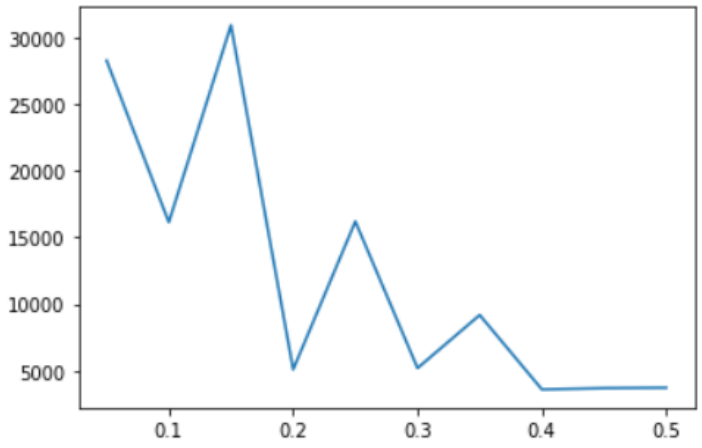
Zvunka Lorena Maria
Portfolio report

Part 1 - Two-layer perceptron

The Neural Network was implemented as a separate class. The only thing necessary to initialize the perception is to mention it's structure. For this case, we have [4, 4, 1], an input layer and a hidden layer with 4 elements and an output layer with a single output element. The weights are all initialized at the beginning with random numbers between 0 and 1. As for the bias, it is added as a column on 1's at the end of both the weight matrix and the input matrix.
The activation function chosen is the simple sigmoid function 1/(1+np.exp(-x)).

The dataset consists of all 4 number combinations of 0 and 1. For the training of the model a subset of this dataset was chosen, consisting of 10 entries. For different learning rates, the accuracy of the predictions were different. For a learning rate of 0.05, the training takes almost 40.000 epochs to complete and the accuracy of the final predictions is 73.3%. One of the best accuracy for this way of splitting the training as testing data was 80%. Depending on the way in which the data is split, accuracy can vary from 40% to 100&.

Another factor in training a neural network model, aside from the data, is the learning rate. Since for this we are not operating with a specific number of epochs, rather a stop condition, the learning rate determines how long it takes to train a model.

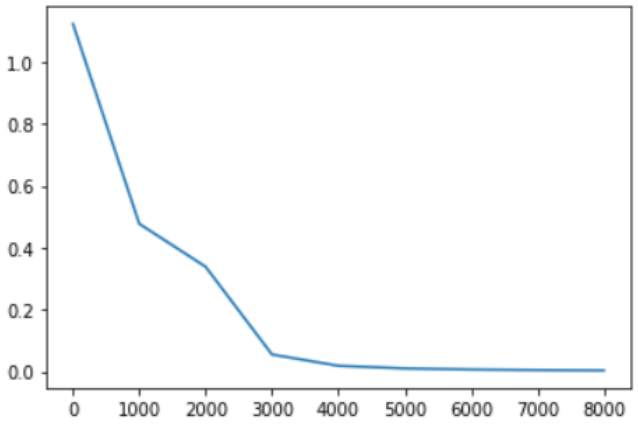


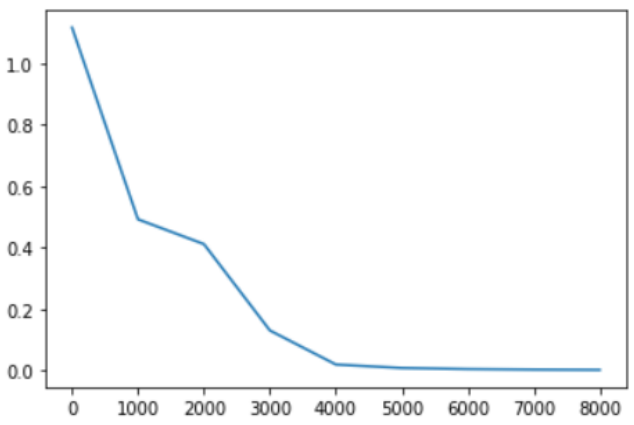
When changing the learning rate from 0.05 to 0.5 we can see that the number of epochs necessary to train the model lowers from over 30.000 to less than 5.000.

The overall accuracy when having different learning rates, also varies. For the case described above where the training data contains 10 entries, the accuracy is around 60-80%

As we change the learning rate from 0.05 to 0.5, in increments of 0.05, the number of epochs is, in order: [28275, 16142, 30923, 5091, 16200, 5196, 9180, 3598, 3693, 3722]. This is for a specific data training set, but will change depending on how many entries there are. However, we can see it generally gets lower as the learning rate gets greater.

By looking at the trajectory of the loss function, we can see that a decently accurate model can be achieved by stopping even before the stop condition is met.





Part - 2 Genetic Algorithm

1. Reading and preparing the data

   The first step is retrieving the data in a way that makes it easy to access and manipulate in the future. Storing all the information was done in different classes, corresponding to the type of data being handled. A member of the customer class contains all the information for a client, the coordinates, the quantity to be delivered etc. A member of the class depot remembers the information of a depot and also all the customers that will later be assigned to a particular location.
   In order to create a population we need to decide which clients belong to which depot. This is done in an intuitive and simple way. By assigning each customer to the depot they are closest to in terms of the Euclidean distance. If the closest depot is already full, it will be assigned to the next closest one and so forth.
   It is important that this first assignment of clients is as good as possible because the following implementation does not reallocate customers to other depots.

2. Chromosome representation

   For this implementation one of the first big decisions to be made is the chromosome representation. The way I decided to represent the data is by having each chromosome be a collection of t vectors, where t is the number of depots available. Each of the t vectors will contain the customers being serviced by that depot. The chromosome representation will not keep track of any of the individual routes done by the cars in each depot. It will only remember the customers and the order in which they are to be serviced.
   For the initial population, this order is chosen at random. For example, after assigning clients, a possible chromosome would look like this.

```
[[40, 13, 42, 37, 25, 45, 44, 4, 17, 18, 41, 15, 19],
 [47, 26, 24, 46, 48, 23, 31, 1, 43, 6, 11, 14, 27, 12, 32, 8, 7],
 [33, 5, 39, 34, 49, 30, 38, 9, 50, 10, 16],
 [2, 21, 29, 35, 3, 20, 36, 28, 22]]
```

   Another possible chromosome representation would involve remembering the individual car routes of each depot, however this would make working with the data much harder. The main reason for that is that while there are always a set number of depots, each depot can have a different number of cars. This number of cars can also change from chromosome to chromosome.

```
[[17, 40, 45, 4, 25], [18, 19, 13, 42, 37, 44], [15, 41]]
[[48, 43, 11, 14], [23, 1, 31, 32, 27, 7, 24], [26, 46, 6, 8, 47], [12]]
[[38, 9, 49, 30, 10], [16, 34, 33, 39, 50], [5]]
[[21, 3, 29, 2], [35, 36, 22, 20]]
```

   This makes a chromosome more irregular and would make the mutation and crossover harder to execute. While harder to work with, this representation could also suit the problem since it can remember all the important details and describe a feasible solution.

3. Selection and Fitness function

   Before the crossover and mutation can occur, a selection process is necessary in order to choose the pool of parents. The crossover process implemented can only be applied on 2 parent chromosomes at a time, therefore the selection process involves randomly selecting 2 chromosomes from the population.
   While selecting parents randomly is not the best practice it does ensure that all chromosomes have a chance at becoming parents, regardless of their fitness.
   The fitness of a chromosome is measured by calculating the entire distance traveled through that solution. The overall fitness is given by normalizing all the values previously calculated.

4. Crossover and Mutation

   The crossover used here is the Best Cost Route Crossover. When coded correctly this method for the cross-over ensures that all the offspring will be feasible. Although in theory this should not produce any unfeasible solutions, on certain test sets some vehicles have been assigned routes that exceed the weight of their capacity. Therefore, the implementation I have for this function should be perfected.

   When performing the crossover, each step checks whether the potential child obtained through this operation will be allowed back into the population. If at any given point the answer is no, then the child is changed accordingly.
   Because this method deals with only one depot at a time there is no chance of making a change that will exceed the maximum weight capacity of a depot. The way in which the change is made is by selecting 2 random clients from the 2 chromosomes and inserting at the best feasible position. Before inserting 2 clients into a depot, those clients are deleted from their original place, ensuring that no repeats are going to take place.

```
[32, 27, 24, 7, 47, 6, 23, 26, 8, 1, 46, 11, 14, 43, 31, 48, 12],
```

   Above is an example of a parent and below is a possible child that is created by the BCRC function.

```
[27, 24, 43, 7, 47, 6, 23, 1, 11, 46, 14, 31, 48, 12, 8, 32, 26],
```

   The mutation used is a reversal mutation. It takes a random depot from a chromosome and reverses the order of all of its elements. It poses no threat to creating unacceptable solutions since a simple inversion doesn't interfere with any of the limitations places(ex. max weight).

   Another thing that is implemented while going through the crossover and mutation operations is elitism. Whenever choosing parents, the best solution currently in the population cannot become a parent and be replaced. Thus ensuring that the solution cannot degrade over time.

   The population size stays consistent through all generations because for every step, all the parents are replaced with their offspring.

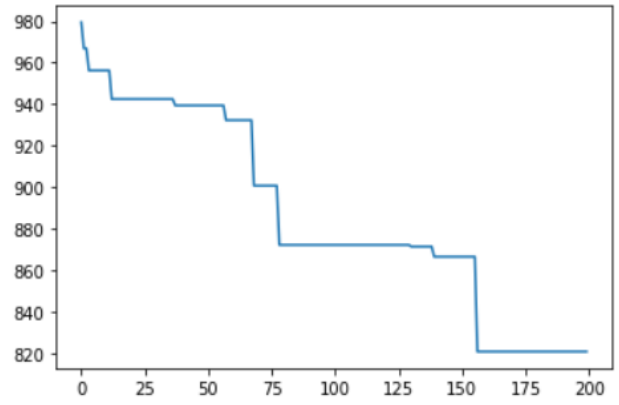
Mutation Rate and Crossover Rate

Because of the implementation chosen for the crossover and mutation functions, the way in which both the mutation and crossover rate affect the overall algorithm is by choosing how many times every generation a crossover or mutation operation should be executed. If the crossover rate is

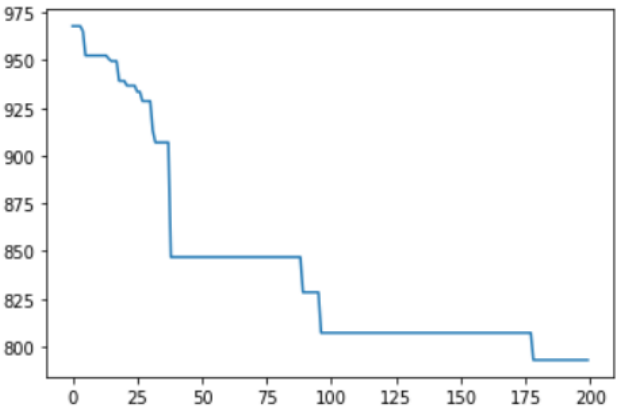
1, for every generation only 2 parents will be chosen and replaced. If the value is 4 it means that 8 parents will be chosen and replaced.

The bigger the mutation and crossover rate, the faster the algorithm will find a better solution.
For the same number of generations(=200) and a population size of 10. The results when both the crossover and mutation rate are 1 differ from when the crossover rate is 3 and the mutation rate is 2. The following images show the progression of the best distance for each generation, given the variables mentioned above.

820.7711331752533



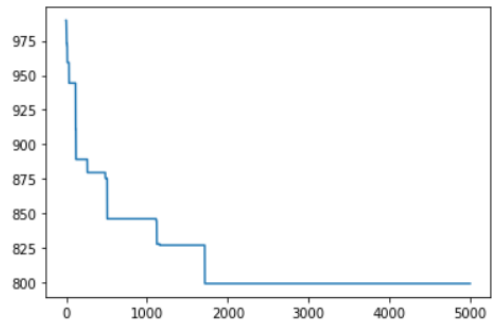
792.9810660073182

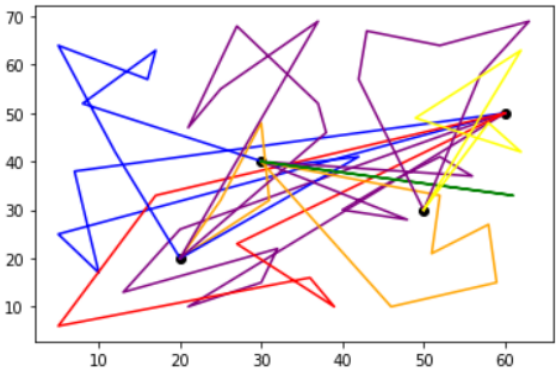


We can see that the bigger the mutation and
crossover rates, the faster our algorithm will reach an optimal solution. However, the bigger those rates, so becomes the chance of an error happening.
When the crossover and mutation rates are too high, for certain populations, errors have occurred, therefore, setting those values too high can damage the performance.

However, probably the most important parameter is the number of generations. Having both the crossover and mutation rates as 1, we can view the differences caused by changing the number of generations.

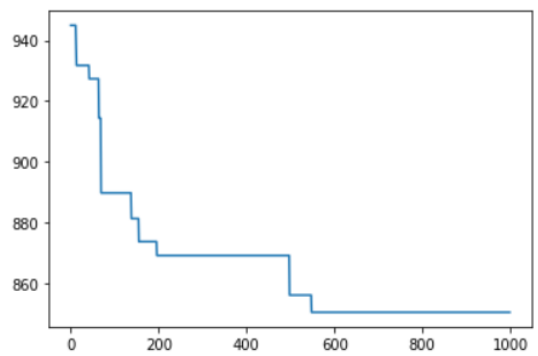
799.2640567696034

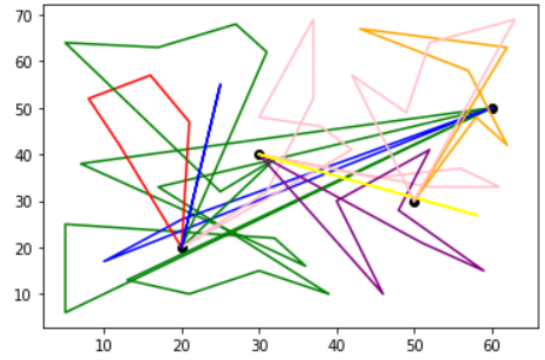




Results when there are 5.000 generations. The distance is 799.26

850.3798730210333

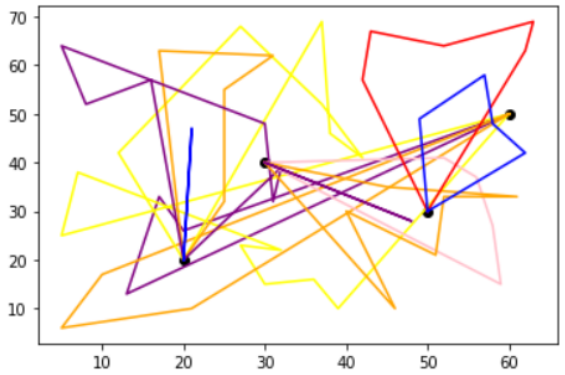




Results when there are only 1.000 generations. The distance is 850.37

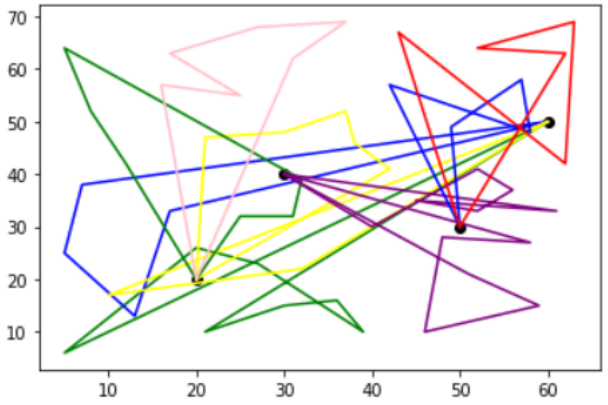
It can be clearly seen that the more generations we have, the better of a solution we will get.
Another factor that should be taken into consideration is the population size. All the results until now have been achieved with a population of size 10. If we take into consideration two drastically different population sizes, we can better see the effect this parameter has on the result.

```
822.3383364439549
```



For a population of size 3 and a generation number of 1.000, the smallest distance found is of 822.3

```
811.9147983263174
```



For a population of 100 with the same generation number of 1.000 the distance found is 811.9
Although this is a better result than the one found with a smaller population size. Going for a big population is not always the best idea because it can get computationally hard and slow down the overall process. When it comes to choosing both population size and generation number, choosing extremely small or large numbers is a worse idea than sticking to something in-between.

Overall, the parameters should be set according to the data and the computational power one has at its disposal.
They should also be in sync with each other. Having a really low crossover rate and a really high mutation rate can lead to little changes from one generation to the next. Having a large generation number and also a high crossover rate can lead to the algorithms taking a lot of time to find a solution. Those numbers should be chosen in such a way that they do not lead to any excess computation and that the change from one generation to the next doesn't become stagnant.

Resources:

1. https://cs.uwlax.edu/~dmathias/cs419/readings/Ombuki-BermanHanshar2008.pdf
2. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3