Proceedings of the Third International Conference on Computing Methodologies and Communication (ICCMC 2019)
IEEE Xplore Part Number: CFP19K25-ART; ISBN: 978-1-5386-7808-4

A Study on Single and Multi-layer Perceptron Neural Network

Jaswinder Singh
School of Computer Engineering
KIIT University
Bhubaneswar, India
Jaswinder9051998@gmail.com

Rajdeep Banerjee
School of Mechanical Engineering
KIIT University
Bhubaneswar, India
rbanerjee140@gmail.com

***Abstract*—Perceptron is the most basic model among the various artificial neural nets, has historically impacted and initiated the research in the field of artificial nets, with intrinsic learning algorithm and classification property. It has boosted the world of neural networks and profoundly impacted the numerous advancements. From the very beginning it has proved to be the key to the way machines perceive, making them artificially intelligent through extensive training processes. In this study, the ideology of perceptron learning, its concepts, working, applications and a very brief introduction to multi - layer perceptron has been discussed.**

Keywords*— *Neural network, Feed Forward Network, artificial neuron, activation functions, training algorithm, neuron structure, prediction, multi layer neuron.

I. INTRODUCTION

Perceptron is the most basic form of neural network architecture with zero hidden layers, most prominently used for classification of linearly separable patterns. The perceptron model falls under the category of supervised learning and can be referred to as a binary classifier. The perceptron algorithm is advantageous in the scenario when we want to prioritize the impact of selected features over the others (using modifiable weights), in contrast to the decision tree algorithm(only a small number of features) and nearest neighbor algorithm(all features are given equal priority). It consists of a single McCulloch-Pitts neuron containing adjustable synaptic weights and bias.

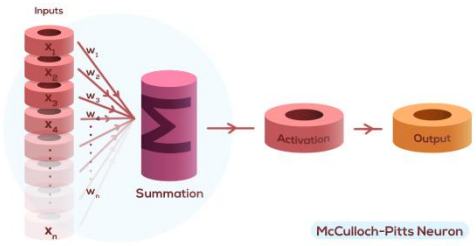

Fig. 1. Neural Network

The perceptron algorithm is the simplest configuration of ANN (Artificial Neural Network) ever being created. It was invented at the Cornell Aeronautical Laboratory in the year of 1957 by Frank Rosenblatt, funded by the United States Office of Naval Research [1]. The perceptron was first implemented in IBM704 software, also built in custom-made hardware as the Mark 1 perceptron, mainly designed to perform image recognition with the help of an array of 400 photocells, randomly connected to the neurons. Potentiometers were used to encode the weights, and their update was performed during the learning process by the electric motors. Single Layered Perceptron cannot be trained to recognize multiple classes of pattern, only capable of learning linearly separable patterns. It was later recognized that a feed-forward neural network having two or more hidden neural layer has faster processing power than any single layer perceptron [2]. Later Stephen Grossberg published three series of papers stating that these networks are capable of contrast enhancing, modeling differential & XOR functions which were published in 1972 and 1973 [3].

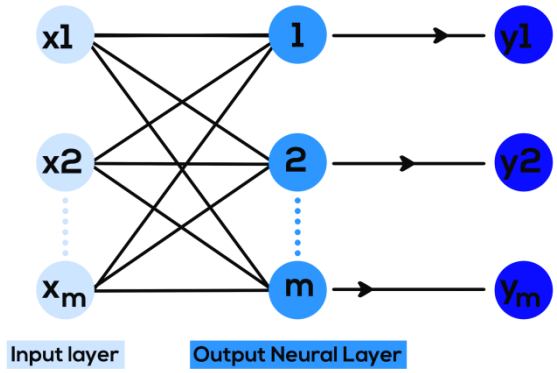

Fig. 2. Single Layered Feed-Forward Network.

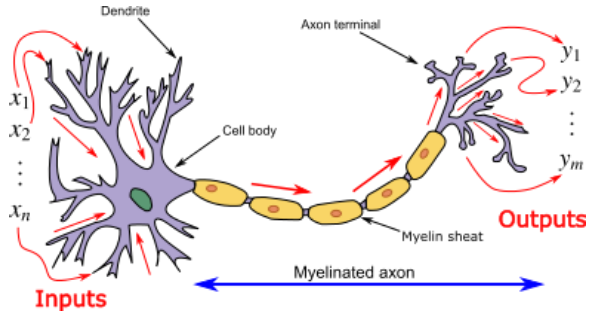

Fig. 3. Biological Neuron (Source: https://commons.wikimedia.org/wiki/File:Neuron.svg)

II. Perceptron Structure and Component

A. Structure

The artificial neuron is derived/inspired from the biological neuron. The biological neuron continuously collects stimuli (electrical impulses) from the environment or other neurons with the help of its thin extensions called "dendrites" [4] .The cell body, in turn, works on the stimuli collected by the dendrites to create an activation potential which determines if the neuron can trigger an electrical impulse along its axon[5]. The cell body passes on the impulse signal only if the summation of the electrical signals received by the dendrites passes a particular and appropriate threshold. The artificial neuron abstracts the functioning and features of the biological one to a very simplistic level. This collection of external stimuli is mapped to the artificial neuron in form of inputs values provided. The functioning of the cell body is mirrored by the use of "weights". These set of weights play the distinctive role which it defines the output. The role of the threshold is played by "threshold/bias" value while the output is determined at the leftmost end of the neuron (after the implementation of suitable activation function).The value calculated by the artificial neuron is parallel to the electrical potential accumulated in the biological neuron when this value (potential) is greater than the threshold; then similar to the biological neuron, the artificial neuron sends a positive output(an electric impulse in case of biological).This is how the decision-making ability of the cell body is implemented in the artificial neuron.

B. Components

Since the concept of artificial neuron inspired from the biological neuron several of its features are analogous.As seen in the figure, the inputs which are coming in, are similar to the impulses gathered by the dendrites of the biological neuron. The set of weights $\{w_1, w_2, \ldots, w_m\}$ are responsible for weighing each of the input values which is achieved by multiplying the weight value with the respective input.

1) *Input Signals* :$\{x_1, x_2, .., x_m\}$ are the values provided to the neuron for computation.These external values are to be worked on by the neuron for favourable output.
2) *Weights* :$\{w_1, w_2, \ldots, w_m\}$ these are the values to be multiplied to the respective inputs to quantify the impact/relevance of the input on the output. The weights decide the extent to which the input will have an impression over the output. The result can be assumed as the weighted sum of inputs.
3) *Linear aggregator* :(Σ) as the name suggests, it sums up all the weighted values
4) *Threshold /Bias* : (Θ) this is the value which the result produced by the linear aggregator must have (or greater than the bias) to generate a trigger(positive) at the output end.
5) *Activation potential (u):* it is the value obtained by the difference between the result of the linear aggregator and the bias. This value defines the shapes of the output
6) *Activation Function* :(g(u)) is a mathematical way to limit the value of output to a range of values which determines whether the result will be positive or not.
7) *Output Signal* :(y) is the final value produced by the neuron.

Thus the result of the neuron can be represented as.

$$y = g\left(\Sigma w_i * x_i - \theta\right)$$

C. Activation Function

There are different types of activation functions. Some of them are Step function, Linear function, Sigmoid Function, Tanh function, ReLu. These activation functions have different characteristic features and are used depending on what kind of problem we want the perceptron to work on. Like, Sigmoid Functions are widely used for the classification problems with the combined feature of the Step function as well as smooth. But, the problem with the Sigmoid function is that towards the end of both the sides of the graph shown below, y corresponds very less to the corresponding changes in the X values. This indicates that in this region their derivatives give a constant as being unresponsive to the changes resulting in independence in its characteristics. Such nature signifies that the network refuses to learn further or very slowly.

The commonly used activation functions are as follows :

1. Step function :

$$g(u) = \begin{cases} 1 \; if \;\; u \geq 0 \\ 0 \; if \;\; u < 0 \end{cases}$$

Where $u = \Sigma w_i * x_i - \theta$

When the potential created by the artificial neuron (similar to the electrical potential created by the biological neuron) is zero or positive, the output assumes a unitary positive value; otherwise, the result reflects null.

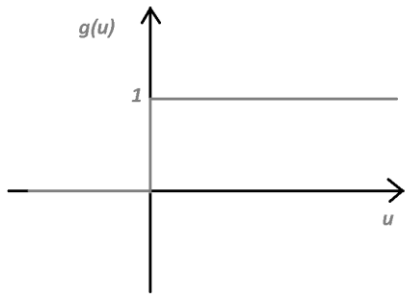

Fig. 4. Step Function

2. Bipolar step function:

$$g(u) = \begin{cases} 1 \; if \;\; u > 0 \\ 0 \; if \;\; u = 0 \\ -1 \; if \;\; u < 0 \end{cases}$$

As the function suggests, if the potential created is greater than zero then the output assumes a unitary positive value, if the potential; created is null, then the output is null and if the potential is negative, the output assumes a negative unitary value.

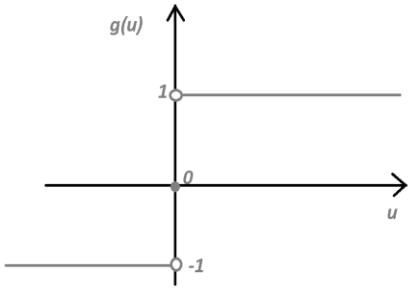

Fig. 5. Bipolar Step Function

3. Sigmoid function [6] :

The sigmoid function is especially used when probability is to be predicted as an output because the range of value produced by sigmoid function lies between 0 and 1. Since the function is differentiable, we can find the slopes between two points. The function is monotonous but, its derivative is not.The only drawback is that the logistic regression model can lead a neural network to get stuck during training. A more generalized logistic activation function, Softmax which is used for multi-class classification problem. Each of the input provided to the perceptron has a desired output associated with it. Some commonly used activation functions for the perceptron are the step and the bipolar step functions.

4. Tanh function [6] :

It is more like a logistic sigmoid function but better. Tanh or hyperbolic Tangent Activation function has a range from -1 to 1. It has the advantage of mapping negative values. The strongly negative values and the inputs which are 0, will be mapped near zero as we can from the graph. This function is differentiable, monotonic but its derivative is not.

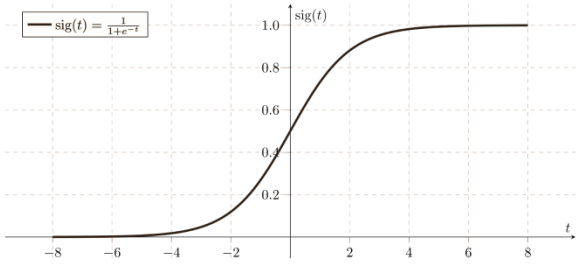

Fig. 6. Sigmoid Function (Source: http://www.stokastik.in/machine-learning-interview-questions-and-answers-part-i/).

5. ReLu function:

The most used activation function which has found its usage in almost all the Deep Learning and Convolutional Neural Network models. The ReLu function, as well as its derivative, is monotonic. The ReLu graph is half rectified, this means that the value of the function is 0 for the negative values. As the other argument is 0, the function value returns the value of z for all the values of z**Error! Reference source not found.**0. This issues a problem that limits the ability of the neural model to train or fit from the data properly as it returns all the negatives to 0, posing a threat for the model to train with the negative data and fitting as well corresponding to it.

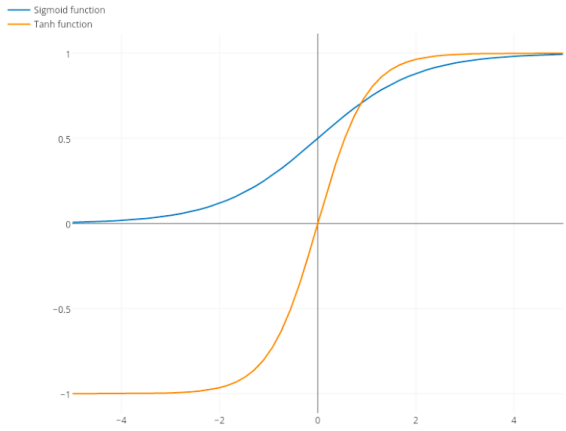

Fig. 7. Tanh (Source: https://stats.stackexchange.com/questions/330559/why-is-tanh-almost-always-better-than-sigmoid-as-an-activation-function)

D. Principle

Perceptron being a simple neural network architecture, its principle of operation is also quite simple. Each input is weighed separately by some value and then added up together to get the final output, with the goal to trace the output behaviour (using activation function) and at last compare with the desired output. The reason for comparing with the target output or the desired output is to find out how the proposed algorithm is working with the given datasets in terms of function approximation which is often considered in the case of supervised learning. As our weight approximation is done on the principle of supervised learning, it is important that only such activation function is in use, which can produce only two values as an output. For the step function, the values are 0 or 1, while 1 or -1 in case of the bipolar step function.

Parameter	Variable	Type
Inputs	X_i (i^{th} input)	Real or Binary (from the external environment)
Synaptic Weights	W_i (associated with x_i)	Real (initialized with random values)
Threshold(bias)	θ	Real (initialized with random values)
Output	y	Binary
Activation function	g(.)	Step or bipolar step function
Training Process	-	Supervised
Learning Rule	-	Hebb's Rule

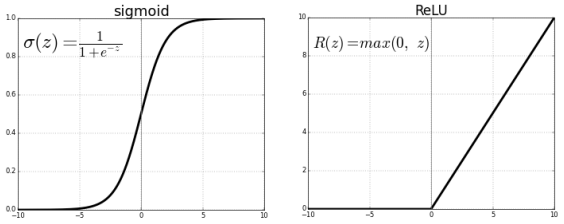

Fig. 8. ReLu(Source: https://cdn-images-1.medium.com/max/1000/1*XxxiA0jJvPrHEJHD4z893g.png)

E. Mathematical Intuition

Upon a closer look at the function and the similarities between the different forms of outputs produced (depending upon the activation function used), we can concur that perceptron works as a linear discriminator i.e it can be effectively used for linearly separable values.This can be proved upon analysis of its functioning.
For a perceptron using bipolar step function as its activation function, the result can be represented as

$$y=\begin{cases} 1 \text{ if } \Sigma w_i * x_i - \theta \geq 0 \\ -1 \text{ if } \Sigma w_i * x_i - \theta < 0 \end{cases}$$

Now consider this perceptron having only two input values. From this, we obtain two inequalities

$$w_1 * x_1 + w_2 * x_2 - \theta \geq 0$$

$$w_1 * x_1 + w_2 * x_2 - \theta < 0$$

A classification boundary is found upon the basis of these inequalities in the form of the straight line(if three inputs were to be considered we would obtain a plane):

$$w_1 * x_1 + w_2 * x_2 - \theta = 0$$

Thus, in conclusion, the perceptron can be called as a pattern classifier. It is important to notice that this application of perceptron is only applicable for linearly separable values. (as the degree itself is linear).

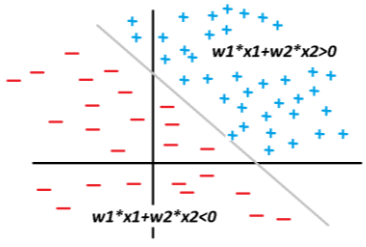

Fig. 9. Perceptron decision boundary.

F. Training

As per seen the working principle of the perceptron, weights play a crucial role in the functioning. The weighted inputs which are used to train the perceptron are optimized in order to get the desired output or to match itself with the targeted output. The adjustments or the optimization of the weights and the threshold values of the Perceptron for the classification problem is performed in the following way, if the value of the calculated output matches with the value of the desired output, the weights considered for the input and threshold remains the same [7]. Otherwise, if the value differs from the value of the desired output, the respective weights and the threshold values are changed accordingly to its input data, in order to match up with the desired output. This condition is often termed as an excitatory condition [8]. This process continues iteratively until the desired result is obtained for the given all the inputs being fed to the perceptron. We can also refer to the algorithm as being "error driven" because as discussed earlier, it won't bother to update the weights as long as the desired output is obtained. In mathematical form, the equation for adjusting the weights is given as:

$$w_i^{current} = w_i^{previous} + (d^{(k)} - y) * x_i^{(k)}$$

$$\theta_i^{current} = \theta_i^{previous} + (d^{(k)} - y) * (-1)$$

$d^{(k)}$ is the desired result for the k^{th} training example
$w_i^{(k)}$ is the i^{th} weight of the k^{th} training example
Applying the vector approach, we can fit the threshold value inside the weight vector as the same principle is applied to both the bias and the weights.
Thus, the equation reached in vector form is:

$$w^{current} = w^{previous} + \eta * (d^{(k)} - y) * x^{(k)}$$

where

$$w = [\theta\ w_1\ w_2\ \dots\ w_m]^T$$

$$x^{(k)} = [-1\ x_1\ x_2 \dots\ x_m]^T$$

Error! Reference source not found. It is the learning rate of perceptron which controls how quickly the training process stabilizes i.e how the classification boundary adjusts itself between the separable classes. The learning rate has a profound impact on the perceptron and should be chosen carefully to avoid instabilities, the value is usually within the range $0 < \eta < 1$.
Thus, the perceptron algorithm can be explained as follow: [9]

1. Training samples are obtained $\{x^{(k)}\}$
2. Obtain desired output $d^{(k)}$ for each respective sample
3. Initialize weight vector with random values
4. Choose an appropriate learning rate
5. Repeat the steps

    5.1. Initialize error to zero
    5.2. For all the training samples $\{x^{(k)}, d^{(k)}\}$, do

        5.2.1. Calculate $w^T \cdot x^{(k)}$
        5.2.2. Pass the calculated value to the activation function
        5.2.3. If $y \neq d^{(k)}$

            5.2.3.1. $w^{previous} = w^{current} + \eta * (d^{(k)} - y) * x^{(k)}$
            5.2.3.2. The error is provided the appropriate value.

6. Repeat until the error is zero or acceptable minimum

It is also suitable to include a variable "iteration" for counting the number of times the input (the entire dataset) is worked with to adjust the weight vector "w", ideally until the error calculated becomes zero. A trained perceptron can then, in turn, classify any new input provided to it.

It is also important to understand the graphical analysis of our training process.

- With each successive update to the weight vector, the classification boundary nudges itself in the appropriate direction to adjust between the classes to be differentiated.
- The solid line representing the acceptable classification boundary is achieved through adjustment from each previous boundary(in dotted line) after successive increment.
- The learning rate impacts the "jump" distance between two adjacent boundaries, thus if the classes are narrowly separated, a small value of the learning rate is required for reaching the appropriate decision boundary.

There can be more than one unique classification boundary for a set of classes, this, in turn, results in achieving different values of "iteration".

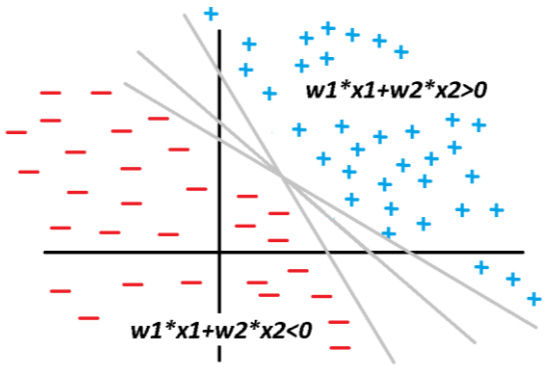

FIG. 10. MULTIPLE DECISION BOUNDARIES POSSIBLE

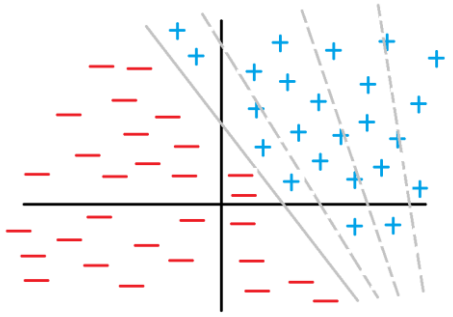

FIG. 11. ADJUSTMENT OF DECISION BOUNDARY

G. Classification Of Test

The following represents the classifying algorithm for the perceptron

1. The sample to be classified {x} is obtained
2. The vector w used in training is obtained
3. $y = g(w^T * x)$
   - 3.1. If y is -1
     - 3.1.1. Then x belongs to the class falling below the classification boundary.
   - 3.2. If y is 1
     - 3.2.1. Then, x belongs to the class falling above the classification boundary.

This illustration depicts the Perceptron Learning Algorithm. The algorithm learns a linear discriminant function, **Error! Reference source not found.** from the training data drawn. [10] From the given training data, it learns specifically W and b. To introduce the learning algorithm, it is considered to have augmented pattern, of the pattern "X" that we considered in the above discriminant function.

$$X_a = (1, x_1, x_2, \ldots, x_l)$$

and the augmented weight vector,

$$W_a = (b, w_1, w_2, \ldots, w_l)$$

Now, we have already seen that $g(X) = W_a{}^T * X_a$. We have assumed that

y = -1 if $X \in C_-$ &

y = +1 if $X \in C_+$

If $X \in C_-$, then $g(X) = W_a{}^T * X_a < 0$.

Hence, $g(yX) = W_a{}^T * X_a > 0$

as y = -1 or

If $X \in C_+$,

Then $g(X) = W_a{}^T * X_a > 0$.

Hence, $g(yX) = W_a{}^T * X_a > 0$ as y = +1.

Therefore, from the above portion, we can surely conclude that **Error! Reference source not found.** whether $X \in C_-$ or $X \in C_+$. This simplifies the learning algorithm for perceptron.

H. Multilayer Perceptron

It is often considered that the perceptrons are generally used to classify linearly separable values which can just be separated by a straight line. But, when the classification is not linearly separable and if such problems are being implemented onto the perceptron, it will iterate infinitely without any prompt result. That's when multilayer perceptron comes in handy. A multilayer perceptron network includes the backpropagation, a procedure to repeatedly adjust the weighted and the threshold values accordingly to minimize the difference between the desire/targeted output and the obtained output [11]. It is also associated with hidden layers or neural nodes which are stacked between the inputs and the output. The hidden layers often help to learn more complicated features (as XOR logic) [12].

The number of input data that are being fed into the network is known as the feature. So, if we take $x_1 x_2 \cdots x_n$ i.e. we are having 'n' features. A feature is a variable whose influence is felt in an output. These features are then multiplied with their weights to perform the summation of all the weighted features which gives us the dot product.

$$\sum_{l=1}^{m} w_l x_l = w_1 x_1 + w_2 x_2 \ldots . w_m x_m$$

The obtained dot product is then added to the bias in order to get **Error! Reference source not found.** (in this case) which is then fed onto the activation function to get the output for the first hidden layer h_1's first neuron h_1^1.

$$Z = \sum_{l=1}^{m} w_l x_l + bias$$

Similarly, the procedure is carried out for the hidden layer h1's second and third neuron, h_2^1 and h_3^1 respectively. The output for the neurons of the first hidden layer is taken as the inputs for the second hidden layer (if present) else for the output.

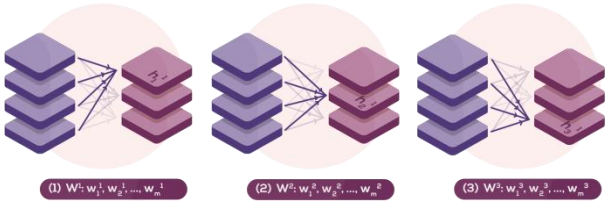

FIG. 12. MULTIPLE LAYERS OF THE MODEL

After feeding the output obtained by the summarization of the weighted input with the bias into the activation function to get the output for the output layer as shown below.

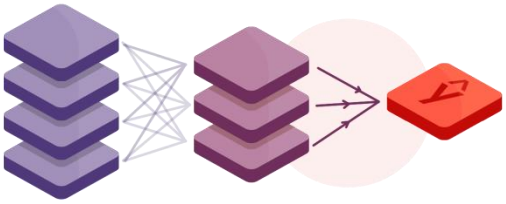

Fig. 13. Feed forward network of multi-layer perceptron.

III. CONCLUSION

In this work, the concept of a simple neural network called perceptron, its components, training algorithm, and prediction analysis has been discussed. Being the basic neural network, there is still a lot of ongoing research and investigation, on the basic intuition of perceptron to obtain countless opportunities. It has already opened many doors towards invention and innovation-generating endless possibilities for the future of Neural Networks and Artificial Intelligence to come.

REFERENCES

[1] *perceptron* . Retrieved from "https://en.wikipedia.org/wiki/Perceptron"

[2] Bishop, Christopher M. (2006). Pattern Recognition and Machine Learning. Springer.

[3] Freund, Y.; Schapire, R. E. (1999). *"Large margin classification using the perceptron algorithm"* (PDF). Machine Learning. 37 (3): 277–296. doi:10.1023/A:1007662407062.

[4] D. Shiffman, S. Fry and Z. Marsh, *The nature of code*, 1st ed. The Nature of Code, 2012, pp. 444-446.

[5] Sinha, U. (2016) *All about biological neurons,* Available at: *http://www.aishack.in/tutorials/biological-neurons/* (Accessed: 28th September 2018).

[6] Kang Nahua, "Multi-Layer Neural Networks with Sigmoid Function—"Deep Learning for Rookies (2)", www.towardsdatascience.com, Jun 27, 2017, Web, https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

[7] Brownlee, J. (2018) *How To Implement The Perceptron Algorithm From Scratch In Python,* Available at: *https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/* (Accessed: 28th September 2018).

[8] Frohlich, F. (2017) *Excitation and Inhibition: The Yin and Yang of the Brain,* Available at: *https://knowingneurons.com/2017/01/25/excitation-inhibition/* (Accessed: 28th September 2018).

[9] A. Silva, I.N., Hernane Spatti, D., Andrade Flauzino, R., Liboni, L.H.B., dos Reis Alves, S.F. *Artificial neural network : a practical course,* 1st ed.Springer International Publishing,2017,

[10] Minsky, M.l., Papert, S.:Perceptrons: *An Introduction to Computational Geometry*, MIT Press(1988)

[11] Kang Nahua, "Multi-Layer Neural Networks with Sigmoid Function—"Deep Learning for Rookies (2)", www.towardsdatascience.com, Jun 27, 2017, Web, https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

[12] Y. Zhao,B. Deng,Z. Wang (2002), "*Analysis and study of perceptron to solve XOR problem"* in The Second International Workshop on Autonomous Decentralized System, Nov. 2002.