Evolutionary Algorithms for Neural Networks Learning

Artificial Intelligence

Submitted To:
DR. JUNAID AKHTAR

Table of Figures

Abstract

Artificial intelligence is powerful branch of computer science. And nowadays participating a lot in problem solving and machine learning. In this project, we are solving classification problem with the help evolutionary algorithm and neural networks. We were given a non-linearly separable data set. And neural networks are very good at the classification of data points into different regions. We are using evolutionary algorithm for the production of new weights from old ones by doing crossover and mutation. Our model will stop learning when it will be succeeded in its learning otherwise it will do crossover and mutation on individuals and follow the process to achieve more and more accuracy in classifying the all data points.

1. Introduction

In the present era, artificial intelligence is major branch of Computer Science. In artificial intelligence, scientist make intelligent machines and these machine can take intelligent decisions. These machine are provided the environment and after perceiving it, takes specific actions that helps to achieve our goals. In this project, we have to classify the given datasets using Evolutionary Algorithm (EA). We were using artificial neural networks (ANN) for this purpose. We were given a dataset of inputs by our instructor. Our object is to classify the data with 100% accuracy.

Evolutionary algorithms (EAs) are very powerful technique to solve the problem like optimization, maximization and many more. Evolutionary algorithm is a subset of evolutionary computation. An evolutionary algorithm uses a mechanism which is inspired by biological evolution like selection, reproduction, mutation, and crossover. Evolutionary algorithms are typically used to provide optimal and approximate solutions to problems which can't be solved easily with simple techniques. Optimization problem one of the examples. It may be too computationally intense to find an exact solution but sometimes an approximate and optimal solution is sufficient. These algorithms work without making any assumption about the underlying fitness landscape.

Artificial neural network (ANN) is the component of artificial intelligence which is meant to simulate the working of a human brain. ANN is the piece of computing system design to simulate the way the human brain analyzes and processes the information. ANN solves problems that could not be solved by the human brain or any other statistical standards. In neural networks each neuron takes inputs and after some processing on information, it gives output.
ANNs have capabilities of self-learning that enable them to produce better results as more data become available. There are two types of data:

1. Linearly separable data
2. Non-linearly separable data

Linearly separable data is data that can be separated by a single line if we draw the graph of data. For example, if we have two types of movies (good and bad movies), graph for the movie is made in two dimensions (Acting (A) and Direction (D)). Red data points in the graph are bad movies and blue data points are good movies. All good and bad movie can be separated by a straight line without lose any data point.

Figure 1: Linearly Separable Data

Non-linear separable data is data that cannot be separated with a single line 100%. If we have non-linearly separable data, and we are classifying the data with a single line, there may be chances that we have to compromise on some data points. But if we use more than one line, we can classify with 100% accuracy. In cases where data is not linearly separable, kernel trick can be applied, where data is transformed using some nonlinear function so the resulting transformed points become linearly separable. A simple example is shown below where the objective is to classify red and blue points into different classes. It's not possible to use linear separator, however by transforming the variables, this becomes possible. In the picture given below, blue points are bounded by both lines (classifiers) and all other points are red points' place.

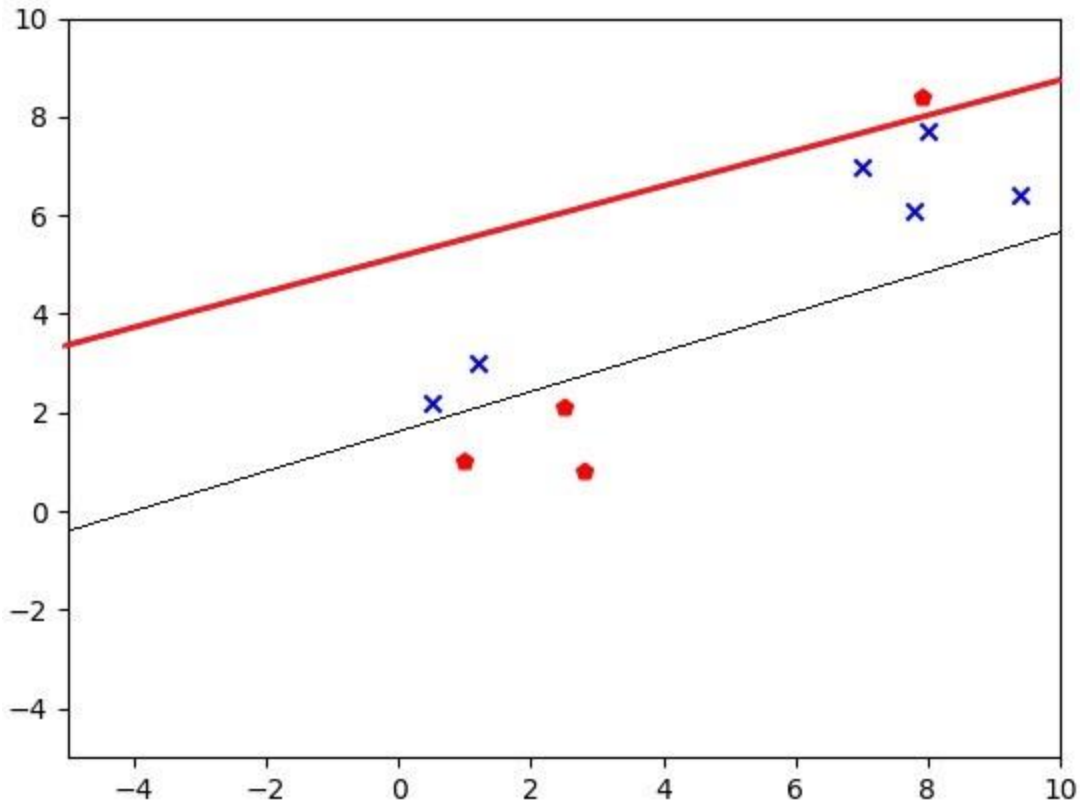

Figure 2: Non-linearly Separable Data

2. Literature Review

2.1 Activation Function

Activation function decides whether a neuron should be should or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Output= f(W.X)

Above is an abstract form of the activation function.
Here we will describe some activation functions that we reviewed

- Step function
- Linear Function
- Sigmoid Function
- Tanh Function

Now, let's discuss each function one by one.

2.1.1 Step Function

It is a threshold-based activation function. **In** this we have to set a certain threshold and checks dot product of weights and input vectors. If the value is greater than threshold then neuron will be activated otherwise not activated.

For example, in our case that we discussed in class session we set a threshold like this.

if W.X >=0 ----------> output =1

else W.X < 0 ----------->output =-1

From below figure you can see that how step function works.

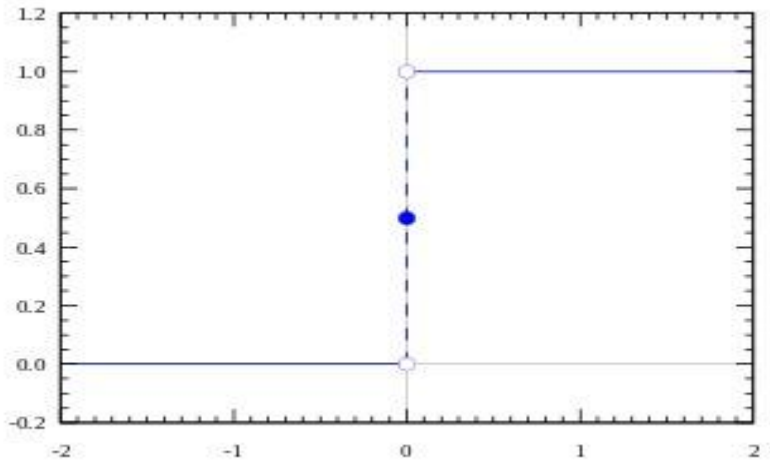

Figure 3: Step Function

Drawbacks:

It is good for binary classification but it can't work in case of multiclass classification. So, when we have to classify multiple classes, we can't use it.

2.1.2 Linear Function

It is a straight line activation function where output is proportional to W.X. Its output is a dot product of weights and inputs. It can also be used when multiple neurons connected. You can see below a linear function.

Output=mx

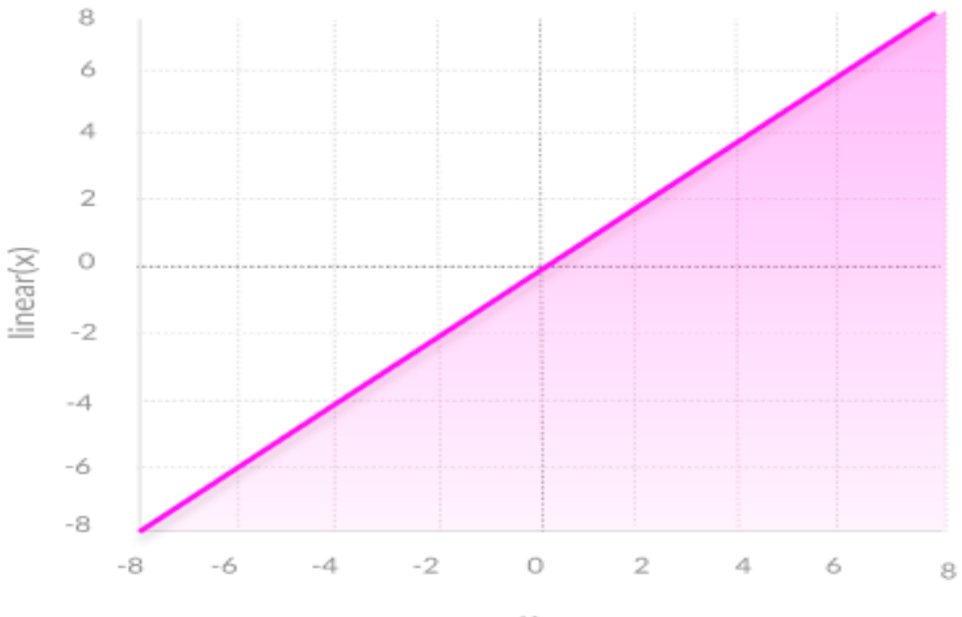

Figure 4: Linear Function

Drawbacks:

It doesn't achieve the main purpose of activation function as we expect. Main purpose of activation function causes non-linearity but it behaves linearly in nature. It is useless if we use many layers because it acts as single layer and gives us the result as of single layer. One more thing its derivative is constant so we are unable to perform gradient descent for training.

2.1.3 Sigmoid Function

It is first nonlinear activation function that we will discuss here. It seems like a step function but it performs better classification due to its smoothness. It can be easily used with nonlinearly separable data. It returns an output value be in range (0,1). It works well in binary classification as well as multiclass classification. One more thing its derivative can be easily calculated for gradient descent. You can see sigmoid function here and can observe how it works.

Function = $1/1+e^{-x}$

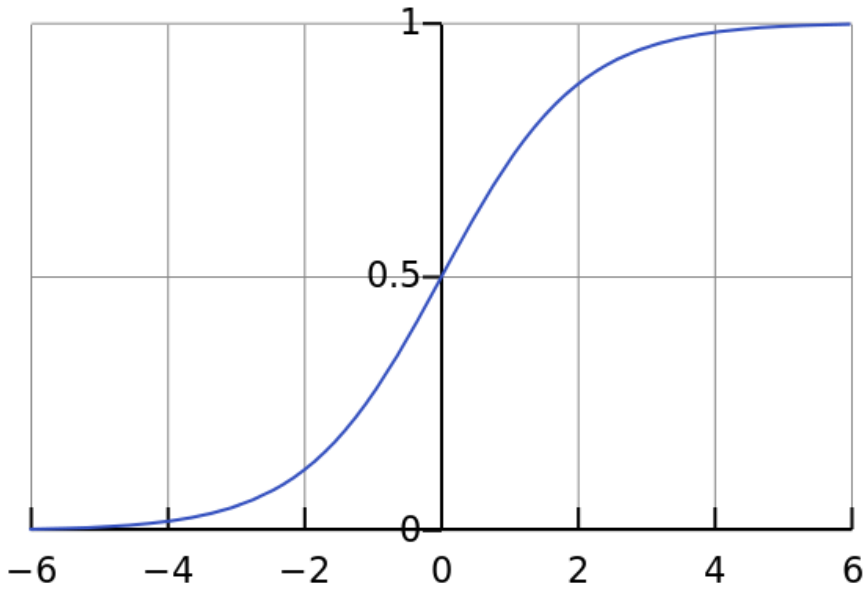

Figure 5: Sigmoid Function

2.1.4 Tanh Function

It is a mathematically shifted version of sigmoid function. It works always better than a sigmoid function. Sigmoid gives output in range (0, 1) but in case of Tanh, its values lie between (-1, 1). It is also non-linear in nature. Its value range (-1, 1) helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier. Here below you can see its mathematical form as well as observe how it acts as classifier.

Function = $2/1+e^{-2x}$ -1

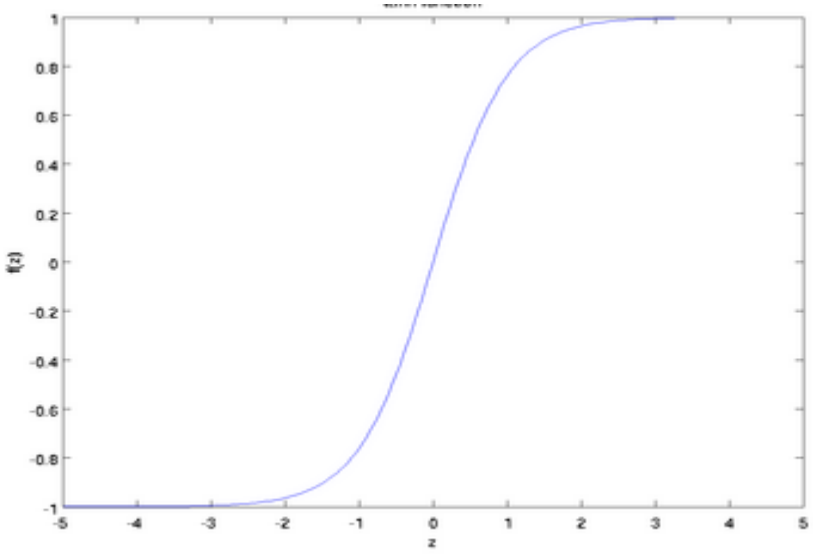

Figure 6: Tanh Function

3. Methodology

First of all, our task was to make model which would have to be followed during whole project.

3.1 Model

We made a model of neural network on playground which we were following during this whole project for the training of our data.

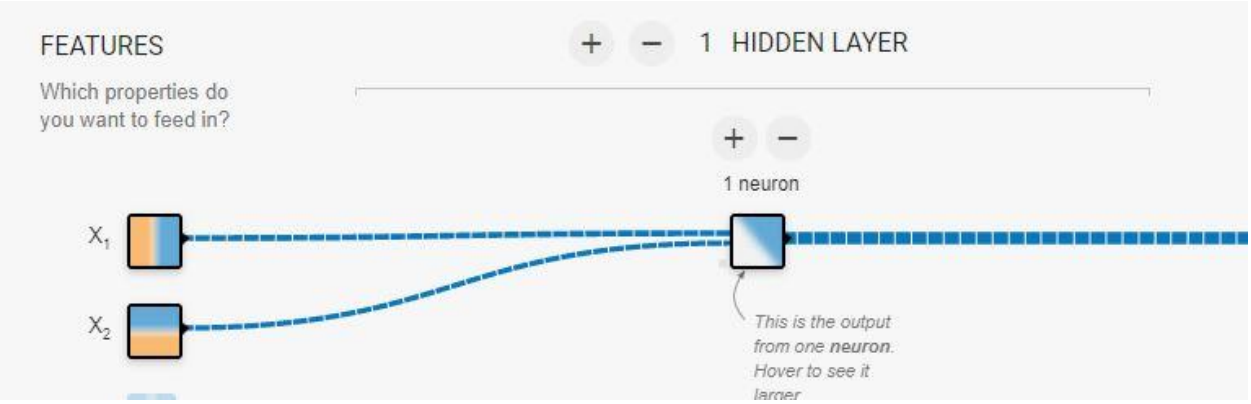

Figure 7: Model

3.2 Initialize Population

We firstly initialize the population of fifty and every individual has nine weights that are generated randomly.

```
def genRandomWeigths():
    for i in range(popSize):
        w = random.sample(range(-10, 10), 9)
        weightsList.append(w)

    return weightsList
```

Figure 8: Initialize Random Weights

3.3 Neurons Structure

In over code we are using two layers of neurons. First layer has two neurons while the second layers (hidden layer) has one neuron. Every neuron has two inputs and a bias having a value of one (1).

```
class LayerOne: #Input Nuerons Layer
    def N1(self, x1, x2, x3, w1, w2, w3):
        s = (x1*w1) + (x2*w2) + (x3*w3)
        return s

    def N2(self, x1, x2, x3, w1, w2, w3):
        s = (x1*w1) + (x2*w2) + (x3*w3)
        return s

#######################################################
#######################################################

class LayerTwo: #Final Neuron Layer
    def N1(self, x1, x2, x3, w1, w2, w3):
        s = (x1*w1) + (x2*w2) + (x3*w3)
        return -1 if s < 0 else 1
```

Figure 9: Layers of Neural Network

3.4 Activation Function

Activation function is mathematical equation that determines the output of a neural network. We are using linear function in layer one (input neurons) and in second layer (final neuron) we are using step function. In neural network, activation function is attached to each neuron in the network. This function determines whether it should be activated or not based on whether each neuron's input is relevant for the model's prediction.

```
def getNeuronOutput(i, w, L1, L2):
    n1Out = L1.N1(1, i[0], i[1], w[0], w[1], w[2]) # Input into Neuron-1
    n2Out = L1.N2(1, i[0], i[1], w[3], w[4], w[5]) # Input into Neuron-2

    # Now feed the output into final neuron

    finalOut = L2.N1(1, n1Out, n2Out, w[6], w[7], w[8])

    return finalOut
```

Figure 10: Getting Output

3.5 Crossover and Mutation

Here comes the part of evolutionary algorithm. In this section we produce the new weights by doing crossover and mutation in the old weights. In this stage we will crossover and mutate the data of every individual of our population if the result is not according to the desired output. If there is no error in the training, then our model will give 100% accuracy and learning will be stopped.

Like if the data retrieved is not equal to our desired output we will pass those values for crossover and mutation function. In crossover and mutation we will first convert them into binary and then change the last bit of the data and again convert them into decimal and then pass those values to the calling function. By following this technique in the crossover and mutation, we were succeeded in achieving 90% accuracy but our model was not giving more accuracy than 90%.

In the end we are storing the best weights of neural network in **maxiScores** and **Generations** and then plotting the graph between both of them. Graph is shown below, which shows the progress of our model which we implemented using evolutionary algorithm and neural networks. As this graph also shows that maximum accuracy that we reach in classifying the data is ninety percent.

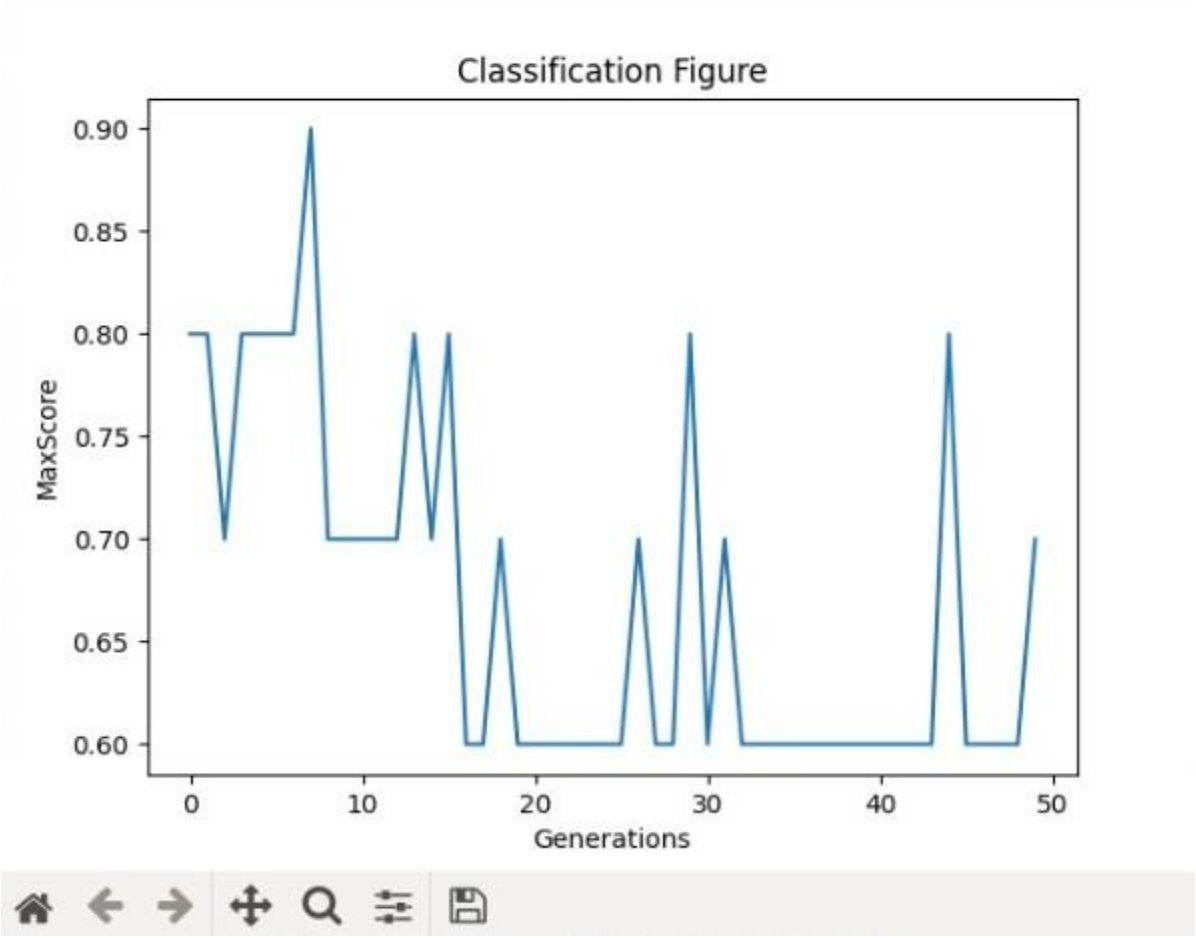

Figure 11: Highest Accuracy showing Graph

4. Result

We fixed the epochs by fifty (stopping criteria). We move generations by generations for having the collective accuracy of data more than or equal to 90 percent. If we achieve that limit accuracy of 90 percent, then the weights we have are optimized weights. Finally we were able to make the machine learn to achieve ninety percent accuracy. As we increased the number of generation of individuals which were randomly generated at initial point, the accuracy was getting much better as compared to low number of generations.

5. References

- https://playground.tensorflow.org/