Clustering Methods Based on Reinforcement Learning

A Thesis

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Eleni Pachi

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WITH SPECIALIZATION
IN SOFTWARE

University of Ioannina
October 2019

Examining Committee:

- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina (Supervisor)
- **Konstantinos Blekas**, Associate Professor, Department of Computer Science and Engineering, University of Ioannina
- **Konstantinos Vlachos**, Assistant Professor, Department of Computer Science and Engineering, University of Ioannina

DEDICATION

To my family and Nikos.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Prof. Aristidis Likas for the guidance and patience throughout my research. His positive outlook and his brilliant ideas not only made this thesis possible but also helped me develop strong research skills and critical thinking.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

Eleni Pachi
MSc, Computer Science and Engineering, University of Ioannina, Greece
October 2019
Title: Clustering Methods Based on Reinforcement Learning
Supervisor: Aristidis Likas

Clustering is one of the most popular problems in machine learning and data mining. It belongs to the category of unsupervised learning problems since no label information is provided to assist in partitioning the data points into coherent groups. Although clustering is an unsupervised problem, it is possible to view clustering from a reinforcement learning perspective. In reinforcement learning, an agent learns an action policy that solves a sequential decision problem using reinforcement signals provided by the environment.

In reinforcement-based clustering, the clustering system learns through reinforcements to follow the desired clustering policy. The previous method of this type (RGCL algorithm) trains a team of binary stochastic units to perform on-line clustering. Each unit corresponds to a cluster and the weights of each stochastic unit correspond to a representative point (centroid) of the respective cluster. The team of stochastic units is trained to perform clustering using the REINFORCE algorithm by exploiting properly defined reinforcement signals provided by the environment.

In this thesis we propose two extensions of the RGCL algorithm based on the use of a single stochastic multinomial unit instead of a team of binary stochastic units. In the first method the unit is trained on-line based on the REINFORCE framework using immediate reinforcement signals. In the second method the stochastic multinomial unit is trained in a batch mode based on the REINFORCE framework using delayed reinforcement signals. In both cases the

weight update equations are derived so that the weight updates lead to the stochastic minimization of the well-known k-means clustering error.

An experimental study has been conducted using synthetic and real datasets to assess the performance of the proposed methods. The experimental results indicate that improved clustering results are obtained in the majority of cases.

ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Ελένη Παχή
MSc, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
Οκτώβριος 2019
Τίτλος: Μέθοδοι Ομαδοποίησης βασισμένες σε Ενισχυτική Μάθηση
Επιβλέπων: Αριστείδης Λύκας

Η ομαδοποίηση των δεδομένων είναι ένα από τα πιο δημοφιλή προβλήματα της μηχανικής μάθησης καθώς και της εξόρυξης δεδομένων. Ανήκει στην κατηγορία των προβλημάτων μάθησης χωρίς επίβλεψη, αφού η μόνη πληροφορία που παρέχεται για τον διαχωρισμό των δεδομένων σε ομάδες, είναι τα ίδια τα δεδομένα και όχι ετικέτες αυτών. Παρόλο που η ομαδοποίηση είναι πρόβλημα χωρίς επίβλεψη, είναι εφικτό να την προσεγγίσουμε και σαν ένα πρόβλημα ενισχυτικής μάθησης. Στην ενισχυτική μάθηση, το σύστημα μαθαίνει μια στρατηγική η οποία λύνει ένα πρόβλημα διαδοχικών αποφάσεων χρησιμοποιώντας σήματα ενίσχυσης που παρέχονται από το περιβάλλον.

Στην ομαδοποίηση με ενισχυτική μάθηση, το σύστημα μαθαίνει μέσα από σήματα ενίσχυσης να ακολουθήσει την επιθυμητή στρατηγική ομαδοποίησης. Σύμφωνα με την ιδέα αυτή μια προηγούμενη μέθοδος (αλγόριθμος RGCL), βασίζεται στην εκπαίδευση ενός συνόλου από στοχαστικές δυαδικές μονάδες και υλοποιεί ένα σειριακό αλγόριθμο ομαδοποίησης. Στη μέθοδο αυτή κάθε στοχαστική μονάδα αντιστοιχεί σε μια ομάδα και οι παράμετροι της στοχαστικής μονάδας αντιστοιχούν στον αντιπρόσωπο της ομάδας. Το σύνολο των στοχαστικών μονάδων εκπαιδεύεται με τη βοήθεια των REINFORCE τεχνικών και με κατάλληλα ορισμένα σήματα ενίσχυσης που στέλνονται έτσι ώστε να υλοποιείται η ομαδοποίηση των δεδομένων.

Στην εργασία αυτή προτείνουμε δυο νέες μεθόδους που επεκτείνουν τον αλγόριθμο RGCL και χρησιμοποιούν μια μόνο στοχαστική πολυωνυμική μονάδα, αντί για ένα σύνολο από στοχαστικές δυαδικές μονάδες. Στην πρώτη μέθοδο η στοχαστική μονάδα εκπαιδεύεται σειριακά, με την εκπαίδευση να βασίζεται στο REINFORCE αλγόριθμο χρησιμοποιώντας

σήματα άμεσης ενίσχυσης. Στη δεύτερη μέθοδο η στοχαστική πολυωνυμική μονάδα εκπαιδεύεται σε ομάδες παραδειγμάτων (batches) και βασίζεται πάλι στο REINFORCE αλγόριθμο, με τη διαφορά ότι λαμβάνουμε υπόψη και παρελθοντικά σήματα ενίσχυσης. Και στις δυο περιπτώσεις παρουσιάζουμε τις εξισώσεις που προκύπτουν για την ενημέρωση των παραμέτρων. Η ενημέρωση των παραμέτρων γίνεται με τέτοιο τρόπο ώστε το γνωστό σφάλμα ομαδοποίησης του k-Means να ελαχιστοποείται στοχαστικά.

Διάφορα πειράματα σε τεχνητά και πραγματικά σύνολα δεδομένων διεξήχθηκαν για να μελετήσουμε την επίδοση των προτεινόμενων μεθόδων, όπου στις περισσότερες περιπτώσεις τα πειραματικά αποτελέσματα έδειξαν ότι οδηγούμαστε σε καλύτερες λύσεις ομαδοποίησης.

CHAPTER 1

INTRODUCTION

1.1 Introduction

As available information and data increase, users seek ways to discover hidden information. Machine learning develops models that learn through examples in order to implement a specific task. There are different types of learning. The most common types are supervised, unsupervised and reinforcement learning. In supervised learning the model ‘learns’ through data and also from correct answers (labels) that are provided. In contrast, in unsupervised learning the model uses only unlabeled data to learn and implement a task.

Additionally, as far as reinforcement learning is concerned, we can claim that this type of learning lies somewhere in the middle of supervised and unsupervised learning. Thus, the model ‘learns’ from the data and through a reinforcement signal provided from the environment. This reinforcement signal provides useful guidance about how the system should operate in future. We should not confuse reinforcement learning with supervised, because reinforcement signal does not contain correct answers as happens in supervised learning, but only an indirect information about the system’s operation.

Supervised, unsupervised and reinforcement learning can be used in order to solve several types of problems, such as classification or clustering of data. The most common unsupervised learning problem is clustering. Suppose the dataset has the form of $X = (x_1, x_2, \dots, x_N)$, with $x_n \in \mathbb{R}^p$ and does not contain any correct answers, i.e. class labels. The goal of clustering is to partition the X dataset in L groups, called clusters, in a such way that data belonging to the same cluster are similar to each other and dissimilar to those in other clusters. Because clustering tries to reveal hidden structures of data without ground truth, it has many applications in various fields such as pattern recognition, image segmentation, medicine, spatial database analysis, finance and other.

It has been shown that is possible to view clustering as a reinforcement learning problem, where the clustering system learns through reinforcements to follow the desired clustering policy. In this thesis, we propose two clustering algorithms that learn stochastically to group data with the help of a reinforcement signal provided from the environment.

The first algorithm trains on-line a single multinomial stochastic unit based on the REINFORCE framework [3] using immediate reinforcement signals. In the second algorithm the unit is trained in a batch mode based on the REINFORCE framework using delayed reinforcement signals. In both cases the weight update equations are derived and is shown that the weights lead to the stochastic minimization of the well-known k-means clustering error.

We have tested the proposed methods in several real and artificial datasets and compared them with other existing clustering algorithms. It can be concluded that our algorithms in many cases achieve a better clustering solution. This mainly happens because of the included stochasticity, which lets the algorithms explore different clustering solutions from the same initialization of the parameters.

In the following subsections we present the basic knowledge which is necessary to introduce and understand our work properly.

1.2 Clustering

Clustering aims at partitioning data into groups (or clusters) with similar properties. The obtained meaningful groups of objects that share common characteristics, reveal the natural structure of the data. Because of this, cluster analysis has many applications to practical problems in biology, medicine, business, finance, etc. Except for understanding the hidden structure of data, clustering can also be used in many cases for data summarization or compression, if we take into consideration that each cluster can be represented by one data point, the cluster representative.

The goal of clustering is to partition a dataset $X = \{x_n\}, i = 1, \dots, N$ in L groups, called clusters, such that the objects inside a cluster are similar to one another and different from the objects in other clusters. So, the greater the similarity within a group and the greater the difference between groups are, the better or more distinct the clustering is. Fig. 1.1 presents a typical clustering paradigm.

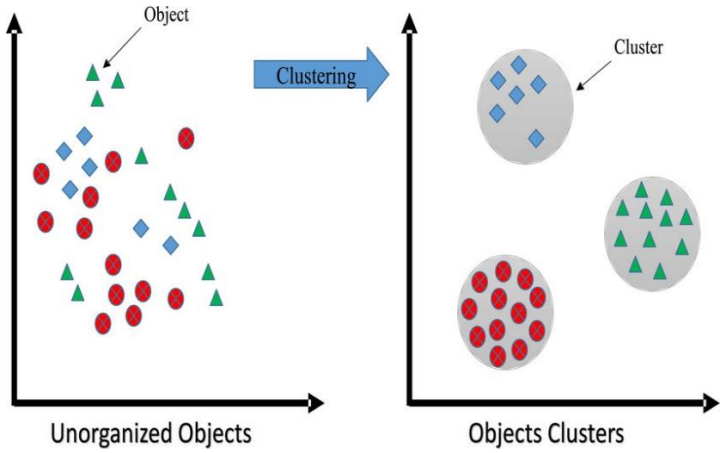


Figure 1.1: A clustering paradigm.

In the definition of clustering, the notion of similarity is important. In order to measure the similarity between objects or clusters, every clustering algorithm needs a proximity measure. We are going to work with algorithms that use distance as a proximity measure and more specifically they use the Euclidean distance. So, for every dataset X containing vectors $x_n \in$

$\mathbb{R}^p$ we consider the distance matrix $d \in \mathbb{R}^{NxN}, d_{nm} = d(x_n, x_m)$, where d_{nm} is the Euclidean distance of x_n from x_m.

There are many types of algorithms that implement clustering. The first big category is hierarchical clustering algorithms. The main idea of these algorithms is that objects are more related to nearby objects than to objects farther away. Thus, in every step we group the closest pairs of groups. Because of this, we can describe hierarchical clustering as a set of nested clusters that can be organized in a tree. Each node of the tree represents a cluster and is the union of its children, that represent the sub clusters. Usually, the leaf nodes are singleton clusters of individual data points and the root is the cluster containing all data points. We can see an example of a dendrogram in Fig. 1.2 below.

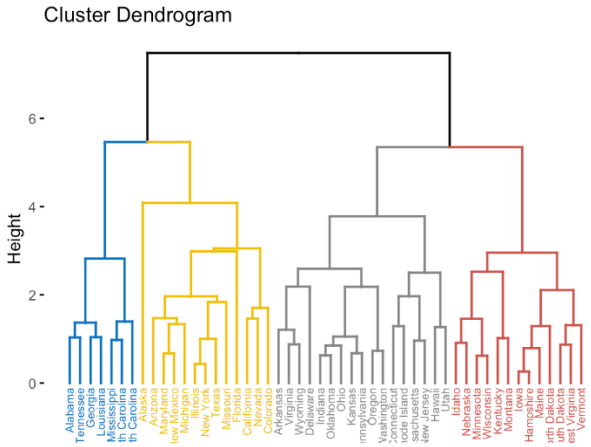


Figure 1.2: A visualization of hierarchical clustering.

A second category is partitional clustering that is simply a division of the set of data points into clusters with no hierarchy. The clusters can be non-overlapping, defining hard clustering, where each object belongs exclusively to one cluster, and overlapping, defining fuzzy clustering which allows objects to belong in different clusters with a degree of membership. We can further divide the partitional clustering algorithms, into density-based algorithms, which define clusters as sets of objects generated by the same distribution and graph-based algorithms, where data are represented as a graph with graph's nodes being the data points and edge links with weights analogous to the similarity among them. Thus, a cluster is defined

as a strongly connected component in graph, i.e. a group of objects that are strongly connected to one another, but have few connections to objects outside the group.

Also, the partitional clustering algorithms can be further divided into prototype-based clustering algorithms that contain parameters, where the clusters are described by a cluster prototype or a representative. This thesis focuses on hard partitional clustering and more specifically on prototype-based algorithms.

1.2.1 Prototype-based Clustering

In this type of clustering algorithms, every cluster is described by a data point that is usually named prototype or representative of the cluster. Thus, in prototype-based clustering a cluster is a set of objects in which each object is more similar to the prototype that describes the cluster than to the prototype of any other cluster. For data with continuous attributes, the prototype of a cluster is often a centroid, i.e. the average of all points in the cluster.

Because a prototype of a cluster is usually the most central point of the cluster, prototype-based algorithms are referred in the literature as center-based algorithms, also. Such clusters tend to be globular. It worth noticing that the most common advantage of prototype-based methods is that they provide an intuitive summarization of the given dataset using a few instances, i.e. the cluster prototypes.

1.2.1.1 k-Means Algorithm

The most popular and oldest prototype-based algorithm is *k-Means*. Because of its simplicity and efficiency, *k-Means* has been used to perform clustering in a large variety of disciplines. k-Means is a prototype-based algorithm, consequently each cluster is represented by a point called centroid or center, which is usually the average of the points of the cluster. This point is not necessarily a point of the dataset. Also, it is a hard partitional clustering algorithm which

tries to split data in L disjoint clusters, in a such way that the distance between a data point and cluster centroid is minimized. In other words, k-Means tries to minimize the variance inside the clusters. In typical k-Means the Euclidean distance is used as a proximity measure.

After defining the parametric clustering model, like in every parametric clustering algorithm, k-Means needs a clustering criterion to optimize, in this case to minimize, with respect to the parameters. The most common clustering criterion used by prototype-based clustering algorithms, is the clustering error. Clustering error is defined as the sum of squared Euclidean distances between each data point x and the cluster centroid w_k of the cluster C_k that x belongs. More specifically, given a set of observations $(x_1, x_2, \dots, x_N)$ where each observation is a p-dimensional real vector, k-Means tries to partition the N observations in L clusters $C = (C_1, C_2, \dots, C_L)$ minimizing the objective function

$$J(w_1, \dots, w_L) = \sum_{i=1}^{L} \sum_{x_n \in C_i} \|x_n - w_i\|^2 \qquad (1.1)$$

where w_i is the centroid of cluster C_i.

k-Means optimizes the above objective function by initializing the L centroids, usually chosen randomly from the dataset, through an iterative procedure. Firstly, the distances between each datapoint and the centroids are computed. The centroid that is closer to the data point is specified. Then, the data point is assigned to its closest centroid cluster and consequently all clusters update their centers. The update of every cluster centroid is equal to the average of all data points that belong to this cluster. This is repeated until convergence, i.e. no data point changes cluster, thus the centroids do not change. We present the k-Means algorithm below.

Algorithm 1.2.1 k-Means

Input: Dataset $X = (x_1, x_2, \dots, x_N)$, number of clusters L, initial centroids $w_1, \dots, w_L$.

Output: Final clusters $C_1, \dots, C_L$, final centroids $w_1, \dots, w_L$.

1. For all points $x_n, n = 1, \dots, N$ do
2. For all clusters $C_i, i = 1, \dots, L$ do

3. Compute the distance $\|x_n - w_i\|^2$
4. Find $c^*(x_n) = \text{argmin}_i(\|x_n - w_i\|^2)$
5. For all clusters $C_i, i = 1, \ldots, L$ do
6. Update cluster $C_i = \{x_n | c^*(x_n) = i\}$
7. For all clusters $C_i, i = 1, \ldots, L$ do
8. Update centroid $w_i = \frac{\sum_{x_n \in C_i} x_n}{|C_i|}$
9. If convergence then
10. Then return final clusters C_i, final centroids w_i.
11. Else
12. Go to step 1.

Fig. 1.3 presents a visualization of k-Means steps. As we see, Fig 1.3(a) presents a set of 2-dimensional data points and Fig 1.3(b) presents the initialization of the centers along with the data. Fig 1.3(c) shows which points are closer to each centroid and Fig 1.3(d) shows the update of the cluster centroids, i.e. the first iteration of k-means. Fig 1.3(e) and Fig 1.3(f) present the second iteration of k-means, where convergence happens.

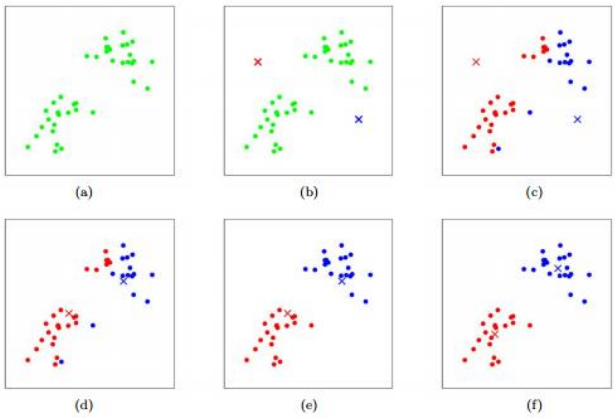


Figure 1.3: A visualization of k-Means algorithm.

Of course, besides Euclidean distance other measures can be used, such as cosine distance or Mahalanobis distance. Note that changes in the proximity measure result in changes in the objective function. Also, there are cases where the centroids are not always the mean of the cluster data.

The computational complexity of k-Means is $O(MN)$, which makes it practical for large datasets. Moreover, it is an easily implemented algorithm. However, it suffers from some limitations. First of all, the solution is highly dependent on the initialization of the centroids. This problem is typically treated by executing k-Means several times, and then store the solutions and select the one with the minimum clustering error. A related important limitation is that k-Means converges to a local minimum of the objective function and therefore it is not a global optimization technique. Moreover, it identifies only linearly separable datasets and it is difficult for k-Means to identify clusters that do not have spherical shape such as datasets with the shape of a ring. Finally, the dataset must be in the form of vectors and the number of clusters is required as input. Despite all the beforementioned limitations, k-Means is widely used because of its simplicity and efficiency.

1.2.2 Competitive Learning for Clustering

Competitive learning is a form of unsupervised learning used commonly in artificial neural networks, in which neurons compete to each other in order to implement desirable task. Competitive learning can also be used for clustering. The basic concept of online competitive learning for clustering is that clusters compete each other in order to cluster data properly. We can model a clustering problem as a competitive learning problem as follows. [4]

Suppose we are given a set $X = (x_1, \dots, x_N)$ of unlabeled data with $x_n = (x_{n1}, \dots, x_{np})^T \in \mathbb{R}^p$ and want to assign each of them to one L clusters. Each cluster is described by a prototype vector $w_i = (w_{i1}, \dots, w_{ip})^T$, $(i = 1, \dots, L)$ and let $W = (w_1, \dots, w_L)$ the matrix of all prototype vectors. Also, let the proximity measure be a distance measure $d(x, w)$. The objective function that choose to minimize in order to find good clusters is the clustering error

$$J(W) = \sum_{n=1}^{N} min_r d(x_n, w_r). \tag{1.2}$$

The clustering strategy in competitive learning techniques can be described as follows. First, randomly select an example x of dataset X, then for each cluster $i = 1, \dots, L$ compute the

distance $d(x, w_i)$ and store the winning cluster i^* where cluster prototype has the minimum distance from x. Next, update the weights w_{i^*} so that the winning cluster prototype moves towards x. This procedure is repeated until a termination criterion is satisfied. It worth noticing that the aforementioned clustering strategy operates in an online mode, because the system updates its parameters immediately after the presentation of a sample. Also, it is a prototype-based clustering strategy, since clusters are represented by prototypes.

1.2.2.1 LVQ algorithm

A well-known competitive learning algorithm is Learning Vector Quantization (LVQ) [4]. LVQ has been mainly used for supervised learning techniques, such as classification but can be applied in unsupervised learning problems, such as clustering, too. Here, we present LVQ for clustering.

Since LVQ is a competitive learning algorithm, it will follow the clustering strategy described in the previous subsection. As far the update of the parameters is concerned, LVQ was designed to update only the winning prototype, moving it towards the pattern x, while leaving the other prototypes unchanged. Thus, the update equation for the winning prototype i^* for a sample x, is the following:

$$\Delta w_{i^*} = a(x - w_{i^*}) \tag{1.3}$$

where a is the learning rate parameter and determines the strength of the update. We present the LVQ algorithm in the table below.

Algorithm 1.2.2 LVQ

Input: Dataset $X = (x_1, x_2, \ldots, x_N)$, number of clusters L, initial cluster prototypes $W = (w_1, \ldots, w_L)$.

Output: Final cluster prototypes $W = (w_1, \ldots, w_L)$.

Specify: Learning rate: a, number of epochs: num_epochs.

1. For all $e = 1$,..., num_epochs do
2. For all data points $x_n, n = 1, \dots, N$ do
3. For all clusters $w_i, i = 1, \dots, L$ do
4. Compute the distance $d(x_n, w_i)$
5. Find the winning prototype i^* such that $i^* = \arg_i \min d(x_n, w_i)$
6. Update only the weights of the winning prototype as $\Delta w_{i^*} = a(x_n - w_{i^*})$.

The LVQ algorithm operates online. Therefore, when a sample x is presented to the system, the update of the parameters occurs exactly after the presentation. Note that LVQ is related to k-Means. Actually, LVQ is the online version of k-Means. Note that k-Means updates the system parameters after all instances have been presented and minimizes the same clustering error. Also, LVQ is a prototype-based clustering algorithm, since clusters are described by prototypes or representatives. We can execute LVQ for a specific number of epochs or until there is no change in parameter values.

1.3 Reinforcement Learning

Reinforcement learning is a machine learning category, where the learning model based on its decisions, receives from the external environment a reinforcement signal that contains indirect information about system's operation. We can consider this operation like the evaluation of human behavior, where we reward someone if he behaves well or penalize him if he does not. Thus, we reward the system if it operates well and implements the desirable task or penalize it if it performs badly.

We can describe the system as a learning agent who receives the reinforcement information and wants to implement an action policy. Since the environment does not provide the correct answers but only sends a reinforcement signal, the agent's actions based on that signal are characterized by stochasticity. Of course, in reinforcement learning, like in every machine learning method, every strategy needs a performance criterion to optimize. Commonly in

reinforcement learning, the objective function that is going to be optimized, contains the reinforcement signal. Thus, the model is trained to take actions that maximize the reinforcement signal, not necessarily immediately but in a long term.

Basically, a reinforcement learning problem can be modeled as a Markov Decision Process (MDP). We have the following.: a set of system states S and a set of actions A taken by the agent. Also, we have a probability of transition $P_a(s, s') = \Pr(s'|s, a)$ from a state s to a state s' under an action a and a reward $r(s, a)$ related to the action a selected from state s. Finally, there is a discount factor $\gamma \in (0,1)$ that quantifies the difference in importance between immediate rewards and future rewards.

As we mentioned before, every reinforcement learning system tries to maximize the reward sent from the environment. Thus, we can define as the objective function, that we want to maximize, the sum of the rewards in a long term, across all future timesteps t, given by the equation

$$R^{(t)} = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} r(s_k, a_k) \tag{1.4}$$

where γ is the discount factor and $r(s_k, a_k)$ is the reward associated with taking the action a at state s at a timestep t.

According to the above MDP modelling, a reinforcement learning agent interacts with the environment at discrete time steps and the reinforcement scheme is described as follows. The agent takes from the environment, that is in a state s_t, a reward r_t and makes an action a_t. This action is sent back to the environment that moves in a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The beforementioned procedure is repeated and the agent changes its action policy in order to optimize the objective function. A visual description is provided in Fig. 1.4 below.

Reinforcement learning due to its generality, has been applied in many disciplines, such as game theory, control theory, object tracking, multi-agent systems etc.

There are many reinforcement learning algorithms that follow different procedures. Also, we can classify the algorithms in two big categories, those with immediate reinforcers and those with the delayed ones. In immediate reinforcement learning, after each action performed by

the agent, the environment sends the reinforcement signal immediately. On the other hand, in delayed reinforcement learning, this signal is sent after several actions have been taken.

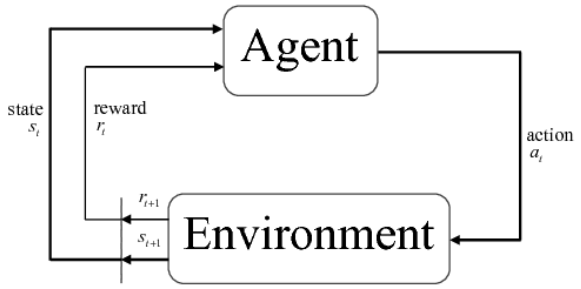


Figure 1.4: The Reinforcement Learning scheme

In this thesis we are going to exploit immediate reinforcement learning algorithms. More specifically our work is based on algorithms of the family of REINFORCE algorithms. In the next subsection, we are going to describe this category of algorithms. Also, in the following chapters we show clustering strategies that based on reinforcement learning, and how REINFROCE algorithms can be adjusted to this concept.

1.3.1 REINFORCE algorithms

In reinforcement learning algorithms [3], the learning agent can be viewed as a feedforward network consisting of several individual units. Because the learning system needs to explore the best decisions, these individual units operate stochastically, so we can call them stochastic units. The network operates by receiving an input from the environment and stochastically propagating the corresponding activity through the net. Then it sends the activity produced from output units to the environment for evaluation. This evaluation is sent through the reinforcement signal r to all units in the net. Then each unit updates its parameters and the cycle begins again.

To describe the above mathematically. For input $x = (x_1, \dots, x_p) \in \mathbb{R}^p$ let denote as y_i the output of ith unit. Each unit has its own parameters $w_i = (w_{i1}, \dots, w_{1p})$ such that $W = (w_1, \dots, w_L)$ denotes all network parameters. Since each unit acts stochastically, the output y_i of a unit is drawn from a distribution, and depends on the input x and unit parameters w_i. Suppose that this distribution is described through its probability mass function. Consequently, for each unit i we define

$$g_i(z, w_i, x) = \Pr\{y_i = z | w_i, x\} \tag{1.5}$$

to be the probability mass function determining the output of the unit as a function of its input and its parameters. It worth noting that all the quantities such as r, y_i, x depend on time, but for convenience, in the following, when they appear in the same equation represent values for the same time instance.

A widely used subclass of stochastic units in connectionist networks are the stochastic logistic or Bernoulli units. In these units, a binary output $y_i \in \{0,1\}$ is drawn from a probability distribution with mass function p_i, which is computed as

$$p_i = f(s_i) \tag{1.6}$$

where f is the logistic function

$$f(s_i) = \frac{1}{1 + e^{-s_i}} \tag{1.7}$$

and

$$s_i = w_i^T x \tag{1.8}$$

is the inner product of w_i and x. In this way, through logistic function we achieve to convert the inner product of w_i and x into a probability that can be used to select the output. In Fig 1.5 below we present the computations of a stochastic logistic unit.

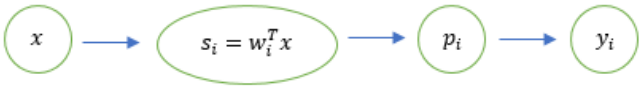


Figure 1.5: A visualization of a stochastic unit computations.

Like in every machine learning method we need an objective function to optimize. A very common objective function that is used in immediate reinforcement learning problems, is the expected value of reinforcement signal r, $E\{r|W\}$ conditioned on the system parameters W. It is necessary to use the expected values because of the randomness of the system. Thus, the learning system searches the space of all possible parameters W for a point where $E\{r|W\}$ is maximum. Also, it worth noting that the $E\{r|W\}$ is well defined because the environment's choice of an input pattern and the reinforcement signal for that input, are determined by stationary distributions. Also, the choice of an input pattern is determined independently of time.

In REINFORCE algorithms the parameters w_{ij} of the learning system, after each step are updated as

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij}) \frac{\partial ln g_i}{\partial w_{ij}} \tag{1.9}$$

where α_{ij} is the learning rate parameter, b_{ij} is the reinforcement baseline and $\frac{\partial ln g_i}{\partial w_{ij}}$ is a quantity called the *characteristic eligibility* of w_{ij}.

An important and interesting result of REINFORCE algorithms is that they make weight adjustments in the direction for which the performance measure $E\{r|W\}$ is increasing. More specifically it can be proved that

$$E\{\Delta w_{ij}|W, x_j\} = \alpha \frac{\partial E\{r|W, x_j\}}{\partial w_{ij}} \tag{1.10}$$

if $\alpha_{ij} = \alpha$ remains constant. Alternatively REINFORCE algorithms claim that the quantity $(r - b_{ij}) \frac{\partial ln g_i}{\partial w_{ij}}$ represents an unbiased estimate of $\frac{\partial E\{r|W\}}{\partial w_{ij}}$. Therefore, they can be used to perform stochastic maximization of the performance measure, because they relate the gradient of the performance measure in the weight space to the average update vector in the weight space, too.

Also, for a Bernoulli unit since it has two possible outcomes it holds that,

$$g_i(z, w_i, x) = \begin{cases} 1 - p_i & \text{if } z = 0 \\ p_i & \text{if } z = 1 \end{cases} \tag{1.11}$$

where the probability p_i will be computed by the Eq. 1.6 presented above.

Therefore, the characteristic eligibility $\frac{\partial ln g_i}{\partial w_{ij}}$ of the update Eq. 1.9 can be further analyzed as

$$\frac{\partial ln g_i}{\partial w_{ij}} = \frac{\partial ln g_i(y_i; p_i)}{\partial p_i} \frac{\partial p_i}{\partial s_i} \frac{\partial s_i}{\partial w_{ij}} \tag{1.12}$$

where because of Eq. 1.11 we have that

$$\frac{\partial ln g_i(y_i; p_i)}{\partial p_i} = \begin{cases} -\frac{1}{1 - p_i} & \text{if } y_i = 0 \\ \frac{1}{p_i} & \text{if } y_i = 1 \end{cases} \tag{1.13}$$

$$\Rightarrow \frac{\partial ln g_i(y_i; p_i)}{\partial p_i} = \frac{y_i - p_i}{p_i(1 - p_i)} \tag{1.14}$$

Also, because of Eq 1.6, Eq. 1.7 and Eq. 1.8 it holds that

$$\frac{\partial p_i}{\partial s_i} = f'(s_i) = p_i(1 - p_i) \tag{1.15}$$

and

$$\frac{\partial s_i}{\partial w_{ij}} = x_j \tag{1.16}$$

Therefore from Eq. 1.14, Eq. 1.15 and Eq. 1.16 the update equation of the parameters in REINFORCE algorithms in the case of Bernoulli units, takes the form

$$\Delta w_{ij} = a_{ij}\big(r - b_{ij}\big)(y_i - p_i)x_j \tag{1.17}$$

where a_{ij} is the learning rate parameter and b_{ij} the reinforcement baseline.

The training of Bernoulli units using REINFORCE algorithms is exploited in the methods presented in next chapter, where we use REINFORCE algorithms to implement a clustering strategy.

1.4 Thesis Roadmap

The structure of this thesis is organized as follows. In Chapter 1, we have introduced our topic providing basic information about the two types of machine learning methods that we use: unsupervised learning, specifically clustering, and reinforcement learning. More specifically, we analyze the clustering and reinforcement techniques, on which our work is based. Chapter 2 presents the closely related work. More specifically, presents how reinforcement learning and especially REINFORCE algorithms can be used to implement a clustering strategy (RGCL algorithm). Next in Chapter 3, we present two new algorithms. An algorithm that performs clustering combining reinforcement learning and operating in on-line mode and an algorithm that makes exactly the same but operating in a batch mode. Then, in Chapter 4 we provide comparative experimental results on synthetic and real datasets and we discuss interesting drawn conclusions as well. Finally, Chapter 5 concludes this thesis by summarizing our findings and also presents interesting directions for future work.

CHAPTER 2

RELATED WORK

2.1 The Reinforcement Clustering Approach

In [1] a method is presented for clustering based on competitive learning and combined with reinforcement learning simultaneously. The main idea is that the clustering system can be viewed as a reinforcement learning system that learns through reinforcement signals to follow the clustering strategy. Actually, the proposed algorithm expands the LVQ algorithm, presented earlier in Chapter 1, by introducing stochasticity to it. Therefore, the closest cluster prototype is not always selected, but it is selected according to some probability distribution. Since the selection is characterized by stochasticity, we need the external environment to evaluate it through a reinforcement signal. This is the point where REINFORCE algorithms are applied in the clustering strategy. To express the reinforcement clustering approach more analytically, the following have been defined.

Suppose we are given $X = (x_1, \dots, x_N)$ of unlabeled data where $x_n = (x_{n1}, \dots, x_{np})^{\,T} \in \mathbb{R}^p$ is a p- dimensional vector and want to assign them to L clusters. Each cluster is described by a prototype vector $w_i = (w_{i1}, \dots, w_{ip})^T$, $(i = 1, \dots, L)$ and let $W = (w_1, \dots, w_L)$ be all the prototype vectors. Therefore, the algorithm deals with prototype-based clustering. Also, in order to apply clustering, a proximity measure is needed. Let $d(x, w)$ the distance of data

point x from a cluster's prototype w, to be the proximity measure. Therefore, goal of clustering strategy is to minimize the objective function

$$J(W) = \sum_{n=1}^{N} min_r d(x_n, w_r). \quad (2.1)$$

which is the well-known clustering error.

Having defined the model of the clustering system and the objective function to be optimized we are going to present how it can be trained with REINFROCE algorithms [1]. Basically, REINFORCE algorithms are used for updating the model parameters, where the parameters here are the cluster prototypes.

It is assumed that each cluster i $(i = 1, \dots, L)$ corresponds to a Bernoulli unit, whose parameter vector $w_i = (w_{i1}, \dots, w_{ip})^T$ corresponds to the prototype vector of cluster i. At each step each Bernoulli unit i is fed with a randomly selected sample x and the following procedure is implemented.

Firstly, the distance $s_i = d(x, w_i)$ of sample x from prototype w_i is computed. After that, p_i is computed as follows:

$$p_i = h(s_i) = 2(1 - f(s_i)) \quad (2.2)$$

where $f(s_i)$ is the logistic function (Eq. 1.7) fined previously and providing values in (0,1) and s_i is the aforementioned distance. In this way, a relation between distances and probabilities is achieved. More specifically because of the form of $f(s_i)$, they are inversely proportional quantities. Therefore, the closer the vector of a unit i is to input sample x, the higher the probability the unit to be active, i.e. $y_i = 1$. As we notice, the probabilities p_i provide a measure of the proximity between data and the cluster prototypes. Therefore, if a unit i is active, it is very probable that this unit is closer to the input data point.

In immediate reinforcement learning, after each output y_i is computed, we need the environment to evaluate it by sending a reinforcement signal r_i to each unit i. This evaluation is made in such a way that the units update their parameters so that the desirable clustering strategy is implemented. Thus, using the update equation of REINFORCE algorithms in Eq.

1.9 and assuming that the learning rate is the same during the whole procedure, the update equation for the reinforcement clustering scheme takes the following form

$$\Delta w_{ij} = \alpha\big(r_i - b_{ij}\big)\frac{\partial ln g_i(y_i; p_i)}{\partial p_i}\frac{\partial p_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}} \tag{2.3}$$

For the Bernoulli units, we know that

$$\frac{\partial ln g_i(y_i; p_i)}{\partial p_i} = \frac{y_i - p_i}{p_i(1 - p_i)} \tag{2.4}$$

Also, from Eq. 2.2 we have that

$$\frac{\partial p_i}{\partial s_i} = -p_i f(s_i) \tag{2.5}$$

since the derivative of logistic function is $f'(s_i) = f(s_i)(1 - f(s_i)$.

Thus, from equation Eq. 2.4 and Eq. 2.5 the parameter update equation corresponding to the proposed reinforcement clustering scheme is

$$\Delta w_{ij} = -\alpha\big(r_i - b_{ij}\big)(y_i - p_i)\frac{f(s_i)}{(1 - p_i)}\frac{\partial s_i}{\partial w_{ij}} \tag{2.6}$$

It is assumed that the term $\frac{f(s_i)}{(1-p_i)} = \frac{1}{1-e^{-s_i}}$, is incorporated to the learning rate a. Therefore, the update equation becomes

$$\Delta w_{ij} = -\alpha\big(r_i - b_{ij}\big)(y_i - p_i)\frac{\partial s_i}{\partial w_{ij}}. \tag{2.7}$$

As we mentioned in Chapter 1, REINFORCE algorithms operate towards maximizing the reinforcement signal. Thus, the above parameter update scheme maximizes the objective function

$$R(W) = \sum_{n=1}^{N}\hat{R}(W, x_n) = \sum_{n=1}^{N}\sum_{i=1}^{L} E\{r_i|W, x_n\} \tag{2.8}$$

where $E\{r_i|W, x_n\}$ denotes the expected value of the reinforcement received by cluster unit i when the input pattern is x_n. Consequently, the reinforcement clustering scheme can be employed in the case of problems whose objective function is in the form of $R(W)$ and the

maximization is achieved by performing updates that at each step maximize the term $\hat{R}(W, x_n)$. The latter is valid, because from Eq. 1.10 we have that

$$E\{\Delta w_{ij}|W, x_n\} = \alpha \frac{\partial E\{r_i|W, x_n\}}{\partial w_{ij}}. \tag{2.9}$$

and since the weight w_{ij} affects only the term $E\{r_i|W, x_n\}$ in the definition of $R(W)$, we conclude that

$$E\{\Delta w_{ij}|W, x_n\} = \alpha \frac{\partial \hat{R}(W, x_n)}{\partial w_{ij}}. \tag{2.10}$$

Therefore, the very interesting result is that through the reinforcement clustering approach the objective function R is maximized in the same sense that LVQ minimizes the objective function J (Eq. 2.1). Next, we are going to present the RGCL algorithm as proposed in [1] that is based on the aforementioned reinforcement clustering scheme.

2.2 The RGCL Algorithm

As we mentioned previously, in LVQ algorithm only the winning cluster i^* updates its parameters and is moved towards pattern x, while the parameters of the other clusters remain unchanged. RGCL algorithm expands the LVQ combining it with the reinforcement clustering scheme. Thus, the strategy is described as follows.

For every data point x there is a closest cluster to it. Let us denote this cluster as i^* and mention it as the winning cluster. The strategy that would like the system to learn is similar to LVQ. Thus, only the winning cluster/unit i^* will update its parameters, while the other units/clusters remain the same. Note that the closest unit may not be active necessarily since its output is computed stochastically depending on the Bernoulli distribution. For this reason, the environment needs to evaluate the unit's decision y_i. Thus, identifies the winning unit i^* and returns a reward signal $r_{i^*} = 1$, if it has decided correctly, i.e. $y_{i^*} = 1$ or a penalty signal $r_{i^*} = -1$, if its decision is wrong, i.e. $y_{i^*} = 0$. The reinforcements sent to the other units, are $r_i = 0$, so their weights are not affected. Therefore, the reinforcement signal is defined as

$$r_i = \begin{cases} 1 & \text{if } i = i^* \text{ and } y_i = 1 \\ -1 & \text{if } i = i^* \text{ and } y_i = 0 \\ 0 & \text{if } i \neq i^* \end{cases} \tag{2.11}$$

Setting the baseline $b_{ij} = 0$, the update equation Eq. 2.7 takes the form

$$\Delta w_{ij} = -\alpha r_i (y_i - p_i) \frac{\partial s_i}{\partial w_{ij}} \tag{2.12}$$

Also, if $s_i = d(x, w_i)$ the distance of input pattern x from the cluster's prototype w_i is the Euclidean distance

$$s_i = \sum_{j=1}^{p} (x_j - w_{ij})^2 \tag{2.13}$$

the term $\frac{\partial s_i}{\partial w_{ij}}$ in the update equation Eq. 2.12 becomes

$$\frac{\partial s_i}{\partial w_{ij}} = -(x_j - w_{ij}) \tag{2.14}$$

and therefore, the update equation of the parameters of the reinforcement clustering scheme takes the form of

$$\Delta w_{ij} = \alpha r_i (y_i - p_i)(x_j - w_{ij}) \tag{2.15}$$

Thus, RGCL algorithm has the following steps presented below. It worth noticing again that the parameter α remains fixed at a specific small value and the reinforcement baseline is not used.

Algorithm 2.2.1 RGCL

Input: Dataset $X = (x_1, x_2, \dots, x_N)$, number of clusters L, initial cluster prototypes $w_1, \dots, w_L$.
Output: Final clusters $C_1, \dots, C_L$, final cluster prototypes $w_1, \dots, w_L$.
Specify: Learning rate a, number of epochs: num_epochs

1. For all $e = 1$,...,num_epochs do

2. For every $x_n, n = 1, \dots, N$ do
3. For every w_i, $i = 1,2, \dots, L$ do
4. Compute the distance s_i using Eq. 2.13.
5. Compute the probability p_i using Eq. 2.2 and decide the output y_i of unit i.
6. Determine the winning unit i^* with $p_{i^*} = \max(p_i)$.
7. Compute the reinforcements $r_i, (i = 1,2, \dots, L)$ using Eq. 2.11.
8. Update the cluster prototypes $w_i, (i = 1,2, \dots, L)$ using Eq. 2.15.

According to the specification of the rewarding strategy, high values of r are received when the system follows the clustering strategy, while low values are obtained when the system fails in this task. Therefore, the maximization of the expected value of r means that the system follows the clustering strategy. Since the clustering strategy aims at minimizing the objective function J, RGCL algorithm achieves an indirect way to minimize J, through the maximization of the immediate reinforcement signal r. This intuition is made more clear in the following proof presented in [1].

The reinforcements in RGCL algorithm are provided by Eq. 2.11, where it holds that: a) $r_i = 1$ when $y_i = 1$ with probability p_i and $i = i^*$ and b) $r_i = 0$ when $y_i = 0$ with probability $1 - p_i$ and $i = i^*$. In any other case $r_i = 0$. Therefore, from equation

$$R(W) = \sum_{n=1}^{N} \sum_{i=1}^{L} E\{r_i | W, x_n\} \tag{2.16}$$

it is derived that the objective function maximized by RGCL for a cluster i, taking into consideration the Eq. 2.11, is

$$R(W) = \sum_{n=1}^{N} [p_{i^*}(x_n) - (1 - p_{i^*}(x_n))] \tag{2.17}$$

where $p_{i^*}(x_n)$ is the maximum probability for input x_n. This leads to

$$R(W) = 2 \sum_{n=1}^{N} p_{i^*}(x_n) - N \tag{2.18}$$

and since N is a constant and the probability p_i is inversely proportional to the distance, we conclude that the RGCL performs updates that minimize the objective function J, since it operates toward maximization of the objective function R.

Moreover, if we notice carefully the update equations of LVQ and RGCL algorithm, we can observe that the actual difference lies in the presence of the term $(y_i - p_i)$ in the RGCL update equation. Because this term depends stochastically on the outcome of output y_i, the strength of the parameter updates w_{ij} can be different depending on the output y_i. That makes the RGCL algorithm more efficient from LVQ, because it can escape from shallow local minima of the objective function by introducing a kind of noise through the different outcomes of y_i.

It worth noticing that RGCL is a local optimization clustering procedure that tries to escape from local minima, but it can be trapped to these as well. Also, in order for RGCL to be executed, the specific number of clusters is needed as input.

Having presented the related work and introduced the necessary theory as well, in next Chapter we present two new methods that implement clustering based on a reinforcement learning scheme by expanding the RGCL approach.

CHAPTER 3

THE PROPOSED ALGORITHMS

3.1 Multinomial Stochastic (MS) Unit

As we mentioned previously, in REINFORCE algorithms the learning agents use stochastic units that draw their output from some distribution. Also, we defined a subclass of stochastic units, the Bernoulli stochastic units where the RGCL algorithm is basically based on. In this subsection, we are going to introduce another subclass of stochastic units that is going to be used in the proposed algorithms, the *Multinomial Stochastic (MS)* unit [2]. As Bernoulli units are based on the Bernoulli distribution, MS units are based on multinomial distribution.

An MS unit characterized by parameters $s = (s_1, \dots, s_L)$, provides an output or selects an action y among L possible outcomes $\{a_1, \dots, a_L\}$ using a probability vector $p = (p_1, \dots, p_L)$. In other words, having a set of actions $\{a_1, \dots, a_L\}$ that each one is chosen with a probability $p = (p_1, \dots, p_L)$, an MS unit selects an action $a_i = y$ with a probability p_i.

The major difference between Bernoulli and MS unit, is that the former is binary and have two possible outcomes, while MS units have L possible outcomes. Therefore, it is obvious that an MS unit is an extension of a Bernoulli unit. Fig. 3.1 below provides a visualization of the operation of an MS unit and actually describes how an MS unit selects an output.

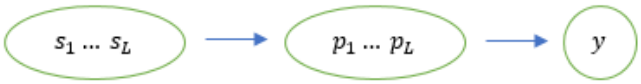


Figure 3.1: A visualization of an MS unit.

During the selection of an output y, it is necessary the probability vector $p = (p_1, \dots, p_L)$ to be computed based on the values of s_i. Thus, using the parameter vector s, we choose to compute the probability vector $p = (p_1, \dots, p_L)$ from the equation

$$p_i = \frac{\exp(-s_i/T)}{\sum_{j=1}^{L} \exp(-s_j/T)} \tag{3.1}$$

where T is a constant. The SoftMin function in Eq 3.1, is used to transform the parameter vector s to a probability vector p. The constant T controls the normalization. If $T \to 0$ we get the min operation, i.e. $p_k = 1$ for $s_k = \min(s_1, \dots, s_L)$ and $p_j = 0$ for $j \neq k$. If $T \to \infty$ all p_i are equal to $\frac{1}{L}$.

3.1.1 REINFORCE algorithms and MS units

Having defined the MS unit and the computation of the probability vector p, we present how the MS unit is trained using the REINFORCE framework.

Let us denote as k the action that the MS unit selects and provides an output y. Obviously, this action has been selected with a probability p_k and it is characterized by parameters s_k. After the selection the environment should send a reinforcement signal r, in order to evaluate the selection. The corresponding update equation of unit parameters that depends on the selection of the MS unit, will be given by the equation

$$\Delta s_i = \alpha_i (r - b_i) \frac{\partial lng}{\partial s_i} \tag{3.2}$$

where $g(y,s) = \Pr\{y|s\}$ is the probability to select the action k and provide the output y having the parameter vector s. Because of the Eq 3.1, every $\frac{\partial lng}{\partial s_i}$ term depends on s_i and on the rest of $s_j, j = 1, \dots, L-1$ terms. Thus, it holds that

$$\frac{\partial lng}{\partial s_i} = \sum_{j=1}^{L} \frac{\partial lng}{\partial p_j} \frac{\partial p_j}{\partial s_i} \tag{3.3}$$

Since we select the action k with a probability p_k, for the first terms of the sum of Eq. 3.3 it holds that

$$\frac{\partial lng}{\partial p_i} = \begin{cases} \frac{1}{p_i} & \text{if } i = k \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Because of Eq. 3.4, $\frac{\partial lng}{\partial p_i} = 0$ for $i \neq k$, thus the terms of Eq. 3.3 where $j \neq k$ will equal to zero. Therefore, the only term that is needed to be calculated is the $\frac{\partial p_k}{\partial s_i}$. Because k is the selected action, from Eq. 3.1 it holds that, if $k = i$

$$\frac{\partial p_k}{\partial s_k} = \frac{-\frac{1}{T}\exp\left({}^{-s_k}/_{T}\right)\sum_{j=1}^{L}\exp\left({}^{-s_j}/_{T}\right) + \frac{1}{T}\exp\left({}^{-s_k}/_{T}\right)\exp\left({}^{-s_k}/_{T}\right)}{\sum_{j=1}^{L}\exp\left({}^{-s_j}/_{T}\right)^2} \tag{3.5}$$

$$\Rightarrow \frac{\partial p_k}{\partial s_k} = -\frac{1}{T}\left[\frac{\exp\left({}^{-s_k}/_{T}\right)}{\sum_{j=1}^{L}\exp\left({}^{-s_j}/_{T}\right)} - \left(\frac{\exp\left({}^{-s_k}/_{T}\right)}{\sum_{j=1}^{L}\exp\left({}^{-s_j}/_{T}\right)}\right)^2\right] \tag{3.6}$$

$$\Rightarrow \frac{\partial p_k}{\partial s_k} = -\frac{1}{T}p_k(1-p_k) \tag{3.7}$$

On the other hand, if $i \neq k$ then from Eq. 3.1 we have that

$$\frac{\partial p_k}{\partial s_i} = \frac{\frac{1}{T}\exp\left({}^{-s_i}/_{T}\right)\exp\left({}^{-s_k}/_{T}\right)}{\sum_{j=1}^{L}\exp\left({}^{-s_j}/_{T}\right)^2} \tag{3.8}$$

$$\Rightarrow \frac{\partial p_k}{\partial s_i} = \frac{1}{T} p_i p_k. \tag{3.9}$$

Thus, combining the Eq. 3.7 and Eq. 3.9 we have that

$$\frac{\partial p_k}{\partial s_i} = \begin{cases} -\frac{1}{T} p_i (1 - p_i) & \text{if } k = i \\ \frac{1}{T} p_i p_k & \text{if } k \neq i \end{cases} \tag{3.10}$$

Because of the Eq. 3.3, Eq. 3.4 and Eq. 3.10, the update equation of system parameters takes the following form

$$\Delta s_i = \begin{cases} -\alpha (r - b_i) \frac{1}{T} (1 - p_i) & \text{if } i = k \\ \alpha (r - b_i) \frac{1}{T} p_i & \text{if } i \neq k \end{cases} \tag{3.11}$$

where α is the learning rate and b_i the reinforcement baseline.

Consequently, Eq. 3.10 presents the update equation of REINFORCE algorithms for training an MS unit. Also, we showed in Chapter 1 that the average update in parameter space S lies in a direction for which the expected value of the reinforcement signal r is increasing. Therefore, using the REINFORCE algorithm to train an MS unit (Eq. 3.11), we achieve stochastic function optimization of the expected reward signal.

3.2 RMS Algorithm

3.2.1 The Reinforcement Clustering Scheme

The related work that was described in Chapter 2, forms the basis of the method presented next. In this subsection we present, how to train the MS unit using the REINFORCE framework to perform clustering. Our goal is to develop an algorithm, that extends LVQ by introducing stochasticity. However, instead of using a team of Bernoulli units, like RGCL does, we aim at using the aforementioned MS unit. Since MS unit is an extension of Bernoulli, we estimate that the new algorithm will be an extension of RGCL and consequently it is expected to perform better.

First of all, the clustering problem is described as follows. We suppose that each cluster i ($i = 1, \dots, L$) is represented by a parameter vector $w_i = (w_{i1}, \dots, w_{ip})^T$ corresponding to the cluster prototype and let $W = (w_1, \dots, w_L)$ be the matrix of all prototype vectors. Thus, the algorithm belongs to the prototype-based clustering category. Let $d(x, w_i)$ the distance metric of a data point x to the cluster prototype w_i. The objective function that we want to minimize is the well -known clustering error J,

$$J(W) = \sum_{n=1}^{N} \min_{\mathrm{r}} d(x_n, w_r). \tag{3.12}$$

We would like to perform clustering through REINFORCE framework and the proposed reinforcement clustering scheme based on MS unit is described in the following. The basic idea is that the actions selected by the MS unit, correspond to the clusters. Through the REINFORCE framework we train the MS unit to select the right clusters in order to cluster data in a proper way.

Thus, at each step, the MS unit is fed with a data point x. For every cluster prototype w_i, the following steps are executed. First of all, we compute the distances $s_i = d(x, w_i)$ of x from each prototype w_i. Thus, we have computed the parameters s_i. In order to select the cluster to which an input x will be assigned, the probability vector $p = (p_1, \dots, p_L)$ is calculated using Eq. 3.1. According to this calculation, a desirable relation between probabilities and distances is achieved: the closest a cluster i to the input is, i.e. the smallest s_i, the higher the probability

p_i this cluster to be selected. Therefore, the probabilities provide a measure of proximity between data and clusters. Next, by selecting using the probability vector $p = (p_1, \dots, p_L)$ an action y is specified, i.e. a cluster is selected for the input example x. In Fig. 3.2 we present a visualization of the aforementioned MS unit computations.

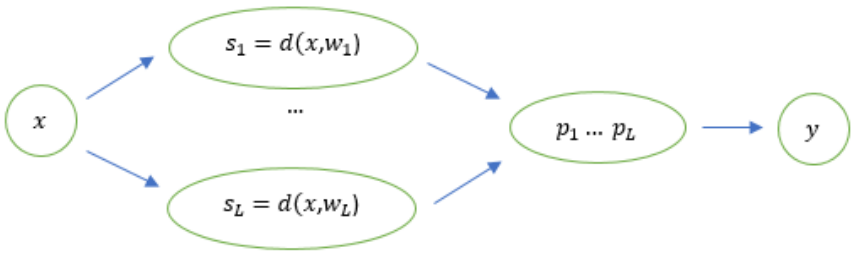


Figure 3.2: Visualization of an MS unit computations.

Having described how a cluster is selected through sampling, we want the environment to evaluate this selection by sending a reinforcement signal r to the system, in order the desirable clustering strategy to be followed by the learning system. As soon as the reinforcement signal is received, the system immediately updates its parameters. This update will be based on the REINFORCE framework.

Based on the REINFORCE update Eq. 3.11, we have that in this case the weight updates will be given by the equation

$$\Delta w_{ij} = \begin{cases} -\alpha(r - b_i)\frac{1}{T}(1 - p_i)\frac{\partial s_i}{\partial w_{ij}} & \text{if } i = k \\ \alpha(r - b_i)\frac{1}{T}p_i\frac{\partial s_i}{\partial w_{ij}} & \text{if } i \neq k \end{cases} \tag{3.13}$$

where α is the learning rate, b_i the reinforcement baseline.

Due to the REINFORCE property the above parameter update equation maximizes the expected value of the reinforcement signal r, because of Eq. 2.10, which indicates that the parameters update lies in the same direction where reinforcements are increasing.

The basic difference between the reinforcement clustering scheme proposed in [1] with this one, is that in RGCL algorithm every cluster corresponds to a Bernoulli unit and consequently we have a team of units equal to the number of clusters. The REINFORCE framework is used

to train this team in order to achieve clustering. On the other hand, in RMS algorithm there is a single MS unit on which the reinforcement framework is applied, which combines the distances to all cluster prototypes to compute the probability vector.

3.2.2 RMS Algorithm

Having introduced the MS unit and the reinforcement framework, we are ready to present our new algorithm. We name it *Reinforcement Multinomial Stochastic – RMS* algorithm.

For an input x we determine a) the winning (the closest) cluster i^* to the input pattern x and b) the selected cluster k for x after sampling of the MS unit. The strategy that we would like the system to learn is to select the closest cluster, i.e. to hold $k = i^*$. In this case the environment rewards the system and sends reinforcement signal $r = 1$. In any other case, the environment penalizes the system sending a reinforcement signal of $r = -1$. Therefore, the proposed reinforcement signal is designed as

$$r = \begin{cases} 1 & \text{if } k = i^* \\ -1 & \text{otherwise.} \end{cases} \tag{3.14}$$

We choose to use the Euclidean distance

$$s_i = \sum_{j=1}^{p} (x_j - w_{ij})^2 \tag{3.15}$$

and consequently, we have that

$$\frac{\partial s_i}{\partial w_{ij}} = -(x_j - w_{ij}). \tag{3.16}$$

Thus, the update equation Eq. 3.13 becomes

$$\Delta w_{ij} = \begin{cases} \alpha r \frac{1}{T}(1 - p_i)(x_j - w_{ij}) & \text{if } i = k \\ -\alpha r \frac{1}{T} p_i (x_j - w_{ij}) & \text{if } i \neq k \end{cases} \tag{3.17}$$

where k is the selected unit, α is the learning rate and r is given by Eq. 3.14. Also, we do not use the baseline and we set it equal to zero, i.e. $b_{ij} = 0$. The steps of the RMS algorithm are described below.

According to the specification of the rewarding strategy, high values of reinforcement signal r are received when the system follows the clustering strategy, i.e. when $i^* = k$, while low values are obtained when the system fails in this task. Therefore, the maximization of the expected value of r means that the system follows the clustering strategy. Since the clustering strategy aims at minimizing the clustering error J, RMS algorithm achieves an indirect way to minimize J, through the maximization of the immediate reinforcement signal r. This intuition is made more clear in the following.

Algorithm 3.2.1 RMS

Input: Dataset $X = (x_1, x_2, \dots, x_N)$, number of clusters L, initial cluster prototypes $w_1, \dots, w_L$.
Output: Final clusters $C_1, \dots, C_L$, final cluster prototypes $w_1, \dots, w_L$.
Specify: Learning rate a, number of epochs: num_epochs, parameter T.

1. For all $e = 1$,..., num_epochs do
2. For $x_n \in X, n = 1, \dots, N$ do
3. For $w_i \in W, i = 1,2, \dots, L$ do
4. Compute the distance s_i using Eq. 3.15.
5. Compute the probability p_i using Eq. 3.1.
6. Decide the selected cluster k.
7. Specify the winning cluster i^* with $p_{i^*} = \max(p_i)$.
8. Compute the reinforcements r using Eq. 3.14.
9. Update the parameter vectors $w_i, (i = 1,2, \dots, L)$ using Eq. 3.17.

As we presented, the reinforcements in RMS algorithm are provided by Eq. 3.15. Using this and considering that we select a cluster i with probability p_i, from equation

$$R(W) = \sum_{n=1}^{N} \sum_{i=1}^{L} E\{r_i | W, x_n\} \tag{3.18}$$

it holds that

$$R(W) = \sum_{n=1}^{N} p_{i^*}(x_n) - \left(1 - p_{i^*}(x_n)\right) \tag{3.19}$$

where for every x_n, the cluster i either is the closest one ($i = i^*$) and we select it with probability $p_{i^*}(x_n)$ taking reinforcement $r = 1$, or we make wrong and do not select it with probability $1 - p_{i^*}(x_n)$ taking a reinforcement $r = -1$. Thus, Eq. 3.19 becomes

$$R\,(W) = 2 \sum_{n=1}^{N} p_{i^*}(x_n) - N \tag{3.20}$$

where N is a constant. Since the probability p_{i^*} is inversely proportional to the distance, we conclude that the RMS performs updates that minimize the objective function J, since it operates toward maximization of the objective function R.

It worth to notice that RMS is a local clustering procedure that tries to escape from local minima by introducing stochasticity to learning but it can be trapped in local minima as well. Also, the execution of RMS needs the specific number of clusters as input. Finally, the RMS algorithm operates in an online mode meaning that the update of the system parameters is done as soon as an input x is presented to the system.

3.3 Batch-RMS Algorithm

As mentioned before, the RMS algorithm operates in an online mode and updates the system parameters as soon as an input x is presented to the system. We have developed an adaptation of RMS in order to operate in a batch mode as well. We call this algorithm as the *batch-RMS* algorithm.

When operating in batch mode, we gather update information from all input patterns and then we use this cumulative information to update the parameters of the system. In this case, we do not update the parameters as soon as a pattern is presented to the system, but after the presentation of all patterns. The batch-RMS algorithm is presented in the next subsections.

3.3.1 The Reinforcement Clustering Scheme

Because batch-RMS is derived from online RMS, we use basically the same modelling of the clustering problem and the same objective function to minimize, i.e. the clustering error. Thus, suppose that $X = (x_1, \dots, x_N)$ is the matrix that describes the whole dataset, where $x_n = (x_{n1}, \dots, x_{np}) \in \mathbb{R}^p$ is a p-dimensional vector. Also, each cluster i ($i = 1, \dots, L$) is described by a parameter vector $w_i = (w_{i1}, \dots, w_{ip})^T$ which corresponds to the prototype vector of cluster i. Let $W = (w_1, \dots, w_L)$ be the matrix of all prototype vectors. Again, the Euclidean distance $d(x, w)$ of a data point x from a cluster prototype w is selected as the proximity measure and the objective function that we want to minimize is the well-known clustering error $J(W)$ (Eq. 3.12).

Also, we use the same reinforcement clustering scheme as with RMS algorithm, where we use an MS unit to select a cluster for each input x and we use the same update for the parameters, the one presented in Eq. 3.17.

Although, we based on online RMS, in batch-RMS there are some differences. First of all, because the update of system's parameters is done after all input patterns x_n have been presented, the MS unit shares the same parameters w_i for every pattern x_n. Thus, in the Fig. 3.3 below is presented the computations of MS unit in a single epoch of batch-RMS.

Also, another important difference lies on the design of reinforcement signal. In online mode the environment evaluates the decision for an input pattern, and sends a reinforcement signal according to the decision. Because now we operate in batches, the system needs to store information on the decisions for all input patterns and use it properly when all patterns have been presented to it. The reinforcement signal is provided after all inputs have been presented

to the system. Since we have the cluster assignments for all input patterns it is natural to consider the clustering error as the basis for the reinforcement signal r,

$$E = \sum_{n=1}^{N} d(x_n, w_k) \tag{3.21}$$

where w_k is the parameter vector of the selected cluster k for the pattern x_n and $d(x_n, w_k)$ is the Euclidean distance of x_n from w_k.

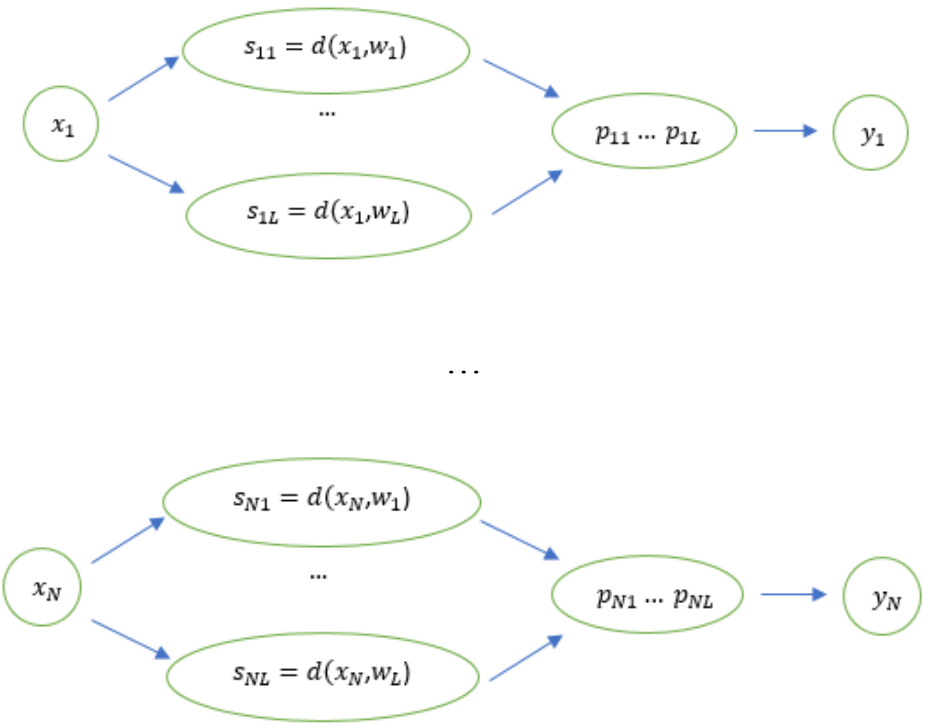


Figure 3.3: Visualization of MS unit computations in batch-RMS.

This function basically is the sum of Euclidean distances of patterns from the corresponding selected clusters. Obviously, the clustering strategy is accomplished correctly if the clustering error is minimized. Since the REINFORCE updates lead to maximization of expected reward, the reinforcement signal sent from the environment should be inversely proportional to

$$r = \frac{c}{E} \tag{3.22}$$

where c is a constant. In this way, it is obvious that E and r are inversely proportional amounts.

Having defined the reinforcement signal the update of system parameters will be based on Eq. 3.17. Since, this update equation derived from REINFORCE algorithms, it updates the cluster prototypes towards maximizing the reward r, therefore the clustering error will be minimized and consequently the clustering operation is implemented. In the following we present the details of batch-RMS algorithm.

3.3.2 The Batch-RMS Algorithm

In our REINFORCE algorithms considered so far, we choose the reinforcement baseline equal to zero. As far as this algorithm is concerned, the exploitation of the baseline is very important for the algorithm. More specifically, we compute the baseline as weighted average of past reinforcement signals:

$$\bar{r}\,(t) = \gamma r(t-1) + (1-\gamma)\,\bar{r}\,(t-1) \tag{3.23}$$

where $0 < \gamma \leq 1$ is a constant that actually measures how much importance we put on past values of r.

For an input x_n we choose to update only the prototype of winning cluster i^*, while leaving the other cluster prototypes unchanged. Thus, the update equation Eq. 3.17 taking into account the reinforcement baseline becomes

$$\Delta w_{ij}{}^{(n)} = \begin{cases} \alpha(r - \bar{r})\dfrac{1}{T}(1-p_i)(x_{nj} - w_{ij}) & \text{if } i^* = i \text{ and } i^* = k \\ -\alpha(r - \bar{r})\dfrac{1}{T}p_i(x_{nj} - w_{ij}) & \text{if } i^* = i \text{ and } i^* \neq k \\ 0 & \text{if } i \neq i^* \end{cases} \tag{3.24}$$

Consequently, the algorithm consists of the following steps. First, for every pattern x_n, the winning cluster (i.e. that with the maximum probability) as well as the selected cluster are computed. Then the clustering error and the reinforcement signal are computed (Eq. 3.22). Next, for every pattern x_n we compute the updates of the cluster prototypes $\Delta w_{ij}{}^{(n)}$ from Eq.

3.24. Finally, we compute the total update Δw_i for each cluster prototype w_i as the average of the weight updates $\Delta w_{ij}{}^{(n)}$

$$\Delta w_i = \frac{1}{N_i} \sum_{n=1}^{N_i} \Delta w_i{}^{(n)} \tag{3.25}$$

where N_i is the number of x_n have been assigned to cluster i.

Algorithm 3.3.1 batch-RMS

Input: Dataset $X = (x_1, x_2, \dots, x_N)$, number of clusters L, initial cluster prototypes $w_1, \dots, w_L$.
Output: Final clusters $C_1, \dots, C_L$, final cluster prototypes $w_1, \dots, w_L$.
Specify: Learning rate a, number of epochs: num_epochs, parameter T, parameter γ.

1. For all $e = 1$,..., num_epochs do
2. For $x_n \in X, n = 1, \dots, N$ do
3. For $w_i \in W, i = 1,2, \dots, L$ do
4. Compute the distance s_i using Eq. 3.15.
5. Compute the probability p_i using Eq. 3.1.
6. Decide the selected cluster and store it.
7. Specify the winning cluster i^* with $p_{i^*} = \max(p_i)$ and store it.
8. Compute J using Eq. 3.21 the reinforcement r using Eq. 3.22 and the baseline $\bar{r}$ using Eq. 3.23.
9. For $x_n \in X, n = 1, \dots, N$ do
10. Compute and store the updates of weight vectors $\Delta w_{ij}^{(n)}, i = 1, \dots, L$ using Eq. 3.24.
11. Update the winning cluster prototypes using Eq. 3.25.

Concluding, batch-RMS is based on the same reinforcement clustering approach like RMS does. The main difference is that it operates in batches, while the RMS operates online. Because of this, it is necessary to use a different approach to specify the reinforcement signal that exploits the property that first all patterns are presented to the system and then the update

of parameters is implemented. To this way the reinforcement signal r is directly related to clustering error E (Eq. 3.22). The batch-RMS performs local stochastic optimization of the clustering error. Finally, we assume that the number of the clusters is known and provided as input to the algorithm.

In Chapter 4 that follows, we present comparative experimental results of four algorithms, LVQ, RGCL, RMS and batch-RMS.

CHAPTER 4

EXPERIMENTAL STUDY

In this chapter we present experiments using the two proposed algorithms, which are compared with LVQ and RGCL. We test them in real and artificial datasets. The code has been implemented in Python 3.5.

4.1 Evaluation

The performance of our algorithms is measured on synthetic data, as well as on real ones. The real datasets concern objects images (*Coil-20* dataset) and collections of handwritten digits (*Pendigits* dataset).

As we mentioned in previous chapter, we evaluate the clustering solution through clustering error. The clustering error is defined as the sum of distances of every input from its closest cluster representative

$$J = \sum_{n=1}^{N} \min_r d(x_n, w_r) \tag{4.1}$$

where $X = (x_1, x_2, \dots, x_N)$ is the dataset and $W = (w_1, w_2, \dots, w_L)$ are the cluster representatives of the L clusters. Thus, the lower the clustering error is, the better the performance of the algorithm.

Also, we use *Normalized Mutual Information* (NMI) criterion to evaluate our clustering performance. NMI derives from entropy in information theory. Let $H(X) = -\sum_x p(x) \log p(x)$ be the entropy of a discrete random variable X. The mutual information of two random discrete variables that measures the mutual dependence between them, can be defined as $I(X,Y) = H(X) - H(X|Y)$, where $H(X|Y) = -\sum_y \sum_x p(x,y) \log\, p(x|y)$ is the conditional entropy between X, Y. According to this, the normalized mutual information (NMI) is defined as follows

$$NMI(X,Y) = \frac{2I(X,Y)}{H(X) + H(Y)} \tag{4.2}$$

Nowadays NMI has become the most widely used criterion for evaluation of clustering methods solutions, if the ground truth labels are known. In our case, because the ground truth labels are given for all datasets, we use the criterion to compare them with the predicted ones provided by each algorithm. NMI score ranges between 0.0 and 1.0, with 1.0 indicating perfect cluster labeling. Obviously, higher NMI values indicate a better match between ground truth cluster labels and the predicted ones. It worth noticing that ground truth labels are used only to compute NMI score and we make no use of them during the clustering phase.

We evaluate the performance of clustering algorithms as follows. For each dataset, we conducted 20 experiments for every of the four algorithms: LVQ, RGCL, RMS and batch-RMS. Consequently, we have 20 different initial states selected randomly. In every experiment all the algorithms are executed from the same random initial state. Also, each algorithm, is executed for 200 epochs. One epoch is completed when all data points have been presented to the system. During the execution of each algorithm we keep track the best solution so far. As best solution we define the clustering solution at a specific epoch where the algorithm achieves the lowest clustering error. Having stored this solution after training, we apply the k-Means algorithm once in order to achieve convergence and store the values of clustering error and NMI for evaluation and comparison.

More specifically, for a fair comparison, we calculate for each algorithm the averages of clustering error and NMI. However, average is not always a good measure since it is affected

from extreme values in the sample. Thus, in order to detect if there is difference between average values indeed, we perform a t-test between the clustering errors obtained from the 20 different experiments.

Basically, a t-test tells how significant the differences between groups are, in this case between clustering errors. In other words, it let us know if these differences happened by chance. Difference is measured by a t-score. The t-score is a ratio between the difference between two groups and the difference within the groups. Thus, the larger the t-score, the more difference there is between groups, or the smaller the t-score, the more similarity there is between them. Moreover, every t-value has a p-value to go with. A p-value is the probability that indicates if the results from our sample occurred by chance. P-values are from 0% to 100% and are usually written as a decimal. Low p-values are good because they indicate that data did not occur by chance.

There are different types of t-tests. In our evaluation we use the independent sample t-test that compares the averages of two groups. The null hypothesis is that the two independent samples have identical average values. So, we reject the null hypothesis of equal averages if the p-value is smaller than a threshold, e.g. 1%, 5%, or 10%. In any other case we cannot reject it. We perform the t-test on the clustering errors of the following pairs of algorithms. RMS vs RGCL, RMS vs LVQ, RMS vs batch-RMS, LVQ vs batch-RMS, RGCL vs batch-RMS. We implement the t-test in Python with the help of *Scipy* package.

Finally, besides averages and t-test, we also measure for every dataset the percentage of the 20 experiments for which the performance of every algorithm was superior. For example, we detect which one algorithm achieves the lowest clustering error in the first experiment, which one in the second, etc. Experimental results are presented in the next subsection.

4.2 Experimental Results

4.2.1 Synthetic Data

We created three different synthetic datasets using a procedure that generates a mixture of various cluster structures, and specifically structures such as gaussian, student-t, uniform rectangle and uniform oval. The function accepts as input the desirable size of dataset, the number of clusters and the dimension of the samples of the dataset. The default type of clusters structure is a random combination of the structures mentioned above, but there is an option to choose any structure that we prefer.

The first artificial dataset (*Synthetic1*) is a 2-dimensional dataset of 500 examples. The structure of the clusters is a mixture of 4 gaussian, 1 student-t, 3 uniform rectangle and 2 uniform oval distributions. Thus, we have 10 clusters. Because every sample is a 2-dimensional vector, we present a visualization of the dataset in Fig. 4.1 (a). The second dataset (*Synthetic2*) is also a 2-dimensional dataset of 1500 examples and 20 clusters; 5 gaussian, 4 student-t, 4 uniform rectangle and 7 uniform oval cluster structures. We also provide a visualization of it in Fig. 4.2 (a). Finally, the last dataset (*Synthetic3*) is a 10-dimensional dataset of 500 examples that has 20 clusters; 3 gaussian, 5 student-t, 7 uniform rectangle and 5 uniform ovals.

As we mentioned previously, in RMS algorithm the update equation (Eq. 3.18), has a parameter a, the learning rate. We choose for all the three synthetic datasets the learning rate of all algorithms to be $a = 0.01$ until epoch 150 and then it becomes $a = 0.001$. We follow this because we want the algorithm to initially "explore" with a larger value of learning rate and then we restrict it at a lower one in order to converge. Moreover, for the batch-RMS algorithm, as we presented in chapter 3, we use the baseline of Eq.3.24 to the update equation Eq. 3.25. We choose the parameter γ to be equal to 0.999 for all experiments. Finally, as far as the T parameter is concerned, in all experiments we choose it equal to 1.

A summarization of the main characteristics of the tested datasets is presented in Table 4.1 containing the synthetic and real datasets as well. Tables 4.2 - 4.7 present the results for *Synthetic1*, *Synthetic2* and *Synthetic3* datasets.

In general, we notice that the RMS algorithm usually demonstrates the best performance compared to the other clustering algorithms.

Table 4.1: A summarization of the tested datasets.

Dataset	Instances	Features	Clusters
Synthetic1	500	2	10
Synthetic2	1500	2	20
Synthetic3	500	10	20
Pendigits	3715	16	5
Coil1	216	1000	3
Coil3	360	1000	5

Table 4.2: Average clustering error (J), average NMI and percentage of algorithm superiority for 20 experiments with *Synthetic1* dataset.

Dataset: Synthetic1	RMS	batch-RMS	RGCL	LVQ
Average J	**19.266**	47.862	39.351	37.028
Average NMI	**0.9522**	0.9015	0.9024	0.9037
Percentage	**86.25%**	6.25%	6.25%	1.25%

More specifically, for Synthetic1, from the Table 4.2 and Table 4.3, it can be observed that RMS has the minimum average clustering error and the maximum average NMI, with the difference in clustering error being statistically significant for p-value threshold 0.1. Also, RMS gives the lowest clustering error at the majority of the experiments (86.25%).

Table 4.3: t-scores and p-values for *Synthetic1* dataset.

Dataset: Synthetic1	(t-score, p-value)
RMS vs RGCL	**(-2.0133, 0.0512)**
RMS vs LVQ	**(-1.9895, 0.0538)**
RMS vs batch-RMS	**(-2.9469, 0.0054)**
LVQ vs batch-RMS	(-0.9581, 0.3440)
RCGL vs batch-RMS	(-0.7003, 0.4879)

Table 4.4: Average clustering error (J), average NMI and percentage of algorithm superiority for 20 experiments with *Synthetic2* dataset.

Dataset: Synthetic2	RMS	batch -RMS	RGCL	LVQ
Average J	**28.536**	68.346	69.019	66.435
Average NMI	**0.9559**	0.9190	0.9217	0.9200
Percentage	**100%**	-	-	-

Table 4.5: t-scores and p-values for *Synthetic2* dataset.

Dataset: Synthetic2	(t-score, p-value)
RMS vs RGCL	**(-5.9459, 6.753e-07)**
RMS vs LVQ	**(-5.6542, 1.6972e-06)**
RMS vs batch-RMS	**(-5.1319, 8.7865e-06)**
LVQ vs batch-RMS	(-0.1960, 0.8456)
RCGL vs batch-RMS	(0.0685, 0.9457)

Table 4.6: Average clustering error (J), average NMI and percentage of algorithm superiority for 20 experiments with *Synthetic3* dataset.

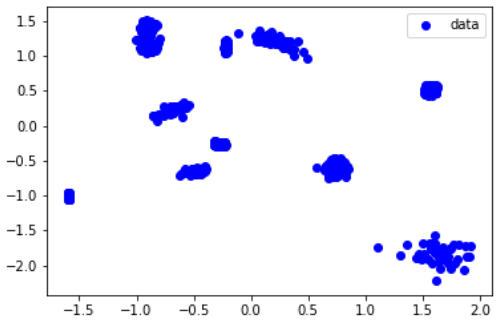
Dataset: Synthetic3	RMS	batch-RMS	RGCL	LVQ
Average J	**765.375**	826.361	963.917	997.313
Average NMI	**0.9121**	0.9041	0.8967	0.8980
Percentage	**60%**	25%	12.5%	2.5%

Again, for Synthetic2 dataset RMS algorithm has the best clustering performance, achieving the lowest clustering error at the 100% of the experiments and the minimum average of clustering error as well, with the superiority being statistically significant.

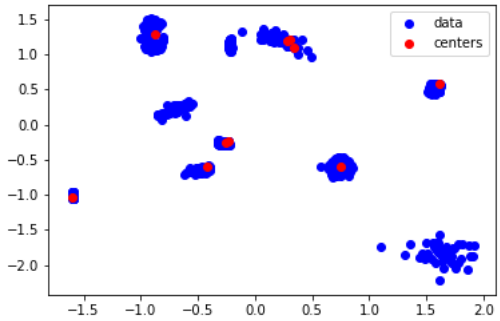
Table 4.7: t-scores and p-values for *Synthetic3* dataset.

Dataset: Synthetic3	(t-score, p-value)
RMS vs RGCL	**(-2.8797, 0.0065)**
RMS vs LVQ	**(-3.7858, 0.00053)**
RMS vs batch-RMS	(-0.9059, 0.3706)
LVQ vs batch-RMS	**(2.7740, 0.0085)**
RCGL vs batch-RMS	**(1.9859, 0.0542)**

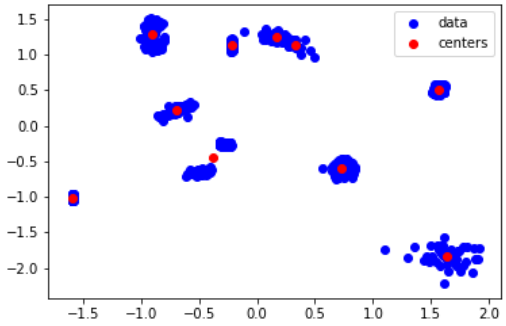
As we understand from the experimental results of the three synthetic datasets, RMS has the best performance. For Synthetic1 and Synthetic2 batch-RMS, LVQ and RGCL seem to have no difference on the averages of the clustering error according to the t-test since the p-value is higher than the threshold of 0.1, although batch-RMS achieves the second best percentage of superiority of the experiments. On the other hand, at Synthetic3, it is clear that batch-RMS has the second best performance after RMS, since Table 4.7 indicates clearly the difference between the averages of LVQ, RGCL versus batch-RMS. Because Synthetic1 and Synthetic2 are datasets with 2-dimensional data points, in Fig. 4.1 and Fig. 4.2 we demonstrate the clustering solution given by each algorithm.



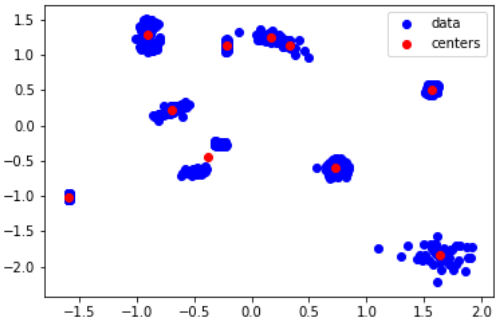
(a)



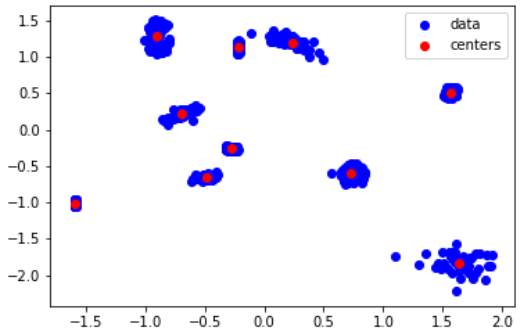
(b)



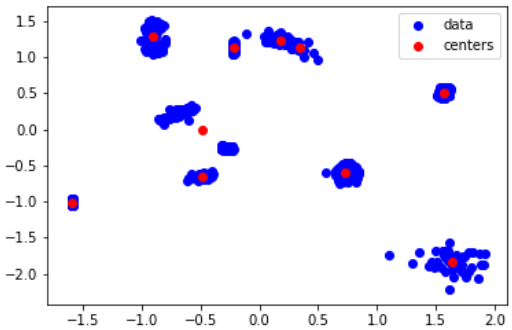
(c)



(d)

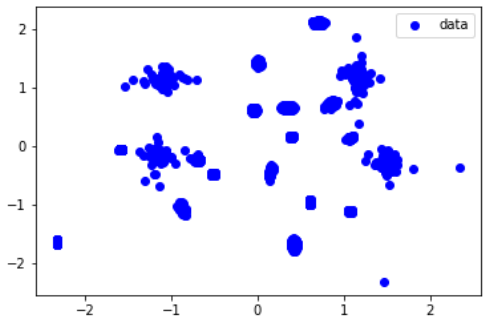


(e)

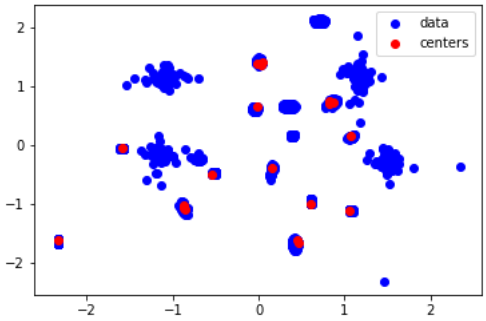


(f)

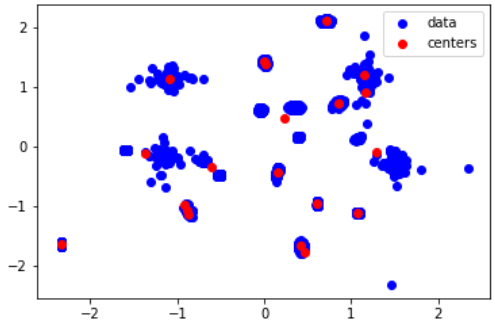
Figure 4.1: (a) Visualization of *Synthetic1* dataset (b) *Synthetic1* dataset and initial canters (c) *Synthetic1* dataset and centers after running LVQ (d) *Synthetic1* dataset and centers after running RGCL (e) *Synthetic1* dataset and centers after running RMS (f) *Synthetic1* dataset and centers after running batch-RMS.



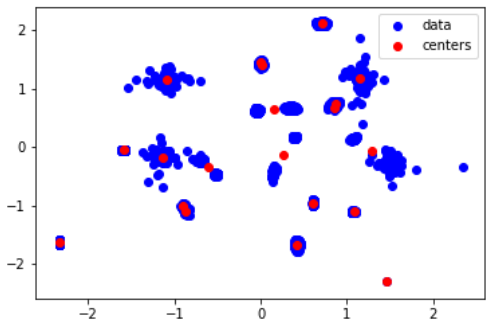
(a)



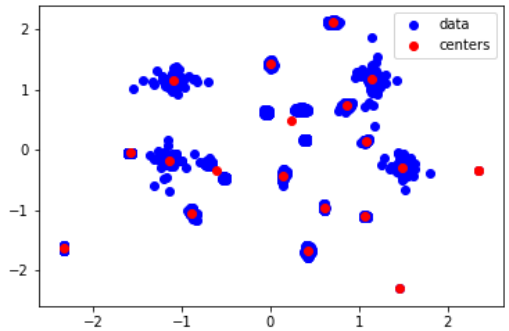
(b)



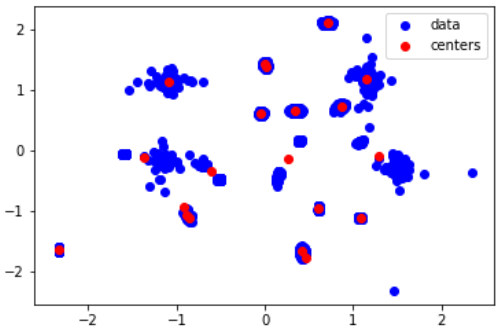
(c)



(d)



(e)



(f)

Figure 4.2: (a) Visualization of *Synthetic2* dataset (b) *Synthetic2* dataset and initial canters (c) *Synthetic2* dataset and centers after running LVQ (d) *Synthetic2* dataset and centers after running RGCL (e) *Synthetic2* dataset and centers after running RMS (f) *Synthetic2* dataset and centers after running batch-RMS.

More specifically, for the Synthetic1 and Synthetic2 datasets, Fig. 4.1 (a) and Fig. 4.2 (a) presents a plot of data and Fig. 4.1. (b) and Fig. 4.2 (b) present along with the data, the randomly initialized centers, respectively. The remaining subfigures present the final solution of the four algorithms; LVQ, RGCL, RMS and batch-RMS respectively. It can be noticed that RMS provides the best clustering solutions.

Since all tested algorithms aim at minimizing the clustering error, in the next figures we present some indicative diagrams of the evolution of clustering error with respect to the number of epochs.

From the plot of the clustering error, it is obvious that the RMS and batch-RMS algorithms exhibit higher stochasticity while RGCL explores in a stricter way and LVQ in a completely deterministic way. LVQ converges very fast to the final solution between epoch 3 and 5.

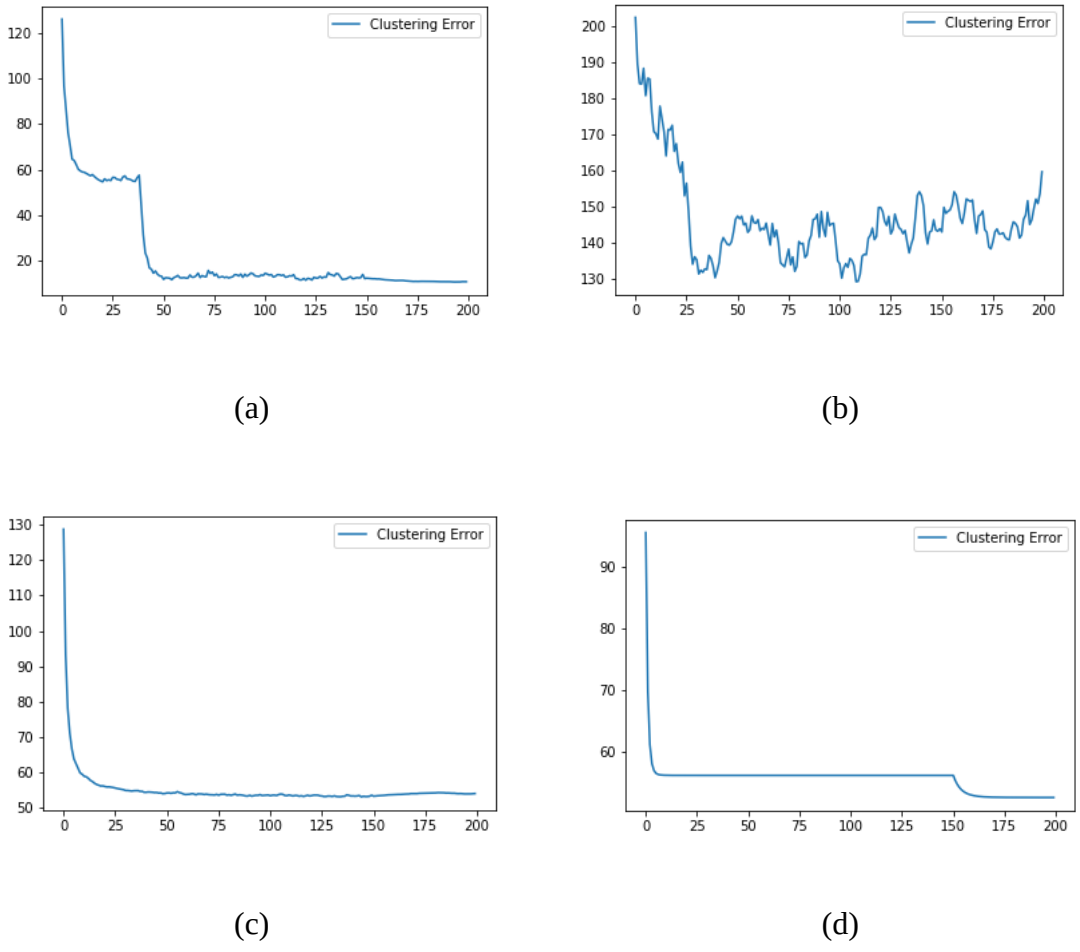
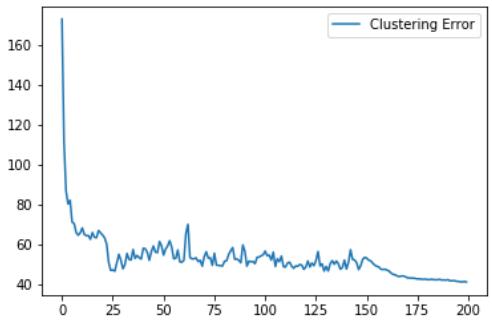
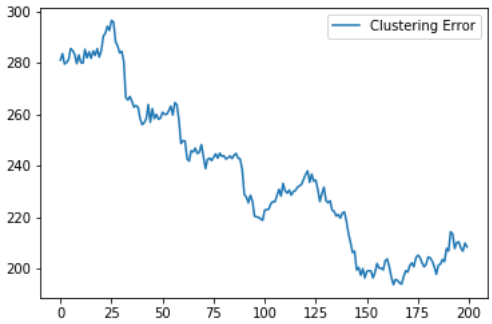


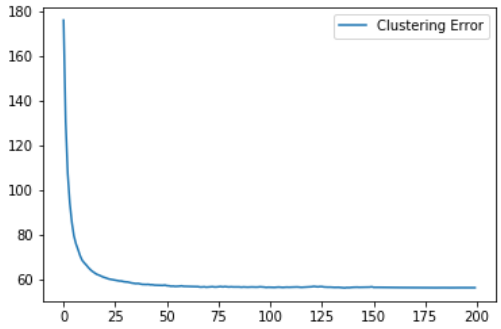
Figure 4.3: (a) Plot of clustering error of RMS (b) Plot of clustering error of batch-RMS (c) Plot of clustering error of RGCL (d) Plot of clustering error of LVQ, for *Synthetic1* dataset.



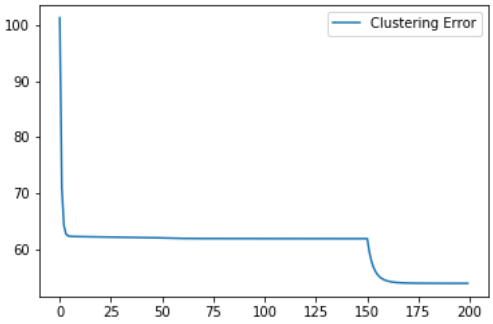
(a)



(b)

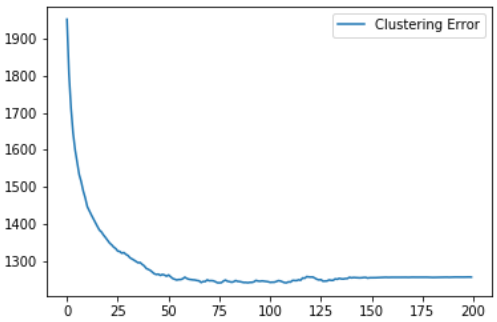


(c)

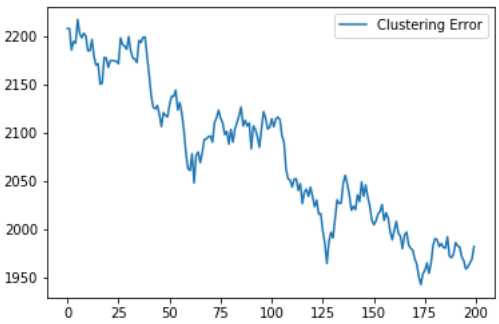


(d)

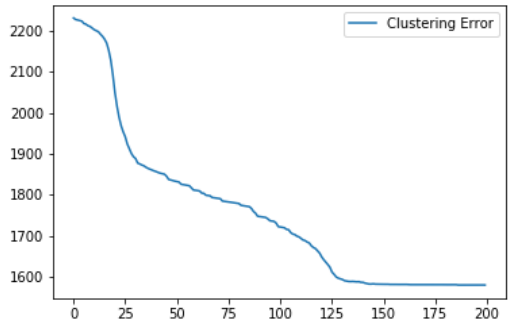
Figure 4.4: (a) Plot of clustering error of RMS (b) Plot of clustering error of batch-RMS (c) Plot of clustering error of RGCL (d) Plot of clustering error of LVQ, for *Synthetic2* dataset.



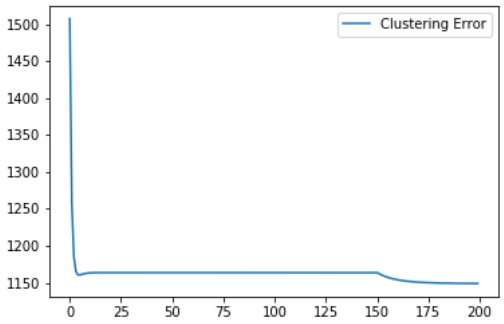
(a)



(b)



(c)



(d)

Figure 4.5: (a) Plot of clustering error of RMS (b) Plot of clustering error of batch-RMS (c) Plot of clustering error of RGCL (d) Plot of clustering error of LVQ, for *Synthetic3* dataset.

From the plots we can see that there are cases where the clustering error of RMS and batch-RMS takes intermediate values higher than LVQ error. This happens because the proposed algorithms seek for a good clustering solution stochastically, thus they may not reach immediately the lowest values of clustering error. Therefore, we apply k-Means at the end, in order for the algorithms to converge deterministically. Through this we observe that the clustering error is lowest eventually than the one found by LVQ. Thus, we conclude that the stochastic exploration property of the algorithms results in a better clustering solution, if it is followed by some exploitation.

4.2.2 Real Data

The four algorithms have been compared in real data as well. We choose two different datasets of real data. The first one is *Pendigits* dataset which is a digit database of 250 samples of handwritten digits (from 0 to 9) from 44 writers. Each sample has 16 features meaning that is a 16-dimensional vector and the total number of samples is 10992. From this dataset, we choose only the five odd digits to test the four algorithms. So, the used dataset is a subset of the Pendigits dataset containing the digits 1, 3, 5, 7, 9. Thus, we have 5 clusters and 3715 examples. The experimental results are presented in tables Table 4.8 and Table 4.9.

Also, we tested our algorithms in *Coil-20* dataset, which is comprised of 72 images taken from different angles for each of 20 selected objects. For our purposes, we use two subsets of the dataset, the first one named *Coil1* dataset, includes 3 objects and has 216 examples and the second one, named *Coil3* includes 5 objects and has 360 examples. Each sample has 1000 features. A summarization of the three datasets has been presented in Table 4.1. Moreover, for the three real datasets we choose the learning rate to be equal to $a = 0.001$. The parameter γ of the baseline at batch-RMS algorithm is 0.999 and T is set equal to 1. Finally, for the t-test we consider the p-value 0.1 to be the threshold and all datasets are normalized. Tables 4.8-4.13 demonstrate the experimental results.

The results in Table 4.8 and Table 4.9 concerning the *Pendigits (1,3,5,7,9)* dataset, demonstrate that all algorithms besides LVQ achieve similar clustering performance in terms of NMI. However, RMS achieves the lowest clustering error in the highest percentage of experiments.

Table 4.8: Average clustering error (J), average NMI and percentage of algorithm superiority for 20 experiments with *Pendigits (1,3,5,7,9)* dataset.

Dataset: Pendigits (1,3,5,7,9)	RMS	batch-RMS	RGCL	LVQ
Average J	**14635.541**	14991.789	14646.963	17014.876
Average NMI	0.5485	0.5297	**0.5507**	0.5187
Percentage	**37.5%**	15%	32.5%	15%

As far as Coil1 and Coil3 datasets are concerned, from the results we conclude that batch-RMS has the best performance, while the others seem to provide similar results. Also, batch-RMS achieves the lowest clustering error in the highest percentage of experiments. The second best percentage corresponds to RMS, although its average clustering error is not different from RGCL and LVQ.

Table 4.9: t-scores and p-values for *Pendigits (1,3,5,7,9)* dataset.

Dataset: Pendigits (1,3,5,7,9)	(t-score, p-value)
RMS vs RGCL	(-0.0219, 0.9826)
RMS vs LVQ	**(-4.7481, 2.906e-05)**
RMS vs batch-RMS	(-0.7060, 0.48446)
LVQ vs batch-RMS	**(4.3456, 9.997e-05)**
RCGL vs batch-RMS	(-0.7089, 0.4826)

Table 4.10: Average clustering error (J), average NMI and percentage of algorithm superiority for 20 experiments with *Coil1* dataset.

Dataset: Coil1	RMS	batch-RMS	RGCL	LVQ
Average J	130.991	**121.993**	129.544	130.056
Average NMI	0.7892	**0.9343**	0.8081	0.8141
Percentage	17.08%	**52.08%**	15.415%	15.415%

Table 4.11: t-scores and p-values for *Coil1* dataset

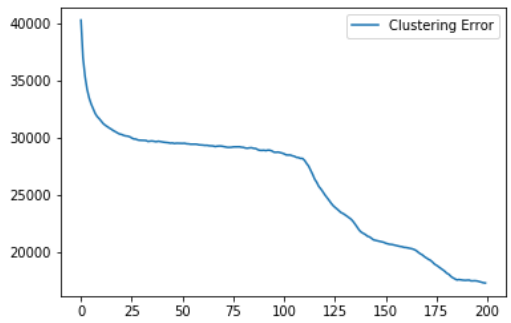
Dataset: Coil1	(t-score, p-value)
RMS vs RGCL	(0.2831, 0.7785)
RMS vs LVQ	(0.1798, 0.8581)
RMS vs batch-RMS	**(2.0063, 0.0519)**
LVQ vs batch-RMS	**(1.9756, 0.0554)**
RCGL vs batch-RMS	**(1.9029, 0.0646)**

Table 4.12: Average clustering error (J), average NMI and percentage of algorithm superiority for 20 experiments with *Coil3* dataset.

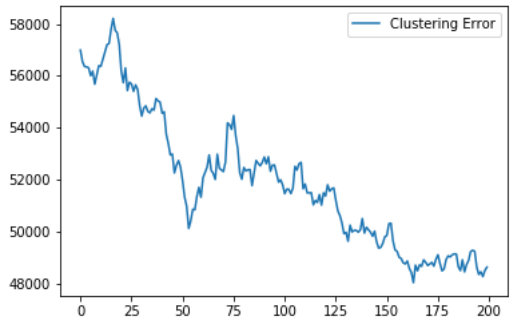
Dataset: Coil3	RMS	batch-RMS	RGCL	LVQ
Average J	264.226	**251.591**	262.563	264.021
Average NMI	0.6974	**0.8529**	0.7091	0.6945
Percentage	15%	**85%**	-	-

Table 4.13: t-scores and p-values for *Coil3* dataset

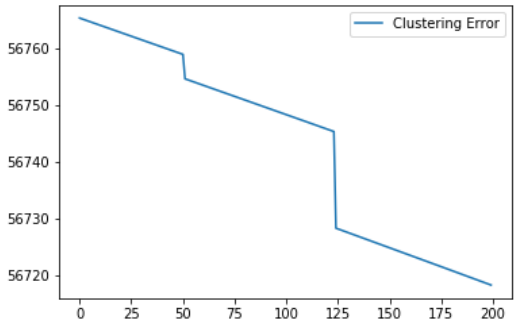
Dataset: Coil3	(t-score, p-value)
RMS vs RGCL	(0.4852, 0.6302)
RMS vs LVQ	(0.0571, 0.9547)
RMS vs batch-RMS	**(3.9142, 0.00036)**
LVQ vs batch-RMS	**(4.2627, 0.00012)**
RCGL vs batch-RMS	**(4.0322, 0.00025)**



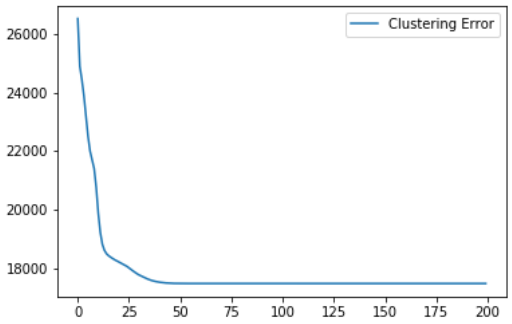
(a)



(b)

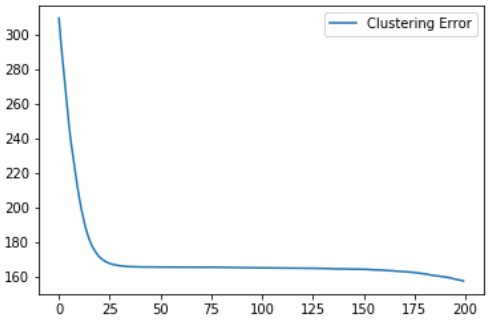


(c)

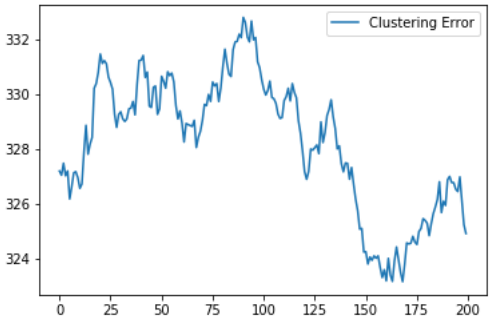


(d)

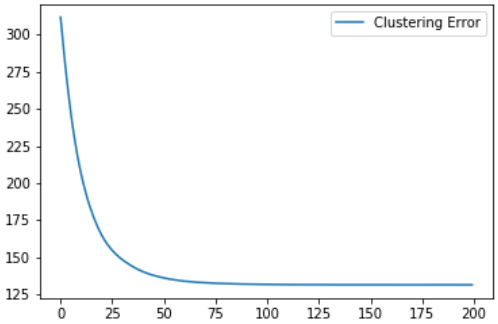
Figure 4.6: (a) Plot of clustering error of RMS (b) Plot of clustering error of batch-RMS (c) Plot of clustering error of RGCL (d) Plot of clustering error of LVQ, for *Pendigits (1,3,5,7,9)* dataset.



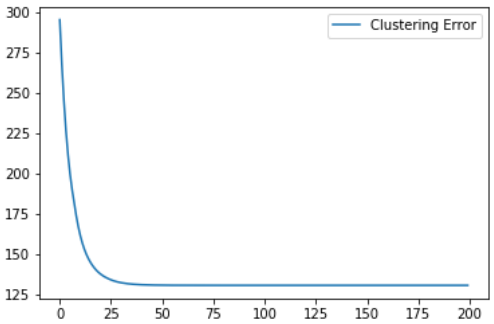
(a)



(b)



(c)



(d)

Figure 4.7: (a) Plot of clustering error of RMS (b) Plot of clustering error of batch-RMS (c) Plot of clustering error of RGCL (d) Plot of clustering error of LVQ, for *Coil1* dataset.

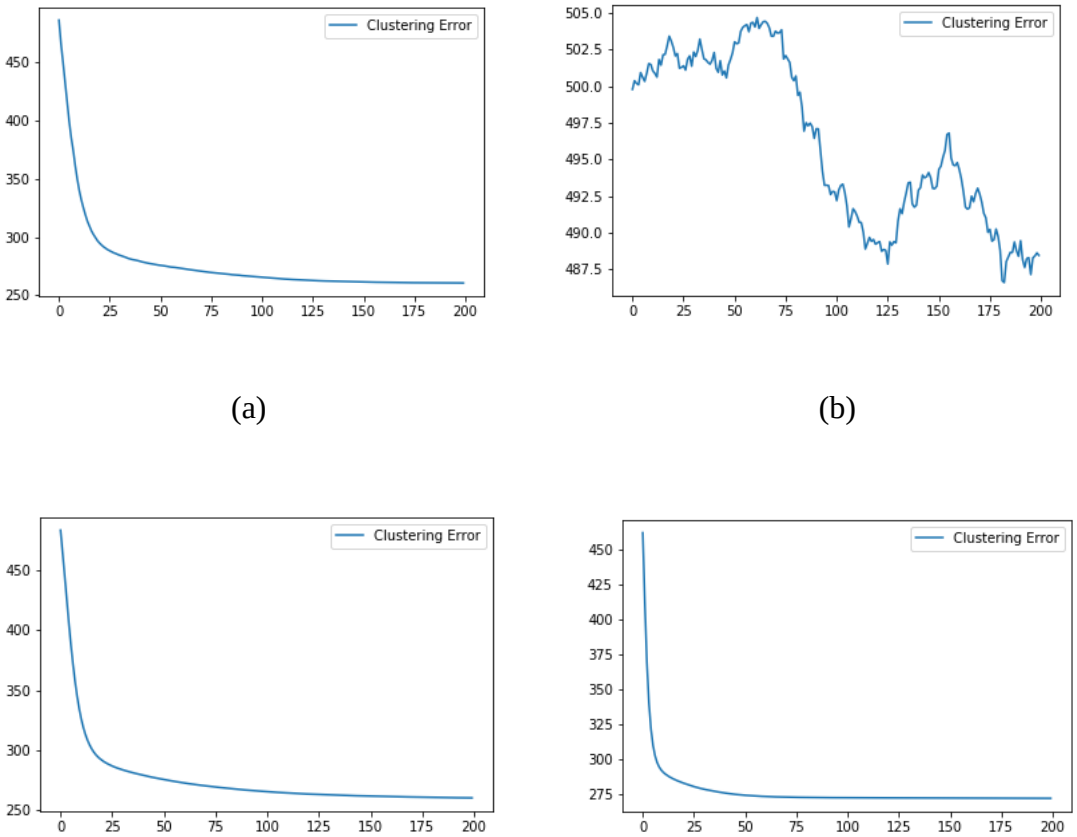


Figure 4.8: (a) Plot of clustering error of RMS (b) Plot of clustering error of batch-RMS (c) Plot of clustering error of RGCL (d) Plot of clustering error of LVQ, for *Coil3* dataset.

In Fig. 4.6-4.8 we provide some indicative plots of the evolution of clustering error for the four algorithms exactly as we did with the synthetic datasets. The plots demonstrate that the clustering error is minimized as iterations proceed. As we mentioned before, the learning rate in real datasets is equal to 0.001. For this reason, RMS and RGCL algorithm seem not to "explore" very much. We select a smaller learning rate because higher values lead to worse results.

4.3 Discussion

Having presented the experimental results, we end up with some interesting conclusions. First of all, the two proposed algorithms, RMS and batch-RMS, lead to better results and better minimization of the clustering error than RGCL and LVQ do. It worth noticing that the four algorithms minimize the same criterion, the clustering error (Eq. 4.1). Therefore, we can claim that by inserting stochasticity in a clustering algorithm based on competitive learning and adjusting it properly with a reinforcement learning scheme leads to more efficient solutions.

Moreover, another interesting result is that RMS and batch-RMS do not depend on the initialization of cluster representatives as much as LVQ does which always ends up in the same clustering solution. This happens because of the stochasticity included in the clustering scheme and implemented by MS units. Consequently, RMS and batch-RMS achieve a disparity in the clustering solutions even from the same initial parameters. As far as the comparison with RGCL is concerned, we notice that a single MS unit performs better than a team of Bernoulli units. Maybe this holds due to the fact that it is more efficient to train a single unit with multiple outcomes than a team of units with binary outcomes.

The RMS algorithm and the RGCL are used without the baseline. We choose not to use it because preliminary results indicated that baseline in RMS does not provide noticeable difference. However, in batch-RMS the baseline is necessary, because the update of the parameters is not performed after each sample has been presented to the system but after all samples have been used. Thus, we need information about the reinforcement values at previous steps in order to judge where the currently selected actions correspond to improved clustering error or not. Based on a lot of experiments, we have decided the parameter γ of the baseline, that actually defines how much important the past is, to be equal to 0.999 in all cases, thus we assign high importance to past values.

Also, the learning rate parameter in the update equations of RMS, RGCL and batch-RMS has been selected empirically after several experiments. In order to achieve a fair comparison, all algorithms are executed with the same learning rate. However, we have empirically found that every algorithm needs its own learning rate to provide best performance. For example, batch-RMS needs a lower learning rate than RMS or RGCL, while RGCL needs the highest values.

Also, we have noticed that in the synthetic datasets we get similar results even with higher learning rate values. On the other hand, in real datasets it is more efficient if the learning rate takes smaller values.

RMS and batch-RMS depend on initialization. However, from the experimental results and especially at synthetic datasets where the cluster structure is known and the number of clusters is large, we notice that RMS can detect the real clusters very efficiently. Thus, we could say that RMS is effective in datasets with a big number of clusters.

Furthermore, in the batch-RMS as we mentioned in Chapter 3, we design the reinforcement signal as

$$r = \frac{c}{E} \tag{4.3}$$

where c is a constant and E is the clustering error. In all experiments, we choose constant c to be equal to the value of the clustering error at the first epoch of each run. This has been chosen empirically based on the observation. Maybe there are other functions that lead to better or an alternative value of constant c could have been selected.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, we proposed two algorithms, the online RMS and batch-RMS, that cluster the data through a stochastic procedure, with the help of a reinforcement signal sent from the environment. The reinforcement signal indicates to the system if it operates well or not and the system learns to implement clustering strategy by maximizing this reinforcement signal. We use the family of REINFORCE algorithms to update system parameters. It was proved that the update of the parameters leads to the stochastic minimization of the well-known clustering error of k-Means.

We tested our algorithms in real and synthetic datasets and compared the algorithms with LVQ for clustering and RGCL. We evaluated the four algorithms based on the clustering error and NMI. The results demonstrate that RMS and batch-RMS outperform RGCL and LVQ giving better clustering solutions in the majority of tested datasets, i.e. solutions with minimum clustering error and maximum NMI. Also, we noticed that our algorithms, because of their stochasticity, overcome more efficiently the limitation of a bad initialization of the cluster representatives by exploring better the solution space and providing superior results. Furthermore, RMS proved to give promising results in datasets with a large number of clusters.

.

5.2 Future Work

Finally, we present some thoughts about potential future work and some preliminary studies on deep clustering: deep learning and clustering simultaneously.

5.2.1 Deep Clustering

Many reasons can affect the performance of a clustering algorithm. For example, one such reason can be the need of data preprocessing such as dimensionality reduction. With the term of dimensionality reduction, we mean that our data are projected from an initial space to another space with a lower dimension. This is very popular task nowadays since data usually suffer from the curse of dimensionality. Thus, in the case of clustering, instead of applying a clustering algorithm in the initial space, we may apply it in the projected one.

Because of the success of deep neural networks (DNNs) in supervised and unsupervised learning, unsupervised deep learning approaches are widely used for dimensionality reduction prior to clustering. One such approach is the autoencoder, an unsupervised deep learning technique that uses DNNs in order to perform dimensionality reduction. More specifically, an autoencoder is a neural network that learns to copy its input to its output. It has an intermediate hidden layer, the code, and its constituted by two parts; an encoder that maps the input to the code and a decoder that maps the code to a reconstruction of the original input. A visualization of the autoencoder is presented in Fig. 5.1.

Thus, the objective function which an autoencoder tries to optimize, is the reconstruction error described by the equation

$$E_{rec} = \sum_{i=1}^{N} \min \left\| x_i - x_{i_{rec}} \right\|^2 \tag{5.1}$$

where $X = (x_1, \dots, x_N)$ is a set of unlabeled data and $x_{i_{rec}}$ is the reconstruction of x_i.

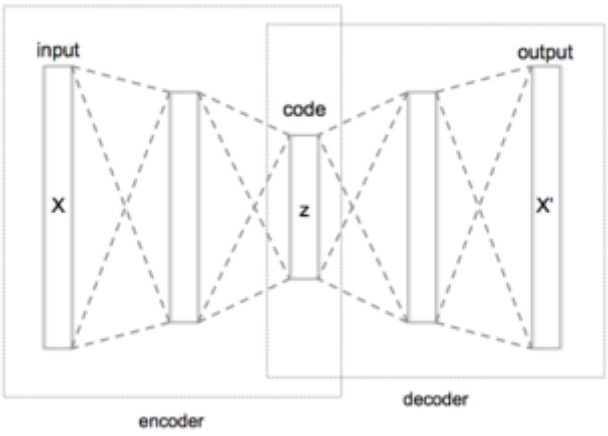


Figure 5.1: Autoencoder.

In several learning approaches, first the dimensionality of data is reduced and then any clustering algorithm is applied. In [7] a method has been proposed where dimensionality reduction (with deep learning techniques) and clustering are performed simultaneously. Inspired from this, we thought to use an autoencoder in order to reduce the dimensionality of data and in the corresponding latent space to cluster our data by applying the RMS algorithm simultaneously with the autoencoder training. This is made more clear in Fig 5.2. More specifically, the input x_i will be projected through the encoder to z_i, the z_i which correspond to the cluster prototypes will be updated by RMS update equation in order the clustering strategy to be implemented and then will be projected back to $x_{i_{rec}}$ through the decoder.

Consequently, the objective function to train the system is the sum of what the RMS minimizes and what the autoencoder minimizes, i.e. the objective function will be the sum of clustering error J (in the latent space) and reconstruction error E_{rec},

$$E = J + E_{rec} \tag{5.2}$$

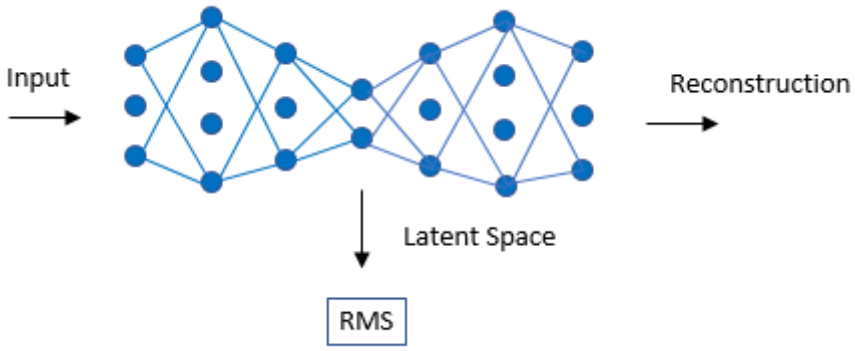


Figure 5.2: Deep clustering framework.

In this way, the deep clustering network will update their parameters in order to achieve both clustering and x_i's reconstruction. According to this deep clustering framework, we cluster our data in a latent space and not in the original one. This method may lead to more clustering friendly latent spaces.[7]

Although this deep clustering network seems promising, our preliminary experimental results have shown that the deep clustering strategy does not lead to more improved results than those obtained by treating the dimensionality reduction with an autoencoder first and then apply RMS clustering in the latent space, separately. Thus, we are still not certain that a method that proposes RMS clustering and dimensionality reduction simultaneously is more efficient. For this reason, additional experimentation is needed to draw reliable conclusions.

5.2.2 Other Future Work

As far as the RMS and batch-RMS is concerned, some directions of future work are the following. First of all, alternative reward functions might be designed that lead to better performance. Furthermore, besides Bernoulli or MS units, maybe other types of stochastic units could be defined based on different probability distributions.

Another interesting potential work is to introduce the stochasticity not necessarily in the selection of a cluster prototype, but somewhere else, like in the distance computation for example. As we mentioned, the distance used here is the Euclidean and is calculated deterministically. It is possible that this distance can be modified by adding some noise resulting from some known probability distribution. Moreover, besides Euclidean distance, other distance metrics can also be used.

RMS and batch-RMS optimize locally the clustering criterion, exactly like LVQ and RGCL do. Since the optimization is local, all the algorithms can be trapped in local minimum. A way to overcome this limitation is to use sustained exploration suggested in [1]. Another possible future work could be to find a better way to handle exploration versus exploitation. For example, through the execution of the algorithm, some steps could be deterministic and some other stochastic. Consequently, we will not apply the k-Means only at the end, but also somewhere in the intermediate steps of the algorithms.

Finally, as we presented the batch-RMS algorithm updates only the winning cluster prototype based on the REINFORCE framework, making the algorithm stricter to the parameter updates. As future work we want to let the batch-RMS algorithm to “explore” more the solution space. Thus, instead of leaving the cluster prototypes that are different form the winning one unchanged, they maybe will be updated from the second term of the update equation (Eq. 3.24) as well.

REFERENCES

[1] A. Likas, "A Reinforcement Learning Approach to Online Clustering", *Neural Computation 11*, pp. 1915-1932, 1999.

[2] A. Likas, "Multivalued Parallel Recombinative Reinforcement Learning", *HERCMA '98*, 1998.

[3] R. J. Williams, "Simple Statistical Gradient – Following Algorithms for Connectionist Reinforcement Learning", *Machine Learning 8*, pp. 229-256, 1992.

[4] T. Kohonen, *Self-organization and associative memory* (3rd ed.). Berlin: Springer-Verlag, 1989.

[5] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. AddisonWesley, Boston, MA, USA, 2005.

[6] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.

[7] B. Yang, X. Fu, N. D. Sidiropoulos, M. Hong, "Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering", *ICML*, 2017.

SHORT CV

Eleni Pachi was born in Ioannina, Greece in 1992. In 2010, she enrolled in the department of Mathematics of Aristotle University of Thessaloniki and received the BSc degree in 2014. In 2014, she enrolled as a MSc student in the same department and received the MSc degree "Theoretical Computing and Theory of Systems and Control" in 2016. In continuation of her studies, she enrolled as a MSc student in the department of Computer Science & Engineering of University of Ioannina. After fulfilling her responsibilities as a graduate student, she presented her thesis in October 2019 in order to complete the Master's Degree. Her main interests are in the area of data analysis and machine learning.