11-2021

Reducing Catastrophic Forgetting in Self-Organizing Maps

Hitesh Ulhas Mangala Vaidya
hv8322@rit.edu

Reducing Catastrophic Forgetting in Self-Organizing Maps

by

Hitesh Ulhas Mangala Vaidya

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science
in Computing and Information Sciences

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY

November 2021

B. THOMAS GOLISANO COLLEGE OF COMPUTING AND
INFORMATION SCIENCES
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

MS DEGREE THESIS

The MS degree thesis of Hitesh Ulhas Mangala Vaidya
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
MS degree in Computing and Information Sciences

Alexander Ororbia, Thesis Advisor

Travis Desell, Thesis Reader

Rui Li, Thesis Observer

Date

Abstract

An agent that is capable of continual or lifelong learning is able to continuously learn from potentially infinite streams of pattern sensory data. One major historic difficulty in building agents capable of such learning is that neural systems struggle to retain previously-acquired knowledge when learning from new data samples. This problem is known as catastrophic forgetting and remains an unsolved problem in the domain of machine learning to this day. To overcome catastrophic forgetting, different approaches have been proposed. One major line of thought advocates the use of memory buffers to store data where the stored data is then used to randomly retrain the model to improve memory retention. However, storing and giving access to previous physical data points results in a variety of practical difficulties particularly with respect to growing memory storage costs. In this work, we propose an alternative way to tackle the problem of catastrophic forgetting, inspired by and building on top of a classical neural model, the self-organizing map (SOM) which is a form of unsupervised clustering. Although the SOM has the potential to combat forgetting through the use of pattern-specializing units, we uncover that it too suffers from the same problem and this forgetting becomes worse when the SOM is trained in a task incremental fashion. To mitigate this, we propose a generalization of the SOM, the continual SOM (c-SOM), which introduces several novel mechanisms to improve its memory retention – new decay functions and generative resampling schemes to facilitate generative replay in the model. We perform extensive experiments using split-MNIST with these approaches, demonstrating that the c-SOM significantly improves over the classical SOM. Additionally, we come up with a new performance metric α_{mem} to measure the efficacy of SOMs trained in a task incremental fashion, providing a benchmark for other competitive learning models.

Acknowledgments

This journey of pursuing my Master's thesis was a long and arduous one. It was filled with ups and downs, failures, breakdowns, buggy code, shifting research topics and algorithms and missed deadlines. But in hindsight, it has sharpened my personality in total by imparting in me not just technical skills but also things beyond books and algorithms. It has taught me tenacity, equanimity and patience.

I can't thank enough, my advisor, Prof. Alexander Ororbia for guiding me throughout this process. You gave me the opportunity and freedom to pursue my ideas and channelled my thoughts in a structured manner. You gave me exposure to different aspects of academia by involving me in the activities of NAC lab and by allowing me to work as the teaching assistant for your courses. You kept me motivated and encouraged me to pursue a Ph.D. in Machine Learning. I am thankful to you for all your support.

I am grateful to Prof. Travis Desell for giving this thesis the concrete direction of Self Organizing Maps. You gave this work a proper structure and procedure. Thank you for patiently teaching me how to weave a proper story and explain things in a definitive flow. Special thanks to Ankur Mali for your valuable suggestions from time to time. I appreciate all of the NAC lab and D2S2 lab members for their precious feedback. Thank you James Craig (Linus) for managing all the system hardware and solving all of the server issues. I could obtain the results only because of the server's availability and your support. Thank you Prof. Reynold Bailey and Prof. Joe Geigel for giving me access to the cloud computing lab. I also want to mention and thank my friends and room mates (Gucci Gang) for sticking together and making my graduate life a celebration.

Lastly, I would like to express my gratitude towards the Computer Science department at the Rochester Institute of Technology for giving me a wonderful experience through this graduate program.

I offer this work at the feet of my Guru (Saint Shree Gajanan Maharaj) and my family. In the words of Shree Das Ganu Maharaj,

"A pen is only a means to do that writing. Same is the case here; I have become a pen and the writer is Shree Gajanan Maharaj. It is by His grace, that I have written this ... O listeners, I owe no credit for this at all."
Shree Gajanan Vijay Granth, 20:207, 20:208

"O Swami Gajanan, whatever I have said here, is entirely as per your inspiration."
Shree Gajanan Vijay Granth, 21:47

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human beings have the capability to learn continuously throughout their entire lifetimes [Parisi et al., 2019]. Modern applications like self-driving cars contain intelligent systems or neural networks that receive data from multiple sources in multiple forms. In a practical environment they no longer receive data from single task apriori. In fact, neural networks in such machines receive streams of data belonging to different tasks in a random order. It is expected that a neural network would extract knowledge from these data streams and use them to generalize its knowledge base. In fact, one of the qualities of a generalized neural network is that it extracts knowledge from one task and uses it in the learning of some other task. This process continues in real-time due to the nature of the mode of operation of such neural networks in applications like self-driving cars or voice assistants. This characteristic of neural networks to learn things belonging to different categories or classes in a never-ending manner without forgetting them motivates the enterprise of continual or lifelong learning [Silver et al., 2013], [Thrun and Mitchell, 1995], [Chen and Liu, 2016].

Statistical learning models, at present, struggle to perform such efficient learning without completely forgetting old knowledge. Solving the problem of forgetting still remains a grand challenge in the field and will be the focus of this thesis.

1.1 Motivation

In 2012, the work presented by [Krizhevsky et al., 2012] demonstrated impressive results that brought back machine learning to the attention of the computer science community. After this work, there was an substantial rise in the amount of papers published in this field and, consequently, computer scientists started pursuing topics in machine learning. Computer vision remains one of the fastest evolving area with numerous papers being published every day. One of the breakthroughs in the field of machine learning occurred when [Silver et al., 2016] (AlphaGo) beat Lee Sedol in the game of Go. Demis Hassabis, who was the CEO of Deepmind (company that developed Alphago) had issued a press statement explaining that AlphaGo was developed with the idea of building a general purpose AI. This sparked the idea of working upon an algorithm that is not restricted to just one problem statement. As explained earlier, continual learning is one such sub-domain of machine learning that follows this idea of building intelligent systems that could solve objectives from multiple tasks. One of the main advantages of having such a system is that it could be deployed on edge computing devices that are not rich in terms of hardware resources. A robust continual learning based system can solve problems with speed and accuracy on edge computing devices. Moreover, a continual learning based neural network can be upgraded to identify new task mappings while maintaining its current knowledge. These properties of continual learning motivated us to pursue a problem in this field.

Further, we realized that there is still a fundamental problem in continual learning that still remains unsolved – catastrophic forgetting [French, 1999], [McCloskey and Cohen, 1989]. Unlike a human brain, an artificial neural network (ANN) cannot update its knowledge dynamically. To consider a very simple example, a simple feedforward network trained on the MNIST dataset [Lecun et al., 1998] performs well only on MNIST. If this neural network was upgraded on another dataset like Fashion-MNIST [Xiao et al., 2017], it will forget its older knowledge about MNIST. In order for it to recognize images from both the datasets, it has to be retrained from scratch on input containing a mixture of both the datasets passed as input. An ideal artificial neural network that replicates human brain like behaviour would not need to be trained on both the datasets again. Instead, it will just need the new data samples from Fashion-MNIST and parallelly able be preserve its older map-

pings from MNIST. Such a neural network would have the ability to augment its knowledge gained from a new dataset without compromising its mapping from the previous one. In modern times, there is a need for such a behaviour in various applications like self-driving cars, stock market prediction systems and voice assistants.

The goal of continual or lifelong learning is to be able to upgrade and augment the knowledge of an artificial neural network. It could be attained by using novel architectural or training approaches.

Self-organizing maps (SOM) are a type of neural network that are trained using competitive learning. They are widely used in dimensionality reduction or clustering methods [Vesanto and Alhoniemi, 2000]. The fundamental structure of SOMs is loosely inspired from the working of human brain, more specifically retina-cortex mapping. [Yin, 2008a]. This makes them more biologically inspired and thus a favourable choice for continual learning. As with other neural networks, they perform well on independent and identically distributed (i.i.d.) data. Also similar to other neural networks, we have experimentally found out that SOMs perform very poorly if trained in a lifelong learning fashion. This thesis also introduces a great potential of SOMs in the form of generative networks in lifelong learning domain due to their simplicity. That is why, we chose SOMs for the scope of this thesis.

1.2 Lifelong Machine Learning

Lifelong learning or continual learning [Thrun, 1995, Ring, 1994] is a process in which an intelligent agent or model receives continuous streams of pattern vectors, that might come from multiple different tasks, from which it must learn and aggregate knowledge from. In practical real-world scenarios, where memory storage is limited, often the constraint is introduced where a continually learning model is allowed to see the training data just once (thus only one epoch or pass through the data is permitted), after which the data is lost forever, e.g, this is a form of online learning (where data is processed one sample at a time). This makes building continually-learning agents even more difficult. Models trained under this particularly challenging setting often suffer from a phenomenon historically known as *catastrophic forgetting* [French, 1999]. A statistical learning model in the lifelong learning setting tends to forget the knowledge that it has acquired in the past, only remembering recent informa-

tion learned in current or neighbouring time steps. This is a part of a larger problem called the *stability plasticity dilemma* [Grossberg, 1982, Abraham and Robins, 2005]

There are multiple challenges [Schwarz et al., 2018, Javed and White, 2019a] faced by a continual learning system, described as follows:

- The system should not suffer from catastrophic forgetting, i.e. it should retain knowledge gained for each task and perform well as it progresses further.
- It should be able to extract knowledge from the current task and use it to aid in learning upcoming ones. This is also known as *positive forward transfer*.
- Learning current tasks must have a positive impact on the recognition of previous tasks, which is also known as having *positive backward transfer*. At minimum, a backward transfer of zero means old knowledge was at least not destroyed.
- It should be scalable.
- It should be able to learn without the explicit provision of task descriptors or task boundaries as a part of an input dataset. Which means, it should be able to determine these parameters internally.

Lifelong machine learning systems must satisfy at least the above items in order to be useful for intelligent systems in real-world applications. As mentioned in Section 1.1, we need lifelong machine learning systems in order to develop general purpose intelligent systems. This is because, it is not ideal in every case to have multiple sub-networks performing different tasks and that we train all of them from scratch. Such a system causes wastage of time and resources. Instead, a system designed by keeping the above mentioned objectives in mind would help us to have fewer neural networks that have the ability to share and reuse the knowledge gained while solving one task to solve another. This would help us build systems that have low space and computational complexity and would run efficiently at speed on low resource devices. There are various approaches used for development of such kind of system. A detailed description of these works is presented in Chapter 2.

In this thesis, we intend to find and develop a neuro-cognitively inspired method for reducing catastrophic forgetting in neural systems, focusing on the domain of unsupervised learning. Along with this, we wish to investigate if, when a continually learning agent extracts information related to one single

class, it is capable of learning subsequent ones more readily. In essence, we aim to design agents that can leverage knowledge gained from training on one class when attempting to model and learn about new, subsequent encountered classes.

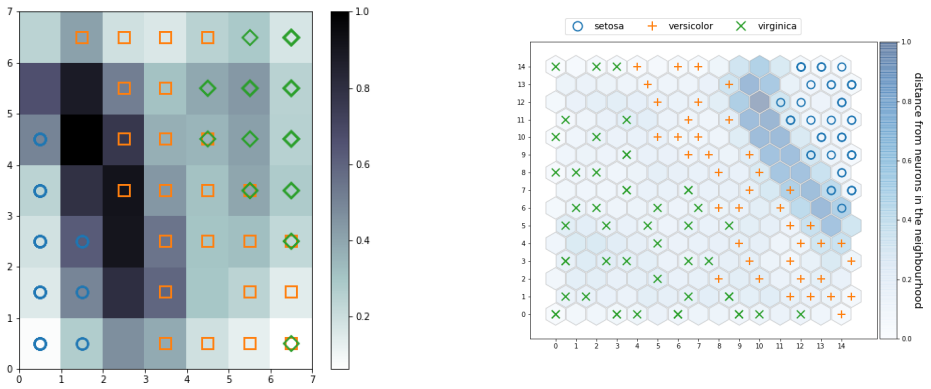
Recently, SOMs have started being used in continual learning. Self-organizing incremental neural networks (SOINN) [Furao and Hasegawa, 2006] are a class of algorithms that use growing SOMs to tackle the problem of task incremental learning. [Lu et al., 2014], [Wiwatcharakoses and Berrar, 2020], [Wiwatcharakoses and Berrar, 2021] propose further advanced versions such as the SOINN-RBF, SOINN+, GSOINN+, respectively. However, for the purpose of this thesis, we focus on fixed-size/capacity SOMs. SOMs, due to their fundamental architecture can be used to consolidate task-specific data in lifelong learning. The trained units of the SOMs that store this consolidated task specific information could be used to perform replay/rehearsal in a lifelong learning model. Most replay approaches use buffers to store clusters of raw/physical data and then retrain the network on this stored information. However, having memory buffers imposes memory constraints (and it seems unlikely that human brains store raw data directly). SOMs, as we will show in this thesis, can be extended to give us topological covariance among a certain class' data samples. Therefore, SOMs might prove to be a more appropriate choice for a lifelong learning system.

1.3 Problem Definition

SOMs were first introduced by Teuvo Kohonen [Kohonen, 1990], which is why they are also referred to as Kohonen maps or Kohonen networks. It is sometimes argued to be computationally the closest abstracton of how memory could be arranged in a human brain [Dresp, 2020], [Kohonen, 1997], [Yin, 2008b]. Despite being an artificial neural network (ANN), it is trained using a competitive Hebbian learning [Yin, 2008b, Choe, 2013] algorithm instead of backpropagation.

The goal of this thesis is to reduce forgetting in SOM models. An SOM contains a set of neuronal units that have the same vector dimension as that of their input samples. These units are arranged in a topology or structure, often square or hexagonal in shape. U-matrices of these topologies are presented in Figure 1.1. At training time, these member neurons or units compete

with each other to match with the given input. The one neuron that wins is called *best matching unit* (BMU). It performs a weight update over itself based on the difference between its activity values and the given input pattern. After this, the SOM unit shares this opportunity in updating weights with its neighbouring units. The weight update decreases exponentially as the units get farther away from BMU. As the training progresses, the competition increases further among the SOM's units to win. With this, the neighbourhood of BMUs in each training step also decreases to reduce the collaboration and thus win more than other units. Thus, at the end of the training phase, each unit assumes the form of a certain class of inputs. Thus, each SOM unit, after training is completed, can be viewed as the cluster center for a certain class of inputs.



(a) Square Topography. (b) Hexagonal Topography.

Figure 1.1: SOM on the Iris dataset, obtained using minisom [Vettigli, 2018].

SOMs will be explained in more detail in section 3.1. A group of SOM units learn representations that capture general patterns about the input data. For example, if an SOM had 100 units and it was trained on MNIST, ideally 10 out of them would capture each digit class from MNIST. Assuming the variance among the classes is similar, these 10 units would generally represent a summary of all the samples belonging to that class which are present in

the input dataset. In the lifelong learning setting, multiple tasks are passed as input to the model in an incremental fashion, where each task could be treated as a separate dataset in general. A task thus contains several classes within itself. When SOMs are trained in such a manner, they forget the representations learned from previous tasks in the same manner as neural networks. In this thesis, we will propose a novel method through which the problem of forgetting in an incrementally-trained SOM is noticeably reduced.

Chapter 2

Literature Review

In this chapter provide a brief survey on the various topics and concepts that will be used in this thesis. The main problem are that this thesis will focus on is known as lifelong learning, continual learning or never-ending learning [Chen et al., 2018]. Domains like meta-learning, multi-task learning, or transfer learning also share similarities with lifelong learning in some aspects.

As described earlier, the main problem of catastrophic forgetting occurs in a lifelong learning system due to the stability-plasticity dilemma [Redondo and Morris, 2011]. The tendency of a network to update or change its weights as it encounters new data is called *plasticity* whereas the contrary is called *stability*. The network has to maintain a balance in order to keep the older representations and learn new ones at the same time. This is common problem when the network receives data incrementally or if the network capacity is low. There are numerous approaches that have been proposed so far, which, at their core, try to attain a balance between the stability and plasticity of a system overall.

Continual learning has been extensively studied in its application to various fields such as robotics [Thrun and Mitchell, 1995], recurrent neural networks [Ehret et al., 2021], [Cossu et al., 2021], visual question answering [Greco et al., 2019], and graph neural networks [Carta et al., 2021], [Liu et al., 2021]. This thesis focuses on reducing the problem of catastrophic forgetting in the case of unsupervised learning, specifically with respect to the SOM neural model. Table 2.1 compares all the approaches in this domain.

Methods	approach	sparsity in model	computational		scalable
			space	compute	
[Kirkpatrick et al., 2017]	regularization	✓	medium	high	×
[Hayes et al., 2019]	rehearsal	×	high	high	✓
[Mallya and Lazebnik, 2018]	parameter isolation	✓	less	high	×
[Ayub and Wagner, 2021]	rehearsal	×	less	high	✓
c-SOM (our approach)	rehearsal	✓	medium	high	✓

Table 2.1: Comparison of various approaches to tackle catastrophic forgetting in continual learning.

2.1 Continual Learning Approaches

Continual learning has used several approaches such as those based on dropout [Srivastava et al., 2014] and through the use of different activation functions [Goodfellow et al., 2014]. There have been many efforts published about using dual architectures [Shin et al., 2017, Kang et al., 2020] and using meta-learning or few-shot learning [Javed and White, 2019b,Gidaris and Komodakis, 2018a]. Other approaches include reducing representational overlap [French, 1992, French, 1994]. More related to this thesis, there has been a strong focus on replay or rehearsal methods which we discuss in detail in the upcoming sections.

Bayesian Approaches to Continual Learning [Kirkpatrick et al., 2017] use the Fisher information matrix [Ly et al., 2017] to obtain the importance of model weights, while mode-IMM [Lee et al., 2017] uses variance information to selectively balance between two sets of weights. According to [Lee et al., 2017], since a large number of pixels in MNIST have values close to zero, the fisher information matrix works in the case of EWC. However, they argue that using only the diagonal of a variance matrix is a very naive way to tackle the problem, which is why EWC fails in case of a disjoint MNIST experiment.

Continual learning has used several approaches based on: 1) regularization, 2) rehearsal, and 3) architectural or parameter isolation. [De Lange et al., 2021] provides a taxonomy of the various approaches in continual learning in addition to comparing the methods that fall under the above three categories.

2.1.1 Regularization Based Approaches

Regularization [Kirkpatrick et al., 2017], [Aljundi et al., 2018], [Li and Hoiem, 2018] introduced a new regularization term to the loss function. It focuses on preserving task-specific weights by either controlling the weight update or using distribution of weights from previous task as prior to learn incoming new task data. This approach can be further divided into two methods called data-focused and prior-focused methods.

Data-Focused Methods

In these sets of methods, knowledge is distilled from one model to another which means that knowledge from one task is used to learn another task. Either a bias is created from the previous task or soft labels are extracted from previous tasks to then be used to learn patterns from the current task. The model from which knowledge is distilled is trained on previous data and the model in which the knowledge is transferred to is trained on the upcoming/current task data. [Schwarz et al., 2018, Michieli and Zanuttigh, 2021, Castro et al., 2018, Rajasegaran et al., 2019] represent some key works that use knowledge distillation. The limitation of this approach, however, is that it is prone to domain shift between tasks [Aljundi et al., 2017]. The success of this approach depends on the degree of inter-task similarity. Dissimilarity among tasks leads to a rise in task error.

Prior-Focused Methods

In this set of methods, importance of weights are learned for the current task. When the next task dataset arrives, this importance is used to avoid a drastic shift of the weight values in the model acquired from the previous tasks. Generally, the importance of all neural connections is used in this method. EWC [Kirkpatrick et al., 2017] is the most popular work that embodies this methodology. Additionally, [Zenke et al., 2017, Aljundi et al., 2018, Lee et al., 2017] also use this method. However, this method of adding a soft penalty does not scale well to large scale datasets because changes to previous parameters are penalised while training on later tasks. Once the neural model encounters a certain number of tasks, it will not be able to preserve all of the older parameters and keep on forming newer synaptic connections. A moderately sized neural network using this method does not cope well with continuous streams of data [Farquhar and Gal, 2018].

2.1.2 Rehearsal Based Approaches

Replay or rehearsal [Hayes et al., 2019, Rebuffi et al., 2017, Lopez-Paz and Ranzato, 2017, Isele and Cosgun, 2018] stores exemplars which are data points that are representative of a cluster of class-specific data, in memory/buffers. These exemplars of past tasks are passed again to the model while it trains

on the current task data in order to help it to remember historic mappings. The argument made in support of this approach is that replay/rehearsal occurs in the human brain when we are asleep [Hayes et al., 2021, Dhar et al., 2019]. However, [Ostapenko et al., 2019] rejects this argument by saying that such an approach is not biologically inspired and, moreover, due to memory constraints, such an approach is not scalable. It is also computationally expensive as it requires computing a forward pass through the old task's network for every new data point [Zenke et al., 2017]. This approach is comparatively more difficult to implement in the case of data streams with no task descriptor (vectors that tell the system what the task's goals are or what is to be done in the given task) because the detection of task boundaries and the classes to which input data belongs to is a hard problem. Detection of spikes in the neural activity patterns as the input data changes is not an accurate way to detect such class boundaries [1]. It can be empirically shown that, quite often, such spikes may be false-positive or false-negative. Like in the case of MNIST, there might not be a spike when classes change from digits 1 to 7. Since the memory at hand or in the buffer is of a fixed size for the mode, as the number of tasks increase, the exemplars representing each task must be shuffled or reselected. Either upcoming tasks are paired with other virtual clusters based on class similarity or new clusters are formed. However, this may put a toll on the size of each task cluster. Overfitting [Lee et al., 2019] is another issue with replay/rehearsal. However, data privacy is also an issue in such approaches as pointed out by [Wu et al., 2018, Dhar et al., 2019]. If a task/class occurs rarely, then it may fall out of buffer [Lipton et al., 2016].

To address the issues of rehearsal, several other approaches such as pseudo-rehearsal that uses a generative network such as one based on the variational autoencoder (VAE) [Ororbia et al., 2019, Ororbia and Kifer, 2020, Shin et al., 2017, Ayub and Wagner, 2021, Venkatesan et al., 2017], attention mechanisms [Serra et al., 2018], and schemes based on few-shot visual learning [Gidaris and Komodakis, 2018b] have been proposed.

2.1.3 Architectural Approaches

Architectural approaches alter the neural model architecture itself to avoid forgetting. There have been efforts to build dynamic neural networks in com-

[1] In communication with William Gebhardt (wdg1351@rit.edu)

bination with generative models [Ostapenko et al., 2019]. [Yoon et al., 2018,lai Li et al., 2019] mentions dynamically expanding neural networks whose layers or nodes could be intelligently increased to cope the increasing size of input dataset. Spacenet [Sokar et al., 2021] and Packnet [Mallya and Lazebnik, 2018] are methods proposed use compression techniques to create space for learning new incoming tasks continually. However, they can suffer from scalability issues, i.e., this approach, in general, increases the architectural complexity of a model [Zenke et al., 2017]. Progress and compress [Schwarz et al., 2018] applies a unique architectural approach to combat this issue of expanding networks by trying to reduce architectural complexity. [De Lange et al., 2021] states that such approaches require a task oracle to activate a task branch/sub-network or to manipulate masks during prediction – this restricts such methods to a multi-head setup. Furthermore, this setup is restricted to a task incremental setup working only when the model size is large and performance is a priority.

There is another approach that utilizes architecture modifications to avoid replay or rehearsal. Exemplars/centroids of classes or tasks are stored using consolidation or clustering techniques. New samples are generated from these centroids and passed to the model along with new input data at current time step. In this way, the model is reminded about its past input-output mappings to avoid catastrophic forgetting. This is an under-explored area and has much potential, as we will see in this thesis. Encoding episodes as concepts (EEC) [Ayub and Wagner, 2021, Ayub and Wagner, 2020] uses this unique technique to consolidate knowledge of a task in the form of a centroid using autoencoders. Images belonging to previous tasks are then generated from these centroids which are passed further to the model while training it on new tasks. In this way, an effort is made to preserve the historic connections in hidden layers.

In expanding deep neural networks (DNNs), one solution is to keep Θ_{t-1} fixed when learning $\Theta_t = \Theta_{t-1} \cup \theta_t$ to avoid catastrophic forgetting completely, where θ_t are the new parameters introduced/learned for new task T_t. In regularization approaches or EWC [Kirkpatrick et al., 2017] method use statistically-inspired functions like the fisher information matrix to control the change of parameter values from Θ_{t-1} to $\Theta - t$. Such approaches are good at preserving parameters for initial tasks by controlling the weight updates for upcoming tasks. However, these approaches are not robust and are not effective for large number of tasks. As we will see in this thesis, our SOM-based approach combines the idea of rehearsal in a fixed-capacity model with

task-aware mechanisms to better deal with the stability-plasticity dilemma.

2.2 Competitive Learning

Competitive learning [Hartono, 2012] is a type of unsupervised learning where neurons compete to match with a given data input vector. It is a variant of Hebbian learning [Choe, 2013]. It is generally used for finding clusters in data. Competitive learning includes algorithms such as vector quantization and self-organizing maps (Kohonen maps). Unlike noncompetitive learning algorithms, where all the neurons take part in the learning procedure in each learning step, in competitive learning, only the neurons that satisfy certain criteria get the right to proceed ahead and update their relevant synaptic weight values. This type of learning could help to identify useful features for a data mapping in models like deep neural networks.

[Rumelhart and McClelland, 1987] gives three basic elements for competitive learning rules:

1. Start with a set of units that are highly similar except for some random noise which makes each of them respond slightly differently to a set of input patterns.

2. Limit the 'strength' of each unit.

3. Allow the units to compete in some way for the right to respond to a given subset of inputs.

The general flow for a competitive learning algorithm is shown in Algorithm 1.

Algorithm 1 Competitive learning algorithm

1: Normalize all input patterns
2: Randomly select a pattern $x^{(n)}$
3: Find the winner neuron
4: $i = arg \max_j \left[W_j^T X^{(n)} \right]$
5: Update the winner neuron
6: $W_i = W_i + \eta X^{(n)}$
7: Normalize the winner neuron
8: $W_i = \frac{W_i}{\|W_i\|}$
9: Go to step 2 until no changes occur in N runs

where, W is the weight vector of SOM unit, η is the learning rate, and X is the input data sample

Chapter 3

Self-Organizing Maps

In this chapter, we examine some of the key concepts needed to understand the core topics of the SOM. The proposed model is presented in the next chapter.

3.1 The Kohonen Self-Organizing Map Model

Self-organizing maps (SOMs) learn via a Hebbian update rule [Morris, 1999], which is one reason why they can be considered as a closer approximation of how learning might occur in the human brain. This worked nicely with our motivation to develop a more neurocognitively-plausible neural model for the setting of continual learning. In an SOM, spatially-arranged clusters are formed gradually around s over the course of the model's training. Each unit in the SOM could be a centroid or cluster center of input samples. The SOM can also be used as a data exploration technique, wherein the clusters in it each represents summary of a certain class of data. Unlike the K-Means algorithm, SOMs perform soft clustering over the input data i.e. the weight update is maximum for the BMU and it gradually decreases in the general neighbourhood. SOM units become similar to their input patterns during the iterative training process. This process is more intuitive than having buffers for storing clusters or exemplars of given task's data in buffers. Intuitively, the SOM learning scheme is better equipped to helps us catch the subtle differences or variances in a task's data patterns.

3.2 The Standard SOM Algorithm

3.2.1 Variable Definitions

The variables presented are scalars unless presented in bold.

- m is the number of units in SOM
- d is the dimension of each SOM unit
- s is the current iteration of training, where $0 < s < S$
- τ is the time constant used for decaying (σ, λ)
- E is the number of epochs
- i is the index of mnist image in the task specific dataset $\mathcal{D}_{train}^{(t)}$. For conciseness, we refer this as x_i as ith image from $\mathcal{D}_{train}^{(t)}$
- v is the index of the node in the map
- $\boldsymbol{W}_v$ is the weight vector of unit v
- u is the index of the Best Matching Unit (BMU)
- λ_t is the decayed learning rate for current task, t
- σ is the radius of neighbourhood of BMU
- σ_t is the decayed radius σ for current task, t
- $h(\sigma)$ is the neighbourhood or influence function for learning in the vicinity of BMU. It is as a function of current task, t
- $\phi(v)$ gives us the count of number of times unit v is assigned to any class.
- ω_v stores the running variance of W_v i.e. the difference between pixel values of W_v before and after training step
- ρ_v stores the class name to which unit v is trained. Example, Wv has been trained and appears like mnist digit 0 then $\rho_v = 0$.

3.2.2 The Process of Neural Competition

In this stage, there is competition amongst SOM units for being chosen as the BMU. The way to select a BMU is as follows:

$$u = arg\min_{v} \|W_v - x_i\| \qquad (0 < v < m) \tag{3.1}$$

Another way to find out BMU is by using dot product which follows the principle of Hebbian theory,

$$u = arg\max_{v} (W_v \cdot x_i) \qquad (0 < v < m) \tag{3.2}$$

Generally, however, using a dot product generally favors normalizing the data to some suitable range (in order to ensure that the dot product is maximal for the BMU). For the first variant, the BMU can be found using various other distance metrics such as MSE loss (or Euclidean distance) or Charbonnier loss [Charbonnier et al., 1994].

3.2.3 The Process of Neural Collaboration

Once the BMU is chosen, it is time for the BMU to collaborate with its neighbouring units. Initially, the collaboration is higher when there is less competition for a winning unit. As the number of training iterations increases, the competition increases and the collaboration decreases. Therefore, the BMU progressively collaborates with smaller neighborhoods. This occurs because of the decay used in both the neighbourhood radius of the BMU and the learning rate of the weight update in the new neighbourhood [Kohonen, 1990].

On Setting the Radius σ

This is one of the most crucial parts in training a SOM. A decay function controls the spread of a class in a SOM. Both σ, or the radius of training neighbouring units of the BMU, and λ, or the learning rate of training (this controls the strength of the Hebbian weight updates), decay as we progress throughout the course of training. Decay for a standard SOM is updated for each iteration, which can be done in the following ways:

$$\lambda(iteration) = \frac{\lambda}{\left(1 + \frac{s}{(S)}\right)} \tag{3.3}$$

$$\lambda(iteration) = \lambda \ \exp\left(-\frac{s}{\tau}\right) \tag{3.4}$$

In case of the standard SOM, the above decay function is applied to both λ and σ.

The Neighbourhood Function

The neighbourhood function controls the extent to which a BMU will share its learning with its neighbouring units. This is done in the following way for

a vanilla SOM:

$$h_{i,j}(d_{i,j}) = \exp\left(-\frac{d_{i,j}}{2\sigma^2}\right) \tag{3.5}$$

where, i, j are units in the SOM, and $d_{i,j}$ is the lateral or topological distance between unit i and unit j and $h_{i,j}(d_{i,j})$ is the neighbourhood function. Apart from Gaussian, several other functions like the bubble [Natita et al., 2016] or the mexican hat can be used as neighbourhood functions. An example of mexican hat function is as follows:

$$h_{i,j}(d_{i,j}) = \left(1 - \frac{d_{i,j}^2}{\sigma^2}\right) \exp e^{-\frac{d_{i,j}^2}{2\sigma^2}}. \tag{3.6}$$

Equation 3.5 is applied when the units are randomly arranged and the distance between them is topological in nature. This means, the distance is calculated based on the pixel values of units. There exists a variety of SOM models where the units are arranged in an euler plane and hence obey the laws of cartesian coordinate axes. That means we consider the row and column index of units to calculate the Euclidean distance between two units. The former condition is applied for the neural gas model [Martinetz and Schulten, 1991, Fritzke, 1994] whereas the latter one is use by Kohonen's SOM. Equation 3.5 is still applied in the same way independent of whether or not the model units are arranged in a structured topology or not.

3.2.4 Weight Udpate

Based on all of the above equations, we can write the SOM's weight update rule as follows:

$$W_v(s+1) = W_v(s) + h(u, v, s) \cdot \lambda_t \cdot (x_i - W_v(s)) \tag{3.7}$$

where, $h(u, v, s)$ is the neighbourhood function based on the distance value, σ between units u, and unit v at iteration s. Algorithm 2 shows the steps to execute standard SOM.

The function on line 13 of Algorithm 3 gives us the radial function for the distance of all the units from a selected best matchine unit (BMU). Further working of the SOM is given in chapter 3.

Algorithm 2 The Standard SOM Algorithm

Input: i.i.d. MNIST data, D

Parameter: W_v, σ, λ

for $s = 0$ to n($iterations$) **do**
    x_i = Randomly pick an input vector, $D(index)$
    $u = \arg\min d = \{\|W_v - x_i\| \mid \forall\ v\ \epsilon\ V\}$
    $\sigma_s = \sigma \exp(-s/\tau)$
    $\lambda = \lambda \exp(-s/\tau)$
    d = distMatrix(u, v, σ)
    $h(d) = \exp(-d/2\sigma_s^2)$
    $W = W + h(d) * \lambda_s * \|W - x_i\|$
end for

Algorithm 3 Pseudocode for storing precomputed distances between SOM units

Input: (x,y) coordinates of SOM units, u, v, and σ

function preCompute(m)
    for $x1 = 0$ to m **do**
        for $y1 = 0$ to m **do**
            for $x2 = 0$ to m **do**
                for $y2 = 0$ to m **do**
                    $node\_dist[x1, y1, x2, y2] = (x1 - x2)^2 + (y1 - y2)^2$
                end for
            end for
        end for
    end for
end function
node_dist = preCompute(m)
function distMatrix(k, p, σ_t)
    nd = node_dist[k, p, :, :]
    denom = $2 * \sigma^2$
    return $\exp(\frac{-nd}{denom})$
end function

Chapter 4

Lifelong Learning Kohonen Neural Systems

In this section, we develop self-organizing map (SOM) models for the continual learning setting. Specifically, all extensions that we propose fall under the general SOM family that we label as the continual SOM (or c-SOM). First, we start with our motivation for choosing the SOM.

[Dresp, 2020] mentions seven properties of SOMs based on functional investigations of the human brain. They are: 1) modular connectivity, 2) unsupervised learning, 3) adaptive ability, 4) functional resiliency, 5) functional plasticity, 6) from-local-to-global functional organization, and 7) dynamic system growth. SOMs are interesting candidate models for lifelong learning because:

1. SOMs are simple and effective networks for clustering that have not been explored in continual learning for consolidating task data.
2. Since the SOM follows a competitive learning approach, which is a branch of hebbian learning, it is biologically inspired.
3. It exhibits a lower space complexity and has the potential to scale (depending on the implementation)
4. [Bashivan et al., 2019] used SOMs to cluster previous input images for a neural network to maintain their experiences. The SOM learned an input distribution of each task and formed an output mask for every input to the neural network based on the distance between input and the weights of SOM. This mask was then multiplied with the output of the fully

connected layer of the feedforard neural network. Using this process, the network allocated nodes to the input-output associations per task and masked out irrelevant nodes. In this way, the SOM shared nodes between similar tasks and separated nodes for dissimilar tasks. According to this thesis, the idea of forming masks based on learned representation of input distribution does not address the stability plasticity dilemma for larger datasets. Rather, this approach only addresses the representations of the input layer by totally ignoring the post activations of the feedforward network's hidden layers.

5. Hebbian learning can be summarized as, 'neurons wire together if they fire together' [Löwel and Singer, 1992]. This same principle applies for how neurons are trained in the human brain. In a human brain, knowledge is spread across neurons that match closely or have similar activations [Gepperth et al., 2015]. This does not mean that the neurons necessarily have to be in the same physical neighborhood, however they are in the same topological neighborhood. Therefore, instead of storing data clusters in a buffer or any data structure, storing memory or data-label mapping in a neural structure like a SOM is more biologically plausible. More about neighborhoods will be explained in Section 4.2.
6. SOMs are of two types: i) static SOMs and ii) growing SOMs or neural gas. With growing SOMs, we can scale them according to our choice of geometry. This means, SOMs' units can either be arranged in the form of a square (which is the most popular arrangement used) or a hexagon (see Figure 1.1 for its visualization). Moreover, we can create more space in SOMs by emptying redundant or similar units. This can also be termed as selective forgetting in order to create space or room for new knowledge. This same principle is also used in the human brain.

4.1 Self-Organizing Maps for Lifelong Learning

Many approaches mentioned in Section 2.1.2 propose storing task data exemplars in memory buffers. These exemplars are then passed again as input to a lifelong neural model to remind it about previous tasks. [Ostapenko et al., 2019] describes that retaining real samples is very much against the notion of bio-inspired design, as natural brains do not feature the retrieval of information identical to originally exposed impressions.

There are multiple challenges in solving this problem. First of all, there is no mathematical way to derive the best parameters for SOMs. This includes the best size and topology for SOM for set of input data, initial neighbourhood radius and learning rate of weight update in it. Due to bad selection of σ and λ, there can be an explosion of weight updates in the neighbourhood of BMUs. This means, if a unit was selected as BMU, an extensive number of units in its neighbourhood received a weight update. Due to multiple iterations of training, the neighbourhoods expand even further which leads to all the units in the SOM receiving the weight update unrestrictively. As a result, all the units in the SOM assumed same pixel values as the current class label. In an incremental learning paradigm, this led to replacing of older trained unit pixel values with newer current class' pixel values. Thus, SOMs became insufficient in maintaining old trained units.

Moreover, when assigning SOMs for vision tasks, such as MNIST where all the images in the dataset have common black background, the majority of pixels from the input image matched with already trained (fully or partially trained) unit instead of opting for a new unit so as to avoid forgetting. To solve this problem, we had to allow large number of units get trained on initial task's data. Consecutively, we had to decay the parameters so that the current or upcoming tasks do not overspread their learning or weight update and replace already trained units on previous tasks with current task patterns. If the current task exploded their training among SOM's units, we perform replay over old task data to preserve their patterns in SOM's units. Since SOMs perform soft clustering, many of their trained units form blurry images. Further, SOMs trained incrementally with decaying (σ, λ) values had decreasing number of trained units per class. Therefore, we had to come up with approaches that would lead to equal number of trained units per class in an SOM. Such an algorithm that reduces forgetting and provides equal number of trained units per input class targets to produce an ideal SOM for split-MNIST dataset.

The following are the objectives that we wanted to solve with respect to continual learning SOMs:

- A vanilla SOM maintains topological neighbourhood of input space if it is provided with iid (independent and identically distributed) data at once. However, it fails miserably in an online learning setting. If it is trained in a task incremental fashion on a split-MNIST dataset, it

remembers only the last task i.e. digits 8 and 9 with the addition of few units remembering digit 6. It was a surprising to know that a SOM remembers just digit 6 from previous classes apart from the last task of split-MNIST.

- We aimed at coming up with a method that not just remembers all the tasks in a fixed sized SOM but also assigns nearly equal number of trained units to each task.
- We wanted to avoid the concept shift [Gepperth and Karaoguz, 2017] caused in SOM due to online learning.
- It is equally important to determine the class label of each trained unit in SOM on the fly. We tried different methods for this which we discuss below

4.2 Task-Incremental Self-Organizing Maps

The classical vanilla SOMs are trained on all data samples at once. Contrary to this, we train a SOM in a task incremental manner wherein, each task contains multiple classes of input data points. We describe in chapter 3 how SOMs are trained in their vanilla form and then we describe our method of training them in Section 4.2.7.

We train the SOM using a stochastic process. This means, we train the SOM on one randomly selected task dependent input datapoint at a time. With this process, finding a BMU by comparing with each SOM unit became a bottleneck step and thus slowed the algorithm to the extent that it took more than 6-7 hours to complete one experiment. Therefore, we perform this as a matrix operation where all the units' values are stored in a matrix form. This scaled down the experiment time to 1-2 hours.

Now, we define the problem of task incremental SOMs and its notations in Section 4.2.1

4.2.1 Problem Definition: The Lifelong Learning Setup

We use the notation as described in [lai Li et al., 2019]. Consider a sequence of N tasks, denoted by $\mathbf{T} = \{T_1, T_2, T_3, ..., T_N\}$. Each task T_t has a training

dataset, $\mathcal{D}_{train}^{(t)} = \{(\mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)}); i = 1, ..., n_t\}$, where $y_i^{(t)}$ is the ground-truth annotation, also known as class label, and n_t is the total number of training examples per task. Let $\mathcal{D}_{train} = \bigcup_{t=1}^{N} \mathcal{D}_{train}^{(t)}$ be the entire training dataset for all tasks. Similarly, $\mathcal{D}_{test}^{(t)}$ denotes the test dataset for a task T_t. The mapping function $f(.; \Theta_t)$ represents the model (example, SOM in our case) learning. Here, Θ_t is collection of all learned parameters till current task T_t (including current t). When a lifelong learning model is finished training on task T_t using its training dataset $\mathcal{D}_{train}^{(t)}$, both $\mathcal{D}_{train}^{(t)}$ and $\mathcal{D}_{test}^{(t)}$ will be lost when the model proceeds to task T_{t+1} to T_N. The main objective of continual learning is to maximize the performance of $f(.; \Theta_t)$ at current task T_t while minimizing the forgetting for tasks from T_1 to T_{t-1}.

There are two popular paradigms in continual learning problems: incremental learning and streaming learning.

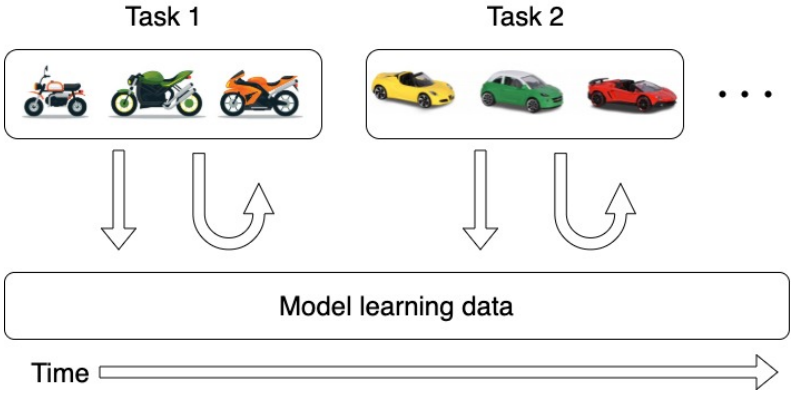


Figure 4.1: A visual depiction of the task incremental learning setup.

4.2.2 Incremental Batch Learning

In the incremental batch learning paradigm, data for each task, $\mathcal{D}^t$ (as described in section 4.2.1) is divided into several batches $\mathcal{B}$, such that $\mathcal{D}_t = \bigcup_{i=1}^{n} \mathcal{B}_i^t$. The learning agent gets to have multiple passes or epochs over each $\mathcal{B}_i^t$. This is a very commonly used setting in lifelong learning problems [Hayes et al., 2019,Lee et al., 2019]. However, while this setting is useful for testing the abil-

ity of a system to remember old knowledge, it does not as accurately reflect a real-world setting of lifelong learning, especially for problems at scale.

Incremental batch learning can be graphically described as shown in Figure 4.1.

4.2.3 Online Streaming Learning

Streaming learning or online learning [Roady et al., 2020], [Ororbia, 2021], [Ororbia et al., 2019], [Ororbia and Mali, 2021] is the extreme form of incremental batch learning. In this setting, the batch size is $\mathcal{B}_i^t = 1$ and the learning agent is only permitted one epoch over each $\mathcal{B}_i^t$. This creates a more challenging problem than incremental batch learning. Notably, unlike in incremental batch learning, a learning agent can possibly revisit tasks in the data ordering. In other words, in incremental batch learning, $\mathcal{D}_t$ follows a linear order i.e. $t = 0, 1, 2, ...$ Once $\mathcal{D}_0$ is over and the learning agent proceeds to $\mathcal{D}_1$, it never receives $\{(x_{i'}^0, y_{i'}^0)\}$ again. However, streaming learning does not follow any such data ordering. It may receive data belonging to any previous task before proceeding to next task. This makes the problem all the more challenging and more closely reflects real life scenarios. Figure 4.2 depicts the process of streaming learning.

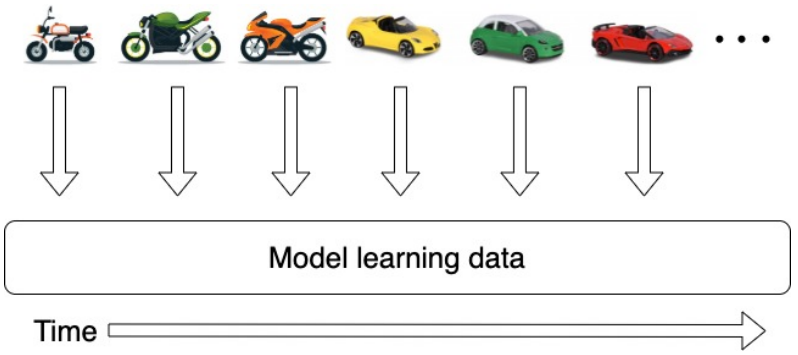


Figure 4.2: A visual depiction of the streaming/online learning setting.

For the scope of this thesis, all SOMs we investigate will receive data in a streaming fashion.

4.2.4 Welford's Algorithm for Running Mean and Variance

Key to our proposed method, we use Welford's online algorithm [Welford, 1962], [Knuth, 1997] to calculate the running variance for each of the SOM's units. Each time a unit of the SOM trains and updates its value, we record this change and use it to calculate the pixel-wise running mean of SOM units. The final trained values of the pixels[1] in units of SOM are used as mean along with the running variance values from the Welford's online algorithm to generate new samples from each unit. Refer to 'Welford's online algorithm' section in [Wikipedia contributors, 2021].

4.2.5 Generating Samples from the SOM

Once the SOM is trained using the algorithm 2, we use the trained values of its units as means for generating new samples. We obtain pixel-wise variance values per SOM unit from Welford's algorithm. The function to generate samples is as follows:

$$mean, \mu = W_v \tag{4.1}$$

$$standard\ deviation, \sigma = \sqrt{\omega_v.variance()} \tag{4.2}$$

$$randomnumber\epsilon = \mathcal{N}([batch\_size, 1]) \tag{4.3}$$

$$generated\_images = \mu + \epsilon * \sigma. \tag{4.4}$$

Note that the above is an example of the re-parameterization trick often used to generate samples from certain continuous distributions (i.e., a statistical one-liner).

4.2.6 Task Incremental Decay Functions

We observed that an SOM trained incrementally on split-MNIST remembered trained units from previous classes for only a few iterations of each current task and that they were overwritten by current classes as the task iterations increased. To alleviate this, we experimented with an exponential decay function with radius (σ) and learning rate (λ) that decayed based on the number of iterations per task. We also tried decaying σ and λ based on number of

[1]Note that while this thesis focuses on pixel-valued inputs, the proposed SOM model would, in principle, work with other types of input data.

epochs per task training. However, both of these approaches did not work as expected. Therefore, we decayed σ and λ based on the task number. To simplify our approach even further, we performed experiments in the class incremental learning setting where we decay parameters based on class numbers. Hence, all of the benchmarks mentioned in Section 5 are based on this setting.

In this thesis, we explored five types of decay functions for both σ, λ. In the following list we show decay functions for σ (which were applied to λ as well):

$$\sigma_t = \sigma * \exp\left(\frac{-t}{\tau}\right) \tag{4.5}$$

$$\sigma_t = \frac{\sigma}{t * \exp\left(t/\tau\right)} \tag{4.6}$$

$$\sigma_t = \frac{\sigma}{\log(t+\epsilon) * \exp\left(t/\tau\right)} \tag{4.7}$$

$$\sigma_t = \frac{\sigma}{1 + t * \exp\left(t/\tau\right)} \tag{4.8}$$

$$\sigma_t = \frac{\sigma}{\log(1+t+\epsilon) * \exp\left(t/\tau\right)} \tag{4.9}$$

where t = current task, $\sigma_t = \sigma$ for task, t, τ = time constant, and the ϵ = offset (10^{-6} or 10^{-7}).

The idea behind all of the equations mentioned above is that both σ and λ must decay distinctively per task. Using exp(.) or log(.) made more sense as they are monotonically increasing functions. Multiplying them with the task number, t in the denominator makes a clear distinction between the parameter values of two consecutive tasks i.e. t_0 and t_1 and so on.

4.2.7 A Task-Incremental SOM Training Algorithm

Crucially, we use all of the mechanics/concepts presented earlier in this chapter and formulate Algorithm 4 for what we call the task incremental SOM. Please note that this algorithm is a modified version of the original SOM algorithm. While Algorithm 2 is trained on i.i.d. MNIST, i.e., all of the images are passed in as input to the SOM model at once, whereas in Algorithm 4, the SOM is trained on split-MNIST which is a dataset that is divided into multiple tasks. Also note that a standard SOM has decay functions as shown in Equation 3.4.

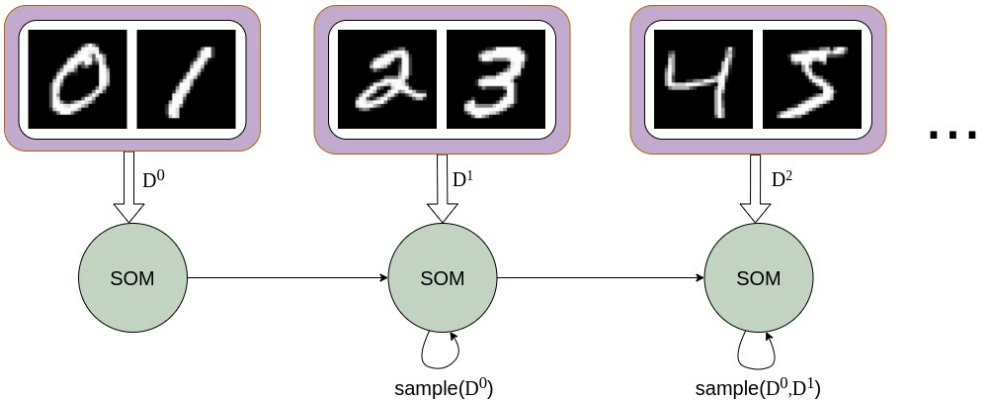


Figure 4.3: Replay in the SOM through re-sampling images from previously learned unit prototypes.

For a task incremental SOM we change the decay function to those shown in Section 4.2.6. This change can be seen in steps 4 and 5 of Algorithm 4.

Algorithm 4 Task incremental SOM

Input: $\mathcal{D}_{train} = (x_i^t, y_i^t)$, where $0 < t < T$
Parameter: Weights of SOM, W_v

1: **for** $t = 0$ to T **do**
2: $\quad \mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)} \leftarrow \mathcal{D}_{train}^t[random\_index]$
3: $\quad u = \arg\min d = \{\|W_v - \mathbf{x}_i\| \mid \forall\ v\ \epsilon\ V\}$
4: $\quad \sigma_t = \sigma/(1 + t * \exp(t/\tau_1))$
5: $\quad \lambda = \lambda/(1 + t * \exp(t/\tau_2))$
6: $\quad$ d = distMatrix(u, v, σ)
7: $\quad h(d) = \exp(-d/2\sigma_s^2)$
8: $\quad W = W + h(d) * \lambda_s * \|W - \mathbf{x}_i\|$
9: **end for**

4.2.8 K-Resampling in SOMs

While training an SOM in a task incremental manner, we use Function 4.8 to decay (σ, λ) for each task, t. As a result of this, the spread of each successive

task decreases at a steady rate. Despite the use of such a task-dependent decay, an SOM still experiences forgetting in its training process. In order for an SOM to preserve its old trained units, it must be retrained on previous tasks every time that it receives a new one. Therefore, we perform resampling on older units and retrain SOM on images generated from them. In other words, new (sampled) images are generated from the old trained units of SOM and then the SOM's weights are updated based on both current task data as well as the sampled/generated images. We use the previously trained weights of a SOM's units as the mean and the running variances of their weight updates (provided through Welford's algorithm) to generate new images and pass them back into the SOM for every new input training sample. This behaviour is similar to generative replay [Shin et al., 2017]. This replay mechanism (which we call 'self-induced replay') helps the SOM to not only learn current task specific data but to also remember its old trained weights. For each input of the current task data, we performed resampling once and twice using old trained units i.e. K=1 and K=2. Algorithm 5 describes the pseudocode for the SOM with K-resampling.

This proposed idea of resampling follows the concept of replay in continual learning. More specifically, it resonates with the idea of sleep in humans. The mechanism of sleep helps the brain to form new neural connections as well as to associate and reinforce memory and forget unnecessary experiences in order to create space for new neural patterns to be formed. For example, with respect to seemingly outdated information, "where did you park your car last month?" is most of the time useless information. When we perform resampling, we reinforce older weights that have learned previous task data and thus maintain the information contained in patterns.

4.2.9 Competitive Resampling

The previous approaches of class decay combined with K-resampling failed in producing robust SOMs. As seen in Figure 5.9 the class decay method produced similar results as a vanilla SOM without any decay. Even though K-resampling showed significant improvement in the class incremental vanilla SOM as seen in Figures 5.10 and 5.11, they failed to acquire an equal number of trained units per class. In addition to that, their trained units did not have a clear visualization of class data. Therefore, we needed a new method

Algorithm 5 SOM Training with K-resampling

Input: One task specific input image, $x_i^{(t)}$
Parameter: Weights of SOM, W_v

for $t = 0$ to T **do**
  $indices \leftarrow length(\mathcal{D}_{train}^{(t)})$
  $\sigma_T \leftarrow decay\_radius(t)$
  $\lambda_T \leftarrow decay\_learning\_rate(t)$
  $indices \leftarrow shuffle(indices)$
  for $index$ in $indices$ **do**
    $x_i^{(t)}, y_i^{(t)} \leftarrow \mathcal{D}_{train}^{(t)}(index)$
    $u \leftarrow find\_bmu(x_i^{(t)})$
    $class\_name = current\_task * task\_size + y_i^{(t)}$
    $update\_som\_count(u, class\_name)$
    $node\_dist = \|u - W_v\| \qquad (0 \leq v < m)$
    $influence = neighbourhood(node\_dist)$
    $W_v = W_v + \lambda_t * influence * (x_i - W_v)$
    $\omega_u = welford.add(x_i - W_u)$
    Replay: select K random W; repeat from 9 to 15
  end for
end for

that would have an equal number of units per class as well as have clear visualization of the units. To solve this problem, we propose a new method called competitive resampling. Figure 4.3 shows how we perform competitive resampling and Algorithm 6 presents its pseudocode. Out of all the methods proposed and investigated in this thesis, competitive resampling produced the best results.

4.3 Evaluation Metrics

There are several metrics that can be used to evaluate the performance of an SOM [Polzlbauer, 2004], [Breard, 2017]. However, these metrics do not measure the performance of an SOM that is trained in an incremental manner. This is likely due to the fact that continual learning is not a typical setting to investigate in context of SOMs (at least classically, it does not appear to be the case). Therefore, we propose a novel metric to evaluate the performance of an incrementally trained SOM. To design our metric, we used quantization error which was originally proposed by Kohonen in the original SOM paper.

In the earlier forms of the SOM described in this thesis, in order to determine the label of any unit in the SOM, we calculated the class number for which that unit was selected a maximum number of times as the BMU. This class was assigned as the label for the respective unit. However, we realized that an upcoming class might have selected a unit a fewer number of times but the update rule still changes the unit/weight representation in a few number of iterations. For example, if a unit was selected as BMU 225 times for class 2 and 100 times for 7. According to this rule, the label for this unit should have been 2. However, due to the update rule, the representation of that unit changes faster for recent digits and the unit will appear like a 7 instead of a 2. Therefore, it is incorrect to label it as a 2 when it appears like a 7. To correct this problem, we instead decided to calculate the arg min over the per class quantization error of a unit.

$$C_u = \arg\min_c \|x_{test}^{(c)} - W_u\| \qquad (c = class \mid 0 < c < C). \tag{4.10}$$

The main objective behind this thesis was to have equal number of trained units per class for an SOM trained on the split-MNIST dataset. This may not hold true in case of a dataset which had more variation in one class in

Algorithm 6 Competitive resampling

for $c = 0$ to $n\_classes$ **do**
  $indices \leftarrow length(\mathcal{D}_{train}^{(c)})$
  $\sigma_C \leftarrow decay\_radius(c)$
  $\lambda_C \leftarrow decay\_learning\_rate(c)$
  trainSom(1)
  if $c > 0$ and $unit\_labels \geq 0$ and $som\_count[u][c] > som\_count[u][unit\_labels[u]]$ **then**
    $C' = som\_count[u].keys() - c$
    $C_s = random.choice(C')$
    units $\leftarrow$ units having (labels = sample_class)
    $random\_unit \leftarrow random.choice(units)$
    rnd = select random number from $\mathcal{N}(0, 1)$
    $\sigma_\omega = \omega[random\_unit].var\_population()$
    $img_s = rnd * \sqrt{\sigma_\omega} + W[random\_unit/W.shape[0],\ random\_unit\%W.shape[1]]$
    $u_s = find\_bmu(img_s)$
    $node\_dist = precomputed\_distance[u_s[0], u_s[1], :, :]$
    $\sigma_s = decay\_sigma(C_s, decay\_choice)$
    $\lambda_s = \lambda_0$
    $influence = get\_neighbourhood(node\_dist,\ \sigma_s)$
    $difference = img_s - W$
    $W = W + (\lambda_s * influence * difference)$
    $update\_som\_count(u_s, C_s)$
    $\delta = u_s[0] * W.shape[0] + u_s[1]$
    $\omega_\delta = welford.add(difference[u_s])$
    $unit\_labels[u_{num}] = max(som\_count[u_{num}].items(),$
                $key = operator.itemgetter(1))[0]$
  end if
  $unit\_labels[u] = max(som\_count[u].items(),$
            $key = operator.itemgetter(1))[0]$
end for

comparison to others. Since split-MNIST has an equal number of samples per class and has even variation within every class, the condition of equal number of trained units per class holds true. We take this as a baseline and compare

Algorithm 7 The α_{mem} metric

Input: $\mathbf{X}_{test}$, where $\mathbf{X}^{c}_{test}$ is all $\mathbf{x}_i$ w/ $\arg\max(\mathbf{y}_i) \equiv c$
Parameter: Weights of SOM, W_v

1: $N_c = V/C$, where C = total number of classes
2: $\rho_v := \arg\min\left(d = \{\|\mathbf{w}_v - \mathbf{X}^{(c)}_{test}\| : \forall\ c\ \epsilon\ C\}\ \right)$
3: $\phi = \left\{\sum_{v=0}^{V} \mathbb{1}(c = \rho_v) : \forall\ c\ \epsilon\ C\right\}$
4: $\alpha_{mem} = (\phi_i - N_c)_{RMS}$ // RMS = "root-mean-square"

it with number of trained units per class of a trained SOM to evaluate it. The lower the difference between number of trained units per class and the $\mathcal{N}_{true}$ in Equation 4.11 , the better the SOM. It can be represented as follows:

$$\begin{aligned} N_c &= \text{total number of classes} \\ O &= \left[\sum \mathbb{1}(C_u = 0), \sum \mathbb{1}(C_u = 1), ..., \sum \mathbb{1}(C_u = N_c - 1)\right] \\ \mathcal{N}_{true} &= (\text{total units in SOM})/N_c \end{aligned} \tag{4.11}$$

$$\alpha_{mem} = \sqrt{\frac{\sum_{c=0}^{N-1} (O_c - \mathcal{N}_{true})^2}{N_c}} \tag{4.12}$$

where, $\mathbb{1}(\cdot)$ is an indicator function that returns 1 if the condition passed in it becomes true and 0 otherwise. For example:

$$\mathbb{1}(C_u = 0) = \begin{cases} 1 & \text{if } C_u = 0 \\ 0 & C_u \neq 0 \end{cases}$$

The lower the value of α_{mem} the better the SOM. An ideal SOM would have $\alpha_{mem} = 0$ because all of its classes will have equal number of trained units i.e. $\mathcal{N}_{true}$. Algorithm 7 explains steps to calculate α_{mem} metric for a SOM. We show empirical results for all variants of SOM in Section 5.6

Chapter 5

Experimental Results

In this chapter we conduct various experiments focusing on SOMs using the various approaches described in the last chapter and analyze their results.

5.1 Experimental Design

We adopt the following experimental design as described in this section.

5.1.1 Experimental Setup

Model: We use an SOM that has a square topology/grid with either 10×10, 20×20 or 40×40 number of units/weights.
Dataset: We used the split-MNIST benchmark and tested our experiments in the task incremental setting as well as the class incremental MNIST. A task consists of multiple classes extracted from the dataset. We divided MNIST into 5 tasks each containing 2 classes. When each task is passed incrementally in this way, the setting follows the task incremental setup (with the caveat that there is only 1 epoch/pass through each dataset allowed) as described earlier in this thesis. On the other hand, when the task size is one, i.e. there exists only one class per task, it is known as class incremental learning.

5.1.2 Model Architecture

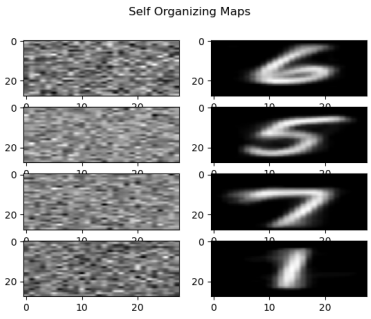
As stated in Section 1.3, for the scope of this thesis, we focus on using a square topology to arrange the SOM's neuronal units. Each unit of the SOM has the same dimensionality d as the input images. So, if there is a square SOM of size **(m x m)** then there are $m * m$ number of units each having dimension d. In the case of mnist, d would become $28 * 28 = 784$. Therefore, $\mathbf{W = m\ x\ m\ x\ d}$. These units are arranged in an Euclidean grid. The Euclidean distance between the units of the SOM are pre-computed and stored such that later they are re-used when performing a weight update in the training process. The pseudocode for calculating the distances between two SOM units is given in Algorithm 3.

5.2 Preliminary Results for the Classical SOM

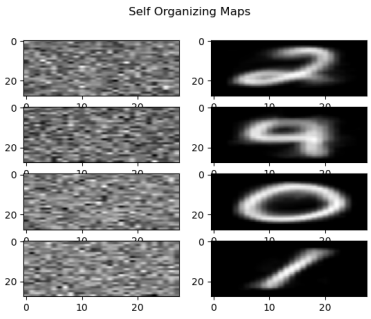
We trained a naive SOM on the MNIST dataset where all of the data samples were passed in independently and identically distributed (i.i.d.) manner. The results for such a naive SOM are shown in figure 5.1. We implemented the design of a neural gas model for this specific set of results. This means, we used the topological distance or pixel-wise distance between two units to calculate distance between them.

5.3 The Vanilla Task-Incremental SOM

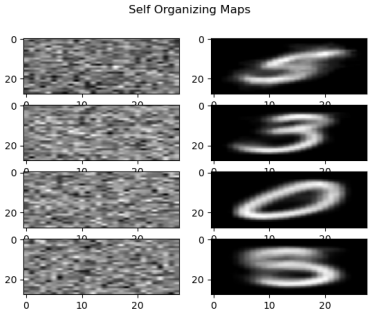
We first trained the SOM in a task incremental fashion where the initial sigma and learning rates are not decayed. In these SOMs, units were not topologically arranged in a Euclidean plane. This means that when calculating the distance between two SOM units we considered their pixel-wise distance . If the units were arranged in an Euclidean plane, the distance between the units were calculated as if they were arranged in a Cartesian coordinate plane. We performed several experiments in this category with different values of the initial radius, σ and learning rate, λ hyperparameters. The values of σ ranged randomly between 0.5 and 5 whereas the learning rates were randomly set between 0.001 to 0.02. Figure 5.2 shows one such example. All of the SOM units were initialized with a Gaussian distribution, $\mathcal{N}(0, 1)$



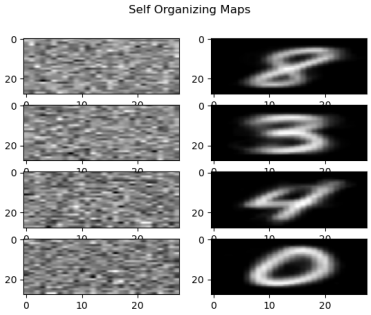
(a) SOM units 0-3 before and after training



(b) SOM units 4-7 before and after training



(c) SOM units 8-11 before and after training

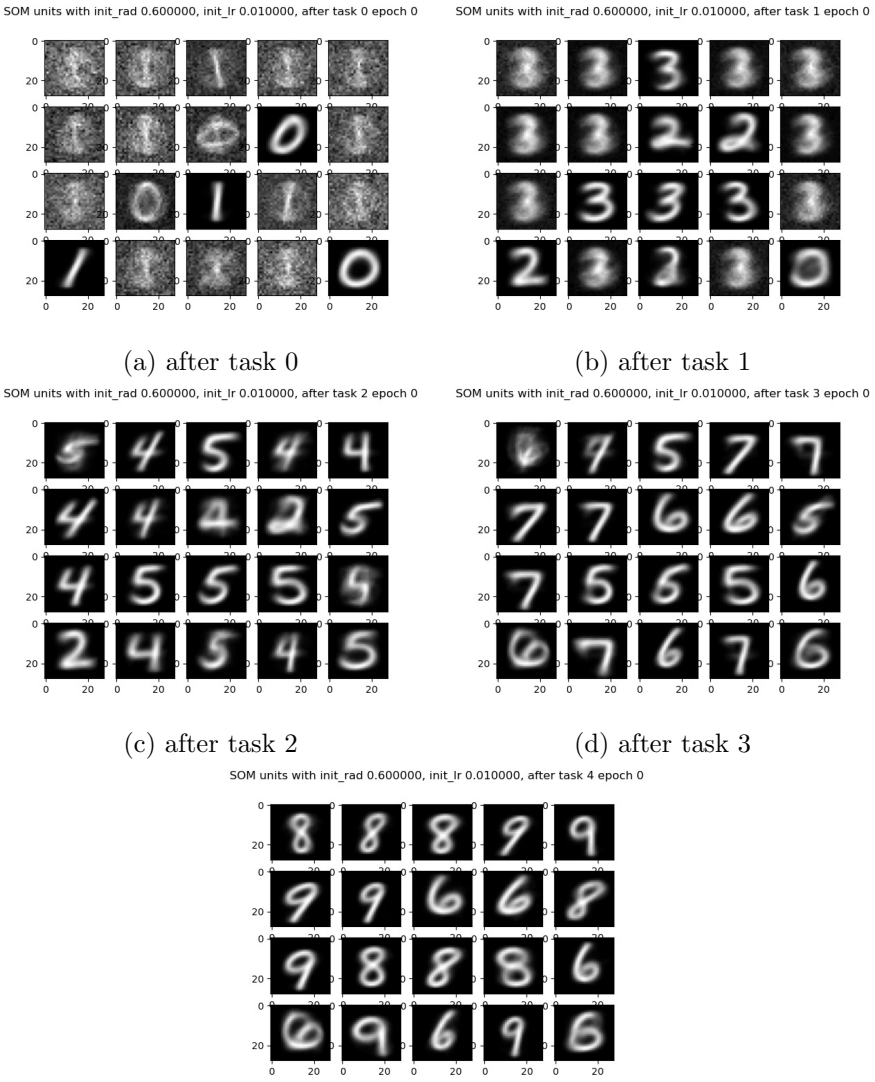


(d) SOM units 12-15 before and after training

Figure 5.1: units 0 to 15 of a SOM containing 20 units

From sets of initial experiments, we observed that there was severe forgetting in task-incrementally trained SOMs. These initial set of experiments were performed with an SOM containing 20 units arranged in 4×5 order as shown in figure 5.2. The distance between units were still considered based on their pixel values and not based on the position of units in a Cartesian plane. We further observed that lowering the value of σ trained fewer units in the SOM. Furthermore, lowering the value of the learning rate resulted in faded units in comparison to those shown in Figure 5.2. An example of this can be seen in figure 5.3.

During this set of experiments, we inferred one more weakness of these



(a) after task 0

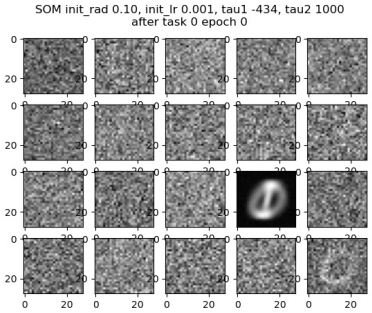
(b) after task 1

(c) after task 2

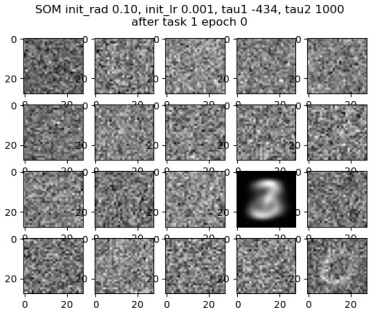
(d) after task 3

(e) after task 4

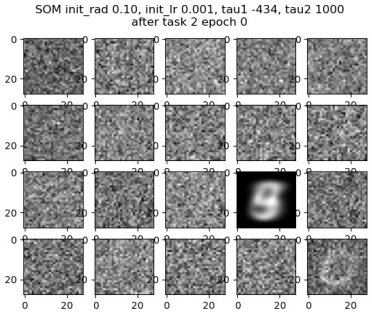
Figure 5.2: Vanilla SOM trained task incrementally with initial $\sigma = 0.6$ and initial $\lambda = 0.01$



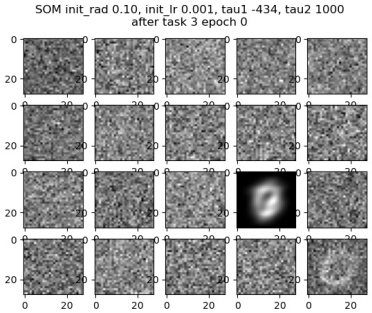
(a) after task 0



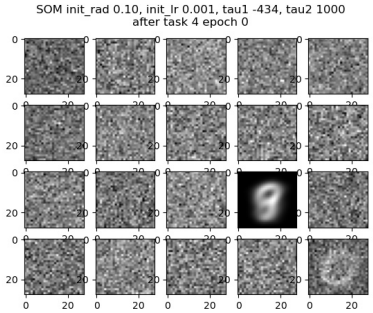
(b) after task 1



(c) after task 2



(d) after task 3



(e) after task 4

Figure 5.3: Vanilla SOM trained task incrementally with initial $\sigma = 0.10$ and initial $\lambda = 0.001$

SOMs. The basic logic of training an SOM involves selecting a BMU. This BMU is selected based on the pixel-wise distance between the input image and the units of SOM. The units trained on task 0 appeared either like MNIST digits 1 or 0. Some units appear to have been trained on both classes thus having a mixed appearance. Due to this, the pixel values of later input images belonging to different classes that arrived in task incremental fashion matched more closely to already trained units instead of untrained units. This is why every time a new task was passed as input to the SOM, the BMUs got automatically selected only out of the already-trained units instead of picking up and using the untrained units. As a result, the pixel representations of older classes were more swiftly forgotten, increasing the amount of forgetting of the SOM units.

From all the experiments conducted with these SOMs using various parameter configuration, we inferred following:

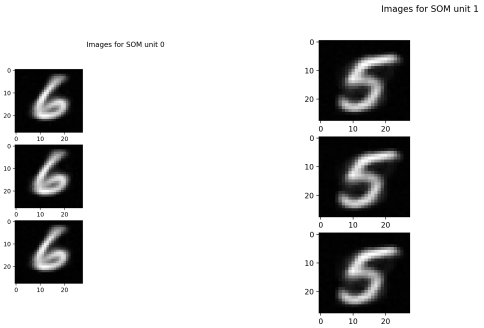
- Increasing the learning rate of SOM increases the spread of the learned units or weights
- Initial radius $\propto \frac{1}{\text{sparcity in SOM}}$
  This means that the greater the value of initial σ, the wider the spread of the trained units in SOMs whereas, the smaller the initial σ value, the smaller the spread of the trained units (leaving a higher quantity of free or untrained units in the SOM). In some cases, if σ is low and λ is high, the units will train faster but will not spread as widely in the SOM.
- Increasing the time constants in the decay functions increases the spread of trained or learned units.

5.4 SOM with K-Resampling

To overcome the problem of units being trained on a current task overwriting previously trained units, we came up with the idea of retraining the SOM on images regenerated or resampled from older trained units. Examples of images generated using Welford's algorithm can be seen as shown in Figure 5.4.

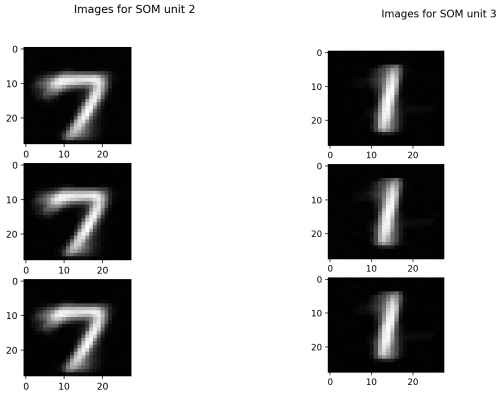
Here, we experimented with different decay functions for σ and λ as shown in Section 4.2.6 where, $offset = 10^{-6}$ or 10^{-7}. The same functions were also applied to λ_t where τ_1 is replaced with τ_2.

Out of all of the decay functions, Equation 4.8 produced the best result. We experimented with both single and double resampling, i.e., $K = 1$ and $K = 2$.



(a) generated from SOM unit 0

(b) generated from SOM unit 1



(c) generated from SOM unit 2

(d) generated from SOM unit 3

Figure 5.4: Images sampled from the SOM units

This means that for every single input image on which the SOM is trained, our model generates either one or two images from its already trained units on previous tasks. Figure 5.5 shows how the SOM performed with double resampling when using Equation 4.8 for decaying σ and λ parameters. It performs well but misses the digit 4. In addition, there are very few trained units for the digit 4 in the final SOM. To overcome such problems and make the SOM capture digits better, we came up with competitive resampling explained

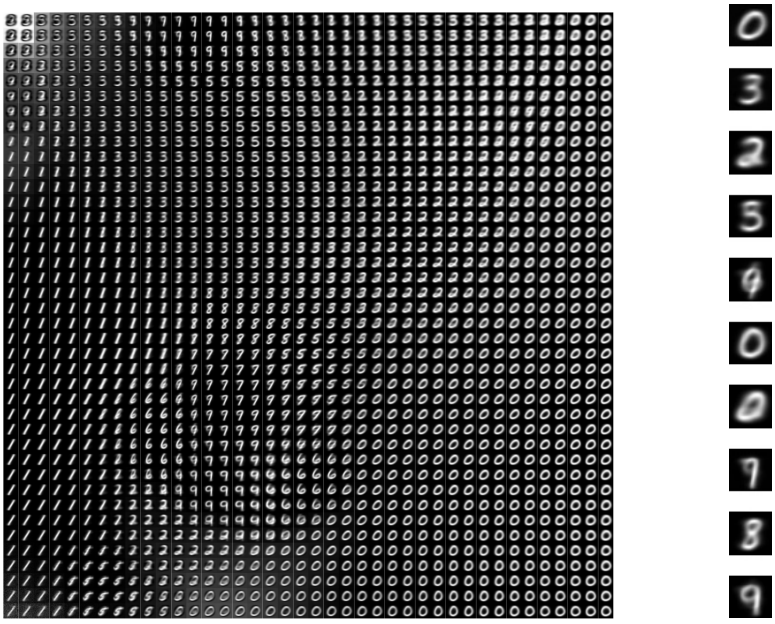
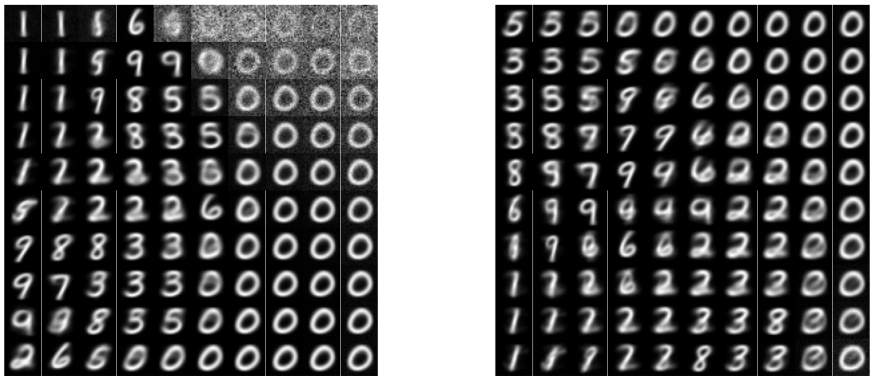


Figure 5.5: SOM of size 40x40 with $\sigma_0 = 5$ and $\lambda_0 = 0.01$ trained using decay function 4.8

in subsection 5.5.

5.5 Competitive Resampling in the SOM

Figure 5.6 shows examples of 10×10 SOM using this approach. As seen in this image, the number of trained units depends upon the value of σ. We downsized the SOMs from 40×40 to 10×10 to exacerbate forgetting in it.



(a) $\sigma = 3, \lambda = 0.002$ (b) $\sigma = 4, \lambda = 0.002$

Figure 5.6: 10x10 SOM with Competitive resampling having only λ decayed and not σ.

5.6 Benchmarks

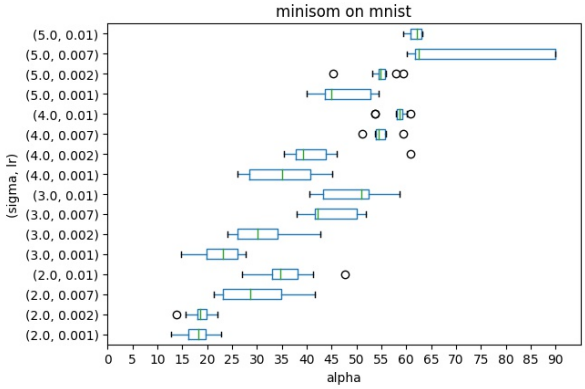
In this section, we provide experimental results for the various SOM versions that we implemented. For every parameter setting of (σ_0, λ_0) we perform 10 trials. Thus, we report the mean, standard deviation, best, and worst across the 10-trial results. Tables 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 present these results. Their corresponding boxplots and SOM outputs are displayed in Figures 5.7, 5.8, 5.9, 5.10, 5.11, 5.12.

5.6.1 miniSOM

miniSOM [Vettigli, 2018] performs very well in normal circumstances where all the data samples are passed at once in an i.i.d. manner. However, when it is trained in a task incremental manner, it clearly suffers from forgetting. It can be clearly observed in Figure 5.7b that it remembers the digits 6, 8 and 9. There are a few units that look like combination of 5s and 8s indicating that there could be a representational overlap between patterns for those two classes. As compared to the results of Section 5.6.2 and 5.6.3, it produces clearer representations of input samples among its trained units. Table 5.1 shows that the best α value scored by the task-incremental SOM is 12.748, produced with (σ, λ) set as (2, 0.001). Furthermore, it indicates that increasing the initial values of (σ, λ) increases the value of α metric.

SOM variant	description	σ_0	λ_0	α			
				mean	std	best	worst
minisom	vanilla	**2**	**0.001**	17.969	3.016	**12.748**	22.825
		2	0.002	18.562	2.400	13.842	22.118
		2	0.007	29.514	6.974	21.370	41.705
		2	0.010	35.747	5.861	27.028	47.707
		3	0.001	22.682	4.077	**14.855**	27.740
		3	0.002	31.079	6.020	24.052	42.786
		3	0.007	45.030	5.256	37.956	51.856
		3	0.010	48.821	6.023	40.587	58.728
		4	0.001	34.994	6.818	**26.153**	45.181
		4	0.002	41.861	7.581	35.412	60.959
		4	0.007	54.869	2.085	51.225	59.464
		4	0.010	58.117	2.422	53.814	60.959
		5	0.001	47.453	5.292	**39.942**	54.489
		5	0.002	54.563	3.733	45.336	59.464
		5	0.007	73.033	14.621	60.208	90.000
		5	0.010	61.878	1.282	59.464	63.253

Table 5.1: Results table for miniSOM trained on split-MNIST



(a) Boxplot

(b) $\sigma_0 = 2, \lambda_0 = 0.001, \alpha = 12.748$

Figure 5.7: MiniSOM performed on MNIST.

5.6.2 Vanilla/Standard SOM with No Decay of Parameters

Figure 5.8b demonstrates the presence of some sparsity as compared to Figure 5.7b. Additionally, there are fewer 6s in Figure 5.8b as compared to Figure 5.7b and Figure 5.9b. Figure 5.8a presents higher values as compared to Figure 5.7a. The best α metric value for this variant SOM is 19.647 obtained by setting initial (σ, λ) to (2, 0.001). α becomes constant 90 for higher values of (σ, λ).

5.6.3 Vanilla/Standard SOM with Class Decay

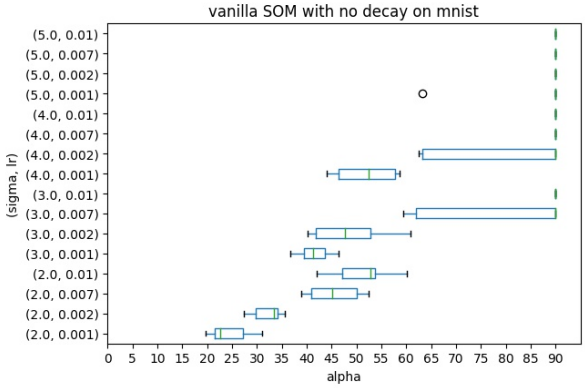
Figure 5.8a and Figure 5.9a loosely show similar values of α. The best α values of both standard SOMs with and without class decay are very similar i.e. 19.647 and 20.396. In general, using class decay did not improve SOM results.

5.6.4 K=1 Resampling

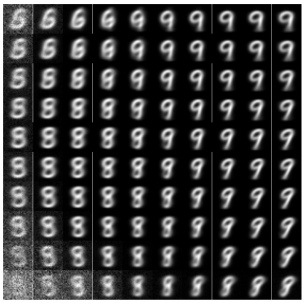
We start obtaining better results with this variant when compared to previous variants of SOMs i.e. miniSOM, SOMs without and with class decay. Figure 5.10b shows that the spread of digits decreases as a function their class

SOM variant	description	σ_0	λ_0	α			
				mean	std	best	worst
Vanilla	without decaying σ_0 and λ_0	**2**	**0.001**	24.173	3.810	**19.647**	31.04
		2	0.002	32.0157	2.92392	27.3404	35.5996
		2	0.007	45.4557	5.08582	39.0043	52.4976
		2	0.01	51.0701	5.8792	42.0872	60.208
		3	0.001	41.5028	3.00352	**36.7605**	46.3753
		3	0.002	47.8526	6.92243	40.1912	60.959
		3	0.007	78.538	14.8165	59.4643	90
		3	0.01	90	0	90	90
		4	0.001	52.0235	6.26454	**44.0681**	58.7282
		4	0.002	79.1471	14.0134	62.482	90
		4	0.007	90	0	90	90
		4	0.01	90	0	90	90
		5	0.001	87.3253	8.458	**63.2535**	90
		5	0.002	90	0	90	90
		5	0.007	90	0	90	90
		5	0.01	90	0	90	90

Table 5.2: Results table for vanilla SOM with no decay of parameters trained on split-MNIST



(a) Boxplot



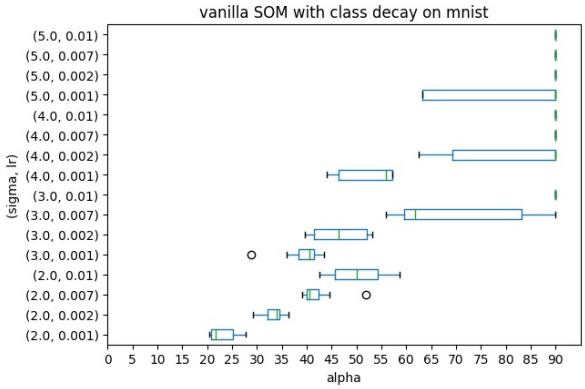
(b) $\sigma_0 = 2, \lambda_0 = 0.001, \alpha = 19.647$

Figure 5.8: vanilla SOM with no decay performed on MNIST

number. This happens due to the nature of the decay function we use. The

SOM variant	description	σ_0	λ_0	α			
				mean	std	best	worst
Vanilla	decay	**2**	**0.001**	23.161	2.953	**20.396**	27.749
	(σ, λ)	2	0.002	33.196	2.319	29.235	36.380
	per class	2	0.007	42.012	3.814	39.183	51.856
		2	0.010	50.126	5.226	42.497	58.728
		3	0.001	39.054	4.152	**28.862**	43.505
		3	0.002	46.510	5.750	39.682	53.151
		3	0.007	68.873	14.718	55.866	90.000
		3	0.010	90.000	0.000	90.000	90.000
		4	0.001	52.194	6.033	**44.068**	57.280
		4	0.002	81.745	13.292	62.482	90.000
		4	0.007	90.000	0.000	90.000	90.000
		4	0.010	90.000	0.000	90.000	90.000
		5	0.001	79.301	13.812	**63.253**	90.000
		5	0.002	90.000	0.000	90.000	90.000
		5	0.007	90.000	0.000	90.000	90.000
		5	0.010	90.000	0.000	90.000	90.000

Table 5.3: Results table for vanilla SOM with class decay trained on split-MNIST



(a) Boxplot

(b) $\sigma_0 = 2, \lambda = 0.001, \alpha = 20.396$

Figure 5.9: Vanilla SOM with class decay trained on split-MNIST.

decay function more strongly affects the σ of digits whereas a decayed λ value

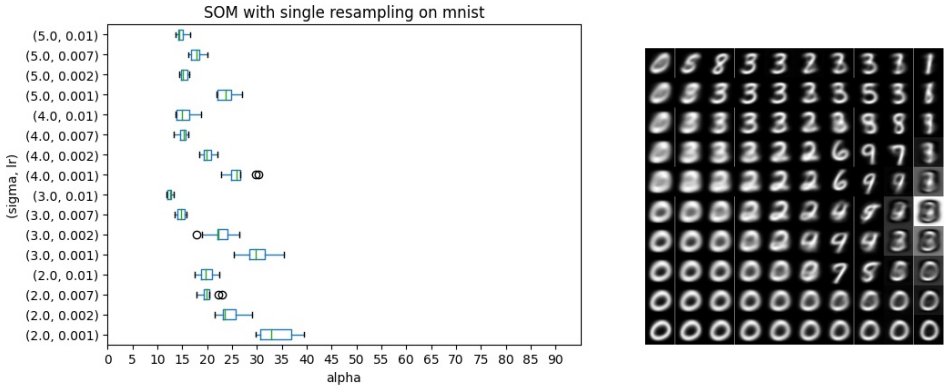
per class does not seem to have any effect on the visibility of the learned representations in Figure 5.10b. Since SOMs perform soft clustering, we see blurred boundaries between two classes in Figure 5.10b. The best α value for this variant of SOM is 11.851 obtained by setting initial (σ, λ) to (3, 0.01). It is significantly lower than previous variants. Table 5.4 demonstrates that $\lambda \propto (1/\alpha)$.

SOM variant	description	σ_0	λ_0	α			
				mean	std	best	worst
Single resampling	resample one image from previously trained units and use current class' (σ_c, λ_c) for them	2	0.001	33.929	3.774	29.757	39.516
		2	0.002	24.415	2.495	21.481	29.009
		2	0.007	20.057	1.536	17.965	23.038
		2	0.010	19.873	1.477	**17.544**	22.422
		3	0.001	29.863	3.173	25.330	35.369
		3	0.002	22.660	2.771	17.907	26.556
		3	0.007	14.674	0.895	13.454	15.825
		3	**0.010**	12.488	0.494	**11.851**	13.266
		4	0.001	26.045	2.460	22.891	30.239
		4	0.002	20.090	1.129	18.373	22.121
		4	0.007	15.067	0.933	**13.251**	16.152
		4	0.010	15.415	1.768	13.663	18.708
		5	0.001	23.868	1.876	21.817	26.972
		5	0.002	15.392	0.785	14.353	16.432
		5	0.007	17.776	1.240	16.142	19.986
		5	0.010	14.740	0.902	**13.679**	16.562

Table 5.4: Results table for SOM with single resampling trained on split-MNIST

5.6.5 K=2 Resampling

The results of this section indicate that α decreases when $K = 2$ in comparison to $K = 1$. However, due to double resampling, it takes more time to finish experiments in this case. Therefore, an α value comes at the cost of execution time for this variant of SOM. Figure 5.11b shows more blurry images though in comparison to Figure 5.10b.



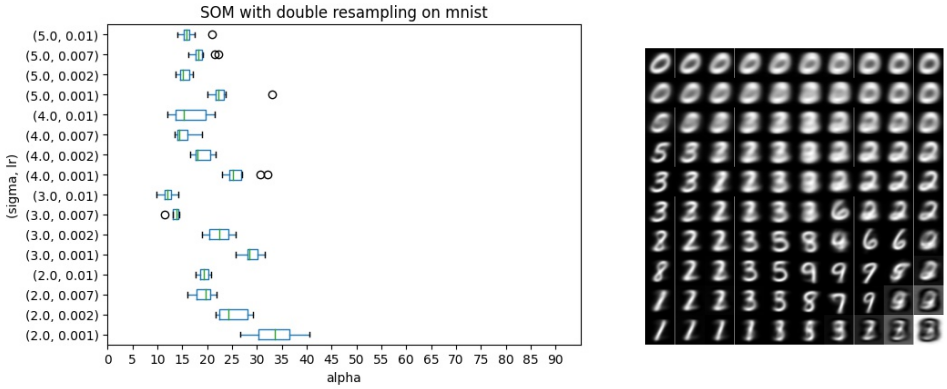
(a) Boxplot

(b) $\sigma_0 = 3, \lambda_0 = 0.01, \alpha = 11.851$

Figure 5.10: SOM with single resampling trained on split-MNIST

SOM variant	description	σ_0	λ_0	α			
				mean	std	best	worst
Double resampling	resample two images from previously trained units and use current class' (σ_c, λ_c) for them	2	0.001	33.795	4.563	26.728	40.587
		2	0.002	25.164	3.033	21.811	29.252
		2	0.007	19.236	1.828	**16.138**	21.909
		2	0.010	19.309	1.079	17.652	20.778
		3	0.001	28.740	1.858	25.768	31.639
		3	0.002	22.378	2.337	19.039	25.729
		3	0.007	13.581	0.865	11.524	14.384
		3	**0.010**	12.147	1.248	**9.899**	14.142
		4	0.001	26.291	2.920	23.058	32.101
		4	0.002	18.942	1.801	16.576	21.686
		4	0.007	15.266	1.817	13.574	18.888
		4	0.010	16.509	3.603	**12.024**	21.594
		5	0.001	23.237	3.607	20.000	33.005
		5	0.002	15.314	1.182	**13.704**	17.110
		5	0.007	18.697	1.897	16.310	22.303
		5	0.010	16.214	1.954	14.081	21.078

Table 5.5: Results table for SOM with double resampling trained on split-MNIST



(a) Boxplot

(b) $\sigma_0 = 3, \lambda_0 = 0.01, \alpha = 9.899$

Figure 5.11: SOM with double resampling performed on split-MNIST

5.6.6 Competitive resampling

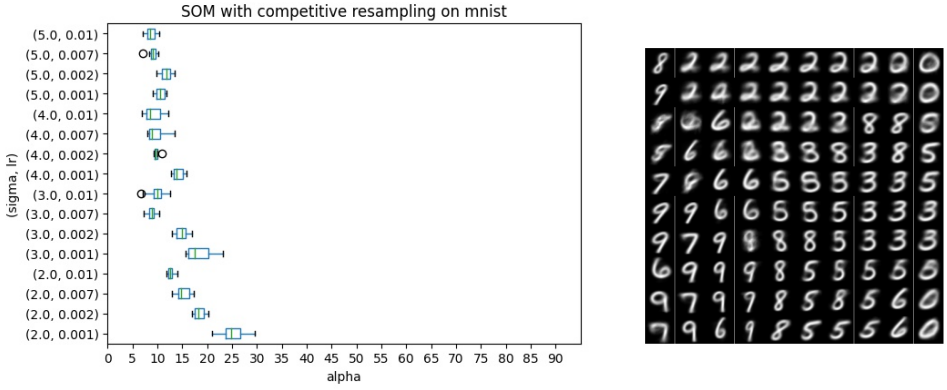
Despite our good experimental results, we were unable to achieve the objective of obtaining an equal number of trained units per class. We tried several variations of this approach such as selecting random σ and λ values and decaying only one out of both of them. Out of all of these, the approach where we decay just λ and not σ worked the best. This approach produces the minimum α value out of all the variants of SOMs i.e. 6.782 with the parameters set as $(\sigma, \lambda) = (3, 0.01)$.

Table 5.7 summarizes the best results from each variant of SOM:

Observe in Table 5.7 that SOMs without and with decay perform poorly as compared to the miniSOM model. By looking at Figures 5.8b, 5.9b and 5.7b, we can say that this pattern in the *alpha* values for these three variants is caused by the number of 6s in their final outputs. The final outputs of all three variants show that all them fail to preserve all 10 classes at the end of training. Therefore, none of them is better overall. We see some units with inverted pixel values in Figures 5.10b and 5.11b. Although competitive resampling produces the best results out of all of the variants, further improvement is required in order to obtain clear and crisp representations of input samples in the final output of our SOMs.

SOM variant	description	σ_0	λ_0	α			
				mean	std	best	worst
Competitive resampling	Decay σ but not λ for resampled images	2	0.001	25.294	2.785	20.899	29.655
		2	0.002	18.435	1.113	17.036	20.310
		2	0.007	15.093	1.541	13.013	17.276
		2	0.010	12.644	0.683	**11.926**	13.968
		3	0.001	18.302	2.541	15.640	23.206
		3	0.002	14.855	1.255	13.015	16.984
		3	0.007	8.820	0.839	7.211	10.305
		3	**0.010**	9.834	1.798	**6.782**	12.506
		4	0.001	14.152	1.137	12.783	15.853
		4	0.002	9.776	0.464	9.286	10.844
		4	0.007	9.879	1.968	8.014	13.474
		4	0.010	9.122	1.697	**6.992**	12.145
		5	0.001	10.580	1.032	9.044	11.916
		5	0.002	11.826	1.119	9.920	13.435
		5	0.007	8.999	0.845	7.159	10.128
		5	0.010	8.762	1.033	**7.124**	10.296

Table 5.6: Results table for SOM with competitive resampling trained on split-MNIST



(a) Boxplot

(b) $\sigma_0 = 3, \lambda_0 = 0.01, \alpha = 6.782$

Figure 5.12: SOM with competitive resampling performed on split-MNIST.

SOM variant	σ_0	λ_0	α			
			mean	std	best	worst
miniSOM	2	0.001	17.969	3.016	12.748	22.825
Vanilla - no decay	2	0.001	24.173	3.810	19.647	31.04
Vanilla - class decay	2	0.001	23.161	2.953	20.396	27.749
Single resampling	3	0.01	12.488	0.494	11.851	13.266
Double resampling	3	0.01	12.147	1.248	9.899	14.142
Competitive resampling	**3**	**0.01**	**9.834**	**1.798**	**6.782**	**12.506**

Table 5.7: Summmary of results from all variants of SOM

5.7 Limitations

In this section, we present some of the limitations of c-SOM based on comparison with some of the existing state-of-the-art approaches. Despite taking a new approach and producing significantly improved results, our method has several limitations which we discuss next.

5.7.1 Equal Number of Trained Units per Class

There is no concrete way to make sure that there are an equal number of trained units for each input class in each task dataset. This problem is exacerbated in an online learning system where there is no information known apriori about the number of incoming classes. Additionally, our approach does not specifically take into account the variance of each input's class data. All of the trained units for any particular class should have a moderate representation of various data samples presented to it. If a certain type of samples are higher in quantity, they should not overshadow the trained representations of low quantity samples of any given class. Moreover, an SOM should not completely forget a class even if that class has low variance. Instead, it could potentially be represented with either a single or a small number of units.

5.7.2 On the Quality of Trained Units

The α metric proposed in this thesis measures the number of trained units per class. However, there is no way to identify the quality of units, i.e., how well trained a unit is for any given class. For example, some of the trained units in

Figure 5.6 are blurry and perhaps need additional training in order to acquire accurate representations in SOM units.

5.7.3 No Study on Concept Drift

Section 5.7.2 raises another concern related to concept drift [Gepperth et al., 2015], which is a very well known issue when building agents for mutli-task learning or continual learning. Our experiments demonstrate the poor performance of a naive SOM trained on incremental data. This poor performance is likely caused by concept drift. There needs to be a concrete way or metric to measure the degree of concept drift.

5.7.4 Determination of the Class Label for Resampled Images

Initially we determined the labels of resampled images using the count of the number of times a unit is selected as the BMU for a given input image. However, we realized that it was the wrong way to determine class label. Simply put, a unit might change its representation even when it is selected as the BMU for a fewer number of times for successive tasks. Therefore, a requirement is needed to develop a concrete way to determine the label of a resampled image from the trained units of an SOM. In the future, one possible way to do this might be by using the quantization error over the current task. Algorithm 8 presents a sketch on how to potentially overcome this limitation.

Another issue with a real lifelong learning problem is that, during training time, there is no concrete way to find out when the tasks changed from T_t to T_{t+1}. Extensive experimentation and testing is required to validate our c-SOM algorithm 8 and extensions to handle a lack of task descriptors. This will be the subject of future work.

Algorithm 8 Algorithm for determining class of resampled images

1: Train the SOM over task 1
2: Label all the units for classes from task 1
3: Train the SOM on upcoming tasks
4: Calculate quantization error over classes belong to current task
5: Perform clustering over these error values and units that have lower error values match with corresponding classes from current task
6: Change the labels of only those units that formed cluster of lower quantizaion error values.

Chapter 6

Conclusions

In this thesis, we studied continual learning in the context of self orgranizing maps (SOMs) and developed various approaches to improve SOM memory retention across tasks. Specifically, we experimented with multiple ways to reduce forgetting in the SOM such as: 1) various task based decay functions, 2) resampling images generated from trained units of SOM, and 3) performing competitive resampling. We empirically demonstrated that our approaches produced much better results than naive SOMs in the task incremental setting.

The problem of reducing forgetting in neural architectures is important because there is a great need for models that have the ability to continually learn, unlearn, and update their internal knowledge representations given real life data. Catastrophic forgetting presents a massive hurdle in crafting such kind of models. Hence, we proposed the following approaches to mitigate this problem:

1. Vanilla SOMs with task dependent decay of hyperparameters.
2. Resampling images from SOM units trained on previous tasks to create samples for retraining the SOM to improve its memory of old task representations. We experimented with two variations of this algorithm, which were: i) resampling once and ii) resampling twice.
3. Competitive resampling – we resample images to maintain an equal number of trained units per class in SOM.

6.1 Future work

In the future, we plan to extend our approach to mainstream architectures like feedforward neural networks (FFN). Forming a dual network architecture using the SOM as a generative model could prove valuable in reducing catastrophic forgetting.

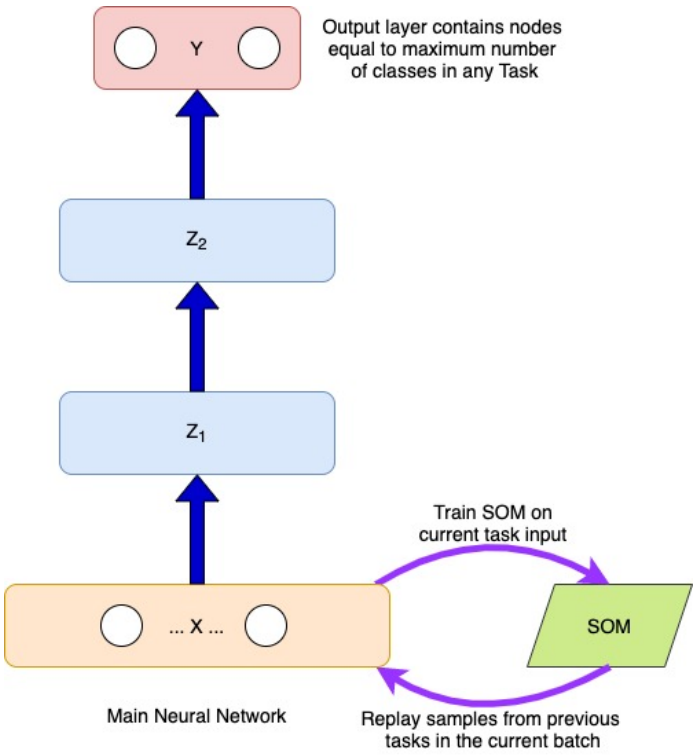


Figure 6.1: Proposed architecture for Input replay

As shown in Figure 6.1, we envision this extension. In essence, we pass the images from the input layer of feedforward neural network to an SOM. The units trained on these inputs are then used to generate new data which will then be replayed to the input layer of the feedforward network for future tasks. We will experiment with another variant of this where the SOM is trained on the hidden layers of the feedforward network. The architecture for this is

shown in Figure 6.2. Here, for a current task, T_n the feedforward network is trained as normal. However, the post activations of the hidden layers are then fed to their respective SOMs. When the feedforward network is trained on later tasks, T_{n+i}, $(0 < i < N)$, the SOMs trained on post-activations of T_n will be used for carrying out replay. This means that new data will be generated from the units of the SOM trained on the post activations of T_n and finally fed back into the hidden layers of the FFN.

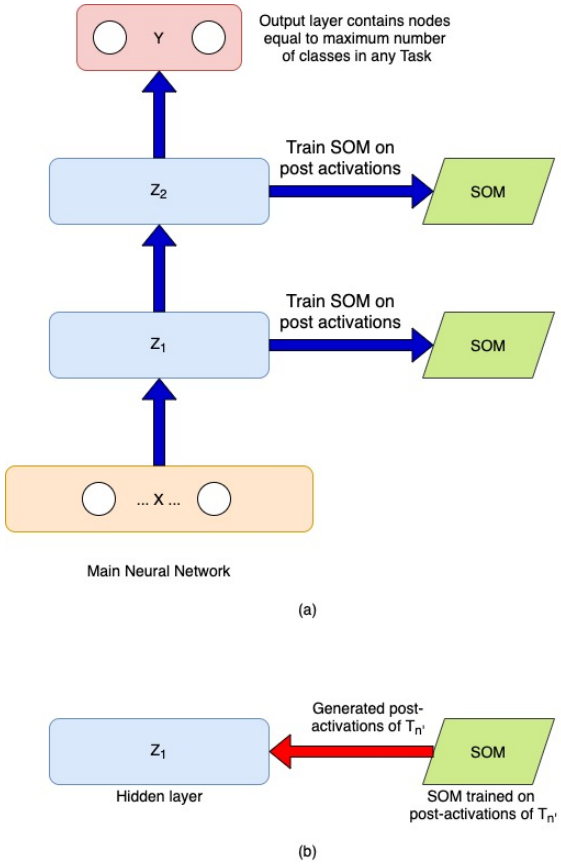


Figure 6.2: (a)Forward pass on current task T_n
(b)Forward pass on next tasks T_{n+1} where $T_{n'}$ is all previous tasks
Proposed architecture for hidden replay

6.1.1 Evaluation Metrics for the FFN

We would measure the performance of our dual model based on standard continual learning metrics such as *Average Accuracy* (ACC) and *Backward Transfer* (BWT) metrics [Lopez-Paz and Ranzato, 2017]. BWT is the influence on the performance of a model on previous tasks, $k < t$ after learning current task, t.

For every t, we will reserve a separate test set, $\mathcal{D}_{test}^{t} = \bigcup_{i=1}^{\mathcal{N}_{test}} \{(x_i^t, y_i^t)\}$ for evaluation. A matrix $R \in \mathbb{R}^{T \times T}$ will then be formed which contains accuracy measures. Each row of R contains model accuracy on the test set of every t. After training is completed on every t, we add a new row, $\mathbb{R}^{1 \times T}$, containing the task-wise evaluations to R. Thus, R_{ij} is the test accuracy of model on T_j after training on T_i. [Lopez-Paz and Ranzato, 2017] defines these metrics as follows,

$$Average\ Accuracy\ (ACC) = \frac{1}{T} \sum_{i=1}^{T} R_{T,i}$$

$$Backward\ Transfer\ (BWT) = \frac{1}{T-1} \sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$$

, where, $\overline{b}$ is the vector of containing the test accuracies for each task at random initialization i.e. the model before its training begin.

A positive value for BWT indicates that the model has positive influence (i.e. it remembers) on previous tasks after training current task. Therefore, the larger/greater (more positive) the values of these metrics, the better the performance of a model.

6.2 Closing Thoughts

This thesis explored unsupervised continual learning for the case of self-organizing maps (SOMs). Several novel techniques were developed to train SOMs in an incremental learning fashion while reducing the amount of forgetting observed in it. Empirical results of the proposed approaches show improved performance over classical SOMs. SOMs are not widely used with red-green-blue (RGB) images and we intend to explore how they perform on RGB image

datasets. Notably, our SOM models could be plugged into convolutional neural networks to form an even more robust continual learning system for image data. Using our proposed techniques with growing SOMs or growing neural gas is another fruitful direction to pursue. Overall, SOMs are a strong alternative to buffer-based storage of raw data often used in modern-day continual learning methods. Despite being classical algorithms, SOMs as demonstrated in this thesis, have great potential to become a part of solutions to modern challenges in the artificial intelligence.

Bibliography

[Abraham and Robins, 2005] Abraham, W. C. and Robins, A. (2005). Memory retention – the synaptic stability versus plasticity dilemma. *Trends in Neurosciences*, 28(2):73–78.

[Aljundi et al., 2018] Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[Aljundi et al., 2017] Aljundi, R., Chakravarty, P., and Tuytelaars, T. (2017). Expert gate: Lifelong learning with a network of experts. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7120–7129.

[Ayub and Wagner, 2021] Ayub, A. and Wagner, A. (2021). {EEC}: Learning to encode and regenerate images for continual learning. In *International Conference on Learning Representations*.

[Ayub and Wagner, 2020] Ayub, A. and Wagner, A. R. (2020). Cognitively-inspired model for incremental learning using a few examples. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 897–906.

[Bashivan et al., 2019] Bashivan, P., Schrimpf, M., Ajemian, R., Rish, I., Riemer, M., and Tu, Y. (2019). Continual learning with self-organizing maps. *ArXiv*, abs/1904.09330.

[Breard, 2017] Breard, G. (2017). Evaluating self-organizing map quality measures as convergence criteria. page 68.

[Carta et al., 2021] Carta, A., Cossu, A., Errica, F., and Bacciu, D. (2021). Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification. *CoRR*, abs/2103.11750.

[Castro et al., 2018] Castro, F. M., Marin-Jimenez, M. J., Guil, N., Schmid, C., and Alahari, K. (2018). End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[Charbonnier et al., 1994] Charbonnier, P., Blanc-Féraud, L., Aubert, G., and Barlaud, M. (1994). Two deterministic half-quadratic regularization algorithms for computed imaging. *Proceedings of 1st International Conference on Image Processing*, 2:168–172 vol.2.

[Chen and Liu, 2016] Chen, Z. and Liu, B. (2016). Lifelong machine learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*.

[Chen et al., 2018] Chen, Z., Liu, B., Brachman, R., Stone, P., and Rossi, F. (2018).

[Choe, 2013] Choe, Y. (2013). *Hebbian Learning*, pages 1–5. Springer New York, New York, NY.

[Cossu et al., 2021] Cossu, A., Carta, A., Lomonaco, V., and Bacciu, D. (2021). Continual learning for recurrent neural networks: an empirical evaluation. *ArXiv*, abs/2103.07492.

[De Lange et al., 2021] De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Dhar et al., 2019] Dhar, P., Singh, R. V., Peng, K.-C., Wu, Z., and Chellappa, R. (2019). Learning without memorizing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5133–5141.

[Dresp, 2020] Dresp, B. (2020). Seven properties of self-organization in the human brain. *Big Data Cogn. Comput.*, 4:10.

[Ehret et al., 2021] Ehret, B., Henning, C., Cervera, M., Meulemans, A., Oswald, J. V., and Grewe, B. F. (2021). Continual learning in recurrent neural networks. In *International Conference on Learning Representations*.

[Farquhar and Gal, 2018] Farquhar, S. and Gal, Y. (2018). Towards robust evaluations of continual learning. *ArXiv*, abs/1805.09733.

[French, 1992] French, R. (1992). Semi-distributed representations and catastrophic forgetting in connectionist networks. *CONNECTION SCIENCE*, 4:365–377.

[French, 1994] French, R. (1994). Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. *Proceedings of the 16th Annual Cognitive Science Society Conference*.

[French, 1999] French, R. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3:128–135.

[Fritzke, 1994] Fritzke, B. (1994). A growing neural gas network learns topologies. In *NIPS*.

[Furao and Hasegawa, 2006] Furao, S. and Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural networks : the official journal of the International Neural Network Society*, 19 1:90–106.

[Gepperth et al., 2015] Gepperth, A., Hecht, T., Lefort, M., and Korner, U. (2015). Biologically inspired incremental learning for high-dimensional spaces. In *2015 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 269–275.

[Gepperth and Karaoguz, 2017] Gepperth, A. and Karaoguz, C. (2017). Incremental learning with self-organizing maps. In *2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 1–8.

[Gidaris and Komodakis, 2018a] Gidaris, S. and Komodakis, N. (2018a). Dynamic few-shot visual learning without forgetting. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4367–4375.

[Gidaris and Komodakis, 2018b] Gidaris, S. and Komodakis, N. (2018b). Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375.

[Goodfellow et al., 2014] Goodfellow, I., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. (2014). An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211.

[Greco et al., 2019] Greco, C., Plank, B., Fernández, R., and Bernardi, R. (2019). Psycholinguistics meets continual learning: Measuring catastrophic forgetting in visual question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3601–3605, Florence, Italy. Association for Computational Linguistics.

[Grossberg, 1982] Grossberg, S. (1982). Neural principles of learning, perception, development, cognition, and motor control. In *Studies of Mind and Brain*, volume 70. Springer Netherlands.

[Hartono, 2012] Hartono, P. (2012). *Competitive Learning*, pages 671–677. Springer US, Boston, MA.

[Hayes et al., 2019] Hayes, T. L., Cahill, N. D., and Kanan, C. (2019). Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776.

[Hayes et al., 2021] Hayes, T. L., Krishnan, G., Bazhenov, M., Siegelmann, H., Sejnowski, T., and Kanan, C. (2021). Replay in deep learning: Current approaches and missing biological elements. *ArXiv*, abs/2104.04132.

[Isele and Cosgun, 2018] Isele, D. and Cosgun, A. (2018). Selective experience replay for lifelong learning. In *AAAI*.

[Javed and White, 2019a] Javed, K. and White, M. (2019a). Meta-learning representations for continual learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[Javed and White, 2019b] Javed, K. and White, M. (2019b). Meta-learning representations for continual learning. In *NeurIPS*.

[Kang et al., 2020] Kang, W.-Y., Han, C.-H., and Zhang, B.-T. (2020). Discriminative variational autoencoder for continual learning with generative replay.

[Kirkpatrick et al., 2017] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

[Knuth, 1997] Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA.

[Kohonen, 1990] Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

[Kohonen, 1997] Kohonen, T. (1997). The self-organising map, a possible model of brain maps. *Perception*, 26:204 – 204.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90.

[lai Li et al., 2019] lai Li, X., Zhou, Y., Wu, T., Socher, R., and Xiong, C. (2019). Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *ICML*.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lee et al., 2019] Lee, K., Lee, K., Shin, J., and Lee, H. (2019). Overcoming catastrophic forgetting with unlabeled data in the wild. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 312–321.

[Lee et al., 2017] Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. (2017). Overcoming catastrophic forgetting by incremental moment matching. *ArXiv*, abs/1703.08475.

[Li and Hoiem, 2018] Li, Z. and Hoiem, D. (2018). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

[Lipton et al., 2016] Lipton, Z. C., Gao, J., Li, L., Chen, J., and Deng, L. (2016). Combating reinforcement learning's sisyphean curse with intrinsic fear. *ArXiv*, abs/1611.01211.

[Liu et al., 2021] Liu, H., Yang, Y., and Wang, X. (2021). Overcoming catastrophic forgetting in graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):8653–8661.

[Lopez-Paz and Ranzato, 2017] Lopez-Paz, D. and Ranzato, M. (2017). Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6470–6479, Red Hook, NY, USA. Curran Associates Inc.

[Lu et al., 2014] Lu, J., Furao, S., and Zhao, J. (2014). Using self-organizing incremental neural network (soinn) for radial basis function networks. *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2142–2148.

[Ly et al., 2017] Ly, A., Marsman, M., Verhagen, J., Grasman, R., and Wagenmakers, E. (2017). A tutorial on fisher information. *arXiv: Statistics Theory*.

[Löwel and Singer, 1992] Löwel, S. and Singer, W. (1992). Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science*, 255(5041):209–212.

[Mallya and Lazebnik, 2018] Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7765–7773.

[Martinetz and Schulten, 1991] Martinetz, T. and Schulten, K. (1991). A "neural gas" network learns topologies. *Artificial Neural Networks. Elsevier*, page 397–402.

[McCloskey and Cohen, 1989] McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

[Michieli and Zanuttigh, 2021] Michieli, U. and Zanuttigh, P. (2021). Knowledge distillation for incremental learning in semantic segmentation. *Computer Vision and Image Understanding*, 205:103167.

[Morris, 1999] Morris, R. (1999). D.o. hebb: The organization of behavior, wiley: New york; 1949. *Brain Research Bulletin*, 50(5):437.

[Natita et al., 2016] Natita, W., Wiboonsak, W., and Dusadee, S. (2016). Appropriate learning rate and neighborhood function of self-organizing map (som) for specific humidity pattern classification over southern thailand. *International Journal of Modeling and Optimization*, 6:61–65.

[Ororbia, 2021] Ororbia, A. (2021). Continual competitive memory: A neural system for online task-free lifelong learning. *ArXiv*, abs/2106.13300.

[Ororbia and Kifer, 2020] Ororbia, A. and Kifer, D. (2020). The neural coding framework for learning generative models. *ArXiv*, abs/2012.03405.

[Ororbia and Mali, 2021] Ororbia, A. and Mali, A. (2021). Backprop-free reinforcement learning with active neural generative coding.

[Ororbia et al., 2019] Ororbia, A., Mali, A., Kifer, D., and Giles, C. L. (2019). Lifelong neural predictive coding: Sparsity yields less forgetting when learning cumulatively. *ArXiv*, abs/1905.10696.

[Ostapenko et al., 2019] Ostapenko, O., Puscas, M., Klein, T., Jähnichen, P., and Nabi, M. (2019). Learning to remember: A synaptic plasticity driven framework for continual learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11313–11321.

[Parisi et al., 2019] Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

[Polzlbauer, 2004] Polzlbauer, G. (2004). Survey and comparison of quality measures for self-organizing maps. *Proc. fifth Work. Data Anal.*, pages 67–82.

[Rajasegaran et al., 2019] Rajasegaran, J., Hayat, M., Khan, S. H., Khan, F. S., and Shao, L. (2019). Random path selection for continual learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[Rebuffi et al., 2017] Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5533–5542.

[Redondo and Morris, 2011] Redondo, R. and Morris, R. (2011). Making memories last: The synaptic tagging and capture hypothesis. *Nature reviews. Neuroscience*, 12:17–30.

[Ring, 1994] Ring, M. B. (1994). Continual learning in reinforcement environments. In *GMD-Bericht*.

[Roady et al., 2020] Roady, R., Hayes, T. L., Vaidya, H., and Kanan, C. (2020). Stream-51: Streaming classification and novelty detection from videos. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[Rumelhart and McClelland, 1987] Rumelhart, D. E. and McClelland, J. L. (1987). *Feature Discovery by Competitive Learning*, pages 151–193.

[Schwarz et al., 2018] Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y., Pascanu, R., and Hadsell, R. (2018). Progress & compress: A scalable framework for continual learning. *ArXiv*, abs/1805.06370.

[Serra et al., 2018] Serra, J., Suris, D., Miron, M., and Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4548–4557. PMLR.

[Shin et al., 2017] Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 2994–3003, Red Hook, NY, USA. Curran Associates Inc.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. V. D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

[Silver et al., 2013] Silver, D., Yang, Q., and Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*.

[Sokar et al., 2021] Sokar, G., Mocanu, D. C., and Pechenizkiy, M. (2021). Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[Thrun, 1995] Thrun, S. (1995). Is learning the n-th thing any easier than learning the first? In *NIPS*.

[Thrun and Mitchell, 1995] Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics Auton. Syst.*, 15:25–46.

[Venkatesan et al., 2017] Venkatesan, R., Venkateswara, H., Panchanathan, S., and Li, B. (2017). A strategy for an uncompromising incremental learner. *ArXiv*, abs/1705.00744.

[Vesanto and Alhoniemi, 2000] Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600.

[Vettigli, 2018] Vettigli, G. (2018). Minisom: minimalistic and numpy-based implementation of the self organizing map.

[Welford, 1962] Welford, B. P. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420.

[Wikipedia contributors, 2021] Wikipedia contributors (2021). Algorithms for calculating variance — Wikipedia, the free encyclopedia. [Online; accessed 19-November-2021].

[Wiwatcharakoses and Berrar, 2020] Wiwatcharakoses, C. and Berrar, D. P. (2020). Soinn+, a self-organizing incremental neural network for unsupervised learning from noisy data streams. *Expert Syst. Appl.*, 143.

[Wiwatcharakoses and Berrar, 2021] Wiwatcharakoses, C. and Berrar, D. P. (2021). A self-organizing incremental neural network for continual supervised learning. *Expert Syst. Appl.*, 185:115662.

[Wu et al., 2018] Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., Zhang, Z., and Fu, Y. (2018). Incremental classifier learning with generative adversarial networks. *CoRR*, abs/1802.00853.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

[Yin, 2008a] Yin, H. (2008a). *The Self-Organizing Maps: Background, Theories, Extensions and Applications*, volume 115, pages 715–762.

[Yin, 2008b] Yin, H. (2008b). The self-organizing maps: Background, theories, extensions and applications. In *Computational Intelligence: A Compendium*.

[Yoon et al., 2018] Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2018). Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*.

[Zenke et al., 2017] Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995. PMLR.