GENETIC ALGORITHM

TEAM NO: 18

Sharadha Iyer 2019101048
Hasvitha Varma 2019101030

SUMMARY OF GENETIC ALGORITHM:

The genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

The stepwise explanation of our implementation is as follows:

STEP 1: GENERATE INITIAL POPULATION

- ➔ To start the genetic algorithm, we require a population of some pop_size number of individuals. These individuals are generated by mutating the vector (overfit on the training dataset) that is provided to us. This produces the starting generation for the GA.
- ➔ We have ensured that the first generation of individuals is diverse by keeping the probability of mutation factor high, that is the probability with which a gene is mutated in a chromosome is high. This is so that the algorithm does not converge to a local minima in the very beginning itself

```
# generate the population
it2 = 0
while it2 < pop_size:
    temp = np.copy(overfit_weights)
    it2 += 1
    population[it2-1] = np.copy(mutateall(temp,0.85,mutate_range))
```

STEP 2: OBTAIN ERRORS FOR THIS GENERATION

➔ Once the first generation is ready, we obtain the errors of all the individuals (parents for the next generation).

➔ This is done to determine the fitness of each individual and to decide their progress into the next generation.

```
# generate errors for each individual in the population
it2 = 0
while it2 < pop_size:
    # passing a list to the get_errors function
    temp = population[it2].tolist()
    it2 += 1
    err = client.get_errors(key, temp)
    # adding the two errors and storing in parenterror - fitness function
    tp = err[0] + 1.5*err[1]
    parenterrors[it2-1] = np.copy((err[0]+1.5*err[1]))
    # parenterrors[j] = np.copy((err[0]+err[1]))
    parenterrors1[it2-1] = np.copy((err[0]))
    parenterrors2[it2-1] = np.copy((err[1]))
```

STEP 3: CROSSOVER OF PARENT POPULATION

➔ Once the parent errors have been obtained, the parent population is sorted in the increasing order of their errors. Then the crossover step takes place.

➔ Crossovers happpen until pop_size children have been produced. The parents for crossover are chosen from a set of top select_sure parents in the parent population.

➔ The parents chosen from the previous step are sent to the crossover function which returns two children (two vectors).

➔ The crossover function returns mutated the child vectors. These 2 child vectors are then appended to the child population.

➔ If the child vector is identical to the parent vector, it is not included in the child population and that iteration is discarded.

```
child_population = np.zeros((pop_size, 11))
it2 = 0
while(new_iter < pop_size):

    # arr = crossover_select(parentprobalities)
    # TODO: Select randomly among top k parents  (For now k =10)
    arr = random.sample(range(8), 2)

    # Sending parents for crossover
    temp = crossover(population[arr[0]], population[arr[1]],mutate_range,prob_mut_cross)
    it2 += 1
    if temp[1].tolist() == population[arr[0]].tolist():
        continue
    elif temp[0].tolist() == population[arr[0]].tolist():
        continue
    elif temp[0].tolist() == population[arr[1]].tolist():
        continue
    elif temp[1].tolist() == population[arr[1]].tolist():
        continue

    for it in range(0, 2):
        arrchoices[d2(new_iter)][it]=np.copy(arr[it])

    arrposswap[d2(new_iter)]=np.copy(np.sort(temp[4]))

    arrchildren[new_iter] = np.copy(temp[2])
    arrchildrenmutated[new_iter] =  np.copy(temp[0])
    child_population[new_iter] = np.copy(temp[0])

    new_iter = new_iter + 1

    arrchildren[new_iter] = np.copy(temp[3])
    arrchildrenmutated[new_iter] = np.copy(temp[1])
    child_population[new_iter] = np.copy(temp[1])
    new_iter = new_iter + 1
```

STEP 4: OBTAIN ERRORS FOR CHILD POPULATION

➔ The errors for the child population are obtained.

➔ The child population is then sorted in the increasing order of their errors (sum of the train and validation errors obtained).

```
childerrors = np.zeros(pop_size)
childerrors1 = np.zeros(pop_size)
childerrors2  = np.zeros(pop_size)

# generate errors for each child
it2 = 0
while it2 < pop_size:
    temp = child_population[it2].tolist()
    err = client.get_errors(key, temp)
    childerrors[it2] = np.copy((err[0]+1.5*err[1]))
    childerrors1[it2] = np.copy((err[0]))
    childerrors2[it2] = np.copy((err[1]))
    it2 += 1
    arrchilderrors[it2-1]=np.copy(childerrors[it2-1])

# Sort children
childinds = np.copy(childerrors.argsort())
childerrors = np.copy(childerrors[childinds[::1]])
childerrors1 = np.copy(childerrors1[childinds[::1]])
childerrors2 = np.copy(childerrors2[childinds[::1]])
child_population = np.copy(child_population[childinds[::1]])
```

STEP 5: CREATE THE NEXT GENERATION

- ➔ Now that we have the pop_size parents and pop_size children we have to prepare a population of pop_size individuals by selecting them from the parents and children.

- ➔ The new generation will have the top select_sure number of parents and children with certainty. This is so that the best (best fitness and least error) chromosomes of both populations are advanced to the next generation. This is stored in a tempbank.

```
# now the children are sorted and stored in child and parents are
# sorted in population
# we will now create a tempbank array to store top k parents,
# top k childs and rest being sorted taking from the top
tempbankerr = np.zeros(pop_size)
tempbankerr1 = np.zeros(pop_size)
tempbankerr2 = np.zeros(pop_size)
tempbank = np.zeros((pop_size, 11))

for j in range(0, select_sure):

    #choosing the top jth parent and putting in the array
    tempbank[j] = np.copy(population[j])
    tempbankerr[j] = np.copy(parenterrors[j])
    tempbankerr1[j] = np.copy(parenterrors1[j])
    tempbankerr2[j] = np.copy(parenterrors2[j])

    #choosing the top jth child and putting it into the array
    tempbank[j+select_sure] = np.copy(child_population[j])
    tempbankerr[j+select_sure] = np.copy(childerrors[j])
    tempbankerr1[j+select_sure] = np.copy(childerrors1[j])
    tempbankerr2[j+select_sure] = np.copy(childerrors2[j])
```

➔ The parent and children population is combined into a single candidates array and the remaining pop_size - select_sure number of individuals are selected.

```
# combining parents and children into one array
# TODO: Concatenating remaining parents and children and selecting from them
candidates = np.copy(np.concatenate([population[select_sure:], child_population[select_sure:]]))
candidate_errors = np.copy(np.concatenate([parenterrors[select_sure:], childerrors[select_sure:]]))
candidate_errors1 = np.copy(np.concatenate([parenterrors1[select_sure:], childerrors1[select_sure:]]))
candidate_errors2 = np.copy(np.concatenate([parenterrors2[select_sure:], childerrors2[select_sure:]]))

# sorting all the candidates by error
candidate_errors_inds = candidate_errors.argsort()
candidate_errors = np.copy(candidate_errors[candidate_errors_inds[::1]])
candidate_errors1 = np.copy(candidate_errors1[candidate_errors_inds[::1]])
candidate_errors2 = np.copy(candidate_errors2[candidate_errors_inds[::1]])
candidates = np.copy(candidates[candidate_errors_inds[::1]])

# TODO: Select the best popsize - 2*(select_sure)
cand_iter = 0
selec2 = 2*select_sure

while(cand_iter + selec2 < pop_size):
    tempbank[cand_iter+selec2] = np.copy(candidates[cand_iter])
    tempbankerr[cand_iter+selec2] = np.copy(candidate_errors[cand_iter])
    tempbankerr1[cand_iter+selec2] = np.copy(candidate_errors1[cand_iter])
    tempbankerr2[cand_iter+selec2] = np.copy(candidate_errors2[cand_iter])
    cand_iter = cand_iter + 1
```

STEP 6: SETTING THE NEW POPULATION

➔ This is now set as the next generation of individuals for the GA. Their errors are computed and the population is sorted.

```
#now setting the next population
parenterrors = np.copy(tempbankerr)
parenterrors1 = np.copy(tempbankerr1)
parenterrors2 = np.copy(tempbankerr2)
population=np.copy(tempbank)
min_parent_error = min_error

#we will now sort before updating min_error
parenerrorsinds = parenterrors.argsort()
check_error = np.copy(parenterrors[parenerrorsinds[::1]])
parent_error = np.zeros((pop_size,3))
population = np.copy(population[parenerrorsinds[::1]])
parenterrors = np.copy(parenterrors[parenerrorsinds[::1]])
parenterrors1 = np.copy(parenterrors1[parenerrorsinds[::1]])
parenterrors2 = np.copy(parenterrors2[parenerrorsinds[::1]])
```

STEP 7: CHECK MINIMUM

➔ If the error of the fittest individual is lesser than the current minimum error, the min_error and the to_send vector are updated.

```
if(min_error == -1 or parenterrors[0] < min_error ):
    nochange=0
    to_send = np.copy(population[0])
    min_error = np.copy(parenterrors[0])
    min_error1 =  np.copy(parenterrors1[0])
    min_error2 = np.copy(parenterrors2[0])

else:
    print("no improvement :(")
    nochange = nochange + 1
```

STEP 8: REPEAT STEPS 3 TO 7

- ➔ The Genetic Algorithm is repeated (Step 3 - Step 7) for iter number of iterations (this was done considering the requests that can be made to the server in order to obtain the errors for the different vectors was limited)

DIAGRAMS

The diagrams representing the 3 iterations of the genetic algorithm are as follows.

Each diagram is accompanied by tables that contain their own set of Population, Children, and Mutation vectors which are indexed and shown along with it. The diagram makes use of these indices.

NOTE: *The DIAGRAMS were created with the code that was run with the specified sample values only for demonstration purposes and may not be used in the final submission.*

Table details:

Table 1: this table shows the following:

- ➔ **Col 1:** Indices of the initial population (P0,P1,P2,P3...)
- ➔ **Col 2:** Individuals of the populations (Vectors)
- ➔ **Col 3:** Errors corresponding to each individual in the population

Table 2: this table shows the following:

- ➔ **Col 1:** Indices of the children population produced (C0,C1,C2,C3...)
- ➔ **Col 2:** The Vectors selected for crossover (That is the parents from which the children have been produced)
- ➔ **Col 3:** The indices at which the parent vector genes are swapped.
- ➔ **Col 4:** The child vectors produced after applying the crossover

Table 3: this table shows the following:

- ➔ **Col 1:** Indices of the mutated children (M0,M1,M2, M3...)
- ➔ **Col 2:** Mutated child vectors after applying mutation
- ➔ **Col 3:** Errors corresponding to each mutated child

DIAGRAM 1

Table 1:

Index	Population	Population Errors
P0	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.90477420e-11 -1.85221708e-10 -1.80586373e-15 9.13468881e-16 2.12394891e-05 -1.87585566e-06 -1.69252514e-08 9.49703462e-10]]	2.84947e+11
P1	[[0.00000000e+00 -1.41532326e-12 -2.28980078e-13 4.82729262e-11 -1.68898185e-10 -2.00252333e-15 9.32306686e-16 2.07956883e-05 -2.08559424e-06 -1.74285109e-08 1.07347650e-09]]	7.34539e+11
P2	[[0.00000000e+00 -1.55095294e-12 -2.24198774e-13 4.38137376e-11 -1.79371080e-10 -1.83669770e-15 8.43144249e-16 2.13310957e-05 -2.07973302e-06 -1.60858646e-08 9.98214034e-10]]	8.88923e+11
P3	[[0.00000000e+00 -1.57703128e-12 -2.50642669e-13 4.31572045e-11 -1.75760959e-10 -1.67638957e-15 8.04457298e-16 2.42204795e-05 -2.04721003e-06 -1.61690456e-08 1.01297625e-09]]	1.85282e+12
P4	[[0.00000000e+00 -1.50771694e-12 -2.28980078e-13 4.47154413e-11 -1.75214813e-10 -1.70670425e-15 8.44237385e-16 2.42156587e-05 -1.94303701e-06 -1.69516071e-08 9.98214034e-10]]	3.25199e+12
P5	[[0.00000000e+00 -1.34406713e-12 -2.28980078e-13 4.62790844e-11 -1.59081903e-10 -1.68225163e-15 7.86564015e-16 2.32246844e-05 -2.10655715e-06 -1.59792834e-08 9.50035159e-10]]	4.76263e+12
P6	[[0.00000000e+00 -1.57265148e-12 -2.10121193e-13 4.84294651e-11 -1.72635270e-10 -1.87042903e-15 8.15553621e-16 2.38020466e-05 -1.85213805e-06 -1.67683456e-08 9.96827826e-10]]	9.1423e+12
P7	[[0.00000000e+00 -1.57132179e-12 -2.36825739e-13 4.52159788e-11 -1.82919172e-10 -1.99413849e-15 8.60052927e-16 2.28346157e-05 -2.22157045e-06 -1.69904718e-08 9.98214034e-10]]	1.10976e+13
P8	[[0.00000000e+00 -1.49522256e-12 -2.45701333e-13 4.90307233e-11 -1.75214813e-10 -1.88806959e-15 8.29505237e-16 2.29423303e-05 -2.24207639e-06 -1.70120701e-08 9.94442634e-10]]	1.35742e+13
P9	[[0.00000000e+00 -1.54218838e-12 -2.37385160e-13 4.58031132e-11 -1.67622395e-10 -1.69172280e-15 8.25752043e-16 2.29423303e-05 -2.18520293e-06 -1.57562231e-08 9.35799547e-10]]	1.46715e+13

Table 2:

Index	Parents	Crossover positions	Children

C0	[['P0' 'P1']]	[[1. 2. 6. 7. 9.]]	[[0.00000000e+00 -1.41532326e-12 -2.28980078e-13 4.90477420e-11
			  -1.85221708e-10 -1.80586373e-15 9.32306686e-16 2.07956883e-05
			  -1.87585566e-06 -1.74285109e-08 9.49703462e-10]]

C1	[['P0' 'P1']]	[[1. 2. 6. 7. 9.]]	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.82729262e-11
			  -1.68898185e-10 -2.00252333e-15 9.13468881e-16 2.12394891e-05
			  -2.08559424e-06 -1.69252514e-08 1.07347650e-09]]

C2	[['P0' 'P7']]	[[0. 5. 7. 8. 10.]]	[[0.00000000e+00 -1.57265148e-12 -2.10121193e-13 4.84294651e-11
			  -1.72635270e-10 -1.99413849e-15 8.15553621e-16 2.28346157e-05
			  -2.22157045e-06 -1.67683456e-08 9.98214034e-10]]

C3	[['P0' 'P7']]	[[0. 5. 7. 8. 10.]]	[[0.00000000e+00 -1.57132179e-12 -2.36825739e-13 4.52159788e-11
			  -1.82919172e-10 -1.87042903e-15 8.60052927e-16 2.38020466e-05
			  -1.85213805e-06 -1.69904718e-08 9.96827826e-10]]

C4	[['P0' 'P6']]	[[4. 5. 8. 9. 10.]]	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.90477420e-11
			  -1.72635270e-10 -1.87042903e-15 9.13468881e-16 2.12394891e-05
			  -1.85213805e-06 -1.67683456e-08 9.96827826e-10]]

C5	[['P0' 'P6']]	[[4. 5. 8. 9. 10.]]	[[0.00000000e+00 -1.57265148e-12 -2.10121193e-13 4.84294651e-11
			  -1.85221708e-10 -1.80586373e-15 8.15553621e-16 2.38020466e-05
			  -1.87585566e-06 -1.69252514e-08 9.49703462e-10]]

C6	[['P0' 'P0']]	[[2. 3. 4. 9. 10.]]	[[0.00000000e+00 -1.55095294e-12 -2.08934108e-13 4.90477420e-11
			  -1.85221708e-10 -1.83669770e-15 8.43144249e-16 2.13310957e-05
			  -2.07973302e-06 -1.69252514e-08 9.49703462e-10]]

C7	[['P0' 'P0']]	[[2. 3. 4. 9. 10.]]	[[0.00000000e+00 -1.52862353e-12 -2.24198774e-13 4.38137376e-11
			  -1.79371080e-10 -1.80586373e-15 9.13468881e-16 2.12394891e-05
			  -1.87585566e-06 -1.60858646e-08 9.98214034e-10]]

C8	[['P0' 'P4']]	[[1. 2. 7. 8. 9.]]	[[0.00000000e+00 -1.50771694e-12 -2.28980078e-13 4.52159788e-11
			  -1.82919172e-10 -1.99413849e-15 8.60052927e-16 2.42156587e-05
			  -1.94303701e-06 -1.69516071e-08 9.98214034e-10]]

C9	[['P0' 'P4']]	[[1. 2. 7. 8. 9.]]	[[0.00000000e+00 -1.57132179e-12 -2.36825739e-13 4.47154413e-11
			  -1.75214813e-10 -1.70670425e-15 8.44237385e-16 2.28346157e-05
			  -2.22157045e-06 -1.69904718e-08 9.98214034e-10]]

Table 3:

Index	Mutated Children	Mutated Children Errors

M0	[[0.00000000e+00 -1.34214696e-12 -2.15811626e-13 4.44360117e-11
	  -1.72425873e-10 -1.78391498e-15 9.23806089e-16 2.27885706e-05
	  -1.75153966e-06 -1.68697667e-08 8.86742734e-10]]
		3.11363e+11

M1	[[0.00000000e+00 -1.42816774e-12 -1.99869330e-13 4.96081677e-11
	  -1.84567258e-10 -1.86946903e-15 9.57015428e-16 2.03514169e-05
	  -2.20186034e-06 -1.74718813e-08 1.07347650e-09]]
		2.18763e+12

M2	[[0.00000000e+00 -1.48300094e-12 -2.01786273e-13 5.11026428e-11
	  -1.72635270e-10 -1.99413849e-15 8.73138513e-16 2.08837244e-05
	  -2.06867722e-06 -1.67683456e-08 1.05982796e-09]]
		1.12419e+12

M3	[[0.00000000e+00 -1.52505571e-12 -2.58691390e-13 4.95523316e-11
	  -1.82919172e-10 -1.83276878e-15 9.22288149e-16 2.46316942e-05
	  -1.93947152e-06 -1.58344218e-08 9.96827826e-10]]
		7.06965e+12

M4	[[0.00000000e+00 -1.66472079e-12 -2.04466716e-13 4.78220342e-11
	  -1.59970473e-10 -1.80159205e-15 8.71014711e-16 1.91512466e-05
	  -1.85213805e-06 -1.71672584e-08 9.96827826e-10]]
		1.9664e+12

M5	[[0.00000000e+00 -1.53692780e-12 -2.17754825e-13 4.84153039e-11
	  -1.90740058e-10 -1.95780476e-15 7.36432243e-16 2.58092130e-05
	  -1.69320983e-06 -1.63166332e-08 9.49703462e-10]]
		2.20093e+13

M6	[[0.00000000e+00 -1.60043842e-12 -2.20869787e-13 4.90477420e-11
	  -1.68027453e-10 -1.97223778e-15 8.09986689e-16 2.28669770e-05
	  -2.03173992e-06 -1.72516689e-08 9.80675328e-10]]
		1.55654e+12

M7	[[0.00000000e+00 -1.52862353e-12 -2.16275727e-13 4.18761415e-11
	  -1.76153012e-10 -1.85358334e-15 9.73198889e-16 2.31012154e-05
	  -1.77606491e-06 -1.60858646e-08 9.98214034e-10]]
		2.06229e+13

M8	[[0.00000000e+00 -1.63323139e-12 -2.50949411e-13 4.68521916e-11
	  -1.92198708e-10 -1.99413849e-15 8.37898914e-16 2.66276689e-05
	  -1.87758618e-06 -1.56762599e-08 9.01058478e-10]]
		8.29955e+11

M9	[[0.00000000e+00 -1.47361088e-12 -2.45841733e-13 4.65037336e-11
	  -1.75214813e-10 -1.60281988e-15 8.78367033e-16 2.16754254e-05
	  -2.04989862e-06 -1.73980366e-08 9.89721172e-10]]
		1.99966e+12

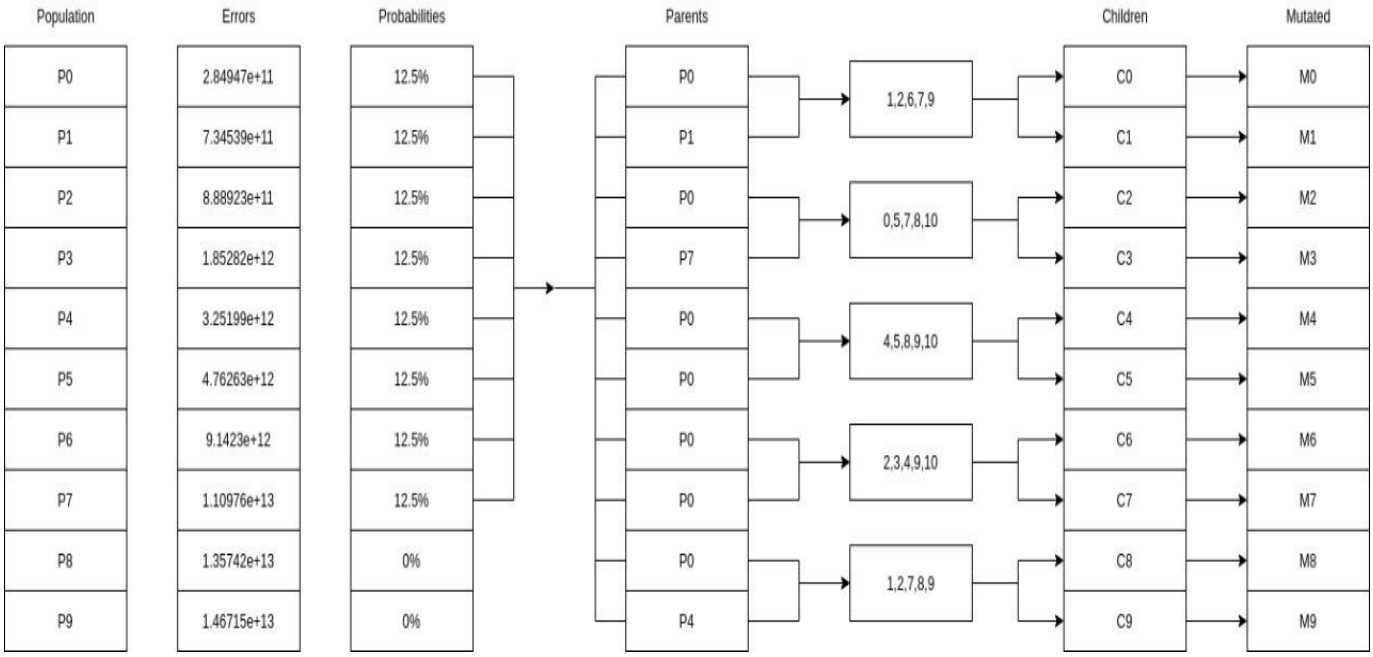

Population
Errors
Probabilities
Parents
Children
Mutated
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
2.84947e+11
7.34539e+11
8.88923e+11
1.85282e+12
3.25199e+12
4.76263e+12
9.1423e+12
1.10976e+13
1.35742e+13
1.46715e+13
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
0%
0%
P0
P1
P0
P7
P0
P0
P0
P0
P0
P4
1,2,6,7,9
0,5,7,8,10
4,5,8,9,10
2,3,4,9,10
1,2,7,8,9
C0
C1
C2
C3
C4
C5
C6
C7
C8
C9
M0
M1
M2
M3
M4
M5
M6
M7
M8
M9

DIAGRAM 2

Table 1:

Index	Population	Population Errors

P0	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.90477420e-11
	  -1.85221708e-10 -1.80586373e-15 9.13468881e-16 2.12394891e-05
	  -1.87585566e-06 -1.69252514e-08 9.49703462e-10]]
		2.84947e+11

P1	[[0.00000000e+00 -1.34214696e-12 -2.15811626e-13 4.44360117e-11
	  -1.72425873e-10 -1.78391498e-15 9.23806089e-16 2.27885706e-05
	  -1.75153966e-06 -1.68697667e-08 8.86742734e-10]]
		3.11363e+11

P2	[[0.00000000e+00 -1.41532326e-12 -2.28980078e-13 4.82729262e-11
	  -1.68898185e-10 -2.00252333e-15 9.32306686e-16 2.07956883e-05
	  -2.08559424e-06 -1.74285109e-08 1.07347650e-09]]
		7.34539e+11

P3	[[0.00000000e+00 -1.63323139e-12 -2.50949411e-13 4.68521916e-11
	  -1.92198708e-10 -1.99413849e-15 8.37898914e-16 2.66276689e-05
	  -1.87758618e-06 -1.56762599e-08 9.01058478e-10]]
		8.29955e+11

P4	[[0.00000000e+00 -1.55095294e-12 -2.24198774e-13 4.38137376e-11
	  -1.79371080e-10 -1.83669770e-15 8.43144249e-16 2.13310957e-05
	  -2.07973302e-06 -1.60858646e-08 9.98214034e-10]]
		8.88923e+11

P5	[[0.00000000e+00 -1.48300094e-12 -2.01786273e-13 5.11026428e-11
	  -1.72635270e-10 -1.99413849e-15 8.73138513e-16 2.08837244e-05
	  -2.06867722e-06 -1.67683456e-08 1.05982796e-09]]
		1.12419e+12

P6	[[0.00000000e+00 -1.60043842e-12 -2.20869787e-13 4.90477420e-11
	  -1.68027453e-10 -1.97223778e-15 8.09986689e-16 2.28669770e-05
	  -2.03173992e-06 -1.72516689e-08 9.80675328e-10]]
		1.55654e+12

P7	[[0.00000000e+00 -1.57703128e-12 -2.50642669e-13 4.31572045e-11
	  -1.75760959e-10 -1.67638957e-15 8.04457298e-16 2.42204795e-05
	  -2.04721003e-06 -1.61690456e-08 1.01297625e-09]]
		1.85282e+12

P8	[[0.00000000e+00 -1.66472079e-12 -2.04466716e-13 4.78220342e-11
	  -1.59970473e-10 -1.80159205e-15 8.71014711e-16 1.91512466e-05
	  -1.85213805e-06 -1.71672584e-08 9.96827826e-10]]
		1.9664e+12

P9	[[0.00000000e+00 -1.47361088e-12 -2.45841733e-13 4.65037336e-11
	  -1.75214813e-10 -1.60281988e-15 8.78367033e-16 2.16754254e-05
	  -2.04989862e-06 -1.73980366e-08 9.89721172e-10]]
		1.99966e+12

Table 2:

Index	Parents	Crossover positions	Children

C0	[['P7' 'P5']]	[[0. 1. 2. 9. 10.]]	[[0.00000000e+00 -1.48300094e-12 -2.01786273e-13 4.31572045e-11
			  -1.75760959e-10 -1.67638957e-15 8.04457298e-16 2.42204795e-05
			  -2.04721003e-06 -1.67683456e-08 1.05982796e-09]]

C1	[['P7' 'P5']]	[[0. 1. 2. 9. 10.]]	[[0.00000000e+00 -1.57703128e-12 -2.50642669e-13 5.11026428e-11
			  -1.72635270e-10 -1.99413849e-15 8.73138513e-16 2.08837244e-05
			  -2.06867722e-06 -1.61690456e-08 1.01297625e-09]]

C2	[['P7' 'P0']]	[[1. 4. 5. 7. 9.]]	[[0.00000000e+00 -1.52862353e-12 -2.20869787e-13 4.90477420e-11
			  -1.85221708e-10 -1.80586373e-15 8.09986689e-16 2.12394891e-05
			  -2.03173992e-06 -1.69252514e-08 9.80675328e-10]]

C3	[['P7' 'P0']]	[[1. 4. 5. 7. 9.]]	[[0.00000000e+00 -1.60043842e-12 -2.08934108e-13 4.90477420e-11
			  -1.68027453e-10 -1.97223778e-15 9.13468881e-16 2.28669770e-05
			  -1.87585566e-06 -1.72516689e-08 9.49703462e-10]]

C4	[['P7' 'P3']]	[[0. 2. 8. 9. 10.]]	[[0.00000000e+00 -1.60043842e-12 -2.50949411e-13 4.90477420e-11
			  -1.68027453e-10 -1.97223778e-15 8.09986689e-16 2.28669770e-05
			  -1.87758618e-06 -1.56762599e-08 9.01058478e-10]]

C5	[['P7' 'P3']]	[[0. 2. 8. 9. 10.]]	[[0.00000000e+00 -1.63323139e-12 -2.20869787e-13 4.68521916e-11
			  -1.92198708e-10 -1.99413849e-15 8.37898914e-16 2.66276689e-05
			  -2.03173992e-06 -1.72516689e-08 9.80675328e-10]]

C6	[['P7' 'P2']]	[[2. 3. 4. 5. 10.]]	[[0.00000000e+00 -1.63323139e-12 -2.28980078e-13 4.82729262e-11
			  -1.68898185e-10 -2.00252333e-15 8.37898914e-16 2.66276689e-05
			  -1.87758618e-06 -1.56762599e-08 1.07347650e-09]]

C7	[['P7' 'P2']]	[[2. 3. 4. 5. 10.]]	[[0.00000000e+00 -1.41532326e-12 -2.50949411e-13 4.68521916e-11
			  -1.92198708e-10 -1.99413849e-15 9.32306686e-16 2.07956883e-05
			  -2.08559424e-06 -1.74285109e-08 9.01058478e-10]]

C8	[['P7' 'P6']]	[[0. 1. 4. 6. 9.]]	[[0.00000000e+00 -1.60043842e-12 -2.50642669e-13 4.31572045e-11
			  -1.68027453e-10 -1.67638957e-15 8.09986689e-16 2.42204795e-05
			  -2.04721003e-06 -1.72516689e-08 1.01297625e-09]]

C9	[['P7' 'P6']]	[[0. 1. 4. 6. 9.]]	[[0.00000000e+00 -1.57703128e-12 -2.20869787e-13 4.90477420e-11
			  -1.75760959e-10 -1.97223778e-15 8.04457298e-16 2.28669770e-05
			  -2.03173992e-06 -1.61690456e-08 9.80675328e-10]]

Table 3:

Index	Mutated Children	Mutated Children Errors

M0	[[0.00000000e+00 -1.47981101e-12 -2.01786273e-13 4.12305482e-11
	  -1.71770204e-10 -1.67638957e-15 8.34297807e-16 2.53456585e-05
	  -2.09612368e-06 -1.84265055e-08 1.02322625e-09]]
		4.68352e+11

M1	[[0.00000000e+00 -1.57703128e-12 -2.50722603e-13 5.50733865e-11
	  -1.79993965e-10 -1.96535590e-15 8.94187329e-16 2.12003149e-05
	  -2.18061203e-06 -1.53985882e-08 9.31464981e-10]]
		1.94508e+13

M2	[[0.00000000e+00 -1.61658088e-12 -2.12349399e-13 4.72084426e-11
	  -1.85221708e-10 -1.93622828e-15 7.80638647e-16 2.19644846e-05
	  -1.91930642e-06 -1.58981301e-08 1.07695595e-09]]
		2.38339e+13

M3	[[0.00000000e+00 -1.67602124e-12 -1.92265283e-13 5.33414054e-11
	  -1.55582732e-10 -2.10006176e-15 9.65978920e-16 2.16001882e-05
	  -1.87585566e-06 -1.73733917e-08 9.98096196e-10]]
		2.11576e+12

M4	[[0.00000000e+00 -1.45903141e-12 -2.47764875e-13 4.61213376e-11
	  -1.60135471e-10 -2.02037308e-15 7.67808638e-16 2.42530930e-05
	  -1.78012226e-06 -1.56762599e-08 9.39069509e-10]]
		8.42842e+12

M5	[[0.00000000e+00 -1.63323139e-12 -2.25626861e-13 4.68521916e-11
	  -1.91699785e-10 -1.87355871e-15 7.85046451e-16 2.52795971e-05
	  -1.93962842e-06 -1.69437333e-08 9.80675328e-10]]
		2.12082e+12

M6	[[0.00000000e+00 -1.72372492e-12 -2.44756681e-13 4.42351253e-11
	  -1.66834065e-10 -1.93073229e-15 7.88211871e-16 2.70614462e-05
	  -2.06174537e-06 -1.59991743e-08 1.05323936e-09]]
		1.14294e+13

M7	[[0.00000000e+00 -1.46723815e-12 -2.52234759e-13 5.04883175e-11
	  -1.90573274e-10 -2.16218834e-15 8.50471188e-16 2.12725661e-05
	  -1.91123620e-06 -1.76015692e-08 8.63555393e-10]]
		1.98389e+13

M8	[[0.00000000e+00 -1.63013293e-12 -2.43935602e-13 4.31572045e-11
	  -1.78943183e-10 -1.70970746e-15 8.29201362e-16 2.39749916e-05
	  -2.05674401e-06 -1.75589353e-08 9.92750013e-10]]
		8.31927e+11

M9	[[0.00000000e+00 -1.57703128e-12 -2.20869787e-13 4.70827744e-11
	  -1.92074210e-10 -1.97223778e-15 8.39863238e-16 2.24680355e-05
	  -1.92238729e-06 -1.73259328e-08 9.80675328e-10]]
		4.36508e+11

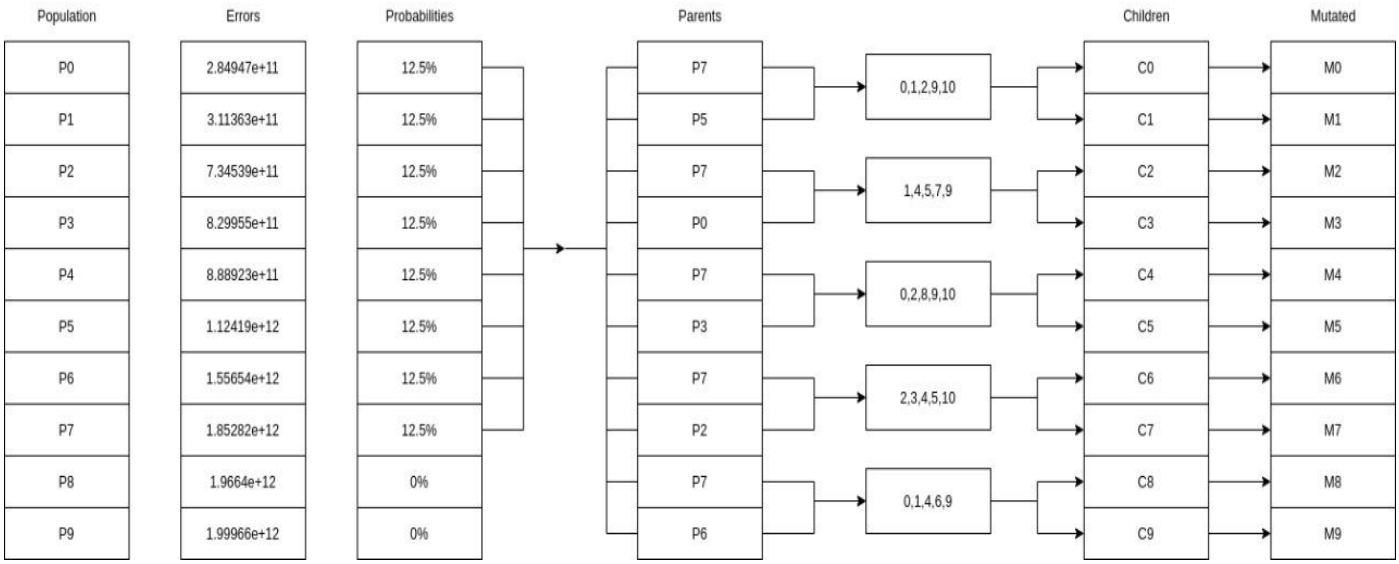

Population
Errors
Probabilities
Parents
Children
Mutated
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
2.84947e+11
3.11363e+11
7.34539e+11
8.29955e+11
8.88923e+11
1.12419e+12
1.55654e+12
1.85282e+12
1.9664e+12
1.99966e+12
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
0%
0%
P7
P5
P7
P0
P7
P3
P7
P2
P7
P6
0,1,2,9,10
1,4,5,7,9
0,2,8,9,10
2,3,4,5,10
0,1,4,6,9
C0
C1
C2
C3
C4
C5
C6
C7
C8
C9
M0
M1
M2
M3
M4
M5
M6
M7
M8
M9

DIAGRAM 3

Table 1:

Index	Population	Population Errors
P0	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.90477420e-11 -1.85221708e-10 -1.80586373e-15 9.13468881e-16 2.12394891e-05 -1.87585566e-06 -1.69252514e-08 9.49703462e-10]]	2.84947e+11
P1	[[0.00000000e+00 -1.34214696e-12 -2.15811626e-13 4.44360117e-11 -1.72425873e-10 -1.78391498e-15 9.23806089e-16 2.27885706e-05 -1.75153966e-06 -1.68697667e-08 8.86742734e-10]]	3.11363e+11
P2	[[0.00000000e+00 -1.57703128e-12 -2.20869787e-13 4.70827744e-11 -1.92074210e-10 -1.97223778e-15 8.39863238e-16 2.24680355e-05 -1.92238729e-06 -1.73259328e-08 9.80675328e-10]]	4.36508e+11
P3	[[0.00000000e+00 -1.47981101e-12 -2.01786273e-13 4.12305482e-11 -1.71770204e-10 -1.67638957e-15 8.34297807e-16 2.53456585e-05 -2.09612368e-06 -1.84265055e-08 1.02322625e-09]]	4.68352e+11
P4	[[0.00000000e+00 -1.41532326e-12 -2.28980078e-13 4.82729262e-11 -1.68898185e-10 -2.00252333e-15 9.32306686e-16 2.07956883e-05 -2.08559424e-06 -1.74285109e-08 1.07347650e-09]]	7.34539e+11
P5	[[0.00000000e+00 -1.63323139e-12 -2.50949411e-13 4.68521916e-11 -1.92198708e-10 -1.99413849e-15 8.37898914e-16 2.66276689e-05 -1.87758618e-06 -1.56762599e-08 9.01058478e-10]]	8.29955e+11
P6	[[0.00000000e+00 -1.63013293e-12 -2.43935602e-13 4.31572045e-11 -1.78943183e-10 -1.70970746e-15 8.29201362e-16 2.39749916e-05 -2.05674401e-06 -1.75589353e-08 9.92750013e-10]]	8.31927e+11
P7	[[0.00000000e+00 -1.55095294e-12 -2.24198774e-13 4.38137376e-11 -1.79371080e-10 -1.83669770e-15 8.43144249e-16 2.13310957e-05 -2.07973302e-06 -1.60858646e-08 9.98214034e-10]]	8.88923e+11
P8	[[0.00000000e+00 -1.48300094e-12 -2.01786273e-13 5.11026428e-11 -1.72635270e-10 -1.99413849e-15 8.73138513e-16 2.08837244e-05 -2.06867722e-06 -1.67683456e-08 1.05982796e-09]]	1.12419e+12
P9	[[0.00000000e+00 -1.60043842e-12 -2.20869787e-13 4.90477420e-11 -1.68027453e-10 -1.97223778e-15 8.09986689e-16 2.28669770e-05 -2.03173992e-06 -1.72516689e-08 9.80675328e-10]]	1.55654e+12

Table 2:

Index	Parents	Crossover positions	Children

C0	[['P4' 'P1']]	[[5. 6. 7. 8. 9.]]	[[0.00000000e+00 -1.41532326e-12 -2.28980078e-13 4.82729262e-11
			  -1.68898185e-10 -1.78391498e-15 9.23806089e-16 2.27885706e-05
			  -1.75153966e-06 -1.68697667e-08 1.07347650e-09]]

C1	[['P4' 'P1']]	[[5. 6. 7. 8. 9.]]	[[0.00000000e+00 -1.34214696e-12 -2.15811626e-13 4.44360117e-11
			  -1.72425873e-10 -2.00252333e-15 9.32306686e-16 2.07956883e-05
			  -2.08559424e-06 -1.74285109e-08 8.86742734e-10]]

C2	[['P4' 'P4']]	[[4. 5. 6. 9. 10.]]	[[0.00000000e+00 -1.34214696e-12 -2.15811626e-13 4.44360117e-11
			  -1.68898185e-10 -2.00252333e-15 9.32306686e-16 2.27885706e-05
			  -1.75153966e-06 -1.74285109e-08 1.07347650e-09]]

C3	[['P4' 'P4']]	[[4. 5. 6. 9. 10.]]	[[0.00000000e+00 -1.41532326e-12 -2.28980078e-13 4.82729262e-11
			  -1.72425873e-10 -1.78391498e-15 9.23806089e-16 2.07956883e-05
			  -2.08559424e-06 -1.68697667e-08 8.86742734e-10]]

C4	[['P4' 'P5']]	[[1. 2. 5. 7. 10.]]	[[0.00000000e+00 -1.63323139e-12 -2.50949411e-13 4.70827744e-11
			  -1.92074210e-10 -1.99413849e-15 8.39863238e-16 2.66276689e-05
			  -1.92238729e-06 -1.73259328e-08 9.01058478e-10]]

C5	[['P4' 'P5']]	[[1. 2. 5. 7. 10.]]	[[0.00000000e+00 -1.57703128e-12 -2.20869787e-13 4.68521916e-11
			  -1.92198708e-10 -1.97223778e-15 8.37898914e-16 2.24680355e-05
			  -1.87758618e-06 -1.56762599e-08 9.80675328e-10]]

C6	[['P4' 'P5']]	[[0. 6. 8. 9. 10.]]	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.90477420e-11
			  -1.85221708e-10 -1.80586373e-15 8.37898914e-16 2.12394891e-05
			  -1.87758618e-06 -1.56762599e-08 9.01058478e-10]]

C7	[['P4' 'P5']]	[[0. 6. 8. 9. 10.]]	[[0.00000000e+00 -1.63323139e-12 -2.50949411e-13 4.68521916e-11
			  -1.92198708e-10 -1.99413849e-15 9.13468881e-16 2.66276689e-05
			  -1.87585566e-06 -1.69252514e-08 9.49703462e-10]]

C8	[['P4' 'P7']]	[[3. 5. 6. 7. 10.]]	[[0.00000000e+00 -1.52862353e-12 -2.08934108e-13 4.38137376e-11
			  -1.85221708e-10 -1.83669770e-15 8.43144249e-16 2.13310957e-05
			  -1.87585566e-06 -1.69252514e-08 9.98214034e-10]]

C9	[['P4' 'P7']]	[[3. 5. 6. 7. 10.]]	[[0.00000000e+00 -1.55095294e-12 -2.24198774e-13 4.90477420e-11
			  -1.79371080e-10 -1.80586373e-15 9.13468881e-16 2.12394891e-05
			  -2.07973302e-06 -1.60858646e-08 9.49703462e-10]]

Table 3:

Index	Mutated Children	Mutated Children Errors

M0	[[0.00000000e+00 -1.43847331e-12 -2.45675268e-13 4.65657383e-11
	  -1.79090034e-10 -1.63224299e-15 9.23806089e-16 2.47903649e-05
	  -1.75308294e-06 -1.74247959e-08 1.06512089e-09]]
		4.92981e+13

M1	[[0.00000000e+00 -1.43009549e-12 -2.31492513e-13 4.03903335e-11
	  -1.86657592e-10 -1.89973930e-15 9.52203462e-16 2.25286300e-05
	  -1.93887538e-06 -1.82397029e-08 8.77024775e-10]]
		1.8184e+13

M2	[[0.00000000e+00 -1.33307625e-12 -2.12265703e-13 4.43393854e-11
	  -1.68898185e-10 -1.89159016e-15 9.95153121e-16 2.27885706e-05
	  -1.76880747e-06 -1.74285109e-08 1.11288769e-09]]
		5.88519e+13

M3	[[0.00000000e+00 -1.31512227e-12 -2.28980078e-13 4.82263774e-11
	  -1.63847843e-10 -1.81394398e-15 9.03522922e-16 2.06978124e-05
	  -2.21127471e-06 -1.62275356e-08 9.67785306e-10]]
		1.80076e+13

M4	[[0.00000000e+00 -1.66688550e-12 -2.62369781e-13 4.75387765e-11
	  -1.75095644e-10 -2.01212823e-15 8.37769577e-16 2.65840962e-05
	  -1.81081738e-06 -1.66554330e-08 9.14907756e-10]]
		2.60544e+12

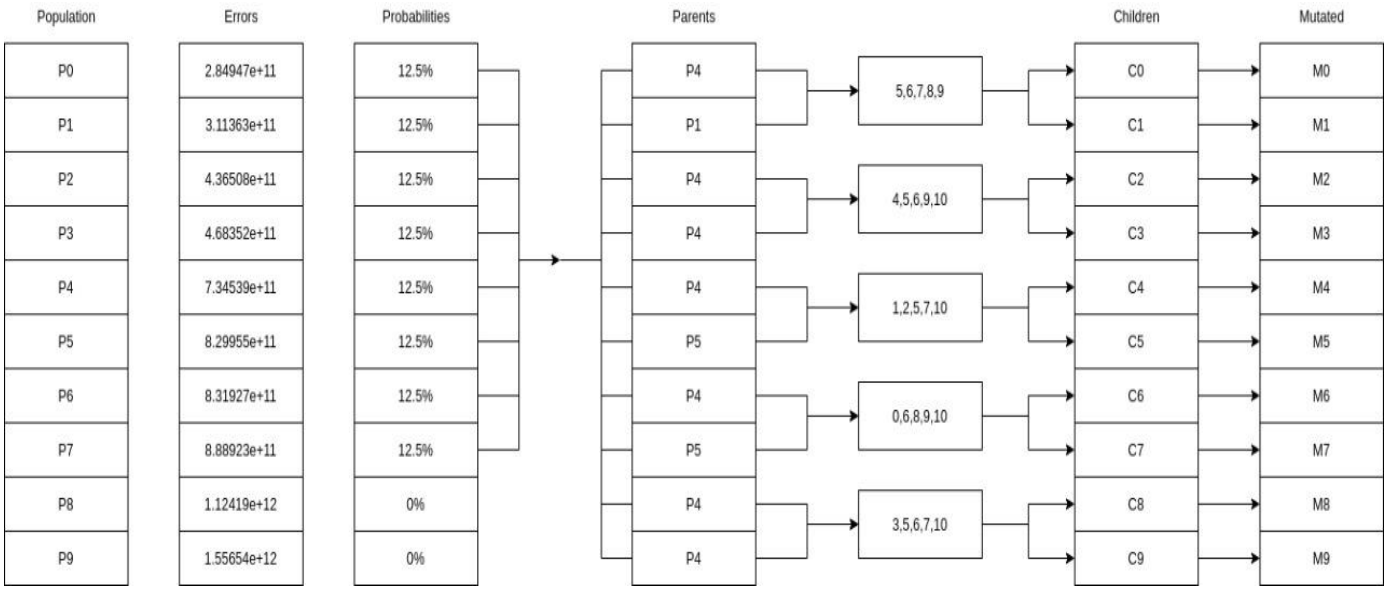
M5	[[0.00000000e+00 -1.50611187e-12 -2.20869787e-13 4.60546763e-11
	  -2.06005670e-10 -1.96948403e-15 8.81017613e-16 2.03621305e-05
	  -1.94260701e-06 -1.41761710e-08 9.17657617e-10]]
		5.39273e+11

M6	[[0.00000000e+00 -1.66259639e-12 -2.27650329e-13 5.38368943e-11
	  -1.93619049e-10 -1.64754508e-15 7.92957055e-16 2.03299011e-05
	  -1.87758618e-06 -1.62222309e-08 8.60133167e-10]]
		8.63747e+12

M7	[[0.00000000e+00 -1.54129237e-12 -2.43554243e-13 4.68521916e-11
	  -1.83070885e-10 -1.98031423e-15 8.34487158e-16 2.60073620e-05
	  -1.87585566e-06 -1.69088911e-08 1.02388944e-09]]
		1.7151e+13

M8	[[0.00000000e+00 -1.42496740e-12 -2.16981885e-13 4.64964230e-11
	  -1.70233226e-10 -1.69655144e-15 7.69528743e-16 1.93910722e-05
	  -1.99688433e-06 -1.65815474e-08 1.05075448e-09]]
		2.79746e+12

M9	[[0.00000000e+00 -1.51064808e-12 -2.40245084e-13 4.90477420e-11
	  -1.79371080e-10 -1.71435096e-15 9.13468881e-16 2.24072471e-05
	  -2.25890502e-06 -1.68572978e-08 9.49474049e-10]]
		3.00651e+13



FITNESS FUNCTION

- ➔ The fitness function is one that is used to decide whether an individual from a population will advance to the next generation or not. Greater the fitness, better is the individual.

- ➔ However, here we have been provided with the train and validation errors of the individual vectors.

- ➔ The relation between fitness and error is: Lower is the error => Better is the fitness

The fitness function we have used while coding the Genetic Algorithm is :

```
err = err1 + err2
```

- ➔ We have not included weights in our fitness function since otherwise, the function would be biased to a particular error. Here, both training and validation error are equally important and hence it is hard to say which is more fit.

CROSSOVER FUNCTION

➔ In the GA, crossover is a function that is performed on two parents to produce offspring. There are two parts in the crossover function :
1. Select the parents for crossover
2. Perform the crossover

➔ In our implementation, we make our selection for the crossover from the top cross_select number of parents, where the parents are sorted in increasing order of their errors

As for the actual crossover function -

➔ It first generates a random list of crossover_no numbers. The chromosomes (elements) at these indices are swapped between the two parent vectors resulting in the formation of two child vectors.

➔ The child vectors are then mutated and returned from the function.

```
def crossover(vector1, vector2, mutate_range,prob_mut_cross, index=-1):
    send1, send2 = (vector1.tolist(), vector2.tolist())
    #crossover_no is the 2nd argument, here it is 5
    a = np.random.choice(np.arange(0, 11),5, replace=False)
    lis = [i for i in a.tolist()]
    it = 0
    for i in lis:
        send1[i], send2[i] = (np.copy(vector2[i]), np.copy(vector1[i]))
        it += 1

    return mutateall(send1,prob_mut_cross,mutate_range), mutateall(send2,prob_mut_cross,mutate_range),send1,send2,a
```

MUTATIONS

Our mutation function takes 3 parameters

➔ **temp :** This is the vector that is to be mutated.

➔ **prob :**
- The probability with which a particular gene in a chromosome will be mutated.
- As the iterations of the GA proceed, we increase the probability of mutation by 0.01 (every 6 iterations). This is also helpful in ensuring that the algorithm does not converge to a local minima.
- In case it is approaching a local minima, then the increased probability of mutation helps in bringing diversity in the population.

➔ **mutate_range :**
- The range within which the element will be mutated. This is a crucial parameter.

- Since the given overfit vector is very sensitive and if the genes are changed by a relatively large amount (changing the order of the element itself) then this could reflect in the train and validation errors in a negative way.
- Further, we have implemented simulated annealing here wherein we start with a mutate_range of 0.1 and then gradually decrease it over the GA iterations (by 0.01 every 6 iterations)

We have used mutations in 2 places :

➔ To produce the first generation (initial population) for the genetic algorithm: Here we pass a high initial prob of mutation so that the degree of mutation is high and the first generation of the population is diverse enough.

➔ On child vectors after crossover: Here the child vectors are mutated based on the parameters that have been passed to the function.

HYPERPARAMETERS

```
crossover_no = 3
pop_size: 10
iter: 15
cross_select_from: 8
select_sure: 3
prob_mut_cross: 0.9
mutate_range: 0.1
```

➔ **pop_size:** Initially we started with a population size of 70. As we progressed in the assignment, we were able to better our GA and realized that a pool size of 10 suited the best as it gave us room to run more iterations and not exceed the number of requests to the server per day.

➔ **select_sure:** In order to ensure that the individuals with the best genes are not lost in the future generations we made sure that we select the top few parents and children for sure (sorted in the ascending order of errors). While experimenting with this value we found that on choosing a very high select_sure value - there was no diversity in the future generations. All the points were clustered together and the algorithm converged to a local minima. On choosing a very low select_sure the algorithm performed poorly. Hence after multiple tries, we fixed a select_sure of 3

➔ **cross_select_from:** This parameter selects the top few parents to send to the crossover function. If this is very low, then the options to choose the parents for crossover are restricted. Likewise, a very high cross_select_from leads to too much randomness. We stuck to a value of 8

➔ **crossover_no:** This number indicates the number of indices at which the elements will be swapped between the two parents in the crossover function. We have chosen a value of 3 to ensure that there is a sufficient degree of crossover that can help in the GA. We also tried randomly generating a number between 0-5 but that did not help.

➔ **mutate_range:** We have set this parameter to 0.1. The overfit vector is sensitive and if the mutation is drastic it will lead to a high error. Hence we made sure that the elements of the vector undergoing mutation change by this formula :

```
fact = random.uniform(-mutate_range, mutate_range)
vector[i] = np.random.choice([vector[i]*(fact+1), vector[i]], p=[prob,1-prob])
```

With simulated annealing, this range decreases by 0.01 every 6 iterations, and prob_mut_cross increases by 0.01.

➔ **prob_mut_cross:** This parameter is set to 0.9 to start with a large degree of mutations. This will ensure diversity and prevent converging to a local minima. Further, the prob_mut_cross increases every 6 iterations by 0.01

HEURISTICS APPLIED

While constructing the Genetic Algorithm, the heuristics that we applied include :

➔ ***Using a fitness function with weights*** :

- We ran the algorithm with a fitness function as follows : err = err[0] + 1.5 * err[1].
- This function however did not seem to help our algorithm to converge to the global minima.
- The train and validation errors we received were still quite high.

➔ ***Modifying the fitness function mid-way*** :

- To try and improve the algorithm, we also tried changing the fitness function midway.
- For first k iterations we had a fitness function of err = err[0] + 1.5*err[1]. After k iterations, we changed the fitness function to simply err = err[0] + err[1].
- We did so considering that with a 'not so random' population in the later iterations of the GA, we can see that both training nad validation errors are equally important and one cannot overpower the other in deciding the fitness of an individual.
- However, this approach did not help in reducing the errors either.

➔ ***Simulated Annealing*** :

- In this method, we reduce the range within which a particular gene/element of the chromosome/vector can be mutated.

- We start off with a mutate_range of 0.1. After every 6 iterations, this is decreased by 0.01.
- As a consequence of the decreased mutate_range the vectors may now come very close to each other.
- To prevent the algorithm from converging to a local minima, we started off with a prob_mut of 0.7 and increased by 0.01 every 6 iterations.
- This method helped us in the process of achieving the global minima.

STATISTICAL INFORMATION

Pop size	iter	Select sure	Cross select from	Prob mut cross	Mutate range	Crossover number	Train error	Validation error	comments
10	15	3	8	0.9	0.1	3	1.2x10^11	7.6x10^10	Fitness function: err = err1 + 1.5*err2 Difference between the errors is very high.
10	15	3	8	0.9	0.1	3	1.2x10^11	1.03x10^11	Fitness function: err = err1 + 1.5*err2 Difference between the errors is very high.
10	15	3	8	0.9	0.1	3	1.38x10^11	5.97x10^10	Fitness function: err = err1 + 1.5*err2 Difference between the errors is very high.
10	15	3	8	0.9	0.1	3	4.9x10^10	4.9x10^10	Error decreased when fitness function gave equal priority to both errors err = err1 + err2 instead of err1 + 1.5*err2 Difference between the errors is decreased.
10	15	3	8	0.9	0.1	3	4.04x10^10	4.8x10^10	Error decreased when fitness function gave equal priority to both errors err = err1 + err2 instead of err1 + 1.5*err2. Difference between the errors is decreased.
10	15	3	8	0.9	0.1	3	4.7x10^10	4.2x10^10	Error decreased when fitness function gave equal priority to both errors err = err1 + err2 instead of err1 + 1.5*err2. Difference between the errors is decreased.

We have tried with different pop_size values but didn't store the data so couldn't put in the table.

TRACE FOR ITERATIONS

The following tables represent the trace for the iterations:

Table 1: this table shows the following:

- ➔ **Col 1:** Indices of the initial population (P0,P1,P2,P3...)
- ➔ **Col 2:** Individuals of the populations (Vectors)
- ➔ **Col 3:** Errors corresponding to each individual in the population

Table 2: this table shows the following:

- ➔ **Col 1:** Indices of the children population produced (C0,C1,C2,C3...)
- ➔ **Col 2:** The Vectors selected for crossover (That is the parents from which the children have been produced)
- ➔ **Col 3:** The indices at which the parent vector genes are swapped.
- ➔ **Col 4:** The child vectors produced after applying the crossover

Table 3: this table shows the following:

- ➔ **Col 1:** Indices of the mutated children (M0,M1,M2, M3...)
- ➔ **Col 2:** Mutated child vectors after applying mutation
- ➔ **Col 3:** Errors corresponding to each mutated child

The parameters for this trace are:

```
crossover_no = 3
pop_size: 10
iter: 15
cross_select_from: 8
select_sure: 3
prob_mut_cross: 0.9
mutate_range: 0.1
```