Introduction to Intelligent Systems - Lab 5

Tom Apol (s2701650)
Nielis Brouwer (s3850706)
Group: 23

October 15, 2018

Contents

1 Introduction **2**

2 Problem Description **2**

3 Results **2**

4 Discussion **3**

5 Conclusion **4**

6 Appendix **5**

6.1 Figures 5

6.1.1 Prototypes over time 5

6.1.2 Quantisation error over time 11

6.2 Code 17

1 Introduction

In this report we will describe the effects of different amount of prototypes (K) and learning rates (η) with regards to Vector Quantisation of a 2D data set.
This algorithm first randomly initialises the prototypes, after which their position is corrected by data points 'pulling' the closest prototypes closer to themselves.
The sets used for the experiments are 2D data sets with 1000x2 values where the first column indicates an x-value and the second column indicates a y-value. For our measurements, we have chosen data set `x` (or `w6_1x.mat` to be precise).

2 Problem Description

The problem as mentioned in the assignment is: "Implement Winner-Takes-All unsupervised competitive learning Vector Quantisation, and apply this method to the provided data sets using the Euclidean distance as a measure." Perform experiments for different number of prototypes K (K=2, K=4) and different learning rates η. The learning quality is represented by the quantisation error, the lower the error the "better" the prototypes represent the cluster of data points they are closest to.

3 Results

The following algorithm was used to compute the quantisation error, where $d_j^\mu = |\xi^\mu - w_j|$ denotes the Euclidean distance between data point μ and prototype j:

$$H_{VQ} = \underset{\text{prototypes}}{\sum_{j=1}^{K}} \; \underset{\text{data}}{\sum_{\mu=1}^{P}} \underbrace{\left|\xi^\mu - w_j\right|}_{d_j^\mu} \; \underbrace{\prod_{k\neq j}^{K} \Theta\left(d_k^\mu - d_j^\mu\right)}_{w_j \text{ is the winner !}}$$

For the results (= plots), see Appendix 6.1. With regards to the code: the code is split into several parts:

- Loading and initialising of variables and data structures: lines 1 - 41
- Computing the quantisation error (H_VQ) and performing the algorithm: lines 43 - 94
- Plotting both the prototypes over time (and data points), as well as the quantisation error over time: lines 96 - 169

What we can observe from our plots is the following: when run with a relatively high η (e.g. $\eta = 0.1$), the prototypes move in a rather volatile manner, where

outliers in particular seem to have a relatively large influence on their position, as seen in figures 1 and 4. When multiple prototypes reside in the same large cluster, this can result in the prototypes violently ‘pushing’ each other away, as seen in figure 4. With lower learning rates, both of these effects seem to lessen (see figures 2 and 5).
This is reflected by the quantisation error over time, where the quantisation error ‘stabilises’ early, but with higher values of η, there is still a lot of fluctuation (see figures 7, 8, 10 and 11).
However, with a learning rate that is very low (e.g. $\eta = 0.0001$), we see that the quantisation error becomes rather stable, yet it takes more time to stabilise. This is to be expected, as the step size (= learning rate) is rather low. (see figures 9 and 12). This is also well illustrated by figure 3, where P_1 only travels in a straight line towards its stabilisation point.
Furthermore, multiple prototypes fighting over the same cluster seem to influence each other less (see figure 6).

4 Discussion

Regarding the question 'what is a reasonable value for t_max, such that the quantisation error has stabilised at that point?': this depends on η and K. Furthermore, the stabilisation point is influenced by the initialisation of the prototypes, since a prototype that has been initialised further from its ‘stabilised’ position will take more epochs to get there. For $0.1 >= \eta >= 0.01$ and data set x, we can say that a t_max of around 20 seems alright. In figures 7, 8, 10 and 11 the stabilisation point generally lies in front of that point, but this is because the initial prototypes lie relatively close to their ‘stabilised’ positions, as seen in their respective ‘Prototypes over time’ figures 1, 2, 4 and 5. To account for this, we chose a t_max that is higher than the ‘stabilisation point’ in our plots.
For $\eta \approx 0.0001$ we can state that it seems to stabilise around 100 epochs or perhaps even later, dependent on K, as seen in figures 9 and 12. The reason for this lower stabilisation rate is the fact that a learning rate of this low magnitude slows down the learning significantly.

This brings us to the next question of interest: "What happens if η is too large or too small?". As discussed in the 'Results' section, for an η that is too large, the quantisation error ‘stabilises’ relatively early, but isn't very stable. This becomes especially evident when comparing figures 10 and 11. This is due to the prototypes being significantly influenced by the other data points (especially outliers) in their cluster, even after having approached their stabilisation point.
For an η that is too small, the quantisation error becomes very stable, but stabilises after a rather large amount of epochs. This becomes evident when comparing figures 8 and 9, and figures 11 and 12.

Finally we will answer the question "How does the final value of the cost function change with η?". For this we will take a look at figures 7 to 12. The final (stabilised) value of the cost function/quantisation error seems to slightly decrease when η decreases. However, since figure 12 doesn't seem to fully stabilise yet at our t_max of 100, it is difficult to tell whether this holds for K=4.

5 Conclusion

The following conclusions can be summarised from the data presented above:

- We found that the value of the initial point of the prototype has great influence when determining a reasonable value for t_max. For data set x, we deem $\eta \approx 0.01$ to be a reasonable learning rate, and keeping the previous statement in mind, we present $t\_max = 20$ as an appropriate corresponding stabilisation point.
- The quantisation error stabilises late when the learning rate is relatively low, but the learning rate becomes very stable.
- A high learning rate results in a early stabilisation, but the quantisation error (and prototype location for that matter) keeps fluctuating around a 'stabilisation point'.
- The final value of the cost function seems to slightly decrease with a lower η.

6 Appendix

6.1 Figures

6.1.1 Prototypes over time

Colours are used to separate our prototypes and their paths, namely red, green, blue and magenta. The initial point of each prototype has been made lighter than the rest for readability purposes. The progression of prototypes is signified by a change in colour from a relatively dark to relatively bright version of their respective colour.
The variables K and η mentioned below represent the number of prototypes and the learning rate respectively. Furthermore, the data used comes from the provided `w6_1.mat` data set.

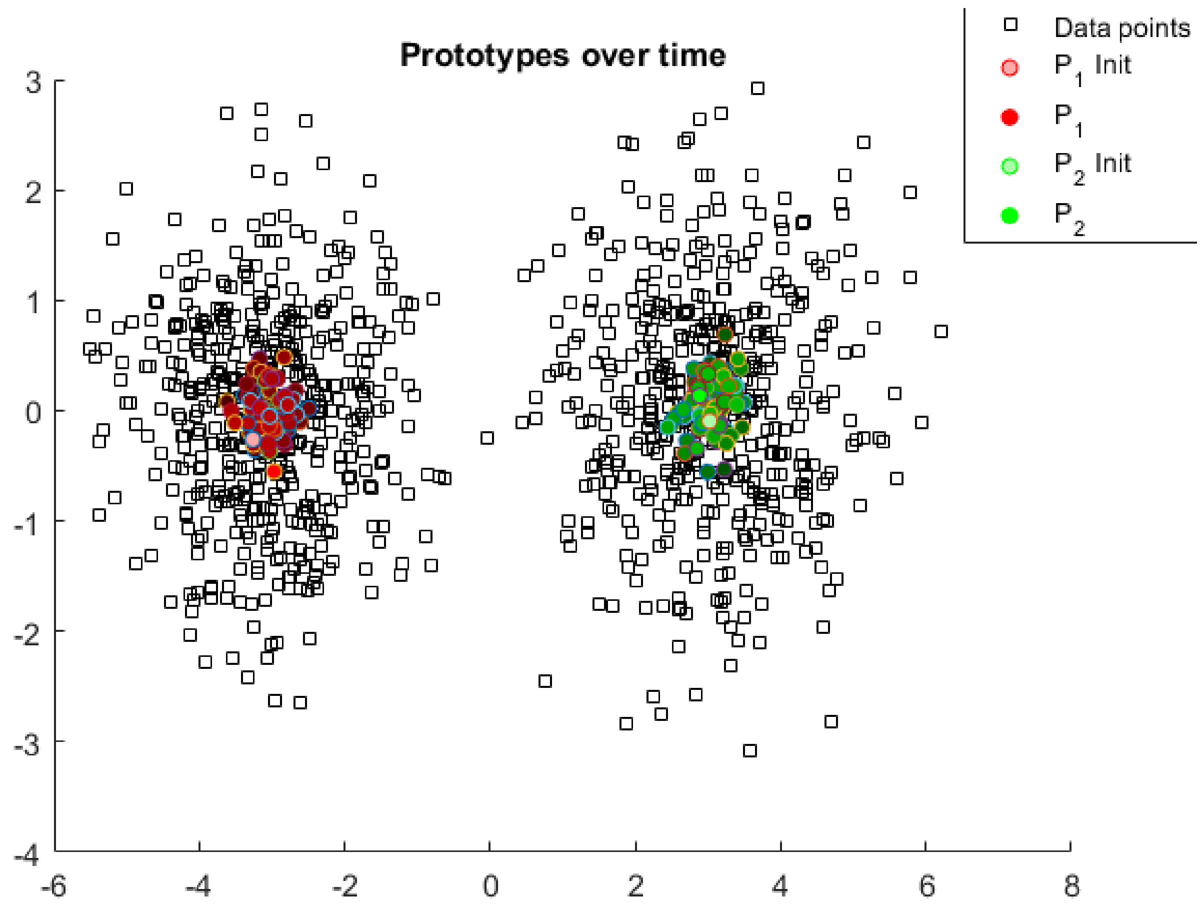

Figure 1: Prototypes over time with K=2 and η=0.1

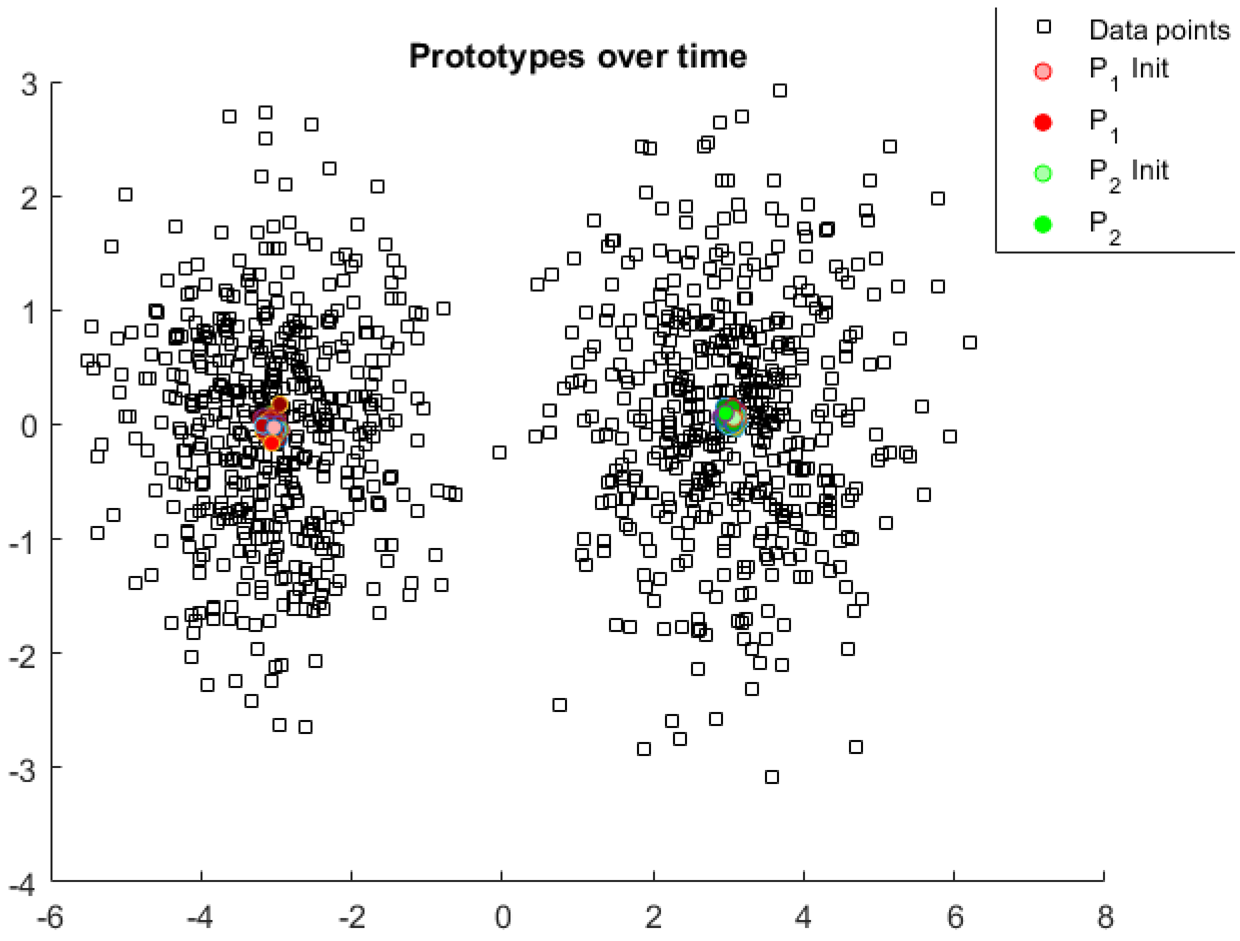

Figure 2: Prototypes over time with K=2 and η=0.01

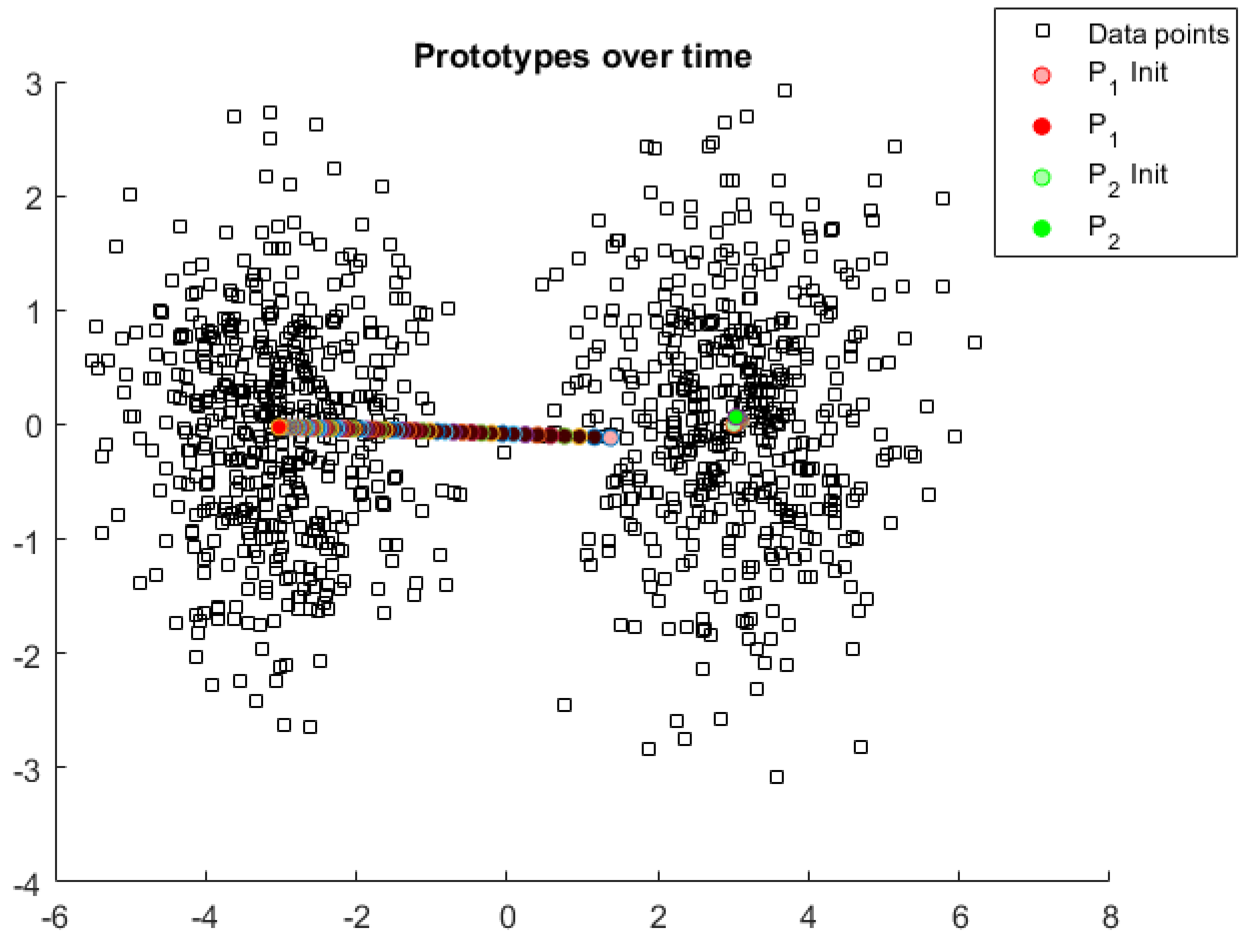

Figure 3: Prototypes over time with K=2 and η=0.0001

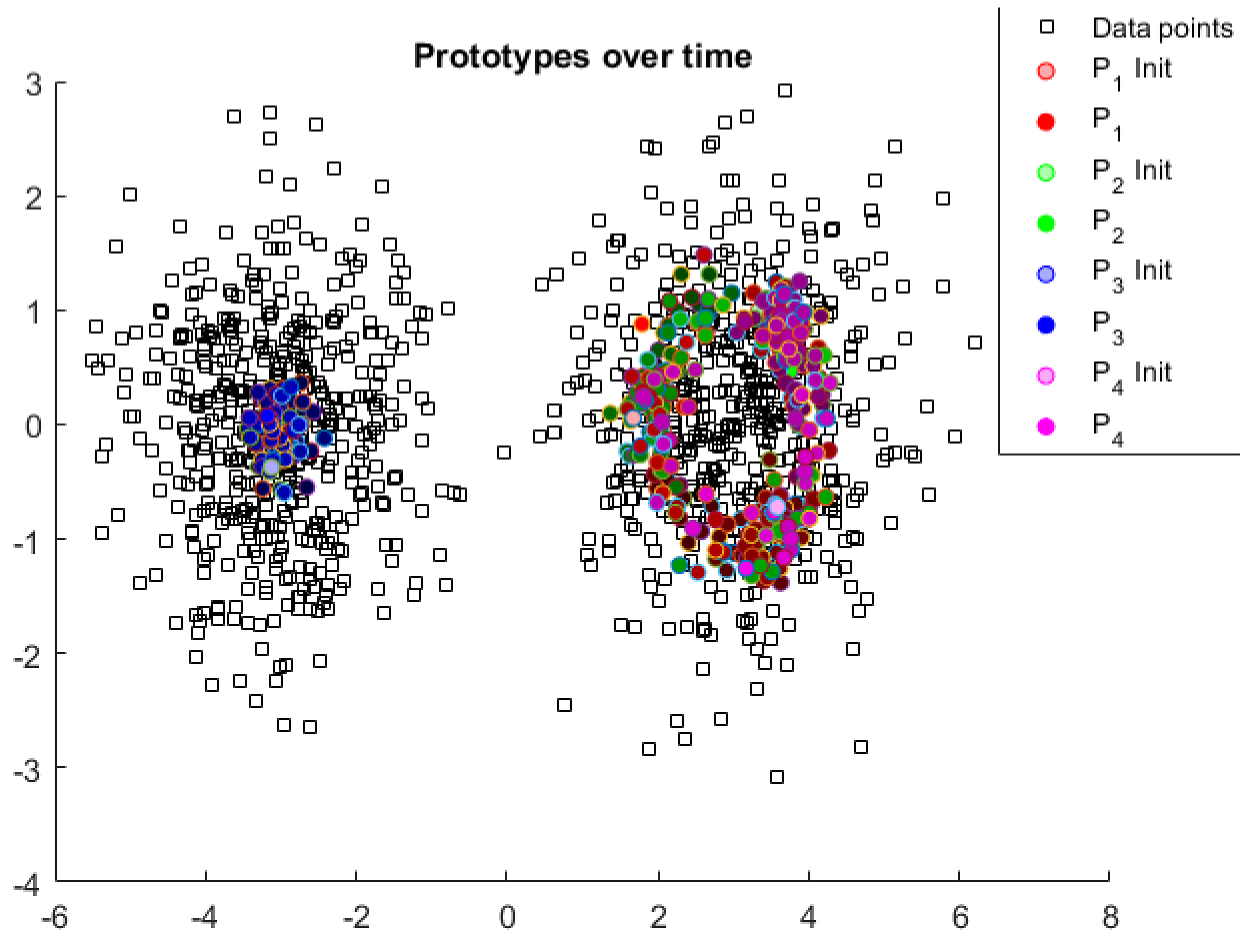

Figure 4: Prototypes over time with K=4 and η=0.1

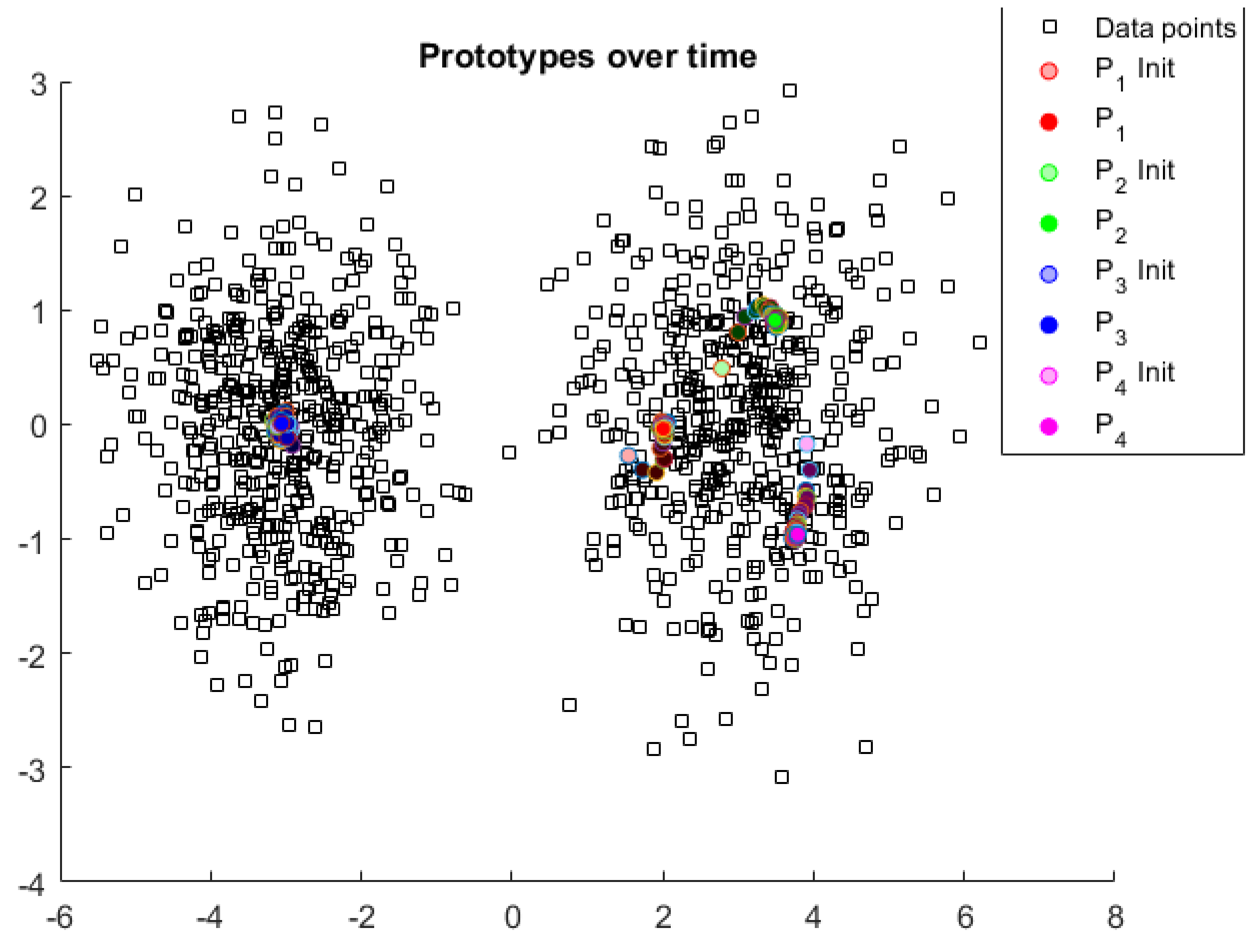

Figure 5: Prototypes over time with K=4 and η=0.01

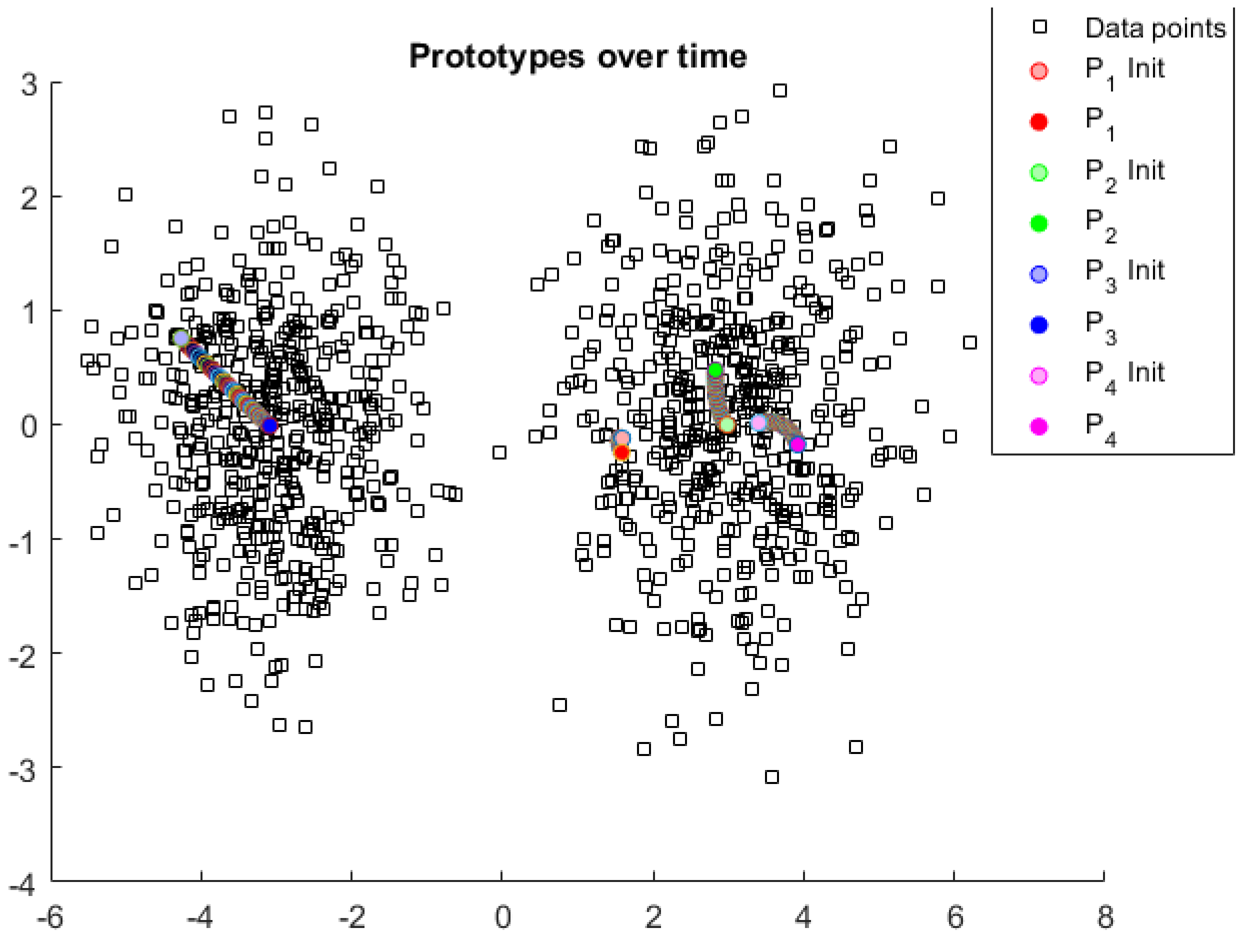

Figure 6: Prototypes over time with K=4 and η=0.0001

6.1.2 Quantisation error over time

The quantisation error has been normalised, such that it has no relation to the size of the dataset. This way we will be able to compare data sets of differing sizes, should we be so inclined.

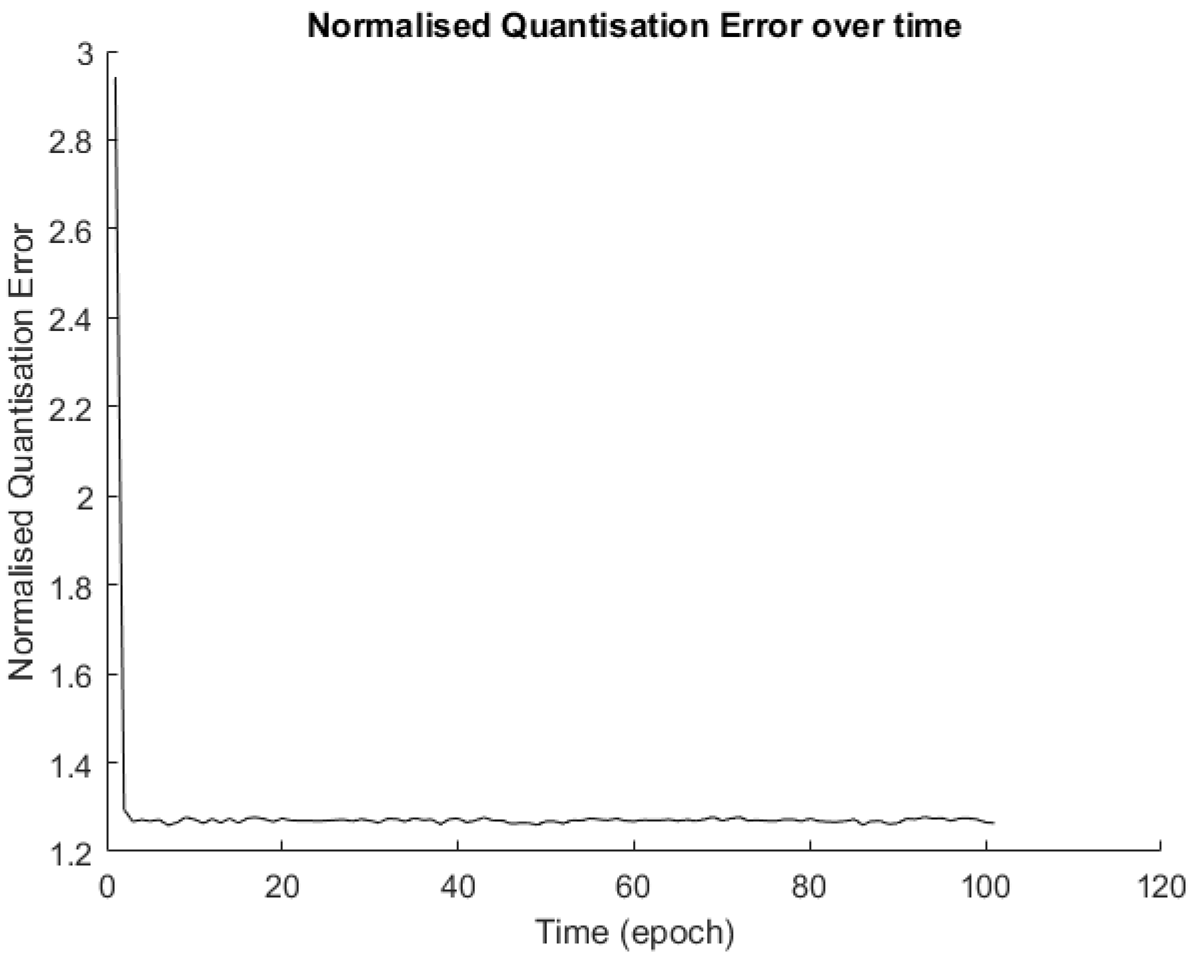

Figure 7: Normalised quantisation error over time with K=2 and η=0.1

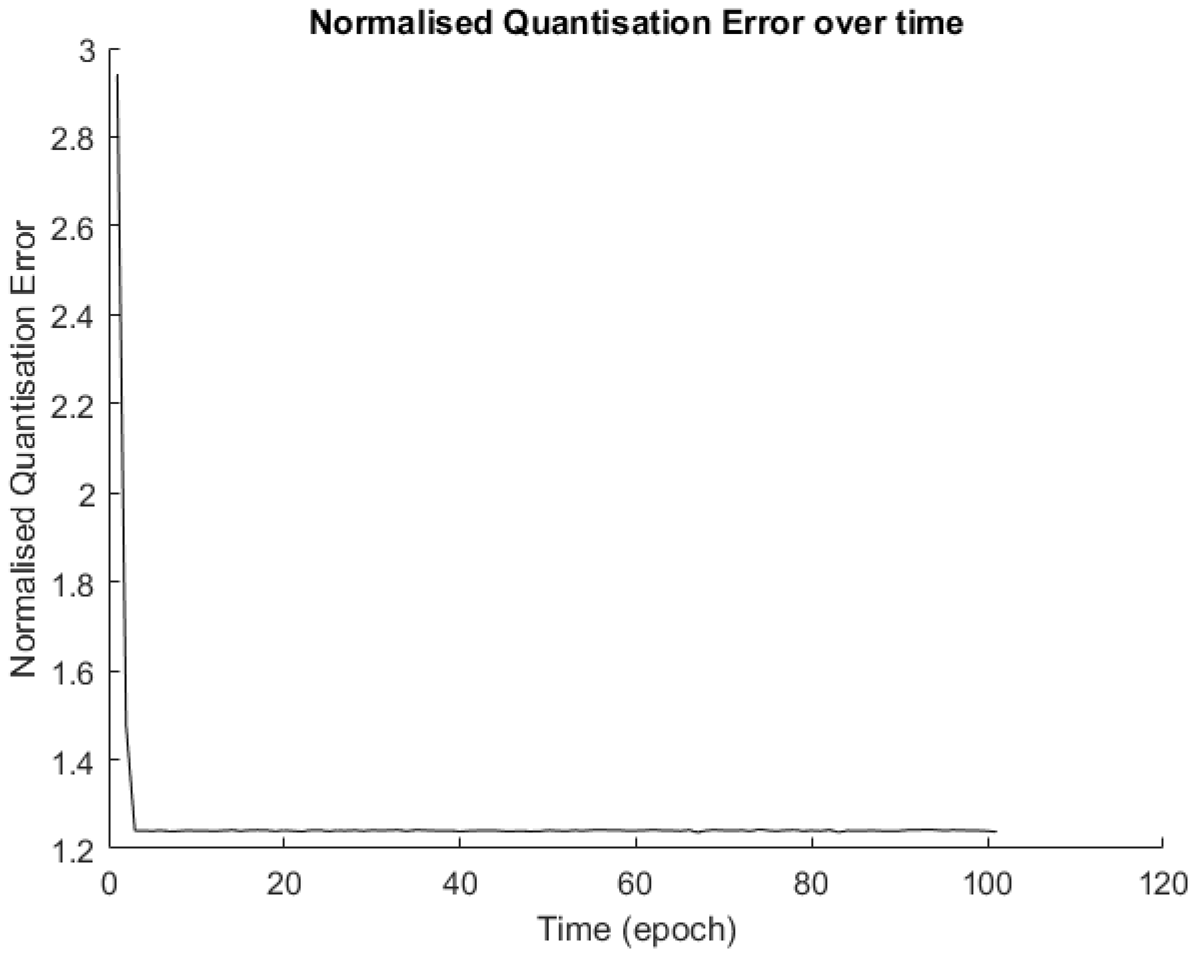

Figure 8: Normalised quantisation error over time with K=2 and η=0.01

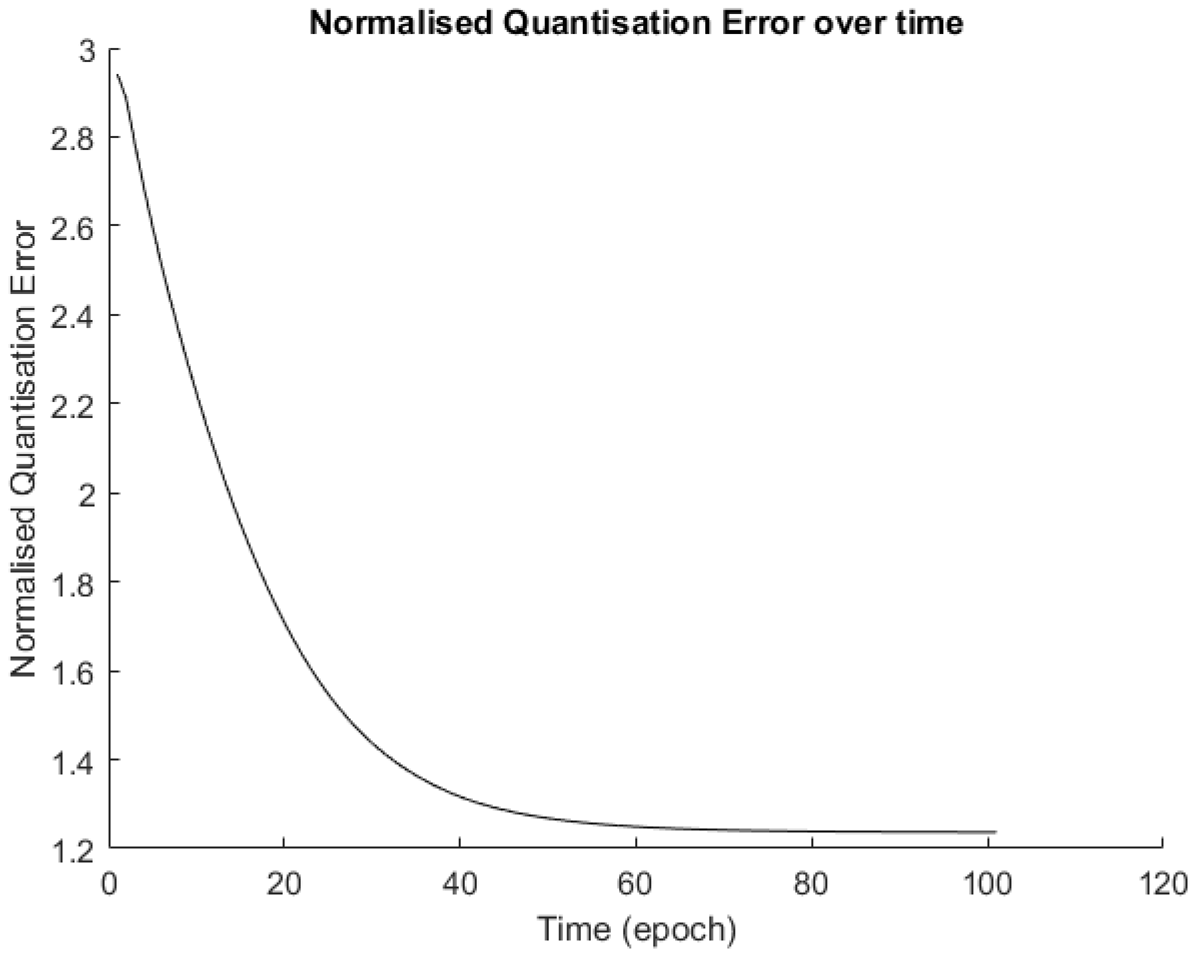

Figure 9: Normalised quantisation error over time with K=2 and η=0.0001

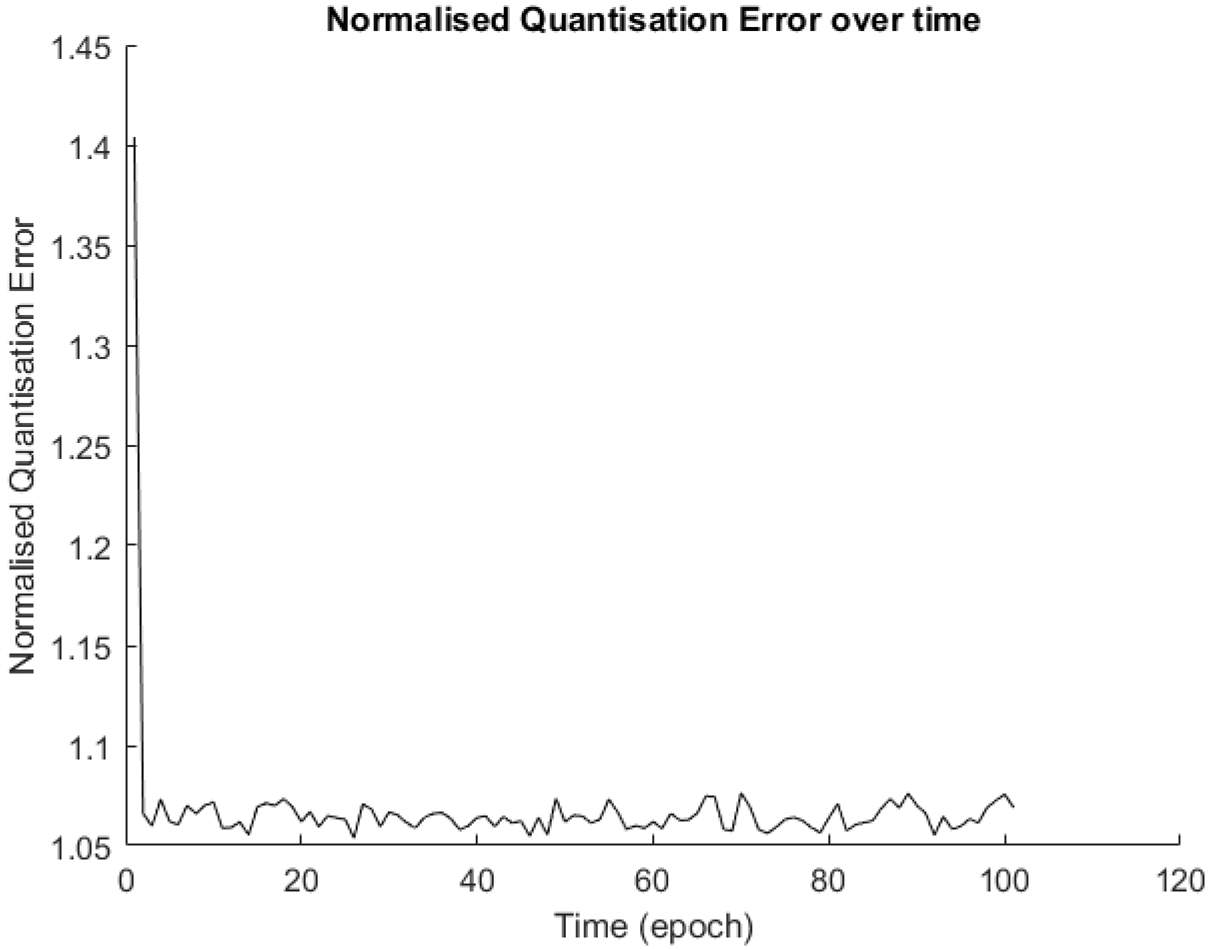

Figure 10: Normalised quantisation error over time with K=4 and η=0.1

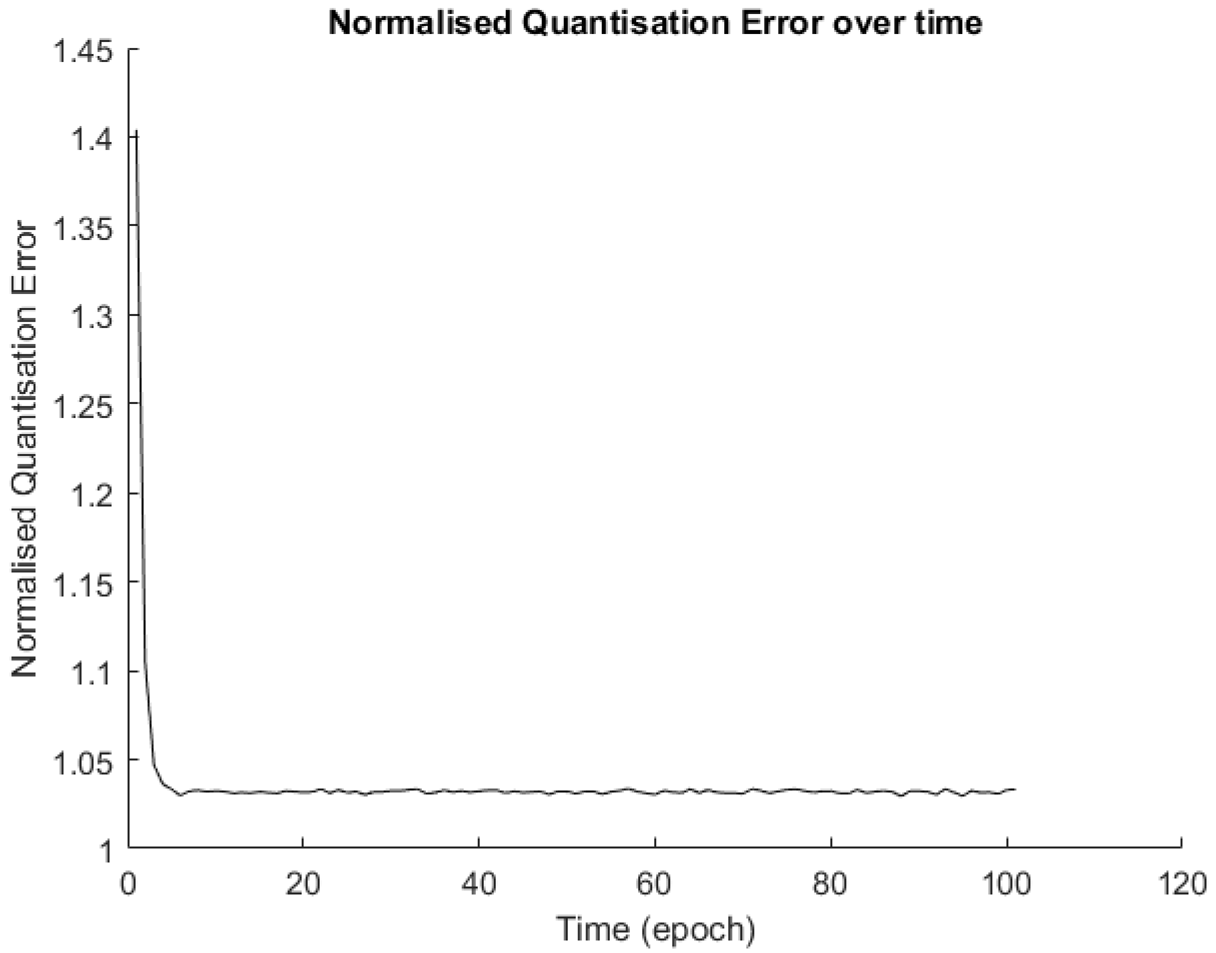

Figure 11: Normalised quantisation error over time with K=4 and η=0.01

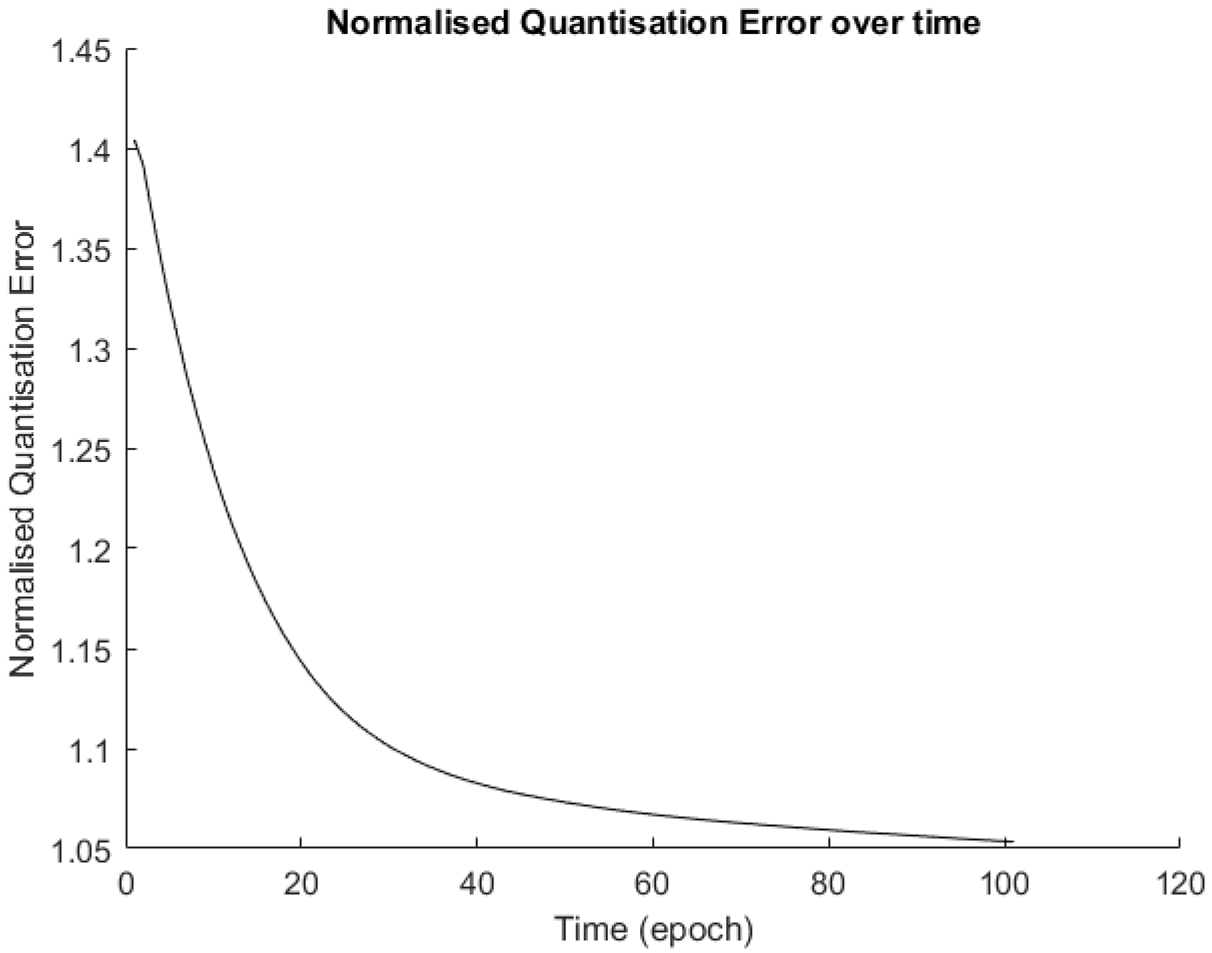

Figure 12: Normalised quantisation error over time with K=4 and η=0.0001

6.2 Code

```
clear all;
close all;

% load data
data_x = load('w6_1x.mat');
data_x = data_x.w6_1x;
data_y = load('w6_1y.mat');
data_y = data_y.w6_1y;
data_z = load('w6_1z.mat');
data_z = data_z.w6_1z;

data = data_x; % replace with dataset of your choice.

% Initialise RNG
rng('default');

% set colour rgb values
red = [1 0 0];
green = [0 1 0];
blue = [0 0 1];
magenta = [1 0 .93];
white = [1 1 1];
colour_init_1 = (red + white*2)./3;
colour_init_2 = (green + white*2)./3;
colour_init_3 = (blue + white*2)./3;
colour_init_4 = (magenta + white*2)./3;

% Initialise variables.
[P, N] = size(data);
K = 4;
learning_rate = 0.1;
t_max = 100;

prototypes_over_time = zeros(t_max, K, N);
H_VQ_over_time = zeros(t_max+1, 1);

% Initialise prototypes
prototypes = zeros(K, N);
for i = 1:K
    prototypes(i, :) = data(randi(P), :);
end

% Compute initial H_VQ
for j = randperm(P)
```

```
      example = data(j, :);

      winner_distance = intmax;
      for k = 1:K
          prototype = prototypes(k, :);
          distance = pdist([example; prototype]);
          if distance < winner_distance
              winner_distance = distance;
          end
      end
      H_VQ_over_time(1) = H_VQ_over_time(1) + winner_distance/P;
end

% Permutations
for t = 1:t_max
  H_VQ = 0; % initialise quantisation error for manual sum.

  %  Shuffle data set and perform permutations
  for j = randperm(P)
    example = data(j, :);

    % initialise winner
    winner = 1;
    winner_distance = intmax; % represents infinity

    % get closest prototype
    for k = 1:K % take rows
      prototype = prototypes(k, :);
      distance = pdist([example; prototype]);
      if distance < winner_distance
        winner_distance = distance;
        winner = k;
      end
    end

    % update winner
    prototypes(winner, :) = prototypes(winner, :) + ...
      learning_rate * (example - prototypes(winner, :));

    % update (partial) sum of quantisation error
    H_VQ = H_VQ + winner_distance;
  end

  % store prototypes and quantisation error for this iteration
  prototypes_over_time(t, :, :) = prototypes;
  H_VQ_over_time(t+1) = H_VQ;

  % normalise H_VQ
  H_VQ_over_time(t+1) = H_VQ_over_time(t+1)/P;
end

% plot prototypes and data over time
%  note that this code is for K = 2 and K = 4
figure; hold on;

% plot data points
data_plot = plot(data(:, 1), data(:, 2), 'ksq');

% plot next iterations with an increasingly coloured line
for t = 2:t_max-1
    plot(prototypes_over_time(t, 1, 1), prototypes_over_time(t, 1, 2), ...
        'o', 'MarkerFaceColor', [t/t_max /2 + .25 0 0]);
    plot(prototypes_over_time(t, 2, 1), prototypes_over_time(t, 2, 2), ...
        'o', 'MarkerFaceColor', [0 t/t_max /2 + .25 0]);
end

% plot first iteration in a whiter form
plot(prototypes_over_time(1, 1, 1), prototypes_over_time(1, 1, 2), ...
    'o', 'MarkerFaceColor', colour_init_1);
plot(prototypes_over_time(1, 2, 1), prototypes_over_time(1, 2, 2), ...
    'o', 'MarkerFaceColor', colour_init_2);

% plot final iteration with fully coloured line
plot(prototypes_over_time(t_max, 1, 1), ...
    prototypes_over_time(t_max, 1, 2), 'o', 'MarkerFaceColor', red);
plot(prototypes_over_time(t_max, 2, 1), ...
    prototypes_over_time(t_max, 2, 2), 'o', 'MarkerFaceColor', green);

% hacky way to add extra plots for when K=4
if K == 4
  for t = 2:t_max-1
    plot(prototypes_over_time(t, 3, 1), prototypes_over_time(t, 3, 2), ...
        'o', 'MarkerFaceColor', [0 0 t/t_max /2 + .25]);
    plot(prototypes_over_time(t, 4, 1), prototypes_over_time(t, 4, 2), ...
        'o', 'MarkerFaceColor', [t/t_max/2+.38 0 t/t_max/2+.33]);
  end

  plot(prototypes_over_time(1, 3, 1), prototypes_over_time(1, 3, 2), ...
      'o', 'MarkerFaceColor', colour_init_3);
  plot(prototypes_over_time(1, 4, 1), prototypes_over_time(1, 4, 2), ...
      'o', 'MarkerFaceColor', colour_init_4);

  plot(prototypes_over_time(t_max, 3, 1), ...
      prototypes_over_time(t_max, 3, 2), 'o', 'MarkerFaceColor', blue);
  plot(prototypes_over_time(t_max, 4, 1), ...
      prototypes_over_time(t_max, 4, 2), 'o', 'MarkerFaceColor', magenta);
end

% title and legend
title('Prototypes over time');
legend_plots = zeros(2*K+1, 1);
legend_plots(1) = data_plot;
legend_plots(2) = plot(nan, nan, 'ro', 'MarkerFaceColor', colour_init_1);
legend_plots(3) = plot(nan, nan, 'ro', 'MarkerFaceColor', red);
legend_plots(4) = plot(nan, nan, 'go', 'MarkerFaceColor', colour_init_2);
legend_plots(5) = plot(nan, nan, 'go', 'MarkerFaceColor', green);
if K == 2
  legend(legend_plots, 'Data points', 'P_1 Init', 'P_1', ...
      'P_2 Init', 'P_2');
else
  legend_plots(6) = plot(nan, nan, 'bo', 'MarkerFaceColor', colour_init_3);
  legend_plots(7) = plot(nan, nan, 'bo', 'MarkerFaceColor', blue);
  legend_plots(8) = plot(nan, nan, 'mo', 'MarkerFaceColor', colour_init_4);
  legend_plots(9) = plot(nan, nan, 'mo', 'MarkerFaceColor', magenta);
  legend(legend_plots, 'Data points', 'P_1 Init', 'P_1', ...
      'P_2 Init', 'P_2', 'P_3 Init', 'P_3', ...
      'P_4 Init', 'P_4');
end

% plot quantisation error over time
figure; hold on;
plot(1:t_max+1, H_VQ_over_time, 'k-');
ylabel('Normalised Quantisation Error');
xlabel('Time (epoch)');
title('Normalised Quantisation Error over time');
```