Introduction to Intelligent Systems
Lab 5

Team 15:
Sara Bardají Serra and Albert Sallés Torruella

October 2019

Introduction

In this assignment we are asked to implement Winner-takes-all unsupervised competitive learning (VQ).

Firstly, lets mention what unsupervised learning consists of. Unsupervised learning is a machine learning technique for which there is no need to supervise the model. It is mostly used to reduce the amount of data by using only a few prototypes to represent it, to represent all data using only a few features instead of all of them, and or clustering, so to identify groups of similar data within the dataset.

VQ Winner-takes-all algorithm is a type of unsupervised competitive learning and the basic idea of how it works is as follows:

- The first step is to initialize K prototype vectors
- Then take points in a random order and present them as examples
- The following step is to identify the closest prototype for each point to identify the winner
- The final step is to move the winner closer to the example

The last three steps form an epoch of the algorithm, and must therefore be repeated as many times as epochs we want the algorithm to perform.

Methodology

The first thing to do is implement the Winner-takes-all algorithm, and to do so we are going to build a function following the steps from the assignment. Our function will have the next signature:

$$VQ(data, k, n, tmax)$$

Therefore, the function has four parameters, where the first one is the dataset of size N, k is the number of prototypes, n is the learning rate and t_{max} is the amount of epochs the

algorithm is going to perform. The structure of the code and the steps it performs are as follows:

- First select k random points from the dataset. These points will now be used as the prototypes.
- Then shuffle the points to randomly permutate the order in which the points will be presented as examples.
- Next, for every point calculate the distance to every prototype, and take the prototype to which it is nearer as the winner.
- Once the winner is found, move the winner closer to the point it is winner to. The distance it is move is n (learning rate) times the distance between the point and the winner.

When these steps are fulfilled for all points, an epoch has been completed. Therefore, they are repeated as many times as t_{max}.

Whilst an epoch is being completed, the quantization error is also computed. The way it is calculated is using the following formula:

$$H_{VQ} = \sum_{j=1}^{k} \sum_{\mu=1}^{N} (\xi^{\mu} - w_j)^2 \prod_{k \neq j}^{k} \Theta(d_k^{\mu} - d_j^{\mu}), \text{ where } \Theta(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

However, we can compute this easier in our algorithm. Firstly, we notice that the function $\Theta(d_k^{\mu} - d_j^{\mu})$ is zero when $d_k^{\mu} - d_j^{\mu} < 0$, therefore when $d_k^{\mu} < d_j^{\mu}$, that is, the data point ξ^{μ} is closer to w_k than to w_j, for all $k \neq j$. But that is true for all the prototypes w_k except one, the closest w_j to ξ^{μ}, because there is no other prototype with less distance to the data point. So, basically, we can rewrite the function to this:

$$H_{VQ} = \sum_{\mu=1}^{N} (\xi^{\mu} - w_j)^2, \quad j \in \{1, ..., k\}$$

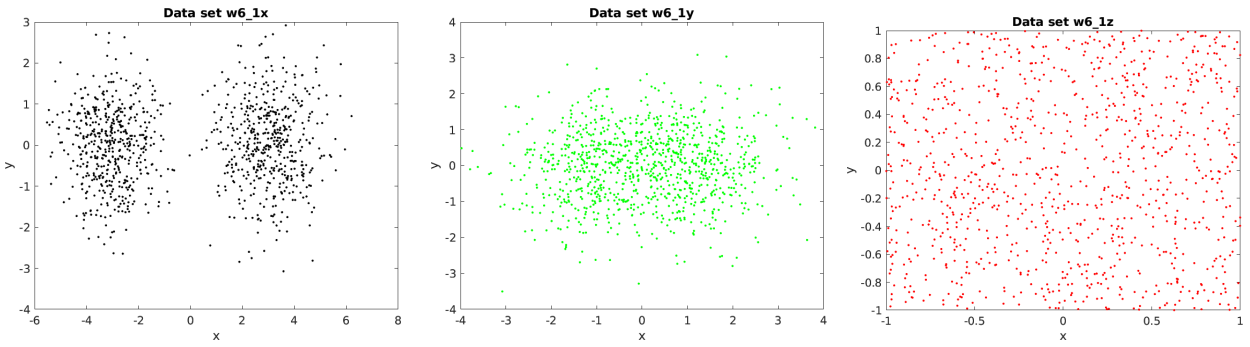
where w_j is the winner for ξ^{μ}.

Now, we know how to compute the quantization error in an easier way. We sum the squares of the minimum distance between each data point and every prototype.

Once the implementation of the winner-takes-all algorithm is done, we perform several experiments using the parameters of the function to test different configurations. Our first aim is to test different values of t_{max} in order to find a reasonable value of t_{max} for which the quantization error function seem to reach a good local minimum. To achieve this, we use 20, 40 and 100 as different values of t_{max}. For values of k we are only going to use 2 and 4, as we are mostly interested in how different value of n (learning rate) affect the quantization error function. Consequently, we are going to perform tests using four different learning rates: 0.5, 0.3, 0.1 and 0.05.

Results

First we have plotted the different datasets in order to visualise the data. The plots we have obtained can be seen in Fig. 1.



(a) (b) (c)

Figure 1: Visual representation of dataset (a) w6_1x (b) w6_1y (c) w6_1z

In the first dataset we can clearly see that there are two clusters defined. Also, in the second dataset there is a big centered cluster. However, in the third dataset we see that the data points are scattered around. So we might think that the last dataset will be hard to cluster.

Next, we have performed our vector quantization algorithm on each dataset to check how they behave. In this case, we have used K = 2 clusters, n = 0.1 for the learning rate and a t_{max} of 20 epochs. In section A, we find the examples for each dataset. In the first case, Fig. 3, we can see the performance of the algorithm for the w6_1x dataset and how fast it approaches a local minimum. Also, in Fig. 4 and Fig. 5 we can see the same examples for the w6_1y and w6_1z datasets respectively. With respect to the w6_1z dataset, we notice that there is no clear single minimum because there are several positions where the prototypes can be placed forming different clusters. Finally, we can see the quantization error function $H_{VQ}(t)$ in each case in Fig. 2.

Now that we have introduced the different datasets, we proceed to find a reasonable value of t_{max} for which the quantization error function $H_{VQ}(t)$ reaches a good minimum. Therefore, using the settings K = 4 and K = 2 clusters and n = 0.1 for the learning rate, we experiment with different values for t_{max}. In section B, in the Fig. 6 and Fig. 7 we can see how the quantization error function behaves for $t_{max} = 100$ and $t_{max} = 40$ respectively. In both cases, we notice that the function does not improve a lot, and therefore we are unnecessarily performing more epochs. So we conclude that the configuration with $t_{max} = 20$ is the best option to find a good minimum, as we have previously seen in Fig. 2.

Finally, we experiment with different values for the learning rate to discuss the effect on the quantization error function. For this experiment, we will only use the first dataset (w6_1x), as we have seen it is rather easy to find a good minimum in.

In section C, we find the results of our experiment. At first, we have used large values like $n = 0.5$ and $n = 0.3$ as we can see in Fig. 8. In this first case, we clearly see that we

never reach a minimum, this happens because the prototypes are moved too much towards the nearest data points, so if, for example, the last data points are very far from the cluster (they are outliers) then the prototype is moved away of the cluster, they are very influenced by outliers.

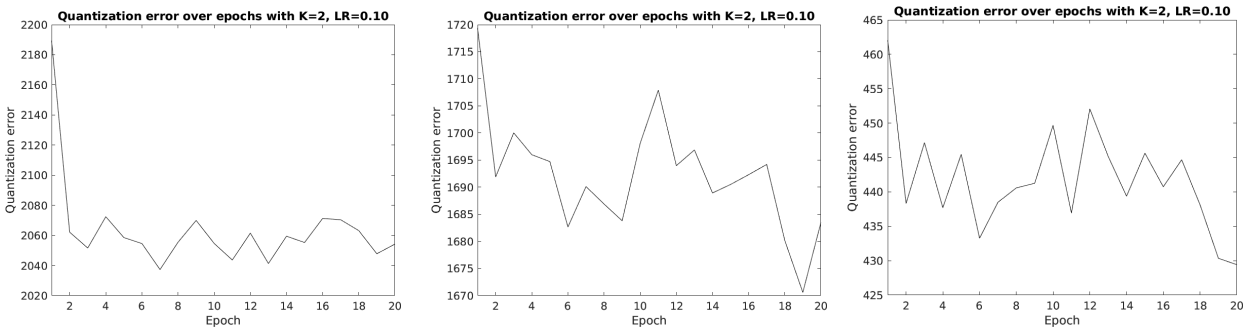
On the other hand, if we take lower values like $n = 0.1$ and $n = 0.05$, as we can see in Fig. 9, this effect does not happen anymore. However, in the case of $n = 0.05$, it seems that the cost function cannot escape a local minimum because it's always around the same values. This means that the prototypes get stuck in a good position but probably not the best, which is much better than the case before. But if we have a look at the cost function of $n = 0.1$, we see that it tries to escape local minima because it finds new values without being affected too much by the outliers.

Therefore, in this experiment, we can conclude that the best learning rate is $n = 0.1$ as it finds different minima but it is not much influenced by data points far away from the cluster.

Work division

During the realization of this assignment, we have been working and making decisions together, so we have divided the work equally. Also, the both of us have written and checked this report.

A Dataset visualization through epochs



(a) (b) (c)

Figure 2: Visual representation of $H_{VQ}(t)$ for (a) w6_1x (b) w6_1y (c) w6_1z

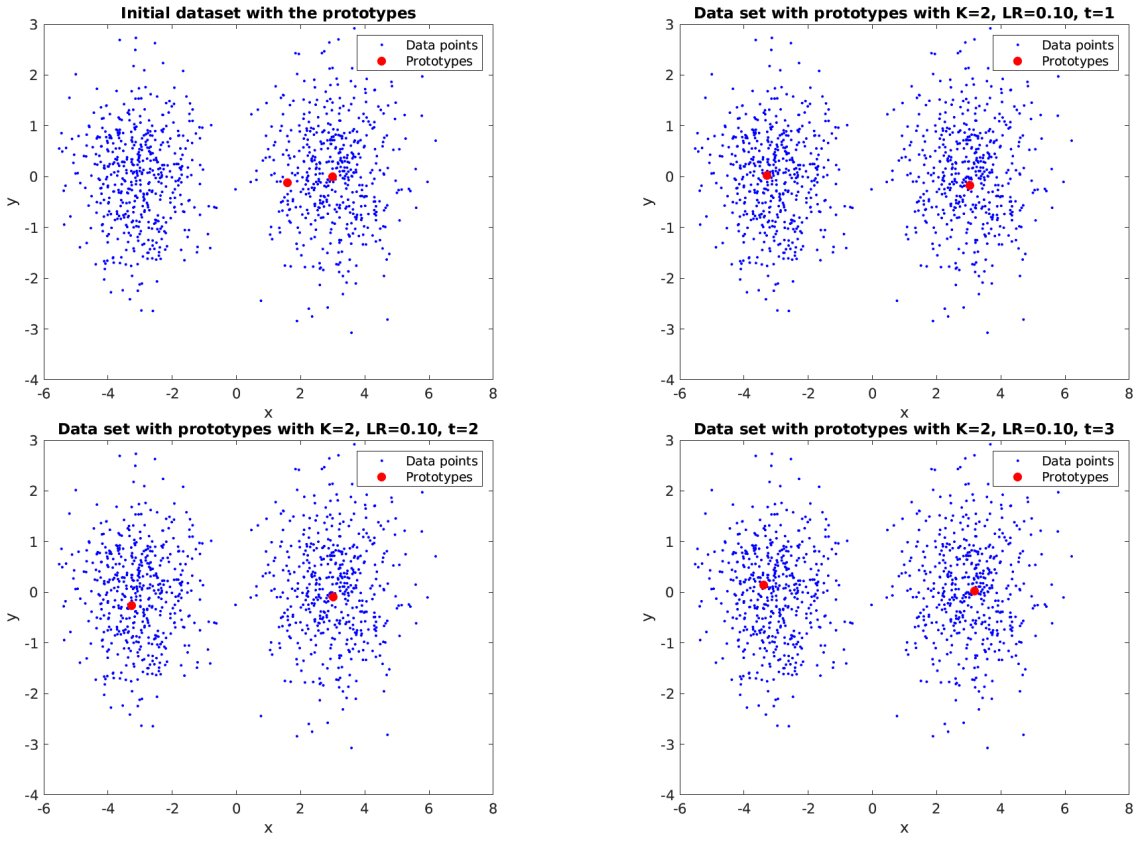


Figure 3: Positions of the prototypes over the w6_1x dataset after 3 epochs

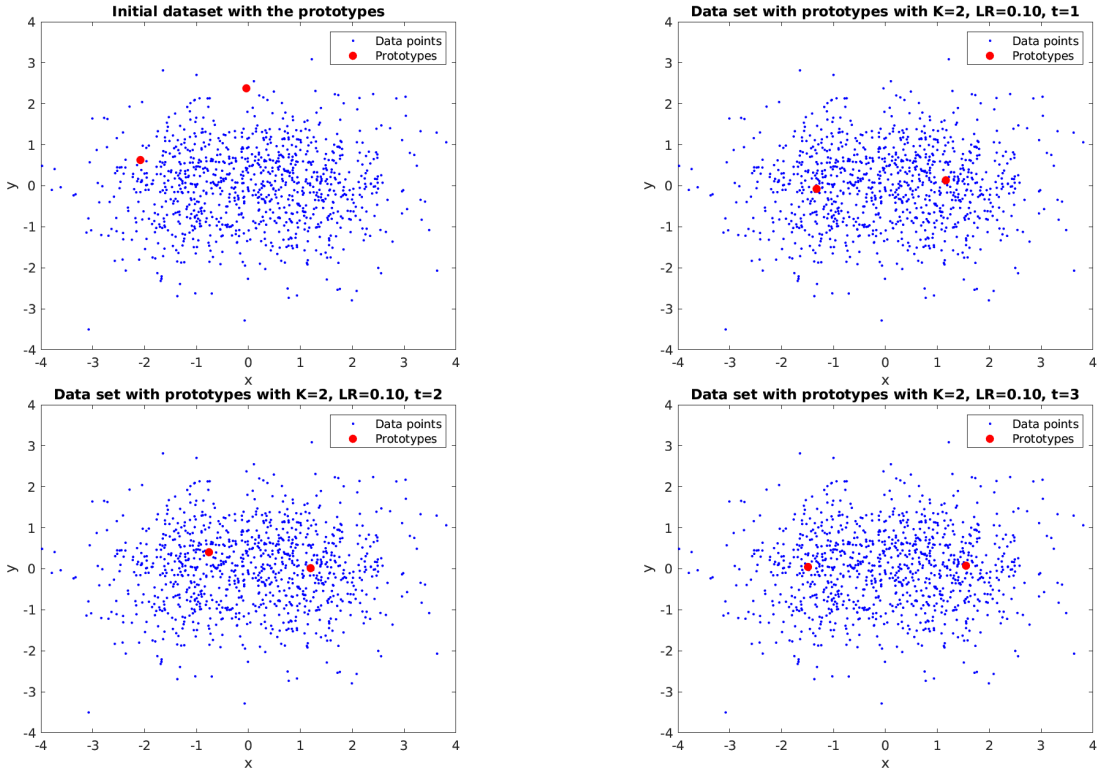


Figure 4: Positions of the prototypes over the w6_1y dataset after 3 epochs

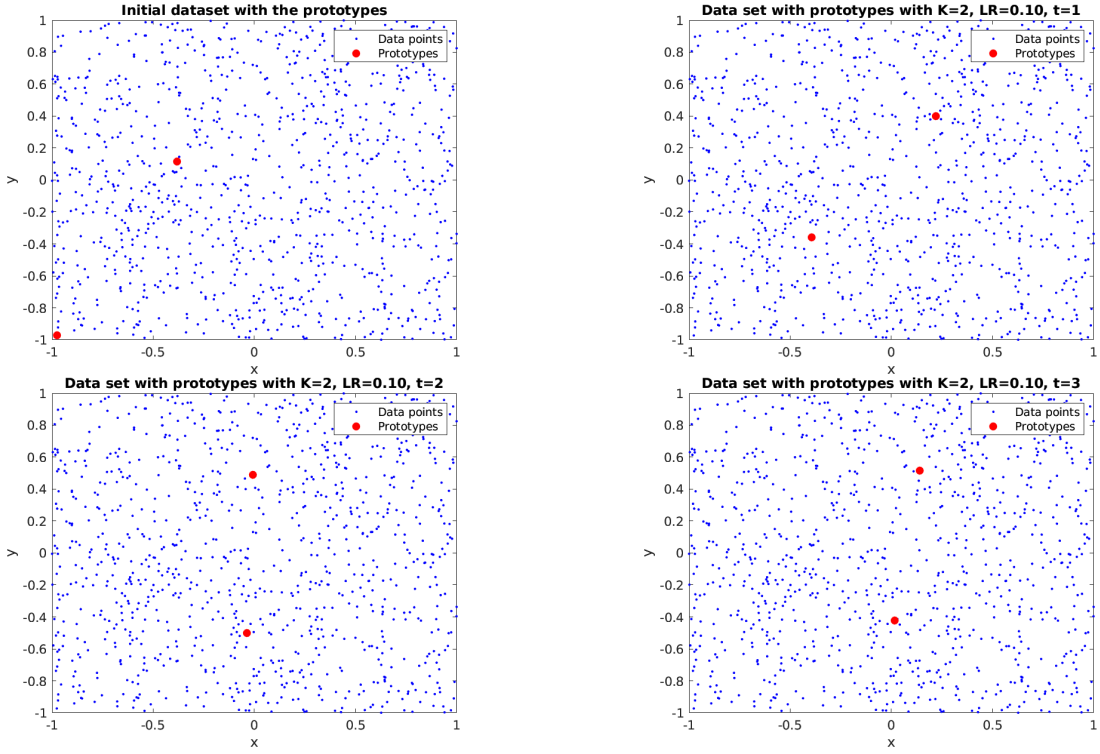
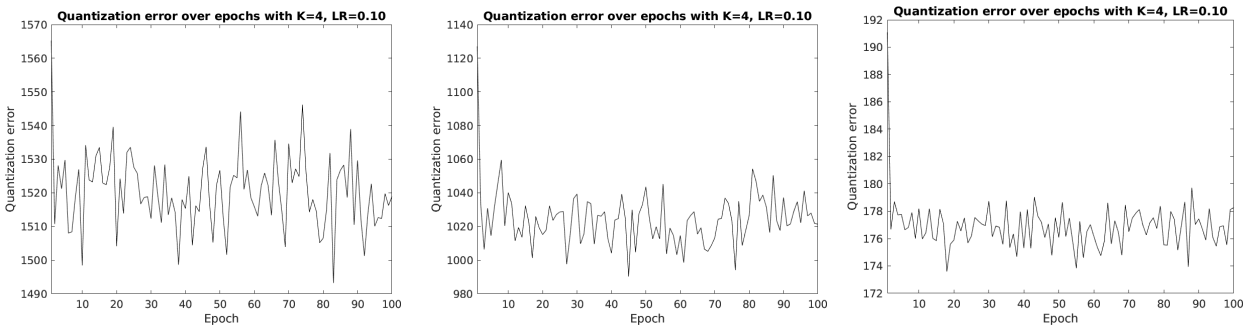


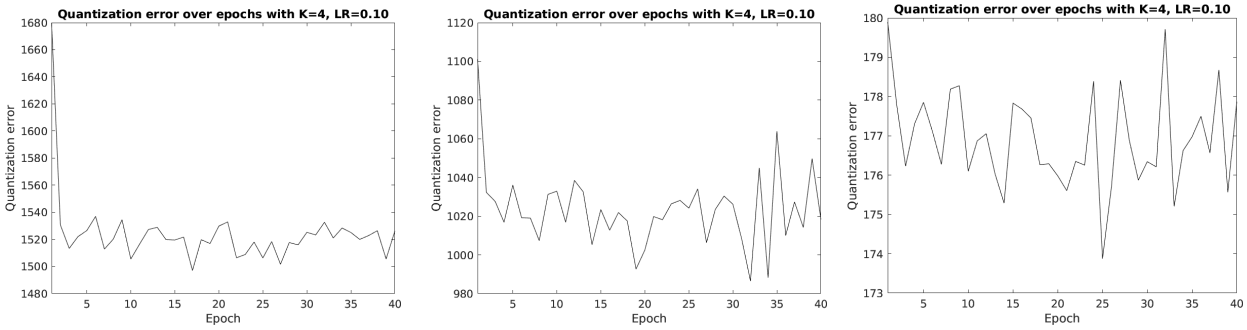
Figure 5: Positions of the prototypes over the w6_1z dataset after 3 epochs

B Experimentation with different maximum epochs



(a) (b) (c)

Figure 6: Visual representation of $H_{VQ}(t)$ for (a) w6_1x (b) w6_1y (c) w6_1z for 100 epochs



(a) (b) (c)

Figure 7: Visual representation of $H_{VQ}(t)$ for (a) w6_1x (b) w6_1y (c) w6_1z for 40 epochs

C Experimentation with different learning rate values

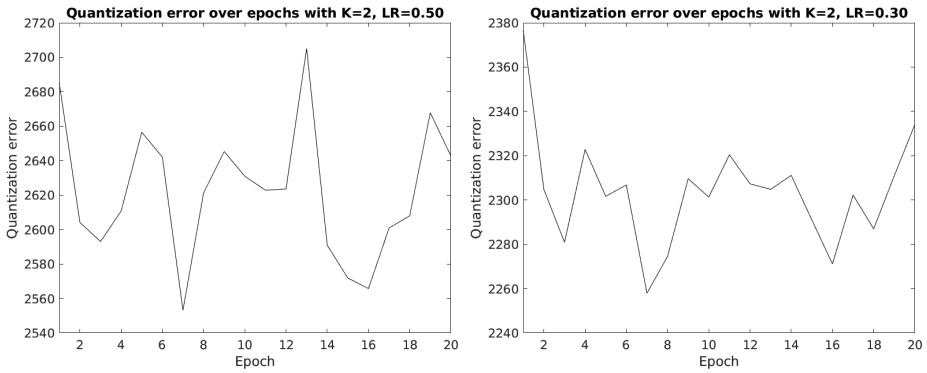


Figure 8: Visual representation of $H_{VQ}(t)$ for $n = 0.5$ (left) and $n = 0.3$ (right)

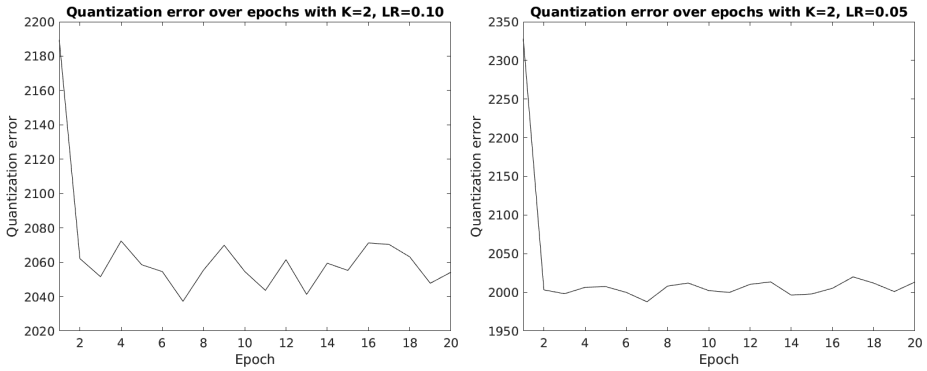
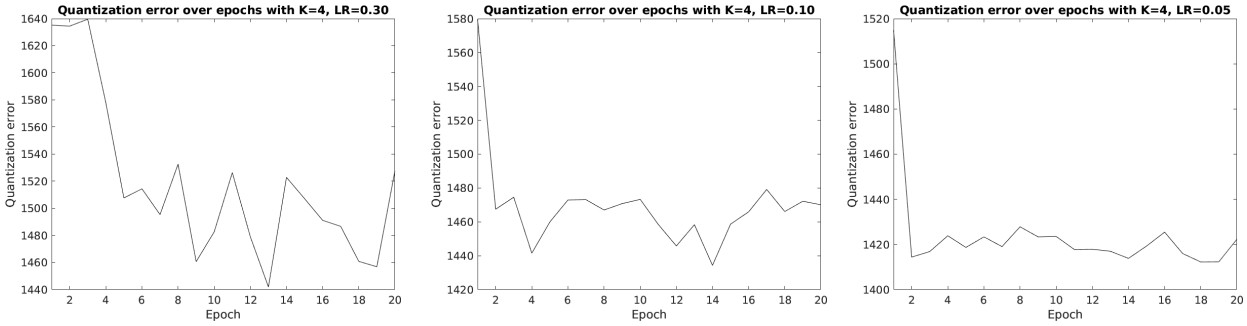


Figure 9: Visual representation of $H_{VQ}(t)$ for $n = 0.1$ (left) and $n = 0.05$ (right)



(a) (b) (c)

Figure 10: Visual representation of $H_{VQ}(t)$ for (a) $n = 0.3$ (b) $n = 0.1$ (c) $n = 0.05$ for w6_1x dataset

D Generated code

```
% Read the data:
load w6_1x.mat
load w6_1y.mat
load w6_1z.mat

% Plot the datasets for visualization
plot(w6_1x(:,1),w6_1x(:,2), 'k.')
title('Data set w6\_1x')
xlabel('x')
ylabel('y')
plot(w6_1y(:,1),w6_1y(:,2), 'g.')
title('Data set w6\_1y')
xlabel('x')
ylabel('y')
plot(w6_1z(:,1),w6_1z(:,2),'r.')
title('Data set w6\_1z')
xlabel('x')
ylabel('y')

```

```
% To set
% K = Number of prototypes
% n = Learning rate
% tmax = Max number of epochs

% Perform tests with different configurations
VQ(w6_1x, 2, 0.5, 20)
VQ(w6_1x, 2, 0.3, 40)
VQ(w6_1x, 2, 0.1, 20)
VQ(w6_1x, 2, 0.1, 40)
VQ(w6_1x, 2, 0.05, 20)
VQ(w6_1x, 2, 0.05, 40)

VQ(w6_1x, 4, 0.3, 20)
VQ(w6_1x, 4, 0.3, 40)
VQ(w6_1x, 4, 0.1, 20)
VQ(w6_1x, 4, 0.1, 40)
VQ(w6_1x, 4, 0.05, 20)
VQ(w6_1x, 4, 0.05, 40)

VQ(w6_1y, 2, 0.3, 20)
VQ(w6_1y, 2, 0.3, 40)
VQ(w6_1y, 2, 0.1, 20)
VQ(w6_1y, 2, 0.1, 40)
VQ(w6_1y, 2, 0.05, 20)
VQ(w6_1y, 2, 0.05, 40)

VQ(w6_1y, 4, 0.3, 20)
VQ(w6_1y, 4, 0.3, 40)
VQ(w6_1y, 4, 0.1, 20)
VQ(w6_1y, 4, 0.1, 40)
VQ(w6_1y, 4, 0.05, 20)
VQ(w6_1y, 4, 0.05, 40)

VQ(w6_1z, 2, 0.3, 20)
VQ(w6_1z, 2, 0.3, 40)
VQ(w6_1z, 2, 0.1, 20)
VQ(w6_1z, 2, 0.1, 40)
VQ(w6_1z, 2, 0.05, 20)
VQ(w6_1z, 2, 0.05, 40)

VQ(w6_1z, 4, 0.3, 20)
VQ(w6_1z, 4, 0.3, 40)
VQ(w6_1z, 4, 0.1, 20)
VQ(w6_1z, 4, 0.1, 40)
VQ(w6_1z, 4, 0.05, 20)
VQ(w6_1z, 4, 0.05, 40)

function VQ(data, k, n, tmax)

    N = size(data, 1);
    prototypes = datasample(data, k);

    % Plot initial state of the data
```

```
    figure
    plot(data(:, 1), data(:, 2), 'b.')
    hold on
    plot(prototypes(:, 1), prototypes(:, 2), 'ro', 'MarkerFaceColor','r')
    title('Initial dataset with the prototypes')
    xlabel('x')
    ylabel('y')
    legend('Data points', 'Prototypes')
    hold off

    cost_function = zeros(1, tmax);
    for t = 1:tmax
        % Randomly permutate the indexes
        shuffled_indexes = randperm(N);
        % Initiate the cost to 0
        cost = 0;
        for i = shuffled_indexes
            % Calculate the distance of point i to every prototype
            distances = pdist2(data(i,:),prototypes);
            % Take the minimum distance
            [d, p] = min(distances);
            % Move the winner
            prototypes(p,:) = prototypes(p,:) + n*(data(i,:) - prototypes(
               p,:));
            % Compute the cost
            cost = cost + d.^2;
        end

        % Build title for the plot
        p_title = 'Data set with prototypes with K=';
        p_title = strcat(p_title, sprintf("%d", k));
        p_title = strcat(p_title, ', LR=');
        p_title = strcat(p_title, sprintf("%.2f", n));
        p_title = strcat(p_title, ', t=');
        p_title = strcat(p_title, sprintf("%d", t));

        % Plot the data for the current epoch
        figure
        plot(data(:, 1), data(:, 2), 'b.')
        hold on
        plot(prototypes(:, 1), prototypes(:, 2), 'ro', 'MarkerFaceColor','
           r')
        title(p_title)
        xlabel('x')
        ylabel('y')
        legend('Data points', 'Prototypes')
        hold off

        cost_function(t) = cost;

    end

    % Plot final state of the data
    figure
```

```
126     plot(data(:, 1), data(:, 2), 'b.')
127     hold on
128     plot(prototypes(:, 1), prototypes(:, 2), 'ro', 'MarkerFaceColor','r')
129     title('Final dataset with the prototypes')
130     xlabel('x')
131     ylabel('y')
132     legend('Data points', 'Prototypes')
133     hold off
134 
135     % Build title for the plot
136     p_title = 'Quantization error over epochs with K=';
137     p_title = strcat(p_title, sprintf("%d", k));
138     p_title = strcat(p_title, ', LR=');
139     p_title = strcat(p_title, sprintf("%.2f", n));
140     % Plot quantization error or all epochs
141     figure
142     plot(1:tmax, cost_function, 'k')
143     xlabel('Epoch')
144     ylabel('Quantization error')
145     title(p_title)
146     xlim([1, tmax])
147     hold off
148 
149 end
```

Code