Report on genetic algorithm

Team name: MDL-GA-1.

Members:
1.P.s.Raghavendra Reddy(2019101074)
2.Abhinav Chowdary Mallampati(2019101109)

Algorithm functions:

1.Mutation algorithm:

This is the mutation function we used here

mutation_operator(temp,prob, mutate_index):

Temp is the vector that need to be mutated.
Prob is the probability for the gene to be mutated or not in the final algorithm we changed this according to get the best fit. We altered it like 0.099 for every five iterations to get the perfect fit.
Mutate_index is the another extra parameter we included, this is to ensure the range within the gene is mutated.

```
def mutation_operator(temp,prob, mutate_index):
    vector = np.copy(temp)
    for i in range(len(vector)):
        fact=random.uniform(-mutate_index, mutate_index)
        vector[i] = np.random.choice([vector[i]*(fact+1), vector[i]], p=[prob,1-prob])
        if(vector[i]<-10) :
            vector[i]=-10
        elif(vector[i]>10) :
            vector[i]=10

    return vector
```

The above is the total function for the mutation we used.

2.crossover algorithm:

For the cross over we used a five different paraments to ensure to get best out of the function.
Cross over function takes two parent vectors to produce output or children for the current two parents.

Cross_select is the parameter for the selcting no of parents
As for the crossover number here we take randomly here in this we choose 6 and for the postions swapping we chosed randomly and swaped.

```
def crossover(vector1, vector2, mutate_index,mutation_crs, index=-1):
    send1 = vector1.tolist()
    send2 = vector2.tolist()


    a = np.random.choice(np.arange(0, 11),5, replace=False)
```

```
    for i in a.tolist():
        send1[i] = np.copy(vector2[i])
        send2[i] = np.copy(vector1[i])

    return mutation_operator(send1,mutation_crs,mutate_index),
mutation_operator(send2,mutation_crs,mutate_index),send1,send2,a
```

3.fitness function:

As for the fitness function we took err(0)+err(1). As the sum of train and validation error. This is the function that choose whether the current vector will qualify or not. Here lesser the sum of error the better is our function.

How our code works:

Process 1:

Generating inital population

Generating inital population. This is done by random operator. This is our geneartion 1 for algorithm. And pop_size is the population size we randomly took pop_size like about 10 to 30 for different scenarious.

population[i] = np.copy(mutation_operator(temp,0.85,mutate_index))

Process 2:

Errors for this generation

For the above generation we obtain errors and get ready for the procees since these are the parents for the next generation.

perrors_main[j] = np.copy((err[0]+err[1]))

perrors_te[j] = np.copy((err[0]))

perrors_ve[j] = np.copy((err[1]))

here main is the compined error and te and ve are training and validation error.

Process 3:

Doing crossover for the parent generation.

After the parent error obtained we sorted the total gen. Crossover will takes place unti pop_size of children is reached.

The parents chosen from the previous step are sent to the crossover function which returns two vectors. The crossover function returns mutated the child vectors. These 2 child vectors are then appended to the child population.

```
 while(new_gen_count < pop_size):
```

```
arr = crossover_new(perrors_main, cross_select)
temp = crossover(population[arr[0]], population[arr[1]],mutate_index,mutation_crs)
    if temp[0].tolist() == population[arr[0]].tolist() or temp[1].tolist() == population[arr[0]].tolist() or temp[0].tolist() == population[arr[1]].tolist() or temp[1].tolist() == population[arr[1]].tolist():
        continue

    arr_1[int(new_gen_count/2)][0]=np.copy(arr[0])
    arr_1[int(new_gen_count/2)][1]=np.copy(arr[1])
    arr_2[int(new_gen_count/2)]=np.copy(np.sort(temp[4]))

    childrens_list[new_gen_count]=np.copy(temp[2])
    children_errors_mutated[new_gen_count]=np.copy(temp[0])
    child_population[new_gen_count] = np.copy(temp[0])
    new_gen_count += 1

    childrens_list[new_gen_count]=np.copy(temp[3])
    children_errors_mutated[new_gen_count]=np.copy(temp[1])
    child_population[new_gen_count] = np.copy(temp[1])
    new_gen_count += 1
```

Process 4:

Child population Errors.

The newly genearted child population is then used to get errors.

```
for j in range(pop_size):
    temp = child_population[j].tolist()
    err = server.get_errors(SECRETKEY, temp)
    childerrors[j] = np.copy((err[0]+err[1]))
    chidren_te[j] = np.copy((err[0]))
    chidren_ve[j] = np.copy((err[1]))
    childern_error_main[j]=np.copy(childerrors[j])
```

Process 5:

Children sorting

Sorting the child population based on the errors obatined in the process 4.

children_population = np.copy(childerrors.argsort())

childerrors = np.copy(childerrors[children_population[::1]])

chidren_te = np.copy(chidren_te[children_population[::1]])

chidren_ve = np.copy(chidren_ve[children_population[::1]])

child_population = np.copy(child_population[children_population[::1]])

Process 6:

New generation creation

we have two different parents and childrens of pop_size and need to create a final gen for the next proceeding generation.
Here we have surity to keed the best parent and children vectors intact so that we can use them to get still best vectors for the upcoming geneartions.

Here this is shown by

tempstore_main = np.zeros(pop_size)
tempstore_te = np.zeros(pop_size)
tempstore_ve = np.zeros(pop_size)
tempstore= np.zeros((pop_size, MAX_DEG))

The parent and children array is stored into into a singlr parent_child

Here this is shown by

parent_child = np.copy(np.concatenate([population[surity:], child_population[surity:]]))
parnet_child_mainerror = np.copy(np.concatenate([perrors_main[surity:],childerrors[surity:]]))
parent_child_te = np.copy(np.concatenate([perrors_te[surity:], chidren_te[surity:]]))
parent_child_ve = np.copy(np.concatenate([perrors_ve[surity:], chidren_ve[surity:]]))

Process 7:

Proceeding to Next generation

Sorting The above process array and storing the error

Here it is shown by as follows

parent_child_sorted = parnet_child_mainerror.argsort()

Process 8:

updating final best fit vector and error evauation

comparing best gen error and updating the final submitting vector.

```
if(min_main_error == -1 or min_main_error > perrors_main[0]):
        final_vector_best = np.copy(population[0])
        min_main_error = np.copy(perrors_main[0])
        min_tainingerror = np.copy(perrors_te[0])
        min_validationerror = np.copy(perrors_ve[0])
        nochange=0

    else:
        print("nothing imporved doing next phase")
        nochange+=1
        print(nochange)
    print("/././/././/././/././/././/././/././/./\n")
    print("Min error = ", min_main_error, "\n\n")
```

Process 9:

Rinse and repeat:)

Doing the phases for 3 to 8 again as per the gen_count here.....

Note:Our best fit vector is came from choosing this parameters pop_size = 10 and gen_count = 50. This is our best fit vector

```
[ 0.00000000e+00, -1.38265250e-12, -2.22702223e-13, 5.32870696e-11,
-1.51978883e-10, -2.06315205e-15, 9.06085461e-16, 2.53922108e-05,
-2.14173024e-06, -1.34763497e-08, 9.64856003e-10].
```