# Introduction to Machine Learning
# Lab Session 5

Group 39
Laura Lall (s6221858) & Rachel Yang (s6213154)

March 18, 2025

## Introduction

Learning Vector Quantization (LVQ) is a type of Artificial Neural Network, which is based on prototype supervised learning classification algorithm. LVQ applies a Winner-Takes-All based approach. The algorithm works by adjusting prototype vectors to better represent the class distributions in the feature space. In this lab, we implemented the LVQ supervised competitive learning algorithm and applied it to a dataset of 100 two-dimensional feature vectors, where the first 50 data points belonged to class 1 and the remaining data points belonged to class 2. The goal is to evaluate LVQ using one or two prototypes per class and analyze the performance of classification. In this report, we outline the methodology, details of the implementation, and results of our experiments with one and two prototypes per class. We also discuss the impact of the number of prototypes on the error rate. The workload was divided between Rachel implementing most of the code and Laura writing most of the report, with both parties assisting each other where needed.

## Methods

Learning Vector Quantization is efficient for moderate-sized datasets and distinct class separations. LVQ can also handle partially labeled data, making it applicable to many datasets. However, the placement of the initial prototypes can significantly impact performance, since suboptimal initialization can lead to low quality decision boundaries. The algorithm is also susceptible to noise and is biased towards dominant classes.

**Initialization**:
The prototypes are initialized by randomly selecting K data points per class from the dataset.
**Training**:

- For each epoch, the data points are presented in a randomized order.
- For each data point, the Euclidean distance is computed for all prototypes.
- The prototype closest to the data point (the "winner") is updated using the learning rate.
- If the winning prototype belongs to the same class as the data point, the prototype is moved closer.
- Else if the winning prototype does not belong to the same class, the prototype is moved further away.
- This process is repeated for all data points in the dataset, and the training error is computed after each epoch.

**Stopping Condition**:
The training stops after a fixed number of epochs tmax.
**Implementation**:

- The number of prototypes per class was set to K = 2.
- The learning rate $\eta$ was set to 0.002.
- The maximum number of epochs tmax was set to 100

## Experimental results

Learning curves:

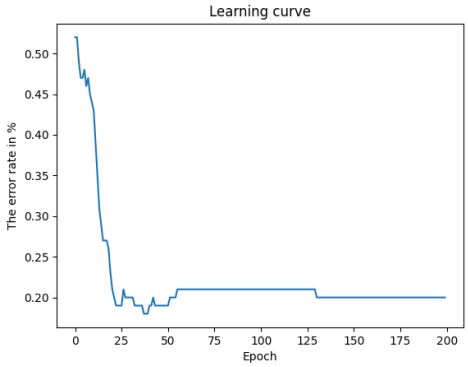


**Figure 1:** *Learning curve for one prototype per class K=1, $\eta$=0.002, tmax=200*

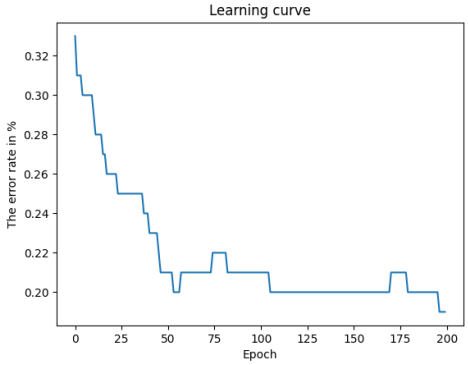


**Figure 2:** *Learning curve for two prototypes per class K=2, $\eta$=0.002, tmax=200*

Trajectories of prototypes:

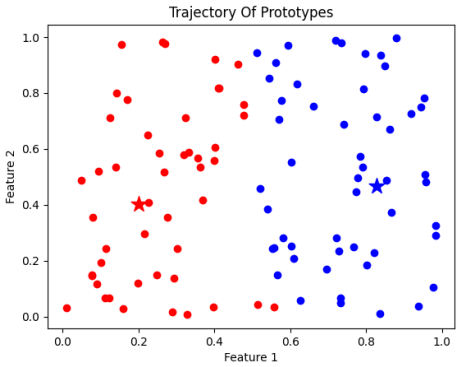


**Figure 3:** *Trajectory for one prototype per class K=1, η=0.002, tmax=200*

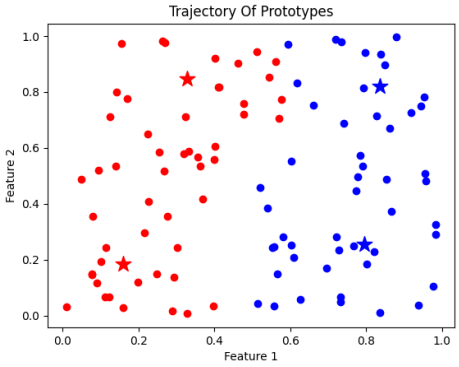


**Figure 4:** *Trajectory for two prototypes per class K=2, η=0.002, tmax=200*

## Discussion

**Learning curves**: For both one prototype per class and two prototypes per class the error rate drops quickly, indicating that the LVQ algorithm learns the classification boundary quickly. For K=1, the error rate starts at a high rate of over 50%, but mostly flattens out around the 60th epoch, stabilizing around the 20% error rate. For K=2, the error rate starts at a rate of over 32%, but mostly flattens out around the 100th epoch, stabilizing around the 20% error rate. The lower initial error rate for K=2 indicates that two prototypes per class provide a more ideal starting point for classification.

**Trajectories of prototypes**: For K=1, the single prototype tends to settle near the center of the class distribution, which may not always yield optimal results, especially for datasets with complex boundaries.

For K=2, the prototypes are more spread out, leading to a more accurate decision boundary. This is particularly evident in the trajectories, where the two prototypes per class cover a wider area of the feature space, improving the classification accuracy.

In conclusion, using two prototypes per class results in slightly more accurate classification compared to using a single prototype. The additional prototype allows for a more flexible decision

boundary, which is better suited to capturing the underlying structure of the data. However, the choice of the number of prototypes should be balanced against computational cost, especially for larger datasets.