SIMPLE PERCEPTRON SELF IMPLEMENTATION TO ILLUSTRATE.

ING Jeison Robles Arias.

The next implementation is based on the concepts and theoretical concepts studied on the Hagan Book:

Hagan M.T., Demuth, H.B., Beale, M.H.., & De Jesus, O (2014). Neural network design (2nd ed.). Stillwater, OK: Martin Hagan.

And basically follows the most trasendental concepts on NN Design. Here I Illustrate the perceptron:

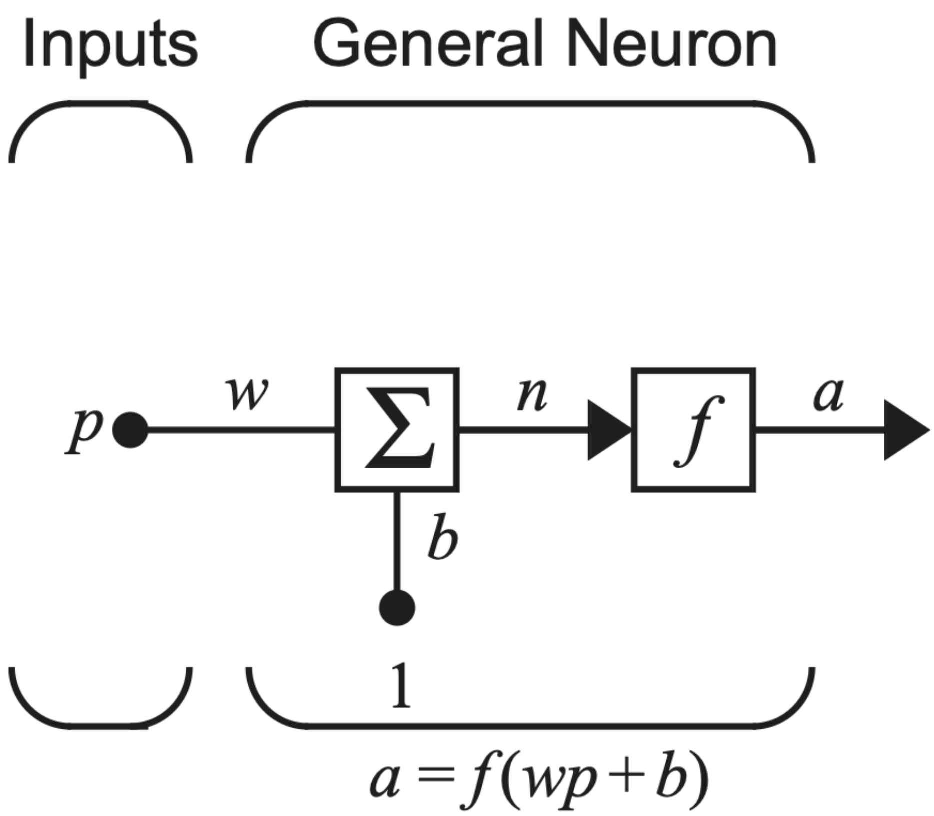
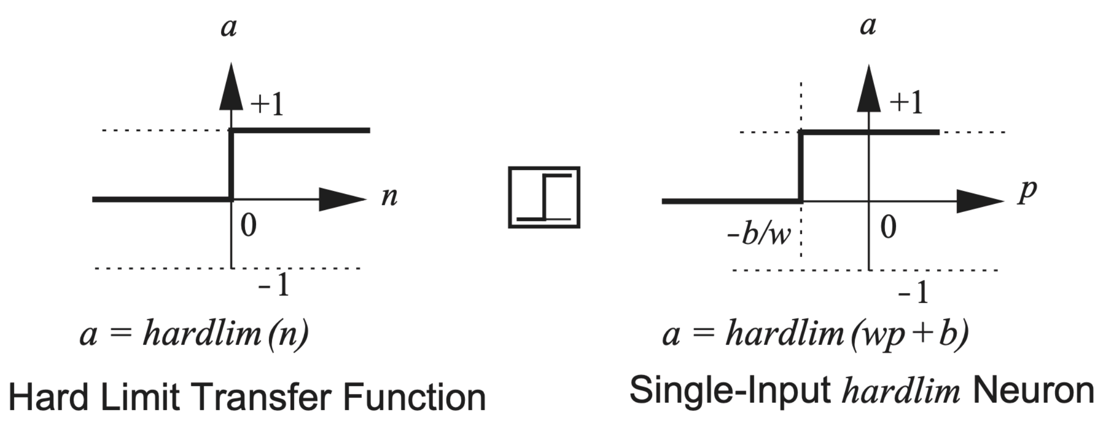


Figure 2.1 Single-Input Neuron

That is going to be executed with a simple hardlim functions as its activation function (understanding that another approaches with relu, etc could be tested.)



In [ ]:
```
import numpy as np
import time
```

In [ ]:
```
#Define an step function for the activation stage
def step_function(x):
    return 1 if x>=0 else 0
```

In [ ]:
```
class Perceptron:

    def __init__(self, input_size, learning_rate=0.1, epochs = 1000, patienc
        self.weights = np.zeros(input_size + 1) # +1 for bias
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.patience = patience

    def predict(self, inputs):
        weighted_sum = np.dot(inputs, self.weights[1:]) + self.weights[0]
        return step_function(weighted_sum)

    def accuracy(self, test_inputs, test_labels):
        correct_prediction = 0
        total_predictions = len(test_labels)

        for inputs, actual_label in zip(test_inputs, test_labels):
            prediction = self.predict(inputs)
            if prediction == actual_label:
                correct_prediction += 1
        #Calculate the accuracy as a percentage way.
        return (correct_prediction / total_predictions) * 100

    def train(self, training_inputs, labels):
        best_accuracy = 0
        patience_counter = 0

        for epoch in range(self.epochs):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - predictior
                self.weights[0] += self.learning_rate * (label - prediction)

            accuracy = self.accuracy(training_inputs, labels)
```

```
            print(f"Epoch {epoch + 1}/{self.epochs} - Accuracy: {accuracy:.2

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                patience_counter = 0
            else:
                patience_counter += 1


            #Stopping training if no improvement after patience epochs
            if patience_counter >= self.patience:
                print(f"Early stopping at epoch {epoch+1}")
                break
```

In [ ]:

```
if __name__== "__main__":
    training_inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
    labels = np.array([0,0,0,1])

    print("Starting Learning Process...")
    perceptron =  Perceptron(input_size=2, epochs=100, patience=10)

    perceptron.train(training_inputs, labels)

    print("Testing Perceptron on AND gate:")
    for inputs in training_inputs:
        print(f"{inputs} -> {perceptron.predict(inputs)}")
        time.sleep(1)
```

```
Starting Learning Process...
Epoch 1/100 - Accuracy: 25.00%
Epoch 2/100 - Accuracy: 50.00%
Epoch 3/100 - Accuracy: 100.00%
Epoch 4/100 - Accuracy: 100.00%
Epoch 5/100 - Accuracy: 100.00%
Epoch 6/100 - Accuracy: 100.00%
Epoch 7/100 - Accuracy: 100.00%
Epoch 8/100 - Accuracy: 100.00%
Epoch 9/100 - Accuracy: 100.00%
Epoch 10/100 - Accuracy: 100.00%
Epoch 11/100 - Accuracy: 100.00%
Epoch 12/100 - Accuracy: 100.00%
Epoch 13/100 - Accuracy: 100.00%
Early stopping at epoch 13
Testing Perceptron on AND gate:
[0 0] -> 0
[0 1] -> 0
[1 0] -> 0
[1 1] -> 1
```

Here one can see that event the NN having be equiped with an strong learning process, the easy the example allows us to learn with high accuracy in just 13 executed epocs.