Genetic Algorithms Report

Problem Statement

Given an overfit vector a dataset, use genetic algorithms to reduce the overfitting by eliminating the noise and getting a lower error.

The function is a linear function on 11 dimensions with mean squared error as the error function.

Constraints

The solution is a 11 dimensional vector of the form:

$$\mathbf{w} = \left[w_0, w_2 \cdots w_{10}\right] -10 \leq w_i \leq 10 \; \forall i \in [0, 10]$$

About Genetic Algorithms

Genetic Algorithms takes the approach similar to the natural evolution. It takes "Survival of the fittest" as an approach to find a solution according to our requirements.

The algorithm takes the same approach as natural selections, but in a very simple manner.

Population Description

A pool of vectors which satisfy the constraints described by the problem.

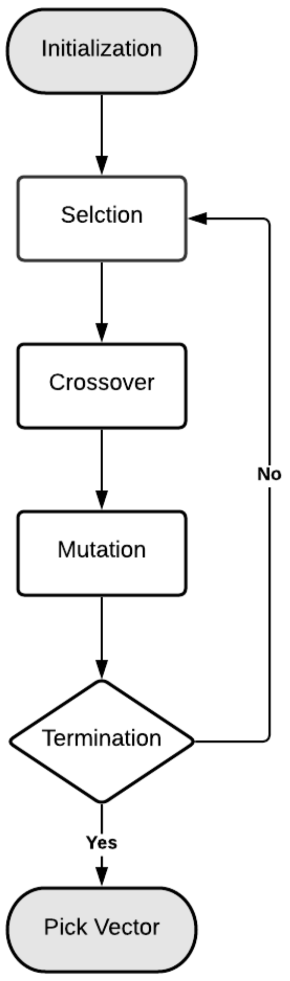
Individual Description

Each individual is a ordered list of values which is called a gene.

The Algorithm

Steps

In a broad sense, the algorithm is a loop with 3 abstract steps with a termination condition.



Initialization

A set of vectors are picked randomly or are given by the problem statement.

Crossover

The population undergoes crossover pairwise where 2 individuals referred as parents create 2 other individuals referred as offsprings which take their genes from either of the parents based on the crossover technique.

Mutation

A subset of the population undergoes changes in a subset of their genes.

Selection

The survival of an individual is based on its fitness, fitness is a function which maps an individual to a number. Higher fitness an individual has, higher chance of it surviving the selection. After selection the population left is the new generation.

Termination

The algorithm can be stopped after a generation if it satisfies the termination condition. The usual termination condition are either the number of generations or the convergence of the fitness.

Implementation Details

The base genetic algorithms which has been described above gives us the abstract understanding of the algorithm. The specifics of each step is dependent on the problem. We had tried multiple implementations for each step, which finally led us from the original over-fit vector to our current submission.

Initialization

Given that the problem statement had already given us the over-fit vector, the goal was to eliminate the noise from the weights.

Choosing zeros

We ran our original program (here by program I shall mean our genetic algorithm implementation) over the given over-fit vector and observed the proceeding generations, in order to identify the noise in the given data.

Why? Because identifying noise was the key factor in eliminating overfitting. So, once we had a rough idea about which elements of the vector corresponded with noise, we set them to 0 manually.

$$\begin{bmatrix} 0.0 \\ -1.45799022 \times 10^{-12} \\ -2.28980078 \times 10^{-13} \\ 4.62010753 \times 10^{-11} \\ -1.75214813 \times 10^{-10} \\ -1.8366977 \times 10^{-15} \\ 8.5294406 \times 10^{-16} \\ 2.29423303 \times 10^{-05} \\ -2.04721003 \times 10^{-06} \\ -1.59792834 \times 10^{-08} \\ 9.98214034 \times 10^{-10} \end{bmatrix} \rightarrow \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -1.8366977 \times 10^{-15} \\ 0.0 \\ 2.29423303 \times 10^{-05} \\ -2.04721003 \times 10^{-06} \\ -1.59792834 \times 10^{-08} \\ 9.98214034 \times 10^{-10} \end{bmatrix}$$

Crossover

Crossover has a large impact when the population size is large, but in our case, we had to set it to be small. Each pair of crossover produces 2 offsprings.

In the entire duration of our project, we have tried and tested a lot of heuristics for crossover.

Keep the Best

Before all the individuals are sent for crossover and mutation, we kept some (in final case just 1) of our population vector in the `nextGeneration` by default.

We allowed it to get cross-overed or in other words "to mate", but before that made sure to keep one copy of it unchanged in our `nextGeneration` vector.

```
# Gurantee that top two will be selected without any mutation or
# crossover: 9 = 8 + 1
nextGeneration = population[:1]
nonMutatedNextGen = population[:1]
```

Prioritizing the fittest

Rather than letting all the individuals be a part of the crossover, we only select the top half vectors for creating the next generation. But the process of picking the individuals out of this filtered population is random.

This is part of the selection since we are eliminating half of the population before making the offsprings - hence, ensuring that only the best vectors are allowed to "mate".

$$\begin{bmatrix} \text{Individual 1} \\ \text{Individual 2} \\ \vdots \\ \text{Individual 10} \end{bmatrix} \rightarrow \begin{bmatrix} \text{Individual 1} \\ \text{Individual 2} \\ \vdots \\ \text{Individual 5} \end{bmatrix}$$

We used this heuristic for the major part of our project. However, towards the end - when we were fine tuning our vectors - we ignored selection in this step and allowed all the vectors to "mate" or get cross-overed.

$$\begin{bmatrix} \text{Individual 1} \\ \text{Individual 2} \\ \vdots \\ \text{Individual 10} \end{bmatrix} \rightarrow \begin{bmatrix} \text{Individual 1} \\ \text{Individual 2} \\ \vdots \\ \text{Individual 10} \end{bmatrix}$$

Crossover Function

We had tried a lot of variants for our crossover function but in the end settled with simulated binary crossover, as mentioned below.

Simulated Binary Crossover

As evident below, in our crossover function we took two parents, namely $parent_a$ and $parent_b$ along with a hyper-parameter β.

The hyper-parameter has the following given range: $0 < \beta < 1)$.

The purpose of β is to serve as a distribution index that determines how far children go from parents. The greater its value the closer the children are to parents.

The entire idea behind simulated binary crossover is to generate two children from two parents, satisfying the following equation. All the while, being able to control the variation between the parents and children using the distribution index value.

$$\frac{\text{offspring}_a + \text{offspring}_b}{2} = \frac{\text{parent}_a + \text{parent}_b}{2}$$

```
def crossOver(
        self,
        parent_a: Individual,
        parent_b: Individual,
        fitness_a: float,
        fitness_b: float,
    ) -> Tuple[Individual, Individual]:
        """Crosses two parents
        to give a two new offsprings"""

        offspring_a = (
            (
              (1 + self.beta) * parent_a) +
                    ((1 - self.beta) * parent_b)
        ) / 2

        offspring_b = (
            (
              (1 - self.beta) * parent_a) +
                    ((1 + self.beta) * parent_b)
        ) / 2

        return offspring_a, offspring_b
```

Example

$$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -5.76450871 \times 10^{-16} \\ 0.0 \\ 1.49845448 \times 10^{-05} \\ -1.04 \times 10^{-06} \\ -5.46 \times 10^{-09} \\ 3.83233413 \times 10^{-10} \end{bmatrix} \rightarrow \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -5.77036367 \times 10^{-16} \\ 0.0 \\ 1.50074509 \times 10^{-05} \\ -1.03949091 \times 10^{-06} \\ -5.46 \times 10^{-09} \\ 3.84 \times 10^{-10} \end{bmatrix}$$

Mutation

For values of such scale with such large error values, mutation was the most crucial part of the algorithm, hence involved a lot of experimentation. Mentioned below is the final method that was used.

Gaussian Mutation

The value by which a gene must be mutated should be of the same order as that of the value itself. The the probability of mutation itself also must be not very large, larger the mutation, smaller should the probability be.

Gaussian distribution with appropriate standard deviation and mean should suffice these conditions.

- The mutated value should be close to the original value since the mutation is small, which means $\mu = w$ where w is the value of the gene.
- The mutation value must of the same order as the value itself, $\sigma = \frac{w}{scale}$ where scale is a hyper-parameter which has to be tuned.
- The scale is the order by which the mutation is smaller than the value itself, for finer changes we set the scale to be high.
- Since there is a strict bound over the weights, if the weights ever cross them after mutation, they are set to 0.

The above function was **vectorized** and was applied to the population in parallel.

```
def mutateOffspring(self, offspring: Individual) -> Individual:
        """
        Mutates an individual Current Algo: Select some indices randomly and
        choose a new number from a gaussian distribution with mean as the
        number at that index
        """
        flagArray = sciStats.bernoulli.rvs(  # type:ignore
            p=self.mutationProbability, size=offspring.shape
        )

        generateGaus = lambda x: np.clip(
            np.random.normal(loc=x, scale=abs(x) / 5e2),
            -self.scalingFactor,
            self.scalingFactor,
        )
        vectorizedGenerateGaus = np.vectorize(generateGaus)
        gausArray = vectorizedGenerateGaus(offspring)

        offspring[flagArray == 1] = gausArray[flagArray == 1]
        return offspring
```

Example

$$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -5.77036367 \times 10^{-16} \\ 0.0 \\ 1.50074509 \times 10^{-05} \\ -1.03949091 \times 10^{-06} \\ -5.46 \times 10^{-09} \\ 3.84 \times 10^{-10} \end{bmatrix} \rightarrow \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -5.74980395 \times 10^{-16} \\ 0.0 \\ 1.50074509 \times 10^{-05} \\ -1.03755096 \times 10^{-06} \\ -5.46 \times 10^{-09} \\ 3.85718176 \times 10^{-10} \end{bmatrix}$$

Selection

The main purpose of selection here is to sort the new offsprings based on their fitness values in descending order.

Selection happens based on the fitness values. In our original implementation, the only form of selection that is happening is during the crossover when we eliminate half the population.

In later implementations, we included the following snippet because we were not eliminating individuals from mating during crossover.

```
population = nextGeneration[:self.populationSize]
```

Saving the fit vector

Other than sorting, we also save the most fit function of the population, we compare it with the best vector so far and replace if it performs better. This gives us the best vector throughout all the populations.

Fitness function

Fitness function is designed based on the problem statement and the requirements. For a simple linear regression problem with mean squared error problem, the fitness function can just be based on the train error.

Fitness function is more complicated when there are chances of falling in a local minima, but in the case of mean squared error there are no local minimas.

Throughout our implementation, we have played around with several forms for the fitness function. What we used the fitness function for is to sort the population — during initialization — based on the **chosen definition of fitness function**.

Below shown are some of our choices:

```
1. sortedIndices = (abs(fitness)).argsort()
2. sortedIndices = (abs(Testfitness)).argsort()
3. sortedIndices = (abs(fitness + testFitness)).argsort()
4. sortedIndices = (abs(fitness - testFitness)).argsort()
```

$$\text{fitness}(\mathbf{w}) = \text{Train Error}(\mathbf{w})$$
$$\text{testFitness}(\mathbf{w}) = \text{Test Error}(\mathbf{w})$$

```
def calculateFitness(self, population):
        """Returns fitness array for the population"""
        # return np.mean(population ** 2, axis=1) ** 0.5

        errorList = [getErrors(indi, False) for indi in population.tolist()]
        return (
            np.array([error[0] for error in errorList]),
            np.array([error[1] for error in errorList]),
        )
```

Termination

```
for iteration in range(steps):
      ----
      ----
```

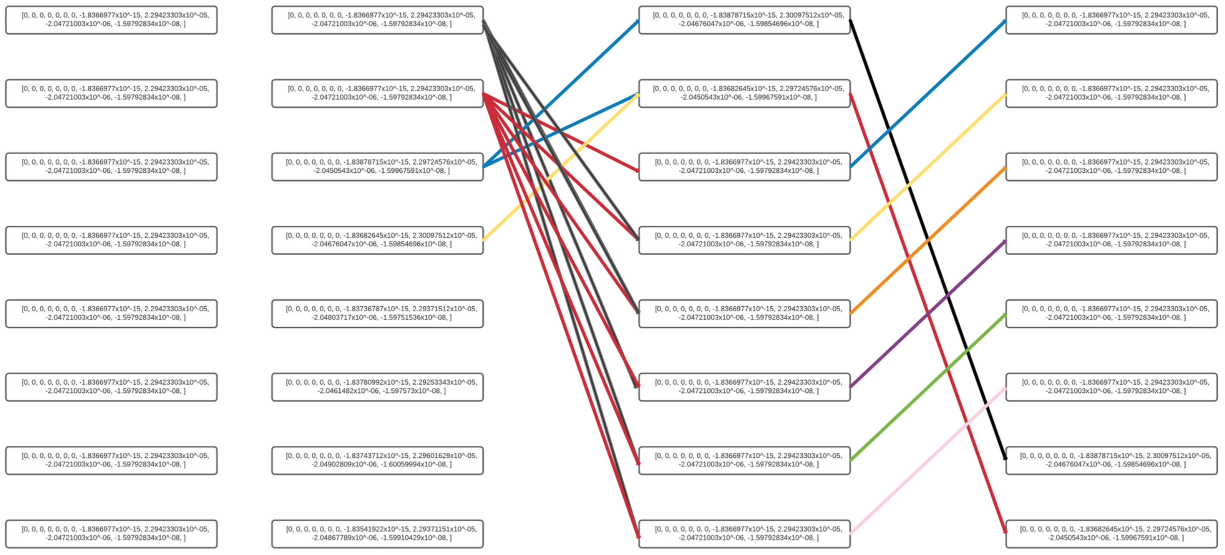
There is no definite termination condition defined for the genetic algorithm, rather it's the choice of the programmer. The termination conditions we used were the number of generations (as `steps` — an hyper-parameter) due to the limited number of queries.

Sample Iterations

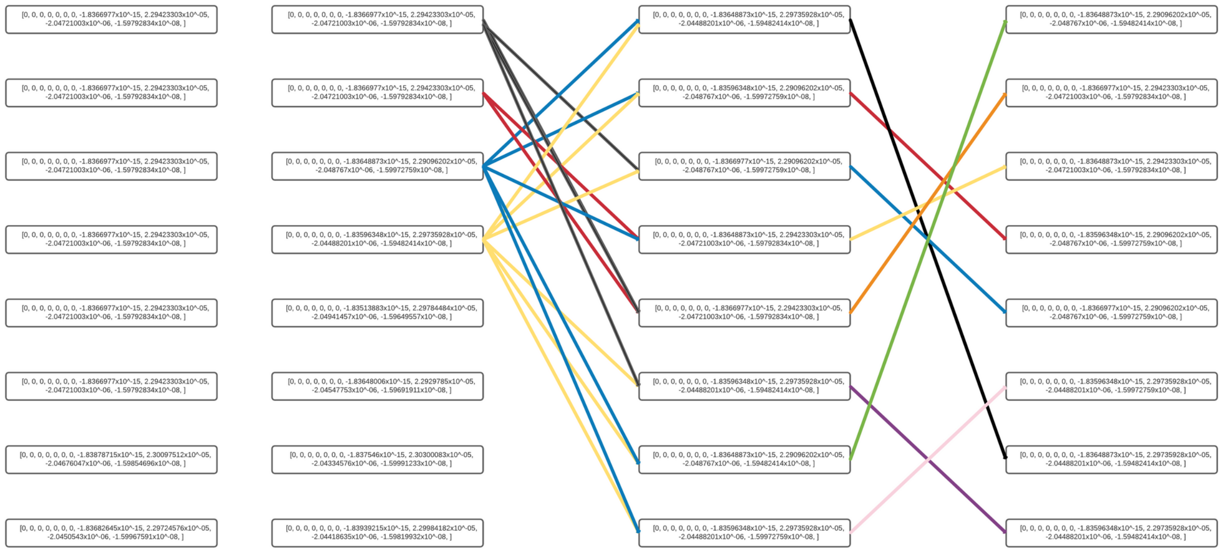
GIven 3 images (below) are 3 iterations of the genetic algorithm with the certain chosen vectors as the population. All our major generations have been documented as `.txt` files under the folder `generations`.

Note: When this following was generated selection was being done within crossover (check Crossover Section for more details).

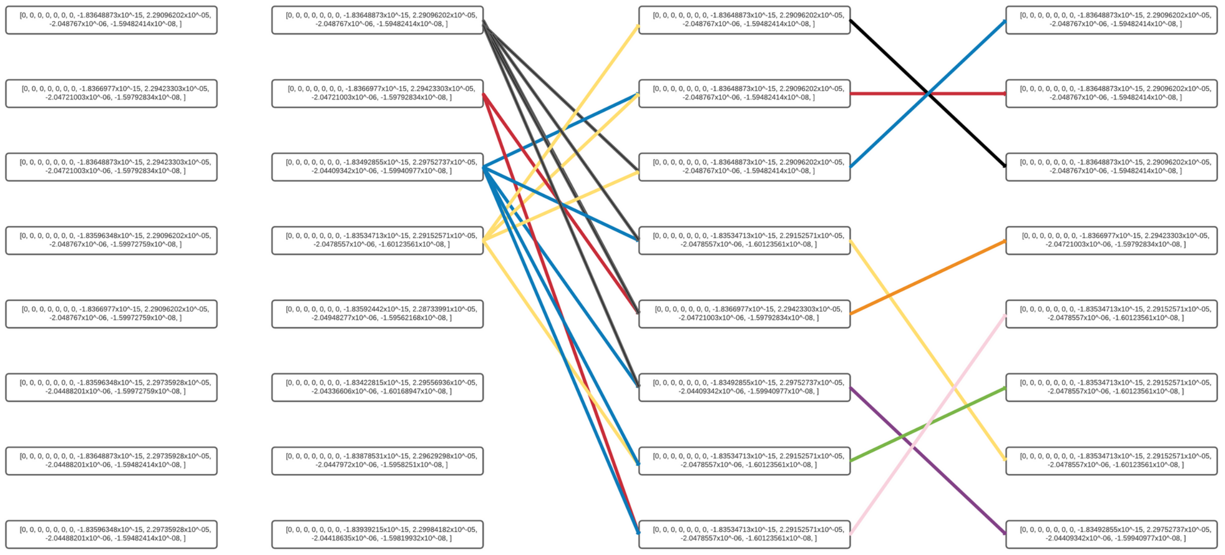
Generation 1


[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83878715x10^-15, 2.29724576x10^-05, -2.0450543x10^-06, -1.59967591x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83682645x10^-15, 2.30097512x10^-05, -2.04676047x10^-06, -1.59854696x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83736787x10^-15, 2.29371512x10^-05, -2.04803717x10^-06, -1.59751536x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83780992x10^-15, 2.29253343x10^-05, -2.0461482x10^-06, -1.597573x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83743712x10^-15, 2.29601629x10^-05, -2.04902809x10^-06, -1.60059994x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83541922x10^-15, 2.29371151x10^-05, -2.04867789x10^-06, -1.59910429x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83878715x10^-15, 2.30097512x10^-05, -2.04676047x10^-06, -1.59854696x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83682645x10^-15, 2.29724576x10^-05, -2.0450543x10^-06, -1.59967591x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83878715x10^-15, 2.30097512x10^-05, -2.04676047x10^-06, -1.59854696x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83682645x10^-15, 2.29724576x10^-05, -2.0450543x10^-06, -1.59967591x10^-08,]

Generation 2


[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83878715x10^-15, 2.30097512x10^-05, -2.04676047x10^-06, -1.59854696x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83682645x10^-15, 2.29724576x10^-05, -2.0450543x10^-06, -1.59967591x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83513883x10^-15, 2.29784484x10^-05, -2.04941457x10^-06, -1.59649557x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648006x10^-15, 2.2929785x10^-05, -2.04547753x10^-06, -1.59691911x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.837546x10^-15, 2.30300083x10^-05, -2.04334576x10^-06, -1.59991233x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83939215x10^-15, 2.29984182x10^-05, -2.04418635x10^-06, -1.59819932x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]

Generation 3


[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59972759x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83596348x10^-15, 2.29735928x10^-05, -2.04488201x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83492855x10^-15, 2.29752737x10^-05, -2.04409342x10^-06, -1.59940977x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83592442x10^-15, 2.28733991x10^-05, -2.04948277x10^-06, -1.59562168x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83422815x10^-15, 2.29556936x10^-05, -2.04336606x10^-06, -1.60168947x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83878531x10^-15, 2.29629298x10^-05, -2.0447972x10^-06, -1.5958251x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83939215x10^-15, 2.29984182x10^-05, -2.04418635x10^-06, -1.59819932x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83492855x10^-15, 2.29752737x10^-05, -2.04409342x10^-06, -1.59940977x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83648873x10^-15, 2.29096202x10^-05, -2.048767x10^-06, -1.59482414x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.8366977x10^-15, 2.29423303x10^-05, -2.04721003x10^-06, -1.59792834x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83534713x10^-15, 2.29152571x10^-05, -2.0478557x10^-06, -1.60123561x10^-08,]
[0, 0, 0, 0, 0, 0, 0, -1.83492855x10^-15, 2.29752737x10^-05, -2.04409342x10^-06, -1.59940977x10^-08,]

Hyper-parameters

Values were changed multiple times until the perfect value was found. All the hyper-parameters are listed as below:

- `POPULATION_SIZE` = 9
- `MUTATION_PROBABILITY` = 0.5
- `BETA` = 0.75

The names of the above parameters are self-explanatory in terms of their functionality. Necessity of β has been furthered explored in Crossover Section.

Results

The errors of the best vector we have submitted are:

$$\text{Train Error : } [1.6251245e+11]$$
$$\text{Test Error : } [1.60388676e+11]$$

Footnotes

Below presented is a rough sketch of our transition from given overfit vector to the best submitted vector.

Removal of noise

Increased both the errors severely since it was no longer overfit.

Fitting on train error

Population is filled with underfit array and train error is used for fitness and is run for generations in batches of 10 with hyper-parameters changed in between.

Hyper-parameter Tuning

Changing the values after every 10 generations was done manually based on the pattern in train and test error for complete population.

Why does it work?

- Since the noise which resulted in overfitting was eliminated, it made the weights underfit the data.
- The weights were selected based on the training error, whereas the hyper-parameter tuning was done based on the validation error.
- This resulted in a good vector which performed well on unseen data.

Tricks

Manual testing

- For finding the correct values for the noise weights, we did manual testing due to the limitation of the queries.
- For checking the variation in the error for each vector to determine one of the hyper-parameters, we checked out the errors by tweaking the values of the weights.

Manual Comparison

- We also compared our vectors manually to check approximate the direction of the minima is.