A study on dual-memory architectures in incremental learning

Behzad Taghipour Zarfsaz

Master Thesis at AI Department of HAW Fulda

Matriculation Number: 1156160
Supervisor: Prof. Dr. Alexander Gepperth
Second Supervisor: Prof. Dr. Reiner Todtenhoefer

Submitted on 17.07.2020
1 Abstract

Although several years passed from the emergence of machine learning science, it is still an infant, and daily progress in this field leads to the exploration of new problems as well. The Artificial Intelligence goal is to mimic the functionality of a biological brain in a robot or any electronic device, and similarly, the emerged problems are sometimes as complicated as the nature of the brain.

Catastrophic forgetting as the central issue in continual learning can be named one of the most critical and challenging issues in machine learning, which explored in the 1980s. Despite the breakneck pace of progress in computer science, it still is not mitigated completely. This article starts with the importance of continual learning, and after explaining the fundamentals, a hybrid bio-inspired model has been introduced. This model implements incremental learning without confronting the catastrophic forgetting.

Inspired by the relations between the hippocampal and neocortical regions of a biological brain, the dual memory concept stands for a composite architecture in which the knowledge acquired by short-term memory, as fast learning network, is transferred to long-term memory to fertile the ground for incremental or online learning. In this article, several experiments conducted to analyze the behavior and the productivity of this model in incremental learning, utilizing a small short-term memory.
Contents

1 Abstract I

List of Figures IV

List of Tables V

2 Introduction 1

2.1 Continual learning .......................... 2
  2.1.1 Necessity of CL in machine learning ... 4
  2.1.2 Catastrophic forgetting ................. 6

2.2 Goals of the study .......................... 8

2.3 Related Works .............................. 9

  2.3.1 Catastrophic forgetting problem in DNN’s 9
  2.3.2 Elastic weight consolidation ............ 10
  2.3.3 Deep Generative Replay .................. 12
  2.3.4 Incrementally Moment Matching ........ 13
  2.3.5 Gepperth and Karaoguz 3-layer architecture 17
  2.3.6 iCarl ..................................... 18

3 Foundations 20

  3.1 Machine learning taxonomy ............... 20
    3.1.1 Supervised Learning ..................... 21
    3.1.2 Unsupervised Learning .................. 21
    3.1.3 Semi-Supervised Learning ............... 26

3.2 Example of problem-solving with DNN ......... 26

4 Implementation 32

  4.1 Deep Neural Network ...................... 34
  4.2 Short-Term Memory ........................ 36

5 Experiments 37

  5.1 Global parameters ........................ 37
  5.2 D5-5 ...................................... 40
  5.3 D9-1 ...................................... 42
  5.4 D2-2-2-2-2 ............................... 44
  5.5 SOM update during training ............... 46
  5.6 Continuous learning ....................... 47

6 Discussion 49

7 Acknowledgement 50
List of Figures

1 Interactions in CLS .................................................. 3
2 Driver-less car ......................................................... 4
3 Object detection in street ............................................ 5
4 Incremental training schema ....................................... 9
5 Testing on all sets ...................................................... 10
6 Testing only on $D_1$ ................................................... 11
7 EWC mechanism Red arrow: EWC Green arrow: L2 Blue arrow: Np Penalty ....................................................... 12
8 Generative Replay performance in incremental learning ... 13
9 Posterior probability computation $\theta$: Initial or previous probability distribution for generating values of weights and biases $y$: The training data .................................................. 14
10 Bayesian Deep Neural Networks Red sinusoidal lines represent the weight generation by distribution .......... 15
11 Graphical illustration of incremental moment matching (IMM) .................................................. 16
12 The performance of IMM in training two disjoint MNIST data sets .................................................. 17
13 The hybrid model used by Gepperth and Karaoguz ........... 18
14 The iCarl algorithm schematic view ............................... 19
15 Sample DNN .......................................................... 20
16 Machine Learning Taxonomy ........................................... 21
17 Supervised Process .................................................... 22
18 Example of a self-organizing map structure .................... 23
19 Example of a self-organizing map of MNIST handwritten digits data set .................................................. 24
20 Example of a 3D self-organizing map for showing special distribution of data .................................................. 25
21 SOM learning process .................................................. 26
22 Applying Global-Ordering to batch process .................... 27
23 Decay reduction difference .......................................... 27
24 Single-layer Perceptron .............................................. 28
25 Images and one-hot vector labels dataset ....................... 29
26 Images and one-hot vector labels dataset ....................... 29
27 Network training process ............................................. 30
28 The steps of Gradient Descent in each iteration. $f$: The Loss value $\alpha$: The learning rate $\theta_n$: The value of loss in step $n$ $\theta$: The minimum value of loss .................................................. 32
29 The training workflow .................................................. 33
30 The Short-Term memory .............................................. 37
31 Feeding the training loop by batches .............................. 39
32 The exponential decay of the SOM radius ....................... 40
33 D5-5 Confusion matrix after $D_1$. The predictions shown as the rows, and the real target values depicted as columns .................................................. 41
34 D5-5 Confusion matrix after $D_2$. The predictions shown as the rows, and the real target values depicted as columns .................................................. 41
D5-5 evaluation. .................................................. 42
D9-1 Confusion matrix after $D_1$. The predictions shown as the rows, and the real target values depicted as columns. .................................. 43
D9-1 Confusion matrix after $D_2$. The predictions shown as the rows, and the real target values depicted as columns. .................................. 43
D9-1 evaluation. .................................................. 44
D2-2-2-2-2 Decay. ................................................. 45
D2-2-2-2-2 evaluation. .......................................... 45
D2-2-2-2-2 Correct classifications in batches. The vertical axis shows the correct classifications of 1000 test samples, and the horizontal axis, indicates the total batch numbers. ($D_1$ contains 2000 samples and with a batch size of 100, the $D_1$ would execute over 20 batches. After replaying the 20% of data from STM in $D_2$, the total sample numbers would be 4000, and consequently, the sub-task would include 40 batches. This pattern repeated for all sub-tasks, and totally the entire task would contain 300 batches.) .................................................. 46
SOM updates during the training phases ......................... 47
D5-5 Continuous learning performance. The vertical axis indicates the number of correct classification per class in a test dataset with a size of 1000. And the horizontal axis shows the total number of train batches. ... 48

List of Tables

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Task parameters</td>
<td>39</td>
</tr>
</tbody>
</table>
2 Introduction

Learning means gaining knowledge and experience or skill, which nowadays is not restricted to human beings exclusively. Robots are at service almost in any industry from medical to automotive or aviation, and their job is not only operating a predefined repetitive task anymore.

A monotonous function is easily definable to an agent by programming it on the production phase. However, a modern AI agent (robot) required to possess the ability to learn progressively, broadening the knowledge-base, improving the interaction quality with the surrounding world, remembering multiple tasks, and consequently increasing the throughput by making better decisions in various situations.\[1\]

A straightforward example of an AI agent can be a smart vacuum. A new electric-vacuum does not possess any information about the topography of the house and furniture positions. The device starts with surfing around the floor and colliding with the objects. It generates a topological map of the house using heuristic-based algorithms. Now, what happens if the decoration of one specific room changes? Should the robot start learning the new topology for all other rooms from scratch? Or is it expected to be capable of determining the latest changes and update its knowledge-base?

Therefore, learning as a primary factor of evolution is a lifelong process due to environmental changes. An environmental change can be the progress in technology or hardware for a robot or habitat destruction from an animal’s perspective. The flexibility in the behavior according to the situation is a characteristic of intelligence for any living organism, and it should be an essential feature of any smart agent, so that the quality of an intelligent system lies in the capability of adaptability with new environments, self-correction, and avoid degradation of performance. Correspondingly considering the dynamic ever-changing environments in the real-world, the incremental learning becomes an essential part of AI.

What is the main difference between legacy engineering solutions and machine learning in finding solutions? In conventional engineering methods, an expert team of engineers starts with analyzing the problem in-depth to find the main factor of it. Then a mathematical model can be generated from this analysis. For example, imagine a team of engineers trying to create a mathematical model for speech translation. This task comprised of generating models by groups of linguistics, psychologists, and signal processing engineers, which is too costly. Besides, it might take too much effort to create a decent model or algorithm to solve the problems. Another drawback is the probable obsolescence of the model due to any slight change in the main factor. \[2\]

While conventional engineering based on domain knowledge, the machine learning approach tries to solve the problem and generate the model by processing a large set
of data and labels and optimizing the resulting model using complicated mathematical equations for comparing the desired and current results per data. [2]

2.1 Continual learning

The "Continual Learning" (CL) or "Lifelong Learning" terms refer to the capability of continuously learning over time and acquiring the new knowledge while retaining the previous knowledge-base. Humans and animals achieve this goal by an astonishing set of neurocognitive mechanisms embedded in their brains. [1]

McClelland, McNaughton, and O’Reilly proposed a theory called "Complementary learning systems" (CLS)[3] that explains the procedure of learning in mammalian brains in two main sub-systems. The first sub-system is the mammalian hippocampus, where the brain quickly learns specific experiences without using any parametric representation of information. The second is the mammalian neocortex, in which the learning accomplished slowly by generalizing the structure of knowledge using a parametric representation. Over time, gradually, the learnings in the first sub-system is consolidated and transferred to the second sub-system.[4]

Figure 1 indicates the interactions between the hippocampus and the neocortex in a mammalian brain. The deep inside the brain has been shown on a medial surface. This figure indicates the primary sensory and motor cortices indicated by dark yellow, Medial Temporal Lobe (MTL) by broken lines, hippocampus by dark grey, and surrounding MTL cortices by light grey. The connections between neocortical association areas, represented by green arrows. Blue arrows demonstrate the connection between neocortical areas and the MTL. The CLS theory describes the blue and green connections as the structure-sensitive neocortical learning system. The connection within the MTL comprised of two red-colored arrows that exhibit rapid synaptic plasticity. Dark-red arrows denote the connections within the hippocampus, and light-red arrows represent the connections between the hippocampus and surrounding MTL cortices. If you consider the size of red arrows, you notice that dark ones are greater, which indicates the rapid binding of elements into the hippocampus. Consolidations happen by the hippocampal activity and spreading it to neocortical association areas via blue arrows, and as a result supporting learning with green connections between neocortical areas. [4]

Imagine a student who learns the most elementary steps of calculation in the first grade. At the same time, he learns reading and writing in his language. He improves his skills in both calculation and language aspects for several years by learning more complicated mathematics and literature. However, this progress is not possible if he cannot recall the most basic calculations he learned in first grade. Besides, he uses the language in defining or understanding the complicated problems, formulates them, and finally
2 Introduction

Bidirectional connections (blue) link neocortical representations to the hippocampus/MTL for storage, retrieval, and replay.

Connections within and among neocortical areas (green) support gradual acquisition of structured knowledge through interleaved learning.

Rapid learning in connections within hippocampus (red) supports initial learning of arbitrary new information.

Figure 1: Interactions in CLS

Source: Adapted from [4]

presenting the solution. Furthermore, it might be necessary to correct some wrongly learned mathematical basics in the past. Consequently, the mathematical subjects from the first moment are useful in any stage of learning and progress, but also the knowledge in other aspects also helps in broadening the knowledge and learning new skills.

Similarly, if an AI agent or robot is not capable of expanding its knowledge-base and acquiring experiences, there would be no difference between it and a mono-functional device. This smart agent can be an automatic electric vacuum, an industrial welding robot, a mobile application for detecting an object, or a much more complicated device like a driver-less public vehicle. An autonomous agent interacting with a dynamic environment in the real world is required to learn, expand the knowledge-base, and gradually summit its productivity to the maximum amount.
2 Introduction

2.1.1 Necessity of CL in machine learning

In general, classification, object detection, and many other functionalities done by a smart agent are based on the digital data retrieved from electronic devices and sensors. A machine learning algorithm deals with an unlimited number of various data captured from the environment. Also, by progressing in hardware technologies and retrieving high-quality data using these sensors, the dimensions of data increases day by day. In consequence, the artificial intelligence requires more sophisticated, progressive, optimized and fast machine learning algorithms.

Driver-less cars are other examples of complicated projects that emphasize the necessity of continual learning. A complex AI agent like this should possess a robust learning algorithm to detect obstacles, other cars, animals, and pedestrians to perform safely. Figure 2 shows a driverless car with a camera in front of it, which intended to capture the front view in designated frequency.

A human driver can perform several tasks concurrently to control the car and prevent collisions:

- Recognize the objects and distinguish the barriers, pedestrians, road signs, and other cars
- Estimate the distance intuitively
- Recognize the approaching or leaving objects
2 Introduction

- Update the parameters in case of any change in car dimensions
- Perform all above items real-time

The only first item above is enough to explain the complexity of this task. We live in a 3D world. However, a captured scene either by a camera or by human eyes is in 2D dimensions. The human brain can combine these 2D images in very high frequency and generate a 3D model for estimating the volume of an object. Considering the numerous types and models of cars in different sizes and their different views from various angles, object detection and recognition is a very complicated task to fulfill. Figure 3 shows an example of object detection by smart car observes from the front camera.

Figure 3: Object detection in street

Source: [5]

Almost every scene captured by the camera is unprecedented, and generating a set of all possible scenes with the objects in different sizes and angles is impossible. Consequently, the desired accuracy cannot be achieved in one run, and the car should re-train its model to acquire new experiences by observing and analyzing new scenes continuously.

The time is passing, and the previously raw data, such as observed street scenes, would not be available at any moment. One simple solution for implementing continual learning is saving these data from scratch, and re-train the model using all former
and new data each time that the agent receives a new signal or data. However, technically speaking, this simple solution is impractical in real-world due to the fact that an agent or robot uses limited resources such as memory or CPU. Consequently, the agent should be able to train and update its knowledge deferentially or incrementally. The data retrieved from the environment can be a vast or small set of samples, or even a single sample at any moment. Inevitably, the smart agent should be able to incrementally re-training the model using this small data set, fine-tuning it, and integrating it with prior knowledge while keeping them consolidated with minimum performance degradation.

The necessity of continual learning can be categorized as follows:

- Since the data retrieving from the environment is sequential in time, accessing previous raw data is not possible.
- Saving the entire data set from the first moment is not possible or at least is not optimal, because of resource limitations in systems.
- Even previously gained knowledge can be changed due to the environment changes over time. Therefore, prior learning should be updated in specific circumstances. But the agent cannot know when and where, or in which stage it happened.

2.1.2 Catastrophic forgetting

As mentioned in the previous section, resources have limitations in the majority of cases, and also the robot is expected to respond in real-time or at least as quickly as possible. Thus, saving and retrieving the raw training data from the first moment, and re-training the network on entire prior and new data sets in each attempt, is a time, memory and CPU demanding process. So using the simple solution mentioned in section 2.1.1 is impossible, or at least useless in real-world.

The life-long learning remains as a challenge in machine learning, because the continuous acquisition of the dynamically distributed data from the environments in reality, usually leads to a chronological phenomenon called "Catastrophic Forgetting" or "Catastrophic Interference" which is considered as the main drawback of modern artificial neural networks.[1]

Forgetting is shared between all neuro-cognitive systems, just like our brains. A human’s brain gradually forgets the old information as new knowledge acquired, and it is a necessary process for learning. An active form of forgetting moves all unnecessary information at that moment to our unconscious memory, and helps us concentrate on a specific task. Human and animals do not forget the entire knowledge at once and the majority of previously learning remains in the unconscious memory and come into mind again when they needed.[6] Contrarily, forgetting in machines can be catastrophic,
meaning that the system faces an abrupt performance decrease and, in the worst-case, overwriting the entire former knowledge with new data.

In the end '80s, many neural network issues in supervised learning, such as error back-propagation, had been solved. However, another fundamental difficulty had been explored by McCloskey, Cohen [7] and Ratcliff [8]. They observed that under specific circumstances, a fundamental limitation emerges in the functionality of Perceptron (section 3.2), by which the process of learning a new set of patterns, erases the complete knowledge of the network that previously learned. The "Catastrophic Forgetting" or "Catastrophic Interference" is the name that they assigned to this phenomenon. The catastrophic forgetting is a variance of a more general problem in memories, called the "stability-plasticity" problem.[6]

Findings in neuroscience reveal a neurocognitive mechanism in the human brain that protects previous learnings from forgetting while acquiring new knowledge. This mechanism helps humans to continually learn, transfer the knowledge and experiences throughout its lifespan, and consolidate the long-term memory simultaneously. Furthermore, this mechanism can adjust a balance between various parts of the brain in the process of experience development, so-called "stability-plasticity" balance. This balance regards that a smart system should be able to integrate the new knowledge while consolidating prior knowledge using an internal mechanism to avoid catastrophic forgetting. In other words, the system should stabilize the knowledge and show its plasticity or flexibility to update the knowledge-base simultaneously. [1]

The catastrophic forgetting is considered one of the most troublesome barriers in artificial learning since it eliminates the ability of continual learning. The learning process cannot be fulfilled by effectively erasing the previous knowledge during the new learning task. Simply this problem is the inability of an artificial neural network in continual learning by a new set of patterns without replicating the previous data. In training deep neural networks (DNN), this dilemma forces to repeat the entire learning process, including all data sets since the first moment, otherwise the probability of facing catastrophic forgetting in training is high when an original pattern is fed to the network.

We live in a fully dynamic world in which the atmosphere, and accordingly, its properties, are unstable over time. Imagine what a disaster can happen if the self-driving car example mentioned in section 2.1.1 forgets every prior knowledge in the middle of a street while driving itself at a dangerous speed? This example, and many other cases, especially in industry, machinery, also intelligent systems that help the pilot to control an airplane, can be situations of life-threatening potential perils of catastrophic forgetting.

Indeed, artificial intelligence, as its name refers, is the simulation of human intelligence in artificial devices. Like a human, a smart agent should interactively commu-
nicate with this dynamic world and recall prior knowledge when demanded. Conse-
quently, discovering real-world practical solutions for preventing catastrophic forget-
ting in the road of incremental learning seems inescapable.

2.2 Goals of the study

Inspired by the stunning nature of the human’s brain, this study aims to challenge a hy-
brid model of a Self-organized Map (SOM), and an ANN, and study the efficiency and
productivity of dual memory architectures in avoiding catastrophic forgetting. In this
model, the neural network performs active learning, while the SOM layer considered as
the consolidator for previously acquired knowledge. Several parameters are assignable
in this implementation to monitor the behavior of the model training. Furthermore,
before explaining the model, few recent studies are briefly explained to have a better
definition of Catastrophic Forgetting in neural networks, and compare the performance
of a few solutions and strategies in overcoming this issue.

Briefly, the goals of this study are:

• Summarizing a few essential related works and studies on catastrophic forgetting
  and methods to prevent it.

• Explaining the implementation of a hybrid model comprising of a Self-Organizing
  Map, and a Neural Network and one Short-term memory for incremental learning
  purpose.

• Conducting three different incremental experiments for monitoring the Self-Organizing
  map changes and analyzing the confusion matrix after each sub-task to confirm
  the learning of new classes.

• Analyzing the performance of class-incremental training in this model on MNIST
  classes by limiting the number of samples and the SOM dimensions (all experi-
  ments have been conducted with $10 \times 10$ SOM and 1000 samples per class).

• Analyzing the performance of the model in alleviating the catastrophic forgetting.

In implementation section (section 4) the exact algorithm of this model has been ex-
plained, and in experiences section (section 5) the model performance has been tested
over three tasks with various combination of MNIST classes and using the diagrams the
performance in each sub-task has been monitored for both training and testing steps.

Also, to challenge the model in alleviating the catastrophic forgetting, the training
should continue in longer periods. Therefore, one of the experiments repeated to sim-
ulate the life-long learning and monitoring the behavior and vulnerability of the model
against forgetting.
2.3 Related Works

As mentioned in section 2.1.2, one of the most annoying obstructions in using artificial intelligence for solving real-world problems is its inability to learn from the ever-streaming flow of data progressively, while preserving previous knowledge. In this section, a few significant studies conducted in this regard have been shortly summarized.

2.3.1 Catastrophic forgetting problem in DNN’s

Let take a deeper look at catastrophic forgetting through a study named “Catastrophic forgetting: Still a problem for DNN”[9]. In this study, deep neural networks in class-incremental training for visual problems have been studied on several DNN and CNN architectures, and the behavior of catastrophic forgetting has been recorded. They divide the entire MNIST data set to 2 subsets $D_1$ and $D_2$. The model is sequentially trained first on $D_1$, and then on $D_2$ subsets.

In figure 5 and 6, the outcome of training and testing two subsets from MNIST data set demonstrated. Figure 4 shows the overall training schema in which, the first phase comprised of training and testing the $D_1$ subset, and the second phase includes the training and testing the model on $D_2$ subset, plus additional testing on $D_1$ to monitor the performance degradation on $D_1$. [9]

Figures 5 and 6 show the training and testing phases in which a white background indicates the first phase, and a grey background shows the second phase.
In figure 6, the CF is evident by curved red lines. With a precise look, in 5000 iterations, the model can predict with high accuracy (near 1). However, after training the $D_2$, the accuracy of the model in $D_1$ is downgraded.

In this study [9], the test cases are applied on sub-sets with "N" randomly selected classes for phase one ($D_1$), and "M" randomly selected sets for phase two ($D_2$), and both sub-tasks have following characteristics:

- $D_1$ is not available in phase 2. and might be much bigger than $D_2$.
- $D_2$ is not available in phase 1.

Tests are conducted on tasks with "DP5-5" ($D_1$=5 classes, and $D_2$=5 classes), "DP9-1", and finally with "DP10-10" combinations, using several DNN architectures. The results showed that although some architectures can manage the CF better than others, however, none of the current DNN architectures can prevent it completely [9].

### 2.3.2 Elastic weight consolidation

As discussed in section 2.2, the brain enables continual learning by utilizing a mechanism for consolidating the previous knowledge and preventing the harmful interference
between newly acquired and prior knowledge. This mechanism implements a reduction on the plasticity of synapses so-called synaptic consolidation.

EWC (Elastic weight consolidation) is a similar operation in an artificial neural network that forces the model to retain the significant parameters close to previous values. In deep neural networks, learning a task is accomplished by changing Weights, and Biases parameters over numerous iterations and many variations of these parameters during the training, result in the same values. Now if we divide the training task, to task A and task B, and show the variable parameters as \( \theta \) (so that the parameters for task A and B would be \( \theta_A^*, \theta_B^* \) respectively), during the training of task B, the EWC forces the parameters to stay in the region of low error of task A (the region that A has the highest accuracy), which results in preserving the performance of task A. The schematic presentation of this mechanism depicted in figure 7. EWC implements this constrain by a quadratic penalty that can be assumed as a spring anchoring the \( \theta_A^* \).

In figure 7 the pink and blue trajectories indicate the areas of the parameter leading to excellent performance in task A and task B, respectively. Obviously, according to above definitions, the parameters are \( \theta_A^* \) after training the task A. At this moment, if the model starts training the task B by using gradient steps of task B only (blue arrow), the loss would be minimal for task B. However, it loses whatever that learned in task A. Contrarily, if the model retains the parameters equal to \( \theta_A^* \), it remembers the task A without learning anything in training task B. EWC can find a solution to minimize the
2 Introduction

2.3.3 Deep Generative Replay

Deep Generative Replay, is another novel framework inspired by the nature of the brain’s hippocampus in the form of a short-term memory.[11] Simply replaying all previously acquired knowledge can alleviate the catastrophic forgetting in incremental learning. However, as mentioned several times before, it needs a lot of memory, which is infeasible in the real world, and memory is always limited. In the proposed deep generative framework in [11], the model keeps the prior knowledge without referring to the past data. This model retains the previously acquired knowledge by a mechanism that replies to the generated pseudo-data concurrently. This pseudo-data is being generated by mimicking the past data using a GAN (generative adversarial networks) [12] network.

The old task can be represented by pairing the generated pseudo-data with the corresponding responses from the solver of the previous task (generator-solver model). Therefore, this generator-solver model can produce fake data with all desired targets for training the new task. The model can train the new task without forgetting the pre-
An experiment proposed in [12] to show the ability of the Generative Replay technique on recollecting the past knowledge (in sequential training of disjoint data), in which the models are trained sequentially on 5 tasks. Each task defined to classify MNIST data set for 2 out of 10 classes so that the networks have given \([0, 1]\) classes in the first task, \([2, 3]\) classes in the second task, and with the same pattern to the end. Figure 8 shows the results obtained in this incremental learning experience. In this figure, the model with the generative replay algorithm is denoted as GR, and ER signifies the model with actual real data of the past solver network. Particularly the GR model can achieve the test performance imminent to perfect bound. [12]

![Figure 8: Generative Replay performance in incremental learning](source)

Source: [11]

2.3.4 Incrementally Moment Matching

Before explaining the study about IMM’s, we should look at Bayesian neural networks (BNN), and their fundamentals. BNN’s involve two main components: a probabilistic or statistical model and a neural network. BNN’s are useful in extracting plenty of missing information from small data sets and calculating the uncertainty in prediction. [13] [14]

A standard neural network optimizes the model by updating the weights using the maximum likelihood estimation (MLE) during the training. However, for many reasons, this approach is unsatisfactory since the uncertainty has been ignored in MLE, and feed-forward usually is prone to over-fitting. Bayesian network corrects this flaw
2 Introduction

\[ Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{Pr(y)} \]

Figure 9: Posterior probability computation

\( \theta \): Initial or previous probability distribution for generating values of weights and biases

\( y \): The training data

Source: Adapted from [13]

in a great deal by calculating the posterior parameters using an equation shown in figure 9 and attaching these distributions on network parameters (figure 10). The outputs of each layer represent the probability distribution that shows the uncertainty of the output. Therefore, in the case of feeding a new sample, this uncertainty would be high. The key idea in this type of neural networks is attaining the next probability distribution for producing values of weights and biases (Posterior), by calculating it using the training data and prior distribution. [13, 14]

Incrementally Moment Matching (IMM) [16] also is a method for overcoming catastrophic forgetting using a technique that matches the moment of the posterior distribution in the neural network in first and second training phases respectively. IMM uses the Bayesian neural networks (BNN) [13], which introduces uncertainty for the parameters in the neural network, and calculates the posterior. IMM generates a mixture of Gaussian posterior with parameters of network in each sub-task, for using it as a Gaussian distribution for a combined task. [16]

The calculations of mean, and mode value shown in figure 11 are as following equations:

\[ \mu^{Mode}_{1:2} = \frac{\sum_{1}^{-1} \mu_1 + \sum_{2}^{-1} \mu_2}{\sum_{1}^{-1} + \sum_{2}^{-1}} \]  \( (1) \)

\( ^1 \)Moment is a mathematical statistics calculation for calculating the probability distribution’s mean, variance, and skewness.
Two different moment matching approaches introduced for generating this mixture in [16]:

- mean-IMM: simply averages the parameters of two networks.
- mode-IMM: uses Laplacian approximation for merging the parameters of two networks.

In addition to the calculation of mean or mode IMM for the combined network, the IMM uses various transfer techniques for finding a feasible $\mu_2$ for better performance in the combined network. Weight transfer, initializes the parameters for the new task, using the parameters of the previous task.

$$\mu_{1:2}^{Mean} = \frac{\mu_1 + \mu_2}{2}$$  \hspace{1cm} (2)

$$\mu_1 \rightarrow \mu_2$$  \hspace{1cm} (3)
L2-transfer is a variance of l2-regularization that uses the distance between $\mu_k$ and $\mu_k - 1$ to calculate the parameters in the second task.

$$\|\mu_2 - \mu_1\|^2_2$$ (4)

Drop-transfer is a variant of dropout in which the $\mu_k - 1$ considered as zero-point.

$$\mu_1 + 2\text{dropout}(\mu_2 - \mu_1)$$ (5)

Figure 11 depicts the performance of IMM in incremental training of two disjoint MNIST data-set with several transfer functions (the $\alpha$ represents a hyperparameter that is used

\[\text{Figure 11: Graphical illustration of incremental moment matching (IMM)}\]

Source: Adapted from [16]

\[\text{Figure 12: Performance of IMM in incremental training of two disjoint MNIST data-set with several transfer functions)}\]

\[\text{Figure 12: Performance of IMM in incremental training of two disjoint MNIST data-set with several transfer functions)}\]

---

2 Regularization is the process of introducing extra calculations on weights, to overcome the overfitting.

3 L2-Regularization or Ridge Regression is a type of regularization in which the calculation of the loss uses the sum of the squares of all the feature weights.

4 Dropout is another type of regularization introduced by Google.
for balancing the data between the previous and the new task). [16]

Figure 12: The performance of IMM in training two disjoint MNIST data sets.

Source: Adapted from [16]

2.3.5 Gepperth and Karaoguz 3-layer architecture

In a study conducted by Gepperth and Karaoguz [17], an objective is a bio-inspired approach for appending a new perceptual object to a trained model without confronting catastrophic forgetting. This study, in particular, targets the perceptual problems with high data dimensionality (> 1000) for robotic scenarios. They intend to create a hybrid model, capable of training the classes as displayed in figure [13][17].

In the three-layer model depicted in figure [13], they used a topographically organized prototype as the hidden layer, which performs the approximation scheme for classes according to their neighborhood or similarity. A readout mechanism between the hidden
and output layer is implemented using a simple linear regression for evaluating the prediction made by neighborhood function. The hidden layer has been updated only when facing the wrong classifications by saving them in short-term queue memory. Another advantage of using this topographical map as a hidden layer is that the excessive deviation detection by the hidden layer indicates the feeding of an unknown class.\cite{17}

Using this architecture, they achieved an adequate performance in high dimensional, real-world perceptual classification. In the model, the incremental addition of new classes to the trained model results in only minimal performance deterioration. Furthermore, by utilizing the short-term memory mechanism, incremental learning could be accomplished with a simpler single-phase learning scheme, and at the same time, the reaction could be almost instant, instead of thousands of iteration over samples to achieve acceptable convergence.\cite{17}

2.3.6 iCaRL

The "iCaRL"\cite{18} (Incremental Classifier and Representation Learning) is the name of another similar training strategy, for learning the classifiers, and at the same time a feature representation. iCaRL uses dynamically selected sets of image exemplars from the data stream for classification. Additionally, iCaRL ensures that the number of sets
for identical exemplar does not exceed from a fixed parameter K (memory size). It uses the "nearest-mean-of-exemplars" rule for classification, which means that the

![Figure 14: The iCarl algorithm schematic view](image)

"mean" value of all exemplars has been calculated, and finally, the classifier's output is the nearest prototype in a feature map of so far observed classes (figure 14). Whenever a new class is available, the algorithm calls an update procedure for updating the internal knowledge of the model. Consequently, this strategy has the following advantages: [18]

- Since it uses a nearest-mean-of-exemplars classifier, it needs to store a small number of exemplars when the representation changes.

- It uses the exemplars combined with distillation[^5] to prevent catastrophic forgetting.

However, despite the above advantages, iCaRL's performance still is lower than models that train all classes simultaneously. [18]

[^5]: Distillation is a machine learning algorithm to improve the performance by extracting the crucial features from the given data, and consequently produce better predictions. [19]
3 Foundations

An ANN (Artificial Neural Network) is a simulation of the human biological brain in which neurons are connected by synapses (weight in neural networks). This architecture can gain experience and resolve problems in numerous fields like economics, technology, statistics, and so forth. In a straightforward format, the data from one neuron or unit passes through weight to a neuron in the next level while a simple linear function has been applied to it. [20]

Deep learning refers to a neural network that functions with one or more layers between input and output. The layers between the input and output called Hidden layers. Figure 15 illustrates a deep neural network with 4 hidden layers.

![Sample DNN](image)

Figure 15: Sample DNN

3.1 Machine learning taxonomy

Machine learning is applicable in multiple different styles that can be categorized by the way of interaction between the model and experiences, environment, or known results. Although a "Semi-Supervised" learning method has been used in this study, it worth explaining three significant methods of machine learning (Supervised, Unsupervised, and Semi-Supervised) briefly, to better understanding the problem in incremental learning.
3 Foundations

3.1.1 Supervised Learning

In supervised learning, the algorithm maps the input data to desired outputs, and it is prevalent in solving the classification problems by neural networks, such as image recognition, spam email detection, and many other similar problems. In this approach, several mathematical equations can be used to calculate the error of the network in conjunction with optimizers such as "Stochastic Gradient-Descent" or "Adam" to minimize the error throughout numerous iterations, by comparing the prediction and the desired output(label) over each iteration. [21] Figure 17 shows the process of a supervised learning.

3.1.2 Unsupervised Learning

Unsupervised learning is a much more laborious method to perform since no label or designated desired value contributes to the learning process, and usually, in unsupervised learning algorithms, a reward system is used to indicate the success. On the contrary, a punish system determines the error so that the goal of the model is to maximize the reward, not classify the inputs. [21]
Among several solutions to unsupervised learning, the SOM (Self-Organizing Map) is the concentration point since used in this study. SOM architecture introduced by Finish Professor Teuvo Kohonen in the 1980s (so-called Kohonen Map) and uses an unsupervised technique to generate a discretized representation of high dimensional data (usually 2D or 3D). A neurobiologically inspired algorithm for generating a topologically-preserved map which used extensively as a method for clusterization and visualization of the data in the exploratory data analysis. [22] The generated representation called the feature map, which preserves the similarity of data in the map, and "Quantization Error" can calculate the error rate of the mapping. [23]

Figure 18 shows an example of SOM architecture. The input vector is mapped to all weight nodes, and the BMU (showed by the white circle becomes similar to the winner vector. Also, the BMU’s close neighbors (red circles) and far neighbors (violet circles)
Figure 18: Example of a self-organizing map structure.

become similar to the vector according to their distance from the BMU. Also in figures 19 and 20 you can see an interesting examples of 2D and 3D samples of SOM from [1].

Let dig into the algorithm and the implementation of a SOM more in detail. Although several implementations of SOM ubiquitous on the internet, in general, they follow almost the same algorithm as following steps for a 2D, $N \times N$ SOM:

1. Initializing a $N \times N$ array with random values, as initial weights.

2. Selecting a vector (or a batch of vectors)\(^{6}\)

3. The selected vector’s similarity is assessed with each weight node, and the most similar node (the **Winner** node) is known as Best Matching Unit (**BMU**).

4. The neighborhood of the BMU is calculated using the Radius, and Learning-Rate parameters. This neighborhood value decreases gradually by passing through each sample.

\(^{6}\)In the case of using a batch process, the weights should be broadcast to the number of samples in zero Axis, and at the same time, each selected vector should be broadcast to $N \times N$ in the first Axis.
5. The BMU (winner weight node) and its neighbors are rewarded by becoming more similar to the sample. The most severe homogenization is applied to BMU, and its severity decreases by getting farther from the BMU.

6. Repeating steps 2 to 5, to cover all samples.

The figure shows the process of learning in a SOM. The number of iterations and sample vectors are equal. Weights in iteration $k + 1$ is the result of updated weights by a neighbourhood function in iteration $k$.

$$w_{ij}(k + 1) = w_{ij}(k) + \alpha_i(k) h_{ij}(k) [x(k) - w_{ij}(k)]$$  \hspace{1cm} (6)

In the equation, the $\alpha$ is the learning rate of the SOM in iteration $k$, and the $h$ shows the neighborhood function in epoch $k$. The neighborhood is a function of the BMU and the radius in each iteration. Both learning-rate ($\alpha$), and radius ($\sigma$) are exponentially decayed by time (iteration):

$$\sigma(k) = \sigma(0) \cdot e^{-\frac{k}{\lambda}}$$  \hspace{1cm} (7)
Figure 20: Example of a 3D self-organizing map for showing special distribution of data

Source: [24]

\[ \alpha(k) = \alpha(0) \cdot e^{-\frac{k}{\lambda}} \tag{8} \]

The calculation of radius, learning-rate, and BMU in each epoch for input vector \( \vec{x} \), is demonstrated by equations 7, 8, and 9 respectively. In 7, 8 the \( \lambda \) is the number of total iterations, or the total number of samples.

\[
d = \min \left( \| \vec{x} - \vec{w}_{ij} \| \right) = \min \left( \sqrt{\sum_{k=0}^{n} \left[ \vec{x}(k) - \vec{w}_{ij}(k) \right]^2} \right) \tag{9}
\]

Finally the neighborhood function that will be applied on previous weights, can be calculated by equation 10.

\[
h_{ij}(k) = e^{\frac{-d^2}{2\sigma^2(k)}} \tag{10}
\]
3 Foundations

In batch processing, when a large batch feeds the SOM, the convergence of SOM would be inadequate, due to the rapid decrease of decay function. Therefore, for resolving this issue, a scalar (is changeable by parameters) named “Global Ordering” (GO) is used to retain the decay function constant until GO'th iteration. In figure 22, the "A" image shows the $10 \times 10$ SOM training with 1000 samples and a batch size of 10. As you can see in the figure, the SOM convergence is insufficient, and in the "B" image, the same training is done by Global Ordering value of 100, meaning that after training the 100'th batch, the radius, and learning rate, start decreasing (figure 23).

3.1.3 Semi-Supervised Learning

The semi-supervised learning is a type of machine learning that is half-way between supervised and unsupervised learning, aiming to overcome the impediments of these learning styles. Supervised learning demands a vast set of data and known labels, moreover it is a time and cost-effective task. On the other hand, unsupervised learning clusters the data based on their similarity without requiring any labeled data, with an inability to accurately clustering unknown data.[26]

3.2 Example of problem-solving with DNN

The term "Perceptron" refers to an algorithm for supervised learning in classification tasks, such as digit recognition. As an example, in an image classification problem, this
algorithm is applied for each neuron in all layers (in a fully connected network like figure [15]). The results passed to all neurons in the next layer and finally to the output layer (Forward Pass\textsuperscript{7}).

In the next steps, the network compares the output layer with desired labels, and the

\textsuperscript{7}Passing the result of the perceptron to neurons in the next layer.
similarity/dissimilarity between the outputs (also called predictions or logits), is determined by Accuracy and Loss functions. The network then tries to adjust the Weights and Biases parameters to minimize the loss and maximize the accuracy, using mathematical algorithms during numerous iterations over the network trace (Back Propagation).

Imagine we want to construct a simple deep neural network like figure 15 to detect weapon, alcohol, and drug images to prohibit the user from uploading an image that contains these objects. To solve image detection problems, usually more advanced algorithms (e.g., Convolutional Neural Network or CNN) are used in the real world, however, to simplify the explanation, a simple DNN is used in this example.

Assume that all images are uncolored (black and white), and the dimensions of all images are 50x50 pixels (image cropping and resizing algorithms are applicable before the process) so that if we flatten the pixels, as a one-dimension array, we would have 2500 pixels. Each pixel is a floating number showing its brightness. Moreover, assume that we have an array of 1000 images with their labels (an array of 1000 one-hot vector as you can see in figures 25 and 26).

---

8 A backward pass through the network while adjusting the parameters to start the forward-pass again.
9 One-hot vector is a flat vector in which the index of 1 indicates the target value. For example, for three items of weapon, alcohol, and drug, the one-hot vector is a 3-element vector. If the value 1 located in index 0, indicates that target is a weapon, 1 in index 1 indicates alcohol, and 1 in index 2 shows that target image contains drug.
Consequently, we would have 1000 samples, and each sample has 2500 input neurons. Since the output is a one-hot vector of 3 elements, the output layer should have 3 neurons. By \(N\) iteration over entire data set on a network like shown in figure 27, applying the perceptron calculation for each transition, and optimize the parameters after each iteration (in the neural network is called an epoch), we would obtain a trained network with satisfactory accuracy.

The activation function is shown in figure 24 called RELU, that maps negative results.
to 0 and keeps the value of positive results.

\[ R(x) = \begin{cases} 
  x & \text{if } x \geq 0 \\
  0 & \text{if } x < 0 
\end{cases} \]  

(11)

The one-hot vector is an ideal format for output and usually in practice, each element of the network output layer would have a value between 0 and 1. As it is evident in one-hot vector terminology, just one element should have value 1 at the time. Thus, the output should be normalized as equations [12] [13]
3 Foundations

\[ \sum_{i=1}^{N} output_i = 1 \] (12)

\[ \forall output_i \in [0, 1] \] (13)

As a common practice in such solutions, the activation function used for mapping the outputs to labels is Softmax:

\[ Sm(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \] (14)

Finally, in the result vector (e.g. \([0.3, 0.5, 0.2]\)), the prediction is the index of the maximum value. In the above example, result is the index of 0.5 which is 1, and prediction shows that the image contains alcohol (\([0.3, 0.5, 0.2]\) map \(\mapsto [\text{weapon, alcohol, drug}]\)).

The accuracy of the result is the most critical factor in a prediction, and the main concentration in machine learning is to increase this factor to the maximum amount. The accuracy of classification is the rate of correct predictions for all results. In classification problems using one-hot vector, usually, the correct prediction is the one that has the maximum value in the same index as target (equation 15).

\[ Accuracy = \frac{\sum \text{argmax}(\vec{x}) = \text{argmax}(\vec{t})}{\sum \text{All}} \] (15)

To increase the accuracy, the neural network iterates over a process, as shown in figure 27. The process of passing the values of neurons to neurons of the next layer after applying the perceptron and activation function called Pass Forward. After each pass, the loss function generates a value named Network Error. The neural network uses an optimizer function such as "Gradient Descent" that generates a matrix off values using a mathematical equation to minimize the network error. Each element of this matrix is associated with one variable of the network (a weight or a bias) that neural network applies these values to the associated variable in the Back Propagation process. After updating the variables, the next pass forward process starts until achieving satisfactory accuracy. Figure 28 shows the steps of gradient descent in which \(\alpha\) represents the learning rate or step size, and \(\theta\) shows the value of gradient in each step. Eventually, after 4 steps, the gradient descent reaches to the minimum point.


4 Implementation

In this section, the general workflow of the model briefly, and then each step has been explained thoroughly. The model discussed in this study, comprised of a Self-Organizing Map as a hidden layer, proceed by a deep neural network and a small short-term memory, which supposed to accelerate incremental learning.

The data is fed to the model in two to five optional non-overlapping subsets of MNIST classes. For example, the first sub-set can be 1,2,3,4,5,6,7 classes, and then the second sub-set would be 8,9,0 classes. The sub-sets named as $D_1$ to $D_5$. As mentioned in section 2.2, this study aims to use this model and evaluate its performance in class-incremental learning on the MNIST data set. Therefore, using repetitive classes in these sub-sets would not be meaningful for this study. However, by repeating all classes in $D_1$ to $D_5$, we can run a non-incremental learning test, and use it in comparing the performance of incremental trials.

The model training involves data streaming, as shown in figure 29, that repeats the same methods for each sub-task. The model generates an untrained SOM, with random values as wights, a DNN with one hidden layer, and also a fixed-size short-term
memory is constructed immediately on the instantiating phase.

The short-term memory supposed to save a few correct classified samples and their associated labels. The model retrieves the Euclidean distances between each sample and all weights and uses them as training and evaluation data for the neural network after passing them through a Gaussian transfer function. The training data encompasses the specified classes in each sub-task, but the evaluation data-set includes all classes each time to evaluate the model. The neural network training can perform with variable
4 Implementation

batch size and number of iterations for each batch in a specific sub-task. The model trains and evaluated the data as following steps:

1. Generate the Short-Term Memory (STM) samples and labels if the STM is not empty, and injecting the generated samples to training data. In the first sub-task, this step would be skipped since the STM is empty.

2. DNN evaluation and capturing the wrong classifications. The decision making about classification accuracy lies on the $ce$ parameter with a default value of 0.2. Therefore, if classification Loss is greater than $ce$ for each sample, the classification would be assumed as wrong.

3. Training the SOM with wrongly classified samples. Obviously, at the first sub-task, all classifications would be wrong, but in the next sub-tasks, the SOM training would be performed on the only newly added classes. By utilizing this approach, the SOM damage would be minimum during the addition of new classes in the next sub-tasks.

4. Training the DNN by the distances retrieved from the SOM.

5. Repeating number all the above steps for each batch until completing the training.

6. Filling the Short-Term Memory with correct classifications.

To perform a realistic incremental learning simulation in the model, as the algorithm displays, the model training is performed in a sequential pattern through all phases without using parallel processes. In the first sub-task, the short-term memory is not involved in the training since it is still empty, but after the first phase, the entire process would be continued after re-generating samples using the short-term memory and injecting them to the training data.

In contrast with the SOM, which uses an exponential decay learning-rate, the learning-rate on the DNN in this model, is constant. The optimizer is SGD (Stochastic Gradient Descent), and the learning rate is not adjusted to a very small value. Hence, when the DNN start learning, in the first steps, overwrites the weights with new data, which leads to forgetting the previous distances retrieved from the SOM. Therefore, by mimicking the prior learnings with re-generating them (which is not memory or CPU baud job), this overwriting can be prevented. As mentioned above, the model uses multiple options as parameters (explained in section 5) for each part of the algorithm.

4.1 Deep Neural Network

As discussed above, the predicting is performed by a neural network, or better saying a simple Linear Regression (LR) that reads the distances from the SOM. This DNN
Algorithm 1: The training workflow

\begin{itemize}
  \item **Data:** MNIST classes in $D_1$ to $D_5$ sub-tasks
  \item **Result:** Incrementally training $D_1$ to $D_5$ sequentially.
  \item Retrieving the class list for $D_1$ to $D_5$ from MNIST data-set and;
  \item Randomly selecting N sample and associating label per each class;
  \item Instantiating the SOM with given parameters;
  \item Instantiating the DNN with given parameters;
  \item Instantiating the STM;

\begin{algorithmic}
  \For{$n$ in sub-task 1 to 5}
    \If{STM is not empty}
      \State Re-generate the prior samples and concatenate to the samples.
    \EndIf
    \State Generating batches.
    \For{batch in batches}
      \State Capture the wrong classifications.
      \If{wrong classifications list is empty}
        \State continue
      \EndIf
      \State Training the SOM with wrongly classified samples.
      \State Training the neural network with the result of distances.
    \EndFor
    \State Fill the STM nodes with correct classifications.
    \State Evaluation.
  \EndFor
\end{algorithmic}

\end{itemize}

uses a hidden layer with size of $2 \times Z_{input}$ in which $Z_{input} = SOM_x \times SOM_y$. The number of iterations in the neural network is adjustable as well to maneuver over the training parameters for investigating the performance with different parameters. In contrast to the SOM layer, the learning rate in the neural network is always constant. The training loop reads the Euclidean distance between each sample ($\vec{x}_{sample}$) and all weight of the SOM ($\vec{w}_{SOM}$) and uses these distances as the input layer of the neural network. Also in order to normalize and standardizing the distances, a transfer function applies equations 17 and 18 on the distances, before passing them to the DNN.

\begin{align*}
  Z^{SOM} &= TF(N(||\vec{x}_{sample} - \vec{w}_{SOM}||)) \\
  N(x) &= x/Ma_{x_d} \\
  TF(x) &= \frac{\vec{x} - \mu}{\sigma}
\end{align*}

Equations 16-18
4 Implementation

In equation 16, the $Z^{SOM}$ is the distances retrieved from the SOM layer. As mentioned in previous pages, the Euclidean Distance has been used for finding the geometric distance between each input vector, and all weights of the SOM. To reduce the DNN confusion, especially between similar samples, the DNN input layer should be normalized. In order to normalize the data, the equation 17 has been used, where the $Max_d$ is the highest distance retrieved so far.

The Equation 18 shows a Gaussian function used in order to standardize the DNN inputs and accelerate the training in which $\mu$ is the mean of data, and $\sigma$ is the standard deviation.

$$P = LR(Z^{SOM})$$

(19)

The activation function between layers is Relu (figure 11), except last layer which uses Softmax (equation 14) as its activation function. The loss function used in the DNN is Categorical Cross-Entropy (CE), in which the values less than the CE threshold is considered as wrong classifications.

$$CE = -\sum_{x=1}^{10} P_x \log(t_x)$$

(20)

Assuming the Softmax outputs as $P$ (predictions), the SOM trains only wrong classifications calculated by equation 20 where the $t$ shows the one-hot target vector of each sample. Since all the code is written in Tensorflow 2, and all SOM and DNN functions are in vector mode, training a large batch of data might lead to OOM error (Out Of Memory), especially in case of using mid-class laptops. In such circumstances, the options for limiting the training task and the size of data are useful. (section 5).

4.2 Short-Term Memory

A simple queue data structure has been used as a short-term memory to store the most accurate final classifications of each sub-task and their associated labels. With pushing new data to the STM, the previously saved ones kicked out from the front of the queue (shown in figure 32). The idea of using short-term memory in this study is to maintain the very exact classification and replaying them in the next sub-tasks to prevent degradation as much as possible.

To achieve the best accuracy values and filling the STM, the memory would be filled by N random samples from a list of classification with accuracy more than $\theta$, resulted from an evaluation step. This evaluation step occurs at the end of each sub-task involving all samples and labels of the task. Evidently, the variety of samples in the STM can
be increased by hiring a larger STM. However, as mentioned several times in this article, the limitation of resources do not allow us to consider just the quality.

![Image of the Short-Term memory](image)

**Figure 30:** The Short-Term memory

Since in each sub-task, the model performs the training only on specified classes, the STM is prone to be filled only by one class, which results in the degradation of prior learned classes. A mechanism has been embedded in the STM that not allows it to be filled just by one class. Therefore, in the next sub-task, re-generated data would include all previously learned classes and avoid accuracy degradation.

## 5 Experiments

In this section, experiments on three different combinations of sub-tasks conducted to evaluate the model performance in class-incremental training. These experiments involve the training of sub-tasks in the loop and evaluating all prior and recent pieces of training using test data. The sub-tasks in this section are named:

- **D5-5:** 0 to 4 classes in first, and 5 to 9 classes fed in the second task.
- **D9-1:** 0 to 8 classes in first, and only class 9 fed in the second task.
- **D2-2-2-2-2:** Includes 5 sub-tasks, and each sub-task contains a sub-set of 2 class.

As mentioned in 4, multiple adjustable options in run-time are utilized to run the test flexibility and maneuver on parameters. Therefore, at first, I take a closer look at these options, and then start to report the test results for a variety of class-incremental training.

### 5.1 Global parameters

In order to develop a flexible model, multiple parameters have been used in both the Self-Organizing Map and Deep Neural Network and in training loop itself. Before reporting the experiment results in section 5 here we explain the parameters.
5 Experiments

- **X and Y**: These options used for the dimensions of the SOM. X for the number of weight nodes in width, and Y for weight nodes in height.

- **$D_1$ to $D_5$**: As explained at this section, the class-specific samples in the range of $D_1$ to $D_5$ sub-sets are fed to a training loop to implement class-incremental learning. The classes for each sub-set can be determined by $D_1$, and $D_5$ parameters.

- **Limit**: By selecting the classes for each $D_1$ and $D_2$ sub-tasks, the entire samples of the selected classes considered for training, which is a large number. The number of samples per class can be limited to avoid OOM error, as explained in section 4.

- **Radius**: Determine the initial radius of the SOM, for $D_1$ to $D_5$ training phases. The default value (for default $10 \times 10$ SOM) is set to 3.5 for first, and 0.5 for all other sub-tasks.

- **Learning Rate**: Determine the initial learning-rate of the SOM, for $D_1$ to $D_5$ training phases. The default value is 0.5 for the first, and 0.005 for all other sub-tasks.

- **CE**: Is the Cross-Entropy threshold for extracting the wrong classifications from the DNN in each training loop, as demonstrated in equation 20. The default value of 0.2 assigned to the $ce$.

- **DNN Iterations**: The number of DNN iteration in each batch, for each sub-task. The default values are 1 for all sub-tasks. However, it is adjustable using this parameter.

- **Epoch**: The number of attempts to run a sub-task and go through all samples.

- **Short-Term Memory**: The $stm$ parameter determines the size of short-term memory. The default value is set to 100, meaning that the STM would not store more than 100 samples.

- **Batch**: This option is used for splitting the data-set to batches for both sub-tasks. For example, if we have 100 samples for each class in a sub-task of 5 classes, we would have 500 samples. Option $batch = 5$ splits the data-set of 500 samples to 5 batches with a size of 100 and runs the training loop on these batches sequentially. The schematic view of the batch process shown in figure 31.

All tests are conducted using a $10 \times 10$ SOM, with a limit of 1000 samples per class. Accordingly, the model has tested on a total of 1000 MNIST images. The Cross-Entropy threshold is 0.2. The DNN iteration for each batch is adjusted to 100 (same size a batch), and radius and learning-rate are mentioned in table 1 for each experiment. The sub-sets are non-overlapped (even one sample from other classes is not involved in the associated training phase).
After training loops for sub-tasks, the model is supposed to remember all classes of previous sub-tasks. Therefore, the incrementally trained model feeds the same classes of each sub-task from $D_1$ to $D_5$, this time using **MNIST Test Data**, to monitor the degradation rates. Consider that in the first stage of experiences, the $Radius$ and $LR$ remain as default.

![Diagram of feeding the training loop by batches](image)

**Figure 31: Feeding the training loop by batches**

<table>
<thead>
<tr>
<th>Task</th>
<th>Radius$_1$</th>
<th>Radius$_2$ to Radius$_5$</th>
<th>LR$_1$</th>
<th>LR$_2$ to LR$_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5-5</td>
<td>3.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.005</td>
</tr>
<tr>
<td>D9-1</td>
<td>3.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.005</td>
</tr>
<tr>
<td>D2-2-2-2-2</td>
<td>3.5</td>
<td>0.75</td>
<td>0.5</td>
<td>0.005</td>
</tr>
</tbody>
</table>
5 Experiments

5.2 D5-5

The aim of this experiment is to monitor the model behavior in incremental learning with large data sets. Only the half of classes have been fed into the model in $D_1$, and the other half in $D_2$. As mentioned in section 3.1.2, the radius, and learning-rate of the SOM would be decreases by time exponentially.

![Figure 32: The exponential decay of the SOM radius.](image)

Since in the $D_1$, the SOM still is untrained, there is nothing to protect, and the SOM training can be started with a high radius and learning rate. Consequently, the SOM can achieve a sufficient convergence, in few steps, and updating it continuously in unnecessary. Contrarily, in the $D_2$, and other sub-tasks, the prior prototypes in the SOM should not be overwritten by new classes. Therefore, the initial radius should be lower, and the SOM should have more time to adapt itself by assembling the new classes in suitable coordination.

By analyzing the confusion matrices in each sub-task, a clear vision of learning status can be achieved. As mentioned previously, the model is trained with 1000 samples per class limit, and the test data includes 1000 randomly selected samples from the MNIST test data.
5 Experiments

Figure 33: D5-5 Confusion matrix after $D_1$.
The predictions shown as the rows, and the real target values depicted as columns.

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>99</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>19</td>
<td>16</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>18</td>
<td>0</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>94</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>40</td>
<td>28</td>
<td>38</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>90</td>
<td>1</td>
<td>38</td>
<td>7</td>
<td>22</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>93</td>
<td>7</td>
<td>30</td>
<td>37</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Figure 34: D5-5 Confusion matrix after $D_2$.
The predictions shown as the rows, and the real target values depicted as columns.

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>93</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>105</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>86</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>78</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>85</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>95</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>84</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
```

Figure 33 demonstrates the confusion matrix of the model, after completing $D_1$. Evidently, there is no prediction of classes 5 to 9, since the model has not been fed by them
yet. And since, this confusion matrix is generated by test images (including all samples) a huge confusion between 4 and 9, and also between 3 and 8 classes is apparent. Figure 34 depicts the same model after training in $D - 2$.

Figure 34 confirms that the model has learned the new classes (5 to 9) in $D_2$, but with a slight degradation in 0 to 4 classes which are expected. The final evaluation of the D5-5 experiment has been depicted in figure 35.

![Figure 35: D5-5 evaluation.]

### 5.3 D9-1

In this experiment, the assumption is that a trained model tries to learn only one new class incrementally. Therefore, the model is trained by classes 0 to 8 in $D_1$, and in $D_2$ only class 9 has been fed into the model to monitor its performance in 1-incremental learning. The confusion matrices of this task have been shown in figures 36 and 37 after accomplishing $D_1$ and $D_2$ respectively. These matrices follow the same pattern as figures 33 and 34 but with 0 prediction for class 9 in $D_1$.

The significance of controlling the STM storage in this experiment, is more evident. Since in this experiment, the STM contains samples of 0-8 classes, and without the mechanism mentioned in 4.2 the STM would completely fill by class 9 samples. The final evaluation of the D9-1 experiment has been depicted in figure 38.
5 Experiments

Figure 36: D9-1 Confusion matrix after $D_1$.
The predictions shown as the rows, and the real target values depicted as columns.

Figure 37: D9-1 Confusion matrix after $D_2$.
The predictions shown as the rows, and the real target values depicted as columns.
5 Experiments

5.4 D2-2-2-2-2

In this experiment, the model has been tested by five consecutive tasks with a small portion of classes in each sub-task. Each sub-task contains only two classes and the model is supposed to learn these classes sequentially while remembering the prior learnings. Figure 39 shows the decay pattern of radius and learning rate in $D_1$ to $D_5$ in order to achieve sufficient convergence in the SOM. The initial radius of all sub-tasks has been adjusted to 0.75 to provide a bit wider space than D9-1 for a new prototype for assembling. However, the learning rate should be adjusted to be a small number in order to minimize the damage of SOM. Figure 40 shows the incremental trend of learning by evaluations conducted after completing each sub-task. The stepped increasing trend in each sub-task is expected since the model feed 20% of the data in each task. Since the experiment comprised of five steps, analyzing the confusion matrix would not be easy. Therefore, figure 41 is provided to detect the learning degradation per class.

This diagram shows the learning process of classes to monitor the their addition during the training. By analyzing this diagram, the incremental addition of new classes and a slight degradation of prior learnings is evident, but there is no sign of completely forgetting a class.
5 Experiments

Figure 39: D2-2-2-2-2 Decay.

Figure 40: D2-2-2-2-2 evaluation.
5 Experiments

Figure 41: D2-2-2-2-2 Correct classifications in batches. The vertical axis shows the correct classifications of 1000 test samples, and the horizontal axis, indicates the total batch numbers. ($D_1$ contains 2000 samples and with a batch size of 100, the $D_1$ would execute over 20 batches. After replaying the 20% of data from STM in $D_2$, the total sample numbers would be 4000, and consequently, the sub-task would include 40 batches. This pattern repeated for all sub-tasks, and totally the entire task would contain 300 batches.)

5.5 SOM update during training

Let take a more profound look at the Self-Organizing Map or hidden layer of this model during the training phases. Figure 42 depicts a $15 \times 15$ SOM which trained by D9-1 sub-tasks. The above image presents the SOM after completing the $D_1$ phase, and the bottom picture shows it after accomplishing the $D_2$ sub-task. The centers of these images have been magnified to facilitate the distinguishing of the changes in these two phases.

The left image depicts the original weights, and as discussed in section 3.1.2 during the training, the weight in the BMU location and its neighbors are rewarded by homogenizing to the winner node. The center images demonstrate the mapped winner samples to each weight, and the right-hand image shows the mapped labels to the same weights. The difference between the two weight matrices is very marginal and intuitively indiscernible. Though, on the mapping of samples, we can see a few 4 and 9 replacements. This difference is apparent in label mapping as well. Since the sub-sets are non-overlapping, the class-9 is not available in the $D_1$ training phase, and by feeding the class-9 to the trained SOM, it has been assembled in the appropriate location with
5 Experiments

Figure 42: SOM updates during the training phases

similar neighbors. This mechanism of Self-Organizing Maps can solidify the previous learning states to overcome catastrophic forgetting.

5.6 Continuous learning

All the above experiments have been performed with a single epoch per sub-task. However, in order to diagnose the model performance in continuous learning, another experiment has been performed with one epoch for $D_1$ and 20 epochs for $D_2$ in the D5-5 test. Usually, the model is prone to downgrading and in the worse case to catastrophic forgetting in continuous learning. However, due to the mechanism that has been embedded as safe-guarding, this model would be immune to such degradation.

As explained in algorithm 1, both SOM would train just wrongly classification, and
DNN would train the data immediately after SOM training to adapt the DNN with new prototypes. Therefore, the training would stop if there is no wrong classification. This mechanism showed a huge effect on preventing catastrophic forgetting or any drastic degradation, during the experiments.

Figure 43 depicts the performance of D5-5 experiment with 1 epoch for $D_1$ and 20 epoch for $D_2$. To demonstrate the model performance consistently during the entire task, the sub-tasks have not been separated. This figure is the plot of training batches during the $D_1$ and $D_2$ sequentially. Since the training has been conducted on a dataset of 1000 samples per class, the $D_1$ contains a total of 5000 samples. Consequently, with a batch size of 100, the $D_1$ has been finished in batch number 50. The $D_2$ sub-task has been started in batch 51 and continues till batch 2050 (20 batches).

![Figure 43: D5-5 Continuous learning performance.](image)

The vertical axis indicates the number of correct classification per class in a test dataset with a size of 1000. And the horizontal axis shows the total number of train batches.

Figure 43 does not show any drastic degradation for any class which means that
6 Discussion

The latest version of Tensorflow, Numpy, and other technologies have been used to benefit from the cutting-edge methods and their precision and performance during the code development of this thesis. As discussed in several sections of this report, the system resources always matter and practically is not possible to perform learning functions entirely by powerful computers any time and anywhere. Therefore, the experiments and development of this thesis implemented by a mid-class laptop with 8 Gigabytes RAM and NVIDIA Geforce 940MX graphic card, which is not considered a high-performance system nowadays.

You can find the full implementation in my personal Github repository: https://github.com/behzadtz86/dualmemory

Despite using low dimensions in SOM, and limiting the samples to 1000 per class in many experiments, the results of evaluations were quite acceptable. During the development, the test runs with only 100 samples per class showed acceptable outcomes.

Undoubtedly, using different reasonable parameters even in such limited learning phases, and small changes in SOM training algorithm or neural network layers would lead to obtaining better results. There are many options and techniques available for increasing the accuracy of DNN training, such as early-stopping or momentum in SGD. Using higher dimensions of SOM and consequently retrieving the distances in higher dimensions for use in the DNN, using hidden layers and even Dropout algorithm, would be beneficial.

In real-life projects, the value of these parameters depends on the system’s performance and availability of data, and many other factors. The optimal value selection for them itself is a challenging and spacious subject. In the experiments section (5), the result of the degradation rates for the $D_1$ after continual training the model with new data in $D_2$ to $D_5$ were expected. However, adjusting the optimal parameters for training and decision-making needs thorough mathematical calculations and thousands of trials. For MNIST data-set, the model shows acceptable performance in incremental learning and alleviating catastrophic forgetting. Considering the adjustable number in data re-generating and DNN epochs, the higher accuracy values are also attainable by this model.

Short-term memory has been embedded in the model for saving the accurate classified samples during the sub-task learning. Although the SOM stores the prior learnings, the final decision-making, and evaluation are always performed by the DNN. Here we
confront a double-edged dilemma. When the learning rate of the DNN is low, it cannot learn the new classes. On the other hand, the DNN weights would be overwritten by new values in high DNN learning rates. Therefore to have the benefit of a high learning rate in the DNN and learning the new classes sufficiently, using the STM and replaying the data is imperative.

Only new classes participate in SOM training. In case of involving the replied sample, the SOM convergence would not be sufficient. Besides, in experiments like D5-5 which feeds the model with a large number of classes in the second task, the structure of the SOM is prone to change completely. Saving the correctly classified samples in the STM and involving them only in DNN training, protects the DNN weights from overwriting by new classes. Especially in classifying similar images, replaying the previous correct classification, prevents the confusion.

Unconditional DNN training in each batch also can lead to the degradation of previously learned classes. The Self-Organizing Map can be considered the main part of this model since it stores all prior knowledge. The DNN is used only for predicting and the evaluation. Therefore, training the DNN in batches unless having any changes in the SOM would be a redundant and risky step. This mechanism can be considered as a safe-guard against drastic degradation in continuous learning.

Since, the model trains only the wrongly classified samples, the Cross-Entropy threshold (CE) option plays a vital role. This value is used for decision making about the correctness of a classification. Consequently, when a large value is assigned to the CE, the model would detect less wrongly classified samples, and accordingly the SOM and DNN would be trained rarely.

Figure 43 confirms the fact that the model alleviates catastrophic forgetting, and also prevents drastic accuracy degradation in prior learnings. With a closer look at the diagram, it is evident that the performance fluctuations show a reducing trend by progressing in the epochs, and the model works even more stable in continual learning.

7 Acknowledgement

Hereby I present my sincere gratitude to Prof. Dr. Alexander Gepperth, who instructed me from scratch during both the development and reporting of this thesis. His support, patience, and motivation paved the way for my progress and increased my enthusiasm for machine learning topics.
8 Conclusion

In this article, the necessity of continual learning in real-world artificial intelligence projects has been discussed by giving examples. Furthermore, the natural ability of humans in incremental learning briefly explained by taking a glance at the fascinating natural mechanism of a biological brain in memorizing and fine-tuning the prior knowledge while acquiring new knowledge and experiences. Moreover, few significant studies in this regard briefly explained. Indeed the role of mathematics and statistics in data sciences and machine learning is inevitable. As explained in section 6, trivial differences in parameters can lead to much better or reversely worse outcomes. Additionally in section 3 the fundamental concepts of machine learning and its taxonomy explained by giving a simple and understandable example.

In the implementation section, the main idea of the model, concepts, and calculations are explained as abstract as possible. Each weight node in the SOM layer is an array of data and can store N-dimensional data. Consequently, this model can be utilized for general-purpose incremental learning methods. However, as always, there is no flawless architecture, and definitely, this model can be optimized by spending more time and effort.

Possibly not only the outcomes but also the network performance and response time could have a rigorous improvement in the case of merging a mechanism to store just essential data to this model. Similar to the mechanism used in iCarl (section 2.3.6), or benefit from more intense statistical equations like Bayesian Neural Networks used in IMM (section 2.3.4).

The Self-Organizing Map, as a consolidator layer, plays the primary role in this model. Numerous implementations of Self-Organizing Maps are available on the internet, and the model can be improved by using better and more precise SOM implementations. A 2D SOM has been used in this thesis for creating the feature map of prior knowledge. Considering the color range of MNIST images (black and white) is sufficient for this project. Nevertheless, in the real world, the image overwhelmingly holds much larger dimensions and are colorful. Therefore, using a 3D SOM layer for saving the RGB color ranges and a convolutional neural network for down-sampling the high definition images also could apply to this model.

Despite numerous substantial studies, we as human kinds have a long way to go in simulating the human’s peculiar nature in the robots and artificial devices.
References


