Introduction to Machine Learning
Lab Session 4

Group 39
Lubor Budaj (s4167376) & Gasan Rzaev (s3553213)

October 2, 2022

1. Introduction

The goal of this assignment is to implement **Winner-Takes-All unsupervised competitive learning (VQ)**, for the given data *simplevqdata*. The dimensions of the data are 1000 by 2, which means there is 1000 data points with 2 values each. In comparison to LVQ that was performed in previous assignment, this implementation will not label the data. Since the data is not classified the closest prototype (the winner) will move only towards the data points. Apart from this the algorithm steps are similar to the LVQ algorithm.

Firstly, the prototypes are initialized randomly by choosing one of the data points from the whole data set. Each epoch is performed on the whole data set in randomized order. The winning prototype is the one with minimal distance (squared Euclidean distance) to the data point, and the update step is identical to LVQ apart from the fact that winning prototype is always moved towards the data point. The position of the prototypes are saved in a 3-dimensional vector in order to keep track of the movement of each prototype, so that in the end of the whole training process the plot of the prototypes' trajectory to the final position can be produced. Finally, after each epoch (**Not after each individual update step**) the quantization error is calculated as the sum of distances from each data point to the closest prototype.

2. Method

2.1. Prototypes vector

Important part of our implementation was storing the trajectory of prototypes' position after each epoch, so that in the end we could make a trajectory plot. We used 3-dimensional vectors to keep track of the prototypes positions throughout all the epochs. The third dimension of the prototypes vector stands for the epoch number, so that the prototype k and its position during epoch number t would be presented in the following way:

- `prototypes(k,1,t)` for the first value of the prototype
- `prototypes(k,2,t)` for the second value of the prototype

Additionally the size of the third dimension is $t_{max} + 1$, since there are t_{max} updates and one initialized position.

2.2. Number of prototypes

Our implementation can output the results for multiple number of prototypes at the same time. The number of prototypes in each run depends on how the variables are initialized in the beginning

of our code, namely K_{min} K_{max} and K_{step}, where K_{min} is the smallest number of prototypes, K_{max} is the largest number of prototypes and K_{step} stands for the increment value, in regards to number of prototypes, between K_{min} and K_{max}. In the code that is provided at the end of this report, the values for number of prototypes (K_{min} and K_{max}) are set to 2 and 4 respectively, such that most of the figures used in this report can be produced.

2.3. t_{max} AND η CHOICE

The values of learning rate η and the number of epochs t_{max} can be easily changed in initialization phase of our implementation. We used this fact to experiment with their values in order to find the best configuration. Firstly we tried to use proposed valued $\eta = 0.1$. This proved to us to be too high - the results seemed to use too unstable, so we chose lower values of $\eta = 0.01$ and $\eta = 0.005$. For each learning rate, the maximum amount of epochs was chosen specifically, so that there is enough time for the learning curve to become stable and so that the quantization error can be seen to be approximately constant.

3. RESULTS

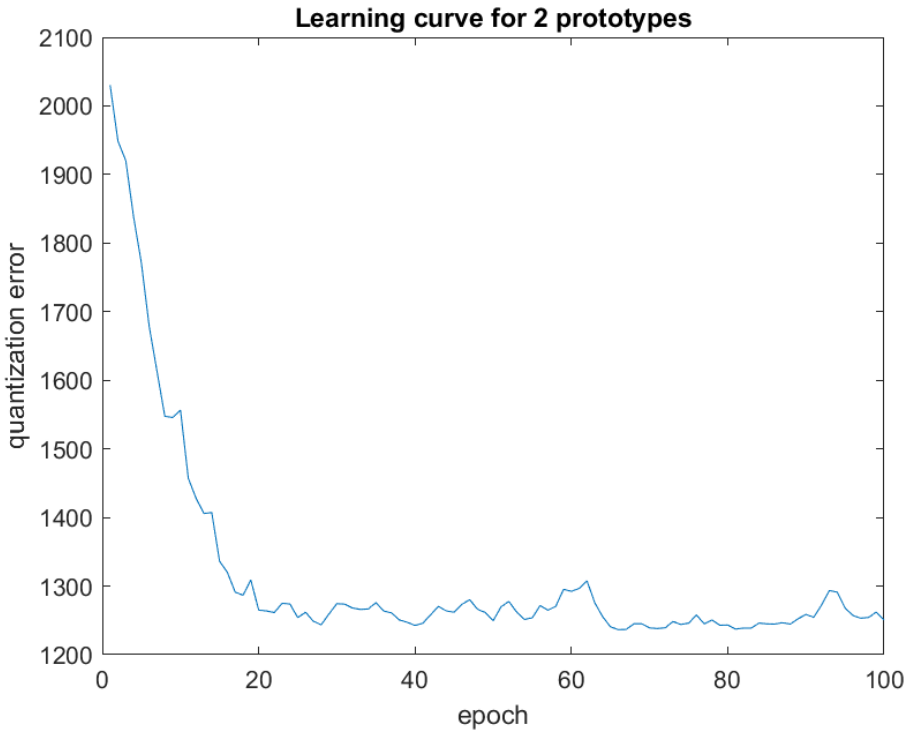

Figure 1: $\eta = 0.1, t_{max} = 100$

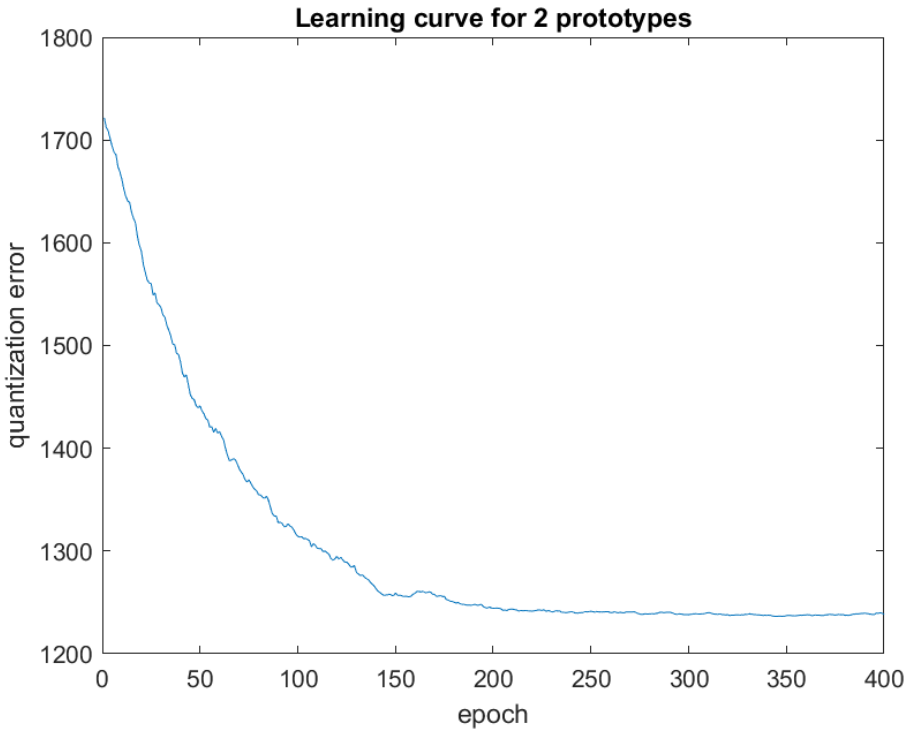

Figure 2: $\eta = 0.01, t_{max} = 400$

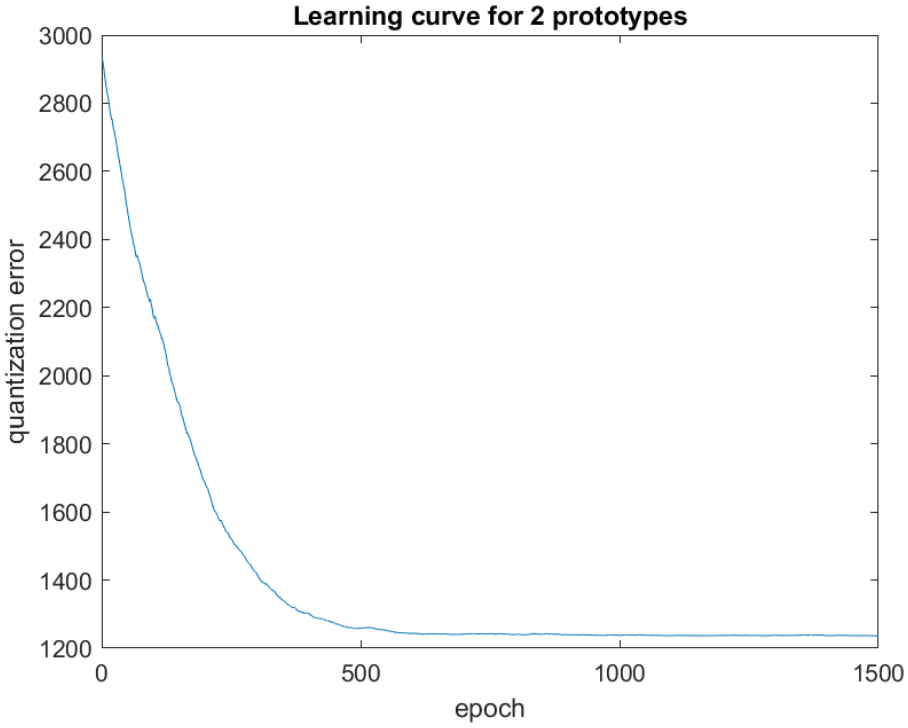

Figure 3: $\eta = 0.005, t_{max} = 1500$

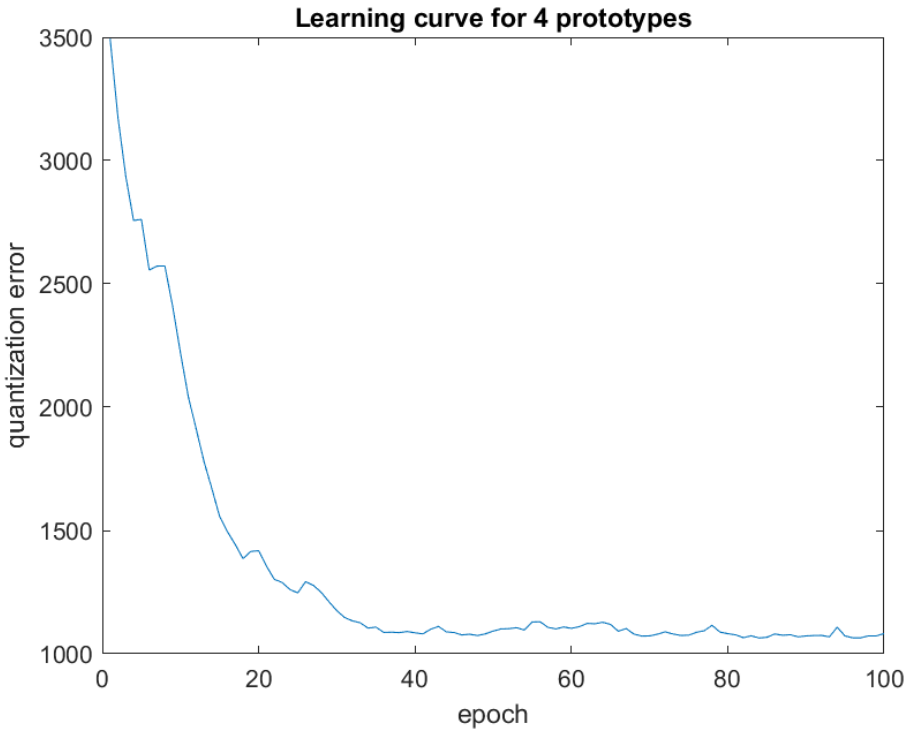

Figure 4: $\eta = 0.1, t_{max} = 100$

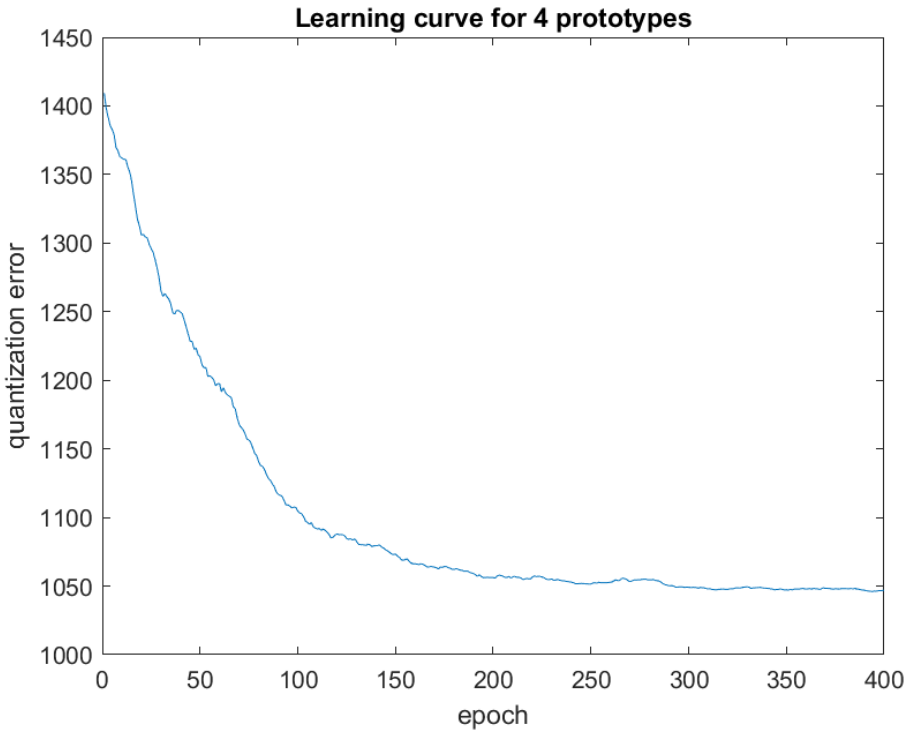

Figure 5: $\eta = 0.01, t_{max} = 400$

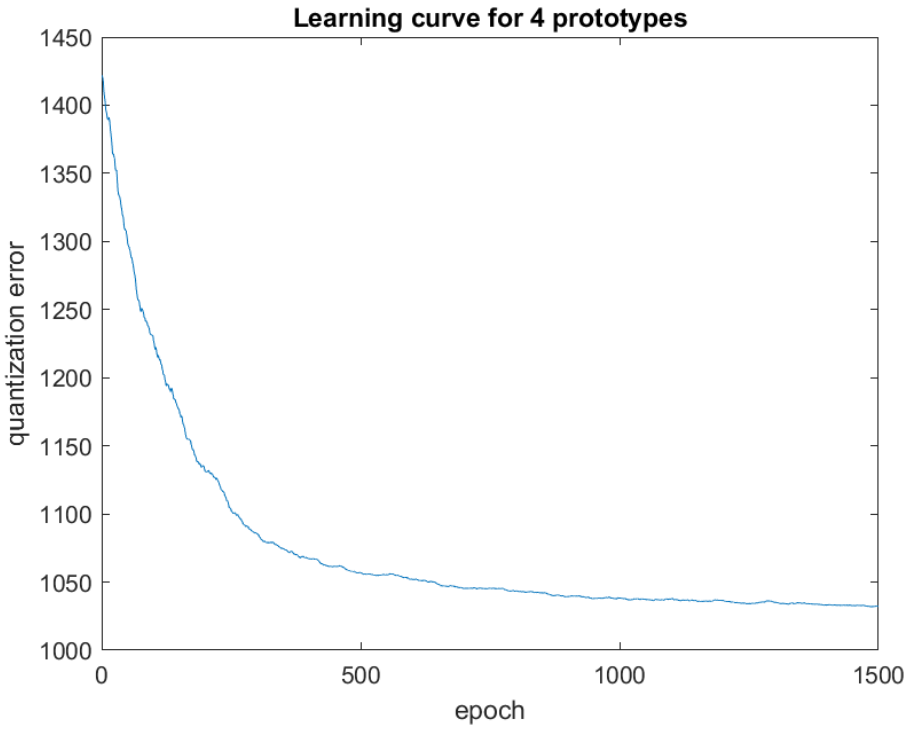

Figure 6: $\eta = 0.005, t_{max} = 1500$

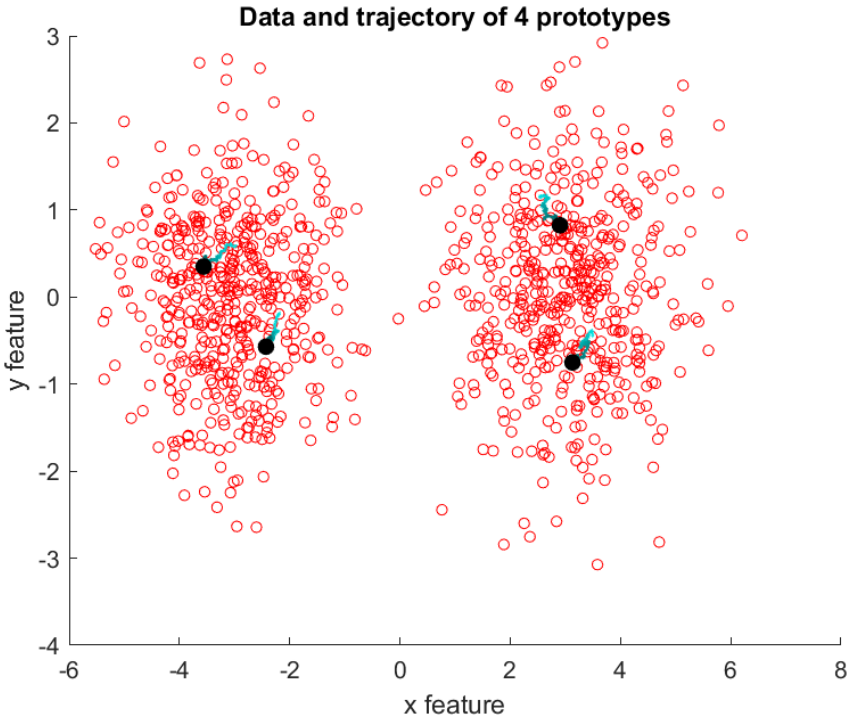

Figure 7: $\eta = 0.01, t_{max} = 400$ *Cyan line segments indicate the trajectory of the prototypes. Darker the color, higher the epoch number at which the prototype was at given point.*

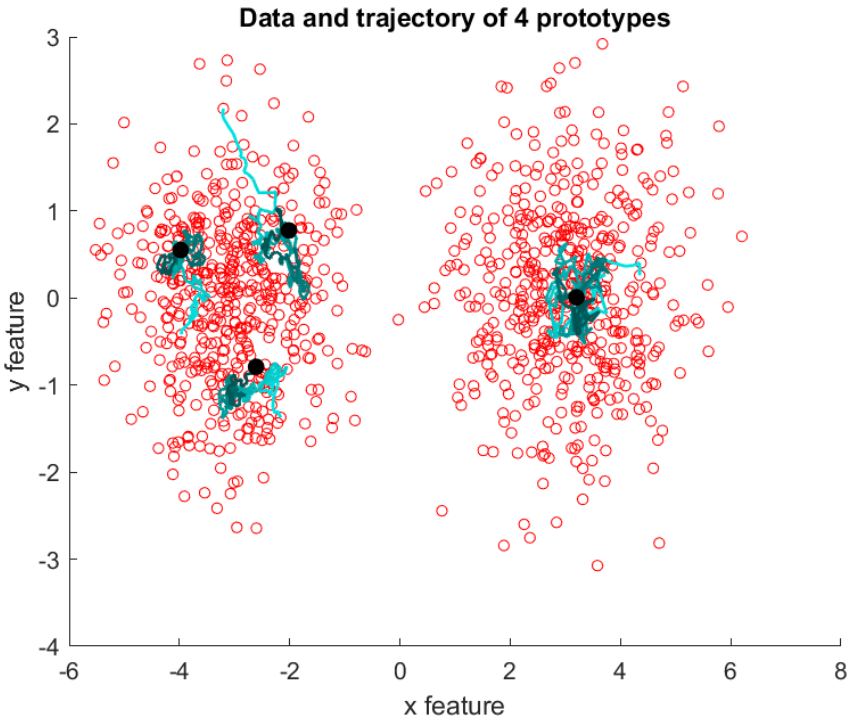

Figure 8: $\eta = 0.1, t_{max} = 100$ *Cyan line segments indicate the trajectory of the prototypes. Darker the color, higher the epoch number at which the prototype was at given point.*

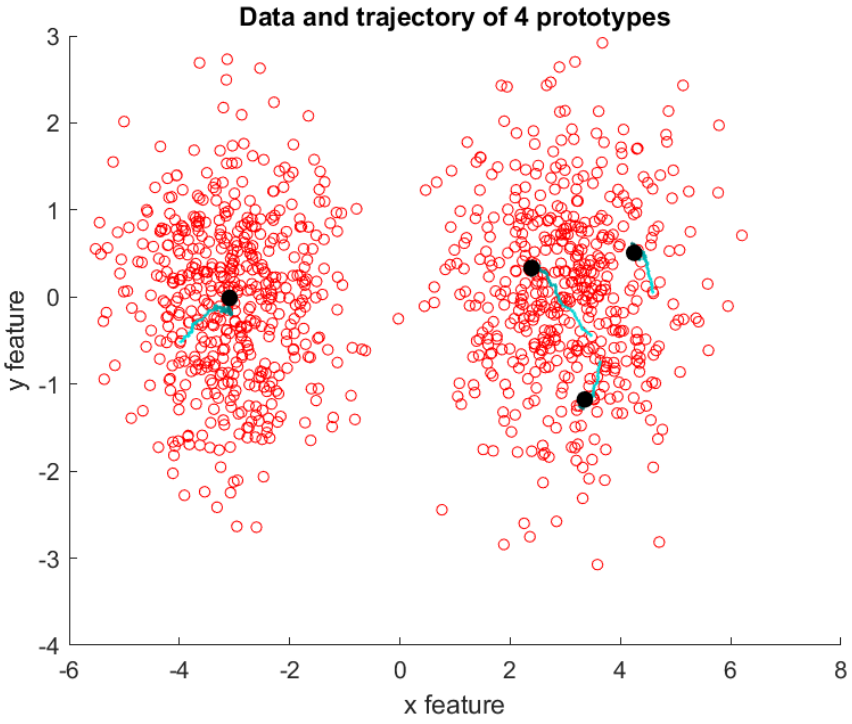

Figure 9: $\eta = 0.01, t_{max} = 400$ *Cyan line segments indicate the trajectory of the prototypes. Darker the color, higher the epoch number at which the prototype was at given point.*

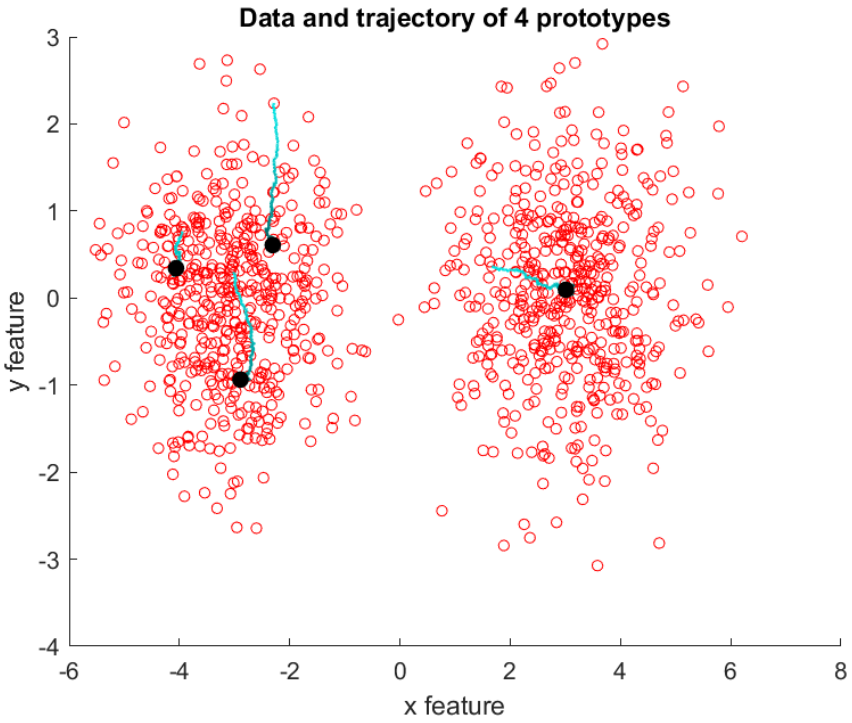

Figure 10: $\eta = 0.005, t_{max} = 1500$ *Cyan line segments indicate the trajectory of the prototypes. Darker the color, higher the epoch number at which the prototype was at given point.*

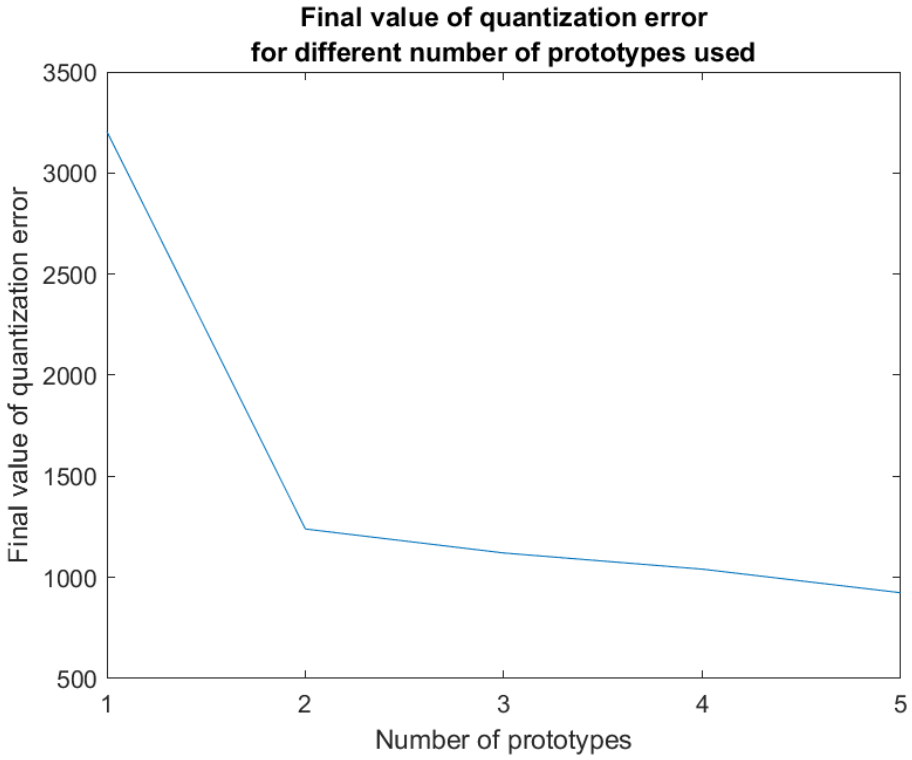

Figure 11: $\eta = 0.01, t_{max} = 400$

4. Discussion

4.1. The influence of the learning rate on quantization error

Firstly, let's discuss the influence of the learning rate (η) on the final value of quantization error, supposing maximum epoch number (t_{max}) is chosen appropriately. In figures (Figure 1, Figure 2, Figure 3) we can see the learning curves for 2 prototypes. When $\eta = 0.1$ (Figure 1), the quantization error is roughly 1250. When $\eta = 0.01$ (Figure 2), the quantization error is approximately 1240 and When $\eta = 0.01$ (Figure 3), the quantization error is approximately 1230. From this observation we can see that the quantization error slightly decreases as we decrease the learning rate (and increase the maximum number of epoch appropriately). We can see similar trend when we compare the learning curves for 4 prototypes (Figure 4, Figure 5, Figure 6). The final learning rate is approximately 1050, 1050 and 1030 for $\eta = 0.1$, $\eta = 0.01$, $\eta = 0.005$ respectively. From these 2 observations we conclude that as we decrease the learning rate, the final quantization error slightly decreases, given t_{max} is increased appropriately. We expected the final quantization error to decrease as we decrease the learning rate, however we expected the change in error to be greater. The change in final quantization error is caused by the fact that a lower learning rate allow for a finer refinements of the positions of the prototypes.

4.2. Learning rate value too high or too small

When η is too high, the resulting learning curve is unstable. There are big differences in error between the consecutive epochs. This may be good in some cases, for example if there are a lot of local minima from which we want the prototypes to get out from, but in general this can cause the learning curve to be more unstable (for example Figure 1). Hence it might be difficult to say when it is a good time to end the learning and the optimal solution may not ever be reached. On the other hand, when η is too small, there are opposite effects. It is possible that the prototypes will get stuck in a local minimum, but the resulting learning curve is more stable and it is easier to say when it stabilizes. Another disadvantage of having too small learning rate is that it takes more epochs for the prototypes to reach their approximate final position, therefore smaller learning rates are more computationally costly. To exploit both advantages of the smaller and the higher learning rates, we suggest to start with a high learning rate ($\eta = 0.1$) and decrease it as the epoch number increases.

4.3. The trajectory of the prototypes

In figures (Figure 7, Figure 8, Figure 9, Figure 10) we can see the trajectory of the prototypes throughout the epochs. Cyan line segments indicate the trajectory of the prototypes. Darker the color, higher the epoch number at which the prototype was at given point. In the figures we can see how prototypes ended up in their final positions - approximately in the middle of each cluster in the data. When we compare higher learning rate in Figure 8 to the rest of figures, we can see that the steps after each epoch are visibly larger and that the trajectory is very unstable. Compared to the other figures it seems that the final position of the prototypes has never reached a local minimum - the prototypes in the other figures don't oscillate as much. This is the result we expected. For its explanation we can use our reasoning from the previous subsection. Learning rate of 0.1 is too high for this particular data set and the prototypes will never settle in a local minimum. What we found interesting in the trajectory plots is the fact that if the 3 prototypes were initialized in one cluster of data and the last one in the other cluster, the prototypes would stay in their initial clusters throughout the experiment. We expected there to be in the end 2 prototypes in each cluster, however, when we thought about the problem further we understood why each prototype remained in it's own cluster - because all data points from the cluster with only one prototype are assigned to it, the second prototype will never have a chance to move over there.

If we wanted to always end up in 2 prototypes in each cluster of data, we would have to implement rank based update system, so that the second closest prototype's position would get updated too. If experimented further to see what would happen if all the prototypes were initialized in the same cluster of data. We found out that one of the prototypes would move to the other cluster, but the would stay in their initial cluster. Lastly, it is important to say that the configuration with 2 prototypes per cluster in the initial configuration (Figure 7) has a lower final quantization error compared to the configuration with 3 to 1 ration of prototypes per cluster (Figure 9) for the same values of η and t_{max}.

4.4. Use of the elbow method to decide the appropriate number of prototypes

Lastly, in Figure 11 we can see a comparison of final quantization error for different number of prototypes, when we set $\eta = 0.01$ and $t_{max} = 400$. In the figure it can be clearly seen, that the final quantization error greatly decreased when we increased the number of prototypes from 1 to 2, but further marginal increases in the number of prototypes do not yield such great increase in performance. Hence, using the elbow method we deem 2 prototypes configuration to be the most appropriate representation of the data. This is the result we expected - from the observation of the scatter plot of the data (Figure 7) it seems to be clear that we can divide the data in 2 clusters - each for one prototype.

5. Work Distribution

The work among the group members was distributed in a following way:

- Code: 50% done by Gasan, 50% by Lubor
- Report: 50% done by Lubor, 50% by Gasan
- Graphs: 50% done by Lubor, 50% by Gasan

The code used to complete this assignment, that has not been included above, is presented in Listing 1 in Appendix A.

A. Appendix

Listing 1: *This is the file* `code.m`

```
%import
data = importdata("simplevqdata.mat");

%initialization of dimensions of the data
dimensions = size(data);
N = dimensions(2);
P = dimensions(1);

%range of number of prototypes, set for same value to run just for one
%value of K
K_min = 2;
K_max = 4;
K_step = 2;
K_num = fix((K_max - K_min) / K_step) + 1;

%learning rate
n = 0.01;

%maximum number of epochs
t_max = 400;

%initialization of error vectors
H_VQ = zeros(K_num,t_max);

%loop for different amounts of prototypes per class
for K = K_min:K_step:K_max

    %initialization of prototypes
    prototypes = zeros(K,N,t_max+1);

    %random prototype selection
    random_prototypes = randperm(P,K);
    for k = 1:K
        prototypes(k,:,1) = data(random_prototypes(k),:);
    end

    %row in error matrix
    pos = fix((K - K_min) / K_step) + 1;

    %for each epoch
    for t = 1:t_max

        %random permutation of examples
        p = randperm(P);

        %for each example
        for x = 1:P
```

```
        %finding closest prototype
        closest_prototype = 1;
        minimal_distance = sqrt((data(p(x),1) - prototypes(1,1,t))
^2 + (data(p(x),2) - prototypes(1,2,t))^2);

        for k = 2:(K)
            distance = sqrt((data(p(x),1) - prototypes(k,1,t))^2 +
 (data(p(x),2) - prototypes(k,2,t))^2);
            if distance < minimal_distance
                closest_prototype = k;
                minimal_distance = distance;
            end
        end

        %moving distance calculation
        distX = n * (data(p(x),1) - prototypes(closest_prototype,1,
t));
        distY = n * (data(p(x),2) - prototypes(closest_prototype,2,
t));

        %moving
        prototypes(closest_prototype, 1,t+1) = prototypes(
closest_prototype, 1,t) + distX;
        prototypes(closest_prototype, 2,t+1) = prototypes(
closest_prototype, 2,t) + distY;

    end

    for x = 1:P

        %finding closest prototype
        closest_prototype = 1;
        minimal_distance = sqrt((data(p(x),1) - prototypes(1,1,t))
^2 + (data(p(x),2) - prototypes(1,2,t))^2);

        for k = 2:(K)
            distance = sqrt((data(p(x),1) - prototypes(k,1,t))^2 +
 (data(p(x),2) - prototypes(k,2,t))^2);
            if distance < minimal_distance
                closest_prototype = k;
                minimal_distance = distance;
            end
        end

        %Error addition
        H_VQ(pos,t) = H_VQ(pos,t) + minimal_distance;

    end

end
```

```
%PLOTS

figure(K*3);
hold on

% Plot the data
scatter(data(:, 1), data(:, 2), 20, 'red');

%Plot the prototype final positions
scatter(prototypes(:, 1, t_max+1), prototypes(:, 2, t_max+1), 50, '
black', 'filled');

hold off
str = sprintf('Data and final positions of %d prototypes', K);
title(str);
xlabel('x feature');
ylabel('y feature');

% Plot the data and the trajectory of the prototypes
figure(1+K*3);
hold on

% Plot the data
scatter(data(:, 1), data(:, 2), 20, 'red');

% Plot the trajectory of prototypes positions
for k = 1:K
    color = [0, 0.9, 0.9];
    for idx = 1:t_max
        color = color - [0, 0.6/t_max, 0.6/t_max];
        x = [prototypes(k, 1, idx), prototypes(k, 1, idx+1)];
        y = [prototypes(k, 2, idx), prototypes(k, 2, idx+1)];
        plot(x,y, 'LineWidth', 1.3, 'Color', color);
    end
end

% Add the final positions of the prototypes
scatter(prototypes(:, 1, t_max+1), prototypes(:, 2, t_max+1), 50, '
black', 'filled');

hold off
str = sprintf('Data and trajectory of %d prototypes', K);
title(str);
xlabel('x feature');
ylabel('y feature');

hold on
figure(2+K*3)
plot(H_VQ(pos,:))
str = sprintf('Learning curve for %d prototypes', K);
```

```
    title(str);
    xlabel('epoch');
    ylabel('quantization error');
    hold off
end

%plot comparing final quantization error for all Ks used
figure(1);
plot(H_VQ(:,t_max));
xticks(K_min:K_step:K_max);
xticklabels(string(K_min:K_step:K_max));
xlabel("Number of prototypes");
ylabel("Final value of quantization error");
title(["Final value of quantization error", "for different number of
   prototypes used"]);
```