Towards Real-World Data Streams for Deep Continual Learning

Scuola Normale Superiore
University of Pisa - Computer Science Department
Pisa, Italy

Candidate

Andrea Cossu

Supervisors

Prof. Davide Bacciu
Dr. Vincenzo Lomonaco
Prof. Anna Monreale

Thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy (Ph.D.) in Data Science (XXXV cycle)

April 13, 2023

Towards Real-World Data Streams for Deep Continual Learning

Ph.D. Thesis — XXXV cycle — Scuola Normale Superiore, University of Pisa

Candidate

Andrea Cossu

Supervisors

Prof. Davide Bacciu
Dr. Vincenzo Lomonaco
Prof. Anna Monreale

Abstract

Continual Learning deals with Artificial Intelligent agents striving to learn from an ever-ending stream of data. Recently, Deep Continual Learning focused on the design of new strategies to endow Artificial Neural Networks with the ability to learn continuously without forgetting previous knowledge. In fact, the learning process of any Artificial Neural Network model is well-known to lack the sufficient stability to preserve existing knowledge when learning new information. This phenomenon, called catastrophic forgetting or simply forgetting, is considered one of the main obstacles for the design of effective Continual Learning agents. However, existing strategies designed to mitigate forgetting have been evaluated on a restricted set of Continual Learning scenarios. The most used one is, by far, the Class-Incremental scenario applied on object detection tasks. Even though it drove interest in Continual Learning, Class-Incremental scenarios strongly constraint the properties of the data stream, thus limiting its ability to model real-world environments.

The core of this thesis concerns the introduction of three Continual Learning data streams, whose design is centered around specific real-world environments properties. First, we propose the Class-Incremental with Repetition scenario, which builds a data stream including both the introduction of new concepts and the repetition of previous ones. Repetition is naturally present in many environments and it constitutes an important source of information. Second, we formalize the Continual Pre-Training scenario, which leverages a data stream of unstructured knowledge to keep a pre-trained model updated over time. One important objective of this scenario is to study how to continuously build general, robust representations that does not strongly depend on the specific task to be solved. This is a fundamental property of real-world agents, which build cross-task knowledge and then adapts it to specific needs. Third, we study Continual Learning scenarios where data streams are composed by temporally-correlated data. Temporal correlation is ubiquitous and lies at the foundation of most environments we, as humans, experience during our life. We leverage Recurrent Neural Networks as our main model, due to their intrinsic ability to model temporal correlations. We discovered that, when applied to recurrent models, Continual Learning strategies behave in an unexpected manner. This highlights the limits of the current experimental validation, mostly focused on Computer Vision tasks.

Ultimately, the introduction of new data streams contributed to deepen our understanding of how Artificial Neural Networks learn continuously. We discover that forgetting strongly depends on the properties of the data stream and we observed large changes from one data stream to another. Moreover, when forgetting is mild, we were able to effectively mitigate it with simple strategies, or even without any specific ones. Loosening the focus on forgetting allows us to turn our attention to other interesting problems, outlined in this thesis, like (i) separation between continual representation learning and quick adaptation to novel tasks, (ii) robustness to unbalanced data streams and (iii) ability to continuously learn temporal correlations. These objectives currently defy existing strategies and will likely represent the next challenge for Continual Learning research.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I need to thank Davide. Having you as a supervisor made my PhD much easier. I have to acknowledge that your ability to instantly answer emails is slightly degrading over time. Nowadays, people may even need to wait minutes for a reply. Although I will never be able to understand how you can always find some more time, I am grateful you do. Thank you for putting together this amazing lab, for your generosity and for constantly providing support in times of need.
Even though Vincenzo joined the party late, he really changed the game. At first, I was really surprised by how quickly he fit in the Department. Little I knew about the guy! From a small team of 3 members (me, you, Antonio), it is nice to see how quickly we are growing. It is fair to say that the last two years have been quite a ride for the three of us. I could not have done half of the things I did without your support. Thank you! Antonio, yes, you are *almost always* right.

Thanks to all the *UnipiML Disgrazie* people for making the Computer Science Department a place worth working in. Thank you Francesco L. for the quite walks, the chat, the snacks, the vinyl, the... Alright, I should probably buy you something at this point.
I would like to thank my Data Science colleagues, with which I shared (many!) offices. A special mention goes to Valentina and Francesco S., for saving me a *panettone*. Three months and it was still soft. Francesca, do not worry, I am also double-checking every time I need to send a CV, it is normal.

I had the opportunity to meet a whole lot of other amazing people in these years. The ContinualAI crew is on to some new adventure and only the future will tell how that will turn out. We were even able to meet each other in person before the end of my PhD, so there is that.
Even though I was not able to follow along, thank you to Nicola and Peppa for keeping me informed about the latest developments in physics. I still do not know what you are talking about.

Outside university, we find family and friends. Family is always first, hence
grazie ai miei genitori per il supporto che mi hanno dato in tutti questi anni. Per quanto gli ultimi siano stati, credo, più facili dal loro punto di vista, immagino anche di aver dato loro un bel da fare in quelli precedenti. Spero siano soddisfatti del risultato, quanto io lo sono di loro.
A warm thank you to Anna and Paolo. It is amazing how far we have gone together, 11 years and counting! "Choosing" KlinK in high-school has been one of the luckiest things ever happened to me.

A heartfelt thank you to "La Birra del Baggia". Even though the undisputed leader is

climbing around the mountains, we are still able (and allowed) to meet every now and then. Thanks to Gabriele and Tommaso for accepting my invitation. I look forward to one hell of a party. And then we will see what happens.

And thanks to Virginia. But that is another book, and we are still writing it.

Chapter 1

Introduction

Artificial Intelligence (AI) (Russell and Norvig 2009) promises to build intelligent agents able to learn and acquire a sense of understanding and meaning with which face the main challenges of the real world. Research in AI has made a leap forward since the adoption of a paradigm called Machine Learning (ML) (Mitchell 1997). The idea behind ML is to train agents (or models) that learn directly from data, with only a minimal amount of human-crafted prior knowledge. Both the availability of a large amount of data and the possibility to process it efficiently have been key enablers that largely contributed to the success of ML, now at the heart of many technologies (LeCun, Yoshua Bengio, et al. 2015). Crucially, ML agents *need* to observe as much data as possible multiple times in order to learn how to respond when observing similar, but unseen, examples. As an example of this behavior, let us consider the task of recognizing classes of objects from images. In order to learn this task, an ML agent requires a, possibly large, dataset of images containing the object classes to be recognized. During the training phase, the agent goes through all the images multiple times in order to learn the appropriate response for each example in the dataset. At inference time, the agent will be prompted with a new image and it should be able to predict the correct object class. One important property of the iterative training process is that the dataset needs to be entirely available *in advance*, before the agent starts to learn. This requirement may seem an unnecessarily, strict one. However, when looking more in depth at the behavior of the agent, it appears clear that it is actually fundamental. What happens, in fact, when *new information* becomes available over time? What if the new information contains completely novel concepts with respect to the ones present in the original data? How will the agent cope with new information?
These questions lie at the heart of Continual Learning.

Continual Learning (CL) (Parisi, Kemker, et al. 2019; Timothee Lesort, Lomonaco, et al. 2020) requires an agent to acquire knowledge *over the entire span of its lifetime* and to incrementally build upon existing information to expand its current abilities. Unfortunately, ML agents are not suitable to learn continuously since, when new data is added over time, they often suffer from *catastrophic forgetting* of previous knowledge (French 1991; Ratcliff 1990; McCloskey and Cohen 1989). Interestingly, forgetting cannot be observed within the static setting of a typical ML framework. In fact, in ML there is no distinction between *novel* and *past* information. All the knowledge is already available from the beginning, within the dataset. How could an agent forget, if it never sees new information? Up to now, the effectiveness of ML pushed to adapt the real world to the ML needs. As a result, the agents are kept as isolated as possible from any source of novel information. They only deal with the task they were designed to solve. CL, instead, targets the opposite approach. It designs

agents capable to operate in dynamic environments where data changes over time. Under these conditions, ML agents often fail to provide a robust performance across *all* the tasks they have seen during their lifetime. Yet, designing a CL agent remains very challenging. When considering practical applications, the predictive accuracy is considered the most important metric to evaluate the performance of an agent, be it a CL or an ML one. Overall, the predictive performance of CL is still behind the one of ML. This is mainly due to the additional challenges introduced when learning in a dynamic environment. On one hand, forgetting of previous knowledge reduces the performance on previous tasks. On the other hand, adapting to new tasks when only part of the information is available can limit the performance.
Ultimately, whenever an agent needs to acquire new concepts, the most effective solution in terms of predictive performance consists in "rebooting" it altogether. The reboot process first adds the new data in the original set of examples and then proceeds to re-train the agent *from scratch* with the aggregated dataset. Re-training solves the two aforementioned issues. Forgetting is prevented since the agent learns in a setup with a fixed, static dataset. Learning new tasks is also easier, since all the information needed has been gathered and included in the dataset. Once again, the effectiveness of ML favors circumventing the problem of how to acquire new knowledge over time, instead of attacking it directly.
Nonetheless, a successful CL agent would still provide a certain number of advantages with respect to an ML one. Advantages which, in our opinion, fully justify the research activity behind CL:

1. CL provides an ever-updated system able to incorporate new knowledge as soon as it is available. In order to acquire new information, it is not necessary to wait until a large amount of data is collected. Each new example can progressively contribute to the creation of a new knowledge base exploited by the agent. On the contrary, ML needs to acquire large amount of data and to jointly train the model with old and new data together. In the meantime, the system runs an out-dated predictive model;

2. since training does not require multiple passes on very large datasets, a CL agent can be trained on the edge with low-power devices. This is not a strict requirement, since CL can be implemented also in large high-performance computing infrastructures. However, the possibility of training an agent directly where data is collected is a huge advantage, for example for Internet of Things applications (S. Li et al. 2015). Data never leaves the device, thus guaranteeing *data-privacy* (Spiekermann 2012) by design. This is an opportunity for environments, like hospitals and medical facilities, where data cannot leave the building where it is originally stored;

3. CL contributes to an energetically sustainable way of designing AI agents. Especially for large ML models, the ones that also deliver the best performance, re-training is a very expensive operation. It requires many computing cycles, performed on parallel on a large number of computational resources (CPUs, GPUs). The result is that training consumes a non-negligible amount of CO2. The cost needs to be payed each time the model requires to be updated. CL would instead perform a sequence of smaller updates with less data, thus dramatically reducing the total computational and energetic cost.

Despite the nice properties of a successful CL agent, it is undoubtedly true that the current state of the art in CL does not allow to match the predictive performance of ML. Closing this gap will become fundamental for the widespread adoption of CL.
This thesis builds on the observation that *the current evaluation system in CL is strongly biased towards few, very specific scenarios and data streams.* In our opinion, it is now time to study the behavior of CL agents outside these few scenarios, a fundamental step to bring

the CL field to maturity. While the existing scenarios still provide a useful test-bed, they also return results with a correspondingly limited scope and validity. We should instead aim for a more complete understanding of what does it mean to learn continuously. Otherwise, the main risk is to erroneously extrapolate results obtained in one specific environment to other ones, where the actual performance may be very different.
A paradigmatic example, particularly relevant for this thesis, is the Class-Incremental CL scenario (Rebuffi et al. 2017; van de Ven and Tolias 2018b). In a Class-Incremental scenario, data is partitioned into multiple sets of non-overlapping classes. Each set is presented one at a time to the CL agent, which never sees previous sets twice. Class-Incremental represents a valid CL environment, since data arrives in a stream and it is not entirely present at the beginning. However, it also represents a very specific case which, alone, cannot be taken as representative of all the CL challenges. One of the main issues lies in the constraint that the model can never revisit previous classes. This constraint exacerbates the problem of forgetting, since the model cannot consolidate its knowledge about previous classes over time. As a consequence, the model is forced to learn everything it can while data is available, and to remember it afterwards without any reinforcement. Real-world data streams, instead, often allow previous concepts to reappear again in the future, making Class-Incremental somewhat an artificial scenarios for many applications.
Still, the vast majority of CL works are conducted on Class-Incremental scenarios, implemented on top of Computer Vision and object detection tasks. It is important to realize that, with respect to the number of possible scenarios and domains, CL literature is currently focusing on an incredibly limited subset.

This thesis addresses the issue by exploring two research directions, strongly related to each other: i) the design of new CL environments inspired by requirements coming from real-world needs and ii) the study of Sequential Data Processing domains with Recurrent Neural Networks.
Both research directions need to first acknowledge the limits and risks of an excessive focus on Class-Incremental scenarios, before proposing some alternatives. The idea is to model real-world data streams that do not currently fit the existing CL scenarios. Each data stream will have specific properties that highly impact on the performance and the design of CL strategies. While we defer a full description of such real-world data streams to later chapters, here we will give a brief introduction to their main characteristics.

In the second Part of the thesis, we introduce the Class-Incremental with Repetition and Continual Pre-Training environments. As the name suggests, Class-Incremental with Repetition creates a data stream in which previous concepts can be repeated, at a varying degree of frequency. Repetition is naturally present in many real-world environments and it can be considered as a powerful source to convey information to the agent.
The Continual Pre-Training environment, instead, is based on the popular *pre-train, fine-tune* setup (Section 2.1 in the Background), where a model acquires knowledge from a large source of data before being adapted to specific tasks. Continual Pre-Training extends this paradigm to CL: it focuses on the possibility to continuously learn *representations* before leveraging them to solve a specific task. When looking at CL through the lenses of Continual Pre-Training, forgetting becomes a much softer issue which can be tackled without any specific CL solution. Continual Pre-Training introduces a different paradigm in learning continuously, which comes with its own advantages and disadvantages. In general, the two environments we propose and study are not meant to replace any of the alternative environments already present in the literature. Instead, they enrich our understanding of the possibility of a CL agent. They highlight that the property and constraints present in the

dynamic environment are fundamental to determine which solution to adopt and to estimate its expected performance.
Following a different, but complementary line of research, in the third Part of the thesis we studied the behavior of Recurrent Neural Networks in existing CL scenarios. Recurrent Neural Networks (Elman 1990; Hochreiter and Schmidhuber 1997) are ML models which are designed to model temporally-correlated information. Quite surprisingly given the ubiquitous presence of time series in the real-world, Recurrent Neural Networks did not receive much attention from the CL community, even though time appears a fundamental correlate in CL. We were the first to conduct a comprehensive study on CL and Recurrent Neural Networks. Importantly, we chose to leverage existing CL scenarios to study the behavior of recurrent models. On one side, this allowed us to create a causal link between the phenomenon we discovered and the introduction of recurrent models. Using previously unexplored scenarios would have required to disentangle the contribution of the model from the one of the environment. On the other side, however, the limitations previously discussed regarding the scope of the evaluation apply as well to the experiments of this Part. To mitigate this effect, our analyses are conducted on two different CL scenarios, such that we are not limited to Class-Incremental only. Moreover, we adapted popular benchmarks to Sequential Data Processing and we also introduced novel ones from domains like Human State Monitoring and Speech Recognition, previously unexplored in CL. The results we obtained validated our previous statement on the importance of the data stream and environment properties. We also obtained a more complete understanding of the factors at play and on whether or not they impact on the final performance.
We discover a peculiar phenomenon, unique to recurrent models and connected to forgetting, which we called the Sequence Length Issue. We observed that when data is temporally correlated, working with recurrent models induces an amount of forgetting which grows with the length of the input sequence. Simply put, the longer the temporal correlation needed to be remembered by a Recurrent Neural Network, the larger the forgetting. This phenomenon affects a large number of CL strategies in different ways, depending on the family they belong to. Luckily, the problem can be attacked in different ways, which we explore in the final Chapter.
Ultimately, the common objective of this thesis is to gain a deeper and more complete understanding of the behavior of CL approaches by focusing on novel environments and data streams. Since CL does not provide a one-size-fits-all solution, it is of paramount importance to understand when a certain strategy may be useful and when, instead, it would be better to opt for a different one. Also, by exploiting the understanding gained in the novel environments we studied, it is possible to devise specific approaches which deliver a better performance than other strategies (Section 6.4.4 for Class-Incremental with Repetition and Section 11.1 for recurrent models).

1.1 Thesis Outline and Contribution

We now detail the structure of the thesis and briefly discuss the main contributions of each Chapter. The thesis is composed by this Introduction, three main Parts and the final Conclusion.
Part 1 covers the background needed to understand the research described in the other two Parts. The background can be safely skipped if the reader is already familiar with the Machine Learning and Continual Learning field. However, we recommend to survey the beginning of Chapter 4, and in particular Section 4.1, where we put forth a CL nomenclature which is adopted throughout the rest of the thesis. The first Part is structured in three

Chapters:

- Machine Learning (Chapter 2): the Chapter covers the main concepts related to learning with iid data. The Chapter presents a set of common ML tasks and the main characteristics of the learning process with iterative optimization methods. It then presents the Artificial Neural Network, the ML model used in all the experiments of Part 2 and Part 3. The Chapter concludes with a brief presentation of Deep Learning and the main families of Artificial Neural Networks developed in the field.
- Non-stationary Environments (Chapter 3): the Chapter describes the properties of learning environments in which the input data distribution is subjected to drifts and may change over time. After covering the main types of drift, the Chapter moves on to describe the main approaches present in the literature to learn in non-stationary environments. In particular, it focuses on techniques to actively detect new drifts and on techniques to effectively adapt the learning model when the drift is detected. Finally, it presents the problem of *catastrophic forgetting*: the degradation of a model performance on old data occurring after acquiring new information.
- Continual Learning (Chapter 4): the Chapter formally describes the characteristics of a Continual Learning agent and highlights the constraints present in non-stationary CL environments where the agent learns. The Chapter formalizes the nomenclature used in the rest of the thesis and presents the main component of a Continual Learning experiment: the type of scenario, the benchmarks, the evaluation protocols and metrics, and the strategies.

Part 2 introduces novel CL scenarios developed over the course of this thesis and which go beyond the Class-Incremental scenario. Part 2 is composed of three Chapters:

- Motivation (Chapter 5): it describes the reasons behind the need of novel CL scenarios. The Chapter describes the limitations and risks due to the adoption of only few, and very similar, CL scenarios. The specific characteristics of the Class-Incremental scenario highly influence the type of CL strategies developed so far, which are mainly focused on forgetting. The evaluation in new scenarios shows that forgetting is not always *catastrophic* and that other CL objectives like forward transfer and knowledge accumulation are as fundamental as forgetting when the properties of the scenario change.
- Class-Incremental with Repetition (Chapter 6), which we published in Hemati et al. 2023: the Chapter introduces a CL scenario in which, unlike Class-Incremental, objects belonging to previously seen classes can reappear later on in time. This is the case in many real-world applications, where a concept which is never revisited is probably a concept not worth remembering. Compatibly with the definition of CL, the repetition of previous concepts present in this novel scenario cannot be controlled by the CL agent. The agent only sees a stream of incoming data, without being able to tweak the amount of repetition, which is a property of the non-stationary environment. Therefore, repetition can be unbalanced, can affect only specific classes and can also change over time. Existing CL strategies mainly target scenarios in which there is no repetition at all (Class-Incremental). In Class-Incremental with Repetition scenarios, such strategies might have a reduced impact. In fact, forgetting is partially mitigated by the environment itself.
- Continual Pre-Training (Chapter 7), which we published in Cossu, Tuytelaars, et al. 2022: the Chapter presents a scenario which decouples the objective of *representation learning* from the one of *adaptation to a given task*. In existing CL scenarios instead,

a model needs to tackle both objectives at the same time. This makes the learning problem much more difficult. For example, when learning simple tasks, the model does not need to learn general features to solve them. Therefore, when the task changes, the features previously learned do not generalize and they will be overwritten by the ones needed for the new task. This is a strong source of forgetting in CL. In the Continual Pre-Training scenario, the model is continuously pre-trained on a stream of data in order to acquire robust representations. Since pre-training is designed to model relationships present in the input data, it does not over-fit on a predictive task. At any point in time during continual pre-training, the model can be fine-tuned by training it on a specific task. The Continual Pre-Training scenario is able to largely mitigate forgetting without any CL strategy, simply by separating the concerns of representation learning and task adaptation.

Part 3 describes our original contribution on CL with temporally-correlated data and Recurrent Neural Networks. Part 3 is composed by 4 Chapters:

- Motivation (Chapter 8): the Chapter describes the reasons behind the study of CL for sequential data processing. It identifies Recurrent Neural Networks as an ideal candidate to study the effect of temporal correlation in non-stationary environments.

- Review of Continual Learning with Recurrent Neural Networks (Chapter 9), which we published in Cossu, Carta, Lomonaco, et al. 2021: the Chapter provides a taxonomy of existing CL approaches dealing with temporally correlated data. The review also focuses on related benchmarks present in the literature, showing that researchers have not yet converged on a set of standard ones.

- The Sequence Length Issue (Chapter 10), which we published in Cossu, Carta, Lomonaco, et al. 2021: the Chapter presents an empirical evaluation of the behavior of Recurrent Neural Networks trained with popular CL strategies. Importantly, the CL strategies were not originally designed to deal with sequential data and recurrent models. The empirical analysis introduces two general-purpose benchmarks which, unlike many of the ones present in the literature, are not tied to a specific application. The results of the empirical evaluation highlight the Sequence Length Issue, common to all strategies: when the input sequence length increases, the amount of forgetting experienced by the model increases as well. The Sequence Length Issue is a phenomenon associated to temporal correlation and recurrent models, not to the specific CL strategy. However, different CL strategies suffer from this problem at different levels. The Sequence Length Issue prevents a blind application of existing CL strategies to sequential data processing problems.

- Overcoming the Sequence Length Issue (Chapter 11), which we published in Cossu, Carta, and Bacciu 2020; Cossu, Bacciu, et al. 2021; Matteoni et al. 2022: the Chapter addresses the Sequence Length Issue by exploring a number of ways with which it can be tackled. The Gated Incremental Memories (Cossu, Carta, and Bacciu 2020) presented in Section 11.1 represent a novel CL strategy that dynamically expands a RNN in order to allocate separate modules for different tasks. The strategy employs a set of recurrent autoencoders to automatically recognize which module use for inference. The performance of Gated Incremental Memories strongly depends on the autoencoder performance, but it frees the model from the Sequence Length Issue. Section 11.2 presents an alternative solution to the Sequence Length Issue: a randomized recurrent model called Echo State Network (Lukoševičius and Jaeger 2009; Gallicchio et al. 2017). Being composed by a random, fixed set of recurrent connections, the Echo State

Network is able to learn continuously without being affected by the Sequence Length Issue (Cossu, Bacciu, et al. 2021). By considering the fixed set of recurrent connections as a sort of pre-trained model, Echo State Networks can also exploit CL strategies specifically designed for pre-trained models, which are often among the most effective ones. Finally, Section 11.3 introduces a practical application for CL with RNNs based on Human State Monitoring tasks (Matteoni et al. 2022) (classification of actions from time-series produced by wearable sensors). We designed a data stream in which new subjects are introduced over time. In this case, the scenario presents a softer drift than other scenarios in which new classes are introduced over time. Therefore, the Sequence Length Issue is less prominent and leaves space for the study of additional objectives like forward transfer and knowledge accumulation.

Chapter 12 concludes the thesis by summarizing the main results and by discussing possible avenues for future research.

Appendix C presents the Avalanche library, which was published in Lomonaco, Pellegrini, et al. 2021. The author of this thesis is one of the maintainer of the library and co-author of the paper.

Avalanche is supported by the non-profit organization ContinualAI[1] and it is currently the leading library for Continual Learning. It is also part of the PyTorch ecosystem and it is publicly available as an open-source project on GitHub[2].

[1] https://www.continualai.org

[2] https://github.com/ContinualAI/avalanche

Part I

Background

Chapter 2

Machine Learning

Learning to achieve a desired objective from data alone is a powerful approach that has enhanced the abilities of Artificial Intelligent (AI) agents (Russell and Norvig 2009) and has allowed them to tackle previously inaccessible tasks. Machine Learning (ML) is a field of AI which focuses on this data-based way of solving a task. A popular definition of what ML is about, provided by Mitchell 1997, states that:

Definition 1. *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 by P, improves with experience E. Mitchell 1997.*

Def. 1 provides an operational characterization of the concept of learning by focusing on the expected behavior a successful ML agent should exhibit. Learning occurs whenever the acquisition of new experience E, that is new *information* or *data*, leads to an increase in the performance P achieved by the agent (or computer program as per the original definition by Mitchell 1997). Compatibly with the *garbage-in garbage-out* principle, feeding the agent with data unrelated to tasks in T does not constitute proper learning. In fact, the performance on such tasks is not expected to improve. On the other hand, learning is expected to continue as long as the agent is provided with new meaningful information related to the tasks in T. Recent results in Deep Learning (Section 2.4) have shown the power of such data-centric view of learning, leading to agents which are able to extract information from gigantic datasets and whose performance improves as the size of these datasets becomes larger and larger (LeCun, Yoshua Bengio, et al. 2015).
Def. 1 also constraints the learning process to consider a set of well-defined tasks and to measure the performance of the agent against the tasks present in this set. While this appears reasonable and almost tautological, it hides some important questions about the learning process, which are at the core of this thesis. Can an agent learn a new task not initially present in T? What happens to the agent's overall performance when learning a new task? As we will see in later chapters, questions of this sort have been posed from the very beginning of ML and they still represent an ambitious challenge. Before delving deeper into this matter, we need to formalize more precisely the way a ML agent learns.

2.1 Different Kinds of ML Tasks

Depending on the nature of the data the ML agent is exposed to during learning, we can define different kinds of ML tasks.
In *supervised* tasks, a dataset is composed of N pairs $\{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1,\dots,N}$, where $\boldsymbol{x_i}$ represents the information the agent is expected to leverage in order to predict the final target output

$\boldsymbol{y_i}$. For example, an object recognition task requires the agent to recognize which object class, out of a fixed set of classes, is present in a given image. This is an instance of a *classification* task. The input received by the agent is a vector $\boldsymbol{x_i}$ containing the pixel intensity values of each pixel in the image. Each dimension of the input vector is usually called a *feature*. The associated target output $\boldsymbol{y_i}$ is the unique code associated with the object class present in the image. In the case of classification, the target can be represented as a one-dimensional vector with as many elements as the number of possible object classes. A given target will have zero in each entry, except for the entry corresponding to the object present in the image, which will contain a 1. Each object/class is therefore assigned an integer number: if class *car* is assigned the code 4, all images containing a car will have an associated target vector with zero in every entry except the fifth - assuming zero-based indexing - which will contain 1. This *one-hot* encoding is commonly used for classification tasks. For the sake of notation, sometimes the target for classification is indicated with a scalar y, assuming that the appropriate conversion to one-hot encoding is performed when needed. A different, and simpler, case of target encoding can be presented for regression task. In regression, the ML model learns to predict one or more scalar values from the input. The target vector will be the vector containing all such values. For example, an ML model trained to predict apartment rental prices will receive as input a vector of features representing the apartment (area in squared meters, number of rooms and so on) and will predict the price in a specific currency. Since the target is already in a numerical form, no conversion is needed.

ML solutions based on supervised learning tasks are often very effective, provided that a large amount of data is available. However, in many applications collecting a target for each sample is costly and can become prohibitive for a very large number of samples. Given these difficulties, learning to perform a task with a low level of supervision (i.e., with few annotated samples) constitutes an important objective for ML. To this end, unsupervised learning tasks do not provide *any* form of supervision to the agent, which only sees the input $\boldsymbol{x_i}$. Unsupervised learning often requires to partition the input space into separate groups, called *clusters*. Learning how to cluster the input space is very useful, since during the process the model acquires useful knowledge about the intrinsic properties of the data. However, a fully unsupervised learning process is unable to perform specific tasks, like classification or regression, due to the absence of targets. Nowadays, unsupervised learning is used mainly for generative tasks or as a proxy to learn meaningful representations without any target information. This latter stage is called *pre-training* and the associated task to be learned is called the pre-train (or proxy) task. The pre-trained model obtained after solving the pre-training task is then taken as the starting point for a follow-up supervised learning phase, called *fine-tuning*. The task associated with fine-tuning is sometimes referred to as *downstream* task. Compared to a supervised learning phase conducted directly on the downstream task, the *pre-train fine-tune* setup usually obtains a better performance given the same amount of annotated labels. However, it also requires a large set of unlabeled data, which may be costly to process during learning. The trade-off between computational resources and availability of annotated samples provides a good indicator of which setup is more appropriate. One of the most used types of pre-training tasks consists in masking out parts of the input and learn to predict the masked content given the unmasked one (Devlin et al. 2019b). In computer vision applications, for example, this approach is called *masked image modeling* (Bao et al. 2021) and it is performed by masking out some pixels of an image and learning to predict the value of the masked pixels given the rest of the image. Similarly, when dealing with natural language, *masked language modeling* is performed by masking out some words in a sentence and learning to predict the masked words given the rest of the sentence (Devlin et al. 2019a; Liu et al. 2019). Since the targets are obtained from the input

itself (or parts of it), this type of pre-training is called *self-supervised* pre-training (Misra and Maaten 2020). Fine-tuning is often confined to the same domain encountered during pre-training. For example, an agent pre-trained on a dataset of images can be fine-tuned on some image-based tasks, like object recognition.
Although quite different from the kind of tasks we have discussed so far, *Reinforcement Learning* (RL) (Sutton and Barto 2018) is another popular ML setting. RL tasks are characterized by a sparse supervisory signal called *reward*. Learning takes place in an environment where the agent repeatedly takes an action, observes the result of its own action and, possibly, receives a reward. The reward tells the agent the value of being in the current state and taking that action. Based on this information, which is not available for all states and actions, the agent needs to learn how to maximize the expected future reward. The overall objective of reward maximization is usually addressed by dividing the problem into different sub-problems (e.g., learning which action to predict next, learning to estimate the value of a state). Supervised and unsupervised learning are tightly integrated with RL, since they are often used to solve many of these sub-problems. RL has been successfully applied to many different problems. Some of the most famous ones are the mastery of games like Atari (Mnih et al. 2013), Chess and Go (Silver et al. 2016) and Dota 2 (OpenAI et al. 2019). In fact, all these problems can be easily modeled with an RL environment. In the case of chess, the agent's actions correspond to legit moves of a piece. The agent's perceived state is the current board configuration. The reward is given only at the end of the game and takes on a positive value in case of a win, a negative value in case of a defeat and zero in case of a draw. The reward is the only external supervisory signal received by the agent, together with the environment state which is usually considered an input. Despite the high degree of reward sparsity, RL agents are still able to succeed in many different tasks.

2.2 Empirical Risk Minimization

The objective of a ML model is to learn how to tackle a specific task, be it supervised or of any other kind. While Def. 1 gives a procedure to check whether a model is learning or not, it gives no information regarding how the learning process should actually be implemented. For simplicity, in this section we will only consider the supervised learning setup. A similar line of reasoning can be adopted for unsupervised learning. We will leave aside learning in RL environments, since RL falls outside the scope of this thesis.
Let us consider a parametric ML model $h_{\boldsymbol{\theta}}$ with parameters $\theta \in \boldsymbol{\theta} \subset \mathbb{R}^d$. The model receives an input $\boldsymbol{x}$ and outputs its predictions $h_{\boldsymbol{\theta}}(\boldsymbol{x})$. In ML, learning a task means predicting the correct target $\boldsymbol{y}$ for each input $\boldsymbol{x}$, that is, $h_{\boldsymbol{\theta}}(\boldsymbol{x_i}) = \hat{\boldsymbol{y}}_{\boldsymbol{i}} = y_i,\ \forall i = 1, \ldots, N$. We can cast this problem into an optimization problem that seeks to find the optimal parameters $\boldsymbol{\theta}^*$ (the optimal model) that minimize a given *loss* function $\mathcal{L}$ over all the inputs $(\boldsymbol{x}, \boldsymbol{y})$ in the joint input-target space $(\mathcal{X}, \mathcal{Y})$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \int_{(\mathcal{X},\mathcal{Y})} \mathcal{L}(h_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})\ d\boldsymbol{x}d\boldsymbol{y}\ . \tag{2.1}$$

The loss function $\mathcal{L}$ quantifies the cost we incur when the model predicts $\hat{\boldsymbol{y}}$ and the correct target is $\boldsymbol{y}$. In case of a correct prediction ($\hat{\boldsymbol{y}} = \boldsymbol{y}$) the loss will be zero, while it will be a positive value otherwise. Unfortunately, evaluating the integral in Eq. 2.1 would require to be able to sample $\boldsymbol{x}$ from the true data distribution $p(\boldsymbol{x}, \boldsymbol{y})$ spanning the whole space $(\mathcal{X}, \mathcal{Y})$. The true data distribution is however not available in all interesting cases. The Empirical Risk Minimization (ERM) framework side-steps the problem by approximating the true input distribution $p(\boldsymbol{x}, \boldsymbol{y})$ with the empirical distribution defined by the available

dataset $D = \{(\boldsymbol{x_i}, \boldsymbol{y_i}),\ i = 1, \ldots, N\}$. The ERM requires the samples in the dataset to be *independent and identically distributed (iid)*. Therefore, each sample is independent from all the others and all the samples are generated by the unknown data distribution $p(\boldsymbol{x}, \boldsymbol{y})$. The optimization problem expressed by Eq. 2.1 can be reformulated in:

$$\boldsymbol{\theta^*} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \mathcal{L}(h_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) \ . \tag{2.2}$$

The ML literature provides different types of loss functions. Two of the simplest and most used ones are the Mean Squared Error (MSE) and the Cross Entropy (CE) losses. The MSE is used in regression tasks. It measures the Euclidean distance between the model prediction and the correct target:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y_i}\|_2^2. \tag{2.3}$$

The CE is used in classification tasks with one-hot encoded targets. It is defined as

$$\mathcal{L}_{\text{CE}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y_i} \log \hat{\boldsymbol{y}}_{\boldsymbol{i}}. \tag{2.4}$$

Depending on the type of ML model h, the optimization problem defined by Eq. 2.2 can be solved in closed form (e.g., when h is a linear model) or with other optimization techniques like gradient descent (when h expresses a differentiable function).

2.2.1 Over-fitting and Generalization

The ERM framework provides an effective way to train a model without needing to access the true input distribution $p(\boldsymbol{x}, \boldsymbol{y})$. However, the goal of ML is still to build models able to correctly predict samples coming from the true distribution of the task. Therefore, once the training phase ends, the model needs to be evaluated in terms of some metric P on *unseen* data (e.g. classification accuracy). Since the dataset is supposed to approximate the true input distribution, reserving part of the data for the final evaluation allows to provide a fair assessment of the model performance. In practice, the original dataset D is split into two non-overlapping sets D^{train} and D^{test}. The dataset D^{train} is used to solve the optimization problem in Eq. 2.2 (model training), while the dataset D^{test} is used to compute the final performance of the trained model (*model assessment*). During model training, the model parameters are updated repeatedly, while during model assessment the model is kept fixed and used only to compute the metric P. The value of P computed after model assessment measures the *generalization* capability of the model. A model is able to generalize when it correctly predicts the target answers for unseen samples coming from the same distribution of training samples.

If the model assessment returns a low performance, there are usually two cases: the model is either *under-fitting* or *over-fitting*. In an *under-fitting* regime, the model performance P is low not only when computed on test data but also when computed on training data. This points to the fact that the model is not able to fit the training data due to lack of expressiveness (e.g., not enough parameters for complex tasks) or a poor optimization method. We say that the model has a large *bias*, since it cannot capture the input-output relationship. In an *over-fitting* regime, instead, the performance on training data is significantly larger than the one on test data. This often happens when the model is too expressive for the task at hand (it has many parameters), especially when combined with few training samples. The model does not need to learn useful representations for the input, since it can simply memorize each input together with the correct answer. We say that the model has a large *variance*, since it

changes its behavior even in the presence of minima fluctuations in the input. According to Def. 1, a model which completely over-fits is not learning, since providing new data would not improve the performance on the overall task T, but just on the seen samples. While avoiding under-fitting often amounts to find a more expressive model or properly tune the optimization method, avoiding over-fitting is more complicated. Popular approaches include adding more data or restricting the flexibility of the model by adding terms to the loss function (*regularization*). Although effective, none of these approaches guarantee beforehand the absence of over-fitting. Running multiple trials with different configurations (e.g., different optimization methods or different models) is often essential to find a good model. All the properties deemed important for the final performance and that require careful tuning are called *hyper-parameters*. Examples of hyper-parameters are the optimizer type, the type of ML model, the loss function and many others. The *model selection* phase estimates the best values for the selected hyper-parameters. Model selection requires an additional data split D^{valid}, called validation set. While model assessment outputs the performance of a model on unseen data, model selection performs model training with multiple hyper-parameters configurations and returns the best configuration in terms of performance P on the validation set. That configuration is then used to train the model before performing model assessment. Other, more advanced form of model assessment are used in the ML literature (like K-fold cross validation) but are not essential to the purposes of this thesis. ML is more of an empirical science than an exact one. While rule of thumbs to train ML models do exist, there is no general rule able to effectively provide a training recipe to build models with good generalization capabilities.
In Chapter 4, we will discuss the challenges that arise when applying model selection to Continual Learning. The robust and well-defined framework of model selection we just introduced will need further adjustments due to the non-stationary and dynamic nature of the Continual Learning environments. More, the idea of *model selection* itself will be questioned. While there is no definitive answer yet, we will present the available options and discuss their merits and caveats.

2.3 Artificial Neural Networks

Although there exists a wide variety of ML models (Decision Trees, Support Vector Machines, etc...), here we will present a family of models called Artificial Neural Networks (ANNs), or Neural Networks (NNs) for short. NNs are one of the most popular ML models and are extensively used throughout this thesis. Let us first focus on Feed-forward Neural Networks, also called Multi Layer Perceptrons (MLPs), which are the simplest type of NN. MLPs are widespread in ML due to their expressiveness power which allows them to fit any continuous function with an arbitrary degree of precision (the Universal Approximation Theorem). MLPs are usually over-parameterized with respect to the difficulty of the task. This allows to avoid the under-fitting regime and only focus on the mitigation of over-fitting (Sec. 2.2.1). An MLP is built out of few simple components which are combined together to give rise to different MLP architectures. The basic building block is the *unit*, a specific variant of which is called Perceptron (Mcculloch and Pitts 1943). A unit simply computes a weighted sum of its input.

Definition 2. *Given a* weights *matrix* $\boldsymbol{W} \in \mathbb{R}^I$*, a* bias $b \in \mathbb{R}$ *and an input* $\boldsymbol{x} \in \mathbb{R}^I$*, a* unit *computes the final output* $\hat{y} \in \mathbb{R}$ *via* $\hat{y} = \boldsymbol{W}\boldsymbol{x} + b$ *.*

A unit learns by adapting the values of the weights and bias which constitute the unit parameters $\boldsymbol{\theta} = [\boldsymbol{W}; b]$, where $[\,\cdot\,;\,\cdot\,]$ indicates vector concatenation. The term *weights* is

sometimes used to indicate all the model parameters. The unit is a simple linear model which can be trained with closed form solutions or with iterative optimization methods like gradient descent.
A linear *layer* is a set of units, each of which with its own parameters, receiving the same input.

Definition 3. *Given a unit u_i with weights matrix $\boldsymbol{W_i}$, and bias b_i, a K-units layer has weights matrix $\boldsymbol{W} \in \mathbb{R}^{KxI} = [\boldsymbol{W_1}; \boldsymbol{W_2}; \ldots; \boldsymbol{W_K}]$, bias vector $\boldsymbol{b} \in \mathbb{R}^K = [b_1; b_2; \ldots; b_K]$ and produces output vector $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ as $\hat{\boldsymbol{y}} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$.*

While a single unit allows to predict only a single number, a layer is able to predict a vectors of numbers. Item i in the output vector is the prediction made by the corresponding unit i in the layer, independently from the other units.
An MLP is built by adding one linear layer on top of the other. Adding a new layer can be viewed as the composition of a new function (a new layer) with the current one (the existing layers). Since the composition of linear function is still a linear function, the output of each layer is passed through a non-linear point-wise function σ before being fed to the next layer. Popular choices for σ are the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent (tanh) $\sigma(x) = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ or the Rectified Linear Unit (ReLU) $\sigma(x) = \text{relu}(x) = \max(0, x)$. This allows to compose together multiple non-linear functions (layers) giving rise to a more expressive function than the single component alone.

Definition 4. *Given a layer l_i with weights matrix $\boldsymbol{W_i}$ and bias $\boldsymbol{b_i}$, a L-layers MLP computes its final output $\hat{\boldsymbol{y}} \in \mathbb{R}^O$ as $\hat{\boldsymbol{y}} = \boldsymbol{W_L}(\ldots\sigma(\boldsymbol{W_2}\sigma(\boldsymbol{W_1}\boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2})\ldots) + \boldsymbol{b_K} = l_L(\ldots\sigma(l_2(\sigma(l_1(\boldsymbol{x})))))$.*

In case of a classification problem with N classes, the final output layer has N units. Its output $\hat{\boldsymbol{y}}$ is passed through a softmax function with temperature T (usually $T = 1$) to obtain the probabilities for each class. The softmax output for the i-th unit returns is defined by

$$\text{softmax}_i(\hat{\boldsymbol{y}}) = \frac{\hat{y}_i^{\frac{1}{T}}}{\sum_{j=1}^{N} \hat{y}_j^{\frac{1}{T}}}. \tag{2.5}$$

The final classification is performed by predicting the class associated with the unit with the largest probability. Obviously, the dimensions of the weights matrices need to be compatible between each other, in the sense that the weights matrix of the layer i needs to be multiplied by the output vector of the layer $i-1$ (where the first layer receives the input vector $\boldsymbol{x}$). The last layer l_L is usually called the *output layer*, while all the other ones are called *hidden layers*.
One interesting case of NN is the *autoencoder*. An autoencoder is a NN whose output layer has the same size of the input layer. The autoencoder maps the input to a hidden activation and then it projects the hidden activation back to the original input. The difference between the original input sample and the reconstructed one defines the *reconstruction error*:

$$R = \frac{1}{I}\sum_{i=1}^{I}(x_i - \hat{y}_i)^2. \tag{2.6}$$

Autoencoders define unsupervised tasks, where an external target is not needed since it is the input sample itself. Autoencoders can be used to recognize specific input distributions (the ones the autoencoder has been trained on) or to compress samples by using the hidden activation. In fact, during training, the hidden activation of the autoencoder learns to represent all the information needed to reconstruct the original input. If the size of the hidden activation is smaller than the input feature space, then the training process learns to compress the input sample.

2.3.1 Back-propagation and Credit-assignment

Solving the optimization problem of Eq. 2.2 requires to find the optimal parameters $\boldsymbol{\theta}^*$ of the ML model h. In the case of NNs, the optimization algorithm (*optimizer*) of choice is the Stochastic Gradient Descent (SGD). The idea behind SGD is, for each input sample, to adjust the value of all the parameters in the direction of the loss steepest descent. To accomplish that, each parameter receives an update which is proportional to the negative derivative of the loss with respect to that parameter. The update rule for a single parameter θ is therefore of the form $\Delta\theta = -\eta\frac{\partial\mathcal{L}}{\partial\theta}(\hat{\boldsymbol{y}}, \boldsymbol{y})$, where η is called *learning rate* and specifies the step-size of the update compared to the full derivative signal. Putting it together for all the parameters, the update is of the form $\Delta\boldsymbol{\theta} = -\eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$. SGD proceeds one sample at a time. In practice, this approach leads to slow convergence. It is instead more effective to compute the gradient averaged over a small number of samples, called *mini-batch*. The mini-batch version of SGD still drives the optimization process towards a local minimum. The size of the mini-batch is an important hyper-parameter which can be selected via model selection. Given the speed-up in NNs training time provided by modern accelerators, like Graphical Processing Units (GPUs), one rule of thumb is to choose the mini-batch size as larger as possible, provided that the resulting memory occupancy (NN activations and parameters) does not exceed the available memory. Mini-batches are sampled from the dataset without replacement. Once all the samples in the dataset have been selected and used to update the model (i.e., after one *epoch*), the process starts over with a new epoch of training.
The problem of computing the gradient of the loss with respect to the model parameters can be seen as solving a *credit-assignment* problem: each parameter must be assigned a number ($\frac{\partial\mathcal{L}}{\partial\theta}$ for parameter θ) which reflects the impact (positive or negative) that parameter had on the correctness of the final prediction. The *back-propagation* algorithm (Rumelhart et al. 1986) allows to compute the quantity $\frac{\partial\mathcal{L}}{\partial\theta}$ for each parameter of any NN. Computing the gradient of the loss with respect to the parameters of the final output layer is straightforward, given the direct dependence between the parameters and the loss function. However, for the parameters of the hidden layers such computation requires more work. The back-propagation algorithm exploits the network architecture and the connectivity between layers. Having computed the gradient for the final layer, the back-propagation computes the gradient of the previous layer via the chain rule. Since, in the case of an MLP, a layer is connected only to the previous and next one, the gradient information can flow backward, from the output layer to the first hidden layer. The whole process requires a chain of multiplications between the gradient computed for the layer above and the local error computed for the current layer. The back-propagation algorithm is general enough to be compatible with any NN architecture, as long as the functions represented by the layers are differentiable. This *end-to-end* training of NNs allows to efficiently update all the network parameters in an iterative way. We can briefly summarize the steps required to train a NN, which are repeated for each mini-batch and for each epoch:

1. compute the network predictions for the current mini-batch;
2. compute the loss given the network prediction with respect to the target prediction;
3. runs the back-propagation algorithm to compute the derivative of the loss with respect to each parameter;
4. given the gradient, update the parameters with an optimizer of choice such as SGD.

This process is repeated until a stopping condition is met. There are many stopping conditions. For example, training can be stopped after reaching a fixed number of epochs. Alternatively, the training process can reach a local minimum, detected by checking that the loss decreased

less than a fixed threshold in the previous iterations. More, the detection of over-fitting can trigger the so-called *early-stopping*. Since the performance on the validation set is significantly lower than the one on the training set (over-fitting condition), training needs to be stopped. Further training may lead to even worse over-fitting. In the early-stopping condition, the model is restored to the version which performed the best on the validation set during training.

2.4 Deep Learning

Deep Learning (DL) (LeCun, Yoshua Bengio, et al. 2015) has recently emerged as a powerful paradigm able to build hierarchical representations by using deep architectures, that is architectures with many layers. Even if NNs are not the only family of models able to provide deep representations, they are certainly the ones which achieved the greater success. In ML and DL, choosing the proper architecture provides an *inductive bias* which can be useful to learn task with specific structures (images, sequences, graphs...). We will now present three popular DL architectures with different inductive biases: the Convolutional Neural Network (CNN) (LeCun, Bottou, et al. 1998), the Recurrent Neural Network (RNN) (Elman 1990; Hochreiter and Schmidhuber 1997) and the Transformer (Vaswani 2017).

2.4.1 Convolutional Neural Networks

CNNs (LeCun, Bottou, et al. 1998) are a type of NN based on the *convolution* operation. We will consider the case of 2-dimensional CNNs, which are widely used in Computer Vision tasks.

A 2-dimensional convolutional layer defines a kernel matrix (also called filter) $\boldsymbol{k} \in \mathbb{R}^{F_1 \times F_2}$ applied on an input $\boldsymbol{x} \in \mathbb{R}^{W \times H \times C}$, where H, W, C are the image height, width and number of channels in pixels ($C = 1$ for gray-scaled images, $C = 3$ for colored RGB images). As common in DL, we will only consider kernels represented by square kernels, where $F_1 = F_2 = F$. A convolutional layer, applied to each channel separately, returns a feature map $\boldsymbol{s} \in \mathbb{R}^{(W-F+1)\times(H-F+1)\times C}$. Each element $s_{i,j}$ of the feature map is obtained by:

$$s_{i,j} = \sum_{w=1}^{F} \sum_{h=1}^{F} x_{i-w,j-h} k_{w,h}. \tag{2.7}$$

The values $k_{i,j}$ of the kernels are the adaptive weights of the convolutional layer which can be trained via SGD or other optimization methods. To enhance the expressive power of CNNs, the convolutional layer uses more than one kernel at a time. Kernels are independently applied to the input image, resulting in a larger number of channels in the output feature map. Given p kernels, the final number of channels will be Cp.

The convolutional layer returns a feature map tensor which is smaller than the original input tensor. Padding allows to keep the same output dimension as the input one after convolution. Padding expands the input $\boldsymbol{x}$ into a new $\boldsymbol{x'} \in \mathbb{R}^{W' \times H' \times C}$, where $W' > W, H' > H$. The original $\boldsymbol{x}$ is placed at the center of $\boldsymbol{x'}$ and surrounded with rows and columns filled with zeros. The values of W', H' can be automatically computed given the original input size and the dimension of the kernel.

The feature map obtained after the application of a convolutional layer is usually passed through an additional *pooling* layer. The 2-dimensional pooling layer reduces the dimension of the input by computing the maximum (max-pooling) or the average (average-pooling) value of the input within a fixed-size region of size $F \times F$. The average pooling operation

applied to an input $\boldsymbol{x}$ returns a feature map $\boldsymbol{s}$ computed by:

$$s_{i,j} = \frac{1}{F^2} \sum_{w=1}^{F} \sum_{h=1}^{F} x_{i-w,j-h}. \tag{2.8}$$

Similarly, max pooling is computed with:

$$s_{i,j} = \max_{1 \leq w \leq F, 1 \leq h \leq F} x_{i-w,j-h}. \tag{2.9}$$

Since convolution and pooling are both linear operation, a non-linear activation σ needs to be applied after each convolutional layer. The ReLU activation is the most common one. In its simplest form, a CNN is built by stacking multiple layers of convolutional-ReLU-pooling blocks. The only parameters of these blocks are the parameters of the convolutional filter k. The last layer is followed by a linear layer, as in MLP, which takes as input the last feature map, flattened to a one-dimensional vector.
CNNs are very effective on images, since the application of convolution enforces a form of spatial locality, making the network robust to small transformations on the input. However, CNNs are also used for time-series (e.g., audio signals) by leveraging the one-dimensional version of both convolution and pooling.
The first Deep CNNs (LeNet by (LeCun, Bottou, et al. 1998), AlexNet by Krizhevsky, Sutskever, et al. 2012) trained with back-propagation were able to surpass the state of the art for object recognition from images. Deep CNNs are paradigmatic of how hidden representations are built hierarchically in a DNN. In fact, by studying the kind of features to which each layer responds the most, it is possible to show that the first layers of a Deep CNN are mostly active in correspondence of very simple patterns like edges, parallel lines, circles and so on. Deeper layers are able to build on the representations learned by previous layers and therefore they encode more abstract and high-level concepts. This property is due to the growing receptive field which a layer in a CNN is exposed to. Deeper layers acquire more context (e.g., are able to see larger portions of the input image) and are therefore able to learn more complex representations. The growing size of the receptive field is directly caused by the depth of the network and by the fact that a NN operates as a composition of many simple functions.

2.4.2 Recurrent Neural Networks

RNNs (Elman 1990) are NNs specifically designed to tackle sequential data. A sequence with T elements can be defined as any pattern of the form $\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_T}), \boldsymbol{x_t} \in \mathbb{R}^I \forall t = 1, \ldots, T$, where the order of the elements is given by the discrete step $t = 1, \ldots, T$. We will refer to the order given by the step t as *temporal* order, even if t may not always express a truly temporal relationship, but just an ordered one. RNNs maintain an hidden state $\boldsymbol{h} \in \mathbb{R}^H$ which is trained such that it represents the memory of the inputs seen so far. Since $\boldsymbol{h}$ is a fixed-sized vector, learning to encode all the past elements of the sequence into $\boldsymbol{h}$ becomes a harder problem as the sequence length T increases. The Elman RNN is one of the first and simplest form of RNN. The evolution of its hidden state when facing an input sequence $\boldsymbol{x}$ is given by:

$$\boldsymbol{h_t} = \sigma(\boldsymbol{W^x x_t} + \boldsymbol{W^h h_{t-1}}), \tag{2.10}$$

where $\boldsymbol{h_0}$ is initialized as the zero vector, $\boldsymbol{W^x}$ and $\boldsymbol{W^h}$ are the weights matrices and σ is the non-linear activation function (usually, the hyperbolic tangent or the sigmoid). Notably, RNNs use the same set of weights to model all the steps in the sequence. This allows to learn and exploit regularities due to the temporal correlation of the input elements.
The Elman's RNN (Elman 1990) is trained via SGD and a version of back-propagation for

sequences called Back-Propagation Through Time (BPTT). The BPTT algorithm proceeds backward from the last hidden state to the first one to compute the gradient of the loss with respect to the parameters $\boldsymbol{W^h}$ and $\boldsymbol{W^x}$. BPTT applies back-propagation to the sequence of feedforward networks. To compute the full gradient $\nabla_{\boldsymbol{W^h}}\mathcal{L}$, the chain rule includes a term $\sum_{k=1}^{t-1} \frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{h_k}}$, with $1 \leq k < t < T$. Each element of the sum can be expanded in

$$\frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{h_k}} = \prod_{i=k}^{t} \frac{\partial \boldsymbol{h_i}}{\partial \boldsymbol{h_{i-1}}} \tag{2.11}$$

$$= \prod_{i=k}^{t} \boldsymbol{W^h} \mathbf{diag}(\sigma'(\boldsymbol{h_{i-1}})), \tag{2.12}$$

where σ' is the derivative of the non-linear activation function applied to the $\boldsymbol{h_{i-1}}$. The product in Eq. 2.11 assumes large values if $\|\boldsymbol{W^h}\| \gg 1$ or very small values if $\|\boldsymbol{W^h}\| \ll 1$. The former case is called *exploding gradient*, while the latter is called *vanishing gradient* (Hochreiter 1998; Y. Bengio et al. 1994). Both of them cause instability in training. The exploding gradient can be controlled by clipping its value to stay in a fixed, small interval. Unfortunately, there is no general solution for the vanishing gradient problem. RNNs exhibiting vanishing gradient are not able to learn long-term temporal dependencies since the gradient signal for early steps approaches zero. The problem becomes worse as the sequence length increases.

Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber 1997) are designed to mitigate the vanishing and exploding gradient problem by enforcing $\|\boldsymbol{W^h}\| \approx 1$. The equations governing the LSTM are given by:

$$\boldsymbol{i_t}, \boldsymbol{f_t}, \boldsymbol{g_t}, \boldsymbol{o_t} = \sigma(\boldsymbol{W^x_{i/f/g/o}} \boldsymbol{x_t} + \boldsymbol{W^h_{i/f/g/o}} \boldsymbol{h_{t-1}}) \tag{2.13}$$

$$\boldsymbol{c_t} = \boldsymbol{f_t} \circ \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \circ \boldsymbol{g_t} \tag{2.14}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \circ \sigma(\boldsymbol{c_t}), \tag{2.15}$$

where the first equation defines the LSTM input, forget, cell and output gate, respectively. The activation σ is the hyperbolic tangent for $\boldsymbol{g_t}$ and $\boldsymbol{h_t}$ and the sigmoid for the other gates. The operator $\circ$ performs element-wise multiplication. The LSTM architecture does not remove the vanishing gradient altogether but, due to the adaptive gates, it largely mitigates it. LSTMs are able to learn longer temporal dependencies than an Elman's RNN.

Non-gated approaches rely on different mechanisms to address the vanishing gradient problem, like parameterizing recurrent connections with an orthogonal matrix (Mhammedi et al. 2017). The Linear Memory Network (LMN) by Bacciu et al. 2019, for example, is one representative of the non-gated approaches. LMNs leverage a conceptual separation between a non-linear functional component computing the hidden state $\boldsymbol{h_t}$ and a linear dynamic memory computing the history dependent state $\boldsymbol{h_t^m}$:

$$\boldsymbol{h_t} = \sigma(\boldsymbol{W^{xh}} \boldsymbol{x_t} + \boldsymbol{W^{mh}} \boldsymbol{h^m_{t-1}}) \tag{2.16}$$

$$\boldsymbol{h^m_t} = \boldsymbol{W^{hm}} \boldsymbol{h_t} + \boldsymbol{W^{mm}} \boldsymbol{h^m_{t-1}}. \tag{2.17}$$

The memory state $\boldsymbol{h_t^m}$ is the final output of the layer, which is given as input to subsequent layers.

A different kind of RNN is the Echo State Network (ESN) (Lukoševičius 2012). The ESN belongs to the Reservoir Computing framework (Lukoševičius and Jaeger 2009), where the recurrent architecture is composed by a set of untrained units (the *reservoir*), randomly connected between each other. The hidden state computation is similar to the one from the Elman's RNN provided by Eq. 2.10. Sometimes, the ESN also leverages a set of feedback

connections $\boldsymbol{W^f}$ giving rise to:

$$\boldsymbol{h_t} = \sigma(\boldsymbol{W^x x_t} + \boldsymbol{W^h h_{t-1}} + \boldsymbol{W^f \hat{y}_{t-1}}). \tag{2.18}$$

All the parameters $\boldsymbol{W^x}, \boldsymbol{W^h}, \boldsymbol{W^f}$ are not trained. The ESN randomly initialize the weights matrix for the recurrent connections $\boldsymbol{W^h}$ and then re-scale it such that its spectral radius (the largest eigenvalue) ρ is close, but smaller, than 1. This is a necessary, but not sufficient, condition for the Echo State Property (ESP). The ESP guarantees that, after a certain amount of steps, the hidden state computed by the ESN is independent on the initialization and only depends on the recent history of the input sequence. Surprisingly $\rho < 1$ works well in many practical cases. ESNs are widely used to predict the evolution of dynamical systems. Due to the untrained set of parameters, ESNs can be efficiently implemented in specialized, neuromorphic hardware. Since they do not need to be trained with BPTT, they enjoy a quicker training phase than fully-trained RNNs like LSTMs or LMNs and they do not suffer from the vanishing gradient problem.
The final output of any RNN is computed with a linear layer. In the case of ESNs, the linear layer (the *readout*) can be trained in closed form or with SGD. Instead, fully-trained RNNs use SGD and BPTT. Given the full sequence of hidden states $\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_T}$, which hidden state to feed as input to the last linear layer depends on the type of task.
Sequence classification tasks require to associate a single class to an entire sequence. RNNs compute the hidden state $\boldsymbol{h_T}$ by unrolling the computation defined by Eq. 2.10 (or Eq. 2.18 for ESNs) for each step $t = 1, \ldots, T$. The linear layer leverages only the last hidden state to generate the output vector $\boldsymbol{\hat{y}} = \boldsymbol{W^y h_T}$. *Sequence modeling* task requires to predict the next element in the sequence, given the previous ones. For each step $1 \leq t \leq T$, the RNN computes $\boldsymbol{h_t}$ and forward it to the linear layer to get the output $\boldsymbol{\hat{y}_t} = \boldsymbol{W^y h_t}$. As a result, while a sequence classification task associates a scalar (the class) to an entire sequence, sequence modeling tasks associate an output sequence to each input sequence. Sequence modeling tasks are sometimes framed as a sequence classification task by taking the input sequence in fixed windows and by predicting the next element of the sequence given the entire window. This way of considering the sequence modeling task achieves considerable speedup since each window can be processed in parallel. Moreover, windows guarantee a fixed sequence length which can be tuned to match the memory capability of the RNN. Finally, the *sequence-to-sequence (seq2seq)* task associates to an input sequence an output sequence which does not necessarily has the same length as the input sequence. One popular case of seq2seq tasks is the machine translation task. In machine translation, an input text in a given language needs to be translated into another language. If each word is treated as a separate element of the sequence, machine translation requires to deal with sequences of different lengths in input and output. RNNs are able to manage seq2seq task by using an *encoder-decoder* architecture. The encoder is an RNN processing the input sequence and returning the hidden state associated with the last element of the sequence. The decoder is a different RNN which takes as input the last hidden state produced by the encoder and generates the output sequence one step at a time, up until a stop vector is generated or a maximum number of steps is reached.

2.4.3 Transformers

CNNs and RNNs have inductive biases which make them very effective on images and temporal data, respectively. Instead, an MLP does not possess a strong inductive bias. As a consequence, its performance is usually lower than CNNs in Computer Vision tasks and RNNs on sequential data.
Similarly to MLPs, the Transformer architecture (Vaswani 2017) does not possess a strong

inductive bias. The core operations of a Transformer are similar to the ones of an MLP, with one important difference: Transformers leverage the *attention* layer, while MLPs do not.
In the last years, attention proved to be a very effective and general tool to tackle many different tasks under different domains: from NLP to Computer Vision and more.
The attention layer deals with sets of input samples. Given a mini-batch with n input samples, one can compute a query matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times k}$, a key matrix $\boldsymbol{K} \in \mathbb{R}^{n \times k}$, a value matrix $\boldsymbol{V} \in \mathbb{R}^{n \times v}$ and define the attention layer A as follows:

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{k}}\right)\boldsymbol{V}. \tag{2.19}$$

The attention expressed by Eq. 2.19 is also called scaled dot-product attention due to the dot-product similarity between the query and the key matrices and the scaling with respect to the dimension k. The multi-head version of the attention linearly project the key, query and value before feeding the result to the scaled dot-product attention. Each linear projection defines a head. The multi-head attention allows to learn different attention policies for the same set of input samples, thus increasing the expressive power of the attention layer.
The query, key and value matrices are obtained by linear projections of the mini-batch of input samples $\boldsymbol{X} \in \mathbb{R}^{n,i}$, where i is the dimension of each input sample:

$$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} = \boldsymbol{W^q X}, \boldsymbol{W^k X}, \boldsymbol{W^v X}. \tag{2.20}$$

Eq. 2.20 defines a *self-attention* operation, since the key, query and value are all obtained from the *same* set of samples $\boldsymbol{X}$. In many cases $\boldsymbol{X}$ is the output of a previous layer or a transformation of the raw mini-batch of input samples. One notable case of the latter is the *embedding*, a technique very popular in NLP. An embedding learns a vectorial, dense representation of a vocabulary of tokens. Each token (e.g., a word) is given a dense vector with a fixed number of components. The vector is called an embedding of the original token. Embeddings are learned as linear projections of the input via SGD and back-propagation, given a corpus in which each token is present. In the context of NLP, given a corpus of text, an Embedding layer learns an embedding for each word. Words which are "similar" will be assigned similar embeddings (e.g., close in the Euclidean space). On the contrary, words which do not express similar concepts will be assigned different embeddings. This simple, powerful technique allows to build a fixed-size representation for each element of a fixed vocabulary.
In the case of embeddings of words in a sentence, the self-attention computes the importance each embeddings has within the sentence, with respect to all the other words. The attention level can then be exploited to solve a specific task, since it will be trained to highlight those words which are more helpful to solve the task. As a matter of fact, the Transformer architecture is based almost entirely on the concept of attention.
A Transformer is an encoder-decoder architecture, similar to the one described for seq2seq tasks with RNNs.
Each layer of the Transformer encoder is composed by a number of identical blocks. Each block is composed by a multi-head self-attention, followed by an MLP with one hidden layer and ReLU activation function. The decoder is also composed by a number of layers. Each layer is composed by a block of masked multi-head self-attention, multi-head attention and feedforward MLP. The masking multi-head self-attention is similar to a multi-head self-attention, with the difference that the masking forbids any token to attend to future ones. This is particularly useful when dealing with sequences, where the causal relationship between the input would not allow to use the future to predict the past. For example, consider a sequence modeling task where the model needs to predict the upcoming word

given the previous ones (*language modeling*). If the model is allowed to compute attention values on the entire sequence of words, it will not be forced to learn any semantic relationship. All the knowledge needed is already present simply by looking at future words given the previous ones. Masking forbids this possibility. The second multi-head attention takes as input the encoder output to produce the attention keys and values and the output of the previous masked attention to produce the query. This block is fundamental, since allows every position in the decoder (the output) to attend to each position in the input. In the case of sequences, each token in the generated output sequence is given the context of the entire input sequence to attend to. This powerful mechanism is at the core of the strong performance obtained by the Transformer architecture on many different tasks. However, it also represents one of its greatest weaknesses. Attending to all tokens in a sequence implies a large computational cost, since the adaptive attention weights (similar to the MLP weights between two layers) are dense and their number scales quadratically with the input sequence length. Sparser, more efficient versions of the Transformer architecture have been developed to address this problem, even though none of them completely solves it.
The final output of the model can be computed by a linear layer taking as input the output of the decoder. For tasks like classification or regression, this can be done even without the decoder, since the output is not composed by a sequence of tokens. In this case, the output of the encoder is directly passed through the final linear layer to compute the Transformer output.
The Transformer architecture had a profound impact on the ML community. It sparked many different research directions and it contributed to dramatically increase the performance in many tasks. In particular, NLP has been the first domain benefiting from the power of Transformers. Transformer-based architectures like BERT (Devlin et al. 2019b), RoBERTa (Liu et al. 2019), GPT-2 (T. Chen et al. 2020) and others have become the standard in NLP tasks. Also, the large availability of data in NLP has made possible to scale the size of these models to billions of parameters. Nowadays, Transformers like the Vision Transformer (ViT) (Dosovitskiy et al. 2020) and BEiT (Bao et al. 2021) are also addressing Computer Vision tasks, with a similar amount of parameters and layers. We will use BERT, RoBERTa and ViT in the analysis of our Continual Pre-Training scenario. Therefore, it is worth introducing them a little bit more in detail.

BERT, RoBERTa. BERT (Devlin et al. 2019b) and RoBERTa (Liu et al. 2019) are popular language models widely used in NLP. The main characteristic of BERT is the pre-training phase, in which the model has access to bidirectional information (left-to-right and right-to-left) thanks to the masked language modeling protocol. BERT also leverage a next-sentence prediction as pre-training protocol to strengthen text-pairs representations. BERT comes in different sizes: the base model has 12 layers and 110 million parameters. The large model has 24 layers and 340 million parameters. BERT is an encoder-only model, meaning that it only leverages self-attention modules in each layer. Being bidirectional, no causal masking is applied during pre-training, unlike other models like GPT-2. BERT is pre-trained on the BooksCorpus (Zhu et al. 2015) and English Wikipedia datasets (Foundation n.d.). After pre-training, BERT can be fine-tuned to specific tasks by employing a separate prediction head.
RoBERTa borrows the architecture and properties of BERT, but it leverage an improved training phase. The BERT model originally published in Devlin et al. 2019b was found to be significantly under-trained. RoBERTa is trained for a larger number of steps on longer chunks of text, taken from the CC-News dataset [1]. BERT removes the next-sentence prediction

[1] `https://github.com/commoncrawl/news-crawl/`

task and randomly change the masking pattern applied to the input text. These simple modifications allowed RoBERTa to surpass BERT performance on a number of standard NLP benchmarks, like GLUE (A. Wang et al. 2018), SQuAD (Rajpurkar et al. 2016) and others.

ViT. ViT (Dosovitskiy et al. 2020) is a popular model that applies Transformers to Computer Vision. ViT works by dividing each input image into a set of patches. The patches are linearly projected before being fed to the ViT encoder, together with a positional embedding representing the spatial location of the patch. The ViT encoder is composed by a number of Transformer blocks. Each block leverages multi-head self-attention and an MLP. Differently from the unsupervised pre-training protocols in NLP, ViT pre-training employs the supervised image classification task. Similar to BERT and RoBERTa, the base version of ViT has 12 layers and 86 million parameters, while the large version has 24 layers and 307 million parameters.

Scaling in DL and CL. The large number of adaptive parameters gives DL models a great flexibility and a strong ability to represent complex relationships present in the input dataset. However, to fully exploit the expressive power of DNNs, the size of the dataset needs to be increased with respect to shallower networks with few layers. As we discussed in Section 2.1, gathering a vast amount of data is prohibitive for purely supervised task, where the availability of annotations is limited. The unsupervised and self-supervised pre-training task allows to overcome this limitation. Large and Deep Transformers pre-trained in an unsupervised/self-supervised way on a large corpus of unlabeled data and then fine-tuned in a supervised way on smaller, labeled corpus are at the basis of many successful real-world applications. Bringing these impressive abilities to their extreme is one of the long-term objectives of DL. One hypothesis sparked from the study of Deep models is that *scaling* the size of the model and the amount of training data will allow to produce better and better results. This view has already attracted many criticisms. Putting it to the test will likely be one of the main research directions for DL in the upcoming years.
Continual Learning is also dealing with scaling: the diffusion of DL models have also impacted on the way Continual Learning experiments are conducted, quickly moving from small datasets and shallow models to large datasets and deep models. Whether this is a positive turn or not, it is still up to debate. Be it as it may, scaling will likely have a profound impact on Continual Learning. Deep models like Large Language Models do not require to adapt much to incoming tasks, thus mitigating many issues related to learning continuously (like catastrophic forgetting). Instead of asking whether large models really need CL at all, we may instead ask whether the current benchmarks are still an effective way to measure their CL performance. Being trained on huge amount of (often undisclosed) data, it is difficult to gauge to what extent the new tasks encountered during CL are really new. Could it be that our, admittedly limited, benchmarks are likely present in a certain form in the model training data? Far from being a pessimistic view, this would instead require to redesign our idea of CL. Perhaps, bringing it closer to real-world applications for which there is hardly data available. Data can be collected *online*, during model lifetime. Continuous adaptation and personalization, we believe, still represent difficult challenges which CL will have something to say about.

2.4.4 Centered Kernel Alignment

Comparing different DL models is usually done by comparing their predictive performance based on a certain metric (e.g., accuracy for classification tasks). The predictive performance

can be used as a proxy to understand some of the properties of the trained model: whether the model is over-fitting, its generalization ability, the gap with respect to some upper bound. The Centered Kernel Alignment (CKA) (Kornblith et al. 2019) provides a flexible approach to inspect the behavior of two models by comparing the similarity of their hidden representations. CKA can be a useful complement to the experimental analysis, since it allows to intuitively grasp differences and similarities in the behavior of two or more models.
We will briefly present the approach when comparing the behavior of 2 layers of the same model, but the same holds when comparing layers from different models.
Given a model M with $L > 1$ layers, we can compute a matrix of neural activations $X^j \in \mathbb{R}^n \times p^j, j \in \{1, 2\}$, representing the output of layer j with p^j units on n input examples. Each column of X^j is centered. The linear CKA is computed as:

$$\frac{\|X_2^T X_1\|_F^2}{\|X_1^T X_1\|_F \|X_2^T X_2\|_F}, \tag{2.21}$$

where $\|X\|_F$ indicates the Frobenius norm of matrix X. There are other versions of the CKA (e.g., RBF CKA, also introduced by Kornblith et al. 2019), which we do not use in this thesis. Eq. 2.21 can be applied to all the pairwise combination of layers in a model or across different models. One possible application is to monitor the change in similarity as the model depth increases. The model should learn to build hierarchical representations, thus making more distant layer more different than closer ones (in terms of model depth). When comparing two different models, one can check which layers behave differently, thus ascribing to them the difference in performance, if any. When the number of examples n is large, the matrix X may not fit into memory. To mitigate this issue, CKA can be applied by aggregating results one minibatch at a time, with $n' << n$ examples, as showed in Nguyen et al. 2020. We will use this equivalent, but more efficient version of the CKA throughout the rest of the thesis.

Chapter 3

Non-stationary Environments

As humans, we are able to explore only a limited portion of the environment we live in. Our experience is always incomplete and subjected to changes and new discoveries. Previously inaccessible phenomena may become available at some point, while well-known conditions can change unexpectedly. The dynamic nature of our experience heavily influences the way we learn new concepts over time, giving rise to a continuous and usually robust process of knowledge acquisition and retention. The learning process of ML models described in Chapter 2, instead, is based on the iid assumption: data is always coming from a fixed, unknown distribution. While this allows to put forward a very effective optimization process, it also greatly limits the versatility of the model, since its performance on Out Of Distribution (OOD) data (Hendrycks and Gimpel 2023) is greatly impaired. Without the opportunity to adapt to new data, a ML model risks to deliver completely wrong predictions when exposed to information outside the training domain.
Learning in dynamic environments where data is subjected to changes and drifts (Ditzler et al. 2015) is a key challenge towards the design of AI agents able to operate autonomously in the real world. An agent unable to adapt to novel situations and contexts will never be able to grasp and navigate the ever changing complexity of a non-stationary environment.

3.1 Characterization of non-stationary Environments

A non-stationary environment (Ditzler et al. 2015) can be formally characterized in terms of the dynamical properties of the probability distributions generating the data. We will define non-stationary environments in the context of supervised learning tasks. The non-stationary environment framework is complementary to the type of tasks described in Section 2.1 and can be readily applied to unsupervised and RL tasks.
Let us consider a data-generating process $\mathcal{P}$ which produces a stream of input-target pairs $\{(\mathbf{x_t}, \boldsymbol{y_t})\}_{t=1,2,\dots}$, where the index t indicates a time-step. The input pair at each time t is sampled from the joint probability distribution $p_t(\boldsymbol{x}, \boldsymbol{y})$, leading to the input distribution $p_t(\boldsymbol{x})$ (evidence) and the conditional distribution $p_t(\boldsymbol{y}|\boldsymbol{x})$ (posterior). Unlike iid supervised tasks described in Section 2.1, here we explicitly consider distributions which may vary over time t. There are different types of distribution shifts, or *drifts* (João Gama et al. 2014), that can occur in a non-stationary environment.

Definition 5. *A* virtual drift *is characterized by a change in the evidence $p_t(\boldsymbol{x})$, which does not affect the posterior $p_t(\boldsymbol{y}|\boldsymbol{x})$.*

If we think about the concepts the agent need to learn as generated by some *hidden contexts* (Tsymbal 2004), a virtual concept drift causes a change in the hidden context, without

affecting the real concept. In the case of classification, the decision boundary stays the same, while the distribution of points around it changes. For this reason, virtual concept drift is sometimes referred to as *sampling shift*, since a change in the data sampling process can be responsible of a virtual drift. As an example, consider the case of spam classification. While the concept of spam remains relatively the same over time, new examples of spam messages may come from a wide variety of different sources. In fact, after being used to receive messages from sport newsletters, receiving a new message from a computer science magazine induces a virtual drift: new parts of the spam/non-spam regions are being observed. This usually affects the prediction performance of the model, which may not have enough information (or worse, may have spurious one) to correctly classify the message.
While virtual drift concerns the change in the evidence, a real drift affects the posterior probability.

Definition 6. *A* real drift *is characterized by a change in the posterior probability* $p_t(\boldsymbol{y}|\boldsymbol{x})$*, which does not affect the evidence* $p_t(\boldsymbol{x})$*.*

In the case of a real drift, what is changing is the real concept we are dealing with. Continuing with the spam detection example, a person suddenly becoming interested in absurd offers promising unrealistic achievements will start to classify as spam those messages coming from otherwise respected senders and vice-versa. This change in the spam concept affects the conditional probability of an input being classified as spam, not the distribution of those inputs.
Other useful categorizations of drifts are orthogonal to the two just described. In terms of drift duration, a drift can be either *gradual* or *abrupt*. In the first case, the drift (be it real or virtual) happens slowly over time, while in the second case it happens almost instantaneously. A drift can also be *permanent* (once happened, the new distribution stays fixed for an unlimited amount of time) or *transient* (the drift is expected to vanish after a certain amount of time). Finally, in recurrent drifts (*recurrent concepts*) a previously encountered change can re-appear later on in time. This particular drift requires the agent to be able to quickly retrieve previously acquired information in order to adapt to the new distribution.
A non-stationary environment, no matter the frequency or the type of drifts, can always be converted into a stationary, iid one. To this end, it is sufficient to store all the data points $(\boldsymbol{x_t}, \boldsymbol{y_t})$ up to the current time T. Then, $\forall T = 1, 2, \ldots$, the dataset $\bigcup_{t=1}^{T}(\boldsymbol{x_t}, \boldsymbol{y_t})$ can be treated as an iid dataset and a ML model can be trained on it with multiple epochs of SGD or similar optimizers. However, as discussed in Ditzler et al. 2015, for very long sequences of data the computational resources in terms of both memory and training time (therefore, energy) may exceed the available capacity. For example, a system which is required to output predictions very frequently (e.g., real-time) does not have the time to be re-trained on all the samples seen up to that point. Instead, learning only the new information is much quicker and allows to save energy, making the approach suitable also for low-powered devices. Nonetheless, there exist situations in which there is the possibility to perform re-training from scratch (starting from a model with randomly initialized parameters), since there are no tight constraints on memory, training time or other factors. In those cases, the performance of an iid learner like a NN is often superior to the one of an agent operating in non-stationary environments, which cannot store or be trained on all previous data.
One extreme case of constrained data availability is *online learning* (Fiat and Woeginger 1998). In online learning, data comes one sample at a time and it is not possible to store any previous sample. These strict limitations are sometimes relaxed in the sense that samples may arrive and be stored in a *very* limited amount (Losing et al. 2018). Importantly, online learning does not necessarily happens in a non-stationary environment (Losing et al. 2018).

Table 3.1: Comparison of the main differences between online learning, batch learning and Continual Learning. Online learning with data drifts is usually considered a separate topic than online learning.

	Drifts	Multiple epochs	Evaluation
Online Learning	×	×	test-then-train
Batch Learning	×	✓	evaluate after training
Continual Learning	✓	✓	evaluate after training

Samples may be generated from the same distribution, as in iid settings. *Streaming learning* is usually considered a non-stationary version of online learning (Gomes et al. 2019).
Online learning can be compared with *batch* learning (Table 3.1), where samples come in mini-batches. The main difference between online and batch learning emerges during evaluation: batch learning trains a model h until convergence on a dataset D and then uses the trained model to assess the performance on a held-out test dataset. In online learning, the model at time h_t is trained on all the samples up to time t and it is directly used to evaluate the performance on the *next* sample at time $t+1$ (prequential evaluation, or test-then-train evaluation). The final performance is the average accuracy obtained by each model h_t. Importantly, online learning allows only one pass on the data. The concept of "epoch" is therefore absent in online learning. While in batch learning only the final model is used for model assessment, in online learning *each* intermediate model is used to measure the performance on the unseen, next input. Since the input used for testing is subsequently used for training, in online learning there is no strict distinction between a training and a test set. From this perspective, Continual Learning is much more similar to batch learning than online learning. In fact, as we will see in Chapter 4, Continual Learning borrows the evaluation protocol of batch learning, but it applies it to non-stationary data streams.

3.1.1 Learning Approaches for Non-stationary Environments

Having discussed the main characteristics of a non-stationary environment subjected to drifting, dynamic distributions, we now turn the attention to the approaches already present in the literature to learn in such environments. There are two main families of strategies, namely *active* and *passive* ones. We will follow the review in Ditzler et al. 2015 and present some of the approaches belonging to the two families.

Active Approaches. Active approaches (Ditzler et al. 2015) operate by monitoring the stream of data and actively looking for drifts. The first phase requires to extract meaningful features from raw data. This can be done by any ML model, like a NN. The feature representations learned or extracted from the data are used by the Change Detector to establish whether the current sample is similar to the previous ones. In case of low similarity, the Change Detector signals the presence of a drift, since the current sample is likely to be generated from a new distribution. When a drift is detected, the Adaptation module changes the Predictor. The Predictor is the module responsible to predict an answer for the current sample. The Change Detector usually focuses on the changes in the prediction error made by the Predictor. The error is used as a proxy for a drift in the input distribution. In fact, a Predictor trained on data from a given distribution is likely to experience a large error when predicting data coming from a different distribution. Importantly, when detecting a drift, active approaches consider the current Predictor obsolete and re-build it to discard

the old knowledge. Especially in the presence of recurring concepts, entirely erasing existing information may be deleterious and slow down the acquisition of re-occurring concepts.
Both the the drift detection phase and the adaptation phase are well studied in the literature. Drift (or change) detection mechanisms are mainly based on statistical tests that can check the presence of a drift from the extracted features. Some methods work with fixed-length sequences, while others are able to work in online learning settings. In the former case, data needs to be gather together before being able to detect whether the sequence contains a drift (Hypothesis Tests approaches, like Patist 2007) and at which point in the sequence the drift is happening (Change-Point Methods, like Hawkins et al. 2003). Therefore, these methods are not able to tackle online learning setting. In the latter case, instead, the drift is dynamically detected by inspecting one sample at a time. Sequential Hypothesis Tests (e.g., Wald 1945) accumulate evidence over time to support either the no-change hypothesis or the opposite one. A huge drawback of this family of approaches is that, once the required statistical confidence is reached and a decision is made, the algorithm discards future samples since they do not contribute anymore to the statistical test verification. The Change Detection Tests (Bifet and GavaldÃ 2007), instead, continuously monitor the data stream for an unlimited amount of time. Change Detection Tests compare the current training error with a target quantity, like the error obtained by previous versions of the Predictor. If the difference surpasses a threshold, a change is detected. Determining the optimal threshold is however quite difficult since it depends on the specific application and on the type of data produced by the environment. In online learning setting is not possible to perform model selection to select the optimal threshold value, given the constraint on the sample availability. As a result, the trade-off between false positives and false negatives in drift detection remains one of the main issues of Change Detection Tests.
When a drift is detected, the Adaptation module is in charge of discarding out-dated information and only considering new one. In the windowing approach (JoÃ£o Gama and Castillo 2006), the adaptation module keeps a fixed-length window of previous samples and considers as meaningful samples only the ones present in the window. Any other sample is considered out-dated. The samples in the windows are then used to re-train the Predictor. The window length shares the problem of the threshold in Change Detection Tests and represents an hyper-parameter which is difficult to be tuned. Techniques like Just-In-Time Classifiers (Alippi et al. 2013) leverage an adaptive window length to overcome this limitation. While windowing considers a fixed amount of samples, weighting approaches (Klinkenberg 2004) include all the samples, but they weight each sample proportionally to a measure of importance. The time since that sample has been seen is one popular importance measure. This gradual-forgetting mechanism is quite flexible and allows to specify a general way of weighting samples. However, it requires to store all data in memory which, as we already discussed, is not realistic for many applications. Sampling techniques, like Reservoir Sampling (Vitter 1985), do not require to store all the samples into memory. Reservoir Sampling guarantees a uniform sampling probability on the entirety of the data. When re-training the Predictor during adaptation, this property is highly desirable to avoid unbalanced datasets. Still, in case of limited memory, the number of samples stored by sampling techniques may not be representative enough to perform an effective re-training of the Predictor.

Passive Approaches. Passive approaches (Ditzler et al. 2015) do not aim at detecting whether a drift is occurring or not. At first, this may seem surprising given the fundamental role played by drifts in non-stationary environments. However, avoiding drift detection frees the model from the problem of dealing with false positives and false negatives. Moreover, especially in environments with gradual drifts, it is often inherently difficult to detect when

a transition to a new distribution is happening. One may need to wait many samples before being able to tell that the input distribution entered a new regime. In the meantime, since the adaptation phase is not triggered, the performance decreases.
Passive approaches continuously adapt the Predictor, by assuming that each new sample comes from a new distribution. Like active ones, passive approaches can still leverage a fixed window of previous samples. These Single Classifier Models re-train the Predictor *at each time-step* on the samples contained in the window. The Very-Fast Decision Tree (Domingos and Hulten 2000) is one of the most used Single Classifier Models. Active approaches perform re-training only in case of a drift detection. While computationally efficient, passive approaches leveraging windowing exhibit a high variance in the loss, especially when the stream presents many drifts. Ensemble Classifier Models (Tsymbal et al. 2008) provide a more robust performance, at the expenses of a larger computational cost. These approaches dynamically expand the Predictor when new data comes in and maintain an ensemble of predictors trained on specific subsets of the input stream. Ensemble solutions allows to tackle multiple drifts by incorporating new knowledge — adding a new Predictor — and by removing out-dated information — removing the corresponding Predictor. The voting mechanism used to produce the final output from the ensemble of Predictors can also be tuned to control the importance of each member of the ensemble.
One of the first Ensemble Classifier Models is called Streaming Ensemble Algorithm (SEA) (Street and Y. Kim 2001). SEA operates by adding a new element in the ensemble each time a new input arrives. In order to manage the large computational cost of keeping many Predictors, SEA removes some of them once reaching a pre-defined number of Predictors. The removal process follows different heuristics: removing the first-added Predictors in a First-In-First-Out manner, or removing the Predictors achieving the lowest performance on the current sample. Learn^{++}.NSE (Elwell and Polikar 2011) adopts a similar approach by computing the final output through a weighted combination of the Predictors. Predictors performing best in recent times are assigned larger weights with respect to the others. Keeping a large set of Predictors is useful in the presence of recurring concepts, since the information related to the re-occurring concept can immediately be leveraged by using the Predictor which was trained on that concept. Working with ensemble methods in passive approaches also allows to leverage the existing literature about popular approaches like bagging and boosting, used extensively in iid online learning environments.

3.2 Catastrophic Forgetting

Non-stationary environments require a continuous update of the way new samples are represented by the model. When facing a new distribution, the model is adapted in order to capture the properties of the new samples. Otherwise, the performance on incoming samples risks to degrade very rapidly. After a short period of adaptation to the new data, the performance is expected to be quickly recovered and to approach the optimal one. The objective of predicting correctly new samples inevitably leads to the *necessity* of adapting the model.
What if we also care about the performance on *previously seen* data? At some point, in a non-stationary environment with recurring concepts, the model will need to predict samples similar to ones it already encountered in the past. If the model does not *forget* the previously acquired knowledge, it will be able to provide correct predictions without the need to be trained on new samples. Training may serve to improve the performance, not to recover the previous one. If the frequency of recurring concept (Widmer and Kubat 1996) is high, the advantage of a model which does not forget will be more pronounced. In the case of

non-stationary environments without recurring concepts, the ability to preserve existing information may not be that relevant. If the model will never observe some concept again, there is no point in keeping the ability to correctly predict that concept. However, in many real-world cases concepts occur multiple times. The time between an occurrence and the next one is also a relevant factor. If a concept re-occurs only after a long amount of time, it can even be counterproductive to keep the ability related to that concept at the expenses of new information. The priority of reaching a good performance for the incoming samples takes precedence over the possibility to re-encounter again previous ones.
To better stress the advantage of a model which does not forget previous knowledge, let us consider the case of a robotic grasping hand working in a production line. During the initial training phase, the robotic hand learns how to grasp the specific set of objects passing by the production line. After some time, however, the product manager conceives a new product which is added to the line. Now, the robotic hand needs to learn how to grasp new objects, while still manipulating objects needed for the previous products. If every time there is a new addition to the line the robotic hand needs to be trained from scratch on every object, the overhead will become significant. Moreover, the training time of the hand will grow over time, since there will always be more objects to learn. If training time is too long, either the production needs to be stopped or it needs to continue with an out-dated robotic hand which is probably going to fail when grasping new or previous objects.
The phenomenon of *catastrophic forgetting*, or simply forgetting, affects many ML models (McCloskey and Cohen 1989; French 1991). In particular, NNs strongly suffer from this problem. When encountering a new data distribution, a naive update of a NN through SGD will lead the model to forget previous knowledge (French 1991; Robins 1995). Forgetting can be easily measured by looking at the change in model performance over time. Let us suppose that, at time $t = 1$, the model h_1 learns to predict samples coming from the distribution p_1. Right after training, h_1 will achieve a certain performance a_1. At time $t = 2$, a new distribution p_2 comes in. The model h_1 is then trained on data sampled from p_2, resulting in a final model h_2. We call a_2 the performance achieved by h_2 on data sampled from the *previous* distribution p_1. This requires to store a subset of samples from p_1, which however are not used to train the model. The difference $a_1 - a_2$ expresses the amount of forgetting experienced by the model on distribution p_1 after being trained on a new distribution p_2. Forgetting is currently one of the main challenges of Continual Learning. We will dive deeper into other metrics for forgetting in Section 4.2.
We have said that NNs are prone to forgetting. To better understand the reason behind the presence of forgetting in NNs, it is useful to consider the *stability-plasticity* dilemma, first described in Carpenter and Grossberg 1986:

Definition 7. *An adequate self-organizing recognition system must be capable of plasticity in order to learn about significant new events, yet it must also remain stable in response to irrelevant or often repeated events.*

The stability-plasticity dilemma requires to find a good trade-off between the ability of a model to learn new information and the ability to preserve previously acquired knowledge. NNs trained with SGD tend to show a strong plasticity, since their only objective is to fit the data they are exposed to. While this is convenient when facing iid data, the presence of drifts in non-stationary environments makes the network focus only on the recently seen samples. The hidden representations learned during early stages are destroyed in favor of the new ones. The stability-plasticity dilemma highlights one great weakness of NNs. It also suggests a possible solution to it: enhancing the stability of a model in order to counter-balance its extreme plasticity. This has been the object of many studies in Continual Learning, which will be addressed in Section 4.3.

Another reason why forgetting plagues NNs is related to the way NNs architectures are designed. As we have seen in Section 2.3, the basic building block of a NN is the linear layer, expressed by $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$. The relation between the weights matrix $\boldsymbol{W}$ and the input vector $\boldsymbol{x}$ is a linear, *dense*, relationship in which each feature of the input is connected to each unit in the layer (rows of $\boldsymbol{W}$). Since many NNs are composed of such linear layers, the resulting architecture is also densely connected. The case of CNNs is different, since the same set of weights used during convolution is repeatedly applied to different patches of the input image (weight sharing). In convolution, each input feature (i.e., pixel) is not connected to all the unit in a convolutional layer, but only to one of them. However, as we will see, even CNNs suffer from forgetting and they benefit from an additional degree of sparsity in the connections. The idea behind designing *sparse* networks is that preserving existing information (stability) is easier when the network can be divided in sub-modules, loosely connected with each other (French 1991). In dense networks, a change in a weight affects all the weights connected, directly or indirectly, to it. The more the weights, the larger the forgetting. The task of keeping a network stable while still guaranteeing a sufficient level of plasticity is therefore more difficult in a dense network than in a sparse one. In fact, in sparse networks each weight is connected to few others. This limits the span of the changes propagated in the network during training. The importance of sparse connections to mitigate forgetting has been known since many years (French 1991; French 1999). Due to the NN over-parameterizaton with respect to the task, it is possible to extract a sparse sub-network from the trained, dense one with minimal loss in performance (Lottery Ticket Hypothesis) (Dettmers and Zettlemoyer 2019). This approach helps when the drifts are easily recognizable or when using active approaches, since the sparse network can be obtained right after a drift detection. However, in all the other cases there is no clear definition of when a training phase ends. Therefore, it is not possible to decide when to *prune* the network. Sparsity needs to be imposed *during* training in order to design networks which are robust to forgetting. Learning how to *grow* sparse networks is still an open challenge of ML research.

Chapter 4

Continual Learning

The ability to learn in non-stationary environments is a long-sought challenge in AI. The objective of Continual Learning (CL), also called Lifelong Learning, is defined in Parisi, Kemker, et al. 2019 as the design of

> an adaptive algorithm capable of learning from a continuous stream of information, with such information becoming progressively available over time and where the number of tasks to be learned (e.g., membership classes in a classification task) are not predefined. Critically, the accommodation of new information should occur without catastrophic forgetting or interference.

This definition, although high-level, sets a number of requirements which a CL agent should possess

1. the need to adapt to new, unknown distributions which change over time;
2. the ability to learn dynamically, without making strong assumptions about the number or the type of tasks the agent will deal with;
3. the ability to incrementally learn new knowledge without destroying the previously acquired information. This tells the fundamental role played by forgetting and the stability-plasticity dilemma in the design of CL agents.

Despite the current focus on forgetting, CL is concerned with many aspects of the learning performance in a non-stationary environment. Learning should be efficient in terms of memory and computational resources. They should be constant with respect to the number of samples learned by the model. Moreover, learning new information should occur rapidly, in a sample-efficient way. Requiring large datasets is a strong disadvantage since the agent has no control on the type of data it will face.
The aforementioned characteristics are almost absent in the definition of offline ML given by Def. 1.
Other research fields have studied problems which are also considered by CL or that are closely related.
Online learning in non-stationary environments (Ditzler et al. 2015), as described in Chapter 3, focuses on the incremental acquisition of knowledge from a stream of incoming data, which is not entirely available at the beginning. However, data does not necessarily introduces new concepts, which is instead a fundamental property of CL. The objective of online learning is to quickly acquire new knowledge and to improve the prediction accuracy on the *incoming* samples. Crucially, and unlike CL, data comes one or few samples at a time. However, online learning does not consider the forgetting problem, but just the expected future performance. The prequential evaluation in online learning uses incoming samples to, first, predict the

outcome of the model and, then, to update the model itself. CL adopts the ML paradigm, and trains the model on incoming samples before evaluating on *separate* data, used only for testing purposes.
Other related research topics include multi-task learning (Caruana 1997) and domain adaptation (Ganin and Lempitsky 2015). The former requires a model to learn a fixed set of different task, all presented at the same time. Multi-task learning is similar to CL in the sense that, at the end of the training phase, the model is able to tackle more than one task. However, in multi-task learning data is entirely available from the beginning and it is not introduced progressively over time. Therefore, the concept of forgetting is completely absent in multi-task learning. Domain adaptation focuses on the performance of a given model on a novel domain. In ML, domain adaptation is tightly connected to pre-trained models (models which already contain a certain amount of knowledge). The objective of domain adaptation is to adapt pre-train models on new data coming from a single, unseen domain and to measure the final performance on that domain. The fine-tuning phase adapts the model to the new domain, without usually caring about the performance on the original domain. Forgetting is again not considered in this area.

To better formalize the properties and requirements of a CL environment and of a CL agent, we will start from the definition provided in Díaz-Rodríguez et al. 2018; Timothée Lesort, Lomonaco, et al. 2020.

Definition 8. *Given a sequence of datasets $\mathcal{S} = (D_1, D_2, \ldots)$, an external memory M, an (optional) task label t which uniquely identifies the task to be solved and its corresponding target function $g_t^*(\boldsymbol{x})$ which solves the task t, a learning algorithm A^{CL} is a CL algorithm with the following signature:*

$$\forall D_i \in \mathcal{S},\ A_i^{CL} :\ (h_{i-1}, D_i, M_{i-1}, t_i) \ \longrightarrow\ (h_i, M_i)\ .$$

CL aims to find an optimal CL model h^* which is able to achieve the best performance on all the datasets encountered up to that point. Def. 8 provides additional details about typical components commonly found in many CL algorithms. The external memory M can be used to store samples coming from previously encountered distributions or any other information useful to improve the performance of a CL agent. Naturally, a large memory size improves the prediction accuracy and contribute to mitigate forgetting. At the same time, it increases the computational cost required to update/use the memory. The task label t and the concept of task itself is still debated in CL (Aljundi, Kelchtermans, et al. 2019). In the literature, the term "task" is used to indicate any dataset present in the stream (van de Ven and Tolias 2018b). Therefore, a stream $\mathcal{S}$ composed by 5 datasets is a stream with 5 tasks. The task label is an external information provided by the environment which allows to distinguish the origin of specific input samples, that is from which distribution/task D_i they are coming from. We will discuss more about the role of task labels when presenting different CL scenarios and strategies (Section 4.1 and 4.3).
It is important to clarify that, although few terms have acquired a somewhat stable meaning over time, there is no standard nomenclature and notation in the CL literature, yet. For example, the term "task" is heavily overloaded, causing lots of confusion in the understanding of which conditions a CL experiment is conducted in. In fact, "task" is used not only to indicate an experience, hence the data currently seen by the model, but also the objective function (as in classification task) and it is also present in the "task label" concept associated with it.
Since the author has actively contributed to its definition, in this thesis we adopt a recently-proposed nomenclature (Lomonaco, Pellegrini, et al. 2021) which builds on common ter-

minology, but also modifies important interpretations of it. This choice does not have the objective of creating a CL niche to exclusively fit our purposes. Rather, it is the result of long discussions with many members of the CL community. We believe our nomenclature provides a flexible framework which reduces the ambiguity often present in existing works. It will help in reasoning about CL beyond the few configurations and experimental conditions currently present in the literature. As we go on, we will highlight the difference between our nomenclature and other alternatives. We hope that this will better clarify the motivation behind the introduction of a new nomenclature.
Following Lomonaco, Pellegrini, et al. 2021, we can slightly redefine the data stream presented in Def. 8. This is one of the major changes adopted in our nomenclature. We define a (possibly unbounded) data stream $\mathcal{S}$ as composed of a sequence of *experiences* $(e_1, e_2, \ldots)$. The term experience, which is also used by Mitchell 1997 in Def. 1, is unambiguous and surpasses the overloaded term "task". Here, an experience e_i identifies not only the dataset D_i but also any additional knowledge the environment provides. For example, the task label is now part of the experience, instead of being a separate piece of information. Equipped with this new formalization, Def. 8 can be rephrased into:

Definition 9. *Given a stream of experiences $\mathcal{S} = (e_1, e_2, \ldots)$, where each e_i provides a dataset D_i with K samples, task labels $\{t_i^j\}_{j=1,\ldots,K}$ associated to each sample and an external memory M, a learning algorithm A^{CL} is a CL algorithm with the following signature:*

$$\forall e_i \in \mathcal{S},\ A_i^{CL} :\ (h_{i-1}, e_i, M_{i-1}) \ \longrightarrow\ (h_i, M_i)\ .$$

The updated definition of a CL agent preserves all the components of Def. 8, with the few modifications already discussed. For the sake of completeness, since the presence of task labels and memory is optional, they can be respectively set to zero and to the empty set in case the environment does not provide them. Importantly, Def. 9 is agnostic with respect to the kind of task the agent is facing. The stream definition encompasses supervised and unsupervised learning problems as well as RL problems. In the latter case, the dataset will contain trajectories in the form of *state-action-reward-next state*, sampled from the environment by the agent's actions. For continual supervised learning, it is enough to consider samples of the form $(\boldsymbol{x}, \boldsymbol{y})$, while for unsupervised learning the target $\boldsymbol{y}$ will not be present.

4.1 Scenarios and Benchmarks

The space of possible non-stationary environments is vast and diverse. As we have seen in Chapter 3, there exist many possibilities regarding the type of drift and its behavior over time. The CL literature offers some examples of environments with specific relaxations and constraints.
In the following, we will indicate an instance of a CL environment with specific properties with the term *scenario*.

Definition 10. *A CL* scenario *provides a characterization of the type of drifts occurring in the CL stream of experiences. The properties of the scenario define the constraints which the model is subjected to during CL.*

The same scenario can be implemented with different types of data, as long as the properties of the scenario are satisfied.

Definition 11. *A* benchmark *consists in the realization of a scenario with a specific set of data.*

Table 4.1: Some examples of Continual Learning benchmarks and their associated scenario.

Benchmark	Scenario(s)
Split MNIST (van de Ven and Tolias 2018b)	CI/TI
Permuted MNIST (van de Ven and Tolias 2018b)	DI
CORe50 (Lomonaco and Maltoni 2017)	NC/NI/NIC
OpenLORIS (She et al. 2019)	DI
Stream-51 (Roady et al. 2020)	Online/OOD
vCLIMB (Villa et al. 2022)	CI
Endless Procedural Driving Simulator (Hess et al. 2021)	DI/NIC
Atari (Mnih et al. 2013)	CI/TI
OGBG-PPA (W. Hu et al. 2020)	CI

The difference between a CL scenario and a CL benchmark is usually not stressed much. It contributes to the ambiguity currently present in the CL nomenclature. We have already seen an example of ambiguity in the case of "task" and "dataset". We think that the distinction between "scenario" and "benchmark" matters, since it allows to disentangle the properties of a scenario from the peculiarities of the input data. These two factors both affect the final outcome, but in different ways. In fact, learning in different scenarios with the same dataset may often produce very different results. Table 4.1 provides an overview of the most popular CL benchmarks and where they fit in the landscape of CL scenarios.

4.1.1 Continual Learning Scenarios

One of the most used taxonomy for scenarios is the one defined in van de Ven and Tolias 2018a; van de Ven and Tolias 2018b. It includes Class-Incremental (CI), Task-Incremental (TI) and Domain-Incremental (DI) learning scenarios. Such scenarios are usually employed in supervised learning problems and classification tasks, especially in Computer Vision tasks with image data.

Class-Incremental Scenario. The CI scenario was originally proposed in Rebuffi et al. 2017, where the properties of the scenario are first laid out. In CI scenarios 1) examples of different classes need to occur at different times and 2) at any time, the classifier should predict all the classes seen so far. The additional requirement of bounded memory and computational complexity specified by Rebuffi et al. 2017 prevents inefficient solutions such as storing all samples in an external memory for re-training. The work by van de Ven and Tolias 2018b further describes CI scenarios, by specifying that they do not provide task labels at test time, but they do provide experience boundaries during training. The drift occurring between one experience and the other is abrupt, since the introduction of a new experience (new classes) happens instantaneously. The main characteristic of CI scenarios is that a certain class is never seen outside of the experience it first appeared in. This constraint prevents repetition of previous classes and it has a strong impact on the evaluation metrics (Section 4.2). In the case of NNs, for example, adding new classes dynamically causes lots of interference in the output layer (Timothee Lesort, George, et al. 2021). Due to SGD updates which strengthen the recently used weights, the model will always prefer the new classes over the previous ones. In CI scenarios, NNs show around 0% accuracy on previous classes, no matter the original performance. Since in recent years CL has been mostly focused with the mitigation of forgetting, CI scenarios are amongst the most used ones. They exacerbate

the forgetting issue, perhaps more than needed in most of the cases. As such, they offer a easy-to-use test-bed to design strategies which mitigate the problem. In Chapter 6, we will discuss more extensively some disadvantages coming from an exclusive focus on CI scenarios.

Domain-Incremental Scenarios. DI scenarios are introduced in van de Ven and Tolias 2018b. In DI scenarios, each experience introduces new examples of previously seen classes. This means that, in DI, *all* the classes are seen in the first experience. This is the main difference with respect to CI scenarios. Otherwise, the two scenarios are very similar, since they both show abrupt drifts with known experience boundaries. Even in DI scenarios, task labels are not available at test time. Admittedly, the drift in DI scenarios is a softer one with respect to CI scenarios. The introduction of new instances does not require to learn arbitrarily new decision boundaries like the introduction of new classes. Usually, new instances of the same object class lie closer in the feature space of a model than completely new objects. Again, this has an impact on the evaluation metrics and on the challenges associated with the scenario.

Task-Incremental Scenarios. The TI scenario (van de Ven and Tolias 2018b) is characterized by the availability of task labels at test time. A model can therefore leverage this information to automatically choose the predicted class only among the ones seen in that task. This makes it very easy to employ techniques which treat each task as separate from the others.*Multi-headed* models are an example of such techniques (De Lange, Aljundi, et al. 2021), similar to the ensemble discussed for passive approaches in Section 3.1.1. However, unlike ensembles, the appropriate predictor is selected via task label information provided by the environment. Multi-headed models first allocate a separate head (e.g., classifier) for each task. Then, at test time, they predict the output by using the head corresponding to the current task, which is identified by the task label. This simple configuration is able to mitigate a large percentage of forgetting (De Lange, Aljundi, et al. 2021). However, in many real-world scenarios it is unlikely to be able to partition the input space, since a task label is not available. In those cases, DI and CI scenarios are a better fit. As in the example provided by van de Ven and Tolias 2018b, TI scenarios are usually implemented by introducing new unseen classes with each new experience. This corresponds to a CI scenario with task labels availability.
Even if van de Ven and Tolias 2018b explicitly distinguish between these two cases, TI scenarios are sometimes thought to imply the usage of multi-headed models. This aspects often blend together and provide again an example of ambiguity in the CL nomenclature. Still, they are profoundly different: TI identifies a specific scenario which constraints the structure and the content of a CL stream. The multi-headed approach is a way to tackle a CL problem and it is a property of the model, not the scenario. In fact, multi-headed models can be implemented in any kind of scenarios, even in ones which does not provide task label information. In those cases, one can develop techniques to infer the task label at training and test time. This module must be more powerful than a simple drift detector. Its role is to recognize which distribution the input belongs to, not only to detect a change in the distribution.

More than three. A different way of classifying CL scenarios is provided in Lomonaco and Maltoni 2017; Maltoni and Lomonaco 2019. The authors identify two dimensions along which to position different CL scenarios. One dimension represents the availability of task labels provided by the environment. The other the content type included in new experiences. This formulation gives rise to 9 possible combinations, described in Table 4.2.

Table 4.2: The 9 different CL scenarios from Lomonaco and Maltoni 2017. The correspondence with the framework proposed in van de Ven and Tolias 2018b is provided in the table cells. Dashes indicate incompatible combinations. Empty cells indicate scenarios which are not explicitly named in the literature.

	New Classes	**New Instances**	**New Instances and Classes**
Single Increm. Task	Class-Incremental	Domain-Incremental	
Multi Task	Task-Incremental		
Multi Increm. Task	—	—	

The task label dimension generates the Single Incremental Task (SIT), the Multi Task (MT) and the Multi Incremental Task (MIT) scenarios. SIT does not provide any task label, MT provides a different task label for each experience and MIT provides a task label for each experience, where a task label can re-occur in subsequent experiences. Similar to the framework of van de Ven and Tolias 2018b, task labels are defined for the whole experience. The content type dimension generates the New Classes (NC), the New Instances (NI) and the New Instances and Classes (NIC) scenarios. NC associates to each new experience a new, unseen set of classes. NI associates to each new experience new samples from previously seen classes. NIC associates to each new experience both new classes and new samples of previously seen classes. The framework of van de Ven and Tolias 2018b can be mapped to the framework of Lomonaco and Maltoni 2017; Maltoni and Lomonaco 2019. CI scenarios are built out of SIT+NC scenarios, DI scenarios corresponds to a SIT+NI configuration and TI scenarios are equivalent to MT+NC scenarios. The correspondence between the two framework is provided in Table 4.2. Interestingly, Table 4.2 shows empty cells. They indicate that there exist valid combinations (e.g., SIT+NIC, MT+NI) which are not currently considered in the literature.

Online/Task-Free/Data-Incremental Scenarios. While the aforementioned scenarios cover the vast majority of CL works, there are additional scenarios that deserve to be taken into consideration. For example, Task-Free/Task-Agnostic scenarios (Aljundi, Kelchtermans, et al. 2019) remove the assumption of knowing in advance the experience boundaries and do not provide task labels at test time. This scenario is also called data-incremental scenario (De Lange and Tuytelaars 2021). Moreover, Task-Free scenarios where data comes one sample at a time or in small mini-batches are called Streaming/Online Continual Learning scenarios (Aljundi, Lin, et al. 2019; Tyler L Hayes, Cahill, et al. 2019). These scenarios require to detect incoming drifts or to use a passive CL strategy which assumes that each new input may come from a drifting distribution. Online CL scenarios pose a serious challenge in the representation learning phase, since it is not easy to optimize NN models with a very small number of examples at a time. Other scenarios operate in batch mode, where the learning model is allowed to perform multiple epochs over the training dataset and where the training dataset contains a fairly large number of examples (e.g., tens-hundreds of thousands). Still, Online CL differs from traditional online learning in non-stationary environments, because the former assumes a separation between training and test set, while

the latter usually does not. The objectives are therefore different. Online learning focuses on optimizing the performance on incoming data (the present and the immediate future), while CL cares about preservation of existing knowledge (the past).

Meta-Learning Scenarios for CL. The field of Meta-Learning (Finn et al. 2017) has also contributed to the definition of a number of CL scenarios.
The Continual Meta Learning scenario (X. He et al. 2019) drives away from the strong focus of current scenarios on forgetting. Instead, it highlights the need for fast remembering of previously acquired knowledge. In case of a hidden context generating the current input distribution, the target associated to the same input may change if the hidden context drifts. Therefore, it is not possible to learn an optimal predictor which is immune to forgetting, since it is not possible to predict the correct target without knowledge about the hidden context. Moreover, inferring the context allows to leverage a predictor tailored to the current input sample. Meta-Learning approaches like Model-Agnostic Meta Learning (MAML) (Finn et al. 2017) fit the Continual Meta Learning scenario. When empowered with CL strategies, they improve their ability to quickly recover previously acquired concepts which have been lost in subsequent model updates.
Meta Continual Learning (Vuorio et al. 2018), instead, works the opposite way. The objective of this scenario is to apply meta-learning to CL strategies. In this way, CL strategies *learn to learn* (Hochreiter, Younger, et al. 2001) continuously how not to forget. The Meta Continual Learning scenario trains a CL strategy on a set of multiple streams. It requires the application of meta-learning algorithms to find a proper model initialization. Such learned initialization should be able to tackle a wide variety of different non-stationary distributions. However, this approach is very costly since the meta-training phase requires the model to be trained on many streams. Also, it is still unclear whether Meta Continual Learning approaches are able to generalize well to OOD streams. In fact, Meta Continual Learning approaches requires the meta-training set and the meta-testing set to be iid. As we have seen, such assumption cannot be met in CL.

From the description of the different scenarios given so far, it appears evident the predominance in CL of supervised problems, in particular classification tasks. This trend is slowly changing with the introduction of new problems, both supervised and unsupervised: from generative models to semantic segmentation and object detection. Still, the way of thinking about non-stationary environments in CL is often tied to supervised classification problems.

4.1.2 Continual Learning Benchmarks

The CL scenarios presented in the previous section provides a characterization of specific non-stationary environments and their properties. However, a scenario alone is not enough to define a CL experiment. In fact, the same scenario can be implemented with many different datasets.
The CL community has focused since the very beginning on the study of forgetting in supervised tasks, mostly image classification. As a consequence, the vast majority of popular CL benchmarks are tied to the Computer Vision field. The objective of this section is not to provide a complete review of existing benchmarks, but rather to offer a view on the different types of CL benchmarks present in the literature and on their distinctive characteristics with respect to the others.

Split Benchmarks for CI/TI Scenarios. Split MNIST (SMNIST) (van de Ven and Tolias 2018b) is one of the most popular CI benchmarks, due to the ubiquitous presence of MNIST in Computer Vision. MNIST (LeCun, Bottou, et al. 1998) is a dataset of 28x28 gray-scaled images of hand-written digits. SMNIST is constructed by taking the entire MNIST dataset and by splitting it in groups containing non-overlapping classes. The data stream is composed of 5 experiences, each of which provides examples related to 2 MNIST classes. The CL model is trained sequentially on each experience. It is tested on the entire MNIST test set or on a portion of it after training on each experience. Even this simple formulation causes complete forgetting for previous experiences. At the end of training, the model reaches a 20% final accuracy. That is, it correctly predicts only the last 2 classes out of the 10 present in the stream.
Along the same line, other CI benchmarks have been built out of different datasets: Split-CIFAR10/100 (Rebuffi et al. 2017; Maltoni and Lomonaco 2019), Split-ImageNet (De Lange, Aljundi, et al. 2021), Split-Omniglot (Schwarz et al. 2018) and so on. Any image classification dataset can fit the CI scenarios simply by taking it a subset of classes at a time. The number of experiences in the stream varies from benchmark to benchmark and, sometimes, even among different implementations of the same benchmark. For example, Split-CIFAR100 can be implemented with a stream of 10 experiences with 10 classes each, or with a stream of 20 experiences with 5 classes each. The number of experiences is one factor defining the difficulty of CL . On long streams, the mitigation of forgetting is more challenging. Also, long streams require an efficient CL strategies with bounded complexity.
TI benchmarks are usually built out of CI benchmarks, by associating a progressive task label to each experience. For example, we can build a TI Split MNIST by including task labels $0, 1, 2, 3, 4$ associated to each experience in the stream.

Benchmarks for DI Scenarios. For the DI benchmarks, Permuted MNIST (PMNIST) (I. J. Goodfellow et al. 2013) has a long history which predates CL. Permuted MNIST is adapted from the MNIST dataset by producing a stream in which each experience introduces *all* the MNIST examples. In each experience, pixels are modified by a fixed, random permutation which changes from experience to experience. Compatibly with the DI scenario, all the classes are introduced in the first experience and the model needs to adjust its ability to recognize those classes when encountering new examples. The test is performed on the MNIST test set, augmented with all the permutations encountered during training. A peculiarity of PMNIST is the fact that the number of possible experiences is very large. There exist (28x28)! different permutations for an image of size 28x28. This allows to build very long streams. As already mentioned, DI scenarios like PMNIST induce a smaller amount of forgetting compared to CI ones (van de Ven and Tolias 2018b). Nonetheless, since CL is not concerned only about forgetting, they provide an additional perspective on the performance of a CL agent. Currently however, DI benchmarks are still used to investigate forgetting. The idea behind PMNIST is borrowed by other DI scenarios, like Rotated MNIST (Lopez-Paz and Ranzato 2017). In Rotated MNIST, instead of a random permutation, each experience introduces a random rotation. This can be extended to any reasonable transformation on images and on any image classification dataset (Permuted CIFAR10/100, Rotated ImageNet and so on).

CL-specific Benchmarks. The benchmarks mentioned so far are all related to datasets built for image classification. There are also benchmarks specifically designed for CL. CORe50 (Lomonaco and Maltoni 2017) is a dataset for Continual Object Recognition composed by 50

classes of objects grouped into 10 categories (classification can therefore be performed at either class or category level). The peculiar characteristic of CORe50 is that images from each object have been collected from short videos recorded in 11 different sessions under different conditions (location, lightning, etc). The dataset offers one video for each session and for each object. In each video, the object is slightly manipulated by an external operator, with a point-of-view perspective. Such temporally-coherent representations of the same object may be exploited to improve the performance in case of sparse or no target labels (semi-supervised/unsupervised learning). The main disadvantage is that, when considered in terms of videos rather than images, the dataset is quite small (550 videos in total). This makes it difficult to treat the dataset as a dataset of time-series and to fully exploit temporal coherence. Nonetheless, the benchmark has been widely used in CL, especially for object recognition, object detection and semantic segmentation (since it provides bounding boxes and segmentation maps). There are multiple CL benchmarks proposed out of CORe50: a NC/CI version, a NI/DI version and a NIC version. Each version is associated to a specific set of runs, describing the order of appearance of the samples to guarantee full reproducibility.
Video benchmarks are quite interesting for CL, due to the high complex and non-stationary input space they cover. The OpenLoris benchmark (She et al. 2019) targets Visual Simultaneous Localization and Mapping (SLAM) problems for long-term robot deployment. The videos, taken from the point of view of a walking robot, exhibit drifts in the form of changed viewpoints, new encountered objects and novel environmental conditions. The OpenLoris-Object benchmark offers a similar setup but it targets object detection tasks. These benchmarks are associated mostly to DI/NI scenarios, since the change in environmental conditions can be considered as a way to produce new instances of existing classes. Stream-51 (Roady et al. 2020) is a dataset similar to CORe50. The CL benchmark built out of it however not only focus on object recognition from video frames but also on OOD detection at test time. In fact, the test set contains classes never seen during training, mixed with classes which the model has been trained on. Stream-51 works in online/streaming scenarios without task boundaries and task labels (task-free/agnostic). The resulting benchmark can be tuned to work in both NI and NC scenarios.
vCLIMB (Villa et al. 2022) proposes a way to blend together different video datasets for action recognition and to build a CL stream in which each new experience provides new action classes.

Simulators Benchmarks The world of simulators and procedurally-generated environment is very appealing for CL, since it enables to create a wide variety of different situations with different degrees of similarity with respect to the real-world. In such environments, most of the information can be controlled and modified at will. As an example, the Endless Procedural Driving Simulator (Hess et al. 2021) is a realistic urban-driving simulator based on the Unreal Engine[1] 4 to support high-resolution 3D representations of urban scenes from the point of view of a driver. As the driver goes on, the environment surrounding the car is subjected to different kinds of drifts, all procedurally-generated and fully controllable by the user: the weather can change both abruptly or gradually, new objects can be placed on the road/sidewalks, the surface of the objects in the environment can change. Moreover, the environment provides depth and surface annotations which can be useful to design proxy task for unsupervised learning.
While the Endless Procedural Driving Simulator is suitable to Continual RL tasks, its interface and API is mainly designed for supervised and unsupervised applications. The agent receives as input the stream of images and cannot directly control the car trajectory

[1] https://www.unrealengine.com/

with its own actions. Existing RL environments like Atari (Mnih et al. 2013) are used within CL: in a TI scenario, the agent must learn to play one game at a time from a sequence of multiple Atari games (Kirkpatrick et al. 2017). The drift is represented by the introduction of a new game, which represents the experience. Non-stationary environments can also be built inside a RL framework in a subtler way, instead of simply adding new types of completely disconnected environments. For example, the drift may affect the way reward are assigned or the way the environment manages the state transitions given the current action from the agent. In any case, RL represents a very interesting topic for CL. Unfortunately, given the difficulty of training RL agents in non-stationary environments, continual RL is still under-studied with respect to continual supervised learning.

More than Computer Vision. Outside the realm of computer vision benchmarks, CL lacks alternatives. Carta et al. 2022 proposed a benchmark to evaluate the effect of forgetting in graph classification tasks, where the objective is to classify inputs coming in the form of graphs. A graph g is defined as a tuple (V, E, X, A) where V is the set of nodes, E is the set of oriented edges connecting ordered pairs of nodes, X and A are node and edge features, respectively. Learning in structured data like graphs require to exploit the information present in the node-edge structure to solve the task. Carta et al. 2022 selected datasets coming from the graph learning community and adapted them to the CI scenario by dividing it into experiences containing non-overlapping classes. In order to provide connections with the CL literature, the authors leveraged SMNIST and Split-CIFAR10 in their super-pixels versions (Dwivedi et al. 2022). The graph for each image can be constructed by treating each super-pixel as node and by creating an edge if two super-pixels are adjacent. The third benchmark is the CI version of OGBG-PPA dataset (W. Hu et al. 2020), representing undirected protein association neighborhoods taken from protein-protein interaction graphs. The results show that, in CI scenario, Deep Graph Networks specifically designed to learn from graphs are not able to properly exploit the graph structure. Instead, they focus solely on the node features. In fact, an MLP which completely discards the structured information performs on par with the Deep Graph Networks. These CI benchmarks for graphs raise important question on the optimization process in CI scenarios outside the commonly used image recognition task. The work proposes a possible solution by applying a structure-preserving regularization to the Deep Graph Networks. The penalization forces the network to focus on the input structure and leads to a small improvement in the final performance. However, the benchmarks have not yet been fully explored and additional research is definitely required to validate this direction.

4.2 Evaluation

Offline, iid ML evaluates a model according to the ERM principles: the comparison between training and validation performance (e.g. accuracy) establishes the under-fitting or over-fitting regime of the model. The final assessment on unseen data (test set) establishes its generalization ability. In CL however, data are not entirely available at once and they drift over time. The objectives of the evaluation are therefore multifaceted and they require to adapt both the evaluation protocol (the way data are structured to perform evaluation) and the metrics (what is actually interesting to monitor).

Evaluation Protocols Following the ERM and the model assessment framework of ML, the most used evaluation protocol in CL creates a *train-test* split for each experience in the stream. The dataset D_i coming from experience e_i is split into non-overlapping

sets D_i^{train} and D_i^{test}. Sometimes, as in CORe50 (Lomonaco and Maltoni 2017), the test set is unique and consists in a single dataset D^{test}. The model is trained sequentially on D_i^{train} $\forall i = 1, 2, \ldots$. The frequency of testing may vary from case to case. At least, the model is tested on the test set of all the experiences after being trained on the last experience. In the case of a single test set, the performance is reported as a single number, while in the case of a test set for each experience the test performance can also be reported separately for each experience.

The evaluation protocol described above is consistent with the model assessment framework used in ML. The model selection, instead, poses fundamental challenges in CL. How can the best model be selected if the data is not available in advance? How can the best model resulting from model selection be trained on the stream of data if the data cannot be stored? On which set of data model should model selection be performed?

It would seem more natural to avoid model selection *tout-court*, since such process requires to know which data the model will see in the future, conditions forbidden by the non-stationary nature of the CL environment. However, the number of hyper-parameters present in many CL strategies requires a tuning phase. The literature has accepted the controversial idea of performing model selection on a single stream of data. Multiple models are trained on the same stream and then the best model is selected based on the average final performance on the test set.

In the context of mitigation of forgetting, the issue of over-fitting on the data is usually not taken into consideration. The model selection is performed on the same test set on which the final model assessment is carried out. While this would not be admissible in a ML setup, the problems behind forgetting are not easily overcome by memorizing the training data. The stability-plasticity dilemma also affects models which over-fit the data. More in general, finding a way to mitigate forgetting by performing model selection on the test set is not obvious. However, the CL community has rarely explicitly addressed this issue.

An evaluation protocol which takes into consideration model selection in CL is proposed in Chaudhry, Rohrbach, et al. 2019. Their protocol reserves a subset of the experiences for model selection and another subset for model assessment. In practice, instead of splitting each dataset D_i into train and test, the authors split the stream into training, validation and test experiences. The model selection is conducted on the stream $\mathcal{S}^{\text{CV}} = (e_1^{\text{CV}}, e_2^{\text{CV}}, \ldots, e_K^{\text{CV}})$ where CV stands for Cross-Validation (a type of model selection). Each experience e_i^{CV} is associated with the corresponding training and validation dataset $D_i^{\text{CV, train}}$ and $D_i^{\text{CV, valid}}$. The model assessment is conducted on the stream $\mathcal{S}^{\text{EV}} = (e_1^{\text{EV}}, e_2^{\text{EV}}, \ldots)$, where EV stands for evaluation. Critically, datasets present in any experience of $\mathcal{S}^{\text{CV}}$ are different from the ones present in the experiences of $\mathcal{S}^{\text{EV}}$. This guarantees that, compatibly with the ERM framework, data seen during model assessment is not seen during model selection and vice-versa. Although very robust, this evaluation protocol is rarely used in practice, since it requires to reserve K experiences for the model selection. The result being that the experiments are conducted on shorter streams/less data.

Metrics "When a measure becomes a target, it ceases to be a good measure". The warning expressed by the Goodhart's law (phrased by Strathern) must always be taken seriously.

CL experiments monitor the model performance with a different set of metrics, each of which targets one of the CL objectives. In the following, we will refer to the performance in terms of classification accuracy, even if the formulation of each CL metric is general enough to cover for different measures like the CE loss or the MSE loss and others.

The average training accuracy on the current experience serves as a sanity check to verify

whether the model is actually learning something or not during training. Similar to what happens in offline ML, this metric is also useful for comparison with the test accuracy on the same experience to detect model generalization on unseen data. The average training accuracy can be computed across a single minibatch or over an entire epoch.
The most popular set of CL metrics is defined in Lopez-Paz and Ranzato 2017. The authors defined a matrix R, where each element $R_{i,j}$ is the average accuracy on the test set of experience j after the model has been trained on experience i. Based on R, Lopez-Paz and Ranzato 2017 defined the Average Accuracy (ACC) as

$$ACC = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}. \tag{4.1}$$

This metric represents the average accuracy on the union of the test sets of each experience after the model has been trained on the last experience. As such, the larger the accuracy, the smaller the forgetting. ACC tells us the expected average performance on all the experiences. However, forgetting also depends on the test performance on a certain experience achieved when learning that same experience. A low value of ACC can indicate large forgetting or low performance during training. To formalize this concept and disambiguate, Lopez-Paz and Ranzato 2017 proposed the Backward Transfer (BWT) metric:

$$BWT = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}. \tag{4.2}$$

With respect to ACC, BWT compares the final performance obtained by the model with the one obtained during training ($R_{i,i}$). BWT provides a way to check whether training on new experiences hurts accuracy on the previous ones. As it is easy to see, a negative BWT value indicates forgetting, while a positive BWT value indicates that training on new information actually improves performance on previous experiences. While this may be surprising, it actually helps us to remind that one of the objectives of CL is also the exploitation of a positive interplay among different tasks. A positive BWT highlights exactly this phenomenon: achieving zero BWT is the best result as far as forgetting is concerned, but it still leaves lots of space for improvement for CL. Pushing BWT to positive values is usually very difficult and sometimes even impossible if two tasks are incompatible with each other.
Another important objective of CL is to understand how training on some information affects the performance on *future*, unseen data. To this end, Lopez-Paz and Ranzato 2017 introduced the Forward Transfer (FWT) metric:

$$FWT = \frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - \hat{b}_i, \tag{4.3}$$

where $\hat{b}_i$ is the test accuracy on experience i achieved by a model at random initialization. The idea behind FWT is to verify whether a model which has already acquired some knowledge during training is able to outperform a randomly initialized model on future data. A positive FWT value indicates that, indeed, the knowledge is transferred from previous to future experiences. On the contrary, a negative value indicates a strong interference and the incompatibility between the data present in the experiences. It is important to note that this formulation of FWT does not consider the notion of transfer with respect to a model *trained* on the same experience, but only with respect to a randomly initialized one. A different, perhaps more useful, objective may be expressed by $FWT^* = \frac{1}{T-1}\sum_{i=2}^{T} R_{i,i} - \hat{b}_i^*$, where $\hat{b}_i^*$ is now the performance of a randomly initialized model trained on experience i. This metric gives the relative improvement gained by a CL model with respect to an offline learner.

However, FWT^* is not currently used in the CL literature. A notable exception in Continual Reinforcement Learning is Wolczyk et al. 2021, where Forward Transfer on experience e_i is defined as $\frac{\text{AUC}_i - \text{AUC}_i^b}{1 - \text{AUC}_i^b}$, where AUC is the Area Under the Accuracy Curve, the subscript i refers to the metric computed after training on experience i and the superscript b refers to the metric computed for a randomly initialized model. This metric, similar to our proposal above, takes into consideration the performance *after* training the models on the experience e_i used for evaluation.
The Forgetting metric introduced in Chaudhry, Dokania, et al. 2018 mirrors the (negative) BWT formulation. Differently from BWT, Forgetting considers as reference performance (last term of Eq. 4.2) the maximum test accuracy achieved on an experience during any point in the learning trajectory. This difference points to the fact that the performance of the model on experience i may increase after training on future experiences. In practice, this is rarely the case in current CL experiments, and these two metrics are often one the negative of the other.
Similarly to FWT, the Intransigence metric defined by Chaudhry, Dokania, et al. 2018 indicates the inability of a model to learn new information. The Intransigence I_k for an experience k is computed by $I_k = R_k^* - R_{k,k}$. The term R_k^* represents the accuracy obtained by a model on the test set of experience k after being trained on the training set of *all* experiences in an iid manner. With respect to FWT, Intransigence does not use a randomly initialized model for comparison, but a model trained offline, thus providing a complementary measure.

The metrics proposed by Díaz-Rodríguez et al. 2018 also include the efficiency of the learning process in terms of model size and sample storage size. The Model Size (MS) metric checks whether the number of parameters used to learn the first experience grows in subsequent experiences. Considering $|\boldsymbol{\theta_i}|$ the number of parameters of the model at experience i, MS monitors the quantity

$$MS = \min\left(1, \frac{1}{N}\sum_{i=1}^{N}\frac{|\boldsymbol{\theta_1}|}{|\boldsymbol{\theta_i}|}\right). \tag{4.4}$$

Similarly, the Sample Storage Storage Size (SSS) monitors the external memory M growth over time by computing

$$SSS = 1 - \min\left(1, \frac{1}{N\sum_{i=1}^{N}|D_i^{\text{train}}|}\sum_{i=1}^{N}|M_i|\right). \tag{4.5}$$

The MS and SSS metrics monitor two conditions that, as we will see in the next section, are very important for many CL strategies. Checking whether a strategy heavily exploits the model size or the external memory is an easy way to understand the feasibility of the strategy in different environments. For example, in environments with strict constraints on the amount of memory resources, a strategy with large MS may not be applicable.

4.3 Strategies

CL strategies are mostly centered around the mitigation of catastrophic forgetting in NNs (De Lange, Aljundi, et al. 2021; Masana et al. 2020). Since Computer Vision received lots of attention by the CL community, many of the existing strategies are tested in vision benchmarks, like the ones described in 4.1.2. However, the applicability of CL strategies is usually much more general and makes them effective on a number of NNs architectures and

domains, even outside Computer Vision.
The popular review by Parisi, Kemker, et al. 2019 identifies three main families of CL strategies: regularization strategies, architectural strategies and rehearsal/Replay strategies. We will go through the main representatives for each of them and conclude with a presentation of hybrid strategies based on the combination of two or more families of strategies.
CL experiments usually compare the performance of novel strategies with a lower-bound performance given by the *Naive* strategy, and an upper-bound performance given by the *Joint Training* strategy. The Naive strategy simply fine-tunes the model on new experiences without any CL technique. Therefore, in terms of forgetting Naive achieves its largest amount and it is therefore considered as a lower bound. Joint Training is not a CL strategy since it trains the model *offline* in a iid way. Joint Training concatenates the data of all the experiences together and trains the model in one-shot, as in ML. In terms of forgetting, Joint Training provides an upper-bound since iid training does not suffer from the problem. However, the final performance of Joint Training could be in principle surpassed by a CL strategy due to positive backward/forward transfer. In practice, it is very challenging to achieve a positive forward and backward transfer and forgetting in CL still plays a major role with respect to the other two metrics.

4.3.1 Regularization Strategies

Regularization strategies (Parisi, Kemker, et al. 2019) aim at balancing the stability-plasticity trade-off by imposing more stability on the model through loss regularization. The way the penalty term is built depends on the specific method and on the kind of stability one wants to achieve.

Elastic Weight Consolidation. One of the first regularization strategies is Elastic Weight Consolidation (EWC) (Kirkpatrick et al. 2017). EWC estimates the importance of each parameter θ and then imposes a regularization such that parameters that are important for previous experiences change less than others when learning the current experience. We will call this class of strategies *importance-based* regularization strategies. The general form of a loss $\mathcal{L}_{\boldsymbol{\theta}}^{tot}$ in importance-based regularization strategies is given by

$$\mathcal{L}_{\boldsymbol{\theta}}^{tot}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \mathcal{R}_{\boldsymbol{\theta}}, \tag{4.6}$$

where the term $\mathcal{R}_{\boldsymbol{\theta}}$ acts as the regularizer, weighted by the hyper-parameter λ. In EWC, the parameters importance is estimated at the end of training on each experience e_i. The model performs an additional forward and backward pass on D_i^{train}. The objective of this phase is not to update the parameters, which are kept fixed, but to compute the average gradient vector on the dataset samples. The importance vector associating an importance value to each parameter is represented by the average squared gradient computed during this additional phase. Formally, the importance vector $\boldsymbol{\Omega_i}$ of parameter vector $\boldsymbol{\theta}$ for experience e_i is computed by

$$\boldsymbol{\Omega_i} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y} \sim D_i^{\text{train}}} \left[\nabla \mathcal{L}_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \nabla \mathcal{L}_{\boldsymbol{\theta}}^T(\hat{\boldsymbol{y}}, \boldsymbol{y}) \right], \tag{4.7}$$

where $\hat{\boldsymbol{y}} = h_{\boldsymbol{\theta}}(\boldsymbol{x})$ with model $h_{\boldsymbol{\theta}}$. The outer product defines a positive semi-definite matrix. By strictly following Eq. 4.7, the elements of the matrix $\boldsymbol{\Omega_i}$ should be computed by averaging over each single sample of the dataset. However, in practice, approximating this result by using a mini-batch of more than one sample does not hurt performance and increases the computational efficiency. The importance vector can be considered as an approximation of the diagonal of the Fisher Information Matrix. Since the off-diagonal elements of the matrix are considered to be zero, one underlying assumption of EWC is the independence of each

parameter with respect to all the other parameters.
In order to tackle multiple experiences, EWC saves one importance vector $\boldsymbol{\Omega_i}$ for each experience i. It also saves the parameters $\boldsymbol{\theta}^k$ of the model at the end of training on each experience k. The regularizer term used when training on experience i can be expressed by:

$$\mathcal{R}_{\boldsymbol{\theta},i} = \sum_{k=1}^{i-1} \frac{1}{2} \sum_{j=1}^{|\boldsymbol{\theta}|} \boldsymbol{\Omega_k} (\theta_j^i - \theta_j^k)^2. \tag{4.8}$$

EWC memory complexity grows as $\mathcal{O}(T|\boldsymbol{\theta}|^2)$, where T is the number of experiences. Online EWC (Schwarz et al. 2018) mitigates this cost by keeping a single importance vector. The importance vector is updated at the end of each experience. After training on experience i, the temporary importance vector $\hat{\boldsymbol{\Omega}}_{\boldsymbol{i}}$ is computed by Eq. 4.7. The result is used to produce the final importance vector $\boldsymbol{\Omega_i} = \hat{\boldsymbol{\Omega}}_{\boldsymbol{i}} + \alpha \boldsymbol{\Omega_{i-1}}$. The hyper-parameter α allows to control the degree of forgetting of previous experiences. Large α is associated to a large regularization effect for previously seen experiences. A small α gives more importance to recently seen experiences.

A similar version of Online EWC, called EWC++, is designed by Chaudhry, Dokania, et al. 2018. EWC++ uses a single importance vector but, unlike Online EWC, it updates the importance for the current experience after each training iteration. EWC++ removes the requirement of an additional forward pass at the end of each experience. Similarly to EWC++, Synaptic Intelligence (SI) (Zenke et al. 2017) computes a single importance vector online by updating it at each training iteration. Instead, Memory Aware Synapses (MAS) is able to compute importances in an unsupervised way, without needing targets from the training set. In fact, the gradient is estimated at the output layer and not after computing the loss. In MAS, the gradient estimates the sensitivity of the output layer to changes in the parameters, providing an alternative characterization of the importance.
Importance-based regularization strategies work well in TI and DI scenarios, while they are well-known to not work in CI scenarios (Timothée Lesort, Stoian, et al. 2020), due to their inability to deal with the interference at the output layer.

Learning without Forgetting. The branch of importance-based regularization strategies is not the only one pursued by researchers. For example, regularization strategies based on distillation (Hinton et al. 2015) are quite popular and effective also in CI scenarios (Z. Li and Hoiem 2016; Jung et al. 2018; Dhar et al. 2019; Douillard, Cord, et al. 2020b). They are often combined with other families of strategies (Buzzega et al. 2020; Douillard, Cord, et al. 2020a), giving rise to hybrid approaches.
Learning without Forgetting (LwF) is the first distillation-based approach within regularization strategies. It works by keeping a copy of the model $h_{\boldsymbol{\theta}^{i-1}}$ at the end of training on the previous experience e_{i-1}. During training, the loss is augmented with a distillation penalty which keeps the predictions of the current model $h_{\boldsymbol{\theta}^i}$ close to the ones of the saved model. Predictions of the previous model on the current input are taken as soft targets to train the current model. The previous model is kept fixed when training the current one. The LwF regularizer has the following form:

$$\mathcal{R}_{\boldsymbol{\theta}} = -\sum_j \hat{\boldsymbol{p}}_{i-1}^j \log \hat{\boldsymbol{p}}_i^j, \tag{4.9}$$

where j indexes the samples in the current dataset D_i and each prediction probability $\hat{\boldsymbol{p}}_i^j$ is computed from the logits $\hat{\boldsymbol{y}}_i^j$ with a softmax with temperature T. It is important to note that LwF computes the output of the previous model $\boldsymbol{p}_{i-1}$ on the samples of the *current* experience. The regularizer tells the current model predictions $\boldsymbol{p}_i$ to stay close to the ones

produced by the previous model.

For all regularization strategies, the role of the hyper-parameter λ (Eq. 4.6) associated to the regularizer is fundamental. Large values of λ produce a strong regularization effect which might result in low accuracy values on the current experience. On the contrary, small values of λ will result in forgetting of previous knowledge. Despite the existing critiques against model selection in CL (Section 4.2), the optimal value of λ needs to be determined by a model selection procedure.

4.3.2 Replay Strategies

One of the most important constraints in any CL scenario is that training data cannot be entirely stored in memory for future re-use. However, storing a *subset* of examples into the external memory M (called *Replay memory*) is often allowed and not considered as a complete violation of the CL assumptions. This idea, although quite simple, is very effective. Most of the best performing CL strategies, especially in CI scenarios, are strategies based on Replay. Due to the connections between RL and Replay (Mnih et al. 2013), where it was first introduced, Replay approaches have also been used in Continual RL environment (Isele and Cosgun 2018; Riemer et al. 2019; Rolnick et al. 2019).

Experience/Random Replay. At each training iteration, Replay strategies concatenates to the current mini-batch a sample of examples taken from the memory M. The resulting, augmented, mini-batch is used for training. In this way, the model is able to preserve the performance on previous examples, since it is not trained only on new ones but also on previous ones stored in M. Obviously, the larger the memory, the better the performance in terms of forgetting. However, using very large memories is in stark contrast with CL principles. Even if there is no general agreement on when a memory is *too large*, existing approaches show the impact of the memory size on the finale performance. An effective Replay strategy should exhibit a strong performance also with low memory values (e.g., few samples stored per class).
The Replay memory management poses different challenges: how to select examples from the current training set to be added to the memory (*storage policy*)? How to manage the memory size over time? How to sample from the memory (*retrieval policy*)? For classification tasks, memory sampling is usually performed randomly on a per-class basis. At each iteration, the mini-batch of samples from the current experience is concatenated with a mini-batch sampled from the memory containing K samples per previous class. Therefore, the model sees at each iteration at least one example for all the previous classes. The sample size is often chosen to be equal to the size of the current mini-batch. If the model sees a mini-batch of 64 examples, 32 will be related to the current experience, while the other 32 will be sampled from the memory. Aligned with the original Class-Incremental constraints (Rebuffi et al. 2017), the memory size is kept constant over the number of experiences. As a consequence, when new classes are added to the memory, the amount of examples per class K stored in the memory is given by $K = S/C$, where S is the constant memory size and C is the number of classes currently stored in the memory.
The procedure we just described is called Random Replay (RR) and it is one of the most common baselines used in CL experiments. More advanced Replay strategies focus on the storage policy by adding only those examples which are considered more informative for the current experience (Aljundi, Rohrbach, et al. 2019). Rosasco et al. 2021; Merlin et al. 2021 followed the same approach by leveraging Dataset Distillation (T. Wang et al.

2020) to condense the training set of the current experience into few, very informative patterns. Using these patterns for Replay improves the performance with respect to RR in the few-samples regime, even with only 1 sample per class. Similar results are also shown by Chaudhry, Ranzato, et al. 2019. The fact that Replay retains most of its effectiveness even with low memory requirements has been used to apply different Replay approaches in streaming/online environments, where samples come one at a time. One notable example is the REMIND approach (Tyler L Hayes and Kanan 2020), which exploits compressed memory representations to perform Replay in streaming scenarios. There are still many open questions related to the behavior of different Replay strategies under different experimental conditions. We studied some of these open challenges in a broad empirical evaluation on Replay (Merlin et al. 2021). While Replay remains one of the most effective approaches in CL, it may also reduce the performance in certain cases. In fact, the model can sometimes overfit on the Replayed samples (Verwimp et al. 2021), hurting the model generalization on unseen data. This behavior is reflected in values of BWT close to zero (low forgetting), but also in sub-optimal values of ACC (low overall accuracy). The research on Replay strategies is far from being completed. The role of memory consolidation played by Replay in the human brain (Tyler L. Hayes et al. 2021) can perhaps provide a grounded way of designing new, more effective Replay strategies.

Generative Replay. Storing previous examples for future re-use proved to be very effective against the mitigation of forgetting. However, storing *exactly* the same samples seen during training may not be necessary. On the contrary, as showed by Rosasco et al. 2021, *any* sample which approximates the input distribution is actually a good Replay sample. Starting from this idea, Generative Replay approaches (Shin et al. 2017; van de Ven and Tolias 2018b) substitute the external memory M with a generative model, like a Generative Adversarial Network (I. Goodfellow et al. 2014). The generative model is in charge of generating examples that are then used as Replay examples and concatenated to the current minibatch. If a single generative model is used, the storage cost still remains fixed with respect to the number of experiences. The diversity of the generated examples can instead increase with respect to Replay, if the generative model does not suffer from mode collapse. However, the problem of forgetting is moved from the predictive model to the generative model. Unfortunately, training a generative model continuously is not an easy task. If the model is trained sequentially on multiple experiences, the generated samples at each experience will be mostly associated with the last experience, thus vanishing the Replay effect. Storing one generative model per experience, although effective, would result in a linear memory cost with respect to the length of the stream. Generative Replay also struggles when scaling to high-dimensional, natural images. While it constitutes an appealing solution, designing effective generative Replay strategies remains an open challenge in CL.

Latent Replay. Any NN is composed by a stack of layers, whose output is the input of the next. Replay stores the raw input data $\boldsymbol{x}$ into an external memory M and trains all layers of the model with that information. However, there is also the possibility to store, instead of the raw input, a set of *latent activations* of the model into M. In practice, instead of Replaying the input starting from the first layer, the latent activation can be fed as input to the hidden layer which produced it, and continue learning from that point. This usually requires the layers before the selected hidden layer to be frozen or updated with a smaller learning rate. Latent Replay (van de Ven, Siegelmann, et al. 2020b; Pellegrini et al. 2020; Ostapenko et al. 2022; Graffieti et al. 2022) also better reflects the bio-inspired motivation of Replay, since the hippocampus in the human brain is well-known to actively

participate in the plasticity of neural pathways through re-activation of previous (latent) memories (Tyler L. Hayes et al. 2021). Latent Replay "splits" the network into two parts: the one including layers up to the chosen latent one, and another including layers from the target latent one up to the classifier. While deciding which latent layer to select for Replay and how to treat the rest of the architecture is not obvious, latent Replay presents some advantages when compared to generative Replay. In particular, latent Replay does not require to train a generative model, a procedure which is always costly and difficult to tune. It also does not transfer the problem of forgetting from the predictive model to a second one. Ultimately, latent Replay represents a promising technique to perform Replay without storing raw samples from the stream of data.

4.3.3 Architectural Strategies

Architectural strategies (Parisi, Kemker, et al. 2019) are a vast and diverse branch of CL strategies. The other two families of approaches presented so far provide a specific direction along which develop new methods. Architectural strategies, on the contrary, only share the idea of tweaking the model architecture to mitigate forgetting. Usually, architectural strategies operate by expanding the model capacity during learning in order to allocate more units/layers for incoming experiences (Draelos et al. 2017). At the same time, in order to balance the stability-plasticity trade-off, previous components can be frozen and new modules can be sparsely connected with the existing ones. Architectural approaches need to devise a policy to expand the model capacity and a policy to compress the model representations, to keep the memory and computational cost bounded.
The fine-tuning of pre-trained models can be seen as a sort of architectural strategy, where the feature extractor component is kept fixed and only the classifier is updated continuously. In fact, pre-trained models are widely used in CL and recent results show that they provide an advantage in terms of forgetting with respect to a model trained from scratch on a stream of data.

Multi-Head. Another simple type of architectural approach is the multi-head approach (De Lange, Aljundi, et al. 2021). Multi-headed models allocate one output layer for each task separately. In order to do that, the CL scenario must provide task labels associated to each input at both training and test time. Otherwise, the CL strategy needs to infer the task label, for example by using expert detectors (Aljundi, Chakravarty, et al. 2017). Multi-head approaches remove forgetting in the output layer by design, since the representations at the output level do not interfere with each other. However, task labels are rarely available in practice and inferring them is not an easy task and may produce blatant errors: a wrong task label would lead to the usage of the wrong head, which would generate a wrong prediction.

Copy-Weight with Reinit. Working on the output layer is a very effective way of mitigating forgetting in CI scenarios, since the output layer is the part of the network suffering the most from forgetting (Timothee Lesort, George, et al. 2021). Copy-Weight with Reinit (CWR*) (Lomonaco, Desai, et al. 2020) is an architectural approach which does not use any task label information. It leverages a fixed pre-trained model as feature extractor and two sets of output layer weights, initialized to the zero vector: the consolidated weights *cw* and the temporary weights *tw*. The main idea is to train the temporary weights on the current minibatch and then to consolidate the learned information into *cw*. When learning *tw*, only the weights related to the classes present in the mini-batch are updated, while the others are kept fixed. This does not require any additional information apart from the supervised signal provided by the target classes. Then, the model performs inference by using the consolidated

weights *cw*. Applying CWR* with an adaptive feature extractor, however, often results in a reduced performance, since the statistics of the activations produced by the feature extractors change over time.

Deep Streaming Linear Discriminant Analysis. Similarly to CWR*, Deep Streaming Linear Discriminant Analysis (DSLDA), works on the model output layer and requires a fixed feature extractor. DSLDA is known to be a very effective strategy in CI scenarios, since the output layer is not trained with SGD. Instead, it performs a Linear Discriminant Analysis in the feature space produced by the pre-trained model. Since the model is fixed, the mean and covariance statistics gathered during training are reliable and produce strong "centroids" which effectively cluster the target classes. CWR* and DSLDA both suffer from limited expressiveness since they can only detect linear relationships in the feature space and cannot update the representations produced by the pre-trained model.

Progressive Networks. The progressive network by Rusu et al. 2016 is an architectural approach which applies the multi-head approach to the whole model. Progressive networks employ a NN (called a *column*) for each task. Columns are connected by adapters, which should provide knowledge sharing and positive forward transfer. However, each column is connected to all previous others, resulting in a quadratic cost in memory with respect to the number of tasks. Progressive networks rely on task labels from the CL scenario to generate the predictions, which, as we have discussed, is a problematic assumption in many real-world cases.

Sparse Architectures. Sparse architectural methods are also a very active area of research. SpaceNet (Sokar et al. 2021) achieves adaptive training of sparse representations by allocating a sub-network for each task and by keeping fixed the connections related to previous tasks. The training procedure also compresses the amount of weights requested by a task such that the network is able to scale to many experiences. Similarly to importance-based regularization approaches, sparse approaches also estimate the importance of a weight for a given task in order to understand which weight to prune to make space for future learning.

4.3.4 Hybrid Strategies

Regularization, Replay and architectural strategies all provide different principles with which combat forgetting in NNs. Given their formulation, nothing prevents the combination of multiple families into a single strategy. These *hybrid* strategies are often very effective, since they exploit the advantages provided by multiple CL strategies while at the same time mitigating their disadvantages.

Progress & Compress. Progress & Compress (P&C) (Schwarz et al. 2018), for example, is based on an architectural approach with two networks combined with an online version of EWC and distillation. Therefore, P&C represents an architectural-regularization hybrid strategy. P&C employs two networks: a knowledge base network and an active network. The active network is trained on the current task and then it is distilled into the knowledge base. The knowledge base, which represents the long-term knowledge, mitigates forgetting via EWC.

Architectural-Regularization. AR1* (Lomonaco, Desai, et al. 2020) is a hybrid strategy combining architectural and regularization approaches. The output layer of AR1*

is managed by CWR*, while the update of the feature extractor is performed via Synaptic Intelligence. Due to the usage of CWR*, AR1* performs best when the model is initialized with a pre-trained model. Under this condition, AR1* is one of the best performing method in CI scenarios, highlighting the effectiveness of a proper combination of different CL strategies.

(Average) Gradient Episodic Memory. Gradient Episodic Memory (GEM) (Lopez-Paz and Ranzato 2017) combines Replay strategies with regularization strategies. Examples in the Replay buffer are not used in combination with the current ones to update the model. Rather, they are used to project the gradient signal computed on the training examples in the orthogonal direction with respect to the gradient computed for the Replay examples. Since orthogonal directions do not interfere with each other, the amount of forgetting experienced by the model is greatly reduced. Computing the new direction amounts to solve a constrained optimization problem for each experience e_t of the form:

$$\boldsymbol{\theta_t^*} = \arg\min_{\boldsymbol{\theta_t}} \mathcal{L}(h_{\boldsymbol{\theta_t}}(\boldsymbol{x}), \boldsymbol{y}), \tag{4.10}$$

$$\text{subject to } \mathcal{L}(h_{\boldsymbol{\theta_k}}, M_k) \leq \mathcal{L}(h_{\boldsymbol{\theta_{t-1}}}, M_k),\ \forall k < t. \tag{4.11}$$

GEM computes the optimal parameters $\boldsymbol{\theta_t^*}$ up to experience t with the additional constraint that the loss on the examples in memory must not increase. If the examples approximate the original distribution, then forgetting is effectively mitigated. The authors formulated a dual Quadratic Optimization problem in which the number of unknowns scales with the number of experiences. While affordable in terms of memory, the computational time required to solve the quadratic optimization remains large. Therefore, GEM does not scale well to a large number of experiences.
Average GEM (A-GEM) (Chaudhry, Rohrbach, et al. 2019) aims to surpass the computational cost of GEM by approximating the optimization problem described in 4.10. Instead of forbidding a loss increase for each previous experience ($k < t$ in Eq. 4.10), A-GEM samples a subset of examples from the memory and imposes the constraints only on those examples. This leads to a closed form projection of the gradient computed by:

$$\tilde{\boldsymbol{g}} = \boldsymbol{g} - \frac{\boldsymbol{g}^T \boldsymbol{g_{ref}}}{\boldsymbol{g_{ref}^T g_{ref}}} \boldsymbol{g_{ref}}, \tag{4.12}$$

where g is the current gradient computed on the current training samples and g_{ref} is the gradient computed on the sample from the memory. The resulting gradient $\tilde{g}$ provides the final gradient direction. While computing the updated gradient $\tilde{g}$ is very quick, the approximation resulting from the fact that g_{ref} is computed only on a sample from the memory heavily impacts on the A-GEM performance with respect to GEM. A-GEM requires task labels, while GEM does not. A-GEM also introduces a task vector information as input to the model. The task vector describes the current task tackled by the model (it can be as simple as a simple numerical index up to a learned embedding representative of the task objective).

iCarl iCarl (Rebuffi et al. 2017) is a hybrid strategy combining Replay and regularization approaches. iCarl also leverages a nearest-mean classifier as output layer. iCarl adds examples to the Replay memory one at a time. At each iteration and for each class, iCarl computes the average feature vector of the Replay examples concatenated to one training example. It then compares the feature vector with the average feature vector for the training examples. Finally, iCarl adds to the memory the training example which makes these two vectors more similar. Therefore, iCarl implements the Replay memory as a prioritized list and the removal

policy follows a First-In-First-Out policy. Moreover, the loss function of iCarl consists of both a CE loss for classification and a distillation-based regularization to preserve the behavior for previous classes.

The short review of the main families of CL approaches we presented in this Section show that, currently, there is no winning strategy in CL. Each strategy has its own advantages and disadvantages, with different sets of assumptions. Solving a CL task requires to understand the principles behind each strategy and to carefully choose the one which better fits the constraints and the characteristics of the environment.
A CL experiment requires a careful implementation of strategies, benchmarks, evaluation protocols and metrics. A crucial factor for the consolidation of a fast-growing research topic like CL is the availability of common tools that can ease the implementation of all the steps of a CL experiments and ensure a fair comparison of results (Pineau et al. 2020).
The CL community has put much effort into addressing these issues, by providing shared code bases and libraries aimed at increasing the reproducibility of CL experiments (De Lange, Aljundi, et al. 2021; Hsu et al. 2018; Masana et al. 2020; Serra et al. 2018; van de Ven and Tolias 2018a; van de Ven and Tolias 2018b). However, these first attempts lack the ability to create different and complex benchmarks and are not backed-up by the support of a large community. One notable exception is the Continuum library (Douillard and Timothée Lesort 2021) which provides a flexible API to build CL benchmarks, but which does not offer any programming abstractions to implement CL strategies and algorithms. This thesis is largely based on the Avalanche library Lomonaco, Pellegrini, et al. 2021, developed and supported by the ContinualAI community[2]. The author of this thesis is one of the lead contributors of the library and co-author of the paper. A complete description of the library is provided in Appendix C.

[2] `https://www.continualai.org`

Part II

Beyond Class-Incremental Scenarios

Chapter 5

Motivation

The space of non-stationary environments is very large. Different types of distribution drifts give rise to different environments, with specific properties which can be exploited to learn continuously. Also, each environment provides a set of constraints which the CL agent cannot violate. As a result, the same agent learning in different non-stationary environments can obtain very different performance. Since it is not possible to evaluate an agent on the complete space of all environments, one needs to consider very carefully the impact each constraint will have on the agent. Moreover, the performance of an agent learning in a specific scenario may change when deploying the agent in another one. In order to acquire a deep understanding of the performance of a CL strategy, it is necessary to study it under different constraints.

How broad is the current understanding of CL strategies? Do we know how a given CL strategy would perform in different environments and scenarios? To answer these questions, let us consider the popular CI scenario (Rebuffi et al. 2017; van de Ven and Tolias 2018b). The properties and constraints of CI scenarios are very simple. Since the scenario is very well studied, most of the tools required to build CI benchmarks require minimal effort from the researcher. This is fundamental to standardize the experimental setup, to compare results across different works and, ultimately, to make progress in a rigorous way. Moreover, the constraints of CI scenarios are shared across many other CL scenarios, with the exception of DI scenarios. In fact, TI scenarios are often implemented as CI scenarios with task labels availability at training and test time. Data-incremental, streaming/online and task-free scenarios are also often based on CI configurations: they remove the boundaries between experiences and consider single-pass training with few examples at a time.

Since CI has received so much focus, it is worth asking: *"is Class-Incremental a scenario well grounded in real-world data streams?"*. Our answer is a negative one. This Part is devoted to explain the reasons behind our answer. We believe CI is an interesting tool for CL, but we also consider it a tool that needs to be handled with caution.

For example, it is interesting to note that almost all CL strategies are designed and tested to work in CI scenarios. At the same time, all these strategies are considered and called *CL* strategies, not CI strategies. While this might seem a minor detail, it actually contributes to the confusing overlap between CL and CI scenarios. We should start from the very simple statement that the latter represents just one, and very specific, instance of a CL stream.

Historically, the role of CI has been fundamental to the development of CL. CI scenarios introduce constraints that surreptitiously induce a large amount of forgetting in a CL agent. Forgetting is by far one of the most challenging issues in CL and it is therefore only natural that many researchers started to be interested in how to mitigate it. CI scenarios provided the perfect test-bed to develop and assess new CL strategies for this objective. However, in

many real-world data streams drifts are not as abrupt and strong as in CI scenarios. Admittedly, encountering completely new object classes is a rare phenomenon. More importantly, preventing repetition of previous classes seems an arbitrary constraint. A convenient one though, when the objective is to study forgetting.
We believe that different CL scenarios can provide an alternative and complementary view of what it means to learn continuously. Currently, our understanding of the performance of many CL strategies is restricted to a very specific class of scenarios. We do not know what the performance would be if the same strategies were to be implemented in a different environment. In this Part, we will introduce and study novel CL scenarios which differ from CI. We will also provide strong empirical evidence about interesting properties of each new scenario, and show in which ways CL strategies are able to cope with them.

In Chapter 6 we introduce the family of Class-Incremental with Repetition (CIR) scenarios. Unlike CI scenarios, CIR scenarios are not based on a stream of mutually exclusive classes. In CIR, samples from one class can reappear later on in times. This property is almost neglected in CL, with few exceptions (Lomonaco, Maltoni, and Pellegrini 2020; Stojanov et al. 2019) which however do not take general conclusion but focus on one specific dataset (CoRE50 in Lomonaco, Maltoni, and Pellegrini 2020 and CRIB in Stojanov et al. 2019).
The repetition present in CIR is different from the one implemented by Replay strategies. Replay repeats samples in an even, balanced way between all the classes previously encountered. Also, Replay allows to carefully choose different policies to update and use the external memory buffer. Repetition in CIR does not rely on an external buffer and they cannot be controlled. It is the environment that generates them (*natural repetition*). This results in a variety of different CIR scenarios which can be implemented by changing the characteristics of the repetition, like the frequency and amount of repeated samples, the distribution of the repeated classes and so on. The presence of repetition better fits many real-world data streams in which objects and situations appear (in the same or in a different form) over and over again. Neglecting this simple fact exacerbates a problem, the *catastrophic* forgetting, which might not be that impactful in the real-world. We are interested in the study of CIR scenarios from the perspective of both existing CL strategies and the design of new ones. Our main objective is to understand whether the presence of repetition makes current CL strategies less effective, given the fact that they mainly focus on the mitigation of forgetting. In case, we are interested in designing specific solutions that exploit the repetition to improve the final performance.

All the CL scenarios described in Section 4.1 offers a single stream of data on which the CL model is trained. The data stream fully describes the tasks to be solved. One important paradigm in modern ML is the pre-train fine-tune setup, where a model is first pre-trained on an unlabeled corpus of data and the fine-tuned on a (usually smaller) dataset to solve a given task. In the pre-train phase the model acquires general features which do not depend on a specific task. In the fine-tuning phase, the model features are adapted towards the target task. This framework has achieved a remarkable success. However, CL still blends together the representation learning phase (pre-training) with the supervised learning of a task (fine-tune). We believe this to be another important source of forgetting. The CL model trained on the first experience of a stream will tend to build its features based solely on the objective given by the first experience. The resulting features are likely to over-fit to the task, instead of capturing general relationships in the input data. For example, in the Split MNIST benchmark a CL model trained to classify the 0 and 1 digits (first experience) is not required to really understand the shape of the digit 0 and 1. It just needs to find those few pixels telling the two digits apart. When a new experience arrives, the features learned for 0

and 1 are not sufficient to quickly learn to classify 2 and 3. The model needs to erase previous knowledge and start over, since there can be no transfer when features are specific to a given task. In Chapter 7, we study the Continual Pre-Training scenario, where a pre-trained model is kept updated over time by applying the pre-training objective over a stream of data. The pre-trained model can therefore update its internal representations whenever new data is available. For example, it can be updated to include information about recent news which were not available during the previous pre-training stages. After any of these additional pre-training steps, the model can be fine-tuned to tackle one or more tasks. Continual Pre-Training decouples the feature representation stage from task learning. Our objective is to understand what happens to the performance on down-stream tasks during fine-tuning when the pre-trained model changes continuously. From a CL perspective, one would expect the performance of earlier pre-trained models to be better on tasks where they were originally pre-trained on. Additional pre-training stages should deteriorate the fine-tuning performance. We observe that supervised pre-trained models show indeed a decrease in the performance. Instead, self-supervised pre-trained models are able to effectively mitigate forgetting.

Our results point out that the design of novel CL scenarios is fundamental to truly understand the impact of forgetting on CL models. While in some scenarios forgetting still remains one important issue to face, in other ones forgetting is naturally mitigated. CIR and Continual Pre-Training scenarios highlight the need to focus on new challenges, beyond catastrophic forgetting in CL.

Chapter 6

Class-Incremental Learning with Repetition

In this Chapter we present a new scenario for CL, which we call Class-Incremental with Repetition (CIR).
The Chapter is structured as follows:

- We highlight the limits behind the CI scenario (Section 6.1).
- We formalize the CIR scenario by providing a description of possible data-stream generators with repetition (Section 6.2).
- We design a specific Replay strategy to mitigate forgetting in CIR (Section 6.3).
- We empirically validate the effectiveness of the proposed approach as well as the behavior of existing strategies (Section 6.4).
- We close the Chapter by summarizing the experiments with CIR and the impact the scenario may have on CL research.

Section 6.1 is based on the published paper Cossu, Graffieti, et al. 2022. The other Sections of this Chapter are based on the paper (under review) Hemati et al. 2023.

6.1 Is Class-Incremental Enough for Continual Learning?

In the CI scenario, each experience e_i contains a dataset $D_i = \{(\boldsymbol{x_j}, \boldsymbol{y_j})\}_{j=1,\dots,N}$, where $\boldsymbol{x_j}$ is the input pattern and $\boldsymbol{y_j}$ is its target class. As discussed in Section 4.1, the peculiar characteristic of CI scenarios is that they partition the target class space by assigning a disjoint set of classes to each experience. Formally, be $\mathcal{C} = \{c_k\}_{k=1,\dots,C}$ the set of all classes seen by the model during training and be $\mathcal{C}_i$ the subset of classes present in experience e_i, a CI scenario must satisfy the following condition:

$$\mathcal{C}_i \bigcap \mathcal{C}_j = \emptyset, \quad \forall i \neq j. \tag{6.1}$$

We will refer to the constraint expressed by Equation 6.1 as the *no repetition* constraint. It simply states that classes present in one experience are never seen again in future experiences or, likewise, that each class is present in one and only one experience.

CI is, by far, the most studied CL scenario in the literature. As a result of the no repetition constraint, CI scenarios exacerbate catastrophic forgetting, making its mitigation particularly

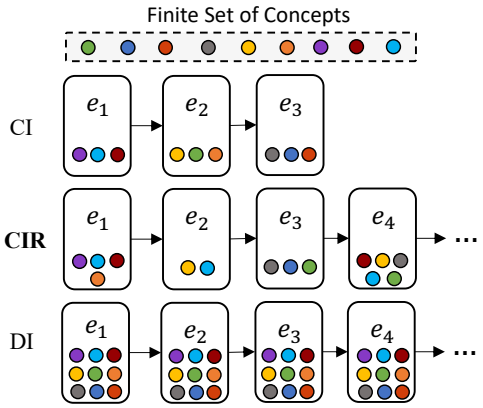
challenging. On the contrary, in many real-world scenarios forgetting is less pronounced. For example, the no repetition constraint is not present in many applications, where a degree of repetition may be naturally present. CI scenarios risk to narrow the variety of possible research paths in CL by overshadowing some objectives like forward transfer (Lopez-Paz and Ranzato 2017) and sample efficiency (Díaz-Rodríguez et al. 2018) in favor of a strong focus on forgetting. We do not advocate for the complete dismissal of the CI scenario, which has been proven useful to spark the interest around CL and to foster the design of new solutions. Instead, we aim at also promoting the usage of alternative CL scenarios, in which previously encountered patterns may be re-occur in the future. We call this family of scenarios *Class-Incremental with Repetition* (CIR).
Before exploring CIR scenarios, it is useful to dive deeper into the advantages and problematic assumptions of CI scenarios.
The discovery of the forgetting phenomenon in NNs triggered the need to develop proper benchmarks on which to test the stability of a model. Splitting a given dataset by classes provided two immediate advantages: a large availability of different datasets to work with and a simple way to produce a large amount of forgetting. As a result, CI benchmarks based on Computer Vision datasets started to attract attention and quickly became the most used ones in CL. Forgetting being so exacerbated in CI environments, the space for improvement was, and still is, large. CI really contributes to spark the attention on CL and to drive the focus on to the forgetting phenomenon. Still, CI scenarios suffer from some serious limitations, caused by the *no repetition* constraint.

Issue 1. *Repetition occurs naturally in many real-world environments.*

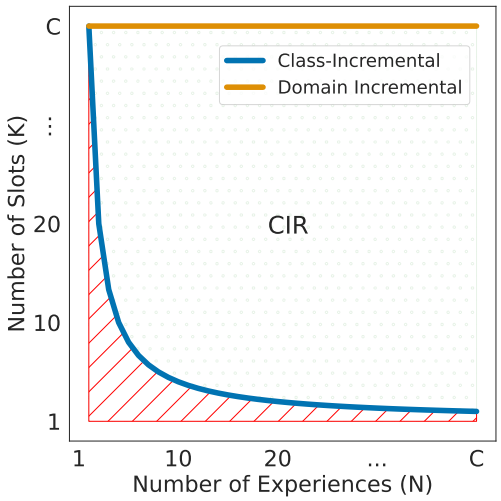
CI learning is not aligned to many applications in which repetition comes directly from the environment. Examples include robotic manipulation of multiple objects, prediction of financial trends, autonomous driving tasks, etc. A learning agent exposed to a continuous stream of information should be able to incrementally acquire new knowledge, but also to forget unnecessary concepts and to prioritize learning based on some notion of importance. Not all the perceived information should be treated equally: if a certain pattern never occurs again, it may be useless to still pretend to predict it correctly. In fact, the statistical re-occurrence of concepts and their temporal relationship could be considered as important sources of information to help determine the importance of what the agent perceives (Maltoni and Lomonaco 2016; Cossu, Carta, Lomonaco, et al. 2021). It is very hard to discern what to forget and what concepts to reinforce if all the information is treated equally.
Learning in a compartmentalized fashion hinders many of the possible insights an agent may draw from the complexity of the environment, eventually limiting its possibility to create its world model suitable to the changing task it has to tackle.

Issue 2. *Lack of repetition induces large forgetting effects.*

Focusing on catastrophic forgetting would not be inconvenient if real-world problems were actually aligned with the characteristics of the CI scenario (Thai et al. 2021). As expressed by Issue 1 however, this is not the case.
Moreover, Issue 2 led to the generally accepted statement that *Replay strategies are the most effective strategies for CL* (van de Ven, Siegelmann, et al. 2020a). This is true in CI scenarios, where Replay is extensively used (Rebuffi et al. 2017; Lopez-Paz and Ranzato 2017; Chaudhry, Ranzato, et al. 2019; Y. Wu et al. 2019; Castro et al. 2018; Belouadah and Popescu 2019; C. D. Kim et al. 2020; Douillard, Cord, et al. 2020b). However, as we will show in later sections, whenever natural Replay occurs in the environment the advantage provided by Replay is greatly reduced. While there are many works hinting at the fact that



(a) Illustration of scenario types that can be generated with episodic partial access in a closed-world setting.



(b) Illustration of how various scenarios can be generated by G_{slot}, by changing K and N. The red area under the blue curve represents invalid scenarios.

CL is not only about catastrophic forgetting (Thai et al. 2021; Díaz-Rodríguez et al. 2018), the CL scenario in which most of the research operates is still one in which the forgetting is by far the most pressing problem.

6.2 Class-Incremental with Repetition Generators

CIR (Fig. 6.1a, center) defines a family of CL scenarios which ranges from CI (Fig. 6.1a, top) to DI (Fig. 6.1a, bottom). Although appealing, currently there exists neither a quantitative analysis nor an empirical evaluation of CL strategies learning in CIR scenarios. Mainly, because it is not obvious how to build a stream with repetition, given the large amount of variables involved. How to manage repetition over time? How to decide what to repeat? What data should we use? We provide two generators for CIR that, starting from a single dataset, allow to build customized streams by only setting few parameters. The generators are as easy to use as CI or DI ones. We built both generators with Avalanche (Lomonaco, Pellegrini, et al. 2021) and we will make them publicly available to foster future research. The generators are general enough to fit any classification dataset and are fully integrated with Avalanche pipeline to run CL experiments.
The Slot-Based Generator (G_{slot}) generates streams by enforcing constraints on the number of occurrences of classes in the stream using only two parameters. G_{slot} does not repeat already observed samples, therefore the stream length is limited by the number of classes. However, it guarantees that all samples in the dataset will be observed exactly once during the lifetime of the model. The Sampling-Based Generator (G_{samp}) generates streams according to several parametric distributions that control the stream properties. It can generate arbitrarily long streams in which old instances can also re-appear with some probability.

6.2.1 Slot-Based Generator

The Slot-Based Generator G_{slot} allows to carefully control the class repetitions in the generated stream with a single parameter K. G_{slot} takes as input a dataset $\mathcal{D}$, the total

number of experiences N and the number of slots per experience K. It returns a CIR stream composed by N experiences, where each of the K slots in each experience is filled with samples coming from a single class.
G_{slot} constrains the slot-class association such that all the samples present in the dataset are seen once and only once in the stream. Therefore, G_{slot} considers repetition at the level of the whole class, while the same sample is never revisited. To implement this logic, G_{slot} first partitions all the samples in the dataset into the target number of slots. Then, it randomly assigns without replacement K slots per experience. At the end, the $N \mod K$ blocks remaining are assigned to the first experience, such that the rest of the stream is not affected by a variable number of slots.
Algorithm 1 provides a formal description of G_{slot}.
The Slot-Based Generator is useful to study the transition from CI scenarios to DI scenarios, obtained by simply changing the parameter K (Figure 6.1b). For example, let us consider a dataset with 10 classes such as MNIST. By choosing $N = 5$ and $K = 2$ we obtain the popular Split-MNIST, a CI scenario with no repetition and 2 classes for each experience. Conversely, by setting $N = 5$ and $K = 10$ we obtain a DI stream where all the 10 classes appear in each experience with new unseen samples. More in general, given a dataset with C classes, we obtain a CI scenario by setting $K = \frac{C}{N}$ (N must divide C). We obtain a DI scenario by setting $K = C$. Fig. 6.2 provides an overview of the transitions produced by G_{slot}.

Figure 6.2: From left to right: transitioning from CI to DI in G_{slot}. Each class is represented with a unique color.

6.2.2 Sampling-Based Generator

The Sampling-Based Generator (G_{samp}) generates arbitrarily long streams and controls the repetitions via probability distributions. The stream generator allows to control the *first occurrence* of new classes and the *amount of repetitions* of old classes. Unlike G_{slot}, it allows to generate infinite and even unbalanced streams.
G_{samp} parameters:

- N: Stream length, i.e. number of experiences in the stream.
- S: Experience size which defines the number of samples in each experience.
- $\mathcal{P}_f(\mathcal{S})$: Probability distribution over the stream $\mathcal{S}$ used for sampling the experience ID of the first occurrence in each class.
- P_r: List of repetition probabilities for dataset classes.

Note that $\mathcal{P}_f$ is a probability mass function over the stream $\mathcal{S}$ which means it sums up to 1.0 and determines in which parts of the stream it is more likely to observe new classes for the first time. However, the list of probabilities $\{p_1, p_2, ..., p_C\}$ in P_r are independent and each probability value $0.0 \leq p_i \leq 1.0$ indicates how likely it is for each class to repeat after its first occurrence.
For each experience, G_{samp} samples instances from the original dataset $\mathcal{D}$ according to a two step process. First, G_{samp} defines a $C \times N$ binary matrix T called *Occurrence Matrix*

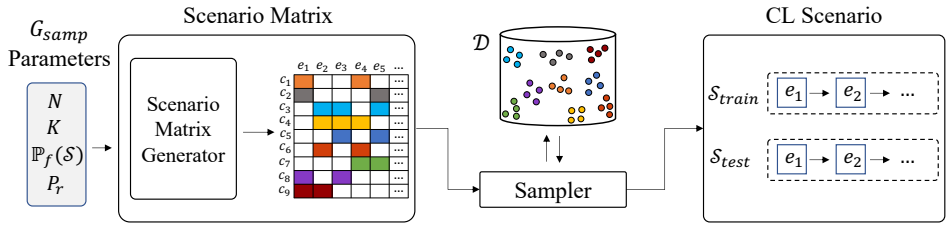


Figure 6.3: Schematic view of G_{samp} generator.

that determines which classes can appear in each experience. Then, for each experience $e_i, 1 \leq i \leq N$ we use the i-th column of T to sample data points for that experience. The generator uses the inputs N, $\mathcal{P}_f(\mathcal{S})$ and P_r to generate T. Therefore, it first initializes T as a zero $C \times N$ matrix. Then for each class c in the dataset, it uses $\mathcal{P}_f(\mathcal{S})$ to sample the index of the experience in which class c appears for the first time. Different probability distributions can be used to either spread the first occurrence along the stream or concentrate them at the beginning, which allows a good flexibility in the stream design. After sampling the first occurrences, the classes are then repeated based on P_r probability values to finalize matrix T. In the simplest case, P_r can be fixed to the same value for all classes to generate a balanced stream.

Once the matrix T is constructed, a random sampler is used to sample patterns for each experience. Since each experience may contain an arbitrary number of classes, another control parameter that could be added here is the fraction of samples per class in experience size S. For simplicity we keep the fractions equally fixed and thus the number of datapoints sampled from each class in experience e_i is $\lfloor \frac{S}{|\mathcal{C}^i|} \rfloor$ where $|\mathcal{C}^i|$ indicates the number of classes present in e_i. Since the sampler is stochastic, each time we sample from a class, both new and old patterns can appear for that class. Given a large enough stream length N, the final stream will cover the whole dataset with a measurable amount of average repetition. In Figure 6.3 we demonstrate the schematic of the generator G_{samp}.

Although we assume a fixed number of instances per class in $\mathcal{D}$, G_{samp} can be easily extended to settings where the number of instances can grow over time. Moreover, the sampler can also be designed arbitrarily depending on the stochasticity type, e.g., irregular or cyclic.

6.3 Frequency-Aware Replay

Experience Replay (ER) is the most popular CL strategy due to its simplicity of use and high performance in class-incremental scenarios. The storage policy, which determines which samples to keep in a limited buffer, is the major component of ER methods. Class-Balanced (CB) and Reservoir Sampling (RS) Vitter 1985 are the most popular policies. CB keeps a fixed quota for each class, while RS samples randomly from the stream, which leads to the class frequency in the buffer being equal to the frequency in the stream. CB and RS are great choices for balanced streams such as class incremental scenarios, where the number of samples per class is the same over the whole stream. However, as in most real-world scenarios, CIR scenarios are naturally unbalanced, and different classes may have completely different repetition frequencies. Accordingly, CB and RS storage policies may suffer a big accuracy drop in the infrequent classes of an unbalanced stream. For example, in highly unbalanced streams, RS will store an unbalanced buffer replicating the the original distribution of the stream, which is sub-optimal because the less frequent classes will require more repetition to prevent forgetting, while the frequent classes will be repeated naturally via the stream

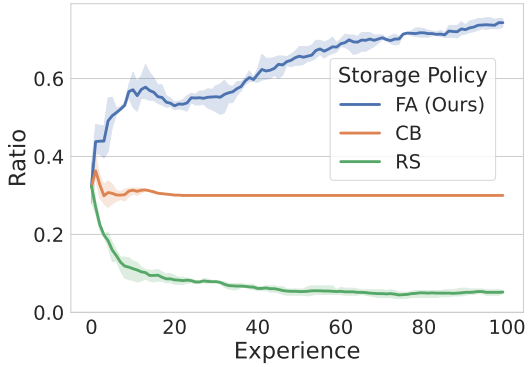


Figure 6.4: Ratio of buffer slots for infrequent classes for three random seeds.

occurrences.

We propose *Frequency-Aware* (FA) storage policy that addresses the imbalance issue in CIR streams by online adjustment of the buffer slots in accordance with the amount of repetition for each class (Algorithm 2). Given a buffer B with a maximum size of M, a list of previously observed classes P initialized as $P = \{\}$ with a corresponding list O indicating the number of observations per class c in C, and a dataset D_i from experience e_i, the algorithm first checks the present classes P_i in D_i and adds them to P ($P \leftarrow P \cup P_i$). Then, for each class c in P_i it increments the number of observations $O[c]$ by 1 if the class was previously seen, otherwise it initializes $O[c] = 1$. After updating the number of observations, FA computes the buffer quota Q for all observed classes by inverting the number of observations ($Q = [\frac{1}{O[c]} \forall c \in C]$) and normalizes it . This way, the algorithm offers the less frequent classes a larger quota. Finally, a procedure ensures the buffer is used to its maximum capacity by filling unused slots with samples from more frequent classes sorted by their observation times. This is a crucial step since it is possible that an infrequent class which is not present e_i will be assigned with a larger quota than its current number of samples in B, and therefore the buffer will remain partially empty. In Figure 6.4 we show how our method assigns higher ratio of samples for infrequent classes to overcome the imbalance issue in the stream. Examples of imbalanced scenario are provided in Fig. 6.5.

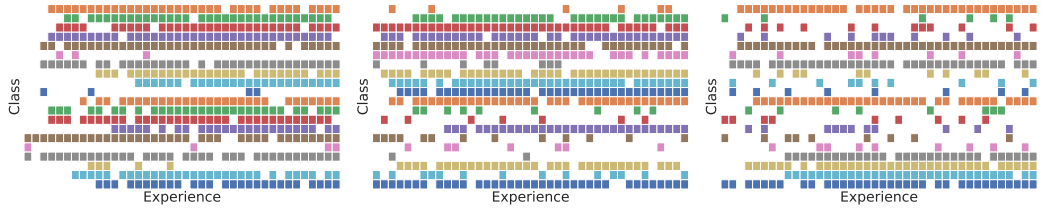


Figure 6.5: Unbalanced scenarios with two modes of repetition. The fractions of infrequent classes from left to right are 0.2, 0.4 and 0.6 respectively.

6.4 Empirical Evaluation

We study CIR scenarios by leveraging our generators G_{samp} and G_{slot}. First, by using G_{slot} we provide quantitative results about forgetting in CL strategies when transitioning from CI to DI scenarios (Sec. 6.4.1). Then, by using G_{samp} we focus on long streams with 500 experiences and study the performance of Replay and Naive (Sec. 6.4.2). The long

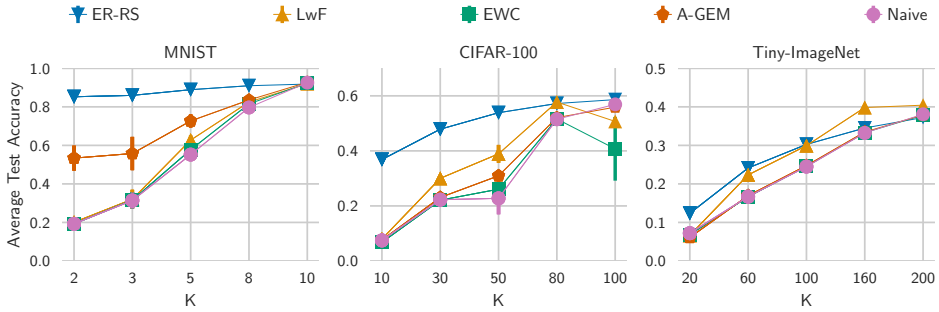


Figure 6.6: Average Test Accuracy for different values of K in CIR scenarios generated with G_{slot}. Class-Incremental scenarios are represented by the left-most point of each plot, Domain-Incremental scenarios by the right-most point. Results averaged over 3 seeds.

streams give us the opportunity to study knowledge accumulation over time in the presence of repetition. We also provide an intuitive interpretation of the model dynamics over long streams (Sec. 6.4.3). Finally, we show that our Frequency-Aware Replay is able to exploit the repetitions present in the stream and to surpass the performance of other Replay approaches not specifically designed for CIR scenarios (Sec. 6.4.4).

The experiments were conducted using the CIFAR-100 Krizhevsky, Hinton, et al. 2009 and Tiny-ImageNet LeCun, Bottou, et al. 1998 datasets with the ResNet-18 model. For G_{slot}, we run experiments for Naive (incremental fine tuning), LwF Z. Li and Hoiem 2016, EWC Kirkpatrick et al. 2017, Experience Replay with reservoir sampling C. D. Kim et al. 2020 (ER-RS) and AGEM Chaudhry, Ranzato, et al. 2019 strategies. For G_{samp} we run experiments for Naive and ER (CB/RS/FA) strategies. We set the default buffer size for CIFAR-100 to 2000, and for Tiny-ImageNet to 4000 in the Replay strategies. We evaluate all strategies on the Average Test Accuracy (TA).

6.4.1 Transition from Class-Incremental to Domain Incremental

DI and CI scenarios are heavily studied in the CL literature. However, little is known about what happens to the performance of popular CL strategies when gradually transitioning from one scenario to the other. By changing the value of K in G_{slot}, we provide a quantitative analysis of such behavior in CIR scenarios. Figure 6.6 shows the Average Test Accuracy over all classes for different CL strategies when transitioning from CI (left-most point of each plot) to DI (right-most point of each plot).

Replay is one of the most effective strategies in CI scenarios. As expected, in CIR scenarios the advantage provided by ER-RS with respect to other CL strategies diminishes as the amount of repetition increases. However, in order for the other strategies to match the performance of ER-RS, the environment needs to provide a large amount of repetition.

LwF guarantees a consistent boost in the performance, both in CIFAR-100 and Tiny-ImageNet. In particular, and quite surprisingly, on Tiny-ImageNet LwF is able to quickly close the gap with ER-RS and even surpass it as the amount of repetition increases. The positive interplay between distillation and repetition provides an effective way to mitigate forgetting in CIR scenarios, without the need to explicitly store previous samples in an external memory. EWC showed different sensitivity to the regularization hyper-parameter λ. We experimented with $\lambda = 0.1, 1, 10, 100$. While on MNIST we did not see any difference in performance, on CIFAR-100 and Tiny-ImageNet large values of λ lead to a dramatic decrease,

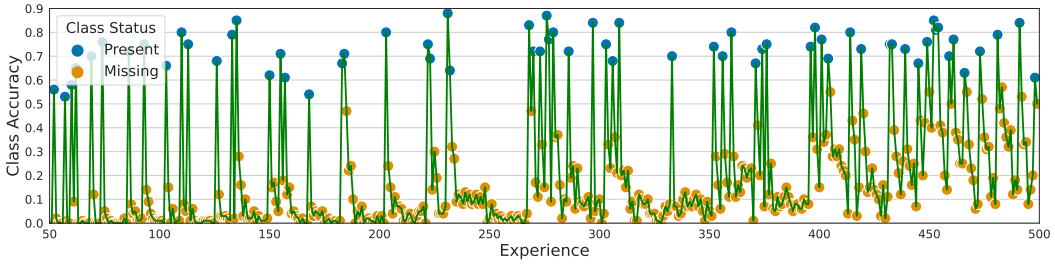


Figure 6.7: Accuracy of a particular class over the stream. The target class is either present or absent in the experiences indicated by the blue and orange points, respectively.

dropping as low as Naive. We found 0.1 to be the best value on both CIFAR-100 and Tiny-ImageNet. This configuration only provides a low amount of regularization. Overall, the role played by the natural repetition already guarantees a sufficient stability of the model, which is additionally boosted only in the case of LwF.

6.4.2 Impact of Repetition in Long Streams

We investigate the impact of repetition in long streams ($N = 500$) generated with G_{samp}. For the long-stream experiments we also report the missing-classes accuracy (MCA) and seen-classes accuracy (SCA). MCA measure the accuracy over the classes that were seen before but are missing in the current experience, and SCA measure the accuracy over all seen classes up to the current experience.

Missing Class Accuracy Increases Over Time In CI scenarios, a Naive strategy catastrophically forgets each class as soon as it starts learning on new classes. Surprisingly, we found that in CIR scenarios there is knowledge accumulation over time for all the classes. Figure 6.7 shows the accuracy of a single class over time, highlighting whether the class is present or not in the current experience. At the beginning of the stream missing classes are completely forgotten, which can be noticed by the instant drop of the accuracy to zero. However, over time the model accumulate knowledge and the training process stabilizes. As a result, the accuracy of missing classes tends to increase over time, suggesting that the network becomes more resistant to forgetting. Notice that this is an example of continual learning property that is completely ignored when testing on CI scenarios. This finding prompts the question, *"What is happening to allow knowledge accumulation even for Naive finetuning?"*. We investigate this question by analysing the model's accuracy over time and the properties of the learned model in the next experiments.

Accuracy Gap Between Naive and Replay Decreases Over Time To study the impact of long streams with repetitions we monitor the accuracy gap between ER and Naive fine-tuning by comparing their accuracy after each experience. For the scenario configuration, we set $\mathcal{P}_f(\mathcal{S})$ as a *Geometric* distribution with a coefficient of 0.01 and fix the probability of repetition $\mathcal{P}_r$ as 0.2 for all classes. In such scenarios, the majority of classes occur for the first time in the first quarter of the stream, and then repeat with a constant probability of 0.2 which makes them appropriate for our experiments since all classes are observed before the middle of the stream and the repetition probability is low enough.
As can be seen in Figure 6.8, while the accuracy of ER saturates over time, the accuracy of Naive increases, closing the gap between the two strategies from around 25% in experience

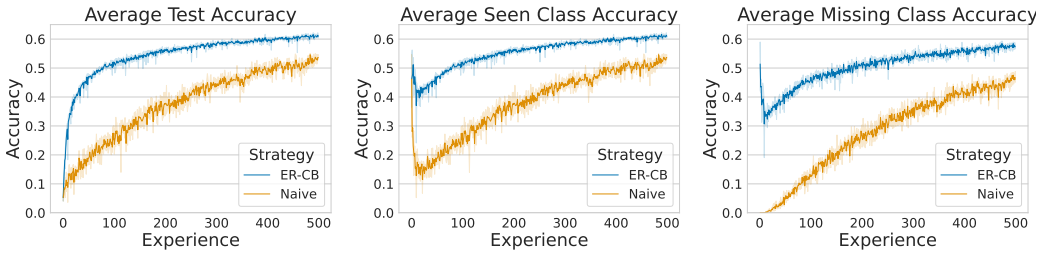


Figure 6.8: Average test accuracy and average missing class accuracy plots for long streams streams with 500 experiences.

100 to 7% in experience 500. This supports our hypothesis that neural network's ability to consolidate knowledge is significantly influenced by "natural" repetition in the environment.

The Role of Repetition The amount of repetition is one of the key aspects of a CIR scenario. To find out how strategies perform under different repetition probabilities, we consider a setting where all components of a scenario are fixed except for P_r. For this experiment, we set $\mathcal{P}_f(\mathcal{S})$ as geometric distribution with $p = 0.2$ and let P_r change. In Figure 6.9 we demonstrate the seen class accuracy (SCA) for the Naive and ER-CB strategies in CIFAR-100. It is clear from the plots, that the model's rate of convergence can be significantly affected by the amount of repetition in the scenario. Although, it may seem obvious that higher repetition leads to less forgetting, it is not very intuitive *to what extent* different strategies may gain from the environment's repetition. While the Naive strategy gains a lot from increased repetition, the Replay strategy saturates after some point for higher repetitions and the gaps close between different repetition values.

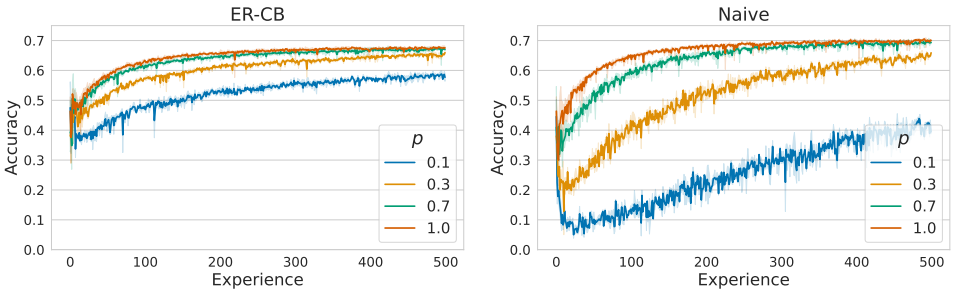


Figure 6.9: Retained accuracy for different values of p in P_r.

6.4.3 Model Similarity and Weight Space Analysis

Weight Interpolation Based on the "gradual loss drop" observation in missing classes, we study how the loss surface changes over time if we perturb the weights. We interpolate between the model weights from two consecutive checkpoints with an interval of 10 experiences in various segments of the stream. Assuming that w_t^* and w_{t+10}^* are the obtained solutions for experiences t and $t + 10$ respectively, we generate eight in-between models $w_k = \alpha * w_t^* + (1 - \alpha) * w_{t+50}^*$ by by increasing α from zero to one, and then compute the accuracy of w_k for the data of experience t. We show the interpolation accuracy for various pairs of experiments in different segments of the stream for the Naive strategy in Figure 6.10 (left). In the beginning of the stream, the accuracy of experience t in each pair drops significantly, while we observe a milder loss drop towards the end of the stream. The findings suggest

that, towards the end of the stream, even a relatively big perturbation does not have a large negative effect on the model's accuracy and the optimal solutions of the consecutive experiments are connected with a linear low-loss path.

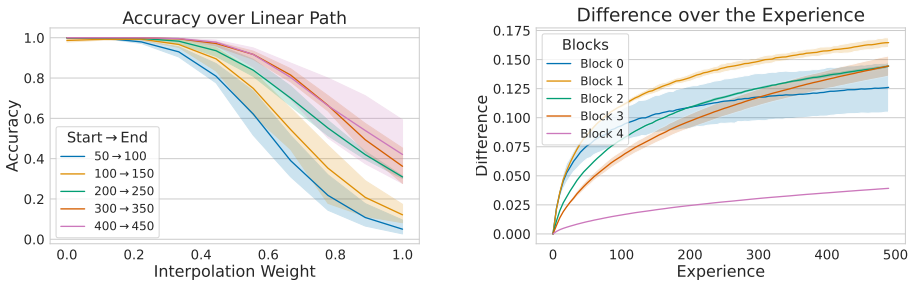


Figure 6.10: (left) Interpolation accuracy. (right) Weight changes in each block. The difference used in (right) is calculated as $D_j = \frac{1}{|\theta_0|} \sum_i^{\theta_b} \left\| \frac{(\theta_{0,i} - \theta_{j,i})}{\|\theta_{0,i}\|_2} \right\|$, where the weights of experience j are compare with the initialization θ_0 for each block i

Weight changes Another approach to analyzing the gradual drop of the accuracy is by dissecting how much, when, and where the weight changes occurs. As shown in Figure 6.10 (right), we can observe that within the first experiences, there is a significant difference for blocks 0, 1, and 2. This difference then stalls, showing that as we continue training experiences, the weights of these blocks stabilize. On the other hand, blocks 3 and 4 show a linear increase in the difference with the number of experiences. An explanation for this phenomenon, is that the first layers of the model capture knowledge that can be useful for several classes (more general), so it is unnecessary to change them after several experiences. On the other hand, the last blocks are the ones that memorize or learn more specific patterns, so they adapt to each experience.

CKA Analysis Finally, we show the CKA Kornblith et al. 2019 of the model in the beginning, middle and the end of the stream with an interval difference of 50 experiences. As shown in the visualizations in Figure 6.11, the longer the model is trained on more experiences, the less significant the changes in the representations become especially for the final layers. We can see that the diagonal of the CKA becomes sharper propagating forward with more experiments. This indicates that although the model is trained on different subsets of classes in each experiment, the representations change less after some point in the stream.

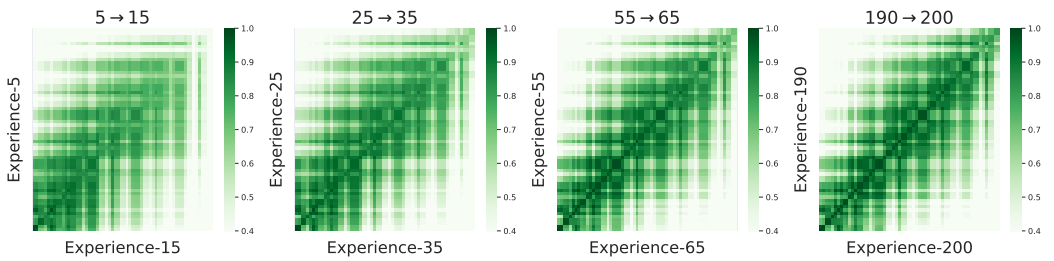


Figure 6.11: CKA of the model in different parts of the stream.

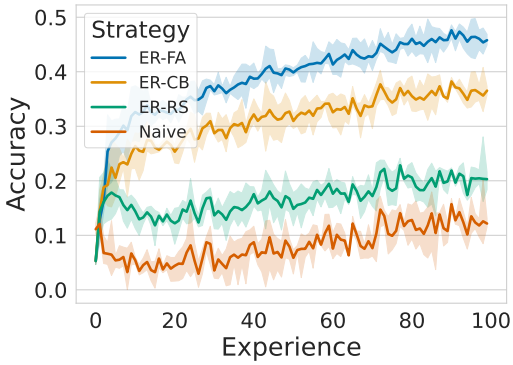


Figure 6.12: Test Accuracy of Infrequent Classes.

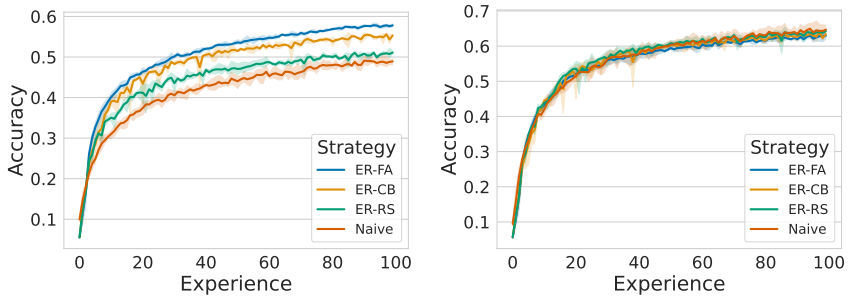


Figure 6.13: Test Accuracy over all classes (left) and frequent classes (right) in a bi-modal unbalanced scenario with Fraction=0.3.

6.4.4 Frequency-Aware Replay in Unbalanced Scenarios

We conduct experiments for bi-modal unbalanced scenarios where classes can have a high frequency of 1.0 or a low frequency of 0.1. We use a fraction factor that determines the amount of infrequent classes in the scenario, e.g., Fraction=0.3 means that 30% of the classes are infrequent. In Table 6.1 we compare ER-FA with the Naive, ER-RS and ER-CB strategies. The numbers show the MCA and average Test Accuracy (TA) metrics for each strategy in the end of the stream averaged over three runs. Fig. 6.12 shows the TA over all and frequent classes. Our strategy outperforms all other scenarios in almost all settings in both CIFAR-100 and TinyImageNet datasets in terms of TA, and significantly outperforms other methods in terms of MCA (in the last experience). Moreover, we illustrate the accuracy of infrequent classes in CIFAR-100 experiments for Fraction=0.3 in Figure 6.12 where ER-FA achieves considerably higher accuracy in the whole stream by assigning larger quota to infrequent classes without losing its performance on frequent classes.

6.5 Discussion

CIR scenarios represent CL environments where repetition is naturally present in the stream. Our experiments provide a number of different insights that, for the first time, focus solely on the role of repetition in CL environments. Although the concept of repetition is quite intuitive, it is not obvious how to realize it in practice for research purposes. We proposed two CIR generators that address this issue. The generators are able to create (even unlimited) streams with repetition from a single dataset. We leveraged our generators to run an extensive empirical evaluation of the behavior of CL strategies in CIR scenarios. We found

Table 6.1: Unbalanced scenario results for the CIFAR-100 (C-100) and TinyImageNet (TIN) dataset. "Fraction" refers to the fraction of infrequent classes having repetition probability of only 10%.

DS	Strategy	Fraction= 0.1		Fraction= 0.3		Fraction= 0.5	
		MCA	TA	MCA	TA	MCA	TA
C-100	Naive	5.0 ± 0.7	58.0 ± 0.1	7.3 ± 2.2	49.0 ± 0.8	8.0 ± 2.0	40.8 ± 1.4
	ER-RS	11.4 ± 0.9	57.7 ± 0.7	16.7 ± 3.6	51.1 ± 0.4	20.5 ± 1.9	45.6 ± 0.8
	ER-CB	30.9 ± 2.7	59.5 ± 0.1	34.5 ± 1.7	55.3 ± 0.1	35.7 ± 0.5	52.0 ± 1.5
	ER-FA	$\mathbf{52.2 \pm 1.1}$	$\mathbf{60.8 \pm 0.3}$	$\mathbf{44.7 \pm 1.5}$	$\mathbf{57.8 \pm 0.4}$	$\mathbf{40.9 \pm 1.2}$	$\mathbf{54.2 \pm 1.2}$
TIN	Naive	2.0 ± 0.8	$\mathbf{33.5 \pm 0.4}$	2.0 ± 0.1	29.1 ± 0.1	2.0 ± 0.1	24.0 ± 0.4
	ER-RS	3.7 ± 0.6	31.8 ± 1.2	4.4 ± 0.7	28.1 ± 0.2	6.0 ± 0.1	24.0 ± 0.1
	ER-CB	10.4 ± 0.2	32.2 ± 0.7	10.0 ± 1.0	28.8 ± 0.2	11.0 ± 0.2	26.0 ± 0.3
	ER-FA	$\mathbf{22.0 \pm 1.0}$	33.0 ± 0.9	$\mathbf{15.3 \pm 1.0}$	$\mathbf{30.4 \pm 0.1}$	$\mathbf{13.6 \pm 0.1}$	$\mathbf{27.0 \pm 0.1}$

out that knowledge accumulation happens naturally in streams with repetition. Even a naive fine-tuning, subjected to complete forgetting in CI scenarios, is able to accumulate knowledge for classes that are not always present in an experience. We observed that Replay strategies still provide an advantage in terms of final accuracy, even though they are not crucial to avoid catastrophic forgetting. On one side, distillation-based strategies like LwF (Z. Li and Hoiem 2016) are competitive in streams with a moderate amount of repetition. On the other side, existing Replay strategies are not specifically designed for CIR streams. We propose a novel Replay approach, called Frequency-Aware Replay (ER-FA) designed for streams with unbalanced repetition (few classes appear rarely, the other very frequently). ER-FA surpasses by a large margin other Replay variants when looking at infrequent classes and it does not lose performance in terms of frequent classes. This leads to a moderate gain in the final accuracy, with a much better robustness and a reduced variance across all classes. The empirical results about CIR provided in this work open new research directions which depart from the existing ones, mainly focused on the mitigation of forgetting in CI scenarios. We hope that our results will promote the study of CIR scenarios and the development of new CL strategies, able to exploit the inner semantics of repetition, a natural trait of real-world data streams, in many other clever and imaginative ways.

Algorithm 1 Slot-Based Generator G_{slot} pseudocode.

Require: Dataset $D = \{(x_i, y_i)\}_{i=1,\dots,P}$ with C classes, number of experiences N, number of slots K present in each experience.
Ensure: $K \leq N$
Ensure: $C \mod N = 0$
Ensure: $NK \mod C = 0$
cls-idxs = {} ▷ Empty dictionary
for $y \in set(\{y_i\}_{i=1,\dots,P})$ **do**
    cls-idxs[y] = [] ▷ Empty list init
end for
for $i = 1, \dots, P$ **do**
    cls-idxs[y_i].append(i)
end for
slots={} ▷ Empty dictionary
for $y \in$ cls-idxs **do**
    slots[y] = []
    ksample = int(len(cls-idxs[y]) $/K$)
    for $k = 1, \dots, (NK)/C$ **do**
        subset-idxs = pop(cls-idxs[y], ksample)
        subset-samples = [x_{idx} for idx $\in$ subset-idxs]
        slots[y].append(subset-samples)
    end for
end for
stream = []
for $n = 1, \dots, N$ **do**
    experience = dataset()
    seen-classes = []
    for k=1,...,K **do**
        repeat
            y = sample(slots)
        until $y \notin$ seen-classes
        seen-classes.append(y)
        experience.add(pop(slots[y], 1))
    end for
    stream.append(experience)
end forreturn stream

Algorithm 2 Frequency-Aware Buffer.

Require: Current Buffer Set B, Maximum buffer size M, List of Seen Classes C, Number of Observation per Seen Class O

D = GetExperienceDataset(e_i)
P = DetectPresentClasses(D)
$C \leftarrow C \cup P$
for $c \in P$ **do** ▷ For each present class, increment the number of observations
  if $c \in O$ **then** $O[c] += 1$
  else $O[c] = 1$
  end if
end for
$Q = [\frac{1}{O[c]} \forall c \in C]$ ▷ Calculate quota per class
$\hat{Q} = \frac{Q}{|Q|}$ ▷ Normalize quota values
$S = \{\text{ceil}(Q[c] * M) \forall c \in C\}$ ▷ Calculate buffer slot size for each class
UpdateSlots(S) ▷ Update assigned slots according to the current state of B
UpdateBuffer(B, D, S)
  return B, M, C, O

Chapter 7

Continual Pre-Training

In this Chapter, we will introduce and explore the Continual Pre-Training scenario.
The Chapter is structured as follows:

- We introduce the Continual Pre-Training and explain why the pre-train fine-tune setup is interesting for CL (Section 7.1.
- We formally define the Continual Pre-Training scenario (Section 7.2) in terms of data stream and (pre)training objectives.
- We discuss the results of our experiments in terms of forgetting (Section 7.3).
- We present existing literature that supports and frame our proposed scenario (Section 7.4).
- We summarize the results and discuss future research directions (Section 7.5).

The Chapter is based on the paper (under review) Cossu, Tuytelaars, et al. 2022.

7.1 Introduction

In the traditional CL scenario presented in Chapter 4, the CL model needs to learn the hidden features while, *at the same time*, leveraging the same features to solve the supervised task. However, this setup is not the only conceivable one in CL. NLP, for example, often exploits Transfer Learning techniques (Ruder et al. 2019) implemented through the pre-training fine-tuning setup (Section 2.1). In Transfer Learning for NLP, the general linguistic knowledge acquired with pre-training is leveraged as a starting point to target specific downstream tasks during fine-tuning. Pre-trained models are widespread also in CL (S. V. Mehta et al. 2021; T. Wu et al. 2021), where they are mostly used to conveniently initialize the model weights before learning from the non-stationary stream of data. However, the generality and robustness of the neural representations features may be greatly impaired during the continual training on the sequence of tasks, since the model will tend to overfit to the tasks objective. By separating the goal of building robust features from that of solving the task *during* the continual training, we study the Continual Pre-Training scenario (CPT). CPT allows to 1) keep continuously up-to-date the model representations over time and 2) to learn representations which are more robust to catastrophic forgetting. In fact, pre-trained features have been reported to be subjected to softer drifts during adaptation to the task (N. Mehta et al. 2020; Ramasesh et al. 2021). We can better understand the importance of CPT through an example: let us consider the case in which a model is pre-trained on a snapshot of Wikipedia containing articles up to 2018. Part of the knowledge contained inside the model will soon become outdated: on one hand, the information contained in

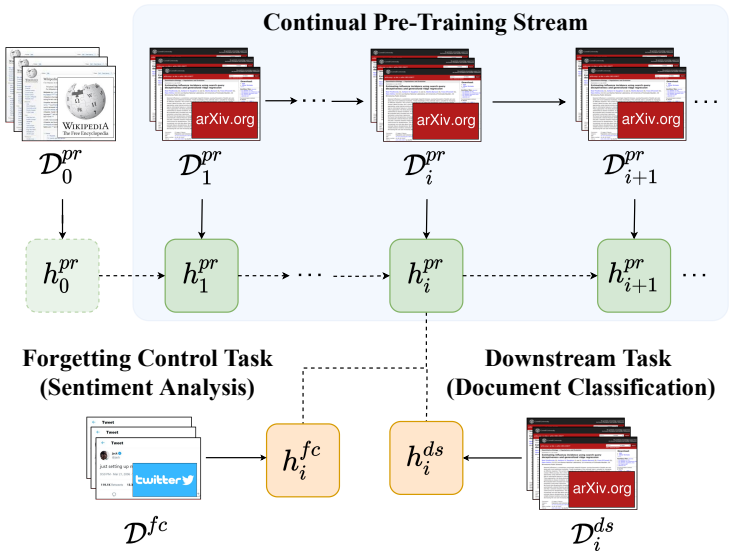


Figure 7.1: The Continual Pre-training scenario. During each stage (experience) i of continual pre-training (top), the model h_i^{pr} is pre-trained (center) on the dataset $\mathcal{D}_i^{pr}$ (e.g., *scientific abstracts*). Subsequently (bottom), the model is fine-tuned against one (or more) downstream task $\mathcal{D}_i^{ds}$ (e.g. *scientific abstracts* classification). Forgetting is measure by fine-tuning on $\mathcal{D}^{fc}$ (e.g. *sentiment analysis*). At each stage, only the current pre-trained and downstream datasets/models are available.

the original articles is likely to be replaced with up-to-date versions (e.g., changes in public figures such as a new President). On the other hand, outdated models do not incorporate the semantics of concepts related to more recent events. For example, the semantics of a term like COVID-19, which becomes important in a short amount of time, cannot be incorporated in the model without additional pre-training. As a consequence, an outdated language model may perform worse on tasks like language generation and Question Answering (Q/A), since it will not be able to generate sentences related to recent events (Jang, Ye, C. Lee, et al. 2022).

We formalize and study the CPT scenario (Figure 7.1), where the model is continuously updated via an appropriate pre-training objective on a non-stationary stream of (possibly unlabeled) data. After each stage of pre-training, we build a new model from the pre-trained one (e.g., by substituting its final classifier head) and we train it on a number of downstream tasks. We monitor whether continual pre-training improves/worsens the performance on tasks which are similar/different with respect to the ones encountered during continual pre-training. We are particularly interested in studying the possible deterioration, which represents catastrophic forgetting. For the sake of the evaluation, we specifically introduce a Forgetting Control (FC) dataset as one of the downstream tasks. The FC dataset contains samples different from the ones present in the non-stationary stream and more similar to the dataset used for the original pre-training phase prior to continual training. Against this FC dataset we compare the performance of the pre-trained model at the beginning of the sequence of tasks with the performance of the model after each stage of continual pre-training. Our aim is *to investigate the behavior of different architectures, pre-training protocols and input modalities in the CPT scenario and how these factors impact on catastrophic forgetting.* In order to explore this broad research question:

1. we formally define the CPT scenario and we describe an evaluation methodology to

assess the impact of catastrophic forgetting (Section 7.2);

2. we build two evaluation environments based on Natural Language Processing (NLP) and Computer Vision tasks (Sections 7.2.1 and 7.2.2, respectively). We thoroughly study them by using different datasets, models architectures and pre-training protocols;

3. we show that unsupervised/self-supervised pre-training protocols play a fundamental role in the mitigation of forgetting, while supervised protocols hurt the performance. The role of the architecture type and depth does not have an equivalent impact (Section 7.3);

4. we study the feature space of our pre-trained models by using linear evaluation and Centered Kernel Alignment (Kornblith et al. 2019) (Section 7.3). We observe that keeping the hidden features fixed during linear evaluation exacerbates forgetting for supervised pre-training. Supervised pre-training also causes a larger drift in the feature space compared to self-supervised pre-training.

7.2 Continual Pre-Training Scenario

The traditional CL scenario (Lomonaco, Pellegrini, et al. 2021) trains a model h_0 on a (possibly infinite) stream of experiences $\mathcal{S} = (e_1, e_2, e_3, \ldots)$, where each experience e_i brings a dataset $\mathcal{D}_i$, representing the current task. The model is trained on $\mathcal{S}$, one experience after the other, and needs to address the non-stationarity and drifts occurring between experiences without having access to the previously encountered data. The model h_0 is sometimes initialized with the weights of a pre-trained model. The pre-training phase is conducted on the dataset $\mathcal{D}^{pr}$ which is however not available during CL.

We provide a formal characterization of the CPT scenario and highlight the differences with respect to the traditional CL setup. The CPT scenario leverages a model h_0^{pr} originally pre-trained on dataset $\mathcal{D}_0^{pr}$, not available anymore. The model is presented with a (possibly infinite) stream of experiences, where each experience e_i brings a dataset $\mathcal{D}_i^{pr}$ for pre-training and a downstream dataset $\mathcal{D}_i^{ds}$ for fine-tuning. For each experience e_i, the last pre-trained model h_{i-1}^{pr} is further pre-trained on $\mathcal{D}_i^{pr}$. After the pre-training step, the model h_i^{pr} is fine-tuned on $\mathcal{D}_i^{ds}$, resulting in h_i^{ds}. We adopt naive fine-tuning, without any CL strategies. In order to measure catastrophic forgetting, we leverage a FC dataset $\mathcal{D}^{fc}$ in place of the $\mathcal{D}_0^{pr}$ originally used during the first pre-training phase. While each $\mathcal{D}_i^{ds}$ contains samples similar to the ones encountered during pre-training, the FC dataset contains knowledge more similar to the one in $\mathcal{D}_0^{pr}$ than the one in $\bigcup_{i=1,2,3,\ldots} \mathcal{D}_i^{pr}$. Forgetting is assessed after each experience e_i by comparing the performance of h_0^{pr} fine-tuned on $\mathcal{D}^{fc}$ with the performance of h_i^{pr} fine-tuned on the same dataset. We use h_i^{ds} to verify that the continual pre-training step actually contributes to learning meaningful features for the downstream task. In this way we avoid the uninteresting case where pre-training leaves features (mostly) unchanged, resulting in no catastrophic forgetting of previous knowledge but also in a lower performance on the downstream task. It is important to note that the head (last layer of the model) used during pre-training is not the one used during fine-tuning. In fact, the pre-training and downstream tasks are different ones and they therefore require different heads. Before fine-tuning on each downstream task, the head of h_i^{pr} is replaced with a randomly initialized head. The model is then trained until convergence to obtain h_i^{ds}. During the continual pre-training step instead, the head is not replaced.

Algorithm 3 provides a high-level description of the CPT scenario, showing the steps of continual pre-training, downstream fine-tuning and catastrophic forgetting evaluation against

Table 7.1: Combinations for the main components of the CPT scenario. MLM=Masked Language modeling, MIM=Masked Image Modeling, CLF=Image Classification.

Pre-training	**Architecture**	**Data**
Unsupervised (MLM)	Transformer	Words
Unsupervised (MIM)	Transformer	Images
Supervised (CLF)	Transformer	Images
Supervised (CLF)	CNN	Images

Algorithm 3 Continual Pre-training scenario

Require: Pre-trained model h_0^{pr}, stream of experiences $\mathcal{S} = (e_1, e_2, e_3, \ldots)$, FC dataset $\mathcal{D}^{fc}$.

1: $h_0^{fc} \leftarrow$ fine-tune$(h_0^{pr}, \mathcal{D}^{fc})$ ▷ Evaluate model on FC dataset before continual pre-training
2: **for** $e_i \in \mathcal{S}$ **do**
3: $\mathcal{D}_i^{pr}$, $\mathcal{D}_i^{ds} \leftarrow$ split$(\mathcal{D}_i)$
4: $h_i^{pr} \leftarrow$ pre-train$(h_{i-1}^{pr}, \mathcal{D}_i^{pr})$ ▷ Choose appropriate pre-train objective
5: $h_i^{ds} \leftarrow$ fine-tune$(h_i^{pr}, \mathcal{D}_i^{ds})$
6: $h_i^{fc} \leftarrow$ fine-tune$(h_i^{pr}, \mathcal{D}^{fc})$ ▷ Evaluate model on FC dataset
7: Compare performance of h_i^{fc} with h_0^{fc} to assess forgetting.
8: **end for**
9: **return** y

the FC dataset. To obtain the configuration we used in linear evaluation, it is sufficient to change *fine-tune* with *linear-eval* in Line 6.

The CPT scenario has different characteristics with respect to the traditional CL setup. Firstly, the CPT scenario updates continuously the pre-trained model and then adapts it to specific tasks. The traditional CL setup does not consider this important distinction, using the same model both for representation learning and to solve incoming tasks. Secondly, model evaluation in CPT requires an additional training phase on the target task, while CL usually requires the model to be readily able to tackle all tasks seen so far without any additional training. Therefore, the model has to focus on the new task without the opportunity to build robust, general features via pre-training protocols. As our results will show, the additional cost of a training phase in CPT can be largely mitigated by a quick adaptation phase (e.g., one epoch of training). In fact, fast remembering of previous knowledge is considered one of the objectives of CL (Hadsell et al. 2020).

Ultimately, *our CPT scenario aims at building models which are general learners, able to quickly adapt to unseen data while still preserving the original knowledge.* We studied CPT by introducing two evaluation environments: one for NLP and one for Computer Vision. They are designed to investigate the impact on forgetting of specific components of the scenario (Table 7.1), namely the input modality, the pre-training protocol and the model architecture.

7.2.1 Natural Language Processing Environment

Current NLP applications are all based on the idea of leveraging large-scale pre-trained models to then solve different tasks under fine-tuning, few- or even zero-shot learning settings.

Therefore, NLP applications based on the traditional pre-training fine-tuning setting seem to be the most natural field for evaluating our CPT scenario. For example, when dealing with a stream of news, it is important to keep the language model updated (Lazaridou et al. 2021) so that it can incorporate information which was not previously available. Our NLP environment employs an unsupervised/self-supervised pre-training protocol and different Transformer architectures (Vaswani 2017). These components are standard ones in NLP and represent the state of the art of the field. We use the popular pre-trained Transformers RoBERTa (Liu et al. 2019) and BERT (Devlin et al. 2019b), pre-trained on Wikipedia. In addition, we study a variant of RoBERTa in which the vocabulary is dynamically expanded with the addition of new tokens. We select the most frequent tokens of the continual pre-training dataset which were not present in the pre-trained tokenizer. Vocabulary expansion is beneficial for downstream performance, as showed by recent works on dynamic token expansion in both Computer Vision (Douillard and Timothée Lesort 2021) and NLP (Zhang et al. 2020; Han and Guo 2021). Our aim is to understand whether the addition of new tokens may result in a larger forgetting of existing knowledge. We apply continual pre-training on a dataset of `scientific abstracts` from arXiv (Geiger 2019). The motivation behind the choice of this dataset is that `scientific abstracts` represent a very specific domain for NLP both in terms of syntactic structures and domain-specific terminology. Indeed, updating the language model before fine-tuning is particularly beneficial under these circumstances. The downstream task is modeled as a document classification problem aiming to associate `scientific abstracts` to their corresponding arXiv classes. The CL stream includes 5 experiences, with 2 scientific domains (classes) in each experience (as in common CL benchmarks like Split-MNIST/CIFAR-10). We test two different FC datasets to measure forgetting: `sentiment analysis` from tweets and Question Answering Natural Language Inference (`QNLI`). The idea behind these choices is that the dataset of `scientific abstracts` should not contain much knowledge neither about sentiments, nor about generic facts for language inference. Pre-training on scientific abstracts may therefore disrupt the knowledge contained in the original language model. We additionally expand our analysis by using the 20 datasets present in the `SentEval` benchmark (Conneau and Kiela May 7-12, 2018 2018) as FC datasets.

Table 7.2: Accuracy on the entire dataset of `sentiment analysis` with RoBERTa model. Continual pre-training has been performed sequentially over each experience of `scientific abstracts`. Base refers to the model pre-trained on Wikipedia, while NT refers to the model with vocabulary expansion.

RoBERTa	**Accuracy**					**1-epoch Accuracy**				
Base	93.40					92.40				
Exp.	e1	e2	e3	e4	e5	e1	e2	e3	e4	e5
Pretr	93.40	93.15	93.35	93.20	92.90	92.40	91.80	92.30	91.85	92.20
Pretr. NT	93.75	93.70	93.75	93.60	94.10	91.75	91.15	92.00	92.30	92.45

7.2.2 Computer Vision Environment

We found Computer Vision to be a useful test-bed to disentangle the importance of the three components in our CPT scenario. In particular, we design the Computer Vision environment to understand *to what extent forgetting depends on the input modality (natural language against vision), on the architecture (Transformer against CNN) and on the pre-training*

Table 7.3: Accuracy on the entire dataset of QNLI with RoBERTa model. Continual pre-training has been performed sequentially over each experience of scientific abstracts. Base refers to the model pre-trained on Wikipedia, while NT refers to the model with expanding vocabulary.

RoBERTa	**Accuracy**					**1-epoch Accuracy**				
Base	92.73					91.76				
Exp.	e1	e2	e3	e4	e5	e1	e2	e3	e4	e5
Pretr.	91.96	91.87	91.96	91.76	92.07	90.68	91.32	90.70	90.83	90.85
Pretr. NT	92.09	91.62	91.31	91.45	91.51	91.49	91.05	91.31	89.99	90.99

protocol (unsupervised/self-supervised against supervised). To limit the large number of experiments needed to explore these three factors, in the Computer Vision environment we do not measure the performance on the downstream task after each step of continual pre-training. Instead, we focus on the study of forgetting on the FC dataset. In fact, the impact of pre-training on downstream tasks similar to the ones in the pre-training stream is assessed both in the discussion of related works (Section 7.4 above) and in the experiments with scientific abstracts classification in NLP environment (results presented below in Section 7.3).
The Computer Vision environment uses iNaturalist (Van Horn et al. 2018) for continual pre-training and CORe50 (Lomonaco and Maltoni 2017) as FC dataset for catastrophic forgetting. We use ResNet101, Vision Transformer (ViT) and BEiT originally pre-trained on ImageNet. The choice of ResNet and ViT is fundamental to disentangle the role of the architecture (NLP uses only Transformers) and the pre-training protocol (NLP uses only self-supervised pre-training). In fact, ResNet (K. He et al. 2016) and ViT (Dosovitskiy et al. 2020) are pre-trained via *supervised image classification.* The choice of BEiT (Bao et al. 2021), instead, allows to understand the role of the input modality. BEiT uses the recent self-supervised *masked image modeling* pre-training, which closely resembles the masked language modeling one used in NLP. The proposed setup allows to run experiments by changing one factor at a time among the three we studied and to keep fixed the other two. In this way, we are able to properly compare results between the NLP and Computer Vision environments.

Table 7.4: Accuracy on the entire dataset of sentiment analysis (ER) and QNLI with BERT model. Continual pre-training has been performed sequentially over each experience of scientific abstracts. Base refers to the model pre-trained on Wikipedia.

BERT	**Accuracy**					**1-epoch Accuracy**				
Base ER	93.05					92.70				
Base QNLI	90.43					90.43				
Exp.	e1	e2	e3	e4	e5	e1	e2	e3	e4	e5
Pr. ER	92.95	92.90	92.90	92.65	92.45	92.25	92.35	91.90	92.15	91.90
Pr. QNLI	90.28	89.75	90.50	89.93	90.01	90.01	89.49	89.31	89.11	89.29

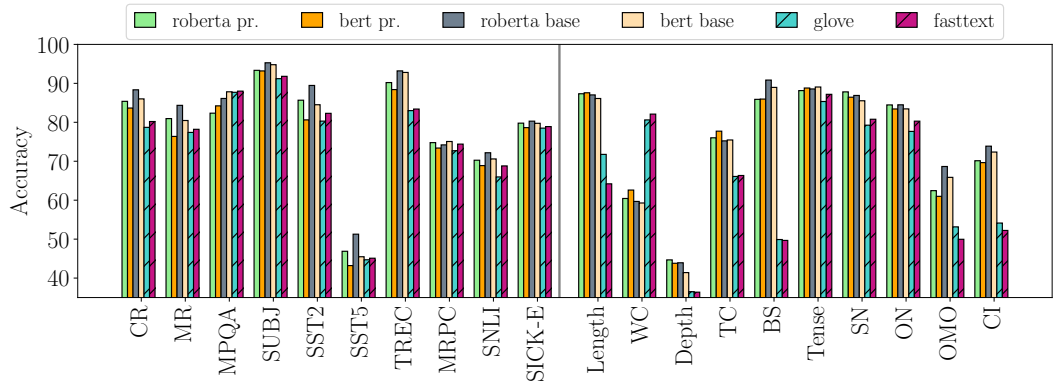


Figure 7.2: Accuracy on the 10 transfer tasks (left) and 10 probing tasks (right) of `SentEval`. Transformers are fine-tuned after 5 experiences of pre-training on the `scientific abstracts`. Base refers to the model pre-trained on Wikipedia.

7.2.3 Experimental Setup

Here, we describe the experimental setup we adopted for both the NLP environment and the Computer Vision environment. All our experiments were run on a single A100 GPU with 80 GB of memory, on a server with 96 cores.

NLP The continual pre-training dataset of `scientific abstracts` is taken from GitHub[1]. We selected 10 ArXiv classes to build our continual pre-training stream, namely 'hep-ph', 'astro-ph', 'hep-th', 'quant-ph', 'cond-mat.mes-hall', 'gr-qc', 'cond-mat.mtrl-sci', 'cond-mat.str-el', 'cond-mat.stat-mech' and 'astro-ph.SR'. For both pre-training and downstream fine-tuning, we selected $10,000$ abstracts for each of the 10 classes for the training set and $1,000$ for the test set. Hence, an abstract present in one of the training/test set of continual pre-training or downstream fine-tuning is not present in the other partitions. We chose similar abstract categories since being able to distinguish very different kinds of abstracts may greatly simplify the problem (e.g., one term may be enough to classify the entire abstract). We will publicly release our version of the scientific abstract dataset used in the experiments. The dataset can be easily loaded via Huggingface.
In order to select new tokens for the expansion of RoBERTa vocabulary at each experience of continual pre-training, we trained from scratch a tokenizer on the WikiText dataset (Merity et al. 2016). This tokenizer quickly approximates the tokens present in Wikipedia. We also train a tokenizer on our `scientific abstracts` dataset and ranked the tokens which were occurring in the latter but not in the former tokenizer. That is, the domain tokens related to the `scientific abstracts` datasets. We selected 426 new tokens for joint training experiments and 39/42/28/30/10 for each of the 5 experiences of continual pre-training.
We added tokens to the tokenizer such that new tokens have precedence over already existing tokens during tokenization process. Within new tokens, we sorted inversely by token length and the precedence is given by the order of addition (First In First Out). The list of new tokens is embedded in the released code. We also ran few experiments (not reported here) by adding with the same procedure sub-word tokens (BPE encoding) instead of word tokens. We did not find significant differences in the results, which do not seem to depend on which specific new tokens are selected, as long as they provide domain knowledge about the task.

[1]R. Stuart Geiger (2020), ArXiV Archive: A Tidy and Complete Archive of Metadata for Papers on arxiv.org, Zenodo: `http://doi.org/10.5281/zenodo.1463242`

The FC dataset `QNLI` is available from Huggingface as part of the GLUE benchmark[2]. The `sentiment analysis` from tweets dataset is also taken from Huggingface[3]. `Senteval` benchmark is taken from the official codebase[4].

During linear evaluation, we removed the feedforward layer right before the classifier. We observed that keeping it frozen yielded a very low training performance. On the other side, fine-tuning it together with the linear classifier did not show the issue but resulted in a non linear fine-tuning procedure, making it difficult to compare results against the Computer Vision setup. Therefore, linear evaluation is performed by taking the representation built for the special CLF token by the last hidden layer of the transformer and decoding it with a trained linear classifier.

Computer Vision We adopted the Masked Image Modeling task for self-supervised pre-training with BEiT. Following the original BEiT paper, we leveraged the DALL-E encoder, which is kept fixed during continual pre-training. Experiments which continually pre-train also the encoder may constitute interesting future works.
For continual pre-training and fine-tuning on FC dataset with ResNet we used a chain of augmentations: RandomResizedCrop with bilinear interpolation, RandomHorizontalFlip and normalization of mean and standard deviation. On the test sets, we resized the image to $256x256$, applied center crop and normalization. ViT uses the same setup without normalization. BEiT applies the ViT setup on the FC dataset only.

We used the Python library CKA[5], which provides the unbiased minibatch estimator of the CKA. For NLP, we use the Huggingface's pre-trained BERT and RoBERTa with 12 layers. The NLP datasets, `SentEval` excluded, are also taken from Huggingface. For `SentEval`, we train our models using the original code. We use the same pre-training protocol across all experiments, with a learning rate of 5e-5 and 30 epochs with early stopping with 2 epochs patience. For fine-tuning, we adopt a similar setup but with a learning rate of 1e-5 and 20 epochs. For Computer Vision, we use ResNet101 and iNaturalist from Torchvision, while we retrieve ViT and BEiT models from Huggingface, using the version with 12 layers in order to properly compare results with NLP experiments. We use Avalanche (Lomonaco, Pellegrini, et al. 2021) to run the continual pre-training and fine-tuning. For fine-tuning on FC task, we try few combinations of learning rates ($1e-5$, $1e-4$, $1e-3$) and batch sizes (64, 128, 256) on a held-out validation set built from `CORe50`. We report the best performance in terms of accuracy on the test set.
CKA is computed incrementally in minibatches, following Nguyen et al. 2020.

7.3 Experiments

We provide strong empirical evidence supporting the hypothesis that *the CPT scenario is less impacted by catastrophic forgetting than the traditional one.* In particular, we found the *unsupervised pre-training objective to be the common factor for the resistance to forgetting* in the proposed environments. Our result adds to the evidences discussed in Section 7.4 for the robustness of unsupervised and self-supervised protocols with respect to catastrophic forgetting. Our evaluation offers similar conclusions for the novel CPT scenario.

[2] `https://huggingface.co/datasets/glue`
[3] `https://huggingface.co/datasets/emotion`
[4] `https://github.com/facebookresearch/SentEval`
[5] `https://github.com/AntixK/PyTorch-Model-Compare`

Table 7.5: Fine-tuning accuracy on the entire dataset of `CORe50`. Pre-training has been performed sequentially over each experience of `iNaturalist`.

Model	**Accuracy**					**1-epoch Accuracy**				
ResNet	94.72					94.28				
ViT Base	90.56					90.56				
BEiT Base	90.15					82.51				
Exp.	e1	e2	e3	e4	e5	e1	e2	e3	e4	e5
ResNet Pr.	89.88	81.29	80.82	77.78	74.35	88.40	69.93	70.43	65.91	57.60
ViT Pr.	90.29	81.36	81.47	79.71	77.42	88.48	79.33	78.60	75.01	75.72
BEiT Pr.	88.37	86.45	86.73	87.07	86.46	80.55	78.06	78.88	77.27	77.06

Continual pre-training improves performance on the downstream task without forgetting on FC datasets. We verified that continual pre-training positively impacts on the performance on the downstream `scientific abstracts` classification task. That is, we observed that acquiring domain knowledge on `scientific abstracts` helps when solving the classification task (on held-out data). Table 7.6 shows the impact of 1 and 5 steps of continual pre-training on the dataset of `scientific abstracts` classification. Each fine-tuning step is performed on the corresponding split of the `scientific abstract dataset`. We see a moderate improvement in the final performance. While the improvement is relatively small, we were able to achieve it by using a smaller number of samples with respect to the common pre-training datasets (e.g. Wikipedia): this points to the fact that continual pre-training does not necessarily need enormous datasets to actually be beneficial (a very useful aspect for CL).
In terms of catastrophic forgetting on the FC dataset after continual pre-training, we show that, quite surprisingly, both RoBERTa (Table 7.2 and 7.3) and BERT (Table 7.4) achieves almost zero forgetting, reaching an accuracy comparable to the one originally obtained by the model before continual pre-training. This happens both for `sentiment analysis` and `QNLI`. Moreover, a single epoch of gradient descent is sufficient to retain most of the original performance, showing the quick adaptation capabilities of the pre-trained models. Notably, the additional pretraining steps on domain-specific texts along with the expansion of the RoBERTa vocabulary does not worsen the effects of catastrophic forgetting. We conducted a broader empirical assessment on a diverse set of NLP tasks by using the `SentEval` benchmark. Figure 7.2 shows the downstream performance of BERT and RoBERTa after the entire continual pre-training stream. GloVe and fastText results are used as baselines and are taken from Conneau and Kiela May 7-12, 2018 2018, except on `SNLI` and on all probing tasks, for which they were not available. Therefore, we computed these results using original code. The results confirm our findings: BERT and RoBERTa do not not show clear signs of forgetting, neither with respect to their original pre-trained version, nor with respect to the baselines.

Self-supervised continual pre-training mitigates forgetting. We found out that self-supervised continual pre-training is the main responsible for the mitigation of forgetting in CPT.
Since all NLP models use the self-supervised masked language modeling task for pre-training, we turned our attention to the Computer Vision environment. In fact, ResNet and ViT both use a supervised image classification during pre-training. In contrast, BEiT uses the recent

Table 7.6: Accuracy on the downstream dataset of `scientific abstracts` classification after continual pre-training. The split used for downstream classification and pre-training contains different documents. The digit next to the model indicates the last experience the model has been trained on (e.g., 5 means that the model has been pre-trained on all 5 experiences sequentially).

Model	Accuracy	1-epoch Accuracy
RoBERTa Pr. 1	82.59	79.88
BERT Pr. 1	82.64	80.91
RoBERTa Pr. NT 1	82.37	80.58
RoBERTa Pr. 5	83.24	81.19
BERT Pr. 5	83.08	81.84
RoBERTa Pr. NT 5	83.06	81.22

Table 7.7: Linear evaluation accuracy on the `sentiment analysis` (`SA`) and `QNLI` datasets. Pre-training has been performed sequentially over each experience of `scientific abstracts`.

Model	ER					QNLI				
RoBERTa	60.05					69.43				
BERT	59.85					77.87				
Exp.	e1	e2	e3	e4	e5	e1	e2	e3	e4	e5
RoBERTa Pr.	59.15	59.85	57.00	54.10	58.05	68.88	68.97	67.16	68.08	67.55
BERT Pr.	60.15	59.15	59.35	58.20	56.70	75.62	74.15	72.93	73.37	73.44

self-supervised protocol of masked image modeling (Bao et al. 2021) (mirroring the NLP setting). We show (Table 7.5) that BEiT shares the same properties of the NLP transformers, showing little forgetting with respect to the original version on the FC dataset (and one epoch of fine-tuning is sufficient to recover the original performance). Interestingly, ResNet and ViT exhibit a qualitatively different trend, with a substantial accuracy drop of around 20% and 13%, respectively. This difference in performance hints towards the fact that *supervised pre-training* in both ResNet and ViT is the main responsible of forgetting.

The negligible role of the architecture. The type of Transformer used in the experiments does not appear to be a fundamental component: we experimented with larger vision models with 24 layers instead of 12 without being able to appreciate any significant difference. The difference between convolutional networks like ResNet and attention-based transformers does not seem to have an impact, either. While ResNet sometimes exhibits worse performance than ViT, there is no clear evidence that this kind of model is more susceptible to forgetting.

Feature Space Analysis: supervised pre-training induces larger drifts. We verified the coherence of our findings by studying the feature space of the models. We leveraged linear evaluation for a quantitative analysis and Centered Kernel Alignment (CKA) (Kornblith et al. 2019) for a qualitatively analysis. Linear evaluation (i.e., training only the

Table 7.8: Linear evaluation accuracy on the entire dataset of `CORe50`. Pre-training has been performed sequentially over each experience of `iNaturalist`.

Model	**Accuracy**				
ResNet	82.50				
ViT Base	91.90				
BEiT Base	52.75				
Exp.	e1	e2	e3	e4	e5
ResNet Pr.	61.99	31.02	34.71	26.41	22.01
ViT Pr.	79.38	55.20	57.98	60.49	48.25
BEiT Pr.	52.34	51.71	51.31	53.12	52.51

Table 7.9: Fine-tuning accuracy on the entire dataset of `CORe50` with large transformers. Pre-training has been performed sequentially over each experience of `iNaturalist`.

Model	**Accuracy**					**1-epoch Accuracy**				
ViT Base	92.95					90.77				
BEiT Base	90.41					89.41				
Exp.	e1	e2	e3	e4	e5	e1	e2	e3	e4	e5
ViT Pr.	91.50	89.37	89.93	89.12	87.72	91.39	89.22	89.30	89.12	87.70
BEiT Pr.	89.78	89.90	89.18	88.50	90.09	86.81	85.94	87.50	88.50	88.50

linear classifier and keeping the rest of the model fixed) is a powerful tool to understand the impact of the learned model representations in terms of catastrophic forgetting (Davari et al. 2022). A model which exhibits forgetting during linear evaluation is likely to posses features which are not representative of the task. Conversely, a good linear evaluation performance points to a set of strong features, since it means that the task is linearly separable in that feature space. We adopted this approach for our CPT scenario. In the NLP environment (Table 7.7), the features built by the models during continual pre-training are robust and do not cause a large deviation of performance with respect to the original pre-trained model. The lower training accuracy with respect to fine-tuning is expected, since we train only a subset of all parameters. In the Computer Vision environment (Table 7.8), both ResNet and Vit suffer from forgetting, while BEiT does not (although it reaches a lower absolute accuracy).

Following D. Hu et al. 2021, we used CKA with linear kernel Kornblith et al. 2019 to compute layers similarity between the original pre-trained model and its continually pre-trained versions. From Figure 7.3, we can see that all models show large correlations across bottom layers (features are not drifting much). Instead, ViT and ResNet show lower correlation values for the final layers than BEiT. This corresponds to a larger drift in those layers. These results are compatible with what showed by Madaan et al. 2021 for unsupervised CL, namely that unsupervised models in the traditional CL scenario have larger correlations in the lower layers than supervised ones. Our results further extend this conclusion to CPT, supporting the idea that pre-training acts mainly in the upper layer of the networks (the ones containing

Table 7.10: Linear evaluation accuracy on the entire dataset of `CORe50` with large Transformers. Pre-training has been performed sequentially over each experience of `iNaturalist`.

Model	**Accuracy**				
ViT Base	82.39				
BEiT Base	52.04				
Exp.	e1	e2	e3	e4	e5
ViT Pr.	85.62	73.75	73.73	75.89	68.27
BEiT Pr.	56.67	55.62	56.12	55.74	56.76

more specific domain knowledge) and that heavy changes in these layers are enough to cause a deterioration of performance on the FC dataset, resulting in forgetting. The full set of CKA results are presented in Appendix D.

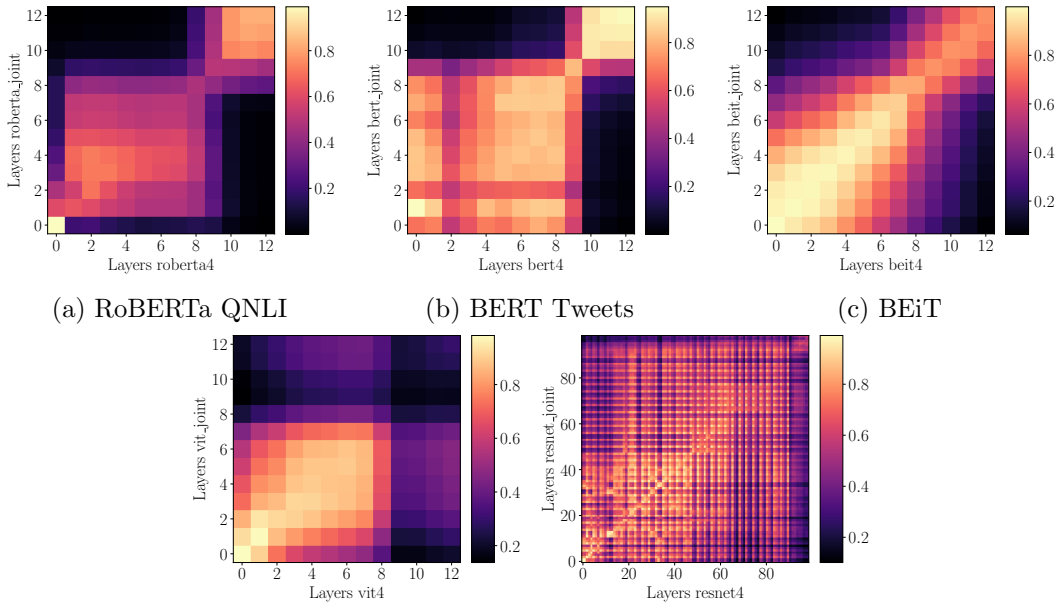
7.3.1 Additional Results

SentEval Results Table 7.12 shows the complete set of results for the `SentEval` benchmark. We compare the performance of continual pre-training after 5 experiences on `scientific abstracts` against two baselines (GloVe and fastText) and the original pre-trained model. For RoBERTa, we also provide the results in case of vocabulary expansion. We used one hidden layer of 50 units for probing tasks and logistic regression for the transfer tasks.

Experiments with Larger Computer Vision Models We report in Table 7.9 and Table 7.10 the performance obtained by larger Vision Transformers models with 24 transformers layers for fine-tuning and linear evaluation, respectively. The results are in line with our main findings with smaller models, except for the ViT, which shows a smaller degree of forgetting. However, the training curves for the large ViT shows an unstable trend: the best accuracy is reached usually after one epoch, after which the value quickly degrades to a lower performance. We believe that future works investigating the impact of model depth on our results may shed a light on this phenomenon.

7.4 Continual Pre-Training in the Literature

The ability of pre-trained models to solve a diverse set of tasks through fine-tuning has led to consider them as almost static models. However, it was recently shown that taking a pre-trained model and performing an additional step of pre-training on domain-specific data is beneficial for the downstream performance in that domain (e.g., Q/A in bio-medicine as showed by Gururangan et al. 2020; C. S. Lee and A. Y. Lee 2020). Pre-trained models are helpful also in CL, where leveraging a pre-trained model as the starting point for the continual training leads to better results with respect to forgetting both in Computer Vision (S. V. Mehta et al. 2021; Ramasesh et al. 2021) and NLP (T. Wu et al. 2021), especially when combined with CL strategies. An additional pre-training step before the continual training also provides advantages in terms of downstream performance on domain-specific tasks (Rongali et al. 2021).
The need to perform continual pre-training is present in many different applications, where



(a) RoBERTa QNLI (b) BERT Tweets (c) BEiT

(d) ViT (e) ResNet

Figure 7.3: CKA for RoBERTa, BERT, BEiT, Vit and ResNet. Pre-trained model h_5^{ds} after the last experience (x axis) is compared with the original pre-trained model h_0^{ds} (y axis). Each row is the similarity of a layer with respect to each layer of the other model.

updating the pre-trained model is fundamental to incorporate new knowledge and update or erase outdated information (Lazaridou et al. 2021; Han and Guo 2021; Jang, Ye, C. Lee, et al. 2022). While models trained directly on a domain task may achieve similar or even better performance on downstream tasks (Gu et al. 2021), the cost of starting from scratch each time is large and mitigating it is one of the objectives of CL. The performance of self-supervised methods in CL is studied also in D. Hu et al. 2021; Fini et al. 2022. In particular, D. Hu et al. 2021 focuses on the performance difference between contrastive self-supervised (MoCo-v2 by X. Chen et al. 2020) and supervised pre-training in Computer Vision, showing that self-supervised leads to robust features in terms of forgetting. Among existing works, the CL scenario used in (Jin et al. 2022) constitutes the most similar setup with respect to our definition of CPT. Like us, the authors used a dataset of research papers as pre-training stream and leveraged RoBERTa in their experiments. Differently from us, though, their work is focused on NLP tasks and on the impact that different CL strategies have on the final performance, rather than on the kind of pre-training protocol and on its impact on a separate FC task. Moreover, the downstream tasks used to measure performance are strongly related to the pre-training stream, making it difficult to understand the impact of each pre-training step on catastrophic forgetting. The results they provided show that the amount of forgetting does not depend on the specific CL strategy used. In line with our findings, a naive fine-tuning approach is robust and does not show a catastrophic loss in performance.

The Continual Knowledge Learning (CKL) framework (Jang, Ye, Yang, et al. 2021) shares some similarities with the CPT scenario we adopted. The CKL considers a pre-trained model updated continuously and, throughout its training, focuses on different objectives: recognizing invariant knowledge which does not change over time, incorporating new knowledge not present before and updating knowledge which is outdated. The proposed benchmark is

Table 7.11: Main takeaways from our experiments. Only the supervised pre-training protocols showed clear signs of forgetting, while unsupervised/self-supervised protocols did not.

Pre-training	Architecture	Data	Forgetting
Unsupervised	Transformer	Words	×
Unsupervised	Transformer	Images	×
Supervised	Transformer	Images	✓
Supervised	CNN	Images	✓

entirely based on NLP: it consists of a continual pre-training dataset of news, a "time-invariant knowledge" dataset hand-crafted from relations dataset and an "updated knowledge" and "new knowledge" datasets built from scratch through Amazon Mechanical Turk and validated by a set of external experts. The empirical evaluation provided in the paper is based on a new metric, called FUAR, which condenses the performance of the pre-trained model in these three tasks into a single number. The experiments are conducted on the T5 transformer endowed with existing CL strategies. The authors found out that that parameter expansion methods are amongst the best performing ones, although they require a larger number of parameters with respect to static alternatives.
The study of D. Hu et al. 2021 focused on the impact of self-supervised pre-training on streaming data subjected to different types of drifts (some of them ascribable to existing CL scenarios like DI, CI, data-incremental). The authors adopted the MoCo-v2 self-supervised technique for pre-training and a vast set of downstream tasks to measure forgetting, all belonging to Computer Vision. Importantly, the authors discussed the problem of catastrophic forgetting. However, differently from our approach, the evaluation is performed on the same data used for pre-training instead of relying on a separate downstream task. In our opinion, reporting results on a FC dataset better fits the CPT scenario and delivers a clearer picture of the effect of continual pre-training. Nonetheless, the results obtained by D. Hu et al. 2021 are compatible with our findings, showing that self-supervised pre-training reduces features drift and mitigates forgetting. The CKA analysis provided by the authors, similar to ours, supports the experimental results. We provide new evidence of the behavior of pre-trained models in the CPT scenario. We propose to evaluate the performance in terms of catastrophic forgetting on a FC dataset not present in the CL stream. We provided results for both Computer Vision and NLP, with experiments on longer streams than most of the existing studies (with the exception of Qin et al. 2022). Unlike prior works, we did not use any CL strategy, but we just employed naive fine-tuning.

7.5 Discussion

Our empirical evaluation provides evidence that forgetting is mitigated in CPT by the usage of self-supervised pre-training protocols (Table 7.11). Fine-tuning for only one epoch allows to recover most of the performance: this is important since an expensive fine-tuning phase might reduce the applicability of CPT in environments with constrained resources. Deciding when to use CPT and when to use the traditional CL scenario is an open question. As previously discussed, the properties of CPT do not fit the case in which a single model has to be readily applicable to different tasks without a step of fine-tuning. Nonetheless, we believe that whenever knowledge must be kept updated over time, CPT can deliver a superior

solution, less affected by forgetting. CPT offers the possibility to shift the focus from the mitigation of forgetting to other CL objectives like quick adaptation and knowledge reuse and transfer.
The main limitation of our study is related to the scale of the experiments, as we were able to experiment with only a limited number of datasets for each environment. While the computational cost of each experiment was reasonable (each experiment took from few hours - fine-tuning - to around one day - continual pre-training on a single A100), the number of experiments per environment was large. We preferred to thoroughly evaluate few environments rather than trying to address a wide range of different datasets without being able to properly explore them (Table 7.1). We are well aware that a comprehensive exploration of CPT in both NLP and Computer Vision domains is an ambitious objective, possible only in the context of a broad research program. However, we are confident of the fact that this study has shed some light on the subject and clearly pointed towards promising research directions. Much like Deep Learning has advanced by disentangling the representation learning objective from the solution to specific tasks, CPT aims to focus on the incremental development of robust features which are kept updated over time. This is a fundamental property towards the achievement of agents who can truly learn continuously in the real-world.

Table 7.12: Accuracy on 10 transfer and 10 probing tasks from `SentEval`. For comparison, we report the performance of the pre-trained models at the end of pre-training on the last experience (e5) of `scientific abstracts` dataset.

			RoBERTa			BERT	
Task	**GloVe**	**fastText**	**Base**	**Pretr.**	**Pretr. NT**	**Base**	**Pretr.**
CR	78.70	80.20	88.34	85.38	86.20	86.01	83.66
MR	77.40	78.20	84.35	80.95	80.65	80.46	76.37
MPQA	87.70	88.00	86.12	82.34	82.04	87.83	84.22
SUBJ	91.20	91.80	95.28	93.34	93.36	94.79	93.19
SST2	80.30	82.30	89.46	85.67	85.17	84.51	80.62
SST5	44.70	45.10	51.27	46.88	46.65	45.48	43.21
TREC	83.00	83.40	93.20	90.20	90.40	92.80	88.40
MRPC	72.70	74.40	74.20	74.78	74.67	75.07	73.39
SNLI	65.97	68.80	72.18	70.26	70.69	70.59	68.88
SICK-E	78.50	78.90	80.29	79.78	79.16	79.74	78.63
Length	71.76	64.20	87.03	87.33	86.17	86.11	87.58
Word Content	80.61	82.10	59.68	60.44	62.63	59.28	62.60
Depth	36.50	36.38	43.93	44.67	44.21	41.41	43.80
Top Const.	66.09	66.34	75.23	76.02	75.91	75.46	77.72
Bigram Shift	49.90	49.67	90.84	85.89	85.75	88.96	85.96
Tense	85.34	87.18	88.56	88.14	87.88	89.06	88.80
Subj Number	79.26	80.78	86.89	87.81	87.44	85.53	86.44
Obj Number	77.66	80.29	84.49	84.46	84.80	83.44	83.42
Odd Man Out	53.15	49.96	68.65	62.45	61.67	65.86	60.99
Coord. Inv.	54.13	52.23	73.87	70.13	70.33	72.36	69.65

Part III

Continual Learning with Recurrent Neural Networks

Chapter 8

Motivation

The study of TI scenarios, first, and CI scenarios, later, has driven the effort of a growing community of CL researchers (De Lange, Aljundi, et al. 2021). Still, if we look at the vast majority of the results, they revolve around a few application domains. In particular, Computer Vision applications are, by far, the most studied in CL. Computer Vision can provide a wide variety of different problems and non-stationary environments in which a CL agent can then test its ability. While the CL literature is historically focused on object detection (i.e., image classification), the trend is slowly changing with the introduction of new non-stationary environments for object detection and image segmentation. CL in RL environments, which contributed to spark the interest in CL, continues to be occasionally studied (M. Ring 1994; Kaplanis et al. 2019; Kaplanis et al. 2018). However, RL is still outnumbered by simpler Computer Vision setup (like object detection) in both the number of works and the amount of results. This is likely due to the fact that CL for Computer Vision provides a much more accessible experimental setup than RL: the benchmarks in Computer Vision are very easy to use and quite cheap to work on in terms of both memory and computational time. CIFAR-100, for example, is one of the most common benchmarks for CL. Its CI version with 10 classes per experience requires to train a model on $10,000$, 32×32 RGB images at each experience. Therefore, a single training phase does not require much time. Also, most of the models configurations and hyper-parameters can be taken directly from the existing Computer Vision literature. On the contrary, RL still requires heavy tuning of the algorithms and the training phase is more sensitive to stochastic components of the environment. Even in iid settings, training an RL agent requires many more trials than a supervised object detection task. The vast literature about CI scenarios for object detection allows to focus almost exclusively on the design of new strategies to mitigate forgetting. Little is currently known about i) the behavior of existing strategies in alternative domains, especially outside Computer Vision and ii) the most important aspects to consider when designing new strategies in under-studied domains.

In this Part, we will explore the Sequential Data Processing domain for CL. In particular, we are interested in sequences and temporally correlated data, where each input $\boldsymbol{x}$ is a sequence of ordered elements $\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_T})$. We believe that real-world data streams are sequential in nature. Time is a fundamental component that can be leveraged to learn meaningful representations.

As we have seen in Chapter 2, RNNs are designed to work with sequential data and are one intuitive candidate to tackle such tasks. They are not the only choice: 1-dimensional CNN, Transformers and, to some extent, even MLP can effectively work with sequences. However, RNNs are the only models able to learn temporal correlation outside a fixed-size window of elements. This is sometimes more of a theoretical consideration rather than

a practical one: due to the vanishing gradient issue, learning long-term temporal correlations is challenging for any RNN. Nonetheless, their behavior is substantially different from the one of a Transformer or a CNN. RNNs are still widespread in many practical applications, from biology and medicine to robotics and finance. Quite unexpectedly, CL for Sequential Data Processing with RNNs has not yet received the attention it probably deserves.

In Chapter 9 we review the state of the art and the existing works for CL in Sequential Data Processing domains (with RNNs and other models). We also provide a review of the benchmarks used in the literature, showing that few works reuse previously proposed benchmarks. Most of them are in fact specific to an application and cannot be reliably used to extrapolate conclusions about the performance of a CL agent dealing with sequences.
Equipped with this knowledge, in Chapter 10 we turn our attention to the behavior of popular CL strategies not specifically designed for sequential data and applied to RNNs. This represents the most important point of this Part and justifies the rest of the work by providing an answer to the question: *do CL strategies exhibit different characteristics and performance when applied to RNNs?* In case of a negative answer, the study of RNNs in CL could only be limited to specific applications, where they are needed. However, as we found out, RNNs introduce additional challenges in the design of CL strategies. In particular, the input sequence length plays a fundamental role and positively correlates with the amount of forgetting experienced by a RNN. We call this phenomenon the Sequence Length Issue. While the impact of forgetting changes depending on the CL strategies used, *many* CL strategies suffer from this phenomenon.
In Chapter 11 we provide a set of possible solutions to the Sequence Length Issue. First, we adapt and improve an architectural strategy like Progressive network to work with RNNs. In line with the results of the previous Part. Then, we propose Randomized RNNs like Echo State Networks as an effective solution to forgetting in CI scenarios. By leveraging a fixed set of recurrent connections, ESNs side-step the Sequence Length Issue. Also, ESNs can easily leverage CL strategies which require fixed feature extractors, since the set of recurrent connections is untrained. This represents probably the only way to leverage such strategies with RNNs. Finally, we conclude with a practical application of CL with RNNs. We study a DI scenario for Human State Monitoring, which presents strong temporal correlation in the input data. We show that, like in the Continual Pre-Training scenario, forgetting is not a strong issue when the drift is a softer one with respect to CI scenarios. Actually in the scenario we experimented on, taking too much care about forgetting can be counter-productive and can lead to worse performance in terms of knowledge accumulation and forward transfer.

Chapter 9

A Review of Recurrent Neural Networks in Continual Learning

The literature on CL focuses on feedforward and convolutional models, with experiments in Computer Vision and, to a lesser extent, Reinforcement Learning applications. Recently, there has been a growing interest towards CL applications with sequential data. In this Chapter, we provide the first comprehensive review of the literature on RNNs in CL settings. In addition, we describe the main datasets and benchmarks for Sequential Data Processing used in CL.
We propose a new classification in which each work is assigned to one of 4 macro-areas: seminal work, natural language processing applications, bio-inspired and alternative recurrent models, and deep learning models.
Table 9.1 provides a fine-grained description of our review. For each paper we highlight 5 different properties: the group to which the paper belongs to, the use of popular deep learning architectures, the application domain, the type of CL scenarios and if the paper provides a large experimental comparison. Table 9.1 shows that we are the only one to provide a large scale evaluation of recurrent models in CI scenario.
The Chapter is structured as follows:

- We review the seminal works that experimented with recurrent models in CL (Section 9.1).
- We review works from Natural Language Processing in CL that involve sequential data (Section 9.2).
- We review bio-inspired approaches and recurrent models that are designed for CL (Section 9.3).
- We review deep recurrent models for CL (Section 9.4).
- We present the main benchmarks used in CL with sequential data (Section 9.5).

The Chapter is based on the published paper Cossu, Carta, Lomonaco, et al. 2021.

9.1 Seminal Works

Historically, interest in CL and Sequential Data Processing traces back to M. B. Ring 1997 and French 1997b. The CHILD model (M. B. Ring 1997) represents one of the first attempts to deal with sequential data in reinforcement learning environments like mazes. Although CHILD lacks an explicit recurrent structure, the author discussed about the possibility to

Table 9.1: Overview of the literature based on our categorization. *Deep RNN* refers to the use of a learning model attributable to the family of deep RNNs (e.g. LSTM). Application-agnostic papers have a dash in the *application* column. A dash in the *CL* scenario indicates that the paper does not provide its clear indication. *Large comparison* refers to experiments with at least 3 CL baselines on two different application domains.

	Reference Paper	Deep RNN	Application Domain	CL Scenario	Large Comparison
seminal	M. B. Ring 1997	×	-	-	×
	French 1997b	×	-	CI	×
	Ans, Stephane Rousset, et al. 2002	×	-	CI	×
	Ans, Stéphane Rousset, et al. 2004	×	-	CI	×
	Coop and Arel 2013	×	-	CI	×
NLP	Asghar et al. 2019	✓	domain adaptation	CI	×
	Y. Li et al. 2020	✓	NMT	CI	×
	Wolf et al. 2018	✓	language modeling	-	×
	Kruszewski et al. 2020	×	language modeling	Online	×
	Madasu and Rao 2020	✓	sentiment analysis	-	×
	Thompson et al. 2019	✓	NMT	-	×
bio-inspired	Cui et al. 2016	×	-	Online	×
	Ororbia 2020	×	-	Online	×
	Parisi, Tani, et al. 2018	×	vision	-	×
	Kobayashi and Sugino 2019	×	motor control	TI	×
	Ororbia et al. 2019	×	-	Online	×
deep learning	Xue et al. 2019	✓	ASR	-	×
	Sodhani et al. 2019	✓	-	-	×
	Schak and Gepperth 2019	✓	synthetic	-	×
	Cossu, Carta, and Bacciu 2020	✓	generic	CI	×
	Duncker et al. 2020	✓	neuroscience	-	×
	Ehret et al. 2020	✓	-	TI	✓
	Cossu, Carta, Lomonaco, et al. 2021	✓	-	CI	✓

include a more powerful memory component into the model (i.e. an RNN) to address tasks with longer temporal dependencies. However, the author also recognizes the difficulties in this process, due to the challenge of learning very long sequences. Moreover, the fixed structure of a RNN compared with a growing model like CHILD highlighted the need for approaches based on model expansion.

In the same period, French 1997b; French 1997a introduced the pseudo-recurrent connectionist network, a model which makes use of pseudo replay (Robins 1995), but it does not address sequential data. Pseudo replay is based on the idea of replaying random, but fixed, input patterns with an associated target. Later on, the pseudo recurrent network together with pseudo-replay inspired the design of the Reverberating Simple Recurrent Network (RSRN) (Ans, Stephane Rousset, et al. 2002; Ans, Stéphane Rousset, et al. 2004). This is a dual model composed by two auto-associative recurrent networks which exchange information by means of pseudo patterns. In the *awake state*, the performance network learns a new pattern through backpropagation. The storage network generates a random pseudo pattern and presents it to the performance network, interleaved with the real one. Once the loss falls below a certain threshold, the system enters the *sleep state*, in which the performance network generates a pseudo pattern and the storage network learns from it. In this way, pseudo patterns produced

in the awake state carry information about previous sequences to the performance network, while pseudo patterns produced in the sleep state carry information about the recently acquired knowledge to the storage network. The authors showed the presence of forgetting on toy sequences and the beneficial effect of pseudo patterns in mitigating it. However, from their experiments it is not possible to assess the effectiveness of RSRN on more realistic benchmarks.
From the very beginning, sparsity has been a recurring theme in Continual Learning (French 1999; French 1991). The Fixed Expansion Layer (Coop and Arel 2012) introduced the use of a large, sparse layer to disentangle the model activations. Subsequently, the Fixed Expansion Layer has been applied to RNNs (Coop and Arel 2013). However, in order to build the sparse layer in an optimal way, the model requires to solve a quadratic optimization problem (feature-sign search algorithm) which can be problematic in real world problems.

9.2 Natural Language Processing

Natural Language Processing is becoming one of the main test-beds for continual and online settings in Sequential Data Processing (Biesialska et al. 2020). Most proposals in this area used modern RNNs and focus on specific problems and strategies. Moreover, they rarely compare against existing CL techniques. It is therefore challenging to draw general conclusions on RNNs in CL from this kind of experiments. Examples of applications are online learning of language models where new words are added incrementally (Y. Li et al. 2020; Wolf et al. 2018; Kruszewski et al. 2020), continual learning in neural machine translation on a sequence of languages (Thompson et al. 2019) and sentiment analysis on multiple topics (Madasu and Rao 2020).
The use of attention mechanisms (Bahdanau et al. 2015), now widespread in NLP, may provide approaches which are widely applicable, since there is no assumptions on the type of information to attend to. As an example, the Progressive Memory Banks (Asghar et al. 2019) augments an RNN with an external memory. The memory grows in time to accommodate for incoming information, while at the same time freezing previous memory cells successfully prevents forgetting. In addition, the author showed that finetuning old cells, instead of freezing them, increases forward transfer on future data. Their experiments are executed on incremental domain adaptation tasks, where the distribution of words shifts as new domains are introduced.

9.3 Bio-inspired and Alternative Recurrent Models

There are a number of contributions that propose customized architectures to address continual learning problems and mitigate catastrophic forgetting. Bio-inspired models for CL, such as the Hierarchical Temporal Memory (Cui et al. 2016) and the Spiking Neural Coding Network (Ororbia 2020), are naturally designed to tackle Sequential Data Processing tasks. More importantly, they adopt learning algorithms which allow to control more efficiently the stability-plasticity trade-off. While promising, these approaches are still missing a thorough empirical validation on real world CL benchmarks.
Echo State Networks (Lukoševičius and Jaeger 2009) are appealing for CL since the reservoir cannot suffer from forgetting. One of the drawbacks may be the fact that the static connections must be able to learn different tasks without adapting their values. The work of Kobayashi and Sugino 2019 tries to overcome the problem by employing a fractal reservoir combined with an external task vector (based on the task label) representing the current task. Different reservoir chunks process different patterns based on their associated task

vector. While the final performance on different motor commands for reinforcement learning environments validated the approach, the requirement of multi task scenarios limits its applicability.
The Backpropagation Through Time (BPTT) is the most used algorithm to train RNNs. The Parallel Temporal Neural Coding Network (Ororbia et al. 2019) introduced a new learning algorithm which is less susceptible to forgetting than BPTT and other variants in sequential benchmarks like MNIST and language modelling.
Temporal information may also arrive from videos. This is particularly important since it allows to exploit the vast literature on CL and Computer Vision. However, it is also possible to develop specific solutions, as it has been done with recurrent Self Organizing Maps (SOM) (Parisi, Tani, et al. 2018). The authors incorporate temporal information into the recurrent SOM and perform object detection from short videos with CORe50 dataset (Lomonaco and Maltoni 2017).

9.4 Deep Learning Models

Recently, there have been a number of papers that studied CL applications in sequential domains using recurrent architectures widely used in the deep learning literature, such as Elman RNNs (Elman 1990) and LSTMs (Hochreiter and Schmidhuber 1997). The advantage of this generic approach is that it can be easily adapted to specialized models to solve any sequential problem. As expected, vanilla recurrent models such as Elman RNNs and LSTMs suffer from catastrophic forgetting in CL scenarios (Sodhani et al. 2019; Schak and Gepperth 2019).
The combination of existing CL strategies, like Gradient Episodic Memory (Lopez-Paz and Ranzato 2017) and architectural strategies, with RNNs has already showed promising results (Sodhani et al. 2019). Contrary to vanilla LSTM networks, their model was able to mitigate forgetting in three simple benchmarks. This important result supports the need for an extensive evaluation of RNNs and CL strategies not specifically tailored to Sequential Data Processing problems.
RNNs are also inclined to be combined with architectural strategies, since most of them are model agnostic. The idea behind Progressive networks (Rusu et al. 2016) has been applied to RNNs (Cossu, Carta, and Bacciu 2020) and also improved by removing the need for task labels at test time with a separate set of LSTM autoencoders, able to recognize the distribution from which the pattern is coming from. The resulting model is multi-headed but it is able to automatically select the appropriate head at test time in CI scenarios. Hypernetworks (previously used for CL in von Oswald et al. 2020) are able to mitigate forgetting when combined with RNNs (Ehret et al. 2020): their work was the first to provide an extensive comparison of traditional CL techniques in TI scenarios with multi-head RNNs. Preserving the space spanned by the connections from being corrupted by the weight update appears to be beneficial also to CL (Duncker et al. 2020). Finance (Philps et al. 2019) and Automatic Speech Recognition (Xue et al. 2019) applications have been explored as candidate application domains for online and continual learning strategies.

9.5 Sequential Data Processing Datasets for Continual Learning

Due to the different application domains and different research communities interested in continual learning for Sequential Data Processing domains, there are no standard benchmarks

Table 9.2: Datasets used in continual learning for Sequential Data Processing. The *scenario* column indicates in which scenario the dataset has been used (or could be used when the related paper does not specify this information).

Dataset	Application	Scenario
Copy Task (Sodhani et al. 2019)	synthetic	TI
Delay/Memory Pro/Anti (Duncker et al. 2020)	synthetic	TI
Seq. Stroke MNIST (Sodhani et al. 2019)	stroke classification	CI/DI
Quick, Draw! †	stroke classification	CI
MNIST-like (Coop and Arel 2013) †	object classification	CI/DI
CORe50 (Parisi, Tani, et al. 2018)	object recognition	CI/DI
MNLI (Asghar et al. 2019)	domain adaptation	DI
MDSD (Madasu and Rao 2020)	sentiment analysis	DI
WMT17 (Bojar et al. 2017)	NMT	TI
OpenSubtitles18 (Lison et al. 2018)	NMT	TI
WIPO (Junczys-Dowmunt et al. 2016)	NMT	TI
CALM (Kruszewski et al. 2020)	language modeling	Online
WikiText-2 (Wolf et al. 2018)	language modeling	CI/DI
Audioset (Cossu, Carta, and Bacciu 2020)	sound classification	CI
LibriSpeech, Switchboard (Xue et al. 2019)	speech recognition	TI/CI
Synthetic Speech Commands †	sound classification	CI
Acrobot (Kobayashi and Sugino 2019)	reinforcement learning	TI

used to evaluate CL strategies. Existing benchmarks vary greatly in terms of complexity. Furthermore, different application domains use a slightly different language. In this section, we provide a review of the different datasets and continual learning scenarios. One objective of this review is to favor the cross-pollination between classic CL techniques and sequential domains and between different sequential domains.

Table 9.2 provides an overview of the different datasets in literature. For each dataset we highlight previous work that used the datasets, the application domain of the data, and the CL scenario. Clearly, most datasets in literature are used by few, or even just one, paper. This is due to the different research questions each paper tries to answer: since few works are spread over very different areas, it is natural to find different benchmarks and evaluation protocols. Unfortunately, these differences in the experimental setups make it impossible to compare different models in the literature or to deduce general and task-independent conclusions.

Different **synthetic benchmarks** have been adapted to continual scenarios. The *Copy Task* (Graves et al. 2014), a benchmark used to test the short-term memory of recurrent models, incrementally increases the sequence lengths in a DI scenario. However, the data generating distribution remains constant, which means that the drift between the different experiences is limited. *Pixel-MNIST* is another popular benchmark for RNN models (Le et al. 2015) where MNIST digits are presented one pixel at a time, either with the original order or using a fixed permutation. Continual learning scenarios based on pixel-MNIST include new classes (CI in Cossu, Carta, and Bacciu 2020; Cossu, Carta, Lomonaco, et al. 2021 or TI in Ehret et al. 2020) or new permutations (DI in Cossu, Carta, Lomonaco, et al. 2021). *Sequlecunential Stroke MNIST* (de Jong 2016) represents MNIST digits (LeCun, Bottou, et al. 1998) as a sequence of pen strokes, with pen displacements as features (plus pen up/down bit). The dataset is adapted to a continual data stream by increasing the sequence length (DI) or by creating new classes (CI in Cossu, Carta, and Bacciu 2020; Cossu, Carta, Lomonaco, et al. 2021, or TI in Ehret et al. 2020). More realistic CL benchmarks for computer vision, like

CORe50 (Lomonaco and Maltoni 2017), can be used also in sequential contexts to leverage temporal correlated information from videos.
In the **Natural Language Processing** domain a common scenario is the DI scenario, where new instances from different topics are gradually introduced (DI). Examples of applications are natural language inference (Williams et al. 2018; Asghar et al. 2019), sentiment analysis (Madasu and Rao 2020) and machine translation in a TI scenario (Thompson et al. 2019). Alternatively, Wolf et al. 2018 studies the problem of online learning a language model. There, EWC is used to keep the previous knowledge when the recurrent model is trained with a single sequence. CALM is a benchmark specifically designed for online CL in language modeling (Kruszewski et al. 2020) which provides a realistic environment in which to test NLP models.
The most common CL scenario in the **audio signal processing** domain is the incremental classification (TI/CI), where new classes are gradually introduced (Cossu, Carta, and Bacciu 2020). For example, *AudioSet* (Gemmeke et al. 2017) is a dataset of annotated audio events. The raw audio signals are already preprocessed, generating sequences of 10 timesteps with 128 features each. Differently from Sequential Stroke MNIST or Copy Task, AudioSet is a real-world application. However, the small sequence length may conceal possible problems when working with RNNs. Other datasets in the literature refer to specific applications, like reinforcement learning (Kobayashi and Sugino 2019), or neuroscience problems (Duncker et al. 2020).

Chapter 10

The Sequence Length Issue

In Chapter 9, we described the fragmented landscape of studies on CL for Sequential Data Processing tasks. However, none of those studies focused on the peculiar behavior that recurrent models might show with respect to forgetting in CL. Are there any differences in learning continuously with recurrent models with respect to other kinds of architectures? Can we identify specific properties that describe the CL process with RNNs? Answering these questions provides solid basis upon which build and design novel CL strategies. To this end, we conducted an empirical evaluation on RNNs combined with strategies belonging to different families of approaches. To better understand the difference between a feedforward (or convolutional) model and a recurrent one, we first considered the former as an instance of a recurrent model dealing with sequences of length 1. As such, by tweaking the *input sequence length* one can obtain a precise idea on the direction followed by the performance of a CL agent when transitioning from a feedforward model to a recurrent one. Perhaps surprisingly, this very simple approach already provides interesting insights on the behavior of RNNs. The input sequence length proves to be a fundamental factor affecting the final performance of recurrent models in terms of forgetting. In our empirical evaluation, we provide strong evidence towards the fact that existing CL strategies are not able to effectively deal with Sequential Data Processing tasks. The amount of forgetting they exhibit increases as the sequence length increases. Learning continuously with temporally correlated data requires to address unique challenges and cannot be performed by blindly applying existing CL strategies.
This Chapter introduces the Sequence Length Issue and it is structured as follows:

- We introduce the experimental setup used to assess the performance of existing CL strategies when applied to RNNs (Section 10.1).
- We present and discuss the results, which highlight the Sequence Length Issue (Section 10.2).

The Chapter is based on the published paper Cossu, Carta, Lomonaco, et al. 2021.

10.1 Experimental Setup

We build two CI benchmarks by leveraging the Synthetic Speech Commands dataset (Buchner 2017) and Quick, Draw! dataset (Ha and Eck 2018). These are two tasks which are easy to learn in an offline, iid setting. However, they become challenging in a continual setting, making them ideal as a CL benchmark. Moreover, they are suitable to evaluate RNNs since they have a sequential nature. In addition, we ran experiments on the pixel-MNIST, both in its DI (Permuted pixel-MNIST or PMNIST) and CI version (Split pixel-MNIST or SMNIST).

Table 10.1: Benchmarks used in the experimental evaluation. Quick, Draw has patterns with *variable* sequence length, from a minimum of 8 time-steps to a maximum of 211 time-steps. Experiments with SMNIST and PMNIST have been conducted with different sequence lengths by taking 28 or 4 pixels at a time, resulting in sequences of length 28 and 196, respectively.

	SSC	QD	SMNIST	PMNIST
Experiences	10	10	5	10
Classes per experience	2	2	2	2
Sequence length	101	variable (8-211)	28/196	28/196
Total number of classes	30	345	10	10
Input size	40	3	28/4	28/4

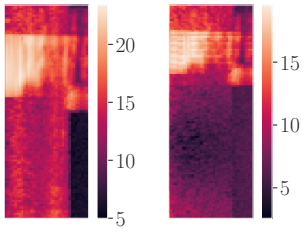
In all the MNIST-based experiments, we took each image a subset of pixels at a time to build the input sequence. Varying the amount of pixels present in each time-step allows to build sequences with different lengths and to study the impact of this factor on forgetting. Table 10.1 summarizes the main characteristics of the benchmarks used in the experiments, while Figure 10.1 provides a graphical representation of their patterns.

Synthetic Speech Commands (SSC) SSC is a dataset of audio recordings of spoken words (Buchner 2017). Each audio represents a single word, taken from a vocabulary of 30 words, sampled at 16kHz. We preprocessed each sample to extract 40 Mel coefficients using a sliding window length of 25 ms, with 10 ms stride. The resulting sequences have a fixed length of 101 time-steps. We did not apply any further normalization. We create a CI scenario from SSC by learning two classes at a time, for a total number of 10 experiences. For model selection and assessment, we used a total of 16 classes out of the 30 available ones.

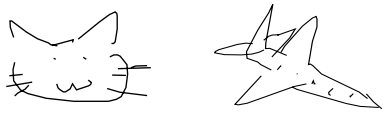
Quick, Draw! (QD) QD is a dataset of hand-drawn sketches, grouped into 345 classes (Ha and Eck 2018). Each drawing is a sequence of pen strokes, where each stroke has 3 features: x and y displacements with respect to the previous stroke and a bit indicating if the pen has been lifted. We adopted a CI scenario by taking 2 classes at a time for a total number of 10 experiences. For model selection and assessment, we used a total of 16 classes out of the 345 available ones. Differently from the previous datasets Quick, Draw! sequences have variable length. Since this is often the case in real-world applications, it is important to assess the performance of RNNs in this configuration.

For model selection, we followed Chaudhry, Ranzato, et al. 2019. We separate the data stream into validation and test experiences. In particular, we hold out the first 3 experiences from our data stream and use them to perform model selection. After selecting the best hyperparameters, we use the remainder of the stream (the test stream, composed of 10 unseen experiences in our experiments), to perform model assessment. Since Split MNIST has a limited number of experiences (5), we decided to perform model selection and assessment following the train-test split protocol described in Chapter 4. The reference RNN that we use in our experiments is the LSTM (Hochreiter and Schmidhuber 1997). We compared its performance with an MLP with ReLU activations. On PMNIST and SMNIST, we ran experiments with a LSTM taking as input 4 pixels at a time (LSTM-4), 16 pixels at a time (LSTM-16) and 28 pixels at a time (ROW-LSTM).

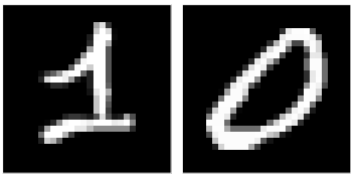
We measure the ACC metric (Lopez-Paz and Ranzato 2017) and we compare the performance of our models in combination with 6 CL strategies: EWC (Kirkpatrick et al. 2017), MAS



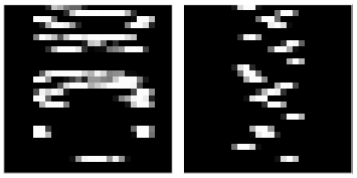
(a) SSC



(b) Quick, Draw!



(c) Row MNIST



(d) Permuted Row MNIST

Figure 10.1: Graphical representation of two patterns for each benchmark used in the experiments. Fig. 10.1a SSC uses 40 Mel features (columns) for each of the 101 time-steps (rows). Both plots use log values. Fig. 10.1b provides sketches of images (cat and airplane). Fig. 10.1c shows MNIST digits, which are provided to the model one row at a time in the Row MNIST version. Permuted MNIST (Fig. 10.1d) permutes images row-wise. Best viewed in color. Figures taken from Cossu, Carta, Lomonaco, et al. 2021.

(Aljundi, Babiloni, et al. 2018), LwF (Z. Li and Hoiem 2016), GEM (Lopez-Paz and Ranzato 2017), A-GEM (Chaudhry, Ranzato, et al. 2019) and Replay. We also report the final performance for the Naive strategy and for iid Joint Training. We did not address architectural approaches in this analysis, since they are a vast and diverse set of CL strategies. It is almost impossible to find good representatives for the existing ones. On the contrary, architectural and Replay approaches are much more restricted in the fundamental way they deal with forgetting. We will provide a study of architectural approaches for RNN and a proposal of a new CL strategy in Section 11.1

10.2 Results

We ran a set of experiments aimed at verifying 1) whether general CL strategies are able to mitigate forgetting in RNNs and 2) the effect of sequence length on the catastrophic forgetting. Table 10.2 reports the mean ACC and its standard deviation (over 5 runs) computed on PMNIST and SMNIST. Table 10.3 reports the same results computed on SSC and QD[1].

Catastrophic Forgetting with CL strategies In CI scenarios, importance-based regularization strategies are subjected to a complete forgetting, as also showed by Timothée Lesort, Stoian, et al. 2020 for feedforward models. We confirmed their results also for RNNs .The effectiveness of Replay in CI scenario is also confirmed by our experiments.

[1]We publish the code and configuration files needed to reproduce our experiments at this link `https://github.com/AndreaCossu/ContinualLearning_RecurrentNetworks`

Table 10.2: Mean ACC and standard deviation over 5 runs on PMNIST and SMNIST benchmarks.

PMNIST	MLP	ROW-LSTM	LSTM-4
EWC	$0.92_{\pm 0.02}$	$0.58_{\pm 0.09}$	$0.29_{\pm 0.05}$
LWF	$0.82_{\pm 0.03}$	$0.77_{\pm 0.05}$	$0.35_{\pm 0.04}$
MAS	$0.58_{\pm 0.04}$	$0.64_{\pm 0.07}$	$0.31_{\pm 0.04}$
GEM	$\mathbf{0.94_{\pm 0.01}}$	$0.90_{\pm 0.01}$	$0.71_{\pm 0.02}$
A-GEM	$0.80_{\pm 0.03}$	$0.58_{\pm 0.08}$	$0.20_{\pm 0.03}$
REPLAY	$0.92_{\pm 0.00}$	$\mathbf{0.92_{\pm 0.01}}$	$\mathbf{0.82_{\pm 0.01}}$
NAIVE	$0.31_{\pm 0.04}$	$0.61_{\pm 0.06}$	$0.30_{\pm 0.03}$
Joint Training	$0.97_{\pm 0.00}$	$0.96_{\pm 0.00}$	$0.94_{\pm 0.01}$

SMNIST	MLP	ROW-LSTM	LSTM-4
EWC	$0.21_{\pm 0.01}$	$0.21_{\pm 0.02}$	$0.19_{\pm 0.00}$
LWF	$0.70_{\pm 0.02}$	$0.31_{\pm 0.07}$	$0.26_{\pm 0.06}$
GEM	$\mathbf{0.91_{\pm 0.01}}$	$\mathbf{0.93_{\pm 0.03}}$	$\mathbf{0.66_{\pm 0.04}}$
A-GEM	$0.20_{\pm 0.00}$	$0.20_{\pm 0.00}$	$0.13_{\pm 0.02}$
MAS	$0.20_{\pm 0.00}$	$0.20_{\pm 0.00}$	$0.19_{\pm 0.00}$
REPLAY	$0.82_{\pm 0.02}$	$0.89_{\pm 0.01}$	$0.61_{\pm 0.06}$
NAIVE	$0.20_{\pm 0.00}$	$0.20_{\pm 0.00}$	$0.19_{\pm 0.00}$
Joint Training	$0.95_{\pm 0.00}$	$0.97_{\pm 0.00}$	$0.95_{\pm 0.01}$

GEM emerges as one of the best performing approaches not only for SMNIST and PMNIST, but also for the more complex SSC and QD. However, its computational cost remains very large, due to the quadratic optimization requested to find a projecting direction. Unfortunately, its more efficient version A-GEM results in complete forgetting. This is caused by the fact that A-GEM is not constrained by each previous experiences, but instead, it computes an approximation of the constraints by sampling patterns randomly from the memory. While this may be sufficient in a TI scenario, it does not work appropriately in a CI scenario.
LwF required careful tuning of hyperparameters, which questions its applicability to real-world CL environment. While LwF works on SMNIST, its performance rapidly decreases on more complex benchmarks like SSC and QD, where it suffers from complete forgetting.

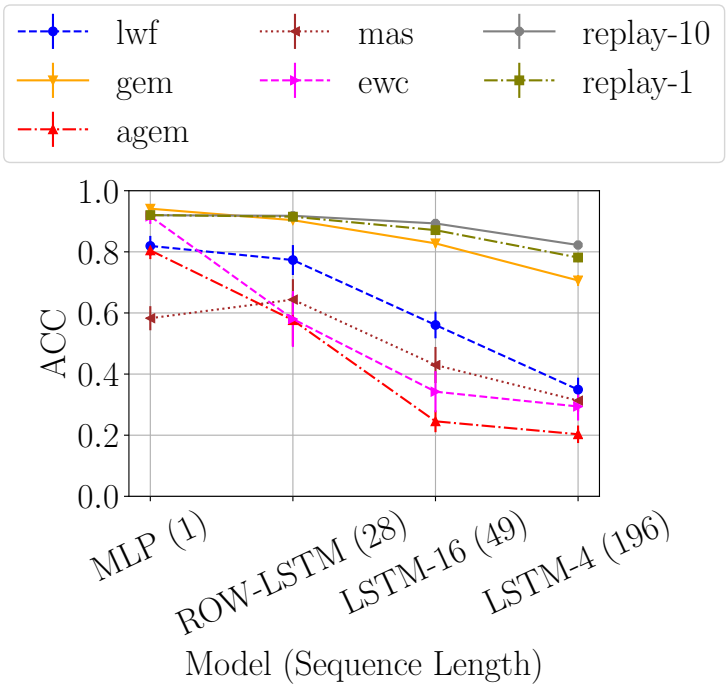
Sequence Length affects Catastrophic Forgetting Our experiments on PMNIST and SMNIST allow to draw some conclusions on the effect of sequence length on forgetting. Fig. 10.2 shows mean ACC results for different sequence lengths. The decreasing trends highlights a clear phenomenon: *long sequences cause more forgetting in RNNs*. Regularization strategies suffer the most when increasing sequence length. Although more robust than the other strategies in terms of final performance, Replay needs more patterns to recover from forgetting as sequences become longer. The results on PMNIST with MAS may seem to contradict the sequence-length effect, since ROW-LSTM performs surprisingly better than MLP. However, the performance has to be measured relatively to the Naive performance, in which no CL strategy is applied. When compared to this value, the ROW-LSTM does not perform any better than MLP. Instead it achieves a worse accuracy with respect to the Naive, as expected. The MAS-Naive ratio is 1.87 for MLP, but only 1.05 for MAS.

Table 10.3: Mean ACC and standard deviation over 5 runs on Synthetic Speech Commands and Quick, Draw! benchmarks.

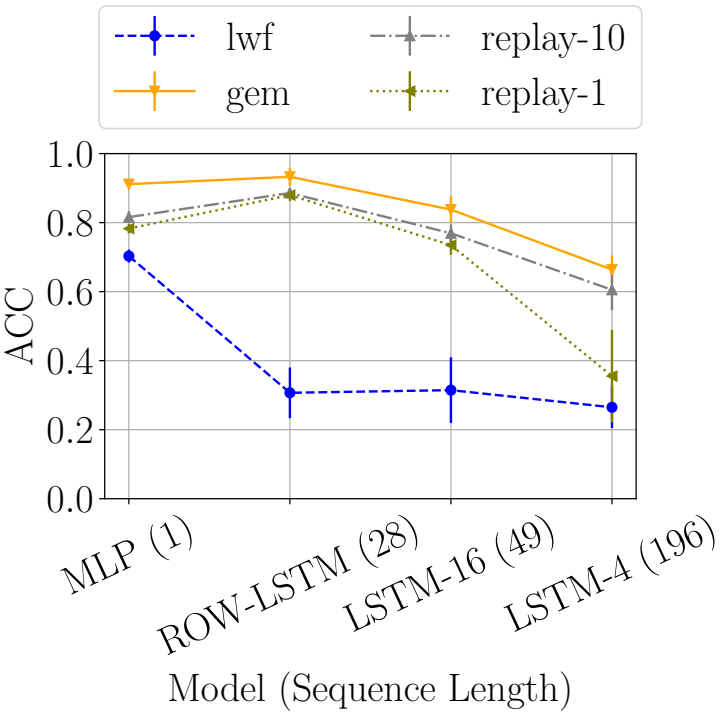
SSC	MLP	LSTM
EWC	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$
LWF	$0.05_{\pm 0.00}$	$0.12_{\pm 0.01}$
MAS	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$
GEM	$0.55_{\pm 0.00}$	$0.53_{\pm 0.01}$
A-GEM	$0.05_{\pm 0.00}$	$0.09_{\pm 0.01}$
REPLAY	$\mathbf{0.81_{\pm 0.03}}$	$\mathbf{0.73_{\pm 0.04}}$
NAIVE	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$
Joint Training	$0.93_{\pm 0.00}$	$0.89_{\pm 0.02}$

QD	LSTM
EWC	$0.12_{\pm 0.02}$
LWF	$0.12_{\pm 0.01}$
MAS	$0.10_{\pm 0.00}$
GEM	$0.47_{\pm 0.03}$
A-GEM	$0.10_{\pm 0.00}$
REPLAY	$\mathbf{0.49_{\pm 0.02}}$
NAIVE	$0.10_{\pm 0.00}$
Joint Training	$0.96_{\pm 0.00}$

We can understand the positive correlation between forgetting and sequence length in light of the BPTT algorithm and the vanishing/exploding gradient phenomena. Consider importance-based regularization strategies, like EWC. The importance associated to each parameter is fundamental to carefully control the trade-off between stability and plasticity of the model. In recurrent model, the norm of the matrix $\boldsymbol{W}$ of adaptive recurrent connection drives the gradient signal. The gradient can take on very large values in correspondence of large norms, and very small values in correspondence of small norms. The sequence length plays a fundamental role: even a norm slightly larger/smaller than 1 may lead to exploding/vanishing gradient if multiplied by itself many times. In BPTT, this multiplication is repeated for each time-step in the sequence (Eq. 2.11). Since the importance values for EWC are proportional to the gradient squared, either the exploding or the vanishing gradient is detrimental to the final performance. In the case of exploding gradient, all the recurrent parameters will be assigned a very large importance values, keeping them almost fixed. In the case of vanishing gradient, all recurrent parameters will be assigned very small importance values, leaving them free to adapt. In both cases, there is no distinction *within* the set of recurrent parameters. They are all treated the same way. The importance metric is not representative anymore of the true importance of a weight for a given task. A similar behavior probably affects the other strategies like LwF and Replay, leading to a progressively lower performance as sequence length increases.



(a) Permuted MNIST



(b) Split MNIST

Figure 10.2: Average ACC on all experiences for different sequence lengths and different CL strategies. Sequence length causes a decrease in performances among all strategies. Best viewed in color. Plots taken from Cossu, Carta, Lomonaco, et al. 2021.

Chapter 11

Overcoming the Sequence Length Issue in Recurrent Models

This Chapter presents three different approaches to the Sequence Length Issue. The Chapter is structured as follows:

- We propose the Gated Incremental Memories. The model is an architectural approach that removes the Sequence Length Issue by combining a task detector with a separate module for each task (Section 11.1).
- We leverage the Echo-State Network model to efficiently address the Sequence Length Issue (Section 11.2). By using a randomized, fixed recurrent component, we show that the Echo-State Network model is very resilient to forgetting. Also, it can exploit CL strategies that are otherwise not applicable to fully-trained recurrent models.
- We present a real-world CL application based on Human State Monitoring (Section 11.3). Here, forgetting is mitigated by the scenario, since the introduction of novel subjects represents a softer drifts than the one of a CI scenario. We leverage recurrent models to predict the state of each subject from sensors in a continual fashion.

The Gated Incremental Memories have been published in Cossu, Carta, and Bacciu 2020. The application of Echo-State Networks to mitigate the Sequence Length Issue has been published in Cossu, Bacciu, et al. 2021. The Human State Monitoring application has been published in Matteoni et al. 2022.

11.1 Gated Incremental Memories

We propose an architectural strategy for RNNs named Gated Incremental Memory (GIM). GIM imbues RNN architectures with CL skills by incrementally adding new modules to capture the drifts in input distribution while avoiding forgetting. The GIM architecture is inspired by the Progressive network but, unlike Progressive networks, GIM leverages a set of autoencoders to automatically recognize input distributions at test time in order to select the correct module. This frees GIM from the need to use task labels at test time, a strong requirement present in Progressive networks. Architectural strategies which leverage different sub-modules for different experiences are not subjected to the Sequence Length Issue. In fact, the weight interference is removed by design and forgetting depends solely on the ability to choose the appropriate sub-module during inference.

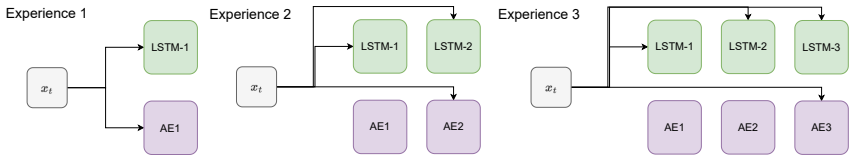


Figure 11.1: Incremental expansion of the GIM-LSTM during training on 3 experiences. When a new experience is encountered a new module is added. Figure taken from Cossu, Carta, and Bacciu 2020.

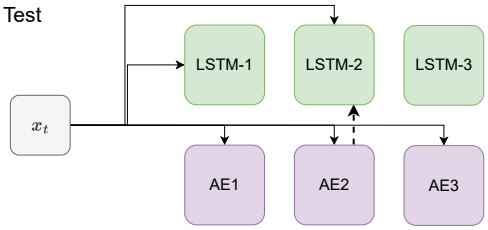


Figure 11.2: GIM-LSTM at inference time. The input $\mathbf{x}$ is encoded by all the autoencoders. The autoencoder with the minimum reconstruction error (AE2 in the example) determines which module to choose (LSTM-2 in the example). The input is passed to the chosen module to compute the output (dashed line). Figure taken from Cossu, Carta, and Bacciu 2020.

Figures 11.1 and 11.2 provide an overview of the entire GIM architecture during training and test.

11.1.1 GIM Modules

The main component of GIM is the RNN *module*. When learning the first experience of the stream, GIM is composed by only one RNN module. As soon as a new experience arrives, a new RNN module is added on top of the existing architecture and connected to the previous one (Figure 11.1). We studied the behavior of GIM wit both LSTM modules (GIM-LSTM) and LMN modules (GIM-LMN). This provides an overview of the impact GIM has on both gated and non-gated, orthogonal RNNs. To compute the output for each element $\boldsymbol{x_t}$ of the input sequence, each GIM-LSTM module takes as input the current sample and the hidden state of the previous module. In an LMN, the hidden state is composed by the concatenation of the functional hidden state $\boldsymbol{h_t}$ with the memory state $\boldsymbol{h_t^m}$. During training, the output produced by the last added module is used as final output for the GIM architecture. Like Progressive networks, forgetting is prevented by freezing the existing modules when a new one is added. Therefore, each module becomes an expert of its own domain. Given an input sequence $\mathbf{x} = \boldsymbol{x_1}, \ldots, \boldsymbol{x_T}$, the output $\boldsymbol{y}$ for a GIM-LMN with N modules can be computed as follows:

$$
\begin{aligned}
\boldsymbol{h^m_{:,1}}, \boldsymbol{h_{:,1}} &= \text{LMN}_1(\mathbf{x_t}, \boldsymbol{h^m_{0,1}}) \\
\boldsymbol{h^m_{:,j}}, \boldsymbol{h_{:,j}} &= \text{LMN}_j([\mathbf{x_t}; \boldsymbol{h^m_{:,j-1}}; \boldsymbol{h_{:,j-1}}], \boldsymbol{h^m_{0,j}}), \quad j = 2, ..., N \\
\boldsymbol{y} &= \sigma(\boldsymbol{W_j^{mo}} \boldsymbol{h^m_{T,N}}),
\end{aligned}
$$

where LMN_j is the RNN module corresponding to the j-th experience, $\boldsymbol{h^m_{:,j}}$ and $\boldsymbol{h_{:,j}}$ are the sequences of memory states and hidden functional states of module j, respectively. The symbol : indexes all the steps in the sequence and $[\cdot;\cdot]$ is the concatenation operator between vectors. The aggregated output $\boldsymbol{y}$ is computed by passing the final memory state $\boldsymbol{h^m_{T,N}}$

through a linear layer.
LSTM modules follow the same logic for the forward pass. They do not have a separate memory state $\boldsymbol{h_t^m}$ and leverage only the hidden state $\boldsymbol{h_t}$. A detailed description is provided in Algorithm 4 for LSTM modules and Algorithm 5 for LMN modules. Algorithm 6 shows the pseudocode for the training procedure.

Gating Autoencoders

At inference time, GIM must choose which module to use to compute the final output. To solve this problem, each module is associated with an LSTM autoencoder (AE)Srivastava et al. 2015, which is a sequence-to-sequence model trained to first encode and then reconstruct the input sequence $\mathbf{x}$. Each AE is added to GIM together with its associated RNN module. When a new module is added (i.e., a new experience is encountered), the previous AEs are kept frozen. This makes each AE able to recognize input data coming from a specific experience. Algorithm 6 shows the procedure to reconstruct the input using the AE encoder and decoder.

Inference

At inference time, a GIM with N modules computes the final output for test input $\boldsymbol{x}$ in three steps:

1. each AE reconstructs the input sequence: $\hat{\mathbf{x}}_\mathbf{i} = \text{AE}_i(\mathbf{x}),\ i = 1, ..., N$;
2. the winning AE, AE_k is the one achieving the minimum reconstruction error: $k = \arg\min_i \text{MSE}(\hat{\mathbf{x}}_\mathbf{i}, \mathbf{x})$;
3. the RNN module (either LSTM or LMN) corresponding to the winning AE is used to compute the final output: $\hat{\boldsymbol{y}} = \text{RNN}_k(\mathbf{x})$.

We formally describe the output computation at inference time in Algorithm 6.

Algorithm 4 GIM-LSTM Forward Pass for Module N

function LSTM-MODULE-FW(GIM, $\mathbf{x}$, N)
Require: GIM-LSTM with at least N modules, $N \geq 1$, $\mathbf{x}$ with T timesteps
  $\boldsymbol{h_{0,1}} \leftarrow \mathbf{0}$
  $\boldsymbol{h_{:,1}} \leftarrow \text{GIM.LSTM}_1(\mathbf{x}, \boldsymbol{h_{0,1}})$
  for $d \leftarrow 2, N$ **do**
    $\boldsymbol{h_{0,d}} \leftarrow \mathbf{0}$
    $\hat{\mathbf{x}} \leftarrow [\mathbf{x}; \boldsymbol{h_{:,d-1}}]$
    $\boldsymbol{h_{:,d}} \leftarrow \text{GIM.LSTM}_d(\hat{\mathbf{x}}, \boldsymbol{h_{0,d}})$
  end for
  $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{W^{out}} \boldsymbol{h_{T,N}}$
  return $\hat{\boldsymbol{y}}$
end function

11.1.2 Advantages of the GIM architecture

GIM, like Progressive networks, is capable of learning multiple distributions without being affected by forgetting. Freezing old parameters easily guarantees that the model will retain the knowledge about previous experiences. Additionally, GIM overcomes one of the major

Algorithm 5 GIM-LMN Forward Pass for Module N

function LMN-MODULE-FW(GIM, $\mathbf{x}$, N)
Require: GIM-LMN with at least N modules, $N \geq 1$, $\mathbf{x}$ with T timesteps
    $\boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{0,1}} \leftarrow \mathbf{0}$
    $\boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{:,1}}, \boldsymbol{h}_{\mathbf{:,1}} \leftarrow \text{GIM.LMN}_1(\mathbf{x}, \boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{0,1}})$
    for $d \leftarrow 2, N$ **do**
        $\boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{0},\boldsymbol{d}} \leftarrow \mathbf{0}$
        $\hat{\mathbf{x}} \leftarrow [\mathbf{x}; \boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{:},\boldsymbol{d-1}}; \boldsymbol{h}_{\mathbf{:},\boldsymbol{d-1}}]$
        $\boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{:},\boldsymbol{d}}, \boldsymbol{h}_{\mathbf{:},\boldsymbol{d}} \leftarrow \text{GIM.LMN}_d(\hat{\mathbf{x}}, \boldsymbol{h}^{\boldsymbol{m}}_{\mathbf{0},\boldsymbol{d}})$
    end for
    $\hat{\boldsymbol{y}}^{\boldsymbol{m}}_{\boldsymbol{N}} \leftarrow \boldsymbol{W}^{\boldsymbol{mo}}_{\boldsymbol{N}} \boldsymbol{h}^{\boldsymbol{m}}_{\boldsymbol{T},\boldsymbol{N}}$
    return $\hat{\boldsymbol{y}}^{\boldsymbol{m}}_{\boldsymbol{N}}$
end function

Algorithm 6 Functions to compute the reconstruction of the autoencoder, for training it, and for choosing the GIM module.

function RECONSTRUCTION(AE, $\mathbf{x}$)
    $\boldsymbol{h}_{\boldsymbol{enc},\mathbf{0}} \leftarrow \mathbf{0}$
    $\boldsymbol{h}_{\boldsymbol{enc},\mathbf{:}} \leftarrow \text{AE.LSTM}_{enc}(\mathbf{x}, \boldsymbol{h}_{\boldsymbol{enc},\mathbf{0}})$
    $\boldsymbol{h}_{\boldsymbol{dec},\mathbf{:}} \leftarrow \text{AE.LSTM}_{dec}(\mathbf{0}, \boldsymbol{h}_{\boldsymbol{enc},\boldsymbol{T}})$
    $\tilde{\mathbf{x}} \leftarrow \text{AE.}\boldsymbol{W}^{\boldsymbol{out}} \boldsymbol{h}_{\boldsymbol{dec},\mathbf{:}}$
    return $\tilde{\mathbf{x}}$
end function
function AE-TRAIN($\mathcal{D}$)
Require: $\mid \mathcal{D} \mid > 1$
    $l_{ae} \leftarrow []$
    while a new experience D_k is available **do**
        AE $\leftarrow$ init-autoencoder()
        l_{ae}.append(AE)
        for training batch $\mathbf{x} \in D_k$ **do**
            $\tilde{\mathbf{x}} \leftarrow$ RECONSTRUCTION(AE, $\mathbf{x}$)
            $J \leftarrow \text{MSE}(\mathbf{x}, \tilde{\mathbf{x}})$
            $\frac{\partial L}{\partial \boldsymbol{\theta}} \leftarrow \text{backprop}(L)$
            Take a descent step along $\frac{\partial L}{\partial \boldsymbol{\theta}}$
        end for
    end while
    return l_{ae}
end function
function GIM-INFERENCE(GIM, $\mathbf{x}$)
    $l_{rec} \leftarrow []$
    for AE $\in$ GIM.l_{ae} **do**
        $\tilde{\mathbf{x}} \leftarrow$ RECONSTRUCTION(AE, $\mathbf{x}$)
        $l_{rec}.append(\text{MSE}(\mathbf{x}, \tilde{\mathbf{x}}))$
    end for
    $m \leftarrow \arg\min l_{rec}$ ▷ index of the best autoencoder
    $\hat{\boldsymbol{y}} \leftarrow \text{LSTM-MODULE-FW}(\text{GIM}, \mathbf{x}, m)$
    return $\hat{\boldsymbol{y}}$
end function

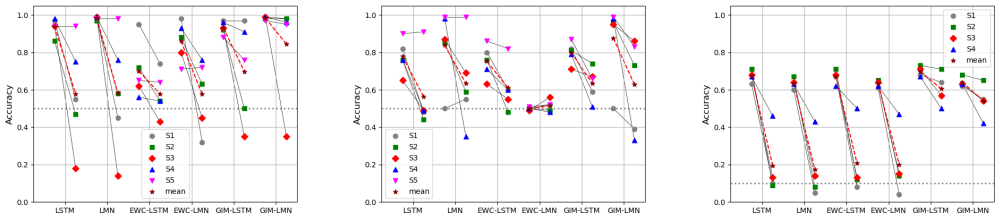
drawbacks of Progressive networks Rusu et al. 2016: it does not require explicit knowledge about input distributions at test time. In fact, gating autoencoders are able to autonomously recognize the current input and use the appropriate module to compute the output. Compared to Progressive networks, GIM simplifies the inter-modules connections: while Progressive networks use feedforward networks, created between a module and *all* the next ones, GIM employs only concatenation between vectors and connects only adjacent modules. Since both Progressive and GIM employ a separate module for each of the N experiences, the number of adaptive parameters in the architecture is quadratic in N for the Progressive network, while it is linear for GIM.

11.1.3 Experiments

We experimented with CI scenarios built out of three datasets. MNIST and Devanagari are image classification datasets, while Audioset (Gemmeke et al. 2017) is a dataset of audio clips of a specific sound. With respect to image datasets, Audioset is more representative of a sequential, temporally-correlated environment[1].
MNIST and Devanagari are both composed of 28×28 gray-scaled images, taken one pixel at a time and resulting in sequences of 784 steps. For Devanagari, the size of the image is obtained after removing the zero-padding present in the original dataset. The length of the sequence makes the tasks challenging for recurrent models, due to the vanishing gradient issue. This might directly worsen forgetting in CL with RNNs, as we have seen in Section 10. Each sequence is shuffled according to a fixed, random permutation. Permuting the images ensures that the RNN performance is not affected by the long sequence of non-informative elements pixels present at the end of each sequence. For Split pixel-MNIST and Split pixel-Devanagari, we built a CL stream with 5 experiences and 2 classes in each experience. Following Schak and Gepperth 2019, for Devanagari we randomly sampled 10 classes out of the available 46. For Audioset, we embedded each 10-seconds audio clip with a CNN acoustic model into 10 vectors, one per second. Each vector has 128 elements. Similar to Kemker et al. 2018, we built Split Audioset with a stream composed of 4 experiences with 10 classes in each experience. Audioset data has already been used in the literature to assess CL skills (Kemker et al. 2018). However, the authors focused on the task from a *static* perspective, relying on the use of feedforward models only. The sequential aspect of the task, however, is completely lost. At the best of our knowledge, we are the first to tackle Audioset in CL scenarios with recurrent models. It is also important to notice that the task difficulty is increased when using recurrent networks, since the model is not able to see the input in its entirety (like in feedforward networks).

On MNIST and Devanagari we trained all models with Adam optimizer Kingma and Ba 2015, learning rate of $1e-4$, mini-batch size of 32. The number of hidden units is set to 128 for both LSTM-based and LMN-based (functional and memory component) models.
It is well known that orthogonal initialization of memory weight matrixes can improve learning when dealing with long sequences and linear memories Mhammedi et al. 2017; Henaff et al. 2016. Therefore, we chose to use such initialization for LMN-based models and also to preserve it during training through an additional penalty in the loss function. The orthogonal-preserving penalization is expressed by $\beta\|(\boldsymbol{W^{mm}})^T\boldsymbol{W^{mm}} - \boldsymbol{I}\|^2$, where $\boldsymbol{W^{mm}}$ is the memory weight matrix of the LMN, $\boldsymbol{I}$ is the identity matrix and β is the hyperparameter associated to the penalization. We used a value of $\beta = 0.1$, which was capable of preserving orthogonality on all experiments.

[1]Code to reproduce results is available at `https://github.com/AndreaCossu/ContinualLearning-SequentialProcessing`



(a) MNIST (b) Devanagari (c) Audioset

Figure 11.3: Paired plots for the three benchmarks. Each pair plot shows, for each model and for each experience, mean validation accuracy (left point) computed after training on that experience, and mean test accuracy (right point) computed at the end of the entire training on all experiences. Validation and test accuracy are connected by a line. Therefore, the drop in performance due to forgetting is the difference between the two points. The red dashed line is the average among experiences. Horizontal dotted line is equal to random classifier performance (0.5 for MNIST and Devanagari, 0.1 for Audioset). Figure taken from Cossu, Carta, and Bacciu 2020.

On Audioset we used smaller RNN models with 16 hidden units for both LSTM and LMN (memory and functional component). This choice was determined by the limited amount of data points available for each class in Audioset: larger models led to over-fitting without any performance improvements. We used the RMSProp optimizer, with learning rate of $3e-5$, momentum of 0.9 and L2 regularization with hyperparameter of $1e-3$ and mini batch size of 4.
For all the benchmarks, LSTM autoencoders use 500 hidden units on encoder and decoder (36.22% compression on MNIST and Devanagari, 60.94% on Audioset) trained with Adam optimizer and learning rate of $1e-4$. On Audioset, we also use L2 regularization with hyperparameter $1e-3$.
We compare GIM-LSTM and GIM-LMN with LSTM and LMN and with the popular EWC method Kirkpatrick et al. 2017. EWC requires to choose the value of the hyperparameter λ regulating the trade-off between new incoming experience and older ones. We choose the value of 0.4 out of $0.01, 0.1, 0.4, 1.0$, since it gave the best performances on a held-out validation set.

11.1.4 Results

Table 11.1 provides the results of our experiments, averaged over 5 runs. Paired plots (Figure 11.3) show the comparison between the validation performance for each experience computed after training on that experience (on the left) and the test performance computed after the final training is completed for all the experiences. Therefore, the forgetting for each configuration can be evaluated by looking at the difference between the left and right points for each experience.
MNIST and Devanagari are binary classification tasks, hence a random classifier would score 0.50 accuracy on all experiences. On Audioset, being composed of 10 classes per experience, a random classifier would score 0.10 accuracy on all experiences.
We also show examples of learning curves (Figure 11.4), comparing GIM-LMN with GIM-LSTM. Audioset shows early overfitting even with small models. A similar behavior, even if

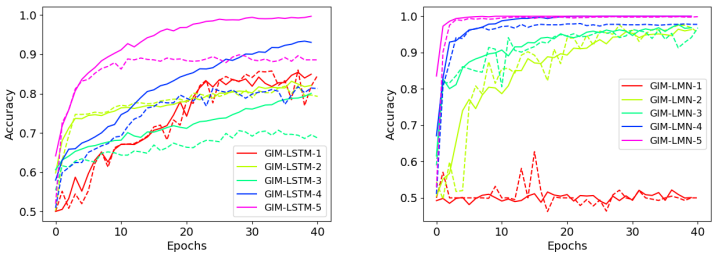
Table 11.1: Validation (top of cell) and Test (bottom of cell) accuracy (± std) on all datasets (D) and experiences (S). Validation accuracy on each experience computed after training on that specific experience. Test accuracy computed at the end of training on all experiences. For each dataset, final row **A** shows Validation / Test accuracy averaged over all experiences. Results averaged over 5 runs.

D	S	LSTM	LMN	EWC LSTM	EWC LMN	GIM LSTM	GIM LMN
MNIST	1	0.97 ± .01 0.55 ± .03	0.99 ± .02 0.45 ± .05	0.95 ± .01 0.74 ± .06	0.98 ± .02 0.32 ± .08	0.97 ± .041 0.97 ± .09	0.99 ± .01 **0.98 ± .07**
	2	0.86 ± .03 0.47 ± .07	0.97 ± .02 0.58 ± .06	0.72 ± .04 0.54 ± .08	0.88 ± .04 0.63 ± .07	0.92 ± .04 0.50 ± .08	0.98 ± .02 **0.98 ± .09**
	3	0.94 ± .04 0.18 ± .08	0.99 ± .05 0.14 ± .04	0.62 ± .03 0.43 ± .03	0.80 ± .03 **0.45 ± .08**	0.93 ± .01 0.35 ± .08	0.99 ± .02 0.35 ± .09
	4	0.98 ± .03 0.75 ± .06	0.99 ± .04 0.76 ± .04	0.56 ± .03 0.54 ± .09	0.93 ± .06 0.76 ± .08	0.96 ± .05 0.91 ± .12	0.99 ± .02 **0.96 ± .09**
	5	0.94 ± .01 0.94 ± .06	0.98 ± .03 0.98 ± .03	0.65 ± .05 0.64 ± .05	0.71 ± .04 0.72 ± .06	0.88 ± .01 0.76 ± .08	0.97 ± .04 **0.95 ± .07**
	A	0.94 / 0.58	0.98 / 0.58	0.70 / 0.58	0.86 / 0.58	0.93 / 0.70	0.98 / **0.84**
Devanagari	1	0.82 ± .04 0.48 ± .03	0.50 ± .04 0.55 ± .06	0.80 ± .05 **0.60 ± .03**	0.50 ± .05 0.52 ± .04	0.82 ± .01 0.59 ± .02	0.50 ± .07 0.39 ± .06
	2	0.76 ± .07 0.44 ± .05	0.85 ± .07 0.59 ± .03	0.76 ± .06 0.48 ± .07	0.50 ± .08 0.49 ± .04	0.81 ± .11 **0.74 ± .07**	0.96 ± .08 0.73 ± .07
	3	0.65 ± .09 0.49 ± .08	0.87 ± .03 0.69 ± .03	0.63 ± .05 0.55 ± .05	0.49 ± .08 0.56 ± .08	0.71 ± .07 0.67 ± .09	0.95 ± .05 **0.86 ± .10**
	4	0.76 ± .05 0.49 ± .04	0.98 ± .03 0.35 ± .05	0.71 ± .07 **0.60 ± .06**	0.50 ± .06 0.48 ± .09	0.79 ± .08 0.51 ± .08	0.98 ± .04 0.33 ± .08
	5	0.90 ± .05 0.91 ± .05	0.99 ± .04 0.99 ± .06	0.86 ± .07 0.82 ± .06	0.51 ± .08 0.52 ± .4	0.87 ± .03 0.66 ± .09	0.99 ± .05 **0.83 ± .03**
	A	0.78 / 0.56	0.84 / 0.63	0.75 / 0.61	0.50 / 0.51	0.80 / **0.63**	0.88 / **0.63**
Audioset	1	0.63 ± .01 0.10 ± .02	0.60 ± .01 0.05 ± .02	0.67 ± .03 0.08 ± .04	0.61 ± .03 0.04 ± .01	0.68 ± .02 **0.64 ± .04**	0.62 ± .03 0.55 ± .01
	2	0.71 ± .04 0.09 ± .02	0.67 ± .03 0.08 ± .01	0.71 ± .03 0.12 ± .02	0.65 ± .03 0.14 ± .02	0.73 ± .03 **0.71 ± .03**	0.68 ± .02 0.65 ± .08
	3	0.68 ± .03 0.13 ± .03	0.64 ± .01 0.14 ± .2	0.68 ± .01 0.13 ± .02	0.64 ± .01 0.15 ± .01	0.71 ± .04 **0.57 ± .03**	0.63 ± .03 0.54 ± .02
	4	0.67 ± .01 0.46 ± .02	0.63 ± .02 0.43 ± .02	0.62 ± .03 0.50 ± .01	0.62 ± .02 0.47 ± .02	0.67 ± .03 **0.50 ± .03**	0.63 ± .04 0.42 ± .05
	A	0.67 / 0.20	0.64 / 0.18	0.67 / 0.21	0.63 / 0.20	0.70 / **0.61**	0.64 / 0.54

less drastic, is detected on Devanagari. It is, however, important to stress the fact that the learning curves cannot show the effect of forgetting because they are computed using the data from the current experience, while we are interested in the final test accuracy, measured after training on all experiences.

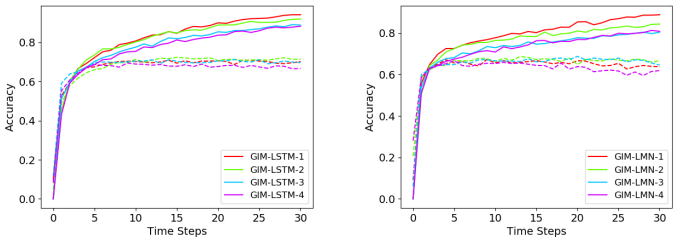
11.1.5 Discussion

In accordance with the results presented in Schak and Gepperth 2019, LSTM and LMN models suffer from catastrophic forgetting of old experiences when trained without any CL strategy. EWC alone is not able to mitigate catastrophic forgetting. This confirms and



(a) GIM-LSTM on Devanagari: 40 epochs

(b) GIM-LMN on Devanagari: 40 epochs



(c) GIM-LSTM on Audioset: 30,000 time steps, printed every 1000 steps.

(d) GIM-LMN on Audioset: 30,000 time steps, printed every 1000 steps.

Figure 11.4: Examples of accuracy curves on training set (solid line) and validation set (dashed line). Figure taken from Cossu, Carta, and Bacciu 2020.

extends the results presented in Section 10 also to Split Devanagari and Split Audioset benchmarks.

GIM models are by far the best performing ones, since they successfully learn with limited forgetting.

On Audioset, GIM-LSTM and GIM-LMN are capable of maintaining comparable performance on all experiences once training is finished, while performance for EWC-based models drops below the random baseline for some experiences. This means that the autoencoders successfully recognize the incoming distribution and select the correct module to produce the output without any information on the incoming input labels.

On MNIST we obtained similar results, with some exceptions: GIM-LSTM underwent complete forgetting on 2 experiences out of 5, while GIM-LMN experienced it on 1 out of 5. In those cases, autoencoders failed to reconstruct the input sequences, leading to the choice of the wrong module for the final classification. On these experiences, the EWC version of LMN and LSTM surpassed GIM architectures.

Devanagari is the most challenging task: GIM models still exhibited complete forgetting on 2 experiences out of 5, with reduced performances on almost all the others. However, this behavior is common to all models on this task. EWC-LMN does not show significant differences between performances on the validation set and final test only because it is unable to learn the experience, achieving an accuracy equivalent to the one of a random classifier.

Architectural strategies represent a flexible approach to effectively mitigate forgetting and circumvent the Sequence Length Issue in recurrent models.

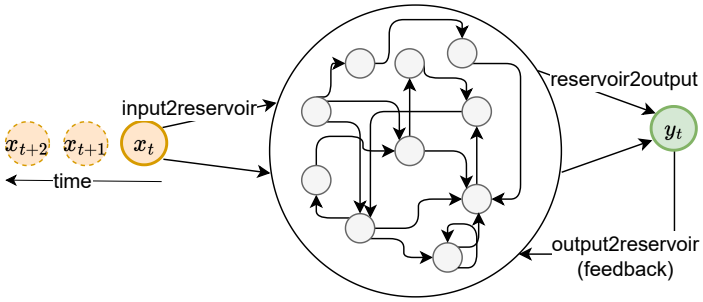


Figure 11.5: Description of an Echo-State Network. The input sequence (left) is fed to the reservoir by the input2reservoir connections. The reservoir (center) processes the sequence one time-steps at a time and maps each time-step to a final output (right) through the reservoir2output connections. Optionally, the output can be fed-back to the reservoir via the output2reservoir connections.

11.2 Randomized Recurrent Models

We address CL in Sequential Data Processing (where each pattern is a sequence) with a family of randomized RNNs called Echo State Networks (ESNs) (Jaeger and Haas 2004). We recall (Equation 2.18) that an ESN computes the output for each time-step t via $\boldsymbol{h_t} = \sigma(\boldsymbol{W^x x_t} + \boldsymbol{W^h h_{t-1}} + \boldsymbol{W^f \hat{y}_{t-1}})$. Figure 11.5 shows a pictorial representation of such equation. In the following experiments, we will not consider the feedback connections $\boldsymbol{W^f}$ and we will consider only the input-to-reservoir connections $\boldsymbol{W^x}$ as well as the reservoir-to-reservoir (recurrent) connections $\boldsymbol{W^h}$.

As we have seen in Chapter 10, many CL strategies behave differently when applied to RNNs than when applied to feed-forward models. The input sequence length is one important factor affecting the final performance which positively correlates with the amount of forgetting a RNN experiences. These results apply to fully-trained RNNs, where the recurrent connections are jointly trained with the linear output layer. We now instead turn our attention to ESNs, which represent un-trained, randomized RNNs.

We provide the first experimental analysis of forgetting in ESNs and we show how this class of models behave in combination with popular CL strategies. By treating the untrained reservoir of an ESN as a feature extractor for sequences, we are able to 1) restrict the application of CL strategies to the final, linear readout and 2) to leverage efficient CL strategies operating solely on the final layer.

The latter point marks a clear advantage in using ESNs with respect to fully-trained models. Many CL approaches for Computer Vision, in fact, often exploit pre-trained feature extractors. However, it is not obvious how to employ the same approaches on recurrent models, which lack effective pre-training protocols. Instead, ESNs allow to circumvent the problem, by treating the fixed reservoir as a pre-trained feature extractor. As we will see, the SLDA strategy is a simple, yet compelling example of how ESNs can be leveraged in CL. SLDA would not be applicable to fully-trained recurrent models since, as we have discussed in Chapter 4, it requires a fixed representation for the application of the Linear Discriminant Analysis.

Our results indicate that ESN exhibits competitive performance with respect to LSTM and it is amenable to be applied with CL solutions operating in challenging settings like streaming learning.

Table 11.2: Mean ACC and standard deviation over 5 runs on SMNIST and SSC benchmarks. SLDA is applied only to ESN since it assumes a fixed feature extractor. SMNIST contains 5 experiences, while SSC contains 10 experiences. † results are taken from Section 10, except for Replay which has been recomputed to guarantee the use of the same Replay policy (200 patterns in memory).

SMNIST	LSTM†	ESN
EWC	$0.21_{\pm 0.02}$	$0.20_{\pm 0.00}$
LWF	$0.31_{\pm 0.07}$	$0.47_{\pm 0.07}$
REPLAY	$\mathbf{0.85_{\pm 0.03}}$	$0.74_{\pm 0.03}$
SLDA	—	$\mathbf{0.88_{\pm 0.01}}$
NAIVE	$0.20_{\pm 0.00}$	$0.20_{\pm 0.00}$
JOINT	$0.97_{\pm 0.00}$	$0.97_{\pm 0.01}$

SSC	LSTM†	ESN
EWC	$0.10_{\pm 0.00}$	$0.09_{\pm 0.02}$
LWF	$0.12_{\pm 0.01}$	$0.12_{\pm 0.02}$
REPLAY	$\mathbf{0.74_{\pm 0.07}}$	$0.36_{\pm 0.07}$
SLDA	—	$\mathbf{0.57_{\pm 0.03}}$
NAIVE	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$
JOINT	$0.89_{\pm 0.02}$	$0.91_{\pm 0.02}$

11.2.1 Experiments

We compared the performance of LSTM and ESN when equipped with 4 CL strategies already presented: EWC, LwF, Replay, SLDA. We tested our methods on 2 different sequence classification tasks, in which each pattern (a sequence) is associated to a target class. We chose 2 popular CI benchmarks. We measure the final performance through the ACC metric. We work in a single-head setup without assuming any knowledge about the input distribution at test time.
We used Split pixel-MNIST (SMNIST) and Synthetic Speech Commands (SSC) as the benchmarks for our experiments. We took each MNIST image one row at a time, resulting in input sequences with 28 steps. We performed grid search on all the strategies for ESN and on Replay for LSTM. The other results for LSTM are taken from the experiments presented in Section 10, since the experimental setup is the same. To perform grid search for SSC, we took 3 held-out experiences for model selection and 10 for model assessment. To fairly compare ESN and LSTM, we select a model configuration whose only requirement is to be able to learn effectively at training time. Then, the performance in terms of forgetting depends mostly on the CL strategy (subjected to grid search) and not on the specific model setting. We train the ESN readout with Adam optimizer and backpropagation, since CL requires to update the model continuously, possibly without storing its activations. We used the Avalanche (Lomonaco, Pellegrini, et al. 2021) framework for all our CL experiments. We make publicly available the code together with configurations needed to reproduce all experiments[2].

Results Table 11.2 reports the ACC metric and its standard deviation over 5 runs for the best configuration found in model selection. Our results show that ESN performs better or comparably to the LSTM network. EWC is not able to tackle CI scenarios, neither with LSTM nor with ESN. It achieves a performance equal to the Naive one. LwF, instead, manages to reduce forgetting in the simplest SMNIST benchmarks. In this context, applying LwF only on the feedforward component results in a better performance with respect to the LSTM. This is compatible with results presented in Section 10 and highlights one of the improvements when using ESNs for CL. However, when facing more complex scenarios like SSC, LwF fails to provide any benefits with respect to Naive finetuning. The advantage in using ESNs clearly emerges when studying the behavior of SLDA strategy. This strategy

[2] `https://github.com/Pervasive-AI-Lab/ContinualLearning-EchoStateNetworks`

effectively tackles CI benchmarks like SSC. The absolute performance of SLDA in SSC is still far from the Joint Training upper bound. However, this does not mean that SLDA suffers from large forgetting effect. In fact, it achieves an average experience forgetting (the difference between the accuracy after training on a certain experience and the corresponding accuracy after training on all experiences) of 0.14 ± 0.02. The remaining 20% of difference from the Joint Training is explained by the fact that SLDA is a streaming strategy which trains the model only on a single epoch. Therefore, on complex scenarios like SSC it achieves a lower accuracy.

11.2.2 Discussion

We studied the ability of ESNs to mitigate forgetting in CL environments. We provided the first experimental evaluation of ESNs trained with popular CL strategies. Our analysis showed that, apart from Replay strategies, ESNs perform comparably with fully trained recurrent networks like LSTM. Moreover, since the reservoir is a fixed feature extractor, it is possible to train ESNs with CL strategies like SLDA which are not applicable to LSTM. SLDA obtains a good performance in CI scenarios.
This work could foster new studies and applications of CL with ESNs: the renowned computational efficiency of ESNs may be particularly interesting for streaming or task-free CL. The possibility to implement ESNs in specialized hardware opens to the continuous training of such models on low resources devices. Neuromorphic hardware, for example, promises to efficiently model neural-like computation as the one required by neural networks and recurrent models. Due to their limited training budget, ESNs are a perfect test-bed to explore these frontiers of research.
ESNs are not the only family of models with an untrained component. More in general, CL with partially trained networks constitutes an interesting avenue of research, due to the fact that fixed connections are not subjected to catastrophic forgetting. Alternatively, reservoir in ESNs may also be finetuned during training to better adapt to new experiences. In particular, unsupervised finetuning through backpropagation-free methods (e.g. Hebbian learning, intrinsic plasticity) may provide quicker adaptation and more robust representations. The design of new CL strategies which exploit reservoir finetuning while keeping forgetting into consideration would provide us with a deeper understanding of the CL learning capabilities of ESNs.

11.3 Continual Human State Monitoring

Table 11.3: HSM datasets summary.

	WESAD	**ASCERTAIN**	**Cust. ASCERTAIN**
Subjects	15	17	17
Classes	4	4	4
Sequence length	100	160	160
Features	14	18	18
Training set size	4500	2160	1836
Test set size	1500	720	720

The Sequence Length Issue was present even in DI scenarios, as showed by the experiments on PMNIST in Chapter 10. However, PMNIST introduces new instances of previous classes with a strong drift. Each digit is subjected to a random permutation, which completely

changes the structure of the example. Instead, in line with CIR scenarios (Chapter 6), what if we introduce new instances which are sampled in a more natural manner? We focus on Human State Monitoring tasks, where a fixed set of actions performed by multiple subjects are monitored via wearable sensors. We created a DI scenario by introducing new subjects over time and by monitoring the performance on a held-out test set composed by one unseen subject. The drift here is much more gradual with respect to PMNIST. In fact, while each subject can perform an action (e.g., sitting down) differently from another, the basic mechanism is shared across all human beings. What will happen to the Sequence Length Issue in such context?

We propose two new benchmarks for the evaluation of Continual Human State Monitoring on time series data. Our benchmarks are based on two existing datasets for time-series classification: WESAD (Schmidt et al. 2018) and ASCERTAIN (Subramanian et al. 2018). Both datasets provide sequences of physiological data and each sequence is classified into 4 labels that denote the self-evaluated mental state of each subject. WESAD provides temperature and movement data, in addition to electromiogram (EMG) data and electrodermal (EDA) data, while ASCERTAIN provides electroencephalogram (EEG) and galvanic skin response (GSR) data. Both datasets provide heartbeat data via ECG measurements. Crucially for the design of our benchmarks, WESAD and ASCERTAIN are organized by subject. We used this property to build a non-stationary environment for CL. For both datasets, we created a stream in which each experience brings data coming from two subjects. The CL models we tested are trained sequentially on each experience of the stream. For each benchmark, we held out one subject to be used as test set. The task is to classify each time-series into one of the possible classes, which does not vary across experiences. We applied a pre-processing pipeline to WESAD by re-sampling each sequence at 32Hz and by normalizing mean and variance of each feature. Following WESAD documentation, we removed the labels 0 (neutral), 5, 6, 7 and regrouped each sequence into subsequences of 100 points per label. For each label, we kept 100 subsequences, leaving us with a training set of 4500 elements and a test set of 1500 elements. For ASCERTAIN, we ignored subjects 44 and 52 whose data quality is highlighted in the dataset as poor. For each remaining subject, we created the labels based on valence and arousal self-reported levels. Similar to WESAD, we re-sampled each sequence at 32Hz, we removed subjects with missing features and we built subsequences of 160 points. We ended up with a training set of 2160 elements and a test set of 720 elements. The ASCERTAIN dataset contains an over representation of the class 0: while the classes 1, 2 and 3 are represented each in around 500 sequences, the class 0 is represented in around 1100 sequences. To assess the impact of unbalanced classes in ASCERTAIN, we produced a custom version by artificially balancing it, removing sequences from over represented class. The pre-processing follows the one used for original ASCERTAIN. Table 11.3 summarizes the main characteristics of the three datasets.

11.3.1 Empirical Evaluation

We conducted an empirical evaluation to assess the behavior of different CL strategies applied to the proposed benchmarks. Our main objective is to understand to what extent training continuously on novel subjects affects the performance on the held-out test subject. That is, we aim to understand if and how much RNNs suffer from catastrophic forgetting in our proposed benchmarks. In order to enable full reproducibility of our results, we used the Avalanche library (Lomonaco, Pellegrini, et al. 2021) for all our experiments.

Table 11.4: Final average accuracy and standard deviation for each HSM dataset over 5 runs.

Strategy	WESAD	ASCERTAIN	Cust. ASCERTAIN
Offline	$99.60_{\pm 0.14}$	$30.39_{\pm 1.74}$	$39.09_{\pm 0.13}$
Naive	$72.41_{\pm 4.50}$	$26.78_{\pm 2.47}$	$32.77_{\pm 7.01}$
Cumulative	$83.62_{\pm 10.47}$	$29.59_{\pm 2.38}$	$35.62_{\pm 4.51}$
Replay	$80.79_{\pm 7.95}$	$26.75_{\pm 2.18}$	$33.07_{\pm 4.59}$
EWC	$71.79_{\pm 4.74}$	$26.66_{\pm 2.17}$	$34.46_{\pm 6.12}$
LwF	$72.75_{\pm 3.82}$	$27.62_{\pm 2.36}$	$33.03_{\pm 6.12}$

Experiments setup We tested three different CL strategies: Replay, EWC (Kirkpatrick et al. 2017) and LwF (Z. Li and Hoiem 2016). We also provide the results for Naive, Cumulative and offline Joint Training. Cumulative is a special case of Replay in which all previous data is used at each experience. For both datasets, we used an RNN composed by 2 layers of 18 GRU units (a RNN similar to LSTM). We conducted model selection on the offline training and the best resulting model has been used to train all the CL strategies used in our empirical evaluation. The hyperparameters involved in the model selection were the learning rate, the β_1, β_2 of the Adam optimizer and the L1L2 regularizer hyperparameter. The Replay memory size has been set at 25% of the training set size, which is 70 patterns for WESAD and 25 patterns for ASCERTAIN. For each experiment, we monitored the average accuracy and computed mean and standard deviation over 5 runs.

Results Table 11.4 reports the ACC metric, while Figure 11.6 shows the average accuracy for each benchmark after training on each experience. The WESAD benchmark is effectively tackled by RNNs. Both Cumulative and Replay strategies are able to accumulate knowledge over time, as the model is trained on more experiences. The gap with offline performance is still present, although it is smaller than the one achieved by the other strategies. There is no clear advantage in the use of regularization strategies, like LwF and EWC, with respect to a Naive finetuning approach. While this may be surprising in a traditional CI setting, in our DI benchmark the deterioration of performance due to catastrophic forgetting is much less severe. Both ASCERTAIN and its custom, balanced version represent a more difficult task with respect to WESAD. The offline performance is low, highlighting the fact that RNNs struggle to properly learn this kind of task. Balancing the dataset, as in Custom ASCERTAIN, improves the offline accuracy of around 9 points. The CL strategies still show a gap with respect to offline Joint Training. However, in the ASCERTAIN cases, the application of Replay does not contribute to improve the final performance compared to the other strategies. This is strongly related to the low accuracy achieved during offline training.

11.3.2 Discussion

Our experiments show that RNNs do not suffer from catastrophic forgetting in any of our benchmarks for Human State Monitoring. This is mainly due to the DI scenario defined by the benchmarks. Especially in the case of time-series, it is likely that the stream of incoming data will not always introduce new classes, but rather new instances of the same task that was required to solve from the beginning. Our benchmarks fit this case and highlight that, in order to model real-world applications, it is in some cases best to focus on knowledge

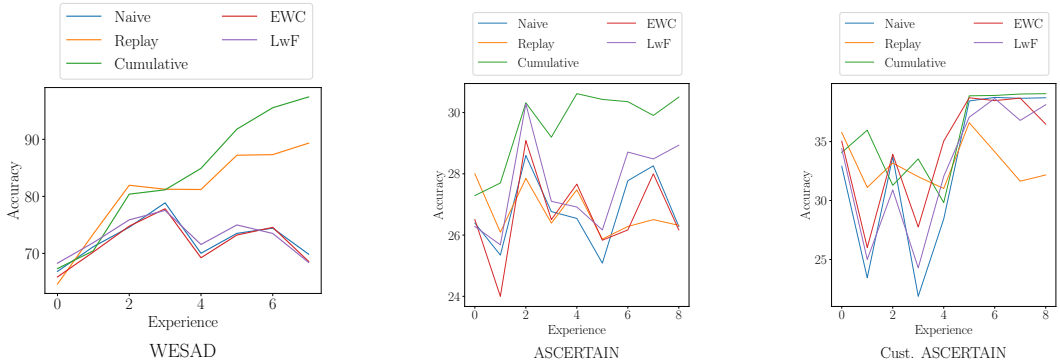


Figure 11.6: Accuracy on each HSM dataset measured on the held-out test set after training on each experience. Plots taken from Matteoni et al. 2022.

accumulation and forward transfer rather than forgetting. The Sequence Length Issue in this case can be safely overlooked. As showed by Figure 11.6, regularization strategies like LwF and EWC are not capable of accumulating knowledge, since they are mostly designed to improve model stability against forgetting. To this end, Replay is the most effective strategy with respect to the alternatives. However, storing previous data may not be possible in all applications due to data-privacy concerns or other constraints. This highlights the need to develop novel CL strategies which are capable of rapidly exploiting new information to improve on the current task.

Chapter 12

Conclusion

We concluded our journey across the wide space of CL environments. We collected new insights and intuitions which make us better understand what learning continuously means. In Chapter 1, we started from a key observation: the current evaluation system in CL is biased towards a very specific set of environments and data streams. In the main Chapters of this thesis, we proceeded to outline a set of novel scenarios designed out of real-world needs, which so far had not received much attention from the literature.
The Class-Incremental with Repetition scenario (Chapter 6) focuses on the concept of repetition, which are embedded within the data stream. We provided two different CIR generators and we released them to facilitate their use. The generators are flexible and allow to create data streams with a variable amount of repetition. We discovered that, when the repetition is unbalanced, Replay strategies can be highly improved. This is due to the fact that Replay has been designed to work in Class-Incremental scenarios and does not usually consider alternative settings. Our Frequency-Aware Replay, instead, is able to take into account a scenario with unbalanced repetitions: unlike other Replay variants, it consolidates over time the knowledge about the most infrequent classes. It performs on par with the other variants on the frequent ones. Exploiting the properties of novel scenarios and data streams is an important point for the future design of CL strategies.
We also studied the Continual Pre-Training scenario (Chapter 7), where a pre-trained model is kept updated over time via a stream of pre-training tasks. Continual Pre-Training is useful whenever a pre-trained model is available and we wish to inject new knowledge in it when needed. In the case of language models, for example, injecting new knowledge is important to avoid predictions based on out-dated information and to make the model aware of new facts happened since the last pre-training phase. By studying the scenario, we discovered that when continual pre-training is performed in an unsupervised way, the model is not subjected to forgetting of previous knowledge. In this scenario, the evaluation is conducted with an additional step of fine-tuning on a set of downstream tasks. The continually pre-trained model is able to improve its performance on tasks related to the newly injected information without deteriorating the performance on tasks associated to knowledge previously acquired. Therefore, Continual Pre-Training provides an elegant solution to the problem of forgetting in pre-trained model, at the cost of an additional training phase on the target task. Fortunately, fine-tuning can be rather quick. In our experiments, one epoch of training was enough to reach a strong performance.
Finally, we reviewed (Chapter 9) and studied CL for temporally correlated data, like time-series. We focus on a family of ML models, the Recurrent Neural Networks, which are designed to model temporal dependence. Given the widespread usage of time-series in ML applications, it is of fundamental importance to understand whether this domain has different

characteristics than the ones studied in CL (e.g., Computer Vision). Our empirical evaluation (Chapter 10) on a number of CL strategies discovered the Sequence Length Issue: a RNN will suffer more forgetting when dealing with longer sequences. To overcome the issue, we experimented with two main approaches (Chapter 11). The Gated Incremental Memories are recurrent models which, by partitioning the architecture in separate modules, are able to effectively learn sequences of different lengths. However, they strongly depends on the performance of the recurrent autoencoder choosing which modules to use at inference time. The other approach we leveraged is based on Echo State Networks, a form of randomized recurrent models. The ESNs keep a fixed recurrent component (the reservoir) which is immune to the Sequence Length Issue, since there is no update flowing through it. The reservoir can also be considered as a fixed feature extractor in order to leverage ad-hoc CL strategies which are otherwise not suitable for recurrent models. Finally, we focused on a Continual Human State Monitoring application from time-series data. We showed that, when the drift in the stream is mild, the Sequence Length Issue does not need to be tackled directly. However, many CL approaches fail to accumulate knowledge over time and they are not able to exploit information coming from new subjects to improve their performance on a different, held-out subject.

This thesis has shown that the objectives and the challenges of CL vary much across different environments. Class-Incremental with Repetition and Continual Pre-Training represent two examples that clearly show the need to focus not only on what a CL agent does, but also on *what* a CL agent "sees". The blind application of any approach may turn out to be hurtful for the agent, depending on the environment it lives in. The design of novel CL scenarios that answer to specific needs is a promising source of new questions and findings. It helps to grasp what are the objectives of a CL agent and it contributes to a more complete understanding of how an agent can learn continuously.

12.1 Future Research Directions

We have not yet unveiled all the opportunities stemming from the study of new scenarios and data streams.
In Chapter 6, we introduced the concept of repetition at the class level by revisiting new and previous instances of already encountered classes. Importantly, the drift introduced by the CIR scenarios we studied in this thesis is completely arbitrary. It is impossible to predict in advance where a drift is going to occur or which kind of drift it will be. This choice has been useful to study the role of repetition alone, by modifying as little as possible the Class-Incremental scenario. However, it also prevents to fully exploit the information carried by repetition. The fact that an agent is revisiting previous concepts can be a strong suggestion that the context around the agent is changing, possibly going back to a state similar to previous ones. The agent is not unexpectedly brought into a different environment, but rather it is gradually introduced to it. This may allow novel CL strategies to anticipate what is coming next and to predict which skills and abilities will be needed in the near future. One advantage of CIR is its flexibility, and the possibility to configure streams of this type. For example, it would be interesting to study a stream with gradual drifts, where new concepts slowly intermix with the current ones before taking over completely (and being overturned by others later on). This type of drift is neglected by CL, and it is sometimes considered by the online and streaming learning communities. A scenario with gradual drifts may also be particularly interesting when studied in the context of sequential data. Learning a pattern of temporal correlation naturally allows to predict what is coming next (sequence

modeling). This knowledge can in turn be exploited when there is a mismatch between what should have happened and what actually happened. When the drift is unpredictable, the prediction error is instantaneous and it does not carry useful information. All the information is determined by the input, already received, which caused the drift. In the case of gradual drift instead, a temporal-aware agent like a RNN can anticipate the drift as soon as its prediction error *starts* to increase. That is, when the drift is still occurring. In line with the predictive coding theory, when an agent expects to see something but see something else, its attention is triggered. One immediate advantage for CL is that a system of the type described should not need any external knowledge about boundaries between experiences. The model itself would be able to detect, as part of its learning protocol, drifting distributions.

Continual Pre-Training represents a novel CL scenario which opens many research questions. While image classification tasks, like NLP, benefit from self-supervised pre-training, we should explore whether this behavior is present also in other Computer Vision tasks. Continual Learning has started to tackle semantic segmentation and object detection tasks. What is the impact of continual self-supervised pre-training on these environments? Are Convolutional Networks able to exploit the benefits of self-supervised pre-training, or are Vision Transformers the most effective option?
Continual Pre-Training requires a fine-tuning step in order to adapt the pre-trained model to the down-stream tasks. This remains a disadvantage with respect to a model which learns continuously directly on the stream of tasks. However, there are hints suggesting that continual pre-training may be enough in many situations and may not require additional fine-tuning. Large pre-trained models are capable of solving down-stream tasks without being fine-tuned on the datasets coming from those tasks (zero-shot adaptation). However, zero-shot performance depends on a very large pre-training phase, in order to incorporate enough knowledge to solve a wide variety of tasks. Continual Pre-Training could ease this constraint, since it allows to learn representations over time. Suppose that the pre-trained model is not able to tackle a new incoming task zero-shot. The only option currently available is to fine-tune the model by using part of the labelled dataset coming from the task. However, Continual Pre-Training may leverage unlabeled corpus to update the internal knowledge of the model without the need to rely on external supervision. The additional pre-training step may suffice to perform zero-shot adaptation on the down-stream tasks. This is also suggested by our results, showing that the fine-tuning phase can be actually rather quick (1 epoch was enough). Continual Pre-Training can be leveraged whenever a model is powerful enough to learn effective internal representations. The scenario provides a strong learning paradigm that allows to keep the model updated over time and to make it able to tackle a wide variety of down-stream tasks with minimal or no adaptation at all.

The study conducted in this thesis contributed to highlight that forgetting, so far at the center of CL research, happens at very different degrees and it is not always the most important challenge to consider. Looking at every CL problem through the lenses of forgetting risks to depict a distorted view of the real phenomena at play. For example, we have seen that in a scenario like Continual Human State Monitoring, forgetting is already limited by the properties of the data stream. On the contrary, all the CL strategies struggle to accumulate knowledge over time. In that specific case, acquiring more information on the state of more subjects did not contribute much to generalize to a new, unseen subject. So far, the vast majority of the existing approaches are focused on the mitigation of forgetting and they overlook other aspects of learning continuously. Aspects which, in some scenarios like Class-Incremental, are often hidden by the large amount of forgetting. We believe that

both Class-Incremental with Repetition and Continual Pre-Training represent interesting scenarios to study additional objectives, like Forward Transfer and knowledge accumulation. A continual learner able to accumulate knowledge over time can be as effective as one which does not forget previous knowledge. Importantly, a model which does not forget is not necessarily pushed to actually *use* the knowledge it is able to remember. On the contrary, a model capable of Forward Transfer is, by definition, able to exploit what it already knows to learn more effectively new tasks. In order to do that, the model needs to remember previous knowledge. Admittedly, it can also forget unnecessary one. However, this is more of an advantage than a disadvantage. We hope that future studies along these lines will lead to CL agent that does not forget simply because they are required to leverage what they already know.

Appendix A

List of Publications

Carta, A., Pellegrini, L., **Cossu, A.**, Hemati, H. and Lomonaco, L. Avalanche: A PyTorch Library for Deep Continual Learning. under review (2023)

Carta, A., **Cossu, A.**, Lomonaco, L., Bacciu, D. and van de Weijer, J. Projected Latent Distillation for Data-Agnostic Consolidation in Distributed Continual Learning. under review (2023)

Cossu, A., Carta, A., Passaro, L., Lomonaco, V., Tuytelaars, T. and Bacciu, D. Continual Pre-Training Mitigates Forgetting in Language and Vision. under review (2022)

Hemati, H., **Cossu, A.**, Hurtado, J., Graffieti, G., Pellegrini, L., Carta, A., Lomonaco, V. and Borth, D. Class-Incremental Learning with Repetition. 3rd Continual Learning in Computer Vision Workshop (CLVISION) at CVPR, (2022)

Sangermano, M., Carta, A., **Cossu, A.** and Bacciu, D. Sample Condensation in Online Continual Learning. In 2022 International Joint Conference on Neural Networks (IJCNN) (2022)

Carta, A., **Cossu, A.**, Errica, F. and Bacciu, D. Catastrophic Forgetting in Deep Graph Networks: A Graph Classification Benchmark. Frontiers in Artificial Intelligence 5, (2022)

Matteoni, F., **Cossu, A.**, Gallicchio, C., Lomonaco, V., and Bacciu D. Continual Learning for Human State Monitoring. In European Symposium on Artificial Neural Networks (ESANN), (2022).

Cossu, A., Graffieti, G., Pellegrini, L., Maltoni, D., Bacciu, D., Carta, A. and Lomonaco, V. Is Class-Incremental Enough for Continual Learning? Frontiers in Artificial Intelligence 5, (2022)

Carta, A., **Cossu, A.**, Lomonaco, V. and Bacciu, D. Ex-Model: Continual Learning from a Stream of Trained Models. in 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (IEEE, 2021)

Cossu, A., Bacciu, D., Carta, A., Gallicchio, C. and Lomonaco, V. Continual Learning with Echo State Networks. in 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (2021)

Lomonaco, V., Pellegrini, L., **Cossu, A.**, Carta, A., Graffieti, G., Hayes, T., De Lange, M., Masana, M., Pomponi, J., van de Ven, G., Mundt, M., She, Q., Cooper, K., Forest, J., Belouadah, E., Calderara S., Parisi, G., Cuzzolin, F., Tolias, A., Scardapane, S., Antiga, L., Ahmad, S., Popescu, A., Kanan, C., van de Weijer, J., Tuytelaars, T., Bacciu, D. and Maltoni, D. Avalanche: an End-to-End Library for Continual Learning. in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) 3595–3605 (IEEE, 2021)

Cossu, A., Carta, A., Lomonaco, V. and Bacciu, D. Continual learning for recurrent neural networks: An empirical evaluation. Neural Networks 143, 607–627 (2021)

Rosasco, A., Carta, A., **Cossu, A.**, Lomonaco, V., and Bacciu, D. Distilled Replay: Overcoming Forgetting Through Synthetic Samples. In Continual Semi-Supervised Learning Lecture Notes in Computer Science, Springer International Publishing), pp. 104–117.

Cossu, A., Ziosi, M. and Lomonaco, V. Sustainable Artificial Intelligence through Continual Learning. In EAI (2021)

Merlin, G., Lomonaco, V., **Cossu, A.**, Carta, A. and Bacciu, D. Practical Recommendations for Replay-based Continual Learning Methods. Workshop on Novel Benchmarks and Approaches for Real-World Continual Learning (CL4REAL) (2021)

Cossu, A., Carta, A. and Bacciu, D. Continual Learning with Gated Incremental Memories for sequential data processing. in 2020 International Joint Conference on Neural Networks (IJCNN) 1–8 (2020)

Appendix B

Open-Source Software

During the development of this thesis, a number of open-source code releases have been produced. All the resources listed below have been entirely created by the work of the author of this thesis:

- GitHub repository for the experiments with the Continual Pre-Training scenario: `https://github.com/AndreaCossu/continual-pretraining-nlp-vision`.
- GitHub repository for the evaluation of Continual Learning strategies with Recurrent Neural Networks: `https://github.com/AndreaCossu/ContinualLearning_RecurrentNetworks`.
- GitHub repository for the experiments with Gated Incremental Memories: `https://github.com/AndreaCossu/ContinualLearning-SequentialProcessing`.
- GitHub repository for the experiments with Echo-State networks: `https://github.com/Pervasive-AI-Lab/ContinualLearning-EchoStateNetworks`.

The Avalanche library (Appendix C) is open-sourced at `https://github.com/ContinualAI/avalanche`. The author of this thesis is one of the contributors and maintainer of the repository.
The repository at `https://github.com/ContinualAI/continual-learning-baselines` provides a set of experiments to reproduce the performance of popular CL strategies, implemented in Avalanche. The author of this thesis is the main maintainer of the repository.

Appendix C

Avalanche: an End-to-End library for Continual Learning

Avalanche is an open-source (MIT licensed) end-to-end library for CL based on PyTorch (Paszke et al. 2019) which provides a shared and collaborative codebase for fast prototyping, training, and evaluation of CL algorithms. Even if still in its infancy, Avalanche has rapidly become the most used CL framework in research. Most of the experiments of this thesis are based on Avalanche. Avalanche is promoted and supported by ContinualAI[1], a non-profit research organization on CL[2].

Avalanche is organized into five main modules: **Benchmarks**, **Training**, **Evaluation**, **Models** and **Logging**. Fig. C.1 shows a representation of the library modules and their interplay within the aforementioned reference framework. The author of this thesis is currently the maintainer of the Evaluation module and one of the authors of the original paper.

Benchmarks The *benchmarks* module is designed with the idea of providing an extensive set of out-of-the-box loaders covering the most common benchmarks (i.e. Split CIFAR (Rebuffi et al. 2017), Permuted MNIST (I. J. Goodfellow et al. 2013), etc.) through the *classic* submodule. A simple example illustrating how to use the SMNIST benchmark (Zenke et al. 2017) is shown in Fig. C.2. Moreover, a wide range of tools are available that enables the creation of customized benchmarks. The benchmark preparation and data loading process can seamlessly handle memory-intensive benchmarks, such as Split-ImageNet (Rebuffi et al. 2017), without the need to load the whole dataset into memory in advance.
Further, the benchmarks module is entirely standalone, meaning that it can be used independently from the rest of Avalanche. As such, it represents a valid alternative to other libraries such as Continuum.
Starting from the higher-level API, Avalanche offers explicit support for creating benchmarks that fit one of the ready-to-use scenarios. If the benchmark to be implemented fits either in the *New Instances* or *New Classes* scenarios, one can consider using one of the specific generators `nc_scenario` or `ni_scenario`. Most classic benchmarks are based on these generators.
If the benchmark does not fit into a predefined scenario, a *generic generator* can be used. At the moment, Avalanche allows users to create benchmark instances from lists of files, Caffe-style filelists (Jia et al. 2014), lists of PyTorch datasets, or even directly from Tensors.

[1] `https://www.continualai.org/`

[2] Disclaimer: the author of this thesis is, at the time of writing, Board Member and Treasurer of ContinualAI.

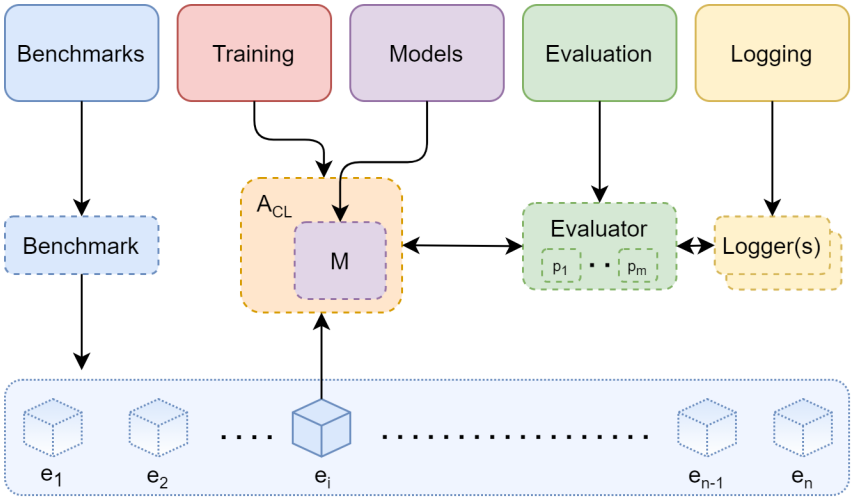


Figure C.1: Representation of Avalanche with its main modules (top), the main object instances (middle) and the generated stream of data (bottom). A *Benchmark* generates a stream of experiences e_i which are sequentially accessible by the CL algorithm A_{CL} with its internal model M. The *Evaluator* object directly interacting with the algorithm provides a unified interface to control and compute several performance metrics (p_i), delegating results logging to the *Logger(s)* objects. Figure taken from Lomonaco, Pellegrini, et al. 2021.

Classic Benchmarks

```
benchmark_instance = SplitMNIST(
    n_experiences=5, seed=1)
    # Other useful parameters
    # return_task_id=False/True
    # fixed_class_order=[5, 0, 9, ...]
```

Figure C.2: Simple instantiation of a *Classic* CL benchmark.

Avalanche also offers the possibility to define multiple streams for the same scenario. Therefore, one can design a scenario with a train stream, a test stream, a validation stream and/or any other stream that can be used by the model. This allows to implement many evaluation protocols based on customized streams.

Training The `training` module implements both popular CL strategies and a set of abstractions that make it easy for a user to implement custom CL algorithms. Each strategy in Avalanche implements a method for training (`train`) and a method for evaluation (`eval`), which can work either on single experiences or on entire slices of the data stream. Currently, Avalanche strategies can be readily used to train baselines in a few lines of code, as shown in Fig. C.3.

Avalanche `training` module is built on a simple callback mechanism. This is used by strategies, metrics, and loggers to interact with the training loop and execute their code at the correct points using a simple interface and provides an easy interface to implement new strategies by adding custom code to the generic loops. This also allows for multiple strategies to be combined together. For example, we can define a hybrid strategy that uses

Training Strategies

```
strategy = Replay(model, optimizer,
                  criterion, mem_size)
for train_exp in scenario.train_stream:
    strategy.train(train_exp)
    strategy.eval(scenario.test_stream)
```

Figure C.3: Simple instantiation of an already available strategy in Avalanche.

Elastic Weight Consolidation (EWC) (Kirkpatrick et al. 2017) and Replay using the snippet of code shown in Fig. C.4.

Hybrid Strategies

```
replay = ReplayPlugin(mem_size)
ewc = EWCPlugin(ewc_lambda)
strategy = Naive(
    model, optimizer,
    criterion, mem_size,
    plugins=[replay, ewc])
```

Figure C.4: Example of an on-the-fly instantiation of hybrid strategies through Plugins.

Evaluation The performance of a CL algorithm should be assessed by monitoring multiple aspects of the computation (Section 4.2). The `evaluation` module offers a wide set of metrics to evaluate experiments.
Avalanche's design principle is to separate the issues of *what to monitor* and *how to monitor* it: the `evaluation` module provides support for the former through the metrics, while the `logging` module fulfills the latter (Section C). Metrics do not specify in which format their output should be displayed, while loggers do not alter metrics logic. Metrics can work even without a logger: the strategy's *train* and *eval* methods return a dictionary with all the metrics logged during the experiment.
Few lines of code are sufficient to monitor a vast set of metrics: *accuracy*, *loss*, *catastrophic forgetting*, *confusion matrix*, *timing*, *ram/disk/CPU/GPU usage*. The `EvaluationPlugin` is the component responsible for the orchestration of both plugin metrics and loggers. Its role is to gather metrics outputs and forward them to the loggers during training and evaluation loops. All the user has to do to keep track of an experiment is to provide the strategy object with an instance of the `EvaluationPlugin` with the target metrics and loggers as parameters. Fig. C.5 shows how to use the evaluation plugin and metric helper functions.

Logging Nowadays, logging facilities are fundamental to monitor in real time the behavior of an ongoing experiment (which may last from minutes to days). The Avalanche `logging` module is in charge of displaying to the user the result of each plugin metric during the different experiment phases. Avalanche provides three different loggers: `TextLogger`, `InteractiveLogger` and `TensorboardLogger` (Martin Abadi et al. 2015). They provide reports on textual file, standard output and Tensorboard, respectively. As soon as a metric

Evaluation Plugin

```
eval_plugin = EvaluationPlugin(
    accuracy_metrics(experience=True),
    loss_metrics(minibatch=True, stream=True),
    forgetting_metrics(experience=True),
    timing_metrics(minibatch=True),
    gpu_usage_metrics(gpu_id, epoch=True),
    loggers=[InteractiveLogger(),
             TextLogger(open('out.txt', 'w')),
             TensorboardLogger()])
```

Figure C.5: Avalanche evaluation plugin (or *evaluator*) object instantiation example.

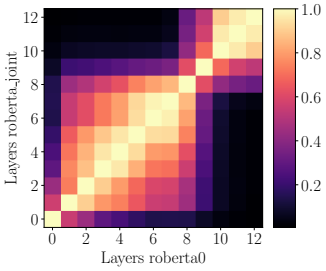
emits a value, the Text Logger prints the complete metric name followed by its value in the destination file. The `InteractiveLogger` reports the same output as `TextLogger`, but it also uses the *tqdm* package[3] to display a progress bar during training and evaluation. `TensorboardLogger` is able to show images and more complex outputs, which cannot be appropriately printed on standard output or textual file.
Integrating loggers into both training and evaluation loops is straightforward. Once created, loggers have to be passed to the `EvaluationPlugin`, which will be in charge of redirecting metrics outputs to each logger during the experiment. See Fig. C.5 for an example of loggers creation.

Models The Avalanche `models` module offers a set of simple machine learning architectures ready to be used in experiments. For example, Avalanche provides `DynamicModule` which can automatically adapt during training. The multi-head mechanism is already provided and fully integrated in the Avalanche training and evaluation loop. The main purpose of these models is to let the user focus on Avalanche features, rather than on writing lines of code to build a specific architecture.

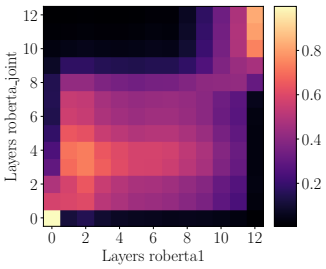
[3] `https://tqdm.github.io`

Appendix D

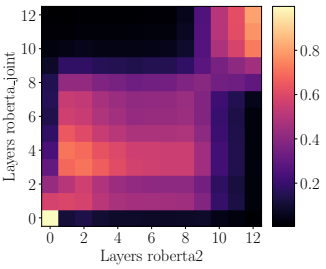
Full Set of CKA Results for Continual Pre-Training Scenario



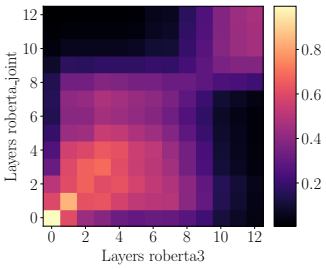
(a) RoBERTa QNLI 1



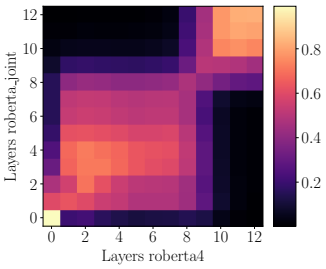
(b) RoBERTa QNLI 2



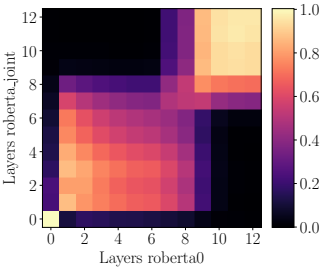
(c) RoBERTa QNLI 3



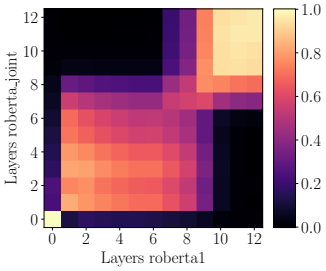
(d) RoBERTa QNLI 4



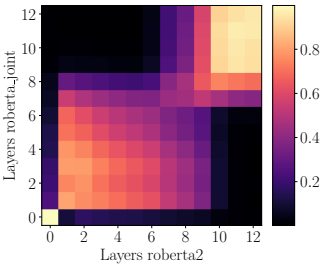
(e) RoBERTa QNLI 5



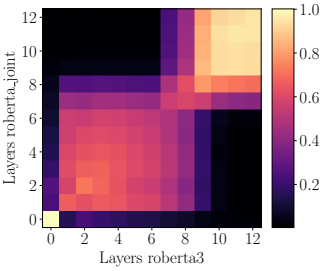
(f) RoBERTa Tweets 1



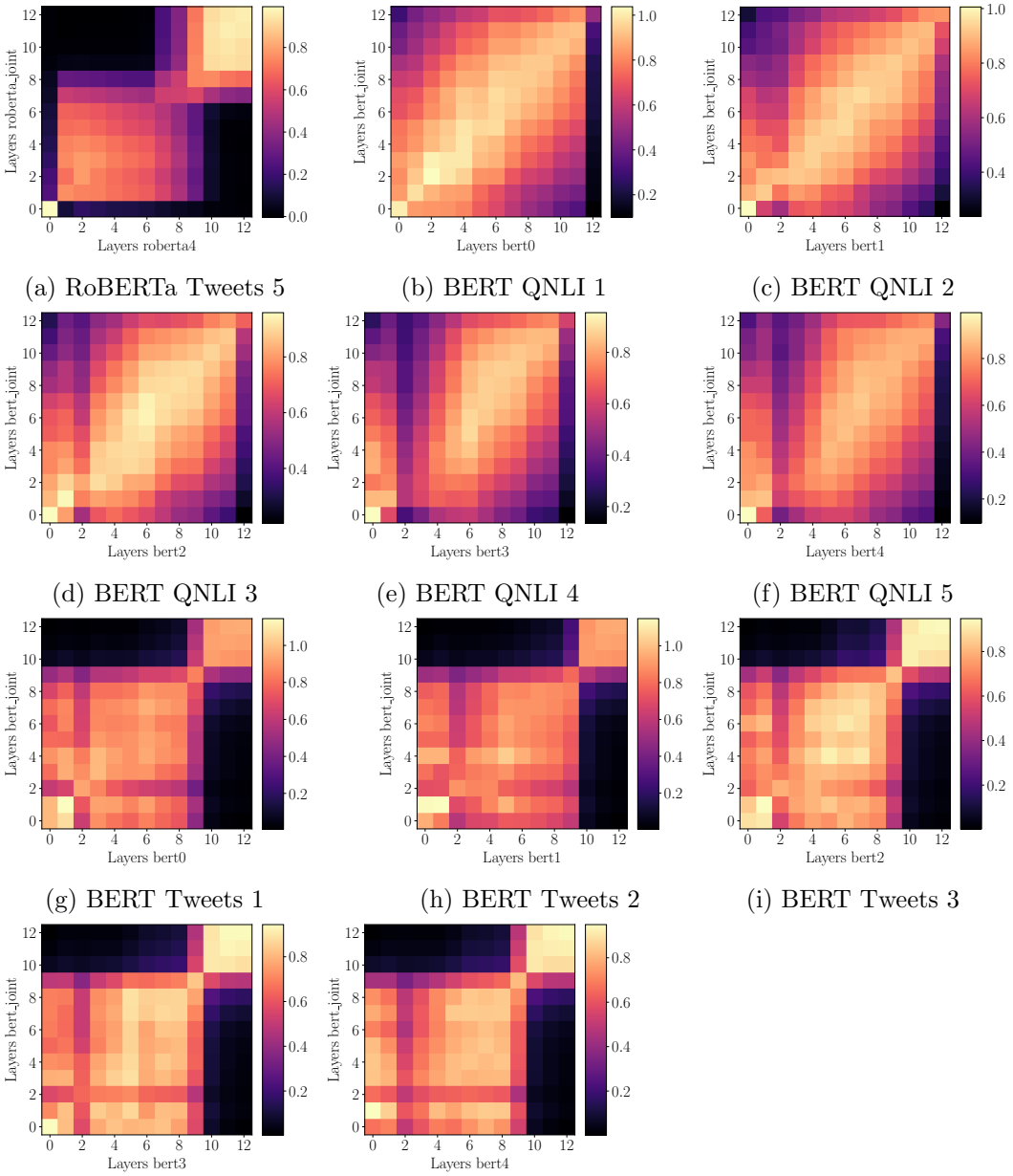
(g) RoBERTa Tweets 2



(h) RoBERTa Tweets 3



(i) RoBERTa Tweets 4

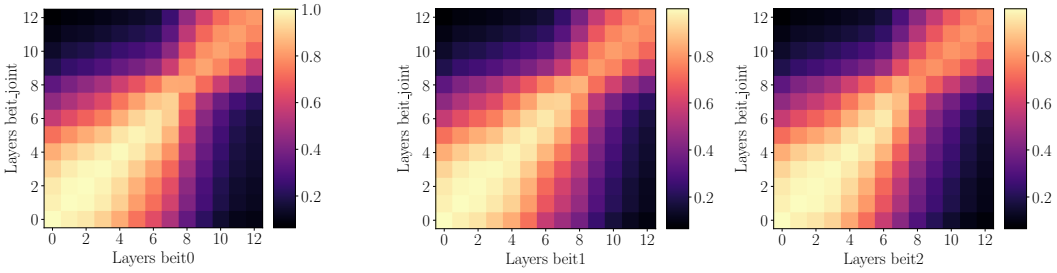


(a) RoBERTa Tweets 5 (b) BERT QNLI 1 (c) BERT QNLI 2

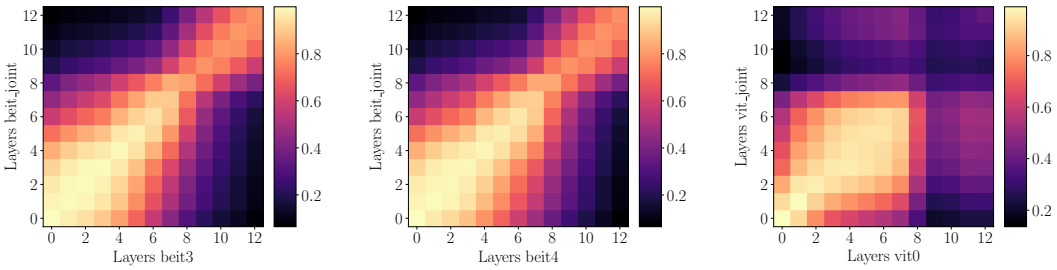
(d) BERT QNLI 3 (e) BERT QNLI 4 (f) BERT QNLI 5

(g) BERT Tweets 1 (h) BERT Tweets 2 (i) BERT Tweets 3

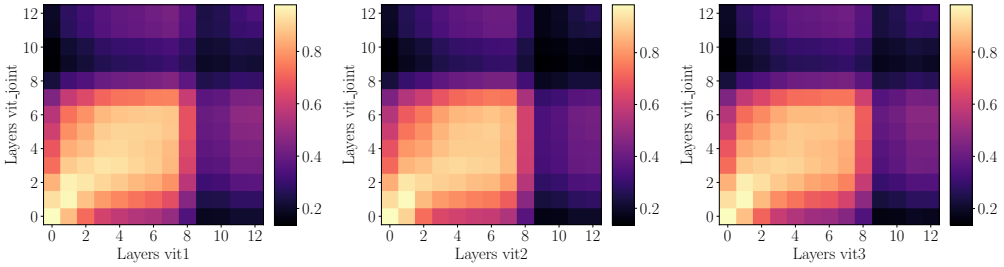
(j) BERT Tweets 4 (k) BERT Tweets 5



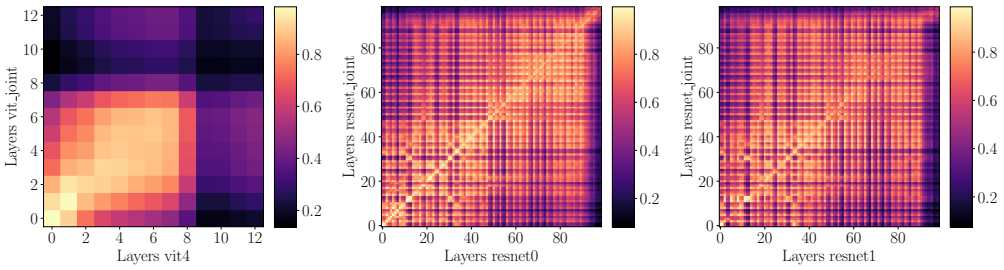
(a) BEiT 1 (b) BEiT 2 (c) BEiT 3



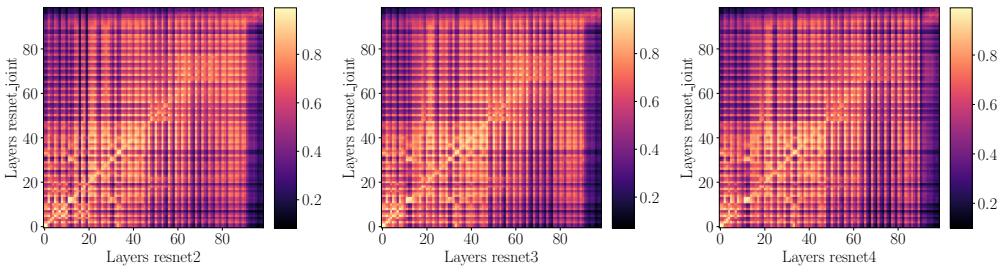
(d) BEiT 4 (e) BEiT 5 (f) ViT 1



(g) ViT 2 (h) ViT 3 (i) ViT 4



(j) ViT 5 (k) ResNet 1 (l) ResNet 2



(m) ResNet 3 (n) ResNet 4 (o) ResNet 5

Bibliography

Alippi, Cesare, Giacomo Boracchi, and Manuel Roveri (2013). "Just-In-Time Classifiers for Recurrent Concepts". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.4 (cit. on p. 27).

Aljundi, Rahaf, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars (2018). "Memory Aware Synapses: Learning What (Not) to Forget". In: *The European Conference on Computer Vision (ECCV)* (cit. on p. 97).

Aljundi, Rahaf, Punarjay Chakravarty, and Tinne Tuytelaars (2017). "Expert Gate: Lifelong Learning with a Network of Experts". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 48).

Aljundi, Rahaf, Klaas Kelchtermans, and Tinne Tuytelaars (June 2019). "Task-Free Continual Learning". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 32, 36).

Aljundi, Rahaf, Min Lin, Baptiste Goujaud, and Yoshua Bengio (2019). "Gradient Based Sample Selection for Online Continual Learning". In: *Advances in Neural Information Processing Systems 32*. Ed. by H Wallach, H Larochelle, A Beygelzimer, F dAlche-Buc, E Fox, and R Garnett. Curran Associates, Inc., pp. 11816–11825 (cit. on p. 36).

Aljundi, Rahaf, Marcus Rohrbach, and Tinne Tuytelaars (2019). "Selfless Sequential Learning". In: *ICLR* (cit. on p. 46).

Ans, Bernard, Stephane Rousset, Robert M. French, and Serban C. Musca (2002). "Preventing Catastrophic Interference in MultipleSequence Learning Using Coupled Reverberating Elman Networks". In: *Proceedings of the 24th Annual Conference of the Cognitive Science Society* (cit. on p. 90).

Ans, Bernard, Stéphane Rousset, Robert M. French, and Serban Musca (2004). "Self-Refreshing Memory in Artificial Neural Networks: Learning Temporal Sequences without Catastrophic Forgetting". In: *Connection Science* 16.2 (cit. on p. 90).

Asghar, Nabiha, Lili Mou, Kira A Selby, Kevin D Pantasdo, Pascal Poupart, and Xin Jiang (2019). "Progressive Memory Banks for Incremental Domain Adaptation". In: *International Conference on Learning Representations* (cit. on pp. 90, 91, 93, 94).

Bacciu, Davide, Antonio Carta, and Alessandro Sperduti (2019). "Linear Memory Networks". In: *Proceedings of the 28th International Conference on Artificial Neural Networks (ICANN 2019),* Lecture Notes in Computer Science. Springer-Verlag (cit. on p. 18).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ICLR* (cit. on p. 91).

Bao, Hangbo, Li Dong, Songhao Piao, and Furu Wei (2021). "BEiT: BERT Pre-Training of Image Transformers". In: *International Conference on Learning Representations* (cit. on pp. 10, 21, 75, 79).

Belouadah, Eden and Adrian Popescu (2019). "Il2m: Class incremental learning with dual memory". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 583–592 (cit. on p. 57).

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning Long-Term Dependencies with Gradient Descent Is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (cit. on p. 18).

Biesialska, Magdalena, Katarzyna Biesialska, and Marta R. Costa-jussà (2020). "Continual Lifelong Learning in Natural Language Processing: A Survey". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 6523–6541 (cit. on p. 91).

Bifet, Albert and Ricard GavaldÃ (Apr. 26, 2007). "Learning from Time-Changing Data with Adaptive Windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*. Proceedings. Society for Industrial and Applied Mathematics, pp. 443–448 (cit. on p. 27).

Bojar, Ondřej, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi (2017). "Findings of the 2017 Conference on Machine Translation (WMT17)". In: *Proceedings of the Second Conference on Machine Translation*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 169–214 (cit. on p. 93).

Buchner, Johannes (2017). "Synthetic Speech Commands: A Public Dataset for Single-Word Speech Recognition." In: *Kaggle Dataset* (cit. on pp. 95, 96).

Buzzega, Pietro, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara (2020). "Dark Experience for General Continual Learning: A Strong, Simple Baseline". In: *arXiv:2004.07211 [cs, stat]* (cit. on p. 45).

Carpenter, Gail and Stephen Grossberg (1986). "Adaptive Resonance Theory: Stable Self-Organization of Neural Recognition Codes in Response to Arbitrary Lists of Input Patterns". In: *Proceedings of the Eight Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, pp. 45–62 (cit. on p. 29).

Carta, Antonio, Andrea Cossu, Federico Errica, and Davide Bacciu (2022). "Catastrophic Forgetting in Deep Graph Networks: A Graph Classification Benchmark". In: *Frontiers in Artificial Intelligence* 5 (cit. on p. 40).

Caruana, Rich (1997). "Multitask Learning". In: *Machine Learning* 28.1 (cit. on p. 32).

Castro, Francisco M, Manuel J Marin-Jimenez, Nicolas Guil, Cordelia Schmid, and Karteek Alahari (2018). "End-to-end incremental learning". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 233–248 (cit. on p. 57).

Chaudhry, Arslan, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr (2018). "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 532–547 (cit. on pp. 43, 45).

Chaudhry, Arslan, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny (2019). "Efficient Lifelong Learning with A-GEM". In: *ICLR* (cit. on pp. 47, 57, 62, 96, 97).

Chaudhry, Arslan, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip H S Torr, and Marc'Aurelio Ranzato (2019). "On Tiny Episodic Memories in Continual Learning". In: *arXiv* (cit. on pp. 41, 50).

Chen, Tianlong, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang (2020). "Long Live the Lottery: The Existence of Winning Tickets in Lifelong Learning". In: *International Conference on Learning Representations* (cit. on p. 21).

Chen, Xinlei, Haoqi Fan, Ross Girshick, and Kaiming He (2020). "Improved Baselines with Momentum Contrastive Learning". In: *arXiv:2003.04297 [cs]* (cit. on p. 82).

Conneau, Alexis and Douwe Kiela (May 7-12, 2018 2018). "SentEval: An Evaluation Toolkit for Universal Sentence Representations". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Ed. by Nicoletta Calzolari (Conference chair), Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga. Miyazaki, Japan: European Language Resources Association (ELRA) (cit. on pp. 74, 78).

Coop, Robert and Itamar Arel (2012). "Mitigation of Catastrophic Interference in Neural Networks Using a Fixed Expansion Layer". In: *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, pp. 726–729 (cit. on p. 91).

– (2013). "Mitigation of Catastrophic Forgetting in Recurrent Neural Networks Using a Fixed Expansion Layer". In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. Dallas, TX, USA: IEEE, pp. 1–7 (cit. on pp. 90, 91, 93).

Cossu, Andrea, Davide Bacciu, Antonio Carta, Claudio Gallicchio, and Vincenzo Lomonaco (2021). "Continual Learning with Echo State Networks". In: *29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (cit. on pp. 6, 7, 101).

Cossu, Andrea, Antonio Carta, and Davide Bacciu (2020). "Continual Learning with Gated Incremental Memories for Sequential Data Processing". In: *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN 2020)* (cit. on pp. 6, 90, 92–94, 101, 102, 106, 108).

Cossu, Andrea, Antonio Carta, Vincenzo Lomonaco, and Davide Bacciu (2021). "Continual Learning for Recurrent Neural Networks: An Empirical Evaluation". In: *Neural Networks* 143 (cit. on pp. 6, 57, 89, 90, 93, 95, 97, 100).

Cossu, Andrea, Gabriele Graffieti, Lorenzo Pellegrini, Davide Maltoni, Davide Bacciu, Antonio Carta, and Vincenzo Lomonaco (2022). "Is Class-Incremental Enough for Continual Learning?" In: *Frontiers in Artificial Intelligence* 5 (cit. on p. 56).

Cossu, Andrea, Tinne Tuytelaars, Antonio Carta, Lucia Passaro, Vincenzo Lomonaco, and Davide Bacciu (2022). "Continual Pre-Training Mitigates Forgetting in Language and Vision". In: (cit. on pp. 5, 70).

Cui, Yuwei, Subutai Ahmad, and Jeff Hawkins (2016). "Continuous Online Sequence Learning with an Unsupervised Neural Network Model". In: *Neural Computation* 28.11 (cit. on pp. 90, 91).

Davari, MohammadReza, Nader Asadi, Sudhir Mudur, Rahaf Aljundi, and Eugene Belilovsky (2022). "Probing Representation Forgetting in Supervised and Unsupervised Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (cit. on p. 80).

de Jong, Edwin D. (2016). "Incremental Sequence Learning". en. In: *arXiv: 1611.03068 [cs]* (cit. on p. 93).

De Lange, Matthias, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars (2021). "A Continual Learning Survey: Defying Forgetting in Classification Tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (cit. on pp. 35, 38, 43, 48, 51, 87).

De Lange, Matthias and Tinne Tuytelaars (2021). "Continual Prototype Evolution: Learning Online from Non-Stationary Data Streams". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8250–8259 (cit. on p. 36).

Dettmers, Tim and Luke Zettlemoyer (2019). "Sparse Networks from Scratch: Faster Training without Losing Performance". In: *arXiv:1907.04840 [cs, stat]* (cit. on p. 30).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (May 2019a). "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [Cs]* (cit. on p. 10).

– (2019b). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186 (cit. on pp. 10, 21, 74).

Dhar, Prithviraj, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa (2019). "Learning without Memorizing". In: *CVPR* (cit. on p. 45).

Díaz-Rodríguez, Natalia, Vincenzo Lomonaco, David Filliat, and Davide Maltoni (2018). "Don't Forget, There Is More than Forgetting: New Metrics for Continual Learning". In: *arXiv* (cit. on pp. 32, 43, 57, 58).

Ditzler, Gregory, Manuel Roveri, Cesare Alippi, and Robi Polikar (2015). "Learning in Nonstationary Environments: A Survey". In: *IEEE Computational Intelligence Magazine* 10.4 (cit. on pp. 24–27, 31).

Domingos, Pedro and Geoff Hulten (Aug. 1, 2000). "Mining High-Speed Data Streams". In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. New York, NY, USA: Association for Computing Machinery, pp. 71–80 (cit. on p. 28).

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2020). "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations* (cit. on pp. 21, 22, 75).

Douillard, Arthur, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle (2020a). "PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning". In: *European Conference on Computer Vision (ECCV)* (cit. on p. 45).

– (2020b). "Podnet: Pooled outputs distillation for small-tasks incremental learning". In: *European Conference on Computer Vision*. Springer, pp. 86–102 (cit. on pp. 45, 57).

Douillard, Arthur and Timothée Lesort (2021). "Continuum: Simple Management of Complex Continual Learning Scenarios". In: *arXiv* (cit. on pp. 51, 74).

Draelos, Timothy John, Nadine E Miner, Christopher Lamb, Jonathan A Cox, Craig Michael Vineyard, Kristofor David Carlson, William Mark Severa, Conrad D James, and James Bradley Aimone (2017). "Neurogenesis Deep Learning". In: *IJCNN* (cit. on p. 48).

Duncker, Lea, Laura N Driscoll, Krishna V Shenoy, Maneesh Sahani, and David Sussillo (2020). "Organizing Recurrent Network Dynamics by Task-Computation to Enable Continual Learning". In: *Advances in Neural Information Processing Systems*. Vol. 33 (cit. on pp. 90, 92–94).

Dwivedi, Vijay Prakash, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson (Dec. 27, 2022). *Benchmarking Graph Neural Networks*. URL: http://arxiv.org/abs/2003.00982 (visited on 03/31/2023). preprint (cit. on p. 40).

Ehret, Benjamin, Christian Henning, Maria Cervera, Alexander Meulemans, Johannes Von Oswald, and Benjamin F. Grewe (2020). "Continual Learning in Recurrent Neural Networks". In: *International Conference on Learning Representations* (cit. on pp. 90, 92, 93).

Elman, Jeffrey L. (1990). "Finding Structure in Time". In: *Cognitive Science* 14.2 (cit. on pp. 4, 16, 17, 92).

Elwell, Ryan and Robi Polikar (2011). "Incremental Learning of Concept Drift in Nonstationary Environments". In: *IEEE Transactions on Neural Networks* 22.10 (cit. on p. 28).

Fiat, Amos and Gerhard J. Woeginger, eds. (1998). *Online Algorithms*. Red. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 1442. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer (cit. on p. 25).

Fini, Enrico, Victor G. Turrisi da Costa, Xavier Alameda-Pineda, Elisa Ricci, Karteek Alahari, and Julien Mairal (2022). "Self-Supervised Models Are Continual Learners". In: *CVPR* (cit. on p. 82).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". en. In: *ICML*. Chap. Machine Learning, pp. 1126–1135 (cit. on p. 37).

Foundation, Wikimedia (n.d.). *Wikimedia Downloads*. URL: https://dumps.wikimedia.org (cit. on p. 21).

French, Robert (1991). "Using Semi-Distributed Representations to Overcome Catastrophic Forgetting in Connectionist Networks". In: *In Proceedings of the 13th Annual Cognitive Science Society Conference*. Erlbaum, pp. 173–178 (cit. on pp. 1, 29, 30, 91).

– (1997a). "Pseudo-Recurrent Connectionist Networks: An Approach to the 'Sensitivity-Stability' Dilemma". In: *Connection Science* 9.4 (cit. on p. 90).

– (1997b). "Using Pseudo-Recurrent Connectionist Networks to Solve the Problem of Sequential Learning". In: *Proceedings of the 19th Annual Cognitive Science Society Conference* (cit. on pp. 89, 90).

– (1999). "Catastrophic Forgetting in Connectionist Networks". In: *Trends in Cognitive Sciences* 3.4 (cit. on pp. 30, 91).

Gallicchio, Claudio, Alessio Micheli, and Luca Pedrelli (2017). "Deep Reservoir Computing: A Critical Experimental Analysis". In: *Neurocomputing* 268 (cit. on p. 6).

Gama, JoÃ£o and Gladys Castillo (2006). "Learning with Local Drift Detection". In: *Advanced Data Mining and Applications*. Ed. by Xue Li, Osmar R. ZaÃ¯ane, and Zhanhuai Li. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 42–55 (cit. on p. 27).

Gama, João, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia (2014). "A Survey on Concept Drift Adaptation". In: *ACM Computing Surveys (CSUR)* 46.4 (cit. on p. 24).

Ganin, Yaroslav and Victor Lempitsky (June 1, 2015). "Unsupervised Domain Adaptation by Backpropagation". In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1180–1189 (cit. on p. 32).

Geiger, R. Stuart (2019). *ArXiV Archive: A Tidy and Complete Archive of Metadata for Papers on Arxiv.Org, 1993-2019* (cit. on p. 74).

Gemmeke, J. F., D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter (2017). "Audio Set: An Ontology and Human-Labeled Dataset for Audio Events". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 776–780 (cit. on pp. 94, 105).

Gomes, Heitor Murilo, Jesse Read, Albert Bifet, Jean Paul Barddal, and JoÃ£o Gama (2019). "Machine Learning for Streaming Data: State of the Art, Challenges, and Opportunities". In: *ACM SIGKDD Explorations Newsletter* 21.2 (cit. on p. 26).

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., pp. 2672–2680 (cit. on p. 47).

Goodfellow, Ian J, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio (2013). "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211* (cit. on pp. 38, 122).

Graffieti, Gabriele, Davide Maltoni, Lorenzo Pellegrini, and Vincenzo Lomonaco (2022). *Generative Negative Replay for Continual Learning* (cit. on p. 47).

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural Turing Machines". In: *arXiv:1410.5401 [cs]* (cit. on p. 93).

Gu, Yu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon (2021). "Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing". In: *ACM Transactions on Computing for Healthcare* 3.1 (cit. on p. 82).

Gururangan, Suchin, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith (2020). "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 8342–8360 (cit. on p. 81).

Ha, David and Douglas Eck (2018). "A Neural Representation of Sketch Drawings". In: *ICLR* (cit. on pp. 95, 96).

Hadsell, Raia, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu (2020). "Embracing Change: Continual Learning in Deep Neural Networks". In: *Trends in Cognitive Sciences* (cit. on p. 73).

Han, Xuejun and Yuhong Guo (2021). "Contrastive Continual Learning with Feature Propagation". In: *arXiv:2112.01713 [cs]* (cit. on pp. 74, 82).

Hawkins, Douglas, Peihua Qiu, and Chang Kang (2003). "The Changepoint Model for Statistical Process Control". In: *Journal of Quality Technology* 35 (cit. on p. 27).

Hayes, Tyler L, Nathan D Cahill, and Christopher Kanan (2019). "Memory Efficient Experience Replay for Streaming Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)* (cit. on p. 36).

Hayes, Tyler L and Christopher Kanan (2020). "Lifelong Machine Learning with Deep Streaming Linear Discriminant Analysis". In: *CLVision Workshop at CVPR 2020*, pp. 1–15 (cit. on p. 47).

Hayes, Tyler L., Giri P. Krishnan, Maxim Bazhenov, Hava T. Siegelmann, Terrence J. Sejnowski, and Christopher Kanan (2021). "Replay in Deep Learning: Current Approaches and Missing Biological Elements". In: *Neural Computation* 33.11 (cit. on pp. 47, 48).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778 (cit. on p. 75).

He, Xu, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu (2019). *Task Agnostic Continual Learning via Meta Learning* (cit. on p. 37).

Hemati, Hamed, Andrea Cossu, Antonio Carta, Julio Hurtado, Lorenzo Pellegrini, Davide Bacciu, Vincenzo Lomonaco, and Damian Borth (2023). "Class-Incremental Learning with Repetition". In: *arXiv* arXiv:2301.11396 (cit. on pp. 5, 56).

Henaff, Mikael, Arthur Szlam, and Yann LeCun (2016). "Recurrent Orthogonal Networks and Long-Memory Tasks". en. In: *ICML*, pp. 2034–2042 (cit. on p. 105).

Hendrycks, Dan and Kevin Gimpel (Mar. 29, 2023). "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: International Conference on Learning Representations (cit. on p. 24).

Hess, Timm, Martin Mundt, Iuliia Pliushch, and Visvanathan Ramesh (Nov. 4, 2021). "A Procedural World Generation Framework for Systematic Evaluation of Continual Learning". In: Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (cit. on pp. 34, 39).

Hinton, Geoffrey, Oriol Vinyals, and Jeffrey Dean (2015). "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop* (cit. on p. 45).

Hochreiter, Sepp (1998). "The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (cit. on p. 18).

Hochreiter, Sepp and Jurgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9 (cit. on pp. 4, 16, 18, 92, 96).

Hochreiter, Sepp, A. Steven Younger, and Peter R. Conwell (2001). "Learning to Learn Using Gradient Descent". en. In: *Artificial Neural Networks — ICANN 2001*. Ed. by Georg Dorffner, Horst Bischof, and Kurt Hornik. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 87–94 (cit. on p. 37).

Hsu, Yen-Chang, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira (2018). "Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines". In: *NeurIPS Continual learning Workshop* (cit. on p. 51).

Hu, Dapeng, Shipeng Yan, Qizhengqiu Lu, Lanqing Hong, Hailin Hu, Yifan Zhang, Zhenguo Li, Xinchao Wang, and Jiashi Feng (2021). "How Well Does Self-Supervised Pre-Training Perform with Streaming Data?" In: *International Conference on Learning Representations* (cit. on pp. 80, 82, 83).

Hu, Weihua, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec (2020). "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc Aurelio

Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (cit. on pp. 34, 40).

Isele, David and Akansel Cosgun (2018). "Selective Experience Replay for Lifelong Learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 46).

Jaeger, Herbert and Harald Haas (2004). "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication". In: *Science* 304 (cit. on p. 109).

Jang, Joel, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo (2022). "TemporalWiki: A Lifelong Benchmark for Training and Evaluating Ever-Evolving Language Models". In: *arXiv:2204.14211 [cs]* (cit. on pp. 71, 82).

Jang, Joel, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo (2021). "Towards Continual Knowledge Learning of Language Models". In: *International Conference on Learning Representations* (cit. on p. 82).

Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell (2014). "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678 (cit. on p. 122).

Jin, Xisen, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren (2022). "Lifelong Pretraining: Continually Adapting Language Models to Emerging Corpora". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics* (cit. on p. 82).

Junczys-Dowmunt, Marcin, Bruno Pouliquen, and Christophe Mazenc (2016). "COPPA V2.0: Corpus of Parallel Patent Applications. Building Large Parallel Corpora with GNU Make". In: *Proceedings of the 4th Workshop on Challenges in the Management of Large Corpora, Portorož, Slovenia, May 23-28, 2016* (cit. on p. 93).

Jung, Heechul, Jeongwoo Ju, Minju Jung, and Junmo Kim (2018). "Less-Forgetful Learning for Domain Expansion in Deep Neural Networks". In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 45).

Kaplanis, Christos, Murray Shanahan, and Claudia Clopath (2018). "Continual Reinforcement Learning with Complex Synapses". In: *ICML* (cit. on p. 87).

– (2019). "Policy Consolidation for Continual Reinforcement Learning". In: *ICML* (cit. on p. 87).

Kemker, Ronald, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan (2018). "Measuring Catastrophic Forgetting in Neural Networks". In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 105).

Kim, Chris Dongjoo, Jinseo Jeong, and Gunhee Kim (2020). "Imbalanced Continual Learning with Partitioning Reservoir Sampling". In: *ECCV* (cit. on pp. 57, 62).

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *ICLR*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 105).

Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell (2017). "Overcoming Catastrophic Forgetting in Neural Networks". In: *PNAS* 114.13 (cit. on pp. 40, 44, 62, 96, 106, 113, 124).

Klinkenberg, Ralf (2004). "Learning Drifting Concepts: Example Selection vs. Example Weighting". In: *Intelligent Data Analysis* 8.3 (cit. on p. 27).

Kobayashi, Taisuke and Toshiki Sugino (2019). "Continual Learning Exploiting Structure of Fractal Reservoir Computing". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*. Ed. by Igor V Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis. Vol. 11731. Cham: Springer International Publishing, pp. 35–47 (cit. on pp. 90, 91, 93, 94).

Kornblith, Simon, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton (2019). "Similarity of Neural Network Representations Revisited". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 3519–3529 (cit. on pp. 23, 65, 72, 79, 80).

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). *Learning multiple layers of features from tiny images*. Tech. rep. (cit. on p. 62).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc. (cit. on p. 17).

Kruszewski, Germán, Ionut-Teodor Sorodoc, and Tomas Mikolov (2020). "Evaluating Online Continual Learning with CALM". In: *arXiv* (cit. on pp. 90, 91, 93, 94).

Lazaridou, Angeliki, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomáš Kočiský, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom (2021). "Mind the Gap: Assessing Temporal Generalization in Neural Language Models". In: *Thirty-Fifth Conference on Neural Information Processing Systems* (cit. on pp. 74, 82).

Le, Quoc V, Navdeep Jaitly, and Geoffrey E Hinton (2015). "A simple way to initialize recurrent networks of rectified linear units". In: *arXiv preprint arXiv:1504.00941* (cit. on p. 93).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep Learning". In: *Nature* 521.7553 (cit. on pp. 1, 9, 16).

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (cit. on pp. 16, 17, 38, 62, 93).

Lee, Cecilia S and Aaron Y Lee (2020). "Clinical Applications of Continual Learning Machine Learning". In: *The Lancet Digital Health* 2.6 (cit. on p. 81).

Lesort, Timothee, Thomas George, and Irina Rish (2021). "Continual Learning in Deep Networks: An Analysis of the Last Layer". In: *arXiv* (cit. on pp. 34, 48).

Lesort, Timothee, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Diaz-Rodriguez (2020). "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges". In: *Information fusion* 58 (cit. on p. 1).

Lesort, Timothée, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez (2020). "Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges". In: *Information Fusion* 58 (cit. on p. 32).

Lesort, Timothée, Andrei Stoian, and David Filliat (2020). "Regularization Shortcomings for Continual Learning". In: *arXiv* (cit. on pp. 45, 97).

Li, Shancang, Li Da Xu, and Shanshan Zhao (2015). "The Internet of Things: A Survey". In: *Information Systems Frontiers* 17.2 (cit. on p. 2).

Li, Yuanpeng, Liang Zhao, Kenneth Church, and Mohamed Elhoseiny (2020). "Compositional Language Continual Learning". In: *Eighth International Conference on Learning Representations* (cit. on pp. 90, 91).

Li, Zhizhong and Derek Hoiem (2016). "Learning without Forgetting". In: *European Conference on Computer Vision*. Springer, pp. 614–629 (cit. on pp. 45, 62, 67, 97, 113).

Lison, Pierre, Jörg Tiedemann, and Milen Kouylekov (2018). "OpenSubtitles2018: Statistical Rescoring of Sentence Alignments in Large, Noisy Parallel Corpora". In: *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA), pp. 1742–1748 (cit. on p. 93).

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov (2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *arXiv:1907.11692 [cs]* (cit. on pp. 10, 21, 74).

Lomonaco, Vincenzo, Karan Desai, Eugenio Culurciello, and Davide Maltoni (2020). "Continual Reinforcement Learning in 3D Non-Stationary Environments". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 248–249 (cit. on pp. 48, 49).

Lomonaco, Vincenzo and Davide Maltoni (2017). "CORe50: A New Dataset and Benchmark for Continuous Object Recognition". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, pp. 17–26 (cit. on pp. 34–36, 38, 41, 75, 92, 94).

Lomonaco, Vincenzo, Davide Maltoni, and Lorenzo Pellegrini (2020). "Rehearsal-Free Continual Learning over Small Non-IID Batches." In: *CVPR Workshops*, pp. 989–998 (cit. on p. 54).

Lomonaco, Vincenzo, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido M. van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas S. Tolias, Simone Scardapane, Luca Antiga, Subutai Ahmad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni (June 2021). "Avalanche: An End-to-End Library for Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3600–3610 (cit. on pp. 7, 32, 33, 51, 58, 72, 77, 110, 112, 123).

Lopez-Paz, David and Marc'Aurelio Ranzato (2017). "Gradient Episodic Memory for Continual Learning". In: *NIPS* (cit. on pp. 38, 42, 50, 57, 92, 96, 97).

Losing, Viktor, Barbara Hammer, and Heiko Wersing (2018). "Incremental On-Line Learning: A Review and Comparison of State of the Art Algorithms". In: *Neurocomputing* 275 (cit. on p. 25).

Lukoševičius, Mantas (2012). "A Practical Guide to Applying Echo State Networks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 659–686 (cit. on p. 18).

Lukoševičius, Mantas and Herbert Jaeger (2009). "Reservoir Computing Approaches to Recurrent Neural Network Training". en. In: *Computer Science Review* 3 (cit. on pp. 6, 18, 91).

Madaan, Divyam, Jaehong Yoon, Yuanchun Li, Yunxin Liu, and Sung Ju Hwang (2021). "Rethinking the Representational Continuity: Towards Unsupervised Continual Learning". In: *International Conference on Learning Representations* (cit. on p. 80).

Madasu, Avinash and Vijjini Anvesh Rao (2020). "Sequential Domain Adaptation through Elastic Weight Consolidation for Sentiment Analysis". In: *arXiv* (cit. on pp. 90, 91, 93, 94).

Maltoni, Davide and Vincenzo Lomonaco (2016). "Semi-Supervised Tuning from Temporal Coherence". In: *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2509–2514 (cit. on p. 57).

– (2019). "Continuous Learning in Single-Incremental-Task Scenarios". In: *Neural Networks* 116 (cit. on pp. 35, 36, 38).

Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org (cit. on p. 124).

Masana, Marc, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer (2020). "Class-incremental learning: survey and performance evaluation". In: *arXiv preprint arXiv:2010.15277* (cit. on pp. 43, 51).

Matteoni, Federico, Andrea Cossu, Claudio Gallicchio, Vincenzo Lomonaco, and Davide Bacciu (2022). "Continual Learning for Human State Monitoring". In: *30th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (cit. on pp. 6, 7, 101, 114).

McCloskey, Michael and Neal J. Cohen (1989). "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem". In: *Psychology of Learning and Motivation*. Ed. by Gordon H. Bower. Vol. 24. Academic Press, pp. 109–165 (cit. on pp. 1, 29).

Mcculloch, Warren and Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5 (cit. on p. 13).

Mehta, Nikhil, Kevin J Liang, and Lawrence Carin (2020). "Bayesian Nonparametric Weight Factorization for Continual Learning". In: *arXiv* (cit. on p. 70).

Mehta, Sanket Vaibhav, Darshan Patil, Sarath Chandar, and Emma Strubell (2021). "An Empirical Investigation of the Role of Pre-training in Lifelong Learning". In: *arXiv:2112.09153 [cs]* (cit. on pp. 70, 81).

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher (2016). "Pointer Sentinel Mixture Models". In: *arXiv:1609.07843 [cs]* (cit. on p. 76).

Merlin, Gabriele, Vincenzo Lomonaco, Andrea Cossu, Antonio Carta, and Davide Bacciu (2021). "Practical Recommendations for Replay-based Continual Learning Methods". In: *Workshop on Novel Benchmarks and Approaches for Real-World Continual Learning (CL4REAL)* (cit. on pp. 46, 47).

Mhammedi, Zakaria, Andrew Hellicar, Ashfaqur Rahman, and James Bailey (2017). "Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections". en. In: *ICML*, pp. 2401–2409 (cit. on pp. 18, 105).

Misra, Ishan and Laurens van der Maaten (2020). “Self-Supervised Learning of Pretext-Invariant Representations”. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6707–6717 (cit. on p. 11).

Mitchell, Tom (1997). *Machine Learning*. McGraw Hill (cit. on pp. 1, 9, 33).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). “Playing Atari with Deep Reinforcement Learning”. In: *arXiv:1312.5602 [cs]* (cit. on pp. 11, 34, 40, 46).

Nguyen, Thao, Maithra Raghu, and Simon Kornblith (2020). “Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth”. In: *International Conference on Learning Representations* (cit. on pp. 23, 77).

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, PrzemysÅ‚aw DÄ™biak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal JÃ³zefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang (Dec. 13, 2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. URL: http://arxiv.org/abs/1912.06680 (visited on 03/30/2023). preprint (cit. on p. 11).

Ororbia, Alexander (2020). “Spiking Neural Predictive Coding for Continual Learning from Data Streams”. In: *arXiv* (cit. on pp. 90, 91).

Ororbia, Alexander, Ankur Mali, C Lee Giles, and Daniel Kifer (2019). “Continual Learning of Recurrent Neural Networks by Locally Aligning Distributed Representations”. In: *arXiv* (cit. on pp. 90, 92).

Ostapenko, Oleksiy, Timothee Lesort, Pau Rodríguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin (2022). “Foundational Models for Continual Learning: An Empirical Study of Latent Replay”. In: *arXiv:2205.00329 [cs]* (cit. on p. 47).

Parisi, German I, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter (2019). “Continual Lifelong Learning with Neural Networks: A Review”. In: *Neural Networks* 113 (cit. on pp. 1, 31, 44, 48).

Parisi, German I, Jun Tani, Cornelius Weber, and Stefan Wermter (2018). “Lifelong Learning of Spatiotemporal Representations With Dual-Memory Recurrent Self-Organization”. In: *Frontiers in Neurorobotics* 12 (cit. on pp. 90, 92, 93).

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett. Curran Associates, Inc., pp. 8024–8035 (cit. on p. 122).

Patist, Jan Peter (Oct. 2007). “Optimal Window Change Detection”. In: *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007), pp. 557–562 (cit. on p. 27).

Pellegrini, Lorenzo, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni (2020). “Latent replay for real-time continual learning”. In: *Proceedings of the 2020*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (cit. on p. 47).

Philps, Daniel, Artur d'Avila Garcez, and Tillman Weyde (2019). "Making Good on LSTMs' Unfulfilled Promise". In: *arXiv* (cit. on p. 92).

Pineau, Joelle, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché-Buc, Emily Fox, and Hugo Larochelle (2020). "Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program)". In: *arXiv preprint arXiv:2003.12206* (cit. on p. 51).

Qin, Yujia, Jiajie Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou (2022). "ELLE: Efficient Lifelong Pre-training for Emerging Data". In: *Findings of ACL* (cit. on p. 83).

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang (2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *arXiv e-prints*, arXiv:1606.05250 (cit. on p. 22).

Ramasesh, Vinay Venkatesh, Aitor Lewkowycz, and Ethan Dyer (2021). "Effect of Scale on Catastrophic Forgetting in Neural Networks". In: *International Conference on Learning Representations* (cit. on pp. 70, 81).

Ratcliff, R (1990). "Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions." In: *Psychological review* 97.2 (cit. on p. 1).

Rebuffi, Sylvestre-Alvise, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert (2017). "iCaRL: Incremental Classifier and Representation Learning". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 3, 34, 38, 46, 50, 53, 57, 122).

Riemer, Matthew, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro (2019). "Learning to Learn without Forgetting by Maximizing Transfer and Minimizing Interference". In: *ICLR* (cit. on p. 46).

Ring, Mark (1994). "Continual Learning in Reinforcement Environments". PhD thesis. University of Texas (cit. on p. 87).

Ring, Mark B (1997). "CHILD: A First Step Towards Continual Learning". In: *Machine Learning* 28.1 (cit. on pp. 89, 90).

Roady, Ryne, Tyler L. Hayes, Hitesh Vaidya, and Christopher Kanan (2020). "Stream-51: Streaming Classification and Novelty Detection From Videos". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 228–229 (cit. on pp. 34, 39).

Robins, Anthony (1995). "Catastrophic Forgetting; Catastrophic Interference; Stability; Plasticity; Rehearsal." In: *Connection Science* 7.2 (cit. on pp. 29, 90).

Rolnick, David, Arun Ahuja, Jonathan Schwarz, Timothy P Lillicrap, and Greg Wayne (2019). "Experience Replay for Continual Learning". In: *NeurIPS*, pp. 350–360 (cit. on p. 46).

Rongali, Subendhu, Abhyuday Jagannatha, Bhanu Pratap Singh Rawat, and Hong Yu (2021). "Continual Domain-Tuning for Pretrained Language Models". In: *arXiv:2004.02288 [cs]* (cit. on p. 81).

Rosasco, Andrea, Antonio Carta, Andrea Cossu, Vincenzo Lomonaco, and Davide Bacciu (2021). "Distilled Replay: Overcoming Forgetting through Synthetic Samples". In: *1st International Workshop on Continual Semi-Supervised Learning (CSSL) at IJCAI* (cit. on pp. 46, 47).

Ruder, Sebastian, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf (2019). "Transfer Learning in Natural Language Processing". In: *Proceedings of the 2019*

Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 15–18 (cit. on p. 70).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning Representations by Back-Propagating Errors". In: *Nature* 323.6088 (cit. on p. 15).

Russell, Stuart and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd. Pearson (cit. on pp. 1, 9).

Rusu, Andrei A, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell (2016). "Progressive Neural Networks". In: *arXiv* (cit. on pp. 49, 92, 105).

Schak, Monika and Alexander Gepperth (2019). "A Study on Catastrophic Forgetting in Deep LSTM Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*. Ed. by Igor V Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 714–728 (cit. on pp. 90, 92, 105, 107).

Schmidt, P., A. Reiss, R. Duerichen, C. Marberger, and K. Van Laerhoven (2018). "Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection". In: *Proceedings of the 20th ACM International Conference on Multimodal Interaction*. ICMI 2018. Boulder, CO, USA: Association for Computing Machinery, pp. 400–408 (cit. on p. 112).

Schwarz, Jonathan, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell (2018). "Progress & Compress: A Scalable Framework for Continual Learning". In: *International Conference on Machine Learning*, pp. 4528–4537 (cit. on pp. 38, 45, 49).

Serra, Joan, Didac Suris, Marius Miron, and Alexandros Karatzoglou (Oct. 2018). "Overcoming Catastrophic Forgetting with Hard Attention to the Task". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 4548–4557 (cit. on p. 51).

She, Qi, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, Fei Qiao, and Rosa H M Chan (2019). "OpenLORIS-Object: A Robotic Vision Dataset and Benchmark for Lifelong Deep Learning". In: *arXiv* (cit. on pp. 34, 39).

Shin, Hanul, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim (2017). "Continual Learning with Deep Generative Replay". In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett. Curran Associates, Inc., pp. 2990–2999 (cit. on p. 47).

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016). "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (7587) (cit. on p. 11).

Sodhani, Shagun, Sarath Chandar, and Yoshua Bengio (2019). "Toward Training Recurrent Neural Networks for Lifelong Learning". In: *Neural Computation* 32.1 (cit. on pp. 90, 92, 93).

Sokar, Ghada, Decebal Constantin Mocanu, and Mykola Pechenizkiy (2021). "SpaceNet: Make Free Space for Continual Learning". In: *Neurocomputing* 439 (cit. on p. 49).

Spiekermann, Sarah (2012). "The Challenges of Privacy by Design". In: *Communications of the ACM* 55.7 (cit. on p. 2).

Srivastava, Nitish, Elman Mansimov, and Ruslan Salakhutdinov (2015). "Unsupervised Learning of Video Representations Using LSTMs". en. In: *ICML* (cit. on p. 103).

Stojanov, Stefan, Samarth Mishra, Ngoc Anh Thai, Nikhil Dhanda, Ahmad Humayun, Chen Yu, Linda B. Smith, and James M. Rehg (2019). "Incremental Object Learning From Contiguous Views". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8777–8786 (cit. on p. 54).

Street, W. Nick and YongSeog Kim (Aug. 26, 2001). "A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification". In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '01. New York, NY, USA: Association for Computing Machinery, pp. 377–382 (cit. on p. 28).

Subramanian, Ramanathan, Julia Wache, Mojtaba Khomami Abadi, Radu L. Vieriu, Stefan Winkler, and Nicu Sebe (2018). "ASCERTAIN: Emotion and Personality Recognition Using Commercial Sensors". In: *IEEE Transactions on Affective Computing* 9.2 (cit. on p. 112).

Sutton, Richard and Andrew Barto (2018). *Reinforcement Learning: An Introduction*. Second Edition. MIT Press (cit. on p. 11).

Thai, Anh, Stefan Stojanov, Isaac Rehg, and James M. Rehg (2021). "Does Continual Learning = Catastrophic Forgetting?" In: *arXiv* (cit. on pp. 57, 58).

Thompson, Brian, Jeremy Gwinnup, Huda Khayrallah, Kevin Duh, and Philipp Koehn (2019). "Overcoming Catastrophic Forgetting During Domain Adaptation of Neural Machine Translation". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 2062–2068 (cit. on pp. 90, 91, 94).

Tsymbal, Alexey (2004). *The Problem of Concept Drift: Definitions and Related Work*. Tech. rep. Trinity College, Dublin (cit. on p. 24).

Tsymbal, Alexey, Mykola Pechenizkiy, PÃ¡draig Cunningham, and Seppo Puuronen (2008). "Dynamic Integration of Classifiers for Handling Concept Drift". In: *Information Fusion*. Special Issue on Applications of Ensemble Methods 9.1 (cit. on p. 28).

van de Ven, Gido M., Hava T. Siegelmann, and Andreas S. Tolias (2020a). "Brain-Inspired Replay for Continual Learning with Artificial Neural Networks". In: *Nature Communications* 11 (cit. on p. 57).

– (2020b). "Brain-like Replay for Continual Learning with Artificial Neural Networks". In: *International Conference on Learning Representations (Workshop on Bridging AI and Cognitive Science)* (cit. on p. 47).

van de Ven, Gido M. and Andreas S. Tolias (2018a). "Generative Replay with Feedback Connections as a General Strategy for Continual Learning". In: *arXiv* (cit. on pp. 34, 51).

– (2018b). "Three Scenarios for Continual Learning". In: *Continual Learning Workshop NeurIPS* (cit. on pp. 3, 32, 34–36, 38, 47, 51, 53).

Van Horn, Grant, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie (2018). "The INaturalist Species

Classification and Detection Dataset". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8769–8778 (cit. on p. 75).
Vaswani, Ashish (2017). "Attention Is All You Need". In: *arXiv:1706.03762 [cs]* (cit. on pp. 16, 19, 74).
Verwimp, Eli, Matthias De Lange, and Tinne Tuytelaars (2021). "Rehearsal Revealed: The Limits and Merits of Revisiting Samples in Continual Learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9385–9394 (cit. on p. 47).
Villa, Andrés, Kumail Alhamoud, Juan León Alcázar, Fabian Caba Heilbron, Victor Escorcia, and Bernard Ghanem (2022). "vCLIMB: A Novel Video Class Incremental Learning Benchmark". In: *arXiv* (cit. on pp. 34, 39).
Vitter, Jeffrey S (1985). "Random sampling with a reservoir". In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (cit. on pp. 27, 60).
von Oswald, Johannes, Christian Henning, João Sacramento, and Benjamin F Grewe (2020). "Continual Learning with Hypernetworks". In: *International Conference on Learning Representations* (cit. on p. 92).
Vuorio, Risto, Dong-Yeon Cho, Daejoong Kim, and Jiwon Kim (2018). "Meta Continual Learning". In: *arXiv* (cit. on p. 37).
Wald, A. (1945). "Sequential Tests of Statistical Hypotheses". In: *The Annals of Mathematical Statistics* 16.2 (cit. on p. 27).
Wang, Alex, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman (Nov. 2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, pp. 353–355 (cit. on p. 22).
Wang, Tongzhou, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros (2020). "Dataset Distillation". In: *arXiv:1811.10959 [cs, stat]* (cit. on p. 46).
Widmer, Gerhard and Miroslav Kubat (1996). "Learning in the Presence of Concept Drift and Hidden Contexts". In: *Machine Learning* 23.1 (cit. on p. 28).
Williams, Adina, Nikita Nangia, and Samuel Bowman (2018). "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". en. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1112–1122 (cit. on p. 94).
Wolczyk, Maciej, Michal Zajkac, Razvan Pascanu, Lukasz Kucinski, and Piotr Milos (2021). "Continual World: A Robotic Benchmark For Continual Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., pp. 28496–28510 (cit. on p. 43).
Wolf, Thomas, Julien Chaumond, and Clement Delangue (2018). "Continuous Learning in a Hierarchical Multiscale Neural Network". In: *ACL* (cit. on pp. 90, 91, 93, 94).
Wu, Tongtong, Massimo Caccia, Zhuang Li, Yuan-Fang Li, Guilin Qi, and Gholamreza Haffari (2021). "Pretrained Language Model in Continual Learning: A Comparative Study". In: *International Conference on Learning Representations* (cit. on pp. 70, 81).
Wu, Yue, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu (2019). "Large scale incremental learning". In: *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 374–382 (cit. on p. 57).

Xue, Jiabin, Jiqing Han, Tieran Zheng, Xiang Gao, and Jiaxing Guo (2019). "A Multi-Task Learning Framework for Overcoming the Catastrophic Forgetting in Automatic Speech Recognition". In: *arXiv* (cit. on pp. 90, 92, 93).

Zenke, Friedemann, Ben Poole, and Surya Ganguli (2017). "Continual Learning Through Synaptic Intelligence". In: *International Conference on Machine Learning*, pp. 3987–3995 (cit. on pp. 45, 122).

Zhang, Rong, Revanth Gangi Reddy, Md Arafat Sultan, Vittorio Castelli, Anthony Ferritto, Radu Florian, Efsun Sarioglu Kayi, Salim Roukos, Avi Sil, and Todd Ward (2020). "Multi-Stage Pre-training for Low-Resource Domain Adaptation". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, pp. 5461–5468 (cit. on p. 74).

Zhu, Yukun, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler (Dec. 2015). "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *The IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 21).