MACHINE DATA AND LEARNING

Assignment-3

Genetic Algorithm

Team - 14

Sartak Periwal-2018101024

Bhavesh Shukla-2018101036

Summary of our Genetic Algorithm

Following steps are covered in the algorithm implemented:

1. Create Initial Population
2. Fitness Calculation
3. Find probability values for the solutions/chromosomes of population
4. Create next population/generation
    a. Fittest 30% chromosomes pass into next population directly
    b. Parent Selection
    c. Crossover
    d. Mutation

Create Initial Population

In this step we create the initial population.
Each chromosome (solution) in the population will have 11 genes, one gene for each weight.
To create an initial population,we start with the overfit_vector given to us.
We make all the chromosomes in the initial population = overfit_vector

```
overfit_vector= [
0.0 , 0.1240317450077846 , -6.211941063144333,
0.04933903144709126, 0.03810848157715883 ,
 8.132366097133624e-05 ,-6.018769160916912e-05 ,
 -1.251585565299179e-07 ,3.484096383229681e-08 ,
 4.1614924993407104e-11 ,-6.732420176902565e-12
]
```

Fitness Calculation

A Fitness Score is given to each individual which **shows the ability of an individual to “compete”**. The individuals having optimal fitness score (or near optimal) are preferred.

We store the train and validation error in error_val.
error_val=get_errors(id,vector) => Function call to server for getting errors.
error_list.append([i,total_error]) where
total_error=error_val[0]+4*error_val[1]

Similarly we find errors corresponding to all the chromosomes in the population.
The fitness value for a vector is defined as :
Fitness_value = train_error + 3*validation_error = error_val[0]+3*error_val[1]
In our algorithm,lower the fitness value,fitter/better is the vector.This is quite the opposite of normal implementation where higher the fitness value,fitter/better is the vector.
The fitness value should be low but the difference between the 2 vectors should also reduce

Find probability values for the solutions/chromosomes of population

With the help of fitness values,we assign probabilities to the corresponding solutions using the given algorithm:
Let summ_fitness=fitness[0] + fitness[1] + ……… + fitness[8] + fitness[9]
temp_probability[i]= fitness[i] / (summ_fitness)
Probability[i] = (1 - temp_probability[i]) / (num_solutions - 1)

Create next generation/population

The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce better offspring by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating a new generation.It is hoped that over successive generations better solutions will arrive while least fit die.

A. Fittest 30% chromosomes pass into next population directly

The lower the fitness value,better is the chromosome/solution. Thus, to conserve the survival of these fitter chromosomes,the fittest 30% solutions/chromosomes are passed to the next generation directly.

B. Parents Selection

Fitness Proportionate Selection : In this every individual can become a parent with a probability which is proportional to its fitness.

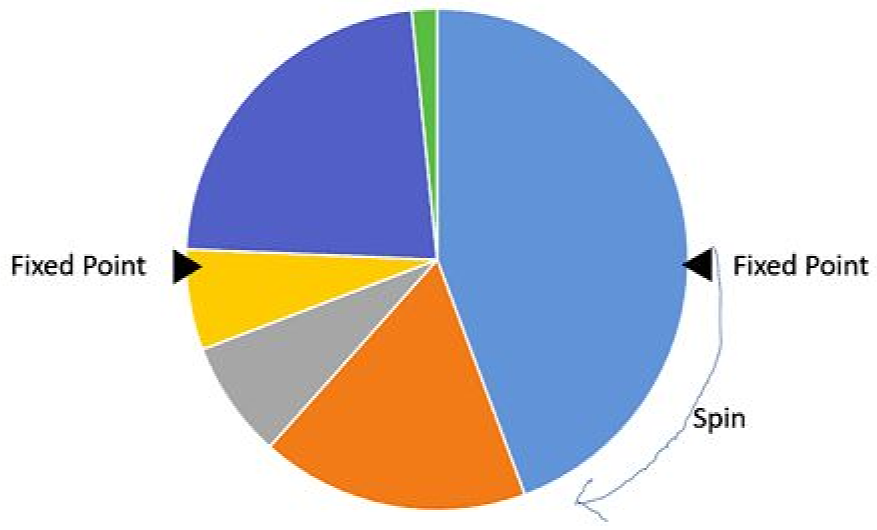
Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

Randomly,select 2 parents based on above conditions.

```
pos = ['0']*7 +['1']*6 + ['2'] *5 + ['3']*4+['4']*3 +['5']*2
+['6']*1
ind2=int(random.choice(pos))
```

If the population is 7

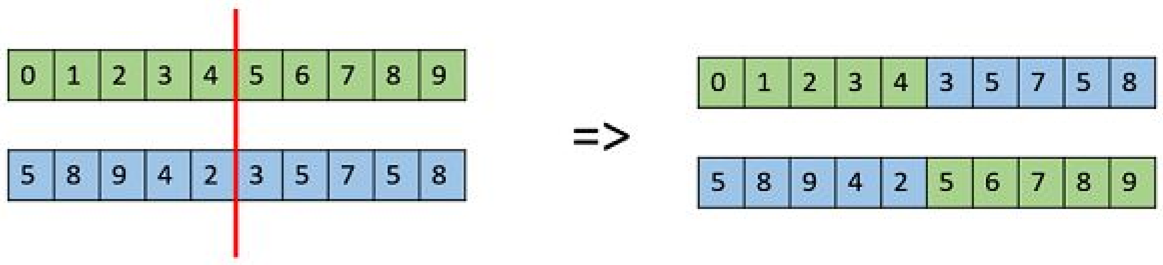
It is similar to Roulette wheel selection.



C. **<u>Crossover</u>**

It is the most significant phase in a genetic algorithm. In this step,parents reproduce to produce a new chromosome/solution.

One Point Crossover : In this,a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



Only first of the 2 produced childs is copied to the next generation.

D. Mutation

The key idea is to insert random genes in offspring to maintain and introduce diversity in the population to avoid the premature convergence and is usually applied with a low probability –> $\boldsymbol{p_m}$.

Random Resetting : In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

```
## here a random gene is selected from range (0,10) both inclusive
## and mutation is done only if probability < 0.3
index =random.randrange(0,11,1)
probability = random.random()
if probability < 0.3:
arr[index]=arr[index]*(random.uniform(.8,1.2))
```

---------------------- ALGORITHM FINISHES -------------------------

Fitness function

A fitness function should return higher values for better state our fitness functions aim is to reduce the total error and the difference.The target should be to reduce train and validation weighted sum.We had to reduce both the errors so we try and reduce the sum.

Also weighted sum is to take care that the difference between sum and validation is minimum as a given error which has fallen way too much will lead to the difference increasing.

To reduce the difference we use the max(validation and train).

The functions we used at different points are----

A) error_val[0]+error_val[1];

B) error_val[0]+error_val[1]*(weight);

C)max(error_val[0],error_val[1])

D)sum +max

Crossover function

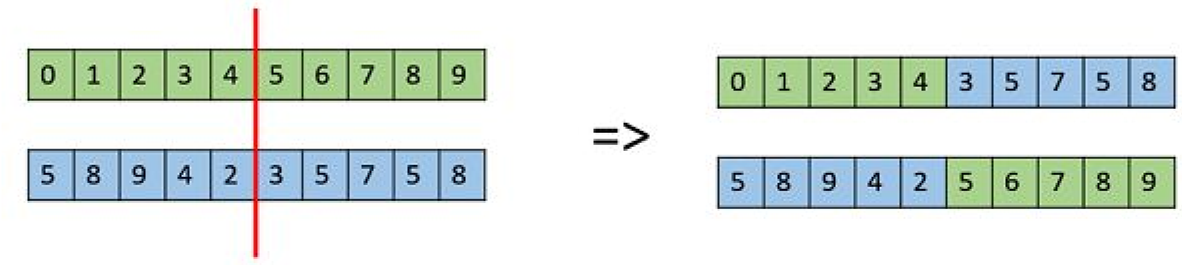
The crossover functions used were choosing a parent probabilistically depending on whether the fitness value is low .Meaning parent with higher probability of selection

Whose fitness value is low. Also the crossover is one point crossover as mentioned above it shows how an offspring is generated.

Following algorithms of crossover were tried:-

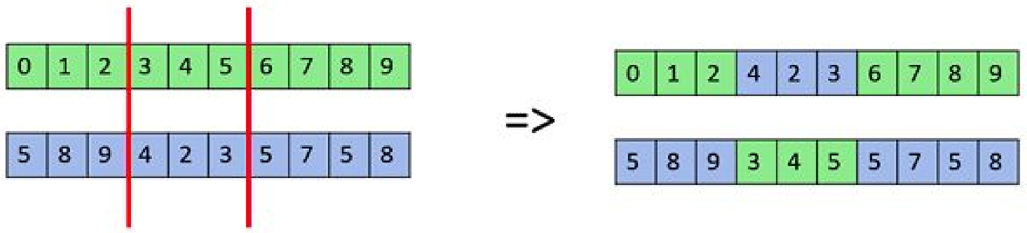
1. One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



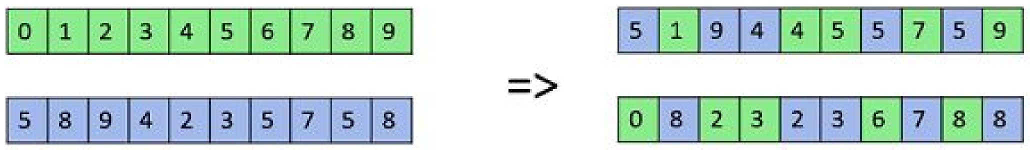
2. Two Point Crossover

Same as One -point crossover instead 3 segments are swapped to generate offsprings.



3. Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.



Mutation Function

The mutation we have used is to change a random value of the vector probabilistically depending on randomly generated value.Also it helps in changing the value in a small range.**Probability of mutation is 0.3**.

Following algorithms of mutations were tried:-

1. **Random Resetting-1**

   Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

   For a given value x **we change to between(0.8x,1.2x) randomly**

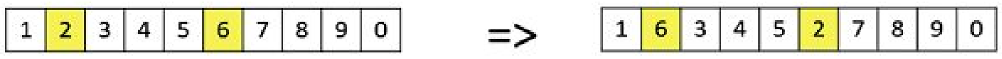
   This mutation has although fluctuated to change the position from a deep local minima.

2. **Random Resetting-2**

   For a given value x we change to between**(0.0009x,1.0001x)** randomly

3. **Swap Algorithm**

   In swap mutation, we select two positions on the chromosome at random, and interchange the values.This algorithm was not much useful because sometimes it creates very unfit offspring.



Hyperparameters-

- Pool size -: Initially we started with a pool size 15 but later changed it to 10 as it was better off if we have as many generations possible to train so that we get an ideal vector.
- Splitting point for new genes has been chosen randomly as this will bring in more diversity in the crossover and might give a better offspring.
- The best population is selected for subsequent training.

Heuristics applied-:

We choose the parents probabilistically depending on their rank in the fitness function using the sort options giving more chance to the parents which have better values according to the fitness function.As mentioned above in the Roulette wheel model

We choose the point of crossover randomly to bring in diversity

The probability to mutate any given vector is .3 and it keeps the value of the element as specified above.

The genetic algorithm converges depending on the overfit vector we are training on.

There might be a steep local minima which is not the global minima.It does not converge in those steep local minimas if the mutation and crossovers can bring some diversity in the population.And after iterating for 30-35 times the population converges to a value which is at the present best and it fills the whole population with these values.It causes the vector to not change it's values it does not essentially converge but it might get stuck in a local minima .

Tricks /brute forces applied-:

Manually tweaking the vector to reduce the difference in train and validation and also reduce both of them independently