# Vector Quantisation Implementation Report

Team 30
Dekel Viner(s2612925)  Andreea Glavan (s3083691)

October 2018

## Problem description

In the given assignment we implemented Winner-Takes-All unsupervised competitive learning (VQ) to the different data sets provided, taking varying values for $\eta$ and K, the learning rate and the number of prototype vectors, respectively. Using VQ, we aim to cluster data by using a few prototype vectors representing a large amount of data. Each time a feature vector is presented, the closest prototype based on a the squared Euclidean distance is moved in the direction of this feature vector based on $\eta$. VQ updates only the winner, and considers one feature vector at a time, as opposed to K-means. The quantization error is optimized based on the total distance(based on the squared Euclidean distance) of feature vectors to prototype vectors.

## Results

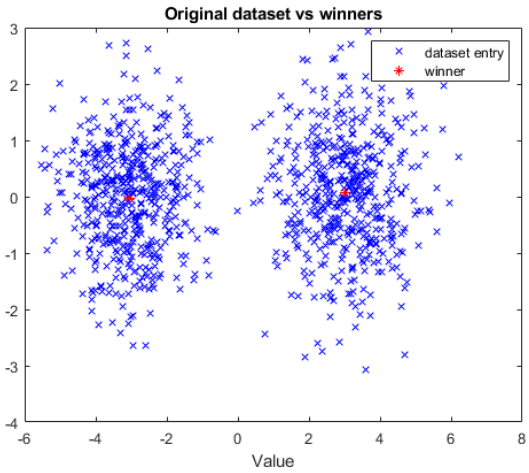


Figure 1: Dataset population(blue x's) vs final positions of the winners(red asterisks)

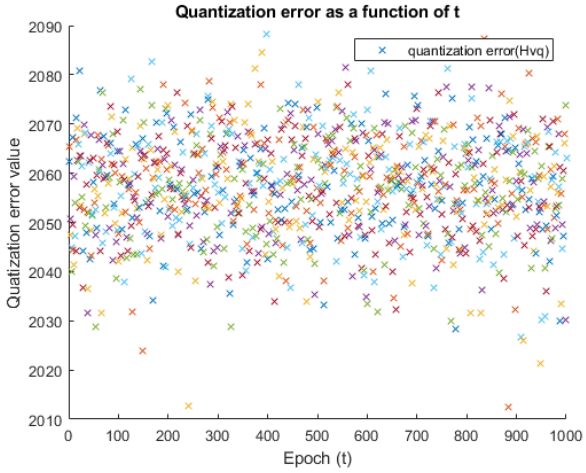


Figure 2: Learning curve for $\eta = 0.1$ and k=2

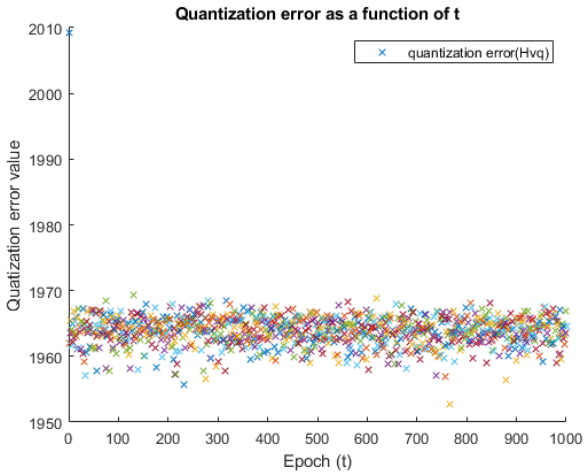


Figure 3: Learning curve for $\eta = 0.01$ and k=2

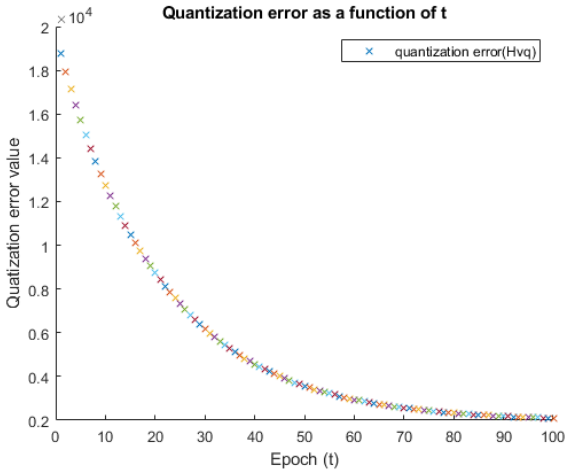


Figure 4: Learning curve for $\eta = 0.0005$ and k=2

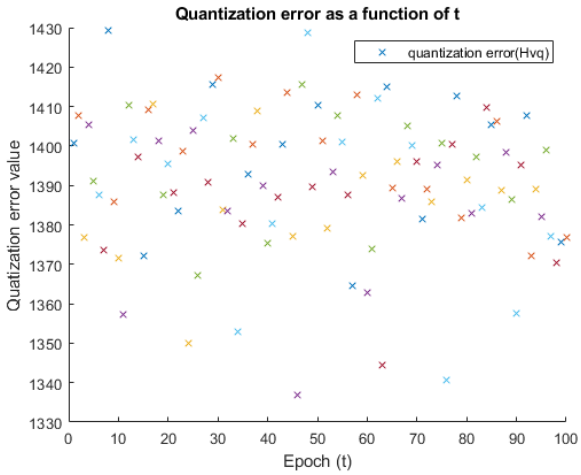


Figure 5: Learning curve for $\eta = 0.1$ and k=4

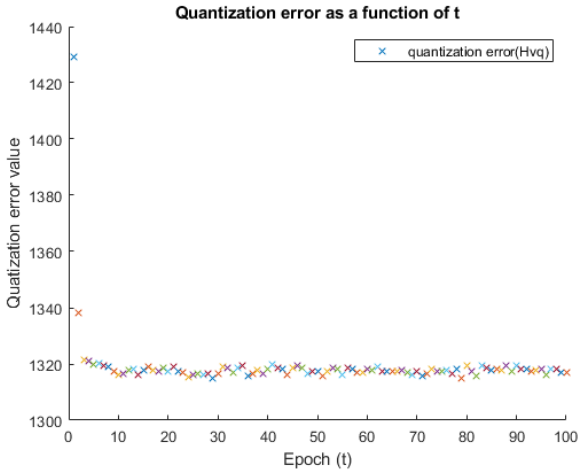


Figure 6: Learning curve for $\eta = 0.01$ and k=4

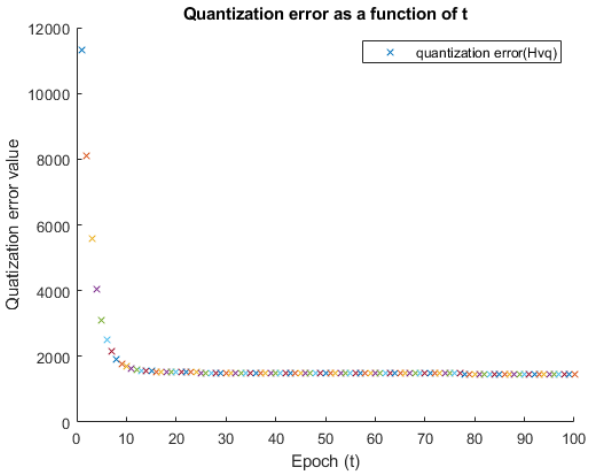


Figure 7: Learning curve for $\eta = 0.0005$ and k=4

# Discussion

This discussion is based on the plots illustrated above, for which the code is available in the appendix. We used the given dataset $w6_1x$ to generate them, thus the results are linked to those values.

Figure 1 illustrates the plot of the datasets and the final positions of the winners after applying VQ competitive learning. The figure showcases that the final prototypes(winners) are placed in the center of the two clearly distinct clusters formed from the original dataset. What is more, the winners are relatively symmetric to each other. This could of course be attributed to the fact that the approximate centers of the two clusters are parallel. It is notable that there are plenty of outliers in the original dataset, as shown in the plot.

The plot of the quantization error($H_{VQ}$) as a function of t is directly affected by the $\eta$ value. Up to a certain value, the smaller the $\eta$ value is, the clearer the curve of the plot function. However, if $\eta$ is too small, the plot will look like a straight line descending from the left hand side. If, on the other hand, $\eta$ is too large, the plot will appear closer to a straight line, with only the first value being very high while the others are clustered in the lower part of the plot. A large $\eta$ value means a high contribution of a single data point to the prototype. Hence, with a higher $\eta$ value the plot also becomes messier with a higher variance between the points after the convergence of the curve.

Due to the fact that the data set consists of two clearly distinguishable clusters made up of 1,000 entries the learning occurs very fast. A learning rate of 0.01 means that even after one epoch the prototypes reach the centers of their clusters. This is clearly visible in figure 3 as the Quantization error converges onto its equilibrium values. On the contrary with a learning rate of 0.0005 a

single epoch is not enough to adjust the prototypes enough for them to reach their final value, in fact it takes around a 100 epochs for that to happen.
The final cost value is also proportional with $\eta$. The lower $\eta$ is, the lower the final cost. However, it is notable that the starting cost for lower $\eta$ is greater than the starting cost of greater $\eta$ values.

# Appendix

Listing 1: `Source code`

```
%load data
load('w6_1x.mat');
load('w6_1y.mat');
load('w6_1z.mat');

%determine variables
n=size(w6_1x,2);
p=size(w6_1x,1);

%set parameters
k=2;
eta=0.00005;
t_max=100;

%initialize prototype vectors
w=zeros(k);
w=datasample(w6_1x,k);

%begin h_vq as a func of t plot
figure
hold on;

%for all epochs
for t=1:t_max
    %shuffle dataset
    aux=randperm(p);
    %reset hvq
    h_vq=0;
    %perf 1 epoch of training
    for i=1:p
        %get shuffled values based on aux
        xi= w6_1x(aux(i),:);

        %calculate distances
        diss1 = pdist2(xi,w(1,:),'squaredeuclidean');
```

```
        diss2 = pdist2(xi,w(2,:),'squaredeuclidean');

        %compare distances, update vector and h_vq
        if(diss1<diss2)
            w(1,:) = w(1,:) + eta*(xi-w(1,:));
            h_vq=h_vq+diss1;
        else
            w(2,:) = w(2,:) + eta*(xi-w(2,:));
            h_vq=h_vq+diss2;
        end
    end
    %plot new hvq point
    plot(t,h_vq,'x');
end

%labels for first plot
title('Quantization error as a function of t');
xlabel('Epoch (t)');
ylabel('Quatization error value');
legend('quantization error(Hvq)');

%plot dataset and final winners
figure
plot(w6_1x(:,1),w6_1x(:,2),'bx');
hold on;
plot(w(:,1),w(:,2),'r*');
title('Original dataset vs winners');
xlabel('Value');
ylabel('');
legend('dataset entry','winner');
```