CS 551 Introduction to AI

Homework 02

Arash Mehrabi (S027783)

1 Introduction

Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. [1]

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. [2]

In this homework, we used an genetic algorithm as an optimizer for a simple neural network. As it is shown in figure 1-1, the NN has two hidden layers and one output layer. In the first hidden layer, there are two hidden units. A weight matrix, called $\mathbf{W_1}$, of shape [2*3] is multiplied to a the input matrix, **X**, of shape [3*N] where N is the number of samples. No activation function is applied to these hidden units and no bias terms are added to the result of the matrix multiplication. Hence, the result **H**

$$H = W_1 * X$$

is resulted from a simple matrix multiplication of $\mathbf{W_1}$ and **X** and its shape is [2*N].

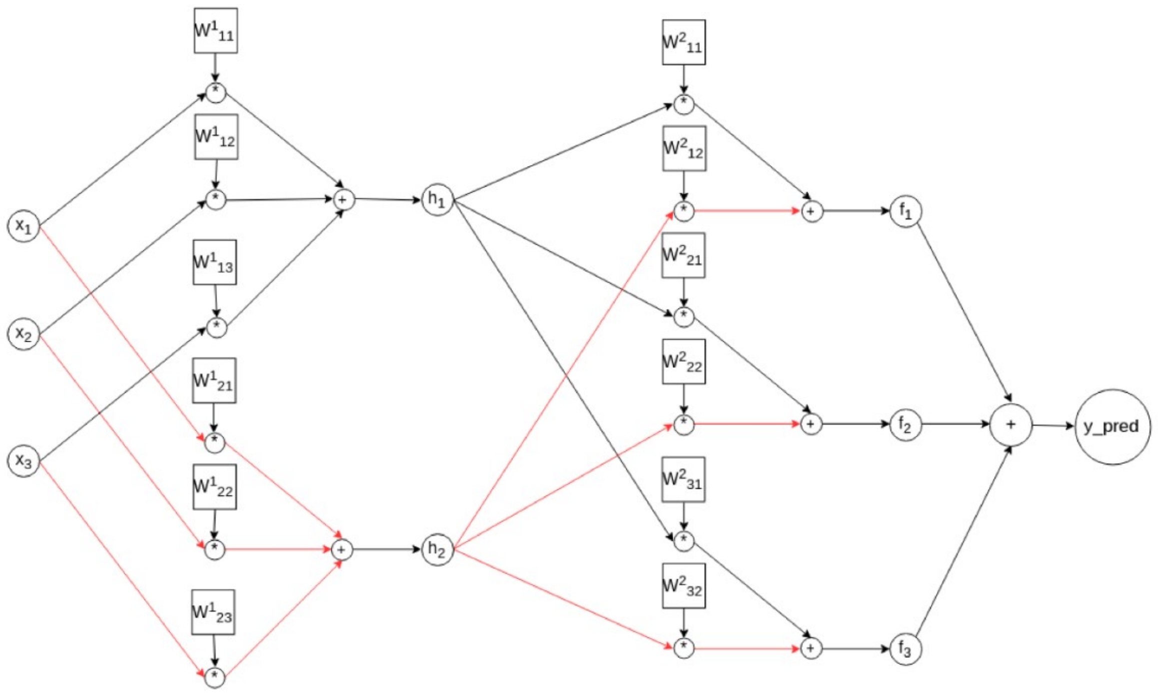

Figure 1-1 The architecture of the NN

In the second hidden layer, the result **F** is the product of the second weight matrix $\mathbf{W_2}$ of shape [3*2] and **H**. Therefore, **F**, must be of size [3*N]. In the output layer, the elements of each column of the **F** matrix are summed to each other. Hence, for an input matrix X of size [3*N], the prediction matrix is of size [1*N] where N is the number of the samples.

Our goal is to use GA to find the best values for the weight matrices that achieve the least minimum square error value. Minimum square error (MSE) is our measure for evaluation and the set of weights that achieve the least MSE value are the optimal values for weight matrices.

1.1 Implementation Details

The population size of our implementation was 28. Hence, at each generation, we had 28 different set of weight values. Also, we generated 2800 generations. The given input of the program should be a matrix of size [N * 3], as it was requested in the homework description; however, we transpose it and use it with shape [3 * N] inside our program because of matrix multiplication standards.

For chromosomes, which consist of 12 genes, I defined a class. Each object of the class, saves a list of weights, the MSE value of the chromosome, and when needed, it splits and reshapes the list of weights to W_1 and W_2. For finding the K-best individuals of each generations, I used the priority queue class of the previous homework.

2 Algorithm

As it is mentioned in the previous section, we used a genetic algorithm as an optimizer for our neural network. In this section, we are going to explain how we achieved the optimal values for weights of our NN.

2.1 Initial Population

At first, we randomly choose real values in range [-1, 1] for the 12 genes in the weight vector. We do this for the number of individuals we want to have in our population. For example, if we want to have 28 individuals (chromosomes) in our population, we should select 12 random real values 28 times.

2.2 Finding the Fitness

Then, we find the fitness of each individual in the current generation of population. I defined the fitness of each individual as one over the MSE value of that individual. ($^{1}\!/_{MSE}$.) This is reasonable since as much as the error value of an individual be higher, its fitness will be lower and vice versa.

It is worth noting that the MSE value is calculated as

$$MSE = \sum_{i}^{column} \sqrt{(y^i - y^i_{pred})^2}$$

meaning that for each sample in the input matrix, **X**, we calculate the difference between the true y value and NN’s prediction and we square the difference. We add all these squared differences and divide by the number of the samples; hence, the mean squared error value is derived.

As it was mentioned in the homework description, the true value for *y* is derived by averaging the values of each column of the **X**. If **X** be a matrix with size [3*N], *y* will be a vector of size [1*N].

2.3 Best Individuals

By experiment, it was decided that 0.05 of the population directly be added to the next generation. For example, if the population size be 28, it means that only 1 best individual (according to its fitness value) of current generation will be also present in the next generation. According to the experiments of the author, this is proved to increase performance of the algorithm.

2.4 Selecting Parents

By intuition, individuals with higher fitness values should have higher chance of reproduction. To implement the mentioned intuition, we randomly selected parents of the next generation; however, weighted each individual according to its fitness value. Our intention was to reproduce by couples, like most animals; therefore, we forbade reproduction of an individual with itself. However, an individual was able to reproduce with more than one individual.

2.5 Crossover

Crossover point was randomly selected by choosing a random integer in range [1, 11]. For example, if the crossover point was 5, for the first child, the first 5 genes will come from the first individual (let’s say male), and the last 7 individuals come from the second individual (let’s say female). For the second child, the first 5 genes will come from the female and the last 7 genes from the male.

2.6 Mutation

With chance of $^{1}/_{18}$ = 5.55%, each gene can be mutated which means that the gene value will be replaced by random value in range [-1, 1]. Mutation helps to achieve diversity in the population and avoids getting stuck in local optima.

2.7 Repetition

The new generation will be achieved by adding the offsprings and best individual of previous generation. We repeat the the processes of sections 2.2 to 2.6 as long as it is desired. At each iteration, we derive a new generation.

3 Results

In this chapter, we will show the results of the algorithm. At first, it is helpful to look at the MSE (Error) value of the best individual of each generation (figure 3-1).

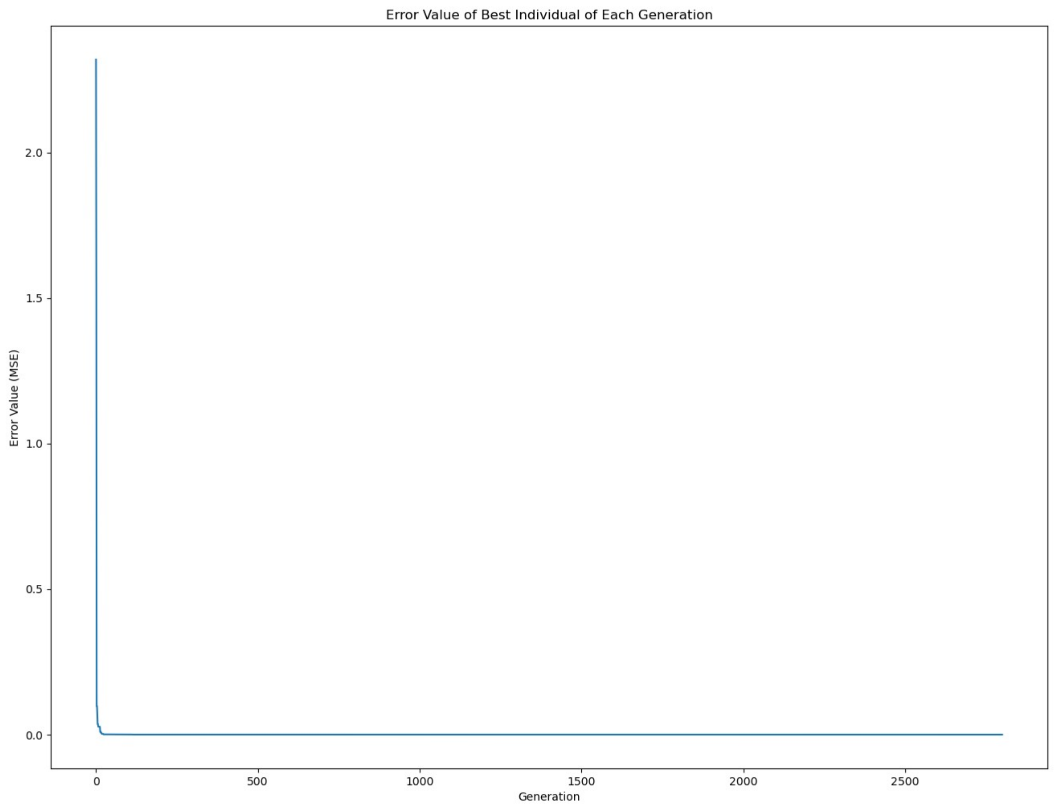

Figure 3-1 Error value of best individual of each generation

Since the MSE values fell down from large values like 2 in the first generations to small values like 1e-6 in middle generations, figure 3-1 is not able to cover this large range; hence, it is not very helpful. But, if we zoom to the first 14 generations (figure 3-2), we can see the decline of the MSE values from 2 to values near 0.

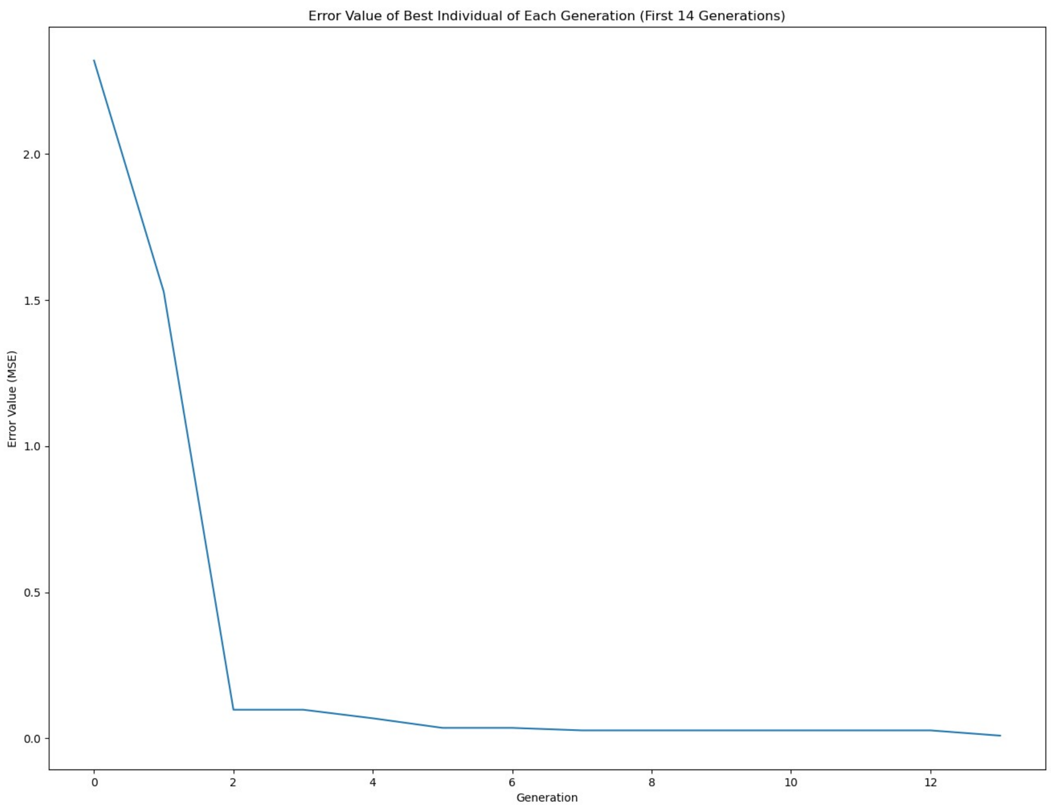

Figure 3-2 Error Value of Best Individual of Each Generation (First 14 Generations)

And if we look at the generations from 1000 to 1014 (figure 3-3), we can see that at middle generations the MSE values of best individuals are very small, e.g., around 2e-6.

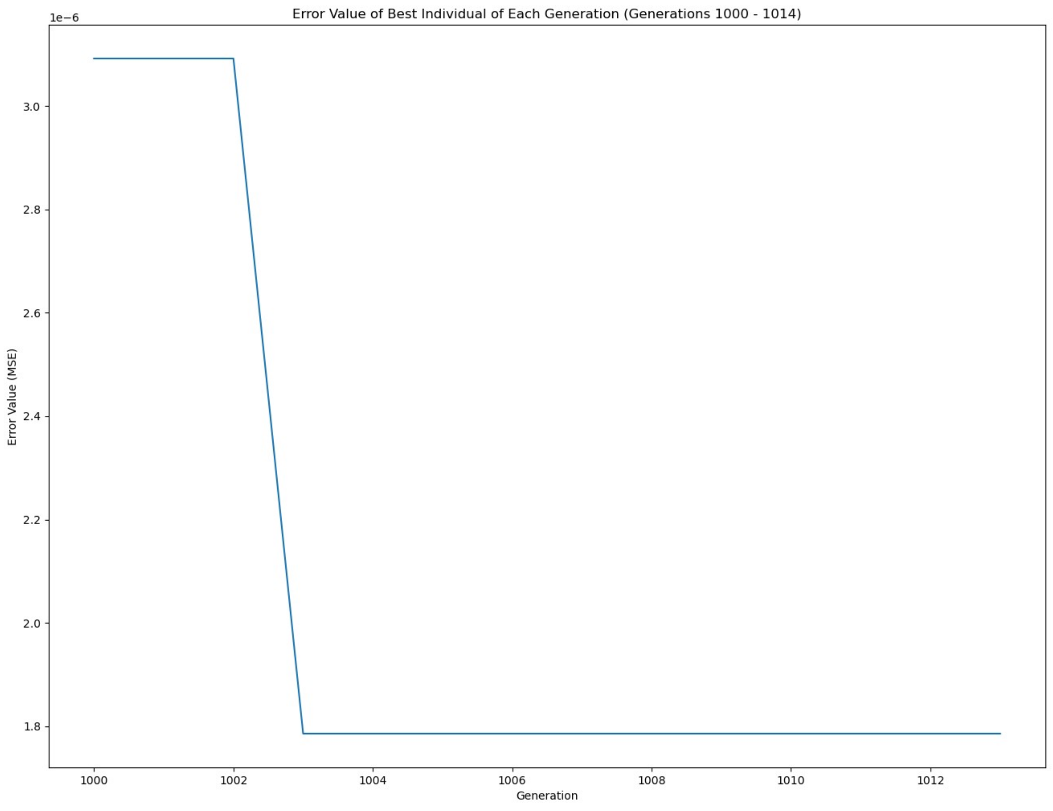

Figure 3-3 Error Value of Best Individual of Each Generation (Generations 1000 – 1014)

Furthermore, we can average the MSE values of all chromosomes in each generation. From this, we can see if our algorithm was capable of producing better generations overall throughout the time. As it is shown in figure 3-4, the mean MSE value oscillates around value 2.5 after a quick fell down in first generations. So, it is safe to say that our algorithm was able learn and produce generations that achieve better fitness values over time.

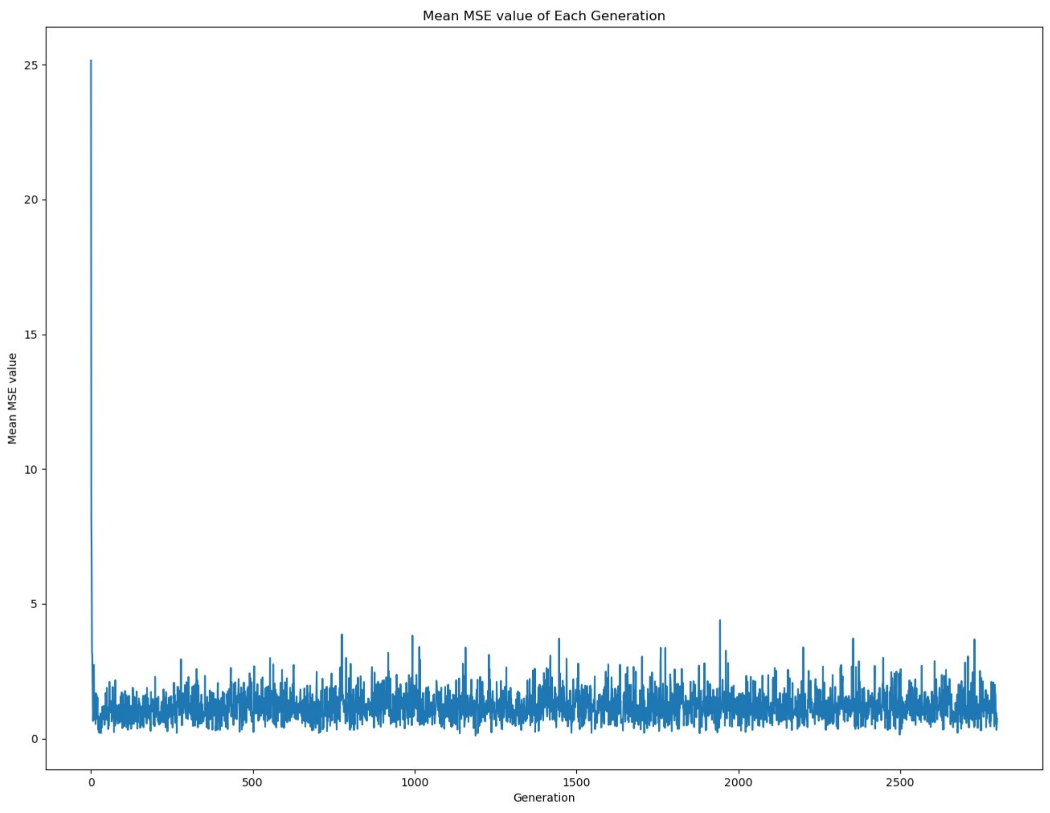

Figure 3-4 Mean MSE value of Each Generation

If we zoom into the first generations of figure 3-4, we can see that at the first generations the mean MSE value was around very high values like 25. However, it fells down quickly and after generation 10 it oscillates around value of 2.5 (figure 3-5). We can see that further, at middle generations, it oscillates around value of 1.5 (figure 3-6).

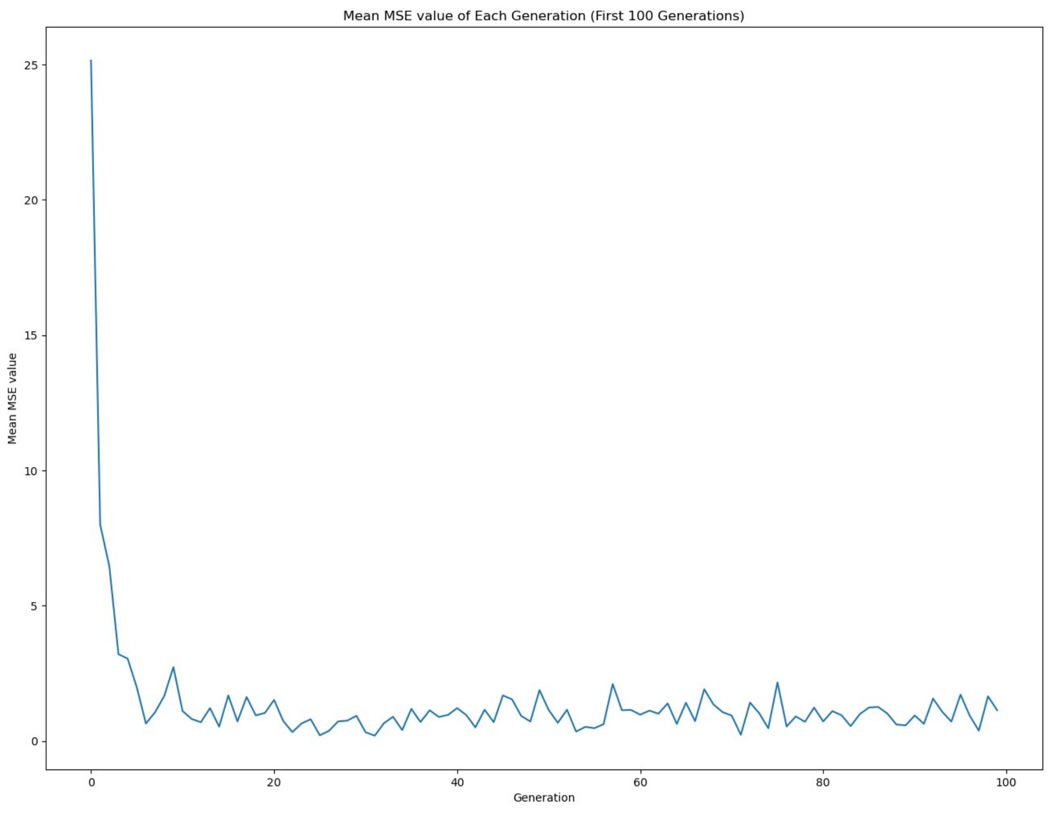

Figure 3-5 Mean MSE value of Each Generation (First 100 Generations)

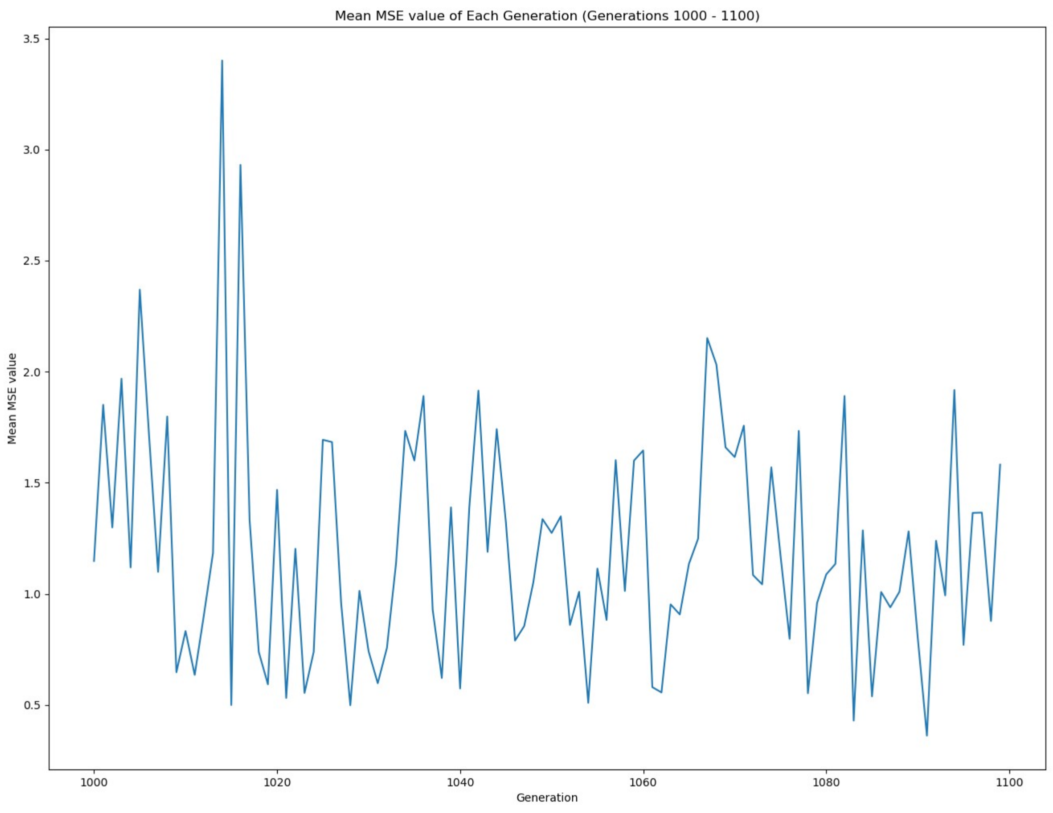

Figure 3-6 Mean MSE value of Each Generation (Generations 1000 – 1100)

At the end, we would like to show the error of each chromosome of each generation. As it is shown in figure 3-7, the majority of MSE values are below 60 and most of them are very close to 0. However, the figure is not very informative. If we plot this figure again; but, this time at each 400 generations, we can see that after first generations, the MSE values become very close to 0 (figure 3-8).

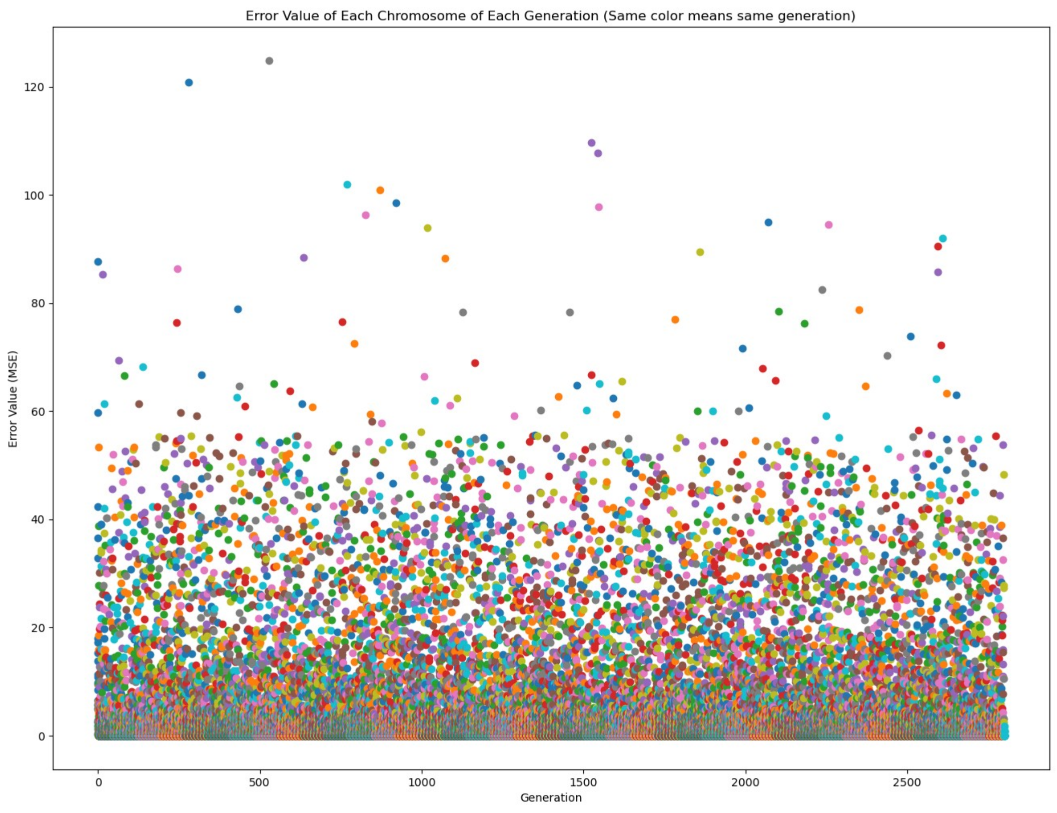

Figure 3-7 Error Value of Each Chromosome of Each Generation

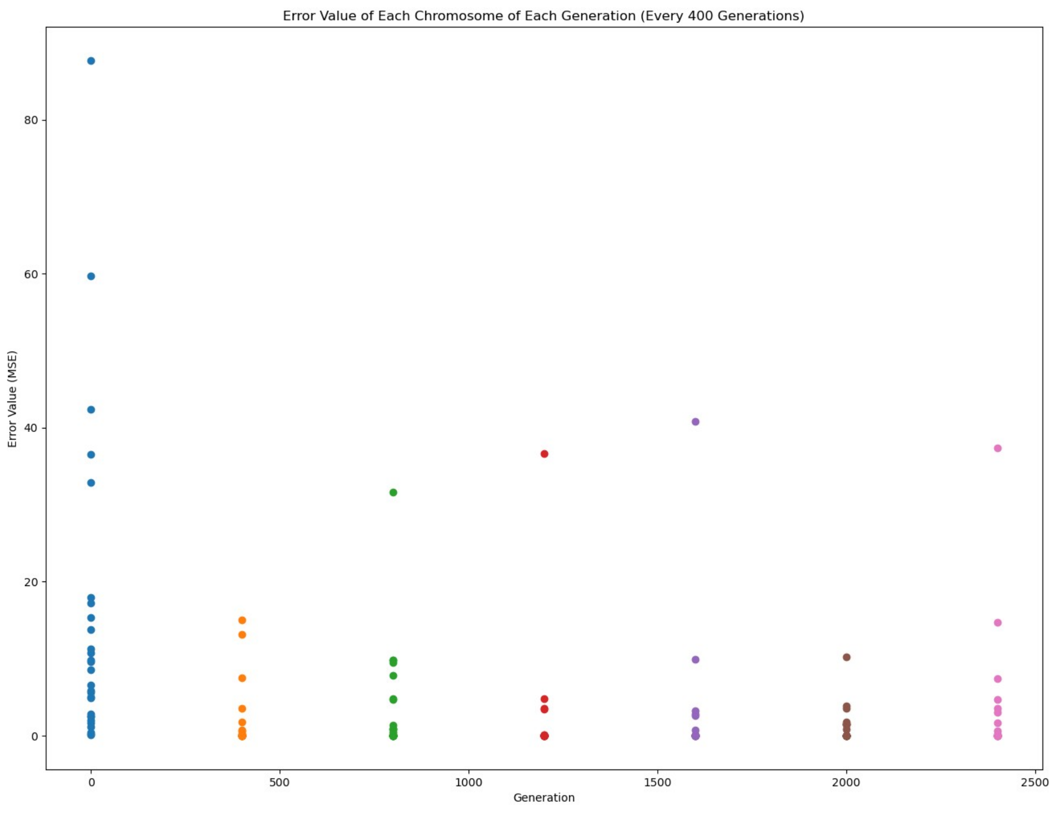

Figure 3-8 Error Value of Each Chromosome of Each Generation (Every 400 Generations)

4 Conclusion

From the results we can conclude that our GA was able to optimize the weights of NN. Surprisingly, it converged very fast and even at the 10th generation it showed a very good results. One reason might be that the range of the weight values, i.e., [-1, 1], is very small and if we increase this range, e.g., [-100, 100], it might take much longer for the GA to converge. Another reason might be that the architecture of the NN is very simple and the ground truth values are linear combination of **X** values (i.e., mean of **X**); hence, it is easy to learn suitable weight values.

5 References

1. Artificial Neural Network. (2022). Retrieved April 24, 2022, from https://en.wikipedia.org/wiki/Artificial_neural_network
2. Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press.