.

1.2 Perceptron Function

Perceptron is a function that maps its input "x," which is multiplied with the learned weight coefficient; an output value "f(x)"is generated.

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

In the equation given above:

"w" = vector of real-valued weights

"b" = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
"x" = vector of input x values

$$\sum_{i=1}^{m} w_i x_i$$

"m" = number of inputs to the Perceptron

The output can be represented as "1" or "0." It can also be represented as "1" or "-1" depending on which activation function is used.

1.3 Inputs of a Perceptron

A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The image below shows a Perceptron with a Boolean output.

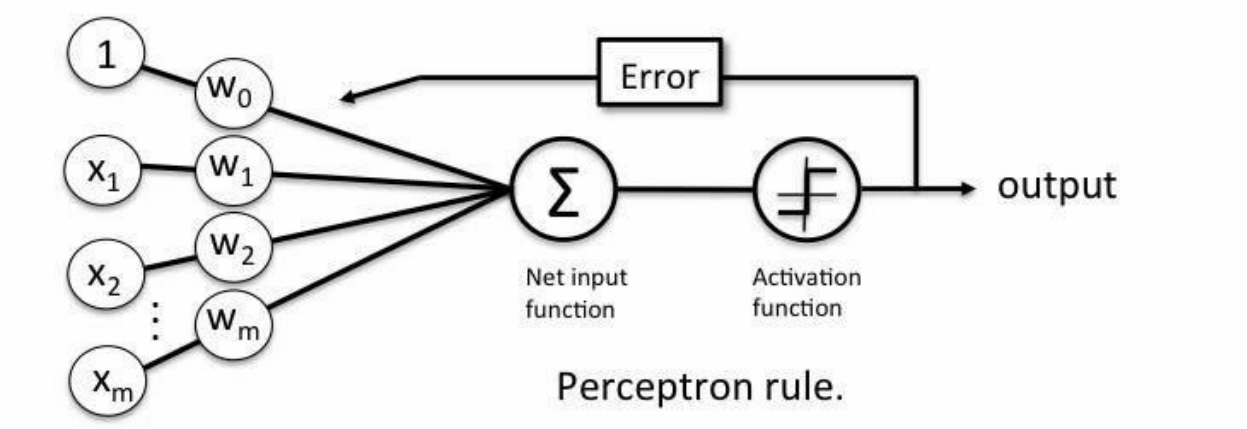

Figure 2 Inputs in a Perceptron Learning Process

A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function "∑" multiplies all inputs of "x" by weights "w" and then adds them up as follows:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

1.4 Activation Functions of Perceptron

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.

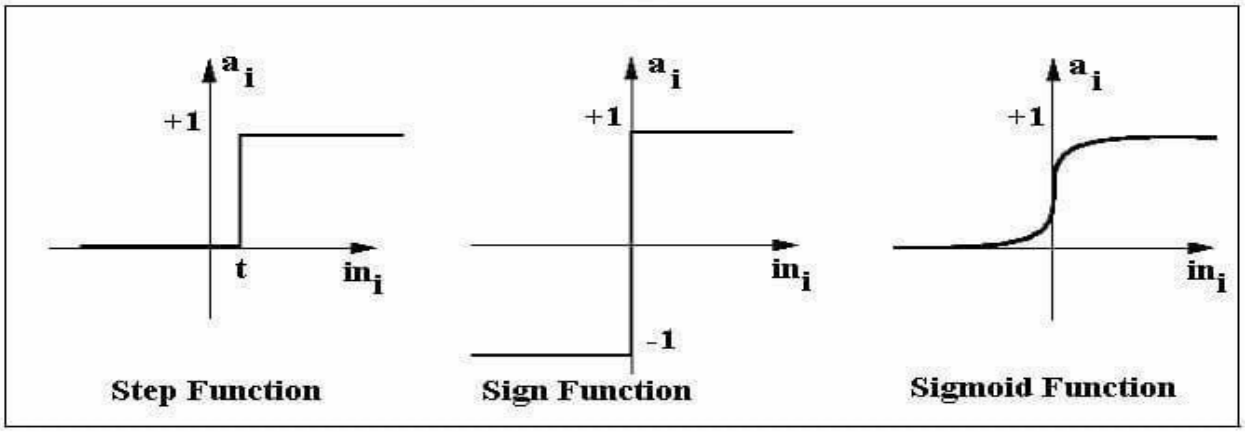

Figure 3 Various Activation Functions

For example:

```
If ∑ wixi> 0 => then final output "o" = 1 (issue bank loan)
Else, final output "o" = -1 (deny bank loan)
```

Step function gets triggered above a certain value of the neuron output; else it outputs zero.

Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not.

Sigmoid is the S-curve and outputs a value between 0 and 1.

1.5 Perceptron at a glance

- Weights are multiplied with the input features and decision is made if the neuron is fired or not.
- Activation function applies a step rule to check if the output of the weighting function is greater than zero.
- Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1.
- If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.
- Types of activation functions include the sign, step, and sigmoid functions.

2 Perceptron Training Algorithms

In this course, we have covered two training algorithms namely, perceptron learning rule and gradient descent. These algorithms would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.

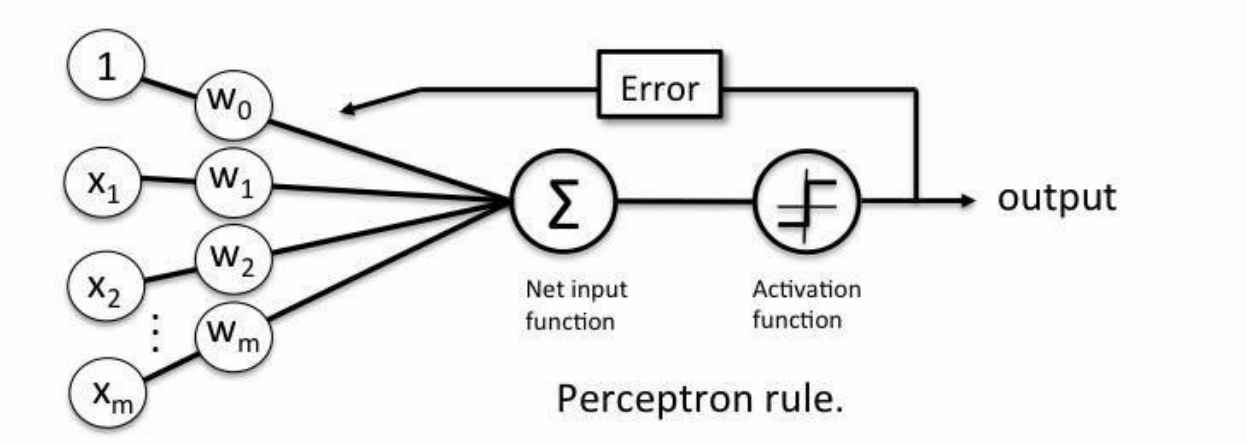

Figure 4 Learning Process

The perceptron receives multiple input signals and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output.

In this lab session, we only focus on the perceptron learning rule. This rule requires the data to be linearly separable. The weight update rule is given by:

$$W_{new} = W_{old} - \alpha \underbrace{\frac{dJ}{dW}}_{\text{gradient}}$$

5 Logic Gates

Logic gates are the building blocks of a digital system, especially neural network. In short, they are the electronic circuits that help in addition, choice, negation, and combination to form complex circuits.

Using the logic gates, Neural Networks can learn on their own without you having to manually code the logic. Most logic gates have two inputs and one output.

Each terminal has one of the two binary conditions, low (0) or high (1), represented by different voltage levels. The logic state of a terminal changes based on how the circuit processes data. Based on this logic, logic gates can be categorized into seven types:

- AND
- NAND
- OR
- NOR • NOT • XOR

5.1 Implementation of Logic Gates with Perceptron

The logic gates that can be implemented with Perceptron are discussed below.

AND

If the two inputs are TRUE (+1), the output of Perceptron is positive, which amounts to TRUE. This is the desired behavior of an AND gate.

```
x1= 1 (TRUE), x2= 1 (TRUE) w0 =
-.8, w1 = 0.5, w2 = 0.5
=>  o(x1, x2) => -.8 + 0.5*1 + 0.5*1 = 0.2 >  0
```

OR

If either of the two inputs are TRUE (+1), the output of Perceptron is positive, which amounts to TRUE.

This is the desired behavior of an OR gate.

```
x1 = 1 (TRUE), x2 = 0 (FALSE) w0 = -
.3, w1 = 0.5, w2 = 0.5
=>  o(x1, x2) => -.3 + 0.5*1 + 0.5*0 = 0.2 >  0
```

Lab Tasks

1 Implement a **single-layer perceptron** to model the **AND** and **OR** logic gates. (10+10)

x_1	x_2	y_{AND}
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	y_{OR}
0	0	0
0	1	1
1	0	1
1	1	1

Do not use built in functions.

```
# Step function (Activation function)
def step_function(z)

# Perceptron Learning Algorithm
def train_perceptron(X, y, epochs=10, learning_rate=0.1)

# Predict function
def predict(X, weights, bias)
```

```
#Plot decision boundary
def plot_decision_boundary(X, y, weights, bias, title)
```

You can get help here: https://medium.com/analytics-vidhya/implementing-perceptron-learning-algorithm-to-solve-and-in-python-903516300b2f

2 Implement XOR using Multilayer perceptron (10)

Since XOR is **not linearly separable**, a **single-layer perceptron** cannot solve it. You will use a **hidden layer** to enable the MLP to learn the XOR logic.

Input A	Input B	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

- **Build an MLP Model**
  - Input Layer: **2 neurons** (for x1 and x2).
  - Hidden Layer: **2 neurons**, activation function **ReLU**.
  - Output Layer: **1 neuron**, activation function **Sigmoid**.
- **Train the Model**
  - Use **Binary Cross-Entropy** as the loss function.
  - Use **Adam optimizer**.
  - Train for **500 epochs**.
- **Evaluate the Model**
  - Compute **accuracy on the XOR dataset**.

- Display the **predictions** for all possible inputs.

Get help here: https://www.geeksforgeeks.org/how-neural-networks-solve-the-xor-problem/