Continual forgetting-free deep learning from high-dimensional data streams

Andrey Besedin

le cnam

École doctorale Informatique, Télécommunications et Électronique (Paris)

Centre d'études et de recherche en informatique et communication

THÈSE DE DOCTORAT

présentée par : **Andrey BESEDIN**

soutenue le : **10 décembre 2019**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Spécialité : **Informatique**

Continual Forgetting-Free Deep Learning from High-dimensional Data Streams

THÈSE dirigée par

M. Michel Crucianu — *PR1, CNAM*

et co-encadrée par

M. Blanchart Pierre — *Cadre scientifique des EPIC, CEA LIST*

M. Ferecatu Marin — *Maître de Conférences, CNAM*

RAPPORTEURS

M. Gosselin Philippe-Henri — *Professeur, InterDigital*

M. Lefévre Sébastien — *Professeur, IRISA*

PRÉSIDENT DU JURY

Mme Gouet-Brunet Valérie — *Directeur de Recherche , IGN*

EXAMINATEURS

M. Sabhi Hichem — *Directeur de Recherche , LIP6*


DE LA RECHERCHE À L'INDUSTRIE
cea
list


EDiF

Acknowledgements

I would like to thank everyone who supported me during these 3+ years and helped me make this journey possible.

First of all, I would like to thank CEA LIST and in particular LI3A (former LADIS) lab for the research grant that supported my research and for having provided me with everything I needed to accomplish this work. Additional thanks to the lab for accepting me into its outstanding work environment with very talented, open-minded and highly motivated people.

I would like to thank my thesis advisers. I am extremely grateful to Prof. Michel Crucianu for having accepted me to this thesis under his direction. Thank you for sharing your scientific rigor, expertise, and vision. I would like to thank Dr. Marin Ferecatu for all the long scientific and general discussions we had. Thank you for your guidance through all the different aspects of academic work, and for your inspiring scientific curiosity. Finally, I thank Dr. Pierre Blanchart for answering all my questions, helping me to get the real taste of problem-solving and for being so responsive. Thank you for your constructive criticism, it made my progress through the whole Ph.D. much faster.

I would like to individually thank Dr. Marine Depecker who kindly proposed to help me structure my manuscript, and could find the right words to keep me motivated and concentrated when I needed it the most.

Big thanks to my favorite CEA tea-team, Sandra and Shivani. Our frequent conversations were of huge value for me, professionally and personally. You became much more than just colleagues and friends to me. I also thank all the people from the lab who were joining us for weekly team-building football matches, it was great to see you outside of work.

Special thanks to my closest friends, Ksenia and Anna, who were and will always be there to help me, give advice or just hear me out.

I am extremely grateful to my parents, Tatyana and Alexandre, who always encouraged my curiosity and my desire to learn. Thank you for giving me the opportunity to study, without your help and support this thesis would not be possible.

Special thanks to my wife Mélody. You shared with me all the highs and lows, were there when I needed it the most, and always believed in me no matter how things were going. It is hard to believe that becoming a parent during the last year of Ph.D. can have a positive impact on the latter. It was the case for me. I would like to thank my son, Sasha, first, for being a wise kid who respects his sleeping hours not making things even more complicated, and, second, for helping me (probably, involuntarily) to improve my self-organizing skills and giving me extra motivation to succeed.

Résumé

Dans cette thèse, nous proposons une nouvelle approche de l'apprentissage profond pour la classification des flux de données de grande dimension. Au cours des dernières années, les réseaux de neurones sont devenus la référence dans diverses applications d'apprentissage automatique. Cependant, la plupart des méthodes basées sur les réseaux de neurones sont conçues pour résoudre des problèmes d'apprentissage statique. La caractéristique principale d'un tel apprentissage est que toutes les données sont disponibles pendant la durée complète de l'entraînement.

Effectuer un apprentissage en ligne, particulièrement une classification, à l'aide de réseaux de neurones est une tâche difficile. La principale difficulté est que les classificateurs basés sur les réseaux de neurones reposent généralement sur l'hypothèse que la séquence des batches de données utilisée pendant l'entraînement est stationnaire; ou en d'autres termes, que la distribution des classes de données est la même pour tous les batches (hypothèse i.i.d.). Lorsque cette hypothèse ne tient pas, ce qui est très souvent le cas dans l'apprentissage en ligne, les réseaux de neurones ont tendance à oublier les concepts temporairement indisponibles dans le flux. Cet effet est dû au fait que la rétropropagation tend à renforcer les classes présentes dans le batch actuel. Dans la littérature scientifique, ce phénomène est généralement appelé *oubli catastrophique*.

Le but de la première approche de cette thèse est de garantir la nature i.i.d. de chaque batch qui provient du flux et de compenser l'absence de données historiques. Pour ce faire, nous entraînons des réseaux génératifs (Generative Adversarial Networks, GAN), un par classe de données. Ensuite, lors de l'entrainement du classificateur, les GAN génèrent des échantillons synthétiques à partir des classes absentes ou mal représentées dans le flux, et complètent les batches du flux avec ces échantillons.

Nous testons notre approche dans un scénario d'apprentissage incrémental et dans un type spécifique de l'apprentissage continu. Nous démontrons la capacité de notre approche à s'adapter à des classes de données jamais vues ou à de nouvelles instances de classes déjà vues tout en évitant d'oublier les classes / instances de classes apprises précédemment qui n'apparaissent plus dans le flux de données.

L'approche proposée ne peut pas résoudre de larges problèmes avec des milliers de classes de données; ni être appliquée à des types plus généraux de flux continus. Pour pallier ces limitations, nous proposons un nouvel autoencodeur génératif doté d'une fonction de perte auxiliaire qui assure une convergence rapide spécifique aux tâches. De plus, une telle approche permet un échantillonnage conditionnel de toutes les classes à partir d'un modèle unique. Notre approche est en mesure d'atténuer l'oubli catastrophique dans le scénario d'apprentissage continu, sans hypothèse sur la stationnarité des données dans le flux, en nécessitant très peu de stockage de données historiques. Pour évaluer notre méthode, nous effectuons des expériences sur le jeu de données d'image MNIST bien connu, et sur le jeu de données LSUN plus complexe en mode de diffusion continue. Nous étendons les expériences à un grand ensemble de données synthétiques multi-classes, ce qui permet de vérifier les performances de notre méthode dans des environnements plus difficiles, comprenant jusqu'à 1 000 classes distinctes. Notre approche effectue une classification sur des flux de données dynamiques avec une précision proche des résultats obtenus dans la configuration de classification statique où toutes les données sont disponibles pour la durée de l'apprentissage. En outre, nous démontrons la capacité de notre méthode à s'adapter à des classes de données invisibles et à de nouvelles instances de catégories de données déjà connues, tout en évitant d'oublier les connaissances précédemment acquises.

Mots-clés: Classification de Données, Reseaux de Neurones, Apprentissage Profond, Apprentissage Incrémental, Apprentissage Continu, Flux de Données, Oublie Catastrophique

Summary

In this thesis, we propose a new deep-learning-based approach for online classification on streams of high-dimensional data. In recent years, Neural Networks (NN) have become the primary building block of state-of-the-art methods in various machine learning problems. Most of these methods, however, are designed to solve the static learning problem, when all data are available at once at training time. Performing Online Deep Learning, and specifically online classification using Neural Networks is exceptionally challenging. The main difficulty is that NN-based classifiers usually rely on the assumption that the sequence of data batches used during training is stationary, or in other words, that the distribution of data classes is the same for all batches (i.i.d. assumption). Because backpropagation tends to reinforce the classes present in the current batch, when this assumption does not hold – which is a likely situation in an online learning setting – Neural Networks tend to forget the concepts that are temporarily not available in the stream. In the literature, this phenomenon is known as *catastrophic forgetting*.

To ensure the i.i.d. nature of each training batch in online learning and to make up for the absence of the historical stream data at some point, we proposed a first approach relying on Generative Adversarial Networks (GANs), by training such models to represent and re-generate elements from each data class. Thus, during the online training of the classifier, the GANs model can generate synthetic samples for the classes absent or not well represented in the stream and complete the current training batches with these samples. We test our approach in an incremental and specific type of continuous learning scenario. We demonstrate its ability to adapt to previously unseen data classes or new instances of previously seen classes while avoiding forgetting of previously learned classes/instances of classes that do not appear anymore in the data stream.

Unfortunately, such an approach does not scale to large problems with thousands of data classes; neither it can be applied to more general types of continuous streams. To make up for these limitations, we propose a new approach based on pseudo-generative autoencoder-based models. To train these models, we designed a specific loss function that ensures fast task-sensitive convergence and allows efficient class-conditional sampling from a single model. While requiring very little historical data storage, the proposed approach is able to alleviate catastrophic forgetting in the scenario of continual learning without requiring the class distribution to be stationary in the stream. To evaluate our approach, we perform experiments on the well-known MNIST image dataset and the more complex LSUN dataset, in a continuous streaming mode. We extend the experiments to a large multi-class synthetic dataset. It allows us to assess the performance of our method in a more challenging setting with up to one thousand distinct classes.

The conducted experiment proves that our approach performs classification on dynamic data streams with an accuracy close to the results obtained in the offline classification setup where all the data are available at once for the training. Besides, the experiments show the ability of our method to overcome the main problem of continual learning, which is to adapt to unseen data classes and new instances of already known classes while avoiding catastrophic forgetting of previously learned classes.

Keywordss: Classification, Neural Networks, Deep Learning, Incremental Learning, Continual Learning, Data Streams, Catastrophic forgetting

Contents

List of Figures

List of Tables

Part I

Background material

Chapter 1

Introduction

1.1 Motivation

In the last decades, computers and algorithms have become an essential part of our lives. In industries, they are used to increase productivity by thousands of times, thus saving billions of human working hours. They are actively employed in public and private services such as transportation and health-care, improving our daily routines and life quality. Moreover, computer-based technologies entered private life and changed the way people socialize, learn, and spend their free time.

This success is mainly due to the ability of modern computers to work with enormous amounts of data, efficient algorithms of data processing, and the growing capacity of computational systems. However, the most significant part of the solutions for real-life problems until recently was based on theoretical models and detailed algorithmization of the tasks developed and implemented by specialists in corresponding domains, engineers, and scientists. Such solutions suffer from several limitations: they are hard to design, require a considerable number of parameters to fine-tune, and are usually based on the human understanding of the phenomenon, which can be very approximate and imprecise. As a result, such methods are hard to generalize outside the task for which they are designed.

More importantly, when it comes to the problems considered "easy" by an average human, e.g., reading the hand-writings, detecting objects on the image, or recognizing a song from just the first few seconds of a recording, the direct algorithmization of the process is often extremely complicated. For instance, object detection, segmentation, and classification from natural images are intuitive tasks for humans.

However, one could hardly give step-by-step instructions for a computer to perform those tasks. This algorithmization complexity is the reason why the described tasks form a whole field of scientific research called Computer Vision. This domain has been studied for decades to achieve close to human performance in vision-related tasks.

In contrast to the approaches based on modeling and algorithmization, Machine Learning (ML) aims to make machines perform those tasks without giving explicit instructions. Instead, ML models extract knowledge directly from the raw data and corresponding annotations, if those are provided. Thus, the idea behind ML is to approximate a physical phenomenon by retrieving and organizing knowledge based on observations. Most of the ML methods store the knowledge in the parameters of the model. These parameters are optimized during the learning phase.

In contrast to a biological learning system that can continuously learn during its lifetime, most of the modern ML approaches learn in a static way from a predefined dataset of a limited size. In the era of robotics, social networks, and personalized gadgets that continuously collect user information, the demand for systems that can learn in real-time from vast amounts of data coming from multiple sources is rapidly growing. Due to this increasing need, the domain of online learning that aims to integrate new knowledge into the learning system in real-time has recently started to receive significant attention from the ML community.

1.2 Problem statement

The potential advantages of an online learning system over a system learning "in-place" from a predefined static dataset are numerous. The ability to learn continuously provides the learning system with the capacity to adapt to the changes in the environment and explore things never seen before. Besides, such a system should also be able to adapt to the evolving trends, tasks, updates in software and hardware, under the critical condition of not losing the previously acquired knowledge. The applications of an online learning system can vary from predicting the hash-tags of the images taking into account the latest trends to creating the humanoid bots, which can adapt to the rich and rapidly changing real-world environment.

Compared to training on static data, online learning from the streams introduces several new relatively unstudied challenges. First of all, data streams can suffer from drifts in the underlying distribution due to the changes in the emitting environment, sensors used to retrieve the data can be replaced or updated, or the task of interest can change over time. In such conditions, models tend to fit the distribution of currently available data, which may result in drastic changes in the inference mechanism. Moreover, online learning systems have to learn from the continuously arriving data, which means that the learning mechanism should be able to retrieve all the necessary information in real-time.

To summarize the challenges mentioned above and formalize the requirements one should impose on the online learning system, let us state that such a system should be provided with the following characteristics:

1. Real-time data processing.
2. Fast knowledge incorporation from limited data.
3. Mechanism to protect already acquired knowledge.
4. Efficiency on data of a very high complexity.
5. Scalability to large problems with a significant number of classes.

One can obtain most of the described characteristics by imposing certain conditions on the training procedure (1), the model's design (5), or even the type of the model employed (4). In contrast, (2) and (3) are challenging to handle and often interfere with each other. The literature dedicated to learning in biological and computational systems often addresses the relation between the ability of the system to incorporate new knowledge (2) and retain the old one (3) as the stability-plasticity dilemma ([Gro82], [MBB13]). This notion, while not frequently mentioned in the rest of the thesis, is the core problem of our study.

More specifically, **in this thesis** we focus on the problem of online image classification in a continuous stream context. While useful in a large number of practical computer vision applications, image classification is an acknowledged difficult problem, largely still unsolved in the general case. Moreover, image classifiers need large training sets to perform reasonably well, while dealing with image databases

is very resource-intensive both in terms of storage and computing power. These conditions make it very challenging to perform online training on streams of image data, moreover so because the incoming data is heavy and difficult to store, while the task to solve is complex and requires updating large learning models.

Until recently, the most widely used methods for classification on data streams included Hoeffding trees ([DH00]), Bayesian trees ([SAK+09]), Support Vector Machines ([RDIV09]) and ensemble methods ([Oza05]). A comparative overview of these methods is presented in ([NWN15]). The conclusion is that, even though they allow real-time testing, can efficiently handle concept drift ([WHC+16]) and do not face memory issues, those methods do not perform well on complex high-dimensional data, thus not satisfying (4).

In comparison, methods based on Deep Learning (DL) are efficiently handling complex classification tasks on high-dimensional data with up to a few thousand classes. In recent years, DL-based approaches have become state of the art in numerous applications, such as image and signal classification ([KSH12]), object detection ([SKCL13]) and segmentation ([HGDG17]), natural language processing ([SVL14]), ([CWB+11]) and many others. Despite its popularity and efficiency on high-dimensional data of significant structural complexity, most of the currently existing deep learning approaches are aiming to solve offline learning problems where all the data are constantly available during training. Training DL models on non-stationary stream data with a changing distribution of data classes generally leads to a phenomenon of catastrophic forgetting ([MC89]). The knowledge encoded in the neural connections is gradually overwritten by new information in the absence of data reinforcing previous knowledge, i.e., data corresponding to previously learned classes or to different modes of the current classes.

Currently existing solutions to overcome catastrophic forgetting in Neural Networks include three types of approaches: methods based on training regularization where the network updates depend on the importance of the neural connections for the historical tasks; networks with evolving architectures, able to grow and adapt to the increasing complexity of the problem; and the approaches that make use of a supplementary memory unit that provides the model with missing data classes. In Chapter 4,

we discuss in detail the existing methods from each of the three groups. This bibliographic chapter shows the reader the reasoning behind our choice of direction for the study.

1.3 Conventions and notations

In this section, we introduce several notions and conventions that hold for the rest of the manuscript. We do so to facilitate the understanding of the proposed methodology, avoid confusion in terminology, and provide the reader with a brief overview of the experimental scenarios on which we validate our approaches.

Dataset and data stream. We discriminate between the notions of the **dataset** as a predefined static collection of data samples, and the **data stream** (sometimes referred to as **data source**) as a process of continuous retrieval of data samples from the changing environment. The essential difference between the two is that in the case of the dataset, one can manipulate the collection of data prior to learning, e.g., balance data classes, perform several steps of pre-processing, study the data statistics, etc. All this becomes hardly conceivable when working with data streams.

Learning system. In our study, we employ sophisticated systems that contain several learning blocks, each having its own learning rules, structure, and schedules. For the sake of simplicity of appealing to the full learning mechanism, we call the complete set of architectures used to perform learning for a given problem as a **learning system**.

Learning tasks. In most cases, the term **task** is used in a global context referring to the objective of learning (e.g., classification task, generative task). This notion, however, is sometimes employed to describe a sequence of sub-problems of the global learning problem. To make it more clear, let us consider an example of classification on a dataset containing N separate classes. In these settings, we can imagine a situation in which only a subset of all the different classes is available at each time interval. Thus, we can describe learning on each of these intervals as a separate task.

Online learning This term is used to describe in a very general manner any learning process that involves incorporating knowledge from data streams.

1.4 Contributions of the thesis

As mentioned before, one of the biggest challenges in deep online classification training is to avoid forgetting the already acquired knowledge. A straightforward solution to this problem is rehearsal ([Rob95]) that consists in storing all (or at least a significant part of) the historical data together with newly arriving stream data and retraining the classifier on all the available information every time the model has to be updated. However, if the system is required to learn on high-speed data streams and keep acquiring new knowledge for an extended period, or has to perform life-long learning, such an approach fails to scale to the size of the problem. To avoid the difficulties related to historical data storage, we take the line of research that focuses on using generative models to approximate the statistical distribution of the source ([PKP+18, SLKK17, KGL17]).

In the **first contribution** of this work (Chapter 5), we address the incremental classification problem, where classes appear in the stream one by one, and a particular type of continuous stream with several classes being available at each time interval. In the proposed approach, we employ Deep convolutional Generative Adversarial Networks (DCGANs [RMC15]) to make up for the absent data classes. We initialize a separate DCGAN for each newly appearing class. We then train them during the stream phase when corresponding data are available to produce images that are visually close to the input source. Each time a data class disappears from the input stream, we use the previously learned GANs to generate new samples for the missing categories. This procedure allows us to ensure that all classes (past and present) are equally balanced in the current learning batch. The proposed approaches for incremental and continuous cases resulted in papers [BBCF17] and [BBCF18] correspondingly.

While this approach has merits and works reasonably well on many databases, the use of GANs has some drawbacks. First, the quality of the generated samples is very class-dependent as the internal structure of the objects influences the learning process. Second, training GANs is an unstable process, especially for data of high complexity. It may affect the quality of the generated samples when new

real examples of a class arrive. Finally, GANs are extremely slow to train. In the incremental learning settings where one can learn each new class until convergence, slow training is not an issue. However, such behavior makes the proposed approach fail to scale to more complex scenarios where real-time training is required.

The **second contribution** of the thesis (Chapter 6) presents a more general approach that can work with more complex continuous data streams with no specific order in class appearance and is scalable to a much larger pool of available classes. The proposed learning framework uses auto-encoders for long-term memory purposes to overpass the previously described limitations of GANs. The auto-encoders are significantly faster to train than GANs. At the same time, using auto-encoders allows us to train a single model to learn all the historical classes. This significantly reduces the memory requirement of the learning system. To further improve the performance of the proposed approach, we enhance the auto-encoders training procedure by adding a supplementary loss function that measures the classifier's performance on produced samples. Moreover, to avoid the drift in the distribution of samples reconstructed by the auto-encoders when the latter is trained on its own reconstructions, we propose a technique of adaptive gradient weighting. This method modifies the influence of the synthetic samples depending on the number of full encoding-decoding passes they went through.

We validate the proposed method on the continuous streams simulated from static natural image datasets and a large-scale synthetic dataset designed for this study. We demonstrate that the proposed approach is fast, scalable to large problems with thousands of classes, and has a stable training process. The described work was submitted to the Computer Vision and Image Understanding journal.

The rest of the thesis is organized as follows. In Chapter 2, we describe the theoretical background of online learning and give detailed information about the experimental scenarios we address. Chapter 3 provides a brief discussion on the DL principles. We explain how the standard learning procedure in Neural Networks leads to catastrophic forgetting. In Chapter 4, we review existing methods that address the problem of online learning on data streams and position our proposals with respect to these. We give a detailed presentation of the most relevant proposals in three types of methods for online learning: dynamic architectures, regularization, and dual-memory based approaches, and

highlight their main advantages and limitations. In Chapter 5, we describe our first proposal that uses GANs to preserve and replay historical knowledge. We demonstrate that the proposed method can efficiently perform incremental learning by validating it on two image datasets simulated as incremental data streams. In Chapter 6, we present our second approach, able to perform continual learning on dynamic data streams using auto-encoders. We discuss the advantages of auto-encoders over GANs to replace historical data in such an advanced scenario. We describe our learning framework, assess its performance by an experimental validation on three different datasets, and analyze its behavior. Finally, in Chapter 7, we discuss perspectives and future work.

Chapter 2

Learning from Data Streams

The goal of this thesis is to develop approaches that can learn in a fast and efficient way from data streams with dynamically changing data distributions and multiple factors that can influence training in real-time. To develop such a method, we first need to give a more formal definition of the problem and discuss its currently existing solutions.

In this chapter, we formalize the context of continual learning from data streams. We start by introducing the main definitions of the domain and discussing the challenges that one faces when dealing with it, notably the issues related to the phenomenon of the **concept drift**. We then discuss different types of learning scenarios that fall into the domain of online learning.

2.1 Online learning

Online learning has only recently received serious attention from the ML community. The field is therefore not well established yet, missing agreement in definitions and baselines. As an example, the following recent works [HKCK18], [KMA+17], [LP+17] and [MCH+18] all deal with online learning. A quick look at the learning scenarios and definitions proposed in these papers is enough to see that different authors try to solve relatively different problems. In this section, we propose our definitions of the discussed concepts. These definitions differ, to some extent, from what the reader may find in the literature. However, for the sake of clarity, we hold on to proposed notions for the rest of the thesis.

Let us first introduce the notion of the learning **environment** as the surroundings and conditions in which a learning agent lives or operates. As previously stated, in this work, we mostly deal with supervised learning. Let us, therefore, consider that each environment is partially or entirely annotated by human experts.

Let us introduce a group of agents $(a_i)|_{i=1}^{A}$ dynamically exploring a potentially infinite set of environments $(\mathcal{E}_j)_{j=1}^{\infty}$. Each agent a_i is provided with a set of sensors $(c_k^{(a_i)})|_{k=1}^{C}$ that capture specific representations from the environment (e.g., images, sounds, other sensory information). We denote these representations as data samples

$$s = \{\vec{x}_1, \ldots, \vec{x}_k, \ldots, \vec{x}_C\} = \{c_k^{(a)}(\mathcal{E})\}_{k=1}^{C}$$

where each data feature $\vec{x}_k$ is a specific numerical representation of the environment acquired by the k^{th} sensor of an agent. In the supervised learning context, one or several features provided by the agent have to correspond to environment annotations (training targets). To keep the notations simple and homogeneous, we put the target value (or values) in the last positions of the data sample in the following way:

$$s = \{\vec{x}_1, \ldots, \vec{x}_k, \ldots, \vec{x}_C, \vec{y}\}$$

We denote $\mathcal{E} = \bigcup_{j=1}^{\infty} \mathcal{E}_j$ the complete environment available for the agent population. To link the data and the environment, let us say that data are the perception of the environment by the learning agents.

We then assume that the data acquisition procedure is discrete and that data samples can be rapidly transferred to a centralized learner L with no time overlap between separate transfers. This assumption allows us to introduce the following definition of a Data Stream..

Def. *We call* ***Data Stream*** *the following agent–learner communication procedure:*

$$\mathcal{S} = [s_t = c_{k(t)}^{a_{i(t)}}(\mathcal{E}_{j(t)})]_t|_{t=0}^{\infty}$$

where s_t is a data sample acquired from the environment $\mathcal{E}_j$ by the sensor c_k of the agent a_i and transferred to the learner L at the moment of time t; i, j and k being time-dependent indices.

In real-life applications, a data stream in the provided formulation certainly has one or several properties that make real-time learning very challenging. First of all, depending on the application, data can arrive extremely fast. Second, in contrast to learning from static data, when learning from streams, there is no guarantee that data classes are balanced and that the system does not suffer from dramatic drifts in the underlying data distribution. However, even in the presence of distribution changes and incrementally appearing classes, different data categories can still share internal representations and have temporal dependencies that may be extremely useful for the fast incorporation of new knowledge. Therefore, an efficient online learning system should be provided with the following characteristics:

1. Learn fast from few examples and from unequally represented concepts.
2. Continuously learn for long periods of time on long data sequences, potentially for the life duration (life-long learning).
3. Detect and handle drifts in data distribution and adapt to different types of data.
4. Efficiently use previously acquired knowledge when learning new tasks (knowledge transfer).
5. Adapt to new data concepts never seen before, with the total number of categories unknown in advance and potentially very large.
6. Long-term memory capacities – ability not to forget already acquired knowledge.

To be able to generalize from the training set to the test set, one needs to assume that there is some common structure in the data. The most commonly used set of assumptions are the i.i.d. assumptions. They state that the data is independently and identically distributed: each example is generated independently from the other examples, and each example is drawn from the same distribution P_{data}.

The property of unequally distributed data classes in a given batch of the stream and dynamically changing data concepts is often addressed in the literature ([HKCK18], [LP+17], etc.) as the non-i.i.d. nature of the data in the stream. This implies that the batches (1) are not randomly sampled from the training data (streaming data may have temporal/spatial dependencies due to the continuity of the observed environment) and (2) are sampled from a changing distribution (not identically distributed).

Depending on the type of the stream and the tasks the model has to solve, online learning can be divided into several sub-domains.

Incremental learning is the sub-case of continual learning in which a single agent explores environments one by one, with each environment being locally i.i.d. In the most frequent case, this type of learning consists of introducing new objects to be explored by the system. Such a problem formulation can find its utility in real-life applications where data arrive in a very specific and well-organized order.

Transfer learning is another type of learning which aims to transfer knowledge from one task to another. The transfer learning paradigm usually does not require retaining previously acquired knowledge but instead aims to use this knowledge to accelerate and stabilize the procedure of incorporating the information from the current task.

In this thesis, we do not claim any contribution to the domain of transfer learning. However, we use some transfer learning techniques, such as feature extraction by pre-trained models. We employ them as auxiliary tools to facilitate our experiments.

Continual learning is the learning paradigm that is of the main interest to this thesis. Similarly to incremental learning, data are continuously arriving at the learning system and have to be rapidly processed. However, in continual learning, we do not make any assumption about the proper separation between the tasks and the availability of task descriptors at training time – instead, we consider the full learning procedure as a single task that evolves with time.

Remark on centralized learning. The requirement for centralized learning, mentioned earlier in this section, may produce security and privacy concerns for various applications where only in-place training is allowed. However, we argue that this condition is of high importance in designing a comprehensive life-long learning system. However, one can easily project the proposed definitions onto the case of a pre-trained system that has to be locally fine-tuned. To do so, one can provide a single agent with a pre-trained learner, updated exclusively on locally available data. Such a system

usually has to fit a specific application in a limited environment; therefore, learning is relaxed on several requirements that we impose in the case of life-long learning. We, therefore, assume that this scenario is of no particular interest to this thesis.

2.2 Concept drift

In most real-life applications, data distribution is changing over time. This phenomenon is usually referred in the literature as **concept drift**. The notion of concept drift describes the characteristics of a data stream. In this section, based on the review papers ([WHC+16], [GŽB+14]), we provide a brief overview of the existing ways to detect and handle concept drifts in data streams. We believe that this introduction can help the reader better understand the type of data and the complexity of the task we aim to solve in this thesis.

Multiple factors can cause changes in the data distribution. To demonstrate it, let us consider the following scenarios:

1. A learning agent is pre-trained on a base environment containing data classes $1,\ldots,n$, and is sent in its original configuration, with no modifications in its receptors and data pre-processing mechanisms, to explore new environments and learn new classes $n+1,\ldots,n+m$.
2. The same learning agent, instead of being sent out, is kept training on the base environment, but the learning conditions are changed, e.g. it is now learning from the images captured by different sensors, in a much lower level of light (night mode) or in the presence of other artifacts that introduce noise.
3. The data environment stays exactly the same, but the learning task is changed, e.g. the original task of detecting and classifying animals on a farm switches to distinguishing between indoor and outdoor images.

All three described situations can be challenging to handle by a learning system. To understand the potential difficulties one can have when dealing with those situations, let us formalize the problem. Let us consider a scenario of supervised learning from the stream (see Sec. 2.1), where the learning

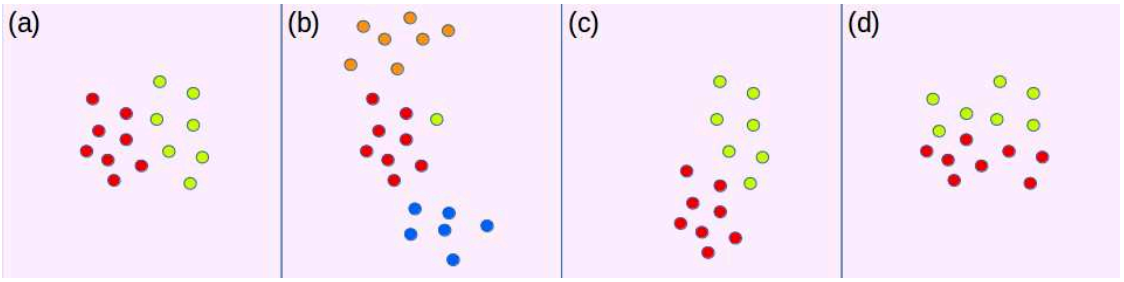


Figure 2.1: Representation of different possible changes in data distribution. (a) Initial distribution with two classes; (b) $p(y)$ is changed, the green class almost disappeared, two new classes are added; (c) changes in $p(X|y)$ (virtual concept drift); (d) data set stays the same, but the label distribution $p(y|X)$ changes (real concept drift)

system continuously receives data samples $s = (X_t, y_t)$ from the environment. We also consider that data emitted by the source at time t follow the joint input-output distribution $p_t(X, y)$. We then say that the data stream suffered from **concept drift** in the time period $[t_0, t_1]$ if

$$p_{t_0}(X, y) \neq p_{t_1}(X, y)$$

With these notations, the previously discussed situations can be seen as follows (see Fig. 2.1 for an illustration):

1. Changes in the prior probabilities $p(y)$ (Fig. 2.1 [b]).
2. Changes in class conditional probabilities $p(X|y)$ – *virtual* concept drift (Fig. 2.1 [c]).
3. Changes in posterior probabilities $p(y|X)$ – *real* concept drift (Fig. 2.1 [d]).

In standard learning paradigms often discussed in the DL literature, dealing with the first two types of drift can be seen as improving the generalization capacities of the learning model. Indeed, in both cases, the drifts in data distribution are most probably temporal, with many possible variations of those changes. Therefore the main objective of learning on such a stream is to create a general model able to perform its main task independently from the changes in the data distribution. One of the most significant issues that can arise when dealing with such drifts is **catastrophic forgetting** – the core problem this thesis is addressing, that is discussed in detail in the next sections. In contrast, *real concept drift* mostly reflects the permanent changes in the learning system objectives aiming to adapt

to a relatively new task in the fastest and most efficient way – a problem often referred to as transfer learning.

From the other point of view, drifts in prior probabilities $p(y)$ mostly represent the global changes in the modeled phenomenon rather than changes in the data distribution. Therefore, only the types of data evolution described at points (2) and (3) are usually considered concept drift.

To deal with the previously described changes in streaming data distribution, a learning system must be able to detect concept drift as soon as possible, distinguish it from noise and adapt to it. Besides, one of the main ideas of online learning is that the most recent data should be the most relevant to the model of interest. Therefore the efficient concept drift handling mechanism should not only be able to incorporate new knowledge, but also remove the out-of-date information. Depending on the desired result, one can choose a forgetting method with an adequate trade-off between its reactivity and robustness to noise. Ideally, the system is forced to forget already trained parts of the model if and only if it detects a drift in data distribution with certainty that this drift is not due to noise.

A very intuitive solution to adapt to changes in data distribution is to introduce a *memory* term to the online learning model. It can be done by adding a sliding window that tells how many most recent data samples one can store in our model and use to verify if the data distribution has recently changed.

There exist many methods to detect concept drifts in data. Generally, these methods characterize and quantify changes in the data distribution at a given time point or during some time interval and can be based on: 1) sequential analysis, 2) control charts, 3) differences between two distributions and 4) heuristics.

Detectors based on sequential analysis. *The Sequential Probability Ratio Test* (SPRT) is based on detecting the potential changes in data distribution at time point $w: \ t_0 \leq w \leq t_1$ by computing the ratio between the probabilities of the data sub-sequence, arriving after w, to come from $P_1 = P_{t_1}$ rather than $P_0 = P_{t_0}$.

$$T^n{}_w = log \frac{P(x_w ... x_n | P_1)}{P(x_w ... x_n | P_0)} = \sum_{i=w}^{n} log \frac{P_1[x_i]}{P_0[x_i]}$$

We say that the change at time point w is detected if $T^n{}_w$ is above a user-defined threshold. Methods developed on this basis include the cumulative sum, Page-Hinkley test and some others.

Detectors based on Statistical Process Control. Let us consider a sequence of examples (X_i, y_i). For each data sample X_i our classification model predicts $\hat{y}_i$ which is either *true* ($\hat{y}_i = y_i$) or *false* ($\hat{y}_i \neq y_i$). For each example we define the probability p_i of detecting a wrong label with the standard deviation $\sigma_i = \sqrt{p_i(1-p_i)/i}$. We also define two parameters p_{min} and σ_{min} that are updated if, for a given sample p_i, $p_i + \sigma_i < p_{min} + sigma_{min}$.

For a newly arriving data sample (X_j, y_j) with p_j and σ_j defined by the current model, we say that the sample is **in-control** if $p_j + \sigma_j < p_{min} + 2\sigma_{min}$, **out-of-control** if $p_j + \sigma_j > p_{min} + 3\sigma_{min}$ and in **warning** state otherwise.

With enough data, we can use **Out-of-Control** examples as Concept drift detectors. One more useful measure that can be defined based on obtained information is the rate of change – the time it takes to pass from **Warning** to **Out-of-Control**.

Monitoring distributions on two different time windows. Methods from this group usually consist of fixing windows on the historical data and new arriving data and introducing a statistical test that has a hypothesis that distributions in two windows are equal. If the hypothesis is rejected, then data drift is declared.

Methods can vary from more reactive to more stable depending on the window size. The main limitation of these approaches is that we have to store the data from the chosen windows. The advantage is usually a much more precise localization of the change point compared to sequential methods.

2.3 Experimental scenarios for online learning

In previous sections, we presented the theoretical background of online learning. However, this theory is not straightforward to apply to real problems. First of all, real massive data streams are very difficult to obtain, process, and especially make reproducible studies on them due to the dynamic nature of the data. To overcome this, the validation of the methods for online learning is usually performed by casting the data from classic static datasets in the form of streams. In this section, we review several existing setups of the experimental validation of online learning, notably the cases proposed in [GMX+13].

The authors of [GMX+13] propose to evaluate forgetting when learning from a sequence of tasks. While in the original paper, the authors only perform experiments on two tasks, we extend the described framework to the series of K distinct tasks. Let us consider a sequence $T = \{T_1, \ldots, T_t, \ldots, T_K\}$ of K tasks, each provided with a corresponding dataset $D_t = \{X_i^t, Y_i^t\}_{i=1\ldots N_t}$, where N_t is the size of the corresponding dataset. The goal of each task is to optimize the parameters θ of the model C that learns to map each input X to the corresponding output Y by minimizing the task-specific loss function $L_t(C_\theta(X), Y)$. The global objective of this multi-task learning is to minimize the loss over the full set of tasks:

$$\min_\theta \sum_{t=1}^{K} L_t(C_\theta(X^t), Y^t).$$

While the described framework is very general and can be adapted to any task sequence, in the discussed work, the authors consider three different types of tasks that allow studying model behavior under different angles.

Input reformatting: permutation tasks. Training dataset $D = \{X_i, Y_i\}$ is the same across all the tasks. For each new task, the input data distribution is changed by applying a task-specific transform F_t while corresponding outputs stay unchanged so that for each task T_t its corresponding dataset can be described as $D_t = \{T_t(X_i), Y_i\}_{i=1...K}$. As an example of such a transform, the authors propose to apply a fixed **pixels permutation** on the input data for each new task. Such a scenario can be naturally related with the applications prone to the *virtual* concept drifts (see Chapter 2.2).

Similar tasks. Similarly to the previously described tasks, here, the input-output domains are similar – the tasks differ from one another by a slight drift in the data distribution. As an example, the authors perform their validation on two product categories in the Amazon reviews dataset ([BDP07]) with the binary sentiment analysis as the primary learning objective.

Dissimilar tasks. Data for each new task are coming from a different distribution, for instance, from a completely different dataset. Learning in this way is natural for humans, who can learn from various sensory inputs in different environments and successfully transfer knowledge across different domains of expertise. To the best of our knowledge, there exists no artificial learning system that proved its capacities of efficient knowledge transfer across entirely dissimilar tasks where the inputs don't share any common characteristics. In the current state of the art, the described problem is referred to as Artificial General Intelligence. It is mostly approached by incrementally adding new modules to the model, each solving a new task, with almost no positive knowledge transfer between the modules. From this point of view, existing approaches do not differ from training one model per task and do not represent any particular interest for our study.

Online classification task. In this study, we address the problem of online learning for data classification. In accordance to the notations introduced in Sec. 2.1, we can formalize the global objective as learning from a long sequence of tasks, where each task corresponds to training in the environment E_j which is a part of the global environment E containing only a limited number of data classes. In a real-life application with a large population of agents, each task would correspond to a separate time interval during which only a specific group of agents are actively learning, or only a small part of the global environment is explored.

Chapter 3

Deep learning background

As we have already mentioned in the introduction, there exist several types of non-DL methods that proved their ability to perform online learning. These methods are, however, not efficient when applied to streams of high-dimensional data (e.g., natural images), which brings us to the necessity to build our learning system based on Neural Networks.

The main technical challenge that we address in this thesis is catastrophic forgetting in NNs. However, going into a more in-depth discussion of this phenomenon, it is crucial to understand its nature and origins. We argue that this is only possible with a fundamental understanding of the Deep Learning machinery. Basic Deep Learning concepts are not the primary objective of this study. However, in this chapter, we provide a brief introduction to the DL background, which helps us to understand the difficulty of training Neural Networks on dynamic data and to get the first idea on how to approach the problem.

Neural networks, the core model of Deep Learning, are originally designed to reproduce in a very simplified way the biological brain functioning. The formal objective of such models is to approximate a function that maps some input vector to the desired output, corresponding to this input. The term "Deep Learning" is related to the deep multi-layer structure of these networks. Each layer is a composition of a simple affine transformation of the output of the previous layer and of a non-linearity. In theory, such a structure is supposed to allow the network to approximate any function of interest. The output layer of the network represents the space of desired outputs, which depends on the application – the actual function the network has to approximate. It can be represented by a single value for the case of binary classification and fault detection, by a vector of probabilities for multi-class classification, or lie in the input space when we want to build a model that recovers corrupted data (autoencoders) or to perform some modification or processing on the input.

3.1 Inference in Neural Networks

Neural Networks in their standard formulation are models built from a sequence of linear layers (affine mappings), each followed by an activation function (a non-linearity). More formally, denoting the

input vector by X, linear layers by l_i and activation functions by σ_i, an n-layers Neural network can be represented as a composition of functions in the following way:

$$y^* = l_0 \circ \sigma_0 \circ l_1 \circ \sigma_1 \circ \cdots \circ l_n(X)$$

Linear layers in these notations are simply the following matrix operation: $l(X) = WX + b$, where the matrix W is usually called neural weights, and b is the vector of bias. Weights and biases form the parameter space of the model. In recent applications where extremely deep models are used on complex high dimensional data, this space can reach the dimension of several billions of parameters. The activation function, in turn, can be any almost everywhere differentiable non-linear function defined over the vector space. However, for reasons like numerical stability and empirically proved effectiveness, most of the currently employed DL models use the following activation functions:

- Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh function: $\sigma(x) = \frac{2}{1+e^{-2x}} - 1$
- Rectified Linear Activation (ReLU): $\sigma(x) = max(0, x)$

3.2 Backpropagation: principles behind NN optimization

As it was previously said, Neural Networks are models having as parameters neural weights W and bias terms b. To simplify the further notations, let us group the neural weights and the bias terms in the single parameter set W. To approximate a function, one needs to find the values of those parameters that map the inputs of the network (X) as close as possible to the desired output (y). The very first step is to define the objective (or loss) function that computes the proximity of the actual output of the network (y^*) and the desired one. The goal of optimization in Deep Learning is to search for a set of parameters W^* that minimizes the loss of the network over the available training data samples:

$$W^* = \text{argmin}_{W \in R^n} \sum J(F(x, W), y),$$

where J is a loss function that evaluates how far the real output is from the desired one.

There exist various approaches to optimize parametric models. The most widely used methods for Neural Networks are derived from the simple Gradient Descent:

$$W_{i+1} = W_i - \frac{\alpha}{N} \sum_{x \in X, y \in Y} \nabla_W J(F(x, W), y),$$

where N is the number of samples in the training set and α is the learning rate. Computing the gradient of high dimensional parametric models is, however, not an easy task.

The most straightforward way to evaluate the gradient of the cost function with respect to the network parameters is to compute its numerical estimation by adding tiny variations separately for each

parameter w_i of the network and verifying how much it influences the output:

$$\frac{dC}{dw_i} = \frac{C(w_i + \epsilon) - C(w_i)}{\epsilon}$$

This method was widely used in the early ages of NN history. However, to perform a single update, such methods require propagating information through the big part of the network for as many times as there are parameters, which is extremely slow and becomes almost unfeasible even for problems that are nowadays considered small and simple.

A more efficient way to compute gradients, that became the gold standard in Neural Network optimization, is gradient backpropagation ([WH86]). The method is an application of the classic differentiation "chain rule" and consists of an analytic computation of the derivative of the objective function over each layer of the network. As it was already mentioned, all the layers of Neural Networks are almost everywhere differentiable. Moreover, the widely used types of layers are designed in such a way that each layer i of the network can be easily derived over the layer $i-1$ with the analytic derivative represented by either a matrix multiplication or, in the case of activation layers, by the element-wise vector operations. In this way, once the loss function is used to compute the error of the network on a given input-output pair, this error can be "back-propagated" through the full network in a single pass, thus estimating the gradient of the loss over the full set of parameters.

3.3 Optimization in Neural Networks

In the previous section, we discussed the mechanism of gradient descent applied to NN optimization. However, we only mentioned how the optimization is supposed to work to correctly map a given sample. Deep learning problems usually consist of optimizing the model to learn huge datasets of up to thousands of different classes, each represented by millions of data samples. The global objective, in this case, is usually to approximate a function that produces the smallest possible error on the available training data, rather than just fitting a given data sample. Ideally, one could first compute the error on the totality of the available training data and only then back-propagate it and update the parameters. However, such a procedure is extremely slow.

On the other hand, one can try to compute the error and update the network sequentially on each separate data sample. This way of performing optimization is known to have a very low generalization. Due to the high-dimensional parametrization of Neural Networks, consecutive data samples can have very distant local minima provoking high oscillations in the computed loss and thus resulting in gradient descent divergence.

It was empirically demonstrated that what works best is performing the parameter updates on mini-batches of randomly selected data samples. Indeed, such a procedure averages the error over a small batch of data samples, potentially significantly different. This provides a better approximation of the direction of global minima and smooths the optimization trajectory.

One of the possible ways to further increase the smoothness of the learning trajectory is to provide

the gradient descent with an additional term – the momentum ([SMDH13]). In this case, the network parameters are updated in the following way:

$$v_{t+1} = \mu v_t - \epsilon \nabla f(\theta_t)$$

$$\theta_{t+1} = \theta_t + v_{t+1},$$

where θ_t are the model parameters at step t, μ and ϵ are the training hyper-parameters, usually called momentum coefficient and learning rate. As can be seen from the formula, the momentum term is accounting for the recent history of parameter updates, thus, to some extent protecting the gradient from dramatically changing directions. The idea of smoothing the gradient by accounting for its history was pushed even further in the method called Adam ([KB14]), which also adapts the learning rate separately for each layer of the network. Due to its numerical stability and fast convergence, Adam is nowadays one of the most widely employed optimization techniques. We, therefore, decided to use it in our experiments.

3.4 Importance of the initialization in neural networks

Until recently, Neural Network parameters were often randomly initialized from the normal distribution. Due to the often occurring divergence during the early training stages, very deep NNs initialized in this way were considered extremely difficult to train until [HOT06] proposed to initialize deep networks from pre-trained blocks. While this technique slightly improved the situation, training in such a way required significant computational resources and still had stability issues.

The problems mentioned above are often characterized by the exploding or vanishing values in the activations of the network. The reason for that is the nature of the basic operations of the neural networks – matrix multiplications. For the simplicity of demonstration, let us for now consider neural networks with no activation functions and no bias terms. In this case, each activation $a_j^{(l)}$ of a layer l of the neural network is a weighted sum of the activations from the previous layer:

$$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)}$$

Assuming that both weights and input data ($x = a^{(0)}$) have zero mean (normalizing data to have zero mean is a standard procedure), the expected value of the activations at all the layers will also be zero:

$$E[a_j^{(l)}] = N^{(l-1)} E[w_{ij}^{(l)}] E[a_i^{(l-1)}] = 0$$

The variance of the activations, in this case, depends on the distribution of network parameters, input activations and, importantly, on the size of the previous layer:

$$
\begin{aligned}
Var(a_j^{(l)}) &= E[(\sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)})^2] \underset{s\neq t: E[w_{sj}w_{tj}a_s a_t]=0}{=} \\
&= E[\sum_{i=1}^{N^{(l-1)}} (w_{ij}^{(l)})^2 (a_i^{(l-1)})^2] = N^{(l-1)} Var(w^{(l)}) Var(a^{(l-1)})
\end{aligned}
$$

Therefore, if all the parameters of the network are initialized from the normal distribution ($Var(w^{(l)}) = 1$), then the standard deviation of the activation will be scaled by $\sqrt{N^{(l-1)}}$ at each new layer which will rapidly result in exploding activation values. Initializing the weights with lower variance can result in a completely inverse "vanishing" effect with the network activations having a lower deviation from one layer to the next. At the same time, when performing the backpropagation, the variance of the gradient of the parameters between layers $l-1$ and l depends on the size of the layer l instead of $l-1$, as it was for the forward pass. It is therefore clear that one should initialize the weights separately for each layer with the variation that depends on the sizes of the layers $l-1$ and l.

While the mechanism inevitably changes when one adds the non-linear activation functions at each layer, the previous logic holds. However, in addition to the layer size dependency, one now also has to adapt the initialization to the type of activation functions. In [GB10] the authors propose the **Xavier** initialization method that takes into account all the discussed points for the case of hyperbolic tangent and softsign activation functions:

$$
W^{(l)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{N^{(l-1)}+N^{(l)}}}, \frac{\sqrt{6}}{\sqrt{N^{(l-1)}+N^{(l)}}}\right]
$$

To train the networks with ReLU activations, the authors of [HZRS15] propose the following **Kaiming** initialization:

$$
W^{(l)} \sim \mathcal{N}(0,1) \cdot \sqrt{\frac{2}{N^{(l-1)}}}
$$

According to the mathematical evidence presented in the paper, in the case of ReLU activation function initialization requires only one parameter of the layer size (input or output of the layer) to ensure stable training with no value explosion.

3.5 Generalizing to unseen data

Over-fitting is a significant problem in ML in general, and in DL in particular. It naturally arises from the way learning is performed. The NN's optimization fits the data in the training set, aiming to build the precise input-output mappings. In the static learning case, the problem usually comes from the extremely high dimension and variability of the input data. Indeed, Neural Networks can be seen as the statistical approximations of the input data. When the data are high dimensional

and complex, building the statistical approximations requires enormous amounts of training samples. When the available data are not enough to describe the full variability of the data distribution, NNs fit the available samples, which often results in sub-optimal performance on unseen data from the test split.

Several regularization techniques are frequently used in the static learning setup to reduce overfitting and thus improve the network's capacity to generalize. These techniques include Dropout ([SHK+14], that selects random network pathways to train for each new batch, thus not overfitting the whole network but only small parts of it), L1 and L2 regularization (avoid moving far from the previous optimum when training on new tasks by introducing a supplementary penalty), early stopping (stop training when the performance on the validation set starts decreasing, which means that the model is overfitting the training set), and data augmentation.

Over-fitting is the main reason why Deep Learning is challenging to apply to continual learning – it is closely related to the phenomenon of catastrophic forgetting. When training on a sub-population/sub-environment, network parameters are optimized to fit those sub-populations. This procedure finds the local optimum for the network parameters, which often does not generalize to the historical sub-populations. In this work, we aim to build a system that, while being able to match the new data classes, keeps the ability to generalize on the temporarily unseen data. Standard regularization techniques, while sometimes slowing down the forgetting effect, are not able to solve the issue.

Chapter 4

Alleviating catastrophic forgetting in Neural Networks

In the context of life-long learning, one of the main requirements for a machine learning system is the ability to avoid forgetting the already acquired knowledge. Any such system is supposed to be able to learn in an evolving environment with no assumption on the i.i.d. nature of the data. While being very efficient and easy to train on static datasets, Neural Networks tend to overwrite already learned concepts that disappear from the training set. This phenomenon is known as *catastrophic forgetting* and represents a significant challenge undergoing active research in the machine learning community. It is one of the reasons why neural networks are almost exclusively used for cases where the full dataset is available at any point in time. Indeed, to be able to incorporate the information about all the data classes/concepts/types, the vanilla neural network needs to receive information about all of those during the whole duration of training.

In this chapter, we describe situations in which catastrophic forgetting usually appears and discuss state of the art methods to deal with it depending on applications, datasets, training scenarios, and network architectures. A very naive solution to train DNN in applications with evolving datasets, appearing concepts, etc. is to keep all the historical data while acquiring the new data. In this case, at the moment, when we need to update the model, we re-train it from scratch on the entire dataset. The drawbacks of such a method in the context of life-long learning are as straightforward as the method itself. While the dataset grows with time and is joined to the historical data, the cost of re-training the model grows with time. At some point, we are not able anymore to catch up for new arriving data since training becomes too expensive.

To summarize, a system aimed at life-long learning should be able to incorporate knowledge in real-time from a potentially fast-changing environment. At the same time, it should preserve the already acquired knowledge even if it doesn't receive any recall of it for a long time.

Humans, while tending to forget concepts through their life-span, rarely have new knowledge that interferes with the acquired information. This capacity of the human brain to acquire new information while preserving the old memories is usually referred to as brain plasticity.

The paper [PKP+18] provides an up-to-date review on continual life-long learning and gives a complete overview of the biological inspiration of neural network-based systems. The authors summarize the main reasons for forgetting and existing approaches to handle this issue in neural networks. They define a life-long learning system as 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 learn (e.g., number of classes for classification) is not predefined.

In this chapter, we extend the discussion from the review paper mentioned above and demonstrate the main advantages and limitations of existing methods that aim to alleviate catastrophic forgetting in continual learning. We discuss the three types of existing solutions that aim to avoid as much as possible the loss of the historical knowledge – methods based on training regularization, evolving neural architectures, and systems based on rehearsal or pseudo-rehearsal. We conclude the chapter by the comparative overview of these methods and a discussion on the advantages and limitations of each type of method.

4.1 Regularization-based approaches

As was already mentioned, in the domain of Deep Learning, the term "regularization" is usually employed for imposing special constraints that control the way neural weights are updated. In most cases, it is done to avoid overfitting and therefore generalize training (see Section 3.5 for a detailed explanation). However, regularization techniques can also provide a way to perform selective task-specific network retraining, which allows sequential training on a series of tasks with no or limited forgetting. Regularization-based methods are often biologically inspired, suggesting that the mammalian brain has specific mechanisms to consolidate knowledge and regulate the plasticity of its synapses.

Inspired by the competitive behavior of neurons in the brain, in [SMK+13], the authors propose to replace standard non-linearities in NN training by a Local Winner Take All (**LWTA**) mechanism. In the proposed approach, the network is organized into blocks. In each block at each training step, only the strongest activation is kept active and passed to further layers, while all the other activations are set to zero (Fig. 4.1). Back-propagation is then performed over non-zero activations.

LWTA network was initially designed as a new type of "non-linearity" supposed to improve the performance in offline training settings. However, incremental learning experiments on MNIST-split demonstrated that compared to other often used non-linearities, such as ReLU or sigmoid, the proposed method significantly reduced catastrophic forgetting. LWTA non-linearity reduced the error on the test set for the first half of the split after retraining on the second half from 57.84% (sigmoid) and 16.63% (ReLU) to 6.12%. The error is still relatively large compared to results obtained with modern methods designed to alleviate catastrophic forgetting. However, the LWTA network can be considered the first successful attempt to provide NNs with long-term memory based on training regularization without any computational and memory overhead over vanilla SGD.

One of the first regularization-based techniques explicitly designed to alleviate catastrophic forgetting was introduced in [LH17]. In the proposed method, the model M_Θ is sub-divided into three sub-blocks: $M_{\{\theta_s\}}$ – part of the model with parameters θ_s shared across all tasks; $M_{\{\theta_o\}}$ – task-specific blocks for

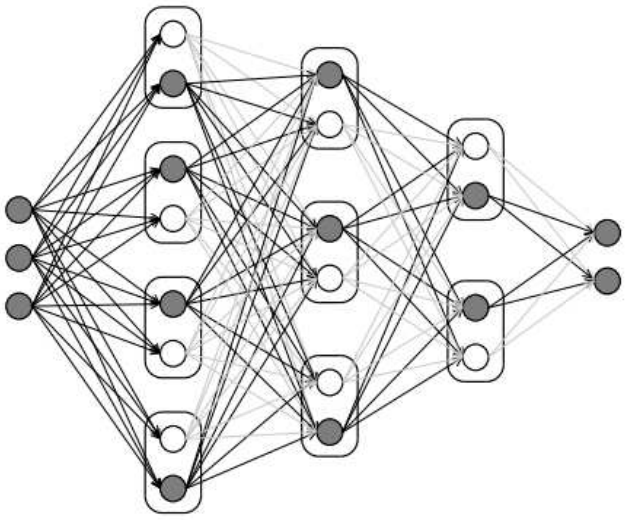


Figure 4.1: Local Winner Takes All (LWTA) mechanism in Neural Network's updates, Fig. 1 from [SMK+13] (Sec. 3). Only the most active neurons (dark gray) propagate information and get updates for a given training batch.

already learned tasks that are usually not modified during training on the new task; and $M_{\{\theta_n\}}$ – task-specific block for the new task.

In the absence of an established evaluation baseline, the authors compare their method to fine-tuning, feature extraction, and joint training on all tasks simultaneously. Using previously provided notations, feature extraction can be seen as fitting $M_{\{\theta_n\}}$ to new task while freezing all the other parameters, and fine-tuning as optimizing $M_{\{\theta_s,\theta_n\}}$ with former having a slower learning rate to prevent big drift on old tasks.

In contrast to fine-tuning and feature extraction, the proposed method optimizes the full model $M_{\{\theta_s,\theta_o,\theta_n\}}$. To avoid catastrophic forgetting, the authors propose the following procedure. For each new task, the sub-model $M_{\{\theta_n\}}$ is optimized on it until convergence. The outputs of the old task solvers on new data $\hat{Y}_o = M_{\{\theta_s,\theta_o\}}(X_n)$ are then computed and used to regularize re-training of $M_{\{\theta_s,\theta_n\}}$. This is done to avoid changes in θ_s that may result in drastic changes for the old task inference (see Fig. 4.2 for details). The proposed training mechanism does not require storing and reusing historical data. It demonstrated fast adaptation to new tasks, over-performing feature extraction, and fine-tuning with only a small computational overhead over the latter. Moreover, it resulted in low forgetting on old tasks.

The big limitation of **LwF** is its computational complexity that grows linearly with the number of tasks. Indeed, for each new batch from the new task, one should perform a forward pass through $(M_{\{\theta_s,\theta_o\}})$ and backward pass through θ_o for every old task before and after the update, and only then full network update can be performed. Unfortunately, the authors only conducted experiments on a short sequence of tasks (4), which does not demonstrate the scalability of the method. Another critical

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

Figure 4.2: Schematic representation of LwF.

limitation of **LwF** is the necessity of storing task-specific output layers for all learned tasks, which makes it challenging to apply to continual learning scenarios where tasks are not strictly separated but are rather continuously evolving. Moreover, to choose the corresponding output layer, the proposed method requires test-time knowledge of the task it is currently solving, and therefore testing cannot be considered entirely unsupervised. This limitation is widespread among incremental learning algorithms discussed in this section. In further discussion, we refer to it as **task-dependent testing**.

The straightforward way to overcome the computational limitations of LwF is to train a single model with all the parameters shared across the tasks, supplied with a mechanism that provides task-specific scheduling based on weighting the importance of each parameter for each specific task.

One of the methods based on this idea, Elastic Weight Consolidation (EWC), was proposed in [KPR+17]. The method is inspired by synaptic consolidation in the mammalian brain – a mechanism that limits the elasticity of small groups of synapses to protect long-term task-specific memories. The proposed framework aims to force the parameters that are important for old tasks to stay close to their historical values supporting long-term memory.

The authors argue that due to the high data dimensionality and rich parameter space in NN-based approaches, for a given problem A, there exists a large variety of local minima with similar close-to-optimal solutions. Therefore, when passing to a new task B, one would ideally want to find a solution θ_B in the close environment of optimal solutions for the task A. Intuitively, one can decide to use L2 regularization to forbid high deviations from θ_A. However, it can result in sub-optimal solutions for both old and new tasks (see Fig. 4.3 for an illustration).

The algorithm consists in introducing the weighted quadratic penalty on the distance between current parameters and optimal parameters for historical tasks. For the new task B and a sequence of old tasks A_i the authors propose to use the following loss term:

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

where F is the diagonal of the Fisher information matrix.

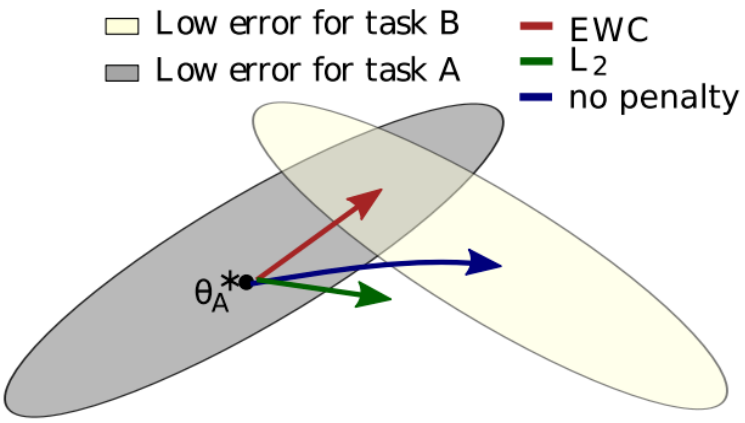


Figure 4.3: Principles behind the regularization by Elastic Weight Consolidation.

Experimental validation provided in the paper demonstrated that in contrast to L2 regularization, EWC could efficiently handle forgetting. However, compared to fine-tuning, EWC achieved sub-optimal solutions on new tasks showing the method's tendency to prioritize stability over plasticity. The proposed method requires storing optimal parameter sets $\theta_{A_i}*$ for every previously learned tasks A_i. Contrary to LwF, stored historical models are only used to compute the parameters' importance estimations at the end of each task, providing lower computational overhead. However, EWC needs diagonal weighting over the parameters of learned tasks, which can only be performed offline, and requires task-dependent testing. All the described limitations make this method not applicable to continual learning problems.

To achieve a more online way to perform neuron-wise regularization, the authors of [ZPG17] propose to explicitly study the internal dynamics of each synapse that forms the network. The work is inspired by a biological model of neurons: in contrast with artificial neural networks, each neuron in a biological brain is complex machinery rather than a single value. In the proposed algorithm, in addition to its actual weight, each network connection stores its local importance measures for every already learned task. Defining synapses in this way allows penalizing the modification of important parameters when training on new tasks. In practice, the importance of each synapse θ_k for a task γ is evaluated based on two values:

- How much this parameter contributed to the loss drop on task t^γ during the full training phase:

$$\omega_k^t = \sum_{(t-1,\dots,t)} \left(\frac{\partial L}{\partial \theta_k} \cdot \frac{\partial \theta_k}{\partial t}\right)$$

- How far it moved from $t^{\gamma-1}$ to t^γ:

$$\theta_k(t^\gamma) - \theta_k(t^{\gamma-1})$$

To take into account the discussed properties, the authors add an approximation of the sum of losses for previous tasks to the loss on the current task (L_μ):

$$L'_\mu = L_\mu + c \sum_{k<\mu} \Omega_k^\mu (\theta'_k - \theta_k)$$

where θ'_k is the parameter value at the end of the previous task, and Ω is the normalized sum of ω for all the previous tasks.

The proposed method showed good performance in the settings of incremental learning and demonstrated an excellent capacity to learn new tasks on a few data rapidly. Moreover, the proposed Synaptic Intelligence (**SI**) framework estimates the importance of each parameter during training, allowing to perform online learning.

Similarly to **SI**, the authors of [ABE+18] propose a learning framework, Memory Aware Synapses (**MAS**), able to select and freeze important neural connections to be kept as long-term memory. In practice, **MAS** differs from **SI** only in the way the importance coefficients Ω_{ij} are defined and computed:

$$\Omega_{ij} = \frac{1}{N} \sum_{k=1}^{N} ||g_{ij}(x_k)||$$

where

$$g_{ij}(x_k) = \frac{\partial [l_2^2(F(x_k;\theta))]}{\partial \theta_{ij}}$$

For a given task, the importance of a given parameter is accumulated from the norms of its gradients on already learned data samples. This estimates how the parameter variations change the output of the model for a given data sample. In these settings, with $\theta_{ij}*$ – the network parameters at the end of the previous task, training loss for task n is defined as follows:

$$L(\theta) = L_n(\theta) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta_{ij}*)$$

When training on the new task, the Ω from the end of the previous task is used. It is updated at the end of training on the given task, thus requiring a short "sleep" phase to update the importance weights. Experimental results demonstrated the method's capacity to alleviate catastrophic forgetting with no need to store and reuse historical data. However, a supplementary forward-backward pass on data is required at the end of each task in order to update Ω. This introduces the need to reuse parts of data from the latest task and thus disconnects the proposed framework from online learning settings. Besides, all the described methods that apply regularization on a single neuron level require a separate output layer for each performed task and thus suffer from the task-dependent testing.

Another way to handle forgetting is to allocate network parts for separated tasks dynamically. In [FBB+17], the authors propose **PathNet**, a method that aims to control the information flow in a single network by creating task-specific pathway bands inside the network. They argue that training a huge multi-task model instead of one model per task allows sharing information and internal repre-

sentations between tasks. The method is inspired by a hypothesis on evolutionary algorithms in the brain.

The proposed model is represented by a single network that consists of L modular layers, with M modules each. Each module is itself a neural network with a non-linear activation function on top of it. The outputs of the modules of each layer are summed and passed to the active modules of the following layer. On each layer, only N modules are active at a time. A module is active if it belongs to the learning pathway chosen for a given training stage. The output layer is unique for each task.

For the evolutionary part, the authors initialize P genotypes (or learning agents) that are represented by pathways through the modules of the network. Each genotype is at most a $N \times L$ matrix. For a single task, two pathways of the network are trained separately one after another, the copy of the winning genotype overwrites the losing one, and the winning genotype is mutated by adding an integer from the range $[-2,2]$ to each of its weights with the probability $1/(N \times L)$.

After the task is learned, modified model weights are frozen, and the rest of the network is reinitialized. For the new task, the new set of genotypes is initialized, and the network is trained as previously.

Experiments showed strong capacities of the proposed method to transfer knowledge between tasks and thus accelerate learning. For each task, only a sub-network of fixed size is used. Therefore the algorithm's computation speed is independent of the full network size. However, in the described setup, each task is learned until convergence, which is impossible in continual learning. Moreover, the model is limited to task-dependent testing.

In [SSMK18], the authors introduce the hard attention mechanism (**HAT**) that aims to perform training regularization. The authors consider the case of performing different tasks on the same data, e.g., classify cats vs. dogs and classify outdoors vs. indoor images from the same dataset (including images of cats and dogs in different environments). When learning this kind of tasks sequentially, the internal differentiating descriptors between two tasks can significantly differ, resulting in catastrophic forgetting, which motivated the authors to introduce a learning mechanism with task-specific scheduling. In the described work, this mechanism is given in the form of the task description embedding, provided to the model as an additional input.

Using the embeddings of the task descriptions the authors introduce their attention mechanism – real-valued gating masks that are applied to perform selective back-propagation. When training for a task t, attention weights are accumulated from all the historical tasks: $a_l^{\leq t} = max(a_l^t, a_l^{\leq t-1})$, where a_l^t is the mask for layer l and task t. During back-propagation, the gradient is weighted element-wise in the following way:

$$g'_{l,ij} = [1 - min(a^t_{l,i}, a^t_{l-1,i})]g_{l,ij}$$

The proposed approach can be seen as a version of **PathNet** on the single-neuron level instead of modules. It has no computational overhead compared to vanilla SGD while showing a competitive performance compared to state of the art incremental learning methods such as **EWC**, **LwF**, **PathNet** etc. However, task embeddings are defined before training and are not learned. Thus, HAT does not take into account the similarity between tasks (if any) and therefore promotes network sparsity that

limits knowledge transfer between tasks. Also, training until convergence and task-specific testing make this approach not applicable in the continual learning setting.

In general, the regularization-based approaches help to efficiently alleviate catastrophic forgetting in particular setups where tasks are well separated, and task environment descriptions are given at test time. Besides, the described methods introduce a very sensitive trade-off between knowledge consolidation and learning something new.

4.2 Evolving neural architectures

Methods based on task-specific regularization of network updates allow to limit the effect of catastrophic forgetting in specific experimental settings and perform well when tasks are similar. But in real-life applications, the learning agent is often exposed to complex problems with various tasks and the rapidly growing complexity and variety of data concepts. Thus, when learning from streams, one would ideally want the learning system to incrementally augment its computational and memory capacities when already allocated resources are insufficient to capture the full richness of the data. Moreover, newly introduced connections usually have high sensitivity to data providing large gradient, thus privileging the "injection" of new knowledge into newly introduced regions of the network and potentially limiting the catastrophic forgetting. On the other hand, uncontrollable model growth can result in over-parametrized models that tend to overfit training data and are slow to train. In this section, we discuss several methods that aim to address these issues.

In [RRD+16], the authors propose a progressively growing model based on the transfer learning paradigm in which, in contrast to standard methods, training on new tasks doesn't introduce catastrophic forgetting of the old ones. The model starts with a single multi-layer Neural Network (column), which is optimized for the first task. For every following task, a new column of the same architecture is initialized. While the new column performs a forward pass on the new task data, it also receives corresponding responses from the previously learned sub-models through lateral connections. In this way, each layer i of the new network receives the activations of the layer $i-1$ on the same input from all the previous models (see Fig. 4.4). Parameters are only updated for the new column and its lateral connections, while previous sub-models stay frozen. At testing time, to solve a given task, one has to choose the corresponding model from the full sequence of sub-models.

The proposed approach showed state-of-the-art performance on several sequences of reinforcement learning tasks and proved to be able to alleviate catastrophic forgetting in the incremental learning scenarios. However, this is done at the expense of linearly growing model complexity and memory requirements with the growing number of tasks. The progressive network's learning procedure can be adapted to handle incremental classification learning, where each new task consists in adding new data classes to the training set. However, by its design and requirements to know which task it is solving, the framework is impossible to apply for the continual classification learning from streams where the data environment is rapidly changing.

One of the first consistent attempts to perform a controlled adaptation of the model architecture to the growing complexity of streamed data was made by [ZSL12]. In contrast to the Progressive Neural

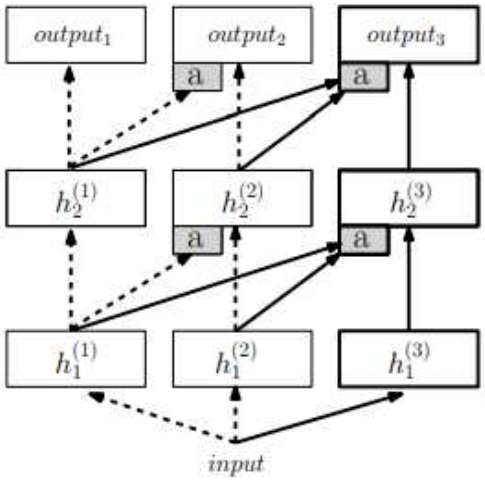


Figure 4.4: Schematic representation of Progressive Networks approach (Fig. 1 from [RRD+16]).

Networks, where the model is growing linearly with the number of tasks, the proposed framework aims to increase model capacities only when already allocated resources are insufficient to handle the growing data complexity.

The authors perform their study on Denoising Autoencoders (DAE) – NN-based models that have as a primary goal learning to recover corrupted data (see more details in Sec. 4.3.2). The study aims to accelerate DAE training and improve reconstruction quality by adapting the model architecture when learning from dynamic data. To perform the discriminative task, the authors propose to attach a classification layer on the top of the encoded features and train the model using a mix of classification and reconstruction losses.

During training, the learning system initializes a buffer B, which collects training samples with the highest training error. Once the buffer is full, the neurons with the most similar activations are merged, and a population of randomly initialized neurons with corresponding connections is added. The model is then optimized on B by freezing all the old connections and only updating newly introduced nodes. This procedure aims to incorporate new knowledge while protecting the historical expertise from being overwritten. However, since the inference in Neural Networks is made through all the learned connections, such a procedure only keeps the internal representations with no guarantees to preserve the correct outputs for the old tasks.

In [YYLH18] the authors propose a more advanced way to adapt network architecture to learning from data streams – Dynamically Extendable Networks (DEN). The introduced framework tackles the problem of incremental learning in life-long learning settings, where a potentially infinite set of tasks $\{1,\ldots,t,\ldots,T\}$ appear sequentially, each task provided with a corresponding dataset $D_t = \{x_i, y_i\}_{i=1}^{N_t}$. Importantly, training is done under the assumption that when task t is processed, data from tasks $1,\ldots,t-1$ are not available.

Under these conditions, the primary motivation of the work is, when learning a new task, to make the

maximum use of the knowledge acquired from previous tasks to accelerate learning and, at the same time, to dynamically extend the capacities of the model when the present capacities are exhausted by already acquired knowledge.

The first key point of the method is the use of a L_1 regularization term for model training to promote learning sparsity. This procedure creates a sort of task-dependent pathways in the network. For further training, the authors introduce a process of *selective retraining*. When the new task is introduced, the model's topmost hidden layer is updated to find the first sparse estimator for the current task solver. Once it is found, the neurons that are mostly affected by training are identified, and, based on a predefined threshold, corresponding weights are updated.

The authors argue that in the case of drastic changes in the data environment from one task to another, selective retraining alone is not sufficient because learned representations for previous tasks are not representative of the new task. For these cases, the authors propose a procedure of *Dynamic Network Extension*, which consists in introducing new neurons to the model when its performance on the new task after the selective retraining is sub-optimal. The less active neurons are further removed to avoid uncontrollable model growth.

While performing comparably well under the conditions of separated sequential tasks, the proposed approach still requires storing task-specific output layers for all the historical tasks, which results in task-dependent testing.

Prototype-based input space partitioning. Deep Neural Networks are by design extremely "input-greedy". Indeed, given an N-class classification problem (N categories in the dataset and a classifier with N output nodes), training a classifier on $M < N$ classes will always result in partitioning the input space into M areas, thus not leaving empty space for missing classes (see the detailed discussion in Sec. 5.1). Such a behavior can be a crucial drawback in the continual learning setting where several data classes can be missing during training. One of the ways to avoid catastrophic forgetting in the context of incremental learning is to perform restrictive local partitioning of the input domain prior to / during training.

In [GK16], the authors propose a model based on self-organizing maps (SOM [Koh82]) that create a topologically organized representation of the input space, and a linear regression layer that aims to classify the obtained representations. The SOM performs spatial reorganization by initializing the prototypes in the input data space and associating each node of the hidden layer to one of those prototypes.

The SOM layer is only updated when the classification probabilities are entirely incorrect or ambiguous, thus preventing the topology of the hidden features from drastic changes when no new concepts are added. The regression layer, in turn, is only updated when the hidden layer activations are situated in the proximity of some prototype, thus promoting the correct topological organization for further learning. The authors claim that such an organization is beneficial for incremental learning because the SOM module tends to map yet unknown data concepts to the unoccupied locations in the space of hidden representations, thus not perturbing previously learned prototypes and avoiding catastrophic forgetting.

However, to function correctly, the learning procedure of the described approach has to be regulated by training on historical data. As a part of the approach, the authors introduce a short-term memory unit that stores the most recent and relevant memories and replay them during sleep phases. The algorithm requires a partial replay of all the previously learned data classes, which can be a big limitation for large-scale problems. Nevertheless, under the condition of having access to the historical data and due to the spatial organization of hidden representations, the proposed model proved to learn new data concepts in a fast and efficient way.

Based on a similar logic, in [PL16] the authors perform topological input space reorganization using the Load-Balancing version of the Self Organizing Incremental Neural Network (LB-SOINN [ZXH14]) as a classifier on top of the off-the-shelf pre-trained convolutional feature extractor. In the proposed approach, each classifier node is associated with a cluster center (or a prototype) represented by a feature vector in the input space. Once a new training example is available, based on the distance between the newly added example and the two closest prototypes, the algorithm decides whether a new node has to be added to the network. In this manner, when new concepts appear, the classifier adapts to the input topology while preventing the learning system from catastrophic forgetting.

The original LB-SOINN algorithm doesn't require any supervision. While the authors of [PL16] use it in a supervised fashion, the unsupervised nature of the LB-SOINN provides the possibility of "knowledge gap" detection – it allows to detect previously unseen data concepts and thus reduce the supervision requirement from the human expert.

One of the most significant disadvantages of the proposed method is that, applied to complex high-dimensional data, it may create an extensive set of data prototypes. As discussed above, introducing each new prototype produces extra complexity in the classifier architecture, which can lead to an excessively heavy model structure. Moreover, due to the high intra-class variability, even the prototypes corresponding to a single class can be situated far from each other in the input space. The results provided in the paper show that this can result in weak generalization capacities of the algorithm in the cases where the unseen examples of learned classes are far from the learned prototypes (e.g. the model trained on a dataset including green and red peppers couldn't correctly classify yellow peppers).

The discussed methods can, to some extent, be seen as pseudo-rehearsal (see Sec. 4.3). Instead of storing and re-using old memories, the proposed algorithms search for statistical representations of the data (class prototypes) in the hidden layer space and re-use these feature approximations in further learning to preserve input-space partitioning and thus avoid catastrophic forgetting.

Discussion on the evolving neural architectures. The described approaches demonstrate strong capacities to alleviate catastrophic forgetting and are resource-efficient compared to regularization methods. Primarily, this efficiency is coming from the methods' ability to share representations among tasks, thus allowing knowledge transfer. However, most of the discussed approaches only allow task-dependent testing, which can be a big issue when dealing with online learning on dynamically changing data where full knowledge about the environment can be unavailable.

4.3 Dual-memory based methods

As already mentioned, catastrophic interference in Neural Networks is mainly caused by the non-i.i.d. nature of the streaming data, which comes with unbalanced or even missing data classes in training batches. The problem posed this way can potentially be solved in a very natural and straightforward way by storing full historical data and completing data batches with randomly selected (from the storage) samples of missing classes when required. However, when considering massive streams, such a procedure can be costly in terms of storage, memory, and time one needs to access the data.

In this section we discuss methods that aim either to develop a strategy to store historical data in a smart way by selecting only the most relevant representatives of each class, or to train generative models – statistical approximations of data that allow to perform random sampling and thus regularize training of the main classification model by feeding it with generated data representing historical knowledge. The first sub-group of algorithms is usually designated in the literature as **rehearsal** methods, while the second one is often called **pseudo-rehearsal**. The approaches we develop in this thesis are mainly based on the ideas of pseudo-rehearsal mechanisms. Therefore the methods presented below are of significant importance for this work and are discussed in detail.

4.3.1 Rehearsal-based methods

The paper [Rob95] can be considered the first consistent study that formalized the problem of catastrophic forgetting in Neural Networks and introduced the notions of rehearsal and pseudo-rehearsal. The authors describe and test multiple regimes of rehearsal: straightforward, that stores and reuses only the latest samples; random rehearsal, that has an equal probability to select any sample from the historical data storage at the beginning of the new task; and sweep rehearsal, i.e. a random rehearsal mechanism that re-selects new samples in the beginning of each training epoch.

The described approaches require permanent access to the full historical data, limiting their application to problems of a very small scale. The work is nevertheless of high importance to this state of the art because it pioneered the dual-memory based approaches to continual deep learning.

The use of rehearsal for online learning received new attention and was pushed forward in [RKSL17]. The authors aim to create an incrementally learning system provided with the following properties:

1. Trainable from a stream where examples from different classes occur separately at different times.
2. Supporting real-time testing with good accuracy for all already learned categories.
3. Memory and computationally efficient (memory is bounded or growing very slowly with the number of classes).

The authors initialize a memory buffer that stores a small number of data samples for each already seen class. The streaming sample is automatically added to the corresponding cell in the buffer until the given class is filled. If no more space is available, the sample is only added if its last hidden activation is close enough to the average activation over the samples from the corresponding class.

During online training, examples from the buffer are randomly sampled, mixed with stream data, and fed to the classifier.

To simulate a stream from a static dataset, the authors initialize the order of classes in the stream prior to training. Classes come one by one, each class appears in the stream only once and is cast until the convergence of the system. The performance is evaluated on a subset of the test set containing all the previously seen classes. Instead of using one-hot label vectors for testing, the authors compare a sample output with the average outputs for each class and assign to the testing sample the label from the class with the closest mean. The authors declare that compared to the standard classification procedure applied to incremental training, this way of performing the inference is less sensitive to the changes in the output layer parameters, which helps to limit catastrophic forgetting.

The proposed method allows continuous learning of a potentially unlimited number of data classes. It has a low computational overhead compared to fine-tuning, not depending on the potential number of classes. Outperforming such methods as LwF ([LH17]) and fine-tuning by a considerable margin, iCarl still results in strong forgetting. The authors argue that forgetting can be reduced by increasing the size of the memory buffer. However, the method performance is limited by the amount of information stored in the buffer, therefore iCarl risks to perform poorly on complex datasets where a large amount of historical data is needed to approximate the dataset variability. Similarly, in [HCK18], the authors propose an online clustering-based rehearsal strategy that aims to maximize the efficiency of training, reduce forgetting, and remove redundancy in historical data.

Instead of searching for a strategy to select and store the most representative samples, the authors of [LP+17] propose a mechanism that makes the best use of the available historical data. The method consists in computing the gradient for each stream batch and slightly modifying it to match a selected historical batch. Working on the triplets x_i, t_i, y_i continuously arriving from the stream, with t_i being task descriptors and $(x_i, y_i) \sim P_{t_i}$ data-label pairs, the authors formulate the problem in the following way:

$$\min_{\theta} l(f_\theta(x,t),y)$$

$$\text{subject to } l(f_\theta, M_k) \leq l(f_\theta^{t-1}, M_k) \text{ for all } k < t$$

The modification of the gradient g is found by the following optimization procedure:

$$\min_{\hat{g}} \frac{1}{2} \|g - \hat{g}\|_2^2$$

$$\text{subject to } \langle \hat{g}, g \rangle \geq 0 \text{ for all } k < t$$

The authors assume that the task description is given at the testing time. Therefore the tasks should be clearly separated. The proposed approach cannot be applied to continual learning without imposing restrictions on the order of the tasks. Moreover, when performing gradient correction, the authors make an assumption that memorized samples from past tasks are highly similar to the new tasks. In the case of complex dynamic non-stationary data and not substantial enough storage, this assumption

risks not to hold. The proposed learning mechanism showed excellent capacities to avoid catastrophic forgetting, but has high memory requirements and is not scalable to larger problems.

Compared to the straightforward retraining on the full historical data, rehearsal-based methods solve the problem of excessive storage and extremely slow training. However, as it was already discussed, generalizing to unseen instances requires Neural Networks to be trained on vast amounts of data. Therefore, not large enough rehearsal buffers for historical classes may result in a reduction of the generalization ability for those classes.

Using smart gradient updates can limit the overfitting of classes with insufficient historical data and thus promote generalization. To the best of our knowledge, the only existing well-performing method that provides such a mechanism ([LP+17]) is limited to the incremental learning cases and is only able to solve similar and well-separated tasks. Therefore, we can argue that existing methods based on rehearsal are highly dependent on the size of the buffer and the quality of the data in it, and thus are not scalable to massive streams of complex high-dimensional data.

In addition to what was already said, due to the security and privacy concerns rehearsal based methods may be impossible to apply for various applications (e.g. military or medical) where the access to historical data may be restricted or completely forbidden. This limitation is well formalized by the authors of [VVPL17]. For a given continual learning problem, they consider the presence of two sites that define access to data: the base site S_b and the incremental site S_i, each provided with significant computational resources. S_b possesses the base dataset $D_b = \{(x_l^b, y_l^b), l \in \{1,\dots,n\}\}$ where $x_l^b \in R^d$ and $y_l^b \in \{1,2,\dots,j\}, \forall l$. S_b possesses the incremental dataset $D_i = \{(x_l^i, y_l^i), l \in \{1,\dots,m\}\}$ where $x_l^i \in R^d$ and $y_l^i \in \{j+1, j+2,\dots,c\}, \forall l$ and $y_l^i \notin \{1,2,\dots,j\}, \forall l$. The authors then define **data membrane** – the property that supposes that the support datasets are not intersecting between S_b and S_I and the learning system has no direct deterministic access to S_b once passed to incremental learning. At the same time, they also suppose that no assumption can be made that learning on the base data gives us any knowledge about the incremental data (**domain-agnostic** criterion).

The solution to most of the discussed problems regarding rehearsal is to have an additional model, or memory module, that consolidates knowledge about the previous experience and provides it to the main model in the form of "fake" data samples, importantly without the explicit access to the original data. In literature, such mechanisms are often called **generative models**. Instead of memorizing the exact copies of data samples, such models usually aim to estimate the statistical properties of the data distribution and capture the internal data representation.

We believe that one does not need to be an expert in generative modeling to understand the principles of the pseudo-rehearsal methods discussed in this chapter. However, generative models are the basis of the methods proposed as the contribution of this thesis. We, therefore, find it important to provide an overview of the most popular generative modeling approaches.

4.3.2 Overview of the high-dimensional generative models to approximate the real data distribution

Human intelligence can create. Based on our knowledge and experience, we can write stories, draw objects, create extremely complex objects and concepts, e.g., space shuttle engines, etc. What is the most impressive is that given the same "sketching" idea of the desired (complex enough) output, two separately working individuals (or groups of individuals) rarely end up with identical or even similar creations.

Until recently, transferring this "creation" ability to machines was either based on exploring the statistics of the big datasets and sampling data that would hold the same statistical properties or was purely algorithmic with step-by-step instructions of what to do, where and in which order. These methods, while being widely used, did not allow to generate complex data that would correspond well to the general concept and, at the same time, have a high enough variability.

In this section, we discuss different existing approaches to build **generative models** – the approaches based on learning from observations aiming to approximate the real data distributions and sample random data instances from it. In this formulation and based on the previous discussion, generative models are not aiming to reproduce already seen samples accurately but rather to acquire a "general understanding" of underlying data concepts and create samples based on this knowledge.

Generative models are usually defined as the conditional distribution of data X over some latent variable z: $P_\theta(X|Z=z)$. In this context, parameters θ are to be optimized on data to find the best fitting model. The term "generative model" is also often used for approaches that do not straightforwardly describe the probability distribution in the space of data. Such approaches (e.g., Generative Adversarial Networks) still make it possible to sample random data instances.

One of the most intuitive ways to describe a complex data population is to fit it with **mixture models** (e.g., Gaussian mixture). Mixture models are used to derive the probability density of a population of complex, often multi-modal data. They are usually described as a weighted sum of probability densities:

$$f(x;\psi) = \sum_{i=1}^{P} \pi_i f(x;\theta_i)$$

with $\sum_{i=1}^{P} \pi_i = 1$ denoting the weights of the P components of the model and corresponding to the probability of a given sample to belong to each sub-population, and $\psi = (\pi_1,\ldots,\pi_P,\theta_1,\ldots,\theta_P)$ denoting the full vector of parameters of the model. The parameters ψ are iteratively optimized on the available population using the expectation-maximization (EM) algorithm. As a big limitation, prior distributions of high-dimensional data are extremely difficult to estimate. Therefore, with no additional engineering and dimensionality reduction, mixture models can only be applied to low-dimensional problems.

Recent research showed that Neural Networks could perform generative tasks in a very efficient manner, capturing the variability inside the observed classes of objects and at the same time, generalizing well. One of the first deep generative approaches able to model high-dimensional data is the **Deep Belief Network** (DBN). Though the idea of Belief Networks was first proposed in the early 90s and has been a subject of numerous studies and experiments since then, its first appearance in a Deep Learning

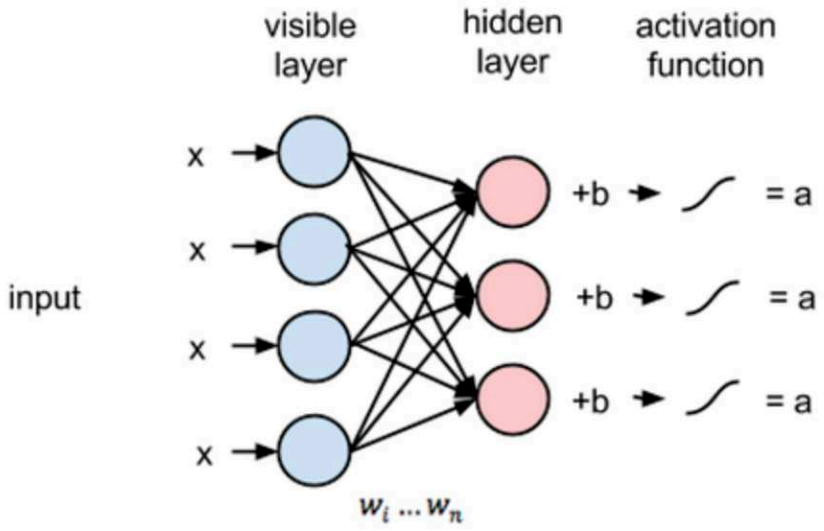


Figure 4.5: Restricted Boltzman Machine representation. Hidden layer **h** is densely connected to visible layer **v** by the weight matrix W.

context dates back to 2006 with the paper [HOT06]. The authors studied the possible approach of training densely-connected Bayesian networks with several hidden layers for data generation and classification tasks. Nowadays, this work is considered a breakthrough in the domain of statistical modeling using graph-based approaches.

To describe DBN, we should first introduce its primary building block, the Restricted Boltzmann Machine (RBM). An RBM is a directed graph that consists of two layers – a layer of visible variables (**v**) and one of the hidden variables (**h**). Nodes that belong to the same layer are not connected. In contrast, the visible layer is densely connected to the hidden layer by undirected connections (Fig. 4.5, taken from [1]). The RBM used in [HOT06] is built of several layers. The top layer represents undirected associative memory. Multiple hidden layers convert information from associative memory into observable variables.

To construct the DBN, the authors stack the RBMs on top of each other. The output of each previous model is used as the input for the next one and can be considered an intermediate representation of the input data. The connections between the top two layers of this architecture are undirected, which is equivalent to having infinitely many higher layers with tied weights. Models are trained in a consecutive way. The first "brick" of the model is trained, its parameters are frozen and the second model is put on the top. Parameters are now updated only for the newest part of the model. Training is continued until the last level is added and trained. Compared to the more modern techniques, DBNs demonstrate relatively limited generative abilities, producing samples of low quality, mainly when applied to high-dimensional natural images.

[1] https://skymind.ai/wiki/restricted-boltzmann-machine

Another technique that deserves special attention, the **Variational Auto-Encoder (VAE)**, was proposed in [KW13]. In contrast to the standard Auto-Encoder (AE), where the codes are simply the encoded low-dimensional representations of the original data, in VAEs, the latent space is forced to fit the unit Gaussian. Such constraints have as the main objective to ensure the continuity of the latent space. They, therefore, allow random sampling from the approximated distribution instead of storing and decoding the explicit codes, as it is done in vanilla AEs.

Variational autoencoders are trained using two losses. First, the reconstruction loss, similarly to standard AE, is computed as the mean squared error or the cross-entropy between the input and the output. The second loss is the Kullback–Leibler divergence that is applied to the latent variables to fit the unit Gaussian.

Variational autoencoders find their applications in multiple domains. One of the most important reasons for this is that they inherit the excellent reconstructive capacities and relatively pure clustering of the separate data classes in the latent space from the AEs, and at the same time, ensure the latent space continuity. This allows generating mixed concepts (e.g., human face + sunglasses = human with sunglasses) by performing the arithmetical operations on the code vectors. However, when real-time learning is required, VAEs are limited to the non-conditional generative process. Indeed, to perform sampling from a given class, one would first need to estimate the distribution of the latent codes corresponding to the given sub-population.

The **Generative Adversarial Network (GAN)** [GPAM+14] is a recently developed approach of generative models that made a breakthrough in the field of image generation. GANs consist of two sub-models, a generator $G(z;\theta_g)$ and a discriminator $D(x,\theta_d)$. G is trying to generate samples according to the data distribution, and D is trained to distinguish between real samples, and those generated by G. Training GANs is based on a process that is very similar to the confrontation of counterfeiters and police. The first attempt to produce a currency that is indistinguishable from the real one (to use it without been detected), while the second tries to improve their methods to distinguish the real and fake currencies.

The generator receives randomly generated samples from the input noise $p_z(z)$ and maps it to the data space. The discriminator, in turn, randomly gets either images from the real data distribution or the outputs of the generator and predicts the probability for these samples to come from real data (Fig. 4.6).

In the described approach, both $G(z;\theta_g)$ and $D(x,\theta_d)$ are multi-layer perceptrons with the parameters θ_g and θ_d correspondingly. The choice is driven by the straightforward optimization techniques and the ability of Neural Networks to capture the variability in internal data representations.

In this context, the discriminator is trained to maximize the probability of assigning the correct labels to the presented samples and the generator, in contrast, to minimize it and thus force the discriminator to make a mistake. To summarize, given the value function

$$V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))]$$

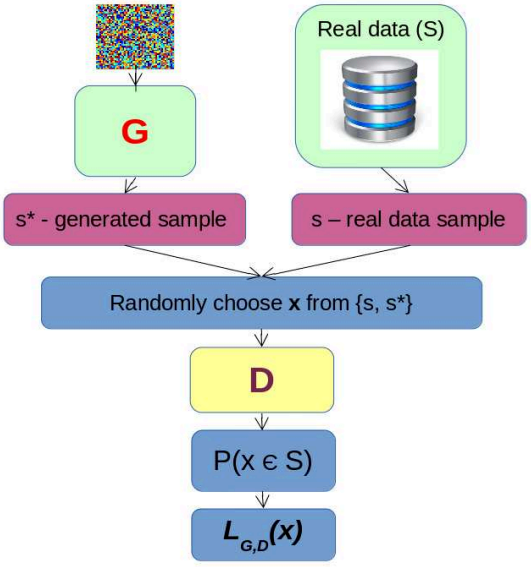


Figure 4.6: Schematic representation of a GAN.

the optimization process consists in finding

$$\arg\min_{\theta_g} \arg\max_{\theta_d} V(D_{\theta_d}, G_{\theta_g})$$

Generative adversarial networks provided the state-of-the-art generative technique for relatively complex data. However, GANs, in their original formulation, rapidly reached their performance limit, not being able to learn from average-size natural images efficiently. Training them on such data demonstrated the unstable behavior of GANs with the minimax game often falling into local minima, which results in the unreasonably poor quality of the generated samples. On the other hand, simple to understand and use, GANs became an excellent platform for further development in the domain of generative modeling.

In [RMC15] the authors propose a convolutional version of GANs (**DCGAN**). The motivation of that work was two-fold: to make use of recent advances in convolutional networks applied to unsupervised tasks and, at the same time, to improve the quality of generated data compared to previously developed methods.

The proposed architectures have some topological constrains to ensure the stability of training. Specifically, the authors replace all the pooling layers by strided convolutions, as it was also done before in [SDBR14]. This allows the network to learn its own down-sampling. To avoid the problems related to poor initialization and smooth the gradient, the authors used batch normalization [IS15] for all layers except for the generator output and discriminator input. Also, to make the model more interpretable and reduce the size of the parameter space, the authors removed all the fully connected layers.

Finally, except for the generator output where tanh is used, the proposed architecture only makes use of ReLU [NH10] and LeakyReLU [MHN13] activation functions. All the proposed constraints have a positive impact on the stability of training and the visual quality of the generated samples.

The resulting model was trained on several baseline datasets, including CIFAR-10, LSUN, ImageNet-1k, and a database of human faces. Training on the LSUN bedrooms set with over 3M distinct images for only one epoch was already enough to generate meaningful images and allowed the authors to demonstrate that their method is not simply overfitting the input data but can generalize and learn distributions.

Since the publication of the original GAN paper, various architectural and training procedure modifications were proposed in order to increase the training stability ([SGZ+16], [ACB17]), learn from extremely high-dimensional data ([DCF+15], [ZXL+17], [KALL17]), discover cross-domain relationships ([KCK+17]), etc.

4.3.3 Pseudo-rehearsal

As it was already said, using generative models as the memory units storing the historical knowledge to alleviate catastrophic forgetting in continual learning is the core of a group of methods usually referred to as **pseudo-rehearsal**.

In addition to the already discussed rehearsal-based method, the authors of [Rob95] propose a pseudo-rehearsal mechanism, which, however differs significantly from the modern definition of pseudo-rehearsal and is somewhat similar to one of the regularization-based methods. Each time a new item has to be fed to the network, a random input vector is generated and passed through the model. The obtained input-output pairs are mixed with new data and fed to the network, thus preventing strong variations in the network parameters.

In [AR97], the authors point out that such a pseudo-rehearsal mechanism has two significant drawbacks. First of all, it requires storing synthetic samples for historical tasks to transfer model behavior to future tasks. According to the authors, storing any amount of data does not correspond to the connectionist philosophy. This inspired the authors to implement a "forgetting-protected" behavior "neurally". On the other hand, the authors argue that the proposed procedure is not optimal for capturing the structure of a data distribution into the model weights.

They propose to sequentially train two networks with identical architectures – the first one serving as the actual learner and the second used to learn the behavior of the first. The authors train the first network on input-output pairs from task A. At the end of task A, the first network starts to receive random inputs and generate corresponding outputs. Generated pairs are then fed to the second network to capture the behavior of the first model. When task B appears, the second network stops training and starts a similar procedure of generating random inputs and producing corresponding outputs. The first network is then trained on the union of task B data and generated pairs from the second network.

The described procedure allows the second network to capture the behavior of the first on the historical tasks and provide regularizing information to the first network when it is trained on the new task. Such an approach doesn't require storing any data.

While allowing to avoid forgetting in the proposed experimental setup, the scalability of this approach to complex datasets is not confirmed and is highly questionable. The main reason for this is the hardly predictable nature of NNs optimization, which cannot guarantee that regularizing network training on data from one input domain (noise) will prevent strong output deviations for the input from a different domain (complex high-dimensional data). This discussion leads us to the idea of training generative models that approximate the data distribution and use generated samples for regularization instead of the noise. This idea is the basis of pseudo-rehearsal in its current understanding and is the core principle behind the rest of the papers discussed in this section.

One of the first attempts to explicitly use generative models to replace historical data to alleviate catastrophic forgetting in continual learning settings was made in [CRDP12]. The authors first introduce a procedure of belief regeneration, which consists of incremental training of a sequence of Deep Belief Networks (DBN), where each subsequent model in the sequence is trained on the data generated by the previous model. To perform the classification task, the authors use recognition connections of the DBN that provide generated data samples with labels.

When training the DBN and the classifier on the stream, the authors mix online data with generated samples from the DBN and update both the DBN and the classifier on this mix. The proposed approach has a major limitation – compared to more recent generative models (e.g., GANs) DBNs provide samples of poor quality that result in strong forgetting. Moreover, the proposed training framework requires several training iterations on each stream data interval, which makes it not applicable to real-time learning on massive streams.

The authors of [DML+17] propose a framework inspired by the process of neurogenesis in the mammalian brain – the continual birth of new extraordinary plastic neurons in the hippocampus throughout the lifetime. In the brain, those neurons are mostly used to integrate behavioral and environmental novelties.

The authors propose to train an autoencoder on the part of the data, containing only a small portion of all the available classes. To proceed with continual learning, the authors simulate a stream on the rest of the data classes. The network architecture is updated in the following way: if for a given stream batch, the reconstruction error is above some predefined threshold, then new neurons are added to the network. To preserve already acquired knowledge, the authors propose to perform learning by updating only the connections passing through the newly added nodes. This is followed by optimizing the full architecture on the union of stream data and replayed historical data classes.

To replay old classes, the authors propose the procedure of Intrinsic Replay (IR). At the end of each task, the class-conditional statistics (mean and covariance Cholesky factorization) of the encoded features are computed and stored. During the replay, the authors sample from the normal distribution based on those statistics and decode the acquired features.

The authors showed that IR controls how the network grows during neurogenesis, thus preventing unreasonable growth. Moreover, the stated results show that the procedure of neurogenesis, together with intrinsic replay, provides a big improvement in autoencoder's capacity to integrate knowledge from new domains incrementally while preserving old knowledge.

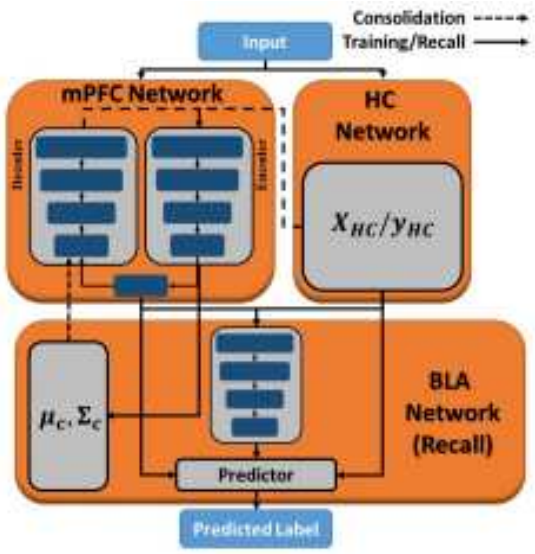


Figure 4.7: Schematic representation of the fearnet framework (Fig. 1 from [KK17]).

A limitation of the work is that one cannot update the covariance matrix on the fly. Thus, if for already learned classes, new samples appear, they will not influence the code distribution, providing no way to control concept drift. Therefore, the method cannot be applied to complex streaming scenarios with possible drifts in the data distribution.

Similarly to the paper on neurogenesis, the authors of [KK17] propose to use generative autoencoders for pseudo-rehearsal. The introduced model consists of three blocks:

1. Hippocampal complex (HC), a recent memory system for quick recall.
2. Medial prefrontal cortex (mPFC), a memory system for long-term storage.
3. Basolateral amygdala (BLA), a sub-system to decide which memory to use for classification at test time.

HC is a data buffer organized by class and filled during stream sessions; mPFC is a symmetric autoencoder that captures the distribution of historical data. The learning process is represented by a sequence of training sessions; each session is a sequence of data batches with no assumption on the i.i.d. nature of the data in a given session. During each session, new data is collected into the HC. At the end of the session, the model enters a sleeping mode where all the collected data are mixed with the generated data for already learned classes produced by mPFC. The latter is then retrained on this mix. The distribution of the codes for each new class is approximated by the Gaussian with the mean and covariance computed over those codes. These distributions are then used to sample data for missing classes during the streaming training. At the end of each session, the HC is emptied.

In the proposed framework, the autoencoder is optimized using the mix of classification $L_{classif}$ and reconstruction L_{recon} loss functions:

$$L_{recon} = \sum_{j=0}^{M} \sum_{i=0}^{H_j - 1} ||h_{encoder,(i,j)} - h_{decoder,(i,M-j+1)}||_2^2$$

counting for the mean of the sum of quadratic errors in each layer j for all units i of this layer. $L_{classif}$ is the supervised classification loss that computes the softmax of the encoded data sample and compares it to the one-hot vectors corresponding to class labels, thus forcing the class-conditioning of the codes distribution.

The authors perform their experiments in the following incremental learning setting. First, the models are pre-trained on several classes from the original dataset in offline mode to create the "base knowledge". Online learning consists of T consecutive sessions, in each session t the model receives N_t data batches from a single class y_t.

The method demonstrated state of the art performance in incremental class learning on image datasets. As a big advantage, the proposed framework is light-weighted and doesn't need long-term historical data storage (only the newest data are stored in the buffer until the sleep phase). However, similarly to [DML+17], once computed, the distribution of class codes cannot be updated. The model is, therefore, only able to assimilate new classes, while in the case of the reappearance of old classes, mPFC would not be able to learn from them. Besides, the proposed classification criterion to train auto-encoders is based on one-hot label vectors. This doesn't encourage the reconstructed samples to be similar to their origin, nor encourages intra-class variability of reconstructions.

Similarly, in [KGL17], the authors propose to use Variational Autoencoders (VAE, [KW13]) as generative models to approximate historical data distributions. Compared to previously described works, the authors propose several novelties. First, they introduce a learning procedure that depends on the "age" of the generative model: older it is and more tasks it has seen, more it is allowed to forget the oldest tasks to promote the assimilation of the new knowledge.

The authors argue that integrating new knowledge directly into a single generative model is not fast enough to process the stream data online. To accelerate learning, they propose to introduce and train a smaller generative model for each new task and train it purely on new data. During the sleep phase, the knowledge of the new models is assimilated into the main generative model by generating samples from small new and large historical models and retraining the historical model on the mix.

The authors of [SLKK17] propose a pseudo-rehearsal mechanism based on Generative Adversarial Networks (GAN). The goal of the approach is to sequentially solve a series of tasks $T = \{T_1, \ldots, T_N\}$. Each task T_i has a corresponding data distribution D_i from which training pairs (x_i, y_i) are drawn.

The authors start with the introduction of a notion of a scholar: a pair $H = (G, S)$ where G is a GAN and S is a task solver (classifier) parametrized by θ. The global learning objective is to minimize $E_{(x,y)\sim D}[L(S(x;\theta), y)]$ where D is the entire data distribution and L is the loss function. Since the scholar is sequentially learning on different tasks acquiring knowledge not only from task data but also from its previous versions, the authors decided to unfold it into a sequence of scholars H_i.

At each new task, G generates data for previously learned tasks. Data are passed through the last solver to obtain labels and mixed with the data from the new task. Both solver and generator are retrained on the mix. The training loss, in this case, looks as follows:

$$L_{train}(\theta_i) = rE_{(x,y)\sim D_i}[L(S(x;\theta_i),y)] + (1-r)E_{x'\sim G_{i-1}}[L(S(x';\theta_i),S(x';\theta_{i-1}))]$$

Unlike previously discussed papers, the last two approaches store and access historical knowledge purely using generative models that are retrained on their own generations. We demonstrate in the experimental sections of this thesis that in the case of imprecise approximation of the data distribution, retraining generative models on their own generations may result in gradual decrease of the quality of generated samples and catastrophic forgetting for classes not appearing in the stream for a long time. The discussed papers do not mention this critical limitation, nor do they propose a solution to this problem.

4.4 Measures and metrics for continual learning

In this section, we discuss the methods used in the literature to assess the performance of incremental/continual learning algorithms. As in a static learning scenario, qualitative and quantitative metrics are used to measure the algorithm performance on a preliminary extracted set of data that is representative of the full data distribution, usually called **validation set**.

Among the widely accepted performance metrics for multi-class classification, we put forward the confusion matrix, which can be used to derive many other metrics since both the precision and the recall can be extracted from it. Deep learning evaluation often consists in measuring the **average classification accuracy**, computed as the mean of the diagonal of the "normalized-per-class" confusion matrix ("normalized-per-class" meaning a row normalization of the confusion matrix). However, when solving a continual learning problem, one rarely has even an approximate knowledge about the full data distribution the model has to deal with. Therefore, standard classification metrics cannot be directly applied to evaluate the behavior of the model when facing problems specific to continual learning, especially when dealing with non-i.i.d. data streams. In the relatively new domain of continual learning, the common agreement on methods of evaluation and comparison is still missing. However, several attempts have recently been made to establish a universal baseline.

In [GMX+13] the authors propose to study the effect of forgetting in a two task scenario by plotting the graphs that relate the error of the model on the first task to its error on the second task. Such a method only suits the cases where the tasks can be learned until convergence. Moreover, it can only be used for at most two separate tasks with no way to scale it to larger problems.

The authors of [LP+17] scale their evaluation framework to the sequence of T separate tasks. Denoting the test classification on task t_j after learning task t_i by $R_{i,j}$ and the accuracy of the randomly initialized classifier on task i by $\hat{b}_i$, the authors propose the following metrics:

$$\begin{aligned}
\textbf{Average accuracy: } ACC &= \frac{1}{T}\sum_{i=1}^{T} R_{T,i} \\
\textbf{Backward transfer: } BWT &= \frac{1}{T-1}\sum_{i=1}^{T-1}(R_{T,i} - R_{i,i}) \\
\textbf{Forward transfer: } BWT &= \frac{1}{T-1}\sum_{i=2}^{T}(R_{i-1,i} - \hat{b}_i)
\end{aligned}$$

In the proposed notations, the **average accuracy** is similar to straightforward average accuracy computed on the joint test set and, as already discussed, is not very informative about the quality of the online learning algorithm. **Backward transfer** sums up the impact the training of the model on task T has on all previous tasks and can thus provide explicit knowledge about catastrophic forgetting. **Forward transfer**, in turn, is designed to measure if learning on one task influences model performance on new, yet related tasks. It indirectly measures the tasks similarity and the capacity of the model to extract shared data representations.

In the similar context of sequential learning on separate tasks $1,\ldots,t,\ldots$, where each task is provided with its own test set, the authors of [SSMK18] propose to compute the model performance separately on all the test sets for tasks $\tau \leq t$, and to compute the forgetting ratio for task τ when learning task t as:

$$\rho^{\tau \leq t} = \frac{A^{\tau \leq t} - A^{\tau}_{R}}{A^{\tau \leq t}_{J} - A^{\tau}_{R}} - 1$$

where A^{τ}_{R} is the accuracy of a random classifier on task τ and $A^{\tau \leq t}_{J}$ is the task τ accuracy of the classifier, jointly trained on tasks $1,\ldots,t$. Overall forgetting is then set to the mean forgetting over all the learned tasks:

$$\rho^{\leq t} = \frac{1}{t}\sum_{\tau=1}^{t} \rho^{\tau \leq t}$$

The provided metrics can only be applied to sequential task learning, with no possibility to extend it to continual learning settings. In [KMA+17] the authors propose an evaluation that fits the continual learning setup where different concepts of data continuously get available with no specific ordering and no precise task definition:

1. $\Omega_{base} = \frac{1}{T-1}\sum_{t=2}^{T} \frac{\alpha_{base,t}}{\alpha_{offline}}$ – the ability to retain base knowledge,
2. $\Omega_{new} = \frac{1}{T-1}\sum_{t=2}^{T} \alpha_{new,t}$ – the ability to learn new knowledge,
3. $\Omega_{all} = \frac{1}{T-1}\sum_{t=2}^{T} \frac{\alpha_{all,t}}{\alpha_{offline}}$ – model performance on all the already seen classes.

where $\alpha_{offline}$ is the test accuracy of the classifier pretrained on the entire data in offline mode, $\alpha_{new,t}$ is the test accuracy on the new class, $\alpha_{base,t}$ is the test accuracy on base knowledge (the part of data used to pretrain the model prior to learning from the stream) and $\alpha_{all,t}$ is the model accuracy on test examples for all already seen classes.

4.5 Discussion

In this chapter, we discussed the existing methods that aim to alleviate catastrophic forgetting when training Neural Networks on non-stationary data streams. The described methods are grouped into three fundamentally different types of approaches, each having its advantages and limitations. To make visible these differences, we summarize the main characteristics of each approach in Fig. 4.8.

The methods based on training regularization impose geometrical constraints either on the function parameters or on the input data space to reinforce the separation of data classes or tasks. They, therefore, prevent network optimization from correcting the model parts responsible for temporarily absent data concepts. Making the link to biological learning systems, such systems correspond to the capacity of the brain to allocate separate resources to separate problems, with brain areas affected only when corresponding tasks are being learned. Such methods are usually computationally expensive and require explicit knowledge of the task environment at testing time. While not being applicable to continual learning and frequently requiring offline training phases, the discussed methods demonstrate outstanding performance on the incremental learning tasks.

Evolving neural architectures, on the other hand, are mainly designed to adapt neural architectures by allocating supplementary memory and computational resources to correspond to the growing complexity and richness of the data in dynamic streams. Not explicitly addressing the problem of catastrophic forgetting, such methods are usually much more "light-weight" than regularization-based techniques, allowing fast online data processing.

Lastly, dual-memory based algorithms aim to regularize training by accessing the historical knowledge either in the form of stored data samples or as generative models that reproduce the acquired knowledge from the stored statistical approximation of the historical data distribution. While often having significant computational overhead compared to the other methods, rehearsal-based approaches allow addressing the continual learning problems with no restrictions on stream structure and task-independent testing.

We argue that despite demonstrated differences, the discussed approaches can be considered complementary, from both a biological and a computational point of view. Comparing the discussed approaches to a dynamically learning biological system, we can relate regularization-based approaches to chemical and physical processes that help to consolidate knowledge in the brain, evolving neural architectures to the processes of neural evolution during the lifetime, especially in the childhood when the brain plasticity is extremely high, and dual-memory based approaches to biological memory that serves not only to store the exact knowledge but also to generalize learning during the lifetime. We consider, however, that the dual-memory mechanism is significantly better adapted to the task of continuous learning that we address in this thesis. For this reason, the approaches we propose in this thesis are based on the idea of pseudo-rehearsal.

	Method	Incremental learning	Continual learning	Online trainnig	Task-independent testing	Computational overhead	Memory overhead	Scalability
Regularization	LWTA	+	+	+	+	No	No	+
	LwF	+	-	-	-	High	Low	-
	EWC	-	-	-	+	Medium	High	-
	SI	+	-	+	-	Low	Medium	-
	MAS	+	-	+	-	Medium	Medium	-
	Pathnet	+	-	-	-	Medium	Low	+
	HAT	+	-	-	-	Low	Low	+
Dynamic architectures	Incremental Feature Learning	+	-	-	+	Medium	Low	+
	DEN	+	-	-	-	Medium	Low	+
	Progressive nets	+	-	+	-	High	High	-
	Prototype based methods	+	+	-	+	Medium	Medium	-
Rehearsal	Robins	+	+	+	+	High	High	-
	iCarl	+	+	+	+	Low	Medium	-
	GEM	+	-	-	+	Medium	Medium	-
Pseudo-rehearsal	Reverbating nets	+	+	-	+	High	Low	-
	DBS	+	+	-	+	High	Medium	-
	Neurogenesis	+	-	-	+	Medium	Medium	+
	Fearnet	+	-	-	+	Low	Medium	+
	Uncompromized incremental learning	+	-	-	-	Low	Low	+
	DGR	+	+	+	+	Medium	Medium	+

Figure 4.8: Comparison of existing methods aiming to alleviate catastrophic forgetting

Part II

Contributions

Chapter 5

Using GAN-based pseudo-rehearsal for online classification

As it was previously discussed, catastrophic forgetting is the core issue when applying neural networks to dynamic data with changing distribution. In this chapter, we propose a method to deal with the problem of catastrophic forgetting and validate this method on the online learning problem in incremental and continual settings. We argue that before tackling the problem and discussing the methodology of our work, it is essential to deeply understand the phenomenon of catastrophic forgetting and the reasons it appears in neural networks. For this reason, we start this chapter by an experiment on synthetic data that provides the reader with a visually understandable image of the nature of this phenomenon. Also, this discussion allows us to justify the choices of the building blocks we use to design our learning framework.

5.1 Experimental analysis of the long-term memory in neural networks

Catastrophic forgetting (or catastrophic interference) can take place in any type of NNs, independently from the objective function in use and application of interest. In this section, we demonstrate how catastrophic forgetting influences the learning process on the example of the classification task. To do so, we design a simple experiment that reproduces the non-i.i.d nature of data in the stream. We start by introducing a small synthetic dataset containing 5 classes, with each class being sampled from a 2D Gaussian with the following parameters (see fig. 5.1 for visual support):

- Class 1: Mean: [0, 0]; Cov: [[1, 0.5],[0.5, 2]]
- Class 2: Mean: [5, 7]; Cov: [[3, -3],[-3, 4]]
- Class 3: Mean: [4, -8]; Cov: [[5, 0],[0, 1]]
- Class 4: Mean: [-5, -10]; Cov: [[2, -2],[-2, 2.5]]
- Class 5: Mean: [-8, 5]; Cov: [[4, 2],[2, 4]]

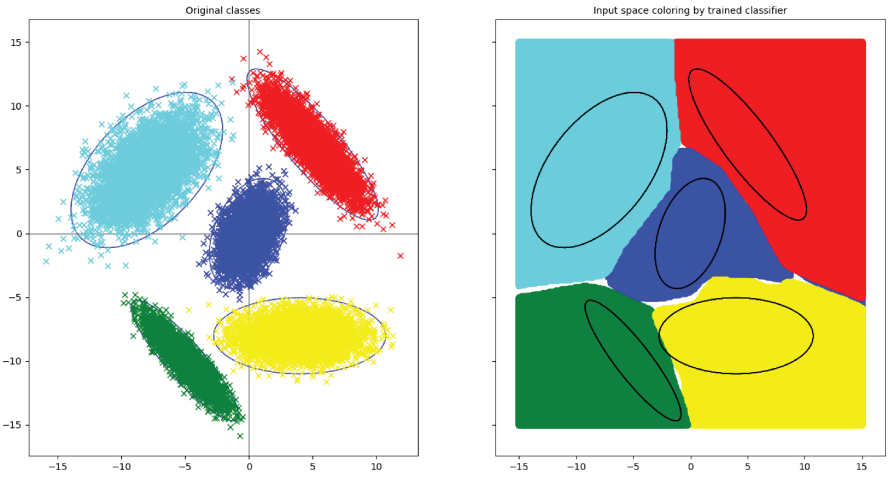


Figure 5.1: Simple synthetic 2D data. (Left) sampled data classes, described by 2D Gaussian distributions, (Right) input space partition by a NN classifier trained on the synthetic data from the left, white regions correspond to the uncertainty zones ($c_u = 0.8$)

To spatially separate classes, the moments and covariances are set in such a way that the confidence ellipses corresponding to 3σ for each class are not intersecting. We proceed by initializing and training a simple 5-class classifier (Linear(2, 6) -> ReLU -> Linear(6, 6) -> ReLU() -> Linear(6, 5)) on all the introduced classed.

Such a classifier, when applied to the entire input space, by its design, is meant to create an input space partition. Any point from the input space, no matter how far it is from the original classes, is assigned with a label corresponding to one of the classes. To make this statement more clear we propose to pass the rectangular region of the input space containing all the classes (all the drawn samples have more than 99.9% chance to belong to that region) through the classifier and color its point depending on the assigned labels. To do so we initialize a 2D grid in the range $[-15,15]$ and step 0.1 for both dimensions, pass it through the classifier and apply the Softmax function to obtain the vectors of probability $[p(y = i|x)]_{i=1}^{5}$ of a giving point x to belong to each class. Usually, when performing classification tasks, in testing time, the label is assigned to the class with the highest probability p_{max}. However, to account for the network uncertainties, we introduce an additional parameter – uncertainty threshold c_u. In these settings, the point is only colored when $p_{max} > c_u$, and is left blank otherwise (see fig. 5.1, right).

Provided visualization confirms our previous statement – the classifier indeed occupies the totality of the input space. Such logic, on the one hand, means that the classification Neural Networks do not have any explicit mechanism for novelty detection since the classifier has a by default "opinion" with high certainty about any possible input. On the other hand, and we believe this to be the actual reason for catastrophic forgetting, in the absence of some data classes, even already learned, the network is forced to fill the regions previously attributed to those classes by the currently available concepts.

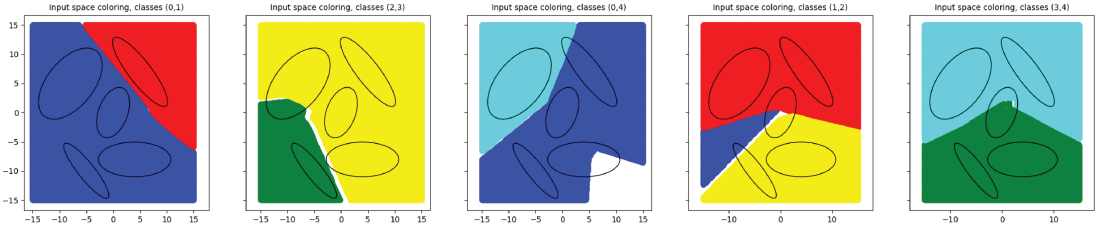


Figure 5.2: Space partition during sequential classifier training on class pairs.

To confirm this last statement, we modify the previous experiment. We now take the same network, re-initialize it, and train on a sequence of randomly selected class pairs, with each pair, learned until convergence. Similarly to the previous experiment, we visualize the coloring results after each training step (fig. 5.2). We can see that at each step, regions corresponding to previously learned classes are almost fully covered by the currently available classes. At the same time, the network is not fed with any type of false information about those regions and thus is not explicitly forced to forget unavailable data. This experiment shows that in the case of the distribution changes of the input training data, approximated function suffers from drastic changes in its topological properties. However, the provided demonstration only shows the behavior of the classifier as a black box – we check how changes in the input influence the output with no understanding of the underlying processes in the hidden layers.

To go forward and understand not only **what** is happening but also **why** it is happening, we visualize the activations of the hidden layer. Let us first introduce some notations. Let a_{1j} and a_{2j} be the activations on the input $x = (x_1, x_2)$ of j-th neuron in the first and second hidden layers correspondingly:

$$a_{1j} = ReLU(W^0_{j1}x_1 + W^0_{j2}x_2 + b^0_j)$$

$$a_{2j} = ReLU(\sum_{i=1}^{6} W^1_{ji}a_{1i} + b^1_j)$$

In provided notations, with D_k denoting the data corresponding to the class k, let us introduce

$$a^{(k)}_{ij} = \text{mean}_{x \in C_k} a_{ij} \ ,$$

the **average activation of the j-th neuron of the i-th layer on the data class k**.

Using proposed notations, in the previously described scenario with switching class pairs, we compute the average activations at the end of each training scenario separately for each class and visualize them (see fig. 5.3, brighter pixels correspond to stronger activations). Each element on the figure corresponds to the network's hidden layers activations (each sub-figure has two columns = two hidden layers of the network) after a given scenario (columns) for a given class (rows). To have a comparative baseline, we perform the same visualization in the case of training on the full dataset, with each column being the activation snapshot at the end of every 20 training epochs (fig. 5.4).

To better understand the nature of provided figures, let us perceive Neural Networks as a sequence

of spatial mappings of the input data onto the intermediate representation spaces. While the goal of classification training is to separate classes in the output layer, from both figures, we can see that the classes are already getting separated by the intermediate mappings – they are mapped onto the separate regions of the space of the hidden representations.

What changes between the experiments is the nature of those mappings during learning. As we can see, on the fig. 5.4 corresponding to the training on the i.i.d. data, the mappings stay almost unchanged between training sessions leaving the topological organization of the intermediate data representations close to the previous states. In contrast, when introducing drastic drifts in input data distribution during training (fig.5.3), the internal mappings undergo dramatic changes between sessions. Such behavior completely reorganizes the topology of hidden representations with each learned task. This makes training significantly slower and introduces huge and often unpredictable changes in the output.

Based on the previous discussion, we argue that there exist at least two potential ways to approach the problem of catastrophic forgetting. The first one consists in putting spacial layer- and class-wise constrains on the mappings performed by the neural network to preserve the topology of those mappings (this corresponds to the regularization based methods in the state of the art chapter). The second way, in turn, consists in conserving the input space repartition and ensuring the i.i.d. nature of the training data (rehearsal and pseudo-rehearsal methods).

We have seen in Chapter 4 that the first group of methods often requires storing relative information about the network's structure, gradients, and even the full history of the parameters of the network for each separate task. Besides, imposing too many constraints on the network's updates can significantly slow down training and prioritize the model's stability over its plasticity – the capacity to incorporate new knowledge.

From the discussed perspective the rehearsal based approaches can be seen as a way to "reserve" the input space regions corresponding to the already learned classes and therefore protecting the model from overwriting, or in previous terminology "coloring", the regions of the missing classes by the currently available concepts. As it was already discussed in the Chapter 4, approximating those regions directly by storing the original samples from the stream have either generalization (stored data are too few to represent the full richness of the missing class) or memory (stored data are too large) limitations. We, therefore, decided to take the idea of pseudo-rehearsal as the basis of our online learning methods, proposed in the following sections. This choice was mainly guided by the ability of the modern generative models to efficiently approximate the distribution of missing data, which makes them a potential solution to overcome both described limitations of rehearsal.

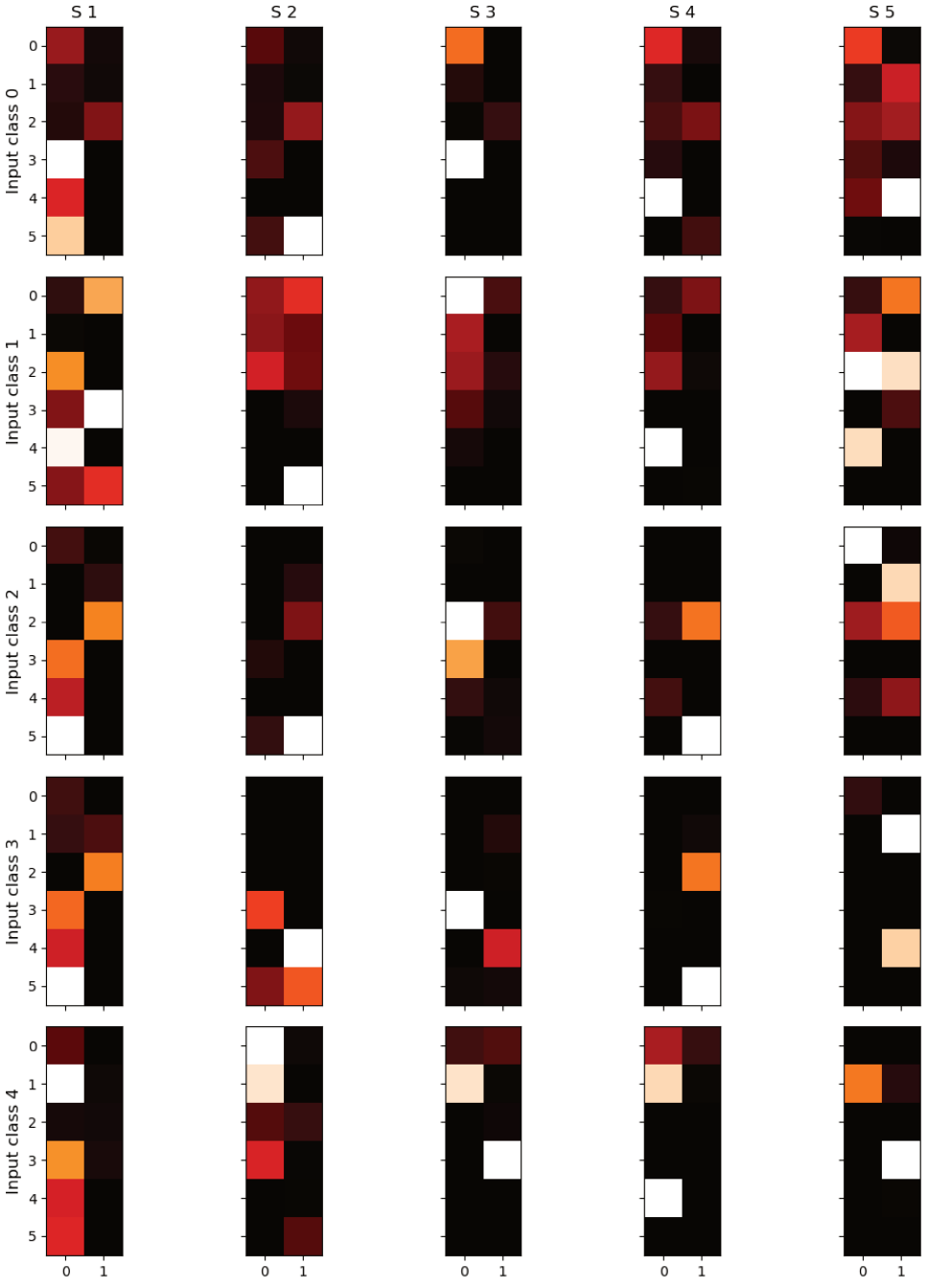


Figure 5.3: Class-wise hidden layer activations for sequentially learned scenarios

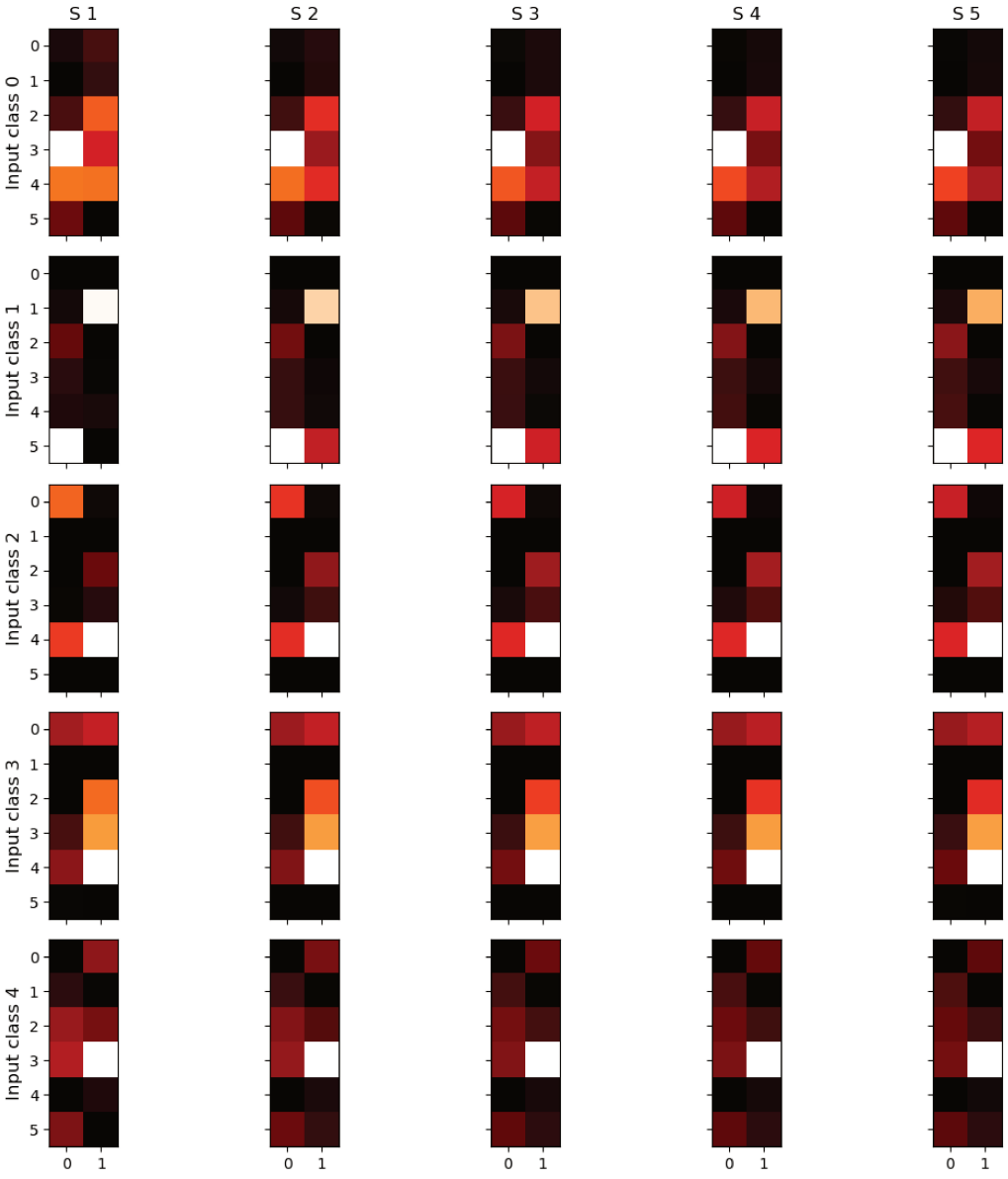


Figure 5.4: Class-wise hidden layer activations when trained on full data, snapshot taken every 20 training epochs

5.2 Classification-based evaluation of generative model performance

In this thesis, we aim to build a framework of efficient continuous learning and, as it was previously mentioned, decided to construct it following the pseudo-rehearsal philosophy. To do so, we need a generative model able to rapidly and efficiently learn from the massive stream of high-dimensional data. In addition, we have chosen image classification as the application of interest in our experiments. Following the discussion on the state of the art in generative modeling provided in Chapter 4.3.2, we argue that **DCGANs** [RMC15] match perfectly described objectives. We, therefore, use them as the source of pseudo-rehearsal in the online learning approach proposed in this chapter.

The global objective of the thesis is to investigate the performance of our online learning approach, more precisely to see how the data sampled from the generative model performs when used to train networks in an online scenario. However, before proceeding to that, we first make sure that the enabling assumption is correct: that is, verify that trained generators are able to represent well the initial training data and generalize on the data distribution. Despite their recent popularity, generative models in general, and GAN-based models in particular, until now, were almost not studied for their ability to generalize outside the training set. At the same time, the standard technique to evaluate the quality of generated samples and verify how close the generated samples are to the original concepts consists of human-expert evaluation. Such evaluation can be extremely time consuming and biased.

In this section, we introduce the notions of **generalizability** and **representativity** of generative models and propose quantitative metrics to evaluate them. By generalizability of the generative model, we understand its capacity to focus on learning concepts and patterns and become a representation of the data distribution rather than reproducing data samples it encounters during training. The term representativity is used to describe the ability of generative models to represent the original dataset it was trained on with all its internal variability.

In the case of classification algorithms, measuring the generalization capacities of a given model is very straightforward and can be evaluated by the classification accuracy decay when passing from training to validation set. Creating a generative model that would focus on learning concepts rather than memorizing single data samples is a problem of very high importance since it can be viewed as the machine's capacity to generalize to unseen instances (similar to what creativity and imagination are for human intelligence). In other words, we are more interested in creating a model that would be able to "imagine" objects, that are similar to those from data distribution, rather than just reproducing the data samples it has seen during training, especially in the online learning context where data distributions tend to change in time. This brings us to the question of defining and measuring the generalizability of the generative model. The measure of the model's capacity to generalize cannot be directly transferred from a classification model to a generative model. Instead, we propose a new notion of generalizability that captures much the same principles but adapted to the generative case.

We shall say that a generative model G trained on some support subset D^{supp} of a dataset D, with D^{val} being the validation set of D such that such that $D^{val} \cap D^{supp} = \emptyset$, generalizes well on D over some measure μ, evaluating the semantic resemblance between two data sets, if the similarity over μ between D^{val} and D^{Gen} – the data sampled from G, approaches the similarity between D^{val} and

$D \backslash D^{val}$. In a more formal way:

$$|\mu(D \backslash D^{val}, D^{val}) - \mu(D^{Gen}, D^{val})| < \epsilon,$$

where ϵ is the parameter that determines the quality of generalization.

Choosing the metric to measure the semantic similarity between the two datasets is not straightforward. In our study, we decided to use a neural network-based classification model for this purpose. Since neural networks are known and much appreciated for their ability to learn the abstractions and internal data representations, the mean classification accuracy of this model, when trained on one dataset and tested on another one, represents the desired property.

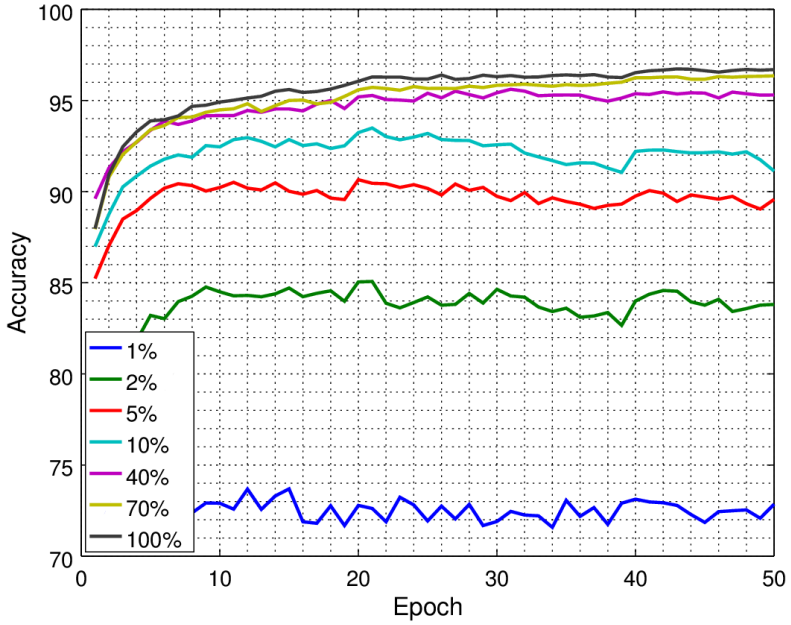
One of the main assumptions on the generalization capacities of any machine learning algorithm is that to improve it, one often would want to get a bigger training set with more representative data samples. In our experimentations, we adopt this idea and adapt it to the case of generative models. In the following experiment, D is the MNIST dataset, and G is the set of generative models $G_1, \cdots, G_{10}$ – one for each data class. To assess the ability of G to generalize on unseen content, we designed a test that consists in varying the size of the set D^{supp} used to train generative models from 60 (1% of D) to 6000 (100%) samples per class, and, at the same time, comparing the mean classification accuracy of the classifier, trained on the data generated by G_i, to the one trained and tested on the original data.

Fig. 5.5(a) represents the classification accuracy as a function of the training time, averaged over 10 runs of classifier training on generated data G, with G trained on the datasets of different sizes. We can see from the figure that the classification performance on the validation set significantly improves when increasing the size of the support set to train the generative models. We observe that with only 1% support size, the classification accuracy is pretty high (70-75%), and, with a support size of 5%, the accuracy goes up to 90%. To quantify this effect, Fig. 5.5(b) shows the curve of the improvements through all the tested support size values and makes a link with the generalizability definition proposed earlier, with $\epsilon = 5\%$ and $\mu(D \backslash D^{val}, D^{val}) = 99.6\%$. We can see from the figure that using only 40% of the initial dataset (2400 samples per class) allows us to obtain a highly generalizing generative model with the generalization error below 5% (the values in the blue box). We can also observe that the curve keeps going up, which means that having more data for training the generative models should improve the final classification accuracy.

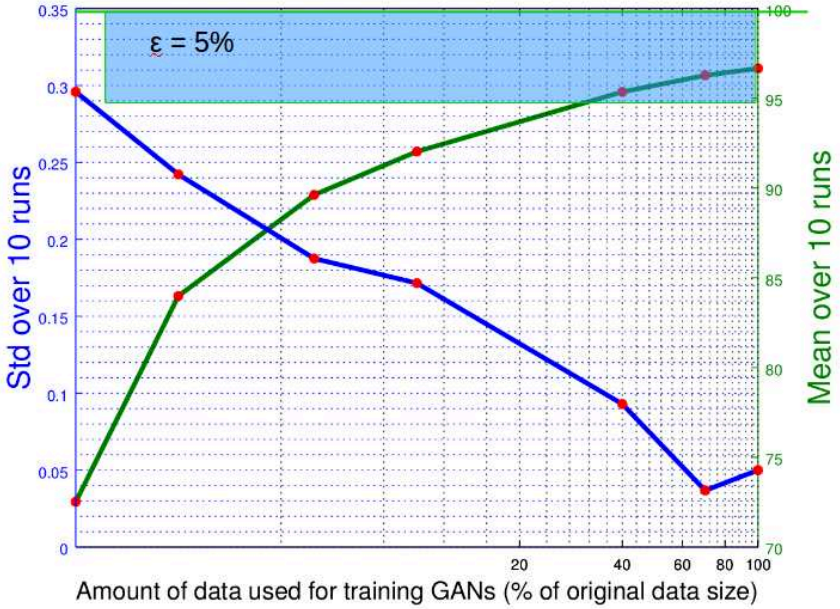
Fig. 5.6 shows random examples of synthetic samples generated by DCGAN when trained on a different amount of original images from the MNIST dataset. We see that starting for 10% support size, the generated data is visually very consistent, confirming the results above.

Another important question we needed to answer before passing to the online scenario is how much data do we need to sample from pre-trained generators to represent the full richness of the information learned by generative models from the original dataset. This problem is essential since the amount of data we need to generate while training online classifiers influences directly the learning reactivity. It would be reasonable to sample the smallest amount of data that, at the same time, wouldn't affect too much the final classification accuracy.

Similar to the generalization experiments, we trained one generative model for each MNIST class. We



(a)



(b)

Figure 5.5: Results of the generalizability test on the MNIST dataset. (a) Classification accuracy for different GANs support sizes as a function of training time. Average over 10 runs; (b) Mean/std of the classification accuracies for different GANs support sizes over 10 runs after 50 training epochs for the generalizability tests. Blue box represents the area in which the generalization error does not exceed 5%

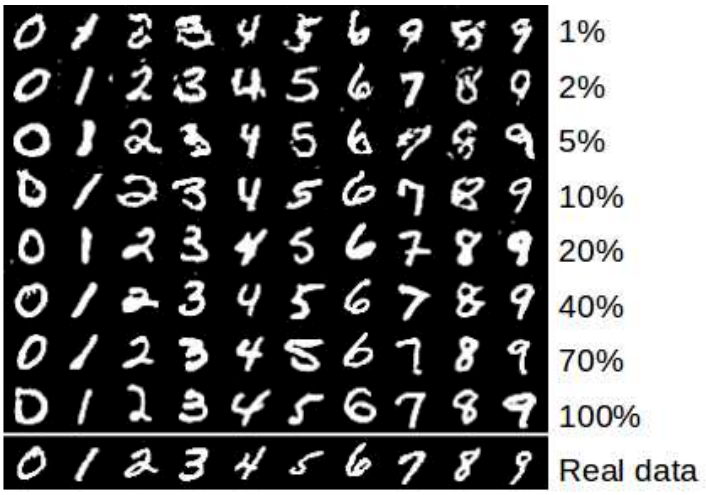


Figure 5.6: Samples, produced by DCGAN-based generator, when using 1 to 100% of the original MNIST dataset to train it

used the generated data to train a classifier with the difference that, this time, we used the full dataset as support for generative models training.

To verify the representative capacities of DCGAN, we studied the influence of the amount of generated data on the final training accuracy by varying its size from 1 to 100 % of the original dataset. Fig. 5.7 represents the mean and standard deviation of the classification accuracy over ten runs for each chosen size of the generated dataset. We can see that for MNIST dataset, generating samples of only 30% of the original dataset is enough to reach almost the same training accuracy and stability, represented by low standard deviation, as for the case of generating 100% of the original dataset size.

Comparing to the training on original data, where with the classification model we use, we obtain the accuracy of $F^{orig} = 99.6\%$, training on generated data reduces the maximum accuracy down to $F^{gen} = 97.2\%$. We can consider this decrease as the price we pay for not storing the data. Based on these two values we can introduce the metric to evaluate representativity of generative model:

$$R_M(D) = \frac{F_M^{gen}(D^{val})}{F_M^{orig}(D^{val})} = 0.976,$$

where M is the model we evaluate and D^{val} is the validation set – the subset of initial data that was not used to train the generative model. In these notations, a value of $R_M(D)$ close to one represents the case of D being well represented by G. In this case, generative models not only work as a memory to store data representations but also act as a filtering mechanism that extracts useful information from data samples. Bad representativity correspond to values of $R_M(D) << 1$ and $R_M(D) > 1$.

To check if DCGAN-based generators can represent more complex datasets, we train one DCGAN generator per class on the LSUN dataset. We then train two classifiers: the first one on the original data that was used to train the generators, and the second one on purely generated data produced by pre-trained DCGANs. We then test the obtained classifiers on a validation set consisting of previously

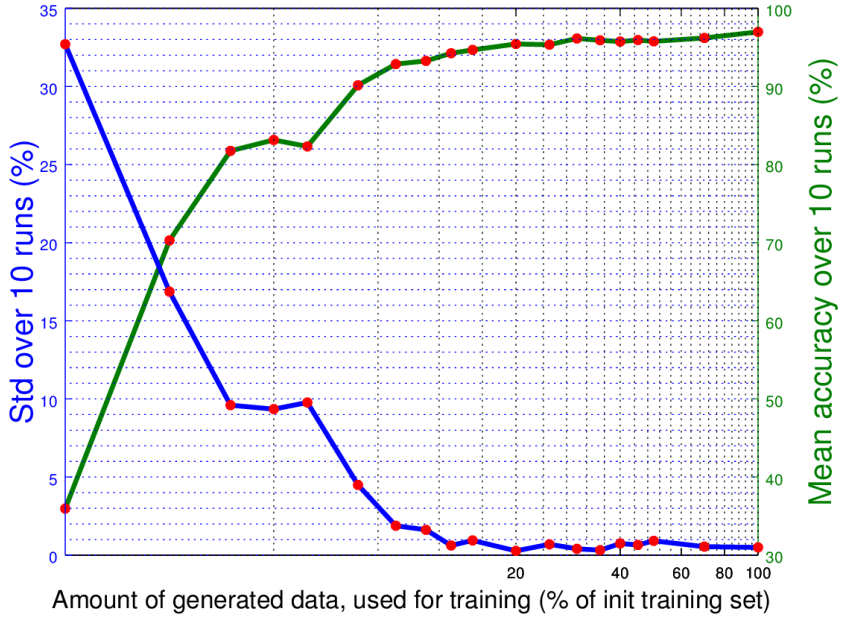


Figure 5.7: Representativity check of the DCGAN on MNIST dataset. This experiment demonstrates the performance of the classifier trained on generated data, depending on the amount of data sampled from the generator (% of the size of the original dataset)

unseen real images and compare the classification accuracies of both. From Fig. 5.8 we observe that using generated data to train a classifier results in a decrease in classification accuracy (MNIST: 99.14% $\rightarrow$ 97.16%; LSUN: 88.69% $\rightarrow$ 70.22%), especially for the LSUN dataset. Nevertheless, we find this decrease to be an acceptable trade-off to pass to a completely online classification training scenario with no necessity to store historical data, especially taking into account the data complexity of the LSUN dataset.

5.3 Proposed Method

In this section, we propose a method to learn online from dynamic data and build learning frameworks that allow us to use it in two streaming scenarios: incremental and continuous data streams.

Incremental scenario For the first experimental scenario, we model the streams in a way that the data arrives continuously with distinct classes coming separately one by one.

More formally, let $S = \{S_i | i = 1,..,N\}$ be the partition of real data into N distinct classes. In our learning framework, we take the first coming class S_1 from S and train a generator G_1, able to represent this data. We save G_1 and discard S_1. We then start training G_2 on the data from S_2, and in parallel, train a classifier C_1^2, feeding it with samples from S_1^* – synthetic data, generated by G_1, and newly

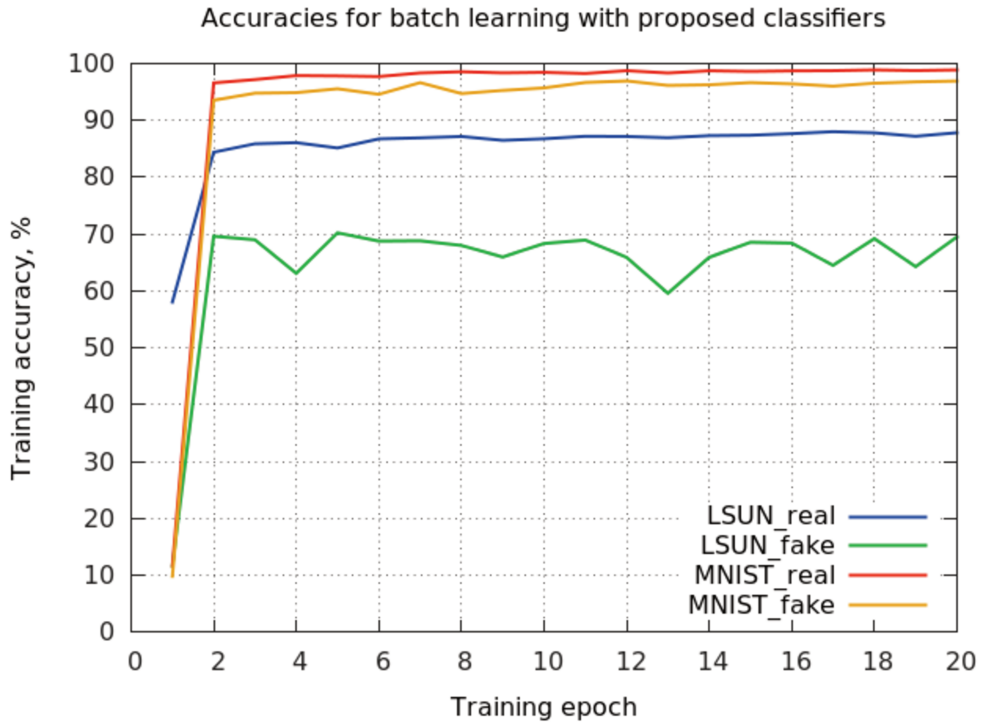


Figure 5.8: Batch training accuracy on the original validation data for MNIST and LSUN dataset when trained on real vs. generated data

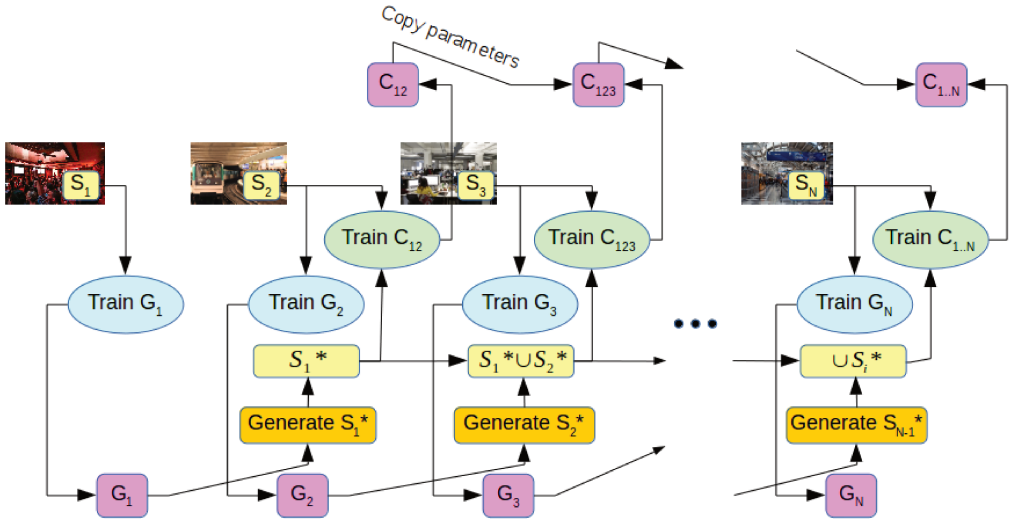


Figure 5.9: Schematic representation of our online learning approach. Original data is presented to the model class by class. Each time new class of data appears we start training a new generator modeling that class. At the same time we train a classifier on the generated data from the previously learned classes and the original data from the new class that come from the stream.

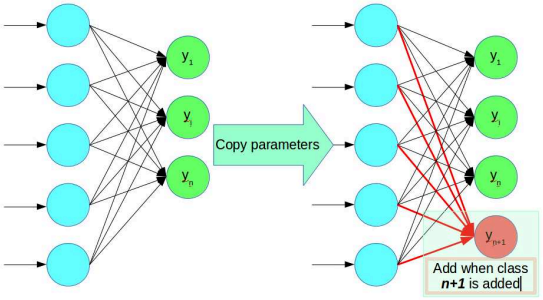


Figure 5.10: Adding a node to the output layer and initializing the connections with the previous layer in the online learning scenario when new data class appears in the stream.

arriving real data from S_2. After that, data from S_2 are discarded. We continue this procedure with all available classes from S, one by one, each time generating equal batches of data from all the previously trained generators. Each time a new class is added, we also add a node to the output layer of the classifier and initialize its connections with the previous layer (fig. 5.10). The rest of the network weights are copied from the previous state. See algorithm 1 for the pseudo-code and fig. 5.9 for the schematic representation of proposed framework.

Continual learning scenario We now extend the previous scenario to the case where data are coming continuously and in random order, which corresponds to a much more realistic situation. Let $E = \bigcup_{k=1}^{\infty} E_k$ be an environment emitting data continuously in time, where E_k represents the subset of E corresponding to class k. Generally, in the continuous stream, we assume that classes are mixed, and on each stream interval, we may have several classes. For simplicity, we consider that we receive data from the stream in the form of small batches of size b, where each batch contains only the elements of one class.

We start by assuming that the stream is divided into time intervals. Every interval contains at least two and, at most, M distinct data classes. Each time a new interval is started we remove several classes from the previous interval and add new classes from E so that the new interval always contains at least one class from the previous one (to simulate environment continuity) and never exceeds M classes. Every class, when it appears in the stream, emits a random number of batches. The duration of each interval and sub-interval, corresponding to a given class, is set randomly from corresponding predefined ranges.

Let us also initialize a data buffer B of size $N \times b$, which serves to collect data from batches to use it later to train a classifier. We fill in the buffer until the number of batches of one of the classes reaches the buffers limit size. After that, we complete buffer to have an equal number of images for each class by generating samples from all the pre-trained generators (Fig. 5.11, forming the buffer), and send obtained data to train the classifier. We then empty the buffer and start filling it again.

Require: $S = \bigcup_{i=1}^{\infty} S_i$: data stream, with i – class number
Require: n : number of already learned classes
Require: G_i : generative model for class i
Require: C_1^n : classification model for data from $\bigcup_{i=1}^{n} S_i$

$G_1 \Leftarrow$ initialize model
$n \leftarrow 1$
while are receiving samples from S **do**
  $d \Leftarrow$ get batch from S_j, j – current class
  if $j = n+1$ **then**
    $n \leftarrow n+1$
    $G_n, C_1^n \Leftarrow$ initialize models
    if $n > 2$ **then**
      $C_1^n \Leftarrow$ copy parameters from C_1^{n-1}
    end if
  end if
  $d^* \leftarrow \bigcup_{i=1..n}^{i \neq j} d_i^*$ generate synthetic data from $\{G_i\}$
  $C_1^n \Leftarrow$ train with $d \bigcup d^*$
  $G_j \Leftarrow$ train with d
end while

Algorithm 1: Online learning model, proposed in Sec.

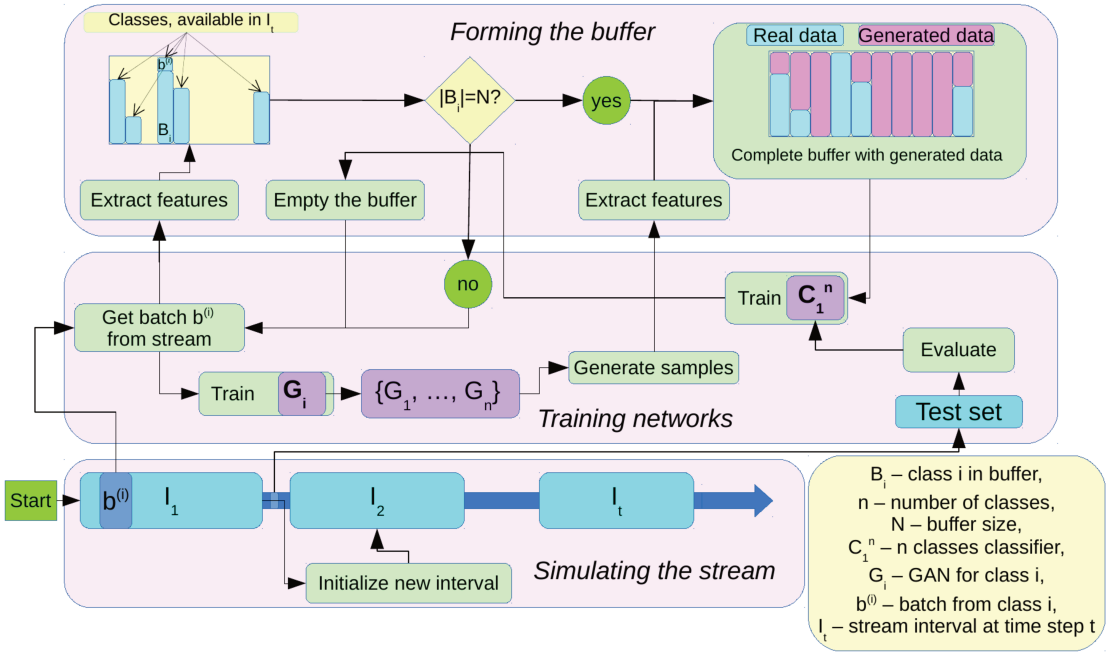


Figure 5.11: Stream classification scheme (for LSUN dataset, on MNIST no feature extraction is performed), described in this work. Stream is represented as an infinite sequence of data intervals.

When starting the online training on stream, we consider that we already have some base knowledge – we pre-train generative models and a classifier for several selected classes from the dataset. Each time a new class appears in the stream, we initialize a new DCGAN for it. We train the DCGANs with batches of corresponding classes directly when they appear in the stream. The classification network is trained each time the data buffer is complete. The performance of the classifier is evaluated on the test set at the end of each stream interval. Fig. 5.11 shows the full schematic representation of the proposed framework.

5.4 Experimental setup

For all the experimental scenarios described in this thesis, we assume that sampled data are of a unique type and format (e.g., RGB images of the same size/depth and represented in the same color space with the same range of values, sound recordings of the same length/sampling rate, etc.). The experiments described in this chapter were held on the following two datasets:

MNIST is a collection of gray-scale images of hand-written digits of 28×28 pixels each. This database is widely used as a baseline in NN benchmarking. The images include 10 data classes, each corresponding to a separate number from 0 to 9. The training set includes 6000 images per class, and the test set 1000 images per class. No spatial transformation was applied on MNIST for either classification or GAN training. The classification network we use on MNIST consists of two convolutional with max-pooling layers (with resp. 16 and 32 feature maps using 4×4 kernels), followed by three fully connected layers (512×512, 512×128 and 128×10) with ReLU activation function except for the output layer.

LSUN is a collection of RGB images of size at least 256×256 pixels each. The original dataset includes 10 classes of scenes (bedroom, bridge, church outdoor, classroom, conference room, dining room, kitchen, living room, restaurant, and tower), with the smallest class containing around 126k images and the biggest one over three millions of images. We extracted 5k images from each class to use them as a validation set for classification; the rest of the images were used to form the stream and train both the generative models and the classifier.

Every image is transformed into a square shape by cutting its sides. DCGAN, in its original formulation, does not work on big size images but works perfectly well on images of size 64×64 pixels and less. Also, since our goal was to simulate a data stream with unique samples, we needed to perform some data augmentation on the dataset. For these reasons, we rescaled LSUN images to the size of 96×96 pixels and randomly cropped them to 64×64 pixels each time they appeared in the stream.

Model architecture All the experiments in this chapter used a DCGAN [RMC15] for the generative model, both for the online and offline training settings. We used DCGAN exactly as described in the original paper, with no modifications.

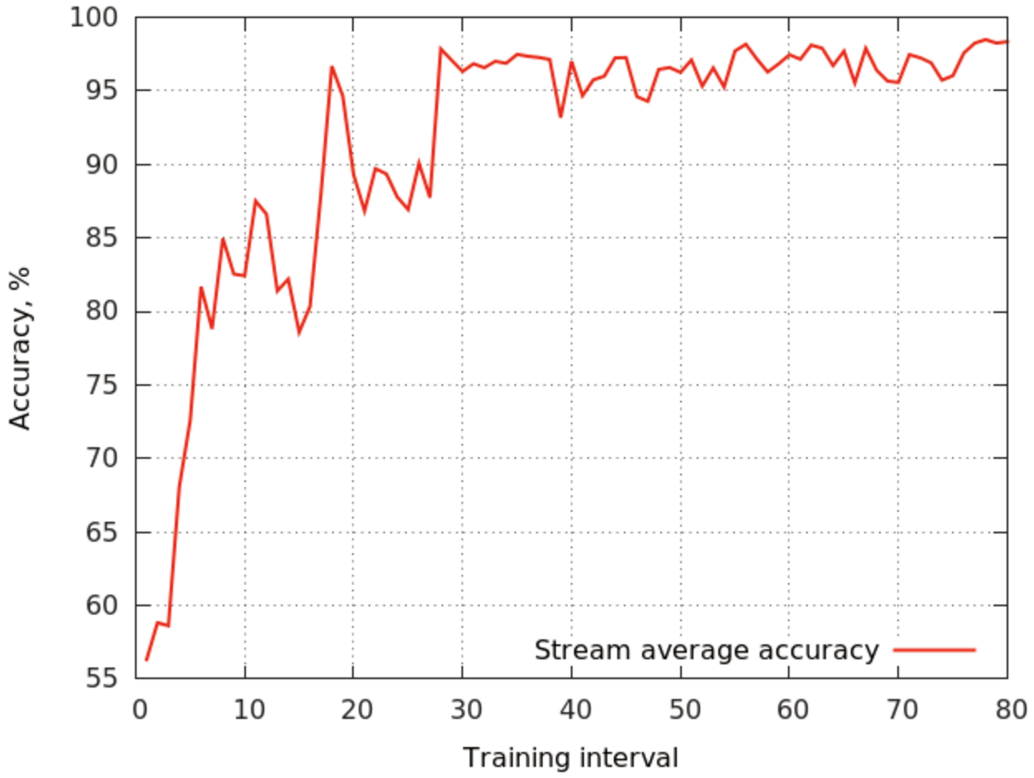


Figure 5.12: Classification accuracy during online stream training for MNIST dataset

The classification model hyper-parameters were adjusted according to two criteria:

- the performance of the model should be comparable to the state-of-art
- the network should not be too deep to allow real-time training.

The classification model we retain for MNIST is constituted of one convolutional layer followed by two fully-connected linear layers. A Rectified Linear Unit activation function follows each layer except the last one. Batch normalization [IS15] and dropout [SHK+14] are applied during training on all layers except the output layer. Stochastic Gradient Descent using Adam [KB14] is used to perform model parameters optimization. The classification network described in this paragraph is used for all experimental scenarios.

Training state-of-the-art classifiers for large complex datasets usually requires very deep network architectures and takes a lot of time and resources. Getting a performance similar to the state-of-the-art batch learning scenario on LSUN dataset was not the intention of this work, so we used a few shortcuts that allowed us to speed up training, to the expense of a slight decrease in classification accuracy. More precisely, we rescaled the original images, as well as the generated 64×64 pixels images, to a 224×224 size, and passed them through the convolutional layers of the ResNet-200 network ([HZRS16]), a convolutional network pre-trained on the ImageNet dataset. The latter was thus used as a feature extractor. Four fully-connected layers with ReLU activations (except for the output

layer) were added on top of the feature extraction block to form the 10-class classification network ($2048 \times 1024 \rightarrow 1024 \times 512 \rightarrow 512 \times 128 \rightarrow 128 \times 10$).

5.5 Results of the online experiments

Incremental learning One of the possible limitations of our online classification method is that to avoid forgetting, we need to continuously generate data from all the previously learned classes when receiving samples from new classes. The dependency between the amount of generated data and the total number of classes in the dataset may become a problem for classification tasks with a large number of classes. On the other hand, synthetic data in our model is used to ensure generalization and stability for the learning process and should not be considered as the main source of information for the parameters update. Generating too much synthetic data can thus reduce the importance that the model gives to original data we receive in streaming mode.

Similar to the tests on representativity of generated models, described in Sec. 5.2, we evaluate the performance of our incremental learning algorithm depending on the amount of data we generate. The only difference is that in the case of data streams, we cannot know in advance the size of the dataset, neither the total number of classes.

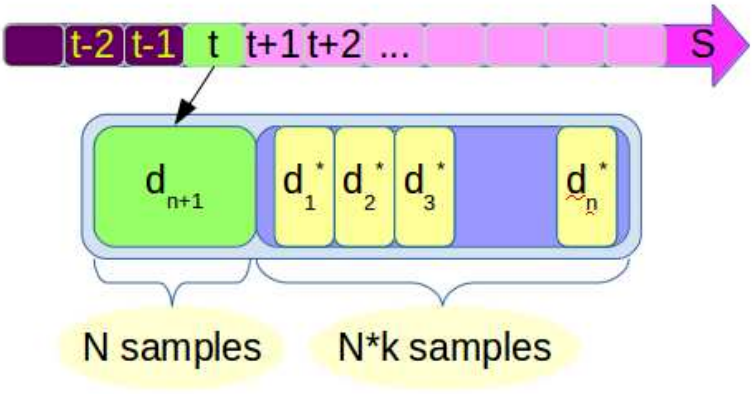


Figure 5.13: Schematic representation of the way batches for the incremental learning are organized. N is the size of real data batch, coming from stream, n is the number of already learned classes.

To deal with the outlined remarks, we design our experiments in the following way. Each time we receive a batch of stream data of size N, we generate $\frac{\min(n,k)*N}{n}$ data samples for each previously learned class. Here k is a parameter, fixed at the beginning of each experiment, and n is the current number of learned classes, so that total volume of generated data is equal to $\min(k,n) * N$ (Fig. 5.13). The size of the generated data batch depends on the number of classes we have already learned only when $n \leq k$. In each experiment, a fixed value of parameter k in the range between 1 and 9 is taken. The value of 1 corresponds to the case where the total amount of generated data is equal to the size of the

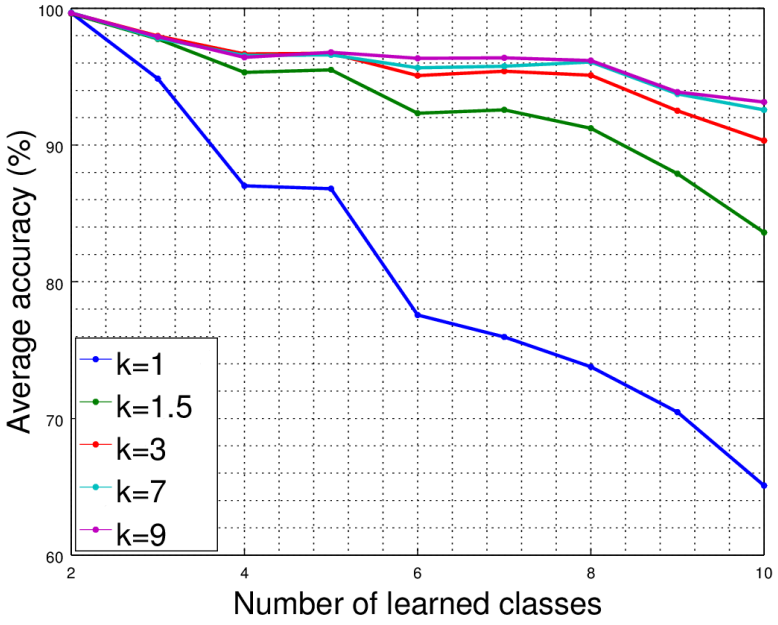


Figure 5.14: Accuracy of the incremental learning on MNIST with different values of scaling parameter k for data regeneration

received batch. The value of 9 in the 10 class classification problem represents the case where, for each already learned data class, the number of generated data is equal to the number of received data.

An important aspect to mention is that running the tests directly in streaming mode with just random initialization resulted in poor performance. To overcome this problem, the classification network was bootstrapped by pre-training for several epochs on the first two data classes and then passing to online mode.

Fig. 5.14 represents the results of incremental classification training on the MNIST dataset. The figure shows the performance of the proposed learning algorithm on the evolving dataset with data classes added in an incremental way. Each line is an average over fifty independent runs, corresponding to one of the five tested values of k.

Our method achieves a classification accuracy above 90% in a completely online adaptive scenario starting from $k = 3$, which is close to the state-of-the-art performance in the offline learning setting. The performance increases for higher values of k, i.e., bigger sizes of generated data. As a comparison point, in a similar experimental online setting on the 10 classes MNIST dataset, [CRDP12] obtained only 60% accuracy. We can also see that with $k \geq 3$, the accuracy decreases a little with every newly added class, but total forgetting never occurs.

Continual learning In our continual learning scenario, we achieved a maximum accuracy of 98.64% on MNIST (fig. 5.12) and 77.59% on 10-classes LSUN, which is comparable to the results of our

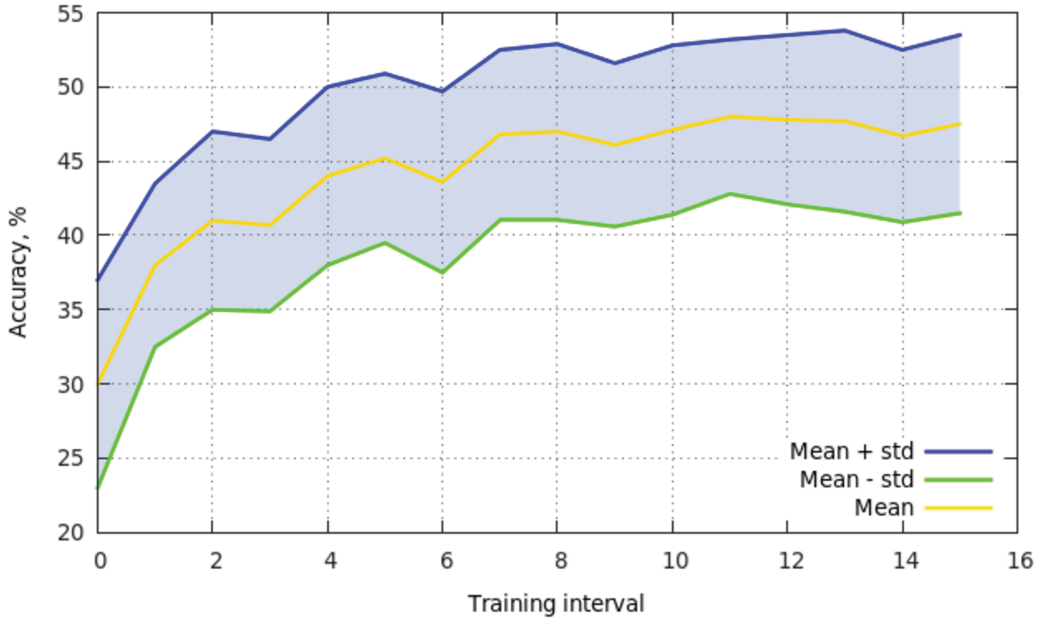


Figure 5.15: Classification accuracy during online stream training for the LSUN dataset. Each point on the graph corresponds to the average accuracy over 80 training intervals.

batch mode experiments where the classifiers were trained only on generated data from **pretrained** DCGANs. Comparing the results of stream online classification with batch mode classification allows us to quantify the loss of performance. To the best of our knowledge, there exist very few works on online classification on evolving streams of complex data and no established baseline for evaluation. Hence, the performance assessment score in batch learning mode is the only one we can provide here for comparison.

The online training on a stream is quite unstable for the LSUN dataset, and classification accuracy varies a lot from one interval to the other, as can be seen from the accuracy standard deviation (Fig. 5.15, light blue curve). Still, performing mean-filtering of classification accuracy over several successive intervals results in a curve that shows stable progression (the results for LSUN presented on Fig. 5.15 are averaged/mean-filtered over 80 successive training intervals). To find the reasons for the training instability in the stream scenario, we performed a test in a similar scenario with the only difference that the original images from the stream were used only to train generative models. The classifier, in turn, was trained with only generated data at the end of each interval (Fig. 5.16). The yellow curve on the diagram of Fig. 5.16 represents the average classification accuracy as a function of the average number of images ($\times 1000$) seen by each DCGAN. The red and blue lines represent respectively the average accuracy over pre-trained classes and the average accuracy over the classes appearing at some point in the stream. Fig. 5.16 shows how the amount of images fed to the DCGAN improves the classification accuracy. Each generative model received about 1 million real images before it started to have a positive influence on classification training, while after 5 million images per class,

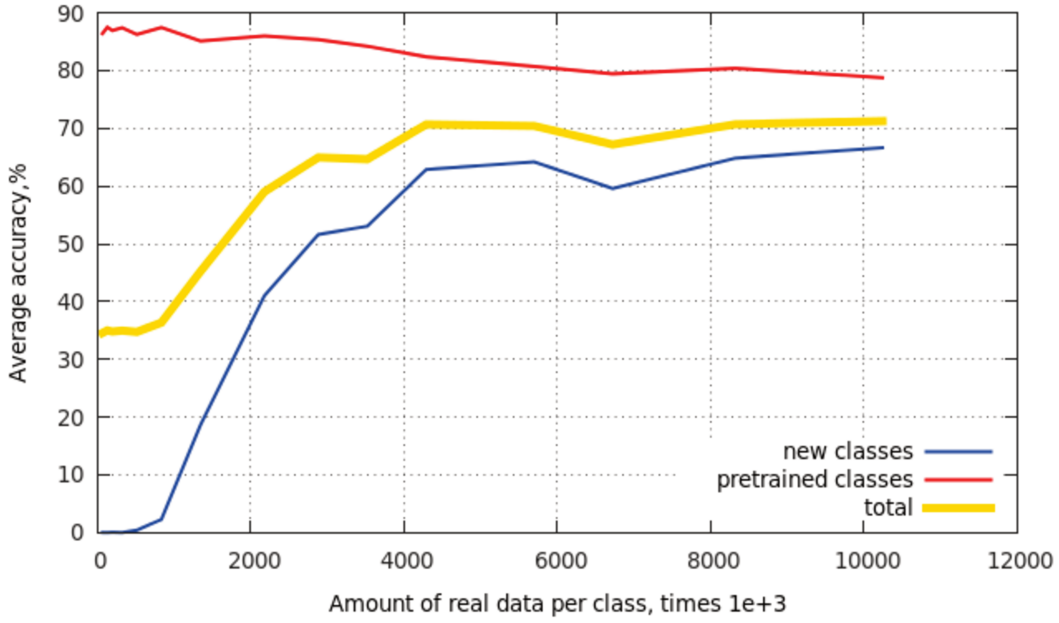


Figure 5.16: Average classification accuracy during stream training when the classifier is trained only on generated data. The curves correspond respectively to the average performance over all classes (gold), the average performance over classes pretrained before the beginning of the stream (red), and, the average performance over classes introduced during the stream (blue)

we do not see global improvements in classification accuracy. The training in such a scenario appeared to be much more stable than when using a mixture of real and generated data to train the classifier.

We think that the training instability on the LSUN dataset is due to the complexity of the database (for the MNIST dataset, the training is rather stable), but, also, to multiple image rescaling steps rendered necessary by the incapacity of DCGANs to work directly with full-size 256×256 images. The classification accuracy during training might also be limited because we use a pre-trained network on ImageNet to perform the feature extraction step for the classifier and that this feature extraction layer is not retrained in our stream scenario.

5.6 Discussion

In this chapter, we proposed a method able to learn online from streams of high-dimensional natural images efficiently. The proposed method is based on the idea of pseudo-rehearsal. It uses DCGANs as an additional memory module that allows us to approximate the distribution of the stream data and draw random samples from this distribution when corresponding data classes are missing. The experiments performed on the MNIST and LSUN datasets showed that our approach performs efficient incremental and continual learning from non-iid data streams while remaining well preserved from catastrophic forgetting.

We should note that training one generative sub-model per data class causes the size of the system to grow linearly with the number of already learned classes. Besides, our experiments demonstrated that the proposed learning framework is extremely slow to converge: on the 10-classes LSUN dataset, the model had to learn from over five million images per class to reach its validation accuracy limit. In real online learning applications, one rarely possesses such a large number of data elements per class. Moreover, the complexity of the problems to solve usually goes far beyond ten classes. Therefore it is clear for us that the proposed method, while indeed being a step towards efficient deep online learning, requires significant improvement to scale to the size of real applications. In the following chapter, we propose a set of techniques to tackle the described issues.

Chapter 6

Learning from large-scale unordered streams

6.1 Problem statement

In the previous chapter, we designed an approach that demonstrated good performance on incremental and simplified continuous data streams. However, we only trained our models on a relatively small classification problem that contained at most 10 classes. In this chapter, we extend this study to a significantly larger scale and work on a more realistic streaming scenario. The work described in this chapter is under revision as a full paper to the Computer Vision and Image Understanding journal (CVIU).

We define a data stream S as the result of sequential data acquisition from one or multiple sources, batch after batch. We suppose that data acquisition is performed by standardized sensors so that the data samples are all of the same nature and format. Since we consider classification problems, we suppose that during the training phase, all data samples come with corresponding class labels, while for testing, no labels are available. We also suppose that each data sample belongs to a single class, i.e., that it is associated with one single label. The latter condition is not mandatory for data stream classification and is rather a sub-case of the general learning case. However, we argue that learning from multi-label data does not increase the difficulty from the forgetting point of view and mostly requires network and data engineering. In our experiments, we, therefore, stick to the one label per image case.

We assume that small batches of data are continuously sent to the learning system from the dynamically changing environment. Let us denote by $\mathcal{D}$ the set of labeled samples from which we simulate the stream. Similarly to the definitions we provide in Chapter 2.1, we denote by $\mathcal{E} = \{l_1, \ldots, l_N\}$ the set of labels of all the available classes. To simulate the stream S we divide it into non-i.i.d. (in terms of class label distribution) time intervals I_t, each containing a number of distinct class labels $N_t < N$.

We denote $\mathcal{E}^t$ the set of N_t distinct labels that form I_t:

$$\mathcal{E}^t = \{l_1^{(t)}, \ldots, l_{N_t}^{(t)}\}$$

where the $l_i^{(t)}$ are the distinct class labels. The set of classes already seen by the system is referred to as $\mathcal{E}^{hist}$.

We simulate streams by sampling data according to their class label distribution. We use the notation

$$D_{\mathcal{E}} = \{(X,y) \in \mathcal{D} | y \in \mathcal{E}\}$$

to refer to the subset of data with the corresponding class label in $\mathcal{E}$. For each time interval I_t, we sample uniformly class labels from $\mathcal{E}^t$ and form associated data subsets $D_{\mathcal{E}^t}$. We simulate a batch b_s of data arriving from the streaming environment at time t by sampling uniformly the elements of b_s from $D_{\mathcal{E}^t}$.

As it was already mentioned, the main challenge of learning from streamed data is to be able to continuously learn from the new data while preserving the knowledge acquired on past data. With our notations, on each time interval I_t, we want to optimize the classification accuracy of the model on the data classes $\mathcal{E}^t$ present in that interval while preserving the model performance on historical classes $\mathcal{E}^{hist}$.

Such a way of defining the learning task imposes some serious constraints on the learning system. In contrast to the approach we proposed in the previous chapter when the variety of classes in the stream increases from tens to hundreds and even thousands, we cannot allow our model to grow linearly with the number of classes seen so far. Such a learning framework, therefore, requires a bounded size of the generative sub-system.

The most intuitive way to bound the size of such a generative system is to have a single multi-class generative model. Compared to a system containing one model per data class, a single model would allow the knowledge sharing between different classes. In practice, however, such generative models provided with the conditional sampling mechanism are not easy to build. Several methods based on adversarial training that match this requirement were recently proposed ([OOS16], [MO14]). Our preliminary experiments, however, showed that while having good capacity to approximate the data distribution and generate realistic samples, such methods require tens of training epochs through the full data even for very large datasets, and, thus, do not match the reactivity and learning efficiency requirements for online learning.

In the method we propose in this chapter, instead of using a generative model, such as a GAN, to supply (when needed) samples similar to the historical ones, we store enough historical data compressed with some loss into very low dimensional representations (or codes), by using a type of autoencoders specifically enhanced for this task. We train the main classifier with a mix of newly arrived stream data and historical data reconstructed from randomly selected stored representations. In the following, we call this approach of recalling historical data a "pseudo-generative" model.

We propose the use of autoencoders as pseudo-generative models to avoid catastrophic forgetting

while keeping the system's memory requirements low. Auto-encoders project input samples in low dimensional feature space by minimizing the reconstruction error. Instead of learning a generative model that tries to mimic a high-dimensional statistical distribution, which is known to be a difficult problem, we store projections via the autoencoder of samples from the input stream, in a number that is small but sufficient to alleviate the loss of memory. When training on a new batch of arriving data, we add reconstructed samples from the missing classes by using a part of the stored projections.

In the further sections of this chapter, we first present our approach of continuous learning from complex unordered streams based on pseudo-rehearsal powered by auto-encoders. To enhance the efficiency of the auto-encoders for this task, we propose a new loss function that takes into account our specific goal: the reconstructed samples should behave well with respect to reinforcing the classes already learned (and not merely minimize the reconstruction error). Following this logic, the loss function we put forward takes into account the error that the generated samples produce during classification. We continue by a discussion on the difficulty of training multi-class generative models on data streams. We point out that in such settings, one needs to retrain (pseudo-) generative models on their own generations, which may result in a decrease in generation/reconstruction quality. To limit this effect, we propose to modify the training procedure by introducing an adaptive gradient weighting scheme based on the "age" of the samples (i.e., the number of times a given sample was reconstructed). We then provide details on the datasets that we use for our extended experiments. We briefly introduce the 30-classes extension of the LSUN dataset and the large-scale synthetic data source that we use to generate streams of up to a thousand separate classes. We conclude the chapter by validating our method on the described datasets and by discussing the results.

6.2 Proposed method

As described in Sec. 4, one way to avoid forgetting in online learning is to use supplementary storage modules or models to "replay" historical data or to generate replacement data when necessary (Sec. 4.3). To do so, a supplementary component that serves for long-term memorization is required. A typical implementation consists in using a rehearsal mechanism that stores the most relevant instances of previously seen classes in a buffer. It is then employed to supply the classification model with historical data when needed. The main limitation of this approach is that training DL models on complex high-dimensional data requires a large number of data samples per class, especially in classes with high intra-class variability. This would require storing and reusing large amounts of data, which is impracticable in a massive real-time stream scenario due to storage limitations and high re-training cost.

Similarly to the approach, we presented in the previous chapter, instead of using the historical data directly, we adopt a solution that consists in approximating its distribution by a generative model that allows sampling labeled instances similar to real data at any moment of the stream.

In our experimental scenario, we train two separate models: a NN classifier model C and a generative model G. The classifier C receives data samples as input and outputs the probabilities of the input belonging to one of the already seen classes. Each time a new data class (not already in $\mathcal{E}^{hist}$) appears

in the stream, a new neuron is added to the output layer so that the latter constantly contains as many neurons as there are classes in the current state of the stream environment. The generator G is trained to produce replacement data approximating the distribution of the original data. It receives a latent code as input and outputs a tensor of the shape of the data.

One of the key goals of our approach is to remove the need for storing excessive amounts of historical data. However, experiments in Sec. 6.6.3 show that keeping a small number of real data samples per class is still essential to avoid progressive drifts in the distribution of generated data. Thus, to perform online learning, we initialize a small (compared to the size of the original dataset) buffer B^{hist} that serves as short-term memory support. It can be seen as a set of cells, one for each class already seen by the system, each storing a small portion of the most recent real data samples for the corresponding class (less than 1%).

Require: D : full data environment
Require: $\mathcal{E} = \{l_1, \dots, l_N\}$: set of class labels in D
Require: $S = \{I_t, t = 1, 2, \dots\}$: data stream
Require: s : random data sampler
Require: K : generated-to-real data ratio
Require: bs : batch size
  Initialize $\mathcal{E}^{init} = \{l_1, \dots, l_{N/2}\}$
  Initialize $D^{init} = D_{\mathcal{E}^{init}}$
  Initialize B^{hist} from D^{init}
  Initialize C and G
  Train C and G on D^{init} offline
  for I_t in S **do**
    for b^{stream} in I_t **do**
      $b^{old} \Leftarrow s(B^{hist}, bs)$ – get historical data batch
      $b^{gen} \Leftarrow s(G, K \times bs)$ – get generated batch
      $b \Leftarrow$ concatenate$(b^{stream}, b^{old}, b^{gen})$
      Update C and G on b
      Update B^{hist} from b^{stream}
    end for
  end for

Algorithm 2: The procedure of online learning from a data stream: for each time interval I_t the classifier C and the generative model G are updated with new batches of samples containing a mix of newly arrived data, historical data and generated data. The sampler $s(S, n)$ produces n data points from the source S with a uniform distribution over the set of classes already seen by the system.

In almost any real-life application that requires online learning, we are likely to have prior access to at least a small dataset representing our global task. The most natural approach, in this case, is to make use of this dataset to pre-train both models before training on the stream. In our experimental setup, we consider a similar case: we start with an initial training subset $D^{init} = D_{\{l_1, \dots, l_{N/2}\}}$ containing all the training data associated with half the number of classes in the full data environment. We pre-train

both C and G on D^{init}, then pursue with online training on the stream. The D^{init} subset is also used for fine-tuning different parameters by cross-validation (see Sec. 6.6).

During the online learning phase, for each time interval I_t, the system receives several batches b^{stream} of stream data. For each batch b^{stream}, we sample a batch b^{old} of the same size from B^{hist} and generate K batches using the generator G, where K is a predefined parameter (generated-to-real data ratio) that controls the preservation of the already learned classes. The resulting batches are then joined together and used to perform a gradient descent training step on both G and C. The buffer B^{hist} is then updated with the elements in b^{stream}. If a new class is introduced in b^{stream}, a new cell is added to B^{hist}. Otherwise, the oldest samples from the corresponding cells in B^{hist} are replaced by the elements in b^{stream}. The online learning procedure described above is summarized in Alg. 2.

6.3 Introducing classification error to train auto-encoders.

As it was mentioned in Sec. 6.1, we want the reconstructed data produced by the auto-encoder to influence the training process in the same way as the original data. To encourage this, we introduce an auxiliary loss that measures how far from each other are the outputs of the classifier (C) on the original (x) and reconstructed data ($G(x)$):

$$\mathcal{L}_{cl} = \left\| C(x) \; - \; C(G(x)) \right\|_2$$

We then train the autoencoders with the combined loss function

$$\mathcal{L} = \alpha_{cl}\mathcal{L}_{cl} + \alpha_{rec}\mathcal{L}_{rec} \tag{6.1}$$

where $\mathcal{L}_{rec}$ is the standard mean square reconstruction error usually used to train the autoencoders and α_{cl}, α_{rec} represent the trade-off between classification and reconstruction loss.

A similar additional classification loss to train Denoising Autoencoders was introduced in [ZSL12] with the main goal of performing a discriminative task when training the autoencoders on labeled data. Contrary to the loss we propose, the authors plug the classification layer directly to the output of the encoder and train the encoding part of the network in a standard "classification task" way by fitting the one-hot label vectors. Such a procedure only ensures the separation of classes in the code space, while not explicitly enforcing similar responses of the classifier on the original and reconstructed sample. Applying such evaluation to the stream classification scenario made us rethink the generative models' training objective. When training deep generative models one might want generated samples to be as much similar to real data as possible (memorize the data samples), to generalize on the dataset by understanding and reproducing samples holding the same concepts (describe the data distribution) or, independently from the actual "visual quality", to be well interpreted by a system other than human vision, e. g. to be well classified by a pre-trained classification model. While two first objectives can, at some scale, suit the desired application, it makes sense to optimize generative model to the

application of need directly. The most straightforward way to adapt generative models to given goals is to find a reasonable objective function that considers those goals.

6.4 Adaptive weighting for backpropagation

Since our system stores the most recent samples from the stream (batches b^{stream}) at each iteration as latent codes (buffer B^{codes}), the system continuously updates the autoencoders with the newly arrived data. Over several iterations, this procedure tends to produce in the autoencoders a forgetting effect similar to the one of the main classifier: classes not present in the current set of updating samples tend to be overwritten by the present ones. To avoid this, when updating the autoencoders, we need to include the data reconstructed from the codes already stored in the B^{codes} buffer.

However, recursively training the generator on its own reconstructed data causes a drift of the distribution of the generated samples, which is slowly diverging from the distribution of the real data. This process ends by producing synthetic samples that are harmful to training the main classifier. At the same time, adding data from the short-term memory buffer is not enough to prevent forgetting. To handle the described issue, we propose a penalization scheme that reduces the influence in the training of the samples that have gone through too many recursive regeneration steps. For each code stored in the buffer, we count the number of reconstructions r it has gone through (starting with 1 if the code was produced from the stream data). When evaluating the loss for a given data batch, we compute the weighted average error using the reconstruction counters as "importance" weights in the following way:

$$L = \left(\sum_{i=1}^{k} \frac{1}{r_i} \right)^{-1} \cdot \left(\sum_{i=1}^{k} \frac{l_i}{r_i} \right),$$

where l_i and r_i are respectively the error and the number of regeneration steps associated with the sample i in the batch, and, k the batch size. In proportion, as they are re-generated, codes see their "importance" weights progressively decrease. Thus, the codes that underwent the biggest number of regeneration steps, and, consequently, that are the most likely to have drifted from the original distribution get penalized the most. We also set a hard limit of $r_{max} = 10$ for the maximum number of regenerations. Setting all r_i equal to one (i.e., all samples are equally important for training) amounts to using the usual training loss.

6.5 Experimental setup for the extended study

6.5.1 Datasets

MNIST. In this chapter, we use MNIST only as a proof of concept dataset. We do not add any new information to the dataset. Furthermore, to make our experiments homogeneous between different datasets, we simplify the MNIST experiments compared to the previous chapter, and train all the models on the images unfolded into 1D vectors. All the pre-processing steps stay unchanged.

30-classes LSUN To perform more sophisticated online experiments, we extended the LSUN dataset that we used for the validation of our first approach by adding 20 object classes [1]. We balanced the classes in training set by selecting 100K images per class (indices 1 to 100,000 in the database indexing system) for training. In the original dataset presentation, a test set is only provided for the 10 classes of image scenes, 300 images per class. As this is not sufficient to verify the model's performance on such a large scale, we extracted 10K images per class to form a validation set (indices 100,001 to 110,000), and 10K images per class to form a test set (indices 110,001 to 120,000). Similarly to the experiments of the previous chapter, we applied a ResNet model pre-trained on ImageNet to the input data, and stored the 2048-dimensional obtained representations given by the "before before last" hidden layer. In a real stream scenario, such representations would be extracted online, as data arrives. In our case, we perform this extraction step offline as a way to simplify the stream learning procedure simulation. But it does not alter the problem settings.

Synthetic large-scale data stream To see how our approach behaves when the number of classes dramatically increases, and, as a direct consequence, when a large number of previously learned classes at some point gets missing from the stream, we introduce a synthetic data stream, denoted by **Syn-N** in the following, and counting up to $N = 1000$ data classes. To keep things comparable with the previous experiments using natural images, we use the same size of feature representations ($dim = 2048$).

Require: N : Number of classes in the dataset
Require: dim : Dimension of the data features
Require: d : Minimal distance between classes
  Initialize $\mu_1 \in \mathcal{U}[-1,1]^{dim}$
  for i in $range(2,N)$ **do**
    Initialize $\mu_i \in \mathcal{U}[-1,1]^{dim}$
    while $\min_{j<i} \left\| \mu_j - \mu_i \right\|_2 < d$ **do**
      for k in range$(1, i-1)$ **do**
        $\mu_i = \mu_k + d \frac{\mu_i - \mu_k}{\left\| \mu_i - \mu_k \right\|_2}$
      end for
    end while
  end for
  Initialize $M \in U[-1,1]^{dim \times dim}$
  $O =$ Gram-Schmidt(M)
  for i in range(1, N) **do**
    $D_i = \text{diag}(\mathcal{U}(0.5,1]^{dim})$
    $\Sigma_i = O^T D_i O$
  end for
  return $\{(\mu_i, \Sigma_i)\}_{i=1}^{N}$
  .

Algorithm 3: Syn-1000 dataset initialization. The notation $x \in \mathcal{U}(A)$ means that the value x is sampled from the uniform distribution on the set A

[1] http://tigress-web.princeton.edu/ fy/lsun/public/release/

Data	Classifier	Autoencoder		Params
		Encoder	Decoder	
MNIST	fc(784,256) ReLU fc(256,64) ReLU fc(64,10)	fc(784,256) bn ReLU fc(256,cs) bn ReLU	fc(cs,256) bn ReLU fc(256,784)	0.65M
LSUN, Syn-1000	fc(2048,784) ReLU fc(784,256) ReLU fc(256,30)	fc(2048,512) bn ReLU fc(512,128) bn ReLU fc(128,cs)	fc(cs,128) bn ReLU fc(128,512) bn ReLU fc(512,2048)	4M

Table 6.1: The architectures of the classifier and autoencoder used in this study for the MNIST, LSUN and Syn-1000 datasets.

In the proposed data synthesis procedure, each class follows a multi-dimensional Gaussian distribution. To generate the dataset we start by initializing the class centers μ_i for all the classes $i = \{1,\ldots,N\}$ in such a way that no two centers are closer than a threshold distance d:

$$\forall i \neq j, \left\| \mu_i - \mu_j \right\|_2 \geq d$$

We then compute the covariance matrices Σ_i, which are positive semi-definite and thus can be represented as $\Sigma_i = O^T D_i O$, where O is an orthonormal matrix, and D_i is diagonal with positive elements. To initialize the covariance matrices, we first generate a random square matrix $M \in \mathcal{U}[-1,1]^{dim \times dim}$ and perform Gram-Schmidt orthogonalization on it to obtain O. We then initialize random diagonal matrices D_i for all classes and transform them with the help of O to obtain Σ_i. Therefore we can now sample data for N classes from $\mathcal{N}_{dim}(\mu_i, \Sigma_i)$ (see Alg. 3).

6.5.2 Model architectures and optimization setup

Due to the requirement on learning reactivity when training models on data streams, in our experiments we use shallow models, which are easy to implement and fast to train. Their architectures are provided in Table 6.1, with **fc** standing for fully-connected linear layers, **ReLU** for Rectified Linear Unit ([NH10], **bn** for 1D batch normalization ([IS15]) and **cs** for code size in autoencoders. For each dataset, both the autoencoder and the classifier are separately optimized using Adam ([KB14]) with $\alpha = 10^{-3}$, $(\beta_1, \beta_2) = (0.9, 0.999)$ and weight decay $= 10^{-5}$. When training is performed in a static or offline scenario, the results obtained on the MNIST and LSUN datasets are close to the state of the art as shown in the next subsection.

$\alpha_{cl} \backslash \alpha_{rec}$	0	0.001	0.003	0.01	0.03	0.1	0.3	1	3	10
0	x	96.8	96.9	96.7	96.8	96.9	97.0	96.8	96.9	96.7
0.001	16.9	96.8	**97.2**	96.8	96.8	96.7	96.9	96.8	96.8	96.7
0.003	61.56	96.8	96.8	96.8	96.8	96.9	96.8	96.8	96.8	96.8
0.01	93.97	96.7	96.8	96.8	96.7	96.7	96.7	96.8	96.9	96.8
0.03	96.1	96.7	96.8	96.7	96.8	96.8	96.9	96.9	96.9	96.7
0.1	96.2	96.7	96.6	96.7	96.7	96.9	96.8	96.7	96.8	96.8
0.3	96.4	96.8	96.6	96.8	96.7	96.7	96.9	96.9	96.6	97.0
1	96.5	97.1	97.0	96.7	96.6	96.7	96.7	96.7	97.0	96.8
3	96.4	96.9	97.0	96.8	96.6	96.5	96.7	96.8	96.7	96.9
10	96.3	96.8	96.9	97.1	96.9	96.7	96.8	96.7	96.7	96.9

Table 6.2: Validation set accuracy on the pretrain part of the MNIST dataset, obtained during the grid search aimed to optimize the trade-off between classification and reconstruction losses in the autoencoders.

6.6 Offline experiments

Our online learning framework (see Sec. 6.2) depends on a number of hyper-parameters such as s^{hist} (size of the historical buffer that stores most recent samples), s^{codes} (size of the buffer that stores auto-encoded samples), cs (size of the hidden codes layer in the autoencoder) and $(\alpha_{cl}, \alpha_{rec})$ (the trade-off between classification and reconstruction loss for autoencoder training). The static parameters cs and $(\alpha_{cl}, \alpha_{rec})$ are optimized by cross-validation in the offline scenario, while the sizes of the buffers are optimized in an incremental learning scenario that matches the required characteristics of the online learning system.

For each dataset, we select hyper-parameters that provide the best classification scores on corresponding validation subsets, as described in the following subsections.

6.6.1 Impact of the classification loss term

We first adjust the tradeoff between classification and reconstruction loss in the objective function used to train the autoencoder Eq. (6.3) by performing a grid search on $(\alpha_{cl}, \alpha_{rec})$, making each parameter vary in the range $(0, 10)$. The grid search is conducted for each dataset by training autoencoders with a code size of 32, and by cross-validating each pair $(\alpha_{cl}, \alpha_{rec})$.

One of the most important criteria for the choice of generative model in our online learning approach is the training reactivity, i.e. the ability of the model to learn fast and perform well when trained with few data. Therefore, we select the values of $(\alpha_{cl}, \alpha_{rec})$ for which the model trained for a single epoch on the training set gives the best average accuracy on the validation set over ten independent runs. The experiments provided the following trade-off values $(\alpha_{cl}, \alpha_{rec})$: $(0.001, 0.003)$ for MNIST, $(0.1, 1)$ for LSUN, and, $(0.01, 3)$ for Syn-1000 (Tables 6.2, 6.3 and 6.4).

As shown in the tables 6.2 and 6.4, Syn-1000 and MNIST datasets are too simple, and the use of the reconstruction loss alone is practically enough to achieve optimal performance. As for the LSUN

$\alpha_{cl} \setminus \alpha_{rec}$	0	0.001	0.003	0.01	0.03	0.1	0.3	1	3	10
0	x	3.4	3.6	5.7	10.7	17.3	21.6	25.2	26.0	27.1
0.001	75.1	77.3	79.2	78.8	77.7	76.9	67.5	60.2	52.5	42.6
0.003	82.0	81.2	82.1	82.3	82.0	81.0	78.7	72.3	63.4	52.7
0.01	83.8	84.1	83.0	82.4	83.1	83.3	82.8	80.1	74.6	64.4
0.03	83.1	84.7	84.2	83.9	83.2	83.9	84.0	83.4	80.8	73.0
0.1	85.1	83.5	84.8	84.5	84.8	84.7	83.6	**85.3**	83.8	82.3
0.3	84.9	84.3	84.5	84.8	84.7	83.5	84.1	85.0	83.2	84.4
1	84.5	83.8	84.3	84.2	84.5	83.8	84.6	85.0	84.5	83.9
3	84.2	83.6	85.0	84.1	84.7	84.1	84.1	84.4	84.8	84.6
10	84.1	84.6	84.5	82.6	84.6	84.5	85.1	84.9	84.2	84.8

Table 6.3: Validation set accuracy on the pretrain part of the LSUN dataset, obtained during the grid search aimed to optimize the trade-off between classification and reconstruction losses in the autoencoders.

$\alpha_{cl} \setminus \alpha_{rec}$	0	0.001	0.003	0.01	0.03	0.1	0.3	1	3	10
0	x	rand	58.9	70.1	72.0	73.1	74.0	74.1	75.0	74.9
0.001	56.8	61.4	64.8	68.3	71.3	74.2	75.9	76.0	75.3	74.7
0.003	56.9	58.9	62.4	65.6	70.0	73.4	75.5	76.2	76.1	74.4
0.01	53.5	55.4	57.6	61.9	64.2	69.6	72.1	75.6	**76.3**	75.4
0.03	56.0	56.4	57.3	57.5	61.0	64.9	68.7	72.2	75.2	76.0
0.1	57.9	59.2	59.9	61.5	62.1	62.3	64.5	68.1	71.4	75.7
0.3	54.9	58.7	56.8	52.2	55.9	56.6	60.8	64.5	68.1	71.9

Table 6.4: Validation set accuracy on the pretrain part of the Syn-1000 dataset, obtained during the grid search aimed to optimize the trade-off between classification and reconstruction losses in the autoencoders.

Code	2	4	8	16	32	Base
MNIST	91.3	95.3	97.1	**97.7**	98.2	99.4
LSUN	57.9	72.9	79.5	83.3	**89.1**	90.7
Syn-1000	79.8	91.1	93.3	**94.5**	94.6	96.1

Table 6.5: Accuracies obtained on the validation sets of MNIST, LSUN and Syn-1000 depending on the code size employed by autoencoders. The Base dimension is 784 for MNIST and 2048 for LSUN and Syn-1000.

dataset, full cross-validation performance for each pair $(\alpha_{cl}, \alpha_{rec})$ shown in Table 6.3 is of a particular interest.

As we see from this data, the classification term of the loss proved very useful for more complex/difficult datasets such as LSUN. For the latter, the gradient induced by the classification term of the loss dominates the reconstruction one after a few epochs of training and, as such, contributes to creating more representative codes, which increases the performance.

These values allow us to make an interesting observation about how the influence of each term in the loss function (6.3) changes depending on the nature of the data. We find that while for the MNIST and Syn-1000 datasets the reconstruction error alone is enough to obtain high accuracy (we only get small improvements by adding the classification term), on LSUN the feature reconstruction criterion alone results in an accuracy of only 27%, compared to 85.5% with the optimal combination of two losses. We think that this effect is due to the extremely small norms of the feature vectors extracted by ResNet from LSUN, which results in small L_2 distances between data samples and corresponding reconstructions, leading to vanishing gradients. Classification loss, in turn, computes the error in the space of classifier output. This involves a supplementary mapping that ends up solving the problem of small gradients. Therefore, we assume that the classification term in the autoencoder objective function, in addition to its main use, also performs an adaptive parameter normalization in the hidden layers of the model.

6.6.2 Code size in the autoencoders

As it was already discussed in Sec. 4.3.2, we use autoencoders to reduce the overall storage requirements of the learning system. In the proposed online learning scenario, one has to store the hidden encoded representations for the duration of the stream. Therefore, the choice of the parameter cs (code size) has a direct influence on the storage requirements. For each dataset, we perform a grid search over cs values in an offline manner. We train a classifier and autoencoder on the training split and evaluate models on the validation split. The main goal of this search is to find the smallest value of cs that provides an accuracy higher than a predefined threshold that we set to 98% of the offline classification performance. The results of this evaluation, averaged over 10 independent runs, are shown in Table 6.5 with the selected values highlighted in bold.

6.6.3 Impact of short-term memory on autoencoder behavior

As it was discussed in Sec. 6.2 and 6.4, using pseudo-rehearsal alone to train models in an online setting can result in progressively growing error in a multi-class generator. While adding some real historical data to the pseudo-rehearsal data can solve the problem, storing and reusing historical data for large-scale online learning can be heavy and computationally inefficient. To limit the storage requirements of the online learning system, we bound the maximum size of the historical buffer B^{hist} to 1% of full data size. A very natural question arises: the stored historical data alone are not enough to train all the models without using more sophisticated methods? In this section, we investigate how autoencoders act when trained on their own reconstructions and show that the short-term memory mechanism (using B^{hist}), while not sufficient to solve the problem, can help the autoencoder to avoid forgetting in the online classification scenario.

To evaluate the impact of short-term memory, we first pre-train a classifier and a multi-class autoencoder until convergence. We then pass full training set through the autoencoder, replace data by obtained reconstructions, and retrain the autoencoder on it. This process is repeated for 100 epochs. To measure the forgetting effect, we compute the accuracy of the classifier on the reconstructed validation set after each training epoch. To evaluate the importance of historical data for the described process, we redo the same experiment preserving a small portion of original data for each class (B^{hist} buffer). Finally, to demonstrate that short-term memory alone is not enough to reach desired capacities, we train the autoencoder from scratch on B^{hist} and register the maximum classifier accuracy on the reconstructed validation set. Due to the simplicity of the MNIST dataset, for which training a randomly initialized classifier offline on no more than 50 images per class (size of the streaming batch in our online experiments) provides an accuracy of more than 85%, we confirmed this hypothesis on the LSUN dataset.

Fig. 6.1 shows the results of this evaluation. We can see that retraining the autoencoder on its own reconstructions (blue line) results in a fast decrease of classification accuracy, while the addition of a small portion of the original data to the reconstructed data (orange line) stabilizes training and allows to avoid forgetting in the autoencoder completely. Training models separately either on the reconstructed data or on a small portion of the original data results in forgetting in the first case or overfitting in the second. Using them together solves both problems while requiring only a minimal computation overhead.

6.7 Experiments on high complexity online classification tasks

We perform our main validation on the MNIST, LSUN and Syn-1000 datasets simulated as non-i.i.d. streams (see Sec. 6.1). We consider that a stream is divided in time intervals I_t, each formed of 100 (for MNIST) or 300 (for LSUN and Syn-1000) sequentially arriving batches (of size $bs = 50$ samples each) from at most 10% of all the available classes (respectively 1, 3 and 100 classes for MNIST, LSUN and Syn-1000).

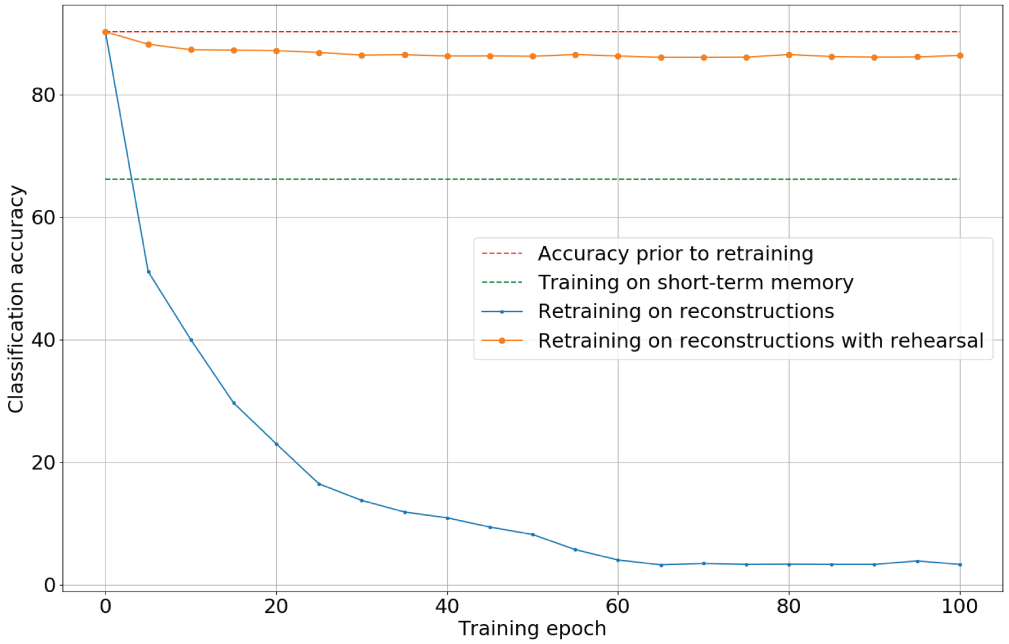


Figure 6.1: On the LSUN dataset, the effect of growing reconstruction error in autoencoders (leading to catastrophic forgetting) when trained on their own reconstructions (blue line) and the positive effect rehearsal has on this process (orange line).

As described in Sec. 6.1, part of the incoming data is stored for later use: the buffer B^{hist} contains the latest 50 raw samples from each class, and the buffer B^{codes} contains the latest 5,000 encoded representations from each class. These values are obtained by cross-validation, as explained in Sec. 6.6. As described in Alg. 2, for each batch from the stream we randomly sample a single batch from B^{hist} and $K = 18$ batches from G (reconstructed from B^{codes} by the auto-encoder), concatenate them and feed the obtained batch of size 1000 ($= (1 + 1 + 18) \times 50$) to the learning system.

For each dataset, we simulate the stream for 1000 intervals and compare the performance of five different online learning methods: naive training on stream data alone (*Pure stream*), training with stream data and limited rehearsal i.e. injections of data only from B^{hist} (*Rehearsal*), and three configurations of the proposed approach — the generator G trained using the reconstruction loss only (*Our method,* $\alpha_{cl} = 0$), using the classification loss only (*Our method,* $\alpha_{rec} = 0$) and using the mixture of classification and reconstruction losses (*Our method,* α_{cl}, α_{rec}). To evaluate the performance of the learning system, at the end of each interval I_t, we measure the classification accuracy obtained by the current classifier on the subset of the test set that contains all the already seen classes (from $\mathcal{E}^{hist}$). Fig. 6.2, Fig. 6.3 and Fig. 6.4 show the results of these continuous learning experiments.

With no surprise, naive online training on stream data results in unrecoverable catastrophic forgetting for each dataset. An interesting observation can, however, be made on the Syn-1000 dataset, Fig. 6.3. After 150 stream intervals, the overall performance starts improving and saturates after 500 intervals at an accuracy of about 70%. This phenomenon can be explained by the nature of the synthetic

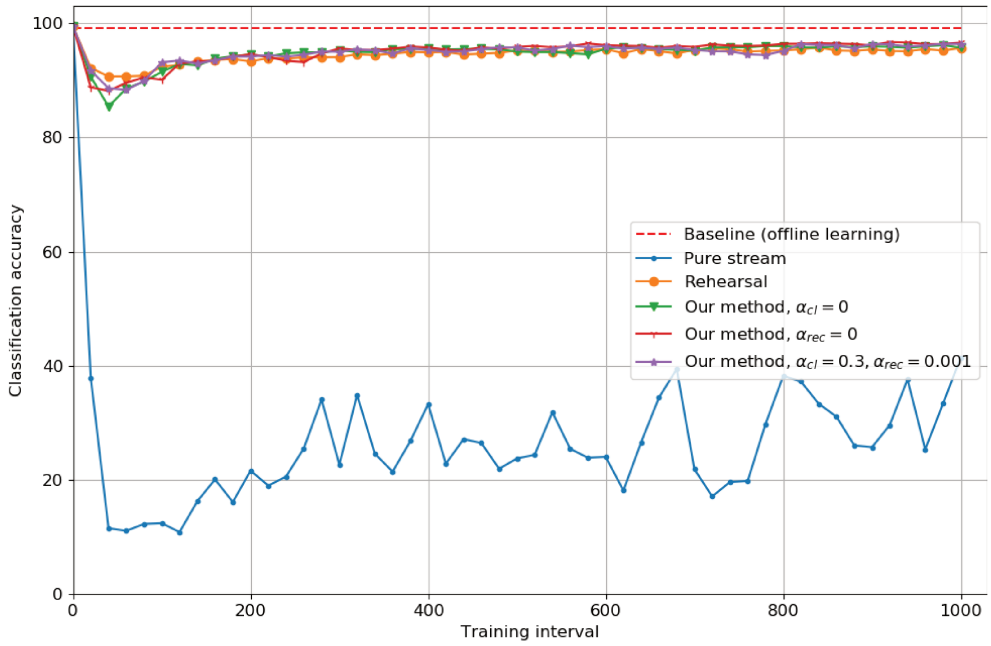


Figure 6.2: Stream training on Syn-100. The results are shown for naive learning from stream data (blue), learning with limited rehearsal (yellow), our method with the autoencoder trained with reconstruction loss only (green), with classification loss only (red) and full method (purple).

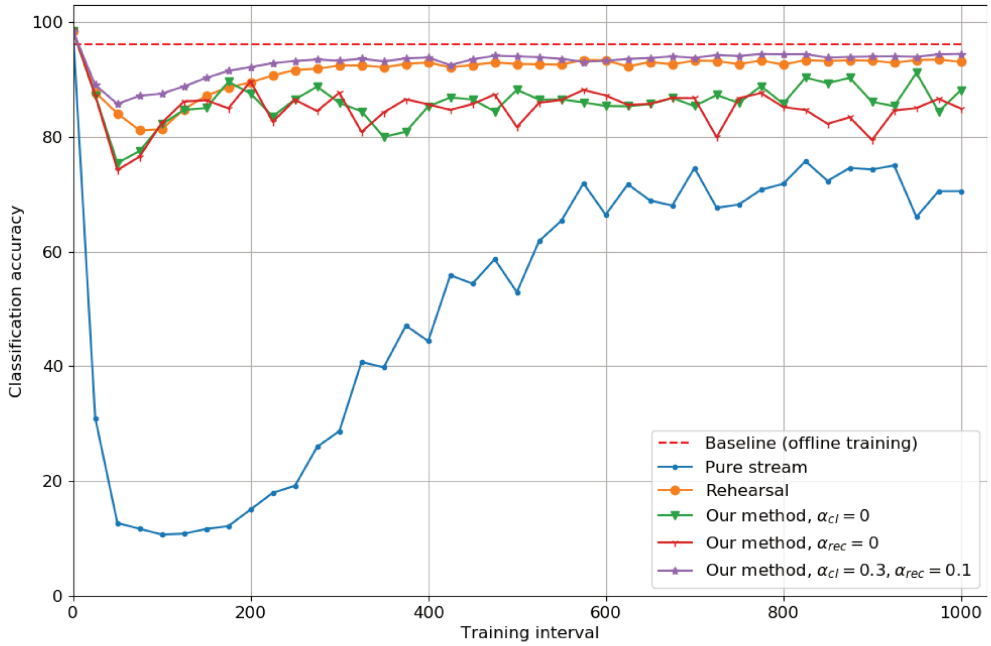


Figure 6.3: Stream training on Syn-100. The results are shown for naive learning from stream data (blue), learning with limited rehearsal (yellow), our method with the autoencoder trained with reconstruction loss only (green), with classification loss only (red) and full method (purple).

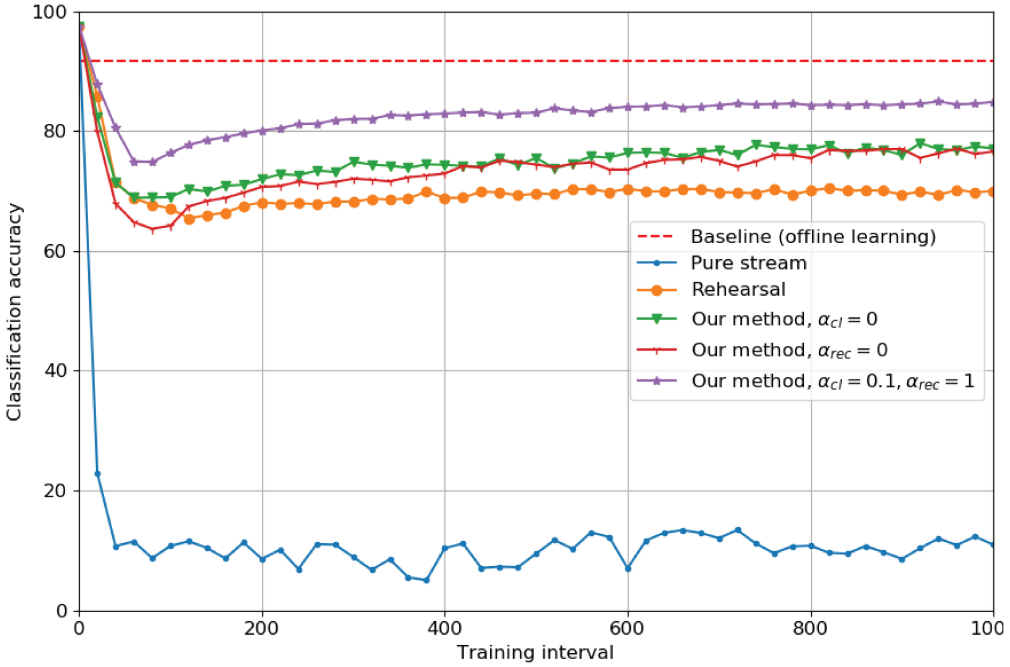


Figure 6.4: Stream training on LSUN. The results are shown for naive learning from stream data (blue), learning with limited rehearsal (yellow), our method with the autoencoder trained with reconstruction loss only (green), with classification loss only (red) and full method (purple).

classes, which take the form of well-separated high-dimensional Gaussian blobs in the feature space. Performance is low in the beginning when many new classes are added, but, once a majority of classes have been seen, the model gradually converges towards a sub-optimal but much better solution.

For similar reasons, limited rehearsal performs well on Syn-1000, resulting in stable and accurate learning comparable to our main method and approaching the offline learning baseline (relative error within 3%). The experiments on Syn-1000 nevertheless show that our approach efficiently scales to problems with a large number of separate classes.

On the MNIST dataset, we can see in Fig. 6.2 that our method and limited rehearsal have similar performance (accuracy of about 96.5%), which is $\leq 4\%$ lower than the offline learning baseline. As for the Syn-1000 dataset, this is due to the fact that the dataset is too easy for the employed classifier. However, even for such a simple database, the catastrophic forgetting phenomenon is very strong (blue curve for pure stream training).

The LSUN dataset is the most difficult of all the presented cases in terms of data complexity. The results of our experiments on LSUN, shown in Fig. 6.4, confirm that the full approach proposed in this paper, with the mixed loss function to train the autoencoder and the mechanism to compensate the negative impact of recursive reconstructions, significantly outperforms limited rehearsal and approaches based on a single-term loss function. On the test set, for the last 100 stream intervals, our approach reaches an average accuracy that is only 5.7% lower than the accuracy of the offline training

baseline. To the best of our knowledge, our results can be considered as state-of-the-art in online classification training on the LSUN dataset simulated as a continuous non-i.i.d. stream.

Chapter 7

Conclusion

In this thesis, we studied the problem of online learning from massive streams of complex high-dimensional data, with an application to online classification in streams of images. We first tackled the case of incremental data streams where the tasks (i.e., the groups of classes to categorize) appear in the stream one by one, allowing a sequential treatment/learning of each task till convergence. We then proceeded with the more general scenario of a continuous stream, where we make no assumption about the distribution of classes in the stream and their order of appearance.

We started our study with an extended review of the currently existing online learning methods. This allowed us, in the first place, to focus on Neural Networks as the main building blocks of our learning system. For performance and scaling reasons, Deep Learning based techniques have been predominant in the domain of high-dimensional multimedia data classification for the last decade. However, training Neural Networks on non-i.i.d. streams is a non-trivial task that requires handling the well-known effect of catastrophic forgetting while incorporating new knowledge. Our literature review provides a detailed discussion of the existing ways to address the catastrophic forgetting phenomenon. The analysis of the literature convinced us that the methods based on the usage of external memory units (that act as explicit or approximate storage of historical stream data) are the most adapted to the general continual learning problem.

As a first technical contribution, we developed a framework for training a DNN classifier on incremental data streams. The learning system we put forward uses data regeneration with DCGANs to compensate for the absence of the historical data and avoid catastrophic forgetting. The proposed approach, therefore, does not require the explicit storage of historical data.

We justify the choice of DCGAN models by showing that they have generalizability and representativity abilities on natural image datasets. Our experiments confirm that DCGAN-generated data can be used instead of the original data to train a classifier with good generalization properties. We validate our online-learning-on-stream method on the MNIST and LSUN datasets by showing that it can efficiently learn to classify complex image data from a time-evolving stream, with no need to store historical data. Our method showed on one side a strong capacity to adapt to unseen data classes appearing

at a different time of the stream, and, on the other side, did not lead to catastrophic forgetting of previously seen data.

However, this approach suffers from several limitations. First, training both generative networks and classifier requires having a sufficiently large amount of original data from each presented class. This requirement is often not satisfied in real-life applications that deal with data streams. Secondly, our learning system requires training one generative model per data class. This implies that the size of the full model grows linearly in proportion as the number of classes increases. Therefore, such an approach cannot scale to complex real-life problems with thousands of data categories.

The second technical contribution of the thesis aimed at overcoming the mentioned problems and extend the application of our work to the more general case of continual learning with memory constraints. In the proposed approach, we used pseudo-generative models (auto-encoders) instead of DCGANs to approximate the stream data distribution. Auto-encoders demonstrated the ability to incorporate new knowledge significantly faster than DCGANS. Also, they provide a straightforward way to perform conditional regeneration of missing data during training from a single model.

We argued that instead of searching for a precise approximation of real data, one should rather train generative models to extract data representations that make sense for an application of interest (classification in our case). To enforce this idea, we introduced an objective function that helps (pseudo-)generative models capture classification-relevant information.

Our approach shows stable training (low variation in validation results) and state-of-the-art performance on the unordered non-i.i.d. streams obtained from the MNIST and LSUN datasets. Moreover, we demonstrated the scalability of this approach on a much larger synthetic dataset with up to 1000 separate classes.

Employing the auto-encoders in our learning framework involves storing very compact data representations (codes). Thus, when necessary, we approximate real data from codes, which is a good compromise allowing us to avoid building online generative models that can be dynamically updated (a very difficult task for high dimensional distributions).

Lifetime continuous learning is an essential characteristic of any biological learning process. In contrast to artificial intelligence models, the biological brain can continuously incorporate new knowledge from a dynamically changing environment without suffering from catastrophic forgetting. Handling catastrophic forgetting in continual learning is thus a crucial step to provide machines with close-to-human intelligence. This work was conducted as a step towards Artificial General Intelligence (AGI).

Being critical towards the continual learning approach proposed in this Ph.D. work, we can mention several leads for further research and improvements.

For the LSUN dataset, all the experiments were performed on the feature maps extracted by the ResNet pre-trained on ImageNet. While the goal of feature extraction was to accelerate learning by reducing the data dimensionality, in theory, this procedure could significantly simplify the learning task. Indeed, similarly to LSUN, ImageNet is a large-scale natural images dataset, with several classes overlapping between the two. While the datasets significantly differ, we can hypothesize that such a feature extractor already provides a good inter-class separation due to this class overlap, making us

solve a much easier problem when training the classifier. One way to verify this hypothesis would be to analyze the statistical properties of the LSUN data classes mapped to the feature space.

As it was already mentioned, to be able to learn from unordered streams and to scale to large problems, we had to pass from generative models (GANs) to pseudo-generative models (auto-encoders). This choice was mainly motivated by the slow convergence of GANs and the unstable training of its versions that allow conditional sampling. Despite this choice, we argue that generative models have a huge potential and are much closer than the AEs to the operating memory in the biological brain. First, generative models do not require explicit knowledge storage, even with a reduced dimension as in AEs. They aim to learn the distribution of the input data and allow us to perform random sampling from a dense approximation of that distribution, rather than reconstructing already seen data samples from the sparse space of codes. This characteristic, in our opinion, should allow a significantly better generalization over the whole distribution. However, to train high-dimensional generative models on the fly, one needs to design a learning procedure with efficient knowledge transfer that would accelerate the incorporation of new information based on what the model already knows.

In addition to supervised learning, biological systems can learn in unsupervised, semi-supervised, and self-supervised manner. The capacity to learn with no or limited access to annotations is essential in continual learning where those annotations are rarely available during training. One of the possible incremental adjustments to our work would be to make use of the temporal dependencies in the streams to make the learning procedure less supervised. Indeed, in this work, we mostly considered streams with no predefined order in the data retrieval process. In various real-life applications, learning agents would be exposed to continuous environments, potentially with many redundancies. Such dependencies can facilitate the label attribution during the learning phase, thus leading to weaker supervision requirements for the learning system. As an example, when a new object is introduced in a continuous video, attributing a label to a single frame containing the object should be enough to label all the following frames where the object is present.

We argue that for large-scale real-life applications, an online learning system must be able to perform efficiently in a changing environment and should be capable of adapting its architecture to a dynamically expanding environment. Moreover, such a model should preserve acquired knowledge not only by approximating and replaying historical data but also by imposing relevant constraints on its parameter space. We thus believe that the potentially best approach to perform human-like life-long learning should be built upon all the three, for now, separate, types of online learning approaches: evolving architectures, regularization-based, and dual-memory-based.

As we discussed in Chapter 4, most regularization-based approaches are designed to solve the incremental multi-task learning. The proposed frameworks in their majority suffer from a significant limitation – due to the learning constraints, at testing time, they require to know the nature of the task they are currently solving. One of the potential solutions to this problem is to automate the task attribution and the detection of new tasks and to perform the "smart" selection of the sub-models/learning sub-mechanisms depending on the detected task. One could do this by training a new neural network that classifies the tasks and learns to attribute corresponding importance weights to each of them. Such

a network would also need to actively detect all the types of concept drift in the data distribution to efficiently switch between tasks.

Last but not least, working on all the discussed points requires a well-established procedure for stream learning evaluation. In our experiments, we evaluated the online learning performance of our method by measuring the classification accuracy of the model at the given point on the test set containing all the learned classes. However, online learning efficiency cannot be estimated by this alone. Indeed, learning from data streams is a complex problem which, depending on the application of interest, might require measuring short- and long-term forgetting effects, evaluating the quality and pace of the new knowledge acquisition, estimating the dynamics of the working memory requirements, etc. We, therefore, argue that the domain of continual learning requires further efforts in establishing the evaluation procedures and metrics. Furthermore, to make the online learning studies consistent and comparable, it is essential to build a reproducible data streaming procedure associated with a representative enough collection of data.

Résumé étendu

Au cours des dernières décennies, les ordinateurs et les algorithmes sont devenus une partie essentielle de nos vies. Dans les industries, ils sont utilisés pour augmenter la productivité par des milliers de fois, économisant ainsi des milliards d'heures de travail humain. Ils sont activement employés dans les services publics et privés tels que le transport et la santé, améliorant nos routines quotidiennes et la qualité de vie. De plus, les technologies informatiques sont entrées dans la vie privée et ont changé la façon dont les gens se socialisent, apprennent et occupent leur temps libre.

Ce succès est principalement dû à la capacité des ordinateurs modernes à travailler avec des quantités colossales de données, à des algorithmes efficaces de traitement des données et à la capacité croissante des systèmes informatiques. Cependant, la partie la plus importante des solutions aux problèmes réels jusqu'à maintenant était basée sur des modèles théoriques et une algorithmisation détaillée de chaque tâche. Ces modèles sont généralement développés et mis en œuvre par des spécialistes des domaines correspondants, des ingénieurs et des scientifiques. De telles solutions souffrent de plusieurs limites: elles sont difficiles à concevoir, nécessitent un nombre considérable de paramètres à affiner et sont généralement basées sur la compréhension humaine du phénomène, qui peut être très approximative et imprécise. Par conséquent, ces méthodes sont difficiles à généraliser en dehors de la tâche pour laquelle elles sont conçues.

Plus important encore, lorsqu'il s'agit de problèmes considérés comme "faciles" par un humain moyen, par exemple, lire les écritures à la main, détecter des objets sur l'image ou reconnaître une chanson dès les premières secondes d'un enregistrement, algorithmisation directe du processus est souvent extrêmement compliquée. Par exemple, la détection, la segmentation et la classification d'objets à partir d'images naturelles sont des tâches intuitives pour l'homme. Cependant, il est difficile de donner à un ordinateur des instructions pas à pas pour effectuer ces tâches. Cette complexité d'algorithmisation est la raison pour laquelle les tâches décrites forment tout un domaine de recherche scientifique appelé Vision par Ordinateur. Ce domaine a été étudié pendant des décennies pour atteindre des performances proches de l'homme dans les tâches liées à la vision.

Contrairement aux approches basées sur la modélisation et l'algorithmisation, l'apprentissage par ordinateur (dite Machine Learning ou ML) vise à obliger les machines à effectuer ces tâches sans donner d'instructions explicites. Au lieu de cela, les modèles ML extraient les connaissances directement des données brutes et des annotations correspondantes, si celles-ci sont fournies. Ainsi, le concept de base de ML est d'approcher un phénomène physique en récupérant et en organisant des connaissances basées

sur des observations. La plupart des méthodes ML stockent les connaissances dans les paramètres du modèle. Ces paramètres sont optimisés lors de la phase d'apprentissage.

Contrairement à l'apprentissage des systèmes biologiques qui peuvent apprendre en continu au cours de la vie, la plupart des approches ML modernes apprennent de manière statique à partir d'un ensemble de données prédéfini d'une taille limitée. À l'ère de la robotique, des réseaux sociaux et des gadgets personnalisés collectant en continu les informations des utilisateurs, la demande de systèmes capables d'apprendre en temps réel à partir de grandes quantités de données provenant de sources multiples augmente rapidement. En raison de ce besoin croissant, le domaine de l'apprentissage en ligne qui vise à intégrer de nouvelles connaissances dans le système d'apprentissage en temps réel a récemment commencé à recevoir une attention significative de la communauté ML.

Dans cette thèse, nous proposons une nouvelle approche basée sur l'apprentissage profond (dite Deep Learning ou DL) pour la classification en ligne sur des flux de données de grande dimension. Ces dernières années, les réseaux neuronaux (dits Neural Networks ou NN) sont devenus le principal élément constitutif des méthodes de pointe dans divers problèmes d'apprentissage automatique. Cependant, la plupart de ces méthodes sont conçues pour résoudre le problème d'apprentissage statique, lorsque toutes les données sont disponibles en même temps au moment de l'apprentissage. L'exécution du DL en ligne, et en particulier la classification en ligne à l'aide des réseaux de neurones, est exceptionnellement difficile. La principale difficulté est que les classificateurs basés sur NN reposent généralement sur l'hypothèse que la séquence de lots de données utilisée pendant l'apprentissage est stationnaire, ou en d'autres termes, que la distribution des classes de données est identique pour tous les lots (hypothèse i.i.d.). Parce que la rétropropagation tend à renforcer les classes présentes dans le lot actuel, lorsque cette hypothèse ne tient pas - ce qui est une situation probable dans un environnement d'apprentissage en ligne - les réseaux de neurones ont tendance à oublier les concepts qui ne sont temporairement pas disponibles dans le flux. Dans la littérature, ce phénomène est connu sous le nom d'oubli catastrophique.

Les potentiels avantages d'un système d'apprentissage en ligne, par rapport à un système d'apprentissage "sur place" à partir d'un ensemble de données statique prédéfini, sont nombreux. La capacité d'apprendre en continu fournit au système d'apprentissage la capacité de s'adapter aux changements de l'environnement et d'explorer des choses jamais vues auparavant. En outre, un tel système devrait également être capable de s'adapter à l'évolution des tendances, des tâches, des mises à jour des logiciels et du matériel, dans la condition critique de ne pas perdre les connaissances acquises précédemment. Les applications d'un système d'apprentissage en ligne peuvent varier de la prédiction des hashtags des images en tenant compte des dernières tendances à la création de robots humanoïdes, qui peuvent s'adapter à l'environnement réel riche et en évolution rapide.

Comparé à l'apprentissage sur les données statiques, l'apprentissage en ligne à partir des flux introduit plusieurs nouveaux défis relativement peu étudiés. Tout d'abord, les flux de données peuvent souffrir de dérives dans la distribution sous-jacente en raison des changements dans l'environnement émetteur, les capteurs utilisés pour récupérer les données pouvant être remplacés ou mis à jour, ou la tâche d'intérêt étant susceptible de changer au fil du temps. Dans de telles conditions, les modèles tendent à s'adapter à la distribution des données actuellement disponibles, ce qui peut entraîner des changements

drastiques dans le mécanisme d'inférence. De plus, les systèmes d'apprentissage en ligne doivent apprendre des données qui arrivent en permanence, ce qui signifie que le mécanisme d'apprentissage devrait pouvoir récupérer toutes les informations nécessaires en temps réel.

Pour résumer les défis mentionnés ci-dessus et formaliser les exigences que l'on devrait imposer au système d'apprentissage en ligne, un tel système devrait être doté des caractéristiques suivantes:

1. Traitement des données en temps réel.
2. Intégration rapide des connaissances à partir de données limitées.
3. Mécanisme de préservation des connaissances déjà acquises.
4. Efficacité sur des données de très grande complexité.
5. Extensibilité à de grands problèmes avec un nombre important de classes.

On peut obtenir la plupart des caractéristiques décrites en imposant certaines conditions à la procédure de l'apprentissage (1), à la conception du modèle (5) ou même au type de modèle utilisé (4). En revanche, (2) et (3) sont difficiles à gérer et interfèrent souvent l'une avec l'autre. La littérature consacrée à l'apprentissage dans les systèmes biologiques et informatiques traite souvent la relation entre la capacité du système à incorporer de nouvelles connaissances (2) et à conserver l'ancien (3) comme dilemme de stabilité-plasticité ([Gro82], [MBB13]). Cette notion, bien qu'elle ne soit pas fréquemment mentionnée dans la thèse, est le problème central de notre étude.

Plus précisément, **dans cette thèse** nous nous concentrons sur le problème de la classification des images en ligne dans un contexte de flux continu. Bien qu'utile dans un grand nombre d'applications pratiques de vision par ordinateur, la classification d'images est un problème difficile reconnu, largement non résolu dans le cas général. De plus, les classificateurs d'images ont besoin de grands ensembles d'apprentissage pour fonctionner raisonnablement bien, tandis que le traitement des bases de données d'images est très gourmand en ressources en termes de stockage et de puissance de calcul. Ces conditions rendent très difficile l'apprentissage en ligne sur les flux de données d'image, d'autant plus que les données entrantes sont lourdes et difficiles à stocker, tandis que la tâche à résoudre est complexe et nécessite la mise à jour de grands modèles d'apprentissage.

Jusqu'à récemment, les méthodes de classification les plus utilisées pour les flux de données comprenaient les arbres Hoeffding ([DH00]), les arbres bayésiens ([SAK+09]), les machines vectorielles de support ([RDIV09]) et les méthodes d'ensemble ([Oza05]). Un aperçu comparatif de ces méthodes est présenté dans ([NWN15]). La conclusion est que, même si elles permettent des tests en temps réel, peuvent gérer efficacement la dérive du concept ([WHC+16]) et ne rencontrent pas de problèmes de mémoire, ces méthodes ne fonctionnent pas bien sur des données complexes de grande dimension, donc pas satisfaisantes pour la condition (4).

En comparaison, les méthodes basées sur l'apprentissage profond (DL) gèrent efficacement les tâches de classification complexes sur des données de grande dimension, avec jusqu'à quelques milliers de classes. Ces dernières années, les approches basées sur DL sont devenues à la pointe de la technologie dans de nombreuses applications, telles que la classification d'images et de signaux ([KSH12]), la détection d'objets ([SKCL13]) et la segmentation ([HGDG17]) , le traitement du langage naturel ([SVL14]), ([CWB+11]) et bien d'autres. Malgré sa popularité et son efficacité sur les données de

grande dimension d'une complexité structurelle importante, la plupart des approches d'apprentissage en profondeur actuellement existantes visent à résoudre les problèmes d'apprentissage hors ligne où toutes les données sont constamment disponibles pendant l'apprentissage. L'optimisation de modèles DL sur des données de flux non stationnaires, avec une distribution changeante des classes de données, conduit généralement à un phénomène d'oubli catastrophique ([MC89]). Les connaissances codées dans les connexions neuronales sont progressivement remplacées par de nouvelles informations, en l'absence de données renforçant les connaissances précédentes, c'est-à-dire des données correspondant à des classes précédemment apprises ou à différents modes des classes actuelles.

Les solutions actuellement existantes pour surmonter l'oubli catastrophique dans les réseaux neuronaux incluent trois types d'approches: des méthodes basées sur la régularisation de l'apprentissage où les mises à jour du réseau dépendent de l'importance des connexions neuronales pour les tâches historiques; des réseaux aux architectures évolutives, capables de croître et de s'adapter à la complexité croissante du problème; et les approches qui utilisent une unité de mémoire supplémentaire fournissant au modèle des classes de données manquantes. Dans le chapitre 4, nous discutons en détail des méthodes existantes pour chacun des trois groupes. Ce chapitre bibliographique montre au lecteur le raisonnement derrière notre choix de direction pour l'étude.

Comme mentionné précédemment, l'un des plus grands défis de l'apprentissage profond en ligne est d'éviter d'oublier les connaissances déjà acquises. Une solution simple à ce problème est la répétition ([Rob95]) qui consiste à stocker toutes les données historiques (ou au moins une partie significative) avec les données de flux nouvellement arrivées et à recycler le classificateur sur toutes les informations disponibles à chaque fois que le modèle doit être mis à jour. Cependant, si le système doit apprendre sur des flux de données à haute vitesse et continuer à acquérir de nouvelles connaissances pendant une période prolongée, ou doit effectuer un apprentissage tout au long de la vie, une telle approche ne parvient pas à s'adapter à la taille du problème. Pour éviter les difficultés liées au stockage des données historiques, nous prenons la ligne de recherche qui se concentre sur l'utilisation de modèles génératifs pour approximer la distribution statistique de la source ([PKP+18, SLKK17, KGL17]).

Dans la **première contribution** de ce travail (chapitre 5), nous abordons le problème de classification incrémentale, où les classes apparaissent dans le flux une par une. Dans l'approche proposée, nous utilisons des réseaux génératifs convolutionnels antagonistes (DCGANs [RMC15]) pour compenser les classes de données absentes. Nous initialisons un DCGAN distinct pour chaque classe nouvellement apparue. Nous les formons ensuite pendant la phase de flux lorsque les données correspondantes sont disponibles pour produire des images visuellement proches de la source d'entrée. Chaque fois qu'une classe de données disparaît du flux d'entrée, nous utilisons les GAN précédemment appris pour générer de nouveaux échantillons pour les catégories manquantes. Cette procédure nous permet de garantir que toutes les classes (passées et présentes) sont équilibrées de façon égale dans le lot d'apprentissage actuel. Les approches proposées pour les cas incrémentaux et continus ont abouti aux articles [BBCF17] et [BBCF18].

Si cette approche présente des avantages et fonctionne raisonnablement bien sur de nombreuses bases de données, l'utilisation des GAN présente toutefois certains inconvénients. Premièrement, la qualité des échantillons générés est très dépendante de la classe car la structure interne des objets influence

le processus d'apprentissage. Deuxièmement, l'entraînement des GAN est un processus instable, en particulier pour les données de grande complexité. Cela peut affecter la qualité des échantillons générés lorsqu'arrivente de nouveaux exemples réels d'une classe. Enfin, les GAN sont extrêmement lents à entraîner. Dans les conditions d'apprentissage incrémental où l'on peut apprendre chaque nouvelle classe jusqu'à la convergence, l'apprentissage lent n'est pas un problème. A l'inverse, un tel comportement fait que l'approche proposée ne parvient pas à évoluer vers des scénarios plus complexes où l'apprentissage en temps réel est requise.

La méthode proposée est basée sur l'idée de pseudo-répétition. Nous utilisons DCGANs comme modules de mémoire supplémentaire qui nous permettent d'approximer la distribution des données de flux et de tirer des échantillons aléatoires de cette distribution lorsque les classes de données correspondantes sont manquantes. Les expériences effectuées sur les ensembles de données MNIST et LSUN ont montré que notre approche effectue un apprentissage incrémental efficace à partir de flux de données non-i.i.d. tout en restant bien préservé de l'oubli catastrophique.

Il est important de noter que l'entraînement d'un modèle génératif par classe de données implique une croissance linéaire de la taille du système avec le nombre de classes déjà apprises. De plus, nos expériences ont démontré que le cadre d'apprentissage proposé est extrêmement lent à converger: sur l'ensemble de données LSUN à 10 classes, le modèle a dû voir plus de cinq millions d'images par classe pour atteindre sa précision de validation optimale. Dans de vraies applications d'apprentissage en ligne, on possède rarement un si grand nombre d'éléments de données par classe. De plus, la complexité des problèmes à résoudre va généralement bien au-delà de dix classes. Par conséquent, il est clair pour nous que la méthode proposée, tout en étant une étape vers un efficace apprentissage profond en ligne, nécessite une amélioration significative pour s'adapter à la taille des applications réelles.

La **deuxième contribution** de cette thèse (chapitre 6) présente une approche plus générale qui peut fonctionner avec des flux de données continus plus complexes sans ordre spécifique d'apparence de classes et est adaptée à des flux beaucoup plus massifs. Le cadre d'apprentissage proposé utilise des auto-encodeurs à des fins de mémoire à long terme pour dépasser les limitations des GAN décrites précédemment. Les auto-encodeurs sont beaucoup plus rapides à entraîner que les GAN. Dans le même temps, l'utilisation d'auto-encodeurs nous permet de former un seul modèle pour apprendre toutes les classes historiques. Cela réduit considérablement les besoins en mémoire du système d'apprentissage.

Pour accroître encore les performances de l'approche proposée, nous améliorons la procédure de l'apprentissage des auto-encodeurs. En effet, nous avons soutenu qu'au lieu de rechercher une approximation précise des données réelles, il faudrait plutôt former des modèles génératifs pour extraire des représentations de données qui ont du sens pour une application d'intérêt (classification dans notre cas). Pour renforcer cette idée, nous avons introduit une fonction objective qui aide les modèles (pseudo-) génératifs à capturer des informations pertinentes pour la classification.

De plus, pour éviter la dérive dans la distribution des échantillons reconstruits par les auto-encodeurs lorsque ces derniers sont entraînés sur leurs propres reconstructions, nous proposons une technique de pondération de gradient adaptative. Cette méthode modifie l'influence des échantillons synthétiques en fonction du nombre de passes de codage-décodage complètes qu'ils ont traversé.

Nous validons la méthode proposée sur les flux continus simulés à partir de jeux de données d'images naturelles statiques et d'un jeu de données synthétique à grande échelle conçu pour cette étude. Nous démontrons que l'approche proposée est rapide, évolutive à de grands problèmes avec des milliers de classes et à un processus de l'apprentissage stable. Le travail décrit a été soumis à la revue Computer Vision and Image Understanding.

L'utilisation des auto-encodeurs dans notre cadre d'apprentissage implique le stockage de représentations de données (codes) très compactes. Ainsi, lorsque cela est nécessaire, nous approximons les données réelles à partir des codes, ce qui est un bon compromis nous permettant d'éviter de construire des modèles génératifs en ligne qui peuvent être mis à jour dynamiquement (une tâche très difficile pour les distributions dimensionnelles élevées).

Étant critique envers l'approche d'apprentissage continu proposée dans ce manuscrit, nous pouvons mentionner plusieurs pistes pour de nouvelles recherches et améliorations.

Pour l'ensemble de données LSUN, toutes les expériences ont été effectuées sur les codes extraits par ResNet pré-entraîné sur ImageNet. Alors que le but de l'extraction de ces codes était d'accélérer l'apprentissage en réduisant la dimensionnalité des données, en théorie, cette procédure pourrait considérablement simplifier la tâche d'apprentissage. En effet, à l'instar de LSUN, ImageNet est un ensemble de données d'images naturelles à grande échelle, avec plusieurs classes similaires. Bien que les ensembles de données diffèrent considérablement, nous pouvons émettre l'hypothèse qu'un tel extracteur de codes fournit déjà une bonne séparation inter-classes en raison de ce chevauchement de classes, ce qui nous permet de résoudre un problème beaucoup plus facile lors de l'apprentissage. Une façon de vérifier cette hypothèse serait d'analyser les propriétés statistiques des classes de données LSUN mappées à l'espace de codes.

Comme cela a déjà été mentionné, pour pouvoir apprendre sûr des flux non ordonnés et évoluer vers de grands problèmes, nous avons dû passer de modèles génératifs (GAN) à des modèles pseudo-génératifs (auto-encodeurs). Ce choix a été principalement motivé par la convergence lente des GAN et l'entraînement instable de ses versions qui permettent un échantillonnage conditionnel. Malgré ce choix, nous soutenons que les modèles génératifs ont un potentiel très important et sont conceptuellement beaucoup plus proches de la mémoire opératoire du cerveau biologique que les AE. Premièrement, les modèles génératifs ne nécessitent pas de stockage explicite des connaissances, même avec une dimension réduite comme dans les AE. Ils visent à apprendre la distribution des données d'entrée et nous permettent d'effectuer un échantillonnage aléatoire à partir d'une approximation dense de cette distribution, plutôt que de reconstruire des échantillons de données déjà vus à partir de l'espace clairsemé des codes. Cette caractéristique devrait, à notre avis, permettre une généralisation nettement meilleure sur l'ensemble de la distribution. Cependant, pour former des modèles génératifs de grande dimension à la volée, il faut concevoir une procédure d'apprentissage avec un transfert de connaissances efficace qui accélérerait l'incorporation de nouvelles informations basées sur ce que le modèle sait déjà.

En plus de l'apprentissage supervisé, les systèmes biologiques peuvent apprendre de manière non supervisée, semi-supervisée et auto-supervisée. La capacité d'apprendre sans accès aux annotations (ou avec accès limité) est essentielle dans l'apprentissage continu où ces annotations sont rarement disponibles pendant la formation. L'un des ajustements incrémentaux possibles de notre travail serait d'utiliser

les dépendances temporelles dans les flux pour rendre la procédure d'apprentissage moins supervisée. En effet, dans ce travail, dans le processus de récupération des données nous avons principalement considéré les flux sans ordre prédéfini. Dans diverses applications réelles, les agents d'apprentissage seraient exposés à des environnements continus, potentiellement avec de nombreuses redondances. De telles dépendances peuvent faciliter l'attribution d'étiquettes pendant la phase d'apprentissage, conduisant ainsi à des exigences de supervision plus faibles pour le système d'apprentissage. Par exemple, lorsqu'un nouvel objet est introduit dans une vidéo continue, l'attribution d'une étiquette à une seule image contenant l'objet devrait suffire pour étiqueter toutes les images suivantes où l'objet est présent.

Enfin, travailler sur tous les points discutés nécessite une procédure bien établie pour l'évaluation de l'apprentissage en continu. Dans nos expériences, nous avons évalué les performances d'apprentissage en ligne de notre méthode en mesurant la précision de classification du modèle au point donné sur l'ensemble de test contenant toutes les classes apprises. Cependant, l'efficacité de l'apprentissage en ligne ne peut être estimée avec seulement une telle metrique. En effet, l'apprentissage à partir des flux de données est un problème complexe qui, selon l'application d'intérêt, peut nécessiter de mesurer les effets d'oubli à court et à long terme, d'évaluer la qualité et le rythme de l'acquisition de nouvelles connaissances, d'estimer la dynamique des besoins en mémoire de travail, etc. Nous soutenons donc que le domaine de l'apprentissage continu nécessite des efforts supplémentaires pour établir les procédures et les paramètres d'évaluation. De plus, pour rendre les études d'apprentissage en ligne cohérentes et comparables, il est essentiel de construire une procédure de streaming de données reproductible associée à une collecte de données suffisamment représentative.

La thèse est organisée comme suit. Dans le chapitre 2, nous décrivons le contexte théorique de l'apprentissage en ligne et donnons des informations détaillées sur les scénarios expérimentaux que nous abordons. Le chapitre 3 fournit une brève discussion sur les principes DL. Nous expliquons comment la procédure d'apprentissage standard dans les réseaux de neurones conduit à l'oubli catastrophique. Dans le chapitre 4, nous passons en revue des méthodes existantes qui traitent le problème de l'apprentissage en ligne sur les flux de données et positionnons nos propositions par rapport à celles-ci. Nous donnons une présentation détaillée des propositions les plus pertinentes dans trois types de méthodes d'apprentissage en ligne: architectures dynamiques, régularisation et approches à double mémoire, et soulignons leurs principaux avantages et limites. Dans le chapitre 5, nous décrivons notre première proposition qui utilise les GAN pour préserver et rejouer les connaissances historiques. Nous démontrons que la méthode proposée peut effectuer efficacement l'apprentissage incrémentiel en le validant sur deux ensembles de données d'images simulées comme des flux de données incrémentiels. Dans le chapitre 6, nous présentons notre deuxième approche, capable d'effectuer un apprentissage continu sur des flux de données dynamiques à l'aide d'auto-encodeurs. Nous discutons des avantages des encodeurs automatiques par rapport aux GAN pour remplacer les données historiques dans un scénario aussi avancé. Nous décrivons notre cadre d'apprentissage, évaluons ses performances par une validation expérimentale sur trois ensembles de données différents et analysons son comportement. Enfin, dans le chapitre 7, nous discutons des perspectives et des futurs travaux.

Bibliography

[ABE+18] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[AR97] Bernard Ans and Stéphane Rousset. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'Académie des Sciences-Series III-Sciences de la Vie*, 320(12):989–997, 1997.

[BBCF17] Andrey Besedin, Pierre Blanchart, Michel Crucianu, and Marin Ferecatu. Evolutive deep models for online learning on data streams with no storage. In *Workshop on Large-scale Learning from Data Streams in Evolving Environments*, 2017.

[BBCF18] Andrey Besedin, Pierre Blanchart, Michel Crucianu, and Marin Ferecatu. Deep online storage-free learning on unordered image streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 103–112. Springer, 2018.

[BDP07] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447, 2007.

[CRDP12] Roberto Calandra, Tapani Raiko, Marc Peter Deisenroth, and Federico Montesino Pouzols. Learning deep belief networks from non-stationary streams. In *International Conference on Artificial Neural Networks*, pages 379–386. Springer, 2012.

[CWB+11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[DCF+15] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[DH00] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[DML+17] Timothy J Draelos, Nadine E Miner, Christopher C Lamb, Jonathan A Cox, Craig M Vineyard, Kristofor D Carlson, William M Severa, Conrad D James, and James B Aimone. Neurogenesis deep learning: Extending deep networks to accommodate new classes. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 526–533. IEEE, 2017.

[FBB+17] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[GK16] Alexander Gepperth and Cem Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934, 2016.

[GMX+13] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[Gro82] Stephen Grossberg. How does a brain build a cognitive code? In *Studies of mind and brain*, pages 1–52. Springer, 1982.

[GŽB+14] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[HCK18] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. *arXiv preprint arXiv:1809.05922*, 2018.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv preprint arXiv:1703.06870*, 2017.

[HKCK18] Tyler L Hayes, Ronald Kemker, Nathan D Cahill, and Christopher Kanan. New metrics and experimental paradigms for continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2031–2034, 2018.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KCK+17] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1857–1865. JMLR. org, 2017.

[KGL17] Nitin Kamra, Umang Gupta, and Yan Liu. Deep generative dual memory network for continual learning. *arXiv preprint arXiv:1710.10368*, 2017.

[KK17] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[KMA+17] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072*, 2017.

[Koh82] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[KPR+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[LH17] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[LP+17] David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[MBB13] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

[MC89] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: the sequential learning problem. *Psychology of learning and motivation*, 24:109–

165, 1989.

[MCH+18] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, B Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[MO14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[NWN15] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and information systems*, 45(3):535–569, 2015.

[OOS16] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

[Oza05] Nikunji C Oza. Online bagging and boosting. 2005.

[PKP+18] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[PL16] Jose L Part and Oliver Lemon. Incremental on-line learning of object classes using a combination of self-organizing incremental neural networks and deep convolutional neural networks. In *Workshop on Bio-inspired Social Robot Learning in Home Scenarios (IROS), Daejeon, Korea*, 2016.

[RDIV09] Piyush Rai, Hal Daumé III, and Suresh Venkatasubramanian. Streamed learning: One-pass svms. In *IJCAI*, volume 9, pages 1211–1216, 2009.

[RKSL17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[Rob95] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[RRD+16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[SAK+09] Thomas Seidl, Ira Assent, Philipp Kranen, Ralph Krieger, and Jennifer Herrmann. Index-

ing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th international conference on extending database technology: advances in database technology*, pages 311–322. ACM, 2009.

[SDBR14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SKCL13] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3626–3633, 2013.

[SLKK17] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[SMDH13] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147, 2013.

[SMK+13] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In *Advances in neural information processing systems*, pages 2310–2318, 2013.

[SSMK18] Joan Serra, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[VVPL17] Ragav Venkatesan, Hemanth Venkateswara, Sethuraman Panchanathan, and Baoxin Li. A strategy for an uncompromising incremental learner. *arXiv preprint arXiv:1705.00744*, 2017.

[WH86] DRGHR Williams and Geoffrey Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[WHC+16] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[YYLH18] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. 2018.

[ZPG17] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*, 2017.

[ZSL12] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. Online incremental feature learning with denoising autoencoders. *Ann Arbor*, 1001:48109, 2012.

[ZXH14] Hongwei Zhang, Xiong Xiao, and Osamu Hasegawa. A load-balancing self-organizing incremental neural networluk. *IEEE Transactions on Neural Networks and Learning Systems*, 25(6):1096–1105, 2014.

[ZXL+17] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

Andrey BESEDIN

Continual Forgetting-Free Deep Learning from High-dimensional Data Streams

Résumé : Dans cette thèse, nous proposons une nouvelle approche de l'apprentissage profond pour la classification des flux de données de grande dimension. Au cours des dernières années, les réseaux de neurones sont devenus la référence dans diverses applications d'apprentissage automatique. Cependant, la plupart des méthodes basées sur les réseaux de neurones sont conçues pour résoudre des problèmes d'apprentissage statique. Effectuer un apprentissage profond en ligne est une tâche difficile. La principale difficulté est que les classificateurs basés sur les réseaux de neurones reposent généralement sur l'hypothèse que la séquence des lots de données utilisée pendant l'entraînement est stationnaire; ou en d'autres termes, que la distribution des classes de données est la même pour tous les lots (hypothèse i.i.d.). Lorsque cette hypothèse ne tient pas les réseaux de neurones ont tendance à oublier les concepts temporairement indisponibles dans le flux. Dans la littérature scientifique, ce phénomène est généralement appelé oubli catastrophique. Les approches que nous proposons ont comme objectif de garantir la nature i.i.d. de chaque lot qui provient du flux et de compenser l'absence de données historiques. Pour ce faire, nous entraînons des modèles génératifs et pseudo-génératifs capable de produire des échantillons synthétiques à partir des classes absentes ou mal représentées dans le flux, et complètent les lots du flux avec ces échantillons. Nous testons nos approches dans un scénario d'apprentissage incrémental et dans un type spécifique de l'apprentissage continu. Nos approches effectuent une classification sur des flux de données dynamiques avec une précision proche des résultats obtenus dans la configuration de classification statique où toutes les données sont disponibles pour la durée de l'apprentissage. En outre, nous démontrons la capacité de nos méthodes à s'adapter à des classes de données invisibles et à de nouvelles instances de catégories de données déjà connues, tout en évitant d'oublier les connaissances précédemment acquises. .

Mots clés : Classification, Apprentissage Profond, Apprentissage Continu, Flux de Données, Oublie Catastrophique

Abstract : In this thesis, we propose a new deep-learning-based approach for online classification on streams of high-dimensional data. In recent years, Neural Networks (NN) have become the primary building block of state-of-the-art methods in various machine learning problems. Most of these methods, however, are designed to solve the static learning problem, when all data are available at once at training time. Performing Online Deep Learning is exceptionally challenging. The main difficulty is that NN-based classifiers usually rely on the assumption that the sequence of data batches used during training is stationary, or in other words, that the distribution of data classes is the same for all batches (i.i.d. assumption). When this assumption does not hold Neural Networks tend to forget the concepts that are temporarily not available in the stream. In the literature, this phenomenon is known as catastrophic forgetting. The approaches we propose in this thesis aim to guarantee the i.i.d. nature of each batch that comes from the stream and compensates for the lack of historical data. To do this, we train generative models and pseudo-generative models capable of producing synthetic samples from classes that are absent or misrepresented in the stream and complete the stream's batches with these samples. We test our approaches in an incremental learning scenario and a specific type of continuous learning. Our approaches perform classification on dynamic data streams with the accuracy close to the results obtained in the static classification configuration where all data are available for the duration of the learning. Besides, we demonstrate the ability of our methods to adapt to invisible data classes and new instances of already known data categories, while avoiding forgetting the previously acquired knowledge.

Keywords : Classification, Deep Learning, Continual Learning, Data Streams, Catastrophic forgetting