## 

## 

## 

## 

## **Artificial Intelligence**

**Assignment 2**

## **Separation of Non-Linearly Separable Data using Evolutionary Algorithm**

**NIM-BSCS-2020-04**

**Syed Muhammad Muneel Raza Naqvi**

**NIM-BSCS-2020-03**

**Shamsa Batool**

**Problem Understanding:**

The problem being tackled in this assignment involves separation of a data set which is linearly non-separable. The training data set consists of some points on the 2-D plane for which there is an output value, given in the output set of training data. To solve this problem, we use the ML approach discussed in the class. As we studied in the class, to separate a linearly separable data, we use a perceptron which is actually a single neuron. This neuron divides the input space into two sections. This division is the decision boundary which converts input space into output space. Due to the linearly separable nature of the data, just one line is enough here. However, when we have to separate a data set which is linearly non-separable, one line is not enough to be the decision boundary. Here, we involve more neurons used to implement a complete neural network consisting of different layers. So, in our case we have to use a neural network for the separation of linearly non–separable data.

**Proposed Solution:**

**Multi-layer Perceptron Introduction:**

The solution to the above discussed problem proposed by this assignment involves use of the perceptron model to implement a neural network with input, hidden and output layers. Such a model is called multi-layer perceptron. In this neural network, we have an input layer which consists of input neurons having input values as their parameters. Next we have hidden layers which may vary depending upon the complexity of the problem. These layers take input from the previous layer computations, perform their own computations and pass the results to the next layer. In the last we have an output layer which takes input from the last hidden layer, applies activation on its computations and gives output. Throughout the neural network, each neuron takes inputs and weights.

**What are Weights and Bias?**

In neural networks, neurons are working on some inputs, performing computations and giving results. Each layer’s neuron’s output becomes inputs for the next layer's neurons. In this way, we reach the final output. It shows that inputs have some impact on the output. This impact is represented and controlled by some data values called weights. With each input to the neuron, we pass a weight vector which is then used in computations of the neuron. For each input variable, there is a weight value in the weight vector which decides the impact of the variable on output.

As from the equation of line’s concepts, weights of input variables (which are x and y) coming from the training data set control the rotation of the boundary, a third weight which is called bias controls the translation of the boundary. A bias is added with the weights and its impact is used to determine the boundary with correct translation.

**Role of Weights in Multilayer Perceptron:**

What a multilayer perceptron does is separation which depends on the weights provided at different layers. As the separation boundary is the output of the network, the weights contribute towards its development. To get a good decision boundary, we have to give a set of weights which efficiently places the cut in the input space so that it is transformed into output space where data is being separated. To train a data set on a network, what we have to do is to get that set of weights. For this purpose, we try different weight vectors, gradually change the goodness of weights and at last get the desired set of weights. Hence, our whole neural network evolves around this gradual change.

**Use of Evolutionary Algorithm:**

As discussed in the above section, our task is to improve the weights. Here comes the theory of learning. In order to get the best weights for our network, we have to make our network learn the weights so that it can separate the two classes of data represented in the graph below.

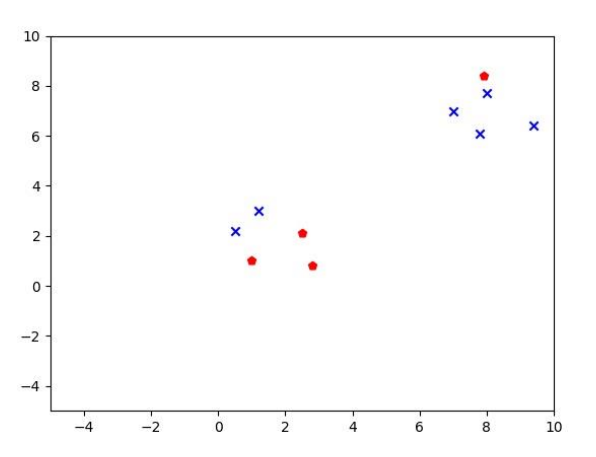


Image 1: Data plotting

There are different techniques used for this purpose. Nevertheless, the method we adopted in this assignment is the use of Evolutionary Algorithm for learning of weights. As an evolutionary algorithm, there is a population which gradually converges towards best fitness, it shows us a way towards our problem. In this way, we simply generate a population of weights, apply them on the network layers, calculate the fitnesses and generate another generation of weights based on the fitnesses. In this way, we move towards improved weights and finally get a set of weights which determine our decision boundary.

**What is Proposed Architecture and Why is it so?**

The architecture setup of this neural network consists of three layers including input layer, hidden layer and output layer. The input layer simply provides the input to the hidden layer neurons. This input comes from the training data set. Further, the hidden layer consists of two neurons which take weights and inputs from the input layer and after computations, pass the results to the output layer as input. Further, the output layer performs its computations and gives a result. The architectural setup of the neural network is presented in the figure.

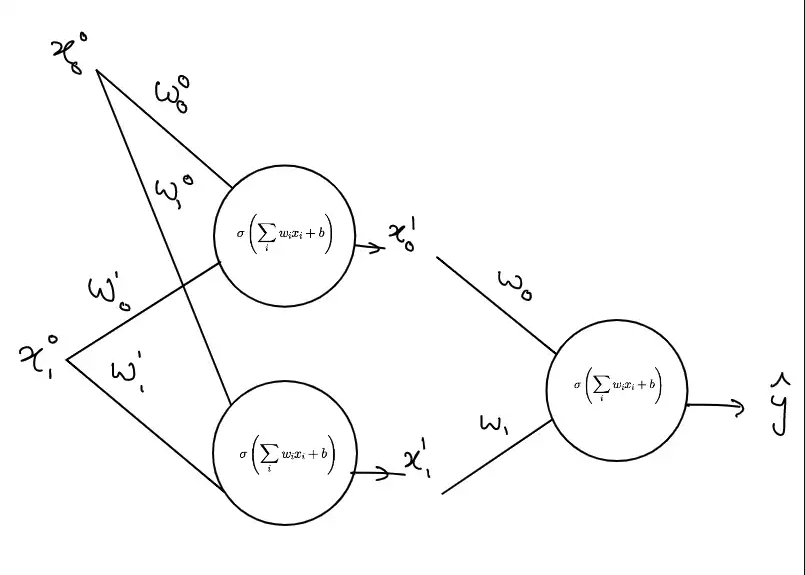


Image 2: Neural Network Architecture

**Why Hidden Layer with Two Neurons?**

Since our data is linearly non-separable, a single neuron is not enough. Therefore, we introduce a hidden layer. Moreover, the number of neurons being used in the hidden layer is two. It is because, as we can see from the 2-D representation of the training data set, the data can be separated by a minimum of two cuts in the input space. Therefore, the hidden layer uses two neurons which individually place a cut in the input space and give an output space. These output spaces of the hidden layer neurons are fed into the output layer.here the processing of output layer places a cut in a different input space than the original one and finally we get our output space. With this architectural setup, we are able to get our decision boundary with a minimum number of neurons in the hidden layer.

**What Operations A Neuron Performs and Why?**

In our network, a neuron performs two operations in its computations. First, it simply performs a calculation of weighted sum which is the dot product of inputs and weights. Second is application of an activation function on the result of dot product so that the output of the neuron can become the input of the next layer neuron in a specific range.

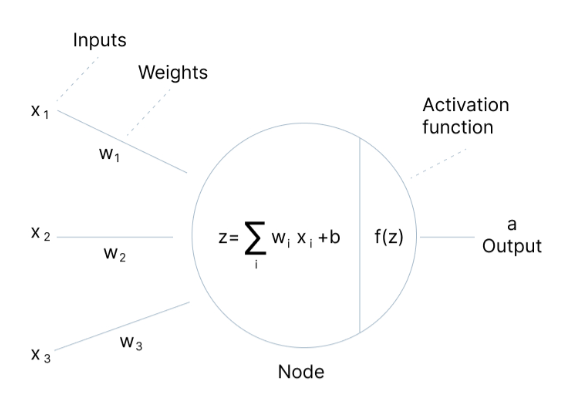


Image 3: Visualizing a Neuron

**What Activation Functions are used?**

In neural networks, different types of activation functions are used depending upon the output desired. In our case, we use the following two activation functions.

1. Tanh Function (ex - e-x / ex + e-x)
2. Step Function

As, from the hidden layer, we are going for a linear combination of outputs of the hidden layer neurons, we are shifting our original input space to another input space with some other dimensions added. We are moving from linearity towards non-linearity. Activation function converts the output of the weighted sum of hidden layer neurons to the input of the output layer on which the output layer can perform computations. Since our decision boundary in the final output space cannot be linear, we use the Tanh function as activation function for the hidden layer. Moreover, the activation function for the output layer is simply a step function since we just have to check the output and compare with threshold to see if the output matches with the train output.

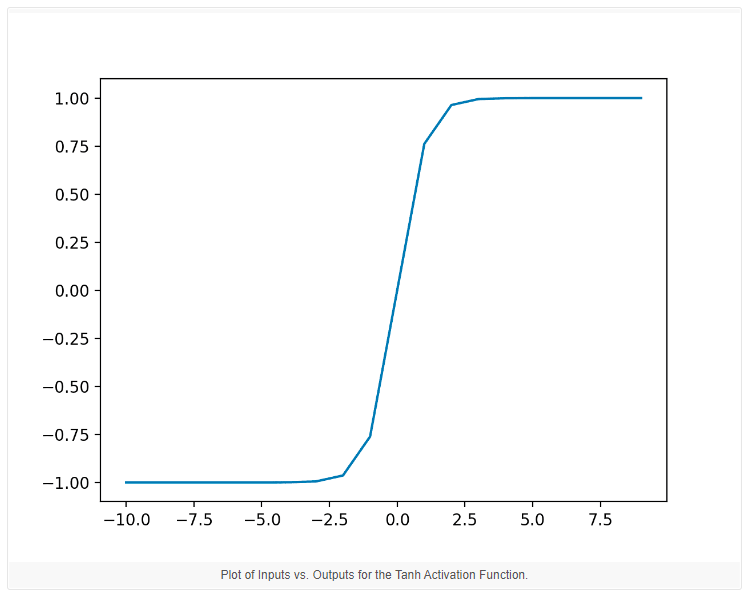


Image 4: Visualizing Tanh Activation Function

**Neural Networks and Evolutionary Algorithms, Combined Implementation**

1. **Population Generation**

This function simply initialises our first population of weights. This initialization is random. Moreover, one individual of this population is represented by a 2-D list containing 3 lists, one weight vector for each neuron.

1. **Fitness Evaluation**

This function combines the neural network and evolutionary algorithms as it computes, for instance of weight, how many values of training output get matched with that of computed output of the network. Before this computation, it performs the calculations of hidden layer and output layer neurons. In this way, the fitness figure for the whole population of weights is calculated. It returns a dictionary containing fiitnesses against weights.

1. **Natural Selection**

This function sorts the fitnesses in descending order and returns a sorted dictionary.

1. **Next Generation**

This function is responsible for generating an evolved generation of weights through crossover. Here, we take two individuals sorted on weights and pass them to crossover which returns two children. In this way, we traverse the whole list of sorted individuals and get a whole population.

1. **Crossover**

This function merges the weights of two parents and after some random change, gives weights combination for children. This crossover is actually producing the evolved generation.

**Hypothesis:**

1. Equation of Neurons in Hidden Layer

x1 w1 + x2 w2 + w3

1. Population Size = 1000
2. Generations = 500

**Results:**

On the above mentioned hypothesis, the network gave a set of weights with 100% accuracy. This is the required set of weights which our network learns through evolution. After getting this weight set, we simply store the weights and then pass it to the model. These are the weights upon which the model is trained.