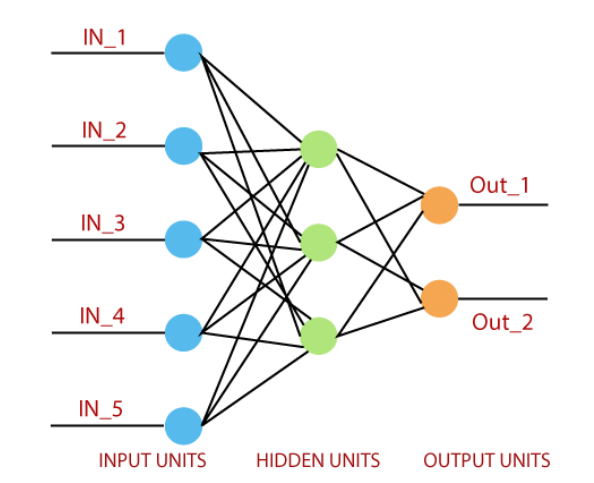
**Lab 7 - Single-Layer Perceptron**

A **Single-Layer Perceptron (SLP)** is one of the simplest types of artificial neural networks used for **binary classification** tasks. It consists of a single layer of weights connecting inputs to an output neuron.

To Understand the single layer perceptron, it's is Important to understand the Artificial neural network (ANN).

An ANN is an information processing system whose mechanism is inspired by the function of biological neural Artificial neural networks have many interconnected computing unit.



**Single Layer Perceptron**

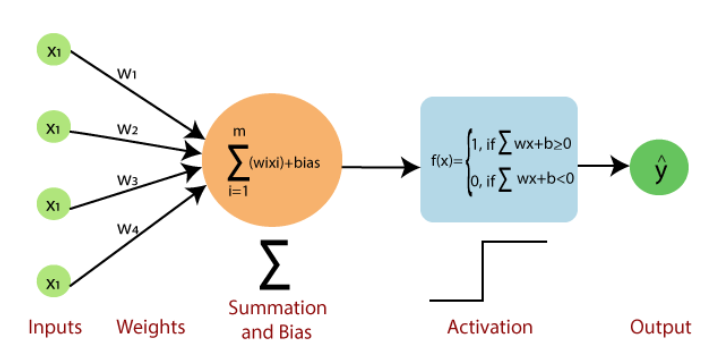
1. Single Layer Perceptron is the second proposed neural model after McCulloch-pitts neuron.

2. It consist of a vector of weights. The calculation of the single-layer is done by multiplying the sum of the input vectors of each value by corresponding elements of the weight vector.

3. Single Layer Perceptron is a feed-forward network based on a threshold transfer function.

4. Single Layer Perceptron is a simple type of artificial neural network.

5. It can only classify linearly separable cases with a binary target (1,0).



**Perceptron Learning Algorithm**

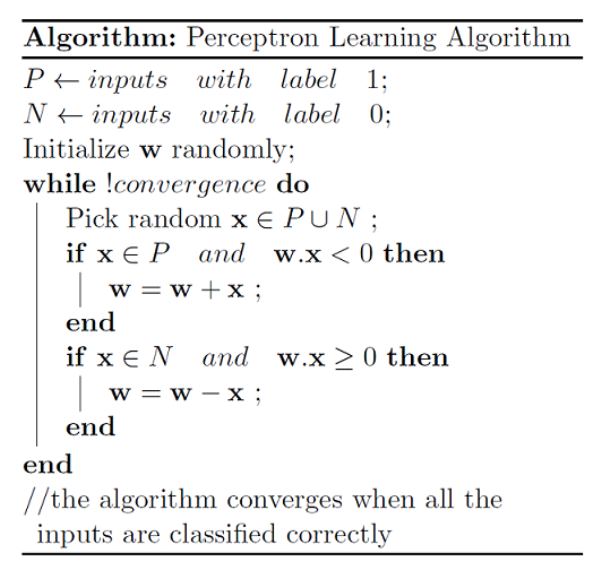
1. The single layer perceptron does not have a prori knowledge. So, the initial weights are assigned randomly.

2. Single Layer Perceptron sums all the weighted inputs and if the sum is above the threshold (predetermined value), then the neuron will be activated otherwise not.

3. If the target value and the predicted value are same, then the performance is considered satisfactory and no changes to the weights are made.

4. However, if they are not same, then the weights need to be changed to reduce the error.

5. Because single layer perceptron is a linear classifier and if the cases are not linearly separable then the learning process will never reach a point where all the cases are classified properly.



**Activation Function**

1. Activation functions are mathematical functions that can be used in Perceptrons to determine the output given its input.

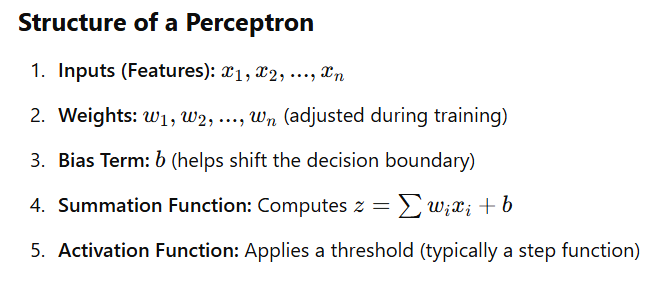
2. As we said it determines whether the neuron (Perceptron) needs to be activated or not.

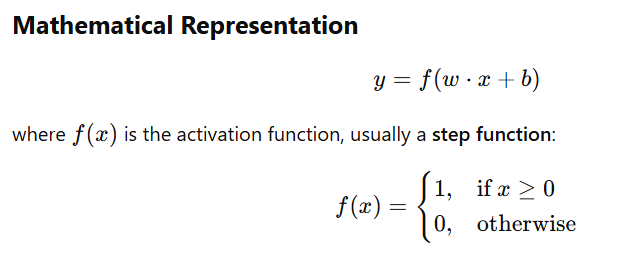
3. Activation functions take in a weighted sum of the input data, called the activation, and produce an output that can be used for prediction.

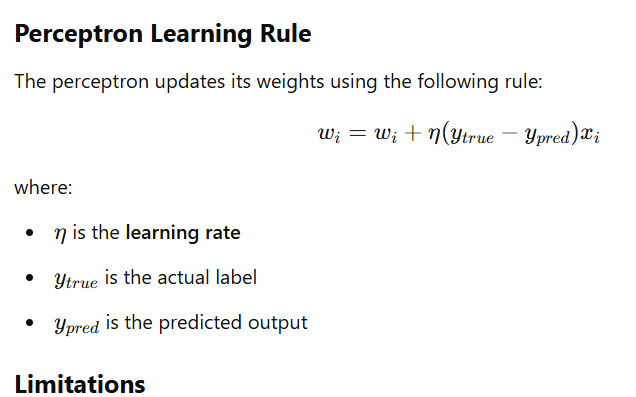
4. Activation functions are an essential part of Perceptrons and neural networks because they allow the model to learn and make decisions based on the input data.

5. They also help to introduce non-linearity into the model, which is necessary for learning more complex relationships in the data.

6. Some common types of activation functions used in Perceptrons are the Sign function, Heaviside function, Sigmoid function, ReLU function, etc.

****

****

****

* **Only works for linearly separable problems** (e.g., AND, OR).
* **Cannot learn XOR function** since it's not linearly separable.

**Sample Code in Python:**

**Single layer perceptron for AND gate**

import numpy as np

class Perceptron:

def \_\_init\_\_(self, input\_size, learning\_rate=0.1, epochs=10):

self.weights = np.zeros(input\_size + 1) # +1 for bias

self.learning\_rate = learning\_rate

self.epochs = epochs

def activation(self, x):

"""Step activation function"""

return 1 if x >= 0 else 0

def predict(self, inputs):

"""Make a prediction based on current weights"""

summation = np.dot(inputs, self.weights[1:]) + self.weights[0] # Bias term

return self.activation(summation)

def train(self, X, y):

"""Train the perceptron using the perceptron learning rule"""

for \_ in range(self.epochs):

for inputs, label in zip(X, y):

prediction = self.predict(inputs)

error = label - prediction

# Update weights and bias

self.weights[1:] += self.learning\_rate \* error \* inputs

self.weights[0] += self.learning\_rate \* error

# Example Dataset: AND Gate

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Inputs

y = np.array([0, 0, 0, 1]) # Labels (AND output)

# Train perceptron

perceptron = Perceptron(input\_size=2)

perceptron.train(X, y)

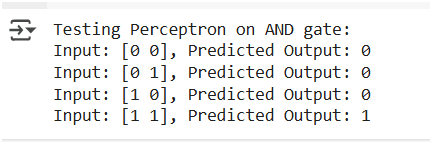
# Test perceptron

print("Testing Perceptron on AND gate:")

for inputs in X:

print(f"Input: {inputs}, Predicted Output: {perceptron.predict(inputs)}")

**Result:**



**Explanation:**

1. Initialize Weights: All weights (including bias) start at zero.
2. Activation Function: Uses a simple step function.
3. Training: Updates weights using the Perceptron Learning Rule.
4. Testing: Predicts output for each input in the dataset.

**Single layer perceptron for OR gate**

import numpy as np

class Perceptron:

def \_\_init\_\_(self, input\_size, learning\_rate=0.1, epochs=10):

self.weights = np.zeros(input\_size + 1) # +1 for bias

self.learning\_rate = learning\_rate

self.epochs = epochs

def activation(self, x):

"""Step activation function"""

return 1 if x >= 0 else 0

def predict(self, inputs):

"""Make a prediction based on current weights"""

summation = np.dot(inputs, self.weights[1:]) + self.weights[0] # Bias term

return self.activation(summation)

def train(self, X, y):

"""Train the perceptron using the perceptron learning rule"""

for \_ in range(self.epochs):

for inputs, label in zip(X, y):

prediction = self.predict(inputs)

error = label - prediction

# Update weights and bias

self.weights[1:] += self.learning\_rate \* error \* inputs

self.weights[0] += self.learning\_rate \* error

def test\_perceptron(perceptron, X, gate\_name):

"""Test perceptron and print results"""

print(f"\nTesting Perceptron on {gate\_name} gate:")

for inputs in X:

print(f"Input: {inputs}, Predicted Output: {perceptron.predict(inputs)}")

# Define input data

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

# OR Gate Training

y\_or = np.array([0, 1, 1, 1]) # OR gate outputs

perceptron\_or = Perceptron(input\_size=2)

perceptron\_or.train(X, y\_or)

test\_perceptron(perceptron\_or, X, "OR")

# XOR Gate Training (Will Fail)

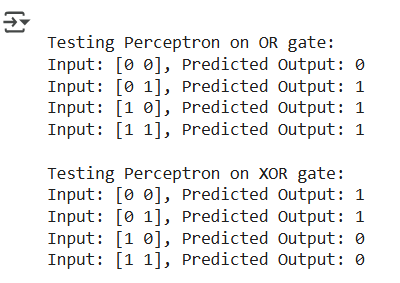
y\_xor = np.array([0, 1, 1, 0]) # XOR gate outputs

perceptron\_xor = Perceptron(input\_size=2)

perceptron\_xor.train(X, y\_xor)

test\_perceptron(perceptron\_xor, X, "XOR")

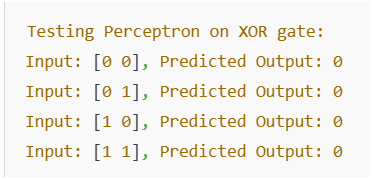
**Result:**



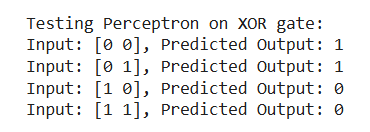
Note:

A Single-Layer Perceptron can only solve linearly separable problems. OR is linearly separable, but XOR is not. The XOR function requires a nonlinear decision boundary, which a single-layer perceptron cannot learn.

**For XOR Gate:** The perceptron will fail to learn XOR, usually outputting **all 0s or all 1s** because XOR is **not linearly separable**.



Sometimes the results may vary



Note:

To correctly classify XOR, we need a **Multi-Layer Perceptron (MLP)** with a **hidden layer** using an **activation function like ReLU or sigmoid**.

Reference:

* + 1. <https://www.javatpoint.com/>
    2. <https://www.kaggle.com/>
    3. <https://openai.com/index/chatgpt/>
    4. <https://towardsdatascience.com/>