Genetic Algorithms Report

MDL PROJECT

Team No : 27

Team Name : Room543

Mihir Bani, 2019113003

Amul Agrawal, 2019101113

Introduction

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

It has few steps involved as listed below, and explained in detail throughout the report.

1. Defining population
2. Choosing a Fitness Function
3. Selection
4. Crossover
5. Mutation

Summary

In our code, we first declared an instance of the model, and then ran the required function for training. These steps are followed in the whole process, for each generation.

- Give it an initial vector, which generates population by random mutations on the same.
- Then with this population, API calls are made to get train and validation errors. The fitness of each vector is calculated and the population is sorted with decreasing order of fitness values.
- The selection process happens, where we select the parents based on fitness (like only the top 4, or etc).
- After selecting these parents, we find the crossover of these pairs of parents. Each pair of parents give us a pair of children. These children and the two best parents are selected for further process.

- The intermediate population generated is now mutated randomly, and thus the new population is generated. Now this new population will undergo the same process for many generations.

And we store the best vector across all these generations.

Now lets see the whole functioning of the project in detail:

Note: instead of making separate section for Heuristics, we have explained it together as we go through the following steps.

1. Defining population

The whole population is stored in 2D Numpy array. With number of rows equal to the Population_Size and the number of columns equal to the Gene_Length.

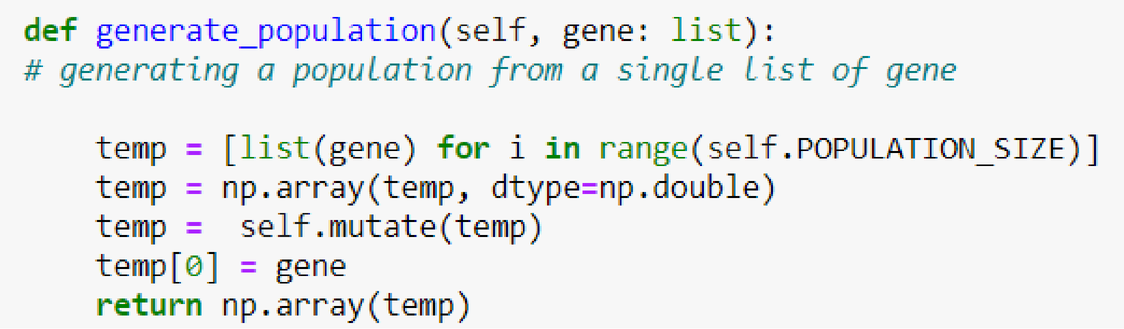
`POPULATION_SIZE = 8` . We also tried for 10, 12, 14.

`GENE_LENGTH = 11`

The population size defines the number of genes that we consider in our algorithm, and the gene length is the length of the vector that was given. Each gene consists of 11 chromosomes.

For the very first iteration, we started with the **overfit vector** given. We create a 2D array by simply copying the starting vector in the whole array, and then apply the mutation on it. For further a single code run, lets say for 20 generations, we carry forward the population. But when we run the code completely again for further training, we take the best vector from the previous batch and use it as the starting point.

So basically the whole population is initialized with the help of only one initial weights vector, that we give to call the function.



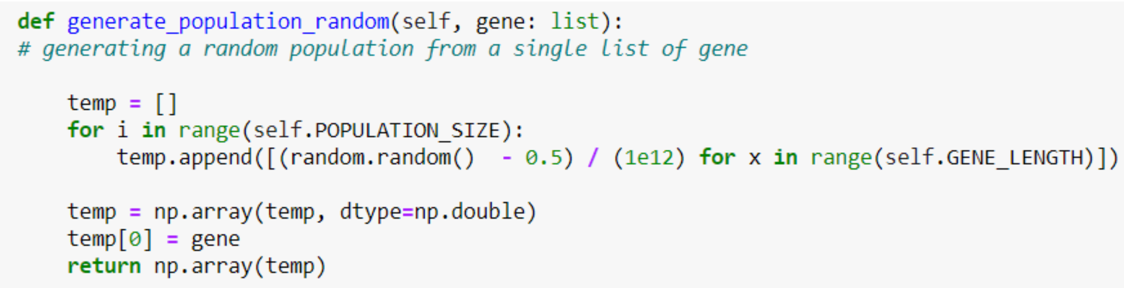
```python
def generate_population(self, gene: list):
# generating a population from a single list of gene

    temp = [list(gene) for i in range(self.POPULATION_SIZE)]
    temp = np.array(temp, dtype=np.double)
    temp =  self.mutate(temp)
    temp[0] = gene
    return np.array(temp)
```

Function to generate the whole population from a single gene. (we used this one)

The Mutation function used here is the same as the one which we use later in the GA after the crossover section. (covered later in this report).

Other approach:



```
def generate_population_random(self, gene: list):
# generating a random population from a single list of gene

    temp = []
    for i in range(self.POPULATION_SIZE):
        temp.append([(random.random()  - 0.5) / (1e12) for x in range(self.GENE_LENGTH)])

    temp = np.array(temp, dtype=np.double)
    temp[0] = gene
    return np.array(temp)
```

Function for generating a complete random population starting from zero vector. (We did not use this much, it was a failed attempt.)

For the very first iteration we also tried initializing the population with Zero vector, but the performance with this was much lower than starting out with the Overfit vector.

2. Choosing a Fitness Function

The Fitness function defines how to evaluate the quality of the gene at hand. We tried out various fitness function and changed between them to train the model, and also trying to go out of local minimas. It was mostly based on trial and error.

The fitness function was some function of `train_err` (training error) and `valid_err` (validation error), which were received from the server, for each gene (vector).

Some of the fitness functions that we used are:

- $-(5 \times valid\_err + 2 \times abs(train\_err - valid\_err))$
- $-(train\_err + valid\_err)$
- $-(valid\_err + 3 \times abs(valid\_err - train\_err))$
- $\frac{1}{(abs(valid\_err - train\_err) + 5 \times valid\_err)}$
- $\frac{1}{(train\_err + 5 \times valid\_err)}$
- $\frac{1}{(abs(train\_err - valid\_err))}$
- $\frac{1}{(2 \times abs(train\_err - valid\_err) + 1.25 \times train\_err + 3 \times valid\_err)}$
- $\frac{1}{(2 \times abs(train\_err - valid\_err) + 5 \times valid\_err + 2 \times train\_err)}$

The **main logic we used was a linear combination of `train_err` and `valid_err`** so that the fitness is greater when these values are closer to each other in magnitude and lesser than the previous time. That's why earlier we used the negation of it, and later decided to take the reciprocal. It did not affect the algo much, given how we implemented the selection function.

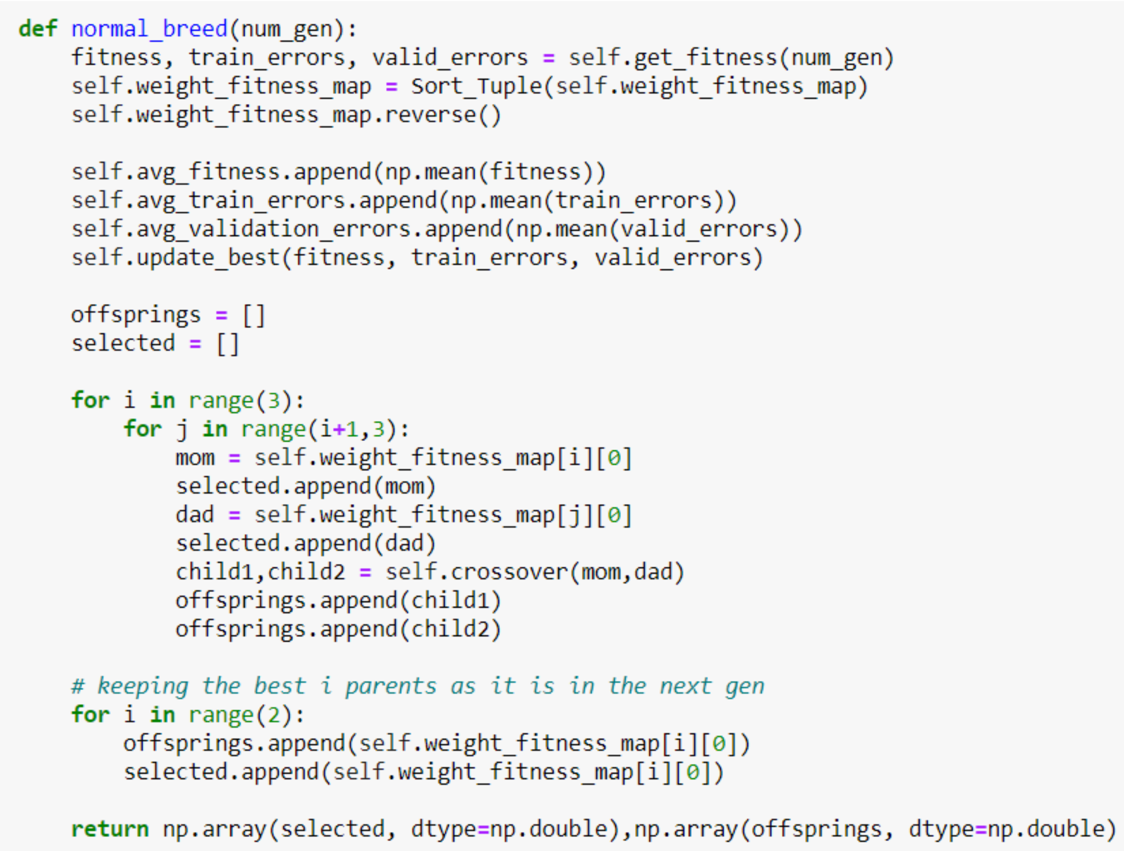
More weightage was given to the variable that we wanted to decrease more in further process. For example in $\frac{1}{(train\_err + 5 \times valid\_err)}$, valid_err is scaled by 5, because we want it to decrease much more than train_err.

In the earlier iterations, we only considered minimizing `train_err` and `valid_err` and only later when these values were reduced, we also brought in the absolute of difference between the two. So that the error values come closer to each other, which means it will be able to generalize better on unseen datasets.

3. Selection Function and process

This is the process in which we select a few of the parents based on their fitness, for further process. Before selection we sorted the population based on their fitness values. And then selected them for crossover.

We tried different selection functions for the model, but we used this one for most time.



```python
def normal_breed(num_gen):
    fitness, train_errors, valid_errors = self.get_fitness(num_gen)
    self.weight_fitness_map = Sort_Tuple(self.weight_fitness_map)
    self.weight_fitness_map.reverse()

    self.avg_fitness.append(np.mean(fitness))
    self.avg_train_errors.append(np.mean(train_errors))
    self.avg_validation_errors.append(np.mean(valid_errors))
    self.update_best(fitness, train_errors, valid_errors)

    offsprings = []
    selected = []

    for i in range(3):
        for j in range(i+1,3):
            mom = self.weight_fitness_map[i][0]
            selected.append(mom)
            dad = self.weight_fitness_map[j][0]
            selected.append(dad)
            child1,child2 = self.crossover(mom,dad)
            offsprings.append(child1)
            offsprings.append(child2)

    # keeping the best i parents as it is in the next gen
    for i in range(2):
        offsprings.append(self.weight_fitness_map[i][0])
        selected.append(self.weight_fitness_map[i][0])

    return np.array(selected, dtype=np.double),np.array(offsprings, dtype=np.double)
```

In this we took the best two parents as it is, and then took all the possible pairs from the best 3 parents. $\binom{3}{2}$ = 3, and each pair gives two children. So total offsprings = 8.

When we took 10 as the population size, we took the best 4 parents as it is and then the same all the possible pairs from the best 3 parents.

For the first few generations, we used a different selection function, known as the Russian Roulette.

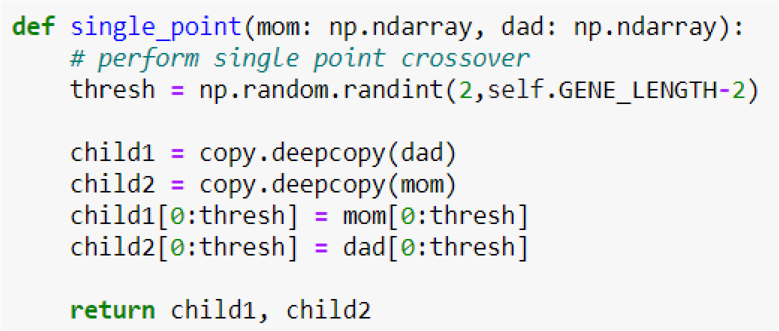
Which gives some probability to each vector depending on the fitness value. We only used it in the beginning but later completely dismissed using this, and only used the technique discussed above. Because we wanted to select only the best parents, and as the fitness values in later stages were similar, we just selected them based on the sorting, to ensure only the good ones filter through.

4. Crossover Function and process

This is the process of combining the selected pair of parents to get children. We tried different crossover functions but used the **Single Point Crossover** function as the default choice.

- **Single Point Crossover**

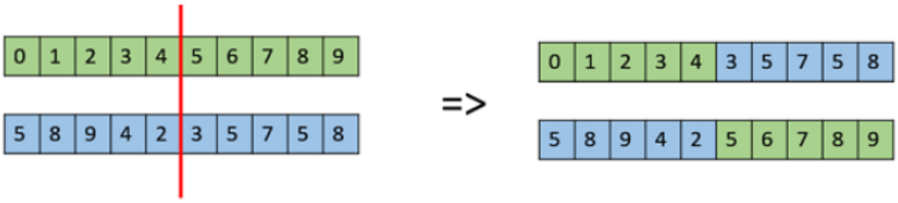
In this the point of crossover is found randomly, and then the two parts are crossed to make children.



```python
def single_point(mom: np.ndarray, dad: np.ndarray):
    # perform single point crossover
    thresh = np.random.randint(2,self.GENE_LENGTH-2)

    child1 = copy.deepcopy(dad)
    child2 = copy.deepcopy(mom)
    child1[0:thresh] = mom[0:thresh]
    child2[0:thresh] = dad[0:thresh]

    return child1, child2
```



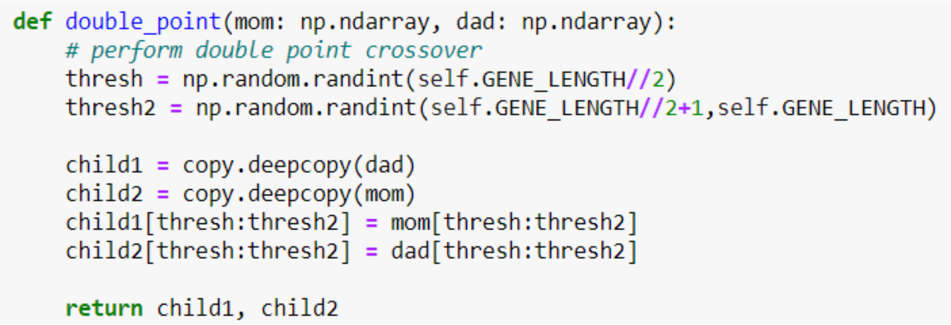
0 1 2 3 4 5 6 7 8 9
5 8 9 4 2 3 5 7 5 8

=>

0 1 2 3 4 3 5 7 5 8
5 8 9 4 2 5 6 7 8 9

- **Double Point Crossover**

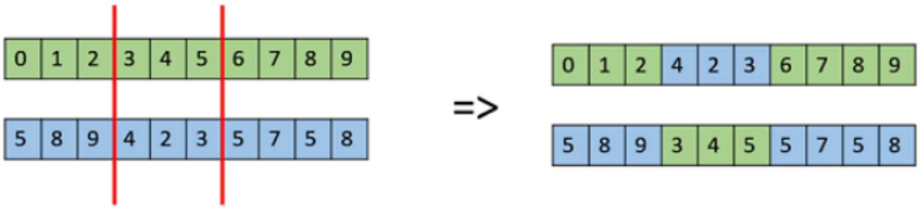
Similar to Single Point, but with two points as partition. This did not do any advantage in our case so after trying it out, we switched back to Single Point.



```python
def double_point(mom: np.ndarray, dad: np.ndarray):
    # perform double point crossover
    thresh = np.random.randint(self.GENE_LENGTH//2)
    thresh2 = np.random.randint(self.GENE_LENGTH//2+1,self.GENE_LENGTH)

    child1 = copy.deepcopy(dad)
    child2 = copy.deepcopy(mom)
    child1[thresh:thresh2] = mom[thresh:thresh2]
    child2[thresh:thresh2] = dad[thresh:thresh2]

    return child1, child2
```



0 1 2 3 4 5 6 7 8 9

5 8 9 4 2 3 5 7 5 8

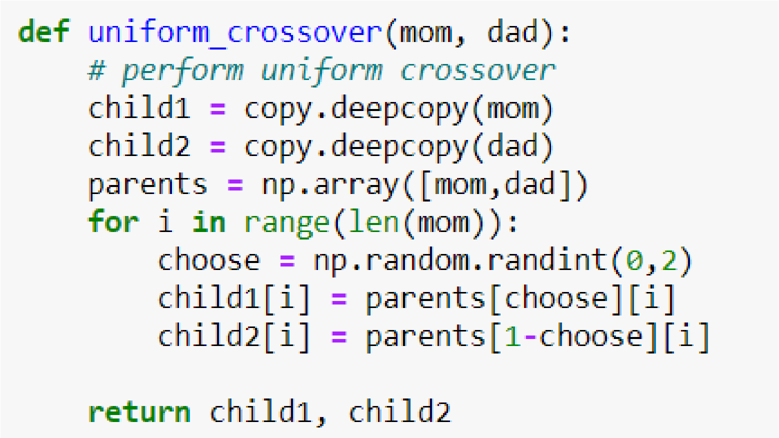
=>

0 1 2 4 2 3 6 7 8 9

5 8 9 3 4 5 5 7 5 8

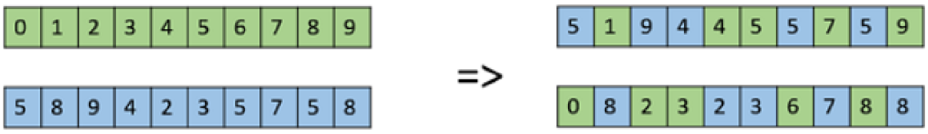
- **Uniform Crossover**

In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We tried this much later, but still not major difference in performance was observed so we switched between this and Single Point based on our mood (just to randomize the things, hoping for different results).



```
def uniform_crossover(mom, dad):
    # perform uniform crossover
    child1 = copy.deepcopy(mom)
    child2 = copy.deepcopy(dad)
    parents = np.array([mom,dad])
    for i in range(len(mom)):
        choose = np.random.randint(0,2)
        child1[i] = parents[choose][i]
        child2[i] = parents[1-choose][i]

    return child1, child2
```



0 1 2 3 4 5 6 7 8 9

5 8 9 4 2 3 5 7 5 8

=>

5 1 9 4 4 5 5 7 5 9

0 8 2 3 2 3 6 7 8 8

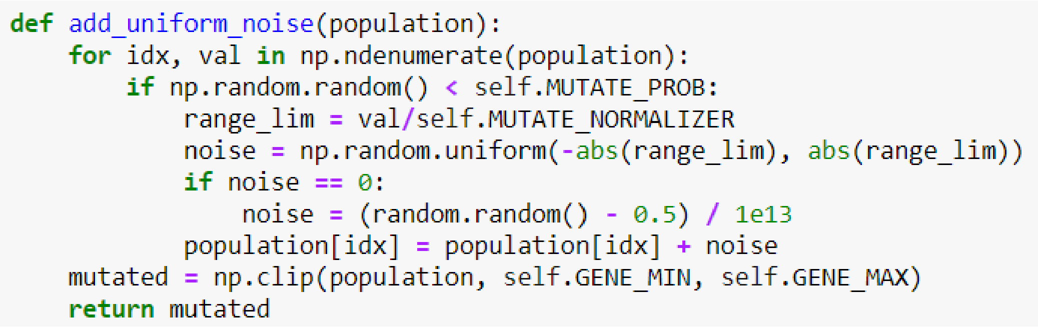
5. Mutation Function and process

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. Mutation alters one or more gene values in a chromosome from its initial state.

For our code, we took two approaches:

- **Adding Uniform noise**

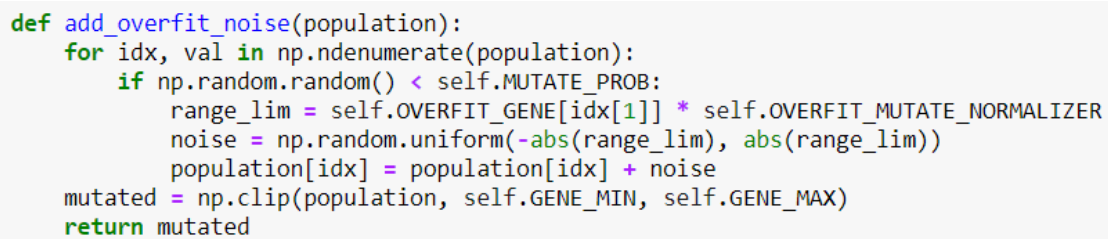
  For every gene, and for every chromosome, with some probability of selecting to mutate that chromosome, we find some percentage of the value of that chromosome, then taking the Uniform distribution for that value, add that to the chromosome.



```
def add_uniform_noise(population):
    for idx, val in np.ndenumerate(population):
        if np.random.random() < self.MUTATE_PROB:
            range_lim = val/self.MUTATE_NORMALIZER
            noise = np.random.uniform(-abs(range_lim), abs(range_lim))
            if noise == 0:
                noise = (random.random() - 0.5) / 1e13
            population[idx] = population[idx] + noise
    mutated = np.clip(population, self.GENE_MIN, self.GENE_MAX)
    return mutated
```

- **Adding overfit vector noise**

  For every gene, and for every chromosome, with some probability of selecting to mutate that chromosome, we find some percentage of the value from the Overfit vector at the same chromosome, then taking the Uniform distribution for that value, add that to the chromosome.



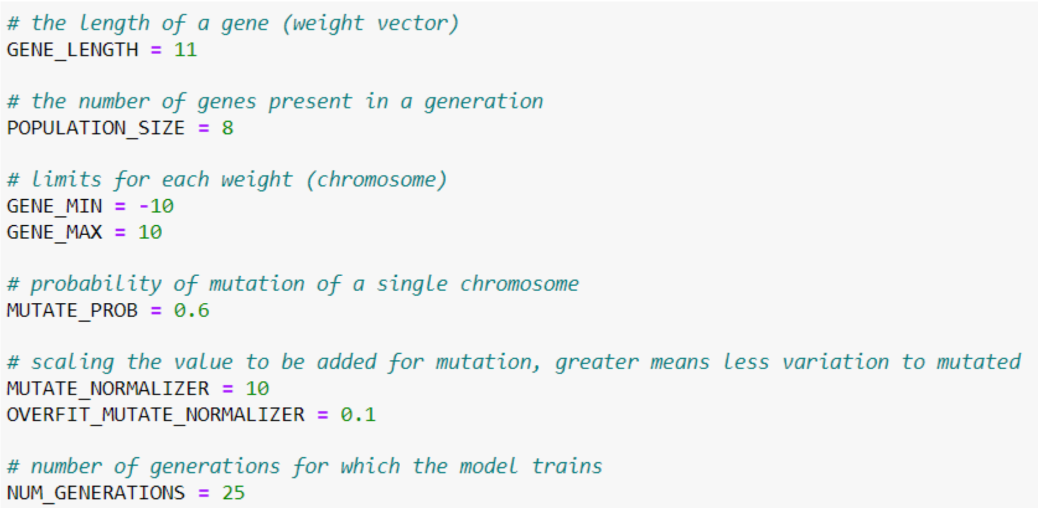
```
def add_overfit_noise(population):
    for idx, val in np.ndenumerate(population):
        if np.random.random() < self.MUTATE_PROB:
            range_lim = self.OVERFIT_GENE[idx[1]] * self.OVERFIT_MUTATE_NORMALIZER
            noise = np.random.uniform(-abs(range_lim), abs(range_lim))
            population[idx] = population[idx] + noise
    mutated = np.clip(population, self.GENE_MIN, self.GENE_MAX)
    return mutated
```

We started with adding Uniform noise, but many a times our population was not improving, so we tried some different things to mutate, and the overfit noise worked well for our case, as it showed some improvement in the training process. We dont know the exact reason for this but assume this happens because overfit vector values give less train error, adding this noise tries to change our vector so it gives lesser train error.

Hyperparameters

These were the hyperparameters that were used, and some of have been discussed above too.

These hyperparameters were used while training.



```
# the length of a gene (weight vector)
GENE_LENGTH = 11

# the number of genes present in a generation
POPULATION_SIZE = 8

# limits for each weight (chromosome)
GENE_MIN = -10
GENE_MAX = 10

# probability of mutation of a single chromosome
MUTATE_PROB = 0.6

# scaling the value to be added for mutation, greater means less variation to mutated
MUTATE_NORMALIZER = 10
OVERFIT_MUTATE_NORMALIZER = 0.1

# number of generations for which the model trains
NUM_GENERATIONS = 25
```

- `NUM_GENERATIONS` was changed before every code run depending, on the number of generations we wished to run.
- `MUTATE_PROB` is the probability with which a chromosome is selected during the mutation process. We kept it in the range **0.3 to 0.8.** Used the higher value when our algo was not improving, so it was stuck in some local minima, so tried this hoping with more mutation the algo can improve. Lower value means, very few mutations happening, this resulted in very slow learning, so we used 0.5 or 0.6 mostly.
- `MUTATE_NORMALIZER` this is the scaling factor of the value that will be added as noise in mutation. Reciprocal of this value is used. **10** means that change can be between -10% to 10%, of the overfit value or the current value. Keeping this number low means very big mutations and was used when stuck. Higher value of this number would mean very slow and gradual changes in the population. We used as low as **2 (50% change)** and as high as **100 (1% change)**.
- `OVERFIT_MUTATE_NORMALIZER` is the scaling factor of the value from overfit vector, that is added as noise. Similar to `MUTATE_NORMALIZER`, but its not in reciprocal.
- `POPULATION_SIZE` this is the number of genes that were in each generation. We used 8 or 10, as 10 would have taken more number of API requests, we switched to 8 and kept it as is. We didnt much difference between 8 or 10. Keeping a much greater population seemed a wastage of requests, though in theory it would have given more diversity in the population.
- The Single Point for crossover was found randomly and took value between 0 and `GENE_LENGTH  - 1`
- For selection of 8 children from the current 8 parents, we selected the first two best parents as it is, and then took all pairs from the first 3 parents. The lower 8 parents were ignored in each generation.

Train and Validation Error

Some of our best vectors gave errors of the order: `[train_err, val_err]`

`[ 10e10, 10e10 ]` , `[ 8e10, 8e10 ]` or a very few about `[ 6e10, 6e10 ]` (slightly close to overfit).

These values are very low as compared to the initial validation error (1e13) of the overfit vector. And also greater than the train error (1e10) of the overfit vector. And also the difference between the train and validation errors is much smaller, in the order of **1e8 or 1e7** in best cases. This means that the vector is not overfitting the train or validation set, and will be able to generalize well.

```
one of the best vectors, that we found
vector = [
          6.375574794612503,
          -6.090882279937824e-18,
          7.468540963434164e-06,
          4.022035369563623e-09,
          -1.2486049836655162,
          7.744241638979762e-20,
          4.497434903624772e-21,
          1.0248913311070943e-13,
          1.0563239599063873e-07,
          -3.9889486689250545e-15,
          -1.1441395707995704e-15
      ]
train_error = 86068070577.39464 (approx 8e10)
validation_error = 86035791318.08537 (approx 8e10)
```

Statistical Information

Number of Generations

The point of convergence. We think that the point of convergence is when we are stuck at a local minima or is at a point with very similar train and validation error. It took nearly 100 (approx) generations starting from the Overfit vector, to reach such condition. But then it got stuck. We further tried to run the algorithm for nearly 100 generations again, but with no luck. So we stopped.

Restarts

Many times we got stuck at local minima, when our vectors gave `[9e10, 9e10]` [train_err, val_err].

These vectors didn't change much, even with 100 generations.

So we had to restart multiple times, like 3 or 4 times.

Once we even considered starting out with Zero vector, but its learning was very slow, and it performed much worse than before, even for earlier iterations.

So we again went with Overfit Vector, increased the mutation probability and scaling. Then again training for about 80~100 generations, we got stuck at a vector with similar order of errors.

By changing the fitness function drastically, like only considering the difference of errors and not the errors individually, we got mixed results, and sometimes we got a vector with `[6e10, 6e10]` [train_err, val_err]. But this vector performed badly on the test set (leaderboard), we believe that it was overfitting slightly.

So in all we chose the vectors with errors about **8e10** in train and validation, and near it, basically from **6e10** to **10e10**.

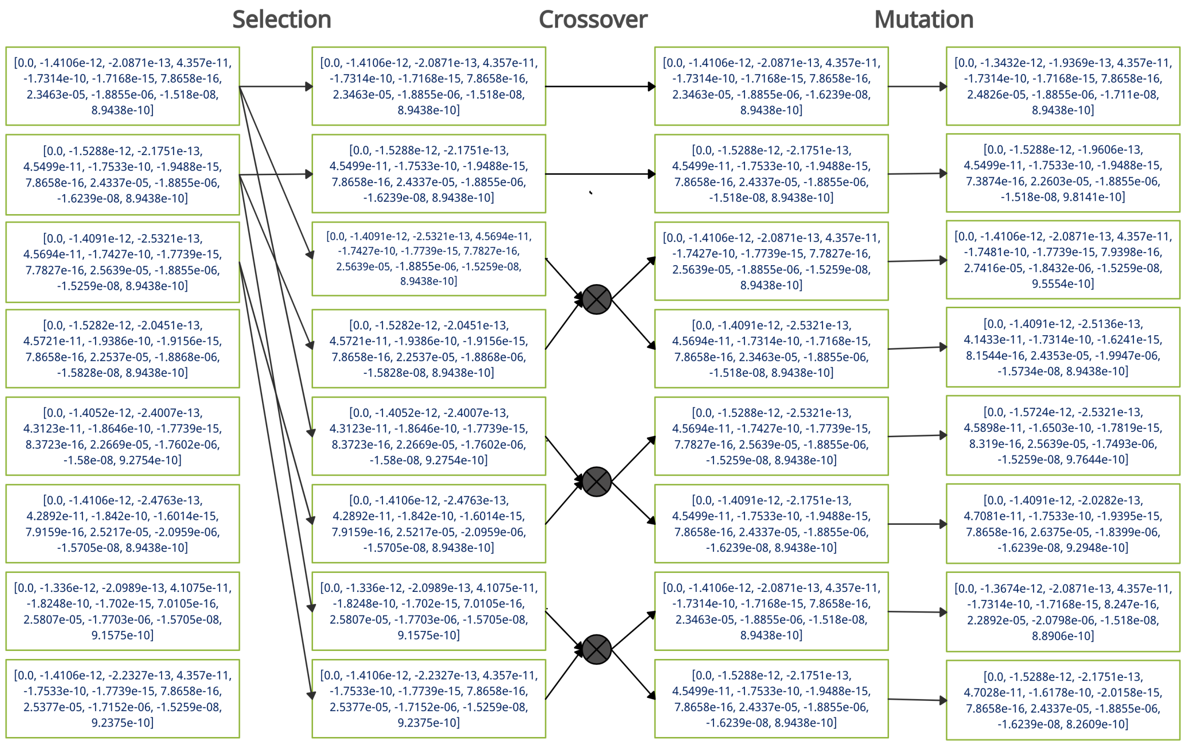
Tips n Tricks, Brute Force

When our algorithm was not improving at all, we tried to change the vector manually, trying different order of the numbers at different indices in the gene vector, to see what positions are more important than others. We didn't use this in our training tho, just sometimes altered the vector manually so it can perform better. For example changing the value of the first chromosome to -10 or 10 or 4 to see what happens to the error. It was found that some positions were more important than others, for example the chromosomes at position 1 and 2 were less critical. We forgot to save our results. We are not sure if its even correct, but we just tried it.

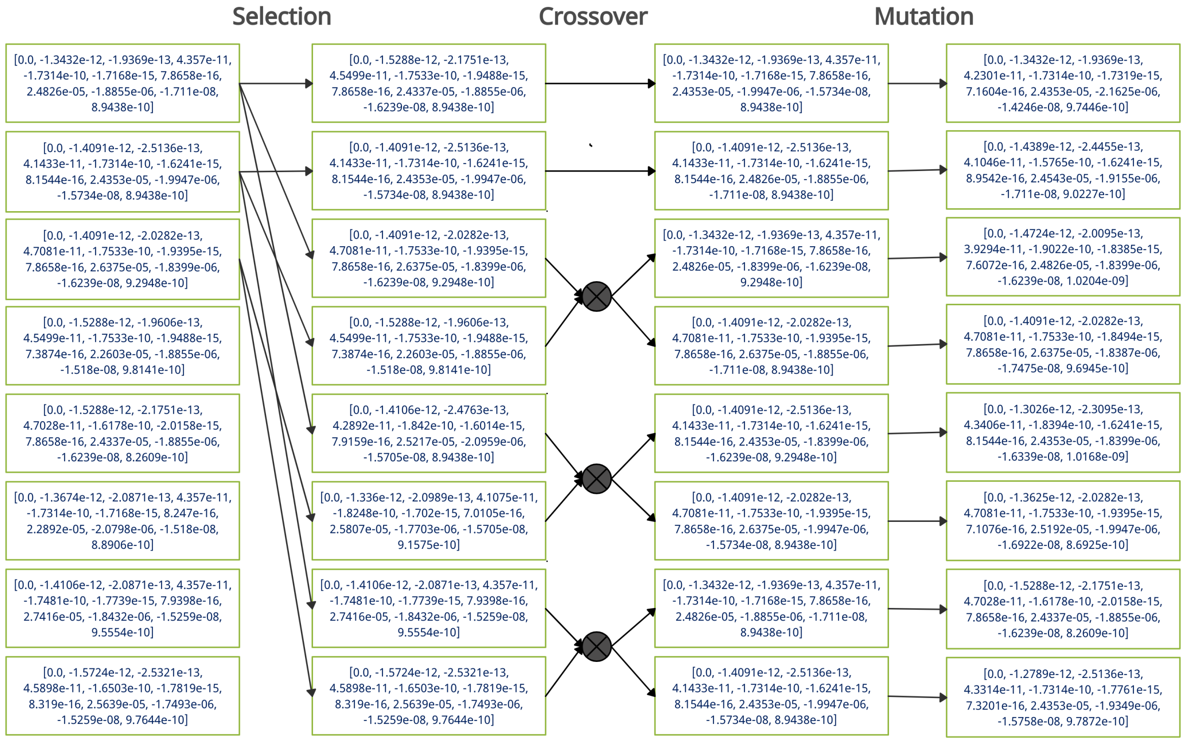
Diagrams

These are 3 iterations of the algorithm, taken from during our training process. The values in the vectors have been rounded to 4 decimal places due to the lack of space to print them in boxes. But in reality these values are significant till more than 12 decimal places. Some mutations are only seen at very small scale, which appear below as not mutated due to this rounding error.

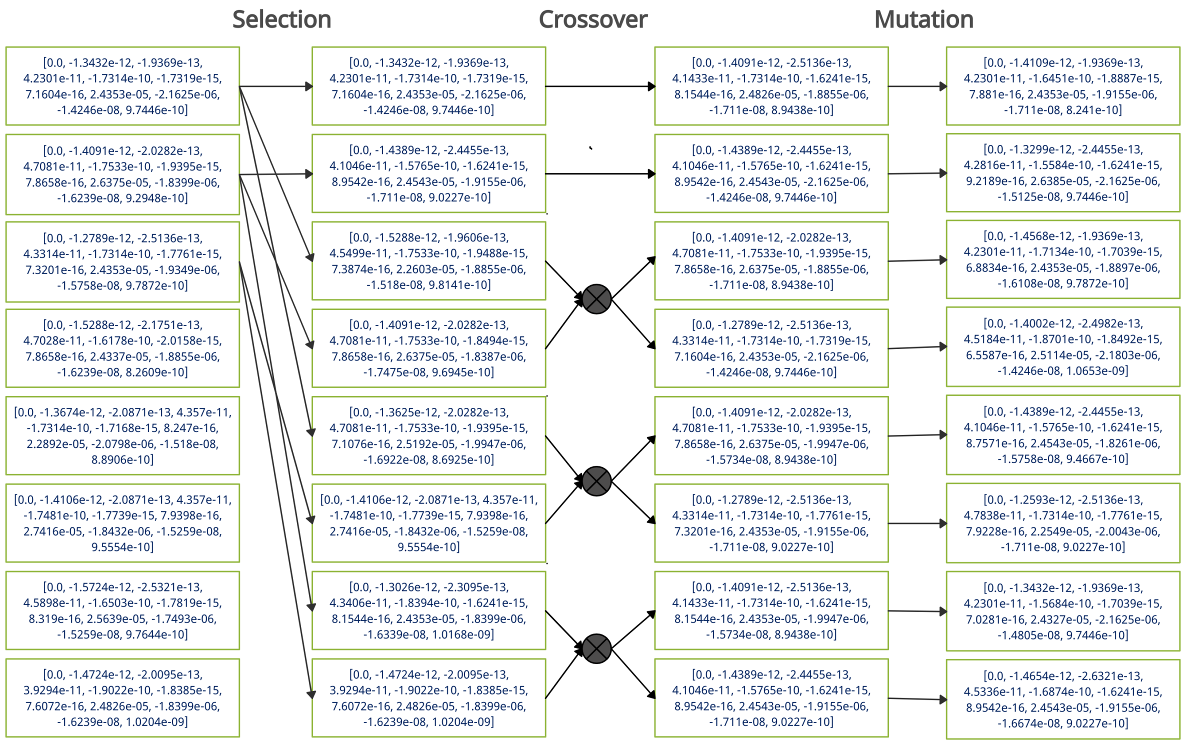
Iteration 1


Selection
Crossover
Mutation
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.3463e-05, -1.8855e-06, -1.518e-08, 8.9438e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5321e-13, 4.5694e-11, -1.7427e-10, -1.7739e-15, 7.7827e-16, 2.5639e-05, -1.8855e-06, -1.5259e-08, 8.9438e-10]
[0.0, -1.5282e-12, -2.0451e-13, 4.5721e-11, -1.9386e-10, -1.9156e-15, 7.8658e-16, 2.2537e-05, -1.8868e-06, -1.5828e-08, 8.9438e-10]
[0.0, -1.4052e-12, -2.4007e-13, 4.3123e-11, -1.8646e-10, -1.7739e-15, 8.3723e-16, 2.2669e-05, -1.7602e-06, -1.58e-08, 9.2754e-10]
[0.0, -1.4106e-12, -2.4763e-13, 4.2892e-11, -1.842e-10, -1.6014e-15, 7.9159e-16, 2.5217e-05, -2.0959e-06, -1.5705e-08, 8.9438e-10]
[0.0, -1.336e-12, -2.0989e-13, 4.1075e-11, -1.8248e-10, -1.702e-15, 7.0105e-16, 2.5807e-05, -1.7703e-06, -1.5705e-08, 9.1575e-10]
[0.0, -1.4106e-12, -2.2327e-13, 4.357e-11, -1.7533e-10, -1.7739e-15, 7.8658e-16, 2.5377e-05, -1.7152e-06, -1.5259e-08, 9.2375e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.3463e-05, -1.8855e-06, -1.518e-08, 8.9438e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5321e-13, 4.5694e-11, -1.7427e-10, -1.7739e-15, 7.7827e-16, 2.5639e-05, -1.8855e-06, -1.5259e-08, 8.9438e-10]
[0.0, -1.5282e-12, -2.0451e-13, 4.5721e-11, -1.9386e-10, -1.9156e-15, 7.8658e-16, 2.2537e-05, -1.8868e-06, -1.5828e-08, 8.9438e-10]
[0.0, -1.4052e-12, -2.4007e-13, 4.3123e-11, -1.8646e-10, -1.7739e-15, 8.3723e-16, 2.2669e-05, -1.7602e-06, -1.58e-08, 9.2754e-10]
[0.0, -1.4106e-12, -2.4763e-13, 4.2892e-11, -1.842e-10, -1.6014e-15, 7.9159e-16, 2.5217e-05, -2.0959e-06, -1.5705e-08, 8.9438e-10]
[0.0, -1.336e-12, -2.0989e-13, 4.1075e-11, -1.8248e-10, -1.702e-15, 7.0105e-16, 2.5807e-05, -1.7703e-06, -1.5705e-08, 9.1575e-10]
[0.0, -1.4106e-12, -2.2327e-13, 4.357e-11, -1.7533e-10, -1.7739e-15, 7.8658e-16, 2.5377e-05, -1.7152e-06, -1.5259e-08, 9.2375e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.3463e-05, -1.8855e-06, -1.6239e-08, 8.9438e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.518e-08, 8.9438e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7427e-10, -1.7739e-15, 7.7827e-16, 2.5639e-05, -1.8855e-06, -1.5259e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5321e-13, 4.5694e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.3463e-05, -1.8855e-06, -1.518e-08, 8.9438e-10]
[0.0, -1.5288e-12, -2.5321e-13, 4.5694e-11, -1.7427e-10, -1.7739e-15, 7.7827e-16, 2.5639e-05, -1.8855e-06, -1.5259e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.1751e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.9438e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.3463e-05, -1.8855e-06, -1.518e-08, 8.9438e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.9438e-10]
[0.0, -1.3432e-12, -1.9369e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.4826e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.5288e-12, -1.9606e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.3874e-16, 2.2603e-05, -1.8855e-06, -1.518e-08, 9.8141e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7481e-10, -1.7739e-15, 7.9398e-16, 2.7416e-05, -1.8432e-06, -1.5259e-08, 9.5554e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.5724e-12, -2.5321e-13, 4.5898e-11, -1.6503e-10, -1.7819e-15, 8.319e-16, 2.5639e-05, -1.7493e-06, -1.5259e-08, 9.7644e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.8399e-06, -1.6239e-08, 9.2948e-10]
[0.0, -1.3674e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 8.247e-16, 2.2892e-05, -2.0798e-06, -1.518e-08, 8.8906e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.7028e-11, -1.6178e-10, -2.0158e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.2609e-10]

Iteration 2


Selection
Crossover
Mutation
[0.0, -1.3432e-12, -1.9369e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.4826e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.8399e-06, -1.6239e-08, 9.2948e-10]
[0.0, -1.5288e-12, -1.9606e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.3874e-16, 2.2603e-05, -1.8855e-06, -1.518e-08, 9.8141e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.7028e-11, -1.6178e-10, -2.0158e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.2609e-10]
[0.0, -1.3674e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 8.247e-16, 2.2892e-05, -2.0798e-06, -1.518e-08, 8.8906e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7481e-10, -1.7739e-15, 7.9398e-16, 2.7416e-05, -1.8432e-06, -1.5259e-08, 9.5554e-10]
[0.0, -1.5724e-12, -2.5321e-13, 4.5898e-11, -1.6503e-10, -1.7819e-15, 8.319e-16, 2.5639e-05, -1.7493e-06, -1.5259e-08, 9.7644e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.8399e-06, -1.6239e-08, 9.2948e-10]
[0.0, -1.5288e-12, -1.9606e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.3874e-16, 2.2603e-05, -1.8855e-06, -1.518e-08, 9.8141e-10]
[0.0, -1.4106e-12, -2.4763e-13, 4.2892e-11, -1.842e-10, -1.6014e-15, 7.9159e-16, 2.5217e-05, -2.0959e-06, -1.5705e-08, 8.9438e-10]
[0.0, -1.336e-12, -2.0989e-13, 4.1075e-11, -1.8248e-10, -1.702e-15, 7.0105e-16, 2.5807e-05, -1.7703e-06, -1.5705e-08, 9.1575e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7481e-10, -1.7739e-15, 7.9398e-16, 2.7416e-05, -1.8432e-06, -1.5259e-08, 9.5554e-10]
[0.0, -1.5724e-12, -2.5321e-13, 4.5898e-11, -1.6503e-10, -1.7819e-15, 8.319e-16, 2.5639e-05, -1.7493e-06, -1.5259e-08, 9.7644e-10]
[0.0, -1.3432e-12, -1.9369e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.4353e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4826e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.3432e-12, -1.9369e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.4826e-05, -1.8399e-06, -1.6239e-08, 9.2948e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.8399e-06, -1.6239e-08, 9.2948e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.3432e-12, -1.9369e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 7.8658e-16, 2.4826e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.3432e-12, -1.9369e-13, 4.2301e-11, -1.7314e-10, -1.7319e-15, 7.1604e-16, 2.4353e-05, -2.1625e-06, -1.4246e-08, 9.7446e-10]
[0.0, -1.4389e-12, -2.4455e-13, 4.1046e-11, -1.5765e-10, -1.6241e-15, 8.9542e-16, 2.4543e-05, -1.9155e-06, -1.711e-08, 9.0227e-10]
[0.0, -1.4724e-12, -2.0095e-13, 3.9294e-11, -1.9022e-10, -1.8385e-15, 7.6072e-16, 2.4826e-05, -1.8399e-06, -1.6239e-08, 1.0204e-09]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.8494e-15, 7.8658e-16, 2.6375e-05, -1.8387e-06, -1.7475e-08, 9.6945e-10]
[0.0, -1.3026e-12, -2.3095e-13, 4.3406e-11, -1.8394e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.8399e-06, -1.6339e-08, 1.0168e-09]
[0.0, -1.3625e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.1076e-16, 2.5192e-05, -1.9947e-06, -1.6922e-08, 8.6925e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.7028e-11, -1.6178e-10, -2.0158e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.2609e-10]
[0.0, -1.2789e-12, -2.5136e-13, 4.3314e-11, -1.7314e-10, -1.7761e-15, 7.3201e-16, 2.4353e-05, -1.9349e-06, -1.5758e-08, 9.7872e-10]

Iteration 3


Selection
Crossover
Mutation
[0.0, -1.3432e-12, -1.9369e-13, 4.2301e-11, -1.7314e-10, -1.7319e-15, 7.1604e-16, 2.4353e-05, -2.1625e-06, -1.4246e-08, 9.7446e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.8399e-06, -1.6239e-08, 9.2948e-10]
[0.0, -1.2789e-12, -2.5136e-13, 4.3314e-11, -1.7314e-10, -1.7761e-15, 7.3201e-16, 2.4353e-05, -1.9349e-06, -1.5758e-08, 9.7872e-10]
[0.0, -1.5288e-12, -2.1751e-13, 4.7028e-11, -1.6178e-10, -2.0158e-15, 7.8658e-16, 2.4337e-05, -1.8855e-06, -1.6239e-08, 8.2609e-10]
[0.0, -1.3674e-12, -2.0871e-13, 4.357e-11, -1.7314e-10, -1.7168e-15, 8.247e-16, 2.2892e-05, -2.0798e-06, -1.518e-08, 8.8906e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7481e-10, -1.7739e-15, 7.9398e-16, 2.7416e-05, -1.8432e-06, -1.5259e-08, 9.5554e-10]
[0.0, -1.5724e-12, -2.5321e-13, 4.5898e-11, -1.6503e-10, -1.7819e-15, 8.319e-16, 2.5639e-05, -1.7493e-06, -1.5259e-08, 9.7644e-10]
[0.0, -1.4724e-12, -2.0095e-13, 3.9294e-11, -1.9022e-10, -1.8385e-15, 7.6072e-16, 2.4826e-05, -1.8399e-06, -1.6239e-08, 1.0204e-09]
[0.0, -1.3432e-12, -1.9369e-13, 4.2301e-11, -1.7314e-10, -1.7319e-15, 7.1604e-16, 2.4353e-05, -2.1625e-06, -1.4246e-08, 9.7446e-10]
[0.0, -1.4389e-12, -2.4455e-13, 4.1046e-11, -1.5765e-10, -1.6241e-15, 8.9542e-16, 2.4543e-05, -1.9155e-06, -1.711e-08, 9.0227e-10]
[0.0, -1.5288e-12, -1.9606e-13, 4.5499e-11, -1.7533e-10, -1.9488e-15, 7.3874e-16, 2.2603e-05, -1.8855e-06, -1.518e-08, 9.8141e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.8494e-15, 7.8658e-16, 2.6375e-05, -1.8387e-06, -1.7475e-08, 9.6945e-10]
[0.0, -1.3625e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.1076e-16, 2.5192e-05, -1.9947e-06, -1.6922e-08, 8.6925e-10]
[0.0, -1.4106e-12, -2.0871e-13, 4.357e-11, -1.7481e-10, -1.7739e-15, 7.9398e-16, 2.7416e-05, -1.8432e-06, -1.5259e-08, 9.5554e-10]
[0.0, -1.3026e-12, -2.3095e-13, 4.3406e-11, -1.8394e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.8399e-06, -1.6339e-08, 1.0168e-09]
[0.0, -1.4724e-12, -2.0095e-13, 3.9294e-11, -1.9022e-10, -1.8385e-15, 7.6072e-16, 2.4826e-05, -1.8399e-06, -1.6239e-08, 1.0204e-09]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4826e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.4389e-12, -2.4455e-13, 4.1046e-11, -1.5765e-10, -1.6241e-15, 8.9542e-16, 2.4543e-05, -2.1625e-06, -1.4246e-08, 9.7446e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.8855e-06, -1.711e-08, 8.9438e-10]
[0.0, -1.2789e-12, -2.5136e-13, 4.3314e-11, -1.7314e-10, -1.7319e-15, 7.1604e-16, 2.4353e-05, -2.1625e-06, -1.4246e-08, 9.7446e-10]
[0.0, -1.4091e-12, -2.0282e-13, 4.7081e-11, -1.7533e-10, -1.9395e-15, 7.8658e-16, 2.6375e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.2789e-12, -2.5136e-13, 4.3314e-11, -1.7314e-10, -1.7761e-15, 7.3201e-16, 2.4353e-05, -1.9155e-06, -1.711e-08, 9.0227e-10]
[0.0, -1.4091e-12, -2.5136e-13, 4.1433e-11, -1.7314e-10, -1.6241e-15, 8.1544e-16, 2.4353e-05, -1.9947e-06, -1.5734e-08, 8.9438e-10]
[0.0, -1.4389e-12, -2.4455e-13, 4.1046e-11, -1.5765e-10, -1.6241e-15, 8.9542e-16, 2.4543e-05, -1.9155e-06, -1.711e-08, 9.0227e-10]
[0.0, -1.4109e-12, -1.9369e-13, 4.2301e-11, -1.6451e-10, -1.8887e-15, 7.881e-16, 2.4353e-05, -1.9155e-06, -1.711e-08, 8.241e-10]
[0.0, -1.3299e-12, -2.4455e-13, 4.2816e-11, -1.5584e-10, -1.6241e-15, 9.2189e-16, 2.6385e-05, -2.1625e-06, -1.5125e-08, 9.7446e-10]
[0.0, -1.4568e-12, -1.9369e-13, 4.2301e-11, -1.7134e-10, -1.7039e-15, 6.8834e-16, 2.4353e-05, -1.8897e-06, -1.6108e-08, 9.7872e-10]
[0.0, -1.4002e-12, -2.4982e-13, 4.5184e-11, -1.8701e-10, -1.8492e-15, 6.5587e-16, 2.5114e-05, -2.1803e-06, -1.4246e-08, 1.0653e-09]
[0.0, -1.4389e-12, -2.4455e-13, 4.1046e-11, -1.5765e-10, -1.6241e-15, 8.7571e-16, 2.4543e-05, -1.8261e-06, -1.5758e-08, 9.4667e-10]
[0.0, -1.2593e-12, -2.5136e-13, 4.7838e-11, -1.7314e-10, -1.7761e-15, 7.9228e-16, 2.2549e-05, -2.0043e-06, -1.711e-08, 9.0227e-10]
[0.0, -1.3432e-12, -1.9369e-13, 4.2301e-11, -1.5684e-10, -1.7039e-15, 7.0281e-16, 2.4327e-05, -2.1625e-06, -1.4805e-08, 9.7446e-10]
[0.0, -1.4654e-12, -2.6321e-13, 4.5336e-11, -1.6874e-10, -1.6241e-15, 8.9542e-16, 2.4543e-05, -1.9155e-06, -1.6674e-08, 9.0227e-10]