MDL Assignment 3 Genetic Algorithms

Team Number - **15**

Kushagra Agarwal *2018113012 -CND* **Shreeya Pahune** *2018113011-CND*

1) Initial Population

We used the overfit vector given to start. To expand it to a population size of the starting generation, we used **broadcasting** to add a random mutation to our vector and make the starting generation. The mutation added was chosen to be the order of **np.random.randn()/10^{-13}**, as the smallest value in the given vector was of the order 10^{-12} and we did not want to corrupt any value at the very start of the iteration.

```
def genalgo(key,iter=10,shape=10,prob=0.2):

    '''This function loops over various iterations and executes the genetic algorithm'''

    givenarray1=[0.0, 0.1240317450077846, -6.211941063144333, 0.04933903144709126, 0.03810848157715883,
    initarray=np.random.randn(shape,11)*(10**-13)
```

```
initial_array= initarray + givenarray1 #perturbation introduced
```

The initarray now has a shape of (shape,11) and this will be our first generation for the algorithm

```
    assert(initial_array.shape == (shape,11))
```

2) Fitness function for the whole generation

```
    for i in range(shape):
        cost=get_errors(key,list(initial_array[i]))
        cost_array[i]=0.75*cost[0]+cost[1]

    assert(cost_array.shape == (shape,1))
```

The cost_array stores the values of costs for each vector in the generation. We used the cost to be **0.75*train_error + 1*validation_error.** This equation was tweaked many times according to

the values obtained. The reason behind this weighted sum is the following: We noticed that when close to certain values, the train error decreases faster than the validation error, which eventually leads to an increase in the validation error as there is a tradeoff between both the errors. We concluded that close to certain local minimas, there is a steeper curve for train error and so weighting with coefficients was used to average out these effects. When we moved out of such minimas, we used a simple **train_error + validation_error** as the cost function. Fitness has been taken to be inversely proportional to the cost function.

---

3) Choosing two parents for each Child

```
for i in range(shape):
    [arr1,arr2]=random_selection(cost_array,initial_array,shape)
```

We found the two parents for each child using the function random_selection which inputs the current generation (initial_array), the fitness of each of the vectors in it (cost_array) and the population size (shape). It returns the two parents for each child (arr1, arr2).

The random_selection function was tried using two implementations, and the **Method1** was the one finally used.

Method 1:

```
def random_selection(cost_array,initial_array,shape):

    '''This function selects two parents based on their fitness and some probabilistic function'''
    cost_array_new2=np.random.randn(shape,1)
    cost_array_new1=[]

    sum_array=np.sum(cost_array)
    mean=np.mean(cost_array)
    std_dev=np.std(cost_array)
    epsilon=10**-14

    #Method number one
    for i in range(shape):
        cost_array_new2[i]=(cost_array[i]-mean)/(std_dev + epsilon) + np.random.randn()/10

    [ind1,ind2]=sorted(range(len(cost_array_new2)), key=lambda i: cost_array_new2[i],reverse=True)[-2:]
```

We calculate the **mean** and **standard deviation** for the cost_array input to the function to make the distribution of the cost_array input a **Gaussian distribution** with mean 0 and variance 1. Then we **add a random number divided by 10** to it. The **epsilon** was added to remove any possible chance of division by zero. The distribution was made gaussian so that the random number added is of similar range. Now to not let the random number dictate the choice, we divide it by a factor of 10 to make it a secondary contributor.

```
return([initial_array[ind1],initial_array[ind2]])
```

Then we choose the top two vectors with the minimum cost to be returned back.

Method 2:

This method wasn't used in the final algorithm as it brought in a lot of randomness in the choice of parents which was not appreciated as there was a lot of noise getting added in the choice of parents.

```
#Method number two
for i in range(shape):
    cost_array_new1.append(cost_array[i][0]/sum_array)

output=np.random.choice(range(shape),size=2,replace=True,p=cost_array_new1)
# return(initial_array[output[0]],initial_array[output[1]])
```

This method chooses 2 parents based on their Probability function which is = **fitness[i]/sum(fitness)**. The replace parameter was set to True so that the same vector is not chosen twice.

---

4) Creating child from Parents

```
assert(arr1.shape == (11,1))
assert(arr2.shape == (11,1))

child=child_maker(arr1,arr2)
```

The **child_maker** function takes two parents as input and creates a child vector as output.

```
def child_maker(arr1,arr2):

    '''This function joins two arrays arr1 and arr2 at the crossover point to form a child array'''
    arr3=np.random.randn(11,1)
    c=np.random.randint(0,11) # choosing the crossover point
    d=np.random.randint(0,2) # choosing which one will be the first parent and which one the second

    if(d==0):
        temp=arr1
        arr1=arr2
        arr2=temp

    assert(arr1.shape == (11,1))
    assert(arr2.shape == (11,1))

    for i in range(11):
        if ( i<=c):
            arr3[i]=arr1[i]
        else:
            arr3[i]=arr2[i]
    return(arr3)
```

It first decides the **crossover point c** which is a random number between 0 and 10. Then it **decides the order of the parents** with the variable d. If d==0 then the arr2 becomes the first parent and vice versa. To do this, we swap arr1 with arr2 if d==0

Then till the crossover point, the parent1 is copied into the child and after it the parent2 is copied Then it returns the child vector thus made.

---

5) Mutating the new child

```
    child=mutate(child,prob)

    assert(child.shape == (11,1))
```

The **mutate** function is used to mutate the child with a **probability prob** sent as a parameter. It returns the mutated child.

```
def mutate(child,prob):

    '''This function mutates the child array according to the probability specified'''
    for i in range(11):
        if(np.random.rand()<prob):
            child[i]=child[i]*(1+np.random.randn()/100)
        if(abs(child[i])>=10):
            child[i]=child[i]*(1- abs(np.random.randn()/100))

    return child
```

For each value in the child vector, we mutate it with a probability sent as a parameter to the function. The mutated value is **= originalvalue*(1+np.random.randn()/100**) which gives us a new value which can be both greater than or lesser than the original value as np.random.randn() can generate both positive and negative values. In case any value in the child vector **crosses the range -10 to 10,** then it is **reduced** a bit so as to fall in the specified range. The parameter /100 was also tweaked multiple times during the iterations. When the algorithm appeared to have **stuck in a local minima** then, the value was **reduced to 10**. During normal operations, it was kept at 100 to make the mutation incremental and not abrupt.

---

6) Creating the new generation

```
    for j in range(11):
        new_array[i][j]= child[j]

assert(new_array.shape == (shape,11))
```

The array **new_array** contains the **newly generated population** with population_size(shape) number of children.

```
    #validating the iteration
    for i in range(shape):

        cost=get_errors(key,list(new_array[i]))
        print(submit(key,list(new_array[i])))

        new_cost_array[i]=0.75*cost[0]+cost[1]
        actual_cost1[i]=cost[0]
        actual_cost2[i]=cost[1]
```

The **costs were recalculated** for the new population using the previously defined cost function and stored in the array new_cost_array which is of dimensions (shape,1) like the cost_array.

```
[ind1,ind2,ind3,ind4,ind5]=sorted(range(len(cost_array)), key=lambda i: cost_array[i],reverse=True)[-5:]
[ind6,ind7,ind8,ind9,ind10]=sorted(range(len(new_cost_array)), key=lambda i: new_cost_array[i],reverse=True)[-5:]
```

Ind1 to ind5 stores the best 5 vectors in the parent population.
Ind6 to ind10 stores the best 5 vectors in the newly created child population.

```
final_array=np.random.randn(shape,11)
final_array[0]=initial_array[ind1]
final_array[1]=initial_array[ind2]
final_array[2]=initial_array[ind3]
final_array[3]=initial_array[ind4]
final_array[4]=initial_array[ind5]
final_array[5]=new_array[ind6]
final_array[6]=new_array[ind7]
final_array[7]=new_array[ind8]
final_array[8]=new_array[ind9]
final_array[9]=new_array[ind10]

initial_array=np.copy(final_array)
```

The **best 5 child vectors** and the **best 5 parent vectors** were used to create the final population which was **copied to the initial_array** which goes and repeats all the above steps 2-6 for the next iteration.

Hyperparameters Used

1) **Population_size:** We have used the population size as 10 after having experimented with 15 and 20 as well. This choice gives us the advantage to run it for more number of iterations without exhausting our requests for the day.
2) **Probability_mutation:** The probability of mutation was kept at 0.2 to start with and when the algorithm got stuck at a local minima, we used a value of 0.5 to increase the probability of it getting enough activation to cross it.
3) **Crossover point**: The crossover point is a random number so that a new child can be generated even if the same two parents get selected by the random_selection function.

Statistical Inference

According to us, the vector that performed the best on the test set would be this:

[-9.94699954e+00, 9.94936112e+00, -6.06250670e+00, 4.60365302e-02, 3.80574614e-02, 7.81803738e-05, -6.01169291e-05, -1.24327056e-07, 3.48072352e-08, 4.05227487e-11, -6.70037317e-12]

All the inferences have been made assuming the same dataset size for train and validation.

The train error was close to 7.77 lacs, while the validation error was close to 9.6 lacs for this vector. We got better values for both validation and train errors individually, yet we think that this would perform better on the following counts:

1) Train error is lower than the validation error, which should indeed happen as the algorithm shouldn't perform better on unseen data as compared to seen one.
2) The difference between train and validation errors is 1.8 lacs = 23% of train error, which falls in the acceptable range
3) The train error is sufficiently increased from the 72000 that we started with. This stands to say that the algorithm is no more overfitting the dataset. This can be further validated with the value of validation error which is in a similar range.
4) We can also conclude that the algorithm is not underfitting the data as the train error isn't very high, which would have been an indication of an underfitting population.

It took us approximately 600 iterations to converge to this value. Due to the less amount of iterations that our algorithm got to optimise, we still had scope for improving the train and validation errors.

Heuristics applied

1) We realised that the initial population must not be identical or else the convergence takes a lot more time. To solve this issue we added a mutation of the order **np.random.randn()/10^{-13}** to the given overfit array to form the initial population.
2) Like we have mentioned in the code explanation for Fitness function (Point 2). We started our algorithm with a cost function = **train_error + validation_error.** Once we converged enough, we started to observe that the train error was reducing much faster than the validation_error and hence we started using a weighted cost function.
   We used a cost function = **0.9*train_error + 1*validation_error** for some time. When the same issue arose again, we further decreased the coefficient of train error and eventually brought it down to **0.75.**
3) We **mapped the cost array to a Gaussian distribution** to **reduce range disparities** between it and the random number that we add during random selection of 2 parents. We also figured that the algorithm was failing as the distribution was going towards infinity due to division by 0, and hence added the **epsilon** term in the equation. The value of the **dividing factor for the random number** was also tweaked multiple times to get to this optimal value for us.
4) The **crossover point** was **initially set to 5**, but this was distorting the true randomness with which the children should inherit characteristics from their parents. We then kept it as a **random number** so that a new child can be generated even if the same two parents get selected by the random_selection function to improve the results.
5) The **order in which parents** give characteristics to the children was important as it defines the order of weights in the particular child, and we observed on **randomising** it, we got more diverse children and consequently better results.
6) We started with **mutation function = np.random.randn()/100 + original_value**. This turned out to work fine, but then when the difference in ranges of the values in the child array increased, the performance deteriorated. To solve the issue we made the mutation function a **multiplicative factor to the current value instead of the additive factor to cancel out range disparity effects.**

Diagram

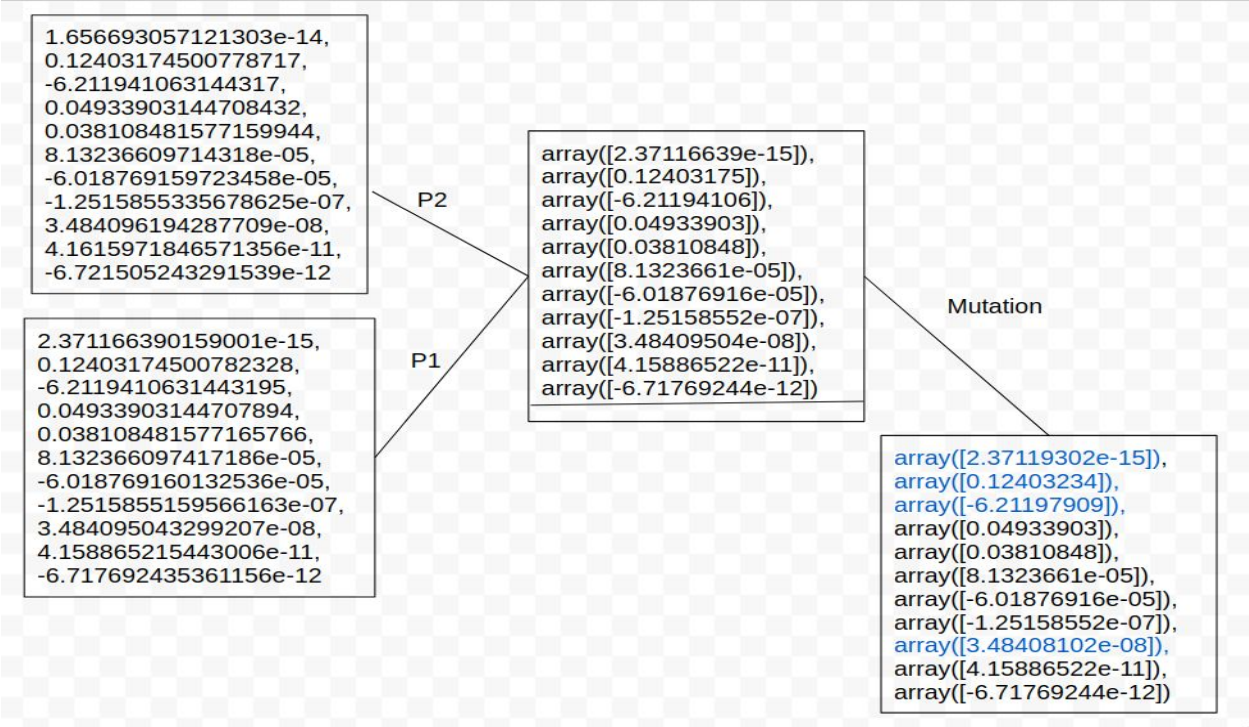
P1 represents **Parent1** and P2 represents **Parent2**
The point of crossover is marked with a **line in the child array**
The **coloured weights** in the **mutated** Child are the ones which got mutated

Iteration 1:
Train Cost: 148219.5240240789
Validation Cost: 2704786.071978009

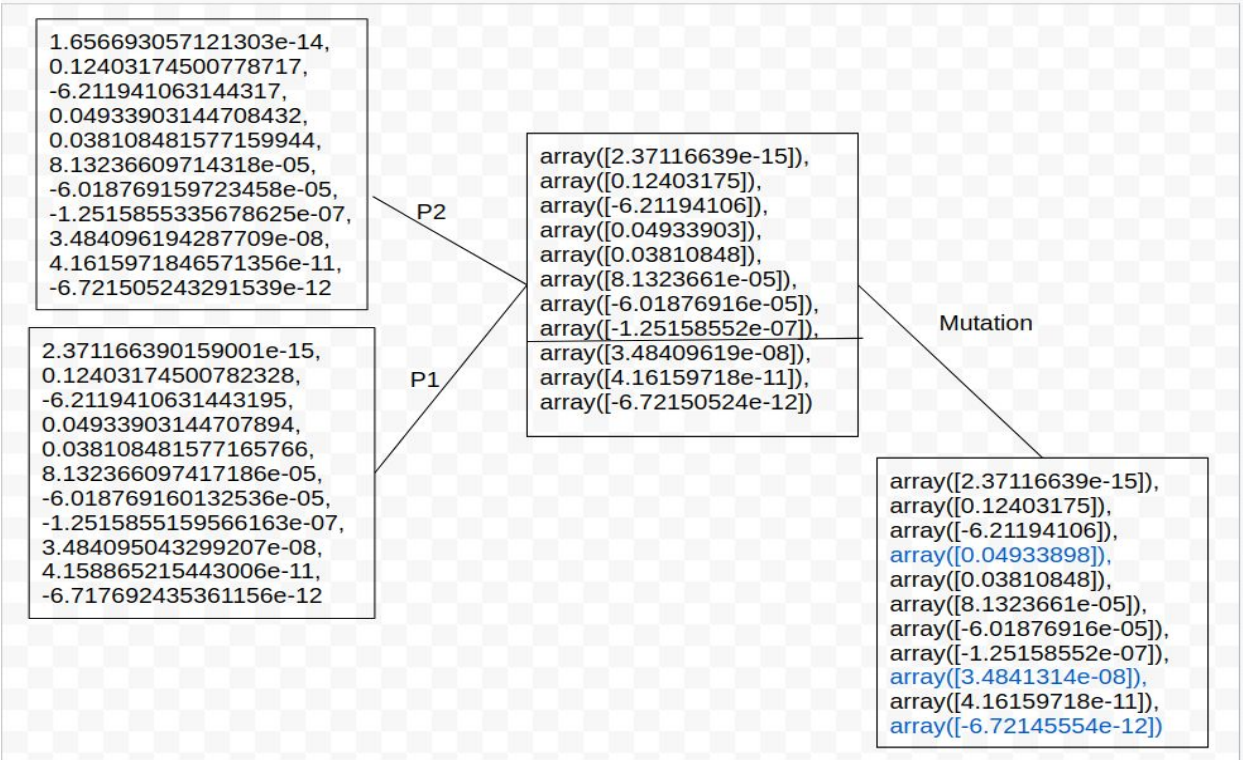
Parent Generation for Iteration 1

-1.05844783e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158571e-07 3.48409620e-08 4.16186726e-11 -6.73900535e-12	-1.50805731e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158576e-07 3.48409603e-08 4.15920192e-11 -6.73781282e-12	1.03562265e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158565e-07 3.48409748e-08 4.15957807e-11 -6.72783961e-12	9.11724352e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158565e-07 3.48409687e-08 4.16263265e-11 -6.73342114e-12	1.65669306e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158553e-07 3.48409619e-08 4.16159718e-11 -6.72150524e-12
4.02232884e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158560e-07 3.48409525e-08 4.16171667e-11 -6.73702101e-12	1.38536437e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158562e-07 3.48409613e-08 4.16133890e-11 -6.73802776e-12	1.09315518e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158555e-07 3.48409629e-08 4.16086369e-11 -6.73696328e-12	2.37116639e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886522e-11 -6.71769244e-12	-4.17872572e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158571e-07 3.48409498e-08 4.16241884e-11 -6.72728334e-12

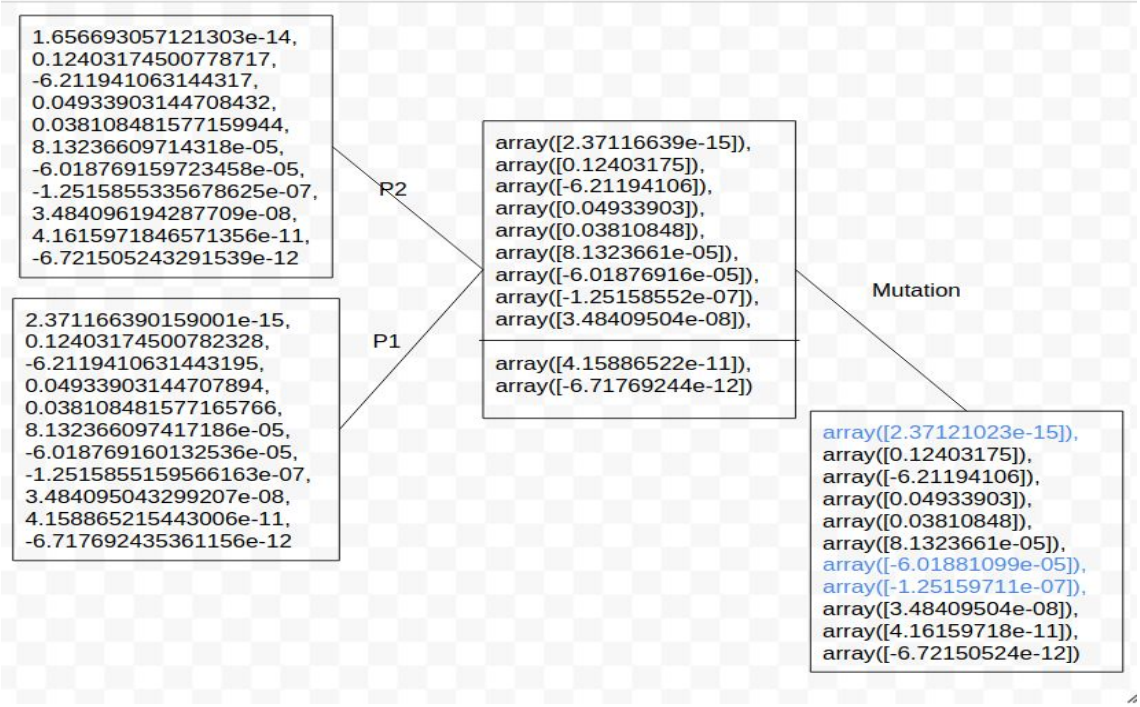
Child1



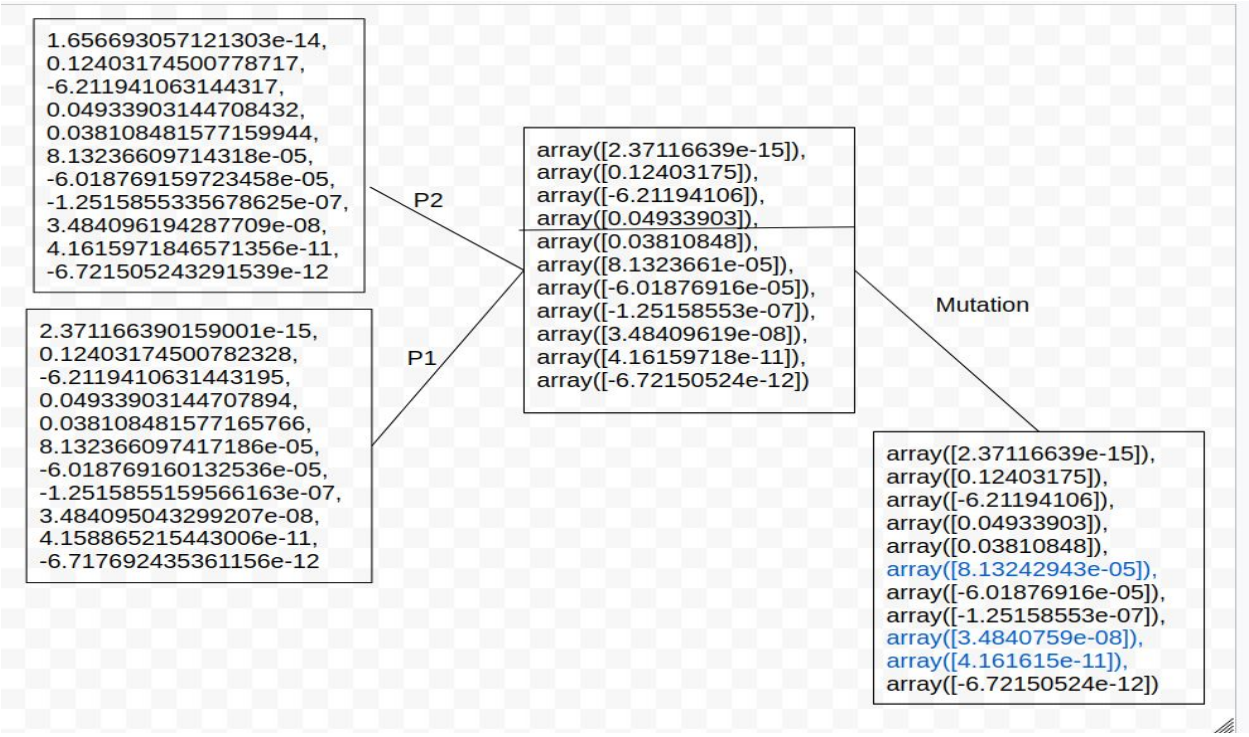
Child2



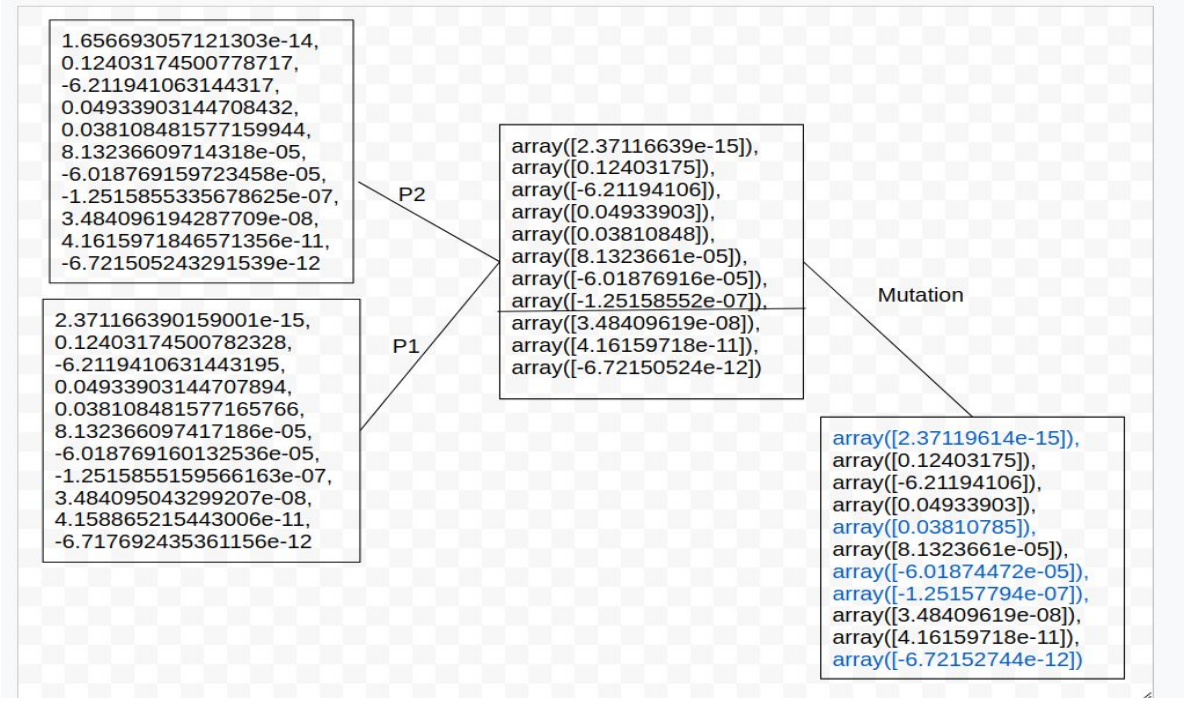
Child3



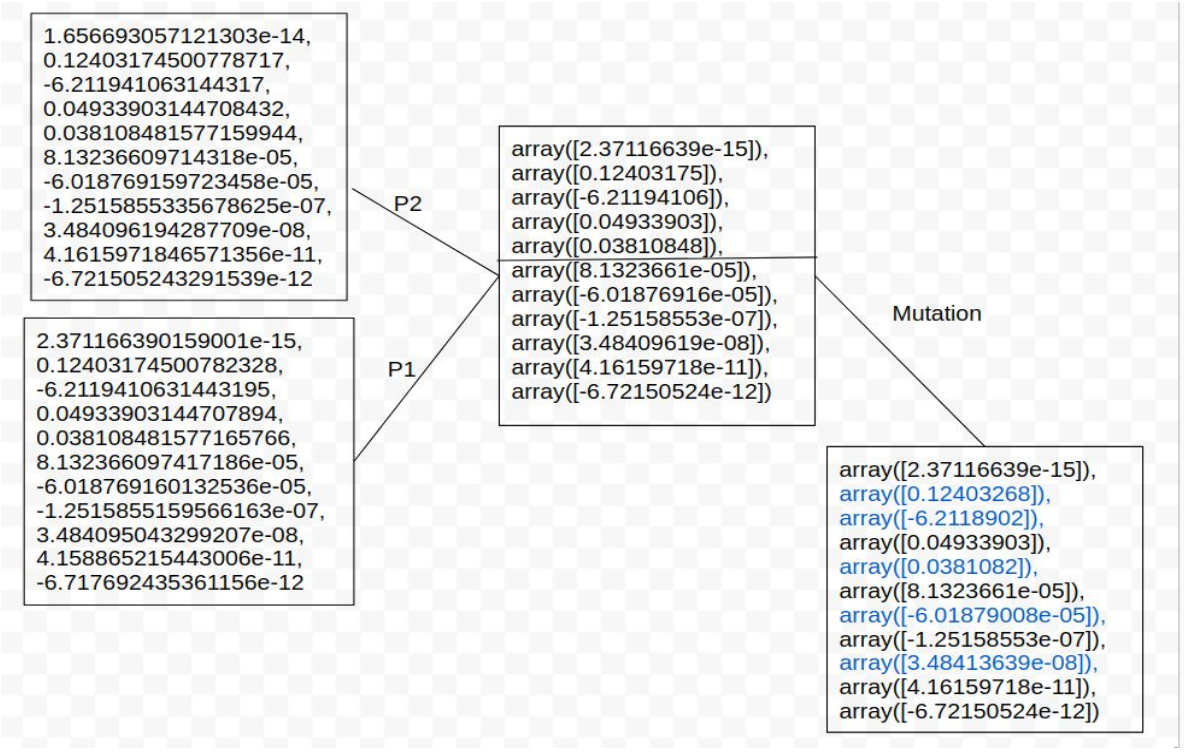
Child4



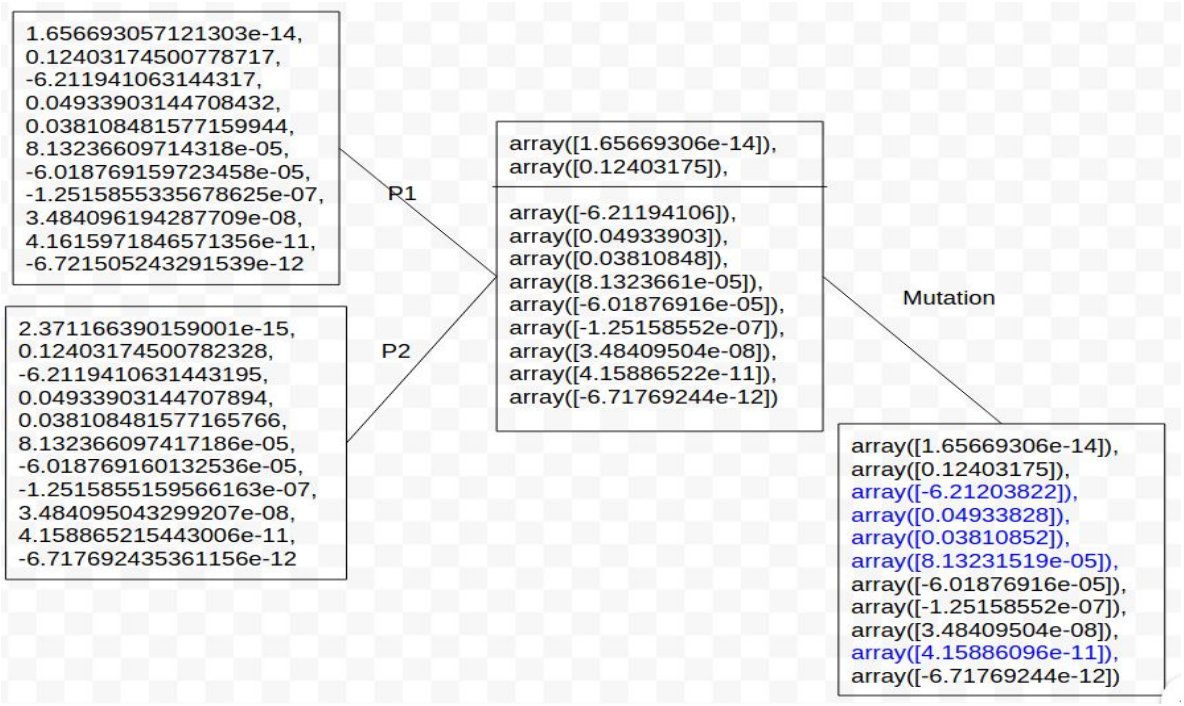
Child5



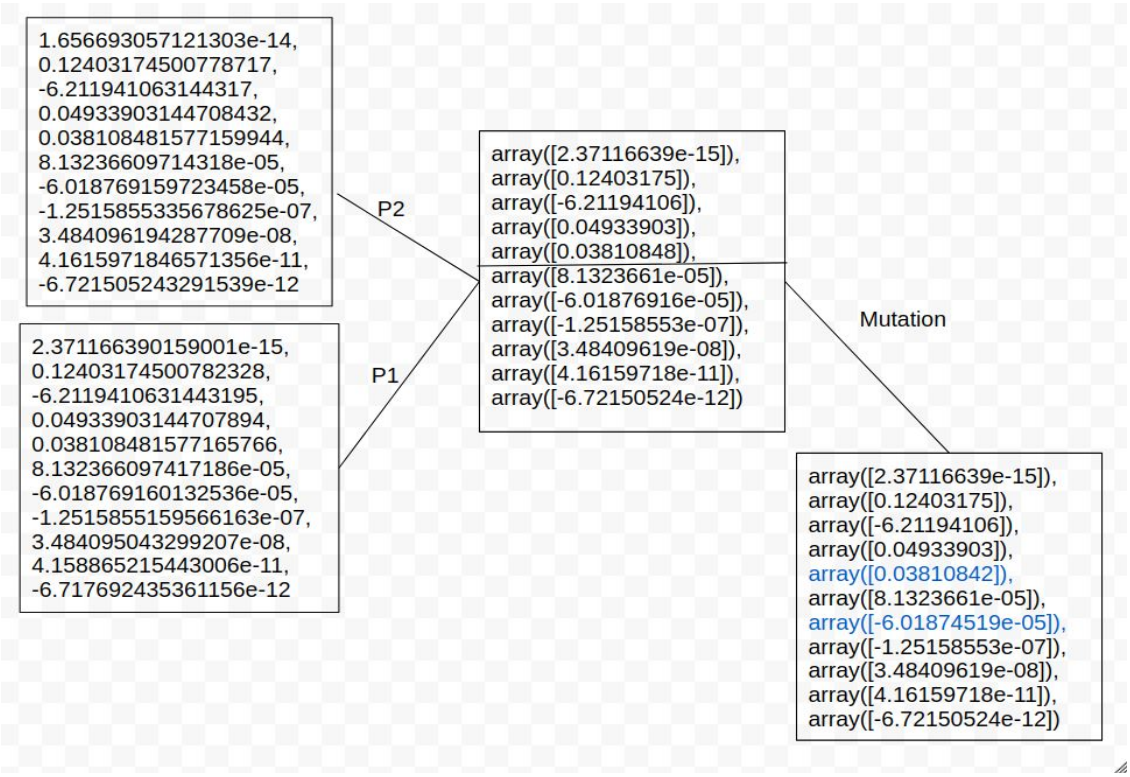
Child6



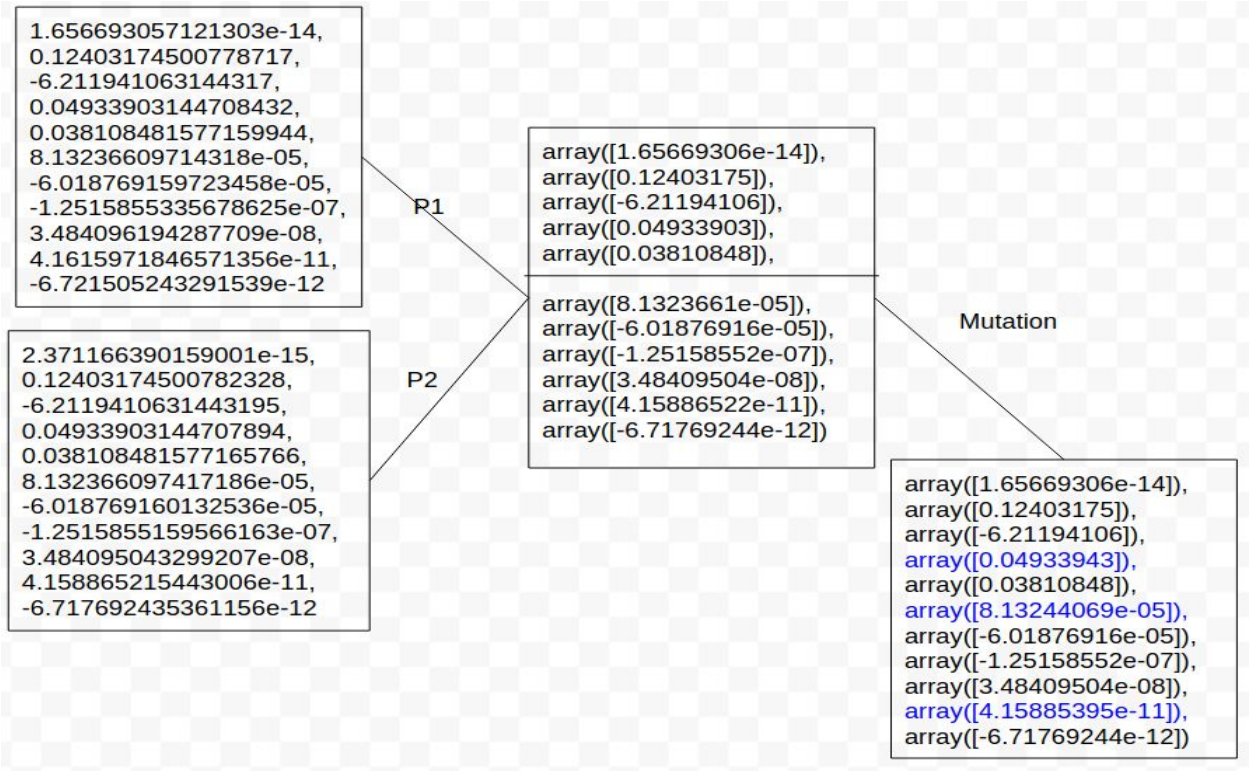
Child7



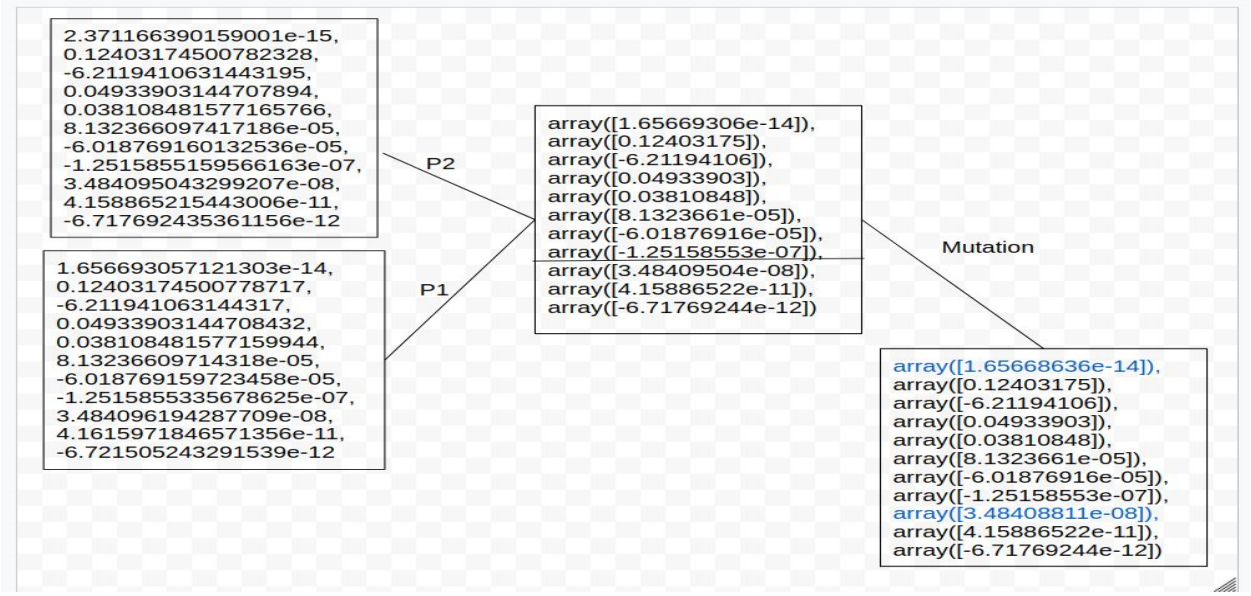
Child8



Child 9



Child 10

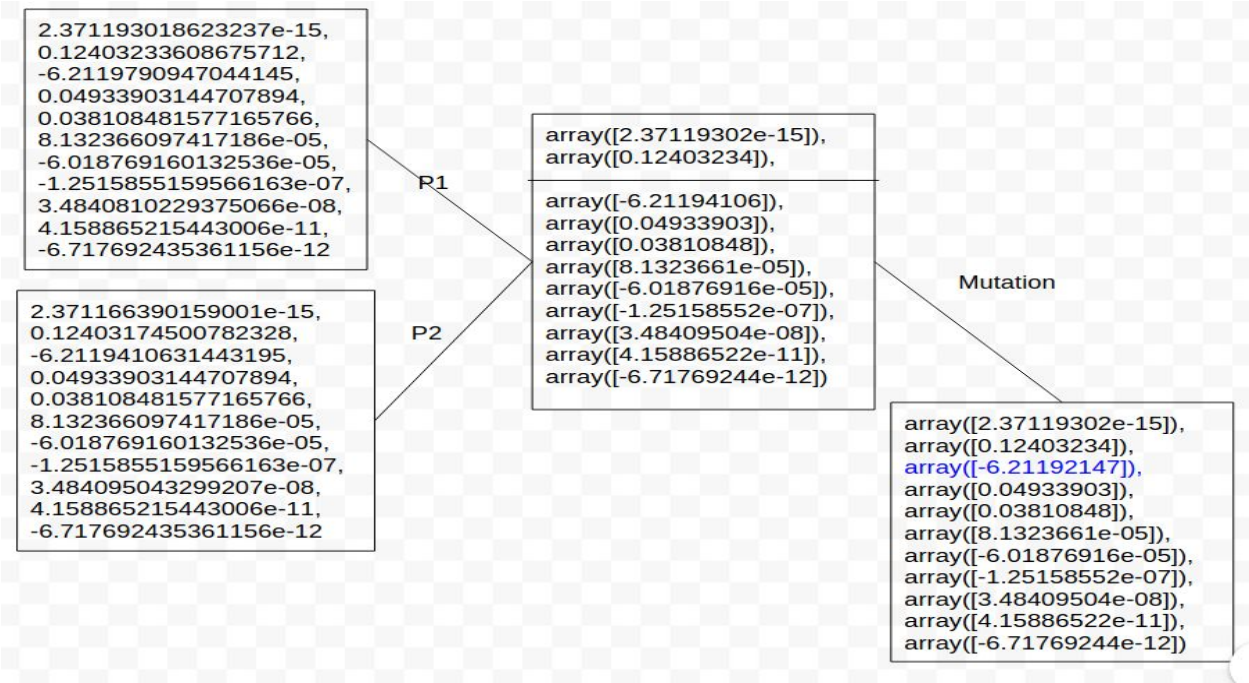


Iteration 2:
Train Cost: 150463.74431823991
Validation Cost: 2696951.371552896

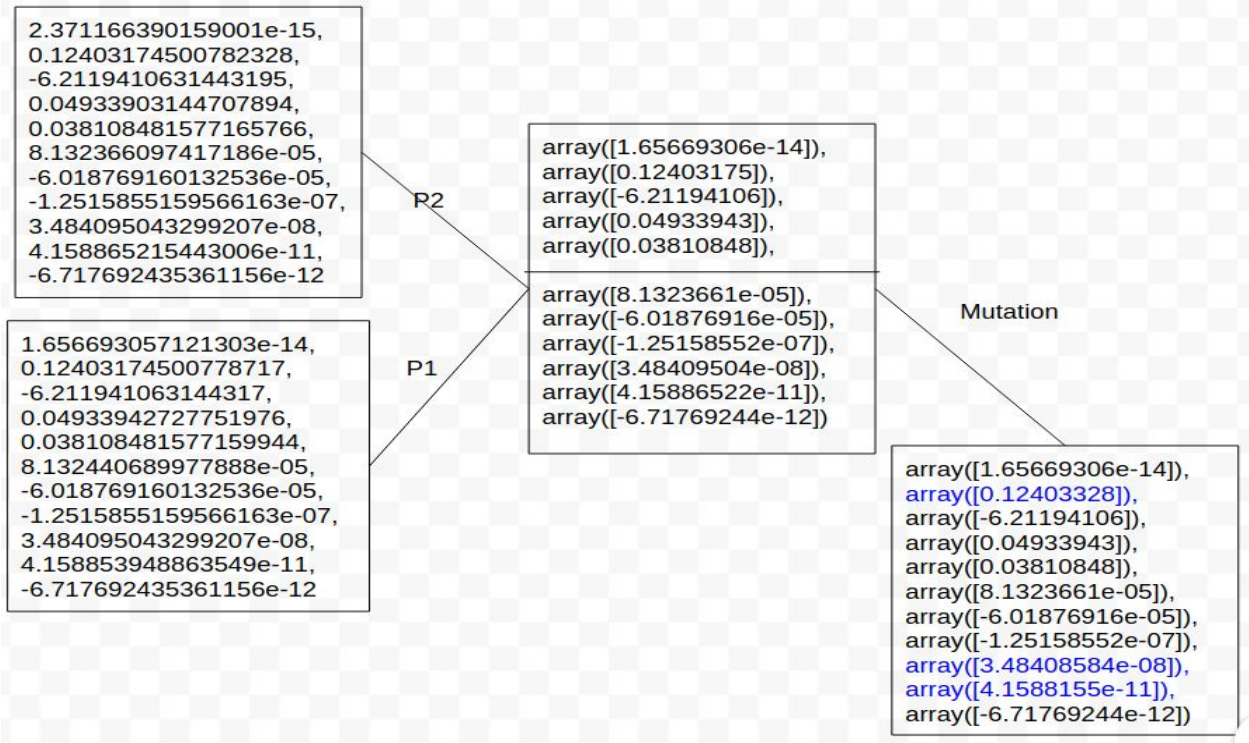
Parent Generation for Iteration 2

9.11724352e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158565e-07 3.48409687e-08 4.16263265e-11 -6.73342114e-12	-4.17872572e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158571e-07 3.48409498e-08 4.16241884e-11 -6.72728334e-12	1.03562265e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158565e-07 3.48409748e-08 4.15957807e-11 -6.72783961e-12	1.65669306e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158553e-07 3.48409619e-08 4.16159718e-11 -6.72150524e-12	2.37116639e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886522e-11 -6.71769244e-12
2.37116639e-15 1.24031745e-01 -6.21194106e+00 4.93389801e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158552e-07 3.48413140e-08 4.16159718e-11 -6.72145554e-12	2.37119302e-15 1.24032336e-01 -6.21197909e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158552e-07 3.48408102e-08 4.15886522e-11 -6.71769244e-12	1.65668636e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158553e-07 3.48408811e-08 4.15886522e-11 -6.71769244e-12	1.65669306e-14 1.24031745e-01 -6.21194106e+00 4.93394273e-02 3.81084816e-02 8.13244069e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15885395e-11 -6.71769244e-12	1.65669306e-14 1.24031745e-01 -6.21203822e+00 4.93382825e-02 3.81085152e-02 8.13231519e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886096e-11 -6.71769244e-12

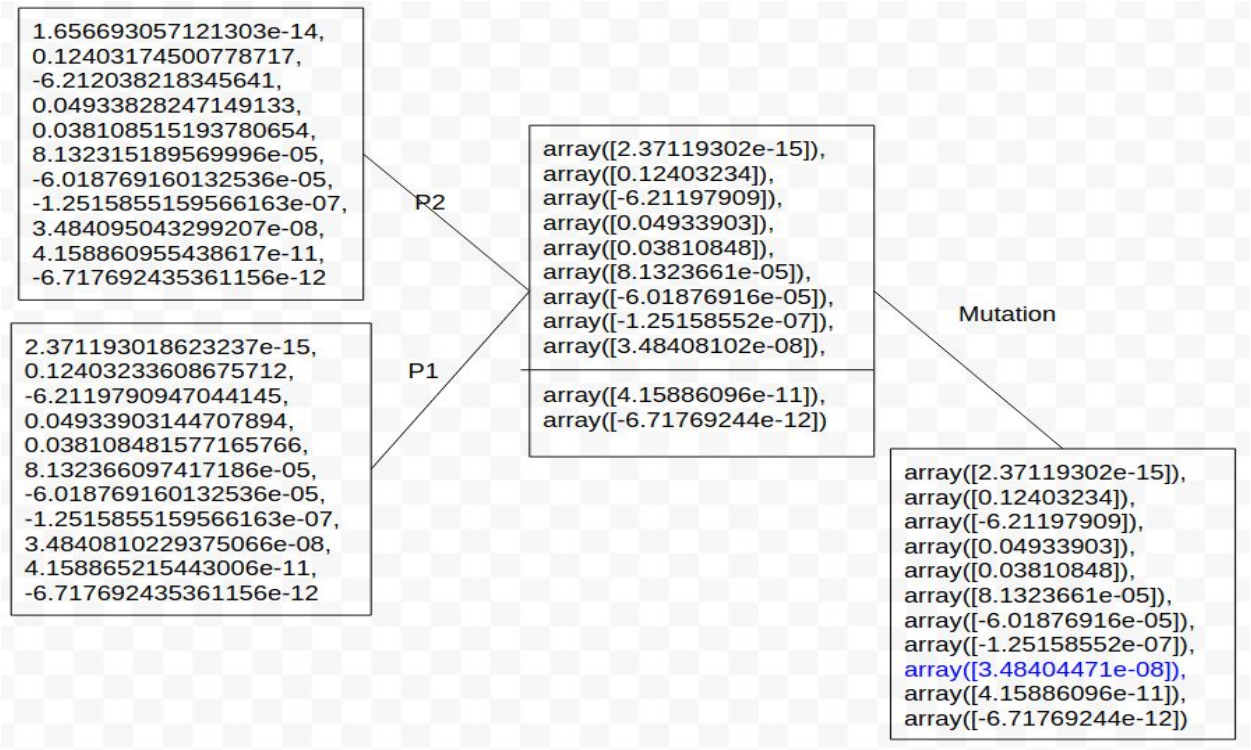
Child 1



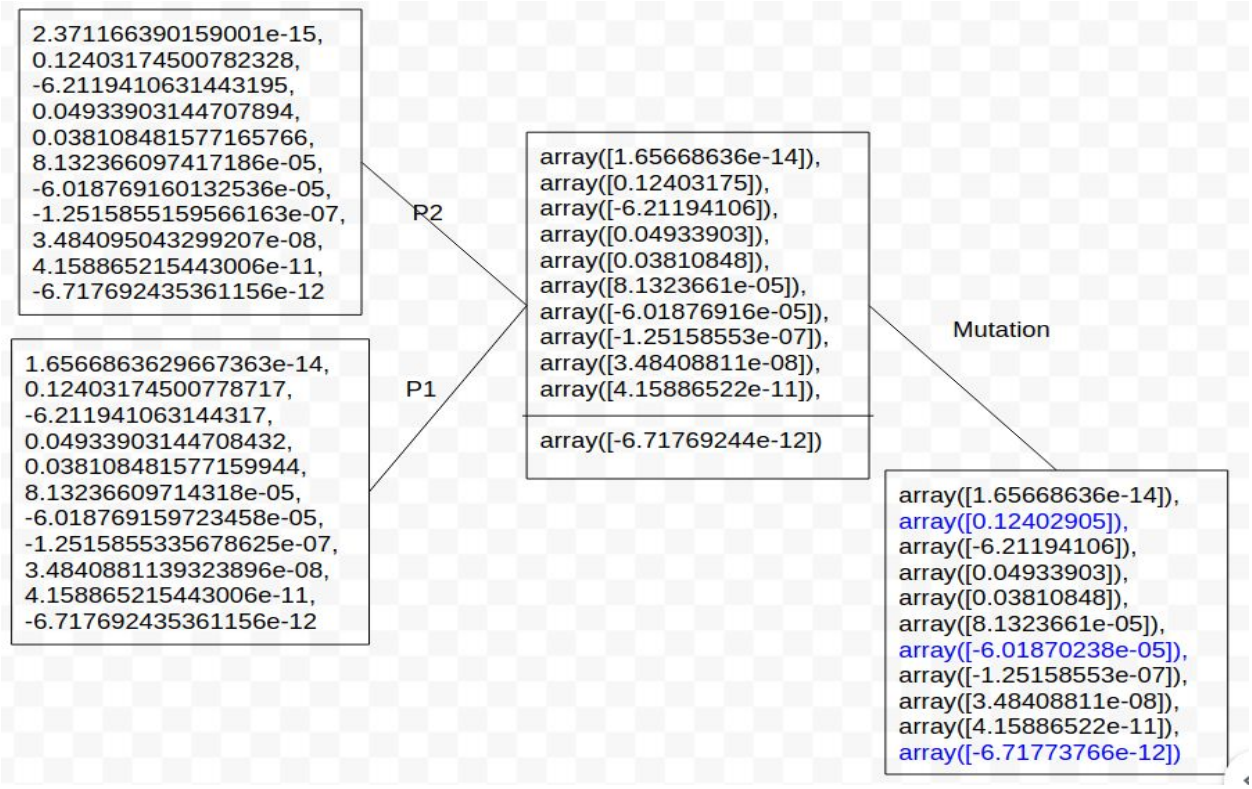
Child 2



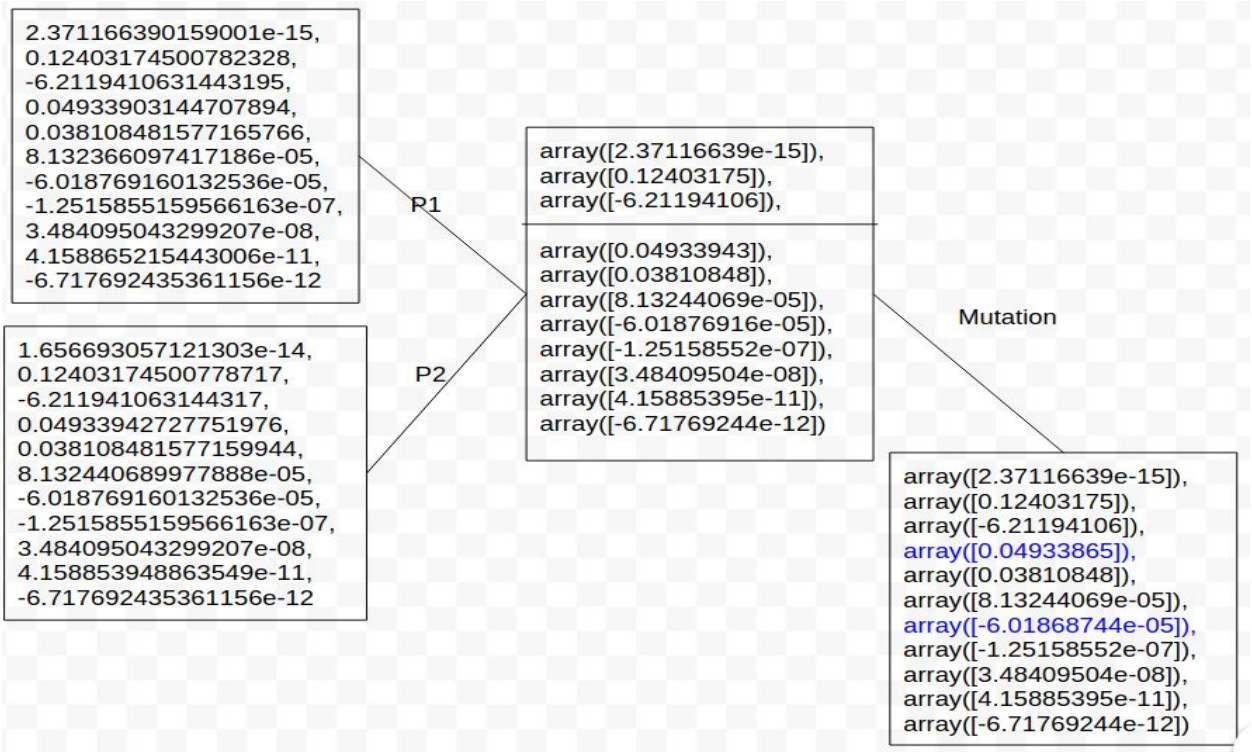
Child 3



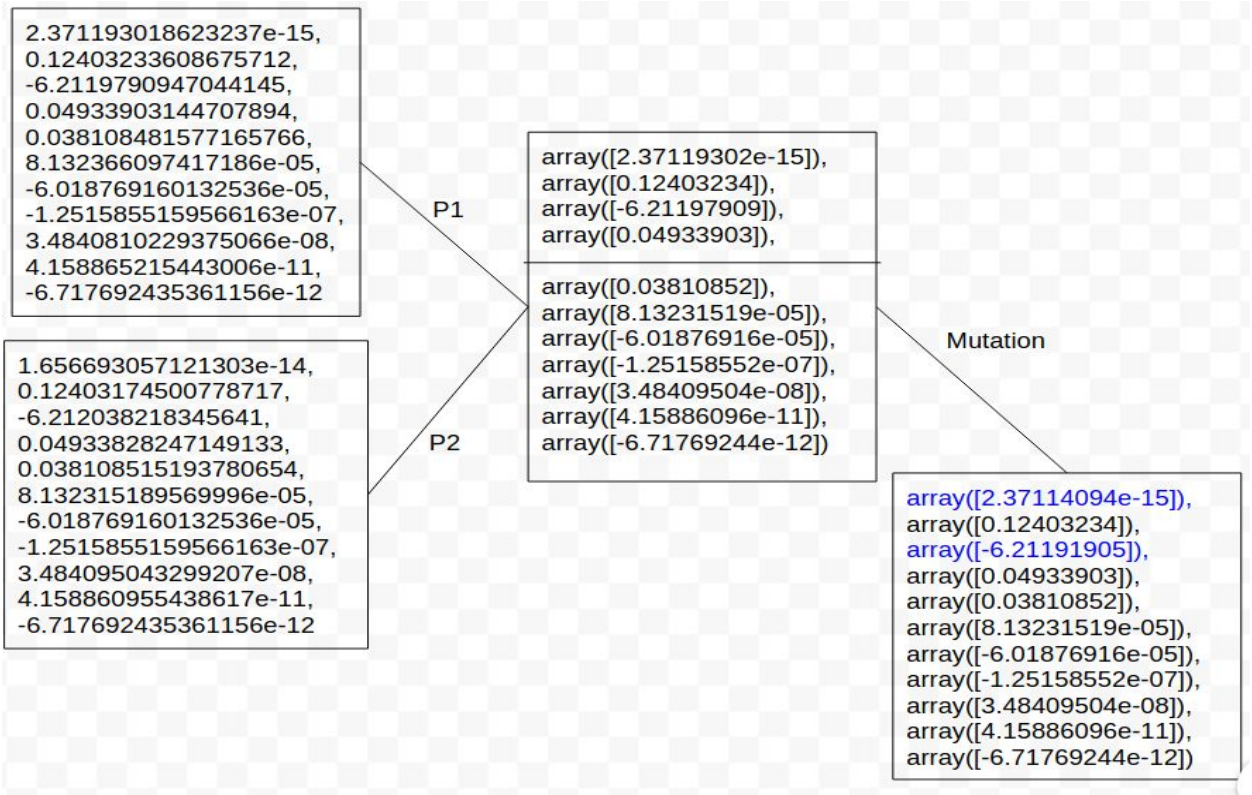
Child 4



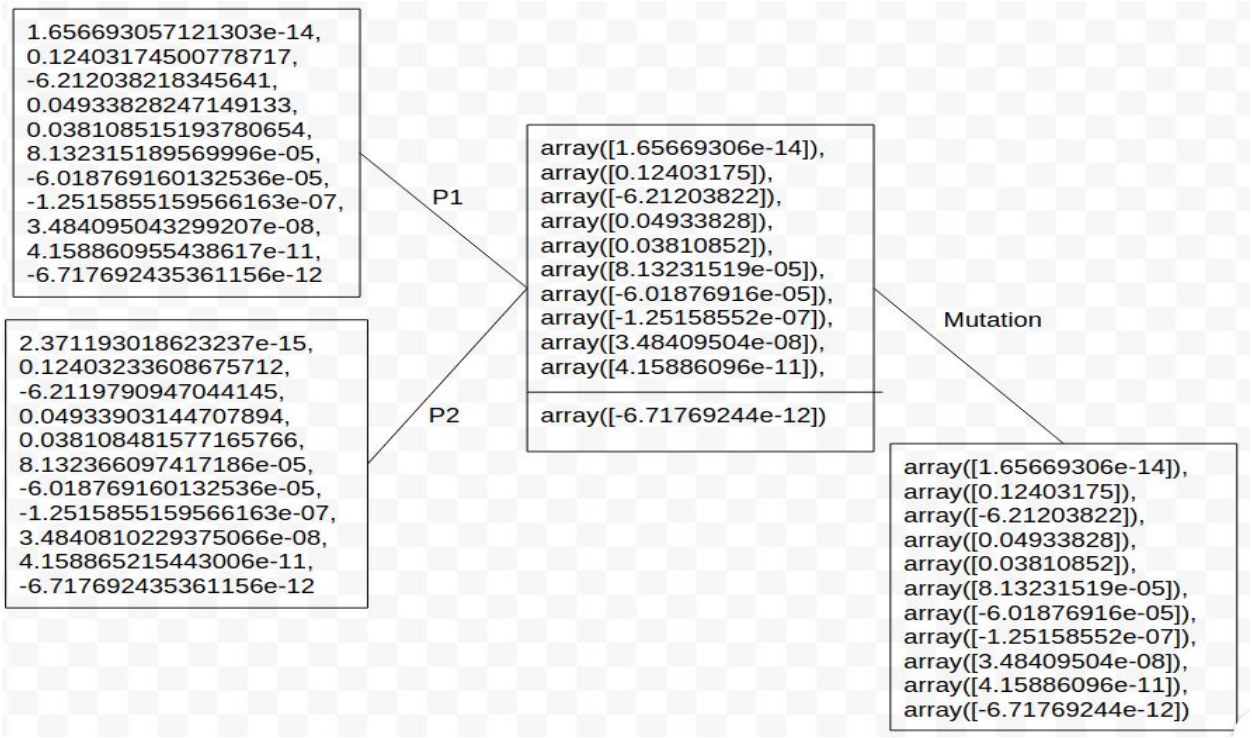
Child 5



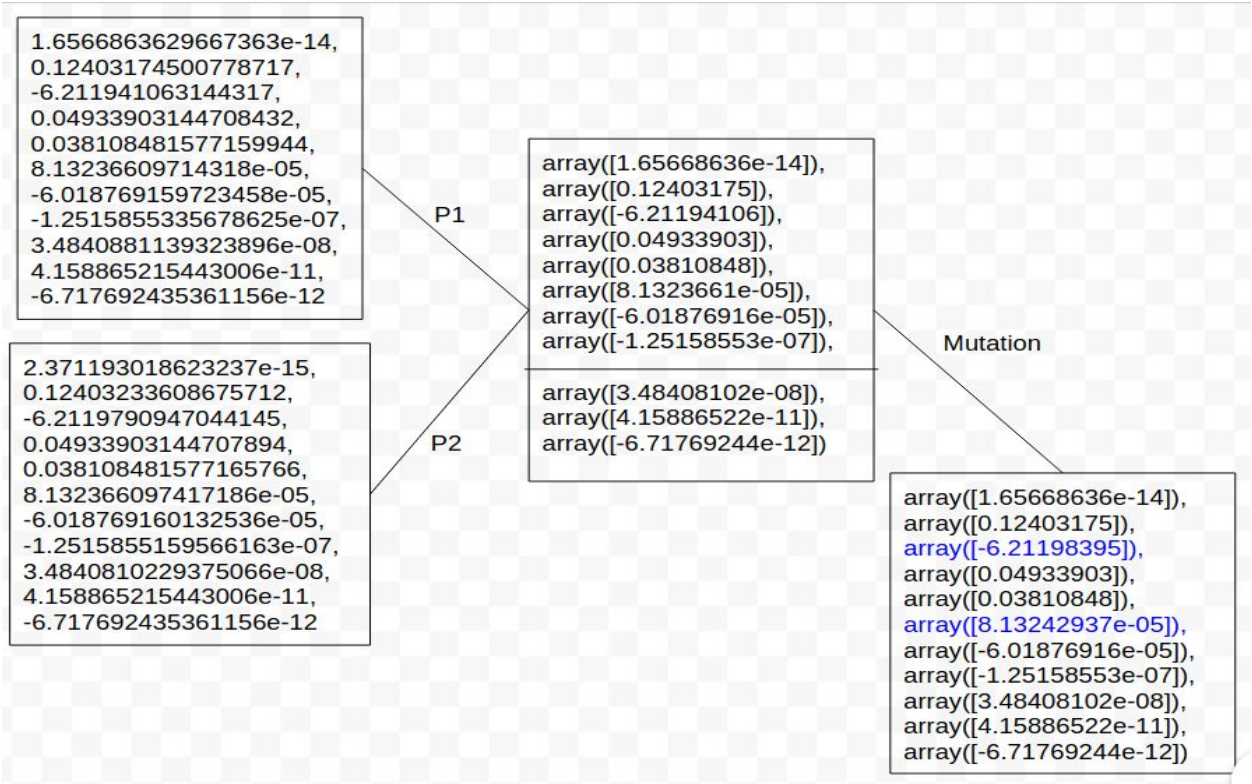
Child 6



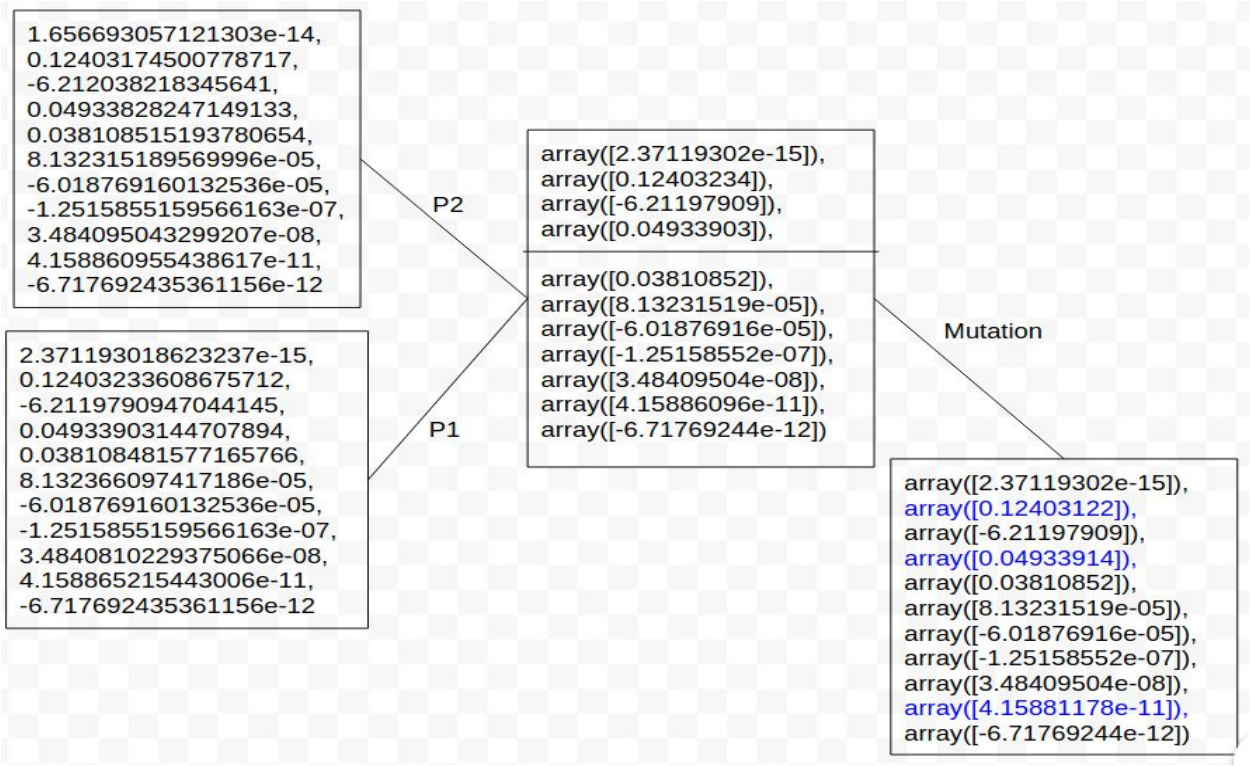
Child 7



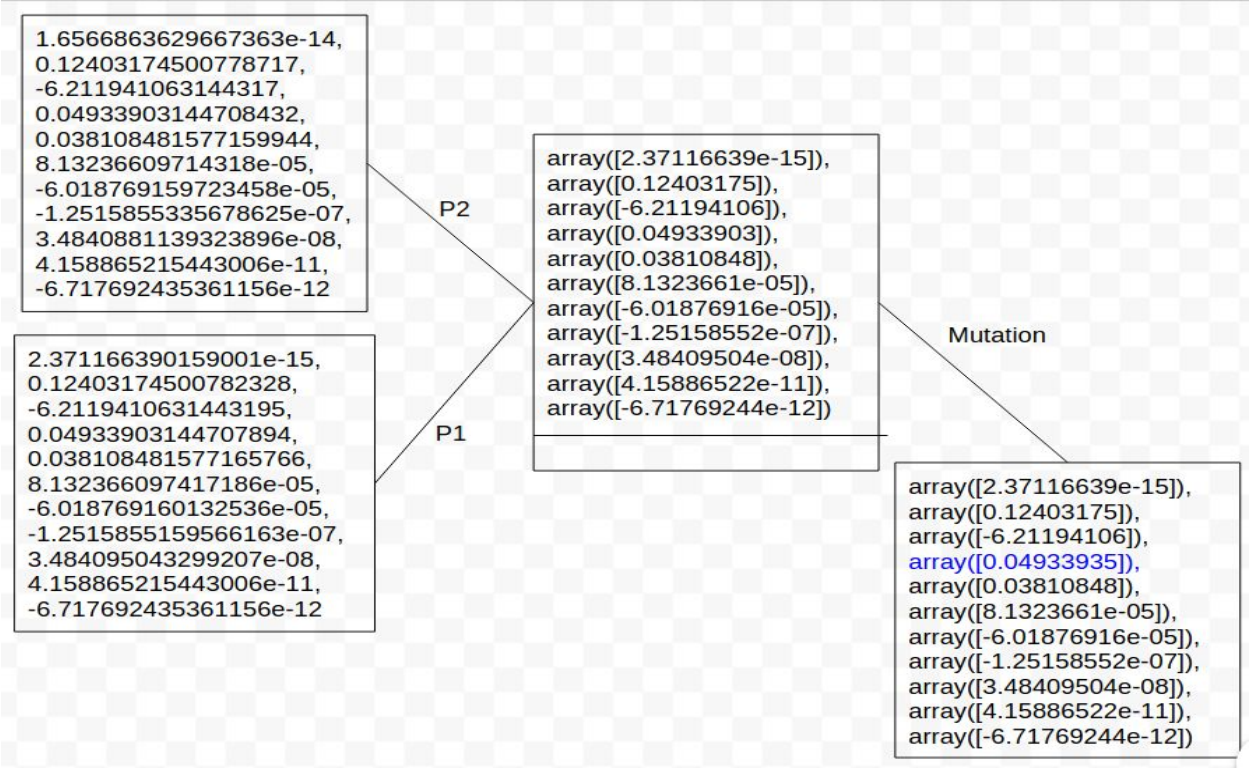
Child 8



Child 9



Child 10

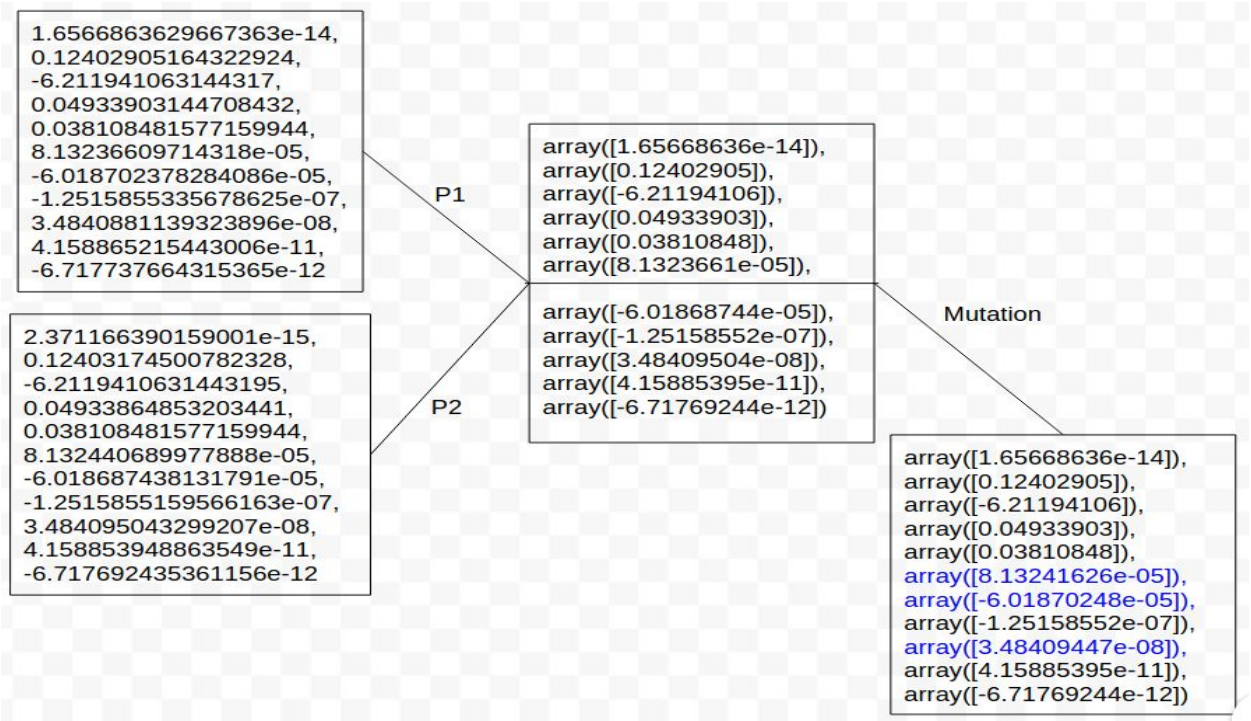


Iteration 3
Train Cost: 151526.159534472
Validation Cost: 2693114.703457

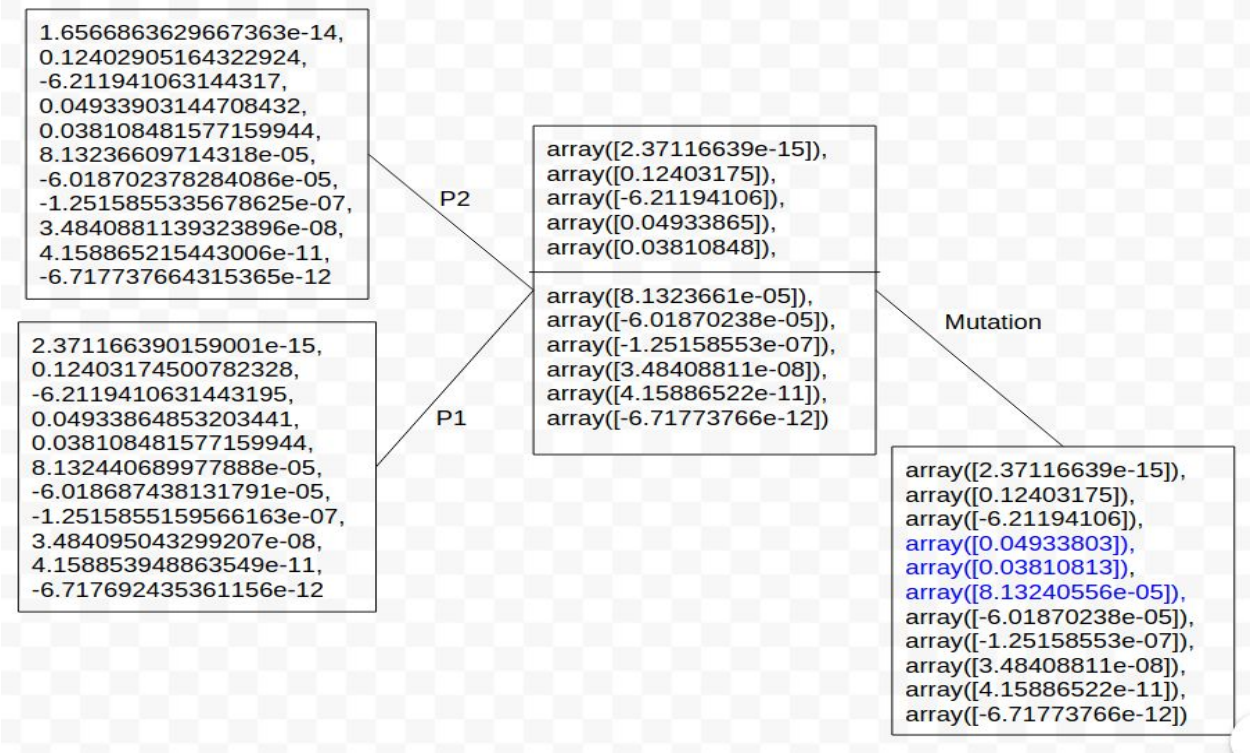
Parent Generation for Iteration 3

2.37119302e-15 1.24032336e-01 -6.21197909e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158552e-07 3.48408102e-08 4.15886522e-11 -6.71769244e-12	1.65668636e-14 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158553e-07 3.48408811e-08 4.15886522e-11 -6.71769244e-12	1.65669306e-14 1.24031745e-01 -6.21194106e+00 4.93394273e-02 3.81084816e-02 8.13244069e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15885395e-11 -6.71769244e-12	2.37116639e-15 1.24031745e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886522e-11 -6.71769244e-12	1.65669306e-14 1.24031745e-01 -6.21203822e+00 4.93382825e-02 3.81085152e-02 8.13231519e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886096e-11 -6.71769244e-12
1.65669306e-14 1.24031745e-01 -6.21203822e+00 4.93382825e-02 3.81085152e-02 8.13231519e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886096e-11 -6.71769244e-12	2.37114094e-15 1.24032336e-01 -6.21191905e+00 4.93390314e-02 3.81085152e-02 8.13231519e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15886096e-11 -6.71769244e-12	2.37119302e-15 1.24031219e-01 -6.21197909e+00 4.93391435e-02 3.81085152e-02 8.13231519e-05 -6.01876916e-05 -1.25158552e-07 3.48409504e-08 4.15881178e-11 -6.71769244e-12	1.65668636e-14 1.24029052e-01 -6.21194106e+00 4.93390314e-02 3.81084816e-02 8.13236610e-05 -6.01870238e-05 -1.25158553e-07 3.48408811e-08 4.15886522e-11 -6.71773766e-12	2.37116639e-15 1.24031745e-01 -6.21194106e+00 4.93386485e-02 3.81084816e-02 8.13244069e-05 -6.01868744e-05 -1.25158552e-07 3.48409504e-08 4.15885395e-11 -6.71769244e-12

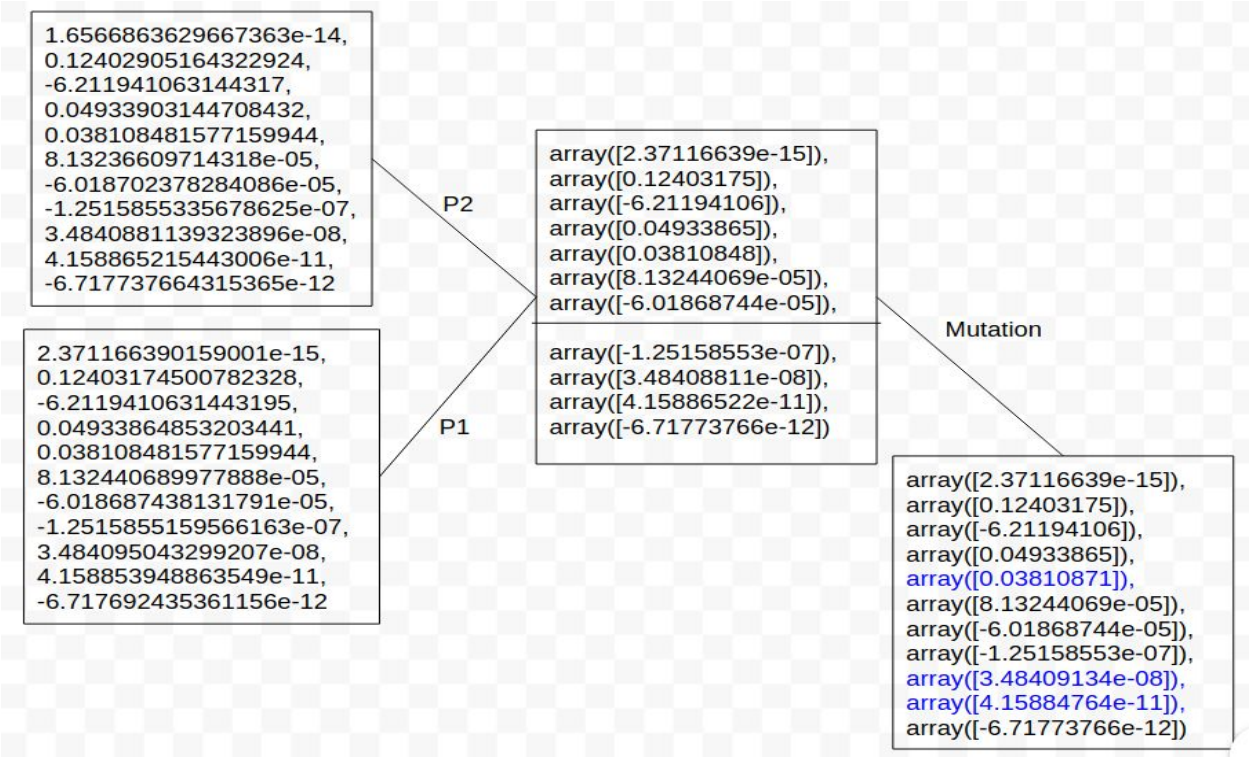
Child1



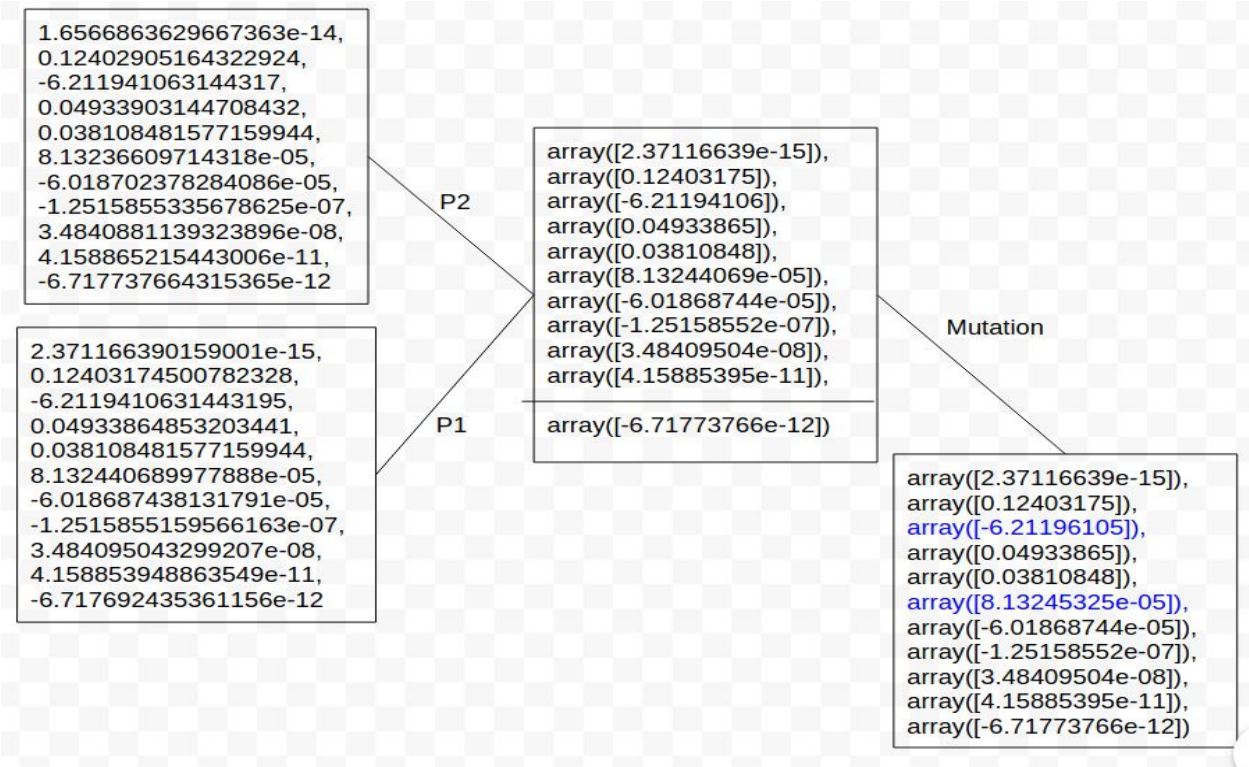
Child 2



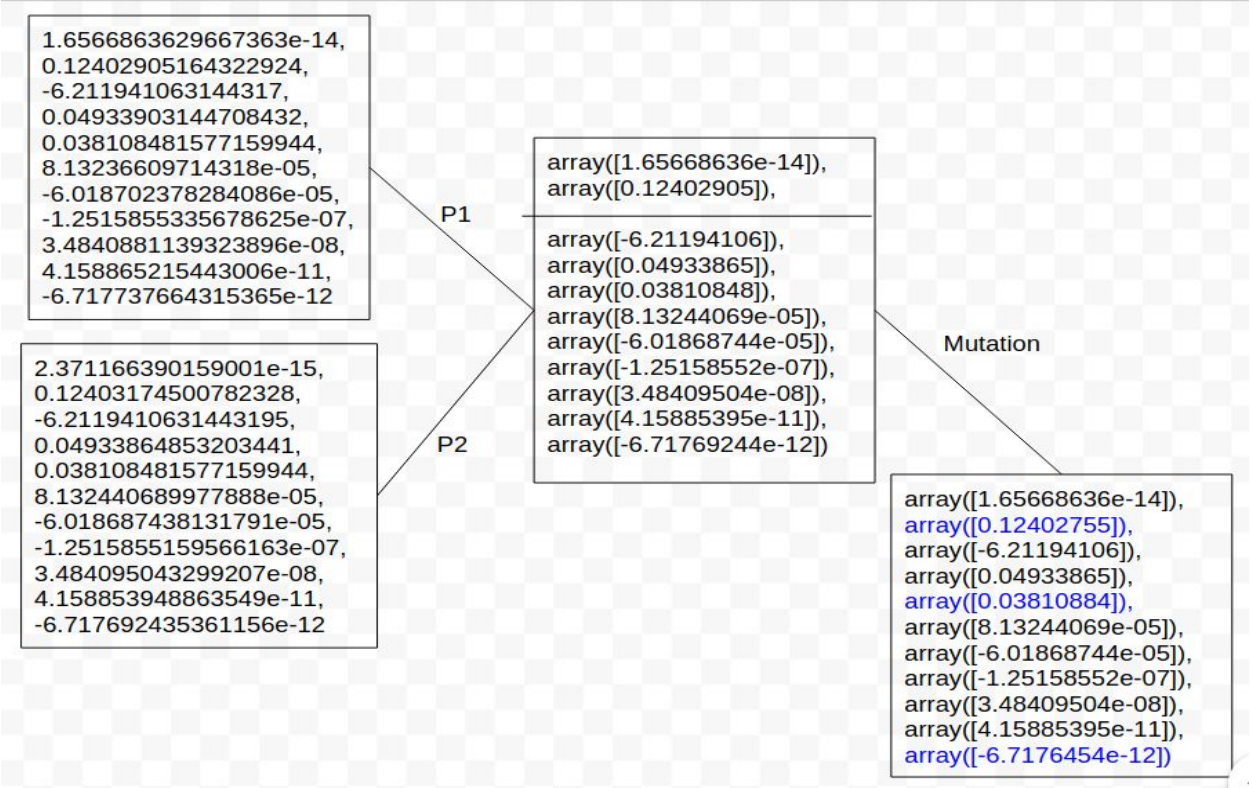
Child 3



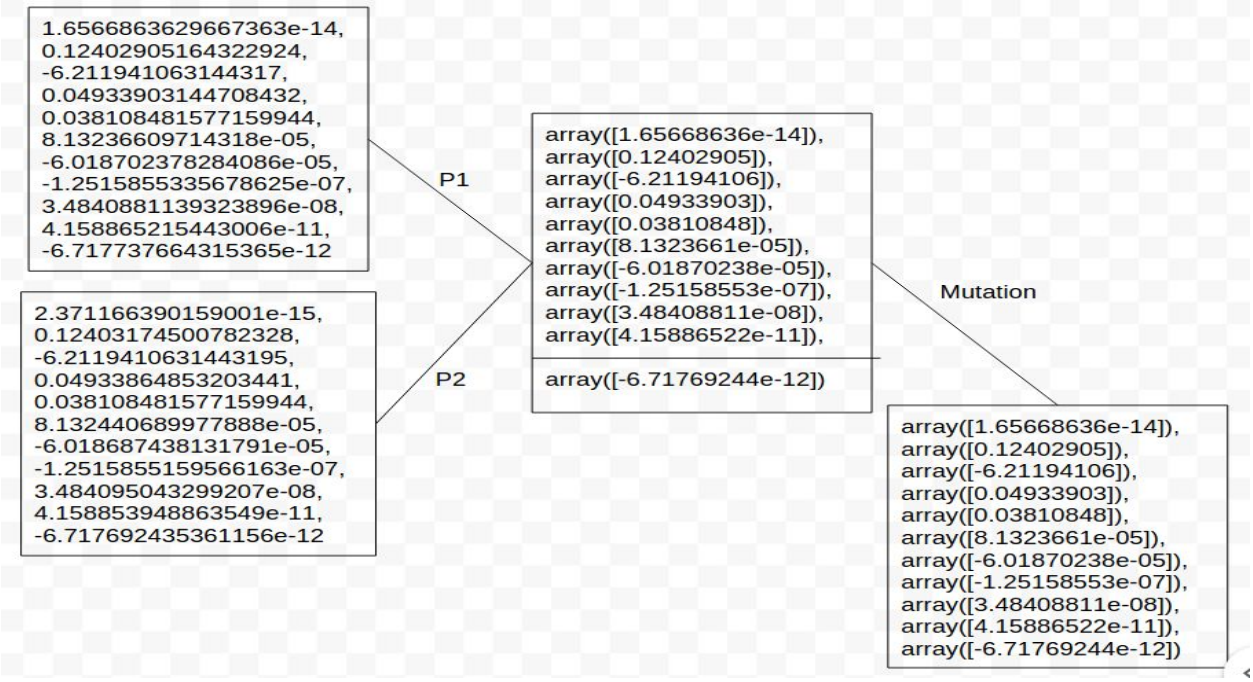
Child 4



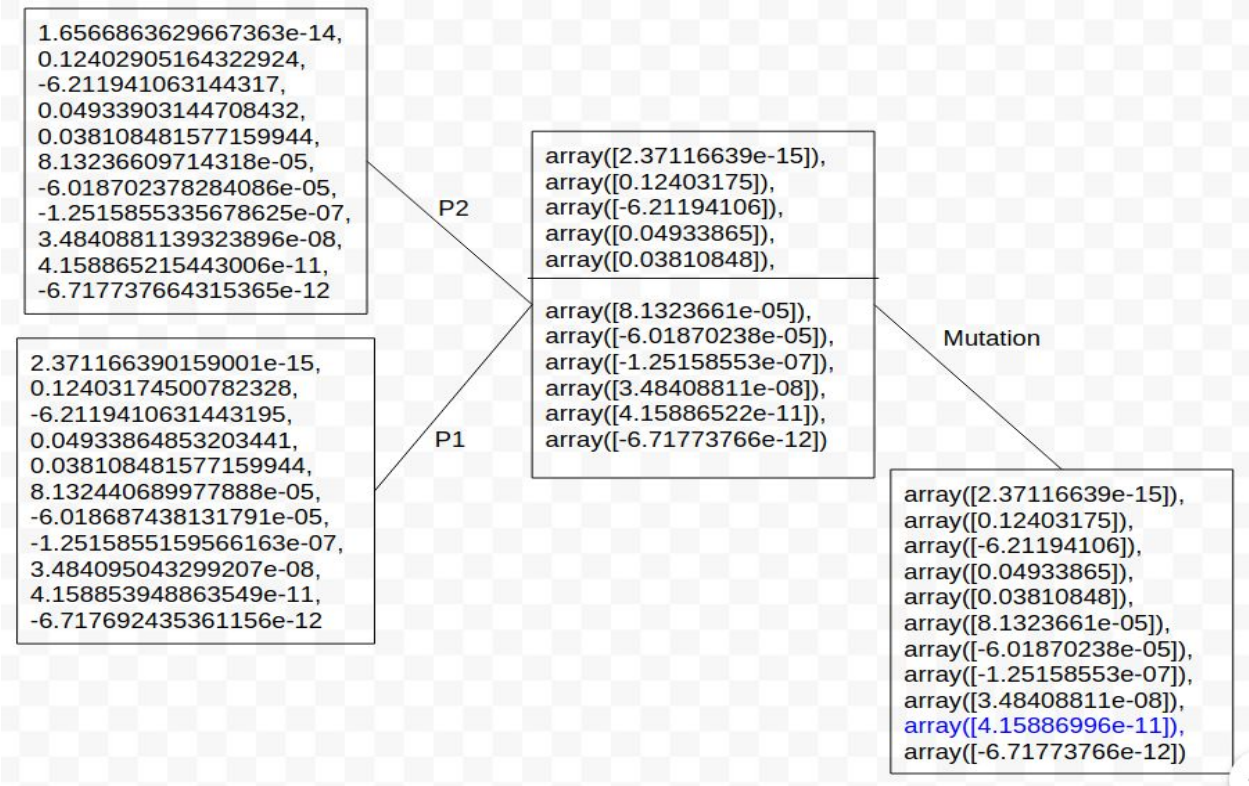
Child 5



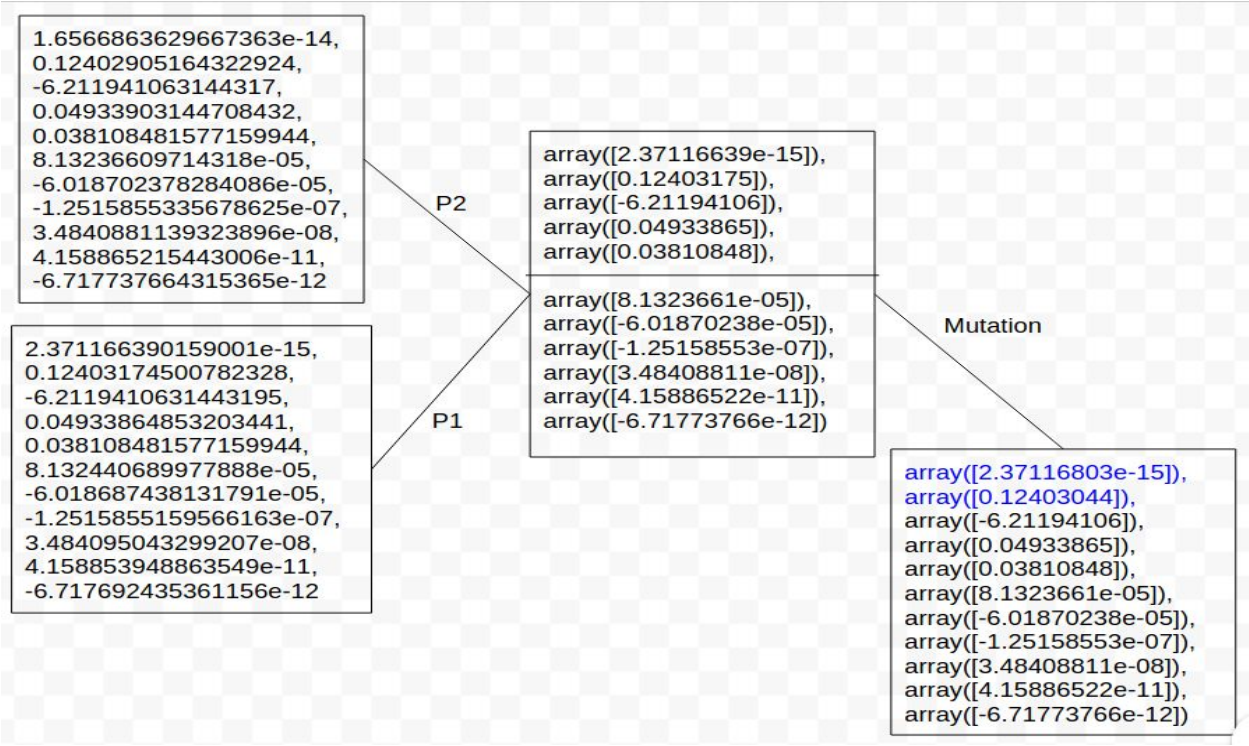
Child 6



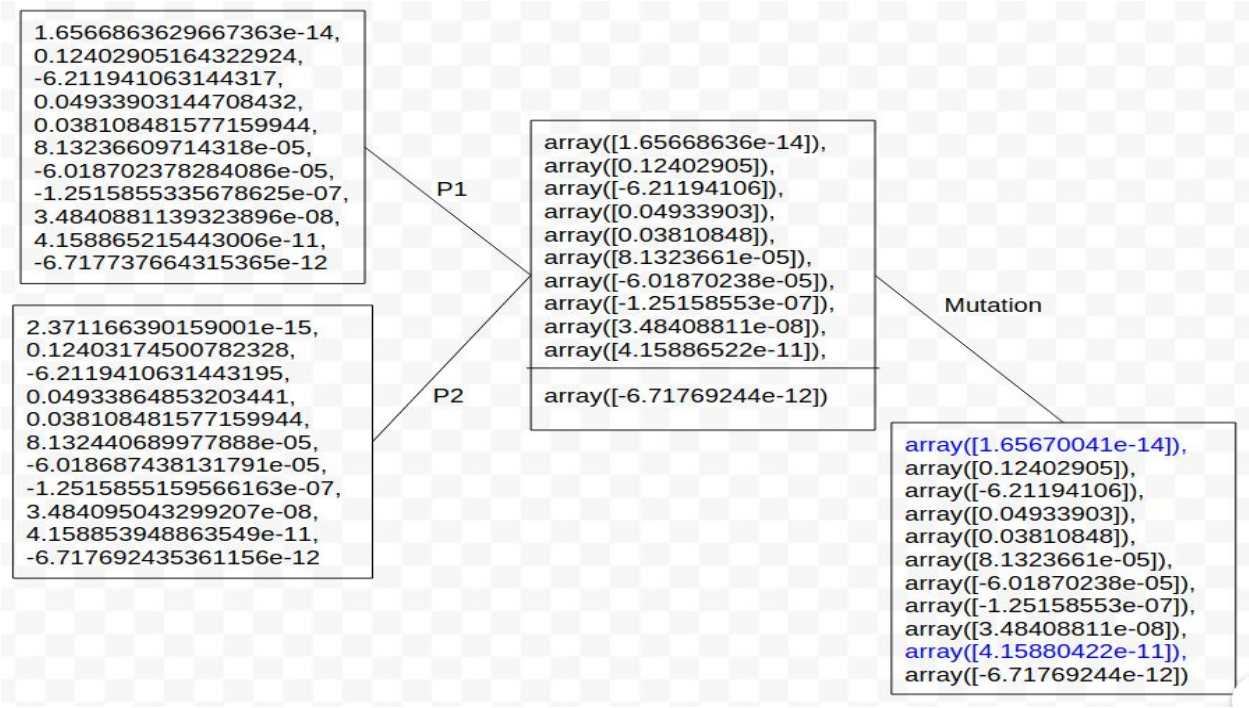
Child 7



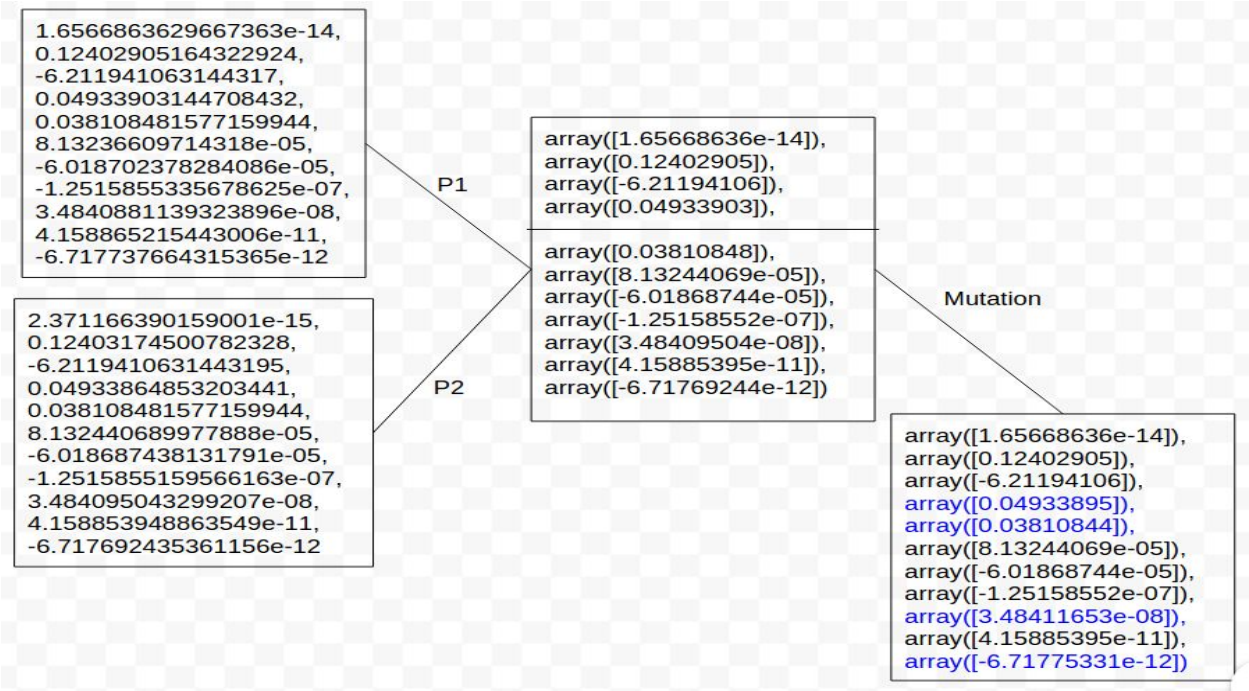
Child 8



Child 9



Child 10



History of Vectors

The trace file exists separately, these are just the best vectors from each iteration that were generated while working through the assignment over the course of time and have been listed out only for reference.

---

givenarray=[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]

givenarray=[-2.82378446e-16, 1.24031745e-01, -6.21194106e+00, 4.93390314e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158556e-07, 3.48409632e-08, 4.16163603e-11, -6.73112059e-12]

givenarray=[[8.89736459e-01, -2.79481201e+00, -6.21194106e+00, 4.93390314e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158557e-07, 3.48409642e-08, 4.16168359e-11, -6.72924983e-12]]

givenarray=[[8.89736459e-01, -3.57767914e+00, -6.21194106e+00, 4.93390314e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158557e-07, 3.48409630e-08, 4.16150287e-11, -6.72795876e-12]]

givenarray=[[8.89736459e-01, -3.57767914e+00, -6.21194106e+00, 4.93390314e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158558e-07, 3.48409612e-08, 4.16159483e-11, -6.72659473e-12]]

givenarray=[[8.89736459e-01, -3.57767914e+00, -6.21194106e+00, 4.93390314e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158558e-07, 3.48409602e-08, 4.16153853e-11, -6.72529058e-12]]

givenarray=[[8.89736459e-01, -3.57767914e+00, -6.21194106e+00, 4.93390314e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158559e-07, 3.48409598e-08, 4.16144805e-11, -6.72318559e-12]]

givenarray=[[8.89733722e-01, -3.57768214e+00, -6.21194106e+00, 4.93162792e-02, 3.81084816e-02, 8.13236610e-05, -6.01876916e-05, -1.25158561e-07, 3.48409595e-08, 4.16134984e-11, -6.72109361e-12]]

givenarray=[[8.89733722e-01, -3.57768214e+00, -6.21194106e+00, 4.93162792e-02,
3.81084816e-02, 7.66353069e-05, -6.01876916e-05, -1.25158561e-07, 3.48409588e-08,
4.16141447e-11, -6.72007676e-12]]

givenarray=[[8.89733722e-01, -3.57768074e+00, -6.21194106e+00, 4.93162792e-02,
3.81084816e-02, 7.66353069e-05, -6.01876916e-05, -1.25158561e-07, 3.48409598e-08,
4.16157269e-11, -6.71905484e-12]]

givenarray=[[8.89733722e-01, -3.57768074e+00, -6.21194106e+00, 4.93162792e-02,
3.81095792e-02, 7.66353069e-05, -6.01876916e-05, -1.25158562e-07, 3.48409581e-08,
4.16153399e-11, -6.71834898e-12]]

givenarray=[[8.89733722e-01, -3.57768074e+00, -6.21194106e+00, 4.93078419e-02,
3.81095792e-02, 7.66353069e-05, -6.01876916e-05, -1.25158563e-07, 3.48409617e-08,
4.16151490e-11, -6.71326617e-12]]

givenarray=[[8.89733722e-01, -3.57768074e+00, -6.21194106e+00, 4.93078419e-02,
3.81095792e-02, 7.66353069e-05, -6.01876916e-05, -1.25158560e-07, 3.48409644e-08,
4.16161190e-11, -6.70061830e-12]]

givenarray=[[8.89733722e-01, -3.57768074e+00, -6.21194106e+00, 4.93139999e-02,
3.81095792e-02, 7.66353069e-05, -6.01876916e-05, -1.25158560e-07, 3.48409659e-08,
4.16185976e-11, -6.70068285e-12]]

Cost function 12,12 lacs

givenarray=[8.89484543e-01, -3.57762285e+00, -6.21206709e+00, 4.94172660e-02,
3.78913218e-02, 7.78738296e-05, -6.01876916e-05, -1.25158578e-07, 3.48409852e-08,
4.15250722e-11, -6.67948159e-12]

givenarray=[8.89648247e-01, -3.57765351e+00, -6.21204342e+00, 4.93310953e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158574e-07, 3.48409780e-08,
4.15150577e-11, -6.68419745e-12]

givenarray=[8.89648247e-01, -3.57765351e+00, -6.21204342e+00, 4.93310953e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158580e-07, 3.48409729e-08,
4.15077575e-11, -6.68332457e-12]

givenarray=[8.89648247e-01, -3.57765060e+00, -6.21204342e+00, 4.93310953e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158578e-07, 3.48409717e-08,
4.15079722e-11, -6.68908121e-12]

Reducing both together, ended at 5,15 lacs

givenarray=[8.89648247e-01, -3.57765060e+00, -6.21204342e+00, 4.93310953e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158576e-07, 3.48409745e-08,
4.15073722e-11, -6.69552321e-12]

givenarray=[9.01146237e-01, -3.57765060e+00, -6.21204342e+00, 4.93310953e-02,
3.81024551e-02, 7.78738296e-05 ,-6.01876916e-05, -1.25158578e-07, 3.48409725e-08,
4.15038523e-11, -6.70044857e-12]

givenarray=[9.01146237e-01, -3.57156651e+00, -6.21204342e+00, 5.21195658e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158577e-07, 3.48409696e-08,
4.15021966e-11, -6.70140483e-12]

Cost starting 12,12 reducing only validation error from 12 lacs

givenarray=[8.89731891e-01, -3.57765351e+00, -6.21199192e+00, 4.93310953e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158577e-07, 3.48409806e-08,
4.15257685e-11, -6.68482225e-12]

givenarray=[8.89731891e-01, -3.57765351e+00, -6.21603962e+00, 4.53401861e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158575e-07, 3.48409765e-08,
4.15223041e-11, -6.68214576e-12]

givenarray=[8.93561286e-01, -3.57550911e+00, -6.21603962e+00, 4.53401861e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158575e-07, 3.48409775e-08,
4.15181164e-11 ,-6.68174209e-12]

givenarray=[1.41796514e+00, -5.70161462e-01, -6.21603962e+00, 4.53401861e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158580e-07, 3.48409792e-08,
4.15056563e-11, -6.68284692e-12]

givenarray=[-2.61920919e+00, 2.35548511e+00, -6.19298898e+00, 4.53401861e-02,
3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158574e-07, 3.48409805e-08,
4.14974879e-11 ,-6.68383697e-12]

givenarray=[-2.61920919e+00, 2.35548511e+00, -6.19298898e+00, 4.53401861e-02, 3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158575e-07, 3.48409812e-08, 4.14928277e-11, -6.68464886e-12]

givenarray=[-5.19101785e+00, 2.35548511e+00, -6.15746922e+00, 4.53401861e-02, 3.81024551e-02, 7.78738296e-05 ,-6.01876916e-05, -1.25158573e-07, 3.48409808e-08, 4.14787624e-11, -6.68611786e-12]

givenarray=[-5.68577849e+00, 3.06958209e+00, -6.15746922e+00, 4.53401861e-02, 3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158573e-07, 3.48409806e-08, 4.14772791e-11, -6.68631574e-12]

givenarray=[-5.68577849e+00, 3.86530309e+00, -6.15746922e+00, 4.53401861e-02, 3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158573e-07, 3.48409824e-08, 4.14631216e-11, -6.68621531e-12]

givenarray=[-5.68577849e+00, 4.26835728e+00, -6.15746922e+00, 4.53401861e-02, 3.81024551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158573e-07, 3.48409801e-08, 4.14465616e-11, -6.68569417e-12]

Costs 10.9 lacs,10.05 lacs

givenarray=[-5.78477849e+00, 4.26835728e+00, -6.15856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158573e-07, 3.48409801e-08, 4.10465616e-11, -6.69569417e-12]

givenarray=[-5.78477849e+00, 4.26835728e+00, -6.15856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158572e-07, 3.48409742e-08, 4.10420140e-11, -6.68622871e-12]

givenarray=[-5.78477849e+00, 5.85560955e+00, -6.15856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158571e-07, 3.48409755e-08, 4.10304130e-11, -6.68646443e-12]

givenarray=[-5.78477849e+00, 5.85560955e+00, -6.15856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158571e-07, 3.48409755e-08, 4.10304130e-11, -6.68646443e-12]

givenarray=[-6.67997053e+00, 5.85560955e+00, -6.15856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158570e-07, 3.48409753e-08, 4.10168847e-11, -6.68624749e-12]

givenarray=[-7.53612586e+00, 5.85560955e+00, -6.15856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158571e-07, 3.48409801e-08, 4.10190119e-11, -6.70177436e-12]

Cost function 9.5 lacs each

givenarray=[-7.53612586e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158571e-07, 3.48409801e-08, 4.10190119e-11, -6.70177436e-12]

givenarray=[-9.18758393e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158578e-07, 3.48409794e-08, 4.10103138e-11, -6.69492176e-12]

givenarray=[-9.23895107e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158582e-07, 3.48409709e-08, 4.09974588e-11, -6.69190994e-12]

givenarray=[-9.99504235e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158583e-07, 3.48409776e-08, 4.09908462e-11, -6.69185227e-12]

givenarray=[-9.99504235e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158580e-07, 3.48409790e-08, 4.09815164e-11, -6.69199345e-12]

givenarray=[-9.99504235e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158577e-07, 3.48409776e-08, 4.09746089e-11, -6.69153221e-12]

givenarray=[-9.99504235e+00, 9.85560955e+00, -6.05856922e+00 , 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158575e-07, 3.48409851e-08, 4.09651178e-11 ,-6.69218935e-12]

Train 12.9 and 8.5 Validation

givenarray=[-9.99504235e+00, 9.85560955e+00, -6.05856922e+00 , 4.53401861e-02, 3.81036551e-02, 7.68038296e-05, -6.01876916e-05, -1.25158575e-07, 3.48409851e-08, 4.09651178e-11 ,-6.69218935e-12]

givenarray=[-9.99504235, 9.85560955, -6.05856922, 0.0453401861, 0.0381036551, 7.60038296e-05, -6.01876916e-05, -1.25158575e-07, 3.48409851e-08, 4.09651178e-11, -6.69218935e-12]

givenarray=[-9.99504235e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.60038296e-05, -6.01876916e-05, -1.25158581e-07, 3.48409885e-08, 4.09564646e-11, -6.69095571e-12]

givenarray=[-9.99504235e+00, 9.85560955e+00 -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.60038296e-05, -6.01876916e-05, -1.25158576e-07, 3.48409868e-08, 4.09468338e-11, -6.69089779e-12]

Costs are 9.5 lacs each

givenarray=[-7.53612586e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876916e-05, -1.25158571e-07, 3.48409801e-08, 4.10190119e-11, -6.70177436e-12]

givenarray=[-7.53612586e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876917e-05, -1.25158504e-07, 3.48408232e-08, 4.09133640e-11, -6.69400719e-12]

givenarray=[-9.99999999e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738298e-05, -6.01876917e-05, -1.25158303e-07, 3.48408724e-08, 4.08237873e-11, -6.69435995e-12]

givenarray=[-9.99999999e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738297e-05, -6.01876919e-05, -1.25158329e-07, 3.48409301e-08, 4.06195473e-11, -6.69305035e-12]

givenarray=[-9.99999999e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738296e-05, -6.01876919e-05, -1.25158386e-07, 3.48409125e-08, 4.06714634e-11, -6.69432539e-12]

givenarray=[-9.99999999e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738297e-05, -6.01876919e-05, -1.25158560e-07, 3.48409838e-08, 4.08210573e-11, -6.70723210e-12]

givenarray=[-9.99999999e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738297e-05, -6.01876919e-05, -1.25158561e-07, 3.48409854e-08, 4.08214542e-11, -6.70420131e-12]

Train 7.89 validation 10.03 lacs

givenarray=[-9.99999999e+00, 9.85560955e+00, -6.05856922e+00, 4.53401861e-02, 3.81036551e-02, 7.78738297e-05, -6.01876918e-05, -1.25158707e-07, 3.48409405e-08, 4.09708941e-11, -6.70736721e-12]

givenarray=[-9.99999999e+00, 9.99999999e+00, -6.05856922e+00, 4.61401861e-02, 3.81136551e-02, 7.78738297e-05, -6.01646018e-05, -1.24958707e-07, 3.48409405e-08, 4.09708941e-11, -6.70736721e-12]

Costs are Train 4.4 lacs, Validation 18.41

givenarray=[-9.96694707e+00, 9.96881596e+00, -6.05983136e+00, 4.60358442e-02, 3.80873328e-02, 7.78781651e-05, -6.01843398e-05, -1.25132831e-07, 3.48053495e-08, 4.09845260e-11, -6.70973486e-12]
givenarray=[-9.96692434e+00, 9.96681593e+00, -6.05976812e+00, 4.60396338e-02, 3.80849605e-02, 7.79272579e-05, -6.02064195e-05, -1.25045697e-07, 3.48160009e-08, 4.10130214e-11, -6.71152757e-12]

Costs are Train 3.46, Validation 21

givenarray=[-9.96198594e+00, 9.96473398e+00, -6.05956687e+00, 4.60544923e-02, 3.80697115e-02, 7.79493994e-05, -6.02114452e-05, -1.24941216e-07, 3.48193014e-08, 4.10324617e-11, -6.71221046e-12]

Costs are 8.88 lacs, 9.4 lacs

givenarray=[-9.96666666, 9.9666669, -6.05856922, 0.0460401861, 0.0381036551, 7.78738297e-05, -6.01646018e-05, -1.25158707e-07, 3.48409405e-08, 4.09708941e-11, -6.70736721e-12]

Costs are 9.17 each

givenarray=[-9.96666666e+00, 9.96666690e+00, -6.05436055e+00, 4.60462110e-02, 3.81036551e-02, 7.78912821e-05, -6.01646018e-05, -1.25158707e-07, 3.48409405e-08, 4.09080132e-11, -6.70736721e-12]

Costs are 7.35 and 10.24

givenarray=[-9.93011979e+00, 9.94841173e+00, -6.05436055e+00, 4.60609255e-02, 3.80641896e-02, 7.79677253e-05, -6.01584192e-05, -1.24630535e-07, 3.48351200e-08, 4.07433518e-11, -6.70642133e-12]

Costs are 8.21 and 9.49 lacs

givenarray=[-9.92402214e+00, 9.94903713e+00, -6.05921755e+00, 4.60385103e-02, 3.80641896e-02, 7.79808066e-05, -6.01169291e-05, -1.24648573e-07, 3.48100575e-08, 4.06261186e-11, -6.70273833e-12]

Costs are 8.09 and 9.55

givenarray=[-9.93997110e+00, 9.94903713e+00, -6.05921755e+00, 4.60385103e-02, 3.80641896e-02, 7.80832578e-05, -6.01169291e-05, -1.24648573e-07, 3.48100575e-08, 4.06261186e-11, -6.70273833e-12]

Costs are 8.89 and 8.97

givenarray=[-9.93997110e+00, 9.94936112e+00, -6.06250670e+00, 4.60643745e-02, 3.80641896e-02, 7.80832578e-05, -6.01169291e-05, -1.24648573e-07, 3.48100575e-08, 4.05849816e-11, -6.70037317e-12]

Costs are 7.77 lacs and 9.66

givenarray=[-9.94699954e+00, 9.94936112e+00, -6.06250670e+00, 4.60365302e-02, 3.80574614e-02, 7.81803738e-05, -6.01169291e-05, -1.24327056e-07, 3.48072352e-08, 4.05227487e-11, -6.70037317e-12]