Genetic Algorithms | Machine Data Learning

Team 35:

Anvita Reddy :2019115009

Stella Sravanthi: 2019101101

Genetic algorithm (GA) is a **metaheuristic** inspired by the process of natural selection that belongs to the larger class of **evolutionary algorithms.**

Main methods involved in finding the fittest:

1) Initial Population
2) Fitness
3) Selection
4) Cross Over
5) Mutation and calculate fitness for new population

Finally, Repeat step-3 to step-8 to finalise the best population.

Summary

Step 1:Initial Population

Initial generation is created by mutating the given overfit vector.
Generation is of around **pop=10** population each of 11 sized vectors.
We then used numpy.random.randn() for creating a vector of size 11 according to standard gaussian distribution and then multiplied it with 10^-17 and stored it in vector **a**.
Later, added this particular vector generated to overfit vector value and returned.
And the starting vectors of 1st generation **a** are stored in multi-dimensional array **b.**

```
def initial():
   a=np.zeros(11)
   vec=np.random.randn(1,11)*((10**-17))
   a=overfit+vec[0]
   a=a.tolist()
   return a
```

Step 2:Getting Errors for Initial Population and Fitness:

Here as soon as each of the 10 initial population vectors are generated and stored in b, they are sent for querying and obtain errors. Based on which their fitness is calculated, for their further progress into the algorithm.

```
b=np.zeros((pop_size,11))
for i in range(pop_size):
        b[i]=initial()
        err=server.get_errors(SECRET_KEY,(b[i].tolist()))
        error[i]=((err[0]+(err[1]-err[0]))+(err[1]-(err[1]-err[0])))
```

Fitness

Fitness is calculated population wise as a combination of train and validation errors.
Errors are obtained using get_errors function and stored in err.
err=server.get_errors(SECRET_KEY,(b[i].tolist()))

- ➢ Now to keep train and validation together and balance both we came up with a simple and perfect fitness calculator that decreases the error and difference as well.
- ➢ This was done to not allow great difference between validation and train error and ensuring higher value of validation error than train

TrainError+(ValidationError-TrainError)+(ValidationError-(ValidationError-TrainError))

```
error[i]=((err[0]+(err[1]-err[0]))+(err[1]-(err[1]-err[0])))
```

Step3: Parent selection

It involved sorting population according to its fitness score and mating poolsize. Here the process we chose for selection was based on the standard deviation of the fitness values in the whole of parent generation. Here **mean, std** has the mean, standard deviation respectively of fitness values stored in **error** for the parent generation's population. Based off of which, a loop is run where each of the fitness values is modified to be an element of a Gaussian Distribution. And all the modified values which are now in the form of a distribution are stored in the **selected** array. Now the values in the **selected** array are sorted in ascending order, since we want least fitness, and we derive the indexes(ind1,ind2) of numbers in **selected** array, and extract the corresponding parents from **b** array to generate a new child.

```
for i in range(pop_size):
                print("Child Number:",i+1)
                children[i]=selection(error,b)
```

```
def selection(error,b):
        selected=np.zeros(pop)
        mean=np.mean(error)
         std=np.std(error)
         for i in range(pop):
            selected[i]=(error[i]-mean)/(std+10**-17)
        [ind1,ind2]=sorted(range(len(selected)), key=lambda i: selected[i],reverse=True)[-2:]
        print("Parent 1:",b[ind1])
        print("parent 2",b[ind2])
```

Step4: Crossover

It is coded in the selection function itself which finally returns the children for every population. Here we used the **Single Point Crossover** function itself, where we randomly generate a number between (0,10)(index) and store it in

chec. Now copies(pp1,pp2) of the parent selected are made since we don't want to change the original **b** array itself. And then we generated another random number(d) between (0,1) to determine the order in which the parents to crossover be selected. Now based on the value of **d,** we define parents' order as d=0, pp1 comes first and d=1, ten pp2 comes first. After which until the point of crossover **chec**, we copy the value from first parent then the remaining (11-**chec)** value from second parent. Which is stored in arr3 and returned.

selection() continued:

```
arr3=np.zeros(11)
   chec=np.random.randint(0,11)
    pp1=b[ind1]
   pp2=b[ind2]
   if(d==0):
          temp=pp1
          pp1=pp2
          pp2=temp
   for i in range(11):
          if ( i<=chec):
                arr3[i]=pp2[i]
          else:
                arr3[i]=pp1[i]
         return(arr3)
```

Step5: Mutation function

For each of the generated children after selection and crossover, we mutate certain of its indexes based off of certain probability. We randomly generate a float number between 0,1 and compare it with the probability sent. If the random number is less than probability, then we again generate a random number and divide it by 1000 and add a one **(1+np.random.randn()/1000)**. Now this value is multiplied to the value in the child vector. This may cause the original value to increase or decrease due to randomisation. But if the

number does not fall within the range(-10,10), we decrease it by multiplying with **(1- abs(np.random.randn()/1000)),** which is surely less than 1 and serves the purpose.

```
for i in range(pop_size):
                    mutated[i]=mutate(children[i],0.2)
```

```
def mutate(child,prob):
          mutated_indices=[]
          for i in range(11):
          if(np.random.rand()<prob):
              mutated_indices.append(i+1)
              child[i]=child[i]*(1+np.random.randn()/1000)
          if(abs(child[i])>=10):
               child[i]=child[i]*(1- abs(np.random.randn()/1000))
          print("Weight Indices that mutated:", mutated_indices)
          return child
```

Step6: Child Fitness:

```
err=server.get_errors(SECRET_KEY,(mutated[i].tolist()))
e2[i][0]=err[0]
e2[i][1]=err[1]
child_error[i]=((err[0]+(err[1]-err[0]))+(err[1]-(err[1]-err[0])))
```

Here using the previously used fitness function again, we generate fitness values for each of the child vectors in this generation and store in array **child_error,** for future reference.

Step 7:Best vectors in this generation

```
[ind1,ind2,ind3,ind4,ind5]=sorted(range(len(error)), key=lambda i: error[i],reverse=True)[-5:]

best_from_parent=[ind1,ind2,ind3,ind4,ind5]
```

```
[ind6,ind7,ind8,ind9,ind10]=sorted(range(len(child_error)), key=lambda i: child_error[i],reverse=True)[-5:]

best_from_children=[ind6,ind7,ind8,ind9,ind10]
```

Here, based off of the fitness values generated from errors for both the parents and children, we select the best of them to send further into the next generation. We sorted the fitness value of the parents stored in **errors** array and chose the best 5 from them. And stored their indexes in **best_from_parent** array. We sorted the fitness value of the children stored in **child_errors** array and chose the best 5 from them. And stored their indexes in **best_from_children** array.

Step 8:Best vectors and their fitness for next generation

```
for i in range(0,5):
        final_generation[i]=b[best_from_parent[i]]
for i in range(5,10):
        final_generation[i]=mutated[best_from_children[5-i]]
b=np.copy(final_generation)
```

```
for i in range(0,5):
        final_errors[i]=error[best_from_parent[i]]
for i in range(5,10):
        final_errors[i]=child_error[best_from_children[5-i]]
error=np.copy(final_errors)
```

Using the previously stored indexes, we copy the best parents and children vectors into **final_generation** array. Similarly using the indexes, we copied the fitnesses from parents and children into **final_errors** array. After printing them, we copied them back into the original **b** array and **errors** array. We

then repeated all the steps from 3 to 8, excluding the initial population generation from overfit vectors. This was repeated for the next 9 times to fill all generations.

Thus new generation is created having a new fitness value

The best vector from this new generation is identified along with fitness.

Diagrams for Consecutive Iterations

Iteration 1:

Initial Vector 1	[1.31915032e-17 -1.45800014e-12 -2.28963867e-13 4.62010698e-11 -1.75214815e-10 -1.82827873e-15 8.50657850e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759873e-10]
Initial Vector 2	[5.14330011e-19 -1.45800111e-12 -2.28991950e-13 4.62010695e-11 -1.75214812e-10 -1.82214380e-15 8.55248111e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759858e-10]
Initial Vector 3	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Initial Vector 4	[-1.26014348e-17 -1.45798762e-12 -2.28987449e-13 4.62010585e-11 -1.75214800e-10 -1.84072384e-15 8.50516680e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759854e-10]
Initial Vector 5	[5.23740920e-19 -1.45799824e-12 -2.28973751e-13 4.62010733e-11 -1.75214813e-10 -1.83250136e-15 8.74931932e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759848e-10]
Initial Vector 6	[1.27736865e-17 -1.45799846e-12 -2.28987254e-13 4.62010666e-11 -1.75214831e-10 -1.84914393e-15 8.44760374e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Initial Vector 7	[1.84378772e-18 -1.45799105e-12 -2.28976709e-13 4.62010699e-11 -1.75214821e-10 -1.83258361e-15 8.43756042e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759868e-10]
Initial Vector 8	[3.44202816e-18 -1.45798786e-12 -2.28976444e-13 4.62010668e-11 -1.75214821e-10 -1.83089071e-15 8.42062362e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759869e-10
Initial Vector 9	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Initial Vector 10	[2.24471228e-17 -1.45799654e-12 -2.28960904e-13 4.62010879e-11 -1.75214819e-10 -1.84345434e-15 8.74005158e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759840e-10

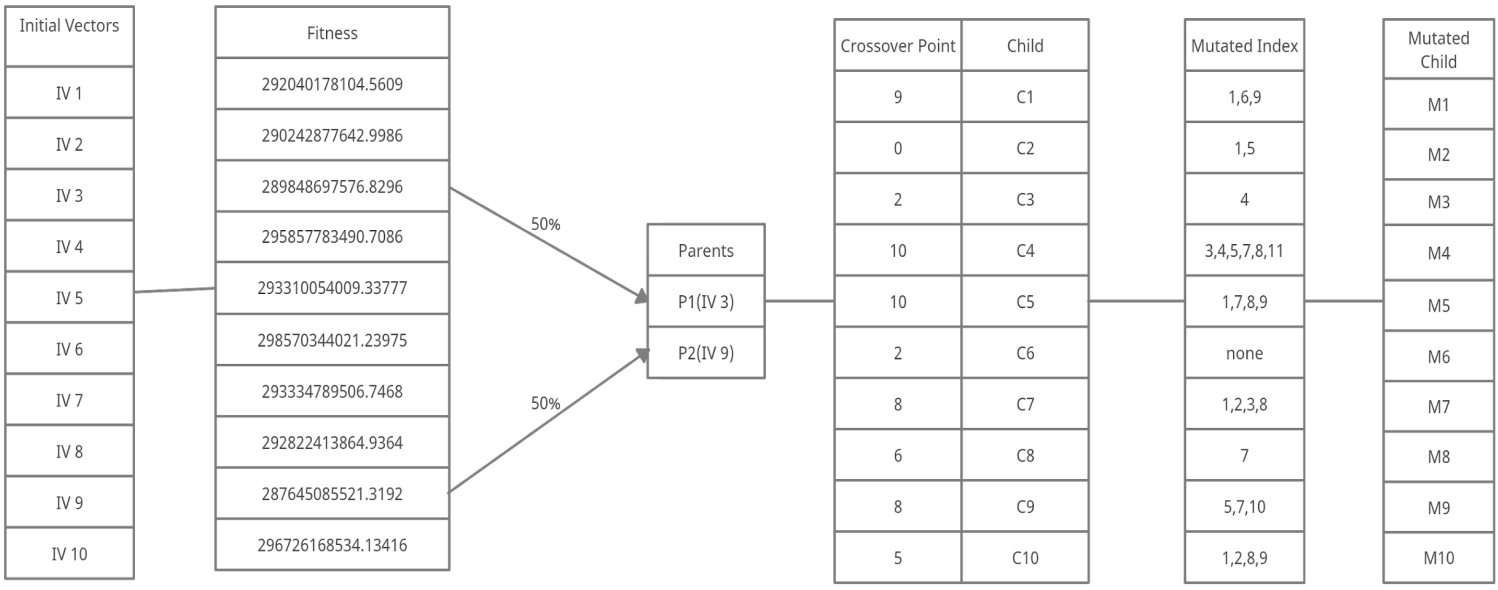
Initial Vectors marked as IV 1-10

Child 1 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05
Child 2 After Crossover	[4.49108363e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 3 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 4 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 5 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 6 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 7 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 8 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 9 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 10 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]

Crossovered Parents to generate Children as C 1-10

Mutated Children Vectors as M 1-10

Mutated Child 1	[-5.15062546e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04845906e-06 -1.59792834e-08 9.83759874e-10]
Mutated Child 2	[4.48748121e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 3	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62618556e-11 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 4	[4.49108363e-18 -1.45798073e-12 -2.28841738e-13 4.62203363e-11 -1.75371538e-10 -1.82077314e-15 8.57899357e-16 2.29268862e-05
Mutated child 5	[-5.13830698e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04846669e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 6	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05
Mutated Child 7	[-5.14454904e-18 -1.45780761e-12 -2.28865640e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29900348e-05 -2.04721003e-06 --08 9.83759874e-10]
Mutated Child 8	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.59546896e-16 2.29423303e-05
Mutated Child 9	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75344654e-10 -1.81292005e-15 8.53222738e-16 2.29423303e-05
Mutated Child 10	[4.48859183e-18 -1.45623880e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.54519834e-16 2.29442165e-05 -2.04744453e-06 -1.59792834e-08 9.83759865e-10]



Iteration 2:

Initial Vector 1	[1.31915032e-17 -1.45800014e-12 -2.28963867e-13 4.62010698e-11 -1.75214815e-10 -1.82827873e-15 8.50657850e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759873e-10]
Initial Vector 2	[5.14330011e-19 -1.45800111e-12 -2.28991950e-13 4.62010695e-11 -1.75214812e-10 -1.82214380e-15 8.55248111e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759858e-10]
Initial Vector 3	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Initial Vector 4	[-1.26014348e-17 -1.45798762e-12 -2.28987449e-13 4.62010585e-11 -1.75214800e-10 -1.84072384e-15 8.50516680e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759854e-10]
Initial Vector 5	[5.23740920e-19 -1.45799824e-12 -2.28973751e-13 4.62010733e-11 -1.75214813e-10 -1.83250136e-15 8.74931932e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759848e-10]
Initial Vector 6	[1.27736865e-17 -1.45799846e-12 -2.28987254e-13 4.62010666e-11 -1.75214831e-10 -1.84914393e-15 8.44760374e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Initial Vector 7	[1.84378772e-18 -1.45799105e-12 -2.28976709e-13 4.62010699e-11 -1.75214821e-10 -1.83258361e-15 8.43756042e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759868e-10]
Initial Vector 8	[3.44202816e-18 -1.45798786e-12 -2.28976444e-13 4.62010668e-11 -1.75214821e-10 -1.83089071e-15 8.42062362e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759869e-10
Initial Vector 9	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Initial Vector 10	[2.24471228e-17 -1.45799654e-12 -2.28960904e-13 4.62010879e-11 -1.75214819e-10 -1.84345434e-15 8.74005158e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759840e-10

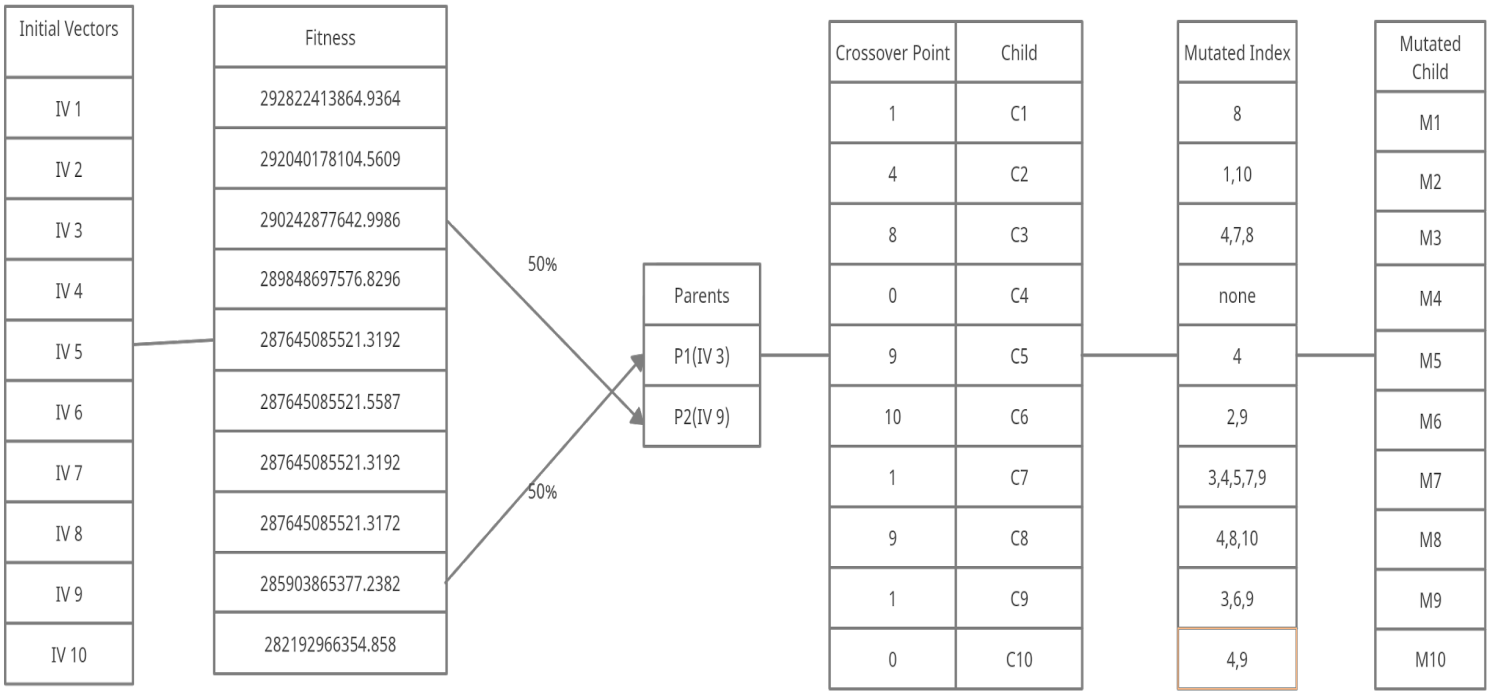
Initial Vectors marked as IV 1-10

Child 1 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05
Child 2 After Crossover	[4.49108363e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 3 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 4 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 5 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 6 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 7 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 8 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 9 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 10 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]

Crossovered Parents to generate Children as C 1-10

Mutated Children Vectors as M 1-10

Mutated Child 1	[-5.15062546e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04845906e-06 -1.59792834e-08 9.83759874e-10]
Mutated Child 2	[4.48748121e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 3	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62618556e-11 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 4	[4.49108363e-18 -1.45798073e-12 -2.28841738e-13 4.62203363e-11 -1.75371538e-10 -1.82077314e-15 8.57899357e-16 2.29268862e-05
Mutated child 5	[-5.13830698e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04846669e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 6	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05
Mutated Child 7	[-5.14454904e-18 -1.45780761e-12 -2.28865640e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29900348e-05 -2.04721003e-06 --08 9.83759874e-10]
Mutated Child 8	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.59546896e-16 2.29423303e-05
Mutated Child 9	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75344654e-10 -1.81292005e-15 8.53222738e-16 2.29423303e-05
Mutated Child 10	[4.48859183e-18 -1.45623880e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.54519834e-16 2.29442165e-05 -2.04744453e-06 -1.59792834e-08 9.83759865e-10]



Iteration 3:

Initial Vector 1	[1.31915032e-17 -1.45800014e-12 -2.28963867e-13 4.62010698e-11 -1.75214815e-10 -1.82827873e-15 8.50657850e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759873e-10]
Initial Vector 2	[5.14330011e-19 -1.45800111e-12 -2.28991950e-13 4.62010695e-11 -1.75214812e-10 -1.82214380e-15 8.55248111e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759858e-10]
Initial Vector 3	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Initial Vector 4	[-1.26014348e-17 -1.45798762e-12 -2.28987449e-13 4.62010585e-11 -1.75214800e-10 -1.84072384e-15 8.50516680e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759854e-10]
Initial Vector 5	[5.23740920e-19 -1.45799824e-12 -2.28973751e-13 4.62010733e-11 -1.75214813e-10 -1.83250136e-15 8.74931932e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759848e-10]
Initial Vector 6	[1.27736865e-17 -1.45799846e-12 -2.28987254e-13 4.62010666e-11 -1.75214831e-10 -1.84914393e-15 8.44760374e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Initial Vector 7	[1.84378772e-18 -1.45799105e-12 -2.28976709e-13 4.62010699e-11 -1.75214821e-10 -1.83258361e-15 8.43756042e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759868e-10]
Initial Vector 8	[3.44202816e-18 -1.45798786e-12 -2.28976444e-13 4.62010668e-11 -1.75214821e-10 -1.83089071e-15 8.42062362e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759869e-10
Initial Vector 9	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Initial Vector 10	[2.24471228e-17 -1.45799654e-12 -2.28960904e-13 4.62010879e-11 -1.75214819e-10 -1.84345434e-15 8.74005158e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759840e-10

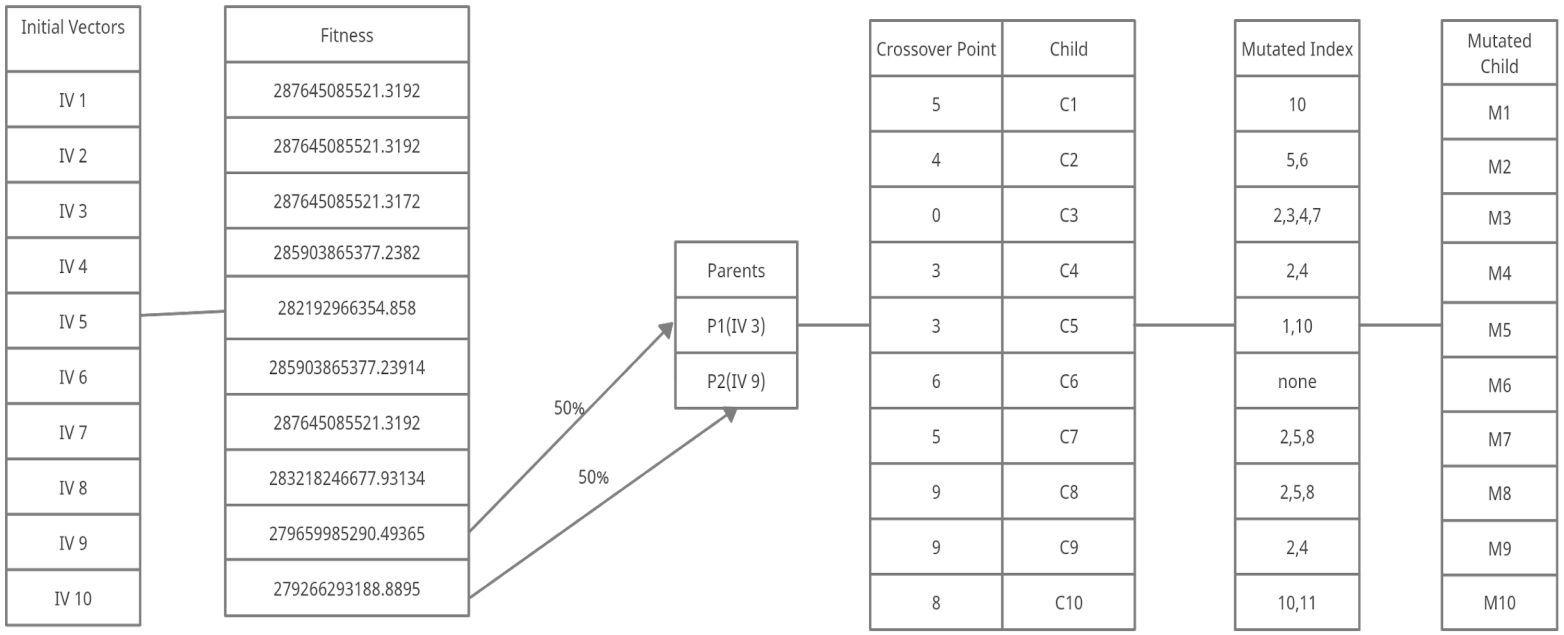
Initial Vectors marked as IV 1-10

Child 1 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05
Child 2 After Crossover	[4.49108363e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 3 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 4 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 5 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 6 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 7 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 8 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.58388170e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Child 9 After Crossover	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759874e-10]
Child 10 After Crossover	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.54519834e-16 2.29423303e-05 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]

Crossovered Parents to generate Children as C 1-10

Mutated Child 1	[-5.15062546e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04845906e-06 -1.59792834e-08 9.83759874e-10]
Mutated Child 2	[4.48748121e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 3	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62618556e-11 -2.04721003e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 4	[4.49108363e-18 -1.45798073e-12 -2.28841738e-13 4.62203363e-11 -1.75371538e-10 -1.82077314e-15 8.57899357e-16 2.29268862e-05
Mutated child 5	[-5.13830698e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -2.04846669e-06 -1.59792834e-08 9.83759865e-10]
Mutated Child 6	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29423303e-05
Mutated Child 7	[-5.14454904e-18 -1.45780761e-12 -2.28865640e-13 4.62010629e-11 -1.75214812e-10 -1.81292005e-15 8.54519834e-16 2.29900348e-05 -2.04721003e-06 --08 9.83759874e-10]
Mutated Child 8	[4.49108363e-18 -1.45798073e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.59546896e-16 2.29423303e-05
Mutated Child 9	[-5.14577735e-18 -1.45798164e-12 -2.28979687e-13 4.62010629e-11 -1.75344654e-10 -1.81292005e-15 8.53222738e-16 2.29423303e-05
Mutated Child 10	[4.48859183e-18 -1.45623880e-12 -2.28986110e-13 4.62010772e-11 -1.75214829e-10 -1.82077314e-15 8.54519834e-16 2.29442165e-05 -2.04744453e-06 -1.59792834e-08 9.83759865e-10]

Mutated Children as M 1-10



Hyperparameters

- **Population size:**

pop

It is the population size for each generation. This parameter is initially set to more than 10 and later as per usage, we reduced it to 7-8, since that was the required minimum and tested. Finally set to 10.This helped to have enough diversity in population and saved our daily requests. We thought it was required and sufficient for convergence

- **Crossover point:**

Chec

We have tried to use various crossovers . At first we tried Simulated binary crossover which doesn't help us to reduce error less than 10^13. Also tried random crossover for limited range.Later, the crossover point is set randomly over 0 to 10 range .By which we get a chance to create a new child irrespective of having repeated parents.

- **Mutation Probability:**

Prob

We have altered the genes with 0.2 probability. Over the assignment, we have used 0.2,0.3 and 0.8.Initially we kept probability either very high or very very low it helped to achieve a minima.But later the algorithm got stuck at that local minima itself.

Inference:

We think that, the **best vector** we could get to is:

[8.88845703e-18 -1.45797740e-12 -2.28980367e-13 4.62010792e-11
 -1.75214800e-10 -1.82846074e-15 8.72363337e-16 2.29423303e-05
 -2.04265077e-06 -1.59792834e-08 9.83759873e-10],

With Train Error:7.87059656e+10

Validation Error:1.91771943e+11

- We think of this as the best vector because the train error has increased from the initial overfit vector as it should. Since the vector earlier overfitted, the train error was considerably low and lower than validation error on unseen data. Now the train error has increased, depicting that the model has overcome overfitting.
- The other thing is that the train error is still lower than validation error, which holds from ear;ier as the model works better on seen data than unseen.
- The difference between the train and validation errors has also considerably decreased as compared to earlier, which should happen as the difference in errors must be less for a good fitting
- Lastly the validation error has decreased as compared to earlier depicting that the model is not moving towards underfitting while decreasing overfitting.

Generation #	Highest fitness	Lowest fitness
1	2.95579349e+11	2.70477909e+11
2	2.90651507e+11	2.70477909e+11
3	2.81473746e+11	2.66775807e+11
4	2.70370421e+11	2.61579681e+11

5	**2.66775807e+11**	**2.60196238e+11**
6	**2.61579681e+11**	**2.60096712e+11**
7	**2.60473356e+11**	**2.60020107e+11**
8	**2.60024423e+11**	**2.57762780e+11**
9	**2.59267943e+11**	**2.57613855e+11**
10	**2.58653833e+11**	**2.57561186e+11**

As the generation increases we can see that the fitness values are decreasing as they should in our case as it is indirectly a measure of error. And the algorithm is converging as we proceed into more generations or out given fitness functions.

Heuristics Applied:

1) As the initial vectors are overfit, we realised modifying those weights doesn't decrease the error less than 10^12 almost we initiated a population with different mutation probability ranges. But the **identical population** is obtained when we add the mutation of order **np.random.randn()/10^{-17}** to overfit vectors. We used numpy.random.randn() for creating a vector of size 11 according to standard gaussian distribution and then multiplied it with 10^-17. We chose to multiply with 10^-17 as the least number in overfit is of order 10^-16, and hence didn't want any vector to be greatly affected
2) For the fitness function we initially tris train*0.7+validation and later even decrease the value 0.7 but was of no use, since there was no convergence to a menial through fitness possible.

TrainError+(ValidationError-TrainError)+(ValidationError-(ValidationError-TrainError))

- We thought this would work as, if both Validation and Train error were high, the difference between them would be low, ultimately leading to a high fitness value.
- If both Validation and train error were low then difference would also be reasonable, ultimately leading to low overall fitness value

3) For better comparison of the fitness value generated using above formula, we decided to use their distributed value according to Gaussian distribution. We included a factor of 10^-17 to avoid any chance of division with zero.

4) We chose crossover points randomly such that, even if the same two parents were to be repeatedly selected, the children would be different on several occasions. The probability of the same two parents was also high hence randomisation of crossover point was required.

5) In order for mutation of children we used a probability factor 0.2.So, we finalised 0.2 with a notion that on an average 2 of the elements of all the vectors would be mutated.

6) For mutation we applied (np.random.randn()/1000)*current child[i]. This helped to solve the worsening performance when we have a large difference in child values. We replaced the value of 1000 with smaller and larger values, but there wasn't much change. Since all of the values were in range(10^-5,10^-16), multiplication seemed ideal considering range limit given

7) We observed selecting parents matters in getting more diverse children and best ordered child.Thus , found order of selecting parents plays an important role.So, instead of fixing a number to separate the parents group, we used previous best vectors and observed better results.

Heuristics that didn't work:

➔ **Initial vectors:**
We almost initiated a population with different mutation probabilities and ranges.

We also initiated randomly in range (-10,10). And even chose the overfit vector repeatedly as the initial population as well. But there was no progress.

➔ **Fitness function:**

From the beginning , we have applied many fitness functions.Tried different arithmetic combinations of train and validation errors.

- ➢ At first,since given overfit vector has low train error, we tried

  TrainError*0.7+ValidationError to avoid overfit problem.

- ➢ Later, the train error increased and to control that we shifted to 0.3 factor.

  TrainError*0.3+ValidationError to control the train error.

- ➢ We even tried multiplying validation error with a factor greater than 1, but it didn't even lead to decrement after the second generation.

➔ We even tried addition for the mutation of children vectors by various factors in range (0,10^-16) but multiplication seemed effective rather than addition.