# University of Padova


Department of Mathematics "Tullio Levi-Civita"


*Master Thesis in Data Science*

# Gated Linear Networks for Continual Learning in a Class-Incremental with Repetition Scenario


*Supervisor*
Prof. Lamberto Ballan
University of Padova

*Co-supervisor*
Dr. Nicolò Navarin
Elena Izzo
University of Padova

*Master Candidate*
Federico Medici

*Student ID*
2005801


*Academic Year*
2022-2023

"Qualunque cosa sia il destino, abita nelle montagne che abbiamo sopra la testa."
— Paolo Cognetti

# Abstract

Continual learning, which involves the incremental acquisition of knowledge over time, is a challenging problem in complex environments where the distribution of data may change over time. Despite the great results obtained by neural networks in solving a great variety of tasks they still struggle in showing the same strong performance in a continual learning environment, suffering from a problem known as catastrophic forgetting. This problem, that consists in a model's tendency to overwrite old knowledge when new one is presented, has been dealt with through a variety of strategies that adapt the models on different levels. Among those, in this work we will focus on Gated Linear Networks (GLNs), a type of architectures that rely on a gating mechanism to improve the storage and retrieval of information over time. This class of models has already been applied to continual learning with promising results, but always in extremely simplified frameworks. In this work we will try to define a more complex continual learning environment and to adapt GLNs to the increased challenges that this environment will present, evaluating their strengths and their limitations. In particular, we found that performing an encoding step can help making a complex dataset more spatially separable and therefore making the GLNs more effective, and that switching to a Class-Incremental with Repetition scenario is useful both to increase the realism of the framework while easing the learning difficulty.

# Contents

# Listing of figures

# Listing of tables

# Listing of acronyms

**CI** ..... Class-Incremental (scenario)

**CIR** ..... Class-Incremental with Repetition (scenario)

**CN-DPM** ..... Continual Neural Dirichlet Process Mixture

**CWR** ..... Copy-Weights with Re-init

**DGN** ..... Dendritic Gated Network

**DGR** ..... Generative Replay

**EWC** ..... Elastic Weight Consolidation

**GEM** ..... Gradient Episodic Memory

**GLN** ..... Gated Linear Network

**HCL** ..... Hybrid generative-discriminative approach to Continual Learning

**ILSVRC** ..... ImageNet Large Scale Visual Recognition Challenge

**LWTA** ..... Local Winner-take-All

**MAS** ..... Memory Aware Synapses

**NC** ..... New Classes (scenario)

**NI** ..... New Instances (scenario)

**NIC** ..... New Instances and Classes (scenario)

**PNN** ..... Progressive Neural Network

**SI** ..... Synaptic Intelligence

**STM** ..... Short-Term Memory

**XdG** ..... Context-dependent Gating

# 1 Introduction

Machine learning and deep learning have become increasingly popular in recent years, and this trend is set to continue in the coming years. The growing popularity of these fields can be attributed to several factors, including the availability of large amounts of data and computational resources, the development of more powerful and efficient algorithms, and the increasing demand for solutions to complex problems that traditional methods are unable to solve. These algorithms have been used to achieve state-of-the-art results in areas such as image recognition, natural language processing, and speech recognition, among others. However as machine learning and deep learning models become more complex and capable of solving increasingly challenging problems, the need to learn and adapt to new information becomes crucial.

Continual learning refers to the ability of machine learning models to incrementally and continuously learn new tasks while retaining their knowledge of previously learned tasks. This is in contrast to traditional machine learning, where models are typically trained on a fixed set of tasks and then tested on new unseen data. The field of continual learning is thus a vital component of deep learning, as it enables deep neural networks to be deployed in real-world applications, where they must continuously face changing situations and environments. However, the same models that are able to achieve impressive results in a traditional machine learning environment are not able to maintain the same performance in a continual learning framework, due to a phenomenon known as catastrophic forgetting.

Catastrophic forgetting refers to the tendency of deep neural networks to forget previously learned information when learning new information. This happens because while the model's parameters are updated to optimize performance on the new task, its lack of capacity to preserve conflicting or competing information leads to the loss of knowledge about the previous tasks. Catastrophic forgetting is a major obstacle to achieving successful continual learning, as it can lead to models losing important knowledge and becoming less effective over time. To address the problem of catastrophic forgetting, various approaches have been proposed in the literature, that

generally involve adapting the architecture of the models, intervening in how the weights are updated or replaying the information at risk of being forgotten. However, many of these approaches are limited by their complexity, scalability, and ability to generalize to new tasks. Among them, this study will focus on a specific class of models, the Gated Neural Networks, that adopt an architectural strategy to mitigate catastrophic forgetting, and it will investigate their use in the context of continual learning.

Gated linear networks are a type of neural network architecture that relies on a gating mechanism to control the flow of information in the network, allowing the model to selectively preserve and retrieve information over time. The gating mechanism is typically implemented by exploiting contextual information about the available data, with the assumption that the environment contains some knowledge about the task being solved. That information is used to select one of the multiple sets of weights associated with each neuron of the neural network, according to a predetermined partition of the context space. Their gating mechanism, combined with a local weight update strategy that does not rely on backpropagation, makes gated linear networks well-suited for being trained in an online environment and a good fit for continual learning. Gated linear networks have already been applied in continual learning for tasks that span from image recognition to multivariate regression to contextual bandits, and in this study, we will try to expand the potential of gated neural networks to solve the problem of catastrophic forgetting by providing insights into their effectiveness in a more complex environment.

In particular, the contribution of this study to the field of continual learning is twofold. The Continual Learning paradigm has been so far modeled and evaluated with easy datasets in simplified scenarios, that manage to grasp only partially the complexity represented by a real-world flow of information and at the same time give up on features that could help in tackling catastrophic forgetting. This work will first try to address this problem by moving to a more realistic and more challenging learning framework, both in terms of how data are presented to the model and the type of presented data. Furthermore, it will test the performance of Gated Linear Networks in this improved framework, pointing out the advantages and the limitations of this type of model and trying to identify strategies to deal with the increased complexity they will have to face.

This dissertation is organized as follows. Chapter 2 introduces the theoretical background necessary for the understanding of this work. After a general overview of the fields of Machine and Deep Learning, it will present in more detail the Continual Learning paradigm and how it can be declined, it will describe the problem of catastrophic forgetting and it will give a summary of the related works in the literature. Chapter 3 will describe Gated Neural Networks, starting from the mathematics that lies behind those models, explaining their main features and why they are a great fit for continual learning, and reporting their usage in prior works. Chapter 4 will first introduce the experimental framework and explain the novelty aspects of this study, and it will then summarize the obtained results. Chapter 5 will finally address the findings and the limitations of this work, it will discuss possible improvements and will suggest directions for future research.

# 2 Background

In this chapter, we will talk about the theoretical background that lies behind this work. After a general introduction to the Machine Learning field (Section 2.1), we will present the concept of Continual Learning together with the challenges it presents (2.2).

## 2.1 Machine and Deep Learning

Machine Learning is a term that generally refers to the branch of Artificial Intelligence that focuses on how a machine is able to learn from data and improve its performance over time, like what humans do during the course of their whole life. A classical definition by Thomas Mitchell [6] describes Machine Learning as follows:

*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured in P, improves with experience E.*

This definition allows us to identify all the main ingredients in machine learning: a task to be solved (classification, regression, etc.), a set of data related to that task, a model to be taught to solve that task, and some metrics to measure the performance of that model. Among various types of machine learning models and algorithms developed in past and recent years, that can span from simple statistical models like linear regression and classification to Support Vector Machines (SVMs) [7] to Decision Trees [8] and many more, Artificial Neural Networks (ANNs) have attracted much attention because of their capability to perform greatly in complex tasks. The branch of machine learning that focuses on neural networks is called Deep Learning, and while their main building blocks, like the perceptron [9], how to combine them [10] and how to train them through techniques such as backpropagation [11][12] have been known for decades it has been only in the past 10-15 years that the necessary computational power and large-scale datasets have been made available in order to efficiently train such complex

models, paving the way for the so-called "deep learning revolution". Taking inspiration from the structure of the biological brain, neural networks generally rely on several layers of artificial neurons connected together in structures that can be very simple as much as incredibly elaborate in order to grasp all the complex patterns present in the available data. Since their introduction neural networks have been able to match or even surpass human performance on a wide range of tasks that include image recognition [13], object detection [14], natural language processing [15], language translation [16], speech-to-text conversion [17], face identification [18] among the best-known ones.

In addition to different types of tasks and models, another important aspect in defining how a machine "learns" is the learning paradigm, which depends on the type of available data and the way they are presented to the model. The most important learning paradigms are Supervised Learning (when the target is available and can be used to give feedback to the model), Unsupervised Learning (no target is available, and the goal is usually to find patterns or approximate distributions), Reinforcement Learning (the model is asked to perform actions in a sequential way receiving feedback in the form of a positive/negative reward), Online Learning (the model receives inputs as a continuous flow of information to be analyzed one by one). This work will focus on a particular paradigm, Continual Learning, that has recently gained a lot of attention because of its increased closeness to real-world scenarios and to human learning patterns and at the same time because of the greater challenge it presents to standard neural networks, and that will be presented in more detail in the next section.

## 2.2 Continual Learning

Continual Learning (also known as Incremental Learning or Life-Long Learning) is a learning paradigm that aims to bring the classical machine learning framework closer to human-like learning, and it is often seen as a necessary step to lead AI systems into achieving "true" intelligence. This paradigm takes inspiration from the fact that for humans it is natural to learn new skills sequentially without forgetting previously learned ones, and that the world around us is in general non-stationary. These are features that in real-world learning settings often translate into a sequential stream of tasks without any knowledge about their relationship or duration in time and with no chances of re-encountering previously seen data. These situations arise in multiple fields, from robotics [19] to reinforcement learning [20], computer vision [21], and many more.

A one-sentence definition of this paradigm, obtained in [22] by adapting Mitchell's definition of Machine Learning, states the following:

*A computer program is said to learn continually from experience if, given a sequence of ephemeral partial experience $E_i$, a target function $h^*$ and performance measure $P$, its performance in approximating $h^*$ as measured by $P$ improves with the number of processed partial experience $E_i$.*

This definition expands the original ones by adding the adjectives *ephimeral* and *partial* to the experience concept: this is a fundamental aspect in the continual learning paradigm, with models that are often trained in an online fashion and with a data distribution that changes over time and does not allow to make permanent assumptions over previously seen data. All these aspects of continual learning can be formalized and declined in multiple

ways, leading to several possible learning scenarios that will be presented in Section 2.2.1. This framework in general is quite different from the standard machine learning framework, that normally deals with stationary tasks and data distributions, and with the assumption that all the data are available during the training phase. That means that conventional neural networks trying to solve problems in a continual learning framework struggle with retaining the previously acquired knowledge and end up overwriting the parameters critical for solving previously learned tasks, a problem commonly referred to as *catastrophic forgetting* [23][24]. The root of this problem has been found in the stability-plasticity dilemma [25][26], a constraint that applies to both artificial and biological neural systems. It is based on the idea that a learning process required both plasticity to integrate new knowledge and stability to retain the old one, and a balance between the two must be obtained. Overcoming catastrophic forgetting means, on one hand, being able to efficiently acquire new knowledge, and on the other hand, preventing this new knowledge from interfering with the existing one. To achieve this a lot of computational strategies have been developed, which will be summarized in Section 2.2.2.

### 2.2.1 Continual Learning scenarios

There are two main characteristics of the learning framework that could be exploited to differentiate the possible continual learning scenarios: the knowledge of the task, and the type of content update [3][22].

The first distinction regards whether the model is aware or not of the task to be solved. The majority of the models assume to know that task, often in the form of a label associated with the input data. In this scenario, called *task-aware*, models can exploit that information to tackle more efficiently the problem of catastrophic forgetting, for example by taking counter-measures whenever a task switch takes place. The alternative is to train a model in a *task-agnostic* (or *task-free*) scenario, where no information about the task is provided. This scenario is more realistic since in real-world cases it often happens that tasks are not clearly defined or change over time, but it also results in greater challenges for the models that have less information to rely on. We will see later on when the main continual learning strategies will be presented, that being in a task-aware or task-agnostic scenario can play a big role in how the forgetting is dealt with.

The second distinction regards the possible kind of data included in each training batch, with three possible cases:

- **New Instances (NI)**: every batch contains new examples of previously encountered classes;
- **New Classes (NC)**: every batch contains only examples of classes never encountered before;
- **New Instances and Classes (NIC)**: a hybrid version of the first two and most realistic case, every batch containing both examples of new classes and new instances of old classes.

In the context of continual learning the second scenario, also called *Class-Incremental* (CI), is the most studied. However, this scenario can be limiting, since it artificially exacerbates the catastrophic forgetting by never allowing the model to revisit previously seen classes. In real-world though it is natural to encounter repeatedly items from the same class without it abruptly disappearing, a behavior that implicitly contributes to mitigating the forgetting of previous knowledge [4]. The third scenario sometimes referred to as *Class-Incremental With Repetition* (CIR), tries to address this limitation, and has been proven to be beneficial for the performance on both learned and novel classes [27][28][29]. For this reason, this work will focus on a NIC scenario, using a task-building protocol proposed in [4].

### 2.2.2 Continual Learning strategies

The relative novelty of this field had led research to move in various directions, sometimes very different from each other, in order to address catastrophic forgetting. In the literature the approaches have been commonly grouped into three main categories [30][31][22][32]:

- **Architectural strategies**: they try to mitigate the effect of catastrophic forgetting by altering the architecture of the model, and therefore by choosing specific architectures, activation functions, or weight-freezing strategies;
- **Regularization strategies**: in this case, the adopted strategy involves adding constraints to the model's weight updates by intervening on the loss function with some regularization techniques;
- **Rehearsal strategies**: involves periodically replaying data from earlier training sessions in order to strengthen the connections relative to previously learned knowledge.

We will now introduce some of those strategies more in detail with a reference to the categories introduced above, keeping in mind that not all of them are fit for both the task-aware and task-agnostic scenarios and that the classes of strategies are not necessarily mutually exclusive.

#### Architectural strategies

Models belonging to this class take inspiration from the internal structure of the brain and the presence of functional areas within it. According to the Complementary Learning Systems theory[33] there are two areas in the brain responsible for the learning process: short-term memories are stored in the hippocampus, while a long-term consolidation of those memories takes place in the prefrontal cortex, allowing the brain to retain the most useful information and discard the rest. This led to developing strategies that focus on having multiple representations inside the model, with different sets of weights getting updated under different rules.

The most naive approach consists in freezing a subset of deemed important weights while leaving the others subject to updates [34], a procedure that can be relaxed by introducing different learning rates for different sets of weights according to how specialized they are to previous tasks leading to an intersection with the regularization strategies. A dual-memory approach has been used also in the GeppNet+STM model [35], where a standard architecture is joined by a short-term memory (STM) buffer that is used to store inputs that produce highly uncertain predictions. Those memories are then consolidated into the main network during a designated "sleep" phase, similarly to how memory works in humans. The Copy-Weights with Re-init (CWR) approach [36] is still based on a dual representation but targets only the neurons of the last layer of a network, being the most specialized ones. For each of them, two sets of weights are defined, a temporary one that gets trained in a regular way only to be re-initialized whenever a shift in the data distribution takes place, and a consolidated one that gets updated only concurrently to those shifts on the basis of the temporary ones, while the rest of the network is left untouched.

Of course, more than two learning areas can be initialized in a model. In the Progressive Neural Networks (PNNs) introduced in [37] new sets of task-related neurons are initialized whenever a new task is encountered, and only those related to the current task are trained while the others are kept frozen. These models, that as the CWR strategy work in a task-aware scenario, ensure a great specialization of different areas of the network, and

are very efficient in preserving weights related to old tasks. On the other hand, these benefits come at the expense of a potential explosion of the architectural complexity, that in a framework with a potentially infinite training time could end up being a heavy limitation.

Another approach is the Context-dependent Gating (XdG) [38], which uses task labels as contextual information to activate or deactivate certain areas of the network and leading to each neuron being trained only on a subset of the tasks, mitigating the forgetting. The result is similar to the one obtained with the PNNs, with each task referring to a part of the architecture, and with a weaker specialization compensated by the lack of complexity explosion. The gating mechanism presented above can also be adapted to be used in a task-agnostic scenario. The Gated Linear Networks (GLNs) [39] present multiple weight vectors for each neuron, and the weight set to be activated is chosen through a gating mechanism based on some contextual information relying on the assumption that the context brings information on the task to solve. This class of models is the object of study of this work and will be described in more detail in Chapter 3. An interesting variant of the GLNs is represented by the Dendritic Gated Networks (DGN) [40]: in an attempt to propose a more biologically plausible process, the gating mechanism is applied to different branches that, like the dendrites in biological neurons, connect the artificial neurons to different subsets of neurons of the lower layer.

An alternative way to introduce specialization inside a neural network consists in using different activation functions. It has been found that different activation functions are preferable depending on the task and the relationship between tasks [41]. With this assumption, in [42] a local winner-takes-all (LWTA) approach is proposed, enhancing local competition between neighboring neurons while exploiting different nonlinearities. Layers in the models that adopt this strategy are composed not of individual neurons but of blocks containing several neurons, each with a different activation function. In each block, a winner is selected through a competition function, while the outputs of the losing neurons are discarded in a winner-takes-all fashion.

### Regularization strategies

In this case, catastrophic forgetting is mitigated by adding regularization terms to the loss function so that the newly acquired knowledge does not interfere with the old one. These approaches are generally based on the assumption that there are multiple sets of weights resulting in the same performance. Therefore, when a new task is introduced, a possible way to preserve previous knowledge is to select a new set of optimal weights that is the closest possible to the old one, taking inspiration from how synaptic consolidation works in biological brains [43]. The easiest way to achieve that is by casually differentiating the learning speed across different neurons, a naive strategy that has been studied in the past years without achieving extremely encouraging results [44][24].

A more targeted method consists in slowing down the learning for the parameters of a model according to how much they contributed to the previous tasks, something that can be measured in different ways leading to different strategies. The Elastic Weight Consolidation (EWC) [31] quantifies the importance of the weights with respect to previous tasks through the diagonal of the Fischer information matrix. In the Synaptic Intelligence (SI) approach [30] an estimate of the importance of each weight is obtained by considering their contribution to the change in the loss while learning the task. In the Memory Aware Synapses (MAS) [45] a similar gradient-based

approach is used, but in this case the importance of each parameter is given by an estimate of how sensitive the performance of the model is to a perturbation of that parameter. In all these cases the computed importance is then incorporated into the loss function as part of an extra term that determines the learning speed for each weight. All these three strategies work only in a task-aware scenario since they rely on a task-change signal to update the importance value associated with each parameter, but they can be extended to the task-agnostic scenario. For example, the Task-Free Continual Learning approach [19] adds a task-switching mechanism to the MAS algorithm to compensate for the absence of task labels. In particular, it bounds the update of the importance values to a quick growth in the loss, assuming that the model performs badly immediately after data from a new task are presented, and freezes the importance values whenever that happens.

A slightly different strategy is used in the Uncertainty-guided Continual learning with Bayesian Neural Networks (UCB) [46]. While the goal is still to adapt the learning speed of each weight, the focus is on their uncertainty rather than their importance. Adopting a Bayesian approach to neural networks, they have each parameter drawn from a normal distribution $N(\mu, \sigma^2)$, and during the learning phase the weights are not updated directly, but rather through their mean and variance. In this context, the variance associated with each weight is interpreted as a measure of the weight's uncertainty. Assuming that the most uncertain weights are the less specialized ones, and thus those more available for further training, their updating speed is determined by multiplying their learning rate by their uncertainty. Notably, this approach doesn't require a knowledge of the task being solved, and is therefore fit for both task-aware and task-agnostic scenarios.

The Learning without Forgetting approach [47] introduces an extra loss term called Knowledge Distillation loss. Knowledge distillation generally refers to the process of passing the knowledge of a large model to a smaller one, and the loss function introduced in [48] does that by encouraging the outputs of the two networks to be similar. In Learning without Forgetting the same loss function is used to minimize the variation in the outputs for old tasks when a new one is introduced. Some variants of this method can be found in the literature: the Active Long Term Memory (A-LTM) network [49] works in a nearly identical way, differing only in the weight decay regularization strategy, while Less-Forgetting Learning differs in using internal representations instead of the final output to control the updates of the weights [50].

### Rehearsal strategies

The rehearsal strategies try to prevent catastrophic forgetting by retraining the model using past information. Also in this case, the most naive approach of storing all the past samples and using them to retrain the model whenever a task shift takes place is extremely inefficient, both in terms of computational complexity and memory requirements. It is possible though to adopt strategies that store past knowledge in an efficient and less memory consuming way, making this approach a viable option.

The ExStream algorithm [51] uses a memory buffer to store only a fixed number of examples from each class that can be seen as prototypes. As long as there's space in the buffer new examples are simply added, while when the buffer gets full a compression process takes place to free some memory: the two closest prototypes are merged through an average weighted on the number of previous mergers of each prototype. The prototypes are then

re-presented to the model together with examples from new tasks.

A less memory consuming approach to present past information to the model is through generative models. These models instead of storing past examples learn the distribution of the previously presented data and generate new data from that distribution. For example, the Deep Generative Replay (DGR) strategy [52] uses a dual model architecture, with a deep generative model ("generator") and a task-solving model ("solver"). The generator produces data based on the distributions of the old tasks that are paired with labels from those tasks. The generated data are then presented to the solver, which works as a standard neural network, interleaved with examples from the new task to reduce the overall forgetting.

### Hybrid strategies

Strategies belonging to different categories can be combined in order to obtain more resistant models. It is the case for example of the AR1 approach [53], obtained by combining CWR [36], an architectural method, with SI [30], a regularization strategy. The weights of the last layer that were kept entirely frozen in CWR got instead a different updated speed each, according to their importance as computed in the SI approach.

While AR1 is a task-aware method, as much as the two methods that it combines, most of the hybrid strategies have been developed for task-agnostic scenarios. The Hybrid generative-discriminative approach to Continual Learning (HCL) [54] combines features from all three strategy groups. It is a method based on generative replay that maps tasks to distributions through generative flows, expanding the model whenever a new task is detected by looking at drifts in the data statistics and adopting a regularized loss to enforce small changes in old representations. Continual Neural Dirichlet Process Mixture (CN-DPM) [32] is also based on a combination of generative replay and model expansion where outliers from the existing distributions are detected in a Bayesian nonparametric framework. Outliers are believed to belong to a new task and subsequently stored in a buffer, and as soon as the buffer gets filled a new task-related part of the network is built using the outliers collected up to that moment, with the model then being trained on samples generated from the distributions. Continual Unsupervised representation Learning works in the same way, differing just in using a Variational Autoencoder for the generative part [55].

Dark Experience Replay [56] works with a combination of regularization and rehearsal techniques. Like Learning without Forgetting [47] a knowledge distillation loss is used but applied only to a subset of the samples stored in a memory buffer. Those samples are selected via reservoir sampling, in order to simulate a uniform sampling among all the past data without the need of storing them all. A knowledge distillation step is used also in Incremental Classifier and Representation Learning (ICARL) [57], but in this case a more elaborate strategy is used to select the samples to be stored. A similar combination is used in the Gradient Episodic Memory (GEM) model [58], which uses a memory buffer to store the last $n$ seen examples from each class and adds a regularization term to the loss based on the performance on the stored examples. However, while the knowledge distillation loss enforces stability on the performance for previous tasks, the loss term introduced with the GEM model allows that performance to grow, in order to enhance the backward transfer of knowledge.

A third class of hybrid methods involves a mix of architectural and rehearsal strategies. These methods gener-

ally rely on a dual-memory system, similarly to the GeppNet+STP model [35] described among the architectural strategies. However, in these cases the short-term memory has two roles: learning short-term conceptual representations for the prediction phase, and generating samples for consolidation into the long-term memory. This is the case for example of FearNet [59] and Growing Dual-Memory (GDM) [60].

# 3

# Gated Linear Networks

In this chapter we will describe in detail the Gated Linear Network model, focusing on the general functioning of the model and the results achieved so far in the literature.

## 3.1 GLN overview

Gated Linear Networks (GLNs) are a family of neural network models first introduced in [39] with some peculiar features that make them particularly fit for working in the continual learning environment. In standard neural networks the task of predicting the final target is left to the final layer of the network, with all the internal neurons and layers learning to detect specific features of the input. In GLNs the approach is different: each neuron is considered an expert that predicts the probability of the target, taking the predictions of the neurons of the previous layer as inputs. Moreover, the model is characterized by a gating mechanism through which each neuron chooses the set of weights to use (and update) according to the region of the space to which the input data belongs, with different neurons having different space partitions.

The first advantage of continual learning comes from the simplification of the learning phase with respect to standard backpropagation. In fact, the weights of each neuron are updated independently by associating a separate convex loss to each neuron, increasing significantly its computational efficiency. The gating mechanism is also very helpful in tackling the problem of catastrophic forgetting by defining different sets of weights to be updated separately; in doing so this model falls into the category of *architectural* strategies following the classification proposed in Chapter 2. Moreover, this comes with no loss in terms of approximation capacity. In fact, similarly to what the Universal Approximating Theorem proved for feed-forward neural networks [61], it has been proved that with sufficiently rich side information the GLNs have a large approximation capacity and that the capacity is effective in the sense a gradient descent algorithm will eventually find the approximation [39].

Another interesting characteristic of this model regards its easy interpretability. Most of the modern neural networks have been criticized for being "black boxes" for which an interpretation is very difficult to obtain, often requiring complex post-hoc analysis [62]. As for GLNs we will see that after the gating process the model collapses into a linear model, specifically into a multilinear polynomial of degree equal to the number of layers. The weight vector associated to the polynomial has the same size of the input, meaning that it can be used to build an intuitive saliency map without any further computational expense. Figure 3.1 shows an example of saliency maps obtained from a GLN used in a classification task with the MNIST dataset.

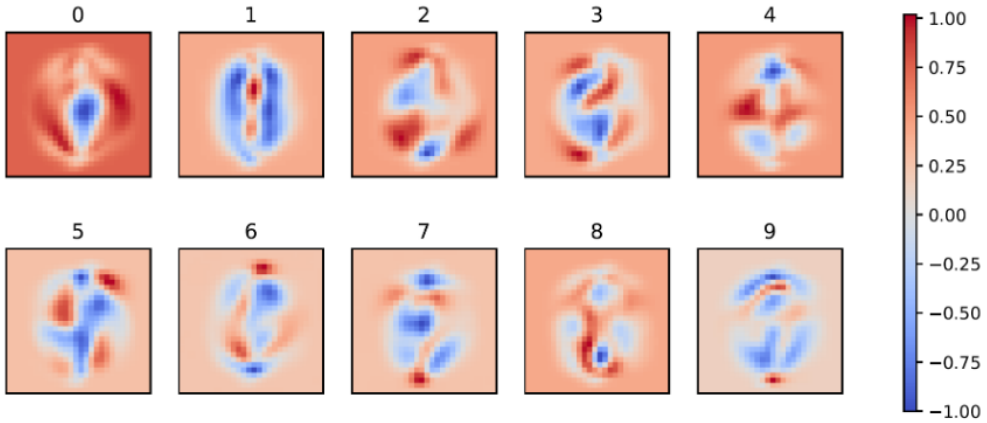



**Figure 3.1:** Saliency map obtained from a GLN used for a binary one vs all classification task on the MNIST dataset after 1 training epoch [1].

Due to their particularly fast training procedure, GLNs have seen their best field of application in online learning, with tasks spanning from contextual bandits [2] to univariate and multivariate regression[63] to image classification [1]. The performance of GLNs in a continual learning setting in countering catastrophic forgetting has been explored only in [3] using the Split-MNIST and Permuted-MNIST benchmarks, leaving room for an evaluation of its efficiency on more complex datasets and tasks, that is the aim of this work.

We will now introduce in more detail the two main building blocks of the Gated Neural Networks, the Geometric Mixing operation and the gating mechanism, explaining how to combine them and how to train a model built on those concepts.

### 3.1.1 Geometric mixing

First introduced by Christopher Mettern in the context of data compression [64][65], Geometric Mixing consists in a parametrized way of combining probabilistic forecasts. Given $p_1, p_2, ..., p_d$ input probabilities predicting a binary event, the geometric mixing of such probabilities takes the form

$$GEO_w(p) = \sigma(w^T \sigma^{-1}(p)) \tag{3.1}$$

where $\sigma(x) := 1/(1+e^{-x})$ defines the sigmoid function, $\sigma^{-1}(x) := \log(x/(1-x))$ defines its inverse, the logit function, $p = (p_1, ..., p_d)$ is the vector containing the input probabilities and $w \in \mathbb{R}^d$ is the weight vector. As shown in [1] the following identity holds:

$$\sigma(w^T \sigma^{-1}(p)) = \frac{\prod_{i=1}^{d} p_i^{w^i}}{\prod_{i=1}^{d} p_i^{w^i} + \prod_{i=1}^{d}(1-p_i^{w^i})} \tag{3.2}$$

that makes it explicit that the geometric mixing operation works as a product of experts, and leads to the following interesting properties:

- if all the weights $w_i$ are the same the geometric mixing works as a simple mean;
- if all the weights $w_i$ are equal to 0 then the prediction is $1/2$;
- every forecaster has a "right of veto", meaning that a single $p_i$ equal to zero, combined with a non-zero weight, drives the whole geometric mixture prediction to zero.

A neuron that carries out the operation defined in 3.1 is called a Geometric Mixer and both its inputs and its output take values in $[0, 1]$. As a result, an L-layers network of geometric mixers can be defined as:

$$\begin{cases} h^{(0)} = \sigma(x) \\ h^{(i)} = \sigma(W^{(i)} \sigma^{-1}(h^{(i-1)})), \ i = 1, ..L \end{cases} \tag{3.3}$$

where $x$ is the input data of the network, $W^{(i)}$ is the matricial parametrization of the weights of all the neurons of the i-th layer and $h^{(i)}$ is the output of the i-th layer.

It is worth noting that a deep network composed solely of geometric mixing neurons as described before is equivalent to a linear network with a final sigmoid function:

$$\begin{aligned} h^{(L)} =& \sigma\left(W^{(L)} \sigma^{-1}\left(\sigma ... \left(W^{(1)} \sigma^{-1}\left(\sigma(x)\right)\right)\right)\right) && (3.4) \\ =& \sigma\left(W^{(L)} \cdot ... \cdot W^{(1)} x\right) && (3.5) \\ =& \sigma\left(\prod_{i=1}^{L} W^{(i)} x\right) && (3.6) \end{aligned}$$

Given this last equivalence, in order to account for the nonlinearity that usually emerges in non-trivial data it becomes necessary to add a gating mechanism to the geometric mixers.

### 3.1.2 Gated geometric mixing

To obtain the neuron of a GLN a contextual gating procedure is added to the geometric mixer as described in the previous section. This means that the contextual information associated with the input data is used to select the

set of weights to be used. This is performed by subdividing the context space into non-overlapping regions, with every region being associated to a different set of weights $W$.

More formally, a context function $c : Z \to C$ is associated to each neuron mapping the contextual information $z$ associated to the input $x$ to a specific set of weights $w_{c(z)}$, where Z is the set of possible side information $z$ and $C = \{0, ..., k-1\}$ is the context space consisting of $k$ regions. Using the notation of equation 3.1 we get that a Gated Geometric Mixer can be defined as

$$GEO^{c}_{w}(p, z) := GEO_{w_{c(z)}}(p, z) \tag{3.7}$$

The standard approach is to simply use the feature vector of each input as context information. As for how to determine the regions of the context space, we will now present two possible strategies: halfspace gating and prototype gating.

### Halfspace gating

This is the method proposed in the original paper [1]. The split of the input space into regions is obtained by combining the halfspaces defined by hyperplanes. Given a vector $z \in \mathbb{R}^d$ and an hyperplane with parameters $a_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$ we can define a context function $\tilde{c_i}$ as

$$\tilde{c_i}(x) = \begin{cases} 1 & \text{if } a_i^T z > b_i \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

What we obtain is that the affine hyperplane $\{x \in \mathbb{R}^d : x \cdot a_i = b_i\}$ is being used to divide the context space into two non-overlapping regions. Assuming the number of regions $k$ to be a power of 2, we can stack $log_2(k)$ context functions of the same kind obtaining a higher-order context function $\tilde{c} : \mathbb{R}^d \to \{0,1\}^{log_2(k)}$, where $\tilde{c} = [\tilde{c_1}, ..., \tilde{c_k}]^T$. On top of that, the set $0, 1^{log_2(k)}$ can be easily mapped into a one-hot encoding of the $k$ regions, and thus we obtain the final context function by combining all those operations: $c : \mathbb{R}^d \to \{0,1\}^k$.

The standard approach for the initialization of the hyperplanes is to sample them randomly from a normal distribution fixed beforehand. In particular, a hyperplane defined through the parameters $a, b$ as described above has those two parameters sampled respectively from

$$a \sim N(0, I)$$
$$b \sim N(0, \sigma^2)$$

with the first being a multivariate normal distribution with the zero vector as mean vector and the identity matrix $I$ as covariance matrix, and the second being a univariate normal distribution with 0 as mean and a small fixed value $\sigma^2$ as variance. This choice has two main properties: with large $d$ such sampled hyperplanes have a high chance of being orthogonal, maximizing the efficiency of the split; secondly, inputs close in cosine distance will be mapped into similar predictions [1]. The choice of a small $\sigma^2$ is driven by the fact that it forces all the hyperplanes

to pass close to the origin; this, combined with a standardization of the input data, will ensure an optimal choice of the hyperplanes in the sense that all of them will cross the data splitting them into two groups. Figure 3.2 shows the result of half-space gating in a two-dimensional context space.

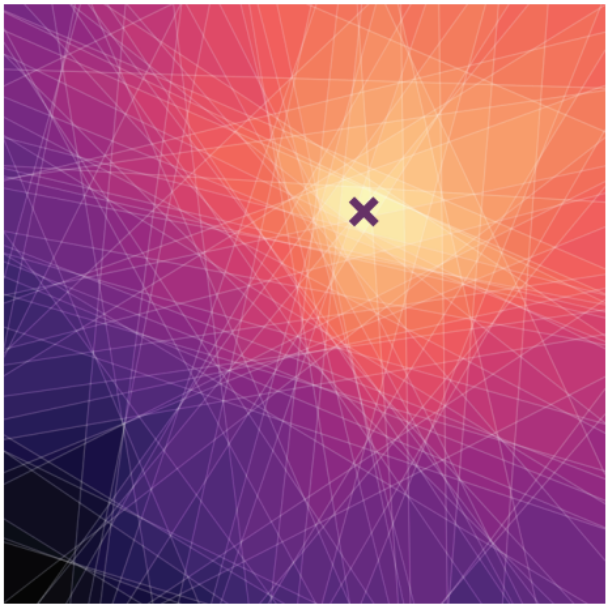


**Figure 3.2:** Illustration of half-space gating for 2D context space. Each white line corresponds to a hyperplane, and each region of the space is determined by its relative position with respect to all the hyperplanes [2].

### Prototype gating

In [3] an alternative gating strategy is proposed, that focuses on prototypes rather than hyperplanes. The region assignment is done by associating every point to its closest prototype, with the result being a Voronoi tessellation of the context space. If we call $P_j^{(i)} \in \mathbb{R}^{k \times d}$ the matrix of the prototypes associated with the j-th neuron of the i-th layer, the gating function can be formalized as:

$$c_j^{(i)}(z) = \text{one hot}\left(\arg\min_l \left(\|p_{j,l}^{(i)} - z\|\right)\right) \tag{3.9}$$

where $z$ is the context vector, $p_{j,l}^{(i)}$ is the l-th row of $P_j^{(i)}$ (corresponding to the parametrization of the l-th prototype of the j-th neuron of the i-th layer) and $\|\cdot\|$ is the Euclidean norm. Figure 3.3 shows how prototype gating differs from half-space gating in splitting a two-dimensional context space.

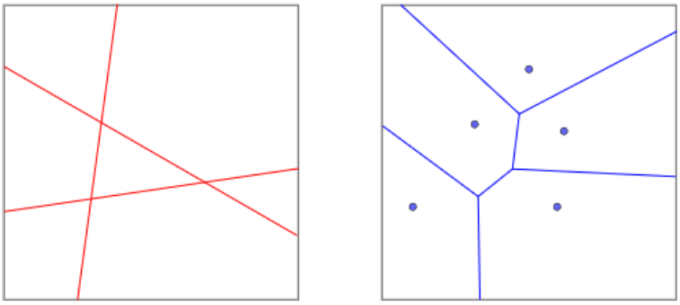


**Figure 3.3:** Difference between the region split produced by halfspace gating (left) and prototype gating (right) in a 2D context space [3].

Also in this case the choice of the prototypes can descend from a random sampling strategy, with the prototypes being sampled from a multivariate normal distribution $p \sim N(0, \mathbf{\Sigma})$ with 0 being the zero vector and $\mathbf{\Sigma}$ being a diagonal covariance matrix with small-value variances. Also in this case, the choice of small-value variances combined with standardized input data ensures an efficient choice of prototypes. Alternatively, a data-driven approach can be used, with prototypes selected among the input data. This can be done in two ways: by sampling among the whole dataset (if it is fully available since the beginning) or, in the case of a continual learning framework, by selecting iteratively the most significant items according to some importance criteria (like the distance from pre-existing prototypes). This last method in [3] leads to the definition of a sub-class of GLNs called Growing ProtoGLNs. While Munari in his work finds the growing approach to be more efficient, its performance with the more complex dataset used in this work will be discussed in Chapter **??**.

### 3.1.3 GLN FORMULATION

Independently from the choice of the gating function $c(z)$, by combining all the building blocks introduced so far with the equation 3.1 we obtain the following formulation for a neuron $j$ at layer $i$:

$$h^{(i)}_{j,(x,z)} = \sigma\left(\left(w^{(i)}_{j,(z)}\right)^T \sigma^{-1}\left(h^{(i-1)}_{(x,z)}\right)\right), \; i \geq 1 \tag{3.10}$$

where $w^{(i)}_{j,(z)}$ represents the set of weights activated by the contest $z$ through the context function $c^i_j$ associated to the neuron. Or, to switch to the matricial notation used in equation 3.3:

$$h^{(i)}_{j,(x,z)} = \sigma\left(W^{(i)}_{(z)} \sigma^{-1}\left(h^{(i-1)}_{(x,z)}\right)\right), \; i \geq 1 \tag{3.11}$$

A schematic depiction of the full model as defined above can be seen in Figure 3.4.

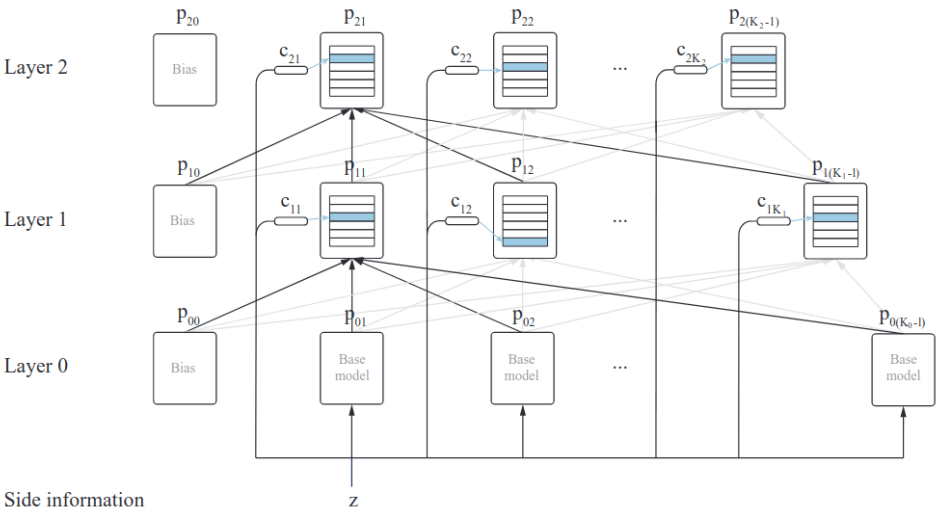



**Figure 3.4:** Illustration of the general architecture of a Gated Linear Network. Each neuron receives as input the outputs of the previous layer, with the side information $z$ broadcasted to every neuron and passed through each context function for the weight selection process [1].

## 3.2 GLN training

As said at the beginning of the chapter each neuron is assumed to be predicting the binary target $y$ at each training step, meaning that they can be trained individually without the need to resort to backpropagation. Thus given the side information $z$, at each training step $t$ the neuron $j$ at layer $i$ suffers a loss that is convex with respect to its active weights $u := w^{(i)}_{j,(z)}$ [65]. Using the notation from equation 3.7 the loss can be expressed as:

$$l_t^{GEO}(u) = -\log(\text{GEO}_u(y; p_t)) \tag{3.12}$$

with $\text{GEO}_u(1; p_t) = \sigma(u^T \sigma^{-1}(p_t))$ and $\text{GEO}_u(0; p_t) = 1 - \text{GEO}_u(1; p_t)$. The gradient of this loss function with respect to the weights $u$ is very easy to compute and is given by:

$$\nabla l_t^{GEO}(u) = (GEO_u(1; p_t) - y)\,\text{logit}(p_t) \tag{3.13}$$

which allows the model to carry out the update of the weights of all the neurons in parallel. The update of the weights can then be done using one of the many different online convex programming techniques, but in this work, we will restrict to Online Gradient Descent. Assuming that the weights belong to a convex set $W \subset \mathbb{R}^d$ then the algorithm is ensured to converge to the solution [66]. The most common way to bound a weight vector into a convex set is by clipping its components into an interval in the form $[-k, k]$ for some constant $k > 1$, which forces the weight vector inside a scaled hypercube of side $2k$.

# 4 Experiments

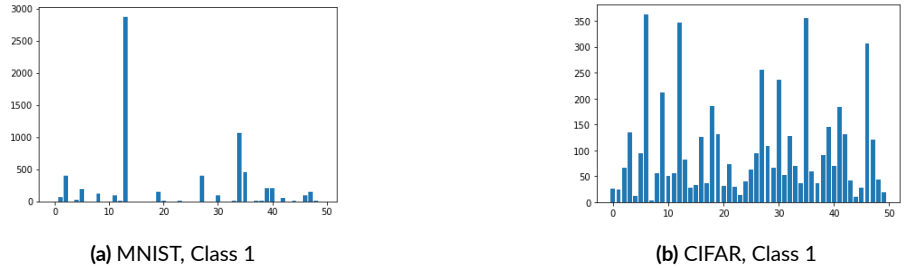
In this chapter, we show and analyze the results of the tests performed. We use the experiments in [3] as starting point for our study, and we will try to increase the complexity of the environment while trying to progressively move towards a more realistic learning scenario.

## 4.1 Experimental setting

This section reports the experimental setting to ease the reproducibility of the experiments.

### 4.1.1 Datasets

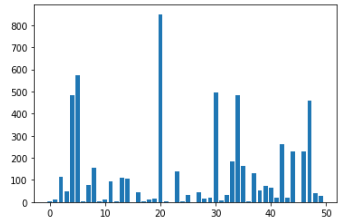
In this work, we will refer to two different datasets. We will try to compare the performance of GLNs on the MNIST dataset, used in the early work of [3] that we will use as a starting point, to the performance on the CIFAR-10 dataset to check how GLNs adapt to a significantly more complex environment. The performance of the GLNs (and other continual learning strategies) on MNIST has been widely assessed [39][1][3], but the relative simplicity, assessed through the two Figures on the next page, calls for test on more difficult datasets. Figure 4.1 compares the separability of classes from MNIST and CIFAR using 50 randomly generated prototypes: we can clearly see that the MNIST images are mostly concentrated in a limited number of areas while the CIFAR ones are more scattered and thus more likely to generate overlappings between different classes. Figure 4.2 shows the performance of GLN on one of the tasks presented in [3] and on the same task using the CIFAR dataset instead. The task in question is called Permuted-MNIST and consists in several training phases, each presenting to the model permutations of the original images. Also in this case we can see how harder it is for the model to perform the same task on the CIFAR dataset.



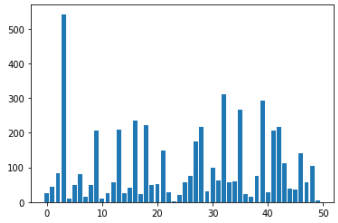
**(a)** MNIST, Class 1



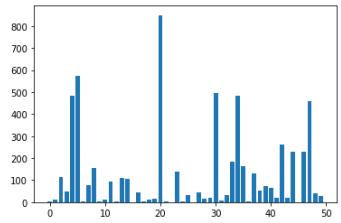
**(b)** CIFAR, Class 1



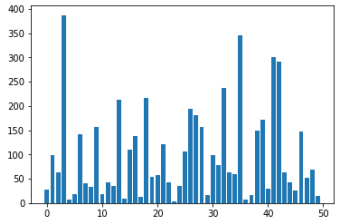
**(c)** MNIST, Class 2



**(d)** CIFAR, Class 2



**(e)** MNIST, Class 3



**(f)** CIFAR, Class 3

**Figure 4.1:** Comparison of the separability of the first 3 classes of the MNIST (left) and CIFAR-10 (right) datasets using 50 randomly generated prototypes.

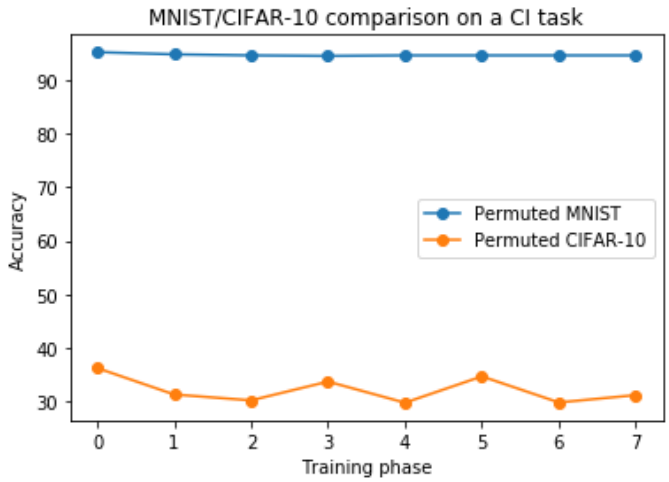



**Figure 4.2:** Comparison of a GLN on the Permuted MNIST task from [3] and on the same task performed using the CIFAR-10 dataset, while keeping all the model hyperparameters fixed.

## MNIST

The MNIST (Modified National Institute of Standards and Technology) database is a large database containing images of handwritten digits [67]. It contains 60,000 training images and 10,000 testing images, all equally split among 10 categories corresponding to the 10 digits. All the images are in black and white and in a 28x28 resolution. Figure 4.3 presents examples from the dataset.

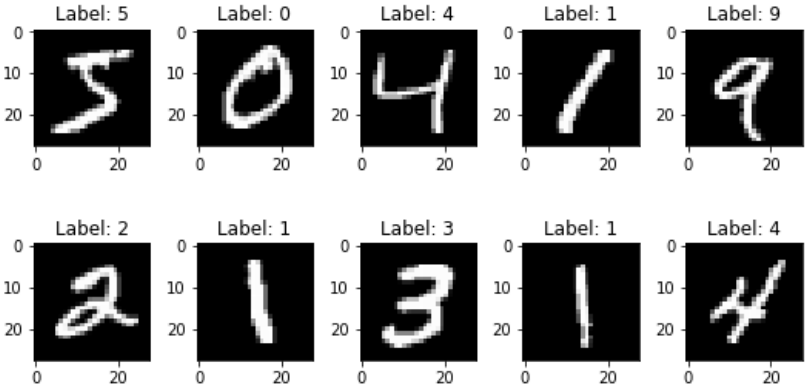



**Figure 4.3:** Examples of images from each of the 10 classes belonging to the MNIST dataset.

## CIFAR-10

The second dataset used in this work is CIFAR-10 [68]. It consists of 60,000 32x32 colored images, equally divided into 10 classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks, and of the 60,000 items in the dataset 50,000 are used for the training phase and 10,000 for the test phase. With respect to MNIST this dataset already results in a more difficult challenge due to a (slightly) bigger dimension of the images, the switch to color images with three different RGB channels, and the presence of significantly more complex figures and patterns.


**Figure 4.4:** Examples of images from all the 10 classes of CIFAR-10.

### 4.1.2 Task

As discussed in Chapter 2 working in a Class-Incremental With Repetition scenario has been proven more effective in dealing with catastrophic forgetting while allowing to present data to the model in a more realistic way. However, to this date, most of the experiments in continual learning are performed in class-incremental scenarios without any kind of repetition, by simply presenting to the model only unseen classes at each training phase, either from the same dataset [21][3], from different datasets [19] or using transformations of previously seen classes [58][3]. In order to address this problem this work will present the data to the GLN model in a CIR fashion, and in particular, referring to two different task-building criteria that increasingly relax the CI scenario. While at first glance it could seem that switching to a CIR scenario, and thus presenting new samples of old classes, would be equivalent to incorporating a rehearsal strategy into the model, there is a significant difference. Rehearsal strategies rely on information that has already been passed to the model and gets passed again either directly or indirectly. In this case, however, the new samples, while still belonging to old classes, represent previously unseen information and thus there is no rehearsal of past information involved.

In order to ease the comprehension of the learning scenario adopted in this study we will try to formalize the concept of task and task sequence. The data flow can be described as a sequence of couples in the form $(x^{(i)}, y^{(i)})$ with $x^{(i)} \in X$ being the input vector and $y^{(i)} \in \{c_1, \ldots, c_n\}$ its target label corresponding to one of the $n$ classes of the dataset. The data are presented to the model in a sequence of disjoint training phases $T_j \in \{T_1, \ldots, T_m\}$, each characterized by a different distribution of the $n$ classes. In the CIR framework, each training phase contains a few samples belonging to all the classes presented in the previous phases and samples belonging to at least one newly introduced class. The two scenarios used in this study differ in the number of samples of each class presented in each phase, which is determined using two different approaches. Figure 4.6 and 4.7 at the end of this section will give a visual comparison of how classes are presented to the model in the two cases.

We will now present in more detail the two scenarios. In both cases the samples will be presented to the model in an online fashion, meaning that the training will happen with just one epoch and batches of size one. A random permutation has also been applied to the order of the classes with the respect to the standard order of the dataset, which will be the order according to which they will be progressively presented to the model. Also in both cases, the first training phase contains 2 classes, and a new one is introduced at each one of the following ones resulting in a total of nine training phases. At the end of each training phase, the model is evaluated on all the classes that it has seen up to that moment.

### First CIR scenario

This first scenario has been applied only with the CIFAR-10 dataset, for which each class contains 5,000 training samples. In this scenario, each class is presented to the model following the same pattern, with the assumption that the majority of the samples of each class are presented at the beginning with only a few of them being used in later training phases. For each class $c_k$, being $T_{j^*}$ the first training phase in which the class is presented to the model, the number of samples from that class presented in each phase $T_j$ is determined according to the following rules:

- 0 for $j < j^*$
- 3,000 for $j = j^*$
- 600 for $j = j^* + 1$
- 200 for $j > j^* + 1$

The following table sums up the whole training process for this scenario, with the number of samples of each class presented in each training phase (with the classes already ordered according to the applied random permutation):

| Class / Task | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | TOT |
|---|---|---|---|---|---|---|---|---|---|---|
| Class 2 | 3000 | 600 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 5000 |
| Class 8 | 3000 | 600 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 5000 |
| Class 4 | | 3000 | 600 | 200 | 200 | 200 | 200 | 200 | 200 | 4800 |
| Class 9 | | | 3000 | 600 | 200 | 200 | 200 | 200 | 200 | 4600 |
| Class 1 | | | | 3000 | 600 | 200 | 200 | 200 | 200 | 4400 |
| Class 6 | | | | | 3000 | 600 | 200 | 200 | 200 | 4200 |
| Class 7 | | | | | | 3000 | 600 | 200 | 200 | 4000 |
| Class 3 | | | | | | | 3000 | 600 | 200 | 3800 |
| Class 0 | | | | | | | | 3000 | 600 | 3600 |
| Class 5 | | | | | | | | | 3000 | 3000 |
| TOT | 6000 | 4200 | 4000 | 4200 | 4400 | 4600 | 4800 | 5000 | 5200 | 42400 |

**Table 4.1:** Scheme of the first CIR scenario. It shows how many elements of each class are presented in each training phase. The classes are presented in the order of the random permutation applied.

Note that in this first scenario since all the classes follow the same pattern the number of presented samples does not depend on the number of training phases the class takes part in, which means that of the 5,000 training items not all of them are necessarily shown to the model.

### Second CIR scenario

This second scenario refers to the protocol introduced in [4] to build the sequence of training phases, which is described through the pseudocode in Figure 4.5:

**Require:** Dataset $D = \{(x_j, y_j)\}_{j=1,...,M}$, number of experiences $N$ in the stream, list of classes per each experience $C = (\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_N)$, where $\mathcal{C}_i = (c_i^1, c_i^2, ..., c_i^{k(i)})$ and $k(i)$ is the number of classes in experience $i$.

$p \leftarrow [0, 0, ..., 0]$ with as many elements as number of classes in $D$.
**for** $j \leftarrow 1, 2, ..., M$ **do** ▷ Number of patterns for each class in $D$
    $p[y_j] = p[y_j] + 1$
**end for**
$e \leftarrow [0, 0, ..., 0]$ with as many elements as number of classes in $D$
**for** $i \leftarrow 1, 2, ..., N$ **do** ▷ Experience count for each class
    **for** $k \leftarrow 1, 2, ..., k(i)$ **do**
        $e[c_i^k] = e[c_i^k] + 1$
    **end for**
**end for**
$p[i] \leftarrow floor(p[i]/e[i])$, $\forall i$

`stream` $\leftarrow$ empty stream
**for** $i \leftarrow 1, 2, ..., N$ **do** ▷ build the stream of experiences
    `stream`.append($E_i$)
    $\mathcal{C}_i \leftarrow C[i]$
    **for** $k \leftarrow 1, 2, ..., k(i)$ **do**
        $c_i^k \leftarrow \mathcal{C}_i[k]$
        $P_i^k \leftarrow$ sample $p[i]$ patterns from $D$ belonging to class $c_i^k$
        $D$.remove($P_i^k$)
        $E_i$.add($P_i^k$)
    **end for**
**end for**
**return** `stream`

**Figure 4.5:** Pseudocode for a protocol to build a class-incremental with repetition benchmark for continual learning.[4]

The protocol is quite intuitive and it can be applied to any dataset regardless of the number of samples it contains. However, it is based on a completely different assumption with respect to the first scenario. In this case, the classes do not follow the same pattern, but instead, for each class, the same number of samples is included in each training phase, obtained simply by dividing the number of samples of each class by the number of training phases the class takes part in. However due to the sizes of the training phases increasing over time the relative size of each class still decreases over time.

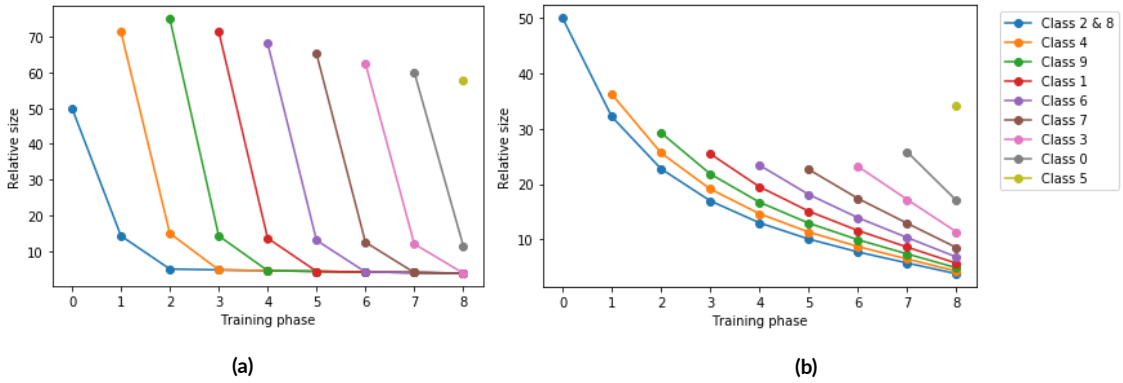



**Figure 4.6:** Relative size of each class of CIFAR-10 across all the training phases of the first CIR scenario (a) and the second (b).

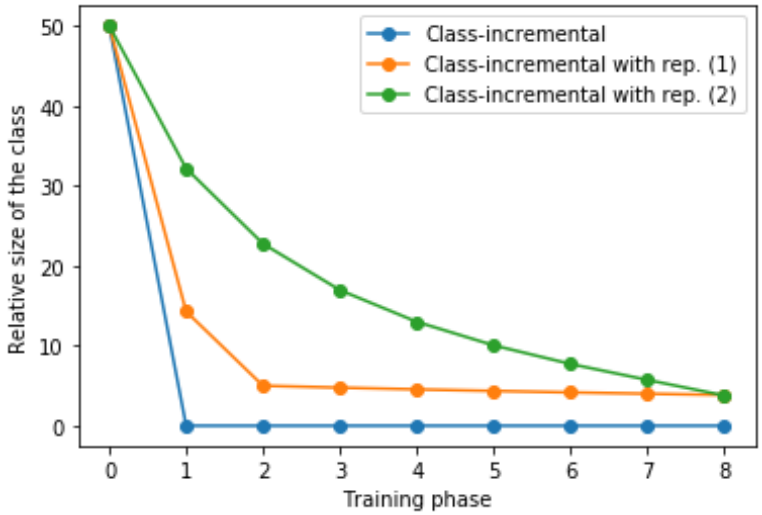



**Figure 4.7:** Visual comparison of the relative size of the first class of CIFAR-10 with respect to each training phase, in the simple class-incremental scenario (blue), in the first class-incremental with repetition scenario (orange) and in the second one (green).

### 4.1.3 Model

The two CIR task sequences outlined above have been used to test the perform of Gated Linear Networks. In particular, the model used in this work is based on the Prototypes GLN (PGLN) proposed in [3], with the addition of an encoder for the input data. The reasons behind the choices of the encoding and the gating mechanism, as much as the details of the normalization strategy used in this online environment, will be discussed in the following paragraphs. The choice of hyperparameters as learning rate, number of prototypes and architecture have been object of study and will be discussed in Section 4.2.

#### ResNet encoding

As we saw in Section 4.1.1 one of the practical consequences of moving to more complex datasets is having classes that are less separable in the input space. Since we are using a model whose strategy to counter catastrophic forgetting is based exactly on efficiently partitioning the input space it is a good idea to increase the separability of the data before presenting them to the GLN. An efficient strategy is to perform an encoding through a pretrained model, with the assumption that such an encoding would highlight the most distinctive features of each class.

In our specific case, we decided to perform the encoding through the ResNet-50 model. ResNet, short for Residual Network, refers to a type of Convolutional Neural Network first introduced in [69] that makes use of *skip connections* to train extremely deep neural networks without encountering the Vanishing Gradient problem, i.e. the tendency of the gradient to decrease significantly in value during backpropagation, resulting in negligible updates of the weights. Its first introduction coincided with its victory at the ILSVRC 2015 and is generally considered a model capable of reaching state-of-the-art results on image classification tasks. Because of their properties, the ResNet architectures have been widely used as a backbone in solving more complex computer vision tasks, covering also the role of encoders in many cases [70][71][72]. The specific architecture chosen for this work, ResNet-50, is as the name suggests a 50 layers-deep model, whose general structure can be seen in Figure 4.8:

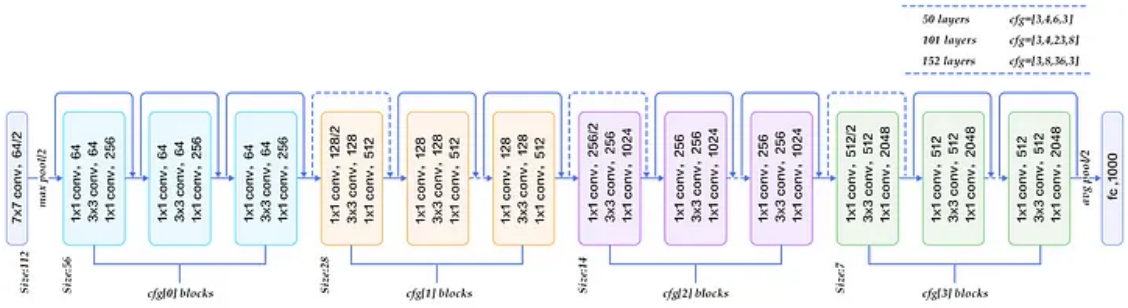



**Figure 4.8:** Scheme of the ResNet-50 architecture [5]

As we can see the architecture is composed of several convolutional blocks interconnected through the skip connections mentioned before. In this study we performed tests using as encoded inputs for the GLN the outputs of the third (pink) or fourth (green) blocks, both providing (after a proper MaxPooling step) a representation

in the form of a vector of size 2048, and we compared the performance of the model in the two cases trying to understand which was the best fit for our model and why.

In order to apply the encoding to samples from the CIFAR-10 dataset we used a ResNet trained on that same dataset, which has been found on the GitHub repository at [73].

### Gating strategy

In Chapter 3 we discussed several possible strategies for what concerns the gating mechanism in GLNs. In Munari's work [3] three different approaches are tested, with increasingly better results: Halfspace Gating, random Prototype Gating and Growing Prototype Gating. However, those results have been obtained with the MNIST dataset. In the preliminary tests that we run, applying the same task sequences used in [3] (Split-MNIST and Permuted-MNIST) to the CIFAR-10 dataset, we obtained different results. In particular, we found that consistently with [3] randomly initialized Prototypes resulted in a better performance than halfspace gating, but the same did not apply to the Growing Prototype approach, which was actually outperformed by the other two.

The cause of this apparently unexpected can be found by analyzing the functioning of the Growing mechanism in relationship with the internal structure of the datasets in question. According to the Growing mechanism proposed in [3] each newest sample is considered a new prototype whenever it falls far enough from the pre-existing prototypes in the context space, with an additional mechanism that removes unused prototypes after a while. This is done through the definition of a distance threshold, in that case, fixed as the average distance between samples within the dataset. Such a mechanism is based on the assumption that different classes naturally fall into a series of more-or-less well-defined non-overlapping cluster(s) so that members of different classes are on average more distant. However we have already shown in Figure 4.1 that this assumption can hold for a simple complex such as MNIST, but it is definitely not true in the case of CIFAR-10. For this reason, we decided to proceed with our study by using the Prototype gating with randomly generated prototypes.

### Normalization

As discussed in Chapter 3 when dealing with GLNs the standardization of the context information $z$ becomes a necessary step in order to ensure an optimal partition of the context space, both with halfspace and prototype gating. This is a procedure that normally relies on the knowledge of the true expected value $\mu_i$ and the true standard deviation $\sigma_i$ for each feature $i$ of the context space. However in an online learning environment like the one where we performed our tests that knowledge is not available. To overcome this problem the authors of the paper where GLNs have been first introduced [39] used Welford's online algorithm [74] to iteratively estimate the two values.

The algorithm makes use of the following formulas to update the values:

$$\mu_i(t+1) = \mu_i(t) + \frac{z_i(t+1) - \mu_i(t)}{t+1} \tag{4.1}$$

$$\sigma_i^2(t+1) = \frac{m_i(t+1)}{t+1} \tag{4.2}$$

where

$$m_i(t+1) = m_i(t) + \big(z_i(t+1) - \mu_i(t)\big)\,\big(z_i(t+1) - \mu_i(t+1)\big) \tag{4.3}$$

initialized with $\mu_i(1) = z_i$ and $m_i(1) = 0$.

Of course, introducing such a standardization strategy comes at a cost, in terms of having an early training performed with standardization parameters substantially different from the current ones. However, this is a scenario consistent with real-world flows of information where it is impossible to predict future shifts in the data distribution and represents a further step in the goal of this work to move towards more realistic learning frameworks.

### 4.1.4 Evaluations

In this work the performance of GLNs has been evaluated using two different metrics:

- the first is the *accuracy* of the model, measured in amount of correctly classified items. It has been computed separately for each training phase to account for the performance of the model over time, both class-wise and averaged over the classes;
- the second is the *backward transfer* (BWT) [58], a measure of how the performance on a class is influenced by the introduction of new classes. When positive, it means that the knowledge about a new class is also used to better classify old ones; when negative, it works as a measure of the catastrophic forgetting. If we call $a_{i,j}$ the accuracy on the j-th class during the i-th training phase, $m$ the number of training phases and $\bar{i}$ the index of the training phase were the i-th class is first introduced, we can formally define backward transfer of the model as

$$BWT = \frac{1}{m-1} \sum_{i=1}^{m-1} a_{m,i} - a_{\bar{i},i} \tag{4.4}$$

The reason behind the choice of this dual system of evaluation lies on what we expect from a model capable of behaving correctly in a continual learning environment: a good classifying performance, and little catastrophic forgetting. Moreover, if considered independently they could be misleading: a relatively stable average accuracy could hide very unstable class-wise patterns with a lot of forgetting, while a low BWT could be just the result of a low accuracy since the early training phases.

## 4.2 Results

With the nature of this study being exploratory, the directions to follow have been determined in a progressive way. Having defined a more complex learning environment through the choice of the dataset, and being conscious thanks to the exploratory tests of the greater challenge it would have represented we first decided to apply an encoding (form the 4th ResNet block) and to choose a first CIR task sequence (the first) and see how the model would have performed in this situation. After having explored the performance of the model in this context we decided to proceed with testing a different encoding and to apply the model in a different CIR scenario. In each of the subsections plots are presented with a graphical comparison of the accuracy trends between different tested models. In the first subsection some tables have been added to better underline the performance with different combinations of hyperparameters, while Table 4.5 at the end of the section will recap the accuracy and BWT scores of the best models found in each subsection.

All the experiments have been performed on Python v3.10.4, using Pythorch v1.12.1, Torchvision v0.13.1 and Numpy v1.23.1.

### 4.2.1 Hyperparameters

We decided to try different learning rates, number of prototypes, layers and neurons, with disappointing results. It was possible to identify an optimal combination of hyperparameter, consisting in a learning rate of 0.5, 50 prototypes and a two layers architecture by looking at the average accuracy trend across all the training phases and prioritizing a better performance on the late phases. However, the differences in performance between different sets of hyperparameters were very tiny. In particular, out of all the tests performed Figure 4.9 shows the average accuracy trends for different hyperparameter combinations based on the optimal one, changing one hyperparameter and keeping the others fixed. It is quite clear from those plots that while still outperforming the random classifier, the differences between different combinations are mostly negligible. Tables 4.2, 4.3 and 4.4 show in more detail the accuracy and BWT scores of all the the combinations presented in the Figures.

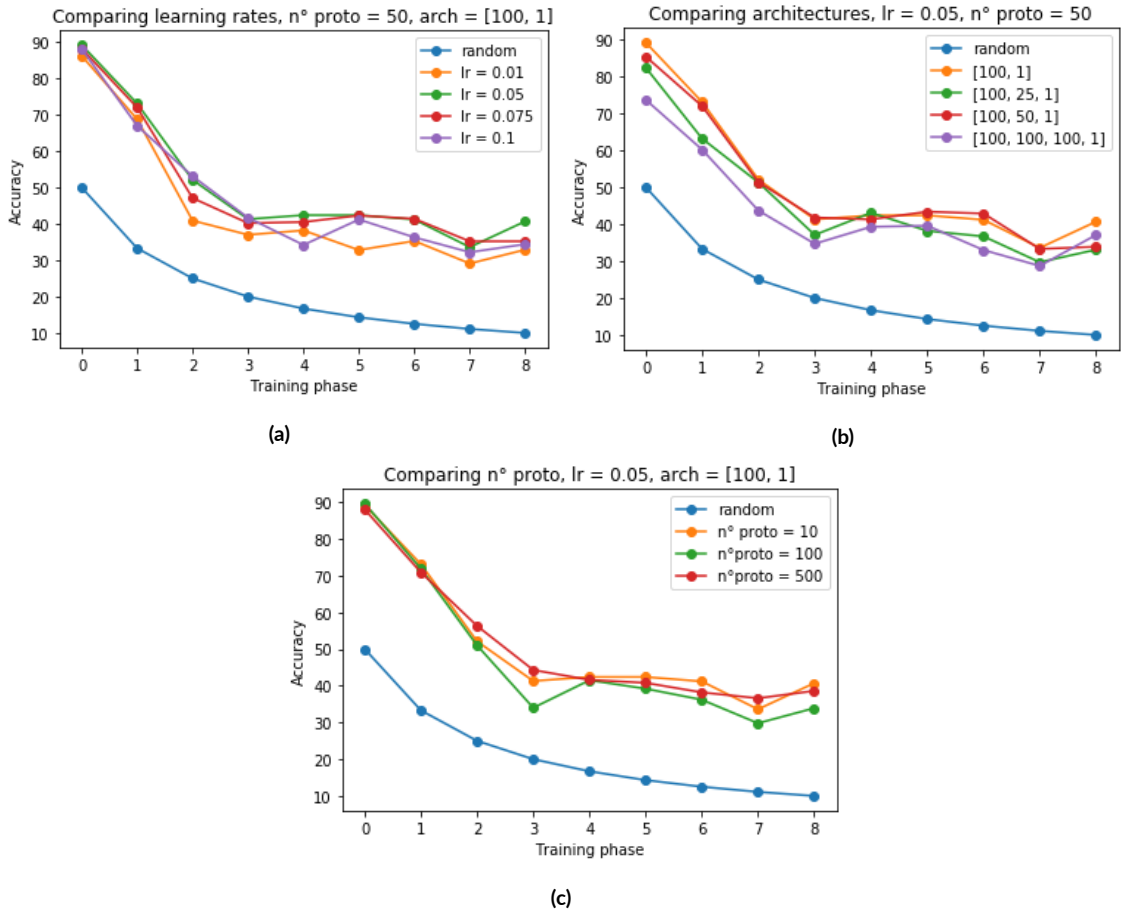



**Figure 4.9:** Results obtained by performing tests on different sets of hyperparameters, specifically learning rates, architectures and number of prototypes, using the random classifier (blue) as reference. Subfigure (a) shows average accuracy trends for different learning rates, Subfigure (b) for different architectures and Subfigure (c) for different numbers of initialized prototypes.

| Model | Accuracy | | | | | | | | | BWT |
|---|---|---|---|---|---|---|---|---|---|---|
| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | |
| Random | 50.0 | 33.3 | 25.0 | 20.0 | 16.7 | 14.3 | 12.5 | 11.1 | 10.0 | -15.9 |
| lr = 0.01 | 86.2 | 68.8 | 40.9 | 37.0 | 38.2 | 32.8 | 35.3 | 29.1 | 32.9 | -58.2 |
| lr = 0.05 | 89.2 | 73.1 | 52.1 | 41.3 | 42.4 | 42.4 | 41.2 | 33.6 | 40.7 | -48.6 |
| lr = 0.075 | 88.2 | 72.0 | 47.1 | 40.2 | 40.5 | 42.3 | 41.4 | 35.2 | 35.2 | -43.1 |
| lr = 0.1 | 88.1 | 67.0 | 53.1 | 41.7 | 34.2 | 41.2 | 36.3 | 32.2 | 34.4 | -52.9 |

**Table 4.2:** Summary of the tests performed on the learning rate.

| Model | Accuracy | | | | | | | | | BWT |
|---|---|---|---|---|---|---|---|---|---|---|
| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | |
| Random | 50.0 | 33.3 | 25.0 | 20.0 | 16.7 | 14.3 | 12.5 | 11.1 | 10.0 | -15.9 |
| [100, 1] | 89.2 | 73.1 | 52.1 | 41.3 | 42.4 | 42.4 | 41.2 | 33.6 | 40.7 | -48.6 |
| [100, 25, 1] | 82.4 | 63.2 | 51.3 | 37.1 | 43.1 | 38.2 | 36.7 | 29.7 | 33.1 | -49.9 |
| [100, 50, 1] | 83.1 | 68.3 | 50.1 | 40.2 | 39.5 | 41.2 | 40.4 | 31.6 | 30.3 | -50.0 |
| [100, 100, 100, 1] | 73.7 | 60.2 | 43.7 | 34.7 | 39.3 | 39.6 | 33.0 | 28.7 | 37.1 | -42.5 |

**Table 4.3:** Summary of the tests performed on the model architecture.

| Model | Accuracy | | | | | | | | | BWT |
|---|---|---|---|---|---|---|---|---|---|---|
| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | |
| Random | 50.0 | 33.3 | 25.0 | 20.0 | 16.7 | 14.3 | 12.5 | 11.1 | 10.0 | -15.9 |
| 50 proto | 89.2 | 73.1 | 52.1 | 41.3 | 42.4 | 42.4 | 41.2 | 33.6 | 40.7 | -48.6 |
| 100 proto | 89.6 | 71.9 | 51.0 | 34.0 | 41.5 | 39.2 | 36.2 | 29.8 | 33.9 | -56.2 |
| 500 proto | 88.0 | 70.9 | 56.3 | 44.3 | 41.6 | 40.8 | 38.2 | 36.6 | 38.6 | -56.3 |

**Table 4.4:** Summary of the tests performed on the number of prototypes.

The cause of these results can be investigated by analyzing in more detail the performance of the best model. Figure 4.12 in the next page shows class-wise accuracy patterns for that model. Interestingly, we see some violent downward peaks in accuracy for some specific classes. In particular, we see a drop for class 4 when class 7 is introduced, for class 8 when class 0 is introduced and for class 3 when class 5 is introduced. Those combinations are not random: by looking at the composition of the dataset we see that those correspond to the couples deer/horse, cat/dog and plane/ship. As we can see in Figure 4.10, the images belonging to those classes are very similar in terms of subjects, shapes or backgrounds. This suggests that the model is still not powerful enough to perform a clear distinction between those classes, resulting in interference between classes that enhance the catastrophic forgetting problem.

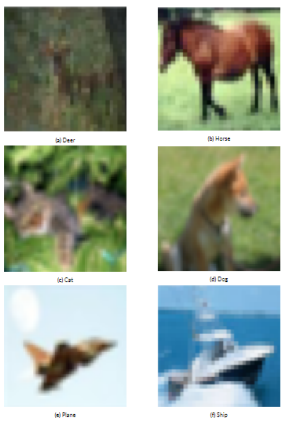


**Figure 4.10:** Similar classes.

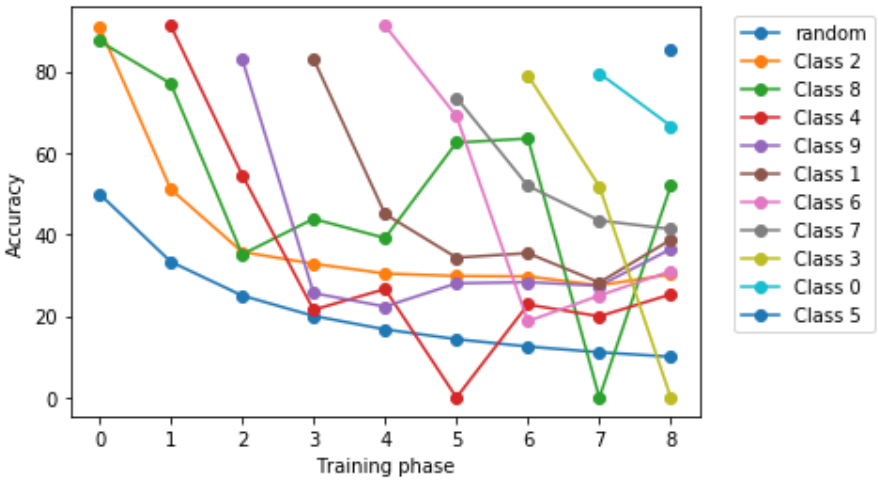



**Figure 4.11:** Class-wise accuracy patterns for the best model with learning rate = 0.05 and 50 prototypes.

A further confirmation of the weakness of the model in this form is given by a comparison of its performance with that of a linear model (equivalent for the reasons explained in Chapter 3 to a GLN with just one prototype). The comparison can be seen in Figure 4.12, that show clearly that while the GLN becomes more effective when more and more classes are introduced, in the early stages its performance is only slightly better than that of the linear model. In order to increase the power of the model we considered two strategies: increasing significantly the number of prototypes, both randomly- and data-generated, and improving the quality of the encoding. While the latter will be presented in the next section, it has not been possible to pursue the former due to its memory load, a problem that will be discussed more in detail in the conclusions.

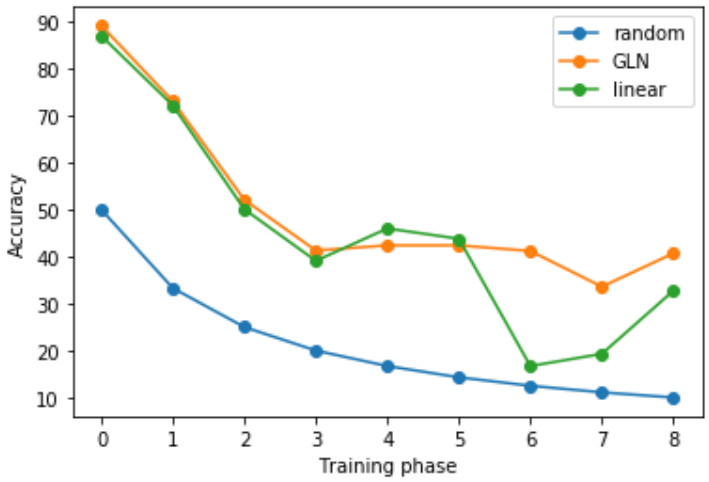



**Figure 4.12:** Comparison between average accuracy trend for the best GLN obtained so far and its linear counterpart.

### 4.2.2 Encoding

Having verified that the main reason behind the suboptimal performance of the GLNs in our tests depended from their lack to grasp the complexity of the presented data, we decided to intervene on the encoding phase by using a different output from the ResNet. In particular, as anticipated in Section 4.1.3 we performed tests comparing the performance of the GLN using inputs encoded up to the 3rd convolutional block of the ResNet with those encoded with the whole model. The tests have been done using be optimal combination of hyperparameters found in the previous section, therefore with a learning rate of 0.05, 50 prototypes and a fully connected layer with 100 neurons.

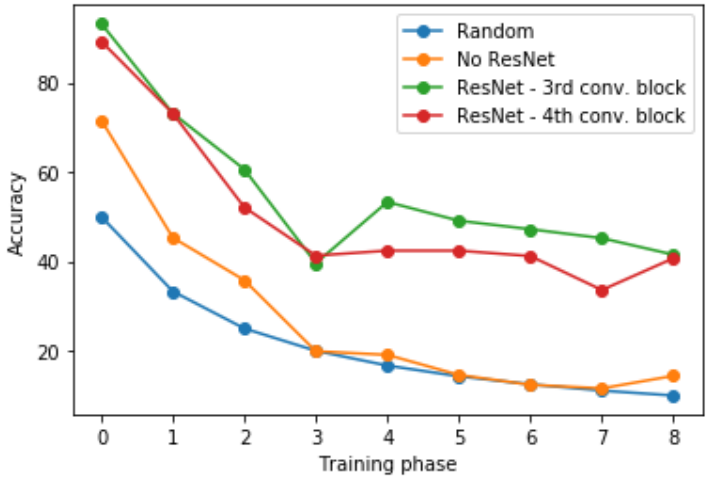



**Figure 4.13:** Average accuracy of the model over the 10 classes in 3 different cases: no ResNet encoding (orange), encoding taken from the 3rd convolutional block (green) and from the 4th convolutional block (red).

The results can be seen in Figure 4.13, that includes also the performance of a model without any encoding at all. While it is quite clear that any encoding increases significantly the performance of the model, with the performance of the model without any kind of encoding being comparable to that of the random classifier, the difference between the two types of encoding is a bit less evident. However, using the output from the 3rd convolutional block outperform the alternative at almost every training phase, especially during the late ones. By looking at the class-wise accuracy we can see another improvement provided by the choice of the the 3rd block output: Figure 4.14 shows that the patterns are more stable and the downward peaks are weaker than what seen using the 4th block in the previous section. The reasons behind this apparently counter intuitive result, i.e. a less processed input leading to a better classification performance, lies in the capability of models relatively simple like GLNs to properly handle heavily processed data and will be discussed more in detail in the conclusions.

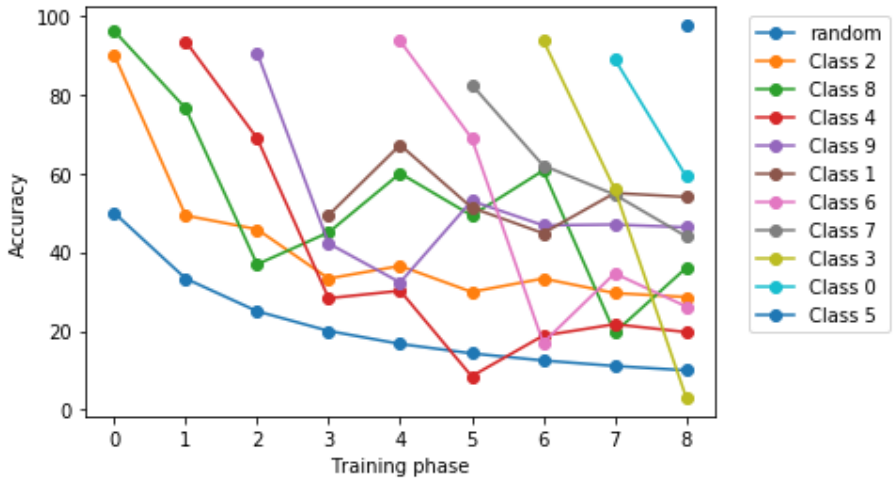



**Figure 4.14:** Class-wise accuracy using the ResNet encoding from the 3rd convolutional block.

### 4.2.3 Task sequence

Similarly to the tests performed with different types of encoding, in this case we tried to compare the performance of the model over the two Class-Incremental with Repetition scenarios introduced in Section 4.1.2. Since the performance of the GLN still showed in the tests performed so far a certain degree of catastrophic forgetting, despite a relative stability of the average accuracy towards the final training phases, it makes sense to relax even more the constraint on the presentation of samples from old classes with the second CIR scenario. Once again, the tests have been performed using a model with 50 prototypes, a learning rate of 0.05 and a fully connected layer of 100 neurons.

The results of this test are shown in Figure 4.15 and, as expected, we see the GLN performing better in every learning phase but the first and generally being more stable in terms of fluctuations of the average accuracy.

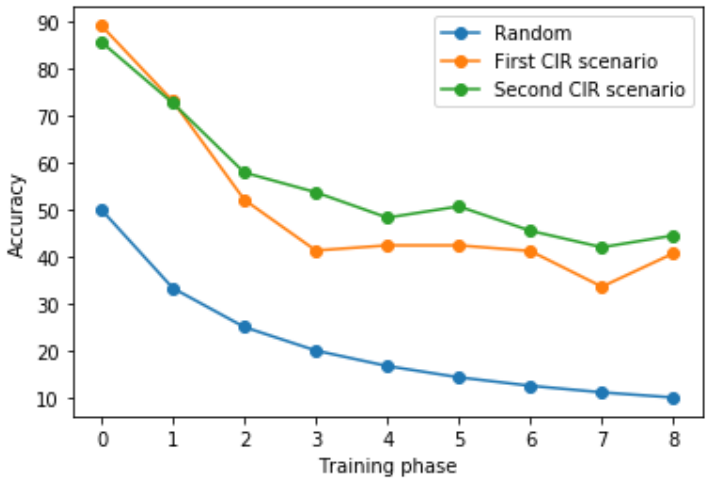



**Figure 4.15:** Comparison of the performance of the best model (learning rate = 0.05, 50 prototypes, one fully connected layer with 100 neurons) in the first (orange) and second (green) CIR scenarios, with the random classifier (blue) for reference.

### 4.2.4 Final summary

To conclude the experimental part of this work, we tried to combine the best set of hyperparameters, the optimal encoding and the second CIR all in one test. The results of this last test are summarized in the next two figures. Figure 4.16 compares the average accuracy before and after adding both the new encoding and the new task sequence. As we can see, the new additions lead the model into significantly outperforming the base version, with the accuracy stabilizing around 60% as soon as with the 5th training phase, compared to the base models that after the third training phase never manages to go back over a 50% average accuracy. Figure 4.17 shows the class-wise accuracy patterns of this final implementations, and even here huge improvements are clearly visible. All the trends appear generally more stable, and the downward peaks are reduced in terms of both quantity and intensity, with non of them ever going below a 30% accuracy, let alone touching values below 10% down to even 0% as it was for the first tests.

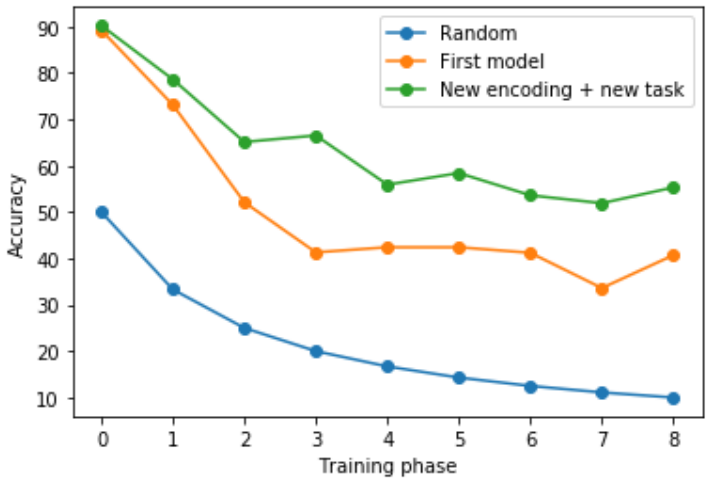



**Figure 4.16:** Comparison of the performance of the best model (learning rate = 0.05, 50 prototypes, one fully connected layer with 100 neurons) tested without any extra additions (orange) and with the new task sequence and encoding (green), with the random classifier (blue) as reference.

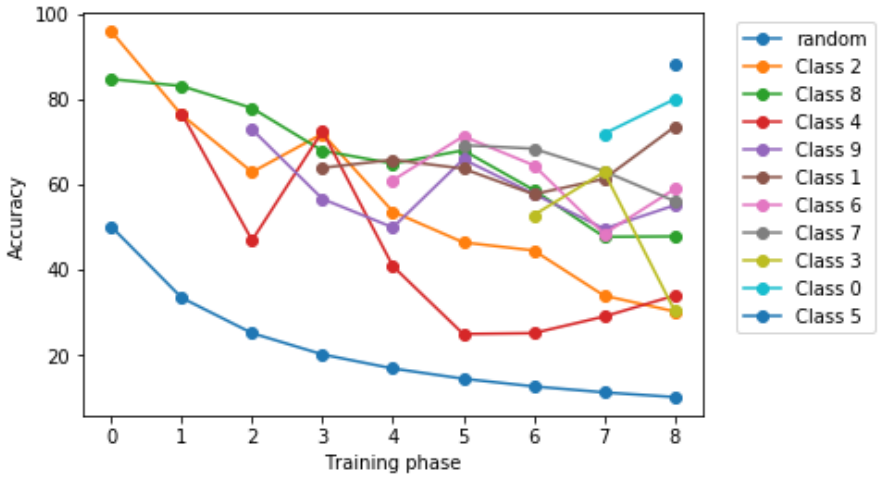



**Figure 4.17:** Class-wise accuracy patterns for the optimal model (learning rate = 0.05, 50 prototypes, one fully connected layer with 100 neurons) with the new encoding and the new task sequence.

Table 4.5 sums up the results for the models that have been progressively tested across this study. First we can observe that the combination of an encoding from a lower layer of the ResNet combined with the second CIR learning environment actually grants the best possible performance, both in terms of average accuracy and in terms of BWT score. Furthermore, we can observe a tangible proof of the necessity to adopt a dual evaluation method: the random classifier is the one that, on paper, shows the less forgetting, but only because it performs very badly since the beginning.

| **Model** | **Accuracy** | | | | | | | | | **BWT** |
|---|---|---|---|---|---|---|---|---|---|---|
| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | |
| Random | 50.0 | 33.3 | 25.0 | 20.0 | 16.7 | 14.3 | 12.5 | 11.1 | 10.0 | -15.9 |
| CIR1+ResNet4 | 89.2 | 73.1 | 52.1 | 41.3 | 42.4 | 42.4 | 41.2 | 33.6 | 40.7 | -48.6 |
| CIR1+ResNet3 | 93.2 | 73.1 | 60.6 | 39.6 | 53.3 | 49.1 | 47.2 | 45.2 | 41.5 | -51.3 |
| CIR2+ResNet4 | 85.7 | 72.7 | 57.9 | 53.7 | 48.3 | 50.7 | 45.5 | 42.0 | 44.5 | -29.1 |
| CIR2+ResNet3 | 90.2 | 78.6 | 65.1 | 66.5 | 55.9 | 58.4 | 53.6 | 51.9 | 55.3 | -21.4 |

**Table 4.5:** Summary of the performances of the best models found in each subsection, classified according to the type of encoding (ResNet3 or ResNet4) and the task sequence used to train them (CIR1 or CIR2). For each of them the average accuracy in each training phase and the final BWT score are shown.

# 5
# Conclusions

In this work we tried to evaluate the performance of Gated Neural Networks in a complex environment, defined by the choice of datasets that are more challenging than those used in prior works. The evaluation has been performed in a Class-Incremental with Repetition framework, that while being closer to real-world scenarios allows to implicitly mitigate the catastrophic forgetting, the main problem that arises when training a model within a Continual Learning paradigm.

We saw that, as expected, the performance of GLNs drops significantly when moving from a simple dataset like MNIST to a more complex one like CIFAR-10. We first observed that while an optimal set of hyperparameters can be found, it will not change the final performance in a significant way. The small improvements obtained by changing the combination of most hyperparameters, if compared with the assumption that a GLN has the potential to approximate any function no matter its complexity, lead us into thinking that the problem resides in the complexity of the model itself. This idea was confirmed by looking at the class-wise accuracy patterns, that present huge downward peaks in correspondence with the introduction of new classes whose samples resemble closely those of old classes, and by the fact that the GLN performance was only slightly better than the one of a basic linear classifier. We decided therefore to implement new strategies to increase the capabilities of the model. While the result of testing different encodings lead to encouraging results and will be discussed in the next paragraph, we haven not been able to carry out the attempt to significantly increase the number of prototypes becasue of some technical limitations that will be discussed later on.

The first main finding of our tests using different encoding is that indeed, as our preliminary tests suggested, introducing an encoding given by a pretrained model like ResNet can help to significantly improve the performance of GLNs. This can be explained by the fact that such an encoding improves the separability of the different classes of the dataset, and therefore diminishes the amount of spatial overlappings between different classes making the gating mechanism more efficient. As a confirmation of that, we could see that the performance of the model

dealing with raw data without any encoding was comparable to that of a random classifier. Moreover, we saw that taking the encoding from an output of an earlier point of the ResNet results in a better performance of the attached GLN classifier. This result, however counter intuitive, can be explained by taking into account that in deep neural networks the specialization and the complexity of the learned representations of the model increase together with its depth, a known and well-verified fact. In a model as deep and as powerful as a ResNet this means that the representations produced by the very last layer are incredibly complex, and very hard to learn for a model such as a GLN, that relies mostly on linear transformation and spatial separability. With this assumption taking the output from the 3rd block instead of the 4th, giving up on around twenty layers of various transformations, represents a good compromise between increasing the separability of the data and learning a representation that is feasible for being learned by a GLN.

While testing the two different Class Incremental with Repetition scenarios, we found as expected that the more we relax the lack of repetition the better the performance becomes. With the perspective of applying continual learning models to flows of data coming from real-world environments like robotics or automated driving, that would most likely result in a CIR flow of information, it makes sense to train and evaluate models in CIR scenarios. From this point of view we found the prior literature to be lacking, with most of the continual learning strategies being tested on Class-Incremental without repetition scenarios, resulting in a consequent lack of benchmarks to be exploited. In view of that, on one hand the pseudocode proposed in [4] for the protocol to generate CIR task sequences should be applied to even more complex pre-existing classification datasets; on the other hand, more task-specific datasets like Core50 [36], with built-in CIR training sequences, should be developed and introduced as benchmarks.

One of the main limitations found in this work regards the memory consumption of GLNs. The problem emerged mainly in two situations: when combining a data-driven initialization of the prototypes, and when trying to significantly scale up the number of prototypes. While if this could become an unsolvable limitation for future developments of GLNs in even more complex environments is a possibility worth of being investigated, it is true that it represented a significant limitation for this study. As we already said testing various combination of hyperparameters lead to mixed results, with the GLN only slightly overperforming the linear baseline. This, together with the separability problems that we already discussed, suggests that a significantly higher number of prototypes (or a relatively similar number of data-driven prototypes) could have lead to a better performance solving some of the stability problems encountered in this study. While this couldn't have been verified in this study it represents for sure a promising way forward.

Other improvements could be done regarding how prototypes are generated. As explained in Chapter 4 the Growing ProtoGLN approach has been proved to not be particularly effective whenever the data are nor clearly separable and the distance between samples is highly variable. However the idea of generating prototypes in a data-driven fashion, instead of generating them randomly, remains still valid and potentially more memory efficient since it reduces the risk of generating useless partitions in the context space. A way to implement such a strategy could be found taking inspiration from the generative rehearsal strategies for deep learning described in Chapter 2. Some information about the classes could be stored, for example in the form of normal distributions approximating the data distribution of each class that get continuously updated while more data from those classes are

detected. The prototypes could then be sampled from those distributions on a regular basis, without the need to initialize all of them since the beginning and thus reducing even more the memory load of the model.

A further step of development is also represented by the detection of new classes. The GLN as developed in this work, while not actively using the knowledge of the task to mitigate catastrophic forgetting, still relied on it to detect when a new class is introduced and expand the model to account for that class in a proper way. It could be interesting to explore the possibility of having GLNs detecting autonomously the introduction of new classes, possibly exploiting information related to the frequency with which different prototypes are activated. This would turn the model completely fit for a task-agnostic scenario, a property that should be considered desirable being that scenario the most realistic one.

# References


[1] J. Veness, T. Lattimore, D. Budden, A. Bhoopchand, C. Mattern, A. Grabska-Barwinska, E. Sezener, J. Wang, P. Toth, S. Schmitt *et al.*, "Gated linear networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 10 015–10 023.

[2] E. Sezener, M. Hutter, D. Budden, J. Wang, and J. Veness, "Online learning in contextual bandits using gated linear networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 467–19 477, 2020.

[3] M. Munari, "Extending gated linear networks for continual learning," 2022.

[4] A. Cossu, G. Graffieti, L. Pellegrini, D. Maltoni, D. Bacciu, A. Carta, and V. Lomonaco, "Is class-incremental enough for continual learning?" *Frontiers in Artificial Intelligence*, vol. 5, 2022.

[5] A. Rastogi. (2022) Resnet50. [Online]. Available: https://blog.devgenius.io/resnet50-6b42934db431

[6] T. M. Mitchell and T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997, vol. 1, no. 9.

[7] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[8] W.-Y. Loh, "Fifty years of classification and regression trees," *International Statistical Review*, vol. 82, no. 3, pp. 329–348, 2014.

[9] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[10] A. G. Ivakhnenko, V. G. Lapa, and V. G. Lapa, *Cybernetics and forecasting techniques*. American Elsevier Publishing Company, 1967, vol. 8.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[12] ——, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[13] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "Coca: Contrastive captioners are image-text foundation models," *arXiv preprint arXiv:2205.01917*, 2022.

[14] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, 2023.

[15] K. Chowdhary and K. Chowdhary, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[16] S. Yang, Y. Wang, and X. Chu, "A survey of deep learning techniques for neural machine translation," *arXiv preprint arXiv:2002.07526*, 2020.

[17] P. Khilari and V. Bhope, "A review on speech to text conversion methods," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 4, no. 7, pp. 3067–3072, 2015.

[18] I. Adjabi, A. Ouahabi, A. Benzaoui, and A. Taleb-Ahmed, "Past, present, and future of face recognition: A review," *Electronics*, vol. 9, no. 8, p. 1188, 2020.

[19] R. Aljundi, K. Kelchtermans, and T. Tuytelaars, "Task-free continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 254–11 263.

[20] M. B. Ring, "Child: A first step towards continual learning," in *Learning to learn*. Springer, 1998, pp. 261–292.

[21] O. Ostapenko, M. Puscas, T. Klein, P. Jahnichen, and M. Nabi, "Learning to remember: A synaptic plasticity driven framework for continual learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 321–11 329.

[22] V. Lomonaco, "Continual learning with deep architectures," Ph.D. dissertation, Università di Bologna, 2019.

[23] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[24] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[25] W. C. Abraham and A. Robins, "Memory retention–the synaptic stability versus plasticity dilemma," *Trends in neurosciences*, vol. 28, no. 2, pp. 73–78, 2005.

[26] M. Mermillod, A. Bugaiska, and P. Bonin, "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects," p. 504, 2013.

[27] S. Stojanov, S. Mishra, N. A. Thai, N. Dhanda, A. Humayun, C. Yu, L. B. Smith, and J. M. Rehg, "Incremental object learning from contiguous views," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8777–8786.

[28] A. Thai, S. Stojanov, I. Rehg, and J. M. Rehg, "Does continual learning= catastrophic forgetting?" *arXiv preprint arXiv:2101.07295*, 2021.

[29] V. Lomonaco, D. Maltoni, and L. Pellegrini, "Rehearsal-free continual learning over small non-iid batches." in *CVPR Workshops*, vol. 1, no. 2, 2020, p. 3.

[30] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3987–3995.

[31] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[32] S. Lee, J. Ha, D. Zhang, and G. Kim, "A neural dirichlet process mixture model for task-free continual learning," *arXiv preprint arXiv:2001.00689*, 2020.

[33] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." *Psychological review*, vol. 102, no. 3, p. 419, 1995.

[34] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.

[35] A. Gepperth and C. Karaoguz, "A bio-inspired incremental learning architecture for applied perceptual problems," *Cognitive Computation*, vol. 8, no. 5, pp. 924–934, 2016.

[36] V. Lomonaco and D. Maltoni, "Core50: a new dataset and benchmark for continuous object recognition," in *Conference on Robot Learning*. PMLR, 2017, pp. 17–26.

[37] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[38] N. Y. Masse, G. D. Grant, and D. J. Freedman, "Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization," *Proceedings of the National Academy of Sciences*, vol. 115, no. 44, pp. E10 467–E10 475, 2018.

[39] J. Veness, T. Lattimore, A. Bhoopchand, A. Grabska-Barwinska, C. Mattern, and P. Toth, "Online learning with gated linear networks," *arXiv preprint arXiv:1712.01897*, 2017.

[40] E. Sezener, A. Grabska-Barwińska, D. Kostadinov, M. Beau, S. Krishnagopal, D. Budden, M. Hutter, J. Veness, M. Botvinick, C. Clopath *et al.*, "A rapid and efficient learning rule for biological neural circuits," *BioRxiv*, 2021.

[41] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[42] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber, "Compete to compute," *Advances in neural information processing systems*, vol. 26, 2013.

[43] C. Clopath, "Synaptic consolidation: an approach to long-term learning," *Cognitive neurodynamics*, vol. 6, no. 3, pp. 251–257, 2012.

[44] G. E. Hinton and D. C. Plaut, "Using fast weights to deblur old memories," in *Proceedings of the 9th annual conference of the cognitive science society*, 1987, pp. 177–186.

[45] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 139–154.

[46] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach, "Uncertainty-guided continual learning with bayesian neural networks," *arXiv preprint arXiv:1906.02425*, 2019.

[47] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[48] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[49] T. Furlanello, J. Zhao, A. M. Saxe, L. Itti, and B. S. Tjan, "Active long term memory networks," *arXiv preprint arXiv:1606.02355*, 2016.

[50] H. Jung, J. Ju, M. Jung, and J. Kim, "Less-forgetting learning in deep neural networks," *arXiv preprint arXiv:1607.00122*, 2016.

[51] T. L. Hayes, N. D. Cahill, and C. Kanan, "Memory efficient experience replay for streaming learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9769–9776.

[52] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," *Advances in neural information processing systems*, vol. 30, 2017.

[53] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Networks*, vol. 116, pp. 56–73, 2019.

[54] P. Kirichenko, M. Farajtabar, D. Rao, B. Lakshminarayanan, N. Levine, A. Li, H. Hu, A. G. Wilson, and R. Pascanu, "Task-agnostic continual learning with hybrid probabilistic models," *arXiv preprint arXiv:2106.12772*, 2021.

[55] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[56] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, "Dark experience for general continual learning: a strong, simple baseline," *Advances in neural information processing systems*, vol. 33, pp. 15 920–15 930, 2020.

[57] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[58] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," *Advances in neural information processing systems*, vol. 30, 2017.

[59] R. Kemker and C. Kanan, "Fearnet: Brain-inspired model for incremental learning," *arXiv preprint arXiv:1711.10563*, 2017.

[60] G. I. Parisi, J. Tani, C. Weber, and S. Wermter, "Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization," *Frontiers in neurorobotics*, p. 78, 2018.

[61] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[62] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[63] D. Budden, A. Marblestone, E. Sezener, T. Lattimore, G. Wayne, and J. Veness, "Gaussian gated linear networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 508–16 519, 2020.

[64] C. Mattern, "Mixing strategies in data compression," in *2012 Data Compression Conference*. IEEE, 2012, pp. 337–346.

[65] ——, "Linear and geometric mixtures-analysis," in *2013 Data Compression Conference*. IEEE, 2013, pp. 301–310.

[66] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th international conference on machine learning (icml-03)*, 2003, pp. 928–936.

[67] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[68] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[70] J. Jiang, L. Zheng, F. Luo, and Z. Zhang, "Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation," *arXiv preprint arXiv:1806.01054*, 2018.

[71] S. Bang, S. Park, H. Kim, and H. Kim, "Encoder–decoder network for pixel-level road crack detection in black-box images," *Computer-Aided Civil and Infrastructure Engineering*, vol. 34, no. 8, pp. 713–727, 2019.

[72] Z. Gu, J. Cheng, H. Fu, K. Zhou, H. Hao, Y. Zhao, T. Zhang, S. Gao, and J. Liu, "Ce-net: Context encoder network for 2d medical image segmentation," *IEEE transactions on medical imaging*, vol. 38, no. 10, pp. 2281–2292, 2019.

[73] H. Phan, "Pytorch cifar10," https://github.com/huyvnphan/PyTorch_CIFAR10, 2021.

[74] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

# Acknowledgments