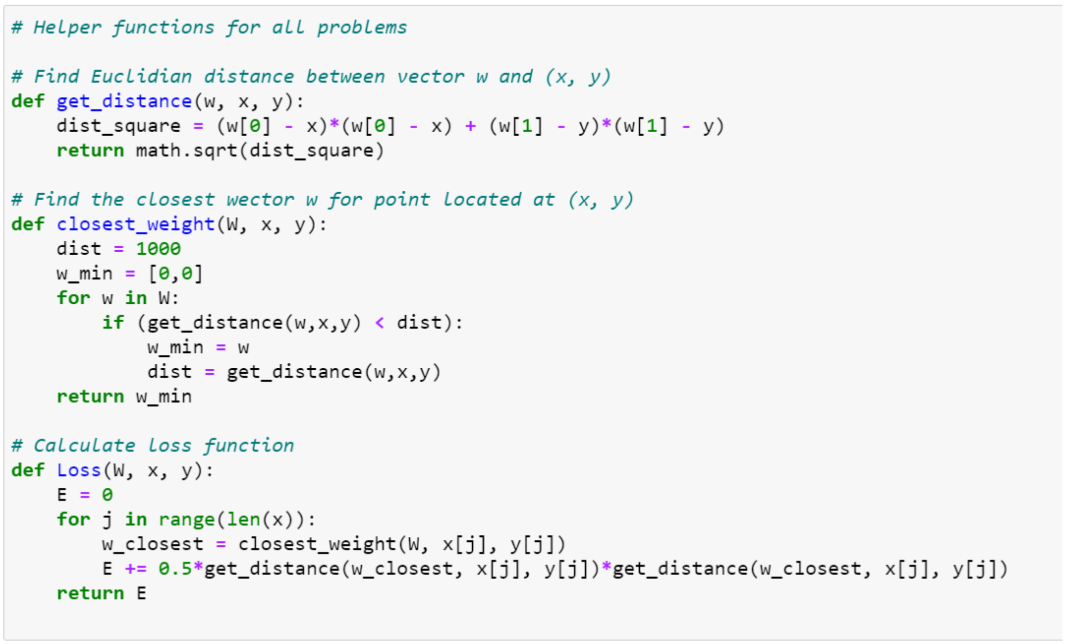
Neural Networks Report #1

Introduction

For this homework I used Jupyter Notebook as python is very simple and effective language, especially where we need to work with data; also, cells in the notebook are very useful for dividing the parts of the algorithm. All problems are made in one file and the comments indicate their name. As code parts are repeating, initially I made the helper functions, then functions for clustering for any data and after that for each problem I generated data and used these functions. Please note that the cells must be run in order as the data changed in each problem. Here I will briefly explain the code algorithm and logic of its performance. Finally, I will show my best result of each problem, add some feedback regarding the performance of my code and further possible improvement and finish it with small conclusion.

Code Explanation

I imported libraries only for math and plotting, for generating data, uniform functions was important, it take the upper and lower bound and generated float in this range with equal probability.



```python
# Helper functions for all problems

# Find Euclidian distance between vector w and (x, y)
def get_distance(w, x, y):
    dist_square = (w[0] - x)*(w[0] - x) + (w[1] - y)*(w[1] - y)
    return math.sqrt(dist_square)

# Find the closest wector w for point located at (x, y)
def closest_weight(W, x, y):
    dist = 1000
    w_min = [0,0]
    for w in W:
        if (get_distance(w,x,y) < dist):
            w_min = w
            dist = get_distance(w,x,y)
    return w_min

# Calculate loss function
def Loss(W, x, y):
    E = 0
    for j in range(len(x)):
        w_closest = closest_weight(W, x[j], y[j])
        E += 0.5*get_distance(w_closest, x[j], y[j])*get_distance(w_closest, x[j], y[j])
    return E
```

The functions that will be called many times are defined as helper functions.

get_distance takes input weight and coordinates of point (x, y) and computes the distance between them.

closest_weight takes point coordinates and W which is list of weight vectors and it finds the weight which is closest to it. And returns this vector.

Loss is needed to calculate the error after iterations. For every point it finds the nearest weight, computes the distance square between and stores it. The half of summation of all point is the overall error in the given iteration. To achieve effective clustering, this error should be minimized.

To generate the data with given probability distributions, hint was used. In problem 1&3 (a) for N1 I create the point at random position between -1<x< 1, -1<y<1 and if it satisfies x^2 + y^2 < 1, I add it to the data. Once N1 points are created, process is finished. Other distributions obey the same logic, like in problem 1&3 (a) N2 is distributed similarly as N1, nut the center of the circle is moved to (2,0), so the data is generated in 1<x< 3, -1<y<1 and it now should satisfy (x-2)^2 + y^2 < 1. Similarly, for N3 in Problem 1&3 (a) unit circle is located at (0,2), so the data is generated in 1<y< 3, -1<x<1 and it now should satisfy (y-2)^2 + x^2 < 1. Problem2&4 (a), not only the center of circle is moved, but also the radius of it and the probability value are changed. The hint is that they changed by the same factor, i.e. if the circle has radius 2, its probability is 1/4(pi). So, the bigger space it has to locate, the smaller becomes the probability to be in 1 unit space. That's why we again use old approach, generate the points with uniform probability in the square of space, and only take the points that occur in the region of that circle. For example for N2 in Problem2&4 (a) data is generated in 1<x< 5, -2<y<2 (center at (3,0)) and it now should satisfy (x-3)^2 + y^2 < 4.

Next, to divide into the cluster, competitive learning algorithm is used which is briefly made in steps:

1. Generate randomly the weight vectors that equal to the number of clusters
2. For each point in the data, find the closest weight and move it along itself by the value etha*distance where etha is the learning rate parameter between 1 and 0.
3. Repeat step 2 for some number of times which is another parameter

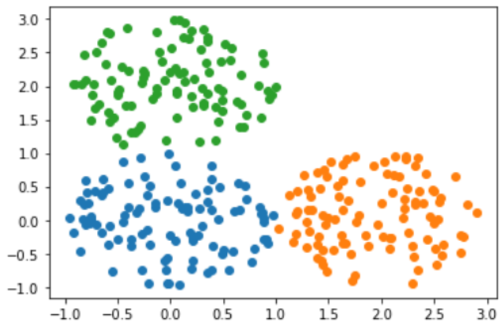
It is competitive as the only closest vector moves in each step. This makes the weight finally end up in the centers of the clusters as they move closer and closer to the data located in some closed space and this makes the error E which is actually the sum of squared distances of cluster points and corresponding weight minimum. Etha parameter should be not large so it will not move into each point and not small to avoid many iterations. The larger the epoch number, the better, but a lot may result in overfitting so it also should be chosen carefully.

The problem is generating random weights as sometimes they happen to be very close and converge to the same cluster, however I optimized parameters well so it happens very rarely and I made some region where weight should be initialized to save the training time and make less number of epoch; it is still valid as I just make assumption in the range where the data should be, not giving information about the clusters. Etha = 0.1 in my network and to optimize it I halve the parameter every 50 iterations as this makes weight to move to the clusters faster, but not to move further them. My epoch varies from problem to problem as I tuned it according loss function such that it reaches its minimum and stops after not changing for long time. For example, in many problems 300 epoch whenever in some only after 450. The loss was calculated 10 iterations.

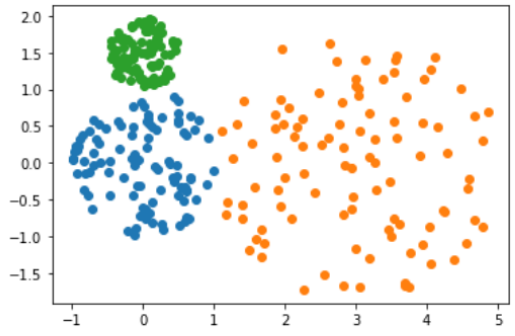
Finally, after the network is trained weights should be located at the center of each cluster. To classify it, for each point we find the closest weight and assign the color according to which cluster is this.

These steps, are shown in the functions three_clusters, two_clusters, four_clusters, for orientation, I commented the name of each step. Functions than were applied for each data, parameter epoch_num was tuned accordingly.

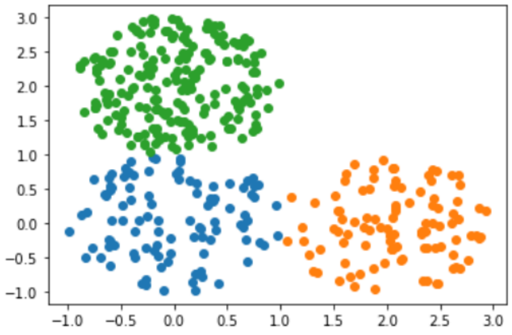
Result analysis



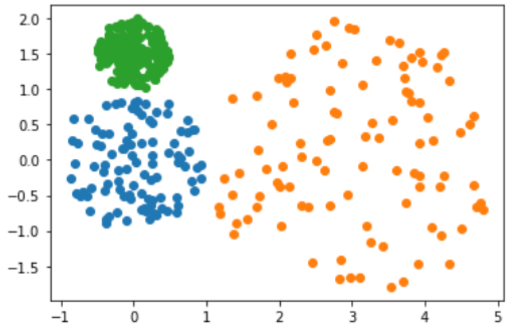
Problem 1 Data



Problem 2 Data



Problem 3 Data

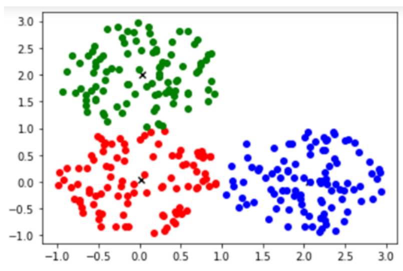
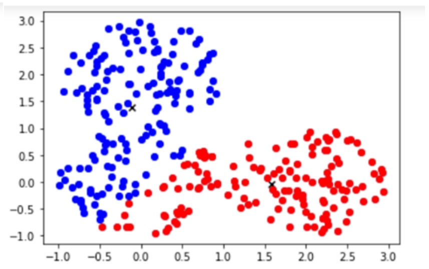
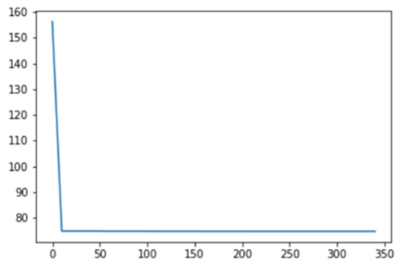
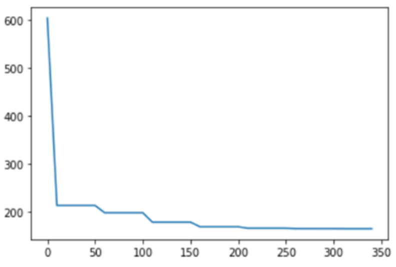
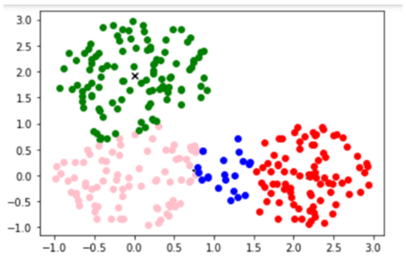
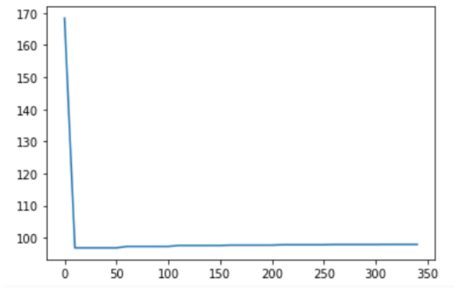


Problem 4 Data

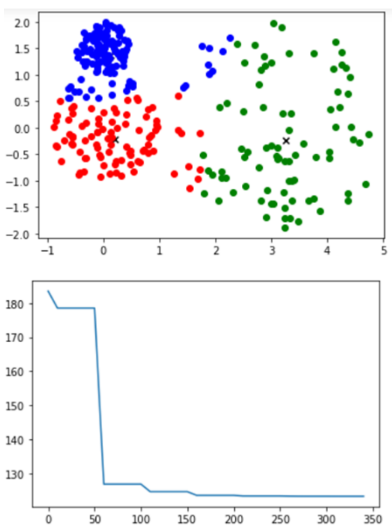
The data generates in from probability density coincides with expected plot, so we can assume that the code is correct.

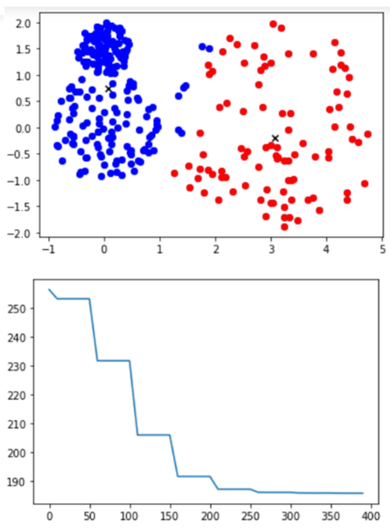
Next, let's view the clustering results for each problem

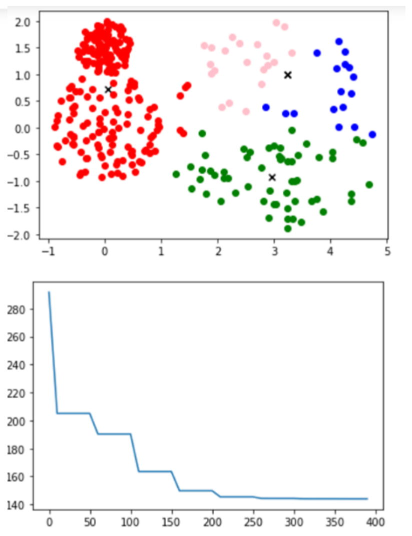
B C D

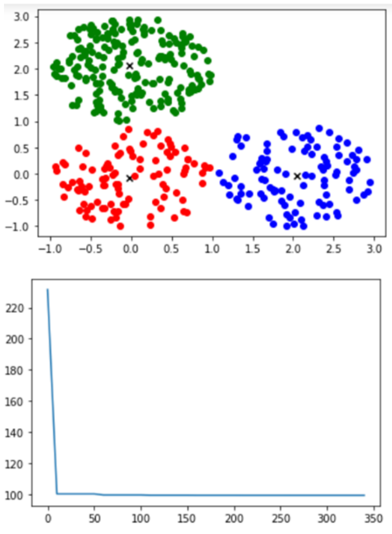
Above are results of problem 1. Note that sometimes they don't converge for 3 and 4 cluster. i.e. sometimes weights are initialized so, that even with 3 weight vectors there are only 2 clusters as one of vectors is very far from others, but this happens rarely. Four clusters don't converge more often, this may be since data naturally cannot be divided so. As this data can be easily divided into 3 clusters, three_clusters function converges very quickly compared to others.

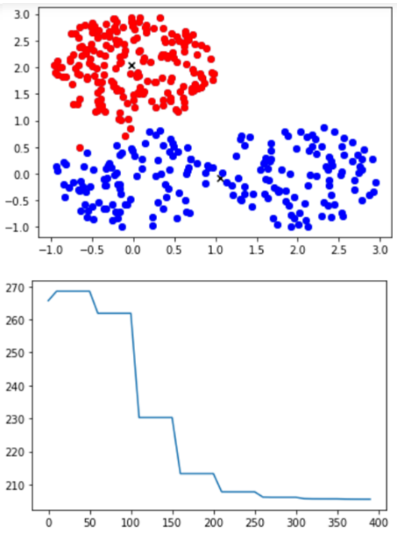

B

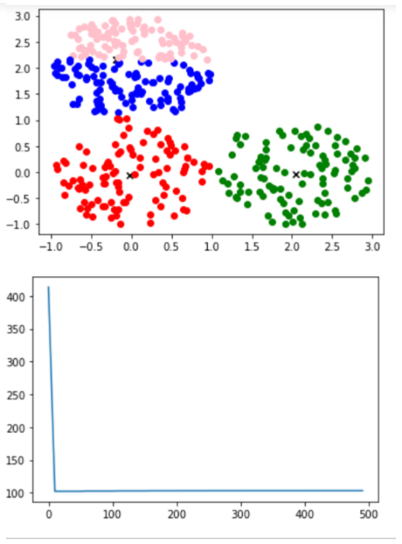

C


D

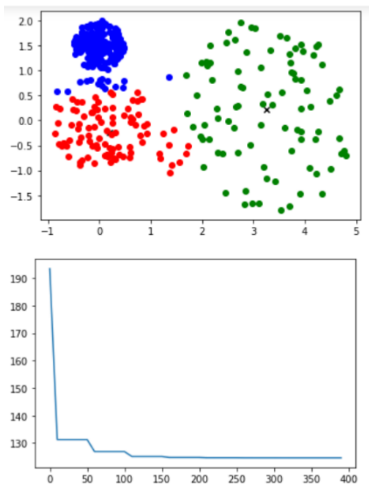
In problem 2 data is more distributed, so the clusters are not so evident. Additionally, the number of epoch are increased in some cases as E was not minimized at that time. As previously, is could not cluster well for the four-cluster case.

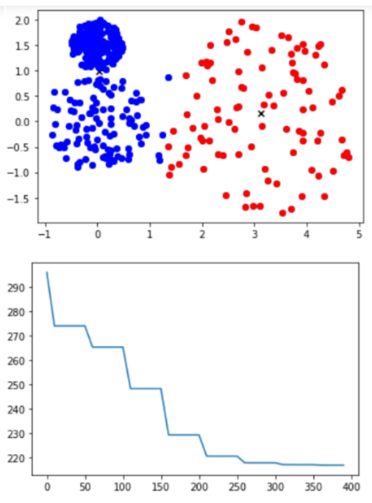

B

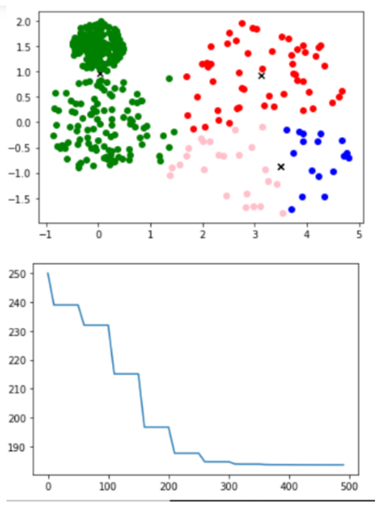

C


D

In the problem 3 again three and two cluster division have no any difficulties whenever four clusters most of time divide only into three or clusters due to the natural distribution, I provided most lucky result when clusters could be made into four. Also, for this time number of epoch should be increased a lot as E cannot be optimized quickly. We see that making learning rate parameter smaller during learning helps scientifically as loss decreases faster.







B C D

In problem 4 all divisions were hard to perform as data is very mixed. That's why number of epoch is big and result of clustering is not so evident. Also, for 4 clusters it NN could not categorize and the result was mostly 2 or 3 clusters. But when I increased epoch, sometimes it loss started to increase at the end which implies overfitting and idea that it is hard to make this data into 4 clusters. So it should be tuned very carefully. Above is the lucky case.

Further modification for making neural network better can be making learning rate parameter as a function which decreases by small amount every number of iterations and learning process continues without epoch number, but while etha is positive, so we reduce parameters and learning stops when step linearly decreases to 0. Also, the weight initialization can be controlled. I could not optimize it, that's why sometimes when random initialization is bad, result is not satisfactory. The good thing is to initialize weights far enough to each other even if you don't know the data at the beginning.

Conclusion

In conclusion, competitive learning code could achieve the goal in these problems, but it depends on lucky initial conditions and 'good' data distribution as there are numerous other modification that can be added and parameters tuned, unfortunately there is time limit for this homework and I cannot work more often with it.