Namal University Mianwali

Perceptron Learning simulation, on linearly separable data.

28.11.2022

Maria Maqsood(NIM_BSCS_2020_37)

Muhammad Abrar Hussain(NIM_BSCS_2020_62)

Overview

Humans are blessed with the ability to think and decide, they observe things by using the sense of hearing, seeing, touching, tasting, and smelling. When humans use these senses, a message or a signal is sent to the brain. The brain gives an output in return. The outcome is intelligently generated by a basic unit of the brain which we call a neuron that controls all human activities and helps humans in decision-making.

Many researchers, scientists, and biologists have been working to understand this working and decision-making process of the neuron and to explain this model.

In this report, we will be explaining the model of neurons. We will also be explaining and relating this model from the perspective of Machine Learning.

What is the neuron?

Neurons or nerve cells are the functional units of the brain and nervous system. These cells receive information from the external environment/surroundings. These neurons after receiving information send a message to the muscles of the body so that the action can be taken. The message sent is in the form of electrical signals after making some decisions. In neurons, the decision is made on the basis of priorities set for different works/tasks. For example: if we have to do two jobs, one is more important than the other, then the decision-making will be such that muscles will be asked to do important work at first. Similarly, if we have to prepare for an exam and at the same time we have to play cricket then our neurons will give importance to exams and send messages to muscles for study. On the other hand, if we are studying continuously for 5 to 6 hours the decision-making will be such that the message will be to entertain. so that the brain can be relaxed. No doubt, neurons make decisions but also set the priority of one task to another(In the ANN case we do this with the help of weight). There are roughly 100 billion neurons in the human brain. Neurons are connected with each other from one end or the other. If there were no neurons present in the human brain, a human wouldn't be able to make decisions and differentiate between things. In fact, all the work that humans have done in the past years wouldn't be done, if they were not blessed with a brain.

structure:

Neurons are just like a tree. There are three parts of a neuron which are dendrites, axons and the third one is soma. As a neuron is like a tree its parts can be represented as branches, roots, and trunk, respectively.

Dendrites:

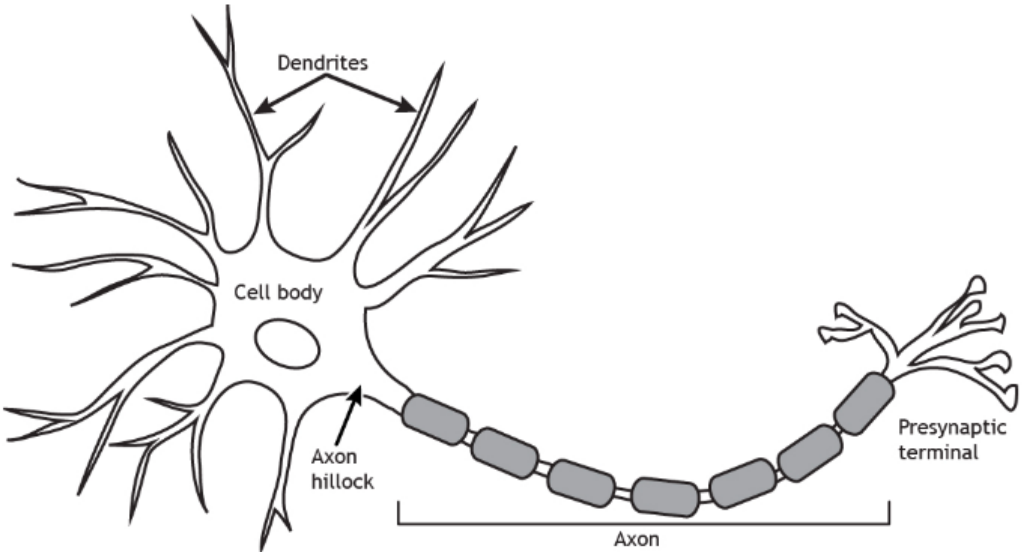
The parts of the neurons which receive information in the form of input are called dendrites. Dendrites carry impulses from other neurons and carry them into the cell body.

Axons:

Axons carry impulses away from the cell body. Axon is many times thinner than human hair. It is the part of the neural network through which information or data is transferred from one neuron to another neuron. When a neuron wants to talk to another neuron, it sends an electrical message called an action potential throughout the entire axon. Neurons cannot work properly if axons get damaged.

Soma:

Where the nucleus lies, soma is present there. Proteins are present in the soma and these proteins are transferred to dendrites and axons. It is just like the CPU of a computer.



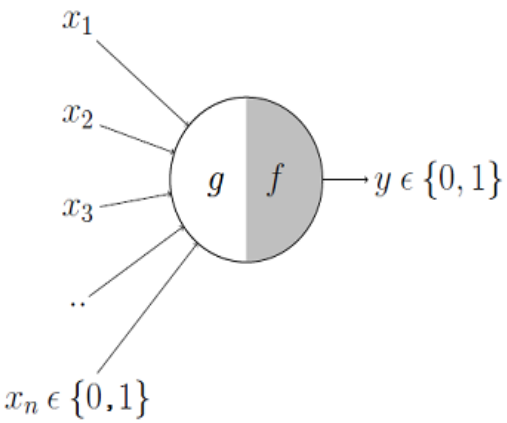
Neuron model in the context of Machine learning:

The concept of artificial neural network(ANN) comes from the human brain(neurons). The human decision-making process is controlled by neurons, as well as we train machines so that machines can make decisions and differentiate between two or more data types.

In 1943, Frank Rosenblatt proposed the idea of a perceptron learning model. Later on, in 1969, Minsky Papert refined this idea of the perceptron learning model. The first

computational model of a neuron was given by Warren McCulloch who was a neuroscientist and Walter Pitts who was a logician.

This is the McCulloch Pitts neuron model:



Here, $x1, x2, x3 \ldots\ldots xn$ are the inputs while y ε {0,1} means that "y" is the output which is in binary form "0" or "1". In this "g" is the input part of the neuron and "f" represents the output based on the inputs. Here, "g" is working as a dendrite, and "f" is that part of the neural network which is used for decision-making. Some inputs/information is given more importance while inhibitory inputs are the inputs that are not so important. For example, I am a student, and the input going into my mind is;

1) x1—-> I have to prepare for the mid-exam which is tomorrow.
2) x2—-> I have to do some shopping.
3) x3—-> I have to play a cricket match in society.

The working of a perceptron is that it will make the decision according to the priority as a student makes a decision that which task is more important than the other.

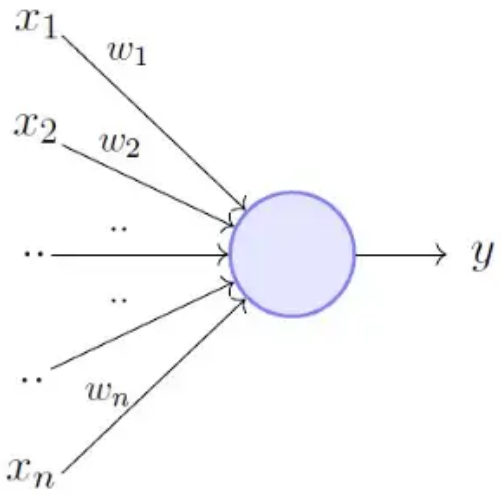
$$(x1, x2, x3 \ldots\ldots\ldots\ldots xn) \;=\; f(x) \;=\; \Sigma\, xi$$

$$y \;=\; 1 \qquad\qquad if\ f(x) \geq 0$$

$$y \;=\; -1 \qquad\qquad if\ f(x) < 0$$

The task which is important should be done first. Then, the perceptron will make the decision that as a student one should study for the exam. i.e; x1 should be done and then the signal will go toward the muscles for the action. Similarly, according to the perceptron learning model, we can say that the weight or task which is important should be represented as 1 while the task or weight which is less important should be represented as 0. But in the McCulloch model, there was no concept of weights.

Perceptron:



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Perceptron learning Algorithm:

Sigmoid function:

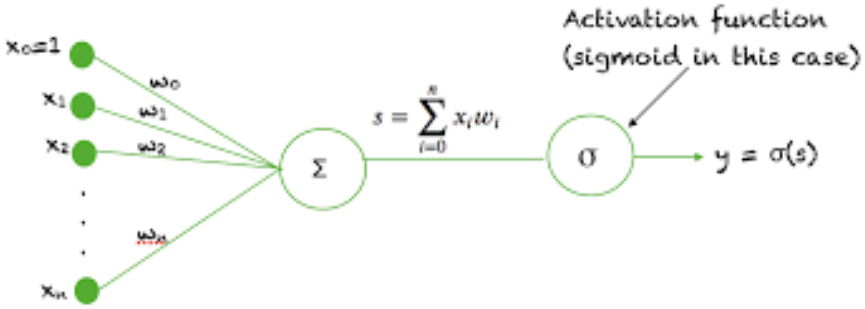
To train perceptrons we have a data set having initial weights(w0, w1, w2), train input data set, and train output set. We have used the sigmoid function here that gives us the sum of all inputs($x1$, $x2$, $x3$) by multiplying it with their respective weights($w0$, $w1$, $w2$). The initial value of $x0$ = 1. This is a bias value of $x0$. we have fixed it, so that the value of $w0$ remain constant. In this way, weight $w0$ is added with the weights $w1x1$ and $w2x2$. It helps the models to shift the activation function toward the positive or negative side.

The activation function maps the sum of $w0x0 + w1x1 + w2x2$ on just two values. In our case, these two values are 1, and -1.

$$sum = w0x0 + w1x1 + w2x2$$

$$If\ sum >= 0: Sum = 1 \qquad else: Sum = -1$$

The sigmoid and activation function diagram and code part is below:




```
import matplotlib.pyplot as plt

initial_weights = [-0.6, 0.75, 0.5]
train_Input = [[1,1],
               [9.4,6.4],
               [2.5,2.1],
               [8,7.7],
               [0.5,2.2],
               [7.9,8.4],
               [7,7],
               [2.8,0.8],
               [1.2,3],
               [7.8,6.1]]

train_Output = [1, 1 ,1, -1, -1, -1, -1, 1, -1, 1]

count = 0
while count < 30:
    for i in range(len(train_Input)):
        sublist = train_Input[i]
        sum = initial_weights[0] + initial_weights[1]*sublist[0] + initial_weights[2]*sublist[1]

        '''---activation function to map output of sum on 1 and -1--- '''

        if sum >= 0:
            actual_output = 1
        else:
            actual_output = -1
```

Importance of activation function and weight updation:

As we have discussed earlier, the activation function maps the output of the sigmoid function between two values “1” and “-1”. The output/value after mapping can be 1 or -1 after comparing it with the output training data set value. If the value

from the sigmoid function is equal to the output list value at the same index no updation in weights occurs, otherwise, the weight will be updated.

The formula we will use for weight updation we use will be:

$$w\_new = w\_old + delta\_w$$

$$delta\_w = learning\_rate(actual\_output - desired\_output)xi$$

Actual_output = output comes after dot product(from sigmoid function)

Desired_output = value from train_output given us to train model

xi = number from train_input w.r.t to w0, w1, w2 it will x0, x1, x2 respectively.

learning_rate = value helps in optimal weight choice.

w_old = old weights are those we find in one step early

After getting a new weight (w_new), the previous list of weights will be updated. Weight updation continues until the model does not get trained. We can also run the model to specific iterations. In our case, we only allow thirty iterations.

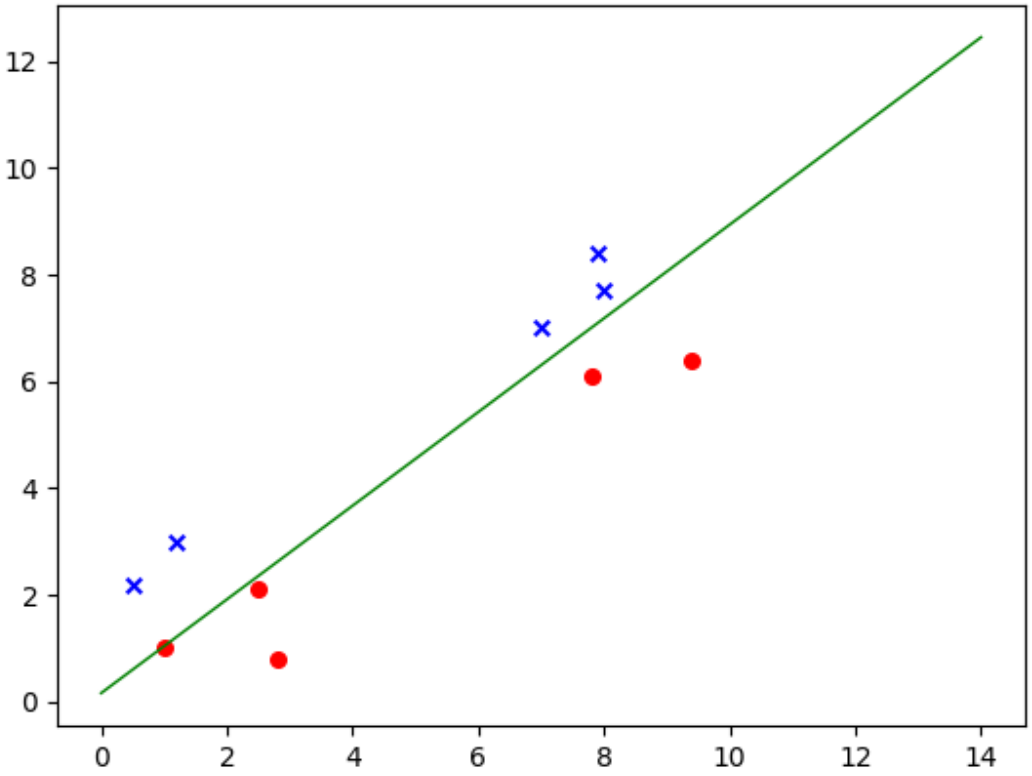
```
    if train_Output[i] == actual_output:
        pass
    else:
        '''
            ---formula to update weight---
        w_new = w_old + delta_w
        delta_w = learning_rate(actual_output - desired_output)xi
        '''
        desired_output= train_Output[i]
        delta_w = (0.02* (actual_output - desired_output))
        w0_new = initial_weights[0] + (delta_w)*1
        w1_new = initial_weights[1] + (delta_w)*sublist[0]
        w2_new = initial_weights[2] + (delta_w)*sublist[1]


        ''' ---weight updation in initial weight list--- '''
        initial_weights[0], initial_weights[1], initial_weights[2] = w0_new, w1_new, w2_new
print(initial_weights)
count += 1
```

Weights:

Weights have key rules in a neural network. Weight is the boundary decider between two or more types of data. We can randomly decide the weight value.

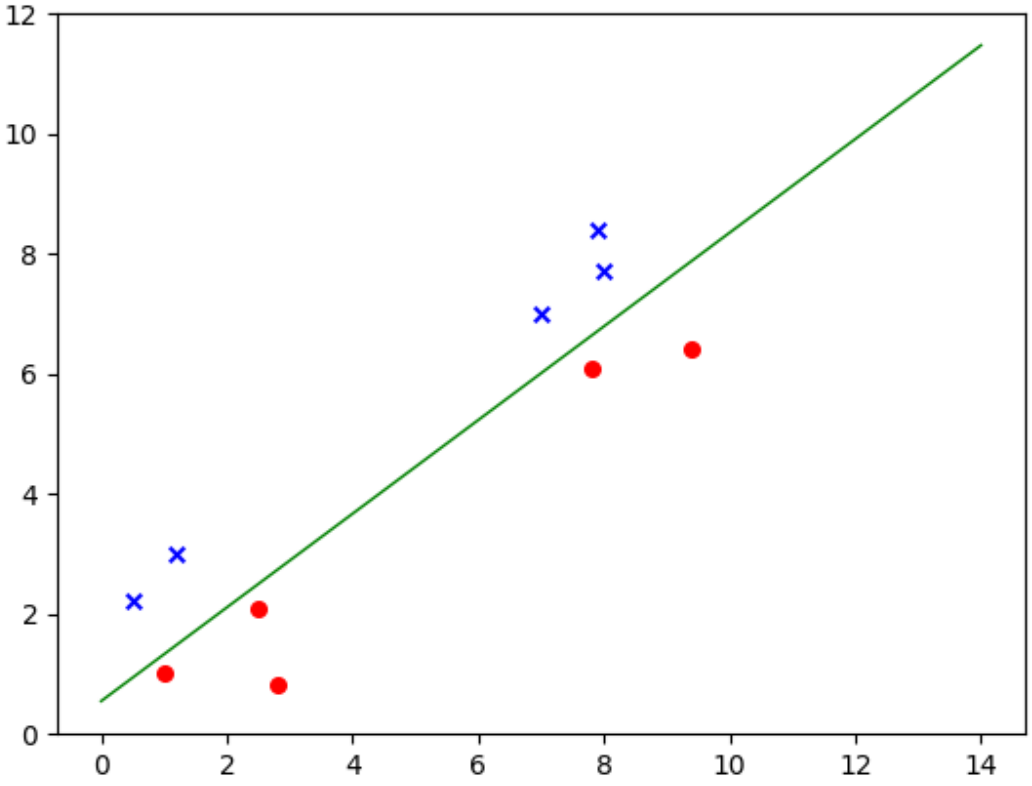
For example, we run the model on $w0 = -0.6,\ w1 = 0.75,\ and\ w2 = 0.5$

The output graph is:



For example, we run the model on $w0 = 500,\ w1 = 700,\ and\ w2 = 900$

The output graph is:



so, we can conclude that no matter how much weights value small or large these can take us to one conclusion after updating on every iteration.

Model training iterations:

We run the model maximum of up to 30 iterations so, we can get the most precise weights.

Our initial weights were: [-0.6, 0.75, 0.5]

After running the model up to 30 iterations our finalized weights were: [5.400000000000004, 30.219999999999974, 34.46]

But we also observed our model trained upon only 5 iterations. The weights after 5 iterations were: [0.39999999999999997, 5.62, 6.160000000000001].

So, we can say the weight after 5 iterations are enough to differentiate between the given data set. Our model was trained after making 5 iterations.

Leaning rate:

Hypothesis:

Learning rate play an important role in weight updation. No matter how small the value of the learning rate. When the value of the learning rate is small it can define weight more precise and more chances of model learning.

If we fix the larger value of the learning rate it creates big jumps in previous weights and new weights. In this way, there is more probability we can miss the right weight for our model.

Observation over hypothesis:

My hypothesis proves wrong when we test it on the model. No matter whether we decide the value of the learning rate large or small. our learning rate does not disturb the model learning. Because it applies to all weight updation equally.

Scenario 1:

When our learning_rate = 0.02

Iterations = 30

Initial weights = [-0.6, 0.75, 0.5]

Final weights = [5.400000000000004, 30.219999999999974, 34.46]

Scenario 2:

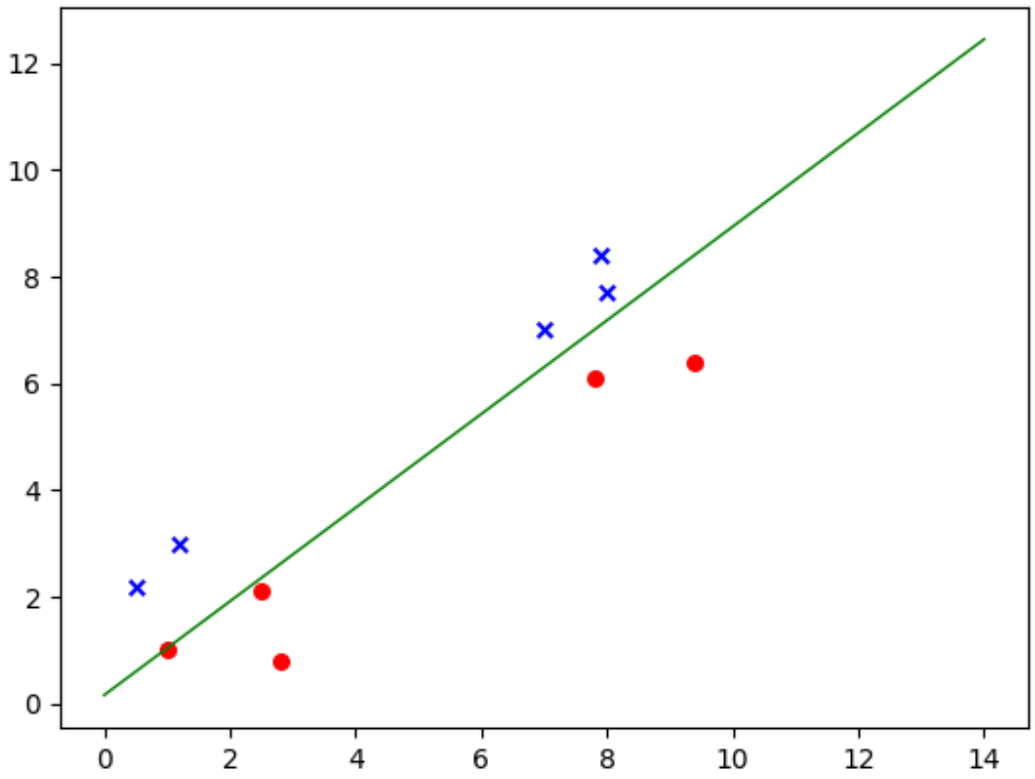
When our learning_rate = 200

Iterations = 30

Initial weights = [-0.6, 0.75, 0.5]

Final weights = [59999.4, 295200.7, 339600.5]

Graph on learning rate 0.02 and 200 remain same:



Graphical representation of model:

For the graphical representation of the model we use the python library matplotlib.pyplot.

As we have linearly separable data, we model it on “1” and “-1”. The data approach to “1” is represented by a cross character in blue color, and the data approach “-1” is represented with a circle in red color as shown in the graph above.

Then we use the equation of the line to draw the boundary between linearly separable data.

The equation is:

$$w2x2 = w0 + w1x1$$

$$x2 = w0 + w1x1/w2$$

Here we use the most precise and updated values of $w0, w1, w2.$ so, we can draw the clearly separable boundary between two types of data. We choose the value of x1 between the range of 1 and 15. So we can find x2 value to get line points.

```
    ''' ---Graph Draw & boundry line between two data--- '''

for i in range(len(train_Input)):
    sublist = train_Input[i]
    desired_output_point = train_Output[i]
    if desired_output_point == 1:
        plt.scatter(sublist[0], sublist[1], label= "circle", color= "Red", marker= "o", s=30)
    else:
        plt.scatter(sublist[0], sublist[1], label= "cross", color= "blue", marker= "x", s=30)



    ''' ---To draw separator line--- '''
x = [] # x_point to draw separation line in graph
y = [] # Y_point to draw separation line in graph
for num in range(0,15):
    x.append(num)

    x2 = ((initial_weights[0]) + (initial_weights[1]*num))/initial_weights[2]
    y.append(x2)

plt.plot(x, y, color="green", linewidth = 1)
plt.show()
```