Perceptrons

Moshiur Rahman

June 2023

1 Introduction

A Perceptron is a single unit of an Artificial Neural Network. It is a supervised model of learning. It is online and error driven. A perceptron takes a vector of real numbers as inputs, computes a weighted sum of them and feeds the sum to an "activation function" to give a binary output. Hence, it is called a feedforward algorithm. The perceptron was invented in 1943 by Warren McCulloch and Walter Pitts and was first implemented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt [2][4]. It is one of the most important and fundamental algorithms in machine learning. The perceptron is formally defined as:

$$f(x) = \begin{cases} 1, & W_i \cdot x_i + b > 0 \\ 0, & W_i \cdot x_i + b \leq 0 \end{cases}$$

where, W_i is the i^{th} weight, x_i is the i^{th} input, and b is the bias term

The perceptron is the most fundamental model of learning. It has led to the development of further learning models. AdaLiNe (Adaptive Linear Neuron) is one such example. The perceptron is limited to binary classification of linearly separable data, and to solve this problem, MultiLayer Perceptron (MLP) has been developed [3][1].

2 Perceptron Structure

The Perceptron is inspired by and modeled after the neurons in our brain, which form a network like structure and make decisions by sending electrical signals to each other. A Perceptron can be decomposed into the following components:

- Inputs: A vector of real numbers
- Weights: A vector of values in range $[-1, 1]$, same size as the input vector.
- Bias term: A constant used to shift the decision boundary to correctly fit the dataset.
- Activation Function: The mathematical function that is used to compute the output.
- Output: A binary value (0/No/False or 1/Yes/True)

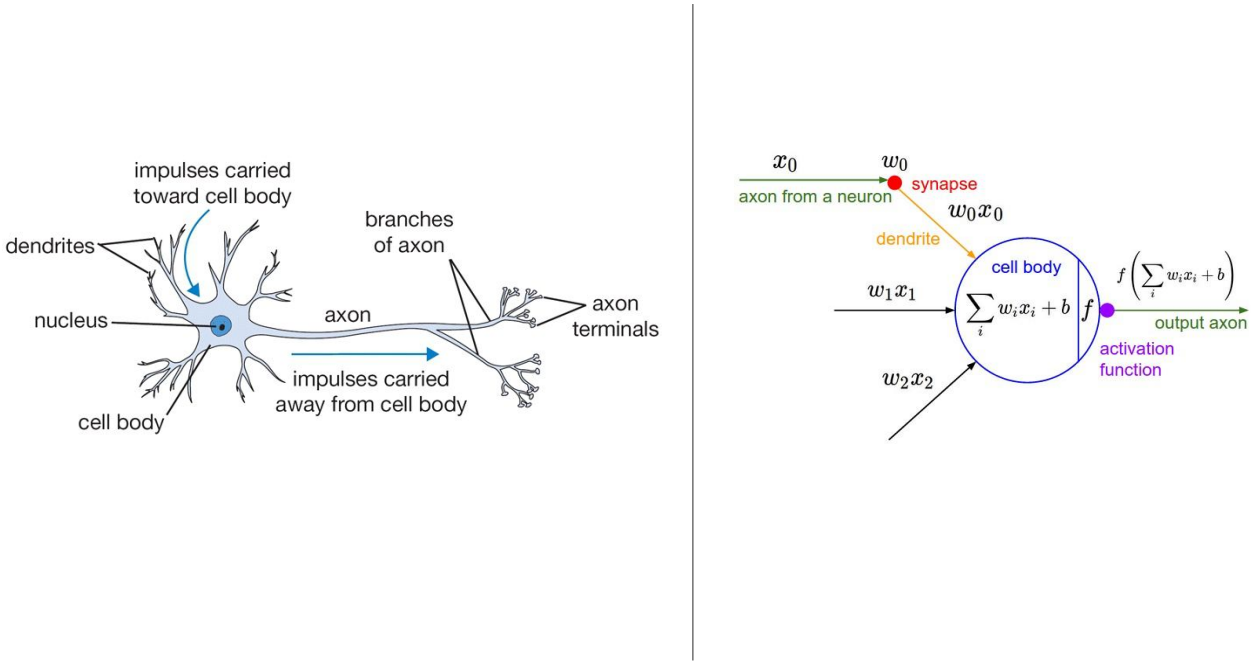


Figure 1: A neuron (left) and its model - the perceptron (right) [Images taken from Stanford CS231n course webpage]

2.1 Perceptron Weights

2.1.1 Weight Initialization

The weights of a perceptron can be initialized in multiple ways. Some of the standard initialization methods are explaines in brief:

- **Zero Initialization:** All the weights are initialized as 0, then each weight is updated on every iteration of the learning process. Zero initialization is a simple but inefficient weight initialization method.
- **Random Initialization:** Each weight is initialized randomly. A well-seeded random number generator is required so that the method works efficiently. But this method is also quite inefficient as weights might be initialized to be too small or too large.
- **Uniform Initialization:** Each weight is initialized to a random number between a maximum and minimum value. This ensures that the weights are not too small or too large and improves efficiency.
- **Gaussian Initialization:** Each weight is initialized by following a Gaussian distribution. This ensures that the weights are spread out more evenly, making the learning process more efficient.

2.1.2 Weight Interpretation

The weights of a perceptron can be interpreted as the strength of the connections between the inputs and output of the perceptron. The value of each weight is between -1 and 1. If the value of a weight corresponding to an input is -1, it indicates that that input has a very strong negative impact on the output. Similarly, an input having weight 1 means that that input has a very strong positive impact on the output. A weight of 0 means that input has no connection to the output. Geometrically, the weight can be thought of as an n-dimensional vector in space, where n is the number of inputs. The weight vector determines the direction and slope of the hyperplane generated by the perceptron.

2.2 The Bias

The bias is a constant term that is added to the weight vector. It shifts the hyperplane by adding a constant dimension to it. Without the bias value, all hyperplanes would have to pass through the origin. It is possible that there is no hyperplane passing through the origin that correctly classifies the data set. This is where the bias term is helpful. To accomodate the bias, an extra constant input with value 1 is added to the input vector. As an example, we can take a perceptron with only one input, x.

$$\text{The weight vector, } W = \begin{bmatrix} b \\ w \end{bmatrix}$$

$$\text{The input vector, } x = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Then, the perceptron equation is

$$f(x) = W \cdot x = b + wx$$

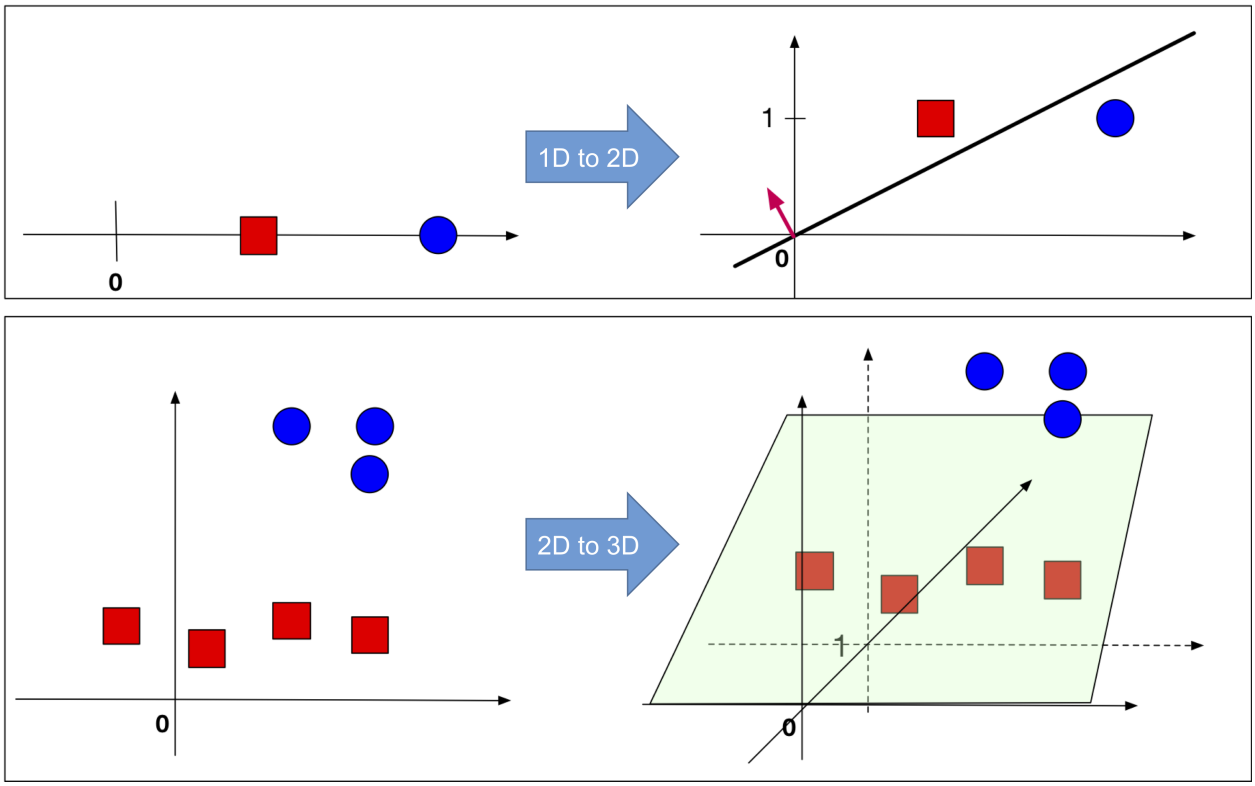


Figure 2: The bias is used to add a constant dimension to the inputs so that a decision boundary exists [Image taken from Cornell CS4780 course webpage]

2.3 Activation Functions

The activation function is a mathematical function that decides whether a perceptron should give a positive output or a negative output based on the inputs. There are different types of activation functions, each has its advantages and disadvantages different use cases. The most popular activation functions are:

1. Heaviside Step Function:

$$H(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

   - Range: $\{0, 1\}$
   - Advantage: Simple implementation
   - Disadvantage: Inefficient in terms of performance

2. Logistic Activation Function (Sigmoid):

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

   - Range: $(0, 1)$
   - Advantage:
     (a) Smooth gradient
     (b) Normalized outputs
   - Disadvantage:
     (a) Vanishing gradient problem
     (b) Computationally expensive
     (c) Outputs are not zero centered

3. Hyperbolic Tangent Function (Tanh):

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

   - Range: $(-1, 1)$
   - Advantage: Outputs are zero centered
   - Disadvantage: Computationally expensive

4. Rectified Linear Unit (ReLU):

$$(x)^{+} = max(0, x)$$

   - Range: $[0, \infty)$
   - Advantage: Computationally efficient
   - Disadvantage: Dying ReLU problem

5. Leaky Rectified Linear Unit (Leaky ReLU):

$$(x)^{+} = max(0.1x, x)$$

   - Range: $[-\infty, \infty)$
   - Advantage: Prevents the dying ReLU problem
   - Disadvantage: Inconsistent predictions

6. Softplus Function:

$$S(x) = ln(1 + e^{x})$$

   - Range: $(0, \infty)$
   - Advantage: Better approximator than ReLU
   - Disadvantage: Computationally expensive

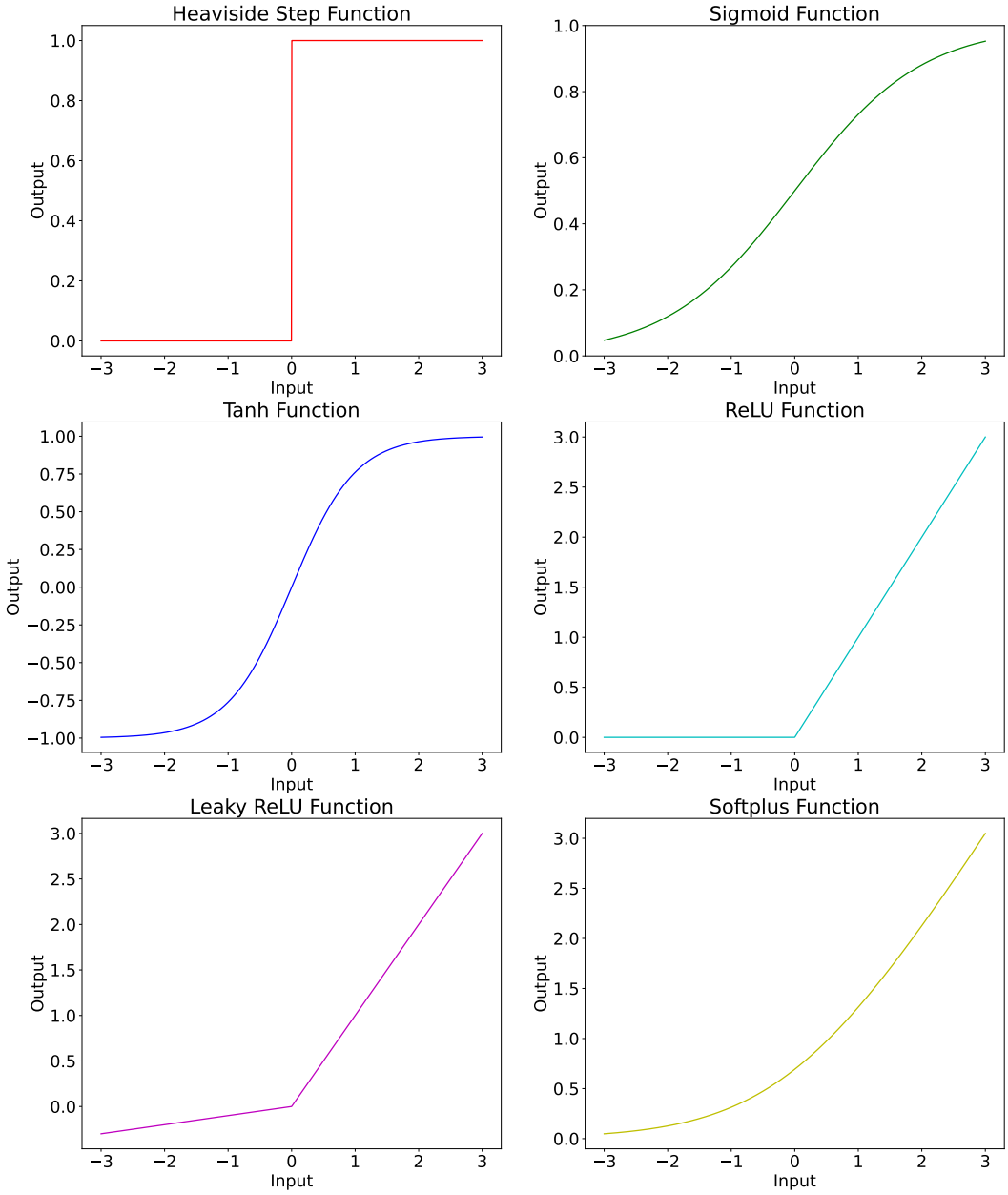


Figure 3: Graphical presentation of different activation functions

3 Perceptron Learning Rule

The goal of the perceptron learning rule is to determine a weight vector using which the perceptron generates a correct hyperplane (linear classifier) to separate the data into two classes. The perceptron learning algorithm is as follows (we assume zero initialization of weights and bias term for simplification):

Algorithm 1 Perceptron Learning Algorithm

```
w ← 0                                          ▷ initialize weights
b ← 0                                          ▷ initialize bias
while TRUE do
    e ← 0                                      ▷ keep track of the errors
    for (x_i, y_i) ∈ D do                      ▷ loop over each (data, label) pair in the dataset, D
        if y_i(w_i · x_i) ≥ 0 then
            w_i ← w_i + y x_i                  ▷ update the weight of the misclassified input
            b ← b + y_i                        ▷ update bias
            e ← e + 1                          ▷ increment error count
        end if
    end for
    if e = 0 then break                        ▷ break out of the while loop if there are no errors
    end if
end while
```

4 Performance Metrics

Performance is an important issue in machine learning. It is necessary to optimize the performance of algorithms and models to get the best results. The performance of a perceptron can be measured by different metrics. Such as:

1. Accuracy: Ratio of number of correct predictions to the total number of input samples.

$$\text{Accuracy} = \frac{n(c)}{n}$$

where, $n(c)$ = number of correct predictions, n = total number of input samples

2. Confusion Matrix: Confusion matrix describes the complete performance of the perceptron. It allows to compute precision, recall, f1 score by generating the values required.

3. Precision: Ratio of true positives and total positives predicted.

$$P = \frac{TP}{TP + FP}$$

4. Recall: The ratio of true positives to the sum of true positives and false negatives.

$$R = \frac{TP}{TP + FN}$$

5. F_1 Score: F_1 score is the harmonic mean of Precision and Recall.

$$F_1 = \frac{PR}{P + R}$$

6. AU-ROC: Area Under Receiver Operating Characteristics (AU-ROC) curve is one of the most widely used metrics for evaluating the performance of a binary classifier. It makes use of true positive rates (TPR) and false positive rates (FPR).

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

5 Perceptron Convergence Theorem

The Perceptron Learning Algorithm will come to a stop for every finite set of linearly separable labeled samples after a certain number of repetitions. In other words, the algorithm produces a vector w that accurately classifies all the cases after a certain number of iterations, given the dataset is linearly separable. If the dataset is linearly inseparable, the Perceptron loops forever.
The Perceptron takes a set of n points each of dimension d. We define the inputs to the Perceptron as

$$(x^1, y^1), ..., (x^n, y^n)$$

where for $i \in \{1, 2, ..., n\}, \mathbf{x}^i \in \mathbb{R}^d$ and $y^i \in \{-1, 1\}$.
In order to prove that the Perceptron Learning Algorithm converges i.e it stops after a finite number of iterations, we make the following assumptions:

Assumption 1. *There exists some* $\mathbf{w}^* \in \mathbb{R}^d$ *such that* $||\mathbf{w}^*|| = 1$ *and for some* $\gamma > 0$*, for all* $i \in \{1, 2, ..., n\}$,

$$y^i(\mathbf{w}^* \cdot \mathbf{x}^i) > \gamma$$

Assumption 2. *There exists* $R \in \mathbb{R}$ *such that for* $i \in \{1, 2, ..., n\}$,

$$||\mathbf{x}^i \leq R||$$

Assumption 1 and 2 ensures linear separability and bounded coordinates respectively. To understand the importance of these assumptions, consult [4].

Theorem 1. *Given that assumptions 1 and 2 hold, the Perceptron Learning Algorithm makes at most* $\frac{R^2}{\gamma^2}$ *iterations.*

Proof. First, we will derived a lower bound on $\mathbf{w}^k + 1$ in terms of k. Suppose, $\mathbf{w}^1 = 0$, and for $k \geq 1, \mathbf{x}^j$ is the misclassified point for k^{th} iteration. Then, we have,

$$\begin{aligned}
\mathbf{w}^{k+1} \cdot \mathbf{w}^* &= (\mathbf{w}^k + y^j \mathbf{x}^j) \cdot \mathbf{w}^* \\
&= \mathbf{w}^k \cdot \mathbf{w}^* + y^j(\mathbf{x}^j \cdot \mathbf{w}^*) \\
&\geq \mathbf{w}^k \cdot \mathbf{w}^* + \gamma
\end{aligned}$$

Applying mathematical induction, we have, $\mathbf{w}^{k+1} \cdot \mathbf{w}^* \geq k\gamma$.

As $||\mathbf{w}^{k+1}|| \times ||\mathbf{w}^*|| \geq \mathbf{w}^{k+1} \cdot \mathbf{w}^*$ and $||\mathbf{w}^*|| = 1$, we have,

$$||\mathbf{w}^{k+1}|| \geq k\gamma \tag{1}$$

Next, we will derive an upper bound on $\mathbf{w}^k + 1$. We have,

$$\begin{aligned}
||\mathbf{w}^{k+1}||^2 &= ||\mathbf{w}^k + y^j \mathbf{x}^j||^2 \\
&= ||\mathbf{w}^k||^2 + ||y^j \mathbf{x}^j||^2 + 2(\mathbf{w}^k \cdot \mathbf{x}^j) y^j \\
&= ||\mathbf{w}^k||^2 + ||\mathbf{x}^j||^2 + 2(\mathbf{w}^k \cdot \mathbf{x}^j) y^j \\
&\leq ||\mathbf{w}^k||^2 + ||\mathbf{x}^j||^2 \\
&\leq ||\mathbf{w}^k||^2 + R^2
\end{aligned}$$

Applying mathematical induction, we have,

$$||\mathbf{w}^{k+1}||^2 \leq kR^2 \tag{2}$$

From (1) and (2),

$$k^2\gamma^2 \leq \mathbf{w}^{k+1}||^2 \leq kR^2$$

which implies

$$k \leq \frac{R^2}{\gamma^2}$$

□

6 Linear Classification

The Perceptron can generate a hyperplane to separate an n-dimensional space into two classes. This is termed as linear classification. Here, a graphical simulation of the linear classification process is presented.

6.1 2D space:

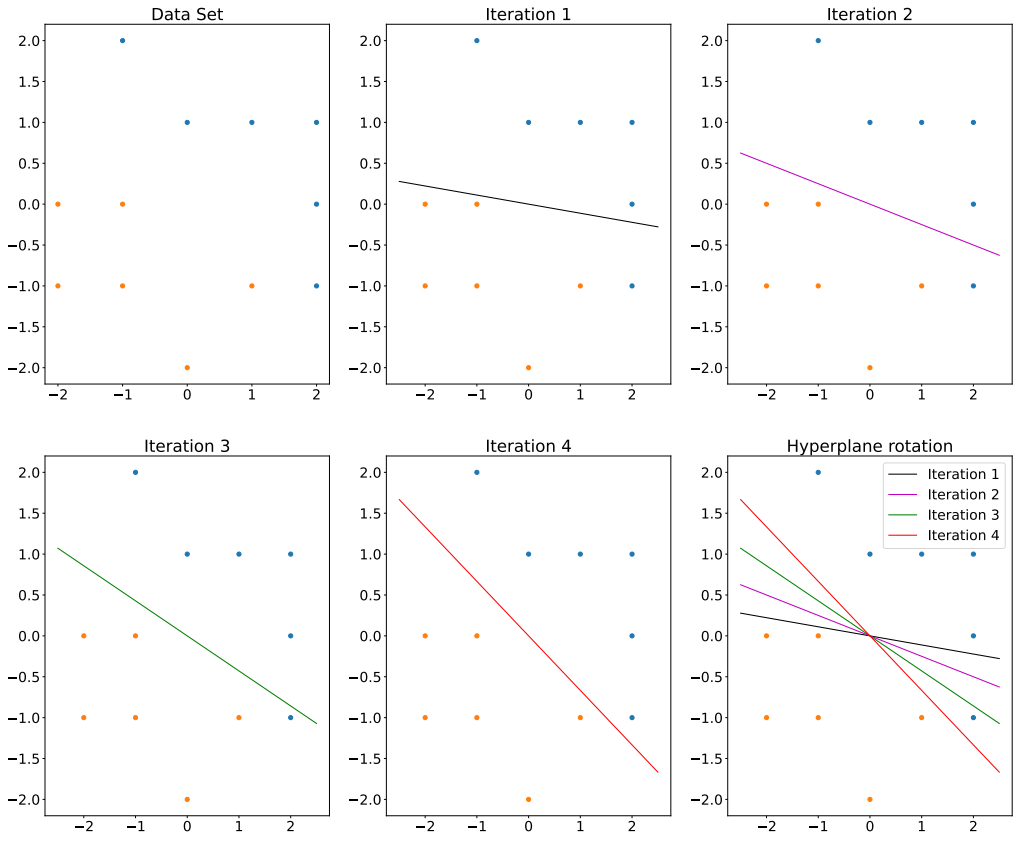
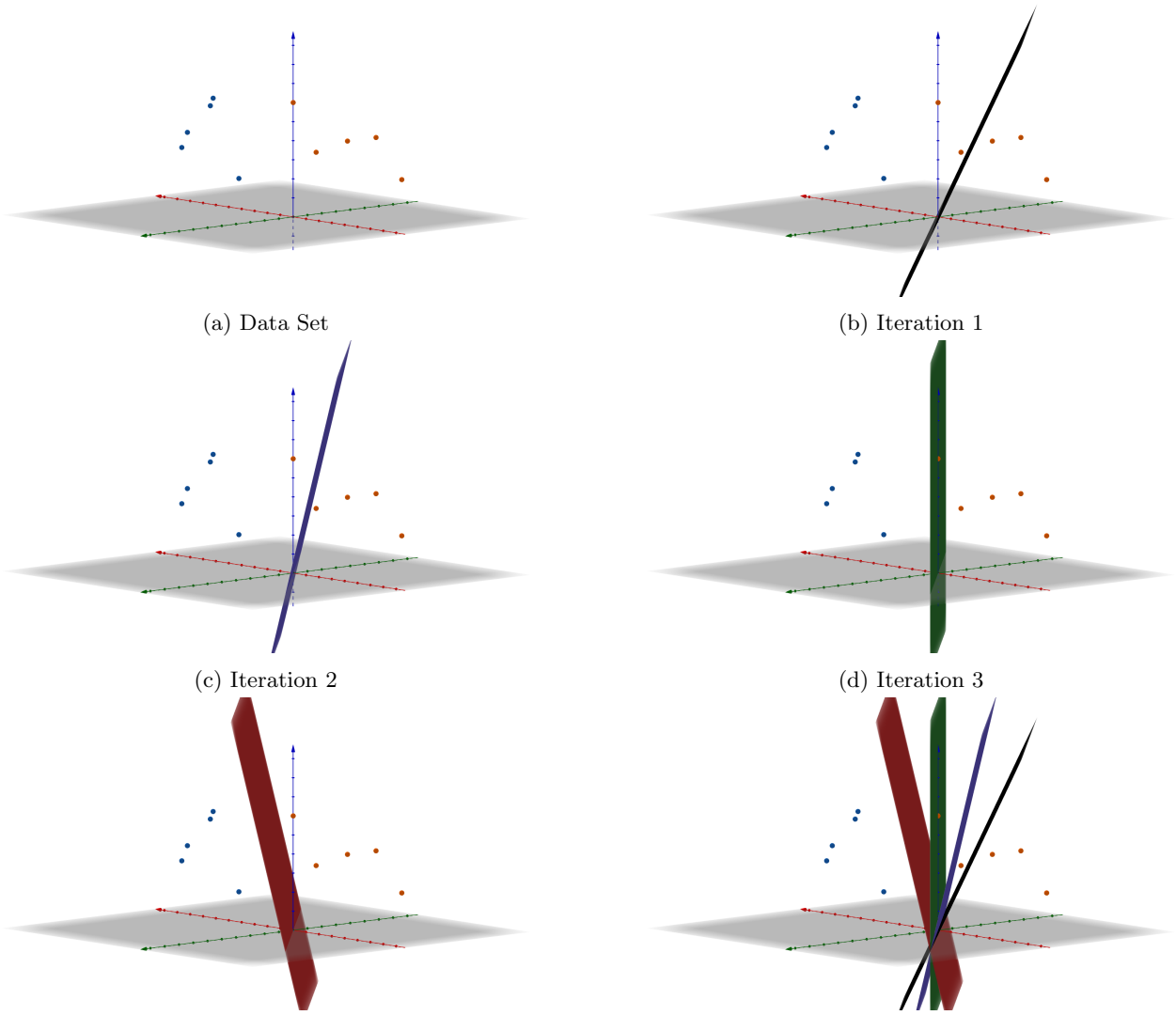


Figure 4: Linear classification in 2D space

6.2 3D space:



(a) Data Set

(b) Iteration 1

(c) Iteration 2

(d) Iteration 3

(e) Iteration 4

(f) Hyperplane rotation

Figure 5: Linear classification in 3D space

7 Limitations of the Perceptron

The Perceptron is a simple classification algorithm that works really well for some problems, specifically binary classification of linearly separable data. The algorithm becomes inefficient in any of the following situations:

1. The data is divided into more than two classes:
   The Perceptron can produce only binary outputs i.e 0 or 1. So, it can only classify data into two classes, not any more. In case of multiclass data, it is not a good idea to use the perceptron. But there is a way around. It is possible to use perceptron on multiclass data by modifying the dataset and dividing it into two classes. For example, if there are 10 classes in a dataset, it can be modified as
   0: Class 1
   1: Not Class 1
   So, data points that fall under class 1 will be classified as 0, and data points that fall under classes 2-10 will be classified as 1.

2. The data is not linearly separable:
   A classic problem in machine learning is the classification of XOR data. The perceptron can classify almost all boolean datasets (AND, OR, NAND, NOR) but it is not possible for the perceptron to classify the XOR dataset as it is not linearly separable. In order to successfully classify a linearly non-separable dataset, the approach is to connect multiple perceptrons in layers to form the Multi Layer Perceptron (MLP).

References

[1] Christopher M. Bishop. *Pattern recognition and machine learning.* Springer New York, 2016.

[2] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[3] Tom Michael Mitchell. *Machine learning.* McGraw-Hill, 1997.

[4] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.