

TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

Dendritic Application to Machine Learning

Author:

Kosmas Pinitas

Committee:

Michalis Zervakis

Apostolos Dollas

Panayiota Poirazi

*A thesis submitted in fulfillment of the requirements
for the degree of 5-year Diploma*

School of Electrical and Computer Engineering

August 27, 2021

Declaration of Authorship

I, Kosmas Pinitas, declare that this thesis titled, “Dendritic Application to Machine Learning” and the work presented in it are my own. I confirm that:

- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Kosmas Pinitas

Date:

August 27, 2021

*“And now, what’s going to happen to us without barbarians?
They were, those people, a kind of solution.”*

Konstantinos Kavafis

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

5-year Diploma

Dendritic Application to Machine Learning

by Kosmas Pinitas

The current deep learning architectures achieve remarkable performance when trained in large-scale controlled datasets. However, the predictive ability of these architectures significantly decreases when learning new classes incrementally due to their inclination to forget the knowledge acquired from previously seen data, also called catastrophic-forgetting. The Self-Organizing Maps can model the input space utilizing constrained-kmeans and thus ensure that the past knowledge is maintained. Hence, we propose the Dendritic-Self-Organizing Map algorithm consisting of a single layer of Self-Organizing Maps, which extract patterns from specific regions of the input space, and an association matrix that estimates the association between units and labels. The best-matching unit of an input pattern is selected using the maximum cosine similarity rule, while the point-wise mutual information is employed for inferencing. Our method performs unsupervised classification since we do not utilize the labels for targeted weight update. Finally, the results indicate that our algorithm outperforms several state-of-the-art continual learning algorithms on benchmark datasets such as the Split-MNIST and Split-CIFAR-10.

Τα υπάρχοντα μοντέλα βαθιάς μάθησης επιτυγχάνουν αξιοσημείωτη απόδοση όταν εκπαιδεύονται σε μεγάλα σύνολα δεδομένων, Ωστόσο απόδοση των μοντέλων αυτών μειώνεται σημαντικά όταν μαθαίνουν σταδιακά νέες κλάσεις λόγω της τάσης τους να ξεχνούν τις γνώσεις που έχουν αποκτηθεί από προηγούμενα δεδομένα, το φαινόμενο αυτό ονομάζεται καταστροφική λήθη (catastrophic forgetting). Οι Αυτοοργανωτικοί Χάρτες μπορούν να μοντελοποιήσουν τον χώρο εισόδου χρησιμοποιώντας **constrained-kmeans** διασφαλίζοντας τη διατήρηση των προηγούμενων γνώσεων. Ως εκ τούτου, εισάγουμε τον Δενδριτικό-Αυτοοργανωτικό Χάρτη που αποτελείται από ένα μόνο επίπεδο Χαρτών Αυτοοργάνωσης, οι οποίοι εξάγουν μοτίβα από συγκεκριμένες περιοχές του χώρου εισόδου και ένα πίνακα συσχέτισης που εκτιμά τη συσχέτιση μεταξύ μονάδων και ετικετών. Η μονάδα που ταιριάζει καλύτερα σε ένα μοτίβο εισόδου επιλέγεται με βάση τον κανόνα της μέγιστου συνημιτόνου, ενώ η αμοιβαία πληροφορία χρησιμοποιείται για συμπερασμό. Η μέθοδος μας εκτελεί ταξινόμηση χωρίς επίβλεψη, καθώς δεν γίνεται χρήση των ετικετών κατά την ενημέρωση των διανυσμάτων βάρους των χαρτών. Τα αποτελέσματα υποδεικνύουν ότι ο προτεινόμενος αλγόριθμος υπερτερεί πολλών αλγορίθμων συνεχούς μάθησης στα σύνολα δεδομένων όπως το Split-MNIST και το Split-CIFAR-10.

Acknowledgements

First and foremost, I would like to thank Dr. Panayiota Poirazi, whose mentoring style and determination motivated me to pursue an academic career. I would also like to thank Dr. Spyridon Chavlis for the intriguing conversations that introduced me to the field of biologically plausible AI and inspired me to conceive the main topic of this thesis. Furthermore, I would like to thank Professor Georgios N. Yannakakis for his suggestions regarding the evaluation of the models and the comments on the experimental results.

In addition, I am grateful to Dr. George Giannakopoulos who introduced me to a more formal research methodology and inspired me to explore the connections of AI with Cognitive Science. I would like to express my thankfulness to Prof. Athanasios P. Liavas for supporting my decision to follow an academic career pathway. Moreover, I would like to thank my close friends and colleagues for their support and trust.

Additionally, I am grateful to Professor Apostolos Dollas and Professor Michalis Zervakis for evaluating my work. Last but not least, I would like to express my gratitude to my family for supporting me throughout these years, and thus gave me the opportunity to broaden my horizons and carve my own path in life.

Contents

Declaration of Authorship	iii
Abstract	vii
Acknowledgements	ix
1 Introduction	1
1.1 Motivation	2
1.2 Machine Learning in a Nutshell	2
1.3 Thesis Contribution	4
1.4 Thesis Outline	4
2 Continual Learning	5
2.1 Chapter Outline	6
2.2 Definition of Continual Learning	6
2.2.1 Historical Background	6
2.3 Continual Learning Terminology & Challenges	7
2.3.1 Basic Definitions	7
2.3.2 Challenges	8
Catastrophic Forgetting	8
Memory Management	9
Concept Drift	9
2.4 A Continual Learning framework with emphasis on classification	10
2.4.1 Definitions	10
2.4.2 Continual Learning Scenarios	11
2.4.3 Datasets	12
2.4.4 Protocols	13
2.4.5 Evaluation Metrics	14
2.4.6 Baselines	14
The essentials of Deep Neural Networks (DNNs)	15
2.5 Methods for addressing catastrophic forgetting	17
2.5.1 Replay	17
2.5.2 Regularization	18
2.5.3 Parameter Isolation	18
2.6 State-of-the-art algorithms	19
2.6.1 Copy Weight with Reinit (CWR)	19
Copy Weight with Reinit + (CWR+)	20
2.6.2 Deep Generative Replay (DGR)	21
2.6.3 Elastic Weight Consolidation (EWC)	22
2.6.4 Learning Without Forgetting (LwF)	23
2.6.5 Synaptic Intelligence (SI)	24
2.6.6 Context-dependent Gating (XdG)	24
2.6.7 Hybrid Algorithms	25

	Deep Generative Replay and Regularization (DGR)	25
	Architectural and Regularization approach (AR1)	25
3	Self-Organizing Maps	27
3.1	Chapter Outline	28
3.2	Historical Background	28
3.3	Quantization	29
	Vector Quantization (VQ)	29
	K-means	29
	Soft-Clustering	30
3.4	Fundamentals of Self-Organizing Maps	31
3.4.1	Online SOM Learning Algorithm	31
3.4.2	Methods for analyzing SOM convergence	32
	Markov Chain Theory (MCT)	32
	Ordinary Differential Equation (ODE)	33
	Robbins-Monro algorithm theory	34
3.5	Theoretical analysis of the 1D SOM case	36
3.5.1	A special case	36
3.5.2	The general 1D SOM convergence case	38
3.6	Theoretical analysis of the multi-dimensional case	38
3.6.1	Continuous setting	38
3.6.2	Discrete setting	39
3.7	The stochastic nature of the online SOM training algorithm	40
3.8	Batch-SOM: A deterministic approach	42
3.9	SOM variations	43
3.9.1	Heskens' rule	43
3.9.2	Soft Topologic Mapping (STM)	44
3.9.3	Other Variations of SOM	45
4	Dendritic Self-Organizing Maps	47
4.1	Chapter Outline	48
4.2	Background	48
4.2.1	Motivation	49
4.3	Dendritic Self-Organizing Maps	50
4.3.1	Incorporating neurobiological concepts into SOMs	51
	MSE Minimization Test	53
	Visual Test	54
	ANN Test	55
	Time Test	57
	Overall	57
4.3.2	DendSOM Architecture	58
	Learning Algorithm	58
	Defining a decision rule for unsupervised classification	60
	Investigating the effects of a new BMU rule	61
4.3.3	DendSOM for CL	63
	Task-Incremental Learning	64
	Domain-Incremental Learning	65
	Class-Incremental Learning	66
4.4	Technical Stuff	67
4.4.1	Implementation Details	67
4.4.2	Hyperparameter Analysis	68

5	Conclusions & Future Work	73
5.0.1	Chapter 2: Continual Learning	73
5.0.2	Chapter 3: Self-Organizing Maps	73
5.0.3	Chapter 4: Dendritic Self-Organizing Maps	73
5.0.4	Future Work	74

List of Figures

1.1	ML definition	3
1.2	AI ML and DL relation	3
2.1	CL task	8
2.2	catastrophic forgetting	9
2.3	concept drift	10
2.4	TIL MNIST	11
2.5	DIL MNIST	11
2.6	CIL MNIST	12
2.7	svhn	12
2.8	cifar-10	12
2.9	mnist	13
2.10	permutation-mnist	13
2.11	rotation-mnist	14
2.12	a simple neural network	15
2.13	continual learning methods	19
2.14	CWR algorithm	20
2.15	CWR+ algorithm	20
2.16	dynamic architecture training	21
2.17	DGR model	21
2.18	EWC training parameters trajectory	22
2.19	LwF algorithm	24
2.20	AR1 algorithm	25
3.1	Voronoi diagram for VQ	30
3.2	A simple Markov Chain	33
3.3	A simple SOM	35
3.4	SOM h function	35
3.5	A 1D SOM	37
3.6	A 2D SOM disposition distortion case	37
3.7	SOM input mapping	41
4.1	Biological Neuron	48
4.2	SOMLP architecture	49
4.3	SOMLP task masks	50
4.4	SOM architecture for images	52
4.5	RSOM architecture for images	52
4.6	DendSOM architecture for images	53
4.7	Visual test results	55
4.8	ANN test procedure	56
4.9	DendSOM training algorithm	58
4.10	DendSOM with association matrix	59
4.11	cosine similarity BMU selection rule	63

4.12	This figure illustrates the behavior of a and σ functions for $r_{exp} \in \{1, 2\}$	64
4.13	This figure illustrates the behavior of a and σ functions for different values of λ	68
4.14	This figure illustrates how the initial learning rate affects the unsupervised classification accuracy of the DendSOM algorithm on the MNOST dataset	69
4.15	This figure illustrates the influence of α_{crit} and r_{exp} hyperparameters over the DendSOM algorithm. It should be noted that the critical lr axis is in logarithmic scale since $\alpha_{crit} \in \{0.5, 0.05, 0.005, 0.0005, 0.00005, 0.000005\}$	70
4.16	This figure illustrates how the receptive field size affects the DendSOM algorithm	70
4.17	The influence of number of units per map over the DendSOM algorithm is illustrated in this figure	71

List of Tables

2.1	CL scenarios	11
3.1	Soft vs Hard Clustering	31
3.2	Kohonen vs Heskes BMU rule	43
4.1	MSE experiment architecture parameters	53
4.2	MSE experiment results	54
4.3	ANN experiment results	55
4.4	Time experiment results	57
4.5	Overall results	57
4.6	The table summarizes the accuracy score of each architecture in unsu- pervised classification on three benchmark datasets	61
4.7	Training hyperparameters for unsupervised classification	62
4.8	Architecture parameters for unsupervised classification	62
4.9	IL training parameters	64
4.10	Task-IL results	65
4.11	Domain-IL results	65
4.12	Class-IL results	66
4.13	Class-IL results	66

List of Abbreviations

AGI	Artificial General Intelligence
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR1	Architectural and Regularization hybrid algorithm
BMU	Best-Matching Unit
CL	Continual Learning
CLA	Continual Learning Algorithm
CPU	Central Processor Unit
CWR	Copy Weight with Reinit
DendSOM	Dendritic Self-Organizing Map
DGR	Deep Generative Replay
DL	Deep Learning
DNN	Deep Neural Network
DSOM	Deep Self-Organizing Map
EWC	Elastic Weight Consolidation
GC	Generative Classification
GPU	Graphic Processor Unit
GSOM	Growing Self-Organizing Map
GTM	Generative Topographic Maps
IL	Incremental Learning
LwF	Learning Without Forgetting
MC	Markov Chain
MCT	Markov Chain Theory
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
ODE	Ordinary Differential Equation
OS-Map	Oriented & Scaling Map
PMI	Pointwise Mutual Information
RL	Reinforcement Learning
RSOM	Receptive Self-Organizing Map
SGD	Stochastic Gradient-Descent
SI	Synaptic Intelligence
SOM	Self-Organizing Map
SOMLP	Self-Organizing Multi-Layer Perceptron
STM	Soft Topographic Mapping
TASOM	Time-Adaptive Self-Organizing Map
VQ	Vector Quantization
XdG	X Context-dependent Gating

Dedicated to my family and friends...

Chapter 1

Introduction

Deep learning (DL) architectures have achieved state-of-the-art performance in several fields such as machine translation, computer vision, and drug design. Nevertheless, DL algorithms are ill-equipped for learning tasks sequentially. Most deep neural networks (DNNs) can be trained when the entire labeled dataset is at one's disposal, and often, re-training is required when new data becomes available due to catastrophic-forgetting. On the other hand, humans can achieve impressive performance on real-world problems since they can adapt to environments where new tasks emerge over time by leveraging their past experiences. Consequently, a human can continuously acquire new knowledge without forgetting what he has previously learned, which is not the case for DNNs.

Although DL techniques have been widely applied in numerous fields, DNNs are still vaguely inspired by the biological neural networks since their computational unit is a simplified model of a biological neuron. Dendrites are thin projections of a neuron that are specialized in receiving information from other neurons. The information is transferred from a neuron to other neurons through electrochemical signals. Hence, the existence of ion channels in dendrites, combined with the permeability of dendritic networks, suggests that a synaptic input can affect its neighboring synapses in a nonlinear manner.

Furthermore, biological neural networks have the ability to change throughout life, a process known as plasticity. Studies in plasticity mechanisms suggest that dendrites are the basic organizational unit for integrating synaptic input, undergoing synaptic plasticity, and storing multiple complex features of the synaptic input [1, 2]. Thus, the incorporation of dendrites can assist current DL models in obtaining greater information processing power. Therefore, drawing inspiration from the structure and properties of dendrites, we introduce a Self-Organizing Map (SOM)-based architecture that models the dendritic nonlinearities and permits local-parallel computations.

The Dendritic-SOM (DendSOM) model consists of a single layer of SOMs. Each SOM is treated as a group of dendrites, and its units are connected to a single soma. In fact, the SOMs model different sub-regions of the input space, and the somata estimate the association between the best-matching units and class labels. We test our model on unsupervised classification and continual learning scenarios and show that the DendSOM is capable of alleviating the effects of catastrophic forgetting.

1.1 Motivation

In the real world, the computational models are required to learn and remember multiple tasks since they are exposed to continuous streams of data from non-stationary distributions. Thus, an agent must acquire, fine-tune, retain, and transfer knowledge to deal with real problems. Learning new tasks on a constant basis without forgetting is called continual learning and constitutes a challenging branch of Artificial Intelligence (AI). Although there have been significant advances in ML theory and algorithms over the last few years, there is still a lot of work to be done towards creating computational systems that can realize life-long learning.

However, as humans, we can acquire new skills and leverage both new and past experience to refine them. Although we tend to forget previously learned information, learning new skills does not usually interfere with consolidated knowledge since our brain can maintain a balance between learning and retaining knowledge which is also called stability-plasticity balance. The stability-plasticity dilemma concerns both artificial and biological neural networks, and it states that learning requires both stability and plasticity. Nevertheless, it is important to strike a balance between those concepts since too much plasticity results in forgetting, while too much stability prevents the assimilation of new knowledge.

Consequently, employing biological concepts into existing AI algorithms and creating biologically plausible architectures can be the key to building human-level artificial general intelligence (AGI). In fact, the human brain is the only system that displays this level of intellect, and thus it is the only proof that such intelligence can exist. Although examining biological intelligence can be proved beneficial for the field of AI, there is no need to enforce the incorporation of biological plausibility to every algorithm since, from a practical standpoint, what works is what matters. Thus a basic understanding of the functions, architectures, and algorithms that the brain uses is enough to help us design more efficient ML algorithms.

1.2 Machine Learning in a Nutshell

ML is a subset of AI and corresponds to the study of algorithms that improve by leveraging data. In brief, ML is primarily used in applications where the task can not be performed by conventional algorithms. ML algorithms are used to construct a model based on training data in order to identify patterns and make accurate predictions without human assistance. According to the nature of the input data, the ML algorithms are typically divided into three broad categories:

- **Supervised Learning:** both input and output data, also called samples and labels, are presented to supervised learning algorithms. The goal of these models is to learn a function that maps the input to the output space.
- **Unsupervised Learning:** mainly deals with unlabeled data, and it allows the model to detect patterns that were undetected and thus discover structures that can group the input data.
- **Reinforcement Learning:** in this category, an algorithm also called an agent interacts with an environment. The ultimate goal of the agent is to find out the strategy that maximized a specific metric.

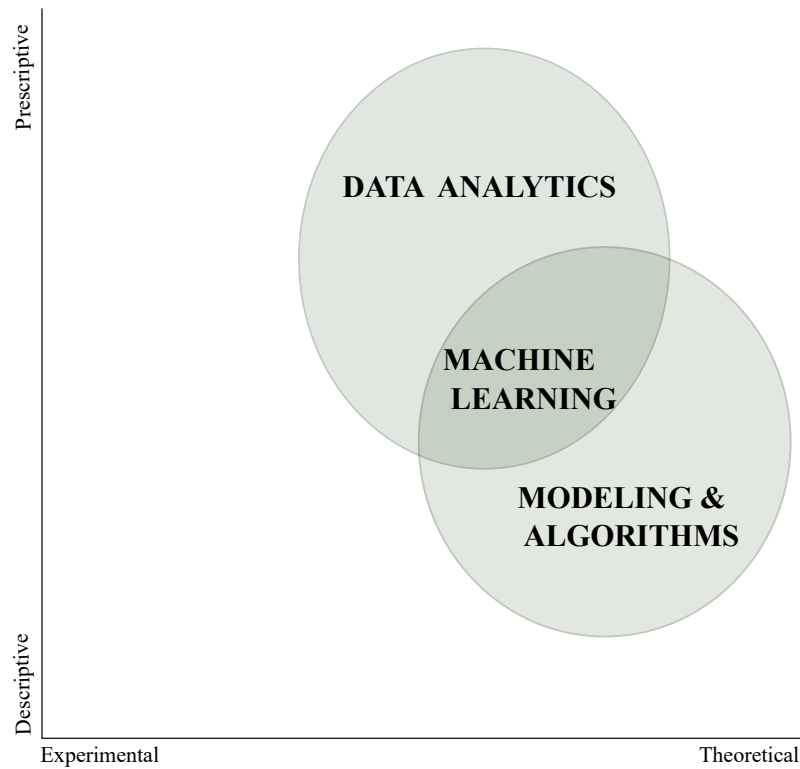


FIGURE 1.1: This figure visually illustrates that ML lies in the intersection of Data Analytics with Modeling & Algorithms.

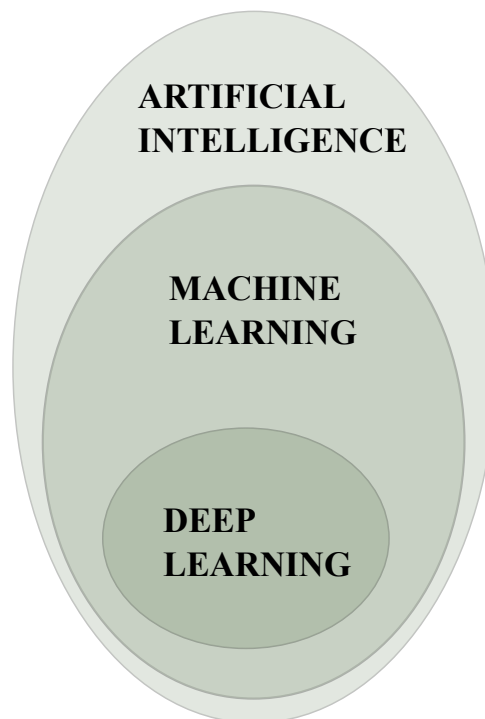


FIGURE 1.2: DL is a subfield of ML, which in turn is a subfield of AI.

1.3 Thesis Contribution

This study aims to take advantage of neuronal cells and networks' properties to reduce the complexity and improve the performance of SOMs. Drawing inspiration from the structure and properties of dendrites, we introduce a SOM-based architecture that models the dendritic nonlinearities and permits local, parallel computations. The DendSOM model consists of a single layer of SOMs. Each SOM is treated as a set of dendrites, and its units are connected to a single soma. In fact, the SOMs model different sub-regions of the input space, and the somata estimate the association between the best-matching units and class labels. We test our model on unsupervised classification and continual learning scenarios and show that DendSOM is capable of alleviating the effects of catastrophic forgetting.

The robustness analysis has been conducted using both the original SOM model and the RSOM model, which is a variation of the SOM algorithm also introduced and implemented in this study, as references in order to investigate the DendSOM's strengths and weaknesses by measuring the accuracy and time complexity on publicly available datasets such as the MNIST, MNIST-FASHION, and CIFAR-10 dataset. Moreover, this thesis presents a framework for continual learning with an emphasis on classification in order to establish a common ground and thus enhance fair comparisons of algorithms. What is more, this thesis reviews both the most widely applied techniques which have managed to alleviate the effects of catastrophic forgetting and the state-of-the-art algorithms in continual learning. It should be noted that the SOM DendSOM and RSOM algorithms are implemented in python using the CuPy, which is a Cuda-enabled version of NumPy.

1.4 Thesis Outline

The rest of this thesis is organized as follows:

- **Chapter 2 - Continual Learning:** The theoretical background and the state-of-the algorithms are presented in this section
- **Chapter 3 - Self-Organizing Maps:** The essential background and the related work in the field of Self-Organizing Maps as described
- **Chapter 4 - Dendritic Self-Organizing Maps:** The Dendritic Self-Organizing Map architecture is described and experiments are conducted
- **Chapter 5 - Conclusions and Future Work:** thesis conclusion and future work directions are provided

Chapter 2

Continual Learning

AI algorithms thrive in solving particular tasks. Over the last few years, several ML systems have been reported to exhibit and sometimes surpass human-level performance on demanding tasks such as Atari Games, Image & Object Recognition, and Speech Generation & Recognition [3, 4, 5, 6]. Although these results are remarkable, they are obtained with architectures and algorithms that lack the ability to adapt their behavior over time, also called static models. Hence, costly re-training is required when new data becomes available.

In our dynamic world, the practice of re-training from scratch our models becomes intractable for data streams, or it may be available only for a short period of time since, in the real world, we have to take into account both time and storage constraints as well as privacy issues [7]. Each day millions of data are produced. These new data may provide new pieces of information regarding new topics and trends. Therefore it is imperative to develop algorithms and deploy systems that perform life-long learning and thus can adapt continually and keep on learning over time.

Human cognition is an example of a system that tends to learn concepts sequentially, one after another. While revisiting some old concepts by observing new examples may occur in real life, it is not crucial for preserving assimilated knowledge and, as a result, for conceiving new concepts and ideas since new knowledge does not interfere with consolidated knowledge. Although we have the inclination to gradually forget old information, a complete loss of previously learned information is infrequently attested, especially in healthful people, due to the ability of biological neural networks to adapt in response to experience, possibly by leveraging complex molecular machinery.

In stark contrast, ANNs are incapable of performing continual learning since they suffer from catastrophic forgetting of old concepts as new concepts emerge. In general, ML algorithms learn data that are sampled from stationary distributions randomly, which makes several application scenarios impossible to solve. However, there are various ways to circumvent this problem. For example, AI research is focused mostly on static tasks. Each task is usually consisted of shuffled data to ensure i.i.d. conditions. Moreover, some techniques incorporate memory or generative replay for revising old concepts while multiple epochs are permitted per task, which allows for a vast performance increase.

2.1 Chapter Outline

In this chapter, we provide the essential background for continual learning. The rest of this chapter is organized as follows:

- **Section 2.2:** In this section, we provide the definition of continual learning along with some historical background
- **Section 2.3:** The basic terminology is provided, and the challenges that a CL system has to overcome are described
- **Section 2.4:** A framework for continual learning with an emphasis on classification is introduced
- **Section 2.5:** The most commonly used methods are discussed
- **Section 2.6:** The state-of-the-art algorithms are revisited

2.2 Definition of Continual Learning

Continual Learning (CL), also called lifelong learning, sequential learning, or incremental learning, is a branch of ML, and it studies the problem of learning from practically an infinite stream of data where data are not available all at once. Thus the ultimate goal of a continual learning algorithm is to gradually extend the acquired knowledge and utilize it for future learning. A problem can be characterized as a CL problem if and only if the learning process is sequential, and consequently, only data from one or a few tasks are available at once. CL algorithms usually deal with numerous data-related issues such as **distribution shifts** [8] and **catastrophic forgetting** [9].

2.2.1 Historical Background

Ever since the birth of the idea of AI, the concept of learning sequentially from experience has been present. The lifelong learning paradigm exists for several years within both the robotics and reinforcement learning (RL) research community [10, 11]. However, the field of CL has begun to be investigated more systematically only recently. During the last 30 years, the AI community has made significant progress towards the incorporation of this topic into the traditional ML approaches (supervised, unsupervised, and reinforcement learning). However, CL research has never been extensively conducted until recent years, despite some pioneering attempts in the late 90s. It can be argued that CL did not become so popular within the scientific community in the past years since there were more fundamental problems to be solved.

- **Limited Computational Power:** Like several DL techniques, CL algorithms have high computational complexity, and thus, executing them on Central Processing Units (CPU) is the least efficient solution due to low parallelism and high power consumption. However, the progress in the field of Graphics Processing Units (GPUs) in recent years has helped CL research advance.

- **Limited Amounts Data:** Data is the blueprint of innovation; Big Data has proven to be an asset to both tech and non-tech setups since it provides a powerful tool utilized for fact-based decisions. The explosion of big data has brought about advancements to ML and consequently to CL. However, before the big data revolution, collecting data was a laborious task
- **Handcrafted features:** Before the recent advances in the field of representation learning, the dominant approach towards solving AI problems, especially computer vision tasks, included handcrafted feature and task-specific solutions. In fact, these solutions may differ significantly depending on the task or domain. For many years, the development of general algorithms was considered practically impossible.
- **The success of ML in classification:** The success of ML algorithms in classification tasks at the time has aroused the interest of the scientific community, which has been systematically involved in the development of such algorithms. Although this situation ultimately led to the development of the first CL algorithms, CL was out of the AI research scope for several years.

2.3 Continual Learning Terminology & Challenges

In this section, we provide some basic definitions, and we describe the challenges that can arise when the samples are not independent and identically distributed (*i.i.d.*), and the data distribution is not static. The definitions used in this Thesis are those which are commonly used in the community, as described in [12, 13, 14]. However, CL lacks a theoretical framework, and thus most of these terms are not directly defined, whereas some terms appear to have multiple meanings in the bibliography. Hence the terms below are redefined in this Thesis. All definitions are detailed here for readability purposes.

2.3.1 Basic Definitions

Definition 1. Problem: Describes what the algorithms is employed to perform, e.g., classification problem.

Definition 2. Learning Objective: The learning objective corresponds to the minimization of a loss function based on data.

Definition 3. Task: A task is characterized by a task label, and it corresponds to a learning sub-objective. In the case of classification, a task may consist of more than one class. (Figure 2.1)

Definition 4. Task Label: It is a variable, usually an integer, that indicated the task boundaries.

Definition 5. Continuum: It is composed of a sequence of tasks, and thus it represents the complete experience.

Definition 6 Data Distribution: Is the statistical distribution from which the data are sampled. It may be constant or dynamic, and it depends on both the task and available data.

Definition 7. Input Distribution: Concerns the theoretical distribution of the input data.

Definition 8. Data Stream: refers to the flux of samples generated by the data distribution. In fact, a task is a set of the data stream.

Definition 9. Forgetting: performance decrease on previous tasks when learning a new one.

Definition 10. Catastrophic Forgetting: refers to the propensity of a model to forget all its previously learned tasks when re-training it on an unseen task.

Definition 11. Interference: is a conflict between two or more learning sub-objectives which leads to predictive power decrease.

Definition 12. Concept Drift: It characterizes the learning objective variations, and it is usually caused by data distribution changes. It should be noted that concept drift causes forgetting.

Definition 13. Catastrophic Interference: Similar to interference, indicating that the learning process of a new task has caused catastrophic forgetting.

Definition 14. Online Learning: is a special case of CL per sample, and thus the batch size equals one.

Definition 15. Joint Training: It corresponds to training in multiple tasks when all data are available.






<i>Split-MNIST</i>				
<i>Task 1</i>	<i>Task 2</i>	<i>Task 3</i>	<i>Task 4</i>	<i>Task 5</i>
				
$\begin{matrix} 0 & 1 \\ (0) & (1) \end{matrix}$	$\begin{matrix} 0 & 1 \\ (2) & (3) \end{matrix}$	$\begin{matrix} 0 & 1 \\ (4) & (5) \end{matrix}$	$\begin{matrix} 0 & 1 \\ (6) & (7) \end{matrix}$	$\begin{matrix} 0 & 1 \\ (8) & (9) \end{matrix}$

FIGURE 2.1: This figure illustrates the notion of the task.

2.3.2 Challenges

Catastrophic Forgetting

As mentioned in **Definition 10**, catastrophic forgetting refers to a model's inability to retain the knowledge acquired from previous concepts when trained incrementally on learning new ones. The phenomenon of catastrophic forgetting in CL algorithms can be detected as a performance decrease on learned tasks when new tasks emerge. Although some studies refer to catastrophic forgetting as catastrophic interference [15], in this work, we argue that the phenomenon of catastrophic interference is the cause of catastrophic forgetting. This is evident from **Definition 12**.

It should be noted that the following figure is from a paper titled "Attention-based selective plasticity"[16].

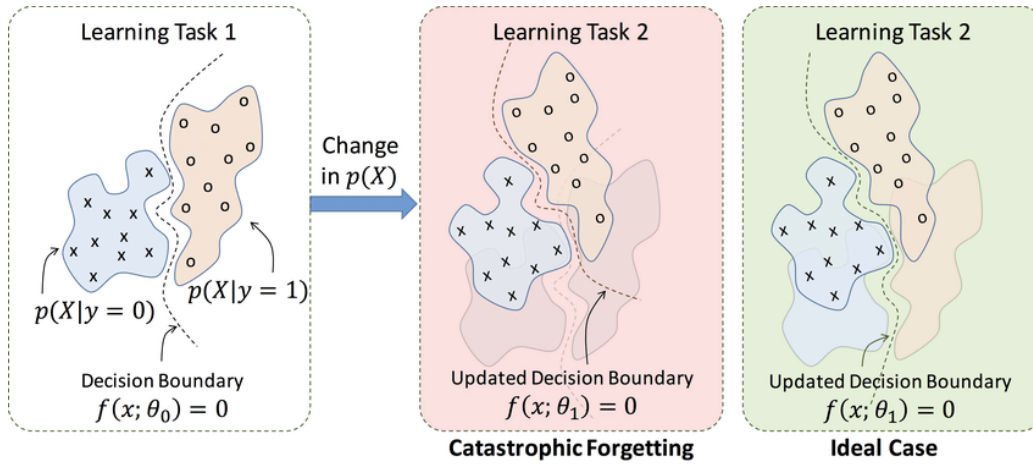


FIGURE 2.2: This figure illustrates the notion of catastrophic forgetting

Memory Management

Retaining knowledge is crucial for avoiding catastrophic forgetting during training in CL settings. Hence, many algorithms utilize memory mechanisms to store information about past tasks. Although memories can be saved in different manners, a memory management system should only save valuable information and transfer knowledge to future tasks to be efficient. The trade-off problem, also called the plasticity-stability problem, states that an algorithm should balance the information saved and forgotten. On top of that, a memory-based algorithm has to assess the already stored information since learning new tasks may lead to the degradation of memories.

Concept Drift

Distribution Shifts are caused by distribution changes that may lead to catastrophic interference and consequently to catastrophic forgetting. Hence the model has to both detect and adapt to these changes in order to alleviate the effects of distributional shifts. The concept drift is related to the Bayesian surprise concept, which measures how an observer can be affected by data in terms of differences between posterior and prior beliefs about the world [17]. In fact, there are two types of concept drifts:

- **Virtual Concept Drift** concerns only the input distribution, and it occurs due to imbalanced classes.
- **The Real Concept Drift** can be detected only by its effect since it is caused by novelty on data.

It should be noted that task changes can cause distributional shifts too.

The following figure is from a survey titled "A survey on classification of concept drift with stream data" [18].

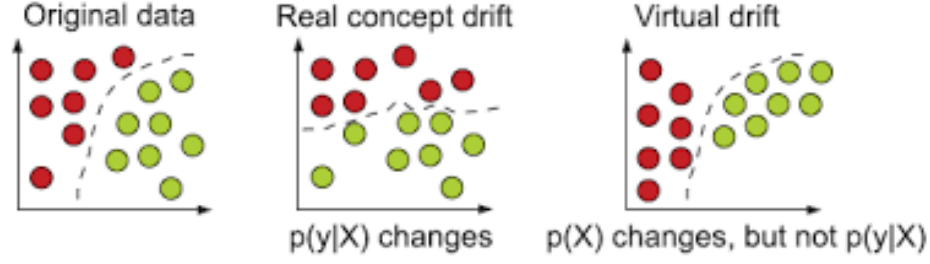


FIGURE 2.3: This figure illustrates the notion of concept drift

2.4 A Continual Learning framework with emphasis on classification

The field of CL has gained significant popularity over the last few years. CL has been the subject of numerous surveys. However, most of these studies are empirical[19], while others do not even provide comparative results [20]. Although some effort has been devoted regarding the formalization of CL in robotics [21], CL lacks a formalization with an emphasis on classification. We argue that a common framework for training and testing CL models is crucial to enhance fair comparisons of techniques and thus assist the field in advancing further.

2.4.1 Definitions

Definition 16. Training Data

Suppose that $D = \{D_1, ..D_n\}$ is a sequence of distributions over $X \times Y$ where X and Y are the input and the output random variables, respectively. At time i the training data, which consist of (x, y) or (i, x, y) observations, are sampled from the distribution D_i and provided to the model. It should be noted that the sequence of distributions D is potentially infinite.

Definition 17. CL Algorithm

Given f as a target function, t as the task label and M as external memory, a continual learning algorithm CLA can be properly defined as:

$$\forall D_i \in D, \quad CLA : \langle f_{i-1}, X_i, Y_i, M_{i-1}, t_i \rangle \rightarrow \langle f_i, M_i \rangle$$

In short, at each time i , (X_i, Y_i) pairs are sampled from the distribution D_i . The objective of the ACL is to utilize these observations and the previously stored information in memory to update both the target function and external memory. It is worth to be mentioned that in the above case, the algorithm can employ the task label information t_i to disentangle the tasks. In the case that the task information is not given the signature becomes:

$$\forall D_i \in D, \quad CLA : \langle f_{i-1}, X_i, Y_i, M_{i-1} \rangle \rightarrow \langle f_i, M_i \rangle$$

On top of that some algorithms do not utilize external memory and consequently we have:

$$\forall D_i \in D, \text{ CLA } :< f_{i-1}, X_i, Y_i > \rightarrow < f_i >$$

Definition 18. CL Scenarios

A scenario is a specific setting in which the tasks follow a specific organizational scheme

2.4.2 Continual Learning Scenarios

Although several machine learning scenarios permit models to operate on the entire dataset, in CL, data arrive incrementally as subsets of samples. According to [22], the CL problems can be categorized into three main scenarios. Using the dataset depicted in figure 2.1 as an example, we summarize the objective of each scenario in the following table:

Continual Learning Scenarios	
Scenario	Objective
TASK-IL	Is it 0 or 1? (Task given)
DOMAIN-IL	Is it 0 or 1? (Task unknown)
CLASS-IL	Which digit is it? (0-9)

TABLE 2.1: This table summarizes the objective of each scenario

Task-Incremental Learning

Task-Incremental Learning is the easiest continual learning scenario since the model can utilize task-related information. The dataset is permuted into non-overlapping tasks, and during training, only data from the current task is available.

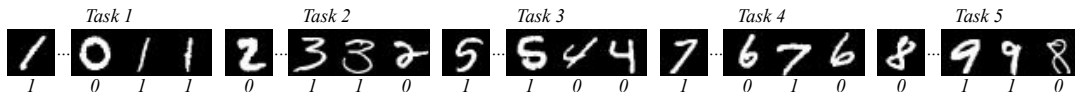


FIGURE 2.4: Illustration of the task-incremental learning scenario on the MNIST dataset

Domain-Incremental Learning

Domain-Incremental Learning is similar to Task-Incremental Learning except that no task information can be accessed by the model. Thus, a given set $D = \{x_i, y_i\}_{i=1...2}$, the model has to solve the task at hand.

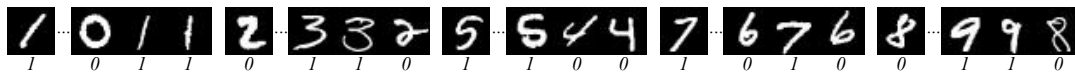


FIGURE 2.5: Illustration of the domain-incremental learning scenario on the MNIST dataset

Class-Incremental Learning

The goal of class-incremental learning is to learn, given a dataset $D = \{x_i, y_i\}_{i=1\dots n}$, a unified classifier. Similar to Domain-Incremental Learning, the model can not utilize task information in this scenario.

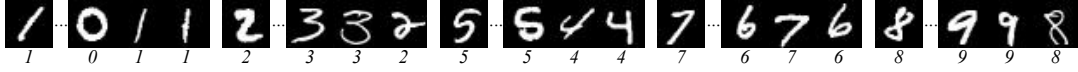


FIGURE 2.6: Illustration of the class-incremental learning scenario on the MNIST dataset

2.4.3 Datasets

SVHN

The SVHN dataset consists of 630420 32 by 32 RGB images, 604388 for training, and 26032 for testing. The images in total correspond to 10 classes, one per digit, and each image is centered around a single digit. It is worth noting that this version of the DVHN dataset is used for continual learning, while there is one more version that is usually used for object recognition [23].



FIGURE 2.7: Classes of the SVHN dataset

CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. The training set consists of 50000 samples while there are 10000 test images [24].

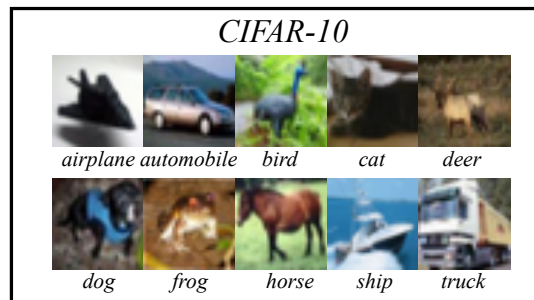


FIGURE 2.8: Classes of the CIFAR-10 dataset

CIFAR-100

This dataset is similar to CIFAR-10. However, it consists of 100 classes organized into 20 categories. Furthermore, there are 500 training, and 100 testing samples for each class [24].

MNIST

The MNIST dataset contains images of handwritten digits and consists of a training set with 60000 samples and a test set with 10000 samples. Each image is centered and size-normalized to 28×28 pixels with 256 gray levels and is associated with a label from 10 classes (0-9) [25].

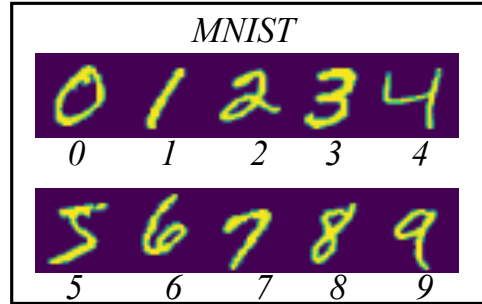


FIGURE 2.9: Classes of the MNIST dataset

2.4.4 Protocols

CL aims to develop algorithms that can acquire knowledge over time. To fairly evaluate a CL method, we need to perform training and testing on several tasks. Hence it is imperative to utilize CL protocols in order to create sequences of tasks.

Permutation

Given a dataset, the permutation protocol generated N task by applying N different permutations to the original dataset

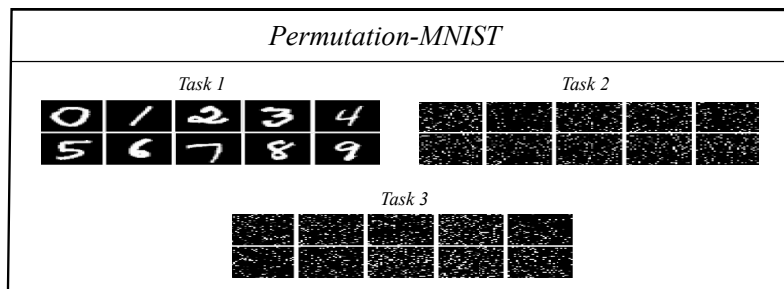


FIGURE 2.10: Illustration of permuted MNIST task protocol

Split

The split protocol operates on the entire dataset by splitting the dataset into clearly separated tasks. The Split MNIST is a typical dataset that is utilized for benchmarking continual learning methods. It was introduced in a five-task form where the ten classes split into five binary classification problems. (See Figure 2.1

Rotation

Given a dataset, the rotation protocol generated N task by rotating the original dataset by multiples of $\frac{360}{N}$ degrees

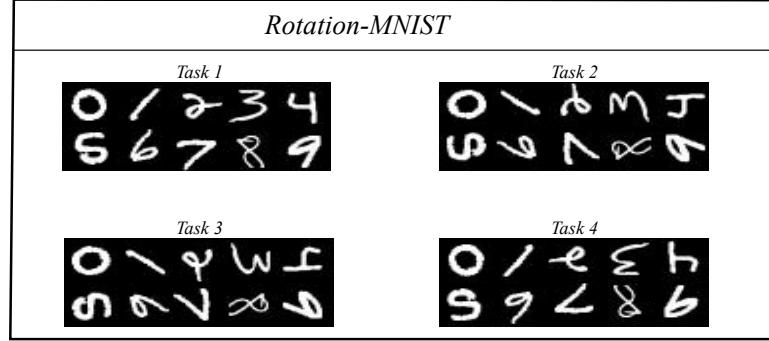


FIGURE 2.11: Illustration of rotation MNIST task protocol

2.4.5 Evaluation Metrics

Evaluating an algorithm is of great significance for building effective AI systems. CL is an emerging field, and thus there are no established evaluation metrics that measure the different aspects of CL. Hence, in this work, we focus on two essential yet straightforward measures:

Accuracy Score

Accuracy is the simple ratio between the number of correctly classified points to the total number of points. We argue that the accuracy score can provide valuable insights about the robustness of the examined models due to lack of data imbalances on the training sets

$$accuracy(v_{pred}, v_{true}) = \frac{\#(v_{pred} == v_{true})}{\#v_{true}}$$

Forgetting Rate We propose an evaluation metric in order to measure the effects of catastrophic forgetting. The forgetting rate is inversely proportional to the ratio of CL accuracy and joint training accuracy (See **Definition 15**).

$$fr = 1 - \frac{accuracy_{CL}}{accuracy_{JT}} \quad (2.1)$$

2.4.6 Baselines

The baselines are important since they serve as performance bounds. In this work, we utilized a deep neural network sequentially trained on all tasks, which can be seen as the lower bound. Furthermore, the same architecture was trained using the data of all tasks, which can be seen as the upper bound.

The essentials of Deep Neural Networks (DNNs)

A neural network is a brain-inspired algorithm that consists of input, output, and hidden layers. Each layer, in turn, consists of neurons which are the computational units of the model. The layers in the case of an ANN are fully connected, which means that all the layer neurons are connected to all neurons of the next one. Each connection maintains a weight value, though the adjustment of those weight values, the neural network can approximate a plethora of functions hence the term universal function approximator.

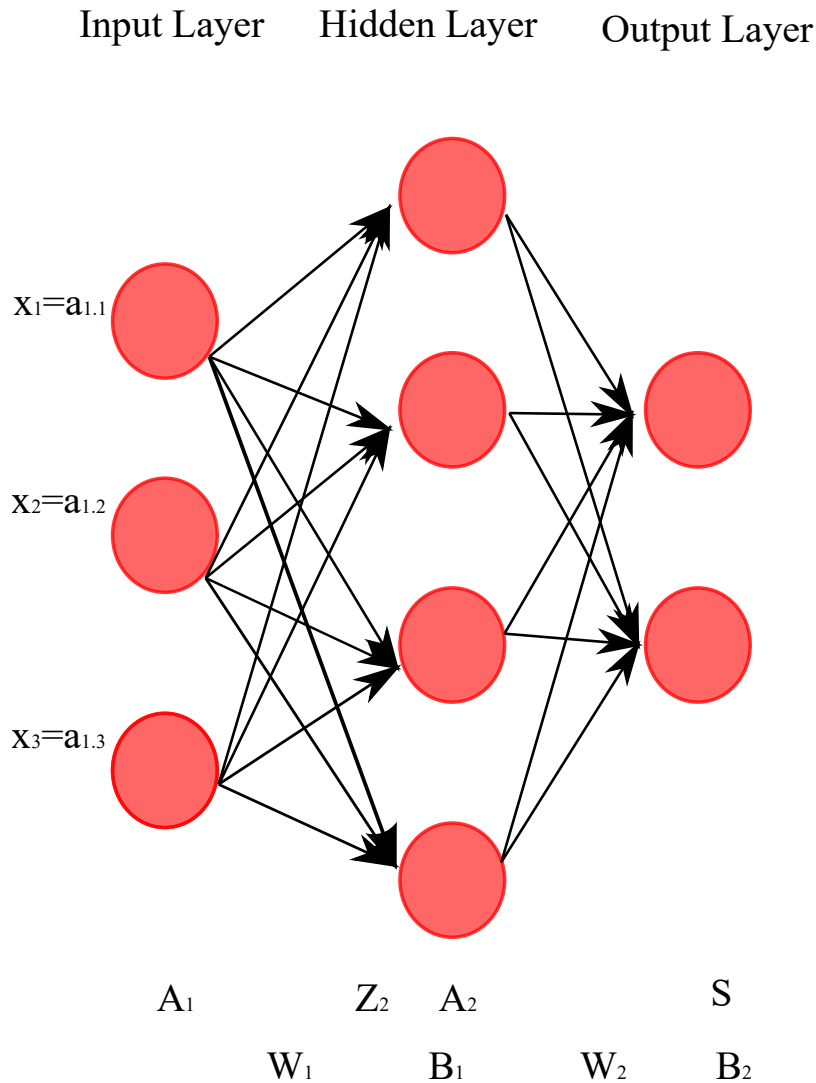


FIGURE 2.12: Illustration of a simple neural network with one hidden-layer

Forward Propagation

During forward propagation, the data is fed to the neural network, and each layer receives, processes, and passes it to the next one.

– Input Layer

The neurons in the input layer represent the input data.

$$x_i = a_{1,i}, \quad i \in 1, 2, 3$$

– Hidden Layer

Each layer applies a linear transformation to the output of the previous layer. Thus the vector Z is produced which in turn is used for the calculation of the activation vector A . It should be noted that the activation function is non-linear.

$$Z_2 = W_1^T A_1 + B_1$$

$$A_2 = \text{activation}(Z_2)$$

– Output Layer

This layer applies a linear transformation to the activation vector of the previous layer. The output vector is produced as follows:

$$S = W_2^T A_2 + B_2$$

Finally we compute a function that evaluates the output vector S against the properly encoded label y . This function is also called cost function and the objective of the neural network is to minimize it

$$C = \text{cost}(S, y)$$

Backward Propagation

As mentioned above the objective of a neural network is to find the values of W, B minimize the cost function and consequently the difference between the actual and the predicted vectors. Hence, the calculation of the gradient of the cost function w.r.t both W and B is needed.

– Gradient Calculation

For batch size equals 1, the gradients can be calculated as follows:

$$\frac{dC}{dW_l} = \frac{dC}{dZ_l} \frac{dZ_l}{dW_l} = \frac{dC}{dZ_l} A_{l-1}^T$$

$$\frac{dC}{dB_l} = \frac{dC}{dZ_l} \frac{dZ_l}{dB_l} = \frac{dC}{dZ_l}$$

For batch size equal to m , the following equation can be used for gradient calculation

$$\frac{dC}{dW_l} = \frac{1}{m} \sum_{i=1}^m \frac{dC_i}{dW_l}$$

$$\frac{dC}{dB_l} = \frac{1}{m} \sum_{i=1}^m \frac{dC_i}{dB_l}$$

– **Parameter Update**

The gradient points to the direction of the greatest increase, and consequently, following the gradient's direction is not a wise choice since it leads to cost function maximization. The solution is to adjust the network's parameters by utilizing the anti0gradient instead. Hence the update rule can be formalized as follows:

$$B_l^{new} = B_l^{old} - \frac{dC}{dB_l^{old}}$$

$$W_l^{new} = W_l^{old} - \frac{dC}{dW_l^{old}}$$

2.5 Methods for addressing catastrophic forgetting

The recent advances in the field of digital design have assisted DL in receiving increased attention. In fact, the use of powerful GPUs has allowed training on vast datasets and thus revealed the remarkable predictive abilities of the DNNs. The advent of DL algorithms has brought about advancements in the field of CL too. Nowadays, AI researchers address CL problems with many tasks of numerous samples each. Hence, several techniques have been adopted to alleviate the effects of catastrophic forgetting. Based on how the information can be stored and used for feature learning and inferencing, these methods can be categorized into three broad categories. In this section, we describe the most common methods that are employed in CL.

2.5.1 Replay

Replay methods aim to defy catastrophic forgetting by relaying memories from previous tasks in order to revisit old concepts when learning new ones. The technique of storing previously encountered examples for rehearsal is used since 1990. In fact, the method of experience replay is widely applied in the field of RL, where the agents have to interact with non-stationary environments and learn to perform the actions that maximize a reward function. According to the replay strategy that they employ, the replay based-algorithms are characterized as follows:

Rehearsal: The rehearsal methods save raw or preprocessed samples of already learned tasks in memory.

Pseudo-Rehearsal: The pseudo-rehearsal methods produce pseudo-samples instead of saving already seen examples. In both the rehearsal and pseudo-rehearsal replay methods, the memories are usually mixed with the current data.

Constrained: Although rehearsal methods tend to overfit in the saved samples, constrained replay methods can circumvent this effect by introducing additional constraints in the update process to prevent the interference between the new and the old tasks

2.5.2 Regularization

This method introduces a regularization term to the loss function of the learner to consolidate the previous knowledge when learning from new data. Thus, the regularization-based algorithms do not make use of external memory systems or generated samples to retain previous concepts, and consequently, they are both memory efficient and privacy-preserving. The regularization approach can also be divided into two main categories:

Prior-focused: The prior-focused regularization-based algorithms utilize the model parameters to estimate the prior distribution when learning from new data. The parameters of the model are assumed to be independent since the estimation of the prior distribution becomes infeasible due to the vast number of parameters in DNNs. On top of that, an importance metric is calculated for each parameter of the model, and changes in important parameters are penalized during the training in new data.

Data-focused: The data-focused regularization approach is based on the concept of knowledge distillation [26]. The goal of the distillation techniques is to transfer the knowledge acquired from one neural network (NN) to another. In fact, if a NN A can solve a specific task, then A's ability can be transferred to a NN B by forwarding the same input to both networks and imposing B to mimic the output of network A. Knowledge distillation is practically less costly than re-training the network B on the entire dataset since it produces a soft target. Thus it is easier for B to distinguish the new from the previously learned tasks.

2.5.3 Parameter Isolation

As the name indicates, the parameter isolation-based algorithms dedicate different subsets of the model parameters to different tasks. Hence the model tends to learn a distinct representation for each concept, and consequently, the catastrophic interference is prevented to some extent, which in turn limits the effects of any possible forgetting of the previous tasks. Once again, according to the size constraints of the architecture, the parameter isolation-based algorithms are categorized as follows:

Fixed Architecture: The fixed architecture method aims to identify the most significant parts of the architecture for solving the old task and mask them out during the training of a new task. Consequently, utilizing this method leads to learning a different mask and thus different features for each task. The masking can be imposed on both the parameters and neurons. However, the task boundaries have to be known for this method to work properly.

Dynamic Architecture: In the case of dynamic architecture parameter isolation methods where there are no constraints on the size of the architecture, the learning process is straightforward. These methods proceed by identifying the parts of the network that solve the previous tasks and freeze them. On top of that, as the name suggests, the architecture is dynamic, and consequently, it can grow new branches when needed. However, much like the fixed architecture method, the task boundaries have to be known for this architecture to achieve the highest possible performance.

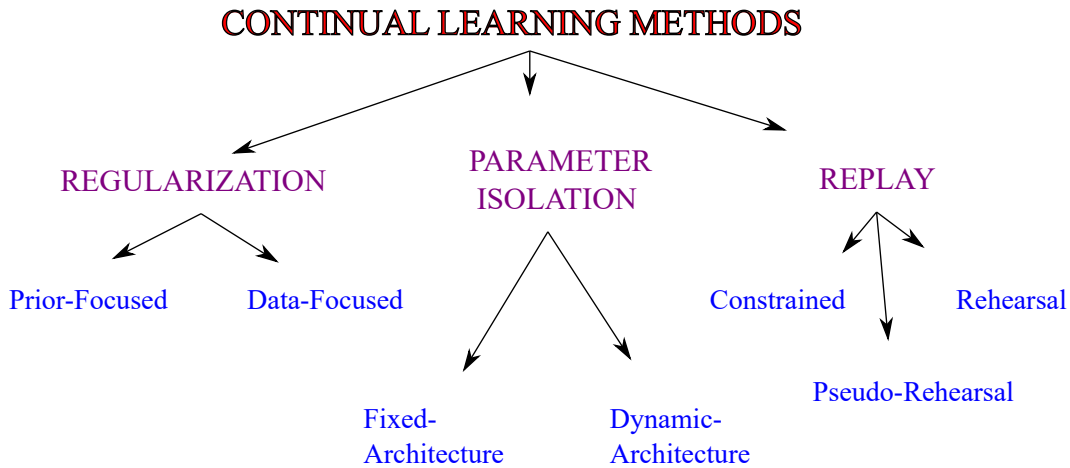


FIGURE 2.13: Illustration of a tree diagram that depicts the types of methods used in continual learning

2.6 State-of-the-art algorithms

In this section, we present some algorithms that can be classified as state-of-the-art in CL. These algorithms fall into at least one of the aforementioned families of methods and learn either task-specific or task-free features. For each algorithm, we provide both a detailed but concise description as well as the pseudocode needed for implementation reasons. It is worth to be mentioned that in chapter 4, these algorithms will be compared with our proposed architecture in each of the three CL scenarios discussed above. Furthermore, the accuracy score on benchmark datasets such as the Split-MNIST and Split-CIFAR-10 will be used as an evaluation metric.

2.6.1 Copy Weight with Reinit (CWR)

This algorithm belongs to the family of dynamic-architecture parameter-isolation methods, and it is considered to be a baseline technique for learning from sequential batches in each CL scenario. When the task label is given, the algorithm freezes the shared weights $\bar{\Theta}$ after the completion of a task and extends the output layer with new randomly initialized neurons for the new one. However, in the case of both domain-incremental and class-incremental scenarios where the environment can not provide information about the task boundaries due to the unavailability of the task label, the algorithm still follows the same procedure, but this time, the architecture is extended for each training batch [27]. As shown in the figure above, the algorithm maintains two sets of output weights the consolidated weights cw used for making predictions and the temporal weights tw utilized for learning while the shared weights remain frozen after learning the first batch.

Algorithm 1 CWR

```

1:  $cw = 0$ 
2: init  $\bar{\Theta}$  random or from a pre-trained model (e.g. ImageNet)
3: for each training batch  $B_i$ :
4:   expand output layer with  $s_i$  neurons for the new classes in  $B_i$ 
5:   random re-init  $tw$  (for all neurons in the output layer)
6:   Train the model with SGD on the  $s_i$  classes of  $B_i$ :
7:     if  $B_i = B_1$  learn both  $\bar{\Theta}$  and  $tw$ 
8:     else learn  $tw$  while keeping  $\bar{\Theta}$  fixed
9:   for each class  $j$  among the  $s_i$  classes in  $B_i$ :
10:     $cw[j] = w_i \cdot tw[j]$ 
11: Test the model by using  $\bar{\Theta}$  and  $cw$ 

```

FIGURE 2.14: This figure illustrates the pseudocode for the CWR algorithm

Copy Weight with Reinit + (CWR+)

The CWR+ algorithm [27] is an extension of the CWR algorithm. In fact, this technique improves upon the original one by introducing the following two simple modifications to the CWR approach:

Modification 1. Mean-Shift: Tuning the parameters w_i in the CWR algorithm is usually a tedious and laborious task, while wrongly tuned w_i can significantly decrease the model's performance. Hence, instead of that, this algorithm normalizes the tw learned at the i_{th} training step by subtracting the global mean of tw

Modification 2. Constant Initialization: It has been empirically proven that the random initialization of the training weights of the last layer can introduce significant bias during the first softmax normalization. Thus instead of random initialization, constant initialization is performed. In fact, the parameters tw are reinitialized to zero since the shared weights remain frozen during training.

Algorithm 2 CWR+

```

1:  $cw = 0$ 
2: init  $\bar{\Theta}$  random or from a pre-trained model (e.g. ImageNet)
3: for each training batch  $B_i$ :
4:   expand output layer with  $s_i$  neurons for the new classes in  $B_i$ 
5:    $tw = 0$  (for all neurons in the output layer)
6:   Train the model with SGD on the  $s_i$  classes of  $B_i$ :
7:     if  $B_i = B_1$  learn both  $\bar{\Theta}$  and  $tw$ 
8:     else learn  $tw$  while keeping  $\bar{\Theta}$  fixed
9:   for each class  $j$  among the  $s_i$  classes in  $B_i$ :
10:     $cw[j] = tw[j] - \text{avg}(tw)$ 
11: Test the model by using  $\bar{\Theta}$  and  $cw$ 

```

FIGURE 2.15: This figure illustrates the pseudocode for the CWR+ algorithm

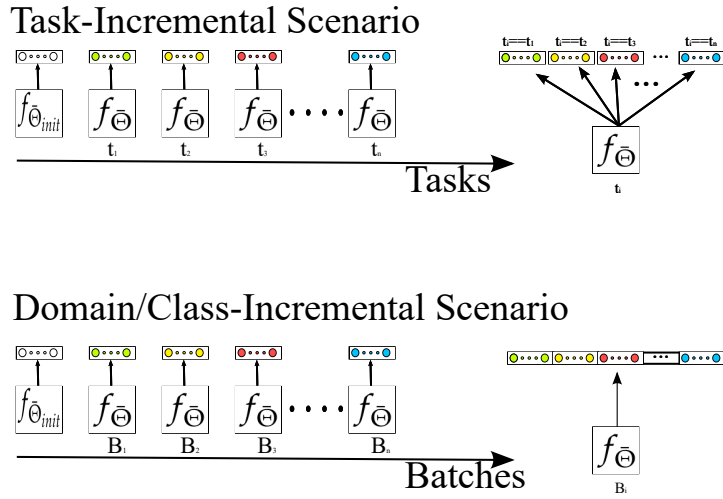


FIGURE 2.16: This figure illustrates the training process of CWR and CWR+ in task, class, and domain incremental scenarios

2.6.2 Deep Generative Replay (DGR)

This model is based on pseudo-rehearsal replay methods. The goal of the classifier is to learn a mapping from the input to the output space, while the generator aims to learn the data distribution in order to sample data from past experience. On the first batch of data, both the classifier and generator are ordinarily trained. However, when the next batch arrives, the generator produces the replay data, which are mixed with the training data of the current batch, and the models are trained in the new upsampled dataset. In this way, it can be ensured that the already learned knowledge is not forgotten [28].

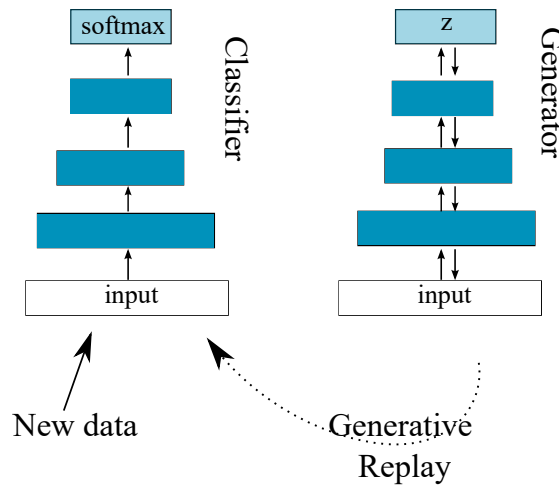


FIGURE 2.17: Illustration of a typical DGR architecture

2.6.3 Elastic Weight Consolidation (EWC)

EWC [29] is a prior-focused regularization-based algorithm since it penalizes updates on important weights. When the model is trained on a new task, the importance of the training parameters can be estimated by the diagonal of the Fisher information matrix F . The diagonal of this matrix corresponds to the curvature of the loss surface near a minimum. Hence, the i_{th} diagonal element F_i corresponds to the importance factor of parameter θ_i . It should be noted that important parameters must be moved as little as possible since a slight change can cause a dramatic increase in the loss function due to the high curvature of the loss surface towards their direction. For two sequential tasks A and B , the loss function of the algorithm is given from the following formula:

$$L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta_{A,i}^*)^2$$

- $L(\theta)$: loss function for task B
- θ_i : current θ_i parameter
- $\theta_{A,i}^*$: the optimal θ_i parameter for task A
- F_i : the i_{th} diagonal element of the Fisher information matrix (FIM)

When another task emerges, the algorithm will try to minimize the loss function of this task without significantly changing the learned parameters of the old tasks by either adding separate penalty functions or a single penalty for the old tasks.

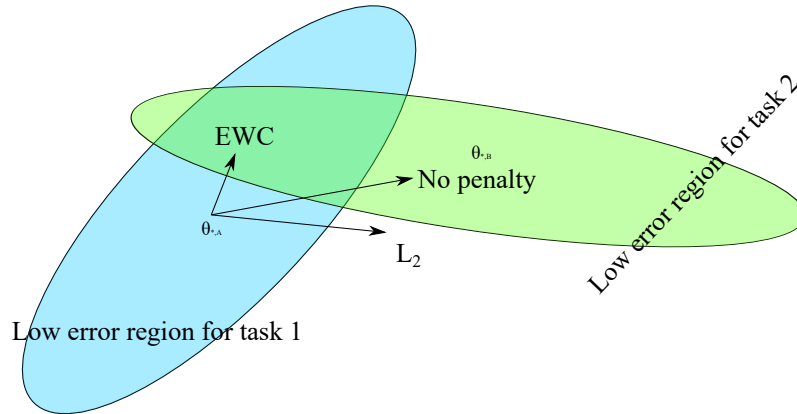


FIGURE 2.18: This figure illustrates the trajectories of the model's training parameters (a) when the EWC algorithm is employed (b) when L_2 regularization is utilized and (c) when no regularization is introduced

2.6.4 Learning Without Forgetting (LwF)

This algorithm is based on data-focused regularization techniques since it tries to address catastrophic forgetting by forcing predictive stability. Given a set of parameters, the ultimate goal of the LwF algorithm is to add task-specific parameters as well as learn parameters that work well on all tasks without using old data [30]. First, the responses \mathbf{y}_o of the algorithm are recorded, and thus, each sample is associated with a prediction vector. Next, the network is trained to minimize the loss for all tasks and the regularization term. Furthermore, the loss function for the new task is defined as follows:

$$L_{new}(\mathbf{y}_n, \hat{\mathbf{y}}_n) = -\mathbf{y}_n \log \hat{\mathbf{y}}_n$$

- \mathbf{y}_n : The one-hot encoded truth label vector
- $\hat{\mathbf{y}}_n$: The softmax prediction vector
- For **multiple new tasks**, the total loss equals to the sum of losses across new tasks
- For **multilabel classification**, the total loss equals to the sum of losses across all labels

On top of that, the loss function for each original task can be computed as follows:

$$L_{old}(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) = -\sum_{i=1}^l \mathbf{y}'_o^{(i)} \log \hat{\mathbf{y}}'_o^{(i)}$$

- l : number of labels
- $\hat{\mathbf{y}}'_o$: The modified version of current probabilities
- \mathbf{y}'_o : The modified version of the initially recorded probabilities
- For **multiple old tasks**, the total loss equals to the sum of losses across new tasks
- For **multilabel classification**, the total loss equals to the sum of losses across all labels

Finally the modified probabilities can be calculated using the following formulas:

$$\mathbf{y}'_o^{(i)} = \frac{(\mathbf{y}_o^{(i)})^{\frac{1}{T}}}{\sum_j (\mathbf{y}_o^{(j)})^{\frac{1}{T}}}, \quad \hat{\mathbf{y}}'_o^{(i)} = \frac{(\hat{\mathbf{y}}_o^{(i)})^{\frac{1}{T}}}{\sum_j (\hat{\mathbf{y}}_o^{(j)})^{\frac{1}{T}}}$$

It should be noted that the parameter $T > 1$ leads to better class similarity encoding [26].

```

LEARNINGWITHOUTFORGETTING:
Start with:
   $\theta_s$ : shared parameters
   $\theta_o$ : task specific parameters for each old task
   $X_n, Y_n$ : training data and ground truth on the new task
Initialize:
   $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$  // compute output of old tasks for new data
   $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$  // randomly initialize new parameters
Train:
  Define  $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$  // old task output
  Define  $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$  // new task output
   $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left( \lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$ 

```

FIGURE 2.19: This figure illustrates the pseudocode for the LwF algorithm

2.6.5 Synaptic Intelligence (SI)

This algorithm is an alternative to the EWC [31]), and thus it is based on the prior-focused regularization technique. In fact, SI differs from EWC in the sense that it computes the weight importance during Stochastic gradient descent (SGD), unlike EWC, which computes the FIM at the end of the training on the batch. During SGD, the loss change caused by the movement of the i_{th} weight can be calculated as follows:

$$\Delta L_i = \Delta \theta_i \frac{dL}{d\theta_i} = (\theta'_i - \theta_i) \frac{dL}{d\theta_i}$$

Hence the F_i parameter can be computed as follows:

$$F_i = \frac{1}{K_i^2 + \xi} \sum \Delta L_i$$

It is worth noting that the EWC can be easily turned into SI regularization since these algorithms differ only in terms of importance calculation.

2.6.6 Context-dependent Gating (XdG)

The XdG method is a neuro-inspired solution for CL. In fact, the algorithm aims to model the ability of the brain to switch between tasks and thus deal with catastrophic forgetting by allowing updates to occur in primarily non-overlapping sets of weights in order to minimize the catastrophic interference between tasks. This easy-to-train algorithm selects a random subset $X\%$ of the total number of units in each hidden layer for each task and sets their activation functions to zero. The hyperparameter X is usually selected through grid search, and as mentioned above, it controls the percentage of units that will be gated in each hidden layer. However, this method can only be utilized only for task-incremental learning problems since it requires a unique signal for each task [32].

2.6.7 Hybrid Algorithms

In this subsection, we describe some hybrid algorithms that aim to alleviate the effects of catastrophic forgetting by combining the merits of two or more techniques.

Deep Generative Replay and Regularization (DGR)

This method aims to combine generative replay with knowledge distillation. The key idea behind this hybrid algorithm is to utilize LwF for replaying input data generated by a deep generative model and associate those data to probability vectors or soft targets in general. Although this approach does not significantly improve the performance of the DGR algorithm, it can be considered a state-of-the-art approach [33].

Architectural and Regularization approach (AR1)

AR1 [27] is a hybrid algorithm that utilizes both parameter isolation and regularization. In fact, AR1 is similar to the CWR+ algorithm since it follows an almost identical training process. However, this hybrid algorithm employs SI to introduce the concept of parameter importance and thus tune the shared weights $\bar{\Theta}$ across the entire dataset.

Algorithm 3 AR1

```

1:  $cw = 0$ 
2: init  $\bar{\Theta}$  random or from a pre-trained model (e.g. ImageNet)
3:  $\Theta = 0$  ( $\Theta$  are the optimal shared weights resulting from the last training, see Section 2.3)
4:  $\hat{F} = 0$  ( $\hat{F}$  is the weight importance matrix, see Section 2.3).
5: for each training batch  $B_i$ :
6:   expand output layer with  $s_i$  neurons for the new classes in  $B_i$ 
7:    $tw = 0$  (for all neurons in the output layer)
8:   Train the model with SGD on the  $s_i$  classes of  $B_i$  by simultaneously:
9:     learn  $tw$  with no regularization
10:    learn  $\bar{\Theta}$  subject to SI regularization according to  $\hat{F}$  and  $\Theta$ 
11:    for each class  $j$  among the  $s_i$  classes in  $B_i$ :
12:       $cw[j] = tw[j] - \text{avg}(tw)$ 
13:     $\Theta = \bar{\Theta}$ 
14:  Update  $\hat{F}$  according to trajectories computed on  $B_i$  (see eq. 10 and 11)
15:  Test the model by using  $\bar{\theta}$  and  $cw$ 

```

FIGURE 2.20: This figure illustrates the pseudocode for the AR1 algorithm

Chapter 3

Self-Organizing Maps

In recent years, enormous amounts of data are created every day. Emails, social networking interactions, and online transactions are only some of the sources that produce data [34, 35]. It has been estimated that almost 5 exabytes of data are created every two days. Thus, it is not a surprise that Big Data and data analytics are at the epicenter of modern science and business in the age of digitalization and technological progress. Actually, data may contain patterns and valuable pieces of information that can be extracted and consequently incorporated into decision-making processes.

However, every analytical task that invokes large datasets requires an information organization process. Distinguishing common patterns and their relationships in the data can provide valuable insights about the structure and thus the organization of data. Clustering belongs to the family of unsupervised ML algorithms that aim to discover the structure of a dataset by grouping data points so that similar samples belong to the same group. Statistical ML methods aim to identify the cluster structure of a high-dimensional dataset and then utilize this structure in order to recognize the patterns that will allow for accurate and persistent predictions for unseen data points.

The SOM [36] is an effective algorithm in producing similarity graphs of input data, considering it converts the statistical relationships between high-dimensional data points to geometric relationships on a space of lower dimensions, usually 2D. On top of that, the SOM, in its basic form, can also be very useful for high-dimensional data visualization since it can compress information while preserving the topological characteristics, at least the most important, of the entire dataset. Hence it is widely applied in several domains such as machine perception, data analytics, and bioinformatics.

Since its introduction by Kohonen, SOM is one of the most commonly used neural network models. The SOM is an unsupervised learning algorithm with a simple structure and relatively low computational complexity compared with the number of data that it processes. Initially developed as associative and inspired by both the retina-cortex mapping and brain maps, the SOM aims to cause neighboring parts of the map to have similar responses to specific input patterns. In general, self-organization is a key process of learning both inter-pattern associations as well as intra-pattern relationships among the stimuli and responses. It should be noted that the process above does not require external influence for discovering the appropriate patterns.

3.1 Chapter Outline

In this chapter we provide the essential background for self-organizing maps. The rest of this chapter is organized as follows:

- **Section 3.2:** In this section we briefly describe the history of the SOM algorithm
- **Section 3.3:** The notion of vector quantization is discussed
- **Section 3.4:** The essential background of SOM is presented
- **Section 3.5:** Theoretical analysis is conducted in the case of 1D SOMs
- **Section 3.6:** Theoretical analysis is conducted in the case of multi-dimensional SOMs
- **Section 3.7:** The stochasticity of the online algorithm is analysed
- **Section 3.8:** The Batch-SOM algorithm is presented
- **Section 3.9:** Variations and extensions of the original SOM algorithm are presented

3.2 Historical Background

In 1982 Teuvo Kohonen introduced a special type of ANN, the model, and ever since, it has encountered large success because it is a simple-to-develop and easy-to-train algorithm that achieves remarkable performance on data clustering as well as it provides a method for producing visualizing the nonlinear relationships between multi-dimensional data in a meaningful way. The basic version of SOM is partly inspired by the way that sensory (e.g., visual, auditory, olfactory) information is handled in the cerebral cortex of humans, and the learning algorithm can be modeled as an online stochastic process.

In 1986 the somatosensory mapping property of SOM was illustrated [37] and consequently was once again affirmed that the SOM learning algorithm was motivated by neuro-biological paradigms in which the learning process is unsupervised and regulated by the experience. For a long time, the algorithm was used only for modeling neural and biological processes in general. However, it slowly started to be applied in a plethora of other fields [38, 39] such as economics, robotics, and natural language processing (NLP) due to its remarkable computational capabilities [40].

Over the last years, numerous variants and extensions of this algorithm have been developed. In fact, the deterministic batch SOM [41] algorithm was proposed for industrial applications where the reproducibility of the results is a crucial factor. On top of that, the original algorithm supports only real-valued vectorial data. Thus the algorithm had to be redefined several times in order to support complex non-vectorial, categorical data, and documents, as well as similarity and dissimilarity indexes [42, 39].

3.3 Quantization

According to [Wikipedia](#) quantization is a mathematical process that maps values from a large, usually continuous, set to values in a smaller set with a finite number of elements. Typical quantization examples include rounding and truncation; thus, quantization is a key step in every lossy compression algorithm. This mapping technique is adopted in the field of digital signal processing since it can be utilized for constructing digital representations of analog signals. However, quantization is a one-way process since the same quantized value may represent a large subset of numbers and not just an input value.

Vector Quantization (VQ)

VQ [\[43\]](#) is the process of partitioning vectorial input data into regions optimally represented by a codebook vector. Thus VQ describes a pattern set using a reduced number of codebook vectors. Consequently, an optimal VQ has been achieved when the regions are partitioned such that the mean distance of an input data point from the best-matching codebook vector is minimized. VQ is widely applied in the field of pattern recognition since it has low computational complexity compared to other models, such as the hidden Markov models (HMMs). However, the classical VQ technique does not account for temporal changes in signals. Thus, a multi-section approach was introduced to address this problem by creating a different codebook for each section of an input signal [\[44\]](#).

For an input d -dimensional pattern $\mathbf{x} \in \mathbb{R}^d$ and the codebook vectors that represent the partitioned regions \mathbf{c} we can find the best-matching codebook vector using the Euclidean distance such as:

$$bm\mathbf{v} = \arg \min_i \{ \|\mathbf{x} - \mathbf{c}_i\|^2 \}$$

Suppose that the probability density of \mathbf{x} is denoted as $p(\mathbf{x})$, then the mean quantization error over the data space V is an energy function E which is defined as:

$$E = \int_V \|\mathbf{x} - \mathbf{c}_{bm\mathbf{v}}\|^2 p(\mathbf{x}) dV$$

Although the above function can be minimized, it may converge in a local minimum due to its high nonlinearity [\[45\]](#). It should be mentioned that a batch method for VQ also exists, which is called k-means clustering [\[46, 47\]](#).

K-means

It is the most used hard-clustering algorithm [\[48\]](#). In each step the probability that an input pattern \mathbf{x} belongs to the region represented by the codebook \mathbf{c}_j is calculated and denoted as $P(j|X)$. Then for each \mathbf{x} in the input space the codebook is updated iteratively such as the best-matching codebook for \mathbf{x} is moved closer to it. The update rule is defined as:

$$c_{bm\mathbf{v}}(t+1) = c_{bm\mathbf{v}}(t) + lrP(bm\mathbf{v}|X)(\mathbf{x} - \mathbf{c}_j(t))$$

- lr : The learning rate
- $c_{bmV}(t)$: The best-matching codebook vector before the update
- $c_{bmV}(t + 1)$: The updated best-matching codebook vector

There is also a batch version of k-means in which each c_j is moved towards the center of its assigned patterns. Although the k-means algorithm can produce results that can be reproduced if trained on the same dataset with the same initial code vectors, it is prone to end at a local minimum, and thus, the initial choice of code vectors can significantly affect the success of the algorithm

Soft-Clustering

In this type of clustering, [49], the codebooks compete for one input pattern x , the probability that the input x belongs to the j_{th} code vector is given by the formula:

$$P(j|x) = \frac{\exp(-\beta||x - c_j||^2)}{\sum_j \exp(-\beta||x - c_j||^2)} \quad (3.1)$$

The above equation indicates that the probability $P(j|x)$ is a Gaussian normalized function. The parameter β controls the code vector's range of influence such that for $\beta \rightarrow \infty$, the algorithm becomes hard-clustering. Furthermore, $\sum_j P(j|x) = 1$ and the energy function E can be calculated as shown above. In fact, the soft-clustering approach can be implemented as an annealing process that can be utilized for avoiding convergence to local minima [50]

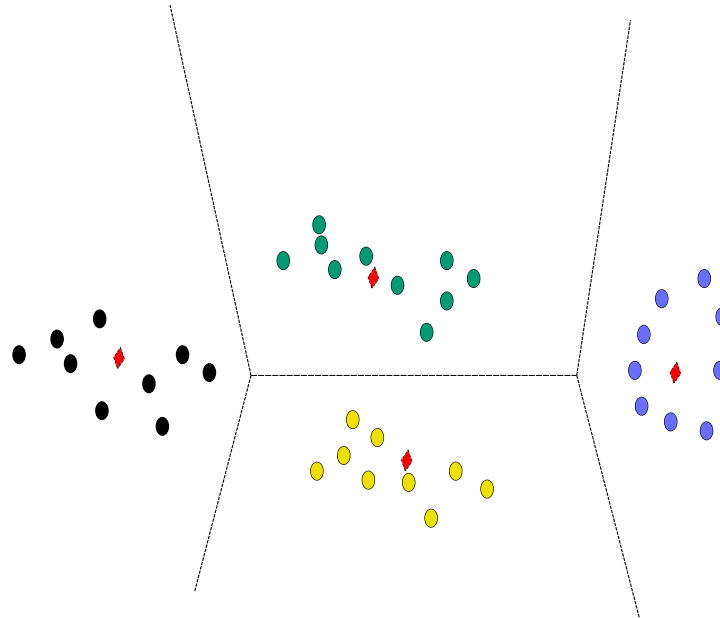


FIGURE 3.1: Illustration of the Voronoi diagram (permutation boundaries) and the codebook vectors (diamonds) for the input training data (circles)

Hard vs Soft Clustering	
Type	Definition
Hard-Clustering	A data point belongs completely to a cluster
Soft-Clustering	A data point can belong to several cluster with a likelihood

TABLE 3.1: Hard vs Soft clustering

3.4 Fundamentals of Self-Organizing Maps

The SOM is an unsupervised projection mapping algorithm similar to VQ in the sense that both aim to find the optimal codebook vectors or units in the case of SOMs. However, The units of a SOM are also spatially globally ordered. These units are represented as nodes on a 2D grid. The SOM algorithm applies competitive learning to construct the map's topology in such a way that the more similar the input data, the shorter their distance on the map [51]. From a theoretical standpoint, we can distinguish two different training settings based on the nature of the input space:

- **discrete setting:** the data space consists of N data points x_i , $i = 1...N$ with $x_i \in \mathbb{R}^d$
- **continuous setting:** The data space can be modeled by a probability density function (PDF) g

3.4.1 Online SOM Learning Algorithm

The SOM consists of a grid of units. Each unit i has a predetermined position \mathbf{p}_i on the grid, where $\mathbf{p}_i \in \mathbb{N}^{+m}$ and m is the grid's dimensions. Furthermore, the i -th unit maintains a weight vector $\mathbf{w}_i \in \mathbb{R}^k$ where k is the number of dimensions of the input vector \mathbf{x}_n with $n = 1, 2, 3...number_of_samples$. It is worth noting that the weight vectors are initialized to small random values.

The SOM is trained iteratively, and at each iteration t , the learning process performs the following steps:

1. Select a sample vector \mathbf{x} randomly (discrete setting) or according to the PDF g (continuous setting) from the input data space \mathbf{X}
2. Identify the index of the Best-Matching Unit (BMU) that satisfies:

$$BMU = \arg \min_i ||\mathbf{x} - \mathbf{w}_i||^2 \quad (3.2)$$

3. Update each weight vector i using the following equation:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + a(t)h_{i,BMU}(t) \left(\mathbf{x} - \mathbf{w}_i(t) \right) \quad (3.3)$$

The learning rate $a(t)$ at training step t is calculated by:

$$a(t) = a_0 \exp \left(-\frac{t}{\lambda} \right) \quad (3.4)$$

$$0 < a_0 < 1$$

Moreover, the neighborhood function $h_{i,BMU}(t)$ at iteration t is calculated as follows:

$$h_{i,BMU}(t) = \exp \left(- \frac{\|\mathbf{p}_i - \mathbf{p}_{BMU}\|^2}{2\sigma(t)} \right) \quad (3.5)$$

where $\sigma(t)$ corresponds to the neighborhood radius at step t and it can be calculated by:

$$\sigma(t) = \sigma_0 \exp \left(- \frac{t}{\lambda} \right) \quad (3.6)$$

During the update step, the BMU and its topological neighbors are moved closer to the input vector. It can also be observed that $a(t)$, $\sigma(t)$ and $h_{i,BMU}(t)$ decrease exponentially over time. Furthermore, a_0 and σ_0 are the initial values of the learning rate and neighborhood radius, respectively. Finally, λ is a hyperparameter whose value controls the decrease rate for both $\sigma(t)$ and $a(t)$.

The above algorithm is simple to develop, easy-to-train (see Figure 3.3) and provides a useful tool for high-dimensional data visualization. However, apart from some empirical results, the theoretical properties have not been entirely proved despite a large number of works and experimental results. In fact, each w_i is a stochastic process in \mathbb{R}^d . Thus for a large number of training steps t , the weight vectors can display different behaviors. Hence, the following questions have to be addressed from a theoretical point of view:

- Have the algorithm approximately converged to a distribution when $t \rightarrow \infty$?
- What happens when $a(t) = a_0 \quad \forall t$?
- What happens when $a(t)$ decreases?
- Is the limit state stable if it exists?
- Can we characterize the organization and how?

The main results, as well as the remaining open problems, can be found in these papers [52, 53].

3.4.2 Methods for analyzing SOM convergence

In this section, we present the mathematical tools that have been utilized to analyze the convergence of stochastic processes and consequently of SOM

Markov Chain Theory (MCT)

Markov chain is a stochastic model [54, 55], and thus it can model the randomness of an environment in which the next state depends only on the current state. The MCT has been applied for studying the convergence and the limit states of a SOM with constant learning rate and neighborhood function. If the algorithm converges in distribution, this distribution, also called limit distribution, has to be an invariant measure for the Markov Chain. It should be noted that for a Markov Chain on a state space M with transition matrix P and the function $f : m \rightarrow \mathbb{R}$ such that $f(i) = \lambda_i, \quad \forall i \in M$. The function f is a neasyre ib M if $f(i) = \lambda_i \geq 0, \quad \forall i \in M$. Furthermore, a measure function f is invariant if $f(i)P = f(i) \leftrightarrow \lambda_i P = \lambda_i$. Thus for every state $j \in M$ it should hold that $f(j) = \sum_{i \in M} f(i)p_{ij} \leftrightarrow \lambda_j = \sum_{i \in M} \lambda_i p_{ij}$

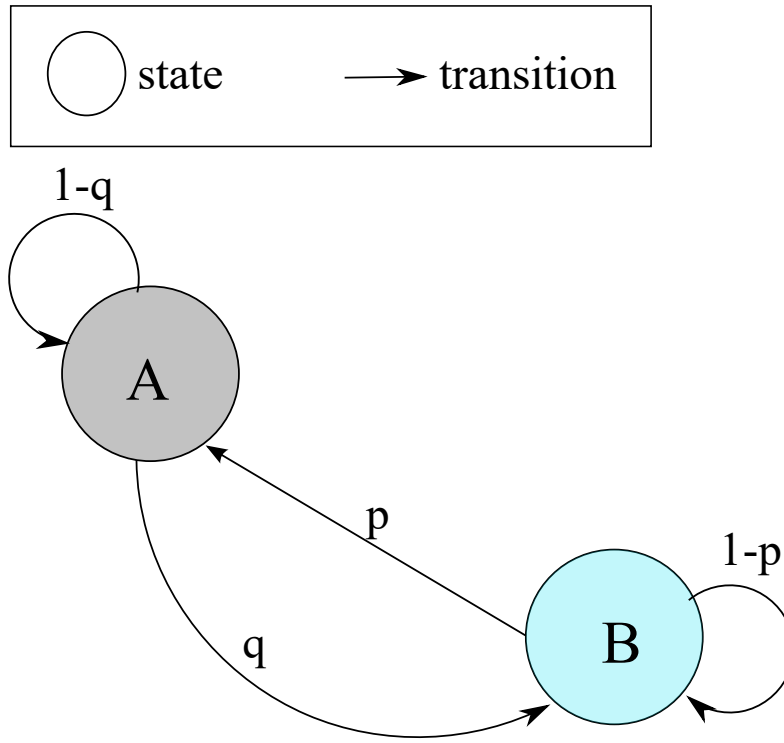


FIGURE 3.2: Illustration of a simple Markov chain with two states and probabilistic transitions

Ordinary Differential Equation (ODE)

To employ the ODE method, the SOM update equation (Equation 3.3) has to be rewritten in vector form as follows:

$$w(t+1) = w(t) + a(t)\Phi(x, w(t)) \quad (3.7)$$

- $w(t)$: SOM's weight vectors before the update
- $w(t+1)$: The updated SOM's weight vectors
- $\Phi(x, w(t))$: a stochastic term

The next step involves the study of the solutions of the deterministic ODE that describes the mean behavior of the process.

$$\frac{dw}{dt} = -\phi(w) \quad (3.8)$$

- $\frac{dw}{dt}$: The derivative of w w.r.t. t
- $\phi(w)$: In the case of continuous setting, it corresponds to the expectation of $\Phi(., w)$ w.r.t. the probability distribution of the input space, while in discrete setting it is the expectation of $\Phi(., w)$ w.r.t. the arithmetic mean

Thus, the m_{th} component of ϕ for the continuous setting can be calculated as follows:

$$\phi_m(w) = \sum_{j=1}^N h_{mj} \int_{C_j} (x - w_m) g(x) dx \quad (3.9)$$

- **N**: The number of SOM's units
- **h_{mj}** : The value of the neighborhood function between the j_{th} and m_{th} unit

Similarly, the m_{th} component of ϕ for the discrete setting can be calculated as follows:

$$\phi_m(w) = \frac{1}{K} \sum_{j=1}^N h_{mj} \sum_{x_i \in C_j} (x_i - w_k) \quad (3.10)$$

- **K**: The number of the input samples
- **C**: The space of BMUs

Finally, the possible limit states of the stochastic process have to satisfy the following equation:

$$\phi(w) = 0 \quad (3.11)$$

It is worth noting that if the solutions of this function are also the minima. Thus gradient descent can be applied in order to find the zeros of the above equation.

Robbins-Monro algorithm theory

This approach is useful for root finding problems where the function can be represented as an expected value. Thus, this algorithm can only be utilized if the learning rate decreases under the following conditions:

$$\begin{aligned} \sum_t a(t) &= +\infty \\ \sum_t a(t)^2 &< +\infty \end{aligned} \quad (3.12)$$

Although some progress has been made in analyzing the SOM convergence, the original SOM algorithm is still difficult to be studied. First of all, an organized state is challenging to be defined for high-dimensional input data and maps. Secondly, it has been proved that the update rule (Equation 3.7) is not an energy function [56], and thus gradient descent can not be utilized for finding the roots of ϕ in the continuous setting. Finally, all the approaches assume that the neighborhood function is of fixed intensity which is not the case for the original SOM algorithm.

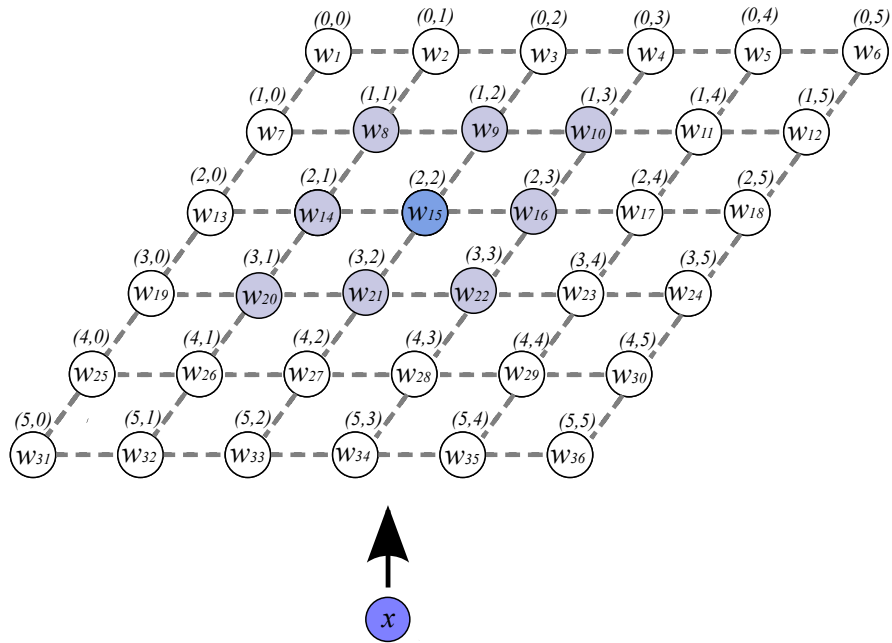


FIGURE 3.3: Illustration of the SOM architecture (grid with nodes w) and the input vector (node x). The weight vector in position $(2, 2)$ as selected as the BMU of the input

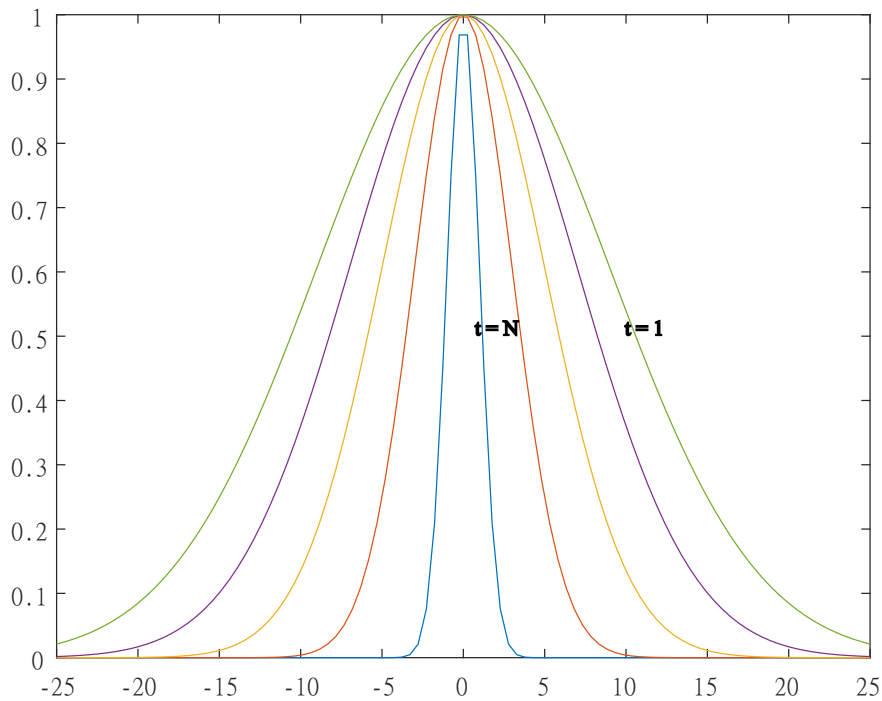


FIGURE 3.4: This figure illustrates the behavior of the neighborhood function h w.r.t. training step t

3.5 Theoretical analysis of the 1D SOM case

Although the theoretical analysis of the one-dimensional case, where the map is a string, and the inputs are scalars, can provide valuable insights about the behavior of the algorithm, the 1D SOM has almost zero practical utility. It should be noted that the 1D SOM algorithm has been extensively analyzed in the past [57, 52, 53]. However, the 1D SOM case is analyzed in a more concise way in this section (see Figure 3.5).

3.5.1 A special case

In this case, the map is one-dimensional and of N units, the input data are sampled from a uniform distribution in $[0, 1]$. Furthermore the learning rate is constant a_0 with $a_0 < \frac{1}{2}$ and the neighborhood function is constant and it can be defined as follows:

$$h_{bmu,i} = \begin{cases} 0, & |bmu - i| > 1 \\ 1, & \text{otherwise} \end{cases} \quad (3.13)$$

If the $w(t)$ update process is a homogeneous (transitions independent of t) Markov Chain with a continuous state space, the organization can be characterized by the ordering of the units. It should be noted that this case has been extensively studied [57]. The core steps of the proof are described below.

1. The number of wrongly ordered triplets has to decrease strictly to sufficiently prove the convergence.
2. The set of ordered dispositions has to be an absorbing state (a state from which there is no escape) of a Markov Chain.
3. Then it can be shown that the topology preservation takes place after a finite time with a probability higher than a positive bound and consequently that the arrival to the absorbing state is finite.
4. Thus, there exists an integer A and a positive constant c such that if the process starts from an ordered state. the probability to enter in a set $E \in [0, 1]^n$ with less than A transitions is smaller than $c \cdot \text{volume}(E)$.
5. The step above indicates that the process converges to a monotonous stationary distribution that depends on a_0 .
6. If $a(t) \rightarrow 0$, the Markov Chain almost surely converges towards a monotonous solution of an explicit linear system if the Robbins-Monro conditions are satisfied.

Proof: $\text{volume}(E) = \frac{1}{n!}$ where $E = \{x = (x_1, \dots, x_n) \in \mathbb{R}^n : 0 \leq x_1 \leq \dots \leq x_n \leq 1\}$:

The statement above can be easily proven as follows:

For $n = 1$ the set represents a unit line segment and thus $V_1 = 1$

For $n = 2$ the set represents a triangle with unit height and E_1 as base so, $V_2 = \frac{1}{2}V_1 = \frac{1}{2}$

For $n > 1$ the set is a simplex and consequently $V_m = \frac{1}{n}V_{n-1} = \frac{1}{n} \frac{1}{n-1} \dots \frac{1}{2} 1 = \frac{1}{n!}$

Therefore, in the case of 1D SOM for $t \rightarrow \infty$ both the convergence to a monotonous and unique ordered solution can be proved.

$$\begin{aligned} w_1(+\infty) &< \dots < w_N(+\infty) \\ w_1(+\infty) &> \dots > w_N(+\infty) \end{aligned} \quad (3.14)$$

However, when the dimension number increases, it becomes impossible to discover absorbing states and consequently a stable and monotonous unit ordering (see Figure 3.6).

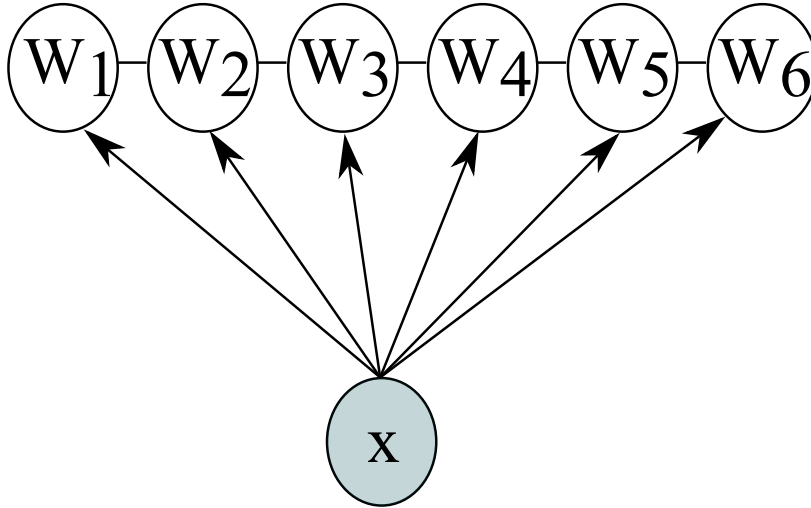


FIGURE 3.5: Illustration of the 1D SOM

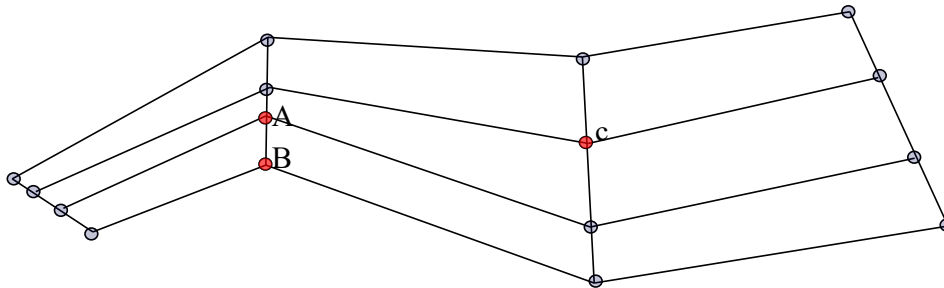


FIGURE 3.6: Illustration of a 2D SOM. In this case, the units are well ordered and connected as depicted. Hence A and B are neighboring units, the same holds for A and C . However, B is not a neighbor of C . Thus if C is repeatedly chosen as the BMU, B is never updated, and consequently, its grid position does not change. On the other hand, A comes closer to C with every update, and as a result, the starting disposition is disordered

3.5.2 The general 1D SOM convergence case

The general case invokes relaxation of the hypothesis on the data distribution and the neighborhood function. In this section, the conclusions that can be drawn for the general 1D case are presented. It should be noted that more detailed discussions on those results can be found in these papers. [52, 53]

- For constant $a(t)$, the order time is usually finite
- The random process $w(t)$ usually converges to a stable and unique equilibrium point if $a(t)$ satisfies the Robbins-Monro conditions
- If the stable equilibrium is unique, for constant $a(t) = a_0$ exists an invariant probability measure m^{a_0} from any ordered state. When $a_0 \rightarrow 0$, this measure becomes a Dirac measure, and the equilibrium state
- If the stable equilibrium is unique, the algorithm converges to this equilibrium from any ordered state provided that it satisfies the Robbins-Monro conditions

Although the 1D case for constant rates is well-known, there is nothing proven about the choice of a proper decreasing function for $a(t)$ that ensures both ordering and convergence. The same holds for the neighborhood function.

3.6 Theoretical analysis of the multi-dimensional case

Although the multi-dimensional case is difficult to be studied theoretically, some partial results have been proved over the last years [56, 58, 59, 60, 61]. In this section, some of the results presented in those studies are discussed.

3.6.1 Continuous setting

In the multi-dimensional case, it has been shown that for a neighborhood function of a finite range, a constant learning rate and a positive probability density function (pdf), at least on an interval. The learning algorithm weakly converges to a unique probability distribution that depends on a_0 . However, the topological preservation properties of this stationary distribution remain unproven due to the difficulty of defining absorbing states in the multi-dimensional continuous setting. Determining the complexity of the problem is also difficult since some contradictory results hold. For instance, if there is a 2D map and a set S that contains ordered units (x,y coordinates), it holds that:

- Given a constant learning rate and a pdf, the process will hit S almost surely in finite time [58]
- The exit time is almost surely finite in the case of 8-neighbor setting [59]

3.6.2 Discrete setting

When the neighborhood function is independent of the training step, the stochastic process derives from a function whose values are positive near positive charges, negative near negative charges, and in general, tend to zero at infinity. The potential functions are associated with energy and thus the configuration of latent variables and the configuration of inputs provided in an example. The update rule is shown below:

$$\begin{aligned} w_u(t+1) &= w_u(t) + a(t)h_{u,bmu}(t)(x - w_u(t)) \\ &= w_u(t) - a(t)\Phi_u(x, w(t)) \\ &= w_u(t) - a(t)\frac{d}{dw_u}E(x, w(t)) \end{aligned}$$

where $E(x, w)$ is a sample function of $E(W)$

$$E(w) = \frac{1}{2N} \sum_{u_1=1}^U \sum_{u_2=1}^U h_{u_1, u_2} \sum_{x_i \in C_j} ||w_{u_1} - x_i||^2$$

or

$$E(w) = \frac{1}{2N} \sum_{i=1}^N \sum_{u=1}^U h_{u, bmu_{x_i}} ||w_u - x_i||^2$$

Hence, the stochastic gradient descent process on $E(w)$ is associated with the update stochastic process $w(t)$. Thus the following conclusions can be drawn:

1. The energy function is associated with the Simple Competitive Learning process, also called VQ algorithm, as a generalization of the intra-classes variance function. The SCL process is a SOM with the following neighborhood function:

$$h_{bmu, i} = \begin{cases} 1, & |bmu - i| = 0 \\ 0, & otherwise \end{cases}$$

Using the above h function the energy function R can be further reduced to:

$$E(w) = \frac{1}{2N} \sum_{i=1}^N ||w_{bmu} - x_i||^2$$

2. The gradient of the energy function is not continuous, and the general hypotheses utilized to prove that the SGD process converges are not valid due to the fact that there are discontinuities on the cluster boundaries since the neighbors involved in the computation change from time to time. Although this process does not ensure convergence, it provides valuable insight into the process behavior.
3. For 0-neighbors, the VQ converges since there are no neighbors, and thus the gradient is continuous. However, the process will most probably converge to one of the local minima since there are a lot of local minima.

3.7 The stochastic nature of the online SOM training algorithm

The stochasticity of the online SOM training algorithm is a result of two main factors:

- random weight vector initialization
- random input vector choice at step t

Although there are deterministic SOM training algorithms, the online algorithm can provide interesting insights into the input data's significance. In fact, assuming that input data that are close in the input space belong to the same or adjacent units of a SOM can be valuable for interpreting the SOM result. However, given observations that have been matched to identical or adjacent units of the map, the hypothesis that these observations belong to the same partition of the input space may not hold. The phenomenon of wrongly mapped samples happens due to the fact that there is no perfect mapping between multi-dimensional and low-dimensional data, especially for one or two dimensions.

Given a pair of input samples x_i, x_j the following three cases can be distinguished (see Figure 3.7):

significant association: The pair is matched to the same or neighboring units because x_i and x_j are close in the input space. Thus the observations are called mutually attracted

significant non-association: The pair is never matched to the same or neighboring units due to the fact that x_i and x_j are not close enough in the input space. Thus the observations are called mutually repulsed

fickle pairs: The pair is sometimes matched to the same or neighboring units, but x_i and x_j are not close enough in the input space. In this case, the proximity of the map is due to randomness

The stochasticity of the SOM can produce results that can not be easily reproduced. However, this stochasticity can also be utilized for quantifying the behavior of every pair of observations by running the algorithm several times; This idea was first introduced in 2002 [62], while in 2015 [63, 64], it was employed for text mining. In fact, the idea at its core is quite simple. Running the algorithm several times will allow for identifying the behavior of every pair of data since the algorithm is stochastic. Hence, if R denotes the total number of runs, and T_{ij}^R is the total number of times that the observations x_i, x_j are classified to neighboring units on the map in the R runs, the **stability index** M_{ij} for these observations can be defined as follows:

$$M_{ij} = \frac{T_{ij}^R}{R}$$

If the data are matched to neighboring units by chance, utilizing an approximation of the binomial distribution with a confidence interval of 5% on a map with N units, the following measures are introduced:

$$A = \frac{9}{R}, \quad B = 1.96\sqrt{\frac{9}{RN}\left(1 - \frac{9}{N}\right)}$$

Hence, the relationships between pairs of input data can be quantified as indicated by the following decision rule:

- if $M_{ij} > A + B$ the association between x_i and x_j is significantly frequent
- if $M_{ij} < A - B$ the non-association between x_i and x_j is significantly frequent
- if $A + B < M_{ij} < A - B$ the non-association between x_i and x_j is due to randomness

The fickle pairs can be utilized in several ways, such as:

- Improving the robustness of a map by distinguishing mutually attracted and repulsive pairs from fickle pairs.
- Removing fickle pairs to improve visualization systems
- In text mining, the removal of flicked words can result in a simplified graph of co-occurrences between words

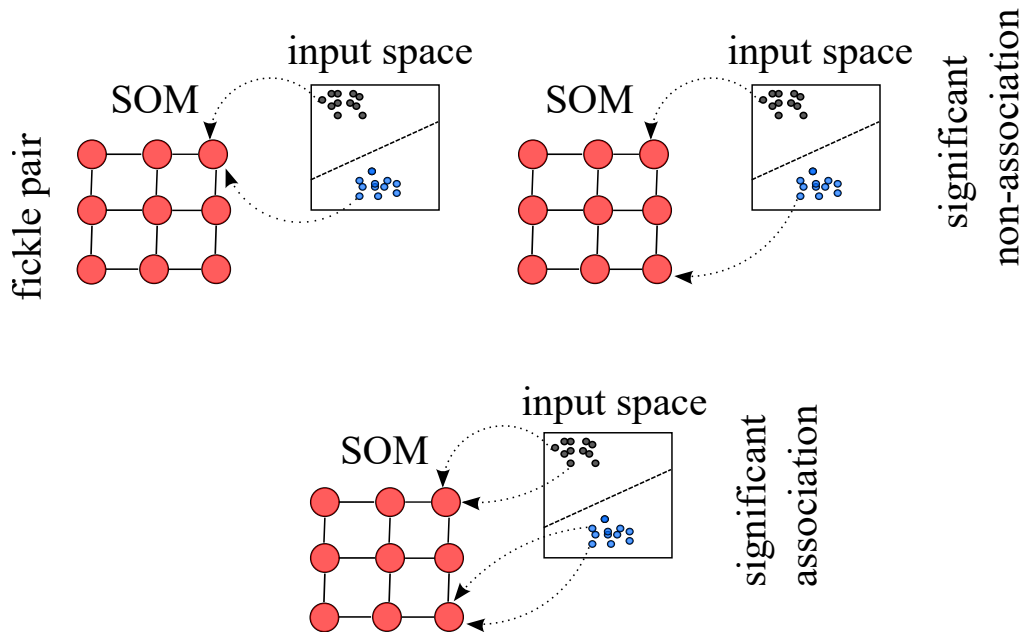


FIGURE 3.7: This figure illustrates the different mapping cases for pairs of input data

3.8 Batch-SOM: A deterministic approach

As mentioned in section 3.4, the possible limit states of the weight update stochastic process $w(t)$ may possibly be the solutions of the ODE equation [65]:

$$\phi(w) = 0$$

Hence, an important question arises: How can someone get the solutions directly? The answer to this question is hidden in the definition of the Batch-SOM algorithm.

As shown previously the k_{th} component of ϕ in the continuous setting is:

$$\phi_k(w) = \sum_{j=1}^K h_{kj} \int_{C_j} (x - w_k) g(x) dx$$

Thus, the continuous setting solution w_k^* can be calculated as follows:

$$w_k^* = \frac{\sum_{j=1}^K h_{kj} \int_{C_j} x g(x) dx}{\sum_{j=1}^K h_{kj} \int_{C_j} g(x) dx}$$

Similarly, the discrete setting solution w_k^* can be calculated as follows:

$$w_k^* = \frac{\sum_{j=1}^K h_{kj} \sum_{x_i \in C_j} x_i}{\sum_{j=1}^K h_{kj} |C_j|} = \frac{\sum_{i=1}^N h_{k,bmu} x_i}{\sum_{i=1}^N h_{k,bmu}}$$

As shown in the above equation, w_k^* is equal to the weighted means of all the inputs which belong to the cluster C_k or its neighboring clusters. The weights are equal to the neighborhood function values.

Thus, the definition of the batch-SOM algorithm can be derived in continuous setting as follows:

$$w_k(t+1) = \frac{\sum_{j=1}^K h_{kj} \int_{C_{j(w_k(t))}} x g(x) dx}{\sum_{j=1}^K h_{kj} \int_{C_{j(w_k(t))}} g(x) dx}$$

Furthermore, the the batch-SOM algorithm can be derived in discrete setting as follows:

$$w_k(t+1) = \frac{\sum_{i=1}^N h_{k,bmu}(t) x_i}{\sum_{i=1}^N h_{k,bmu}(t)}$$

This algorithm is deterministic since the limit states of the code vectors depend only on the initial values of the weight vectors

3.9 SOM variations

The SOM algorithm has been widely applied in numerous fields such as water resources management, skin detection, gene data clustering, and cluster visualization [66, 67, 68, 69]. However, the algorithm lacks some desired theoretical properties in the online continuous setting since it is not a gradient algorithm, while in the discrete setting, the energy function is not continuously differentiable. Thus several extensions of the classic algorithm have been proposed to address these challenges, among other things.

3.9.1 Heskens' rule

Heskes modified the original BMU selection rule to obtain an energy function [61].

$$bmu = \arg \min_{i \in units} \sum_{j=1}^N h_{ij}(t) ||x - m_i(t)||^2$$

The energy function is calculated as follows:

$$E(w) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N h_{ji}(t) \int_{x \in C_i(w)} ||x - m_j(t)||^2 g(x) dx$$

- $C_i(m)$: is the cluster associated with the i_{th} weight vector
- $g(x)$: is the pdf function
- h_{ji} : is the neighbourhood function between the j_{th} and i_{th} unit

The properties of both the energy and gradient function are summarized in the table below:

Kohonen vs Heskens BMU rule	
Kohonen BMU rule	Heskes BMU rule
Discrete Energy: discontinuous & finite Gradient: discontinuous & infinite	Discrete Energy: continuous Gradient: discontinuous & finite
Continuous Energy: continuous Gradient: discontinuous	Continuous Energy: continuous Gradient: discontinuous

TABLE 3.2: Kohonen vs Heskens BMU rule

3.9.2 Soft Topologic Mapping (STM)

This mapping strategy allows for soft winner assignments. In the discrete case, the energy function can be written as follows:

$$E(w, c) = \frac{1}{2} \sum_{k=1}^K \sum_{i=1}^N c_{ik} \sum_{j=1}^K h_{kj}(t) ||w_j(t) - x_i||^2$$

where

$$c_{ik} = \begin{cases} 1, & x_i \in C_k \\ 0, & otherwise \end{cases}$$

However, the c_{ik} can be smoothed by considering $c_{ik \geq 0}$ such that $\sum_{k=1}^K c_{ik} = 1$. Furthermore, an annealing process has to be utilized in order to solve this problem since the gradient-descent algorithm may possibly converge into a local minimum. On top of that, the energy function is smoothed by adding an entropy term such as:

$$F(w, c, \beta) = E(w, c) - \frac{1}{\beta} S(c)$$

If the function above, the term $S(c)$ corresponds to an entropy function parametrized by β such as:

- $\beta \rightarrow \infty$: In this case, the term $\frac{1}{\beta} \rightarrow 0$ and thus the function F becomes equal to function E
- For low values of β : smoothing occurs and consequently the function F has only one global minimum that can be estimated by a gradient descent-like algorithm

The basic steps of the deterministic annealing approach are described below:

1. calculate a minimum at low values of β
2. calculate a minimum at higher values of β
3. repeat 2 until the global minimum of F for $\beta \rightarrow \infty$ is equal to the global minimum of E

If β has a fixed value, the minimization of the function F can be approximated by iterating over the steps denoted by the following two equations:

$$\mathbb{P}(x_i \in C_K) = \frac{\exp(-\beta e_{ik})}{\sum_{j=1}^K \exp(-\beta e_{ij})}$$

- $e_{ik} = \frac{1}{2} \sum_{j=1}^K h_{jk}(t) ||x_i - w_j(t)||^2$

$$w_k(t) = \frac{\sum_{i=1}^N x_i \sum_{j=1}^K h_{jk}(t) \mathbb{P}(x_i \in C_j)}{\sum_{i=1}^N x_i \sum_{j=1}^K h_{jk}(t) \mathbb{P}(x_i \in C_j)}$$

The weight vectors are weighted averages over the corresponding input data. Furthermore, if $\beta \rightarrow \infty$, the process becomes identical to the classical batch-SOM algorithm.

3.9.3 Other Variations of SOM

Over the last years, several efforts have been made towards modifying the algorithm's architecture and learning process. However, there is still much work to be done in this direction since the lack of complete theoretical proofs that describe the convergence and organizational behavior of the multi-dimensional SOMs in combination with the algorithm's heuristic nature constitutes significant challenges that thwart the progress of the field. In this section, several SOM variants are presented.

- **The Elastic Maps:**

This algorithm learns by minimizing the sum of quadratic bending and stretching energy. In fact, it utilizes concepts such as spline interpolation and elastic energy minimization. Consequently, this model is widely applied for visualization and simulation tasks [70, 71]

- **The Generative Topographic Maps (GTM):**

This SOM-based architecture can map high-dimensional input data to low-dimensional spaces in a smooth and continuous way. Thus this algorithm can address some of the theoretical challenges mentioned above. The primary use of this algorithm is in data analysis as a nonlinear version of the principal component analysis (PCA) [72, 73].

- **The Growing Self-Organizing Maps (GSOM):**

This algorithm aims to address the problem of map size identification. The learning process starts with four units and uses a heuristic-based strategy to grow new units on the boundaries. It usually provides better input-space representation than SOM and consequently is widely applied for data visualization [74, 75].

- **The Oriented & Scaling Maps (OS-Map):**

As the name suggests, this algorithm allows for both scale and rotational invariance. The rotational invariance is achieved by replacing the neighborhood function with the matrix exponential. Furthermore, the map can cover the domain several times. Thus, the scale corresponds to the number of best matching units.

- **The Time-Adaptive Self-Organizing Maps (TASOM):**

This architecture extends the original algorithm in the sense that it employs adaptive learning and neighborhood rates as well as a scale factor for scale, translation, and rotation invariance. This algorithm is mainly used for multi-level thresholding [76, 77].

Chapter 4

Dendritic Self-Organizing Maps

In this section, we will introduce a variation of the original SOM algorithm. The Dendritic Self-Organizing Map (DendSOM) incorporates the concepts of receptive fields and local dendritic computations as mentioned above. Furthermore, we will study extensively how these changes affect the performance of SOM in unsupervised classification tasks. Additionally, we will discuss how the DendSOM architecture addresses the challenges of continual learning: concept drift, memory management, and catastrophic forgetting. Finally, we will compare the proposed architecture with the state-of-the-art algorithms in all three continual learning scenarios. It should be noted that the DendSOM algorithm is a bio-inspired algorithm motivated by brain maps.

In fact, there are three types of neural organization that can be called “brain maps”: sets of feature-sensitive cells ordered projections between neuronal layers and ordered anatomical maps of abstract features. The latter manifests the core properties of the experiences of an organism. It has been hypothesized that such feature maps are probably learned in a process involving parallel input to neurons of a particular brain area and adaptation of neurons in close proximity to the cells that respond to this input. Although such maps were considered unrealistic, the biological sensory processing substantiates the presence of such features maps in the brain. For example, tonotopic maps exist in auditory pathways and represent the order of acoustic resonances on the basilar membrane of the inner ear. [78, 79].

It is quite possible that a biological SOM may need days to form or alter neural maps in the neural realm. Thus the effects of reorganization are not always immediately measurable or even observable, and thus the organizing effects may be disguised by other short-term neuronal signals, also called activities. On the contrary, in the case of the SOM learning algorithm, changes that in w_i which affect signal transmission occur gradually and only intermittently during the learning process under the influence of some specific learning-control factor. Nonetheless, it is now evident that several biological components can implement the SOM algorithm.

The SOM algorithm has been gained significant popularity due to its remarkable clustering performance. Although it can be argued that it is inspired by the way that the human brain processes sensory information, there are still several modifications to be made that can result in improved performance, especially in the field of computer vision. For instance, the original algorithm processes the entire image at once while humans sample different regions of a scene by making rapid eye movements which can potentially decrease the root mean square error between the input data and the selected BMU on SOMs. Furthermore, our brains can perform parallel local computations in dendrites, significantly decreasing our response time to a stimulus.

4.1 Chapter Outline

In this chapter, we introduce and test the DendSOM architecture. The rest of this chapter is organized as follows:

- **Section 4.2:** In this section, we provide the biological background and the motivation for creating a SOM-based architecture to address the challenges of CL
- **Section 4.3:** The algorithm and experiments are discussed in this section
- **Section 4.4:** The implementation details and hyperparameter analysis are provided in this section

4.2 Background

Dendrites are thin extensions of nerve cells that receive and transmit electrochemical signals from other neurons to the cell's body, also called soma. The dendritic morphology is highly correlated to the function of the neuron, which indicates that the dendritic branches and grouping patterns play a key role in shaping the brain's information processing power. Dendritic spines are projections contained by some classes of dendrites and allow for better isolation of signal specificity in dendrites. The ability of dendrites to grow and extend is an essential factor in the formation of memory, and thus, it is believed that dendrites influence the learning process as well [80, 81].

Dendrites have several computational components or functional units that can perform the appropriate computations based on the stimuli received. Thus, these dendritic computational components are able to process input information and are also composed of dendritic spines or groups of branches. Hence, the plastic changes observed in dendrites can alter their morphology and consequently affect the neural cell's computational power and processing abilities. Although in the early development stages, the dendritic structure is mainly influenced by the cell's genome, in adulthood, more significant morphological changes are caused by signals from other neurons [82, 83].

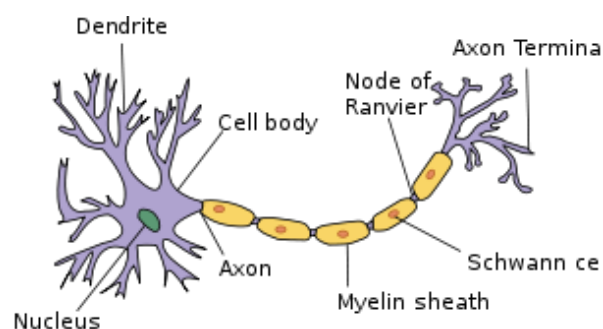


FIGURE 4.1: Diagram of a neuronal cell

Source: [Wikipedia](#)

4.2.1 Motivation

As mentioned in Chapter 2, continual learning is the concept of constructing a model that is able to learn a large number of tasks sequentially without forgetting the knowledge from previous tasks when training on new ones. The basic methods developed for addressing this problem involve regularization, replay, and parameter isolation. The inspiration for using SOM-based architectures to solve CL problems came from a paper entitled "Continual Learning with Self-Organizing Maps" [84]. In short, the authors of this paper proposed a memoryless SOM-controlled ANN for addressing the problem of catastrophic interference and, consequently, catastrophic forgetting.

The so-called Self-Organizing multilayer perceptron (SOMLP) architecture consists of an MLP and a SOM. In this architecture, the SOM layer controls the update of a fully connected layer. In fact, the SOM receives the same input as the fully connected layer and is trained without supervision to create a map that represents the geometrical relationships of the tasks. During training, the important parameters of the MLP are determined based on the SOM's topology, such as the activation of best-matching neurons whose corresponding unit is closer to the input vector is scaled up. Hence the SOM's topology assists the MLP to learn shared representations for similar tasks and separate representations for dissimilar ones.

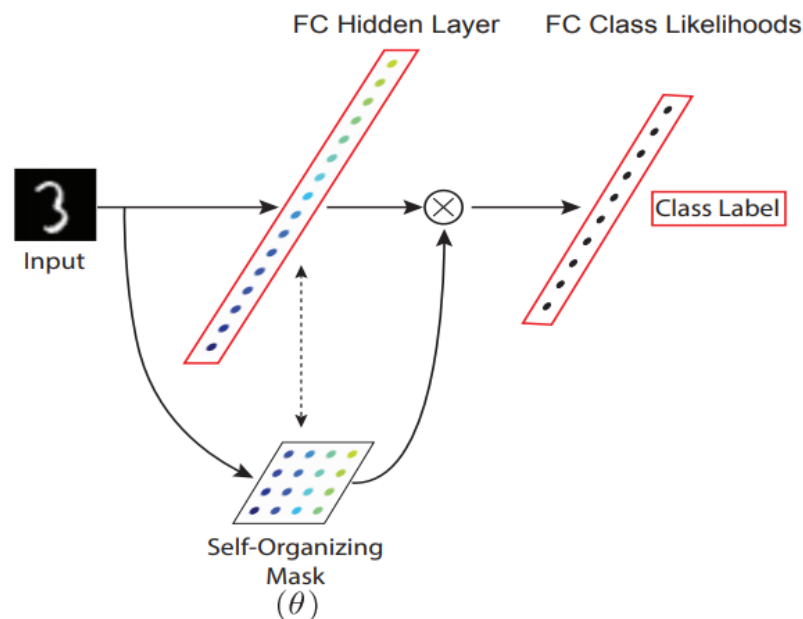


FIGURE 4.2: This figure illustrates the SOMLP architecture. Supervised training is denoted using red color borders

Source: [Continual Learning with Self-Organizing Maps](#)

Apart from the introduction of the SOMLP architecture, the authors also studied the topology of the SOM layer. The results indicated that the SOM is capable of associating each task to a different region of the map and thus produce a separate update mask. In fact, as commented by the authors: "In MNIST permutations, because of the random pixel permutations in each task, masks corresponding to each task are independent of each other. Conversely, in MNIST-rotations, the learned masks share nodes between tasks that are more similar." It should be noted that these results motivated us to further encouraged us to investigate the use of pure SOM models in CL.

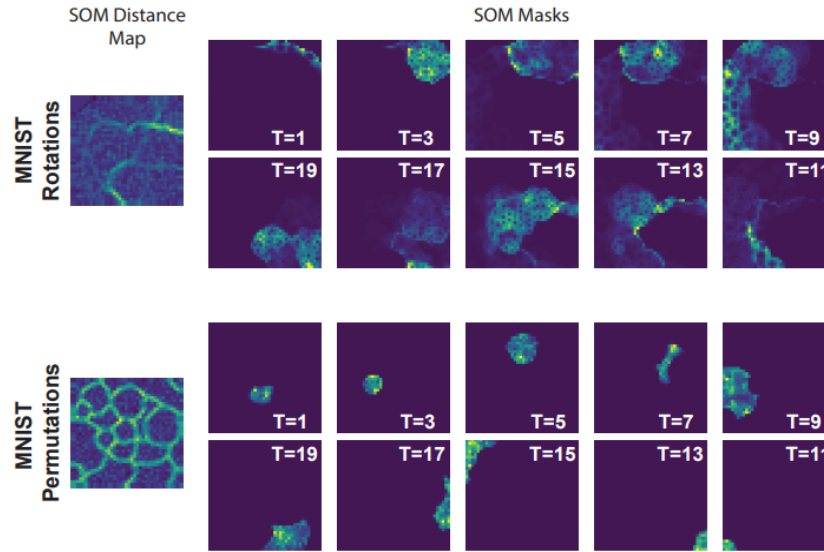


FIGURE 4.3: This figure illustrates the feature masks learned by the SOM layer for the MNIST-permutations and MNIST-rotations datasets

Source: [Continual Learning with Self-Organizing Maps](#)

4.3 Dendritic Self-Organizing Maps

In this section, we incorporate the concepts of receptive fields and computational locality into the original SOM algorithm to reduce the complexity and improve the performance of the online training algorithm. We test this hypothesis by measuring the mean-squared error and the training time of two variations of the original algorithm. Next, we present the DendSOM training algorithm. We derive a decision rule based on information theory and study the effects of different BMU and update rules. Finally, we use this algorithm for solving the CL scenarios as discussed in Chapter 2

4.3.1 Incorporating neurobiological concepts into SOMs

The receptive fields are regions of the observed visual space whose luminance and structural patterns affect the amplitude of the action potential and thus the activity of a single neuron. In the primary visual cortex, receptive fields correspond to visual angles and are organized by their position in the retina. In fact, different neurons tend to have different receptive fields and thus select and process specific orientation information. Receptive fields of different cells also have different structures. For instance, in the retina, the most common receptive fields are of the center-surround structure, whereas most cells in the primary cortex can be activated by simpler or more complex visual space regions [85].

The selectivity in individual neurons is highly correlated to the dendritic spine activity of those neurons since it has been observed that dendritic spines with similar selectivity are clustered together, and neurons whose dendrites consist of such clusters display a greater selectivity. Dendrites have the ability to perform elementary computations locally without interference from other neurons and propagate the result only to the cell body from which they extend due to their biophysical mechanisms. These local computations include the logical operations AND, OR, XOR and addition, multiplication, subtraction, and division as well. Thus, dendrites can perform both linear and nonlinear local computations [86].

Unlike other SOM-based architectures such as Deep SOM, Deep Convolutional SOM, and Layered-SOM [87, 88, 89], the DendSOM is a shallow neural network that can be trained using unsupervised competitive learning algorithms. Hence, this network can be used for unsupervised classification and continual learning tasks. DendSOM employs multiple SOMs to model different sub-regions of the input space (receptive fields). Similar to dendrites, each SOM receives input information and propagates the best matching unit's positional vector to soma, where the association between labels and BMUs is calculated

Hence, it is essential to answer the following questions to evaluate the DendSOM architecture's performance.

- Does the DendSOM minimize the MSE?
- Can the DendSOM's weight vectors properly represent the original images? (Are there significant changes in the content of the original image?)
- Has the algorithm low time complexity?

Although answering the above questions for both DendSOMs and SOMs can help us determine which algorithm performs better, we can not conclude how receptive fields and computational locality affect the performance of the DendSOM architecture. Thus we create the Receptive Self-Organizing Map (RSOM). The RSOM model is similar to SOM, but it sequentially receives the image's receptive fields instead of processing the entire image at once. Hence, the concept of receptive fields is incorporated into the RSOM algorithm. However, the notion of computational locality is absent, and thus different receptive fields can interact with each other. The schematics of SOM, RSOM, and DendSOM are shown below.

It should be noted that each of the following experiments was repeated for 10 rounds

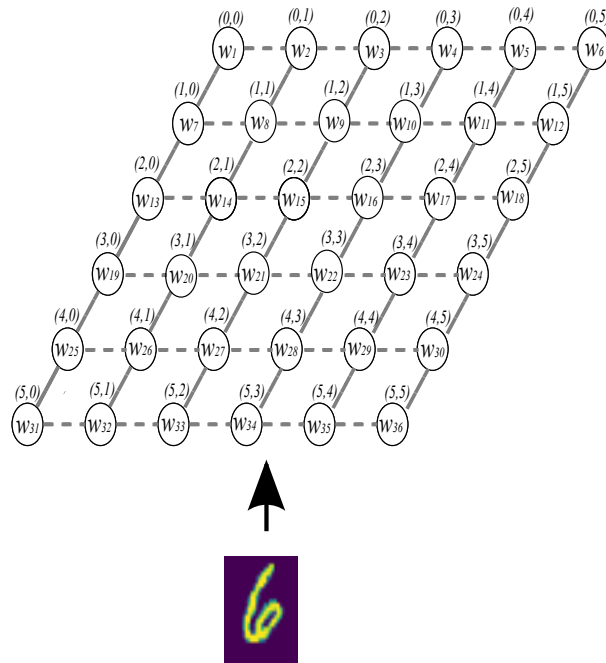


FIGURE 4.4: This figure illustrates the SOM architecture. The input vector is an image

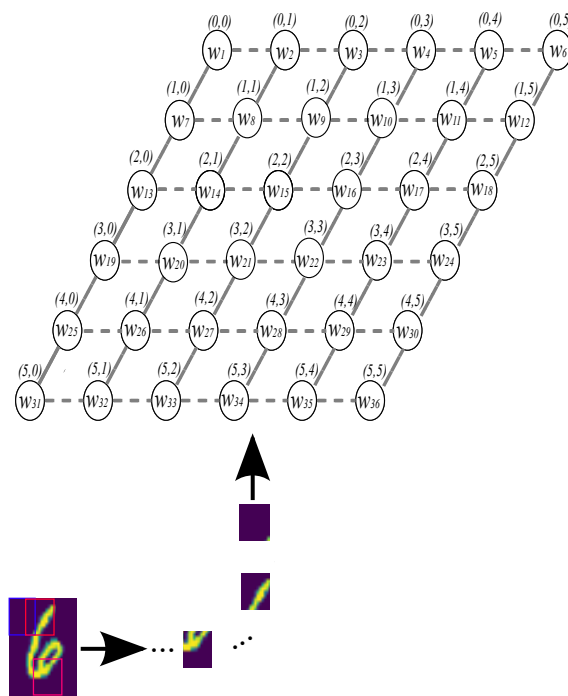


FIGURE 4.5: This figure illustrates the RSOM architecture. An image produces several input vectors, which are sequentially processed by a single SOM

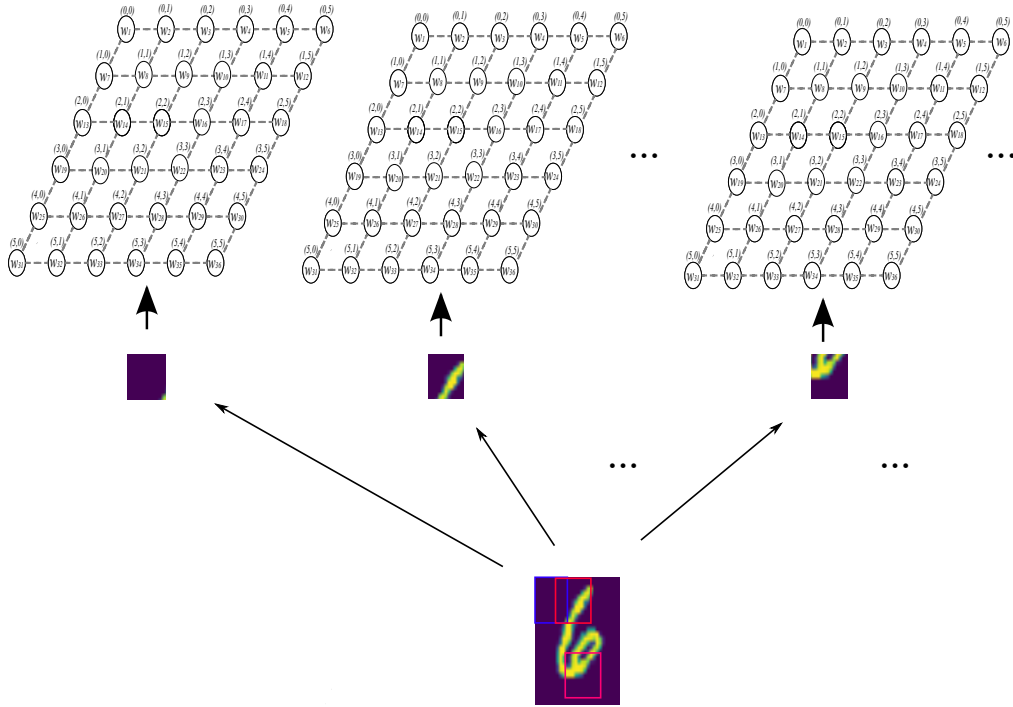


FIGURE 4.6: This figure illustrates the DendSOM architecture. An image produces several input vectors, each input vector is processed by a different SOM

using initial learning rate $a_0 = 0.5$, initial neighborhood radius $\sigma_0 = \frac{\sqrt{\#units}}{2}$ and $\lambda = 10e2$. The experiments were conducted on the MNIST dataset and the architecture parameters are shown in the following table:

MNIST-MSE MINIMIZATION				
Model	maps	units	wv	step
DendSOM	169	4×4	4×4	3
RSOM	1	52×52	4×4	3
SOM	1	8×8	28×28	—

TABLE 4.1: This table contains the parameters of all architectures tested in MSE minimization

MSE Minimization Test

To answer the first evaluation question, we have to calculate the average MSE between the input vectors and their corresponding BMUs for each of the aforementioned architectures. The MSE for each digit can be calculated as follows:

$$MSE(digit) = \frac{1}{\#samples(digit)} \sum_{y_{test}=digit} ||x_{test} - w_{bmu}||_2$$

The results can be summarized in the following table:

MSE per digit			
digit	SOM	DendSOM	RSOM
0	6.057 ± 0.931	3.427 ± 0.344	1.535 ± 0.255
1	3.383 ± 0.816	2.393 ± 0.363	0.976 ± 0.213
2	6.382 ± 0.729	3.495 ± 0.369	1.551 ± 0.280
3	6.022 ± 0.835	3.366 ± 0.361	1.612 ± 0.245
4	5.570 ± 0.835	3.160 ± 0.382	1.494 ± 0.265
5	6.216 ± 0.735	3.372 ± 0.347	1.594 ± 0.268
6	5.745 ± 0.959	3.381 ± 0.397	1.560 ± 0.289
7	5.132 ± 0.973	2.908 ± 0.416	1.261 ± 0.256
8	6.100 ± 0.788	3.588 ± 0.384	1.856 ± 0.294
9	5.229 ± 0.854	3.066 ± 0.377	1.532 ± 0.280
Avg.	5.584 ± 0.846	3.215 ± 0.374	1.498 ± 0.265

TABLE 4.2: This table contains the MSE per digit for each architecture

The table above indicates that the RSOM architecture achieves the minimum MSE for each class while the DendSOM performs better than the original algorithm but worse than RSOM. It is possible that the absence of the computational locality assists RSOM in further minimizing the MSE since there are no constraints regarding the location of the receptive fields.

Visual Test

In this experiment, we randomly choose a sample from each class and compare the reconstructed images, which are visual representations of the weight vectors that correspond to the BMUs of the random samples. This comparison will provide some insights into the ability of the algorithms to maintain the context of the input data. The results are shown in the figure below.

As shown in Figure 4.7, both the RSOM and DendSOM algorithms manage to keep the context of the original image. Although the SOM architecture maintains the context of the image in most cases, the results do not pass the visual test in the case of classes 4 and 5. The visual representation of digit four is closer to the representation for the ninth digit, and the weight vector representation of class five is much closer to digit four. These observations are quite valuable for two main reasons. At first, the inferiority of SOM to the other two architectures in the visual test is clearly demonstrated. Secondly, it becomes evident that the Euclidean distance may not properly map the geometric relations between the classes of this dataset.

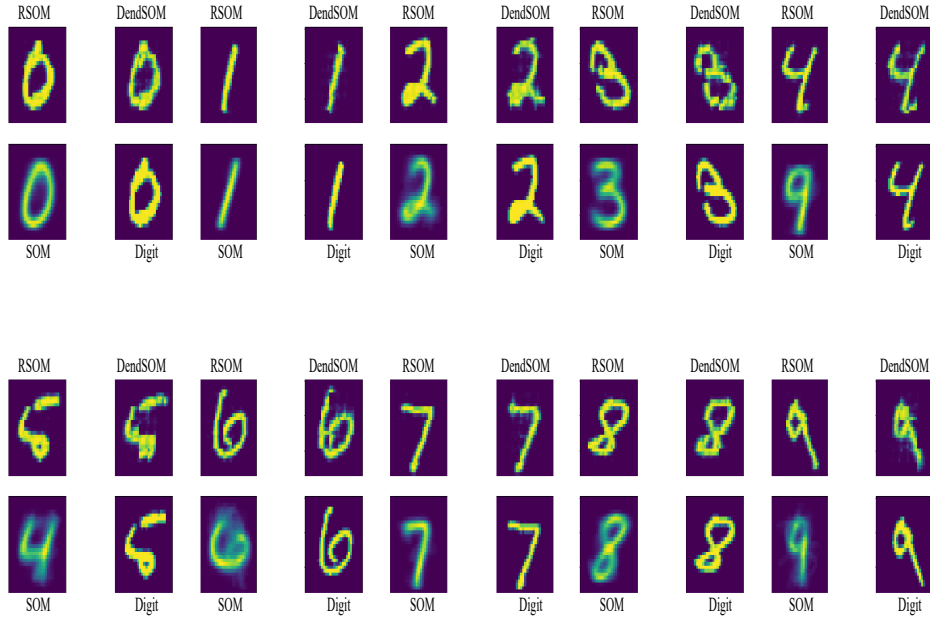


FIGURE 4.7: This figure illustrates the visual representations of some randomly selected samples

ANN Test

Although the visual test provided some valuable insights about the ability of the architectures to maintain the context of the images and the mapping power of the Euclidean distance, it is imperative to evaluate the performance of the SOM-based architectures on the entire dataset systematically. As the name suggests, the ANN test can estimate the ability of the algorithms to maintain the context of the images. In short, the ANN, SOM, RSOM, and DendSOM models are trained on the MNIST dataset, and then the visual representations of the test set are classified. It should be noted that the SOM-based algorithms do not require labels since they perform clustering. The results can be summarized in the following table:

Accuracy score per digit				
digit	SOM	DendSOM	RSOM	ANN
0	0.869 ± 0.003	0.967 ± 0.001	0.980 ± 0.001	0.985 ± 0.003
1	0.981 ± 0.005	0.987 ± 0.001	0.991 ± 0.007	0.991 ± 0.001
2	0.851 ± 0.007	0.968 ± 0.004	0.981 ± 0.002	0.985 ± 0.001
3	0.576 ± 0.004	0.954 ± 0.001	0.967 ± 0.002	0.970 ± 0.002
4	0.698 ± 0.008	0.962 ± 0.005	0.968 ± 0.001	0.973 ± 0.002
5	0.633 ± 0.002	0.950 ± 0.002	0.965 ± 0.005	0.973 ± 0.001
6	0.932 ± 0.003	0.976 ± 0.004	0.980 ± 0.003	0.982 ± 0.002
7	0.753 ± 0.003	0.940 ± 0.006	0.950 ± 0.006	0.961 ± 0.001
8	0.813 ± 0.008	0.950 ± 0.002	0.962 ± 0.004	0.962 ± 0.002
9	0.569 ± 0.005	0.955 ± 0.001	0.970 ± 0.001	0.973 ± 0.000
Avg.	0.768 ± 0.005	0.961 ± 0.003	0.971 ± 0.003	0.976 ± 0.002

TABLE 4.3: This table contains the accuracy score per digit for each architecture

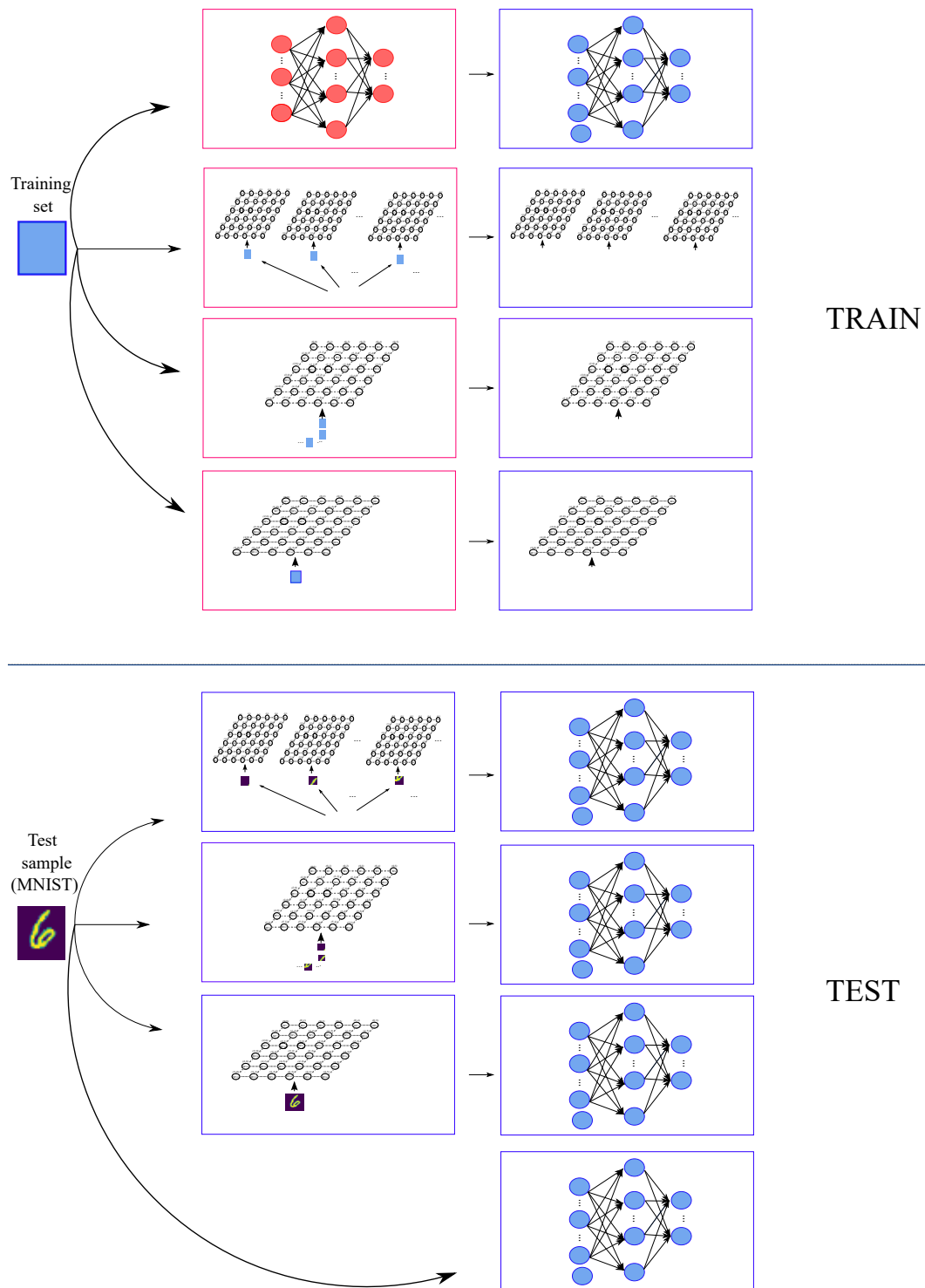


FIGURE 4.8: This figure illustrates the ANN test procedure

Once again, the RSOM algorithm achieves greater performance than the DendSOM model, which in turn is better than the original SOM model. It should be noted that none of the SOM architectures managed to surpass the accuracy score of the ANN, which indicates that there is a loss of information. However, both the RSOM and DendSOM perform comparably with the ANN.

Time Test

The time complexity is defined as the amount of time taken by an algorithm to run for inputs of given lengths. Although most ML algorithms have known time complexity, in practice, the training time can significantly differ due to the size and complexity of the dataset. Thus, in order to determine which algorithm has the lowest time complexity, it is essential to run each algorithm and measure the training time. The results are reported in the table below:

Training time			
Model	SOM	DendSOM	RSOM
Time	55 sec	150 sec	3000 sec

TABLE 4.4: This table contains the required training time (in sec) for each architecture

As indicated from the table above, the SOM has the smallest training cost. The DendSOM requires more training time than the SOM. However, it is still significantly faster in comparison to the RSOM model, which is by far the most expensive algorithm in terms of time since, in the case of RSOM, the receptive fields presented to the algorithm sequentially, whereas, in DendSOM, the receptive fields of an image are processed simultaneously. In fact, the RSOM algorithm can not properly utilize the computational power of the GPU.

Overall

The way that the performance of the original algorithm is affected by the concepts of computational locality and receptive fields affect the original algorithm the following metrics are used:

$$A(model) = \frac{|accuracy(som) - accuracy(model)|}{accuracy(som)} 100\% \quad (4.1)$$

$$T(model) = \frac{|time(som) - time(model)|}{time(som)} 100\% \quad (4.2)$$

$$G(model) = \frac{A(model)}{T(model)} \quad (4.3)$$

The results are shown in the following table:

Rate of Increase		
Metrics	DendSOM	RSOM
T	172%	5354%
A	25%	26%
G	0.14	0.005

TABLE 4.5: This table contains the rates of increase for training time and accuracy score in comparison to those of the SOM algorithm for both RSOMs and DendSOMs

An obvious conclusion that can be drawn based on the results above is that the absence of computational locality and the constraints it imposes leads to better context preservation. However, the incorporation of this concept into the RSOM model can significantly decrease the required training time. In fact, the G metric indicates that the RSOM is inferior to DendSOM since the G factor of RSOM is three times smaller than the G factor of DendSOM.

4.3.2 DendSOM Architecture

As mentioned above, the DendSOM uses multiple SOMs to model the input pattern from different subregions of the input space. Consequently, the order of local regions, also called input vectors, is crucial for the final labeling result due to the fact that the slightest change in the order of the receptive fields is capable of causing significant changes in the temporal evolution of the input image, which in turn will result in the incorrect mapping of the input space.

It should be noted that for an input \mathbf{x} of size $N_1 \times N_2$ and SOMs that extract patterns of size $P_1 \times P_2$ we need $S_1 \times S_2$ SOMs in order to properly model the input space. Assuming the stride s_1, s_2 respectively, then S_1 and S_2 can be calculated by:

$$S_i = \left\lfloor \frac{N_i - P_i}{s_i} \right\rfloor + 1, \quad i \in \{1, 2\} \quad (4.4)$$

In our implementation $s_1 = s_2, N_1 = N_2, P_1 = P_2$ and thus $S_1 = S_2$.

Learning Algorithm

Each SOM can be trained independently by utilizing a similarity metric for best-matching unit identification and a weight update strategy for adjusting the weight vectors. Apart from that, an association matrix is introduced in order to assist us in measuring the association between SOM units and labels. This association corresponds to the number of times that a unit was selected as BMU for a specific label. The association matrix can then be employed in combination with a decision rule in order to predict which class is most likely to be represented by the selected BMUs.

Algorithm 1: DendSOM training algorithm

Input: $X_{train}, y_{train}, \alpha_0, \sigma_0, \lambda, model, \alpha_{crit}, r_{exp}$
Output: $model, assoc_matrix$

```

1  $model.random\_init()$ 
2  $assoc\_matrix = zeros(n\_labels, S_1 \times S_2, U_1 \times U_2)$ 
3  $t = 0$ 
4  $iter\_crit = \lfloor \lambda \log \frac{\alpha_0}{\alpha_{crit}} \rfloor$ 
5 for  $i$  in  $range(0, n\_iter)$  do
6    $sample, label = X_{train}[i], y_{train}[i]$ 
7    $rfs = create\_receptive\_fields(sample)$  // size:  $(S_1 \times S_2, 1, 1, P_1, P_2)$ 
8    $bmus = identify\_bmus(rfs, model)$ 
9    $model = model\_update(model, t, bmus, \alpha_0, \sigma_0, \lambda)$ 
10   $assoc\_matrix[label, bmus] = assoc\_matrix[label, bmus] + 1$ 
11  if  $i \% iter\_crit == 0$  then
12     $t = \lfloor \frac{t}{r_{exp}} \rfloor$ 
13  $t = t + 1$ 

```

FIGURE 4.9: Illustration of the DendSOM online training algorithm for unsupervised classification and CL

It should be noted that for unsupervised classification or clustering $r_{exp} = 1$. What is more, the $iter_crit$ parameter can be derived as follows:

$$\begin{aligned}\alpha_{crit} &= \alpha_0 \exp\left(-\frac{iter_crit}{\lambda}\right) \leftrightarrow \\ \exp\left(\frac{iter_crit}{\lambda}\right) &= \frac{\alpha_0}{\alpha_{crit}} \leftrightarrow \\ iter_crit &= \lambda \log \frac{\alpha_0}{\alpha_{crit}} \leftrightarrow \\ iter_crit &= \left\lfloor \lambda \log \frac{\alpha_0}{\alpha_{crit}} \right\rfloor \\ iter_crit &\in \mathbb{N}\end{aligned}$$

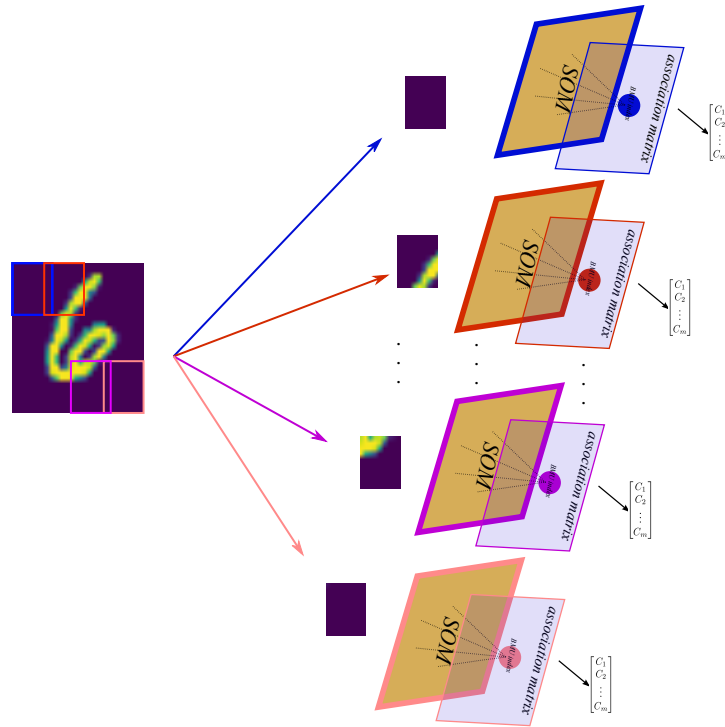


FIGURE 4.10: The image is divided into sub-regions. Each sub-region of the input space is modeled from the corresponding SOM. The SOMs identify the BMUs whose indexes are then propagated to the appropriate association matrices. These matrices estimate the association between the BMUs and the class labels

Defining a decision rule for unsupervised classification

The introduction of the association matrix allows for estimating both the prior and posterior distributions for each map. These pieces of information can be utilized by a function that maps the observed sequence of BMUs to the class that is most likely to be associated with those units. Pointwise Mutual information (PMI) is a measure of association between a pattern and an action. The PMI of two outcomes x_o, y_o can take positive values when x_o, y_o co-occur more frequently than expected under an independence assumption, negative values when the outcomes co-occur less frequently than expected, and it can be equal to 0 when the outcomes are statistically independent. For a single association matrix, we can easily calculate the PMI between a label (l) and a BMU as follows:

$$PMI(l; BMU) = \log \frac{P(l|BMU)}{P(l)} \quad (4.5)$$

where $P(l|BMU)$ corresponds to the conditional probability of a class label given a BMU and it can be calculated using the association matrix:

$$P(l|BMU) = \frac{assoc_matrix[l, BMU]}{\sum_{i \in Labels} assoc_matrix[i, BMU]} \quad (4.6)$$

Where $assoc_matrix[l, BMU]$ is the number of times that the BMU was the closest unit to an image of label l and $\sum_{i \in Labels} assoc_matrix[i, BMU]$ corresponds to the total number of times that the BMU was selected by an image. Moreover, the *a priori* probability $P(l)$ can be calculated by:

$$P(l) = \frac{\sum_{u \in Units} assoc_matrix[l, u]}{\sum_{u \in Units} \sum_{i \in Labels} assoc_matrix[i, u]} \quad (4.7)$$

Where $\sum_{u \in Units} assoc_matrix[l, u]$ is the total number of images in the training set that represent the label l and $\sum_{u \in Units} \sum_{i \in Labels} assoc_matrix[i, u]$ is the total number of images in the training set. According to subsection 4.3.2, in the case of DendSOM, we have more than one BMUs. In fact, we have a BMU for each SOM; hence, the class prediction problem can be formulated in the following way. Let I be an input image of unknown label and $BU = \{bmu_1, bmu_2 \dots bmu_n\}$ a set that contains the best matching units of I . Thus, the objective of the label prediction problem can be expressed as:

$$prediction = \arg \max_{l \in Labels} \sum_{i=1}^n PMI(l; bmu_i) \quad (4.8)$$

It is worth to be mentioned that certain patterns, also called typical patterns, are very likely to appear many times, even in images that represent different labels. For instance, the blank regions in the MNIST dataset are part of the background, and thus, they appear in about the same places in all the images regardless of the label of each image. The use of the PMI decision rule allows for identifying those typical patterns and nullifying their contribution during the label prediction. In fact, for those patterns, it holds that $P(l) = P(l|BMU) \forall l \in Labels$ and thus $\frac{P(l|BMU)}{P(l)} = 1 \forall l \in Labels$. Consequently, $\log \frac{P(l|BMU)}{P(l)} = PMI(l; BMU) = 0 \forall l \in Labels$. Let I be an image of unknown label, $BU = \{bmu_1, bmu_2 \dots bmu_n\}$ a set that contains the best matching units of I and $TBU = \{bmu_a, bmu_b \dots bmu_k\}$ a set that contains the

best matching units of the subregions of I that correspond to typical patterns. The problem of label prediction can be expressed as follows:

$$prediction = \arg \max_{l \in Labels} \sum_{b \in BU} PMI(l; b) = \arg \max_{l \in Labels} \sum_{b \in BU - TBU} PMI(l; b) \quad (4.9)$$

Investigating the effects of a new BMU rule

According to equation 3.2, the BMU is defined as the unit whose weight vector has the minimum Euclidean distance from the input vector. The Euclidean distance corresponds to the L2 norm of a difference between vectors. Thus, the Euclidean distance estimates the similarity between patterns by measuring how close the images are in pixel intensity. On the other hand, the cosine similarity is proportional to the dot product of two vectors and inversely proportional to the product of their magnitudes, and vectors with a high cosine similarity are located in the same general direction from the origin. Consequently, we can express the objective as:

$$BMU = \arg \max_i \cos(\theta_{x, w_i}) \quad (4.10)$$

where θ_{x, w_i} is the angle between the input vector \mathbf{x} and the weight vector of the i -th unit \mathbf{w}_i . Furthermore, the cosine similarity between two vectors \mathbf{v} , \mathbf{u} can be calculated by:

$$\cos(\theta_{v, u}) = \frac{\mathbf{v}^T \mathbf{u}}{\|\mathbf{v}\| \|\mathbf{u}\|} \quad (4.11)$$

Spatial correlation is a signal-matching technique performed in the spatial domain. The cosine similarity can be mathematically interpreted as the cosine of the angle between two vectors x and y , while correlation is the cosine similarity between the centered vectors x^c , y^c . In the spatial domain, the correlation corresponds to the process of moving a filter mask over the image and computing the sum of products. However, in the case of DendSom, the cosine similarity causes highly correlated receptive fields to select the same or neighboring BMUs and thus create a new low-dimensional space that maintains the spatial correlation of the input space

In this following experiment, the hypothesis that the DendSOM algorithm allows for improved classification performance when compared to the original SOM algorithm is tested. To do so, we compare the two algorithms in terms of the accuracy score on three publicly available datasets.

SOM vs Dendritic-SOM			
Model	MNIST	MNIST-FASHION	CIFAR-10
SOM_{euc}	84.51 ± 0.01	73.72 ± 0.00	25.66 ± 0.00
SOM_{cos}	86.96 ± 0.00	76.48 ± 0.00	27.05 ± 0.00
$DendSOM_{euc}$	95.30 ± 0.00	77.74 ± 0.00	38.42 ± 0.00
$DendSOM_{cos}$	95.12 ± 0.00	80.89 ± 0.00	47.13 ± 0.00

TABLE 4.6: The table summarizes the accuracy score of each architecture in unsupervised classification on three benchmark datasets

As the results suggest, the classification accuracy is influenced by both the architecture and BMU selection rule. In short, the results in the table show a consistent

pattern that utilizing multiple SOMs to map different regions improves the performance of the original algorithm since the final matching pattern is not just a weighted average of similar images but it consists of several informative features. Furthermore, in most cases, the cosine similarity rule allows for improved performance compared to the Euclidean distance rule due to the fact that the cosine similarity is only influenced by the inter-pixel relation between the weight vector and the corresponding receptive field while the Euclidean distance takes into account the average pixel value difference. According to the table, the Dendritic-SOM that employs cosine-similarity for BMU selection achieves the highest accuracy score on the MNIST-FASHION and CIFAR-10 datasets while it performs comparably to the Dendritic-SOM that uses the Euclidean distance rule for BMU selection on the MNIST dataset. Finally, all the algorithms achieve their highest and lowest scores on the MNIST and CIFAR-10 datasets, respectively. In fact, the MNIST dataset consists of handwritten digits in a static background, and thus it is the most simple dataset. On the other hand, the CIFAR-10 dataset consists of 10 real-world classes in a dynamic background. Thus, the geometrical properties of the input space can be properly described neither by cosine-similarity nor by the Euclidean distance.

The DendSOM training hyperparameters for this experiment are summarized in the following table:

SOM vs DendSOM				
Dataset	α_0	σ_0	λ	r_{exp}
MNIST	0.95	4	1000	1
MNIST-FASHION	0.95	5	1000	1
CIFAR-10	0.95	6	1000	1

TABLE 4.7: Training hyperparameters for unsupervised classification

It should be noted that the same hyperparameter values are also used in the case of SOM. Furthermore, the architecture parameters used on the MNIST, MNIST-FASHION, and CIFAR-10 datasets are shown in the tables below.

MNIST								
Model	S_1	S_2	U_1	U_2	P_1	P_2	s_1	s_2
DendSOM	7	7	8	8	10	10	3	3
SOM	1	1	21	21	28	28	—	—
MNIST-FASHION								
Model	S_1	S_2	U_1	U_2	P_1	P_2	s_1	s_2
DendSOM	6	6	10	10	8	8	4	4
SOM	1	1	18	18	28	28	—	—
CIFAR-10								
Model	S_1	S_2	U_1	U_2	P_1	P_2	s_1	s_2
DendSOM	15	15	12	12	4	4	2	2
SOM	1	1	29	29	32	32	—	—

TABLE 4.8: Architecture parameters for unsupervised classification

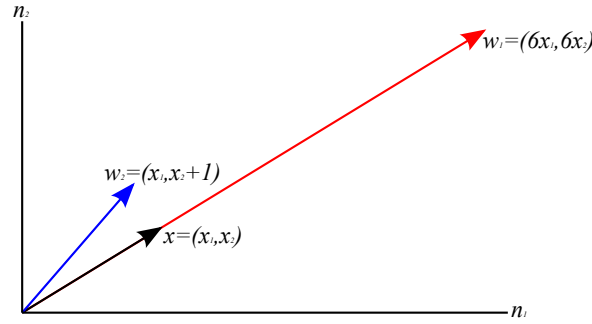


FIGURE 4.11: This diagram shows the 2D vectors x (input vector) w_1 and w_2 (weight vectors). According to the minimum Euclidean distance rule the unit that maintains the w_1 is the best matching unit since $euc(x, w_1) < euc(x, w_2)$. However, the angle between x and w_2 is smaller than the angle between x and w_1 and consequently $\cos(x, w_2) > \cos(x, w_1)$. Hence the maximum cosine rule indicates that the best matching unit is the unit with weight vector w_2 . The cosine similarity rule is a scale-invariant rule since it depends only on the inter-pixel relations and not from the exact pixel values

4.3.3 DendSOM for CL

In this subsection, we will utilize the DendSOM architecture to address the challenges of CL. Furthermore, we will compare the performance of the DendSOM model with the state-of-the-art architectures in CL. The performance of the compared architectures will be measured using the accuracy score on the Split-MNIST and Split-CIFAR-10 datasets. The three challenges of continual learning and the corresponding solutions are listed below:

Memory Management

The DendSOM is a memoryless architecture and thus there is no need for finding an optimal sample saving strategy

Concept Drift

The DendSOM model accounts for both real and virtual concept drift by estimating the prior and posterior probabilities and utilizing the PMI as a decision rule since the PMI is only influenced by the ratio between the posterior and prior probabilities and not by the exact probabilities

Catastrophic Forgetting

Like the original SOM algorithm, the DendSOM performs constrained-kmeans. Thus similar samples are matched to the same or neighboring units while less similar samples are matched to different units. This strategy aims to minimize the catastrophic interference between samples and consequently avoid catastrophic forgetting.

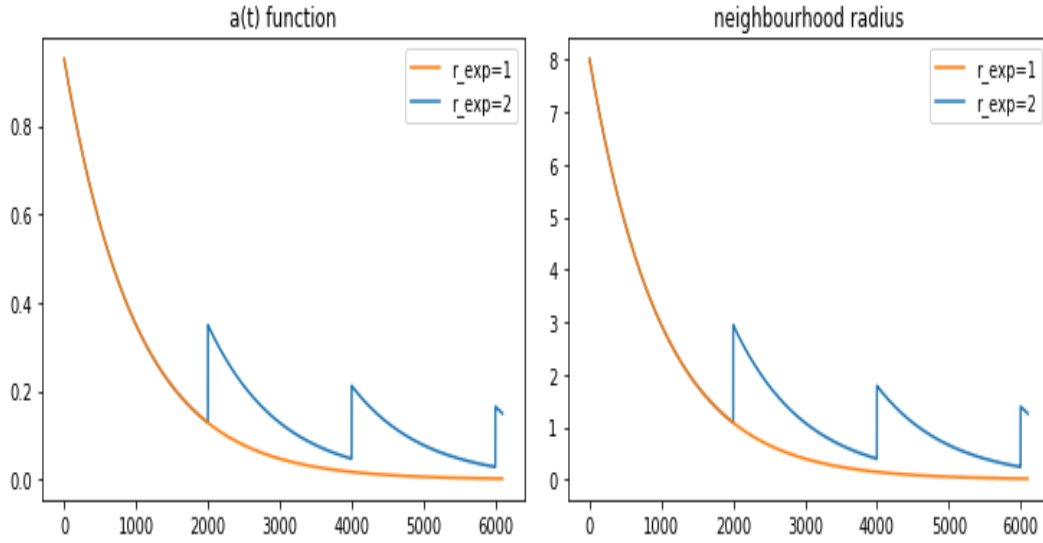


FIGURE 4.12: This figure illustrates the behavior of a and σ functions for $r_{exp} \in \{1, 2\}$

The parameters of the DendSOM architecture are identical to those of Table 4.8. On the other hand, the training parameters are shown in the table below:

Incremental Learning Experiments					
Dataset	α_0	σ_0	λ	r_{exp}	a_{crit}
MNIST	0.95	4	1000	2	$5e - 3$
CIFAR-10	0.95	6	1000	2	$5e - 5$

TABLE 4.9: The IL training parameters are reported in this table

It should be noted that only the hyperparameter r_{exp} changes in the case of incremental learning experiments since it expands the neighborhood radius and increases the learning rate. As shown in the example above (see Figure 4.12) for $r_{exp} = 1$, both the learning rate and neighborhood radius exponentially decrease over time as expected theoretically. On the other hand, for $r_{exp} = 2$, both a and σ values are doubled every 2000 repetitions, which prevents the early completion of input space mapping in the case of continual learning.

Task-Incremental Learning

The performance of our algorithm is evaluated and compared to other state-of-the-art architectures [22] on the Split-MNIST protocol under the limitations of the task-incremental learning scenario.

Split protocol	
Model	MNIST
SOM	82.93 ± 0.08
DendSOM	98.30 ± 0.00
EWC	98.64 ± 0.22
SI	99.09 ± 0.15
XdG	99.10 ± 0.08
DGR	99.50 ± 0.03
LwF	99.57 ± 0.02
DGR+ distill	99.61 ± 0.02

TABLE 4.10: The accuracy scores for the Task-IL scenario are summarized in this table

According to the table above, the models performed well in this scenario. However, the SOM algorithm had the lowest accuracy score; unlike other methods, both the SOM and DendSOM can not use the task information to learn task-specific features by freezing or regularizing its weight vectors. In contrast, it can utilize this piece of information only to estimate the appropriate distributions. It should be noted that the DendSOM algorithm achieves a higher accuracy score than the SOM algorithm because the latter processes the whole image at once and determines the BMU using the Euclidean distance. Consequently, the SOM algorithm does not take into account the temporal evolution of the images.

Domain-Incremental Learning

This experiment is similar to the aforementioned one except that the models operate under class-incremental constraints.

Split protocol	
Model	MNIST
EWC	63.95 ± 1.90
SI	65.36 ± 1.57
LwF	71.50 ± 1.63
SOM	74.33 ± 0.02
DendSOM	90.83 ± 0.01
DGR	95.72 ± 0.25
DGR+ distill	96.83 ± 0.20

TABLE 4.11: The accuracy scores for the Domain-IL scenario are summarized in this table

As shown in the table above, the DendSOM algorithm achieves an accuracy score of 90.83, ranking our proposed model among the top 3 models for domain-incremental learning on the Split-MNIST dataset. However, the other algorithms permit many epochs per task and are trained in batch mode. On the contrary, there is only one stream of samples, and each sample is presented only once in both the SOM and DendSOM cases. Although the SOM algorithm marks the fourth higher accuracy score, still its performance is inferior to the DendSOM algorithm. The superiority of the DendSOM algorithm is due to both the architecture and the BMU selection rule, as mentioned in the previous experiment. On top of that, we calculated the forgetting rate for both RSOM and DendSOM as shown in Equation 2.1. Consequently, we

have that $fr_{SOM} = 0.12$ and $fr_{DendSOM} = 0.05$, which also indicates that the SOM algorithm tends to forget with a higher rate than the DendSOM algorithm.

Class-Incremental Learning

Furthermore, we compare the performance of our algorithm to the state-of-the-art models [90] in class-incremental learning on both the Split-MNIST and Split-CIFAR-10 protocols.

Split protocol		
Model	MNIST	CIFAR-10
DGR	91.30 ± 0.60	17.21 ± 1.88
EWC	19.95 ± 0.05	18.63 ± 0.29
SI	19.95 ± 0.11	18.14 ± 0.36
CWR	32.48 ± 2.64	18.37 ± 1.61
CWR+	37.208 ± 3.11	22.32 ± 1.08
ARI	48.84 ± 2.55	24.44 ± 2.55
SOM	63.35 ± 0.10	25.44 ± 0.04
DendSOM	92.17 ± 0.08	46.02 ± 0.00
GC	93.79 ± 0.08	56.03 ± 0.04

TABLE 4.12: The accuracy scores for the Class-IL scenario are summarized in this table

The table shows the performance of several algorithms on the Split-MNIST and Split-CIFAR-10 protocols. Our model outperformed most of the other algorithms in both cases. In fact, the DendSOM and algorithms achieved accuracy scores of 91.04 and 46.02 on the Split-MNIST and Split-CIFAR-10 datasets, which correspond to the third-highest and second-highest scores, respectively. Furthermore, unlike the other models, the DendSOM was trained in an online mode since the tasks were presented sequentially and only once while the batch size was equal to 1. Once again, the SOM algorithm is proved to be a simple yet effective algorithm in CL that performs comparably with several leading-edge regularization and hybrid algorithms. However, the DendSOM achieves a significantly higher accuracy score than the SOM since the cosine similarity rule does not take into account the sum of individual pixel intensity differences like the SOM. What is more, the restrictive connectivity introduced in the DendSOM algorithm allows for a more detailed mapping of the input space, which in turn accounts for capturing the temporal changes of the input images. Finally, we compute the forgetting rate for the SOM and DendSOM algorithm on each dataset.

Split protocol		
Model	MNIST	CIFAR-10
SOM	0.250	0.009
DendSOM	0.031	0.024

TABLE 4.13: The forgetting rates for the Class-IL scenario are summarized in this table

The results indicate that the SOM appears to forget with a higher rate than the DendSOM since the SOM-mapped space does not properly represent the MNIST input space. On the other hand, the SOM forgets with a lower rate than the DendSOM in the case of the CIFAR-10 dataset. However, we can not draw a clear conclusion

as to which algorithm better addresses the problem of catastrophic forgetting since the accuracy score achieved by the SOM algorithm is significantly low even in the joint-training settings.

4.4 Technical Stuff

In this section, we examine how the training hyperparameters affect the performance of the DendSOM algorithm. In addition, we also provide some implementation details about the SOM and DendSOMs algorithms.

4.4.1 Implementation Details

Initially, a SOM model was implemented using Python 3.8 and CuPy 8.0.0 since the SOM is a special type of neural network, and thus custom features are needed in the model. The results of the SOM model were then evaluated using a Numpy 1.19.2 implemented SOM on the MNIST dataset. The unsupervised classification accuracy score was calculated ten times for each model, and the Wilcoxon Signed-Rank Test [91, 92] was applied on those repeated measurements. This test indicated no statistical significance between the accuracy scores with a confidence interval of 95%. Next, these algorithms were compared in terms of time complexity, and the results suggested that the cupy-enabled SOM was significantly faster than the numpy-based model.

In fact, the cupy-based SOM completed the training process in 180 seconds while the numpy-based SOM in 1554.55 seconds on average, and thus the cupy-based SOM is 8.64 times faster than the numpy-based SOM. It should be noted that the SOM algorithm was implemented based on the online SOM algorithm presented in Chapter 2. Moreover, the DendSOM algorithm was implemented using python and CuPy 8.0.0 and initially required 600 seconds to complete the training process since the computation of the cosine similarity was expensive because it required loops. Hence, it was imperative for this computation to be optimized to decrease the time complexity of the DendSOM algorithm.

Fortunately, the CuPy dependency offers the command `cupy.einsum`, which can assist us in circumventing the problem mentioned above. As the name suggests, the `einsum` command evaluates the Einstein Summation convention on two n-dimensional operands for simplifying expressions, including summations of vectors, matrices, and general tensors. The three rules of Einstein summation notation are shown below:

- Repeated indices are summed over
- Each index can appear at most twice in any term
- Each term must contain identical non-repeated indices.

This optimization significantly decreases the time complexity of the DendSOM algorithm from 600 seconds to 260 seconds since it allowed for parallel computation of the corresponding inner products in cosine similarity. It should be noted that the `einsum` command provides further flexibility to compute array non-classical Einstein summation operations by forcing summation over specified subscript labels.

However, the time complexity results presented in this section are not on par with the results illustrated in table 4.4. In fact, the algorithms in this section are 2-3 times slower than the algorithms presented in the previous section. Although the source code remained the same, at the time of writing, we use an NVIDIA GTX 1050 Ti 4GB GPU, 8 GB DDR 4 RAM, and Ryzen 5 1600 6 Cores 3.2 GHz, whereas the previous experiments were conducted in a machine with RTX 2080 Ti 11GB GPU, 64 GB DDR4 RAM and AMD Ryzen Threadripper 1950X 16 Cores 3.40 GHz provided by the Poirazi Lab. On top of that, the rest of the algorithms mentioned in tables 4.10, 4.11 and 4.13 were already implemented using Python and Pytorch 1.1.0 and the codes are publicly available on GitHub [93, 94, 95]. Although PyTorch provides useful abstractions and has a lower training time than CuPy, it has higher memory usage. Thus at the time of writing, there is no way to measure the time complexity of the other CL algorithms due to the memory limitations of our system. Nevertheless, it is safe to assume that the DendSOM's time complexity is not significantly higher than the complexity of the other models due to the fact that the other algorithms perform 2000-5000 epochs per batch in order to achieve the scores mentioned above while the DendSOM algorithm performs only one epoch. Finally, the introduction of computational parallelism during the creation of receptive fields can further decrease the time complexity of the DendSOM algorithm.

4.4.2 Hyperparameter Analysis

The DendSOM's behavior is controlled by several training hyperparameters. The ultimate goal is to find an optimal combination of hyperparameters that increases the predictive power of our model and thus gives better results. Hence, additional experiments have been performed to determine how the training hyperparameters affect the performance of our model individually. In fact, we discovered a good set of hyperparameters through a trial-and-error process (see Tables 4.7, 4.9) based on which we carried out the following experiments:

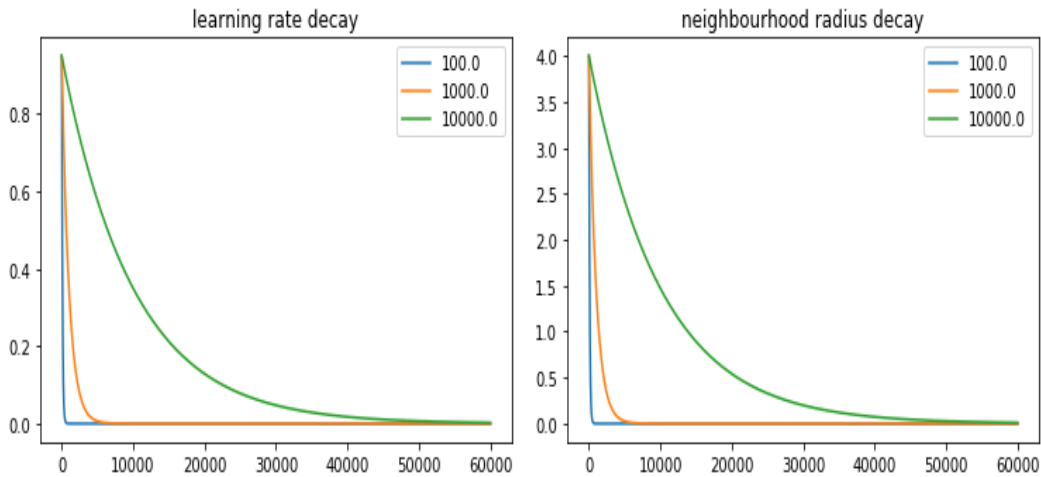


FIGURE 4.13: This figure illustrates the behavior of a and σ functions for different values of λ

As shown in the figure above, for $\lambda = 1000$, both α and σ tend to zero after almost 1000 time steps, and thus the DendSOM can not properly map the input space due to an insufficient number of input samples. However, for higher λ values, the mapping process requires several epochs to be completed, which increases the training time of the algorithm. In the case of the DendSOM model, $\lambda = 10e2$ is a good choice since incorporating the receptive fields allows for averaging in smaller regions of the input space and thus extracting simpler patterns.

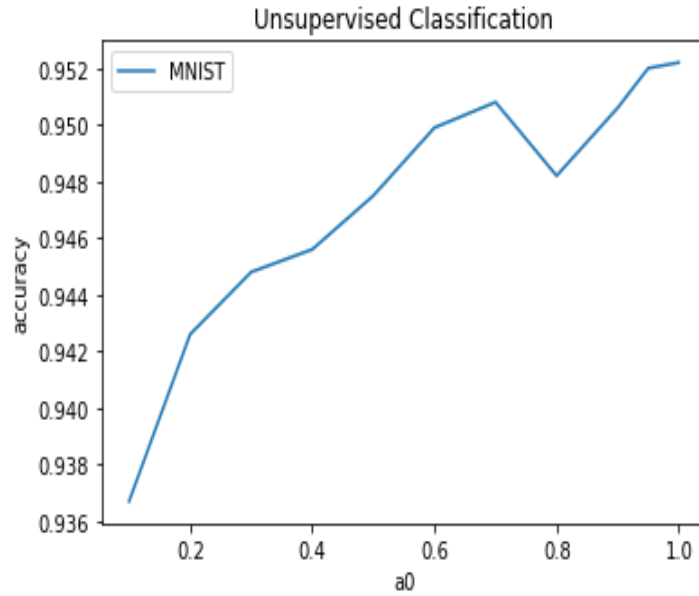


FIGURE 4.14: This figure illustrates how the initial learning rate affects the unsupervised classification accuracy of the DendSOM algorithm on the MNOST dataset

As the figure above suggests, a higher α_0 usually results in a higher accuracy score. However, the original SOM algorithm requires that $0 < \alpha_0 < 1$ the introduction of the maximum cosine similarity rule for BMU identification is a scale-invariant rule, and thus this experiment can be extended for $\alpha_0 > 1$. For $\alpha_0 = 0.1$, the DendSOM model has an accuracy score of 0.936, which in turn suggests that it is possible to make accurate predictions without updating the map during the training process, but this hypothesis has to be tested. It should be noted that σ_0 was initialized using the following heuristic:

$$\sigma_0 = \frac{\max\{U_1, U_2\}}{2}$$

where U_1 and U_2 correspond to the number of units across the horizontal and vertical axis respectively

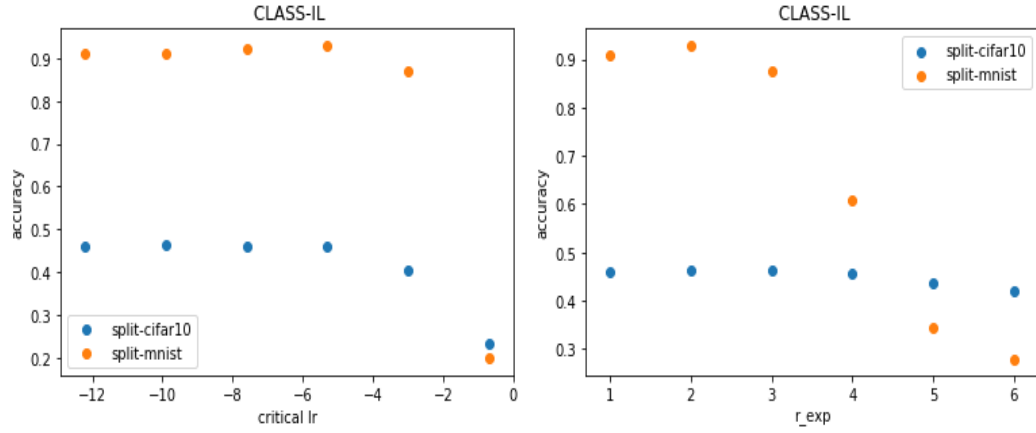


FIGURE 4.15: This figure illustrates the influence of α_{crit} and r_{exp} hyperparameters over the DendSOM algorithm. It should be noted that the critical lr axis is in logarithmic scale since $\alpha_{crit} \in \{0.5, 0.05, 0.005, 0.0005, 0.00005, 0.000005\}$

The figure above indicates that high α_{crit} values tend to decrease the performance of the algorithm while relatively low r_{exp} values tend to improve the predictive ability of our model. In fact, either high r_{exp} or high α_{crit} values tend to increase both α and σ dramatically, which in turn leads to weight vector overwriting. It is worth noting that the algorithm marks the highest accuracy score on MNIST and CIFAR-10 datasets for $\alpha_{crit} = 0.005$ and $\alpha_{crit} = 0.00005$, respectively.

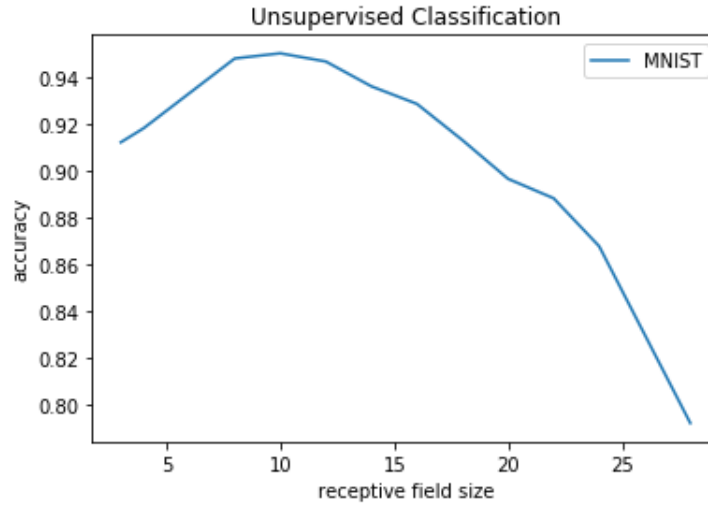


FIGURE 4.16: This figure illustrates how the receptive field size affects the DendSOM algorithm

As shown in the figure above, for receptive field sizes smaller than 10, the accuracy score of our model increases due to the fact that the patterns extracted from the input space are most likely to be informative patterns and thus non-typical patterns. On the other hand, when the receptive field size is greater than 10, the DendSOM is reduced to a SOM, and thus, the accuracy score decreases. It is worth to be mentioned that for receptive field size equal to 1, the accuracy score is 9%.

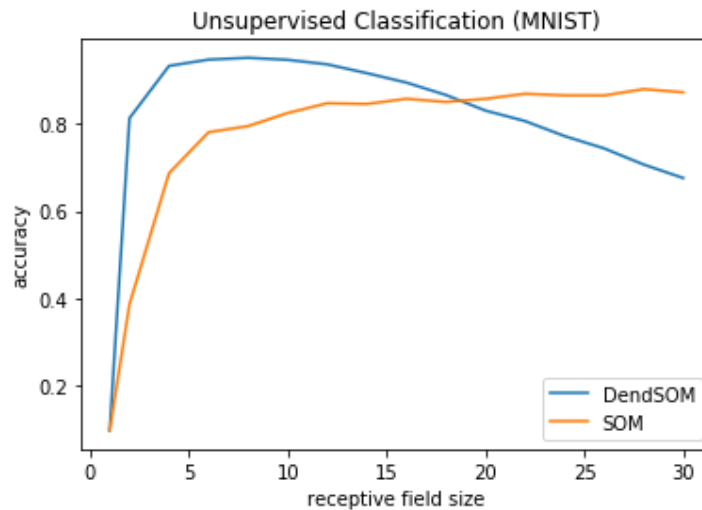


FIGURE 4.17: The influence of number of units per map over the DendSOM algorithm is illustrated in this figure

The results suggest that increasing the number of units improves the performance of the SOM algorithm. However, this does not seem to be the case for the DendSOM model since increasing the number of units above 100 decreases the model's predictive ability. This result is counterintuitive, and thus, additional experiments have to be conducted in order to draw clear conclusions.

Chapter 5

Conclusions & Future Work

In this thesis, we incorporated neuroscience-related concepts into the SOM architecture to create a new algorithm capable of addressing the challenges of CL. In this chapter, the work of this thesis is summed up, and directions for future work are provided.

5.0.1 Chapter 2: Continual Learning

In this chapter, we provided the definition of continual learning along with some historical background. Furthermore, The basic terminology was presented, and the challenges that a CL system has to overcome were described. Moreover, a framework for continual learning in classification was introduced, and the three scenarios of continual learning were presented. On top of that, the broad categories of methods for addressing the challenges of continual learning and the state-of-the-art algorithms were explained.

5.0.2 Chapter 3: Self-Organizing Maps

In this chapter, we briefly described the historical background of the SOM algorithm and discussed the notion of vector quantization which was the forerunner of many well-known clustering algorithms such as SOM and KMEANS. Furthermore, the essential background of SOM was presented. Moreover, theoretical analysis was conducted for both the 1D and multi-dimensional SOM. Moreover, the cause and effects of the stochasticity of the online SOM training algorithm were provided, and the deterministic Batch-SOM algorithm was presented. Finally, algorithms and extensions related to the original SOM algorithm were discussed.

5.0.3 Chapter 4: Dendritic Self-Organizing Maps

In this section, we provided the biological background and the motivation for creating a SOM-based architecture to address the challenges of CL. Moreover, we proposed a new SOM-based architecture that utilizes neuroscience and information theory concepts to address catastrophic-forgetting, which is crucial for avoiding model re-training when new data presented. Our experimental results have shown that the DendSOM consistently outperforms the original SOM algorithm in unsupervised classification, which indicates that both the adaptation of receptive fields and dendritic computational locality play a significant role in improving the predictive power of the SOM. Furthermore, the DendSOM is a memoryless easy-to-train algorithm that performs on par with the state-of-the-art datasets task-incremental learning algorithms. Moreover, it achieves a higher accuracy score than most leading-edge architecture in both the domain-incremental and class-incremental scenarios.

Finally, it should be noted that the DendSOM model is trained for one epoch using only one input stream, which corresponds to a batch size equal to one.

5.0.4 Future Work

This thesis presented a SOM-based architecture capable of alleviating the effects of catastrophic forgetting and concept drift. However, there is still a lot of work to be done in this direction. Some possible extensions of this work are presented below:

- The current model can not process RGB images. However, the retina has two types of photoreceptive cells. Rods are active in low light conditions, while Cones contain color-detecting molecules. Thus an extension of this work could be the incorporation of the cone's color-detecting properties into the DendSOM architecture.
- The DendSOM model is a shallow neural network. Although there have been numerous attempts to stack DendSOMs, the results are at best the same as in the case of a single DendSOM since the upper layer DendSOMs propagate their fickle weight vectors to the lower ones. An unsupervised back-propagation algorithm has to be derived to correct these fickle weight vectors.
- In this work, the preprocessing corresponds to image normalization and grayscale transformation. Thus, a possible extension could concern the investigation of other preprocessing procedures [96].
- The use of DendSOM on new hybrid methods.
- As shown in the unsupervised classification experiments, the BMU selection rule can significantly affect the performance of the model. Hence, utilizing metric learning to increase the number of significantly associated or significantly non-associated samples and consequently decrease the fickle ones may further improve the predictive ability of DendSOM both in CL and unsupervised classification.

Bibliography

- [1] T. Branco and M. Häusser, “The single dendritic branch as a fundamental functional unit in the nervous system,” *Current opinion in neurobiology*, vol. 20, no. 4, pp. 494–502, 2010.
- [2] A. Losonczy, J. K. Makara, and J. C. Magee, “Compartmentalized dendritic plasticity and input feature storage in neurons,” *Nature*, vol. 452, no. 7186, pp. 436–441, 2008.
- [3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [4] M. K. Patrick, A. F. Adekoya, A. A. Mighty, and B. Y. Edward, “Capsule networks—a survey,” *Journal of King Saud University-Computer and Information Sciences*, 2019.
- [5] Y. M. Assael, B. Shillingford, S. Whiteson, and N. De Freitas, “Lipnet: End-to-end sentence-level lipreading,” *arXiv preprint arXiv:1611.01599*, 2016.
- [6] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” *arXiv preprint arXiv:1609.03499*, 2016.
- [7] C. Dwork, “Differential privacy: A survey of results,” in *International conference on theory and applications of models of computation*, pp. 1–19, Springer, 2008.
- [8] A. Gepperth and B. Hammer, “Incremental learning algorithms and applications,” in *European symposium on artificial neural networks (ESANN)*, 2016.
- [9] C. V. Nguyen, A. Achille, M. Lam, T. Hassner, V. Mahadevan, and S. Soatto, “Toward understanding catastrophic forgetting in continual learning,” *arXiv preprint arXiv:1908.01091*, 2019.
- [10] M. B. Ring *et al.*, *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.
- [11] S. Thrun and T. M. Mitchell, “Lifelong robot learning,” *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 25–46, 1995.
- [12] Z. Mai, H. Kim, J. Jeong, and S. Sanner, “Batch-level experience replay with review for continual learning,” *arXiv preprint arXiv:2007.05683*, 2020.
- [13] S. Farquhar and Y. Gal, “Towards robust evaluations of continual learning,” *arXiv preprint arXiv:1805.09733*, 2018.
- [14] R. Aljundi, “Continual learning in neural networks,” *arXiv preprint arXiv:1910.02718*, 2019.

- [15] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, pp. 109–165, Elsevier, 1989.
- [16] S. Kolouri, N. Ketz, X. Zou, J. Krichmar, and P. Pilly, "Attention-based selective plasticity," *arXiv preprint arXiv:1903.06070*, 2019.
- [17] L. Itti and P. F. Baldi, "Bayesian surprise attracts human attention," in *Advances in neural information processing systems*, pp. 547–554, Citeseer, 2006.
- [18] S. Kadam, "A survey on classification of concept drift with stream data," 2019.
- [19] B. Pfülb and A. Gepperth, "A comprehensive, application-oriented study of catastrophic forgetting in dnns," *arXiv preprint arXiv:1905.08101*, 2019.
- [20] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.
- [21] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information fusion*, vol. 58, pp. 52–68, 2020.
- [22] G. M. Van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.
- [23] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.
- [24] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.
- [25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [26] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.
- [27] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Networks*, vol. 116, pp. 56–73, 2019.
- [28] G. M. Van de Ven, H. T. Siegelmann, and A. S. Tolias, "Brain-inspired replay for continual learning with artificial neural networks," *Nature communications*, vol. 11, no. 1, pp. 1–14, 2020.
- [29] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.
- [30] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.
- [31] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*, pp. 3987–3995, PMLR, 2017.

- [32] N. Y. Masse, G. D. Grant, and D. J. Freedman, "Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization," *Proceedings of the National Academy of Sciences*, vol. 115, no. 44, pp. E10467–E10475, 2018.
- [33] G. M. Van de Ven and A. S. Tolias, "Generative replay with feedback connections as a general strategy for continual learning," *arXiv preprint arXiv:1809.10635*, 2018.
- [34] P. Zikopoulos and C. Eaton, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.
- [35] R. D. Schneider, "Hadoop for dummies," *John Willey & sons*, 2012.
- [36] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.
- [37] H. Ritter and K. Schulten, "On the stationary state of kohonen's self-organizing sensory mapping," *Biological cybernetics*, vol. 54, no. 2, pp. 99–106, 1986.
- [38] S. Kaski, T. Honkela, K. Lagus, and T. Kohonen, "Web-som-self-organizing maps of document collections," *Neurocomputing*, vol. 21, no. 1-3, pp. 101–117, 1998.
- [39] M. Oja, S. Kaski, and T. Kohonen, "Bibliography of self-organizing map (som) papers: 1998–2001 addendum," *Neural computing surveys*, vol. 3, no. 1, pp. 1–156, 2003.
- [40] H. J. Ritter and K. Schulten, "Kohonen's self-organizing maps: exploring their computational capabilities," in *ICNN*, pp. 109–116, 1988.
- [41] F. Mulier and V. Cherkassky, "Self-organization as an iterative kernel smoothing process," *Neural computation*, vol. 7, no. 6, pp. 1165–1177, 1995.
- [42] T. Kohonen and P. Somervuo, "How to make large self-organizing maps for nonvectorial data," *Neural networks*, vol. 15, no. 8-9, pp. 945–952, 2002.
- [43] R. Gray, "Vector quantization," *IEEE Assp Magazine*, vol. 1, no. 2, pp. 4–29, 1984.
- [44] M. Faundez-Zanuy and J. M. Pascual-Gaspar, "Efficient on-line signature recognition based on multi-section vector quantization," *Pattern Analysis and Applications*, vol. 14, no. 1, pp. 37–45, 2011.
- [45] T. Kohonen, "Self-organizing maps: optimization approaches," in *Artificial neural networks*, pp. 981–990, Elsevier, 1991.
- [46] A. Gersho, "Asymptotically optimal block quantization," *IEEE Transactions on information theory*, vol. 25, no. 4, pp. 373–380, 1979.
- [47] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1551–1588, 1985.
- [48] K. Teknomo, "K-means clustering tutorial," *Medicine*, vol. 100, no. 4, p. 3, 2006.
- [49] N. Grover, "A study of various fuzzy clustering algorithms," *International Journal of Engineering Research*, vol. 3, no. 3, pp. 177–181, 2014.
- [50] K. Rose, E. Gurewitz, and G. C. Fox, "Statistical mechanics and phase transitions in clustering," *Physical review letters*, vol. 65, no. 8, p. 945, 1990.

- [51] T. Kohonen, "Essentials of the self-organizing map," *Neural networks*, vol. 37, pp. 52–65, 2013.
- [52] M. Cottrell, J.-C. Fort, and G. Pagès, "Theoretical aspects of the som algorithm," *Neurocomputing*, vol. 21, no. 1-3, pp. 119–138, 1998.
- [53] J.-C. Fort, "Som's mathematics," *Neural Networks*, vol. 19, no. 6-7, pp. 812–816, 2006.
- [54] L. Tierney, "Introduction to general state-space markov chain theory," *Markov chain Monte Carlo in practice*, pp. 59–74, 1996.
- [55] G. O. Roberts, "Markov chain concepts related to sampling algorithms," *Markov chain Monte Carlo in practice*, vol. 57, pp. 45–58, 1996.
- [56] E. Erwin, K. Obermayer, and K. Schulten, "Self-organizing maps: ordering, convergence properties and energy functions," *Biological cybernetics*, vol. 67, no. 1, pp. 47–55, 1992.
- [57] M. Cottrell and J.-C. Fort, "Etude d'un processus d'auto-organisation," in *Annales de l'IHP Probabilités et statistiques*, vol. 23, pp. 1–20, 1987.
- [58] J. A. Flanagan, "Self-organisation in kohonen's som," *Neural networks*, vol. 9, no. 7, pp. 1185–1197, 1996.
- [59] J.-C. Fort and G. Pagès, "About the kohonen algorithm: strong or weak self-organization?," *Neural Networks*, vol. 9, no. 5, pp. 773–785, 1996.
- [60] V. Tolat, "An analysis of kohonen's self-organizing maps using a system of energy functions," *Biological Cybernetics*, vol. 64, no. 2, pp. 155–164, 1990.
- [61] T. Heskes, "Energy functions for self-organizing maps," in *Kohonen maps*, pp. 303–315, Elsevier, 1999.
- [62] E. de Bolt, M. Cottrell, and M. Verleysen, "Statistical tools to assess the reliability of self-organising maps," *Neural Networks*, vol. 15, no. 8-9, pp. 967–978, 2002.
- [63] N. Bourgeois, M. Cottrell, B. Déruelle, S. Lamassé, and P. Letrémy, "How to improve robustness in kohonen maps and display additional information in factorial analysis: application to text mining," *Neurocomputing*, vol. 147, pp. 120–135, 2015.
- [64] N. Bourgeois, M. Cottrell, S. Lamassé, and M. Olteanu, "Search for meaning through the study of co-occurrences in texts," in *International work-conference on artificial neural networks*, pp. 578–591, Springer, 2015.
- [65] J.-C. Fort, M. Cottrell, and P. Letremy, "Stochastic on-line algorithm versus batch algorithm for quantization and self organizing maps," in *Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop (IEEE Cat. No. 01TH8584)*, pp. 43–52, IEEE, 2001.
- [66] A. M. Kalteh, P. Hjorth, and R. Berndtsson, "Review of the self-organizing map (som) approach in water resources: Analysis, modelling and application," *Environmental Modelling & Software*, vol. 23, no. 7, pp. 835–845, 2008.

- [67] D. A. Brown, I. Craw, and J. Lewthwaite, "A som based approach to skin detection with application in real time systems.," in *BMVC*, vol. 1, pp. 491–500, Citeseer, 2001.
- [68] F. Nan, Y. Li, X. Jia, L. Dong, and Y. Chen, "Application of improved som network in gene data cluster analysis," *Measurement*, vol. 145, pp. 370–378, 2019.
- [69] J. Himberg, "A som based cluster visualization and its application for false coloring," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3, pp. 587–592, IEEE, 2000.
- [70] A. N. Gorban, A. Y. Zinovyev, and D. C. Wunsch, "Application of the method of elastic maps in analysis of genetic texts," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 3, pp. 1826–1831, IEEE, 2003.
- [71] H. Shaban and S. Tavoularis, "Identification of flow regime in vertical upward air–water pipe flow using differential pressure signals and elastic maps," *International Journal of Multiphase Flow*, vol. 61, pp. 62–72, 2014.
- [72] J. Sublime, N. Grozavu, G. Cabanes, Y. Bennani, and A. Cornuéjols, "From horizontal to vertical collaborative clustering using generative topographic maps," *International journal of hybrid intelligent systems*, vol. 12, no. 4, pp. 245–256, 2015.
- [73] I. Casciuc, Y. Zabolotna, D. Horvath, G. Marcou, J. Bajorath, and A. Varnek, "Virtual screening with generative topographic maps: how many maps are required?," *Journal of chemical information and modeling*, vol. 59, no. 1, pp. 564–572, 2018.
- [74] R. L. do Rêgo, A. F. Araújo, and F. B. de Lima Neto, "Growing self-organizing maps for surface reconstruction from unstructured point clouds," in *2007 International Joint Conference on Neural Networks*, pp. 1900–1905, IEEE, 2007.
- [75] R. Nawaratne, D. Alahakoon, D. De Silva, H. Kumara, and X. Yu, "Hierarchical two-stream growing self-organizing maps with transience for human activity recognition," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7756–7764, 2019.
- [76] H. Shah-Hosseini and R. Safabakhsh, "Tasom: a new time adaptive self-organizing map," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 2, pp. 271–282, 2003.
- [77] M. K. Moghaddam and R. Safabakhsh, "Tasom-based lip tracking using the color and geometry of the face," in *Fourth International Conference on Machine Learning and Applications (ICMLA'05)*, pp. 6–pp, IEEE, 2005.
- [78] G. L. Romani, S. J. Williamson, and L. Kaufman, "Tonotopic organization of the human auditory cortex," *Science*, vol. 216, no. 4552, pp. 1339–1340, 1982.
- [79] M. Saenz and D. R. Langers, "Tonotopic mapping of human auditory cortex," *Hearing research*, vol. 307, pp. 42–52, 2014.
- [80] P. Poirazi and B. W. Mel, "Impact of active dendrites and structural plasticity on the memory capacity of neural tissue," *Neuron*, vol. 29, no. 3, pp. 779–796, 2001.

- [81] G. J. Stuart and N. Spruston, "Dendritic integration: 60 years of progress," *Nature neuroscience*, vol. 18, no. 12, pp. 1713–1721, 2015.
- [82] G. Tavosanis, "Dendritic structural plasticity," *Developmental neurobiology*, vol. 72, no. 1, pp. 73–86, 2012.
- [83] M. Bosch and Y. Hayashi, "Structural plasticity of dendritic spines," *Current opinion in neurobiology*, vol. 22, no. 3, pp. 383–388, 2012.
- [84] P. Bashivan, M. Schrimpf, R. Ajemian, I. Rish, M. Riemer, and Y. Tu, "Continual learning with self-organizing maps," *arXiv preprint arXiv:1904.09330*, 2019.
- [85] M. D. Binder, N. Hirokawa, and U. Windhorst, *Encyclopedia of neuroscience*, vol. 3166. Springer Berlin, Germany, 2009.
- [86] M. London and M. Häusser, "Dendritic computation," *Annu. Rev. Neurosci.*, vol. 28, pp. 503–532, 2005.
- [87] N. Liu, J. Wang, and Y. Gong, "Deep self-organizing map for visual classification," in *2015 international joint conference on neural networks (IJCNN)*, pp. 1–6, IEEE, 2015.
- [88] S. Aly and S. Almotairi, "Deep convolutional self-organizing map network for robust handwritten digit recognition," *IEEE Access*, vol. 8, pp. 107035–107045, 2020.
- [89] A. Nakagawa and A. Kutics, "Classification in big image datasets using layered-som," in *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pp. 143–150, IEEE, 2017.
- [90] G. M. Van de Ven, Z. Li, and A. S. Tolias, "Class-incremental learning with generative classifiers," *arXiv preprint arXiv:2104.10093*, 2021.
- [91] R. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.
- [92] S. Taheri and G. Hesamian, "A generalization of the wilcoxon signed-rank test and its applications," *Statistical Papers*, vol. 54, no. 2, pp. 457–470, 2013.
- [93] G. M. van de Ven and A. S. Tolias, "Generative replay with feedback connections as a general strategy for continual learning," *arXiv preprint arXiv:1809.10635*, 2018.
- [94] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.
- [95] G. M. van de Ven, Z. Li, and A. S. Tolias, "Class-incremental learning with generative classifiers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3611–3620, June 2021.
- [96] G. Lowe, "Sift-the scale invariant feature transform," *Int. J.*, vol. 2, no. 91-110, p. 2, 2004.