Apprentissage continu : S'attaquer à l'oubli foudroyant des réseaux de neurones profonds grâce aux méthodes à rejeu de données

Timothée Lesort

▶ **To cite this version:**

Timothée Lesort. Apprentissage continu : S'attaquer à l'oubli foudroyant des réseaux de neurones profonds grâce aux méthodes à rejeu de données. Machine Learning [cs.LG]. Institut Polytechnique de Paris, 2020. English. NNT : 2020IPPAE003 . tel-02906138

HAL Id: tel-02906138
https://theses.hal.science/tel-02906138

Submitted on 24 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAE003

Thèse de doctorat

Continual Learning: Tackling Catastrophic Forgetting in Deep Neural Networks with Replay Processes

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Ecole nationale supérieure de techniques avancées

École doctorale n°626 École Doctorale de l'Institut Polytechnique de Paris (ED IPP)
Spécialité de doctorat: Informatique, Données, IA

Thèse présentée et soutenue à Palaiseau, le 12/06/2020, par

TIMOTHÉE LESORT

Composition du Jury :

Alexander Gepperth Professor, Fulda University (Computer Science Department)	Président
Irina Rish Associate Professor, Montreal University (MILA)	Rapporteur
Georges Quénot Directeur de Recherche au CNRS, Laboratoire d'Informatique de Grenoble (LIG)	Rapporteur
Razvan Pascanu Research Scientist, Deepmind London	Examinateur
David Filliat Professor, Ensta Paris (U2IS)	Directeur de thèse
Andrei Stoian Research Scientist, Thales (Theresis)	Co-encadrant de thèse

Remerciements

J'ai passé trois années pour ce doctorat pendant lesquelles j'ai pu travailler avec beaucoup de liberté sur mon sujet de thèse. J'ai ainsi pu réaliser de nombreuses collaborations ainsi qu'aller à plusieurs conférences. Tout cela m'a permis de m'épanouir dans ma recherche et dans mon sujet et de pouvoir varier mes activités et mes modes de travail. Je suis ainsi reconnaissant à tous les acteurs de cet environnement de travail qui m'a amené à obtenir mon diplôme de doctorat. Par ailleurs, pendant la période de cette thèse, j'ai aussi continué à exercer des activités diverses à côté grâce à ma famille et à mes amis, merci à tous également.

Pour entrer plus dans le détail, tout d'abord j'aimerais remercier mon directeur de thèse, David, qui a été présent pendant toute la durée de ma thèse pour m'aider, me conseiller et m'écouter. Merci pour ta présence, merci pour ta patience.

Je souhaite remercier aussi Andrei et Jean-François qui ont successivement été mes co-directeurs de thèse côté Thales. Merci, d'avoir accepté de m'encadrer et merci pour le temps que vous m'avez consacré.

Par ailleurs, je suis reconnaissant envers mes collègues de Thales qui ont participé chacun à leur manière à faire régner une bonne ambiance au sein du laboratoire, merci Tiago, Thierry, J-E, David, Stéphanie (x2) :), Michael (x3), Kevin, Cédric, Jean-Yves, Stéphane, Andreina, Yassine, Tom, Rémi, Céline, Louis, Thomas et tous les autres.

Merci également à mes collègues de l'ensta, mes multiples co-bureau, Antonin, Victor, Florence. Les camarades des trois actes déjeuner-café-baby, Thibault, Olivier, Julien, Alex, Thomas et les autres. Les partenaires de sport: Gabriel, Vyshakh (t'es nul!!), Hugo. Merci à tous mes co-auteurs, de l'ensta et d'ailleurs, grâce à qui ma thèse a pu avancer plus vite: Natalia (so much work together !), Hugo, Antonin (again), René, Te, Massimo, Vincenzo, Alexander, Florian, Mathieu, Ashley !

Merci à ma famille, merci à mes parents toujours là pour me soutenir, m'aider et m'accueillir, merci à Zaz, Cacou et Céc, une belle friterie toujours prête à partager un bon repas. Merci aussi à Manf notamment pour ces deux formidables neveu et nièce, Loulou et Milou. Merci à mon grand-père Henri, un exemple de ténacité et de vigueur. Ainsi qu'au reste de la famille, notamment Sabine et Claire qui se greffent occasionnellement aux mardis friterie.

Merci à mes amis d'ici et d'ailleurs pour être ce qu'ils sont et avoir été là afin que je puisse penser à d'autres choses que le doctorat! La TBC : Julaix, Günter, GrandMax, Sylvain, Eddine, Etienne, Jej, Joss et Tang, tous grands amateurs de tarot et de bibines! Les négligés : Mon p'tit fréro LaKajette (surnom favori s'il en est), Dug mon eternel colloc Munichois et traître parmi les traîtres, mon camarade de chant Driss, Cheucheu, Roro, Pauline et les autres... :) Les ex-fumistes : Sophie, Loubier, Lucie, Romain, Lucile, Eline, Antho, Quentin, Alexis, PX, Christophe. Les autres amis de Paris, ex-Parisiens ou d'ailleurs: Minot, Camille, Emmanuelle, Guidg, Hugo, Louis, Nasta, Olivier, Margaux, Claire, Romdav, Paul, Thomas, le brave Remillieux, Sylvestre, Capucine et tant d'autres. Les potes de Montréal : Vincent, Florian, César, Alexandre, Thomas, Massimo, Laurent, Theodora. A bientôt, j'espère ;). Les bons vieux des scouts, Emile, Balt et Martin, ça fait plaisir qu'on se voit encore parfois. Les cibeinsois Sean, Marion, Lucie, Mélanie et bien d'autres. Et enfin Beber et Sylvain, ces personnes respectables et talentueuses, j'espère bien vous voir plus souvent pour travailler efficacement et expérimenter le présent de vérité générale.

Merci également à tous mes relecteurs, ceux de tout temps mais aussi pour ceux qui ont directement participé à ce manuscrit : mon père, ma mère, Zaz, Colin, Cacou, Victor, Vyshakh, Jacko, Günter, Etienne, Capucine, Florence, Andrei et bien sûr, David.

Contents

Contents **3**

List of Figures **5**

List of Tables **7**

1 Introduction **11**
1.1 Context . . . 11
1.2 Contributions and Scope . . . 13
1.3 Publications . . . 13
1.4 Outline . . . 14

2 Deep Learning Backround: Principles and Applications **15**
2.1 Training Deep Neural Networks by Gradient Descent . . . 16
2.2 Learning Paradigms . . . 18
2.3 Learning procedure . . . 30
2.4 Towards Continual Learning . . . 32
2.5 Conclusion . . . 34

3 Continual Learning **37**
3.1 Introduction . . . 37
3.2 Definition of Continual Learning . . . 38
3.3 Key vocabulary . . . 39
3.4 A Framework for Continual Learning . . . 43
3.5 State of the art . . . 49
3.6 Evaluation . . . 55
3.7 Applications : Continual Learning for Robotics . . . 62
3.8 Conclusion . . . 66

4 Supervision Free Inference in Continual Learning **69**
4.1 Introduction . . . 69
4.2 Background . . . 70
4.3 Regularization Approach . . . 71
4.4 Propositions . . . 73
4.5 Experiments . . . 80
4.6 Discussion . . . 84

4.7 Conclusion . . . 86

5 Learning Continually a Generative Models **87**
5.1 Introduction / Motivation . . . 87
5.2 Background . . . 88
5.3 Approach . . . 91
5.4 Experiments . . . 91
5.5 Results . . . 94
5.6 Discussion . . . 103
5.7 Conclusion . . . 103

6 Generative Replay for Classification **105**
6.1 Introduction . . . 105
6.2 Background . . . 106
6.3 Approach . . . 108
6.4 Experiments . . . 112
6.5 Results . . . 113
6.6 Discussion . . . 121
6.7 Conclusion . . . 121

7 Replay for Policy Distillation **123**
7.1 Introduction . . . 123
7.2 Background . . . 124
7.3 Approach . . . 126
7.4 Experiments . . . 128
7.5 Results . . . 130
7.6 Discussion and Future Work . . . 135
7.7 Conclusion . . . 135

8 Discussion **137**
8.1 Rethinking Continual Learning Objectives . . . 137
8.2 Discussion on the thesis choices . . . 141
8.3 Continual Learning Pitfalls . . . 145
8.4 Research recommendations . . . 146

9 Conclusion **149**
9.1 Summary of Contributions . . . 149
9.2 Future Research . . . 150

Bibliography **153**

List of Figures

1.1 Disjoint tasks illustration. . . . 12

2.1 Illustration of a deep neural network (DNN). . . . 15
2.2 Illustration of an artificial neuron. . . . 16
2.3 Illustration of an convolution neuronal network. . . . 19
2.4 Illustration of reinforcement learning environment. . . . 21
2.5 Illustration of famous RL benchmarks. . . . 23
2.6 Illustration of the variational auto-encoder (VAE). . . . 25
2.7 Illustration of the generative adversarial network (GAN). . . . 26
2.8 Illustration of Fitting Capacity method (FiC). . . . 28
2.9 Samples from MNIST-like datasets. . . . 29
2.10 Samples from Cifar10 dataset. . . . 29
2.11 Dog samples from ImageNet dataset. . . . 30

3.1 Task label and concept drift: illustration of the different scenarios. . . . 47
3.2 Venn diagram of some of the most popular CL strategies. . . . 56
3.3 Illustation for robotics tasks 64

4.1 Illustration of a decision boundary learned. . . . 74
4.2 Simple case of continual learning classification in a multi-task setting. . . . 80
4.3 Case of representation overlapping in one task classification setting. . . . 81
4.4 Case of representation overlapping in multi-tasks classification setting. . . . 81
4.5 Illustration of the MNIST-Fellowship dataset's three tasks. . . . 82
4.6 Experiment on disjoint classes without test label vs test label. . . . 83
4.7 Experiments with joint classes. . . . 83

5.1 Illustration of the disjoint setting considered 88
5.2 Fitting capacity evaluation of various generative models on MNIST. . . . 89
5.3 Comparison between generative model evaluations. . . . 90
5.4 Test set classification accuracy on MNIST and Fashion-MNIST 93
5.5 Comparison between FID results and Fitting Capacity results 94
5.6 Fitting Capacity metric evaluation of VAE, CVAE, GAN, CGAN and WGAN 95
5.7 EWC results with CGAN 96
5.8 Reproduction of EWC experiments from [218] 97
5.9 Generated samples with MNIST 98

5.10 Generated Samples with Fashion-MNIST . . . 99
5.11 Fitting Capacity results for GAN and VAE on MNIST . . . 100
5.12 Generated samples with CIFAR10 . . . 101
5.13 Generated samples on CIFAR10 with each method. . . . 102
5.14 FiC and FID on CL tasks with WGAN_GP, on CIFAR10. . . . 102
5.15 Samples at each task for the best working solution, GAN with generative replay. . . . 104

6.1 Illustration of a multi-tasks learning continuum. . . . 108
6.2 MNIST training data for rotation tasks. . . . 108
6.3 MNIST training data for permutation-type tasks. . . . 109
6.4 Samples of MNIST training data for the disjoint tasks . . . 109
6.5 Samples of MNIST training data for the permutations tasks . . . 110
6.6 Illustration of Generative Replay and Conditional Replay for continual learning. . . . 111
6.7 Illustration of classes imbalances for different replay strategies. . . . 112
6.8 Test set accuracies during the training on different tasks . . . 114
6.9 Comparison of the accuracy of each approach on the first task. . . . 116
6.10 Comparison of the FID of each approach's generative models . . . 117
6.11 Generated samples of MNIST for the permutations tasks . . . 118
6.12 Comparison between conditional and marginal replay accuracy. . . . 120

7.1 Image of the three robotics tasks. . . . 124
7.2 Overview of our full pipeline for Continual multi-tasks Reinforcement Learning. . . . 127
7.3 Illustration of *SRL Combination* model . . . 127
7.4 Representation of data sampling strategies to distill the teacher policy. . . . 131
7.5 Accumulated rewards on each task. . . . 132
7.6 Performance comparison with different sampling strategies . . . 132
7.7 Demonstration of the effectiveness of distillation. . . . 133
7.8 Distillation of teacher policies into a student network. . . . 134

List of Tables

3.1 Illustration of continual learning scenarios. 46
3.2 Classification of continual learning main strategies 54
3.3 Continual learning's benchmarks and environments 57
3.4 Illustration of accuracy matrix R 59

4.1 Model architecture for regularization methods evaluation. 82

5.1 Classifier architecture for Fitting Capacity computation 93
5.2 Results table on the 10 disjoint tasks setting 94

6.1 Hyperparameters table for MNIST and Fashion MNIST 111

7.1 Mean normalized performance of a student policies 131

List of Abbreviation

CL	Continual Learning
LLL	LifeLong Learning
DL	Deep Learning
ANN	Artificial Neural Network
NN	Neural Network
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
MLP	Multi Layer Perceptron
i.i.d.	Independent and Identically Distributed
MAP	Maximum A posteriori Probability estimate
MLE	Maximum Likelihood Estimation
PCA	Principal Component Analysis
RGB	Red Green and Blue
RL	Reinforcement Learning
LwF	Learning without Forgetting
GEM	Gradient Episodic Memory
iCaRL	Incremental Classifier and Representation Learning
EWC	Elastic Weight Consolidation

Résumé Français

Les humains apprennent toute leur vie. Ils accumulent des connaissances à partir d'une succession d'expériences d'apprentissage et en mémorisent les aspects essentiels sans les oublier. Les réseaux de neurones ont des difficultés à apprendre dans de telles conditions. Ils ont en général besoin d'ensembles de données rigoureusement préparés pour apprendre à résoudre des problèmes comme de la classification ou de la régression. En particulier, lorsqu'ils apprennent sur des séquences d'ensembles de données, les nouvelles expériences leurs font oublier les anciennes. Ainsi, les réseaux de neurones artificiels sont souvent incapables d'appréhender des scénarios réels tels ceux de robots autonomes apprenant en temps réel à s'adapter à de nouvelles situations et à de nouveaux problèmes [141].

L'apprentissage continu est une branche de l'apprentissage automatique s'attaquant à ce type de scénarios. Les algorithmes continus sont créés pour apprendre des connaissances, les enrichir et les améliorer au cours d'un curriculum d'expériences d'apprentissage.

Il existe quatre grandes familles d'approches pour l'apprentissage continu. Premièrement, les méthodes à **architecture dynamique** consistent à faire évoluer l'architecture du réseau de neurones afin que différentes expériences d'apprentissages soient apprises avec différents neurones ou groupes de neurones. Secondement, les méthodes de **régularisation** évaluent l'importance des neurones ayant déjà été entrainés afin de limiter leurs modifications en conséquence. Il s'agit ainsi d'apprendre de nouvelles tâches préférentiellement avec des neurones jusqu'alors peu ou pas utiles. Troisièmement, les méthodes à **répétitions de données** consistent à sauvegarder des images représentatives des connaissances apprises et à les rejouer plus tard pour se les remémorer. Le quatrième type de méthode, appelé **rejeu par génération** utilise un réseau de neurones auxiliaire apprenant à générer les données d'apprentissage actuelles. Ainsi plus tard le réseau auxiliaire pourra être utilisé pour régénérer des données du passé et les remémorer au modèle principal pour éviter qu'il ne les oublie. Nous présentons en détail l'état de l'art de l'apprentissage continu dans le chapitre 3.

Dans cette thèse, nous proposons d'étudier ces deux dernières méthodes sur des problèmes d'apprentissage sur images. Nous les rassemblont au sein de la famille des méthodes à rejeu de données. Les méthodes de rejeu de données permettent de trouver un compromis entre l'optimisation de l'objectif d'apprentissage actuel et ceux des experiences d'apprentissage passées.

Nous montrons que ces méthodes sont prometteuses pour l'apprentissage continu. Elles permettent la réévaluation des données du passé avec des nouvelles connaissances et de confronter des données issues de différentes expériences. Ces caractéristiques confèrent un avantage certain aux méthodes de rejeu de données par rapport aux méthodes à architecture dynamique ou à régularisation, qui peuvent être incapables d'apprendre à différencier des données provenant de différentes expériences d'apprentissage [139].

Pour mettre en valeur les avantages de ces méthodes, nous expérimentons les algorithmes à redifussion de données sur des séquences de tâches disjointes. Des tâches disjointes sont des tâches d'apprentissage clairement séparées sans intersection dans leur critère d'apprentissage. En classification, par exemple, il s'agit d'apprendre à reconnaitre des images provenant de différents ensembles de classes séparés. Le réseau de neurones apprend les uns après les autres chacun des ensembles de classes. Il doit être capable, in fine, de pouvoir reconnaitre une image provenant de n'importe lequel des ensembles et d'identifer sa classe. Ces expérimentations permettent d'évaluer premièrement, la capacité d'apprendre à distinguer des concepts (e.g classes) appris séparément et deuxièmement, la capacité à se souvenir de concepts appris tout au long de la séquence de tâches.

Afin d'étudier au mieux les méchanismes d'apprentissage et d'oubli continus, les tâches d'apprentissage sont construites à partir d'ensemble de données d'images classiques tels que MNIST [132], Fashion-MNIST [266] ou CIFAR10 [123]. Ceux-ci sont des ensembles de données faciles à résoudre en apprentissage profond classique, mais peuvent toujours être ardu à résoudre dans un contexte continu [186]. Nous expérimentons, par ailleur, dans le chapitre 7 un scénario proche d'une situation réelle en apprenant à un robot à résoudre une séquence de tâche de renforcement.

Nous pouvons ainsi résumer nos contributions de la façon suivante:

- Nous présentons un aperçut approfondit de l'apprentissage continu (Chapitre 3). Nous résumons l'état de l'art sur le sujet ainsi que les différents bancs d'expérimentations et méthodes d'évaluations. De plus, nous approfondissons l'exemple de la robotique pour mettre en valeur les potentielles applications de l'apprentissage continu.

- Nous apportons une preuve théorique des limitations des méthodes dites de régularisation pour l'apprentissage continu. Nous montrons que ces méthodes ne permettent pas d'apprendre à différencier les données provenant de différentes expériences d'apprentissage.

- Nous réalisons une étude empirique sur l'entrainement des modèles génératifs sur des scénarios d'apprentissage continu et nous introduisons une nouvelle méthode d'évaluation des modèles génératifs : la capacité d'adaptation (Fitting Capacity).

- Nous expérimentons différentes méthodes de rejeu de données pour l'apprentissage continu. Nous appliquons en particulier ces méthodes aux scénarios de tâches disjointes pour mettre en avant leurs avantages pour l'apprentissage continu.

Pour résumer, nous démontrons la capacité des méthodes de rejeu de données à apprendre continuellement à travers les paradigmes d'apprentissage non-supervisé (Chapitre 5), supervisé (Chapitres 4 et 6) et de renforcement (Chapitre 7). Ces experimentations nous permettent de présenter et de mettre en valeur les avantages de ces méthodes et de démontrer leur pouvoir à apprendre certains aspects essentiels d'un curriculum d'expérience d'apprentissage faisant défaut aux méthodes concurrentes.

Chapter 1

Introduction

1.1 Context

In recent years, machine learning with deep neural networks has significantly improved the state of the art in solving many research and industrial problems. In vision problems, deep neural networks particularly improve the state of the art in classification and detection. In natural language processing, deep neural are nowadays used for search engines or text analysis. Deep learning also improved reinforcement learning performances. It has made it possible to learn policy and skills in various applications such as video-games, board-games or control.

Robotics is a field which offers significant opportunities for deep learning. Its successes could improve robots cognitive functions such as vision, language processing or exploration. Moreover, deep reinforcement learning can help leverage challenging robotics tasks, such as object manipulation or autonomous driving.

However, the learning algorithms suffer from many shortcomings. An important restriction of deep learning is the dependence on the quality of the data sets, a clean and well-built dataset being a critical condition to an effective learning process. In most machine learning algorithms, training data are assumed to be independent and identically distributed (iid), i.e. the data distribution is assumed static. If the data distribution changes while learning, the new data will interfere with current knowledge and erase it. This phenomenon is so dazzling [76] and it so drastically challenges the algorithms' performance that we call it "catastrophic forgetting". This problem has many implications in the way algorithms train neural networks and in the potential application fields for machine learning.

Let us consider, for example, a robot working in an evolving environment and being assigned the goal of manipulating new objects or solving new tasks. The robot will then need to incrementally learn new knowledge and skills to improve and adapt its behaviour to new situations. With classical machine learning techniques, in order to incorporate new knowledge while avoiding catastrophic forgetting, the model will have to re-learn everything from scratch. A robot which needs only the new data to improve and develop knowledge and skills, would be much more efficient in this situation.

Continual learning (CL) is a branch of machine learning aiming at handling this type of situation and more generally, settings with non-iid data sources. It aims at creating machine learning algorithms able to accumulate a set of knowledge learned sequentially. The general idea behind continual learning is to make algorithms able to learn from a real-life data source. In a natural

environment, the learning opportunities are not simultaneously available and need to be processed sequentially. Learning from the present data and being able to continue later with new data rather than learning only once for all, seems very appropriate. It opens the possibility to improve the algorithms on a certain task or to make them learn new skills/knowledge without forgetting. It also enables transfer between learning experiences. Previous knowledge acquired may help in learning to solve new problems and new knowledge which may improve the solutions found for past tasks. Nevertheless, because of the catastrophic forgetting phenomena, learning continually is a challenging discipline. The strategy consisting in saving everything to avoid any forgetting is not satisfying because it would not be scalable in terms of memory and computation. The amount of memory used might grow too fast. It is therefore important to remember only the essential concepts. Moreover, to deal with catastrophic forgetting, algorithms should identify the potential source of interference between tasks[1] to come up with a smoother forgetting process.

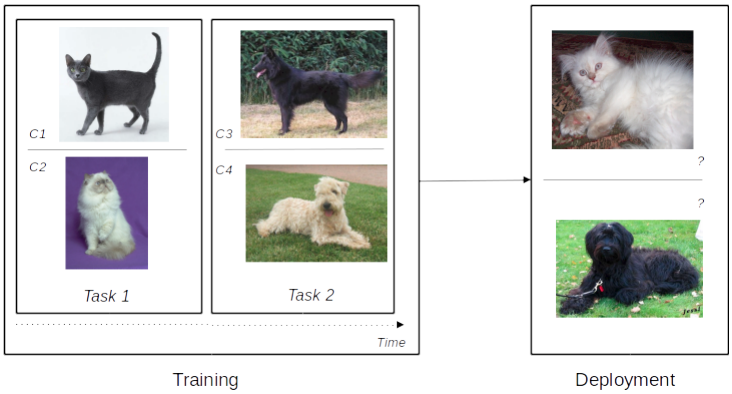


Figure 1.1: Illustration of a disjoint task setting with ImageNet images [124]. There are two tasks learned sequentially, in the first there are two classes, black cats (c1) and white cats (c1), second task is the same but for dogs (c3 vs c4). At the deployment stage, we expect the model to be able to distinguish any class from any other, as white cats from black dogs. Therefore, the model needs both to solve the two tasks and to solve the higher level task which consists of distinguishing classes from various tasks.

In this thesis, we study specifically learning scenarios where the data is static by parts. Each part is different from the other and is referred to as "a task". This setting is called *disjoint tasks setting* and, in classification, it is also called *class-incremental setting*. Each task brings new classes to the learning curriculum and past ones are not available anymore. We show an illustration of class incremental learning in Figure 1.1. In this type of setting, the forgetting happens only when tasks change. Moreover, as classes are available only in one task, it is convenient to evaluate how the neural network learns and memorizes them. On the other hand, this setting makes it possible to assess if algorithms are able to learn to differentiate classes from different tasks, which is challenging in continual learning. Therefore, we study how algorithms are able to deal with this kind of setting as the ability to solve disjoint settings is a necessary condition to be able to deal with real-life settings.

[1] The interferences are phenomena happening when different learning criteria conflict.

1.2 Contributions and Scope

As will be detailed later in the thesis, continual learning approaches can be divided into four main categories: **Regularization**, **Dynamics Architectures**, **Rehearsal** and **Generative Replay**. In our work, we show that *Regularization* and *Dynamics Architectures* methods have theoretical shortcomings for continual learning and therefore we focus on studying applications of replay methods, i.e. *Rehearsal* and *Generative Replay* methods, to categorization and reinforcement learning. More precisely, we study the generative replay and rehearsal methods capacity to learn continually in disjoint settings.

The contributions of this thesis are:

- A global overview of the continual learning research field (Chapter 3). We present the state of the art in continual learning and introduce classical benchmarks and metrics. Moreover, we develop the example of robotics as an application of continual learning solutions.
- A theoretical proof of the shortcomings of regularization methods for continual learning (Chapter 4). We show that regularization methods do not provide learning criterion to differentiate data available at different learning experiences in disjoint tasks settings.
- An empirical study on generative models capabilities in learning continually (Chapter 5) with a new evaluation metric: the Fitting Capacity.
- We experiment with replay methods in continual learning settings (Chapters 5, 6 and 7). We study in particular class-incremental settings with supervision free inference. This benchmark highlights the need for replay in continual learning.

This thesis aims at giving an extended perspective on replay methods in continual learning and insights into the advantages of these approaches for continual learning. We apply replay methods to unsupervised, supervised and reinforcement learning to illustrate our statement.

Moreover, we propose an extensive discussion to stress the real requirements of continual learning, to present the advantages of replay methods in continual learning and spread in light the research direction that should be explored to make it progress.

1.3 Publications

Our work has resulted in the following publications:

1.3.1 Journals

- [141] **Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges** (2019) *T Lesort, V Lomonaco, A Stoian, D Maltoni, D Filliat, N Dìaz-Rodrìguez*, Information Fusion, Elsevier, 2019, ISSN 1566-2535, doi: 10.1016/j.inffus.2019.12.004.

1.3.2 International Conferences

- [135] **Generative Models from the perspective of Continual Learning** (2019) *T Lesort, H Caselles-Dupré, M. Garcia-Ortiz, J-F Goudou, D Filliat*, IJCNN - International Joint Conference on Neural Networks, Budapest, Hungary

- [140] **Training Discriminative Models to Evaluate Generative Ones** (2019) *T Lesort, A Stoian, J-F Goudou, D Filliat*, Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning, Springer International Publishing, pp 604-619
- [137] **Marginal Replay vs Conditional Replay for Continual Learning** (2019) *T Lesort, A Gepperth, A Stoian, D Filliat*, Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning, Springer International Publishing, pp.466-480

1.3.3 Workshops in International Conferences

- [112] **Continual Reinforcement Learning deployed in Real-life using Policy Distillation and Sim2Real Transfer** (2019) *R Traoré*, H Caselles-Dupré*, T Lesort*, T Sun, G Cai, N Dìaz-Rodrìguez, D Filliat*, ICML Workshop on Multi-Task and Lifelong Learning, 2019, Long Beach
- [248] **DisCoRL: Continual Reinforcement Learning via Policy Distillation** (2019) *R Traoré*, H Caselles-Dupré*, T Lesort*, T Sun, G Cai, N Dìaz-Rodrìguez, D Filliat*, Deep RL Workshop, NIPS 2019, Vancouver

1.4 Outline

The organization of the remainder of this manuscript is the following:

- Chapter 2 introduces necessary deep learning background to understand the learning processes applied in the thesis experiments.
- Chapter 3 proposes a global overview of continual learning, its objectives, applications and evaluation.
- Chapter 4 motivates the research in replay methods by pointing out theoretical shortcomings of other methods and by shedding light on replay methods advantages.
- Chapter 5 presents the generative replay method and evaluate its core component ability: the generative model in a continual context.
- Chapter 6 experiments the generative replay method in incremental classification tasks sequences.
- Chapter 7 brings supplementary results on replay methods by applying rehearsal strategies to a continual multi-task reinforcement learning setting. The resulting algorithm is applied on real robots.
- Chapter 8 discusses continual learning objectives and use cases, the choices made in the thesis and the traps of continual learning research.
- Chapter 9 conclude this 3-year work on continual learning and replay methods and opens research directions for its extension.

Chapter 2

Deep Learning Backround: Principles and Applications

Deep learning is a research field that aims at developing learning algorithms. Those algorithms should learn a function that optimizes an objective function on data. In deep learning, this function is implemented as a deep neural network, i.e. a neural network with more than one hidden layer [20, 214] (Figure 2.1[1]).

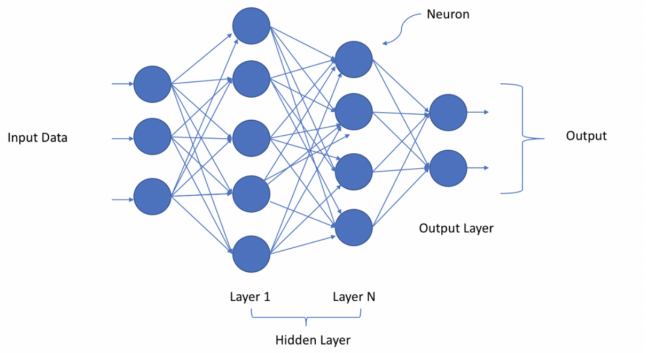


Figure 2.1: Illustration of a deep neural network (DNN).

Deep learning has many applications such as signal processing, language processing or image processing. The scope of this thesis is limited to image processing: we work on algorithms learning from images to understand other images. However, there is no theoretical limitation to transfer results of this thesis in other application fields.

This chapter introduces the basic concepts of classical deep learning in Section 2.1 and its applications in Section 2.2. For a more in-depth understanding of the subject we suggest to refer to the book "Deep Learning" [87]. We also present the global deep learning pipeline in Section 2.3 and introduce in Section 2.4 its constraints leading to continual learning.

[1] Image taken from https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb

2.1 Training Deep Neural Networks by Gradient Descent

We present in this section the simplest method to train deep neural networks: Stochastic Gradient Descent. We also introduce the optimization objective and the libraries dedicated to deep learning.

2.1.1 Deep Neural Networks (DNN)

Deep neural networks (DNN) are artificial neural networks with multiple hidden layers. A layer is composed of a set of neurons connected to neurons from previous layer. They perform a computation and output a single value sent to the next layer. The neurons together form the neural network. A representation of a deep neural structure can be found Fig. 2.1. By combining all the neurons into a coherent ensemble, the neural network should be able to learn complex functions to solve complex problems.

Mathematically, for a set of $n-1$ input values $x_1, x_2, .., x_n$ a neuron will compute the following output:

$$out = \sigma(\sum_{i=1}^{n} x_i \omega_i + b) \tag{2.1}$$

with $\sigma(.)$ a non-linear activation function, b the bias and ω_i the weights of the neuron. An illustration of a single neuron is presented in Fig. 2.2. To train a neural network, we tune the weights (or parameters) and bias of all neurons in order to produce a specific function.

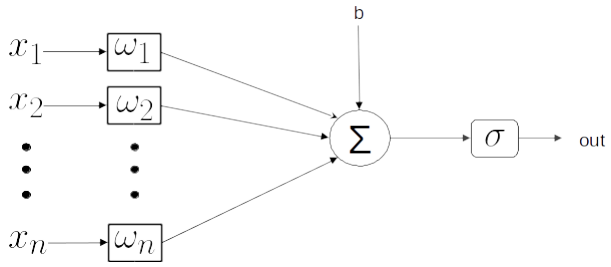


Figure 2.2: Illustration of an artificial neuron.

There are different types of neural networks such as convolutional network or fully connected neural networks. We will present them in Section 2.2.1. In the next section, we will see how to train a DNN.

2.1.2 Stochastic Gradient Descent (SGD)

We define the function $f(.)$ implemented by a neural network. $f(.)$ is parametrized by $\theta \in \mathbb{R}^N$ a vector of N real values corresponding to the connection weights and biases of all neurons. For an input data x we have:

$$\hat{y} = f(x; \theta) \tag{2.2}$$

$\hat{y}$ being the neural network's output.

We assume the dataset composed of pairs (x, y), with x a data point and y its associated expected output. For each data point $x \in \mathbb{D}$, we can compute the output $\hat{y} = f(x; \theta)$ and the loss

$\ell(\hat{y}, y)$. $\ell(\hat{y}, y)$ evaluates the difference between the expected output y and the actual output $\hat{y}$. The loss function is a differentiable function, for example the squared Euclidean distance:

$$\ell_2(y, \hat{y}) = \| \, y - \hat{y} \, \|_2^2 \tag{2.3}$$

The training procedure goal is then to find the best vector θ^* that minimize the cost function $\ell(.)$ on a dataset $\mathbb{D}$.

Deep neural networks are designed such as for each parameter of θ, $\theta_j \in \theta$, we can compute the gradient ∇_{θ_j}:

$$\nabla_{\theta_j} = \nabla_{\theta_j}(x, y) = \frac{\partial \ell(f(x;\theta), y)}{\partial \theta_j} \tag{2.4}$$

One of the assets of deep neural networks is the efficient back-propagation of the gradient through the model. The gradient can use the chain rule to be transmitted from a layer to another.

$$\frac{\partial \ell(f(x;\theta), y)}{\partial \theta_j} = \frac{\partial \ell(f(x;\theta);\theta_j), y)}{\partial f(x;\theta)} \cdot \frac{\partial f(x;\theta)}{\partial \theta_j} \tag{2.5}$$

Hence, $\frac{\partial \ell(f(x;\theta);\theta_j), y)}{\partial f(x;\theta)}$ can be computed once for all and be used after to compute all the ∇_{θ_j}.
The gradient is then used to update the value of all θ_j such that $\ell(\hat{y}, y)$ is minimized.

$$\theta_j \leftarrow \theta_j - \eta \nabla_{\theta_j} \tag{2.6}$$

with η the learning rate.

This operation is then repeated for all (x, y) sampled randomly from the dataset, until convergence to a local minima θ^* of $\ell(f(x;\theta), y)$. This process is called stochastic gradient descent (SGD) [26]. It is the simplest method to train a deep neural network by gradient descent. Data randomly sampled are called i.i.d. (Identically and Independently Distributed). The i.i.d. assumption on the data distribution is often an essential condition to the success of the training algorithms.

The update rule (eq. 2.6) can be modified for a more efficient optimization. Some well known optimization methods are Adagrad [64], Nesterov momentum [239], Adam [118], RMSProp [55]. They add momentum and acceleration components to the gradient in order to learn faster. For the practical applications in this thesis, we mainly used Adam and SGD with momentum to optimize deep neural networks.

2.1.3 Overfitting and Generalization

The optimization process described in Section 2.1.2 minimizes the loss function on the training data until finding a local minima θ^*:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \, \mathbb{E}_{(x,y)\sim\mathbb{D}_{tr}} \ell(f(x;\theta), y) \tag{2.7}$$

with $\mathbb{D}_{tr}$ the training dataset.

However, the true objective of deep learning optimization is to make good predictions on never seen data, i.e. to generalize knowledge from training data to new data. The ability of making good predictions on unknown data is called *Generalization*. It is measured by computing the loss on a test set $\mathbb{D}_{te}$ never seen by the model. If the training loss is very low but the loss on the testing

set is high, the model did not learn a good solution to solve the task. This phenomenon is called *overfitting*. If the loss of the testing set is low then we consider that the model generalized well and the training is successful.

One of the main objectives of machine learning and deep learning is to learn functions that generalize well on new data. However, it is important to note that the test set should be similar to the training set. A neural network can not generalize to completely different data.

2.1.4 Deep Learning Programming Frameworks

The training of neural networks is in most situations achieved thanks to programming libraries that are specific to deep learning. Those libraries allow to compute efficiently and automatically the gradient for all the parameters and to train neural network faster. Using those libraries makes it also possible to develop code faster and have an easy way to use GPU acceleration for deep neural network training. The most famous deep learning library nowadays are Pytorch [182], TensorFlow [1] and Keras [44] but some years ago caffe [106] and Theano [11] were the most used ones. All of those libraries can be used with python, but some of them have an interface to be used with other programming languages such as C++.

In recent years, those libraries have been developed very intensively, making it possible to find pre-trained models and already implemented architectures, neural layers and optimization processes. Today, they are complete frameworks to develop and train deep neural networks.

In this thesis, all the code to train deep neural networks have been developed in python with the Pytorch framework.

2.2 Learning Paradigms

The training of deep neural networks has been applied to different learning paradigm. These paradigms differs in their supervision signal. Supervised algorithms have a true label for all data point, reinforcement learning algorithms have a sparse label referred to as *reward* and unsupervised algorithms have no label at all.

2.2.1 Classification

Images classification (or images recognition) is a typical application of deep learning. It consists of learning to predict the class associated with input data. In this part, we are interested only in supervised training of deep neural networks for classification. Training by supervision is the most common method for learning classification.

History

In the beginning of the 2010s, deep neural networks helped to make significant progress in the image recognition domain, especially with convolutional neural networks (CNN) architectures [77] and hardware computation acceleration with graphical processors units (GPU).

The development of GPU hardware contributed to the acceleration of neural networks training. It substantially helped to develop deep neural networks with more layers growing from a few thousands to hundreds of millions parameters in past few years. Since then, they have been ubiquitous in classification challenges such as PASCAL VOC [67], ImageNet [57], MS COCO [145] or Open Images [126].

Deep neural networks consist of a stack of different neural layers that learn to detect essential features in the data and take decisions. In the early years of classification, feature extractor where hand-engineered, outside the learning algorithm, and only the decision layer was learned. Today, both feature extraction and decision can be learned automatically inside a single neural network. In the next section, we present the important types of neural layers needed for feature extraction and decision making.

Based on these layers, the most famous architectures that helped the development of deep neural networks are LeNet [131], AlexNet [124], Inception [242], VGG [231], ResNet [93]. Those models proposed different types of connections between neurons and layers of neurons to help learning features for image recognition.

Convolution Layers

In image classification, the feature extractor is generally composed of a stack of convolution layers [77].

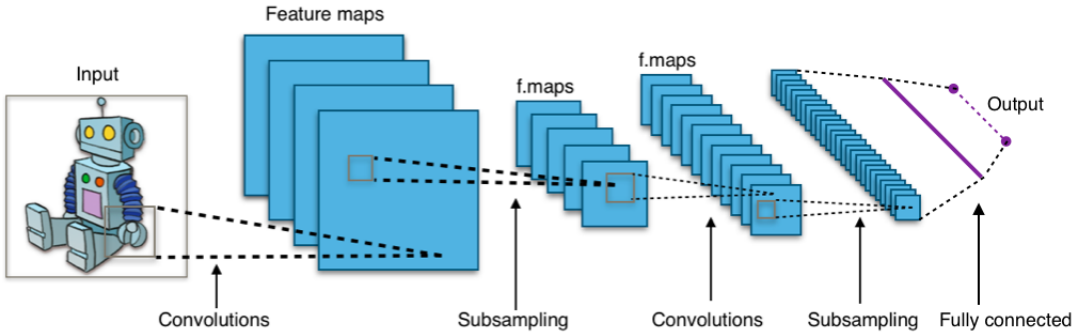


Figure 2.3: Illustration of a multi-layer convolution neural network for image classification. The feature maps contain all the activation output computed with learned filters. The sub-sampling consists of transmitting only a part of the feature map to the next layer.

The convolution layers are designed to limit significantly the number of parameters with respect to a fully connected layer (presented in the next section). They are able to capture local dependencies and benefit from the invariance of certain features to learn better, e.g. a car is still a car whatever its position in the images. The convolution layers are composed of discrete convolution filters (Illustration Figure 2.3). The goal of each filter is to detect a certain pattern. For a given input, the more the input is close to the feature the higher the output of the convolution will be. By stacking convolution layers, the model can detect more and more complex features. Training the neural network consists of learning the right filters to detect discriminative features allowing them to solve the classification tasks.

The output of a convolution layer is a vector $\boldsymbol{h}$ composed of a set of feature map, characteristic of the input features x. It is parameterized by the number of filters, their size and how they are applied to the input vector. $\boldsymbol{h}$ is then transmitted to the next layer.

Fully Connected Layers

The fully connected (FC) layers have the particularity of connecting all the neurons from one layer to another. Those layer can learn to approximate a high variety of functions, however since they contain a lot of connection, they have a lot of parameters. Their training is then more time consuming and energy consuming than convolution layers.

A FC layer of size N realizes the following function:

$$\boldsymbol{o} = W * \boldsymbol{h} + b \tag{2.8}$$

with $\boldsymbol{h}$ the input vector of size H, W a weight matrix of size $H * N$ and $\boldsymbol{b}$ the bias vector of size N.

Output interpretation and loss

The output layer is designed to make the right prediction and to be able to compute a gradient that will be back-propagated through the model.

With respect to the learning procedure introduced in Section 2.1, in the classification case, the expected output y is a label (most of the time an integer) associated with a class of images. The model (i.e. the neural network) should then, for any image x, output a label $\hat{y}$ equal to y.

In order to predict such an integer, the classification models have an customized output layer to learn efficiently a solution. Generally, this layer is a fully connected layer which outputs one value per class, the highest value indicating the class selected by the neural network.

Thus, if there are N classes, the output vector $\boldsymbol{o}$ is a tensor of float with N values. The predicted class is then computed as:

$$\hat{y} = \underset{i \in [\![0, N-1]\!]}{\operatorname{argmax}} \left(o[i]\right) \tag{2.9}$$

For probabilistic interpretation of the output, it is common to apply a softmax operation to the output. Then, each float value $o[i]$ is transformed to $\sigma(o[i])$ with:

$$\sigma(o[i]) = \frac{e^{o[i]}}{\sum_j e^{o[j]}} \tag{2.10}$$

so that all values are mapped between 0 and 1 and they sum to 1. We will note the resulting tensor $\sigma(\boldsymbol{o})$. We can then compute a loss function to compute a gradient and train the neural network by gradient descent. One example commonly used with the softmax is the negative log-likelihood loss:

$$\ell(\hat{y}, y) = -log(\sigma(o)[y]) \tag{2.11}$$

with $\hat{y} = \sigma(o)$.

The gradient descent can then be applied as described in Section 2.1.

In this thesis, these layers will be directly exploited in Chapter 4 and the Chapter 6 dealing with continual learning for classification. Note that the convolutional and fully connected layers presented in this section are used for reinforcement learning and unsupervised learning as well.

2.2.2 Reinforcement Learning

Reinforcement Learning is a machine learning paradigm where the goal is to train an agent to perform actions sequences in a particular environment. The agent should learn a policy which associates the best action to each state of the environment. It is guided by a reward function providing reward according to policy performance. In order to maximize the expected cumulative reward, the agent should explore its environment to discover reward sources and exploit them.

Training methods

Most reinforcement learning processes can be described as Markov decision processes (MDPs). MDPs provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision-maker[2]. At each time-step t, the process is in some state s_t, and the decision-maker may choose any action a_t that is available. The process responds at the next time-step by moving into a new state s_{t+1} and giving the decision-maker a corresponding reward r_t. This process is illustrated in Figure 2.4.

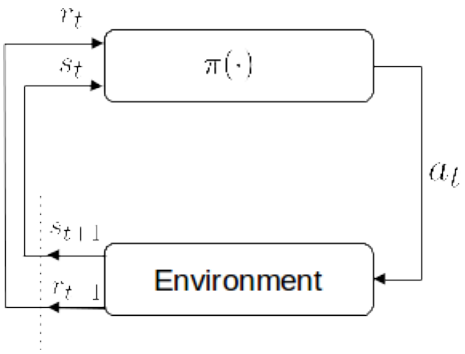


Figure 2.4: Illustration of reinforcement learning environment. With $\pi(\cdot)$ the policy function followed by the agent, r_t the reward at time step t, s_t the state of the environment and a_t the action taken by the agent.

The objective of reinforcement learning is to maximize the accumulated reward gathered in a sequence of actions:

$$R_{tot} = \sum_{i=1}^{T} r_i \tag{2.12}$$

The total reward R_{tot} is the sum of reward received over all the episode (sequence of actions). A discount factor γ can be added to all reward r_i to ponder them.

For each state s_t, the neural network function should choose an action to perform. This function is called the policy $\pi(\cdot)$, such as:

$$p(a|s) = \pi(a, s) \tag{2.13}$$

[2]Definition taken from https://en.wikipedia.org/wiki/Markov_decision_process.

The expected reward received by an agent following a policy starting at a state s_t is computed by the value function $V^{\pi}(s_t)$:

$$V^{\pi}(s_t) = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \tag{2.14}$$

The optimal value function $V^{*}(s_t)$ is the value function for the best possible policy:

$$V^{*}(s_t) = \max_{\pi} V^{\pi}(s_t) \tag{2.15}$$

then the best policy π^{*} is:

$$\pi^{*} = \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s_t) \tag{2.16}$$

Many reinforcement learning algorithms rely on good representation of state quality to maximize the expected cumulated reward. Learning the value function is a way of approximating the state quality and learn a good policy.

However, the value function alone does not give directly the right action to realize, it only evaluates the current state. To find the right action to achieve, the Q-function is introduced which evaluates action quality at each state, such as:

$$V^{*}(s_t) = \underset{a_t}{\operatorname{argmax}}\, Q^{*}(s_t, a_t) \tag{2.17}$$

with $Q^{*}(s_t, a_t)$ the optimal Q-function. The Q-function makes it possible to introduce the *Bellman equation* [17] which links values, rewards and Q-functions:

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}[V(s_{t+1})] \tag{2.18}$$

with $R(s_t, a_t)$ the reward received after realizing action a_t at state s_t.

We can then learn a policy that maximizes the reward based on a neural network implementing Q. A particular method is to learn the optimal Q function which gives directly the best policy. This method is called *Q-Learning* [259]. There also exist other reinforcement learning training methods types.

In the Chapter 7, we will use the PPO (Proximal Policy Optimization) algorithm [216] that belongs to the family of policy gradients methods to directly learn robotics policies. This algorithm is an extension of TRPO (Trust region policy optimization) algorithm [215]. TRPO introduces the trust-region in the policy space. It adds a surrogate loss to constraint the update in the policy space. The goal is to not create drastic changes in the policy and stabilize the learning process. This constraint is implemented as a constraint on the KL divergence between the old and the new policy. KL divergence should not be to high. PPO algorithm does not compute the KL divergence but approximate its action by defining another function that clips the objective function: if the ratio between new policy and old policy is too far from one, the surrogate objective is clipped. PPO is today a commonly used algorithm for various reinforcement learning applications, we used it for many robotics experiments in [189, 190, 112, 248].

For more information about reinforcement learning, we link the reader to the "Reinforcement Learning: An Introduction" book [240].

Reward functions

The reward function defines which behaviour are good or bad for the agent. The reward function can either be sparse and distributes reward only for specific actions (and states) or dense and gives a reward for each action proportionally to the quality of this action. Dense reward functions make the policy easier to learn but it might be expensive to design. Sparse functions can be cheap and easy to design but it might be very hard to learn the best policy from them. The good reward function is just sparse enough to be cheap to design and easy to use for learning algorithms.

For example, a reward function could be one point when a robot put a basketball in the basket. However, the policy that solves those problems might be very difficult to find. It might be really hard to get the first reward and exploit it. The model needs then to explore the environment to find a potential source of reward. The space to explore might be very large, in the example of basketball, it will be very hard for the algorithms to explore all the possibilities of launching the ball in the basket without some more hints beforehand. The reward shaping approach is a way to tune the reward function to help the algorithm find a solution. In particular, it aims make reward more frequent during exploration to give more hints of actions' quality.

Classical benchmarks

Reinforcement learning has known recent big success in games such as chess [230], go [229] or Dota 2 video games [175]. However, commonly used benchmarks are most of the time with simpler robotics settings as in Mujoco [247] or simple video games as Atari [168].

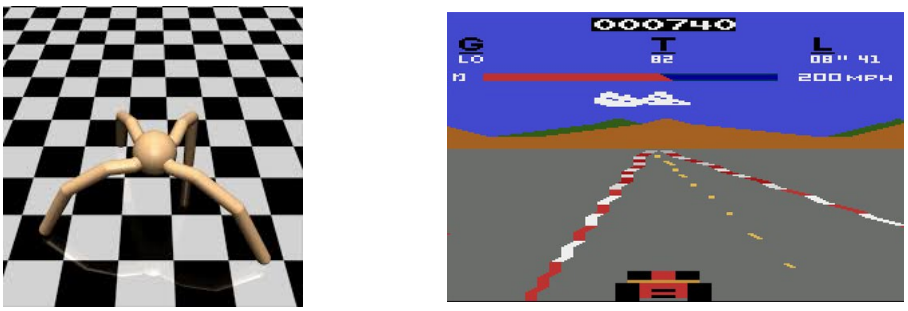


Figure 2.5: Right: Illustration of the Mujoco environment with the ant task. The ant should walk and move as fast as possible. Left: Illustration of the Atari environment with one of the games. The car should go as fast as possible and stay in the circuit.

Reinforcement learning is a very appealing subject because it promises that from a simple reward function we can train an algorithm to execute a difficult task. It has many potential applications as robotics [122], autonomous vehicles [120, 243] etc. However, in practice, the reinforcement learning algorithms are still very unstable and difficult to train. It remains a very interesting research topic, challenging our understanding of how animals and humans learn and questioning the level of supervision needed to learn a given task.

In this thesis, reinforcement learning paradigm will be applied in Chapter 7 in order to learn multiple policies with a robot.

2.2.3 Unsupervised learning

Unsupervised learning is a wide subject in machine learning. In this thesis we are specifically interested in Generative models, as they are used in continual learning for generative replay (see Chapter 3). They are particular types of neural networks designed to reproduce the input data distribution. We call data distribution, in this context, a theoretical probabilistic distribution that generates the dataset $\mathbb{D}$ and could generate any testing data. The goal is to learn to generate data from this distribution, similar but not identical to the data of the training set. For generative models, the generalization is therefore the capacity to generate novel data points. In this thesis, we focus on generative models for images.

In recent years, generative models such as BigGAN [29], VQ-VAE [252] or StyleGAN [114] have shown incredible progress in generating high quality images. In this section, we introduce two generative models frameworks that led to this progress: variational auto-encoders (VAE) [119, 194] and generative adversarial networks (GAN) [88]. We also introduce a tedious challenge of generative models: the evaluation of generated data.

Variational Auto-Encoder (VAE)

Auto-encoders are models that learn to reproduce their input data in their output layer. The variational auto-encoder (VAE) [119, 194] framework is a particular kind of auto-encoder (Illustration Fig 2.6). It is composed generally of an encoder, mapping the input into a latent space and a decoder which learn to regenerate the input image from the latent vector. Those models are useful to learn data compression: if the latent vector is in low dimension, then we can compress input data and decompress it later with the decoder. The VAE learns to map data into a Gaussian latent space, generally chosen as a univariate normal distribution $\mathcal{N}(0, I)$ (where I is the identity matrix). The particularity of the latent space comes from the minimization of the Kullback-Leibler (KL) divergence between the distribution of data in the latent space and a prior distribution $\mathcal{N}(0, I)$. The KL divergence [125] is a measure of the difference between two probability distributions. The decoder then learns the inverse mapping from the univariate normal distribution $\mathcal{N}(0, I)$ to the observation space. However, since the latent space distribution is the univariate normal distribution, we can sample it without encoding data and generate novel data points. This characteristic makes the VAE an interesting option for generating new data after training.

Then, to train the VAE and respecting the prior on the latent distribution, the loss function can be written:

$$\ell = \| x - \hat{x} \|^2 + D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, I)) \tag{2.19}$$

with x the input data, $\hat{x}$ the output of the VAE, μ and σ the two VAE latent vectors, $\mathcal{N}$ is a H dimensional (size of the latent dimension) normal distribution parametrized by a mean and a standard deviation vector.

The Kullback-Leibler between two probabilistic distributions P and Q:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} P(x) log \left(\frac{P(x)}{Q(x)} \right) dx \tag{2.20}$$

We can note that $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$.

In the case of the VAE the Kullback-Leibler can be simplified to:

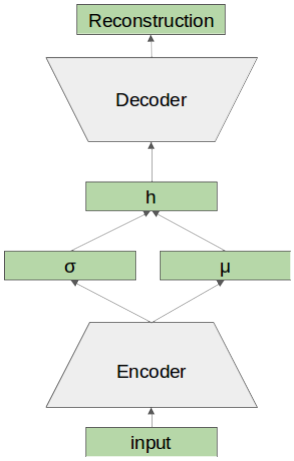


Figure 2.6: Illustration of the variational auto-encoder (VAE). The encoder computes two vectors from the input data, a mean vector μ and a standard deviation vector σ. Then the vector h is sampled from a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ and the decoder output the reconstruction of the input image from h.

$$\forall i \in [\![0, H-1]\!],\ D_{KL}(\mathcal{N}(\mu_i, \sigma_i), \mathcal{N}(0,1)) = log\left(\frac{1}{\sigma_i}\right) + \sigma_i^2 + \mu_i^2 - \frac{1}{2} \tag{2.21}$$

with μ_i and σ_i the i-th dimension of respectively μ and σ.

Generative adversarial networks (GAN)

Generative adversarial network [88] is another framework of generative models (Illustration Fig 2.7). The learning process is a game between two networks: a generator G learns to produce images from the data distribution P and a discriminator D learns to discriminate between generated and true images. The generator learns to fool the discriminator and the discriminator learns to not be fooled. Therefore both D and G have the same loss function $\ell_{GAN}(x, z)$, but G try to minimize it while D try to maximize it:

$$\ell_{GAN}(x, z) = \mathbb{E}_x[log(D(x))] + \mathbb{E}_z[log(1 - D(G(z)))] \tag{2.22}$$

In this function, $D(x)$ is the discriminator's estimate of the probability that real data instance x is real, $\mathbb{E}_x$ is the expected value over all real data instances, $G(z)$ is the generator's output when given noise z, $D(G(z))$ is the discriminator's estimate of the probability that a fake instance is real, $\mathbb{E}_z$ is the expected value over all random inputs to the generator[3].

This class of generative models can produce visually realistic samples from diverse datasets but they suffer from instabilities in their training.

First introduced in [88], many follow-up work have extended and improved upon the original model. Among these, GANs have been extended to Conditional-GANs (or CGANs) to support class-conditional generation [166], and numerous of papers [9, 23, 173, 90] have focused on modifying

[3]Equations and legends are taken from https://developers.google.com/machine-learning/gan/loss

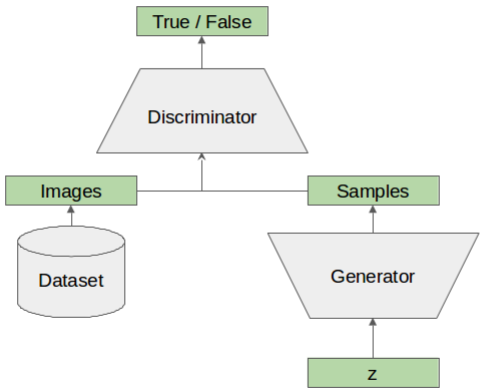


Figure 2.7: Illustration of the generative adversarial network (GAN). An input vector z sampled from a random distribution is given to the generator. The generator output a generated sample. The discriminator receives a mix of true samples and generated samples and predicts if they are true samples or generated one.

the objective function 2.22 to stabilize training and improve the generation quality. One of the model we evaluate, the Wasserstein GAN (WGAN) [9], try to address training issues by enforcing a Lipschitz constraint on the discriminator, i.e. they clip the discriminator's gradient to make training more stable.

There exist many variations of those two frameworks aiming at improving data generation and stability of model with improved losses and architectures.

Evaluation methods

A remaining problem of generative models is the evaluation of the generated samples because generative models produce images, and it is tedious to have a formal definition of a good image. Furthermore, the expectations on the generated images might be different from one application to another. For example, one might expect images that maximize their reality likelihood while others might expect to maximize their variability.

We present here a partial list of evaluation methods for image generation:

- **Visual Turing Test** The visual Turing test [81] is performed by asking humans if images look real or not to assess the generative model quality.
- **Multi-scale structural similarity** Multi-scale structural similarity [258] (MS-SIM) is a measurement that gives a way to incorporate image details at different resolutions in order to compare two images. This similarity is generally used in the context of image compression

to compare images before and after compression. However, it can be used to estimate the variability of features in generated images [174].

- **Inception score** One of the most used approaches to evaluate a generative model is Inception Score (IS) [209, 174]. The authors use an inception classifier model pre-trained on ImageNet dataset to evaluate the sample distribution. They compute the conditional classes distribution $P(Y|X = x)$ at each generated sample x and the general classes distribution $P(Y)$ over the generated dataset.

  They proposed the following score:

$$IS(X) = \exp(\mathbb{E}_X[D_{KL}(P(Y|X) \parallel P(Y))] \tag{2.23}$$

  where D_{KL} is the Kullback-Leibler divergence. The KL term can be rewritten :

$$D_{KL}(P(Y|X) \parallel P(Y)) = H(P(Y|X), P(Y)) - H(P(Y|X)) \tag{2.24}$$

  where $H(P(Y|X))$ is the entropy of $P(Y|X)$ and $H(P(Y|X), P(Y))$ the cross-entropy between $P(Y|X)$ and $P(Y)$.

  The inception score measures if the inception model predictions gives high confidence in varied classes for the generated data. This relies on the hypothesis that if prediction confidence is high, the input image is good.

- **Frechet Inception Distance**

  Another approach to evaluate generative adversarial networks is the Frechet Inception Distance (FID) [97]. The FID, as the inception score, is based on features low moment analysis. It compares the mean and the covariance of activations between real data (μ and C) and generated data (μ_{gen}, C_{gen}). The activation is taken from an inner layer in a pre-trained inception model. The comparison is done using the Frechet distance (see Eq. 2.25). The inception model is trained on Imagenet.

$$d^2((\mu, C), (\mu_{gen}, C_{gen})) = \parallel \mu - \mu_{gen} \parallel_2^2 + Tr(C + C_{gen} - 2(C * C_{gen})^{\frac{1}{2}}) \tag{2.25}$$

  FID measures the similarities between the distribution of the generated feature and the distribution of real features. It assumes a Gaussian distribution of features over the dataset.

- **Fitting Capacity**

  The *fitting capacity* (FiC) approach is to use labeled generated samples from a generator G (GAN or VAE) to train a classifier and evaluate this classifier afterward on real data [140]. It is illustrated in figure 2.8. This estimation of the generative model quality is one of the contributions of this thesis [140]. It is presented more in depth in Chapter 5.

  The fitting capacity of G is the test accuracy of a classifier trained with G's samples. It measures the generator's ability to train a classifier that generalizes well on a testing set, i.e the generator's ability to fit the distribution of the testing set. This method aims at evaluating generative models on complex characteristics of data and not only on their features distribution.

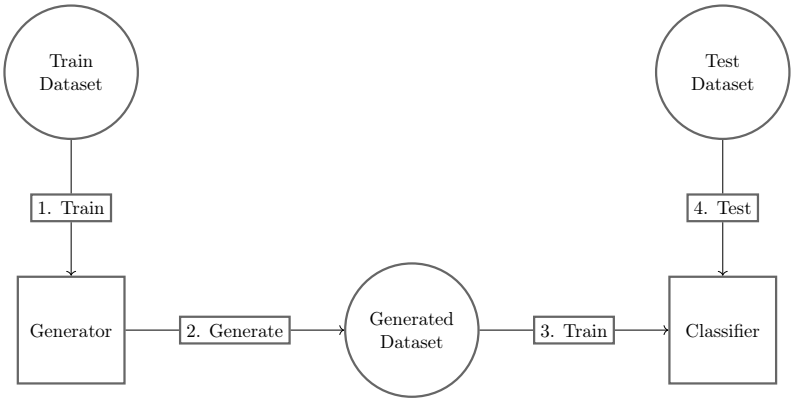


Figure 2.8: Illustration of Fitting Capacity method (FiC): 1. Train a generator on real training data, 2. Generate labeled data, 3. Train classifier with the generated data, 4. Evaluate the generator by testing the classifier on the test set composed of real data

There exist a lot of different evaluation for generative models as listed and discussed in [25]. Anyhow, the best evaluation for a generative model is generally dependent on the future use of the generated data.

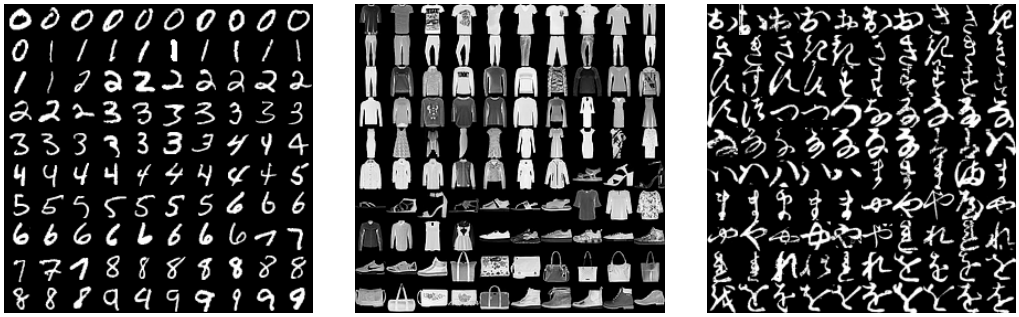
In this thesis, the Chapter 5 deals with continual learning for data generation and the 6 take advantage of generative models for continual classification.

2.2.4 Classical Benchmarks

We now present some classical machine learning benchmarks (MNIST, Cifar10, ImageNet) and some other benchmarks we will use in the experimental work of this thesis (Fashion MNIST and KMNIST).

MNIST and MNIST-like datasets

- **MNIST** [132] is a common benchmark for computer vision systems and classification problems (Fig. 2.9). It consists of gray scale 28x28 images of handwritten digits (ten balanced classes representing the digits 0-9). The train, test and validation sets contain 55.000, 10.000 and 5.000 samples, respectively.
- **Fashion MNIST** [266] consists of grayscale 28x28 images of clothes (Fig. 2.9). We choose this dataset because it claims to be a "more challenging classification task than the simple MNIST digits data [266]" while having the same data dimensions, number of classes, balancing properties and number of samples in train, test and validation sets.
- **KMNIST** [266] consists of grayscale 28x28 images of Kuzushiji (japanese cursive) (Fig. 2.9). As for Fashion-MNIST, we use this dataset because it is a drop in replacement of MNIST dataset and it is a more challenging classification task than the simple MNIST digits data and adds some diversity in the training data.



(a) MNIST (b) Fashion-MNIST (c) KMNIST

Figure 2.9: Samples from MNIST-like datasets.

Cifar10 / Cifar100

Cifar10 [123] dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The classes are completely mutually exclusive. This dataset have been used a lot to design and prototype classification and data generation machine learning algorithms. CIFAR100 is the same dataset with 90 more classes.

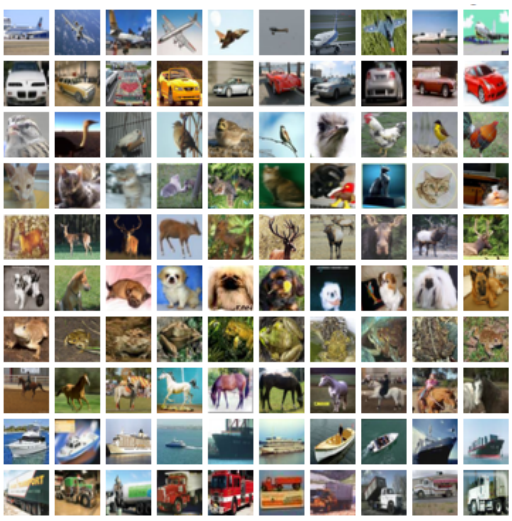


Figure 2.10: Samples from Cifar10 dataset.

ImageNet

ImageNet [124] is a dataset associated with the ILSVRC challenge (ImageNet Large Scale Visual Recognition Challenge). The dataset used in this context is composed of one thousand non-overlapping classes. The images are in color and are from different shapes and resolution but they

are often normalized at 224*224 pixels. This dataset is one that leads to the revolution of machine learning for classification. It is also used nowadays for data generation purposes.

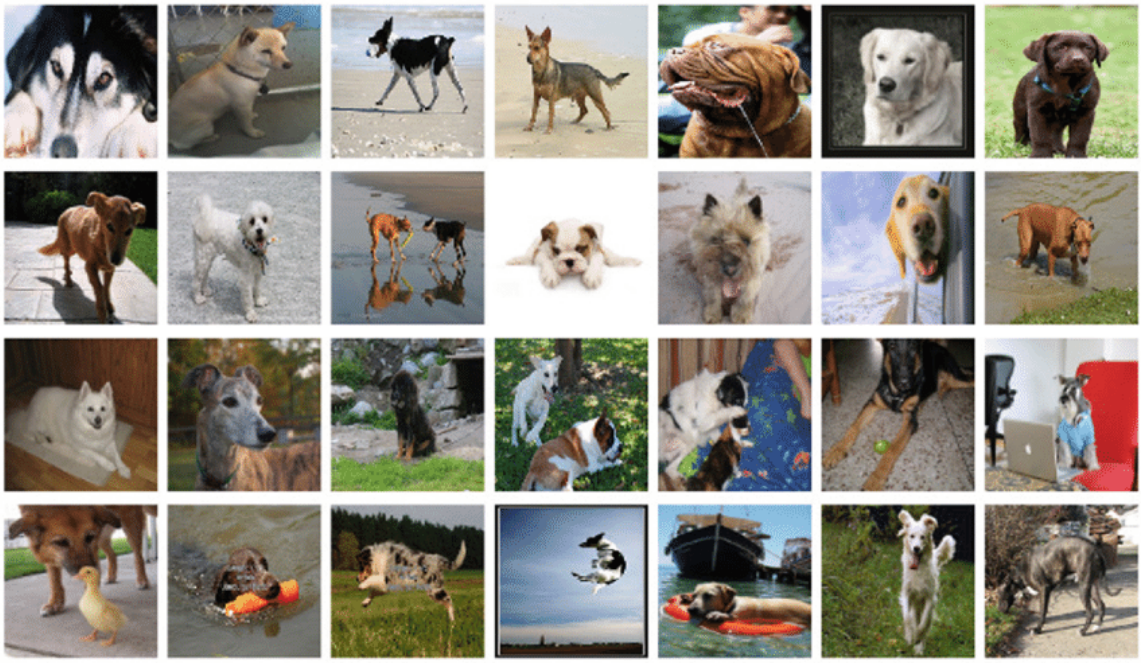


Figure 2.11: Exemples of Imagenet samples with dog pictures.

The benchmark listed above are linked to supervised classification, but they have been adapted for benchmarking other type of machine learning such as data generation, few shot-learning or disentanglement. In this thesis we adapted these classical benchmarks into continual learning benchmarks. Even if some of them can looks almost trivial for nowadays machine learning algorithms they can still be challenging in continual learning settings [186]. In Chapter 3, we will present the different continual learning benchmarks.

2.3 Learning procedure

Besides the training optimization process, the success of deep learning algorithms relies significantly on data handling. In this section, we present the classical full pipeline necessary to train a neural network.

2.3.1 Data Gathering

First of all, to train a DNN, data needs to be gathered to create a dataset. In classification, the image of the different categories needs to be selected and their label set and verified. Building a good quality dataset is difficult, categories need to be balanced, images should be varied and a

trivial solution should not bias the problem if we want to train correctly a neural network. For example, one should avoid that to classify cows from birds, it might be possible to only check if the image is mostly blue (because of sky) or green (because of grass) to solve the classification, instead of looking at intrinsic characteristics from cows and birds. The trivial solutions come from bias into the training set making the learning model believe false causal relationship. A well-designed dataset aims at avoiding misleading correlations in the data.

2.3.2 Data Splitting

Before training the neural network, the dataset is split into three different sets. The training set, used to learn parameters, the validation set, used to select the hyper-parameters and the test set that will be used to evaluate the final performance and verify if the model trained can generalized on new data. It is important to split those set wisely because they should be similar but different enough to be able to measure generalization.

2.3.3 Model Architecture

Once the dataset is built, we choose a model architecture. The model architecture consists of the stack of layers, the characteristics of each layer and how they connected to each other. The architecture defines implicitly the family of functions we are searching our solution θ^* in. Architectures are often parametrized by hyper-parameters that can change the model architecture to better fit a learning problem, e.g. the number of layers, the number of convolutional filters by layer, the size of the filters or the padding...

2.3.4 Model Initialization

Once the architecture is chosen, the model and optimization process should be initialized, i.e. all parameters and hyper-parameters should be set. The architecture hyper-parameter (HP) described in the previous Subsection 2.3.3 are generally chosen empirically based on existing algorithms. The optimizer hyper-parameters such as the learning rate or the ones specific to certain optimizers are also chosen empirically. In the frameworks described in Section 2.1.4, the optimizer functions are proposed with default values that can be used directly.

Once the hyper-parameters are fixed, we initialize the parameters. In the literature, there are several heuristics to initialize parameters, generally following a random policy. The most famous one are *Xavier initialization* [84], or *Kaiming initialization* [94], or a simple normal initialization $\mathbb{N}(0, 1)$.

For example, in the *Xavier* initialization, layers are initialized such as any weight (parameter) w is sampled according to:

$$w \sim \mathcal{U}(-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, +\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}) \tag{2.26}$$

with $\mathcal{U}$ the uniform probabilistic distribution, n_i the incoming network connections and n_{i+1} the out coming network connections from that layer. The bias are initialized to zero.

2.3.5 Data Preprocessing

In order to maximize the learning process chance of converging to a satisfying solution, one could choose to apply specific transformations to the data. First, the data can be modified to be easier to process by the model. For example by applying an heuristic transformation to improve the saliency of crucial components of data. For example, change the color spectre to remove useless colors. In deep learning, a known heuristic for data preprocessing is to normalize value such as the mean is zero and the standard deviation 1. Another common one is the normalization of input data such that all data have the same mean and standard deviation channel by channel. For example, the default values in pytorch to normalize input for model trained on ImageNet are, respectively for channel RGB $mean = [0.485, 0.456, 0.406]$, $std = [0.229, 0.224, 0.225]$.

To improve the model's generalization performance (see Section 2.1), we can perform *data augmentation*, i.e. add a random modification to training images. Those modifications should change the image without changing what it represents, i.e. without changing essential features. Thus, we expect from the model to learn only important features and ignore the rest. As discussed in Section 2.3.1, the construction of a dataset can be difficult and produce some bias. The data augmentation can help to correct them artificially. For example, to detect a car in an image, we expect the model to not give importance to the color (cars can be in any color). If the car's color in the dataset is only white, it will create a bias in the model making it believe that cars are always white. To correct this bias we can then artificially change the color of cars.

2.3.6 Parameters Optimization

The model can now be trained by gradient descent and learn the best set of parameters θ^* as described in Section 2.1. θ^* should optimize the loss on training data but its real objective is to optimize the loss on unknown data. As described in Section 2.1.3, the real application goal is to train a model able to generalize.

2.3.7 Hyper-Parameters Optimization

Frequently, the hyper-parameters set at the beginning do not allows to learn the most satisfying solution. The validation set help then to select the model that has learned and generalizes best. Indeed, the validation data has not been used for training, the error computed on this set is not biased by the optimization process and can be used to select the best hyper-parameters. The hyper-parameters selection is a tough problem, contrary to parameters it is not possible to compute the gradient of the hyper-parameters, their selection should then follow empirical heuristics. Among the possible approaches, grid search consists in searching all the combination of hyper-parameters for specific regularly separated values, but it is also advised to rather search them randomly [22].

There is a research field aiming at finding automatically hyper-parameters, called auto-machine learning [72, 65]. The final goal is still to maximize the generalization capacity on never seen data as presented in Section 2.1.3, but these approaches are computationally extremely intensive and are rarely used in practice.

2.4 Towards Continual Learning

The deep learning pipeline, as presented in the previous section, has shown impressive results in numerous applications such as classification, detection, generation, language processing... However,

it imposes to have all the data at the beginning of the learning process. In particular, it assumes i.i.d. data during training. This assumption is unfortunately difficult to maintain in many situations, for example when data are gathered online. In this section we introduce context in which the pipeline presented in Section 2.3 can not be applied rigorously. Then, we will present briefly the implications in the optimization process.

2.4.1 Context

The constraint of collecting all the data before training a neural network is inconvenient in many cases. De facto, a dataset is very likely to be completed with new data, either to increase the variability of existing concepts in data or to add new concepts. In the classification perspective, either to add data to existing classes or to add new classes to the existing ones.

A trivial solution is to train a neural network from scratch with all data every time new data are available. Nevertheless, training large neural networks can take weeks to months to be trained, then retraining every time can be costly and time-consuming. Especially if new data are regularly available as in a stream of data. Being able to improve a trained network with new data only would be then quite advantageous. Another situation, where training from scratch is not possible, is when the data is not available anymore. For example, if a client buys a pre-trained model but has no access to the initial training data. If this client wants to improve the model with its own data, he can not retrain it from scratch since the initial data are not available. It can also happen if the data have not been saved for legal reasons or memory limitations. From a more general point of view, it would be more convenient to learn from the data available and improve later with new resources. In order to handle such settings, representations should be learned in an online manner [143]. As data gets discarded and has a limited lifetime, the ability to forget what is not important and retain what matters for the future are the main issues that algorithms should targets and focuses on.

The field of deep learning aiming at solving this problem of data availability is called *Continual Learning*. In particular, it aims at finding working solutions for agents which learn from an evolving environment and that need to learn continually to adapt to unseen situations and remember already learned solutions to known situations. For examples robots.

Indeed, from a robotics point of view, CL is the machine learning answer to developmental robotics [151]. Developmental robotics is the interdisciplinary approach to the autonomous design of behavioral and cognitive capabilities in artificial agents that directly draws inspiration from developmental principles and mechanisms observed in children's natural cognitive systems [35, 151] In this context, CL must consist of a process that learns cumulative skills and that can progressively improve the complexity and the diversity of tasks handled. Autonomous agents in such settings learn in an open-ended [61] manner, but also in a continual way. Besides CL, crucial components of such developmental approach consist of learning the ability to autonomously generate goals and explore the environment, exploiting intrinsic motivation [176] and computational models of curiosity [177].

2.4.2 Continual Learning procedure

We have seen that the classical pipeline for deep learning have limitations in continual learning, in particular, because all the data might not be available at the same time. The pipeline from Section 2.3 should therefore be adapted so that data gathering and learning can be simultaneous or interleaved.

Optimization

Concerning the optimization process to train neural networks, the fact that all data are not available at the same time removes the i.i.d. assumption. Many deep learning algorithms are then not adapted anymore for such situations. For most of them, in a continual learning situation, the neural network will automatically adapt to the last data only and forget everything learned on the previous one. This phenomenon is named "catastrophic forgetting" [76].

Data gathering

In fact, gathering data continually is not the essential problem, as theoretically, it is still possible that gathered data are drawn i.i.d. The real problem is the evolution of the underlying data distribution. If this data distribution changes, then "catastrophic forgetting" might happen. This phenomenon is called *concept drift* [82]. In order to find suitable algorithms to avoid catastrophic forgetting, the concept drift needs to be detected and estimated. This process is tedious because the concept drift can be abrupt or very progressive, making it difficult to grasp.

Evaluation

Furthermore, the evaluation process when the data distribution is not static is complicated. As seen in Section 2.3.1, building a dataset to learn should be achieved carefully, there should be a proper balance between classes, a good variability in the data to learn. In the case of continual learning, the evaluation set should be constructed while learning, which might be difficult. Anyhow, it remains essential to have an evaluation set to avoid overfitting.

Hyper-Parameters Optimization

Another tedious process in continual learning is the hyper-parameters optimization. In classical deep learning, the hyper-parameters are tuned to make the algorithms learn better its full task. However, for a convenient set of hyper-parameter at a time t it is possible that at time $t + T$ this set is not good anymore, the hyper-parameter should then be modifiable during the training. Moreover, a HP set can be good to learn but not to remember, so at time $t + T$ if the algorithms forget everything it can not come back at time t to re-learn. The HP selection should both be optimized to present data and withstand to potential future concept drifts.

We will not propose a new pipeline for continual learning because many continual learning cases are different and should not be handled in the same way. Nevertheless, the spirit of the pipeline from Section 2.3 should be maintained. The idea of gathering data, preprocessing them, designing a proper model and train it should be kept but more dynamically than in classical deep learning settings. The less impacted step of the original pipeline is *Model Initialization* since it should be achieved only once at the training's beginning.

2.5 Conclusion

In this chapter, we presented the simplest method to train deep neural networks: Stochastic Gradient Descent (SGD). The training is achieved by optimizing the model parameter on a dataset. The training goal is to make a neural network able to make decisions on never seen data, i.e. able to

generalized its training data. We described the complete pipeline of deep neural network classical training, from data gathering to hyper-parameters selection and we point out the limitation of this pipeline for real-life applications. Finally, we introduced *continual learning*, a research field that aims at overcoming those limitations.

In the next chapter, we present a more in-depth overview of continual learning state of the art, key vocabulary and objectives. We also present a framework to frame any continual learning approach, with a set of benchmarks and evaluation metrics.

Chapter 3

Continual Learning

In the previous chapter, we introduced classical deep learning basic concepts and pipeline and showed their lack of adaptability in practical situations. We also introduced how continual learning aims at solving those shortcomings. In this chapter, we present the continual learning research field more extensively. We illustrate the need for continual learning through the lens of robotics. In particular, we stress the need for better practice in research for better transfer from a field to another, as from simulation to robotics.

This work was the fruit of a collaboration with Vincenzo Lomonaco and Natalia Díaz-Rodríguez. It was published in the "Information Fusion" Journal [141]. The original article has been slightly modified to better fit the thesis thread and updated to add some recent papers.

3.1 Introduction

As described in Section 2.4, machine learning (ML) approaches generally learn from a stream of data randomly sampled from a stationary data distribution. This is often a *sine qua non* condition to learn efficiently, which makes many application scenarios difficult to solve.

For convenience, we can empirically split the data stream into several temporally bounded parts called *tasks*. We can then observe what we learn or forget when learning a new task. Even if there is no mandatory constraint on a task, a task often refers to a particular period of time within which the data distribution may (but not necessarily) be stationary, and the objective function constant. Tasks can be disjoint or related to each other, in terms of learning objectives, depending on the setting.

We now propose a framework for continual learning. This framework also sets the opportunities for continual learning to have a description frame to present approaches in a clear and systematic way. We can summarize the chapter's contributions as following:

- We propose an in depth state of the art of continual learning approaches.
- We present a framework to help characterizing continual learning approaches and benchmarks.
- We gather a list of benchmarks and evaluation metrics for continual learning.
- We develop the example of robotics as an application field of continual learning research.

In the sequel, we first present the context and the history of continual learning. Second, we aim at disentangling vocabulary around continual learning to have a clear basis. Third, we introduce our framework as a standard way of presenting CL approaches to help transfer between different fields of continual learning, especially to robotics. Fourthly, we present a set of metrics that will help to better understand the quality and shortcomings of every family of approaches. Finally, we present the specifics and opportunities of continual learning in robotics that make CL so crucial.

3.2 Definition of Continual Learning

Given a potentially unlimited stream of data, a Continual Learning algorithm should learn from a sequence of partial experiences where all data is not available at once. A non-continual learning setting would then be when the algorithm can have access to all data at once and can process it as desired. Continual learning algorithms may have to deal with imbalanced or scarce data problems [235], catastrophic forgetting [76], or data distribution shifts [82].

We consider continual learning a synonym of *Incremental Learning* [82, 193], *Lifelong Learning* [43, 245] and *Never Ending Learning* [36, 167]. For the sake of simplicity, for the remainder of the chapter we refer to all Continuous, Incremental and Lifelong learning synonyms as Continual Learning (CL). However, in the discussion of the thesis (Chapter 8) we show that we can use them to distinguish different continual learning scenarios.

In this section we first present the history and motivation of continual learning, then we present several definitions of terms related to CL and, finally, we present challenges addressed by CL in machine learning.

3.2.1 History and Motivation

The concept of learning continually from experience has always been present in artificial intelligence and robotics since their birth [250, 261]. However, it is only at the end of the 20^{th} century that it has begun to be explored more systematically. Within the machine learning community, the lifelong learning paradigm has been popularized around 1995 by [245] and [199].

Between the end of the 90s and the first decade of the 21^{st} century, sporadic attention has been devoted to the topic within the supervised, unsupervised and reinforcement learning domains. However, despite the first pioneering attempts and early speculations, research in this area has never been carried out extensively until the recent years [179, 43]. We argue that this is because there were more complex and fundamental problems to solve and a number of additional constraints:

- *Lack of systemic approaches*: Machine learning research for the past 20 years has focused on statistical and algorithmic approaches on simple tasks (e.g., tasks where the distribution of data is assumed static). CL typically needs a systems approach that combines multiple components and learning algorithms in complex and dynamic tasks. The complexity of tasks and their multiple uses in continual learning greatly complicates training and evaluation procedures. Disentangling *"static"* learning performance from continual learning side effects is important for the very incremental nature of the research and to facilitate comparison between approaches in this area.

- *Limited amount of data and computational power*: Digital data is a luxury of the 21^{st} century. Before the big data revolution, collecting and processing data was a daunting task. Moreover, the limited amount of computational power available at the time did not allow complex and

expensive algorithmic solutions to run effectively, especially in a continual learning setting which undoubtedly makes learning more complex by having to deal with multiple tasks at the same time, as well as having to incorporate the concept of time into the learning process.

- *Manually engineered features and ad-hoc solutions*: Before early 2000s and first works on representation learning, creating a machine learning system meant to handcraft features and finding ad-hoc solutions, which may differ significantly depending on the task or domain. Having a general algorithm with a more systematic approach seemed for a long time a very distant goal. Manually engineered features is also a clear limitation to achieve autonomy, as new tasks need to have the same features or re-engineered ones.

- *Focus on supervised learning*: creating labelled data is probably the slowest and the most expensive step in most ML systems. This is why learning continuously has been for a long time not a viable and practical option.

The relaxation of these constraints, thanks to recent advancements and results in machine learning research, as well as the rapid technological progress witnessed in the last 20 years, have open the door for starting tackling more complex problems such as learning continually.

We argue that the robotics community, which has always been intrigued by endowing embodied machines with lifelong and open-ended learning [61] of new skills and new knowledge, would highly benefit from the recent advances of ML in this area. Robotics applications in unconstrained environments, indeed, have always raised questions out of reach for previous machine learning techniques. On the other hand, CL developed in the context of robotics is involved in understanding the role and the impact of the concept of "embodiment" in intelligent machines that learn and think like humans.

Learning, embodiment, and reasoning are presented as the three great families of challenges for robotics in [238]. We postulate that CL tackles the learning problem, taking into account the importance and constraints of embodiment. At best, CL would also benefit from reasoning in order to maximize the learning process. Thus, continual learning lies in the intersection of crucial robotics challenges.

Though lifelong learning approaches do exist in various ML disciplines (such as evolutionary algorithms for example [14, 13, 28, 15]), we will focus, in the rest of this thesis, on recent continual learning developments in the context of gradient-based neural network and deep learning approaches. For a more detailed description of many other classic approaches to continual learning with shallow architectures we refer the reader to [43].

3.3 Key vocabulary

Definition 1. ***Learning objective*** *The learning objective is composed of a data set and a loss function, that has to be optimized. The learning objective change if either the loss or the data change.*

Definition 2. ***Task*** *A task is a learning experience characterized by a unique task label t and its target function $g^*_{\hat{t}}(x) \equiv h^*(x, t = \hat{t})$, i.e., the objective of its learning.*

Definition 3. ***Task label*** *The task label is a variable that define tasks boundaries. It might be available or not depending on the learning scenario.*

Definition 4. ***Continuum*** *The continuum is the full learning experience. It is composed by a sequence of tasks.*

Definition 5. ***Data distribution*** *The data distribution, is a theoretical statistical distribution that generate the data. This distribution can be constant or may variate through time.*

Definition 6. ***Data stream*** *The data stream is the flux of samples generated by the data distribution. A task is a set of the data stream, the continuum is the full data stream.*

Definition 7. ***Forgetting*** *A neural network forget when its performance on a data distribution is decreased by learning on another one.*

Definition 8. ***Interferences*** *In machine learning, interferences are conflicts between two (or more) objective functions leading to prediction errors.*

Definition 9. ***Concept drift*** *The concept drift characterizes the learning objective variations, i.e. when the learning criterion or the data distribution changes. When there is no concept drift the learning objective is constant. Concept drift may lead to forgetting in neural networks models.*

3.3.1 Terminology Clarification

In this section we aim at clarifying the distinction and similarities of continual learning with related topics and terms used in the literature.

Online learning: Online learning is a special case of CL [110] where updates are done on per single data point basis and therefore, the batch size is one. Online learning algorithms are suited to scenarios where information should be processed instantly, either to adapt the model to learn as fast as possible or because data cannot be saved.

Few-shot Learning: Few shot learning [127, 70] is the ability to learn to recognize new concepts based on only few samples of them. It may be used for continual learning problems when the number of data points is very low. The extreme case of zero-shot learning consists of the ability to detect new classes while being trained with a disjoint set of classes [256].

Curriculum Learning: Curriculum learning [21] is a training process that proposes a sequence of more and more difficult tasks to a learning algorithm in order to make it able to learn, at last, a generally harder task. The sequence of tasks is designed in order to be able to learn the last one. Both CL and curriculum learning learn on a sequence of tasks (or partial experience). However, in curriculum learning, tasks are chosen in a way that makes possible to learn tasks of different complexity, by taking into account the difficulty of them, while in CL, tasks are not voluntarily chosen nor ordered. Furthermore, while the interest of curriculum learning ultimately lies into solving the last task, the continual learning objective is to be able to solve all tasks.

Meta-learning: Meta-learning [27] is a learning process that uses meta-data about past experiences, such as hyper-parameters, in order to improve its capacity to learn on new experiences. It also learns several different tasks; however, its goal is not to learn without forgetting but to progressively improve the learning efficiency while learning on more and more tasks. It is also called "learning to learn", and it can be used or not in a continual learning setting.

Transfer learning: Transfer learning [188, 73, 270] is the ability to use what has been learned from a previous task on a new task. The difference with continual learning is that transfer learning is not concerned about keeping the ability to solve previous tasks. In computer vision, transferring

what has been learned from a past environment to new environments would be often referred to as *domain adaptation* [183, 53].

Active Learning: Active learning is a special case of semi-supervised machine learning in which a learning algorithm is able to interactively query the user (or some other information source) to obtain the desired output labels for new data points [220, 221]. Active learning may be used in CL to query new examples and have control of the data the algorithm has access to.

3.3.2 Challenges Addressed by CL

In this section we describe the specific problems addressed by continual learning; the kind of problems that arise when data cannot be assumed i.i.d., and when the hypothesis that the data distribution is static is not valid.

Catastrophic Forgetting

Catastrophic forgetting [162, 76] refers to the phenomenon of a neural network experiencing performance degradation at previously learned concepts when trained sequentially on learning new concepts [162]. Since by definition the continual learning setting deals with sequences of classes or tasks, the catastrophic forgetting is an important challenge to be tackled. Catastrophic forgetting might also be referred to as *catastrophic interference*. The notion of interference is pertinent since the acquisition of new skills interferes with past skills by modifying important parameters as described in definition 8.

Handling Memories

One of the main components that distinguishes two CL approaches is the way they handle memories. In order to deal with catastrophic forgetting, each strategy should find a way to remember what may be destroyed by learning future tasks. Continual learning needs a mechanism to *store* memories of past tasks, which can take very various forms. It is important to note that memories can be saved in different manners: as raw data, as representations, as model weights, regularization matrices, etc. An efficient memory management strategy should only save important information, as well as be able to transfer knowledge and skills to future tasks. In practice, it is almost impossible to know what will be important and what could be transferable in the future; a trade off should then be found between the precision of the information saved and the acceptable forgetting. This trade-off problem is known as the stability/plasticity dilemma [163].

An important challenge inherent to handling memories is to automatically assess them. Learning new tasks may lead to degradation of the memories. As a consequence, the memory process needs mechanisms to evaluate how the memories are degraded, i.e., how it forgets. As no more data and labels from past tasks may be available, this check-up might be very challenging.

Another challenge is the stability of the learning process, which is crucial to learn and remember. Instability might lead to exploding gradients that would accidentally and permanently erase memories.

Detecting Distributional Shifts (concept drift)

When the distribution is not stationary, a shift into the data stream is observed. When there is no external information concerning this shift, the CL model has to detect it, and account for

fixing it by itself. An undetected shift in the data distribution will irrevocably lead to forgetting. Changes in the data distribution over time are commonly referred to as *concept drift*. This idea is related to online change detection algorithms [211, 170] or Bayesian surprise [237] in ML. Two kinds of concept drift are defined [82]: Virtual and real concept drift. Virtual concept drift concerns the input distribution only, and can easily occur, e.g., due to imbalanced classes over time. Real concept drift, on the contrary, is caused by novelty on data or new classes, and can be detected by its effect, on e.g., classification accuracy. However, shift may also happen when the task changes. In RL for example an agent may have to solve a new task. Then the shift is not exactly in the data distribution but in the supervision signal. Regardless of where exactly the shift happened it has to be detected to avoid catastrophic interference with non related skills or knowledge.

3.3.3 Learning Paradigms Orthogonal to Continual Learning

In this section we describe the relationship of continual learning with respect to the main three, generally acknowledged machine learning paradigms introduced in Chapter 2: supervised, unsupervised and reinforcement learning.

Supervised Continual Learning

Supervised learning is the machine learning problem of learning from input-output example pairs [205]. In Chapter 2, we introduced supervised learning and its different use cases.

While the study of continual learning in this context may help disentangling the complexity introduced by algorithms that learn continually, in the context of robotics, the lack of supervision does not allow, most of the time, to apply directly supervised methods.

Unsupervised Continual Learning

Unsupervised learning refers to machine learning algorithms that do not have labels or rewards to learn from. In Chapter 2, we already introduced unsupervised learning and in particular generative models. In the context of robotics, unsupervised continual learning may play an important role in building increasingly robust multi-modal representations over time to be later fine-tuned with an external and very sparse feedback signal from the environment. In order to learn robust and adaptive representations with unsupervised learning, the main objective is to find suitable surrogate and meaningful learning signals, as robotics priors [108, 138], self-supervised models or curiosity driven techniques.

A particular unsupervised task learned in a continual learning setting is the generation of images. Image generation is achieved by training generative models to reproduce images from a dataset. In a CL setting, the distribution changes over time and the generative model should be able to produce at the end images from the whole distribution. This problem has been studied for various generative models as adversarial models [263, 135], variational auto-encoders [172, 191, 2, 69, 135] and standard auto-encoders [249, 271].

There is also a different relation between unsupervised learning and CL, since unsupervised models can be used to learn representations from vast amounts of data sources and can then generate such data (cf Section 3.5.4). This capacity can then be used to perform CL for classification [265, 226, 249, 137] or reinforcement learning tasks [38].

Continual Reinforcement Learning

Reinforcement Learning is a machine learning paradigm where the goal is to train an agent to perform actions in a particular environment in order to maximize the expected cumulative reward. As explained in Chapter 2, in traditional RL, the world is modeled as a stationary MDP: i.e., fixed dynamics and states that can recur infinitely often [198]. Chapter 2 also presented a basic learning process of reinforcement learning. Since in general, complex RL environments have no access to all data gathered at once, RL could often be framed as a CL situation. Moreover, RL borrows several tools used in CL models, such as approximating data to an i.i.d. distribution, via either *i)* setting multiple agents or actors to learn in parallel [157], or *ii)* using a replay buffer (or experience replay [169]), that is equivalent to a particular category of CL (rehearsal, see Section 3.5.3). Another link is found in a popular stable method in RL, the TRPOalgorithm [215], which constrains learning by using an estimate of the Fisher information matrix to improve learning continually, in the same way as some CL strategies (e.g., EWC, see Section 3.5.1). Most of Continual Learning approaches in RL have been applied in simulation settings such as Atari games [121]. However, many approaches [248, 112, 13, 28] also solve use cases on real robots.

As we can see, the continual learning problems are real shortcoming in reinforcement learning algorithms. Improving continual learning will therefore necessarily help improving reinforcement learning performance.

3.4 A Framework for Continual Learning

Despite the rapidly growing interest in continual learning and mainly empirical developments of the recent years [179], very little research and effort has been devoted to a common formalization of algorithms that learn continually in dynamic environments. However, the availability of a common ground for thoroughly evaluating and understanding continual learning algorithms is essential to reduce ambiguities, enhancing fair comparisons and ultimately better advancing research in this direction.

3.4.1 Setting Description

Being able to better compare and evaluate continual learning strategies, while still being general enough to overlook implementation-dependent details over different learning paradigms, becomes essential. This is specially true when targeting deployment of CL paradigms in real-word applications, such as robotics. Nowadays, despite the existence of a basic set of shared practices, many are the fundamental questions often overlooked in recent continual learning research. For example, questions about the data availability during training and evaluation, the amount of supervision with respect to the tasks separation and composition, as well as common but biased assumptions on the nature of the data among others. A list of questions of interest we would like to address and report are the following:

3.4.2 Questions

(a) Data Availability

- $\boldsymbol{Q_1}$: *Does some data need to be stored? if yes, how and what for? (e.g. regularization, re-training, validation)?*

- $\boldsymbol{Q_2}$: *Is the algorithm tuned based on the final performance? I.e. is it possible to go back in time to improve performance?*
- $\boldsymbol{Q_3}$: *Are data distributions assumed i.i.d. at any point?*
- $\boldsymbol{Q_4}$: *Is each task assumed to be encountered only once?*

(b) Prior Knowledge

- $\boldsymbol{Q_5}$: *Is the continual learning algorithm agnostic with respect to the structure of the training data stream? (e.g. number of classes, numbers of tasks, number of learning objectives...)*
- $\boldsymbol{Q_6}$: *Does the approach need a pretrained model for the CL setting? If so, what is the new knowledge that needs to be acquired while learning continually?*

(c) Memory and Computational Constraints

- $\boldsymbol{Q_7}$: *How much available memory does the algorithm require while learning? Does the memory capacity requirement changes as more tasks are learned?*
- $\boldsymbol{Q_8}$: *Is the continual learning algorithm constrained in terms of computational overhead for each learning experience? Does the computational overhead increase over the task sequence?*
- $\boldsymbol{Q_9}$: *Is the continual learning algorithm agnostic with respect to the data type? (e.g. images, video, text,...)*
- $\boldsymbol{Q_{10}}$: *Is the continual learning algorithm able to handle situations where there is not enough time to learn?*

(d) Amount/Type of Supervision

- $\boldsymbol{Q_{11}}$: *In the presence of multiple tasks, is the task label available to the algorithm during the training phase? And during evaluation?*
- $\boldsymbol{Q_{12}}$: *Are all the data labeled? or only the first training set? Can the user provide sparse label/feedback (e.g. active learning) to correct the system errors?*

(e) Performance Expectation

- $\boldsymbol{Q_{13}}$: *What is expected from the algorithm to remember at the end of the full stream? Is it acceptable to forget somehow, when task, context or supervision change?*

To summarize these questions, in any new CL algorithm proposition, it is fundamental to clearly describe the data stream, its use, the algorithm functioning, its assumed prior knowledge, and its requirements in terms of supervision, memory and computation. We will answer those questions in the discussion chapter of the thesis (Chapter 8).

We will now propose a comprehensive and detailed framework to help distinguish and disentangle different approaches in different continual learning settings and help answer these questions.

Early theoretical attempts to formalize the CL paradigm are found in [198] as a combination between reinforcement learning and inductive transfer. More general framework approaches include the one on non i.i.d. tasks of [185]. As in [185], we assume CL is tackling a probably approximately correct (PAC) learnable problem in the approximation of a target hypothesis h^* as well as learning from a sequence of non i.i.d. training sets. Our framework could also be seen as a generalization of the one proposed in [150], where learning happens continuously through a *continuum* of data and a "task supervised signal" t may be provided along with each training example.

3.4.3 Framework Definitions

In continual learning data can be conveniently seen as drawn from a sequence of distributions D_i, and thus the need to redefine a CL framework taking into account this important property is defined as follows.

Definition 10. ***Continual Distributions and Training Sets***
In Continual Learning, $\mathcal{D}$ is a potentially infinite sequence of unknown distributions $\mathcal{D} = \{D_1, \ldots, D_N\}$ over $X \times Y$, with X and Y input and output random variables, respectively. At time i a training set Tr_i containing one or more observations is provided by D_i to the algorithm.

As the framework hereby proposed is supposed to be general enough to cover the orthogonal and classical unsupervised, supervised and reinforcement learning approaches, Tr_i, as better detailed in Definition 12, is a collection of training observations/data samples that act as signal of the joint distribution to be learned.

Definition 11. ***Task***
A task is a learning experience characterized by a unique task label t and its target function $g^*_{\hat{t}}(x) \equiv h^*(x, t = \hat{t})$, i.e., the objective of its learning.

It is important to note that the tasks are just an abstract representation of a learning experience represented by a task label. This label helps to split the full learning experience into smaller learning pieces. However, there is not necessarily a bijective correspondence between data distributions and tasks.

Definition 12. ***Continual Learning Algorithm*** *Given h^* as the general target function (i.e. our ideal prediction model), and a task label t, a continual learning algorithm A^{CL} is an algorithm with the following signature:*

$$\forall D_i \in \mathcal{D}, \qquad A^{CL}_i : \ < h_{i-1}, Tr_i, M_{i-1}, t_i > \rightarrow < h_i, M_i > \tag{3.1}$$

Where:

- *h_i is the current hypothesis at timestep i, or, practically speaking, the parametric model learned continually.*
- *M_i is an external memory where we can store previous training examples or partial computation not directly related to the parametrization of the model.*
- *t_i is a task label, that can be used to disentangle tasks and customize the hypothesis parameters. For simplicity, we can assume N as the number of tasks, one for each Tr_i.*
- *Tr_i is the training set of examples. Each Tr_i is composed of a number of examples e^i_j with $j \in [1, \ldots, m]$. Each example $e^i_j =< x^i_j, y^i_j >$, where y^i is the feedback signal and can be the optimal hypothesis $h^*(x, t)$ (i.e., exact label y^i_j in supervised learning), or any real tensor (from which we can estimate $h^*(x, t)$, such as a reward r^i_j in RL).*

It is worth pointing out that each D_i, can be considered as a stationary distribution. However, this framework setting allows to accommodate continual learning approaches where examples can also be assumed to be drawn non i.i.d. from each D_i over $X \times Y$, as in [82, 91].

Definition 13. ***Continual Learning scenarios*** *A CL scenario is a specific CL setting in which the sequence of N task labels respects a certain "task structure" over time. Based on the proposed framework, we can define three different common scenarios:*

- *Single-Incremental-Task (SIT):* $t_1 = t_2 = \cdots = t_N$.
- *Multi-Task (MT):* $\forall i, j \in [1, .., n]^2, i \neq j \implies t_i \neq t_j$.
- *Multi-Incremental-Task (MIT):* $\exists\ i, j, k :\ t_i = t_j$ *and* $t_j \neq t_k$.

Table 3.1 illustrates an example to clarify the definition of SIT, MT and MIT.

An example of Single-Incremental-Task (SIT) scenario is an ordinary classification task between cats and dogs, where the distribution changes through time. First, there may only be input images of white dogs and white cats, and later only black dogs and black cats. Therefore, while learning to distinguish black cats from black dogs the algorithm should not forget to differentiate white cats from white dogs. The task is always the same, but the concept drift might lead to forgetting.

However, in a classification setting, a Multi-Task (MT) scenario would first consist of learning cats versus dogs, and later cars versus bikes, without forgetting. The task label changes when the classes change, and the algorithm can use this information to maximize its continual learning performance. The Multi-Incremental-Task (MIT) is the scenario where the same task can happen several times in the sequence of tasks.

Table 3.1: Illustration of continual learning scenarios. Sequential task labels (corresponding to different distribution $D_i \in \mathcal{D}$) to reflect differences among CL categorization w.r.t. number and unicity of tasks for SIT, MT and MIT. Notice that a MIT setting requires relaxing the constraint definition of SIT but also relaxing the constraint definition of MT, i.e., it corresponds to the case where not all the tasks are considered having the same *ID*, and not all the task are considered distinct.

Task ID/Session	**CL settings**		
Task ID	**SIT**	**MT**	**MIT**
t_1	0	1	0
t_2	0	2	1
t_3	0	3	0
...	...	...	...
t_i	0	i	...

In any learning problem (be it classification, RL or unsupervised learning), the ability to adapt to new concepts to be learned (from the PAC ML framework [251]), as well as new instances of each concept, should be accounted. This is the objective of the next definition where we formally set three different settings an algorithm is required to manage, as they can have very high impact on the algorithm performance.

Definition 14. ***Task label and concept drift scenarios*** *The task label can specify different assumptions made in a continual learning scenario. We can define three main categories of task label assumptions regarding concept drift:*

- *No task label: Changes in the distribution are not signaled by any task label. The task is always the same (equivalent to SIT scenario).*

- *Sparse task label: Changes in the distribution are sparsely signaled by the task label. There are several tasks but changes in distribution may as well happen inside a task.*
- *Task label oracle: Every change in the data distribution is signaled by the task label, which is given.*

We illustrate the different scenarios in Figure 3.1.

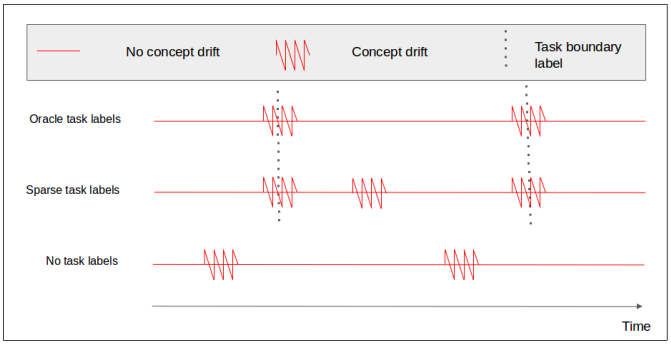


Figure 3.1: Task label and concept drift: illustration of the different scenarios.

Definition 15. ***Availability of task label.*** *When a task label is provided, it is worth distinguishing among two different cases:*

- *Learning labels: Task labels are provided for learning only. At test time, inference should be done without knowing from which task a data point is coming from.*
- *Permanent labels: The task labels are provided for learning, and it is assumed they will also be provided at test time for inference.*

The need for a task label is a need for supervision. It might be legit for training since it helps significantly the learning process. However, it might be a shortcoming in terms of autonomy at test time (for inference).

Definition 16. ***Content Update Type*** *The nature of the data samples or observations contained in each Tr_i can be conveniently framed in three different categories:*

- *New Instances (NI): Data samples or observations contained in the training set at time-step i relate to the same dependent variable Y used in the past.*
- *New Concepts (NC): Data samples or observations contained in the training set at time-step i relate to a new dependent variable Y to be learned from the model.*
- *New Instances and New Concepts (NIC): Data samples or observations contained in the training set at time-step i relate to both, already encountered dependent variables, and new ones (Y).*

In order to exemplify the concept of *Content Update Type* defined in Definition 16, let us recover the aforementioned example of classification. If an algorithm learns the *cat vs dogs* classification problem on a dataset and then new images of cat vs dogs are provided to the algorithm, we are then in a *New Instances* case (NI), we have new data but no new concepts. If the new instances were of different classes (e.g. cars vs bikes) we then would face the *New Concepts* case (NC). The new instances and new concepts case would then have been a mix of both new images of known and new classes.

If a CL algorithm uses a network pretrained on a dataset, the features of such dataset will need to be accounted for as one more task or the same, depending on the distribution of new instances and new classes according to definitions 13 and 16. In other words, using a pretrained model is similar to assume there is a task already learned by the model, and the new learning experiences of the algorithm are just a continuum of learning curricula. If there is any intersection between the pretraining and the new tasks, it should be reported in the setting description. The pretraining effect can then be estimated with the metrics proposed in Section 3.6.2.

3.4.4 Constraints

In this section, we phrase constraints that are advised to be respected to make small experiments scalable to bigger ones.

Constraint 1. *For every step in time, the number of current examples contained in the memory is lower than the total number of previously seen examples*[1].

Constraint 2. *Memory and computation for each iteration step i are bounded. Given two functions $ops()$ and $mem()$ that compute the number of operations and memory occupation required by A_i^{CL}, two reasonably small values* max_ops *and* max_mem *should exist, such that, for each i, $ops(A_i^{CL}) < max_ops$ and $mem(h_{i-1}, M_{i-1}) < max_mem$.*

max_ops and max_mem are the max throughput, in number of operations, and the max memory capacity of the system running A_i^{CL}. Having a memory and computational bounds for each iteration i is an important constraint for a continual learning algorithm. The reason is that the number of training sets Tr_i can potentially be unlimited, and thus, computation and memory should not be proportional to the number of hypothesis updates h_i over time. A finite upper bound should exist and be considered, especially with $n \to \infty$.

3.4.5 Relaxation and desiderata

Given the difficult setting and the additional constraints imposed by Continual Learning with respect to the classic "static" setting, many researchers in the recent literature have proposed new CL strategies in slightly relaxed [206, 121, 153, 150] yet reasonable settings:

Relaxation 1. *Memory relaxation: Removes the fixed memory bound constraint over $ops()$ and $mem()$.*

Relaxation 2. *Computation relaxation: Removes the fixed computational bound constraint $ops(h_i) < max_ops$.*

[1] I.e., if we could fit all previous examples in memory M, it would become a problem of scarce interest for the CL community, given that re-training the entire model h_i from scratch would be always possible [110]

In both cases we assume that for practical applications, either a finite (and reasonable) number of tasks N are encountered or that we are transigent with the memorization of past tasks, hence, for many settings with a generous memory and computational bounds, many continual learning strategies that, in terms of complexity and memory usage, grow somehow proportional to the number of training sets Tr_i may still be a viable option, especially if they can guarantee better performance.

Having defined a formal framework for CL, we can therefore highlight a number of desiderata:

Desideratum 1. *Online Continual Learning: Limit the size of each training set, moving towards online learning so that* $|Tr_i| = 1$.

Being able to learn without storing any raw data would mean a large step towards continual learning. In fact, getting rid of storing raw data means that the learning algorithm is able to extract information from the current task that may be not only useful and accurate for the current task, but also transferable for the future.

In our biological counterparts, namely the brain, a system-level consolidation process is often thought to take place, where memories are encoded, stored and than retrieved for rehearsal purposes [56]. However, the idea of storing high-dimensional perceptual data appears impractical given the incredible amount of information flowing into our brain every day from our multi-modal senses. Being able to process data online as well, is an important desideratum especially for reducing adaptation time and operational memory usage in an embedded or robotics setting. At least, unprocessed data should be selected to keep only essential information.

Desideratum 2. *Task indicator free Continual Learning: Learning continually without help of an external signal t indicating the current task, in particular at test time, is strongly desirable.*

Desideratum 3. *Be ready for the future: Prepare the model to be robust and provided with good representation for handling future learning experiences. Solving only the current task only is probably not sufficient.*

3.5 State of the art

Continual learning can be classified in four families of approaches. Each family has its own method to manage memories and learning without forgetting in non-iid settings.

3.5.1 Regularization

Regularization is a process of introducing additional information in order to prevent overfitting [30]. In the context of Continual Learning, the model should not overfit a new problem because it would make it forget it's previous skills. The regularization approaches in continual learning consist of modifying the update of weights when learning in order to keep memory of previous knowledge.

Penalty Computing

Basic regularization techniques that could be used for CL are weight sparsification, dropout [86], and early stopping [155]. These simple regularization techniques reduce the chance of weights being modified, and thus decrease the probability of forgetting. More complex methods consist of searching for important weights inside the models and protect them afterwards to prevent forgetting.

The Fisher matrix can be used to estimate the importance of weights and produce an adapted regularization as for Elastic Weight Consolidation (EWC) approach [121]. For efficiency purpose, EWC only use the diagonal of the Fisher matrix to estimate importance. [201] proposes an alternative to get a better estimation of the Fisher matrix using the Kronecker factorization. EWC approach needs to have clear task delimitation to compute Fisher matrix at the end of the task, but Synaptic Intelligence (SI) [269] extended the method in an online learning fashion to relax this constraint. [133] propose to use a regularization method called *incremental moment matching* to overcome catastrophic forgetting. This method saves the moment posterior distribution of neural networks weights from past tasks and uses it to regularize learning of a new task. Two different declinations of this method are proposed: one with the use of first order moment *IMM-mean* and one with second order moment *IMM-mode*.

Another method to apply regularization for continual learning is the use of *Conceptor* [102, 95]. Conceptor are memory mechanism that store learned patterns and representation. They are used to guide the gradient of the loss function to prevent forgetting and then favor modification for some weights and penalize others.

The regularization methods have been shown to be efficient in reinforcement learning [121], classification [121, 201, 269, 95] and also generative models [172, 218]. A limitation is that after several tasks the model may saturate because of a too high regularization, and finding a good trade-off between regularization that allows learning without forgetting may be hard. Some regularization methods are described more in depth in Chapter 4.

Knowledge Distillation

Distillation techniques were introduced by [99] in order to transfer knowledge from neural network A to neural network B. The idea is that after A has learned to solve a task, we want B to share this skill with A. We then forward the same input to both A and B and impose B to have the same output as A. Distillation should be more efficient than retraining B because A produces a soft-target that helps B to learn faster. In order to apply this method for continual learning, after network A learned to solve the first task, and while B is learning the second one, we distill knowledge from A to B. In the end, B should be able to solve both tasks. This and related methods have been used in various approaches [263, 217, 78, 207, 112, 248, 58, 164]. A drawback of distillation is that it generally needs to preserve a reservoir of persistent data learned for each task in order to apply distillation from a teacher model to a student model. Distillation can also be used to transfer policy learning from one model to another [207].

3.5.2 Dynamic Architecture

The architecture of learning models has a strong influence on how they learn. One approach to CL is to modify dynamically the architecture of a model to make it learn new concepts or skills without interfering with old ones. We present two types of dynamic architectures. First, when the changes in the architecture are explicit; and second, when changes are implicit architectural changes by freezing weights. We also present an important architectural approach to CL: dual memory models.

Explicit Architecture Modification

Explicit dynamics architecture gather all methods that add, clone or save parts of parameters of the models to avoid catastrophic forgetting.

Progressive neural networks [206] is one of the first approaches within this paradigm for deep neural networks. For each new task to be learned, a new model is created connected to all past ones. The goal of this new model is to learn the new task by using what was already learned by previous models, and so develop the new skills needed. At test time, the method needs to inject data into each neural network previously created, and needs to know the task index to pick the right output. Because the weights are used to connect neural networks together, the growth of parameters is quadratic w.r.t. the number of tasks. This growth is generally to be prevented. Instead, layers may be dynamically expanded in a single network without the need of re-training or freezing previously learned parameters, improving model capacity over time [257].

Another type of dynamic architecture strategy consists of dynamically adding neurons for new tasks. As an example, output layers can be added in order not to change output parameters from previous tasks as in LWF approach [143]. This method ensures that the output layer will not be modified; however, as the feature extraction layers are shared between tasks, some parameters risk to be modified and forgotten. In addition, at test time, the method needs the task label.

It is worth mentioning that we consider as *dynamic architecture*, those approaches that adapt their architecture specifically with the aim of not forgetting, while similar mechanisms can be used for other purposes[2].

Implicit Architecture Modification

Implicit architecture modification is the use of model adaptation for continual learning without modifying its architecture. This adaptation is typically achieved by inactivating some learning units or by changing the forward pass path.

We categorize the fact of dynamically freezing weights as an implicit dynamic architecture approach. It is implicit because the architecture of the model does not change; however, the model's capacity is necessarily affected.

Freezing weights consist of choosing some weights at the end of a task that will no more change in the future. The backward pass will not be able to tune them anymore; however, they can still be used in the forward pass. This method ensures that these weights will not be modified, and tries to keep enough free parameters to learn in the future [154, 153, 219]. The difficulty lies in freezing enough weights to remember, but not too much to still be able to learn new skills. The way weight freezing is implemented in PackNet [154], Piggypack [153] or HAT [219] is by defining a special mask for each task that is used to both protect weights when new tasks are learned, and to define which weights to use at inference time for a given task. The use of masks to freeze important weights can be referred to as hard attention process [219]. Weight freezing can also be used to keep the decision boundary of the output unchanged [109].

An alternative to a weight freezing when tasks change is to define a dynamics path inside the model in order to use a specific path for a specific task and not modify already learned weights. This is the idea exploited in *PathNet* [71].

[2]If the architecture is changed without this objective, it is not considered as part of the CL approach. As an example, when new classes are available, we might choose to make the output size grow to handle these, without making it as a way to not forget.

The use of implicit architecture modifications is not incompatible with explicit architecture modification as it is shown in [153, 219].

Dual Architectures

Dual approaches characterize architectures that are split into two models. One model is used in order to learn the current task and should be easily adaptable, while the second model is used as a memory of past experiences. This approach can be linked to interactions between the hippocampus and neocortex to avoid catastrophic interference in mammals [161]. The stable network plays the role of the neocortex, and the flexible one plays the role of hippocampus [78, 82, 83, 155].

The use of dual architecture is explicit in many bio-inspired approaches such as [78, 83, 178, 235, 116]. Dual architectures are extended in [235] with the addition of an embedding model, and then, continual learning happens in the embedding space. The dual architecture can also be extended to more than two components, as in FearNet [116], which takes inspiration from the basolateral amygdala from the brain to add a third component that is able to choose between the flexible and the stable memory for recall.

3.5.3 Rehearsal

Rehearsal approaches gather all methods that save samples as memory of past tasks either raw or processed.

These samples are used to maintain knowledge about the past in the model. Ideally, those samples are carefully chosen in order to be representative of past tasks; by default, they can be randomly chosen.

The initial strategy is to save the representative samples and incorporate them in the new training set [193, 92, 135]. In the third article samples are chosen randomly for continual learning of generative models but in [193, 92] the set is carefully sorted in order to keep the most representative samples into a coreset. This process allows to dynamically adapt the weights of the feature extractor and strengthen the network connections for memories already learned without forcing to keep previous weights.

However, the coreset can also be used for regularization purpose and not just to be replayed from time to time along with new data in the learning process.

For example, the coreset can be used for distillation in [202], in A-LTM (Active Long Term Memory Networks) [78] and in DisCorl for continual reinforcement learning [248]. They can also be useful to regularize the gradient when learning new tasks as in GEM (Gradient Episodic Memory) [150], A-GEM (Averaged Gradient Episodic Memory) [42] and [7]. Coresets have also been used to regularize the continual learning of a generative model in the CloGAN approach [200]. In a bayesian learning setting the coreset can be incorporated into the prior to regularize learning update as in [172]. The authors experimented the use of a coreset to create a variational continual learning model (VCL).

The coresest should be built in order to maintain knowledge and to be able to learn to distinguish old data from new ones. Moreover, the saved samples should not be used extensively in order to avoid overfitting them. In [5], the authors save samples but only use samples creating interference with current tasks. This makes it possible to use only appropriate memories.

The disadvantage of rehearsal approaches is the utilization of a separate memory for raw and unprocessed data which is a vanilla way of saving knowledge that does not respect data privacy. Nevertheless it ensures that the memories are not degraded through time. In order to mitigate this

problem, some methods aimed at saving data point representation for rehearsal purpose using deep neural networks latent representation [18, 32, 184].

Another problem could arise in rehearsal methods; since memory is limited, there could be a large unbalance between data from previous tasks and new data. However, this problem can be partially addressed by rescaling weights of learning criterion [19, 100, 264].

Pseudo-rehearsal [202, 255], a similar method to rehearsal, is is an approach to continual learning that generates pseudo-input as a memory of past tasks. The pseudo-inputs are data points generated by gradient descent on the classification models to match a randomly sampled output. Pseudo-rehearsal has been abusively used as a synonym to generative replay described in the next section in the continual learning literature. However, the fundamental difference is that the pseudo-inputs depend only on the classifier knowledge and are not produced by a generative model.

3.5.4 Generative Replay

Instead of modeling the past from few samples as it is done in *Rehearsal* approaches, *Generative Replay* approaches train generative models on the data distribution. Therefore, they are able to afterwards sample data from past experience when learning on new data. By learning on current data and artificially generated past data, they ensure that the knowledge and skills from the past is not forgotten. These methods have also been associated with the term *Intrinsic Replay* [62]. They could be understood as methods that perform *regeneration* of samples or internal states, and thus, they can be associated with model-based learning, where the model learns the data distribution of past experiences. The generative models is generally a GAN [88] as in [265, 135, 226] or an auto-encoder as in [62, 116, 38, 113].

A classical method implementing a generative replay normally makes use of dual models [113, 226, 265, 69, 116]. One frozen model generates samples from past experiences and another learns to generate and classify current samples in addition to the regenerated ones. When a task is over, we replace the frozen model by the current one, freeze it, and initialize a new model to learn next task.

Generative Replay models can be categorized into two different approaches: "Marginal Replay" and "Conditional Replay" [137]. Techniques using *Marginal Replay* make use of standard generative models, while *Conditional Replay* are a particular case of the former where the generative model is conditional. Conditional models can generate data from a specific condition, e.g. a class or a task. In continual learning, it allows then to choose from which past learning experience we want to generate. It is important for example to balance data in generated replay [137].

While most of the Generative Replay based approaches are meant to solve classification tasks [116, 113, 226, 265, 200], some models use it for unsupervised learning [135, 263] or reinforcement learning [38].

3.5.5 Hybrid Approaches

Memorization processed described are not incompatible, therefore they can be combined. Most CL approaches have an implicit dual architecture strategy, as they always have a slow learning and a fast learning mechanisms to learn continually. For example, in rehearsal approaches the stable model role is played by a memory that stores samples, in generative replay approaches a generative model plays the role of stable model, in some regularization approaches the stable model is played by the Fisher matrix which saves important weights.

Moreover, most of continual learning approaches do not rely on a single strategy to tackle catastrophic forgetting. As stated in previous sections, each approach offers advantages and disadvantages, but most of the times, combining strategies allows to find the best solutions. We summarize in Table 3.2 and Figure 3.2 the different approaches cited and the strategies they propose.

Table 3.2: Classification of continual learning main strategies

References	Regularization	Rehearsal	Architectural	Generative-Replay
Zhou et al. [271]			✓	
Goodfellow et al. [86]	✓			
Lyubova et al. [152]		✓		
Rusu et al. [207]	✓			
Camoriano et al. [34]	✓	✓		
Furlanello et al. [78]	✓		✓	
Li et al. [143] (LwF)	✓		✓	
Rusu et al. [206] (PNN)			✓	
Jung et al. [109]	✓		✓	
Aljundi et al. [6]			✓	
Rebuffi et al. [193] (Icarl)	✓	✓		
Kirkpatrick et al. [121] (EWC)	✓			
Fernando et al. [71]			✓	
Lee et al. [133]	✓			
Lee et al. [267]	✓			
Triki et al. [249]	✓			
Seff et al. [218]	✓			
Shin [226] (DGR)				✓
Velez et al. [254]	✓			
Lopez-Paz et al. [150] (GEM)	✓	✓		
Zenke et al. [269] (SI)	✓			
Nguyen et al. [172] (VCL)	✓	✓	✓	
Ramapuram et al. [191]	✓			✓
Mallya et al. [154]			✓	
Kamra et al. [113]				✓
Draelos et al. [62]				✓
Serra et al. [219]	✓			
Mallya et al. [153]			✓	
Parisi et al. [178] (GDM)	✓		✓	✓
He et al. [95]	✓		✓	
Hayes et al. [92]		✓		
Wu et al. [265]		✓		✓
Ritter et al. [201]	✓			
Schwarz et al. [217]		✓		
Maltoni et al. [155]	✓		✓	
Achille et al. [2]			✓	✓
Wu et al. [263] (MeRGAN)	✓			✓

Continued on next page

Table 3.2 – *Continued from previous page*

References	Regularization	Rehearsal	Architectural	Generative-Replay
Dhar et al. [58]	✓			
Lesort et al. [135]				✓
Castro et al. [39]		✓		
Caselles-Dupré et al. [38]				✓
Riemer et al. [197] (MER)	✓	✓		
Rios et al. [200] (CloGAN)	✓	✓		✓
Lesort et al. [137]				✓
Sprechmann et al. [235]		✓	✓	
Hayes et al. [92] (ExStream)		✓		
Belouadah et al. [18] (DeeSIL)		✓		
Kemker et al. [116] (FearNet)			✓	✓
Chaudhry et al. [42]	✓	✓		
Wu et al. [264]		✓		
Kalifou1 et al. [112]	✓	✓		
Aljundi et al. [5]	✓	✓		✓
Michieli et al. [164]	✓	✓		
Caccia et al. [32]		✓		
Hou et al. [100]		✓		
Traore et al. [248]	✓	✓		
Aljundi et al. [7]	✓	✓		
Belouadah et al. [19]	✓			

3.6 Evaluation

Before applying CL solutions to autonomous agents, they should be experimented and evaluated in simulation or toy examples. It is crucial to have a set of good evaluation metrics and benchmarks to assess if the approaches are scalable to real problems or may not solve harder ones. It is important to distinguish the evaluation of the performances and the evaluation of algorithms. In research, the evaluation should be able to make predictions of success in applications, while in applications, the evaluation only assesses if the results are satisfying. Therefore, in research, we assess the method while in an application we should assess the result. In this section we summarize existing evaluation methods and benchmarks and highlight some of them we believe worth using when targeting the deployment of practical CL applications.

3.6.1 Benchmarks

In continual learning, the difficulty of learning on a sequence of tasks is first of all dependant on the difficulty of each of the tasks separately. If a task is difficult to learn, a model will have to deeply modify its weights. If those weights contain knowledge from previous tasks, there is a high probability they will be degraded. On the other hand, the risk of forgetting is also dependant on the likelihood of tasks occurring. Indeed, after learning a task T_t, it is easier for a neural network to

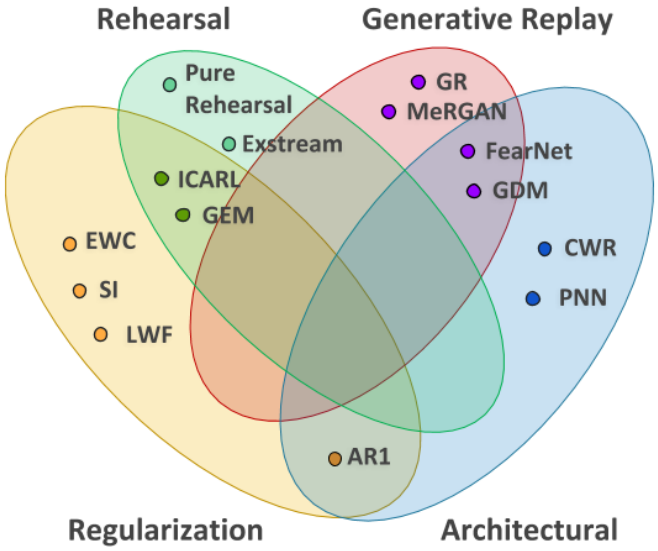


Figure 3.2: Venn diagram of some of the most popular CL strategies w.r.t the four approaches illustrated in Section 3.5: CWR [149], PNN [206], EWC [121], SI [269], LWF [143], ICARL [193], GEM [150], FearNet [116], GDM [178], ExStream [92], Pure Rehearsal, GR [226], MeRGAN [263] and AR1 [155]. Rehearsal and Generative Replay upper categories can be seen as a subset of replay strategies. Better viewed in color.

learn a radically different task T_{t+1} without forgetting, than learning a task T_{t+1} with similarities to T_t [69].

There are several kinds of similarities in a sequence of tasks:

- Similarities in learning objectives: They occur when the objective is similar from task to task. For example, in a classification setting, when the same classes are used from one task to another (e.g. Permuted MNIST), or in RL, the same tasks need to be achieved in different environments.

- Similarities in features: the features from task to task are the same or very similar (e.g. Rotation MNIST).

Beyond the similarity among tasks and the learnability of each task, the availability of data is primordial to evaluate the difficulty of a benchmark. For convenience, most of the classical benchmarks assume that each task is available long enough to learn a satisfying solution. Nevertheless, even when there is no constraint on the time to learn a task, data from the past cannot be available again in the future. In several approaches, past data is used for model selection, however using the performance obtained on task T_t to fine-tune a model that will learn on T_0 violates temporal causality [186]. Data might be saved for later use as in rehearsal approaches, but this must be done before moving on to the next task.

Most CL benchmarks are adapted from others fields, for instance:

Table 3.3: Benchmarks and environments for continual learning. For each resource, paper use cases in the NI, NC and NIC scenarios are reported.

Benchmark	NI	NC	NIC	Use Cases
Split MNIST/Fashion MNIST		✓		[137, 135, 95, 200]
Rotation MNIST	✓			[150, 137, 197]
Permutation MNIST	✓			[86, 121, 71, 226, 269, 137, 95, 197]
iCIFAR10/100		✓		[193, 155, 39, 116, 19, 264, 100]
SVHN		✓		[115, 218, 200]
CUB200	✓			[267, 92]
CORe50	✓	✓	✓	[149, 178, 155, 92]
iCubWorld28	✓			[180, 148, 92]
iCubWorld-Transformation		✓		[181, 33]
LSUN		✓		[263]
ImageNet		✓		[193, 153, 39, 18, 264, 100, 19]
Omniglot		✓		[128, 217]
Pascal VOC		✓		[164, 227, 164]
Atari	✓			[207, 121, 217]
RNN CL benchmark			✓	[233]
CRLMaze (based on VizDoom)	✓			[147]
DeepMind Lab	✓			[157]
MNIST Fellowship	✓	✓		[139]

- **Classification**: MNIST [132], Fashion-MNIST [266], KMNIST [46], CIFAR10/100 [123], Street View House Numbers (SVHN) [171], CUB200 [260], LSUN [268], MNIST-Fellowship [134], ImageNet [124], Omniglot [128] or Pascal VOC [66] (object detection and segmentation).

- **Reinforcement Learning**: Arcade Learning Environment (ALE) [16] for Atari games, SURREAL [68] for robot manipulation and RoboTurk for robotic skill learning through imitation [156], *CRLMaze* extension of VizDoom [147] and DeepMind Lab [157].

- **Generative models**: Datasets that prevail in this domain are the same as those used in classification tasks.

These datasets are then split, artificially modified (e.g., with image rotations or permutation of pixels) or concatenated together to create sequences of tasks and build a continual learning setting. As an example, permuted MNIST [121] and rotated MNIST [150] are continual learning datasets artificially created from MNIST.

Another possible continual learning scenario is the use of naturally non i.i.d. datasets (e.g. NICO [96]) or learning sequentially different datasets either on the same input space [133, 219] or in a multi-modal fashion [115]. However, only few datasets, such as CORe50 [149], OpenLORIS-Object [223], OpenLORIS-Scene [225] or MNIST-Fellowship [134], are specifically built with continual learning in mind.

In robotics, numerous datasets are often recorded in a online fashion through video. Therefore, they are suitable to evaluate continual learning algorithms. As an example, those proposed by [180, 181, 10] are composed of sequences of images captured during robotics object manipulation;

they are used for classification and detection algorithms. A summary of the main datasets and examples of their applications can be found in Table 3.3.

For the remainder of the thesis, we experiment with quite easy datasets, such as MNIST, Fashion-MNIST or KMNSIT (presented in Chapter 2), to focus more on continual learning problems rather than on learning problems. Solving hard problems is important to know if approaches are scalable but for prototyping purposes, it is not always necessary. We discuss this point furthermore in Chapter 8.

3.6.2 Metrics

Following the algorithm evaluation on a benchmark, we should make sure that the evaluation criteria are rigorous and cover the representative aspects of algorithm capacities. A thorough research evaluation should report more than just the final accuracy. We should also evaluate how fast it learns and forgets, if the algorithm is able to transfer knowledge from one task to another, and if the algorithm is stable and efficient while learning. In this section we gather a set of metrics to evaluate a CL approach.

For evaluation purpose, we assume access to series of test sets Te_i. The aim is to assess and disentangle the performance of our hypothesis h_i as well as to evaluate if it is representative of the knowledge that should be learned by the corresponding training batch Tr_i.

For instance, one example of such evaluation is one of the first metrics proposed for CL [91]; it consists of an overall performance $\mathcal{M}$ in a supervised classification setting. It is based on the relative performance of an incrementally trained algorithm with respect to an offline trained algorithm (which has access to all the data at once). In our notation, $\mathcal{M}$ is:

$$\mathcal{M} = \frac{1}{N}\sum_{i=1}^{N}\frac{R_{i,i}}{R^C_{i,i}}. \tag{3.2}$$

Where N is the number of tasks encountered, $R^C_{i,j}$ is the potentially best accuracy we can have on Te^C_i if the model was trained with all data at once, i.e. on Tr^C_i (the accumulation of training sets Tr^C_t from t=0 to t=i). Te^C_i is the accumulation of all test sets Te^C_t from $t = 0$ to $t = i$. $\mathcal{M} = 1$ indicates identical performance to an off-line cumulative setting; an $\mathcal{M}$ larger than one is possible when the offline model is worse than trained in a CL paradigm.

In [219], instead, the authors try to directly model forgetting with the proposed *forgetting ratio* metric ρ after learning i tasks, defined as:

$$\rho^{j\leq i} = \frac{1}{N}\sum_{i}^{N}\sum_{j}^{N}\left(\frac{R_{ij} - R^R_j}{R^C_{ij} - R^R_j} - 1\right) \tag{3.3}$$

Where, R^R_j is the accuracy of a random stratified classifier using the class information of task j.

Always in the same sequential setting, in [150] other three metrics are proposed: *Average Accuracy* (ACC), *Backward Transfer* (BWT), and *Forward Transfer* (FWT). In this case, after the model finishes learning about the training batch Tr_i, its performance is evaluated on all (even future) test batches Te_j.

The larger these metrics, the better the model. The metrics are extended for more fine grained, generic evaluation [59] so that the original accuracy [150] (as well as BWT and FWT) can account for performance at *every timestep in time*. Average Accuracy is defined as:

$$ACC = \frac{\sum_{i=1}^{N}\sum_{j=1}^{i} R_{i,j}}{\frac{N(N+1)}{2}} \tag{3.4}$$

where $R \in \mathbb{R}^{N\times N}$ is the training-test accuracy matrix that contains in each entry $R_{i,j}$ the test classification accuracy of the model on task t_j after observing the last sample from task t_i, Average Accuracy (ACC) considers all accuracies of training set Tr_i and test set Te_j by considering the diagonal elements of R, as well as all elements below it (i.e., averages $R_{i,j}$ where $i >= j$ see Table 3.4).

It should not be confused with the classical accuracy A.

$$A = \frac{\sum_{i=1}^{N} R_{i,i}}{N} \tag{3.5}$$

A computes the average performance at time t and does not take into account the previous performance evolution. We can compare A and ACC to get some more insight of the algorithm behaviour. If $A > ACC$ then the mean performance on past tasks improved through the learning of new tasks, if $A < ACC$ then the mean performance decreased and the algorithm forgot and, finally, if $A = ACC$, then either the algorithm stays very stable or the progress and forgetting compensate each others.

Table 3.4: Illustration of accuracy matrix R: elements accounted to compute ACC (white & cyan), BWT (cyan), and FWT (gray). $R^* = R_{ii}$, Tr_i = training, Te_i= test tasks.

R	Te_1	Te_2	Te_3
Tr_1	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$
Tr_2	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$
Tr_3	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$

Backward Transfer (BWT) measures the influence that learning a task has on the performance on previous tasks. It is defined as the accuracy computed on Te_i right after learning Tr_i as well as at the end of the last task on the same test set (see Table 3.4 in light cyan).

$$BWT = \frac{\sum_{i=2}^{N}\sum_{j=1}^{i-1}(R_{i,j} - R_{j,j})}{\frac{N(N-1)}{2}} \tag{3.6}$$

The original BWT [41, 150] is extended into two terms to distinguish among two semantically different concepts (so that, as the rest of metrics, is to be maximized and in [0,1]).

$$REM = 1 - |min(BWT, 0)| \tag{3.7}$$

i.e., *Remembering*, and (the originally positive) BWT, i.e., improvement over time, *Positive Backward Transfer*:

$$BWT^+ = max(BWT, 0) \tag{3.8}$$

Likewise, the FWT redefined to account for the dynamics of CL at each timestep is

$$FWT = \frac{\sum_{i=1}^{j-1}\sum_{j=1}^{N} R_{i,j}}{\frac{N(N-1)}{2}} \tag{3.9}$$

FWT accounts for the train-test accuracy entries $R_{i,j}$ above the principal diagonal of R, excluding it (see elements accounted in Table 3.4 in light gray). Forward transfer can occur when the model is able to perform *zero-shot* learning.

FWT and BWT can be interpreted as trans-task generalization capacity of an algorithm. It is important to note that FWT and BWT give no insight into the algorithms assets if not compared to another algorithm. It is not possible to easily disentangle the generalization performance from the similarity of tasks.

A Learning Curve Area (LCA) ($\in [0, 1]$) metric to quantify the learning speed by a CL strategy is proposed in [42]. It uses the b-shot performance (where b is the mini-batch number) after being trained for all the N tasks:

$$Z_b = \frac{1}{N} \sum_{i=1}^{N} a_{i,b,i} \tag{3.10}$$

where $a_{i,k,j} \in [0, 1]$ is the accuracy evaluated on the test set of task j after the model has been trained with the k-th mini-batch of task i. This amount is equivalent to previous accuracy matrix entry R_{ij} but at a lower granularity of a batch level. $a_{i,k,j}$ is used to define a forgetting measure $\in [-1, 1]$ that quantifies the drop in accuracy on previous tasks [41]. f_j^k is the forgetting on task j after the model is trained with all mini-batches up to task k:

$$f_j^k = \max_{l \in 1,..,k-1} a_{l,B_l,j} - a_{k,B_k,j} \tag{3.11}$$

where B_i is all mini-batches corresponding to training dataset of task k ($\mathcal{D}_k$).

LCA_β is the area of the convergence curve Z_b during training as a function of $b \in [0, \beta]$:

$$LCA_\beta = \frac{1}{\beta + 1} \int_0^\beta Z_b db = \frac{1}{\beta + 1} \sum_{b=0}^{\beta} Z_b \tag{3.12}$$

The interpretation of LCA is intuitive: an LCA_0 is the average 0-shot performance (FWT), and LCA_β is the area under the Z_b curve, which is high if the 0-shot performance is good and if the learner learns quickly. LCA aims at disambiguating the performance of models that may have the same Z_b or A_T, but very different LCA_β because despite both eventually obtaining the same final accuracy, one may learn much faster than the other.

While forgetting and knowledge transfer could be quantified and evaluated in various ways, as argued in [69, 91, 115], these may not suffice for a robust evaluation of CL strategies. For example, in order to better understand the different properties of each strategy in different conditions, especially for embedded systems and robotics, it would be interesting to keep track and unambiguously determine the amount of computation and memory resources exploited. In this context, the metrics proposed in [150] are extended in [59] to unify in a common evaluation framework different infrastructural and operational metrics. Other practical metrics included are Continual Memory Size (CMS) and Computational Efficiency (CE). We briefly describe them next.

The memory size of model h_i is quantified in terms of parameters θ at each task i, $Mem(\theta_i)$ and the eventual external memory to save data M_{Data_i}; with the idea that it should not grow too rapidly with respect to the size of the model that learned the first task, $Mem(\theta_1)$:

$$MS = min(1, \frac{\sum_{i=1}^{N} \frac{Mem(\theta_1)}{Mem(\theta_i) + M_{Data_i}}}{N}) \tag{3.13}$$

To compute the MS metric, it is important to note that M_{Data_i} may contain data to remember and evaluation data needed for hyper-parameters selection.

A metric that bounds the Computational efficiency (CE) by the number of operations for training set Tr_i is defined as:

$$CE = min(1, \frac{\sum_{i=1}^{N} \frac{Ops\uparrow\downarrow(Tr_i)\cdot\varepsilon}{1+Ops(Tr_i)}}{N}) \tag{3.14}$$

where $Ops(Tr_i)$ is the number of (mul-adds) operations needed to learn Tr_i, $Ops\uparrow\downarrow(Tr_i)$ are the operations required to do one forward and one backward (backprop) pass on Tr_i, and ε is a scaling factor (associated to the nr of epochs needed to learn Tr_i). Overall CL_{score} and $CL_{stability}$ metrics are also finally proposed [59] in order to aggregate different criteria to be maximized that allow to rank CL strategies.

In future evaluation scenarios, particularly in robotics, stability is another important property that should be evaluated since in many robotic tasks and safety-critical conditions, potential abrupt performance drifts would be a major concern when learning continuously. The metrics presented here can also be combined to assess higher-level capabilities. As an example, if we are to assess the *scalability* of a CL algorithm, one could use a weighted average of *MS*, and *CE*.

The metrics presented in a supervised classification context [59] can also be generalized with different performance measure P, instead of task accuracy $R_{i,j}$, and used in the same way in the metrics proposed previously. For example, it could be used for reinforcement and unsupervised learning. For instance, they can be extended to RL; the underlying performance metric is, instead of accuracy, the accumulated reward on test episodes. In general in RL, cumulative reward plots over time are common norm to evaluate policy learning algorithms. Extra performance metrics in RL tasks will very much depend on the task being assessed, the reward function, and other evaluation metrics that act as evaluation *proxies*, as it is common in semi/unsupervised learning settings.

The evaluation of generative models in any setting is challenging. Fréchet Inception Score (FID) [97] is a common metric that compares features from generated data and true data. Inception Score (IS) [209] has also been widely used as a proxy to evaluate the quality of generative models. It measures if the class of generated samples are varied by making use of a model trained on ImageNet. One shortcoming of these scores is that they may be maximized by over-fitting generative models. Another evaluation method is using generated data to train a classifier and evaluate its accuracy on a test set of true data [140]. The test accuracy, called Fitting Capacity (FC) gives a proxy on the quality of the generated data. Fitting Capacity and Fréchet Inception Score were used in a CL setting in [137, 135].
More methods for evaluating generative models are described and assessed more in depth in [25, 107]; however, they have never been used in a CL setting. In any case, the need for real data is mandatory in most evaluation schemes. In a CL setting, evaluating the generation of data from past tasks may need to violate the data availability assumption. The different metrics for generative models may then be useful tools for example for evaluating generative replay methods; however, they have to be manipulated carefully to be incorporated into the continual learning spirit.

For the remainder of this thesis, we are particularly focus on the final accuracy performance (eq. 3.5). Even if the other metrics are interesting to evaluate an algorithm, the most valuable results are the final performance. We discuss more this point in Chapter 8. Moreover, we mostly study disjoint settings and in such setting BWT learning and FWT learning cannot happen, so the corresponding metrics are not relevant. Finally, reporting the experiments computational costs

would be clearly interesting, but would require a higher number of experiments than what has been possible to achieve within this work.

3.7 Applications : Continual Learning for Robotics

In the previous section we listed and described the different existing types of strategies to tackle continual learning. In this section, we will present real use cases of CL with an emphasis on robotics applications. First, we present why continual learning is crucial for robotics, and then, the challenges that robotics face in CL tasks. Finally, we present concrete robotic applications with potential insights to draw from CL.

3.7.1 Opportunities for Continual Learning in Robotics

A robot is an agent that interacts with the real world. It means that it cannot go back in time to improve what it has learn in the past. These particularities of robotic platforms make them a natural playground for CL algorithms. Furthermore, robots suffer from several constraints in terms of power or memory, which CL intends to optimize, in the way it addresses learning problems. On the other hand, robots have rich information about their experiences. They are in control of their interaction with the environment, which may help them understanding the concept of causality, and extracting knowledge from different kinds of sensors (images, sound, depth...). This rich information helps machines to produce strong representations which are crucial for a well performing CL algorithm [136].

We could almost conclude that CL is born for robotics, and it may be true; however, today most of CL approaches are not robotics related and rather focus on experiments on image processing or simulated environments. The next section will present the challenges that make CL difficult to apply in robotic environments.

3.7.2 Challenges of Continual Learning in Robotics

Robotics Hardware

The first challenge to deal with when doing any experiment with robots is the hardware. Robots are known to be unstable and fragile. Robot failures are one of the main restrictions for researchers to propose new approaches on robotics tasks. They add unavoidable delay in any experiment and are expensive to fix. Moreover, if the failure is not hardware but software, since it is not possible to reset the state of the robot automatically, manual help is often needed, e.g., to put back the robot in his starting position or recover it from an irrecoverable state. Furthermore, most of the time building or buying a robot is itself quite costly. Once the robot is correctly working, one new problem arises, which is its autonomy in terms of energy. This aspect is also a main difficulty to deal with when experiments need to be set. It is difficult to program long experiments without manually recharging the robot and making sure that it will not stop by a lack of power supply or failure. Lastly, robots are embedded platforms and, consequently, have limited memory and computation resources, which should be carefully managed to avoid overflow.

The difficulties of using robots in experiments explain why there are so few approaches of continual learning with robots in the literature. In the next section, we will see how robotic environments challenge continual learning algorithms.

Data Sampling

When a robot needs to learn a task in a known or unknown environment, it must collect its own training data in the real world [262]. Data serves as the basis for environment exploration and comprehension. This problematic is exactly the same as the one met by RL algorithms [241]. In infants, a crucial component of lifelong learning is the ability to autonomously generate goals and explore the environment driven by intrinsic motivation [176, 35]. Self-supervised approaches [187, 142, 262, 224] also help to automatically explore environments. Curiosity [31] and self-supervision [60] allow to search for new experiences (or data) and build a base of knowledge useful to achieve current or future tasks via transfer learning [178]. As an example, manipulation tasks [117] such as grasping [187], reaching [189, 48], pushing buttons [138], throwing [236, 117] or stacking [48] objects (cubes, balls...) are common complex tasks built on comprehensive sets of experiments.

Data gathered in this way can then be used on the fly in an online learning process or stored for later processing. However, in order to improve learning algorithms the need for annotations or external help is crucial. In the next subsection we will describe the particular needs for annotations in robotics.

Data Labelling

As seen in previous section, gathering a varied set of raw data is already a difficult task. However, using it and understanding it is even more tedious. In this section, we detail different needs for labelling that autonomous agents such as robots need. First of all, to understand its environment, a robot will need to apprehend the objects that compose it. To do so, the robot will need at some point that an external expert assesses that the object representation learned is good. This is the first kind of label the robot will need, i.e., object labelling [50, 52]. Second, if we want the robot to perform a certain task, it will need to get information about the goals we gave it and also about what it should not do. This is generally done by a reward function that defines credit assignment [165], or it can also be defined internally by more abstract rules such as self-supervision [89, 232], intrinsic motivation or curiosity [176, 213] as in [75, 49, 52, 130]. Third, the robot should know when the task changes, and what task it should try to perform. This process consists of labelling the task; and the label is called the task identifier [150].

All these types of labels are not mandatory, but they drastically help and impact the learning process. The downside of labelling is that it is expensive and time consuming, which slows down the learning algorithms. To tackle those two problems, CL needs to find efficient solutions that can make the best out of the available labels for learning.

The specific fields that aim at answering these questions are few-shot learning [127, 70] and active learning [221]. The former tries to grasp a concept from very few data points. Active learning aims at identifying and selecting the most needed labels in order to maximize learning. By combining optimization procedures in learning from few instances and minimizing the needs for labels, the field of robotics could be more suitable for leveraging continual learning settings in the real world. Furthermore, efficiency in learning reduces the risks of forgetting and degrading memories.

Learning Algorithms Stability

In continual learning, algorithms face several learning experiences in a row. From each learning experience, some memory should be saved to later prevent for not forgetting. The stability of learning algorithms is then crucial: if only one learning experience fails, the whole process may

be corrupted. Moreover, if we respect the continual learning causality, we cannot go back one or several tasks earlier in time in order to fix an current problem. The corruption of one learning experience can lead to the corruption of memories and then to the model degradation when learning later tasks. The needs for robust mechanisms to validate or reject results of a learning algorithm is key to keep sane memories and weights; however, the instability of deep learning models must also be addressed to overcome this drawback. As an example, generative models are powerful tools for continual learning but their instability may make them unsuitable for this kind of setting [135]. Reinforcement learning algorithms are also known to be unstable and unpredictable, which is disastrous for continual learning.

3.7.3 Applications Fields

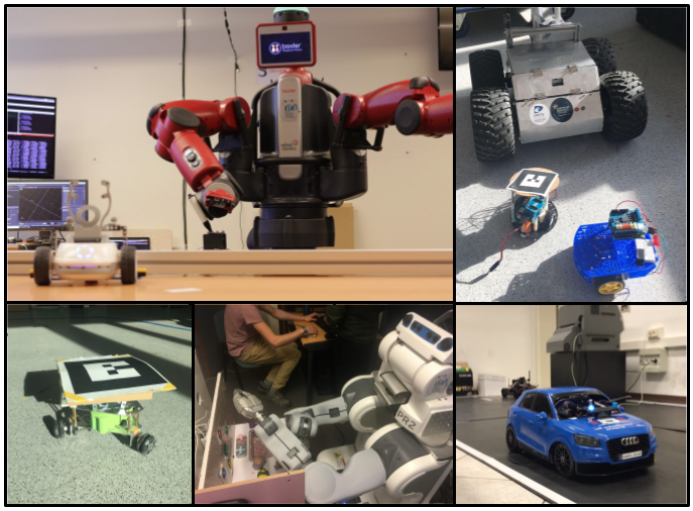


Figure 3.3: Sample tasks tested for unsupervised open-ended learning [189, 61] and continual learning settings [112] in a couple of robotics labs, among others, from the DREAM project.

Real-word applications of continual learning are virtually unlimited. In fact, any learning algorithm that needs to deal with the real world will face a non i.i.d. data stream. This as well happens for autonomous robots that learn new manipulation tasks, for exploration policies, as well as for autonomous vehicles that need to learn and adapt to new circumstances [24, 47, 104, 195]. Non-static settings are also faced by algorithms that learn how to predict trends based on data streams from internet user activities, e.g., among others, for advertisement or finance. This problem is likewise confronted when an already trained algorithms needs to acquire new knowledge without forgetting, e.g., recognize new classes for classification, anomaly detection, etc. However, in this section we focus on specific continual learning use cases on robotics.

Perception

While the world of perception is a multi-faceted topic at the very center of every application on autonomous sytems, the vast majority of CL algorithms in the literature are evaluated on object

recognition tasks. Most models, indeed, are evaluated on datasets including static or moving objects. This is motivated by the fact that before any further action or policy, an autonomous agent (or robot) needs to identify the different component of its environment. In the case of classification, the robot may be pre-trained from an initial dataset. However, in any environment the robot would probably need to learn new objects from the new domain, and new variants (different poses, lighting, aspect) of already learned objects should be leveraged to improve its recognition [152] capabilities. CL is crucial to deal with such dynamic scenarios. Initial progresses in this area have been proposed in [245, 180, 148, 33, 149]. Concrete Continual learning approaches to object segmentation can be found in [164, 164], and in object detection in [227].

Visual saliency for semantic segmentation and unsupervised object detection are other equally important applications in the context of perception which have been recently explored under continual learning and robotics settings [51]. RL-IAC (RL Intelligent Adaptive Curiosity), in particular, explores to learn visual saliency incrementally [52] with an articulated autonomous exploration technique based on curiosity to efficiently and continually learn a saliency model in a complex robotics environment tested in the real-world.

A classic problem in robotics within inherently continual learning settings are simultaneous localization and mapping (SLAM) [40] and navigation [245]. In [245], using a HERO-2000 mobile robot with a radar sensor a continual learning algorithm based on explanation-based neural network learning (EBNN) is proposed to perform room mapping and navigation. Action models in EBNN *explain* (in terms of previous experiences) and analyze observations to transfer task-independent (navigation) knowledge via predicting collisions and their prediction certainty.

Reinforcement Learning

In reinforcement learning the data distribution is dependent on the actions taken by the controlled agent. Therefore, since the actions taken are not random, data is not i.i.d. and the data distribution is not stationary. In the context of reinforcement learning similar techniques to those proposed in CL are often adopted in order to learn over a data distribution which is approximately stationary. An example of such techniques is the use of a external memory for rehearsal purposes, also know as *experience* or *memory replay* buffer [144, 212, 92].

The first challenge for RL is the extraction of representations to understand and compact what is relevant from the input data [136]. Continual learning of state representations for RL is intrinsically close to unsupervised learning or representation learning for classification; the methods used in both cases may then be very similar or worth leveraging across.

The second RL challenge is learning a policy to solve a specific task. The CL challenge of policy learning is different because it often should take into account both state and context. Context is usually handled with recurrent neural networks, and this kind of model is not yet been studied extensively in CL; one example is in [233]. Different robot manipulation tasks such as grasping and reaching objects that are used as benchmarks can be seen in Fig. 3.3 and, for instance, in state representation learning for robotics goal-based tasks [189, 112]. These two challenges face continual learning problems, to learn representations and to learn policies from non stationary data distributions. However, it is worth distinguishing among both problems because learning and transfer between tasks are different challenges. Two tasks may need similar representations with different policies, while similar policies may require dissimilar representations.

In the context of robotics, fewer RL approaches have been proposed than in video-games or simulation settings. In particular, this is due to the low data efficiency of RL algorithms [189].

We can still note several approaches that successfully tackle this problem, either in an end-to-end manner [111, 187], or by splitting the two challenges to address them separately, i.e., by first learning a state representation [136] and later performing policy learning [74, 253, 160, 3, 63, 108]. Nevertheless, a solution to this problem is to learn the policy in simulation and transfer it to deploy it in a real world robot [24, 208, 80, 112].

Model-based Learning

Smoothly moving and interacting with always different, unpredictable environments, while building a consistent model of the external world, is one of the holy grails of robotics. For many years, researchers in this area have tried to propose robust and general enough sensory-motor solutions to complex problems such as navigation or object grasping. However, as it appears to be also true for humans, there will always be an environment or situation in which our biased model of the world fails and would require a further adaptation

Online (inverse dynamics) learning has also been applied in robotics, but generally not using deep learning. In [203, 34], the inverse and semiparametric dynamics of an iCub humanoid robot is learned in an incremental manner. This means both parametric modelling (based on rigid body dynamics equations) and nonparametric modelling (using incremental kernel methods) are used. In [204] it is shown that derivative-free models outperform numerical differentiation schemes in online settings when applied to non parametric (e.g. Gaussian processes with kernel function) model structures.

In the pioneering work by [245], a model of both the external world and the robot itself is incrementally learned through reinforcement learning in complex navigation tasks on a real robot. However, incrementally and autonomously building a causal model of the external world, still remains a poorly explored topic in the context of robotics. Nevertheless, as it has been shown in recent RL literature, a model-based approach may be of fundamental importance in the real-world where millions of trials and errors are not always conceivable.

3.8 Conclusion

Several notions appear to be crucial to clearly describe learning algorithms in CL settings, to compare them fairly enough and to transfer them from simulation to real autonomous systems and robotics. First of all, identifying the exact problem we want to solve, and what are the existing constraints is mandatory. The framework we introduce in Section 3.4 should be an help in achieving the characterization of these settings. This formal step helps finding the proper approach to use and identifying similarities with other settings. Secondly, in the same spirit of defining what we want to learn, it is important to define the level of supervision we are able to give to our learning algorithm. For example, if we can give it the task label, or some kind of information about the structure of the input data stream (number of classes, type of data distribution, number of instances of each task, etc.). This point is also discussed in our proposed framework (Section 3.4). Finally, it is important to exactly clarify what is the expected performance of the algorithm. The set of metrics and benchmarks gathered in Section 3.6 should help defining and articulating the dimension of evaluation for important properties worth considering in the development of embodied agents that learn continually.

To summarize, in this chapter, we proposed a generalized framework to hold a variety of CL strategies and make easier the connection between machine learning and robotics in continual

learning settings. We reviewed the state of the art in continual learning and illustrated how to use the proposed framework to present various approaches. We also discussed benchmarks and evaluation techniques currently being used in continual learning algorithms. Machine learning and robotics are fields undergoing an aggressive development period. We believe that pushing them forward to find formalization solutions to facilitate transfer between both fields is critical in order to understand each other, and make them profit from each other's successes. Moreover, the example of robotics for continual learning is well representative of the challenge addressed for the autonomy of learning agents.

In the following chapters of this manuscript, we will refer to this chapter for state of the art description, continual learning scenarios, benchmarks, and evaluation metrics. But some content will also be summarized in the context of each chapter.

Chapter 4

Supervision Free Inference in Continual Learning

In the previous chapter, we presented extensively the continual learning research field and illustrate its application potential through the lens of robotics. In this chapter, we discuss the need for supervision during the inference in some continual learning approaches and the practical limitations following from this requirement.

4.1 Introduction

In continual learning, some approaches such as *dynamic architectures* need to know from which task a data point is coming from in order to perform an inference. Indeed, since they use different inference paths or different models for prediction they need the task label to choose which one to use. For regularization methods, it is often assumed that the task label is needed only at training time. However, in this chapter, we show that in class-incremental settings, the approach can not distinguish classes from different tasks. Therefore, the task label is necessary at test time.

The class-incremental settings we study is considered iid by parts. Each iid part is referred to as a *task* and the data distribution changes are signalled by a task label. Each task contains different classes. In continual learning, this setting is called a class-incremental or disjoint-task scenario, as introduced in Chapter 3. It consists of learning sets of classes incrementally. Each task is composed of new classes. As the training ends, the model should classify data from all classes correctly.

In this chapter, the setting considers the task label as provided for training but not for inference. Then, without task labels for inference, the model needs to both learn the discrimination of intra-task classes and the inter-task classes discrimination (i.e. distinctions between classes from different tasks). On the contrary, if the task label was available for inference, only the discrimination of intra-task classes needs to be learned. The discrimination upon different tasks is given by the task label. Learning without access to task labels at test time is then much more complex, since it needs to discriminate data that are not available at the same time in the data stream. We study in particular a widely used approach for continual learning: regularization. We show that in the classical setting of class-incremental tasks, this approach has theoretical limitations and can not be used alone. Indeed, it can not distinguish classes from different tasks.

The contributions of this chapter are:

- We prove theoretical shortcomings of regularization based approaches in class-incremental settings.

- We propose an existing alternative to circumvent this problem.

We believe this chapter presents important results for a better understanding of CL which will help practitioners to choose the appropriate approach for practical settings.

4.2 Background

Let's first summarize the information from Chapter 3 relevant to this chapter. In continual learning, algorithms protect knowledge from catastrophic forgetting by saving them into a memory. The memory should be able to incorporate new knowledge and protect them from modification.

In continual learning, we distinguish four types of memorization approaches:

- **Dynamic architecture:** The neural networks create new weights automatically that will learn new tasks. Trained weights are frozen to protect memories [206, 71, 143]. In this case, the memory is composed of the old weights that are not modified anymore.

- **Rehearsal:** In order to maintain knowledge from past learning experiences, the algorithms save a subset of training data as memory [193, 172, 5, 18, 112, 264, 100, 32, 248].

- **Generative Replay:** Instead of saving samples, this method learns generative models that will produce artificial samples as memory of past learning experiences [226, 135, 263, 137].

- **Regularization:** Regularization defines a loss that will constrain weight updates to retain knowledge from previous tasks [121, 269, 155], or distill knowledge [99] from old models to a new one to remember past learning experiences [143, 217].

As presented in Section 3.5.5, many approaches use combinations of these families to allow better memorization. The effectiveness of these approaches is related to the use of the task label defined in Section 3.3. The task label t is an abstract representation built to help continual algorithms to learn. It is designed to give information about the current task and notify if the task changes.t is typically a simple integer indexing the tasks in the learning curricula. The different use cases of task labels are described in Section 3.4.3. Families of approaches have different dependencies to the task label. For example, *dynamic architecture* is an effective approach but it needs the task label at test time for inference. Unfortunately, this necessity of supervision at test time is not desirable in most continual learning settings. Rehearsal and Generative Replay methods generally need the task label for learning but not for inference. As mentioned in the chapter's introduction, for regularization methods, it is often assumed that the task label is needed only at training time. To challenge this belief, we study the case of class-incremental settings. We would like to demonstrate that regularization methods can not distinguish classes from different tasks and therefore, that the task label is also necessary at test time to expect a good prediction.

4.3 Regularization Approach

In the class incremental setting presented in chapter's introduction, we would like to demonstrate that regularization does not help learn the discrimination between tasks. For example, if a first task is to discriminate white cats vs black cats and the second is the same with dogs, a regularization based method does not provide the learning criteria to learn features to distinguish white dogs from white cats.

4.3.1 Formalism

Let's first introduce a more detailed formalization of regularization. We assume that the data stream is composed of N disjoint tasks learned sequentially one by one (with $N >= 2$). Task t is noted T_t and $\mathbb{D}_t$ is the associated dataset. The task label t is a simple integer indicating the task index. We refer to the full sequence of tasks as the continuum, noted C_N. The dataset combining all data until task t is noted $\mathbb{C}_t$. While learning task T_t, the algorithm has access to data from $\mathbb{D}_t$ only.

We study a disjoint set of classification tasks where classes of each task only appear in this task and never again. We assume at least two classes per task (otherwise a classifier cannot learn).

Let f be a function parametrized by $\boldsymbol{\theta}$ that implements the neural network's model. At each task t the model learns an optimal set of parameters $\boldsymbol{\theta}_t^*$ optimizing the task loss $\ell_{\mathbb{D}_t}(\cdot)$. Since we are in a continual learning setting, $\boldsymbol{\theta}_t^*$ should also be an optima for all tasks $T_{t'}$, $\forall t' \in [\![0, t]\!]$.

We consider the class-incremental setting with no test label. It means that an optima $\boldsymbol{\theta}_1^*$ for T_1 is a set of parameters which at test time will, for any data point x from $\mathbb{D}_0 \cup \mathbb{D}_1$, classify correctly without knowing if x comes from T_0 or T_1. Therefore, in our continual learning setting, the loss to optimize when learning a given task t is augmented with a remembering loss:

$$\ell_{\mathbb{C}_t}(f(x;\boldsymbol{\theta}), y) = \ell_{\mathbb{D}_t}(f(x;\boldsymbol{\theta}), y) + \lambda\Omega(C_{t-1}) \tag{4.1}$$

where $\ell_{\mathbb{C}_t}(.)$ is the continual loss, $\ell_{\mathbb{D}_t}(.)$ is the current task loss, $\Omega(C_{t-1})$ is the remembering loss with C_{t-1} represents past tasks, λ is the importance parameter.

4.3.2 Problem

In continual learning, the regularization approach is to define $\Omega(\cdot)$ as a regularization term to maintain knowledge from C_{t-1} in the parameters $\boldsymbol{\theta}$ such as while learning a new task T_t, $f(x;\boldsymbol{\theta}_{t-1}^*) \approx f(x;\boldsymbol{\theta})$, $\forall x \in \mathbb{C}_{t-1}$. In other words, it aims at keeping $\ell_{\mathbb{C}_{t-1}}(f(x;\boldsymbol{\theta}), y)$ low $\forall x \in \mathbb{C}_{t-1}$ while learning T_t.

The regularization term Ω_{t-1} act as a memory of $\boldsymbol{\theta}_{t-1}^*$. This memory term depends on the learned parameters $\boldsymbol{\theta}_{t-1}^*$, on $\ell_{\mathbb{C}_{t-1}}$ the loss computed on T_{t-1} and the current parameters $\boldsymbol{\theta}$. Ω_{t-1} memorizes the optimal state of the model at T_{t-1} and generally the importance of each parameter with regard to the loss $\ell_{\mathbb{C}_{t-1}}$. We note $\Omega_{\mathbb{C}_{t-1}}$ the regularization term memorizing optimal parameters for all past tasks.

When learning the task T_t, the loss to optimize is then:

$$\ell_{\mathbb{C}_t}(f(x;\boldsymbol{\theta}), y) = \ell_{\mathbb{D}_t}(f(x;\boldsymbol{\theta}), y) + \lambda\Omega_{\mathbb{C}_{t-1}}(\boldsymbol{\theta}_{t-1}^*, \ell_{\mathbb{C}_{t-1}}, \boldsymbol{\theta}) \tag{4.2}$$

Eq. 4.2 is similar to eq. 4.1 but in this case the function $\Omega(\cdot)$ is a regularization term depending on past optimal parameters $\boldsymbol{\theta}_{t-1}^*$, loss on previous tasks $\ell_{\mathbb{C}_{t-1}}$ and the vector of current model

parameters $\boldsymbol{\theta}$ only. It could be for example a matrix pondering weights importance in previous tasks [121, 201, 269].

4.3.3 Regularization methods

To illustrate the previous section, we present several well known regularization methods in our formalism.

Elastic Weight Consolidation

(EWC) [121] is one of the most famous regularization approaches for continual learning. The loss augmented with a regularization term is at task t:

$$\ell_{\mathbb{C}_t}(\boldsymbol{\theta}) = \ell_{\mathbb{D}_t}(f(x;\boldsymbol{\theta}), y) + \frac{\lambda}{2} * F_{t-1}(\boldsymbol{\theta}^*_{t-1} - \boldsymbol{\theta})^2 \tag{4.3}$$

We can then by identification, extract our function $\Omega_t(\boldsymbol{\theta}^*, \ell_D, \boldsymbol{\theta})$

$$\Omega_t(\boldsymbol{\theta}^*, \ell_{\mathbb{C}_{t-1}}, \boldsymbol{\theta}) = \frac{1}{2} * F_{t-1}(\boldsymbol{\theta}^*_{t-1} - \boldsymbol{\theta})^2 \tag{4.4}$$

F_t is a tensor of size $card(\boldsymbol{\theta})^2$, specific to task t, characterizing the importance of each parameter θ_k. F_t is computed at the end of each task and will protect important parameters to learn without forgetting. In EWC, the F_t tensor is implemented as a diagonal approximation of the Fisher Information Matrix:

$$F_t = \mathbb{E}_{(x,y)\in\mathbb{D}_t}\left[\left(\frac{\partial log\ p(\hat{y})}{\partial \boldsymbol{\theta}}\right)^2\right] \tag{4.5}$$

where $\hat{y} \sim P(f(x;\boldsymbol{\theta}))$. The diagonal approximation allows to save only $card(\boldsymbol{\theta})$ values in F_t.

K-FAC Fisher approximation

The K-FAC Fisher approximation [201] is very similar to EWC but approximates the Fisher matrices with a Kronecker factorization (K-FAC) [158] to improve the expressiveness of the posterior over the diagonal approximation. However, the Kronecker factorization saves more values than the diagonal approximation.

Incremental Moment Matching

Incremental Moment Matching (IMM) [133] proposes two regularization approaches for continual learning which differ in the computation of the mean $\theta_{0:t}$ and the variance $\sigma_{0:t}$ of the parameters on all tasks.

The idea is to regularize parameters such that the moments of their posterior distributions are matched in an incremental way. It means that each parameter is approximated as a normal distribution and their mean or standard deviation should match from one task to another. This regularization, on the parameters' low-order moments, helps to protect the model from forgetting.

- Mean based Incremental Moment Matching (mean-IMM)

$$\theta_{0:t} = \sum_{i=0}^{t} \alpha_i \boldsymbol{\theta}_i^* \quad \text{and} \quad \sigma_{0:t} = \sum_{i=0}^{t} \alpha_i (\sigma_i + (\boldsymbol{\theta}_i^* - \theta_{0:t})^2) \tag{4.6}$$

  α_i are importance hyper-parameters to balance past task weight into the loss function. They sum up to one.

- Mode based Incremental Moment Matching (mode-IMM)

$$\theta_{0:t} = \sigma_{0:t} \cdot \sum_{i=0}^{t} (\alpha_i \sigma_i^{-1} \boldsymbol{\theta}_i^*) \quad \text{and} \quad \sigma_{0:t} = (\sum_{i=0}^{t} \alpha_i \sigma_i^{-1})^{-1} \tag{4.7}$$

  σ_i is computed as the Fisher matrix (eq. 4.5) at task i.

Then at task t, with $\theta_{0:t-1}$ and $\sigma_{0:t-1}$ we can compute:

$$\Omega_t(\boldsymbol{\theta}^*, \ell_{\mathbb{C}_{t-1}}, \boldsymbol{\theta}) = \frac{1}{2} \sigma_{0:t-1} (\theta_{0:t-1} - \boldsymbol{\theta})^2 \tag{4.8}$$

Synaptic Intelligence

The original idea of Synaptic Intelligence approach (SI) [269] is to imitate synapse biological activity. Therefore, each synapse accumulates task relevant information over time, and exploits this information to rapidly store new memories without forgetting old ones. In this approach, we can identify Ω_t as:

$$\Omega_t(\boldsymbol{\theta}^*, \ell_{\mathbb{C}_{t-1}}, \boldsymbol{\theta}) = M_t (\boldsymbol{\theta}_{t-1}^* - \boldsymbol{\theta})^2 \tag{4.9}$$

M_t is a tensor of size $card(\boldsymbol{\theta})$ specific to task t characterizing the importance of each parameter θ_k over the all past tasks such as:

$$M_t = \sum_{0<i<t} \frac{m_i}{\Delta_i^2 + \xi} \tag{4.10}$$

M_t is the sum over m_i which characterizes the importance of each parameter on task i, with $\Delta_i = \boldsymbol{\theta}_i^* - \boldsymbol{\theta}_{i-1}^*$. ξ is a supplementary parameter to avoid null discriminator.

$$m_i = \int_{T_{i-1}}^{T_i} \nabla_{\boldsymbol{\theta}} \delta_{\boldsymbol{\theta}}(t) dt \tag{4.11}$$

With $\delta_{\boldsymbol{\theta}}(t)$ the parameter update at time step t.

4.4 Propositions

In this section, we present the proposition concerning the shortcomings of regularization methods in class-incremental settings. We also present preliminary definitions and lemmas to prepare for the proposition and we illustrate the proposition with practical examples.

4.4.1 Preliminary Definition / Lemma

Definition 17. ***Linear separability***
Let S and S' be two sets of points in an n-dimensional Euclidean space. S and S' are linearly separable if there exists $n+1$ real numbers $\omega_1, \omega_2, ..., \omega_n, k$ such that $\forall x \in S$, $\sum_{i=1}^{n} \omega_i x_i > k$ and $\forall x \in S'$, $\sum_{i=1}^{n} \omega_i x_i < k$

where x_i the i-th component of x. This means that two classes are linearly separable in an embedded space if there exists a hyper-plane separating both classes of data points.

It can be written, $\forall x \in S$ and $\forall x' \in S'$.

$$(\boldsymbol{q} \cdot \boldsymbol{x} + q_0) \cdot (\boldsymbol{q} \cdot \boldsymbol{x'} + q_0) < 0 \tag{4.12}$$

with $\boldsymbol{q} = [\omega_1, \omega_2, ..., \omega_]$ and $q_0 = -k$ respectively the normal vector

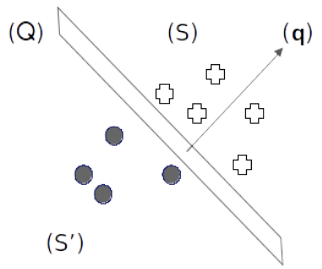


Figure 4.1: Illustration of a decision boundary learned between two sets of points.

and position vector of a hyper-plane $\mathcal{Q}$.

In the case of learning a binary classification with linear model, the model is the best hyper-plane separating two dataset as in Fig. 4.1. As soon as eq. 4.12 can be solved, then it is possible to define a function $f(x, \theta)$ and a loss $\ell(.)$ to learn a hyper-plane that will separate S and S' perfectly.

Definition 18. ***Interferences***
In machine learning, interferences are conflicts between two (or more) objective functions leading to prediction errors.

As such, optimizing one of the objective function increases the error on the other one. In continual learning, interferences happen often after a drift in the data distribution. The loss on previous data is increased with the optimization of the loss for the new data leading to interferences and catastrophic forgetting.

We would like to present the following general lemma, as a preparatory step for the later proposition presented in this chapter:

Lemma 4.4.1. *$\forall (S, S')$ bounded set of discrete points in R^n and linearly separable by a hyper-plane $\mathcal{Q}$. For any algorithm, it is impossible to assess $\mathcal{Q}$ as a separation hyper-plane without access to S' set.*

Proof. Let S and S' be two bounded and linearly separable set of discrete points in R^n. Let $\mathcal{Q}$ be a potential linear separation between S and S'. The hyper-plane $\mathcal{Q}$ can not be assessed as a linear

separation between S and S' if there exists at least one hyper-plane indistinguishable from $\mathcal{Q}$ and which is not a separation boundary between S and S'.

Let $\mathcal{P}$ be a hyper-plane, defined as a normal vector $\boldsymbol{p}$ and position vector p_0. $\mathcal{P}$ is a separation boundary between S and S' if all the point of S are on one side of $\mathcal{P}$ and all point of S' are on the other side. It can be formalized as follows:

$\forall \boldsymbol{x} \in S$ & $\forall \boldsymbol{x'} \in S'$:

$$(\boldsymbol{p} \cdot \boldsymbol{x} + p_0) \cdot (\boldsymbol{p} \cdot \boldsymbol{x'} + p_0) < 0 \tag{4.13}$$

where $< \cdot >$ is the scalar product.

Without the access to S', eq. 4.13 can not be evaluated to verify that S and S' are each entirely on different side of the $\mathcal{P}$.

However, we can evaluate if all the point of S are on the same side of $\mathcal{P}$. By definition if all the point sof S are above $\mathcal{P}$ then:

$\forall \boldsymbol{x} \in S$

$$(\boldsymbol{p} \cdot \boldsymbol{x} + p_0) > 0 \tag{4.14}$$

If all the point are under $\mathcal{P}$ then:

$$(\boldsymbol{p} \cdot \boldsymbol{x} + p_0) < 0 \tag{4.15}$$

And if neither eq. 4.14 nor eq. 4.15 are verified then all the points of S are not on the same side of $\mathcal{P}$. Finally, we can merge both 4.14 and eq. 4.15 and verify only:

$\forall \boldsymbol{x} \in S$

$$sign(\boldsymbol{p} \cdot \boldsymbol{x} + p_0) = constant \tag{4.16}$$

where $sign(.)$ is the function which returns the sign of any real value.

The lemma 4.4.1 is proven if $\exists\ \mathcal{P}$ such as eq. 4.16 is true but not eq. 4.13, because $\mathcal{P}$ would not be a linear separation of S and S' and would not be distinguishable from $\mathcal{Q}$ without access to S'.

Now, we will build an hyper-plan P that respect eq. 4.16 and not eq. 4.13. We know that S is bounded, then it has both upper and lower bounds in all the direction of R^n. If eq. 4.16 is respected, then $\mathcal{Q}$ is a bound of S in the direction of its normal vector $\boldsymbol{q}$. If we move $\mathcal{Q}$ along the direction of $\boldsymbol{q}$ (i.e. if we change the position vector q_0), we can find at least one other plane $\mathcal{P}$ respecting eq. 4.16: the opposing bound of S along the direction $\boldsymbol{q}$.

Since, $\mathcal{P}$ and $\mathcal{Q}$ are two opposing bounds of S in the same direction $\boldsymbol{q}$, then:

$\forall x \in S$

$$sign(\boldsymbol{p} \cdot \boldsymbol{x} + p_0) \neq sign(\boldsymbol{q} \cdot \boldsymbol{x} + q_0) \tag{4.17}$$

If $\mathcal{Q}$ is a lowerbound of S in the direction $\boldsymbol{q}$ and an upperbound of S' in the same direction then, a lowerbound of S' in the direction $\boldsymbol{q}$ is a lowerbound of S in the same direction and an upperbound of S in the direction $\boldsymbol{q}$ is an upperbound of S' in the same direction. (We leave the demonstration to the reader).

Therefore, $\mathcal{Q}$ and $\mathcal{P}$ are both upperbounds or both lowerbounds of S' in the direction of $\boldsymbol{q}$:

$\forall x' \in S'$:

$$sign(\boldsymbol{p} \cdot \boldsymbol{x'} + p_0) = sign(\boldsymbol{q} \cdot \boldsymbol{x'} + q_0) \tag{4.18}$$

Then with 4.17 and eq. 4.18:

$$(\boldsymbol{p} \cdot \boldsymbol{x} + b) \cdot (\boldsymbol{p} \cdot \boldsymbol{x'} + b) > 0 \tag{4.19}$$

Consequently, from eq 4.16 and eq 4.19, $\exists$ a hyper-plane $\mathcal{P}$ which respects eq. 4.16 and not eq 4.13, $\mathcal{P}$ is indistinguishable from $\mathcal{Q}$ and is not a separation boundary between S and S'.

□

Let's summarize this demonstration in a more insightful way. For any bounded set of points S, there is an infinite number of linearly separable set of points. Thus, there exists an infinite number of potential separating hyper-planes. If the second set of points S' is not known, then it is not possible to choose among the infinite number of potential separating hyper-plane which one is a good one. And even if one is chosen, there is no way to tell if it is better or not than another.

In the context of machine learning, this demonstration says that without an assessment criterion for a classification problem, it is not possible to learn a viable solution. Hence, we can not optimize the parameters. For binary classification, the lemma 4.4.1 can be interpreted as: "The decision boundary between two classes can not be assessed nor learned if there is no access to data from both simultaneously".

After presenting the lemma about finding linear separation between sets of points, we would like to add another one concerning finding a projection making two mixed sets of points linearly separable. This lemma will also help in the proposition proof presented in this chapter.

Lemma 4.4.2. *$\forall(S, S')$ two bounded datasets not linearly separable. For any algorithm, it is impossible to assess a function $g(.)$ as a projection of S and S' into a space were they are linearly separable without access to S' set.*

Proof. $g(.)$ is a projection of S and S' into a space where they are linearly separable means:

$\forall \boldsymbol{x} \in S$ & $\forall \boldsymbol{x'} \in S'$, then $g(x)$ and $g(x')$ respect eq. 4.13.

Without access to S' this condition can not be verified. However, we can verify eq. 4.16 with $g(x)$.

The lemma 3.4 is proven if $\forall \boldsymbol{x} \in S$ & $\forall \boldsymbol{x'} \in S'$, $\exists$ a projection f, that respect eq. 4.16 with $f(x)$ but not eq. 4.13 with $f(x)$ and $f(x')$, because then f and g are indistinguishable without access to S'.

Let f be the identity function, $\forall z \in \mathbb{R}\ f(z) = z$. We define S_f and S'_f, the set of point S and S' after projection by f. Since f is the identity function, S and S' are respectively identical to S_f and S'_f. Since S is bounded, S_f is also bounded. Hence there exists a hyper-plane $\mathcal{P}$ that verify eq. 4.16 with $f(x)$ $\forall x \in S$. By hypothesis, S and S' are not linearly separable so S_f and S'_f is also not linearly separable. Then $\exists!$ hyper-plane $\mathcal{P}$ which respect eq. 4.13 with $f(x)$ and $f(x')$.

Thus, f exists and therefore it is impossible to assess any function as a projection of S and S' into a space were they are linearly separable without S' set.

□

In a more insightful way, for any bounded set of points, there is an infinite number of projections of the initial set of point in a space where it could be linearly separable from another set of points. Then, if you don't know the second set of points S' you can not choose among the infinite number of potential projections which one is a good one. And if you ever choose one, you have no way to tell if it is better or not than another.

In the context of binary classification, the previous lemma can be interpreted as: "Two classes representation cannot be disentangled if there is no access to data from both simultaneously".

In those lemma, the concept of "not having access to" a certain dataset can both be applicable to not being able to sample data point from the distributions and to not have a model of the dataset. It can be generalized to not having access to any representative data distribution of a dataset.

4.4.2 Shortcomings in class-incremental tasks

We now prove that in incremental-class tasks, it is not possible to discriminate classes from different tasks using only a regularization based memory. The main point is that, to correctly learn to discriminate classes over different tasks the model needs access to both data distributions simultaneously.

In regularization methods, the memory only characterizes the model and the important parameters as explained in Section 4.3.2. This memorization gives insight on some past data characteristics but it is not a model of their distributions. If we take the cat vs dog example, a model that needs to discriminate white cats from black cats will learn to discriminate black features from white features. This "knowledge" can be saved in Ω but Ω will not save the full characteristics of a cat because the model never has to learn it. We bring then the following proposition:

Proposition 4.4.3. *While learning a sequence of disjoint classification tasks, if the memory Ω of the past tasks is only dependent on trained weights and learning criterion of previous task and does not model the past distribution, it is not possible for deep neural networks to learn new tasks without interference.*

Proof. The proof is organized in the following way: first, we present material necessary for the demonstration, then in a second part, we demonstrate that at any moment the classification task can be reduced to a binary classification task and in a third part we will show that we can not learn to solve this binary classification correctly.

First part

In the context of learning with a deep neural network, we can decompose the model into a non-linear feature extractor $g(\cdot)$ and an output layer to predict a class $y = \text{argmax}(\text{softmax}(A \cdot g(x) + b))$. With A and b, respectively the matrix of projection and the bias of the linear layer. softmax(.) is the output function that for a given class i in a logits output z gives $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=0}^{N-1} e^{z_j}}$. The softmax(.) function does not change the argmax result and only regularize the output values and the gradient for later back propagation. We can thus remove it for our demonstration purposes.

The non-linear projection $g(.)$ should, therefore, disentangle classes and the linear output layer learns to predict the good class. The output layer allows for all classes i to learn hyperplanes $A[:, i]$ with bias $b[i]$ such as: $\forall i \in [\![1, N]\!]$

$$\forall (\boldsymbol{x}, y) \in \mathbb{C}_t, \underset{i}{\text{argmax}}(A[:, i]h + b[i]) = y \tag{4.20}$$

with $h = g(x)$.

Second part

For the sake of the demonstration, we would like to reduce the multi-classes classification problem into a binary classification problem. Hence, we can artificially split classes into two groups: classes from the past $\mathbb{Y}_{C_{t-1}}$ and current classes $\mathbb{Y}_{T_t}$.

We can then $\forall(\boldsymbol{x}, y) \in \mathbb{C}_t$ compute which class $\hat{y}_{C_{t-1}}$ upon the past classes $\mathbb{Y}_{C_{t-1}}$ is the most probable and compute which class $\hat{y}_{T_t}$ upon the current classes $\mathbb{Y}_{T_t}$ is the most probable.

$$\hat{y}_{C_{t-1}} = \underset{i \in \mathbb{Y}_{C_{t-1}}}{\text{argmax}}(A[:, i]h + b[i]) \quad \text{and} \quad \hat{y}_{T_t} = \underset{i \in \mathbb{Y}_{T_t}}{\text{argmax}}(A[:, i]h + b[i]) \tag{4.21}$$

Hence, the equation 4.20 can be rewritten into a binary operation:

$$\forall(\boldsymbol{x}, y) \in \mathbb{C}_t, \quad \underset{i \in \{\hat{y}_{C_{t-1}}, \hat{y}_{T_t}\}}{\text{argmax}} (A[:, i]h + b[i]) = y \tag{4.22}$$

$$\begin{aligned} y &= \text{argmax}(A[:, \hat{y}_{C_{t-1}}] \cdot h + b[\hat{y}_{C_{t-1}}] \ , \ A[:, \hat{y}_{T_t}] \cdot h + b[\hat{y}_{T_t}]) \\ &= \text{argmax}(0, (A[:, \hat{y}_{T_t}] - A[:, \hat{y}_{C_{t-1}}]) \cdot h + b[\hat{y}_{T_t}] - b[\hat{y}_{C_{t-1}}]) \end{aligned} \tag{4.23}$$

Equation 4.23 can directly be rewritten into the linear separability equation from definition 17. To make a proper decision, we should have $\forall(x, y) \in \mathbb{C}_t$, with $g(x) = h$ and $y = \hat{y}_{C_{t-1}}$ and $\forall(x', y') \in \mathbb{D}_t$, with $g(x') = h'$ and $y' = \hat{y}_{T_t}$.

$$(\boldsymbol{q} \cdot \boldsymbol{h} + q_0) \cdot (\boldsymbol{q} \cdot \boldsymbol{h'} + q_0) < 0 \tag{4.24}$$

Then, by identification, the classes $\hat{y}_{C_{t-1}}$ and $\hat{y}_{T_t}$ need to be separated by the hyperplane $\mathcal{Q}$ defined by a normal vector $\boldsymbol{q} = A[:, \hat{y}_{T_t}] - A[:, \hat{y}_{C_{t-1}}]$ and a position vector $q_0 = -(b[\hat{y}_{T_t}] - b[\hat{y}_{C_{t-1}}])$.

This binary classification description highlight that it is essential to be able to discriminate any class $\hat{y}_{C_{t-1}}$ from the past from any class $\hat{y}_{T_t}$ from the present for accurate predictions.

Third part

We will now, to prove proposition 4.4.3, show that we cannot learn the hyperplane $\mathcal{Q}$ from eq. 4.24.

To learn new tasks T_t for $0 < t < N$, there are two different cases: *first* $g(\cdot)$ is already a good projection for C_t tasks, i.e. classes are already disentangled in the embedded space. We assume that if classes are already disentangled, only the output layer has to be trained to solve C_t tasks. *Secondly*, $g(\cdot)$ needs to be adapted, i.e. classes are not yet disentangled in the embedded space and new features need to be learned by $g(\cdot)$ to fix it. We refer as features, intrinsic characteristics of data that a model needs to detect to distinguish a class from another. We will show that it is not possible to learn to discriminate correctly the classes $\hat{y}_{C_{t-1}}$ from $\hat{y}_{T_t}$ from previous part.

First case: Classes are disentangled

Since we are in a regularization setting, at task T_t, we have access to Ω_{t-1} which contains classification information from previous tasks (C_{t-1} tasks). However, by hypothesis, Ω_{t-1} does not model the data distribution from C_{t-1} and therefore it does not model data distribution from C_{t-1} classes.

Following from the second part of the proof, $\forall x \in C_t$ tasks, to make an accurate prediction, we need the right hyperplane $\mathcal{Q}$ that distinguish the most probable class from C_{t-1}, $\hat{y}_{C_{t-1}}$ and the most probable class from T_t, $\hat{y}_{T_t}$.

$\hat{y}_{C_{t-1}}$ and $\hat{y}_{T_t}$ classes images are a bounded set of points and $\hat{y}_{C_{t-1}}$ points are, by definition, not accessible, consequently following lemma 4.4.1, it is impossible to assess a boundary between $\hat{y}_{T_t}$ and $\hat{y}_{C_{t-1}}$ even if by hypothesis this boundary exists. Therefore, we can not learn the hyperplane that discriminate $\hat{y}_{C_{t-1}}$ from $\hat{y}_{T_t}$ and ensure an accurate prediction.

Second case: $g(\cdot)$ needs to be updated with new features.

Let δ_{t-1} be the set features already learned by $g_{t-1}(\cdot)$ the feature extractor from previous task.

Ω_{t-1} should keep δ_{t-1} unchanged while learning T_t. The goal is to make $\hat{y}_{C_{t-1}}$ and $\hat{y}_{T_t}$ linearly separable $\forall x \in \mathbb{C}_t$. Then, either δ_{t-1} already solve the problem and we are in *first case*, or a new set of features δ_t needs to be learned while learning T_t. In the second case, the set δ_t contains features to solve T_t, but features $\delta_{t-1:t}$ that distinguish classes from T_{t-1} to classes from T_t should also be learned. Then two cases raise, $\delta_{t-1:t} \not\subset \delta_t$ or $\delta_{t-1:t} \subset \delta_t$.

- if $\delta_{t-1:t} \not\subset \delta_t$, then supplementary features $\delta_{t-1:t}$ need to be learned. $\hat{y}_{C_{t-1}}$ and $\hat{y}_{T_t}$ classes images are a bounded set of points not linearly separable and since Ω_{t-1} does not give access to C_{t-1} data points, from lemma 4.4.2 we can not assess a projection that put images from $\hat{y}_{T_t}$ and $\hat{y}_{C_{t-1}}$ into a linearly separable space, i.e. we can not learn the set of features $\delta_{t-1:t}$ to discriminate $\hat{y}_{C_{t-1}}$ images from $\hat{y}_{T_t}$ images and solve the continual problem.
- $\delta_{t-1:t} \subset \delta_t$ is possible, however, since data from C_{t-1} are not available anymore, there is no way to project them in the new latent space with δ_t features. Therefore, without access of classes from both C_{t-1} and T_t tasks at time t we can not identify $\delta_{t-1:t}$ features which are in δ_t features. It is also impossible to know if $\delta_{t-1:t} \subset \delta_t$. In other words, this case is not detectable and even if detected the features $\delta_{t-1:t}$ can not be used without data from T_{t-1} (which is by definition prohibited).

In these two cases, there will be in any way conflict between losses leading to interference in the decision boundaries either because classes are not linearly separable or because a separation hyperplane cannot be found. In other words, the regularization methods can not discriminate classes from different tasks and they are then not suited to class-incremental settings.

□

We can note that proposition 4.4.3, still holds if tasks are only partially disjoint, i.e. only some classes appear only once in the continual curriculum.

Indeed, in partially disjoint settings, several classes pairs are never in the same task. If we define two set of disjoint classes Y and Y', that will never be in the same task, the demonstration of proposition 4.4.3 can be applied on Y and Y'. Then, classes Y and Y' will suffer from interference showing a shortcoming of regularization methods for this case too.

Therefore, if there is a class-incremental setting hidden into another setting, the regularization approach will not be able to solve it perfectly either. We could note that in many applications there are latent class-incremental problem to address in the learning curriculum. We mention some applications in Section 4.6.1.

A simple trick used in some regularization approaches to compensate their shortcomings is to use the task label for inferences, it gives a simple way to distinguish tasks from each other. However, it assumes the algorithms rely on a supervision signal for inferences. In the section 4.5, we show that regularization shortcoming is easily highlighted with simple experiments.

4.4.3 Practical examples

To illustrate the proposition from Section 4.4.2, we present two insightful examples of regularization limitations and

The Task Separability Problem

In the first case of proposition 4.4.3 proof, we already have a perfect feature extractor. Classes are already linearly separable and only the output layer needs to be learned continually.

If we have only two classes in the first task, the model will learn one hyper-plane $\mathcal{Q}_0$ separating the instances of these two classes (See Figure 4.2). For the second task, we have two new classes and a regularization protecting $\mathcal{Q}_0$. Then, we can learn a hyper-plane $\mathcal{Q}_1$ that separates our two new classes. In the end, we have learned the hyper-planes $\mathcal{Q}_0$ and $\mathcal{Q}_1$ to distinguish classes from T_0 and classes from T_1. But none of those hyper-planes helps to discriminate T_0 classes from T_1 classes, as illustrated Figure 4.2. This will lead to error in the neural networks predictions.

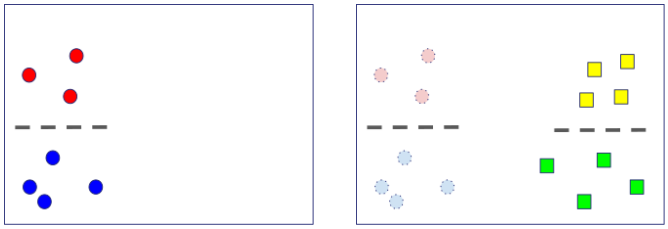


Figure 4.2: Simple case of continual learning classification in a multi-task setting. Left, the task T0: learning a hyper-plane splitting two classes (red and blue dots). Right, the task T1: learning a line splitting two classes (yellow and green squares) while remembering T_0 models without remembering T_0 data (pale red and blue dots).

The Latent Features Problem

In the second case of Proposition 4.4.3 proof, the feature extractor needs to be updated to learn new features extractors.

If we have only two classes in the first task, the model will learn to separate classes instances into two groups with the features extractor g_0 and one hyper-plan $\mathcal{Q}_0$ separating the two classes instances (See Figure 4.3).

For the second task, we have two new classes and a regularization protecting $\mathcal{Q}_0$ and g_0. Then, we can learn a features extractor g_1 to disentangle new class instances in the latent space and a hyper-plane $\mathcal{Q}_1$ that separates them. In the end, we can disentangle classes from T_0 and classes from T_1 and we have two hyper-planes $\mathcal{Q}_0$ and $\mathcal{Q}_1$ to distinguish classes from T_0 and classes from T_1. But we can not disentangle T_0 classes from T_1 classes and none of the learned hyper-planes helps to discriminate T_0 classes from T_1 classes (See Fig. 4.4). It leads to errors in the neural network predictions. At test time, it will not be possible for the model to discriminate between classes correctly.

However, with the task label for inference, we could potentially perfectly use g_0, g_1, $\mathcal{Q}_0$ and $\mathcal{Q}_1$ to make good predictions. Nevertheless, assuming that the task label is available for prediction is a strong assumption in continual learning and involves a need of supervision at test time.

4.5 Experiments

To illustrate the concrete effects of the limitations presented earlier, we propose the dataset "MNIST-Fellowship" for our experiments. This dataset is composed of three datasets (Fig. 4.5): MNIST [132], Fashion-MNIST [266] and KMNIST [46], each composed of 10 classes, those datasets should be learned sequentially one by one. We choose this dataset because it gathered three easy datasets

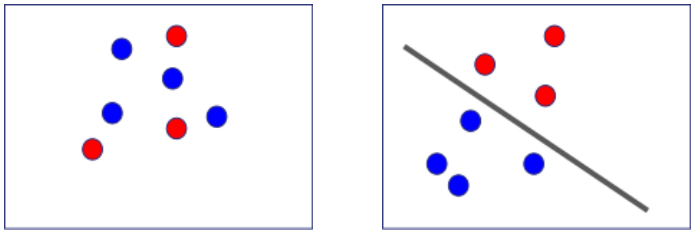


Figure 4.3: $\mathbb{D}_0$ feature space before learning T_0 (Left), $\mathbb{D}_0$ feature space after learning T_0 with a possible decision boundary (Right). Data points are shown by blue and red dots. The line (right part) is the model learned to separate data into the feature space.

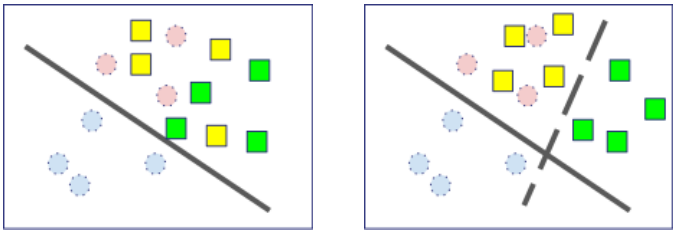


Figure 4.4: Case of representation overlapping while continual learning classification in a multi-task setting. At task T_1, feature space of $\mathbb{D}_1$ before learning T_1 (Left), Feature space of $\mathbb{D}_1$ after learning T_1 with a possible decision boundary (Right). New data are plotted as yellow and green squares and old data that are not available anymore to learn are shown with pale red and blue dots.

for prototyping machine learning algorithms but solving those three quite different datasets is still harder than solving only one.

Our goal is to illustrate the limitation of regularization based methods in disjoint settings. In particular that they can not distinguish classes from different tasks. We would like also to show that the shortcomming happen both in the output layer and in the feature extractor. Thus, we propose three different settings with the *MNIST-Fellowship* dataset.

- **1. Disjoint setting**: all tasks have different classes (i.e. from 0 to 29).
- **2. Joint setting**: all tasks have the same classes (i.e. from 0 to 9) but different data.
- **3. Disjoint setting with test label**: All tasks have different classes but at inference, we know from which task a data-point is coming from.

First setting (disjoint with no test label), is the hardest because all classes need to be discriminated from all the others. The second setting (joint) is a bit easier because we don't need to discriminate task from each other but the model needs to use the same output layer for all task which can produce interferences. Theoretically, the second setting requires only the feature extractor to be learned. The last setting (disjoint with test label) is the easiest, classes from different tasks don't need to be compared and the output layer is different for each task. The model used is presented in Table 4.1.

(a) Task 0 (b) Task 1 (c) Task 2

Figure 4.5: The three tasks of the MNIST-Fellowship dataset.

Table 4.1: Model architecture, convolution have 5*5 kernel size, maxpool have 2*2 kernel size. Parameters not mentioned are default parameters in Pytorch library [182] (in torch.nn). BS is for batch size, which is 64. All layers are initialized with Xavier init method [84].

Layer Name	Layer Type	Input Size	Output Size
Conv1	ReLu(MaxPool2d(Conv2d(input)))	BS*1*28*28	BS*10*12*12
Conv2	ReLu(MaxPool2d(Conv2d(input)))	BS*10*14*14	BS*20*4*4
FC1	ReLu(Linear(input))	BS*320	BS*50
FC2	functional.log_softmax(Linear(input))	BS*50	BS*10

With those three settings, We present two different experiments, a first one comparing disjoint settings with and without a label for inference. The goal is to bring to light that regularization fails in disjoint settings if the task label is not provided. Secondly, we experiment with the joint setting, to show that even if the feature extractor only needs to be learned the approach still struggles to learn continually and forget significantly.

We present EWC results with diagonal Fisher Matrix [121] and with Kronecker Factorization of the Fisher matrix [201]. We add an expert model which learned on the full dataset at once and a baseline model who learned continually without any memorization process. All models are trained with stochastic gradient descent with a learning rate of 0.01 and a momentum of 0.9. Even if continual learning does not support a-posteriori hyper-parameter selection, for fairness in comparison, the parameter lambda has been tuned. The best lambda upon $[0.1; 1; 2; 5; 10; 20; 100; 1000]$ is selected for each model. Then the model is trained on 5 different seeds.

The first experiment (Fig. 4.6), shows that *regularization* methods performances are significantly reduced when there is no test label in the disjoint settings. The experiment also shows that without labels for inference the model forgets almost instantaneously the first task when switching to the second one. Those results support the proposition 4.4.3. Indeed, the low performance of regularization methods without test labels in disjoint settings illustrates the output layer shortcomings in continual learning (task separability problem, Section 4.4.3).

In Experiment 2 (Fig. 4.7), since the classes are the same in all tasks, only the feature extrac-

tor needs to be learned continually. The low performance of the proposed models illustrates the shortcomings in the continual learning of the feature extractor (the latent features problem, Section 4.4.3).

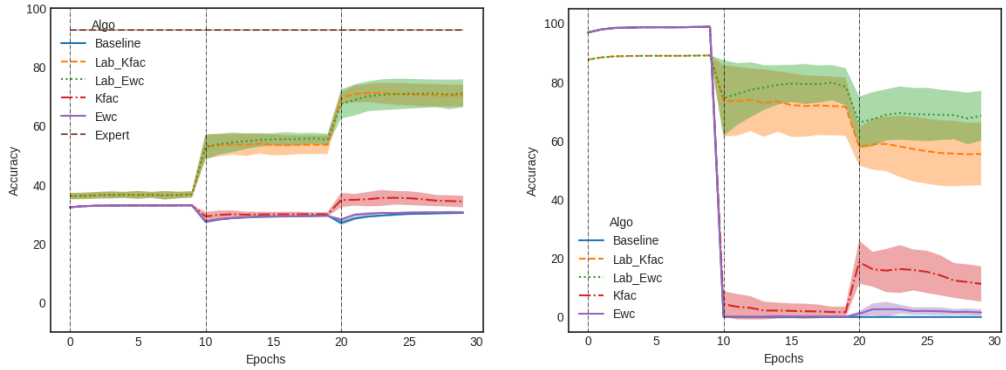


Figure 4.6: Experiment on disjoint classes without test label vs test label. Left, the mean accuracy of all 3 tasks, vertical dashed line are task transitions. Right, accuracy on the first task. Legends with 'Lab_' indicate experiments with task labels for test. The expert model is trained with i.i.d. data from all task and the baseline model is finetuned on each new task without any continual process.

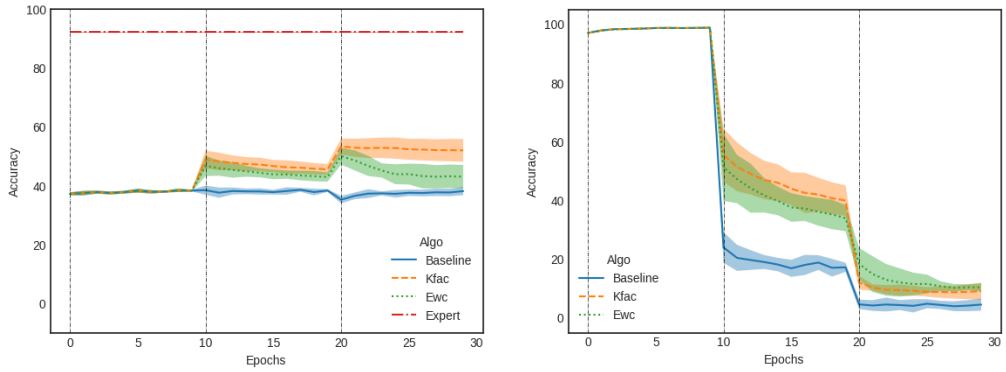


Figure 4.7: Experiments with joint classes. Left, the mean accuracy of all 3 tasks, vertical dashed line are task transitions. Right, accuracy on the first task.

These two experiments show that learning continually with regularization is only efficient in the setting with task label and maintains performance on task 0. The two other settings seem to either have interference in the output layer and in the feature extractor.

4.6 Discussion

The goal of this chapter is to propose a theoretical approach to the shortcomings of regularization methods in class-incremental settings. Regularization methods might have great characteristics for continual learning under certain conditions, but it is important to know their limitations to use the best of their capabilities. Regularization shortcomings could be offset with a replay method such as *rehearsal* or *generative replay*. We discuss in this section the potential applications where the shortcomings described have an impact, we also discuss the relationship between learning representation and memorizing. We finish by introducing how the replay methods could overcome the shortcomings described here.

4.6.1 Applications

In this section, we point out supplementary shortcomings of regularization in other types of learning situations, namely a classification task with one only class and multi-task continual reinforcement learning. We also use proposition 4.4.3 for the case of pre-trained models.

Learning from one class only

A classification task with only one class might look foolish, however, in a sequence of tasks with varying number of classes, it makes more sense and it seems evident that a CL algorithm should be able to handle this situation. Nevertheless, a classification neural network needs at least two classes to learn discriminative parameters. Hence, in a one-class task, the model learns no useful parameters, a regularization term can then a fortiori not protect any knowledge. As noted in [137], the regularization method is not suited for such setting. It is worth noting that in a real-life settings it is mandatory to be able to learn only one concept at a time.

Multi-task Continual Reinforcement Learning

Results from Section 4.4.2 can also be generalized to continual multi-tasks reinforcement learning settings [248]. In this setting, a model has to learn several policies sequentially to solve different tasks. At test time with no task label, the model needs both to be able to run the policies correctly but to infer which policy to run. However, since policies are learned separately inferring which one to run is equivalent to a class incremental task. Therefore, following proposition 4.4.3, the regularization based method will not be able to learn the implicit classification correctly. Hence, in continual multi-tasks RL a regularization method alone will fail if task label is not enabled at test time.

4.6.2 Using pre-trained models for continual learning

We showed in Section 4.4.2 that, in a class incremental classification scenario, regularization methods are not sufficient to learn continually. In the case of a pre-trained classification model on N classes that we want to train on new classes without forgetting, if the training data are not available for some reasons, then we don't even have a *regularization term* Ω to protect some. Following the proposition 4.4.3 and a fortiori without the regularization term, the model will forget past knowledge while learning new classes.

Using pre-trained models can be useful to converge faster to a new task solution but it will undoubtedly forget what it has learn previously.

4.6.3 Representation Learning and Memorization

We presented the theoretical impossibility to distinguish past classes from current classes based only on regularization. Primarily, because we are unable to find the good decision boundaries in the output layer and because we can not learn features to disentangle the past from the present. But also because the model representation overfits the task. The classifier only optimizes the current learning criterion, therefore data representations are restricted to it. Those representations could be insufficient to memorize the learning experiences correctly for future tasks, as in the cat/dog tasks described in the Section 4.3. In any way, to maximize discrimination between tasks with no test label it is mandatory to have good memorization of past tasks. This memorization can be performed either by modelling their data distribution with generative models or samples or by adding surrogate losses that help the model to learn general representations of past tasks. Memorization is intrinsically linked to representation. Hence, adding surrogate loss to improve the learned representation would a priori improve memorization and consequently continual learning.

4.6.4 Toward Replay Methods

In this chapter, we assume that a training "task label" is available, indicating each time a drift happens in the data distribution while learning. These settings make learning easier than when the drifts are not signalled in any way. In case of task label unavailability, it is even more important to have a robust and resilient memorization process to detect data distribution drift and to deal with catastrophic forgetting.

To be easily applicable in real-life scenarios, algorithms should eventually have access to task labels for learning but not for inference. A family of algorithms that fit this requirement is the family of replay methods. It gathers rehearsal methods and generative replay described in Chapter 3. Replay methods emulate the simultaneous apparition of two concepts making it possible to learn good decision boundaries. Replay hence enables the possibility to reinterpret data from past tasks with the current state of knowledge and belief. They can learn task where there is only one concept at a time. The memorization process is agnostic to the past task learning criterion. It is driven by high-level rules. Moreover, the method is agnostic to the number of tasks, the model solving the current task and the task label at test time. Finally, the memory is easy to control, if in any way we want to forget some knowledge, the algorithms just have to stop replaying them and the model will forget automatically.

Some additional assets can be attributed to rehearsal and generative replay separately:

- **Rehearsal**: As long as the samples are saved in the memory, they can not be forgotten and there is no risk of damaging memories. Rehearsal is very suited for online learning, it can save samples as soon as they have been received.

- **Generative Replay:** The generative replay uses a learning criterion to learn the data distribution different from the task criterion. Therefore, it enables the possibility of a supplementary generalization than the generalization of the task learning criterion. Moreover, the generative model synthesizes the information and offers an automatic compression process for memorization.

The replay methods look, therefore, very appealing for research in continual learning. They look adapted, robust and resilient enough to deal with various continual learning settings.

4.7 Conclusion

Regularization is a widespread method for continual learning. However, we proved that for class-incremental classification, no regularization method can learn alone to discriminate classes from different tasks. At test time, this shortcoming makes them dependent on the task label for prediction. This need for supervision during inference restricts significantly the application scenario possible for regularization methods alone.

The class-incremental scenario is a specific benchmark measuring the ability of algorithms to learn sequentially different classes. However, being unable to deal with this setting implies that in a more complex learning environment, all sub-tasks interpretable as class-incremental will be ignored by the algorithm.

This shortcoming in regularization methods is one of the motivations to study replay methods for continual learning in the following chapters of this thesis. In particular, in the next chapter, we will study the capacity of generative models to learn continually. This study allows to determine the best conditions to use a generative model for generative replay.

Chapter 5

Learning Continually a Generative Models

In the previous chapter, we have highlighted the shortcomings of regularization methods for continual learning in the case of disjoint classification. We also stressed the benefits of replay for continual learning. The use of rehearsal is a known method for continual learning and the recent improvements of generative models motivate us to explore generative replay methods. But first, we wanted to evaluate the capacity of generative models to learn continually. In this chapter, we lead an empirical study of generative models in the context of disjoint task generation.

This work has been realized in collaboration with Hugo Caselles-Dupré and led to the publication "Generative Models from the perspective of Continual Learning" published at IJCNN 2019 [135]. The original article has been slightly modified and extended to better fit the thesis and include figures and results cut by article size restriction. Moreover, we incorporate some results of the paper "Training Discriminative Models to Evaluate Generative Ones" [140] in the background section. This second paper has been published at ICANN 20019.

5.1 Introduction / Motivation

In this chapter, we focus on generative models in Continual Learning scenarios. Previous work on CL has mainly focused on classification tasks [121, 193, 226, 217] with approaches such as *regularization*, *rehearsal* and *architectural* strategies, as described in Chapter 3. However, discriminative and generative models strongly differ in their architecture and learning objective (classify images vs generating images). Several methods developed for discriminative models are thus not directly extendable to the generative setting. Once they are trained, generative models can be used as memory of the past for learning continually especially in reinforcement learning and classification. For example in [226], successful CL strategies with generative models have been used, via sample generation as detailed in the next section, to continually train discriminative models. Hence, studying the viability and success/failure modes of CL strategies for generative models is an important step towards a better understanding of generative models but also Continual Learning as a whole.

We conduct a comparative study of generative models with different CL strategies. In our experiments, we sequentially learn generation tasks. We perform ten disjoint tasks, using commonly used benchmarks for CL: MNIST [131], Fashion MNIST [266] and CIFAR10 [123]. In each task, the model gets a training set from one new class, and should learn to generate data from this class without forgetting what it learned in previous tasks, see Fig. 5.1 for an example with disjoint tasks on MNIST.

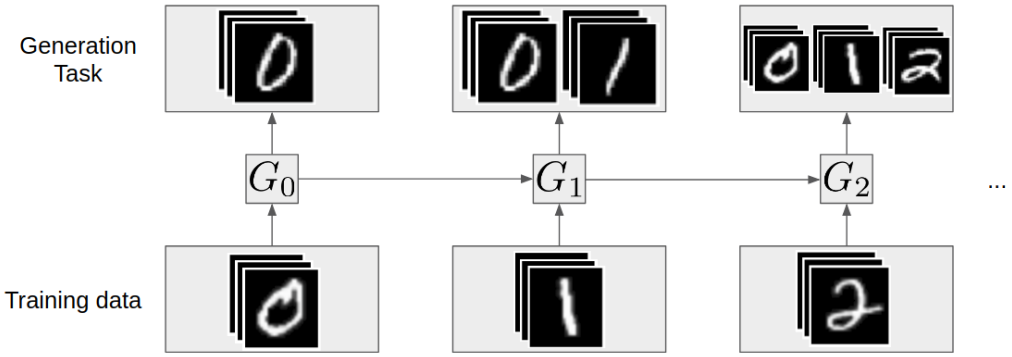


Figure 5.1: Illustration of the disjoint setting considered. At task i the training set includes images belonging to category i, and the task is to generate samples from all previously seen categories. Here MNIST is used as a visual example, we experiment in the same way Fashion MNIST and CIFAR10.

We evaluate several generative models: Variational Auto-Encoders (VAEs)[119], Generative Adversarial Networks (GANs) [88], their conditional variant (CVAE ans CGAN), Wasserstein GANs (WGANs)[9] and Wasserstein GANs Gradient Penalty (WGAN-GP)[90].We introduced those models in Section 2.2.3.

We compare results on approaches taken from CL in a classification setting: *finetuning*, *rehearsal*, *regularization* and *generative replay*. *Generative replay* consists of using generated samples to maintain knowledge from previous tasks. All CL approaches are applicable to both variational and adversarial frameworks. We evaluate with two quantitative metrics, Fréchet Inception Distance [97] and fitting capacity (FiC) [140], as well as visualization. Also, we discuss the data availability and scalability of CL strategies.

Our contributions are:

- We propose the Fitting Capacity, an evaluation method for generative models.
- Evaluating a wide range of generative models in a Continual Learning setting.
- Highlight success/failure modes of combinations of generative models and CL approaches.
- Comparing, in a CL setting, two evaluation metrics of generative models.

We describe related work in Section 5.2, and our approach in Section 5.3. We explain the experimental setup that implements our approach in Section 5.4. Finally, we present our results and discussion in Section 5.5 and 5.6, before concluding in Section 5.7.

5.2 Background

In this section, we will re-introduce the Fitting Capacity with some results from the original paper. The fitting capacity is a contribution from this thesis, we proposed to have a general evaluation of generative models. Then, we will present a brief state of the art on continual learning and generative models.

5.2.1 Fitting Capacity

As presented in Chapter 2, the Fitting Capacity (FiC) is a method to evaluate generative models. It measures the accuracy of a classifier trained on generated data to estimate the quality of the generator. It does not directly assess the realistic characteristics of the generated data but rather if their content and variability contain enough information to classify real data. This method makes it possible to take into account the complex characteristics of the generated samples and not only the distribution of their features. In this background section, we present a summary of [140] results to illustrate FiC evaluation.

The process for the fitting capacity is the following:

1. Train a conditional generative model over a train set
2. Generate a dataset D_{gen}
3. Train a discriminative model (a classifier) over D_{gen}
4. Iterate the process for several generative models and compare the accuracy of the classifiers on the test set.

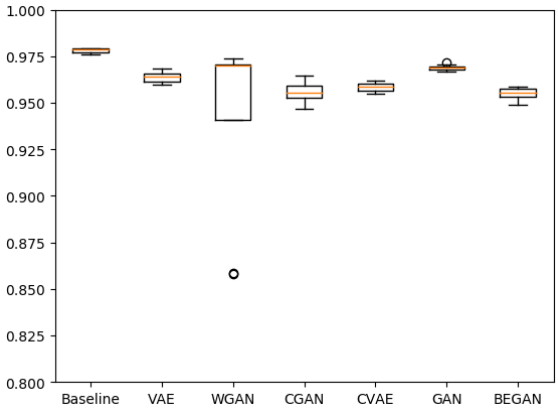


Figure 5.2: Fitting capacity evaluation of various generative models on MNIST. Each model has been trained on 8 different seeds to create plot bars. NB: the figure is zoomed to better appreciate the difference between models but consequently an outlier of the GAN model is hidden and is around 0.1.

The method makes it possible to compare generative models trained on similar datasets. In Figure 5.2, we can find the FiC of various generative models trained on MNIST and appreciate the difference among models. We can also evaluate the relative difference between metrics in Figure 5.3. This figure highlights the differences by normalizing performance on the best models. The baseline is a classifier trained on real data and not on generated ones. We can see that the FiC discriminates real data the most from generated ones and that FiC results do not correlate with other metrics results.

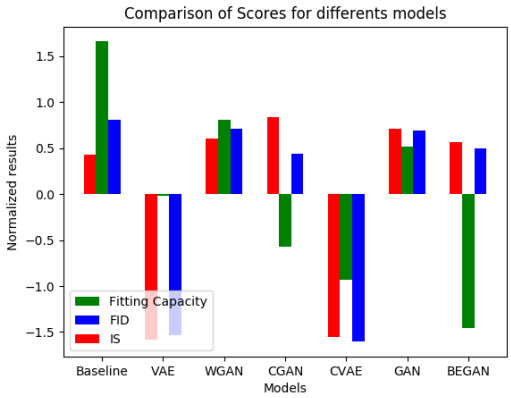


Figure 5.3: Comparison between normalized evaluation with inception score (IS), Frechet Inception Distance (FID) and Fitting Capacity (FiC). The normalization is achieved such as the sum of each metric over all the generative models is 0 with 1 standard deviation.

The Fitting Capacity is time-consuming since it necessitates to train a classifier for evaluation. Nevertheless, we believe that the fitting capacity is a good evaluation to estimate the intrinsic characteristics of generated data.

The outcome of the fitting capacity evaluation of generative models in article [140], was that GAN or WGAN models were producing the best samples but that VAE and CGAN were more stable and had a better mean accuracy. Moreover, conclusion was that inception score and Frechet inception distance do not make possible to predict if the data is good enough for downstream tasks. Similar evaluation methods also came out with that conclusion [210, 228, 54, 192].

In this chapter, we will use the Fitting Capacity to evaluate generative models trained continually.

5.2.2 State of the art

As said in the introduction of this chapter, discriminative and generative models do not share the same learning objective (classify data vs generate data) and architecture (top down vs bottom up). For this reason, CL strategies for discriminative models are usually not directly transferable to generative models. Continual Learning in the context of generative models remains largely unexplored compared to CL for discriminative models. In this section, we present the state of the art of continual learning specifically for generative models.

Among notable previous work, [218] successfully apply EWC on the generator of Conditional-GANs (CGANS), after observing that applying the same regularization scheme to a classic GAN leads to catastrophic forgetting. However, their work is based on a scenario where two classes are presented first, and then unique classes come sequentially, e.g the first task is composed of 0 and 1 digits of MNIST dataset, and then is presented with only one digit at a time on the following tasks. This is likely due to the failure of CGANs on single digits, which we observe in our experiments. Moreover, the method is only shown to work on CGANs. Another method for generative Continual Learning is Variational Continual Learning (VCL) [172], which adapts variational inference to a

continual setting. They exploit the online update from one task to another inspired from Bayes' rule. They successfully experiment on a single-task scenario. However, they experiment only on VAEs. Moreover, since they use a multi-head architecture, they use specific weights for each task, which requires task index for inference. A second method experimented on VAEs is to use a student-teacher method where the student learns the current task while the teacher retains knowledge [191]. Finally, VASE [2] is a third method, also experimented only on VAEs, which allocates spare representational capacity to new knowledge, while protecting previously learned representations from catastrophic forgetting by using snapshots (i.e. weights) of previous model.

A different approach, introduced by [226] is an adaptation of the *generative replay* method. It is applicable to both adversarial and variational frameworks. It uses two generative models: one which acts as a memory, capable of generating all past tasks, and one that learns to generate data from all past tasks and the current task. It has mainly been used as a method for Continual Learning of discriminative models [226, 222]. Recently, [263] authors have developed a similar approach called Memory Replay GANs, where they use Generative Replay combined to replay alignment, a distillation scheme that transfers previous knowledge from a conditional generator to the current one. However, they note that this method leads to mode collapse because it could favor learning to generate few class instances rather than a wider range of class instances.

5.3 Approach

Typical previous works on Continual Learning for generative models focus on presenting a novel CL technique and comparing it to previous approaches, on one type of generative model (e.g. GAN or VAE).

On the contrary, we focus on looking for the best generative model and CL strategy association. For now, empirical evaluation remains the only way to find the best performing combinations. Hence, we compare several existing CL strategies on a wide variety of generative models with the objective of finding the most suited generative model for Continual Learning.

In this chapter, we use two evaluation methods: the fitting capacity (FiC) [140] and the FID[97]. We believe that using these two metrics is complementary. FID is a commonly used metric based solely on the distribution of images features. In order to have a complementary evaluation, we use the fitting capacity, which evaluate samples on a classification criterion rather than features distribution. For unconditional models, we used an adaptation of the FiC where data are labelled by an expert network trained on the dataset.

For all the progress made in quantitative metrics for evaluating generative models [25], qualitative evaluation remains a widely used and informative method. While visualizing samples provides a instantaneous detection of failure, it does not provide a way to compare two well-performing models. It is not a rigorous evaluation and it may be misleading when evaluating sample variability.

5.4 Experiments

In this section, we present the experiments conducted to evaluate the generative models and different training strategies.

5.4.1 Experimental setup

We now describe our experimental setup: data, tasks, and evaluated approaches. The code is available online[1].

Our main experiments use 10 sequential tasks created using the MNIST, Fashion MNIST and CIFAR10 datasets. This setting is referenced to as disjoint classes setting or class-incremental setting. For each dataset, we define 10 sequential tasks, one task consists of learning to generate a new class and all the previous ones (See Fig. 5.1 for an example on MNIST). Both evaluations, FID and fitting capacity of generative models, are computed at the end of each task.

5.4.2 Strategies for continual learning

We focus on strategies that are usable in both the variational and adversarial frameworks. We use 3 different strategies for Continual Learning of generative models, that we compare to 3 baselines. Our experiments are done on 8 seeds with 50 epochs per tasks for MNIST and Fashion MNIST using Adam [118] for optimization. For CIFAR10, we experimented with the CL strategy that performed the best on the previous datatsets.

The first baseline is Fine-tuning, which consists of classical training, ignoring catastrophic forgetting. It is essentially a lower bound of the performance. Our other baselines are two upper bounds: Upperbound Data, for which one generative model is trained on joint data from all past tasks, and Upperbound Model, for which one separate generator is trained for each task.

For Continual Learning strategies, we first use a vanilla Rehearsal method, where we keep a fixed number of samples of each observed task, and add those samples to the training set of the current generative model. We balance the resulting dataset by copying the saved samples so that each class has the same number of samples. The number of samples selected, here 10, is motivated by the results in Fig. 5.4a and 5.4b, where we show that 10 samples per class is enough to get a satisfactory but not maximal validation accuracy for a classification task on MNIST and Fashion MNIST.

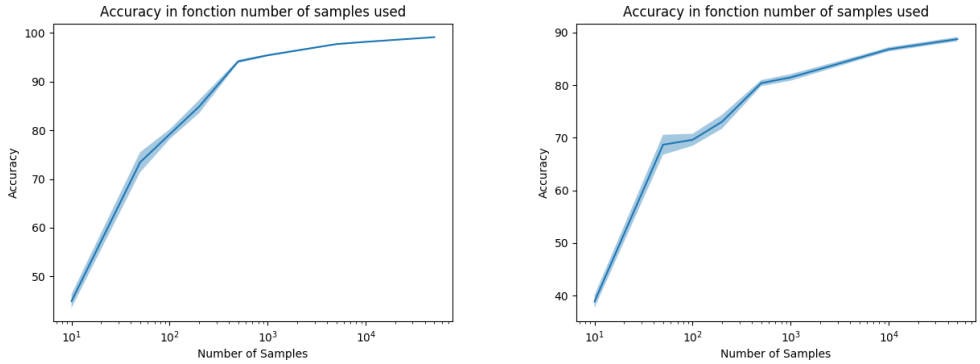
As they use the same test set, the fitting capacity (FiC) and the original accuracy with 10 samples per task can be compared. A higher FiC shows that the memory prevents catastrophic forgetting. Equal FiC means overfitting of the saved samples and lower FiC means that the generator failed to even memorize these samples.

We also experiment with EWC. We follow the method described by [218] for GANs, i.e. the penalty is applied only on the generator's weights , and for VAEs we apply the penalty on both the encoder and decoder. As tasks are sequentially presented, we choose to update the diagonal of the Fisher information matrix by cumulatively adding the new one to the previous one. The last method is Generative Replay, described in Section 5.2.2. Generative replay is a dual-model approach where a "frozen" generative model G_{t-1} is used to sample from previously learned distributions and a "current" generative model G_t is used to learn the current distribution and G_{t-1} distribution. When a task is over, G_t becomes the "frozen" model for training G_{t+1}.

The classification model used for experiments with MNIST and Fashion-MNIST is described in table 5.1. The model is slightly different from the one in Chapter 4, indeed we added the batch-normalization layer [101] to boost the model training speed and performance.

If we describe the experiments as proposed in the framework presented in Chapter 3, then we are in an unsupervised learning setting, multi-task scenario, with 10 disjoint tasks with iid data,

[1] https://github.com/TLESORT/Generative_Continual_Learning



(a) MNIST (b) fashion-MNIST

Figure 5.4: Test set classification accuracy as a function of number of training samples, on MNIST (Left) and Fashion-MNIST (Right). Those figures make possible to estimate the minimal number of samples needed to achieve a high test accuracy. Furthermore, by comparing against the fitting capacity we can estimate how many different images of the dataset a generator can produce.

Table 5.1: Classifier architecture, convolution have 5*5 kernel size, maxpool have 2*2 kernel size. Parameters not mentioned are default parameters in Pytorch library [182] (in torch.nn). BS is for batch size, which is 64. All layers are initialized with Xavier init method [84].

Layer Name	Layer Type	Input Size	Output Size
Conv1	ReLu(MaxPool2d(Conv2d(input)))	BS*1*28*28	BS*10*12*12
Conv2	ReLu(MaxPool2d(Conv2d(Dropout(input,p=0.5))))	BS*10*14*14	BS*20*4*4
FC1	ReLu(Linear(BatchNorm1d(input)))	BS*320	BS*50
FC2	functional.log_softmax(Linear(input))	BS*50	BS*10

with an integer oracle task label for training but not for testing (learning labels). The content update of this setting, is a new concepts type (NC).

Concerning the approaches experimented, the growth of memory is less than linear and the growth of computation is bounded by liner growth.

For memory, the generative replay has only one generative models as a memory and the regularization model only retains a set of weights and a Fisher matrix. It makes the growth in memory less than linear. For regularization, the sample number grows linearly but the memory of the model itself stays constant. Therefore the growth in memory is also less than linear.

The model upperbound method, which saves one mode per task has a linear growth in memory, the upperbound data has also a linear growth since it saves all data (and because tasks have the same size).

For computation, the growth of computation of generative replay is close to linear. The number of samples to generate increases linearly with the number of tasks. For rehearsal, the growth is also almost linear. In regularization, the growth is almost constant after the second task.

The upperbound model has constant computation and the upperbound data has linear growth

Table 5.2: Mean and standard deviations for Fitting Capacity (in %) metric evaluation on last task of 10 disjoint tasks setting, on MNIST and Fashion MNIST, over 8 seeds.

Strategy	Dataset	GAN	CGAN	WGAN	WGAN-GP	VAE	CVAE
Fine-tuning	MNIST	18.43±4.85	11.93±2.97	23.17±5.66	22.79±5.75	38.98±5.57	11.96±2.56
EWC	-	20.34±2.39	11.53±1.42	29.57±5.59	22.00±3.39	34.93±7.06	13.37±3.28
Rehearsal	-	82.69±18.21	66.14±19.2	92.05±0.64	74.79±25.25	92.99±0.64	86.47±1.69
Generative Replay	-	**95.81**±0.31	93.89±0.35	95.41±2.41	91.12±5.09	79.38±4.40	84.95±1.24
Upperbound Model	-	94.50±9.51	96.84±3.22	95.72±6.93	79.41±27.85	97.82±0.17	97.89±0.12
Upperbound Data	-	97.10±0.13	96.65±0.21	96.76±0.29	84.79±27.76	96.88±0.27	96.17±0.19
Fine-tuning	Fashion MNIST	20.82±4.69	12.30±3.33	19.68±3.92	18.75±2.58	18.60±4.24	12.82±3.55
EWC	-	22.22±2.03	12.58±3.48	19.81±4.18	22.63±6.91	17.70±1.83	11.00±1.16
Rehearsal	-	65.34±21.3	57.12±14.4	76.32±0.33	63.28±7.9	76.03±1.77	71.73±1.29
Generative Replay	-	**81.52**±0.87	72.98±1.22	81.50±1.26	75.37±5.49	54.49±3.24	68.70±1.71
Upperbound Model	-	77.93±15.07	80.96±0.69	73.20±5.63	65.5±2.69	78.64±1.36	79.15±0.96
Upperbound Data	-	83.27±0.41	80.09±0.94	83.29±0.52	81.5±0.50	80.21±0.79	79.51±0.55

of computation since the number of data grows linearly.

5.5 Results

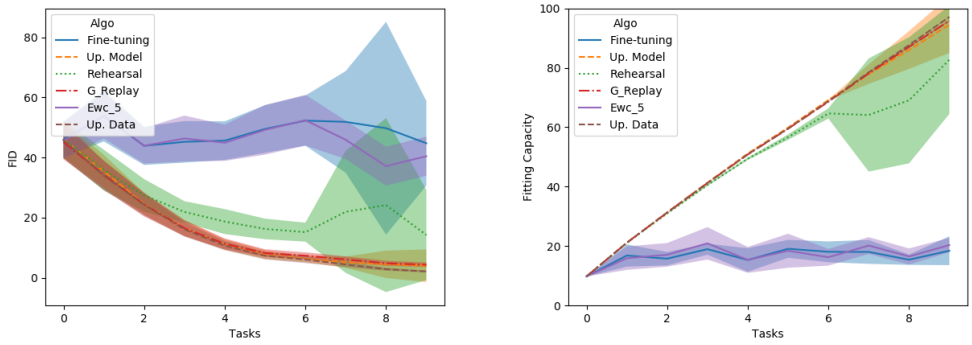


Figure 5.5: Comparison, averaged over 8 seeds, between FID results (left, lower is better) and Fitting Capacity results (right, higher is better) with GAN trained on MNIST.

The figures we report show the evolution of the metrics through tasks. Both FID and FiC are computed on the test set. A well-performing model should increase its FiC and decrease its FID. We observe a strong correlation between the FiC and FID (see Fig. 5.5 for an example on GAN for MNIST).

Nevertheless, FiC results are more stable: over the 8 random seeds we used, the standard deviations are less significant than in the FID results. For that reason, we focus our interpretation on the FiC results.

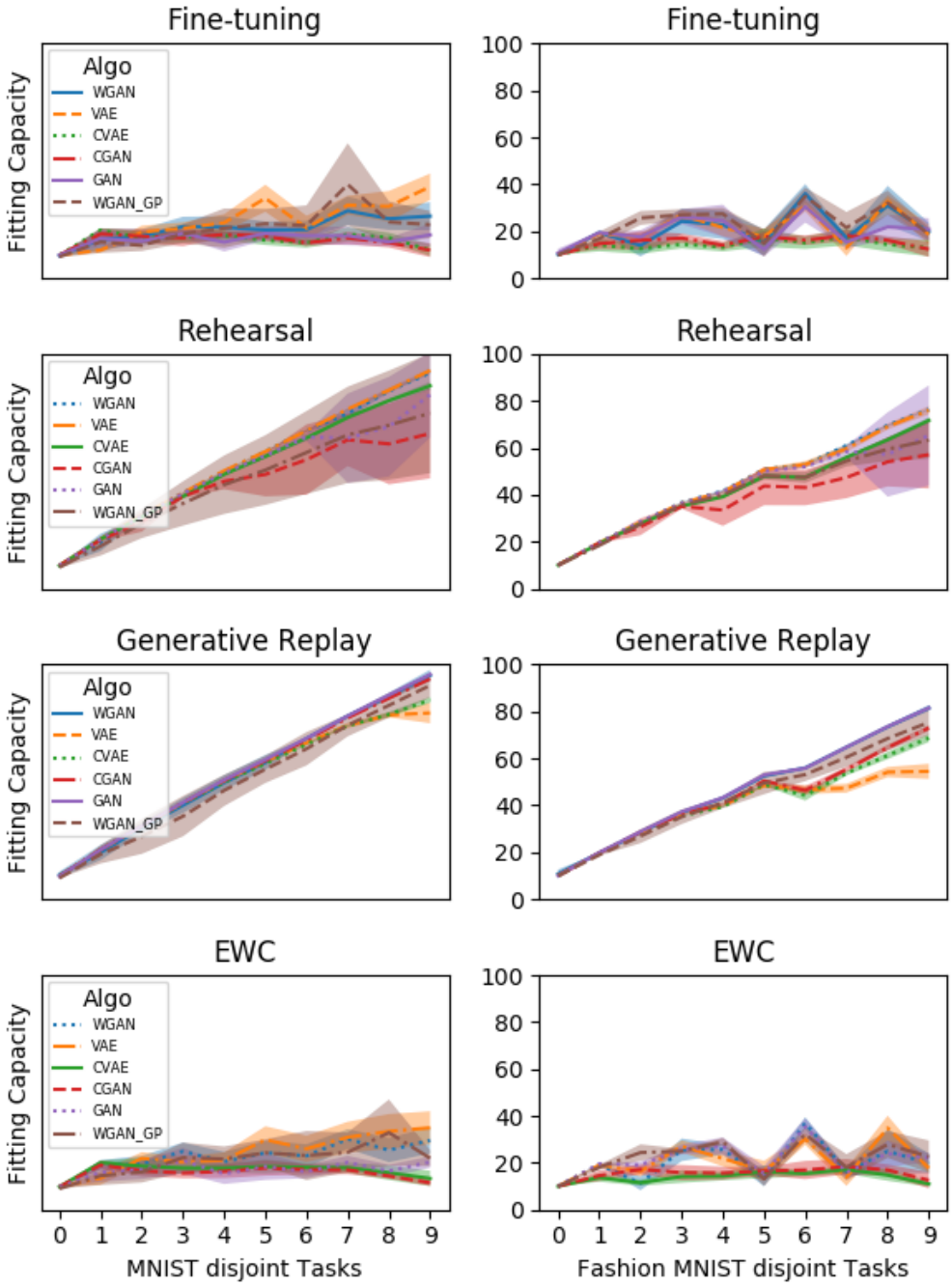


Figure 5.6: Means and standard deviations over 8 seeds of Fitting Capacity metric evaluation of VAE, CVAE, GAN, CGAN and WGAN. The four considered CL strategies are: Fine Tuning, Generative Replay, Rehearsal and EWC. The setting is 10 disjoint tasks on MNIST and Fashion MNIST.

Figure 5.7: CGAN augmented with EWC. MNIST samples after 5 sequential tasks of 2 digits each. Catastrophic forgetting is avoided.

5.5.1 MNIST and Fashion MNIST results

Main results

Our main results with fitting capacity (FiC) are displayed in Fig. 5.6 and Table 5.2. The best combination was Generative Replay + GAN with a mean FiC of 95.81% on MNIST and 81.52% on Fashion MNIST. We observe that, for the adversarial framework, Generative Replay outperforms other approaches by a significant margin. However, for the variational framework, the Rehearsal approach was the best performing. The Rehearsal approach worked quite well but is unsatisfactory for CGAN and WGAN-GP. Indeed, the FiC is lower than the accuracy of a classifier trained on 10 samples per classes (see Fig. 5.4a and 5.4b).

In our setting, EWC is not able to overcome catastrophic forgetting and performs as well as the naive Fine-tuning baseline which is contradictory with the results of [218] who found EWC successful in a slightly different setting. We replicated their result in a setting where there are two classes by tasks, showing the strong effect of task definition (Illustration Figure 5.7).

In [218] authors already found that EWC did not work with non-conditional models but showed successful results with conditional models (i.e. CGANs). The difference comes from the experimental setting. In [218], the training sequence starts by a task with two classes. Hence, when CGAN is trained it is possible for the Fisher Matrix to understand the influence of the class-index input vector c. In our setting, since there is only one class at the first task, the Fisher matrix does not capture the importance of the class-index input vector c. Hence, as for non-conditional models, the Fisher Matrix is not able to protect weights appropriately and at the end of the second task the model has forgotten the first task. Moreover, since the generator forgot what it learned at the first task, it is only capable of generating samples of only one class. Then, the Fisher Matrix will still not capture the influence of c until the end of the sequence.

Moreover, we show that even by starting with two classes, when there is only one class for the second task, the Fisher matrix is not able to protect the class from the second task in the third task (see Figure 5.8).

Our results do not highlight a clear distinction between conditional and unconditional models. However, adversarial methods perform significantly better than variational methods. GANs variants are able to produce better, sharper quality and variety of samples. Hence, adversarial methods seem more viable for CL. Samples for each model can be visualized on figures 5.9 and figure 5.10. We can

(a) Task 2 (b) Task 3 (c) Task 4

Figure 5.8: Reproduction of EWC experiments from [218] with four tasks. First task with 0 and 1 digits, then digits of 2 for task 2, digits of 3 for task 3 etc. The first task results are not shown in the figure but the generated images where accurate 0 and 1 as expected. When task contains only one class, the Fisher information matrix cannot capture the importance of the class-index input vector because it is always fixed to one class. This problem makes the learning setting similar to a non-conditional models one which is known to not work [218]. As a consequence 0 and 1 are well protected when following classes are not. Samples to illustrate that the method works if there are several classes at each tasks are in Fig. 5.7

link the accuracy from 5.4a and 5.4b to the fitting capacity results. As an example, we can estimate that GAN with Generative Replay is equivalent for both datasets to a memory of approximately 100 samples per class.

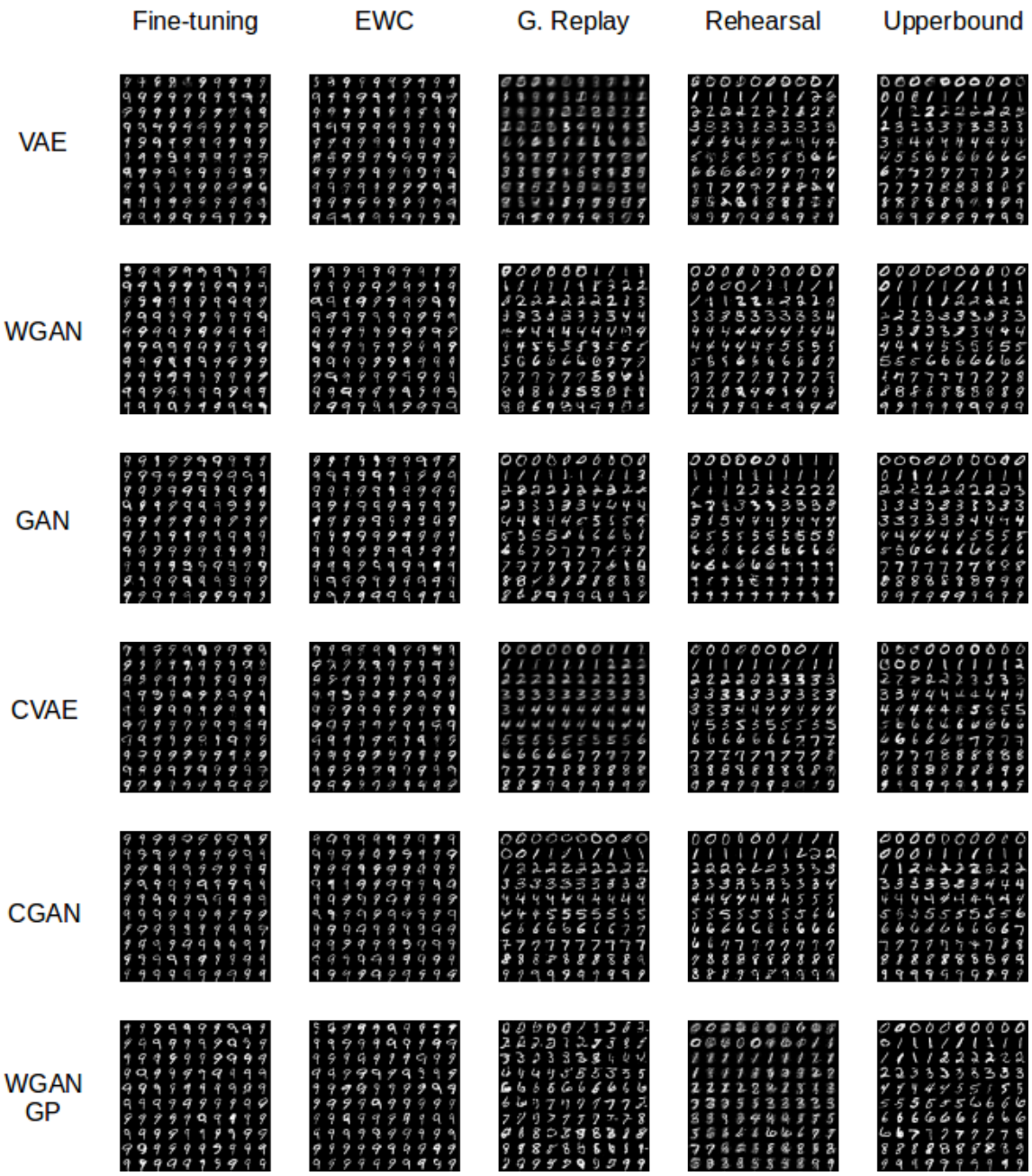


Figure 5.9: For each method and model, images sampled after the 10 sequential tasks with MNIST.

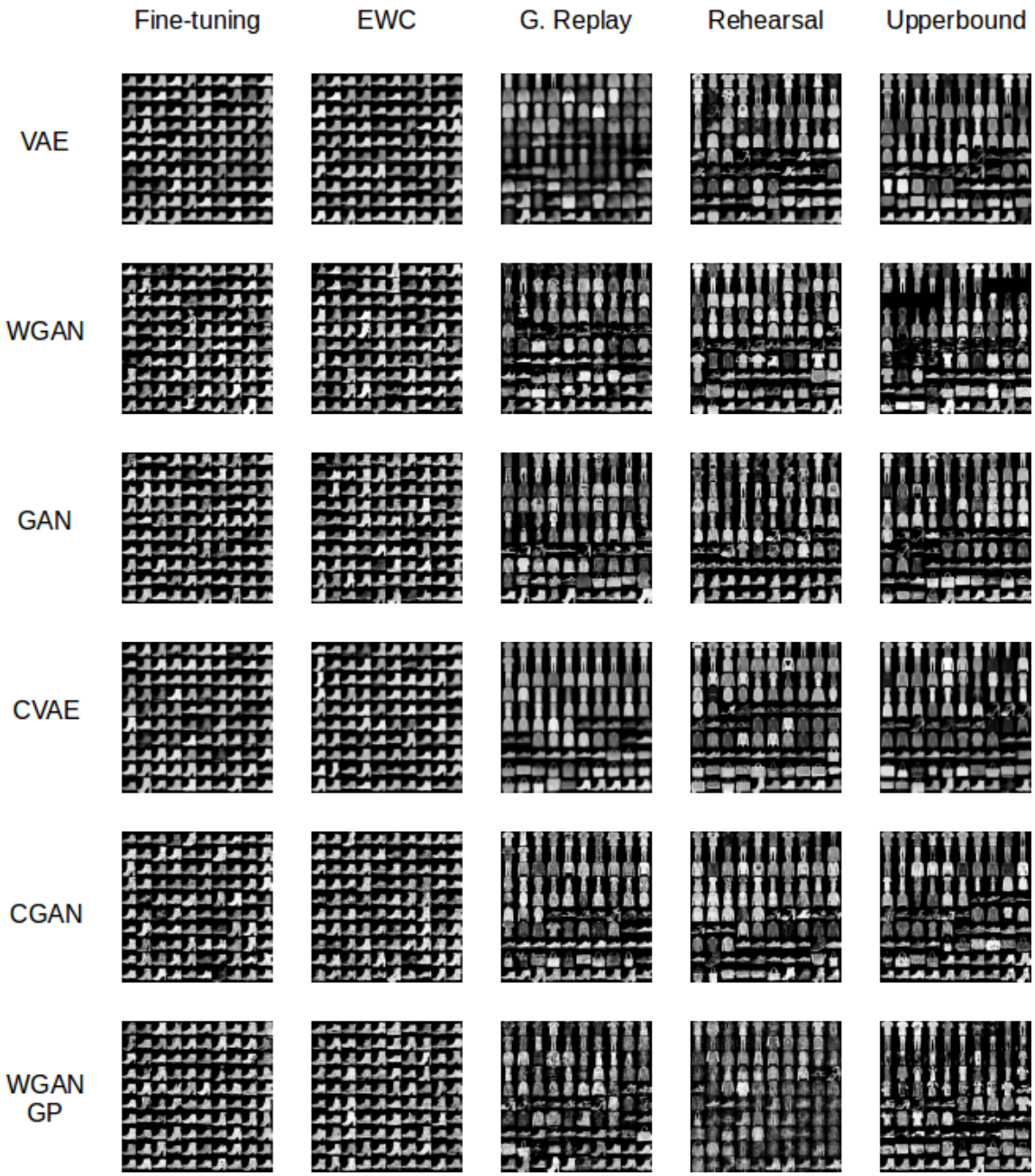
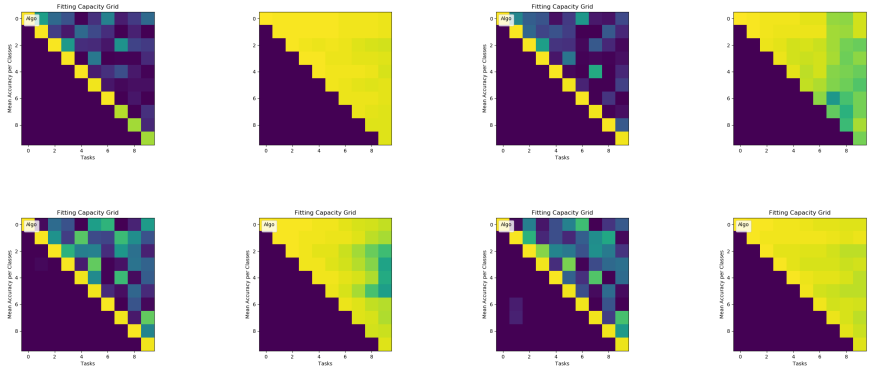


Figure 5.10: For each method and model, images sampled after the 10 sequential tasks with Fashion-MNIST.

Corollary results

Catastrophic forgetting can be visualized in Fig.5.11. Each square's column represents the task index and each row the class, the color indicates the fitting capacity (FiC). Yellow squares show a high FiC, blue ones show a low FiC. We can visualize both the performance of VAE and GAN but also the performance evolution for each class. For Generative Replay, at the end of the task sequence, VAE decreases its performance in several classes but GAN does not. For Rehearsal, we observe the opposite.



(a) Fine-tuning (b) G. Replay (c) EWC (d) Rehearsal

Figure 5.11: Fitting Capacity results for GAN (top) and VAE (bottom) on MNIST. Each square is the accuracy on one class for one task. Abscissa is the task index (left: 0 , right: 9) and orderly is the class index (top: 0, down: 9). The accuracy is proportional to the color (dark blue : 0%, yellow 100%)

Concerning the high performance of original GAN and WGAN with Generative Replay, those models benefit from their samples quality and their stability. In comparison, samples from CGAN and WGAN-GP are moisier and samples from VAE and CVAE blurrier. However, in the Rehearsal approach, GANs-based models seem much less stable (See variance in Table 5.2 and Figure 5.6). In this setting, as the dataset is made of few data samples, the discriminative task is almost trivial for the discriminator and easy to overfit which makes training harder for the generator. In opposition, VAE-based models are particularly effective and stable in the Rehearsal setting (See Fig. 5.11). Indeed, their learning objective (pixel-wise error) is not disturbed by a low samples variability and their probabilistic hidden variables make them less prone to overfit.

However, the FiC of Fine-tuning and EWC in Table 5.2 is higher than expected for unconditional models. As the generator is only able to produce samples from the last task, the FiC should be near 10%. This is a downside of using an expert for annotation before computing the FiC. Fuzzy samples can be wrongly annotated, which can artificially increase the labels variability and thus the FiC of low performing models, e.g., VAE with Fine-tuning. However, this results stay lower than the FiC of well performing models.

Incidentally, an important side result is that the fitting capacity (FiC) of conditional generative models is comparable to results of Continual Learning classification. Our maximal performance in this setting is with CGAN: 94.7% on MNIST and 75.44% on Fashion MNIST. In a similar setting

with two sequential tasks, which is arguably easier than our setting (one with digits from 0,1,2,3,4 and another with 5,6,7,8,9), [95] authors achieve a performance of 94.91%. This shows that using generative models for CL could be a competitive tool in a classification scenario. It is worth noting that we did not compare our results of unconditional models FiC with classification state of the art. Indeed, in this case, the FiC is based on an annotation from an expert not trained in a continual setting. The comparison would then not be fair.

5.5.2 CIFAR10 results

Figure 5.12: Images sampled from a WGAN-GP at the end of each tasks with CIFAR10. The images are similar to real images in their features but still not very likely to be real images.

In this experiment, we selected the best performing CL methods on MNIST and Fashion MNIST, Generative Replay and Rehearsal, and tested them on the more challenging CIFAR10 dataset. We compared the two methods to naive Fine-tuning, and to Upperbound Model (one generator for each class). The setting remains the same, one task for each category, for which the aim is to avoid forgetting of previously seen categories. We selected WGAN-GP because it produced the most visually satisfying samples on CIFAR10 (Figure 5.12). We can note that visual assessment allows detecting failing training even if it is not accurate to distinguish the best models.

FID and fitting capacity (FiC) curves throughout training are provided in Fig. 5.14, and generated samples after the 10 sequential tasks are displayed in Fig. 5.13, where we display images sampled after the 10 sequential tasks. The FiC results show that all four methods fail to generate images that allow to learn a classifier that performs well on real CIFAR10 test data. As stated for MNIST and Fashion MNIST, with non-conditional models, when the FiC is low, it can been artifi-

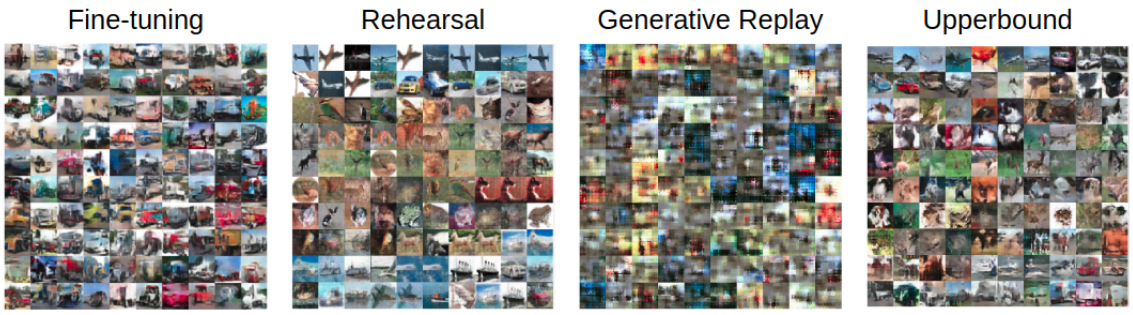


Figure 5.13: For each method, images sampled after the 10 sequential tasks with WGAN-GP trained on CIFAR10.

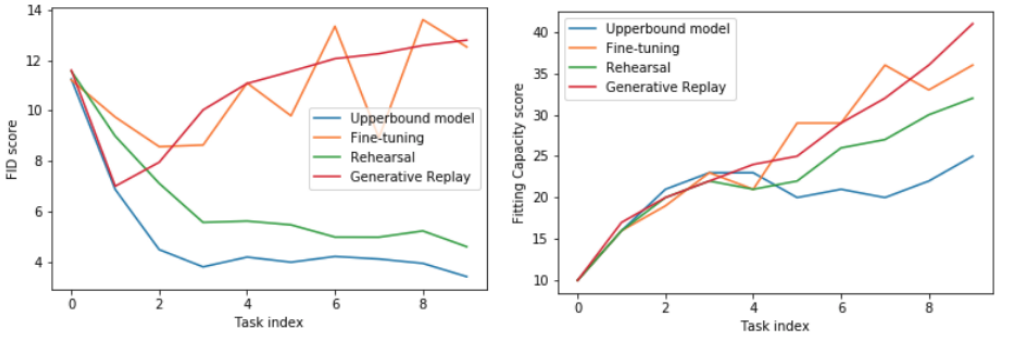


Figure 5.14: Fitting capacity and FID score of Continual Learning methods applied to WGAN_GP, on CIFAR10. The fitting capacity (FiC) (right figure) is too low to be significant. Within this range of fitting capacities, we cannot make any comparison between models. FiC is more suited for models which already work quite well. It explains probably why fine-tuning seems to work better than rehearsal in the CIFAR10 experiments.

cially increased by automatic annotation which makes the difference between curves not significant in this case. Naive Fine-tuning catastrophically forgets previous tasks, as expected. Rehearsal does not yield satisfactory results. While the FID score shows improvement at each new task, visualization clearly shows that the generator copies samples in memory, and suffers from mode collapse. This confirms our intuition that Rehearsal overfits to the few samples kept in memory. Generative Replay fails; since the dataset is composed of real-life images, the generation task is much harder to complete. At task 0, the generator is able to produce images that roughly resemble samples of the category, here planes. As tasks are presented, minor generation errors accumulated and snowballed into the result in task 9 (see Fig. 5.13): samples are blurry and categories are indistinguishable. As a consequence, the FID improves at the beginning of the training sequence, and then deteriorates at each new task. We also trained the same model separately on each task, and while the result might look visually satisfactory (see Upperbound in Fig. 5.13), the quantitative metrics show that generation quality is not excellent.

These negative results show that training a generative model on a sequential task scenario is not equivalent to successfully training a generative model on all data or each category, and that state-of-the-art generative models struggle on real-life image datasets like CIFAR10. Designing a CL strategy for these type of datasets remains a challenge.

5.6 Discussion

Besides the quantitative results and visual evaluation of the generated samples, the evaluated strategies have, by design, specific characteristics relevant to CL that we discuss in this section.

Rehearsal violates the data availability assumption that might be required in CL scenarios, by recording part of the samples. Furthermore, the risk of overfitting is high when only few samples represent a task, as shown in the CIFAR10 results. EWC and Generative Replay respect this assumption. EWC has the advantage of not requiring any computational overload during training, but this comes at the cost of computing the Fisher information matrix, and storing its values as well as a copy of previous parameters. The memory needed for EWC to save information from the past is twice the size of the model which may be expensive in comparison to rehearsal methods. As an example, the model we used to solve MNIST classification problem, is 95.5 kBytes, it is the same memory space as approximately 120 images. We saw that with 10 images per classes the rehearsal is quite effective. Nevertheless, with Rehearsal and Generative Replay, the model has more and more samples to learn from at each new task, which makes training computation cost increase at each new tasks.

Finally, we want to highlight the importance of using metrics that are sensitive to mode collapse (like the one we used). For example, [263] proposes a metric to evaluate CL for conditional generative models. For a given label l, they sample images from the generator conditioned on l and feed it to a pre-trained classifier. If the predicted label of the classifier matches l, then it is considered correct. In our experiment we find that it gives a clear advantage to rehearsal methods. As the generator may overfit the few samples kept in memory and then maximizes the evaluation proposed by [265], while not producing diverse samples. Nevertheless, even if their metric is unable to detect mode collapse or overfitting, it can efficiently expose catastrophic forgetting in conditional models.

5.7 Conclusion

In this chapter, we experimented with the viability and effectiveness of generative models on Continual Learning (CL) settings. We evaluated the considered approaches on commonly used datasets for CL, with two quantitative metrics. Our experiments indicate that on MNIST and Fashion MNIST, the original GAN combined to the Generative Replay method is particularly effective (samples at each task can be visualized Figure 5.15). This method avoids catastrophic forgetting by using the generator as a memory to sample from the previous tasks and hence maintains past knowledge. Furthermore, we shed light on how generative models can learn continually with various methods and present successful combinations. We also revealed that generative models do not perform well enough on CIFAR10 to learn continually. Since generation errors accumulate, they are not usable into a complex continual setting.

In the next chapter, we will study the use of generative models for learning continually on classification tasks.

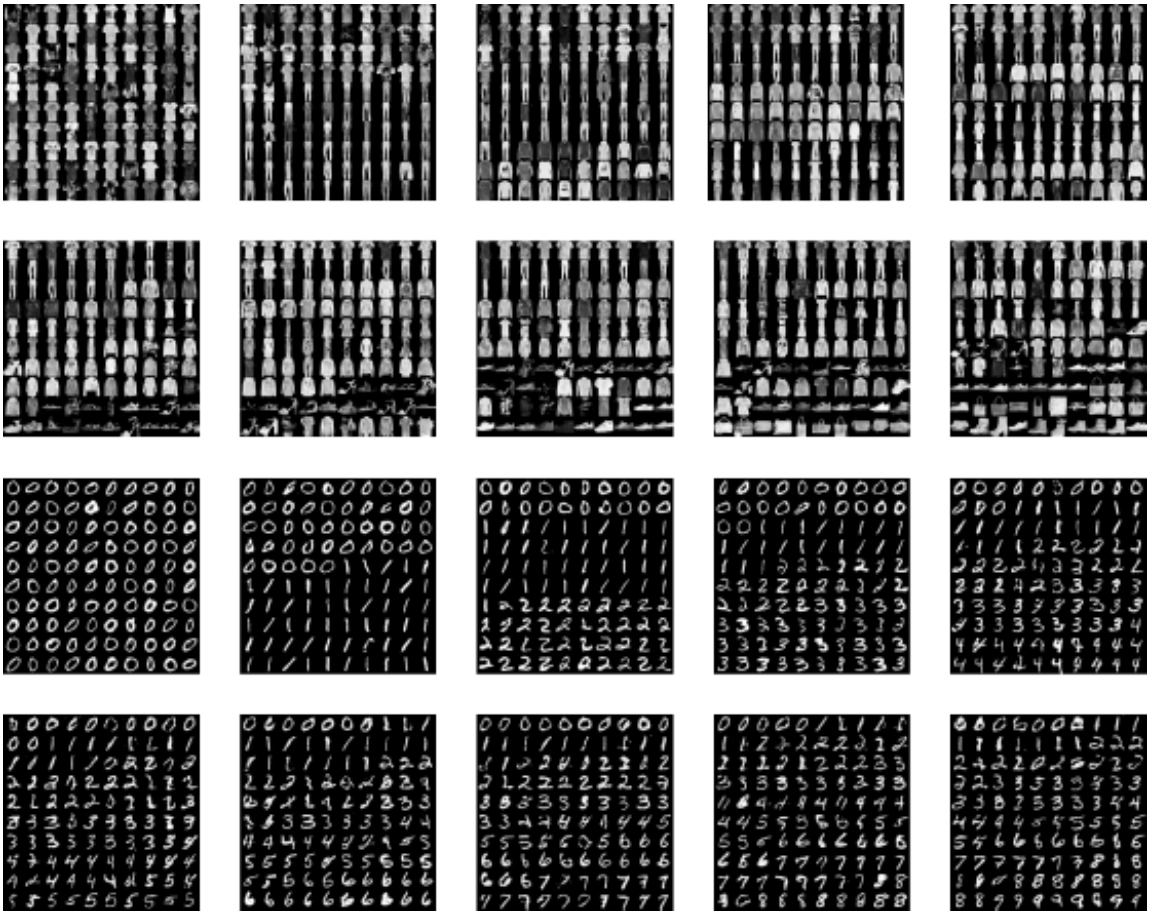


Figure 5.15: Samples at each task for the best working solution, GAN with generative replay.

Chapter 6

Generative Replay for Classification

In the previous chapter, we evaluate different generative models' ability to learn continually. In this chapter, we use the results of the previous section for a classification task. In particular, we study the use of generative replay for classification and the advantage of conditional models over marginal models for continual classification.

This work was done in collaboration with Alexander Geppert from Fulda University (Germany), it led to the publication of "Marginal Replay vs Conditional Replay for Continual Learning" at ICANN 2019 [137]. The original article has been slightly modified and extended to better fit the thesis and include figures and results cut by article size restriction.

6.1 Introduction

As seen in the previous chapter, the generative replay uses a generative model to generate data from past learning experiences to remember them. The *generative replay* approach is in principle model-agnostic. Indeed, it can be performed with a variety of machine learning models such as decision trees, support vector machines (SVMs) or deep neural networks (DNNs). It is also the case more generally to every replay method.

A downside of generative replay and similar approaches, such as rehearsal, is that the time complexity of adapting to a new task is not constant but depends on the number of preceding tasks that should be replayed. Or, conversely, if continual learning should be performed at constant time complexity, only a fixed amount of samples can be generated, and thus there will be forgetting, although it won't be catastrophic.

As in the previous chapter, we decided to investigate two different types of generative models: Generative adversarial networks (GAN) and variational auto-encoder (VAE). On one hand GAN are known to generate samples of high quality but on the other hand VAE directly maximize the likelihood of the learned distribution while training. It was then interesting to experiment both of them to compare their performances.

This chapter proposes and evaluates a particular method for performing replay using DNNs, termed "conditional replay", which is similar in spirit to [226] but presents important conceptual improvements (see next section). The main advantage of conditional replay is that samples can be generated conditionally, i.e., based on a provided label. Thus, labels for generated samples need not be inferred in a separate step as other replay-based approaches, e.g., [226], which we term

marginal replay approaches. Since inferring the label of a generated sample inevitably requires the application of a possibly less-than-perfect classifier, avoiding this step conceivably reduces the margin for error in complex continual learning tasks.

The original contributions of this chapter can be summarized as follows:

- **Conditional replay as a method for continual classification learning** We experimentally establish the advantages of conditional replay in the field of continual learning by comparing conditional and marginal replay models on a common set of benchmarks.
- **Improvement of marginal learning** We furthermore propose an improvement of marginal replay as proposed in [226] by using generative adversarial networks
- **New experimental benchmarks for generative replay strategies** To measure the merit of these proposals, we use two experimental settings that have not been previously considered for benchmarking generative replay: rotations and permutations. In addition, we promote the "10-class-disjoint" task as an important benchmark for continual learning as it is impossible to solve for purely discriminative methods (at no time, samples from different classes are provided for training so no discrimination can happen).
- **Comparison of generative replay to EWC** We show the principled advantage that generative replay techniques have with respect to regularization methods like EWC in a "one class per task" setting, which is a very common setting in practice and in which discriminatively trained models strongly tend to assign the same class label to every sample regardless of content. Those experiences correlate with what was discussed in Chapter 4.

The chapter organization is the following: first, in Section 6.2, we present some background material about generative replay state of the art. Second, in Section 6.3, we describe the methods used as well as the benchmarks. Third, in Section 6.5, we present the experiments conducted to compare conditional replay and marginal replay. Then, in Section 6.6 we show and discuss our results. Finally, in Section 6.7 we conclude about the advantages and limitations of the experimented methods.

6.2 Background

In this section, we present a summary of the state of the art from Chapter 3 oriented on generative replay for continual learning classification and introduce advanced methods for generative replay with improved sampling strategies.

6.2.1 State of the art

As already presented in Chapter 3 and 5, generative replay consists of training a generative model on the current task to replay it later. It benefits from the task learning criterion and the generation learning criterion to learn without forgetting in various types of tasks.

Concerning recent advances in generative replay improving upon [226], several works propose the use of generative models in continual learning of classification tasks [113, 265, 263, 222, 200]. Those approaches generally improves vanilla approach [226] with supplementary losses as with distillation loss [99]. They also experiment with harder datasets than MNIST such as LSUN [268] in [263]. However, their results do not provide comparison between different types of generative models.

Our results from [135] (Chapter 5) offer more insights on which generative models suit continual learning the best. In this chapter, we propose to build on those results to search for best generative replay method for classification.

Generally, each approach to continual learning has its advantages and disadvantages:

- dynamic architecture methods suffer from little to no interference between present and past knowledge as usually different networks or sub-networks are allocated to different learning tasks. The problem with this approach is that, on the one hand, model complexity is not constant, and more seriously, that the task from which a sample is coming from must be known at inference time in order to select the appropriate (sub-)network.

- regularization approaches are very diverse: in general, their advantage is simplicity and (often) a constant-time/memory behaviour w.r.t. the number of tasks. However, they have some theoretical shortcomings in class-incremental learning (See Chapter 4), the impact of the regularizer on continual learning performance is difficult to understand, and several parameters need to be tuned whose significance is unclear (i.e., the strengths of the regularization terms).

- generative replay show very good and robust continual learning performance, although time complexity of learning depends on the number of previous tasks for current generative replay methods. In addition, the storage of weights for a sufficiently powerful generator may prove very memory-consuming, so this approach cannot be used in all settings.

Hence, the choice of the generative model is crucial to correctly learn continually. The way generative models are used is also particularly important to maximize the algorithm's performance and make the best out of a trained generative model. We detail, in next section, the importance of sampling to improve results.

6.2.2 Sampling generative models

A well trained generative model can generate original data from the training data distribution. The generation process is generally the following, a seed z is sampled from a random distribution, typically a normal or uniform distribution. z it then fed to the generative model as input and the generative model outputs a data point. The objective is that for two different z the model generates two different data points.

The sampling can also be generated under conditions. The generative model can be trained to generate different types of data depending on a defined flag. For example, the flag can indicate from which class the model should generate data or from which task. The data is still generated randomly from z but a supplementary input c is given, conditioning the data point generated. This is the kind of sampling we use for *conditional replay*.

The sampling can also be adapted to maximize a certain criterion, like variety or similarity. In continual learning, it can be tuned to find the samples creating the maximum of interferences to use them and solve a conflict between learning criterion [5]. Finding the best sampling method is an interesting research field. However, in this chapter, the samples were selected uniformly as a vanilla method for fair comparison purposes.

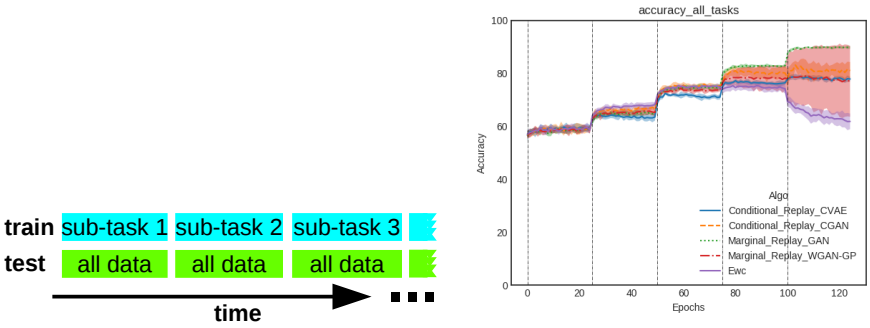
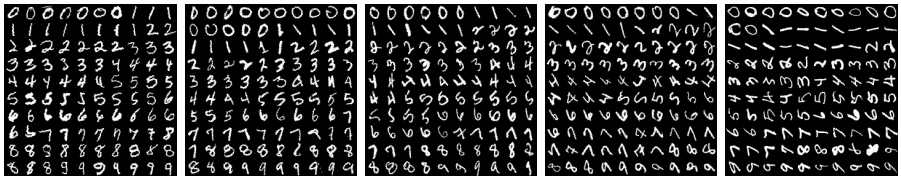


Figure 6.1: Illustration of a multi-tasks learning continuum. Left: The problem setting of continual learning as investigated in this chapter. DNN models are trained one after the other on a sequence of tasks (of which three are shown here), and are continuously evaluated on a test set consisting of the union of all task test sets. This gives rise to results as shown exemplarily on the right-hand side of the figure, i.e., plots of test set accuracy over time for different models, where boundaries between tasks (5 in this case) are indicated by vertical lines.



(a) task 0 (b) task 1 (c) task 2 (d) task 3 (e) task 4

Figure 6.2: MNIST training data for rotation tasks.

6.3 Approach

In this chapter, "learning a task" denotes learning on a classification problem that is composed of two or more tasks presented sequentially to the neural network model. Fig. 6.1 offers a visualization of the problem setting. Here, the tasks are constructed from two standard visual classification benchmarks: MNIST and Fashion MNIST, either by dividing available classes into several tasks, or by performing per-sample image processing operations that are identical within, and different between, tasks. All continual learning models are then trained and evaluated in an identical fashion on all tasks, and performances are compared by a simple visual inspection of classification accuracy plots.

6.3.1 Continual learning tasks

All tasks are constructed from the underlying MNIST and FashionMNIST benchmarks, so the number of samples in train and test sets for each task depend on the precise way of constructing them, as described below.

- **Rotations** New tasks are generated by choosing a random rotation angle $\beta \in [0, \pi/2]$ and

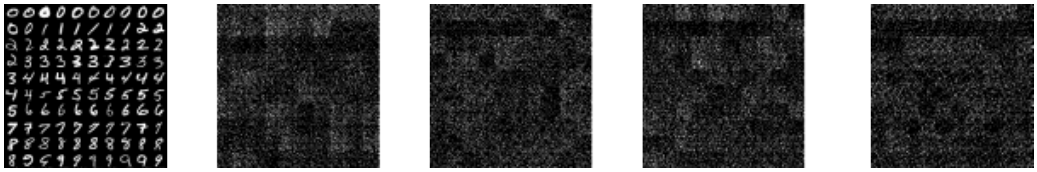


Figure 6.3: MNIST training data for permutation-type tasks.

(a) Task 0 (b) Task 1 (c) Task 2 (d) Task 3 (e) Task 4

(f) Task 5 (g) Task 6 (h) Task 7 (i) Task 8 (j) Task 9

Figure 6.4: Samples of MNIST training data for the disjoint tasks. Each task adds one more visual class, a principle which carries over identically to FashionMNIST.

then performing a 2D in-plane rotation on all samples of the original benchmark. As both benchmarks we use contain samples of 28x28 pixels, no information loss is introduced by this procedure. We limit rotation angles to $\pi/2$ because larger rotations could mix MNIST classes like 6 and 9. Each task in rotation-based tasks contains all 10 classes of the underlying benchmark, leading to 55.000 and 10.000 samples, respectively, in the train and test sets of each task.

- **Permutations** New tasks are generated by defining a random pixel permutation scheme, and then applying it to each data sample of the original benchmark. Each task in permutation-based tasks contains all 10 classes of the underlying benchmark, leading to 55.000 and 10.000 samples, respectively, in the train and test sets of each task.

- **Disjoint classes** For each benchmark, this task has as many tasks as there are classes in the benchmark. Each task contains the samples of a single class, i.e., roughly 6.000 samples in the train set and 1.000 samples in the test set. As the classes are balanced for both benchmarks, this does not unduly favor certain classes. This task presents a substantial challenge for machine learning methods since a normal DNN would, for each task, learn to map all samples to a single class label irrespective of content. Selective discrimination between any two classes is hard to obtain except if replay is involved, because then a classifier actually "sees" samples from different classes at the same time.

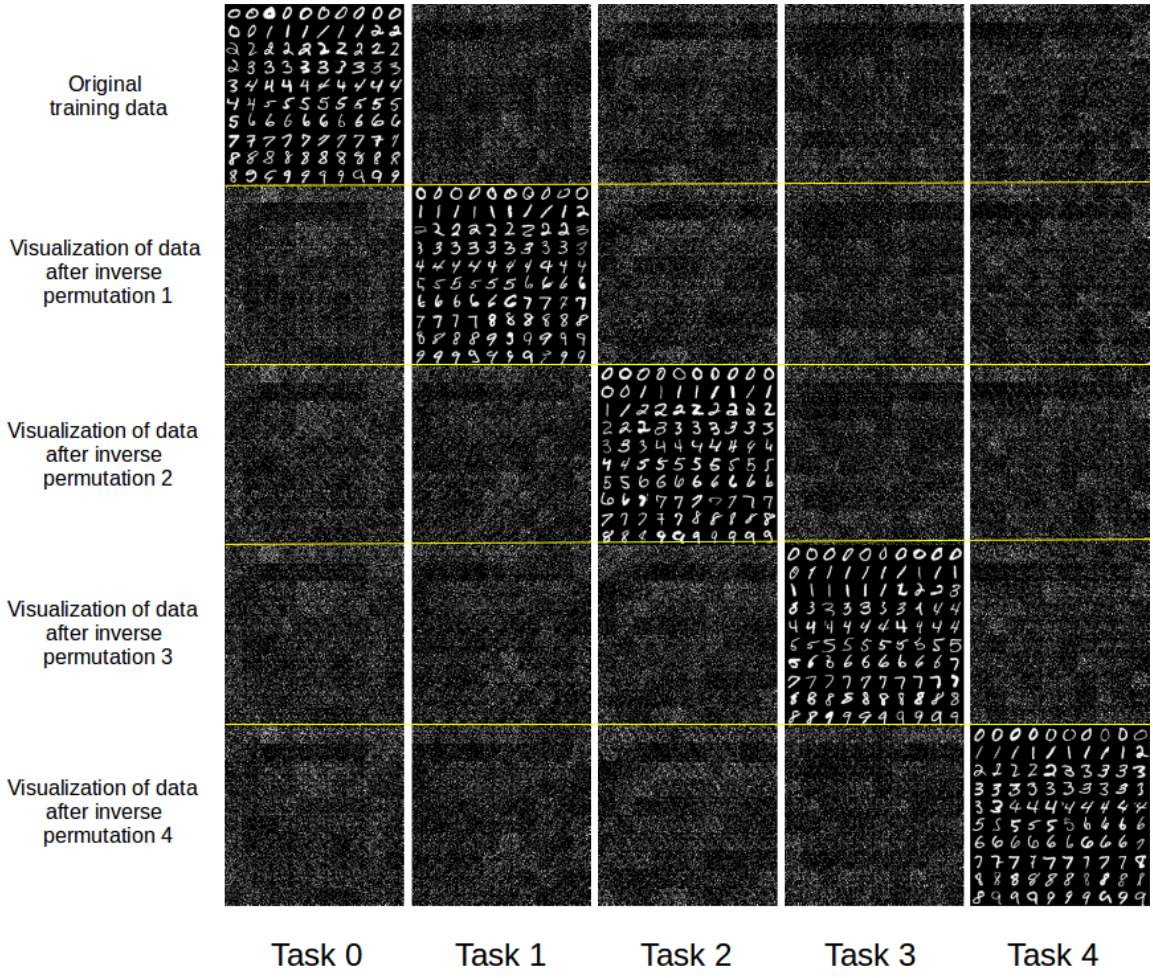
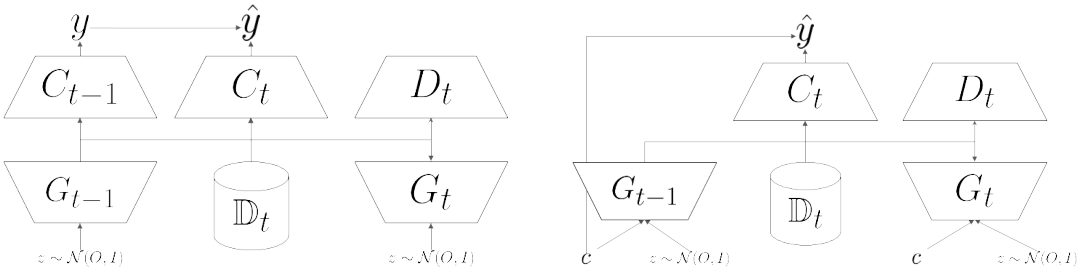


Figure 6.5: Visualization of training data for the MNIST permutations continual learning setting. The first line shows each task original training data. The other lines show the effect of all tasks inverse permutations applied to those data. Obviously, the inverse permutation i, results in original data at task i. This figure should help the interpretation of Figure 6.11

In this chapter, we will experiment with three different scenarios: 10 disjoint tasks (Fig. 6.4), five permutation tasks (See Fig. 6.3 and Fig. 6.5) and five rotation tasks (See Fig. 6.2).



(a) Generative replay (b) conditional replay

Figure 6.6: Illustration of Generative Replay and Conditional Replay for continual learning. The representation is done with a GAN architecture for the generative model but it could be adapted for another one. The Generative replay method train the current generative model G_t and classifier C_t with a mixture of the current dataset $\mathbb{D}_t$ and generated data from G_{t-1} and associated label y given by C_{t-1}. At the end of the task only G_t and C_t are kept frozen as the memory of the past. For Conditional Replay, the label is imposed to the generative model, therefore, we don't need an old classifier C_{t-1} but only G_{t-1}. At the end of the task, only G_{t-1} is kept frozen. Conditional Replay is then lighter to realize.

6.3.2 Models

Table 6.1: Hyperparameters for MNIST and Fashion MNIST all models (all CL settings have the same training hyper parameters with Adam)

Method	Epochs	LR Classifier	LR Generator	beta1	beta2	Batch Size
Marginal Replay	25	0.01	2e-4	5e-1	0.999	64
Conditional Replay	25	0.01	2e-4	5e-1	0.999	64
EWC	25	0.01	-	5e-1	0.999	64

As a reference implementation, we use a fully-connected network (2 hidden layers with 200 neurons each) with ReLU activation function. No batch normalization or dropout is performed. All other training parameters are described in Tab. 6.1. In this chapter, we compare a number of deep learning models: unless otherwise stated, we employ the Rectified Linear Unit (ReLU) transfer function, cross-entropy loss for classifier training, and the Adam optimizer.

- **EWC** We re-implemented the algorithm described in [121], choosing two hidden layers with 200 neurons each.
- **Marginal replay** In the context of classification, the *marginal replay* [135, 226, 263] method works as follows: For each task t, there is a dataset D_t, a classifier C_t, a generator G_t and a memory of past samples composed of a generator G_{t-1} and a classifier C_{t-1}. The latter two allow the generation of artificial samples D_{t-1} from previous tasks. Then, by training C_t and G_t on D_t and D_{t-1}, the model can learn the new task t without forgetting old ones. At the end of the task, C_t and G_t are frozen and replace C_{t-1} and G_{t-1} (see Fig. 6.6a). In the default setting, we use the generator for marginal replay in a way that ensures a balanced

distribution of classes from past tasks D_{t-1}, see also Fig. 6.7. This is achieved by choosing a predetermined number of samples N to be added for all tasks t, and letting the generator produce tN previous samples at task t. Thus, the number of generated samples increases linearly over time. We choose to evaluate two different models for the generator: WGAN-GP as used in [226] and the original GAN model [88] since it is a competitive baseline [140].

- **Conditional replay** The conditional replay method is derived from *marginal replay*: instead of saving a classifier and a generator, the algorithm only saves a generator that can generate conditionally (for a certain class). Hence, for each task t, there is a dataset D_t, a classifier C_t and two generators G_t and G_{t-1} (see Fig. 6.6b). The goal of G_{t-1} is to generate data from all the previous tasks during the training on the new task. Since data is generated conditionally, samples automatically have a label and do not require a frozen classifier. We follow the same strategy as for marginal replay (previous paragraph) for choosing the number of generated samples at each task. However, conditional replay does not require this: it can, in principle, keep the number of generated samples constant for each task since it is trivially possible to generate a balanced distribution of $\frac{N}{t}$ samples per class, from t different classes, via conditional sample generation. C_t and G_t learn from generated data D_{t-1} and D_t. At the end of a task t, C_t is able to classify data from the current and previous tasks, and G_t is able to sample from them also. We choose to use two different popular conditional models: CGAN described in [166] and CVAE [234].

6.4 Experiments

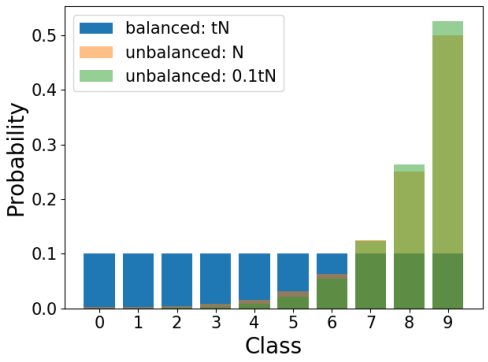


Figure 6.7: Illustration of classes imbalances for different replay strategies. Why marginal replay must linearly increase the number of generated samples: distribution of classes produced by the generator of a marginal replay strategy after sequential training of 10 tasks (of 1 class each). This essentially corresponds to the "disjoint" type of tasks. Shown are three cases: "*balanced*: tN" (blue bars) where tN samples are generated for each task t, "unbalanced: N" (orange bars) where the number of generated samples is constant and equal to the number of newly trained samples N for each task, and "unbalanced: $0.1tN$" where $0.1tN$ samples are generated. We observe that, in order to ensure a balanced distribution of classes, the number of generated samples must be re-scaled, or, in other words, must increase linearly with the number of tasks.

We conduct experiments using all models and tasks described in the previous section. Each class (regardless of the task) is presented for 25 epochs, Results are presented either based on the time-varying classification accuracy on the *whole* test set, or on the class (from the test set) that was presented first. In the first case, accuracy should ideally increase over time and reach its maximum after the last class has been presented. In the second case, accuracy should be stable if the model does not forgot or decrease over time, reflecting that some information about the first class is forgotten. We distinguish two major experimental goals or questions:

- Establishing the performance of the newly proposed methods (marginal replay with GAN, conditional replay with CGAN or CVAE) w.r.t. the state of the art. To this effect, we conduct experiments that increase the number of generated samples over time in a way that ensures an effectively balanced class distribution (see Fig. 6.7). We do this both for marginal and conditional replay in order to ensure a fair comparison, although technically conditional replay can generate balanced distribution even with a constant number of generated samples.
- Demonstrating the advantages of conditional w.r.t. marginal replay strategies, especially when only few samples can be generated, thus obtaining a skewed class distribution for marginal replay (see Fig. 6.7).

Results shedding light on the first question are presented in Fig. 6.8 (showing classification accuracy on whole test set over time, see Fig. 6.9 for accuracy on first task), whereas the second question is addressed in Fig. 6.12 for the disjoint task.

On the other hand, we can also analyze the stability performances of the different models. Results from Fig. 6.8 presenting accuracy performance on the different tasks sequence can be compared with the FID results of generative models Fig. 6.10. We can see a high similarity between the performance of generative models (FID) and the performance of the continual learning approach.

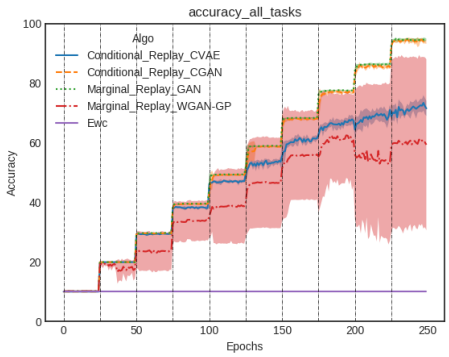
If we describe the experiments as proposed in the framework presented in Chapter 3, then we are in a supervised learning setting with multi-task scenario. They are 10 disjoint tasks with iid data or 5 joint tasks with iid data (for rotations and permutations tasks), with an integer oracle task label for training but not for testing (learning labels). The content update is a new concepts type (NC) for disjoint tasks and new instances for the others. The growth of used memory is less than linear and the growth of computation cost is bounded by linear growth as for generative replay described in Chapter 5.

6.5 Results

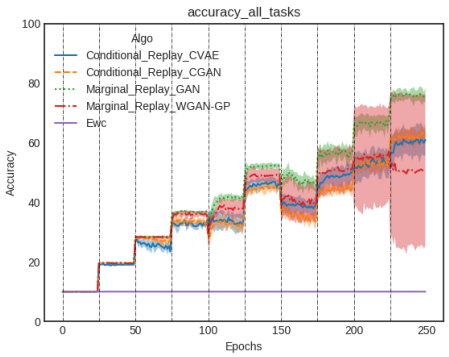
From the experiments described in the previous section, we can state the following principal findings:

6.5.1 Replay methods outperform EWC

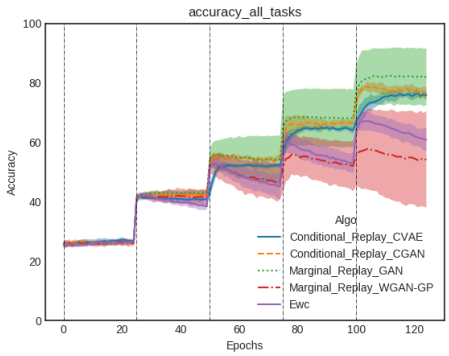
As can be observed from Fig. 6.8, the novel methods we propose (marginal replay with GAN and WGAN-GP, conditional replay with CGAN and conditional replay with CVAE) outperform EWC, on all tasks, sometimes by a large margin. Particular attention should be given to the performance of EWC: while generally acceptable for rotation and permutation tasks, it completely fails for the disjoint task. This is due to the fact that there is only one class in each task, making EWC try to map all samples to the currently presented class label regardless of input, since no replay is available to include samples from previous tasks.



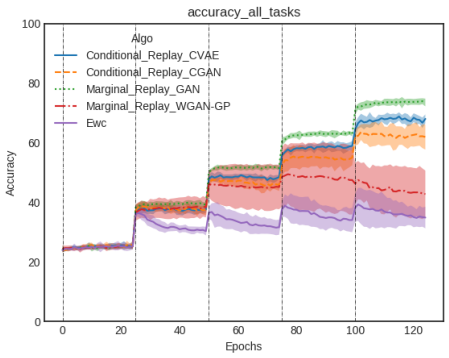
(a) accuracy for MNIST disjoint task



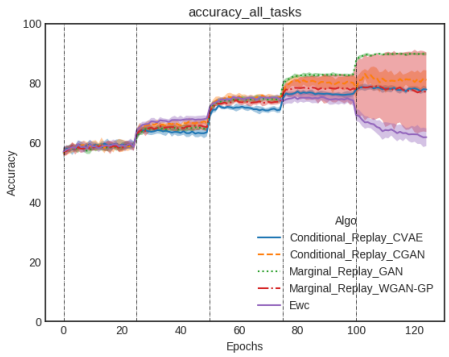
(b) accuracy for Fashion MNIST disjoint task



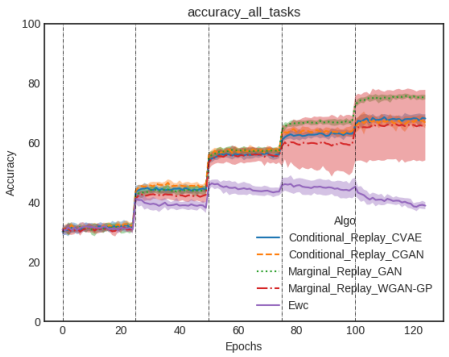
(c) accuracy for MNIST permutation task



(d) accuracy for Fashion MNIST permutation task



(e) accuracy for MNIST rotation task

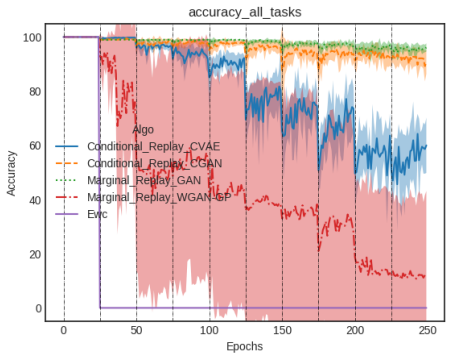


(f) accuracy for Fashion MNIST rotation task

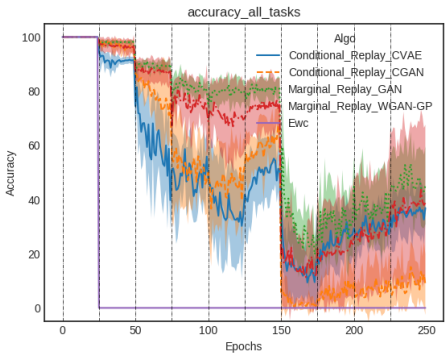
Figure 6.8: Test set accuracies during the training on different tasks, shown for all tasks (indicated by dotted lines).

6.5.2 Marginal replay with GAN outperforms WGAN-GP

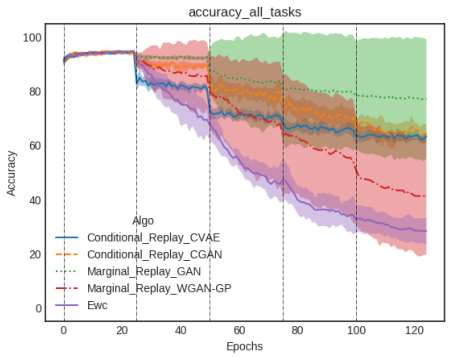
The clear advantage of GAN over WGAN-GP is the higher stability of the generative models. This is not only observable in the accuracy (see Fig. 6.8), but also when measuring performance on the first task only during the course of continual learning (see Fig. 6.9) and in the generative model FID (see Fig. 6.10). We show some samples of GAN for the permutation tasks in Fig. 6.11 (Fig. 6.5 helps to understand Fig. 6.11 by showing how original training data look like and how they look like after applying each inverse permutation. It makes it easier to see that a single trained GAN can generate correct data in all tasks whatever the permutation.). The permutation task is the hardest one for a generative model but we can see that the GAN is able to successfully generate samples from each tasks.



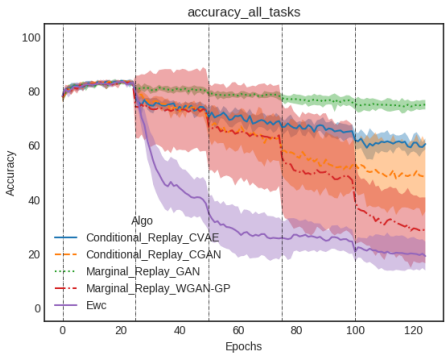
(a) MNIST: disjoint task



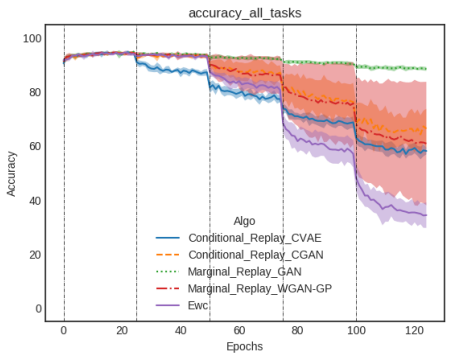
(b) Fashion MNIST: disjoint task



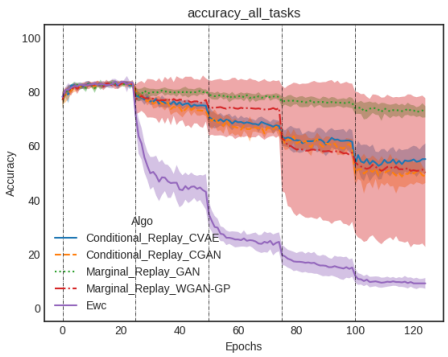
(c) MNIST: permutation task



(d) Fashion MNIST: permutation task

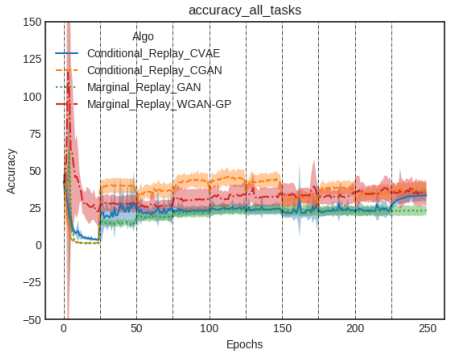


(e) MNIST: rotation task

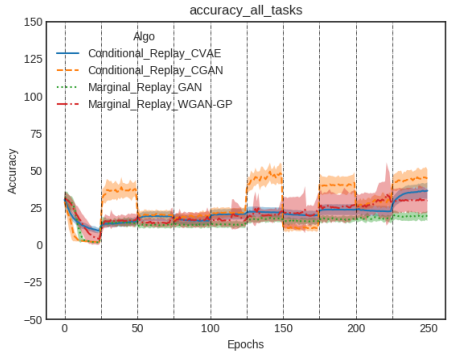


(f) Fashion MNIST: rotation task

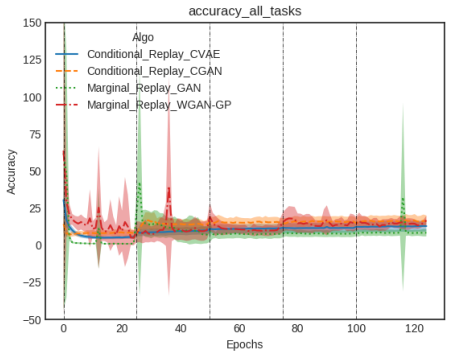
Figure 6.9: Comparison of the accuracy of each approach on the first task. This is another, very intuitive measure of how much is forgotten during continual learning. Means and standard deviations computed over 8 seeds.



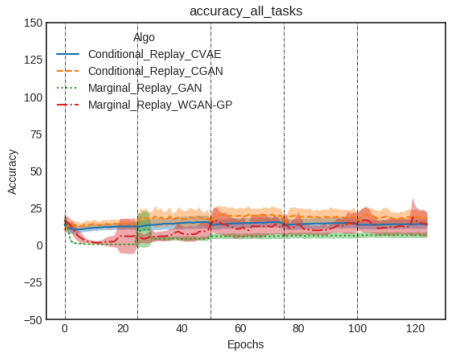
(a) MNIST: disjoint task



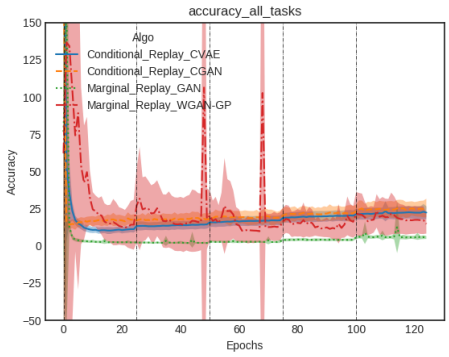
(b) Fashion MNIST: disjoint task



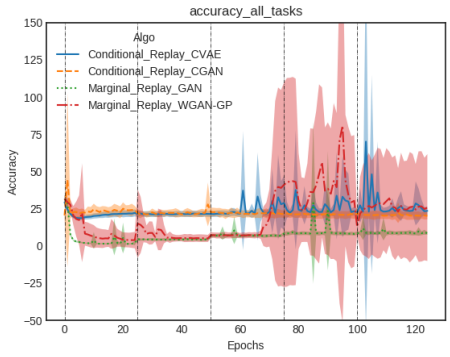
(c) MNIST: permutation task



(d) Fashion MNIST: permutation task



(e) MNIST: rotation task



(f) Fashion MNIST: rotation task

Figure 6.10: Comparison of the FID of each approach's generative models. Means and standard deviations computed over 8 seeds.

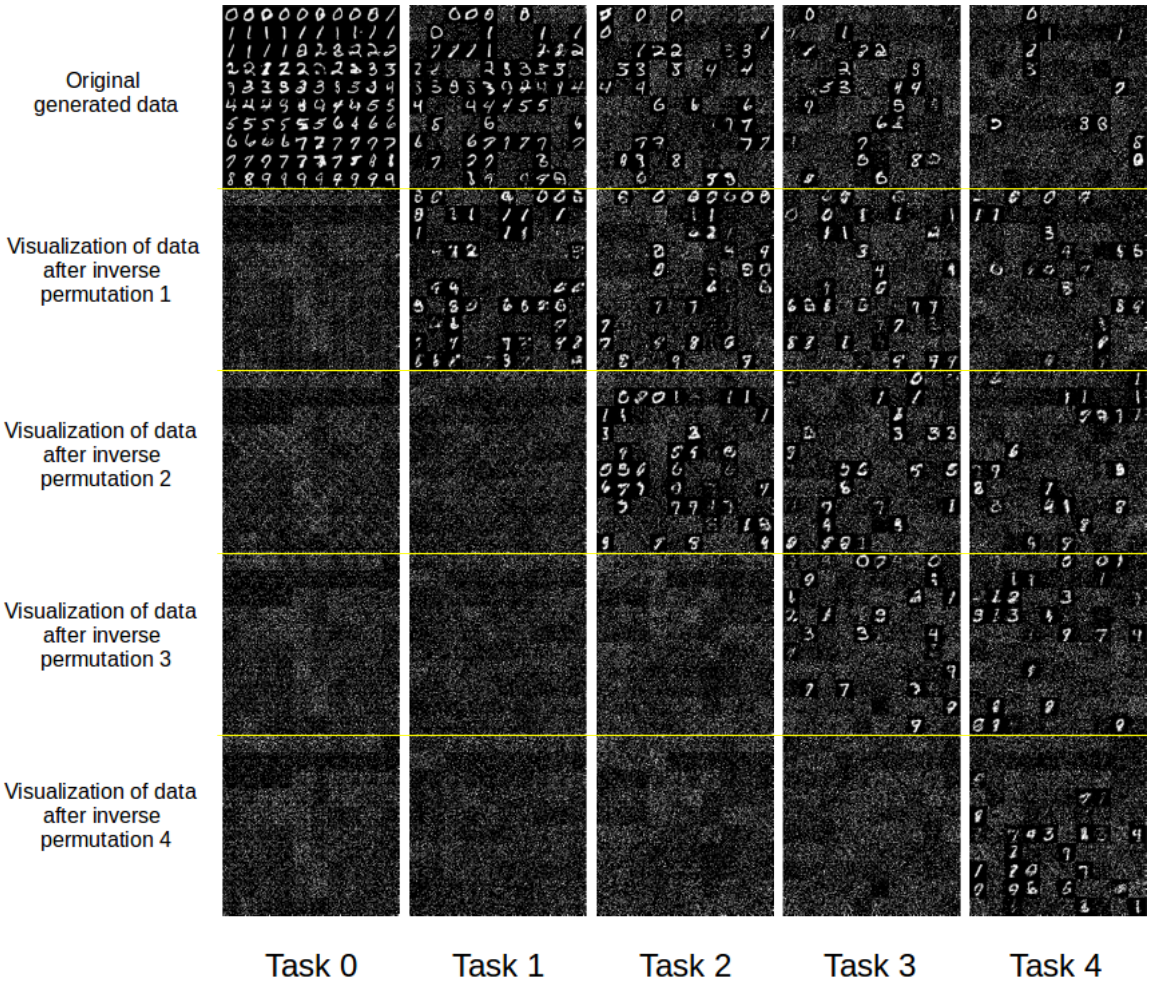


Figure 6.11: Visualization of data generated during the training of marginal replay + GAN on the MNIST permutation tasks. The first line shows each task generated data. The other lines show the effect of all tasks inverse permutations applied to those data as in Fig. 6.5. We observe that at each task the generative model can generate data in all previous permutation spaces. Then, we have a stable retention behaviour as the number of achieved tasks increases, while data from the current task is learned successfully as well.

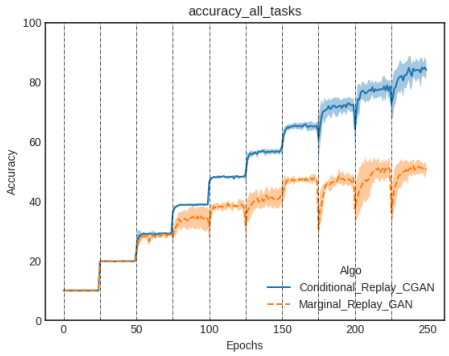
6.5.3 Conditional replay can be run at constant time complexity

A very important point in favour of conditional replay is run-time complexity, as expressed by the number of samples that need to be generated each time a new task is trained. Since the generators in marginal replay strategies generate samples regardless of class, the distribution of classes will be proportional to the distribution of classes during the last training of the generator, which leads to an unbalanced class distribution over time, with the oldest classes being strongly under-represented (see Fig. 6.7). This is avoided by increasing the number of generated samples over time for marginal replay, leading to a balanced class distribution (see also Fig. 6.7) while vastly increasing the number of samples. Conditional replay, on the other hand, can selectively generate samples from a defined class, thus constructing a class-balanced dataset without needing to increase the number of generated samples over time. In the interest of accuracy, it can of course make sense to increase the number of generated samples over time, just as for marginal replay. This, however, is a deliberate choice and not something required by conditional replay itself.

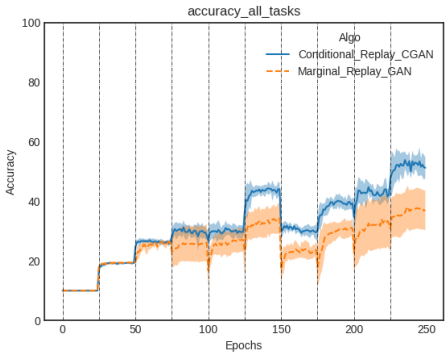
6.5.4 Marginal replay vs Conditional Replay performances

From Fig. 6.8, it can be observed that marginal replay outperforms conditional replay by a small margin. This comes at the price of having to generate a large number of samples, which will become unfeasible if many classes are involved in the retraining.

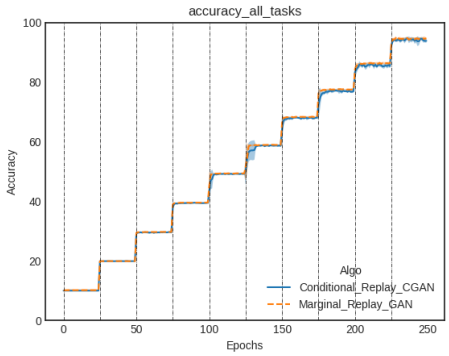
The results of Fig. 6.12 show that conditional replay is superior to marginal replay when generating fewer samples at each task (more precisely: $0.1tN$ samples instead of tN, for task t and number of new samples per task N). This can be understood quite easily: since we generate only $0.1tN$ samples instead of tN samples at each task, marginal replay produces an unbalanced class distribution, see Fig. 6.7, which strongly impairs classification performance. This is a principal advantage that conditional replay has over marginal replay: generating balanced class distributions while having much more control over the number of generated samples.



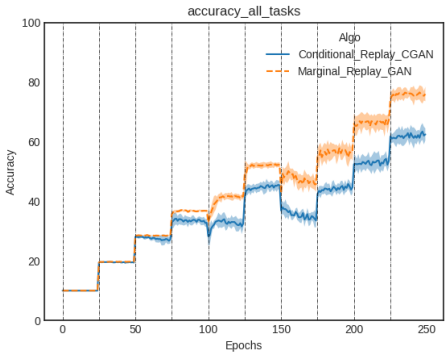
(a) Unbalanced MNIST Disjoint



(b) Unbalanced Fashion Disjoint



(c) Balanced MNIST Disjoint



(d) Balanced Fashion Disjoint

Figure 6.12: Comparison between conditional and marginal replay accuracy. We compare final accuracy when the ratio between size of old task and size of new task is 1 (balanced) or 1/10 (unbalanced, factor was chosen empirically).

6.6 Discussion

In this chapter, we have seen that to ensure the best performance of generative replay, and in replay methods in general, the most straight forward method is to replay samples while taking care of balancing the instance of classes.

Even if balancing the classes offer the best performing results, it is not satisfying in term of computation, indeed since we should balance classes, the amount of computation needs grows linearly with the number of past classes to learn sequentially. It becomes then interesting to look for a solution to reduce the amount of replay needed to reduce the total computational cost. In our results, we show that conditional models are more likely to learn continually if the number of samples is reduced. It would be interesting to also study the impact of balancing the loss instead of the replay to see if we could also improve the computational cost. It might also be interesting to sample more data but processing only a subset of them that could maximize the memorization as proposed in [5] or find smarter methods of sampling.

While one might argue that MNIST and FashionMNIST are too simple for a meaningful evaluation, this holds only for non-continual learning scenarios. In fact, recent articles [186] show that MNIST-related tasks are still a major obstacle for most current approaches to continual learning under realistic conditions. So, while we agree that MNIST and FashionMNIST are not suitable benchmarks in general anymore, we must stress the difficulty of MNIST-related tasks in continual learning, thus making these benchmarks very suitable indeed in this particular context. The use of intrinsically more complex benchmarks, such as CIFAR,SVHN or ImageNet is at present not really possible since generative methods are not really good enough for replaying these data [135] (see or experiments with CIFAR in Chapter 5).

An interesting point is that the disjoint type tasks pose enormous problems to conventional machine learning architectures, and therefore represent a very useful benchmark for continual learning algorithms. As pointed in the Chapter 4, if each task contains a single visual class, training them one after the other will induce no between-class discrimination at all since every training step just "sees" a single class. Replay-based methods nicely bridge this gap, allowing continual learning while allowing between-class discrimination. In our opinion, any application-relevant algorithm for continual learning therefore must include some form of experience replay.

6.7 Conclusion

In this chapter, we have proposed several ways of performing continual learning with replay-based models and empirically demonstrated (on novel benchmarks) their merit w.r.t. the state of the art, represented by the EWC method. A principal conclusion of this chapter is that conditional replay methods show strong promise because they have competitive performance, and they impose less restrictions on their use in applications. Most notably, they can be used at constant time complexity, meaning that the number of generated samples does not need to increase over time, which would be problematic in applications with many tasks and real-time constraints.

Nevertheless, as noted in Chapter 5 one of the biggest limitations of generative replay is the difficulty to train generative models with real images. Generative models are generally very long to train and suffer from high instability.

Ultimately, the goal of our research is to come up with replay-based models where the effort spent on replaying past knowledge is small compared to the effort of training with new samples. Then,

to minimize the amount of replay necessary the model needs also to limit catastrophic forgetting without replay.

Chapter 7

Replay for Policy Distillation

In the previous chapters, we studied generative replay methods for continual learning on two aspects: the continual learning process of generative models and the generative replays applied to classification. We have seen the successes and failures of these uses of generative models for continual learning. In this section, we study the rehearsal method for reinforcement learning. We apply a replay method to train a robot to learn several tasks sequentially and test them in a real-life setting.

This work is a collaboration with René Traoré, Hugo Caselles-Dupré, Te Sun and Guanghang Cai. It has been published at NeurIPS deep Reinforcement Learning workshop as "DisCoRL: Continual Reinforcement Learning via Policy Distillation" [248]. The original article has been slightly modified to include additional figures and results.

7.1 Introduction

An autonomous agent should be able to acquire and exploit knowledge through its entire life. In a sequence of learning experiences, it should therefore be able to build representations and skills online and be able to reactivate and reuse them later. In this chapter, we focus on a setting where an agent learn skills sequentially using a single model, and use them afterwards.

This challenge is partially addressed by a sub-domain of machine learning called multi-task learning. Multi-task learning [37] studies how to optimize several problems *simultaneously* with a single model. However, when those problem can not be optimized at the same time, but sequentially, we identify the learning setting as a continual learning problem. In this chapter, we address a continual learning problem of reinforcement learning (RL). In this setting, each learning experience is called task, and each task solution is called policy.

To propose a learning setting compatible with a real autonomous agent, we define three simulated robotics tasks to solve sequentially. At each task, the agent need to learn a policy based on a reward function and a RL algorithm.

As presented in Section 2.2.2, RL is a popular framework to learn robot controllers that also has to face the continual learning (CL) challenges. In order to learn a multi-task reinforcement learning policy continually , we use a method called policy distillation [207] that allows to transfer several policies learned sequentially into one in a single model. To validate our approach, we evaluate the final results on the three simulated learning settings but also in a real life settings matching the simulation (Figure 7.1). It is important to note that, at test time, the agent does not have access

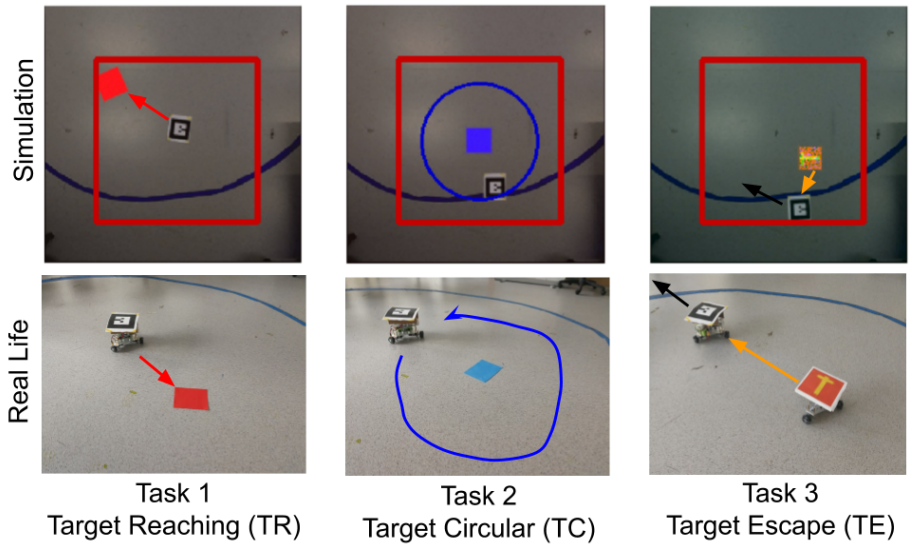


Figure 7.1: Image of the three tasks, in simulation (top) and in real life (bottom) sequentially experienced. Learning is performed in simulation, real life is only used at test time.

to a task label to determine which policy to run, and thus, it needs to figure it out by itself from its observations. As discussed in the previous chapters, it is an important feature for the autonomy of decision making.

Our contribution are:

- We propose **DisCoRL** (*Distillation for Continual Reinforcement learning*): a modular, effective and scalable pipeline for continual RL. This pipeline uses policy distillation for learning without forgetting [76], without access to previous environments, and without task labels. Our results show that the method is efficient and learns policies transferable into real-life scenarios.

- We explore various sampling strategies for policy distillation and compare their performances for task transfer.

The chapter is structured as follows: Section 7.2 introduces related work, Section 7.3 details the methods utilized, Section 7.4 describes the robotics setting and tasks, Section 7.5 presents the experiments performed, and Section 7.6 concludes with future insights from our experiments.

7.2 Background

In this section, we present some background material about multi-task reinforcement learning as well as reinforcement learning in continual learning and robotics.

7.2.1 Multi-task RL

The objective of Multi-task learning (MTL) [37] is to learn several tasks simultaneously; generally by training tasks in parallel with an unique model. Therefore, multi-task RL aims at constructing one single policy that can solve a number of different tasks. Note how in classification this problem is quite simple, as data from all tasks just have to be shuffled randomly and can then be learned all together at once. However, in RL environments, data is sampled on sequences that can not be shuffled randomly with all other environments because the environments are not accessible simultaneously. Learning multiple tasks at once is thus more complicated.

Policy distillation [207] can be used to merge different policies into one single module/network. This approach uses two models, a trained policy (the teachers) to annotate data with soft-labels, and a model to learn from the former (the student). The student is trained in a supervised manner with the soft-labels. The soft-annotation helps the student to learn faster than the teacher did [79]. Policy distillation can then be used to learn several policies separately and simultaneously, and distill them into a single model as in the *distral* algorithm [244]. In our approach, we also use distillation but we do not keep the teacher model. We just label a set of data and then we don't need to keep the teacher model anymore and we can delete it to save memory space. Furthermore, tasks are learned sequentially, and not simultaneously. Other approaches such as SAC-X [196] or HER [8] take advantage of Multi-task RL by learning auxiliary tasks in order to help learning a main task. This approach is extended in the CURIOUS algorithm [48] which selects tasks to be learned that improve an absolute learning progress metric the most.

7.2.2 Continual Learning

We present here a brief recall of Chapter 3 on continual learning for reinforcement learning.

Continual learning (CL) is the ability of a model to learn new skills without forgetting previous knowledge. It is in many ways similar to multi-task learning but with the difference past task can not be accessed directly. In our context, it means learning several tasks sequentially and being able to solve any of the learned tasks at the end of the sequence.

In continual RL, several approaches have been proposed, such as the use of *Progressive Nets* in [208], *Elastic Weight Consolidation (EWC)* [121], *Progress And Compress* (P&C) [217], or *CRL-Unsup* [147]. However, they either need a task indicator at test time to choose which policies to run or, they have some hyper-parameter difficult to tune during a continual learning training, such as the importance of the Fisher information matrix in EWC. Our method does not add any new hyper-parameter to tune during the sequence of tasks and does not need a task label at test time.

7.2.3 RL in Robotics

As discussed in Chapter 3, applying RL to real-life scenarios such as robotics is still a major challenge.

One of the main problem in this setting is that sampling data, and a fortiori learning, is costly. Therefore sample efficiency and stability in learning are highly valuable. One common approach to reduce training cost, is training policies in simulation and then deploying them in real-life hoping that they will successfully transfer, considering the gap in complexity between simulation and the real world. Such approaches are termed *Sim2Real* [85], and have been successfully applied [45, 159] in different scenarios. One of these approaches is Domain Randomization [246], which we use in this chapter. This technique trains policies in numerous simulations that are randomly different

from each other (different background, colors, etc.). Using this technique, the transfer to real life is easier.

Another method we also exploit is to first learn a state representation [136] to compress the observation into a low dimensional embedding and secondly learn the policy on top of this representation. This method helps to improve sample efficiency and stability of RL algorithms [189] and thus can make them directly applicable in real life.

Others have tried to train a policy directly on real robots, facing the hurdle of the lack of sample efficiency of RL algorithms. SAC-X [196] is one example that takes advantage of multi-task learning to improve efficiency, by simultaneously learning the policy and a set of auxiliary tasks to explore its observation space - in search for sparse rewards of the externally defined target task.

In the literature, most approaches focus on the single-task or simultaneous multi-task scenario. In this chapter, we attempt to train a policy on several tasks sequentially and deploy it in real life by combining policy distillation, training in simulation and state representation learning.

7.3 Approach

In this section we present our approach towards continual reinforcement learning for a sequence of vision based tasks. We assume that observations visually allow to recognize the current task from other tasks. We first explain how we learn a single task by combining state representation learning (SRL) [136] and reinforcement learning (RL), then how each task is incorporated in the continual learning pipeline. Finally, we present how we evaluate the full pipeline.

7.3.1 Learning one task

Each task i is solved by first learning a state representation encoder E_i in order to compress input images into a representation of the important underlying factor of variation. This step allows to reduce the input space for the reinforcement learning algorithm and makes it learn more efficiently [189]. To train this encoder, as shown in Fig. 7.2 (left), we sample data from the environment Env_i with a random policy. We call this dataset $D_{R,i}$. $D_{R,i}$ is then used to train the SRL model composed of an *inverse model* and an *auto-encoder*. The inverse model is trained to predict the action a_t that led to transition from state s_t to s_{t+1}, both extracted from respective observations o_t and o_{t+1} by the auto-encoder using E_i. The auto-encoder is additionally trained to reconstruct the observations from the encoded states. The architecture is motivated by the results from [189], and illustrated in the Figure 7.3.

Once the SRL model is trained, we use its encoder E_i to provide features as input of a policy π_i[1] trained using RL. We also experimented to learn the policy directly in the raw pixel space but, as shown in [189], it was less sample efficient.

Once π_i is learned, we use it to generate sequences of on-policy observations with associated actions, which will eventually be used for distillation (Fig. 7.2, right). We call this the distillation dataset D_{π_i}. We generate D_{π_i} the following way: we randomly sample a starting position and then let the agent generate a trajectory. At each step we save both the observation and associated action probabilities. We collect the shortest sequences maximizing the reward for an episode. We also experiment to generate D_{π_i} with a regular sampling and a random policy but annotated with π_i to compare results, as detailed in Section 7.5.1.

[1] Architecture available at: https://github.com/araffin/srl-zoo/blob/438a05ab625a2c5ada573b47f73469d92de82132/models/models.py#L179-L214

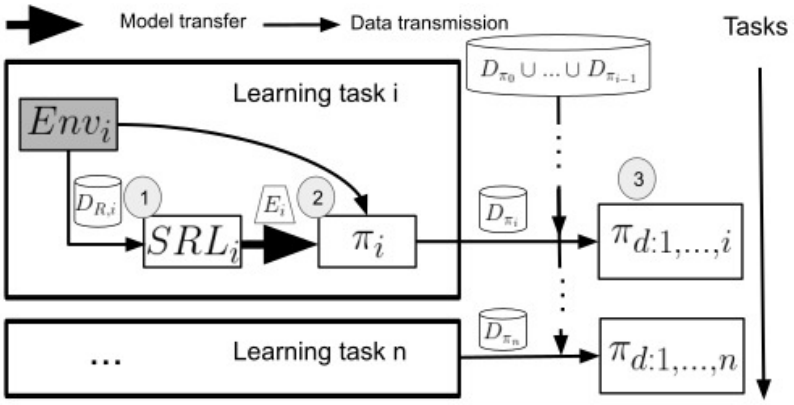


Figure 7.2: Overview of our full pipeline for Continual Reinforcement Learning. White cylinders are for datasets, gray squares for environments, and white squares for learning algorithms, whose name corresponds to the model trained. Each task i is learned sequentially and independently by first generating a dataset $D_{R,i}$ with a random policy to train a state representation with an encoder E_i with an SRL method (1), then we use E_i and the environment to learn a policy π_i in the state space (2). Once trained, π_i is used to create a distillation dataset D_{π_i} that acts as a memory of the learned behaviour. All policies are finally compressed into a single policy $\pi_{d:1,..,i}$ by merging the current dataset D_{π_i} with datasets from previous tasks $D_{\pi_1} \cup ... \cup D_{\pi_{i-1}}$ and using distillation (3).

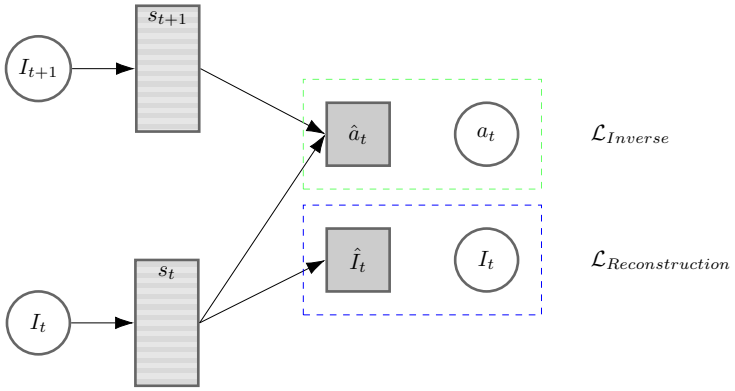


Figure 7.3: *SRL Combination* model: combines the prediction of an image I's reconstruction loss and an inverse dynamics model loss in a state representation s. Arrows represent inference, dashed frames represent losses computations, rectangles are state representations, circles are real observed data, and squares are model predictions; t represents the timestep.

From each task we only keep dataset D_{π_i}. As soon as we change task, $D_{R,i}$ and Env_i are not available anymore. D_{π_i} is split into a training set and a validation set.

7.3.2 Learning continually

Learning policies independently ensures that learning a new policy will not degrade the previous policies. Nevertheless, it does not prevent forward transfer between tasks, since models are initialized with weights from the previous task.

In order to learn continually, we adapt policy distillation [207] to a continual learning setting. The distillation consists of training a student policy to imitate a teacher policy. In our case, a student model learns from a teacher policy the action probabilities associated to each observation. Each dataset D_{π_i} allows to distill the policy π_i (the **teacher** model) into a new network $\pi_{d:i}$ (the **student** model). In classic distillation, both data and models are saved , however saving just soft-labeled data is a lighter solution adapted to a continual setting.

With the aggregation of several distillation datasets D_{π_i}, we can distill several policies into the same network that can achieve all tasks (Fig.7.2, bottom right). By extension of the previous nomenclature, we denote $\pi_{d:1,..,n}$ a model where policies π_1 ... π_n have been distilled in. When distilling all policies into the student, we select our best models with early stopping using the validation set of D_{π_i}, and test later in simulation and in real life settings.

Since we assume that observations visually allow to recognize the current task, $\pi_{d:1,..,n}$ is able to choose the right action for the current task without a task indicator.

The method, termed *DisCoRL* for *Distillation for Continual Reinforcement learning*, allows to learn continually several policies while minimizing forgetting. Regarding scalability, saving data from all past experiments may not look ideal if there is a high number of tasks. However, this solution is highly effective for remembering and letting the reinforcement learning algorithm be absolutely free to learn a new policy without regularization. It is worth mentioning that RL is the real bottleneck in the whole process: Dataset D_{π_i} contains approximately 10k samples per task, which allows to perform the distillation quickly, relative to how long and computationally expensive RL is (few minutes needed to learn $\pi_{d:i}$ while several hours are needed to learn π_i). Thus, in this context, it is better not to curb RL with regularization. Indeed, as will be explained in Section 7.5.4, we tried several regularization based approaches that were not successful.

If we describe the experiments as proposed in the framework presented in Chapter 3, then we are in a reinforcement learning setting, multi-task (MT) scenario. We have 3 disjoint tasks with non-iid data and we have an integer oracle task label for training but not for testing (learning labels). Our setting fall in the NIC (New Instances and New Concepts) content update type for each task. Our approach can be classified into the rehearsal family of approaches where memory is saved as data points. The growth of memory is linear per number of tasks and the growth of computation is less than linear growth. Indeed, the computation is mostly spent on learning the tasks using RL and not much on the continual learning model.

7.4 Experiments

We apply our approach to learn continually three 2D navigation tasks applicable in real life. The software related to our experimental setting is available online[2].

[2] https://github.com/kalifou/robotics-rl-srl

7.4.1 Robotic setup

The experiments consists of 2D navigation tasks using a 3 wheel omni-directional robot similar to the 2D mobile navigation in [190]. The input image is a top-down view of the floor and the robot is identified by a black QR code. The room where the real-life robotic experiments are performed is lighted by surroundings windows and artificial illumination and is subject to illumination changes depending on the weather and time of the day. The robot uses 4 high level discrete actions (move left/right, move up/down in a Cartesian plane relative to the robot) rather than motor commands.

We simulate the experiment to increase sampling and learning speed. The simulation is performed by artificially moving the robot picture inside the background image according to the chosen actions. We use domain randomization [246] to improve the stability and facilitate transfer to the real world: during RL training, at each time-step, the color of the background is randomly changed.

7.4.2 Continual learning setup

Our continual learning scenario is composed of three similar environments, where the robot is rewarded according to the associated task (Fig. 7.1). In all environments, the robot is free to navigate for up to 250 steps, performing only discrete actions within the boundaries identified by a red line. Each task is associated to a visual target, which color depends on the task. This way, the controller can automatically infer which policy it needs to run and thus, does not need task labels at test time.

- **Task 1.** The task of environment 1 is named Target Reaching (TR). The robot gets at each time-step t a positive reward $+1$ for reaching the target (red square), a negative reward -1 for bumping into the boundaries, and no reward otherwise.
- **Task 2.** The task of environment 2 is named Target Circling (TC). The robot gets at each time-step t a reward R_t defined in Eq. 7.1 (where z_t is the 2D coordinate position with respect to the center of the circle) designed for agents to learn the task of circling around a central blue tag. This reward is highest when the agent is both on the circle (red (first) square in Eq. 7.1), and has been moving for the previous k steps (blue, second square). An additional penalty term of -1 is added to the reward function in case of bump with the boundaries (last, green square). A coefficient $\lambda = 10$ is introduced to balance the behaviour.

$$R_t = \lambda * \boxed{(1 - \lambda(\|z_t\| - r_{circle})^2)} * \boxed{\|z_t - z_{t-k}\|_2^2} + \lambda^2 * \boxed{R_{t,bump}} \tag{7.1}$$

- **Task 3.** The task of environment 3 is named Target Escaping (TE). Robot A is being chased down by another robot B with an orange tag. Robot B is hard-coded to follow robot A, and robot A has to learn to escape using RL. Robot A gets at each time-step t a reward of $+1$ if it's far enough from robot B, otherwise, if it is in the range of robot B, it gets a reward of -1. Additionally, robot A gets a negative reward -1 for bumping into the boundaries.

All RL tasks are learned with PPO algorithm [216] from *stable baselines* [98] and the same state representation learning (SRL) model, as described in Section 7.3.1. We select the model architecture as in [190] for RL and SRL. The input observations of all models are RGB images of size $224 * 224 * 3$.

7.4.3 Dataset generation

For each task, a dataset of (image, annotation) pairs is created after the teacher has been trained. The images are the agent observations starting from a random point. The trajectory are sampled using the different strategies described in the next section. The annotations are the soft-labels predicted by the teacher.

Each tasks may use slightly different sampling methods. While generating on-policy datasets $D_{\pi 1}$ (see Section 7.3.1) for task 1 (TR), we allow the robot to perform a limited number of contacts with the target to reach ($N_{contacts} = 10$) in order to mainly preserve the frames associated with the correct reaching behaviour. There are no such additional constraints when recording for task 2 (TC) or 3 (TE), the limit is the standard episode length, i.e. 250 time-steps.

Different tasks will lead to different coverage of the space while sampling, therefore we adapted slightly the sampling for task 1 to maximize the samples space coverage, by modifying the sampling end criterion.

7.5 Results

In this section, we present how we select the best strategy for sampling and distilling policy. Then, we use these choices to present our main result: the distillation of three tasks continually into a single policy that can achieve the three tasks both in simulation and real-life. We provide a supplementary video of this policy deployed in real-life on the robot showing the successful behaviours at https://youtu.be/mzUigGWEfbU. We also present the different strategies we tried but that did not work in our setting.

7.5.1 Sampling and Distilling Methods

This section present the different distillation and sampling methods experimented and the results.

Distillation strategies:

Distillation loss minimizes the difference between the student model's output and the teacher model's output for the same input. As in the policy distillation paper [207], we investigate variations of the loss function:

- Mean Squared Error loss:

$$\mathcal{L}_{MSE}(x, y) = \mathbb{E}\left[||x - y||_2^2\right] \tag{7.2}$$

- Kullback-Lieber divergence, and Kullback-Lieber divergence with temperature smoothing:

$$\mathcal{L}_{KL,\tau}(p|q) = \mathbb{E}\left[\text{softmax}\left(\frac{p}{\tau}\right) ln\left(\frac{\text{softmax}(\frac{p}{\tau})}{\text{softmax}(q)}\right)\right] \tag{7.3}$$

We run a performance comparison of the different losses by computing the mean normalized performance of a student policy trained to perform all three tasks (Tab. 7.1). Using the Kullback-Lieber divergence loss function with temperature smoothing with $\tau = 0.01$ is best, and optimizing the temperature parameter yields a small performance boost. This result is coherent with [207] where they reach the same conclusion.

Distillation loss	Student performance (± std)
MSE	0.71 (± 0.22)
KL ($\tau = 1$)	0.76 (± 0.14)
KL ($\tau = 0.1$)	0.68 (± 0.18)
KL ($\tau = 0.01$)	**0.77 (± 0.13)**

Table 7.1: Mean normalized performance[3] of a student policies trained with distillation using 4 different loss functions. The student policy is trained to perform all three tasks. Kullback-Lieber divergence with $\tau = 0.01$ performs best.

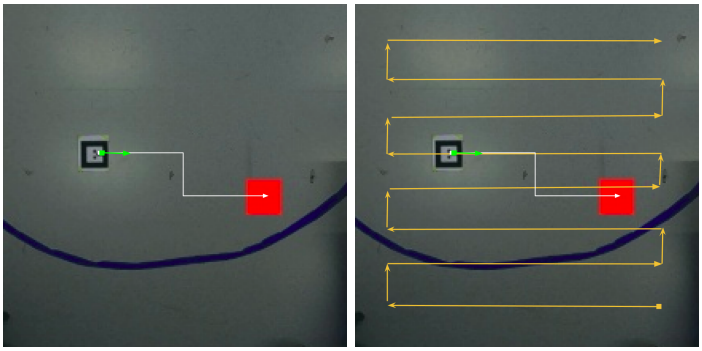


Figure 7.4: Representation of data sampling strategies to distill the teacher policy. Left, on policy sampling. Right, grid sampling.

Data sampling strategies:

We evaluate the effect of three different sampling strategies to create D_{π_i} for policy distillation. Data sampling is a key component as the sampled dataset should be as small as possible but contain sufficient information for student model training. The strategies involved for data generation are:

- *On-policy generation (Fig.7.4, left)*: We start an episode from a random point, then at each timestep t, we collect an observation o_t and perform the action $a_{\pi_i,t}$ of the teacher policy. D_{π_i} is thus composed of tuples $(o_t, p(a_{\pi_i,t} \mid o_t))$, with $p(a_{\pi_i,t} \mid o_t)$ the action probability associated to the action $a_{\pi_i,t}$ taken by the teacher, i.e., a *soft label*, since we use the Kullback-Lieber divergence loss.
- *Off-policy generation from a grid walker (Fig.7.4, right)*: at each time-step t, we collect an observation o_t by performing an action $a_{grid,t}$ of a *grid walker* exhaustively exploring the space of the arena. However, for each o_t we save the probability of action $p(a_{\pi_i,t} \mid o_t)$ that would have been taken by a teacher policy. The goal of this strategy is to provide a more exhaustive sampling of the space of robot positions.
- *Random Walker*: We start an episode from a random point, then at each time-step t, we realize random action for 200 time-steps. This method is proposed as a baseline.

Performance of policies distilled using such strategies (see Fig. 7.6) show that *on-policy generation* (i.e., demonstrations) suffice to reproduce performance close to those of teacher policies on

every task individually, with reasonable stability. In particular cases, see Fig. 7.6 for task TC, this strategy even provides a small boost in performances in the student policy over the teacher policy.

However, using *off-policy data generation from a grid walker* for distillation results in either unstable or poorly performing policies, especially in tasks defined by a reward function requiring the agent to move actively (*TC* task, blue part of eq. 7.1) or anticipate the behaviour of another agent (TE task). In this case, the resulting policy reaches the performances of a lower-bound baseline obtained by distilling from trajectories of an untrained policy (see *Student on off-policy data with a random walker* in fig. 7.6), i.e. from a policy with random weights with input in the raw pixels' space.

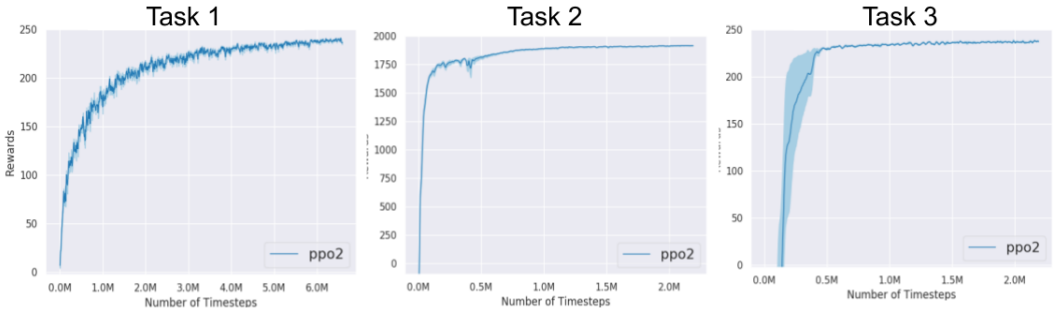


Figure 7.5: Mean and standard error of rewards during RL learning of each task separately. Each task is learned using the same type of SRL model (SRL Combination), trained on each environment. All three tasks are mastered within roughly 2M time-steps.

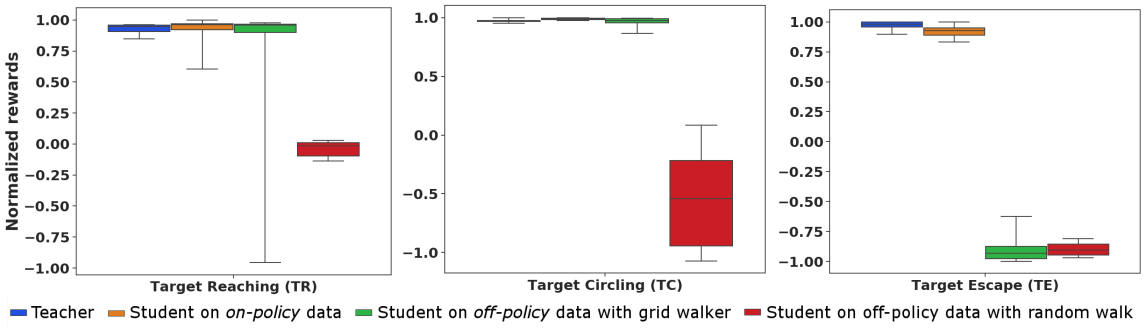


Figure 7.6: Efficiency (normalized rewards w.r.t the best teacher performance) of policies distilled on 8 seeds using various data sampling strategies for each task separately. Each evaluated policy is distilled on 15k tuples of sampled observations and action probabilities, for 4 epochs (see criteria of stopping in Section 7.3.2 and Figure 7.5).

We performed a more explicit evaluation of distillation in the task 2 (Target Circling (TC)). While we train a policy using RL, we save the policy every 200 episodes (50K), and distill it into a new student policy which we test. This is illustrated in Fig. 7.7. Both curves are very close, which indicates that policy distillation enables to reproduce the skills of a teacher policy regardless of the teacher's state of convergence on the evaluated task. Moreover, distillation is able to transfer

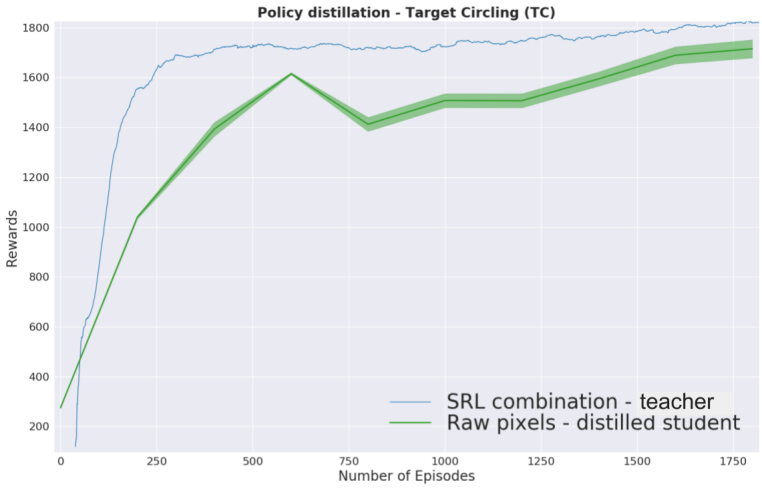


Figure 7.7: Demonstration of the effectiveness of distillation. Blue: RL training curve of an SRL based Policy (SRL Combination) on the target circling (TC) task. Green: Mean and standard deviation performance on 8 seeds of distilled student policy. The teacher policy in blue is distilled into a student policy every 200 episodes (1 episode = 250 time-steps).

knowledge from teacher policy into a student using a small number of observations, i.e only 15k samples (w.r.t. the volume of samples required to learn the teacher policy, see Fig. 7.5).

7.5.2 Evaluation of each task separately

We perform two evaluations on our final models. Our first evaluation is the performance of the final policy on the simulated environment. This evaluation can then be compared with the performance of each teacher policy. For the second evaluation we test if the policy is robust to the reality gap and can be adapted into a real life scenario. The simulation is voluntary close the real life setting but the reality gap is notoriously problematic.

Before moving to a continual setup, we tested if it is possible to solve each task separately using distillation. Therefore, we evaluate distillation process for each task separately. The challenges, is not only the ability to distill knowledge, but also the ability to know when a policy has been distilled, i.e. properly learned by the student. Indeed, due to the hypothesis of real continual learning settings, access to previous environments is not possible. Then, the challenges is to find a proxy task that can help indicate when early stopping of the policy distillation can be applied. That proxy task or signal should be different from the reward achieved in previous environments, which is no longer available. In our experiments, we found empirically that all policy could be distilled and that limiting training to a small number of epochs, i.e $N = 4$, guarantee policy learning.

7.5.3 Main result

In this section, we present our final results. We used *on policy* data generation and training using KL-divergence loss with $\tau = 0.01$, as described in Section 7.5.1. In figure 7.8, we show box plots

over 10 episodes of reward performances for teacher policies in each task, and for the distillation of the same three teachers into a single student using DisCoRL. Each policy is evaluated in simulation and also in real-life on the robot. As a reference, we also show the performance of a random agent in each task. Our approach is effective in a continual reinforcement learning setting: the performance of teachers and student are similar.

More precisely, there are two main challenges to overcome in our setting: learning a behaviour via distillation by using only a limited number of examples, and the reality gap which can notoriously [246] introduce variations that may lead the policy to fail. Fig. 7.8 demonstrates the efficiency of our approach at overcoming both of these issues: only a small fraction of performance is lost from teacher to student, and from simulation to reality. We can see that the single student distilled policy achieves close to maximum rewards in all tasks, in real-life.

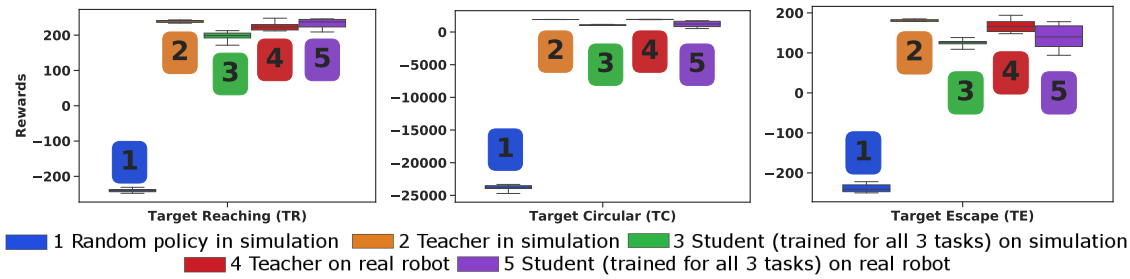


Figure 7.8: Main result: distillation in a continual learning setting of three teacher policies into a single student policy. The resulting policy is able to perform all three tasks both in simulation and in the real world, while minimizing forgetting.

7.5.4 Negative results

While distillation is effective for policy transfer, we also tested other alternatives worth mentioning.

Elastic Weight Consolidation (EWC)

EWC [121] was implemented as a continual learning baseline to compare with the distillation method. EWC has the appealing advantage of not re-using any data from previous tasks. However, in all cases we found the method unsuccessful.

Tuning the λ parameter that controls the trade-off between weight protection and learning the new task showed that either λ is too low and catastrophic forgetting happens, or λ is too high and nothing new is learned (i.e., the full network is frozen). A λ value providing a proper balance in between both effects could not be found for such sequential tasks to be learned.

Progress and Compress (P&C)

P&C [217] was tested but as EWC, we had problems with the importance factor λ and we where not able to learn three policies into a single model with this method.

Adding task labels for distillation

Even if all tasks contain a visually differentiating identifier, they remain visually similar. In cases, we found that a distilled policy trained to perform well on several tasks can mix up tasks and thus not perform adequately. Hence, either adding tasks labels directly, or adding a module in the network that predicts the task label could be a way to improve the efficiency of distillation. However, none of the approaches were successful in practice, yielding the same results with or without task labels.

Gumbel-Softmax action sampling for the student

This trick [103] allows to sample from a categorical distribution using a softmax output layer. It has proven to be useful for action sampling in policy learning [216]. However, in our case we saw no improvement over a simple argmax strategy for action sampling when we used it on the student policy at test time.

7.6 Discussion and Future Work

Even if we believe this work proposes a stable and scalable framework for continual reinforcement learning, several possibilities for improvement exists. For example, we could have not only a policy learned in a continual way, but also the SRL model associated. We would need to update the SRL model as new tasks are presented sequentially. One possible approach would be to use Continual SRL methods like S-TRIGGER [38] or VASE [2]. Moreover, we would like to optimize more the memory needed to save samples by reducing their number and their size.

Moreover, training policies on real robot experiences without the use of simulation would be desirable. However, at the moment, this is more a RL challenge than a CL challenge. One promising approach would be to use model-based RL while learning the state representation learning (SRL) model to improve sample efficiency. Though, nowadays approaches still do not offer solutions working in a reasonable amount of time.

7.7 Conclusion

In this chapter, we presented DisCoRL, an approach for continual reinforcement learning. DisCoRL is a simple yet efficient method for continual multi-tasks reinforcement learning. It performs independent learning of different policies without disabling forward learning transfer with a fast merging policy methodology. Moreover, at test time DiscoRL can run all the policies with a single model without the supervision of a task label.

The method consists of summarizing sequentially learned policies into a dataset to distill them into a student model. It allows to learn sequential tasks in a stable pipeline without forgetting. Some loss in performance may occur while transferring knowledge from teacher to student, or while transferring a policy from simulation to real life. Nevertheless, our experiments show promising results in simulated environments and real life settings.

Chapter 8

Discussion

In the previous chapters, we presented a global overview of continual learning, the strength of replay methods and the analysis of generative replay methods. We focused on the study of generative models in a CL setting and the application of generative replay to classification problems. We also experimented with the rehearsal method for continual multi-task reinforcement learning.

In this chapter, we would first like to discuss continual learning research in a more global picture. Hence, we will discuss popular objectives in continual learning. We also try to disentangle potential use cases and present application scenarios for continual learning. Secondly, we discuss the work done in this thesis and the choices made. Then, we introduce a few continual learning pitfalls that should be avoided to perform healthy continual improvements.

8.1 Rethinking Continual Learning Objectives

Continual learning is a vast research domain with very ambitious objectives. The general aim is to learn from non-static data-sources but more particularly to learn from the real world. In this section, we first present the popular objectives of continual learning. Then we introduce several use cases we believe are the main long term objectives of continual learning.

8.1.1 Popular objectives of continual learning

Several continual learning papers [129, 69, 141, 4] present general desiderata for continual learning. Those desiderata are presented as characteristics that continual learning algorithms should have. The most common ones are:

- **Maximize Final performance:** Algorithm should have the best performance possible at test time.
- **Learning without forgetting:** The ability to learn sequentially and incrementally knowledge or/and skills.
- **Graceful forgetting:** Remembering the essential only.
- **Detecting concept drift:** The ability to detect when the data distribution changes to avoid forgetting.

- **Storage-Free Continual Learning:** Avoiding the storage of raw data.
- **Efficient learning / Few shot learning :** Grasping new concepts thanks to only few data points.
- **Applicable algorithms:** Algorithms able to solve problems with reasonable constraints such as limited memory or power consumption.
- **Transfer between tasks:** The ability to improve a specific knowledge/skill by learning another one (forward learning and backward learning).
- **Transfer between settings:** An algorithm able to learn in an environment/continuum A should be able to learn in a similar environment/continuum B.
- **Execution time:** The ability to learn in a limited time, at best near to on-line.
- **Reproducible results:** Being able to reproduce results by an independent party.

Many of those characteristics are quite easy to evaluate. The metrics proposed in Section 3.6.2 associated with the right benchmarks make it possible to evaluate most of them or, at least, it makes it possible to compare two algorithms. Unfortunately, most of them are not intrinsically interesting for continual learning and they should be put in a more long term perspective to understand what is important to be addressed. For example, the desiderata of transfer between tasks is usually not the true objective. The true objective is either to reduce the execution time or improve the final accuracy. The transfer might be an answer to those objectives but the transfer is finally not exactly a goal but just a way to reach another goal. Similarly, graceful forgetting or few-shot learning are interesting, for us, only if they improve other objectives such as final performance or execution time.

By analyzing the desiderata again we find that the most important one for continual learning are the same as for machine learning:

1. Maximize final performance (and generalization)
2. Applicable algorithms (being able to adapt algorithms to specific constraint on power consumption / memory / training time / inference time)
3. Reproducible results

Indeed, the difference between classical machine learning and continual learning is not the objective or criterion to optimize but the hypothesis on the input data (iid vs non-iid).

Nevertheless, the difference in input data might lead to supplementary constraint to fulfill the global desiderata and overcome some needs such as:

- Minimizing memory
- Minimizing training time
- Minimizing computation power
- Autonomous inference (label free inference)

The tools to answer those desiderata and constraints might be provided by other research domains such as:

- Few-Shot learning
- Transfer (Backward / Forward ; upon tasks / settings)
- Knowledge distillation (for Graceful forgetting)
- Learning from sparse labels
- Detecting concept drift
- Minimizing storage
- Sparsity
- [...]

Hence, progress in those research domains might be useful or necessary to leverage continual learning problems but it always depends on the targeted problem. Nevertheless, it is safer to keep apart desiderata from side constraints and from other research field to keep track of what we are trying to do.

We can also mention the bio-inspiration as a side objective. The biological agents provide many ideas to solves continual learning problems. However, as the other side objective, it is not what continual learning aims at solving. For example, even if biological agents don't store raw data, it does not mean that continual learning should not do it either. In many cases, saving raw data helps, as shown in Chapter 7. Creating false prohibition is deleterious for continual learning. The aeroplane is a working solution illustrating that biological agents do not necessarily have the optimal solution for our needs.

In the same spirit of finding the real goal of continual learning, in the next section, we present what we believe are the potential type of use cases for continual learning.

8.1.2 Potential use cases

The long term objectives are linked to the potential use cases of continual learning. Depending on them, the side objectives described in the previous section are not expected to be optimized in the same way. The application cases can be numerous and quite different. Therefore, to develop specialized approaches for different scenarios, it is necessary to identify the important differentiating factors. We present here a possible classification of potential use case scenarios that are more interesting to target specifically than continual learning in general.

- **Incremental Learning** The objective is to learn new knowledge or/and skills without forgetting. In this case, the goal is not to improve knowledge but only to incrementally grow a set of knowledge. This is typically the case of a trained classification model that should learn a new class without forgetting the previous ones with very limited access to data from the past. This case is the most straightforward to describe, the primary criterion that matters is learning without forgetting. The past learning experiences are not supposed to be accessible again and consequently, everything forgotten is lost forever. In this case, since the forgetting

should be minimized as much as possible, it looks more acceptable to not be too restrictive on the memory needed or the efficiency of learning or computation.

Example: **Adding classes to a classifier**

A company, let's call it AItowardAGI, needs a trained model for a classification purpose. A solution is to buy a trained classification model from a company, training4U, that has access to more computation and more data. However, AItowardAGI has some personal data and would like to improve the model on this personal data without damaging initial models. Then, the company will need an incremental learning method to learn without forgetting and improve the model with new classes. We hope that AItowardAGI has read Chapter 4 of this thesis and therefore they know that they will not be able to improve the model if they don't ask for some more information about the initial training data from training4U company.

- **Lifelong Learning** Lifelong learning consists of learning a never ending task, there are few variations in the tasks but the agent should always improve and be able to handle more and more of those little variations. It could be essential for applications where smart agents need to keep improving and gain efficiency from experiences on a specific task. In this case for example, forgetting is not unconditional, it is somehow acceptable to forget if globally it allows to improve the performance. However, as the agent learns at the same time it is used, it is expected to learn as fast as possible to adapt to changing situations.

  Example: **Gift wrapping robot**

  A company wants to build robots that wrap gift papers. They want to send those robots in shops to wrap gifts automatically. However, each year the gifts are different and each robot needs to be updated. Moreover, shops sell objects of different sizes and shapes. Therefore, if all robots had a learning algorithm that could adapt to new sizes and shapes without selling company intervention, it would be very practical. This company could use an algorithm similar to the one used in Chapter 7.

- **Multi-task agent** Multi-task agent is a mixture of the incremental learner and the lifelong learner. Similar to the lifelong learner, it is in a never ending situation however it has several different tasks to learn and improve in its lifelong learning curricula. In the multi-task learning, as for lifelong-learning, forgetting can be acceptable in order to improve the global performance, however improving on one task should not deteriorate too much another one to keep progress positive.

  Example: **Periodic improvement**

  The training4U company sells pre-trained classification models. So at any time, they should be able to sell a final trained model, however, they regularly receive new data about new or existing classes that could help to improve their models. Training again from scratch is too expensive and they will need a continual process to improve the models on new data without damaging current knowledge. Then, they need multi-task learning methods to keep improving existing classes and still be able to add new ones. We hope they also read the Chapter 4 of this thesis, and therefore they will have developed a correct remembering process for their models.

- **Autonomous agent**

  The autonomous agent is a multi-task agent which can additionally set its own objectives and has the capacity to explore and understand without clear tasks. This agent possess

curiosity [176, 213]. The curiosity is a self-motivated objective leading an agent to explore its environment and the possible interactions. It helps the agent to improve on past and new tasks and enable it to eventually anticipate future tasks. Moreover, the autonomous agent should be able to create its own innovative solution to problems with limited supervision, as in Open Ended Learning [61]. This kind of agent should make the best use of the resource that is available to it. It also has to be curious, creative and have some kind of survival reflex to evolve without auto-destruction.

Example: **Robots on a Foreign Planet**

A space exploration program would like to send a robot to a very far exoplanet for discovery. The problem is that any communication can take weeks or years to be sent, so communication is very limited and the robot needs to be autonomous to survive in this unknown environment. Moreover, we would like the robot to autonomously explore and adapt to the environment to complete future tasks faster. For example, the robot could automatically learn to recognize the type of grounds where it can pass to be able to move better later. The robot then needs a continual learning algorithm to complete its mission, explore and survive.

For each case, we described what we believe are the most emblematic objective to optimize in the potential use cases. However, depending on the use case more objectives may have to be optimized like computation power which could be restricted in certain situations like in embedded platforms. In any case, an approach would be more useful in the global research effort if it targets long-term or/and short-term objectives specifically.

To summarize, in continual learning, the essential expected ability is to be able to improve a model based on new data, either by adding new concepts or by improving/strengthening already known concepts. A second important notion is adaptability to various types of supervision signals to learn, at best an algorithm should be able to learn at least "something" whatever the sparsity of supervision signal or its type (reward or label). Therefore, continual learning is somehow the science of learning autonomously and might have links to the auto-machine learning research field [72].

8.2 Discussion on the thesis choices

In this section, we discuss the different choices made in this thesis and the conducted experiments.

8.2.1 Replay methods

In this thesis, we presented the use of replay to deal with catastrophic forgetting. Indeed, replay methods have both the advantage to make models able to remember and learn on past tasks. We study it in particular in the context of incremental classes or tasks. We showed that in such settings, regularization and dynamic architecture approaches rely on a task label for inference. Replay methods propose a label-free inference model that can be therefore easily deployed after training.

As presented earlier in the thesis, the assets of replay are:

- **A posteriori understanding:** Past learning experiences can be reinterpreted with current knowledge and favor backward transfer.

- **Task agnostics memorization:** Some part of the memorization process is not affected by the current task and just aims at representing the learning experience data.
- **Test labels agnostic:** The learned model does not need any supervision for inference.
- **Model agnostic:** The memorization process is at least partially not affected by the model architecture.
- **Manageable memory:** The memory is easy to control, if a memory is considered useless, it can be erased by just deleting it or not replaying it anymore
- **Unconditional remembering:** (Rehearsal only) By saving raw data the memorization is theoretically robust to misinterpretation and memory modification.
- **Good on-line capabilities:** (Rehearsal only) Saving raw data is quite fast in comparison to learning a memorization model.
- **Auto-Memorization with self synthesizing models:** (Generative Replay Only) The generative model automatically learns to synthesize data and compress it in its weights.
- **Supplementary understanding of data for memorization:** (Generative Replay Only) The generative models can potentially generalize processed data and share its understanding with the inference model.

The continual learning use case that fits perfectly with the project of this thesis is "incremental learning" (Section 8.1.2), a model that learns different tasks sequentially and should and never forgets. Studying this setting produce results that can be transferred to lifelong algorithms and multi-task continual learning. We believe, as demonstrated in Chapter 4, that in all learning situations, two concepts need anyhow to be confronted to be distinguished and replay might be the only method able to achieve it.

8.2.2 Static Deployment

We can define two type of deployments for continual learning:

- **Static Deployment:** The model is trained and used frozen. The training can still be continued later.
- **Never Ending Learning:** In this case, the training never stops; every new experience can be exploited to learn and be integrated in the model.

Moreover, it might be worth distinguishing two types of potential use. First, the "milestone" use case, the model is trained and it should be ready at some point to be used. We don't care about how it learns as long as at some point in time the model is ready. Secondly, the "always ready" use case, where at any time the model could be used and should be aware of the last data point processed.

We target models with milestone use case and static deployment. Those models can learn in a continual environment and learn concept and skills sequentially until deployment start.

We considered as too ambitious the always ready scenario and we argue that static inference is probably the use of continual algorithms that could be the safest. For the static inference

scenario, after training, the model can be assessed and be deployed without fearing for uncontrolled or adversarial modification of the model. A model that would continue to learn after deployment would be harder to assess and the learning processes are nowadays not stable enough to be deployed.

8.2.3 Task labels

The use of training labels can be seen as a limitation of the approaches. The use of the task label allows targeting only the learning-without-forgetting problems without addressing concept-drift detection.

We think that task labelling is cheap and may significantly help continual algorithms to learn. In many applications of continual learning, we can assume that at training time, the algorithm has a bit of assistance to learn. If no task label is provided, the algorithms have a higher risk to diverge and misunderstand its learning experiences.

8.2.4 Data stream distribution

In the same way we assumed task label available for training, we assume that the learning curriculum is independently and identically distributed by part (except in Chapter 7). This is a clear limitation that should be solved before expecting to tackle real environment settings, however, this setting allows to better understand catastrophic forgetting behaviour and memory processes. Since we know exactly at which moment the model will start forgetting, it is easier to analyse it and address it. The iid by part settings is then very interesting for research purpose.

8.2.5 Classification tasks

Ideally, algorithms can learn on-line new concepts from unprocessed data. Therefore, it is unlikely to have a fully annotated dataset correctly preprocessed to maximize learning easiness. Benchmarks using sparse labelling would therefore be more appropriate and more in the spirit of continual learning. However, the use of fully annotated classification benchmarks makes it possible to focus on continual learning problems rather than having to deals with sparse label problems. Nevertheless, in Chapter 7, we did experiments with reinforcement learning environments that have sparse labels. And we have seen that even if learning tasks from those environments is much harder it does not make the continual learning problems significantly harder.

8.2.6 Evaluations

In this thesis, we evaluate essentially our algorithms with the final performance, we believe that, with the computation cost, it is the most valuable metrics to evaluate CL as discussed in Chapter 3. The computation cost is however often difficult to evaluate rigorously and we did not evaluate it to focus on the final performance.

The MNIST, Fashion-MNIST or KMNIST benchmarks we used are simple. Some experiments have been conducted on Cifar10 and Core50 (not described in the manuscript), however, the results were not stable enough to conclude from the experiments. The main conclusion is that generative models are difficult to train on those datasets in a continual setting. Therefore, it is still too early to expect generative replay to work in continual real-life settings. However, with the rapid progress of generative models we expect generative replay to became a viable solution in complex environments. As discussed for the classification task above, the use of simple datasets makes it possible to get

rid of learning shortcomings of models and only focus on continual learning shortcomings. If it is already difficult to train a model in a classical setting, studying it in a continual setting is limited by the machine learning shortcomings.

8.2.7 Answers to the framework questions

In Chapter 3, we compiled a set of questions that continual learning approaches should answer to have a proper explanation of the setting and the method.

Here, we compile the global answers for those questions concerning the thesis results.

- $\boldsymbol{Q_1}$: *Does some data need to be stored? If yes, how and what for? (e.g. regularization, re-training, validation)?*

  Yes, we store data for model selection purposes (validation set) and in the rehearsal approaches for remembering.

- $\boldsymbol{Q_2}$: *Is the algorithm tuned based on the final performance? I.e. is it possible to go back in time to improve performance?*

  Normally no, but we admit that some hyper-parameters have been tuned empirically, so we did not respect the temporal coherence perfectly in our results.

- $\boldsymbol{Q_3}$: *Are data distributions assumed i.i.d. at any point?*

  Yes, the data distribution is assumed iid by part (iid during each task). Except in reinforcement learning experiments.

- $\boldsymbol{Q_4}$: *Is each task assumed to be encountered only once?*

  Yes, for unsupervised learning, no for the other experiments (even if all tasks are only encountered only once in the reported experiments for clearer evaluation purposes).

- $\boldsymbol{Q_5}$: *Is the continual learning algorithm agnostic with respect to the structure of the training data stream? (e.g. number of classes, numbers of tasks, number of learning objectives...)*

  Yes, the number of classes / tasks can be dynamically changed.

- $\boldsymbol{Q_6}$: *Does the approach needs a pretrained model for the CL setting? If so, what is the new knowledge that needs to be acquired while learning continually?*

  No pretrained model is used for continual learning experiments.

- $\boldsymbol{Q_7}$: *How much available memory does the algorithm require while learning? Does the memory capacity requirement change as more tasks are learned?*

  The model architecture is chosen empirically based on the non-continual performance on classical benchmarks, it stays fixed. Only the output layer can be dynamically changed to add more classes. In rehearsal experiments the memory grow as there are more past tasks.

- $\boldsymbol{Q_8}$: *Is the continual learning algorithm constrained in terms of computational overhead for each learning experience? Does the computational overhead increase over the task sequence?*

  There is no particular constraint, the amount of computation increases at least linearly with the number of same size tasks.

- $\boldsymbol{Q_9}$: *Is the continual learning algorithm agnostic with respect to the data type? (e.g. images, video, text,...)*

  The architecture model is designed for images and their dimension is known in advance.

- $\boldsymbol{Q_{10}}$: *Is the continual learning algorithm able to handle situations where there is not enough time to learn?*

  Not yet for generative replay. For rehearsal, yes since the algorithms just need the time to save data to remember them.

- $\boldsymbol{Q_{11}}$: *In the presence of multiple tasks, is the task label available to the algorithm during the training phase? And during evaluation?*

  Task label is used for training but not for testing.

- $\boldsymbol{Q_{12}}$: *Are all the data labeled? or only the first training set? Can the user provide sparse label/feedback (e.g. active learning) to correct the system errors?*

  In supervised experiments, all data are labeled, the labels are also used for evaluation in unsupervised learning, there are sparse labels for reinforcement learning.

- $\boldsymbol{Q_{13}}$: *What is expected from the algorithm to remember at the end of the full stream? Is it acceptable to forget somehow, when task, context or supervision change?*

  Since the number of tasks in the experiments is not very high, the algorithm is expected to remember everything as much as possible.

8.3 Continual Learning Pitfalls

With regard to the potential use cases from Section 8.1.2, it is important to not fall into pitfalls that do not help to push forward true objectives.

8.3.1 The bias of the future

In opposition to classical machine learning, the temporality in continual learning is essential. A CL algorithm should be prepared for its future without knowing it. Therefore hypothesis should be done about the future. In a chaotic world, it would not be possible to make any plan about the future, fortunately, we are not in a chaotic world and we can have reasonable hypothesis of the future to consider what can likely happen or not.

Nevertheless, we should not use the future to improve learning algorithm. Indeed, in research experiments we can virtually control the temporality of learning experiences and potentially use the future. Thus, we should be cautious to not create causal incoherence.

The main problem is that tuning the parameter at time $t = 0$ based on results at time $t > 0$ is aberrant. Algorithms can not be perfectly designed in a one-shot process, however, if they are designed on one curriculum they should be tested on another one to be valuable.

In the scope of continual learning, the hyper-parameter should be selected only based on the present tasks and past tasks. In a lifelong task, we believe the use of populations of models might be a good solution to deal with hyper-parameters selection. Indeed if we have a population of models with different hyper-parameters we could select the best hyper-parameter in one run.

8.3.2 Spread out objectives

As explain in the Section 8.1, continual learning is a wide research domain with ambitious expectations. However, it is clear that no algorithm can tackle all learning situations. Understandably, algorithms have a limited scope of application, aiming at finding the algorithm that can solve all problems is chimeric.

One of the pitfalls in continual learning is to not specify what kind of setting is targeted. It is probably not possible to find an approach that can solve catastrophic forgetting no matter the subject. Then, to better address continual learning approaches and application generally it is appropriate to specify precisely the goals and scope of a research project. The question presented in Section 8.2.7 and Chapter 3 should allow to specify the essential aspects of a continual learning research project such as the learning subject, the learning setting, the learning objectives and the learning tools.

Therefore, we should target reasonable goals to expect to be able to bring useful solutions and not being lost in a far too big space of exploration.

8.3.3 Scalability: a Double-Edged Sword

In machine learning research, a comment often present in article reviews is "Have you tried on harder problems?". There exist many incentives to try to tackle difficult problems in order to validate a theory or an approach. However, those incentives are just a rule of thumb following the adage "who can do more can do less". However, in machine learning, the successful transfer from one settings to another might have low correlation with the difficulties of tasks.

Then, asking for proof of scalability is probably more a mandatory obstacle to overcome to have recognition than a legitimate request for evaluation. Asking for more baseline or comparisons however might be a better request to estimate the legitimacy and the appropriateness of any approach.

In continual learning (and in other machine learning fields), the MNIST dataset is often used, and reviewers often ask for harder settings rather than for stronger baselines. However, it is not clear if continual learning specific difficulties are correlated with machine learning difficulties. Therefore asking for harder datasets is not necessarily a service for the community. Asking for more datasets can however be legitimate to compare approaches with similar results.

In the next section, we gather a set of recommendations that we believe are important to have in mind for research projects in continual learning.

8.4 Research recommendations

For more concrete indications on what we consider worthwhile checking while creating a CL approach, we suggest a set of recommendations. Those recommendation point out research topics that should be privileged or methodology that should be respected.

Recommendation 1. *On-line capabilities: CL algorithms should adapt to new data as soon as they are available without assumptions like the number of tasks or classes.*

Recommendation 2. *Autonomous inference: CL algorithms should be autonomous at least for inference, therefore they should not assume labeling information at test time.*

Recommendation 3. *Scalability evaluation: In order to provide a proper evaluation of the scalability and continual learning performance, we recommend, as the authors from [69], to evaluate algorithms on more than two tasks.*

Recommendation 4. *Resources evaluation: To be practical, CL systems should evaluate resources consumption as for memory or computation.*

Recommendation 5. *Reporting metrics: We recommend reporting exactly the targeted objectives of the method and report the associated metrics.*

Recommendation 6. *Ablation studies: we recommend reporting ablation studies to motivate as best as possible the different components and choices made in the CL algorithm and identify their importance for the different objectives or tools (learning without forgetting, transfer, few-shot-learning,...).*

Recommendation 7. *Distributional shifts: We recommend to formally describe the mechanism to handle distributional shifts, not only when tasks change, but also among batches where data points conform to different distributions.*

Recommendation 8. *Report precisely and clearly how an approach learns and the assumptions it make, as described in the framework (Chapter 3).*

In this chapter, we discussed continual objectives and how specific criteria can be put in long term ambitions. We presented the global aim of the thesis regarding those objectives and justified the different choices made. We compiled a set of pitfalls that can divert continual learning from its progress and its potential applications and finally proposed a set of recommendation rules for continual learning. In the next chapter, we will conclude this thesis and provide some potential future research directions.

Chapter 9

Conclusion

To conclude this thesis, we will first summarize the contributions presented in this manuscript. Then, we present research directions that could extend this work and improve the understanding and efficiency of replay methods for continual learning.

9.1 Summary of Contributions

The overall aim of the thesis was to study methods able to learn on incremental settings and which do not rely on any supervision to be deployable in real world applications.

First of all, we presented a framework for continual learning, built on top of [146] framework. This framework makes it possible to frame any continual learning approach systematically. It helps to rigorously set out the method, the scope and the evaluation of a CL algorithm to ease the comparison of the methods and transfer from one application to another.

Second, we demonstrate the advantages of replay methods in comparison with regularization and dynamic architecture methodology. We show that in the absence of task labels, the replay is the only method that could learn classification incrementally.

Third, we applied replay methods in unsupervised learning (Chapter 5), supervised learning (Chapter 6) and reinforcement learning (Chapter 6) settings. We experimented, in particular, the generative replay methods and introduced the "Conditional Replay" method for continual learning. We showed that generative replay is agnostic to the test label and the number of tasks. Moreover, the generative model learns to memorize in a potentially more compact way than the initial dataset, it can generate never seen samples and offer its generalization capacity to learn downstream tasks. We also used the rehearsal strategy for multi-task continual reinforcement learning, presenting the *DiscoRL* algorithm. DiscoRL advantages are the unconditional ability to remember thanks to the hard memory process, the independence of individual policy learning and global policy learning without preventing forward transfer. We showed the effectiveness of the method on robots both in simulation and in real-life settings.

Finally, we present an extensive discussion on continual learning ultimate objectives, the choices made in the thesis experiments, the pitfalls of continual learning research and we introduce a list of recommendations which should help to push forward the limits of continual learning.

9.2 Future Research

We presented our contributions to research in continual learning with replay methods. However, many research directions may improve these methods. We mention here several of them: improving sampling methods, improving generative models, detecting concept drift, improving hyper-parameters selection with meta-learning and estimating knowledge retention. Progress in those research direction will push continual learning possibilities forward.

9.2.1 Improving Replay Methods

Even if theoretically the replay method has clear advantages, there are still potential improvements, in particular in the construction of the memory either by improving the validation of generative models for memory replay or by improving the selection criterion for coreset. The data replayed should be representative enough to remember the task and general enough to be used in other tasks. Another research subject that would deserve to be studied is the protection against overfitting memory and insuring a good generalization. Sampling the memory should then be carefully done to find the good trade-off between the benefit of the memory without spoiling it. Smart sampling would also helps to reduce the algorithms computation consumption.

9.2.2 Improving Generative Models

Generative models are promising for continual learning. They theoretically propose a satisfying memorization solution for neural networks. However, in the generative replay framework, the generative model is a serious bottleneck in the learning process both in computation and accuracy. Their training is long, computation heavy and they often suffer from instability.

Since the experiments proposed in this thesis were performed, it seems that a lot of progress has been made in generated image quality. Therefore, we hope that those progresses will overcome generative model current limitations and fully exploit their potential for continual learning.

9.2.3 Detecting Concept Drift

In this thesis, we did not tackle the problem of concept drift detection. This problem is probably as crucial as tackling catastrophic forgetting for continual learning when the i.i.d assumption does not hold anywhere in the learning curricula. It would be worth studying it more intensively, as it is essential to make lifelong learning work.

9.2.4 Improving Hyper-Parameter Selection with Meta-Continual Learning

As discussed, in Chapter 2, the hyper-parameters (HP) selection is difficult in continual learning. In classical machine learning, we can select HPs that minimize validation set loss. However, in continual learning, we don't have access to the full validation set. It is then crucial to develop strategies to improve HP selection.

Meta-learning, or learning to learn, is a training concept that aims at improving the learning process on new tasks by learning from many others. Meta-algorithms learn the best parameters or/and hyper-parameters to solve new tasks efficiently. In continual learning, it could help to prepare learning algorithms for future tasks and improve existing strategies. It has already been used in many approaches [197, 105, 12].

For replay methods, meta-continual learning could be useful to improve memorization processes, especially by automatically learning the memorization hyper-parameters. For example, it could improve its learning of criterion for data selection or improve memory sampling to maximize remembering and minimize overfitting.

Nevertheless, meta-learning needs to replay tasks several times to learn (or at least similar tasks) and it does not remember from one task to another. It might thus not be relevant in all continual learning settings.

9.2.5 Estimating Knowledge Retention

In this thesis, the algorithm's goal is, at any time, to remember everything of the past in the active memory. Knowledge retention is only estimated as the knowledge that can be directly used for inference. For example, in classification, we only check if the neural network can correctly classify test images. However, latent representations from past tasks can be hidden in the weights but can not be directly activated, i.e. the neural network can have memories of past tasks in inner layers without being able to use them directly because, for example, the output layer has been modified. It would be interesting to tell the difference between a model that still has latent representations of past tasks and a model that forgot everything.

Therefore, it would be relevant to develop methods that promote latent knowledge which could be easily reactivated and valuable again.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alessandro Achille, Tom Eccles, Loic Matthey, Christopher P Burgess, Nick Watters, Alexander Lerchner, and Irina Higgins. Life-long disentangled representation learning with cross-domain latent homologies. *arXiv preprint arXiv:1808.06508*, 2018.

[3] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016.

[4] Rahaf Aljundi. Continual learning in neural networks. *arXiv preprint arXiv:1910.02718*, 2019.

[5] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximal interfered retrieval. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019.

[6] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3366–3375, 2017.

[7] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11816–11825. Curran Associates, Inc., 2019.

[8] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

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

[10] Pablo Azagra, Florian Golemo, Yoan Mollard, Manuel Lopes, Javier C Civera, and Ana C Murillo. A Multimodal Dataset for Object Model Learning from Natural Human-Robot Interaction. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)*, Vancouver, Canada, September 2017.

[11] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[12] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. *arXiv preprint arXiv:2002.09571*, 2020.

[13] F. Bellas, R. J. Duro, A. Faina, and D. Souto. Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots. volume 2, pages 340–354, Dec 2010.

[14] Francisco Bellas, José Antonio Becerra, and Richard J Duro. Using promoters and functional introns in genetic algorithms for neuroevolutionary learning in non-stationary problems. *Neurocomputing*, 72(10-12):2134–2145, 2009.

[15] Francisco Bellas, Andrés Faiña, Gervasio Varela, and Richard J Duro. A cognitive developmental robotics architecture for lifelong learning by evolution in real robots. pages 1–8, 2010.

[16] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[17] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957.

[18] Eden Belouadah and Adrian Popescu. Deesil: Deep-shallow incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[19] Eden Belouadah and Adrian Popescu. Scail: Classifier weights scaling for class incremental learning. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1266–1275, 2020.

[20] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[21] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[22] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[23] David Berthelot, Thomas Schumm, and Luke Metz. Began: boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[24] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[25] A. Borji. Pros and Cons of GAN Evaluation Measures. *ArXiv e-prints*, February 2018.

[26] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*, 2010.

[27] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[28] Nicolas Bredeche, Evert Haasdijk, and Abraham Prieto. Embodied evolution in collective robotics: A review. *Frontiers in Robotics and AI*, 5:12, 2018.

[29] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

[30] Peter Bühlmann and Sara van de Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[31] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019.

[32] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptative quantization module. *arXiv preprint arXiv:1911.08019*, 2019.

[33] R. Camoriano, G. Pasquale, C. Ciliberto, L. Natale, L. Rosasco, and G. Metta. Incremental robot learning of new objects with fixed update time. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3214, May 2017.

[34] Raffaello Camoriano, Silvio Traversaro, Lorenzo Rosasco, Giorgio Metta, and Francesco Nori. Incremental semiparametric inverse dynamics learning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 544–550. IEEE, 2016.

[35] Angelo Cangelosi and Matthew Schlesinger. From babies to robots: The contribution of developmental robotics to developmental psychology. *Child Development Perspectives*, 2018.

[36] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3. Atlanta, 2010.

[37] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[38] Hugo Caselles-Dupré, Michael Garcia-Ortiz, and David Filliat. S-TRIGGER: Continual State Representation Learning via Self-Triggered Generative Replay. *arXiv e-prints*, page arXiv:1902.09434, Feb 2019.

[39] Francisco M Castro, Manuel J Marin-Jimenez, Nicolas Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248, 2018.

[40] Tommaso Cavallari, Stuart Golodetz, Nicholas A Lord, Julien Valentin, Luigi Di Stefano, and Philip HS Torr. On-the-fly adaptation of regression forests for online camera relocalisation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4457–4466, 2017.

[41] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, 2018.

[42] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019.

[43] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[44] François Chollet. keras. https://github.com/fchollet/keras, 2015.

[45] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

[46] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *CoRR*, 2018.

[47] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio López, and Vladlen Koltun. End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410, 2017.

[48] Cédric Colas, Pierre Fournier, Olivier Sigaud, and Pierre-Yves Oudeyer. CURIOUS: intrinsically motivated multi-task, multi-goal reinforcement learning. *CoRR*, abs/1810.06284, 2018.

[49] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: decoupling exploration and exploitation in deep reinforcement learning algorithms. *CoRR*, abs/1802.05054, 2018.

[50] Alvaro Collet, Bo Xiong, Corina Gurau, Martial Hebert, and Siddhartha S Srinivasa. Herbdisc: Towards lifelong robotic object discovery. *The International Journal of Robotics Research*, 34(1):3–25, 2015.

[51] C. Craye, D. Filliat, and J. Goudou. Exploration strategies for incremental learning of object-based visual saliency. In *2015 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 13–18, Aug 2015.

[52] Céline Craye, Timothée Lesort, David Filliat, and Jean-François Goudou. Exploring to learn visual saliency: The rl-iac approach. *Robotics and Autonomous Systems*, 112:244–259, 2019.

[53] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *CoRR*, abs/1702.05374, 2017.

[54] P. T. Dat, A. Dutt, D. Pellerin, and G. Quénot. Classifier training from a generative model. In *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, pages 1–6, Sep. 2019.

[55] Yann N. Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015.

[56] Jean-Francois Delvenne. Science of memory: Concepts. henry l. roediger iii, yadin dudai, and susan m. fitzpatrick (eds.). oxford university press, new york, 2007. no. of pages 464. isbn 978-0-19-531044-3.(paperback). *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, 23(6):895–896, 2009.

[57] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[58] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[59] Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don't forget, there is more than forgetting: new metrics for Continual Learning. In *Workshop on Continual Learning, NeurIPS 2018 (Neural Information Processing Systems*, Montreal, Canada, December 2018.

[60] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017.

[61] Stephane Doncieux, David Filliat, Natalia Díaz-Rodríguez, Timothy Hospedales, Richard Duro, Alexandre Coninx, Diederik M. Roijers, Benoît Girard, Nicolas Perrin, and Olivier Sigaud. Open-ended learning: a conceptual framework based on representational redescription. *Frontiers in Neurorobotics*, 2018.

[62] 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. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 526–533, 2017.

[63] Wuyang Duan. Learning state representations for robotic control. *M. Thesis*, 2017.

[64] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[65] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[66] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[67] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[68] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, 2018.

[69] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[70] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[71] 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. *CoRR*, abs/1701.08734, 2017.

[72] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[73] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[74] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. *CoRR*, abs/1509.06113, 2015.

[75] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

[76] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[77] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[78] T. Furlanello, J. Zhao, A. M. Saxe, L. Itti, and B. S. Tjan. Active Long Term Memory Networks. *ArXiv e-prints*, June 2016.

[79] Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. *arXiv preprint arXiv:1805.04770*, 2018.

[80] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. *CoRR*, abs/1704.05588, 2017.

[81] Donald Geman, Stuart Geman, Neil Hallonquist, and Laurent Younes. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences*, 112(12):3618–3623, 2015.

[82] Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016.

[83] Alexander Gepperth and Cem Karaoguz. A Bio-Inspired Incremental Learning Architecture for Applied Perceptual Problems. *Cognitive Computation*, 8:924 – 934, 2016.

[84] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[85] Florian Golemo. *How to Train Your Robot - New Environments for Robotic Training and New Methods for Transferring Policies from the Simulator to the Real Robot.* Theses, Université de Bordeaux, December 2018.

[86] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *ArXiv e-prints*, dec 2013.

[87] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[88] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1406.2661, Jun 2014.

[89] Alison Gopnik, Andrew Meltzoff, and Patricia Kuhl. The scientist in the crib: Minds, brains and how children learn. *Journal of Nervous and Mental Disease - J NERV MENT DIS*, 189, 03 2001.

[90] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[91] T. L. Hayes, R. Kemker, N. D. Cahill, and C. Kanan. New metrics and experimental paradigms for continual learning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2112–21123, June 2018.

[92] Tyler L. Hayes, Nathan D. Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. *CoRR*, abs/1809.05922, 2018.

[93] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[94] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[95] Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. In *International Conference on Learning Representations*, 2018.

[96] Yue He, Zheyan Shen, and Peng Cui. NICO: A Dataset Towards Non-I.I.D. Image Classification. *arXiv e-prints*, page arXiv:1906.02899, Jun 2019.

[97] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *ArXiv e-prints*, June 2017.

[98] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[99] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop (2015)*, pages 1–9, 2015.

[100] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[101] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[102] Herbert Jaeger. Using conceptors to manage neural long-term memories for temporal patterns. *Journal of Machine Learning Research*, 18(13):1–43, 2017.

[103] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[104] Maximilian Jaritz, Raoul de Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. *CoRR*, abs/1807.02371, 2018.

[105] Khurram Javed and Martha White. Meta-learning representations for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 1818–1828. Curran Associates, Inc., 2019.

[106] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, page 675–678, New York, NY, USA, 2014. Association for Computing Machinery.

[107] D. Jiwoong Im, H. Ma, G. Taylor, and K. Branson. Quantitatively Evaluating GANs With Divergences Proposed for Training. *ArXiv e-prints*, March 2018.

[108] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

[109] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *CoRR*, abs/1607.00122, 2016.

[110] Christoph Käding, Erik Rodner, Alexander Freytag, and Joachim Denzler. Fine-tuning deep neural networks in continuous learning scenarios. In *Asian Conference on Computer Vision*, pages 588–605. Springer, 2016.

[111] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018.

[112] René Traoré Kalifou, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Natalia Diaz-Rodriguez, and David Filliat. Continual reinforcement learning deployed in real-life using policydistillation and sim2real transfer. In *ICML Workshop on Multi-Task and Lifelong Learning*, 2019.

[113] N. Kamra, U. Gupta, and Y. Liu. Deep Generative Dual Memory Network for Continual Learning. *ArXiv e-prints*, October 2017.

[114] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[115] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan. Measuring Catastrophic Forgetting in Neural Networks. *ArXiv e-prints*, aug 2017.

[116] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018.

[117] Seungsu Kim, Alexandre Coninx, and Stéphane Doncieux. From exploration to control: learning object manipulation skills through novelty search and local adaptation. *CoRR*, abs/1901.00811, 2019.

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

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

[120] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep Reinforcement Learning for Autonomous Driving: A Survey. *arXiv e-prints*, page arXiv:2002.00444, Feb 2020.

[121] 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. *Proc. of the national academy of sciences*, 2017.

[122] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[123] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[124] 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.

[125] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[126] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *CoRR*, abs/1811.00982, 2018.

[127] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33, 2011.

[128] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[129] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks, 2019.

[130] Adrien Laversanne-Finot, Alexandre Péré, and Pierre-Yves Oudeyer. Curiosity driven exploration of learned disentangled goal spaces. *CoRR*, abs/1807.01521, 2018.

[131] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[132] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. *public*, 2010.

[133] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pages 4652–4662, 2017.

[134] Timothée LESORT. Continual learning data former, January 2020.

[135] Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Jean-François Goudou, and David Filliat. Generative models from the perspective of continual learning. In *IJCNN - International Joint Conference on Neural Networks*, Budapest, Hungary, Jul 2019.

[136] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-François Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 2018.

[137] Timothée Lesort, Alexander Gepperth, Andrei Stoian, and David Filliat. Marginal replay vs conditional replay for continual learning. In *International Conference on Artificial Neural Networks*, pages 466–480. Springer, 2019.

[138] Timothée Lesort, Mathieu Seurin, Xinrui Li, Natalia Díaz Rodríguez, and David Filliat. Unsupervised state representation learning with robotic priors: a robustness analysis. In *International Joint Conference on Neural Networks*, 2019.

[139] Timothée Lesort, Andrei Stoian, and David Filliat. Regularization shortcomings for continual learning. *arXiv preprint arXiv:1912.03049*, 2019.

[140] Timothée Lesort, Andrei Stoian, Jean-François Goudou, and David Filliat. Training discriminative models to evaluate generative ones. In Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2019: Image Processing*, pages 604–619, Cham, 2019. Springer International Publishing.

[141] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52 – 68, 2020.

[142] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

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

[144] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[145] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[146] Vincenzo Lomonaco. Continual learning with deep architectures. 2019.

[147] Vincenzo Lomonaco, Karan Desai, Eugenio Culurciello, and Davide Maltoni. Continual reinforcement learning in 3d non-stationary environments. *arXiv preprint arXiv:1905.10112*, 2019.

[148] Vincenzo Lomonaco and Davide Maltoni. Comparing incremental learning strategies for convolutional neural networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 175–184. Springer, 2016.

[149] Vincenzo Lomonaco and Davide Maltoni. CORe50: a New Dataset and Benchmark for Continuous Object Recognition. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 17–26. PMLR, 13–15 Nov 2017.

[150] David Lopez-Paz and Marc-Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6467–6476. Curran Associates, Inc., 2017.

[151] Max Lungarella, Giorgio Metta, Rolf Pfeifer, and Giulio Sandini. Developmental robotics: a survey. *Connection Science*, 15(4):151–190, 2003.

[152] Natalia Lyubova, Serena Ivaldi, and David Filliat. From passive to interactive object learning and recognition through self-identification on a humanoid robot. *Autonomous Robots*, page 23, 2015.

[153] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[154] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[155] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019.

[156] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018.

[157] Daniel J Mankowitz, Augustin Žídek, André Barreto, Dan Horgan, Matteo Hessel, John Quan, Junhyuk Oh, Hado van Hasselt, David Silver, and Tom Schaul. Unicorn: Continual learning with a universal, off-policy agent. *arXiv preprint arXiv:1802.08294*, 2018.

[158] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[159] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

[160] Jan Mattner, Sascha Lange, and Martin A. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *Neural Information Processing - 19th International Conference, ICONIP 2012, Doha, Qatar, November 12-15, 2012, Proceedings, Part V*, pages 126–133, 2012.

[161] James L McClelland, Bruce L McNaughton, and Randall C O'reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[162] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[163] 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(August):504, 2013.

[164] Umberto Michieli and Pietro Zanuttigh. Knowledge distillation for incremental learning in semantic segmentation, 2019.

[165] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, Jan 1961.

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

[167] T. Mitchell, W. Cohen, E. Hruscha, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohammad, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *AAAI*, 2015. : Never-Ending Learning in AAAI-2015.

[168] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[169] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[170] Vincent Moens and Alexandre Zenon. Learning and forgetting using reinforced bayesian change detection. *bioRxiv*, 2018.

[171] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[172] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018.

[173] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pages 271–279, 2016.

[174] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[175] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with Large Scale Deep Reinforcement Learning, Dec 2019.

[176] P.-Y. Oudeyer, F. Kaplan, and V.V. Hafner. Intrinsic motivation systems for autonomous mental development. *Evolutionary Computation, IEEE Transactions on*, 11(2):265–286, April 2007.

[177] Pierre-Yves Oudeyer. Computational theories of curiosity-driven learning. *CoRR*, abs/1802.10546, 2018.

[178] German I. Parisi, Tani Jun, Cornelius Weber, and Stefan Wermter. Lifelong Learning of Spatiotemporal Representations with Dual-Memory Recurrent Self-Organization. *arXiv preprint arXiv:1805.10966*, pages 1–20, 2018.

[179] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54 – 71, 2019.

[180] G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale. Teaching icub to recognize objects using deep convolutional neural networks. In *Proceedings of the 4th International Conference on Machine Learning for Interactive Systems - Volume 43*, MLIS'15, pages 21–25. JMLR.org, 2015.

[181] G. Pasquale, C. Ciliberto, L. Rosasco, and L. Natale. Object identification from few examples by improving the invariance of a deep convolutional neural network. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4904–4911, Oct 2016.

[182] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[183] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE Signal Processing Magazine*, 32(3):53–69, May 2015.

[184] Lorenzo Pellegrini, Gabrile Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. *arXiv preprint arXiv:1912.01100*, 2019.

[185] Anastasia Pentina and Christoph H Lampert. Lifelong learning with non-iid tasks. In *Advances in Neural Information Processing Systems*, pages 1540–1548, 2015.

[186] B. Pfulb and A. Gepperth. A comprehensive, application-oriented study of catastrophic forgetting in DNNs. In *International Conference on Learning Representations*, 2019.

[187] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015.

[188] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[189] Antonin Raffin, Ashley Hill, Kalifou René Traoré, Timothée Lesort, Natalia Díaz-Rodríguez, and David Filliat. Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics. In *ICLR*, 2019.

[190] Antonin Raffin, Ashley Hill, René Traoré, Timothée Lesort, Natalia Díaz-Rodríguez, and David Filliat. S-rl toolbox: Environments, datasets and evaluation metrics for state representation learning. *arXiv preprint arXiv:1809.09369*, 2018.

[191] Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Lifelong generative modeling. *arXiv preprint arXiv:1705.09847*, 2017.

[192] Suman Ravuri and Oriol Vinyals. Classification accuracy score for conditional generative models. In *Advances in Neural Information Processing Systems*, pages 12247–12258, 2019.

[193] 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.

[194] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning,*, 2014.

[195] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *CoRR*, abs/1810.06544, 2018.

[196] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

[197] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[198] Mark B Ring. Toward a formal framework for continual learning. In *NIPS workshop on Inductive Transfer, Whistler, Canada.*, 2005.

[199] Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

[200] Amanda Rios and Laurent Itti. Closed-loop memory gan for continual learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pages 3332–3338. AAAI Press, 2019.

[201] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems*, pages 3738–3748, 2018.

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

[203] Diego Romeres, Mattia Zorzi, Raffaello Camoriano, and Alessandro Chiuso. Online semi-parametric learning for inverse dynamics modeling. *arXiv e-prints*, page arXiv:1603.05412, March 2016.

[204] Diego Romeres, Mattia Zorzi, Raffaello Camoriano, Silvio Traversaro, and Alessandro Chiuso. Derivative-free online learning of inverse dynamics models. *CoRR*, abs/1809.05074, 2018.

[205] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[206] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *ArXiv e-prints*, jun 2016.

[207] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. *arXiv e-prints*, page arXiv:1511.06295, Nov 2015.

[208] Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *CoRR*, abs/1610.04286, 2016.

[209] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[210] Shibani Santurkar, Ludwig Schmidt, and Aleksander Mądry. A classification-based study of covariate shift in gan distributions. *arXiv preprint arXiv:1711.00970*, 2017.

[211] Pradipta Sarkar and William Q Meeker. A bayesian on-line change detection algorithm with process monitoring applications. *Quality Engineering*, 10(3):539–549, 1998.

[212] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[213] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[214] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[215] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[216] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[217] Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *ICML*, 2018.

[218] Ari Seff, Alex Beatson, Daniel Suo, and Han Liu. Continual learning in generative adversarial nets. *CoRR*, abs/1705.08395, 2017.

[219] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *ICML*, 80:4548–4557, 10–15 Jul 2018.

[220] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[221] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[222] Haseeb Shah, Khurram Javed, and Faisal Shafait. Distillation techniques for pseudo-rehearsal based incremental learning. *arXiv preprint arXiv:1807.02799*, 2018.

[223] Qi She, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, Fei Qiao, and Rosa H. M. Chan. Openloris-object: A dataset and benchmark towards lifelong object recognition, 2019.

[224] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

[225] Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, Yangquan Guo, Zhigang Wang, Yimin Zhang, Baoxing Qin, Wei Yang, Fangshi Wang, Rosa H. M. Chan, and Qi She. Are we ready for service robots? the openloris-scene datasets for lifelong slam, 2019.

[226] 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.

[227] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. *CoRR*, abs/1708.06977, 2017.

[228] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. How good is my gan? In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229, 2018.

[229] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[230] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[231] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[232] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *Artificial life*, 11:13–29, 12 2005.

[233] Shagun Sodhani, Sarathcon Chandar, and Yoshua Bengio. On training recurrent neural networks for lifelong learning. *CoRR*, abs/1811.07017, 2018.

[234] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc., 2015.

[235] Pablo Sprechmann, Siddhant M Jayakumar, Jack W Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. Memory-based parameter adaptation. *arXiv preprint arXiv:1802.10542*, 2018.

[236] Freek Stulp, Laura Herlant, Antoine Hoarau, and Gennaro Raiola. Simultaneous on-line discovery and improvement of robotic skill options. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1408–1413. IEEE, 2014.

[237] Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *International Conference on Artificial General Intelligence*, pages 41–51. Springer, 2011.

[238] Niko Sünderhauf, Oliver Brock, Walter J. Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke. The limits and potentials of deep learning for robotics. *CoRR*, abs/1804.06557, 2018.

[239] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[240] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[241] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[242] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[243] Victor Talpaert, Ibrahim Sobh, B. Kiran, Patrick Mannion, Senthil Yogamani, Ahmad El-Sallab, and Patrick Pérez. Exploring Applications of Deep Reinforcement Learning for Real-world Autonomous Driving Systems. In *14th International Conference on Computer Vision Theory and Applications*, pages 564–572, Prague, Czech Republic, February 2019. SCITEPRESS - Science and Technology Publications.

[244] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.

[245] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. In *The biology and technology of intelligent autonomous agents*, pages 165–196. Springer, 1995.

[246] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[247] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct 2012.

[248] René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, Natalia Díaz Rodríguez, and David Filliat. Discorl: Continual reinforcement learning via policy distillation. *CoRR*, abs/1907.05855, 2019.

[249] Amal Rannen Triki, Rahaf Aljundi, Matthew B. Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. *CoRR*, abs/1704.01920, 2017.

[250] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

[251] Leslie G Valiant. A theory of the learnable. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 436–445. ACM, 1984.

[252] Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[253] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934, Oct 2016.

[254] Roby Velez and Jeff Clune. Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks. *PloS one*, 12(11), 2017.

[255] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *CoRR*, abs/1811.10959, 2018.

[256] Wei Wang, Vincent W. Zheng, Han Yu, and Chunyan Miao. A survey of zero-shot learning: Settings, methods, and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2):13:1–13:37, January 2019.

[257] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Growing a brain: Fine-tuning by increasing model capacity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2471–2480, 2017.

[258] Z Wang, E P Simoncelli, and A C Bovik. Multiscale structural similarity for image quality assessment. In *Proc 37th Asilomar Conf on Signals, Systems and Computers*, 2003.

[259] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[260] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[261] Juyang Weng, James McClelland, Alex Pentland, Olaf Sporns, Ida Stockman, Mriganka Sur, and Esther Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001.

[262] Jay M Wong. Towards lifelong self-supervision: A deep learning direction for robotics. *arXiv preprint arXiv:1611.00201*, 2016.

[263] Chenshen Wu, Luis Herranz, Xialei Liu, yaxing wang, Joost van de Weijer, and Bogdan Raducanu. Memory replay gans: Learning to generate new categories without forgetting. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5962–5972. Curran Associates, Inc., 2018.

[264] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019.

[265] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, Zhengyou Zhang, and Yun Fu. Incremental classifier learning with generative adversarial networks. *CoRR*, abs/1802.00853, 2018.

[266] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[267] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[268] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015.

[269] Friedeman Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[270] Chenyang Zhao, Timothy M Hospedales, Freek Stulp, and Olivier Sigaud. Tensor based knowledge transfer across skill categories for robot control. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3462–3468. AAAI Press, 2017.

[271] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. Online incremental feature learning with denoising autoencoders. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 1453–1461, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

INSTITUT POLYTECHNIQUE DE PARIS

Titre: Apprentissage Continu: S'attaquer à l'Oubli Foudroyant des Réseaux de Neurones Profonds grâce aux Methodes à Rejeu de Données

Mots clés: Apprentissage profond, Apprentissage Continu, Régénération, Méthodes de Rejeu

Résumé:
Les humains apprennent toute leur vie. Ils accumulent des connaissances à partir d'une succession d'expériences d'apprentissage et en mémorisent les aspects essentiels sans les oublier. Les réseaux de neurones artificiels ont des difficultés à apprendre dans de telles conditions. Ils ont en général besoin d'ensembles de données rigoureusement préparés pour pouvoir apprendre à résoudre des problèmes comme de la classification ou de la régression. En particulier, lorsqu'ils apprennent sur des séquences d'ensembles de données, les nouvelles expériences leurs font oublier les anciennes. Ainsi, ils sont souvent incapables d'appréhender des scénarios réels tels ceux de robots autonomes apprenant en temps réel à s'adapter à de nouvelles situations et devant résoudre des problèmes sans oublier leurs expériences passées.
L'apprentissage continu est une branche de l'apprentissage automatique s'attaquant à ce type de scénarios. Les algorithmes continus sont créés pour apprendre des connaissances, les enrichir et les améliorer au cours d'un curriculum d'expériences d'apprentissage.
Dans cette thèse, nous proposons d'explorer l'apprentissage continu avec rejeu de données. Les méthodes de rejeu de données rassemblent les méthodes de répétitions et les méthodes de rejeu par génération. Le rejeu par génération consiste à utiliser un réseau de neurones auxiliaire apprenant à générer les données actuelles. Ainsi plus tard le réseau auxiliaire pourra être utilisé pour régénérer des données du passé et les remémorer au modèle principal. La répétition a le même objectif, mais cette méthode sauve simplement des images spécifiques et les rejoue plus tard au modèle principal pour éviter qu'il ne les oublie. Les méthodes de rejeu permettent de trouver un compromis entre l'optimisation de l'objectif d'apprentissage actuel et ceux du passé. Elles permettent ainsi d'apprendre sans oublier sur des séquences de tâches.
Nous montrons que ces méthodes sont prometteuses pour l'apprentissage continu. En particulier, elles permettent la réévaluation des données du passé avec des nouvelles connaissances et de confronter des données issues de différentes expériences. Nous démontrons la capacité des méthodes de rejeu à apprendre continuellement à travers des tâches d'apprentissage non-supervisées, supervisées et de renforcements.

Title: Continual Learning: Tackling Catastrophic Forgetting in Deep Neural Networks with Replay Processes

Keywords: Deep Learning, Continual Learning, Generative Replay, Replay Processes

Abstract:
Humans learn all their life long. They accumulate knowledge from a sequence of learning experiences and remember the essential concepts without forgetting what they have learned previously. Artificial neural networks struggle to learn similarly. They often rely on data rigorously preprocessed to learn solutions to specific problems such as classification or regression. In particular, they forget their past learning experiences if trained on new ones. Therefore, artificial neural networks are often inept to deal with real-life settings such as an autonomous-robot that has to learn on-line to adapt to new situations and overcome new problems without forgetting its past learning-experiences.
Continual learning (CL) is a branch of machine learning addressing this type of problem. Continual algorithms are designed to accumulate and improve knowledge in a curriculum of learning-experiences without forgetting.
In this thesis, we propose to explore continual algorithms with replay processes. Replay processes gather together rehearsal methods and generative replay methods. *Generative Replay* consists of regenerating past learning experiences with a generative model to remember them. *Rehearsal* consists of saving a core-set of samples from past learning experiences to rehearse them later. The replay processes make possible a compromise between optimizing the current learning objective and the past ones enabling learning without forgetting in sequences of tasks settings.
We show that they are very promising methods for continual learning. Notably, they enable the reevaluation of past data with new knowledge and the confrontation of data from different learning-experiences. We demonstrate their ability to learn continually through unsupervised learning, supervised learning and reinforcement learning tasks.

Institut Polytechnique de Paris
91120 Palaiseau, France