Our genetic algorithm is very similar to the base algorithm. We are using a population size of 100 , where while making next generations best n parents (60<=n<=90) simply moves to the next generation while rest of the left spaces are filled by children which are produced by cross-over between randomly selected top 20 or 30 parents . We also mutate these children in order to stop false convergence.

Now as I mentioned that our population size was 100 and it is really visually impossible to understand and represent cross-over and mutations for such large population , so instead i am going to represent it for a subset of my original population only.

First Iteration:-
Starting vectors:-
v1 = [-2.6426889099049466, -7.021822879777007e-13, -3.231316819643275e-13, 3.0446227099508686e-11, -0.12460111602215033, -3.0690529713552496e-16, -9.83468275446705, 2.3950987554711424e-05, -1.3704354083514642e-06, -1.416431991173997e-08, 6.277814034149513e-10]

v2 = [-2.67113182575081, -7.021659756093542e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -10.0, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v3= [-2.697843144008318, -7.021659756093542e-13, -3.251679350961277e-13, 3.1284212026498604e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v4= [-2.67113182575081, -6.951443158532606e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v5= [-2.6630108536562402, -7.104337250873248e-13, -3.2103792818066963e-13, 3.075848573857818e-11, -0.12599673580076243, -3.090425605644727e-16, -9.918330834114158, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

v6= [-2.6630108536562402, -7.033997278092324e-13, -3.275235428913902e-13, 3.075848573857818e-11, -0.12725670315877005, -3.090425605644727e-16, -10.0, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

v7= [-2.67575749252493, -7.081158294483324e-13, -3.2522637934293747e-13, 3.074904162776287e-11, -0.12572262875442358, -3.1096857462136975e-16, -9.949722206509463, 2.3949785411982346e-05, -1.3704285826494409e-06, -1.4163608979292807e-08, 6.277782766358492e-10]

Now after crossovers and mutations:

v1<->v2 = [-2.67113182575081, -7.021659756093542e-13, -3.251679350961277e-13, 3.1284212026498604e-11, -0.1258527003911205, -3.1194926969319223e-16, -9.912052108955898, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

This above vector mutated to = [-2.697843144008318, -7.091876353654477e-13, -3.2191625574516645e-13, 3.159705414676359e-11, -0.12459417338720931, -3.088297769962603e-16, -9.81293158786634, 2.3711303975691414e-05, -1.384156603824382e-06, -1.4305858309481889e-08, 6.340669323045478e-10]

The rest of the 6 parents moves into next generation directly as they are few of the top best vectors.

Iteration-2:(Note that vector produced in last generation is placed at 4th position after sorting)

v1 = [-2.6426889099049466, -7.021822879777007e-13, -3.231316819643275e-13, 3.0446227099508686e-11, -0.12460111602215033, -3.0690529713552496e-16, -9.83468275446705, 2.3950987554711424e-05, -1.3704354083514642e-06, -1.416431991173997e-08, 6.277814034149513e-10]

v2 = [-2.67113182575081, -7.021659756093542e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -10.0, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v3= [-2.697843144008318, -7.021659756093542e-13, -3.251679350961277e-13, 3.1284212026498604e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v4 = [-2.697843144008318, -7.091876353654477e-13, -3.2191625574516645e-13, 3.159705414676359e-11, -0.12459417338720931, -3.088297769962603e-16, -9.81293158786634, 2.3711303975691414e-05, -1.384156603824382e-06, -1.4305858309481889e-08, 6.340669323045478e-10]

v5= [-2.67113182575081, -6.951443158532606e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v6= [-2.6630108536562402, -7.104337250873248e-13, -3.2103792818066963e-13, 3.075848573857818e-11, -0.12599673580076243, -3.090425605644727e-16, -9.918330834114158, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

v7= [-2.6630108536562402, -7.033997278092324e-13, -3.275235428913902e-13, 3.075848573857818e-11, -0.12725670315877005, -3.090425605644727e-16, -10.0, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

Now after cross-over and mutation:
v4<->v7 = [-2.673444932923365, -7.051412546907604e-13, -3.2519716067875e-13, 3.1016595151288153e-11, -0.12578765687404897, -3.114588641115767e-16, -9.930889387362875, 2.3950298693552415e-05, -1.3704403314309931e-06, -1.4163912527710035e-08, 6.277836586235928e-10]

above vector after mutation:

[-2.6467104835941315, -7.12192667237668e-13, -3.219451890719625e-13, 3.0706429199775275e-11, -0.12704553344278946, -3.0834427547046093e-16, -10.0, 2.371079570661689e-05, -1.384144734745303e-06, -1.4022273402432934e-08, 6.215058220373569e-10]

After sorting we and eliminating v7 we get next generation as:(new on 2nd rank)

v1 = [-2.6426889099049466, -7.021822879777007e-13, -3.231316819643275e-13, 3.0446227099508686e-11, -0.12460111602215033, -3.0690529713552496e-16, -9.83468275446705, 2.3950987554711424e-05, -1.3704354083514642e-06, -1.416431991173997e-08, 6.277814034149513e-10]

v2 = [-2.6467104835941315, -7.12192667237668e-13, -3.219451890719625e-13, 3.0706429199775275e-11, -0.12704553344278946, -3.0834427547046093e-16, -10.0, 2.371079570661689e-05, -1.384144734745303e-06, -1.4022273402432934e-08, 6.215058220373569e-10]

v3 = [-2.67113182575081, -7.021659756093542e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -10.0, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v4= [-2.697843144008318, -7.021659756093542e-13, -3.251679350961277e-13, 3.1284212026498604e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v5 = [-2.697843144008318, -7.091876353654477e-13, -3.2191625574516645e-13, 3.159705414676359e-11, -0.12459417338720931, -3.088297769962603e-16, -9.81293158786634, 2.3711303975691414e-05, -1.384156603824382e-06, -1.4305858309481889e-08, 6.340669323045478e-10]

v6= [-2.67113182575081, -6.951443158532606e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

v7= [-2.6630108536562402, -7.104337250873248e-13, -3.2103792818066963e-13, 3.075848573857818e-11, -0.12599673580076243, -3.090425605644727e-16, -9.918330834114158, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

For this final iteration we didn’t crossover , rather we just mutated all of the best vectors and compared them with our initial vectors.

The vectors that we get after mutation are:
a1 = [-2.6426889099049466, -7.021822879777007e-13, -3.231316819643275e-13, 3.0446227099508686e-11, -0.12460111602215033, -3.0690529713552496e-16, -9.83468275446705, 2.3950987554711424e-05, -1.3704354083514642e-06, -1.416431991173997e-08, 6.277814034149513e-10]

a2 = [-2.67113182575081, -7.021659756093542e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -10.0, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

a3 = [-2.697843144008318, -7.021659756093542e-13, -3.251679350961277e-13, 3.1284212026498604e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

a4 = [-2.67113182575081, -6.951443158532606e-13, -3.2841961444708896e-13, 3.159705414676359e-11, -0.1258527003911205, -3.088297769962603e-16, -9.81293158786634, 2.3950812096657993e-05, -1.3704520829944377e-06, -1.416421614800187e-08, 6.277890418856909e-10]

a5 = [-2.6630108536562402, -7.104337250873248e-13, -3.2103792818066963e-13, 3.075848573857818e-11, -0.12599673580076243, -3.090425605644727e-16, -9.918330834114158, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

a6 = [-2.6630108536562402, -7.033997278092324e-13, -3.275235428913902e-13, 3.075848573857818e-11, -0.12725670315877005, -3.090425605644727e-16, -10.0, 2.3952189155913203e-05, -1.3704422309787247e-06, -1.416503052393454e-08, 6.277845287855382e-10]

a7 = [-2.67575749252493, -7.081158294483324e-13, -3.2522637934293747e-13, 3.074904162776287e-11, -0.12572262875442358, -3.1096857462136975e-16, -9.949722206509463, 2.3949785411982346e-05, -1.3704285826494409e-06, -1.4163608979292807e-08, 6.277782766358492e-10]

These were the 3 iterations that we got and note we didn’t crossover and mutated all our parents in final iteration.

Fitness Function: We tried many combinations of fitness functions and kept on changing it in the middle. We basically used some a*valError + b*trainError as fitness function. The best fitness function that seemed to work for us was 4*valError + trainError. The reason for this ratio was basically to minimize valError more as compared to trainError.

We use two crossover functions as :

```
pro = a.score/(a.score+b.score)
  for i in range(0,11):
     arr.append(((1.5+pro)*b.arr[i]+(2.5-pro)*a.arr[i])/4)
```

and

```
pro = a.score/(a.score+b.score)
  for i in range(0,11):
     arr.append(((1+pro)*b.arr[i]+(2-pro)*a.arr[i])/3)
```

where a.score is the fitness score of a.arr vector where a.arr is the vector.
This is basically like taking a weighted mean of two vectors where the weights of two vectors are decided accoring to their relative scores.

Mutations:
We took an element of a vector which has to be mutated and then if the value of this i element of vector is ‘a’ then we changed a as:

```
a = random.choices(  { a/100  ,  -a/100  )  )
```

so we basically reduce or increment the value of a by a/100.
Mutation probability most of the times was set as 0.3 (was set to one when we got false convergence)

So our pool size was 100 . And instead of splitting a parent into multiple childs , we rather combined 2 parents to form single child. Total child that we produced were 40 and 60 best parents simply survive in next gen also. The reason for doing this was basically because we wanted a diverse population as much as we could. Hence we choose 100 as size of our population.
Next reason for making only 40 children and was because we wanted to make sure that we don’t lose good genes which are good for our model in future generations. Because parents for these 40 children were only selected among top 20-30 parents so it was highly probable that good genes will flow into these produced children also.

The next unusual thing that we did in our genetic algorithm was that instead of going into convergence we wanted to train our model as much as we could. So every time when we got some convergence we mutated 90 percent of the population just to remove this convergence . It is not like that we will lose the best vector because we did not mutated best 10 vectors and kept them in generation as it is. Now we start our normal genetic algorithm from then onwards.

Many things did not worked for us like at one point we were just trying to reduce valError. This gave us wrong results probably because you either get low error at some over-trained data or which is very near to minima of valError. Also we started mutating no matter what which gave us wrong results. Over mutation will never-ever converge to good results.

So the best Val error that we got was of order 1.8*10^10 and train error for this was 5.4*10^10. Ideally this should have been our best vector but it was not. Reason is simple that you either reach close enough to minima of val error to get best results or you overtrain your model which is not best but give some reasonable error. The problem with out low val error vector was that it wasn’t that much close to the minima of val error and hence we know by the grapph of error with respect train error that it first decreases to a minima then starts increasing again as we train our model. So what might have happened is that my vector was on the left side of minima (under trained data) and the error here was more than

an over trained vector like all zero vectors. But theoretically we get best results when we reach minima of validation error.

There are no such tricks that we used bu yeah at some point , we used unusual methods for mutation like multiplying whole vector by some factor or rearranging elements of vector etc.