Alma Mater Studiorum
Università di Bologna

Dottorato di Ricerca in
Computer Science and Engineering
XXXI Ciclo

Continual Learning with Deep Architectures

Presentata da
Vincenzo Lomonaco

Supervisore
prof. Davide Maltoni

Coordinatore Dottorato
prof. Paolo Ciaccia

Settore Scientifico Disciplinare
ING-INF/05

Settore Concorsuale
09/H1

Aprile, 2019

"We may regard the present state of the universe as the effect of the past and the cause of the future. An intellect which at any given moment knew all of the forces that animate nature and the mutual positions of the beings that compose it, if this intellect were vast enough to submit the data to analysis, could condense into a single formula the movement of the greatest bodies of the universe and that of the lightest atom; for such an intellect nothing could be uncertain and the future just like the past would be present before its eyes."

Pierre Simon Laplace, *A Philosophical Essay on Probabilities*, 1814

ALMA MATHER STUDIORUM
UNIVERSITY OF BOLOGNA

Abstract

Department of Computer Science and Engineering
Doctoral Degree

Continual Learning with Deep Architectures

Vincenzo Lomonaco

Humans have the extraordinary ability to learn continually from experience. Not only we can apply previously learned knowledge and skills to new situations, we can also use these as the foundation for later learning. One of the grand goals of Artificial Intelligence (AI) is building an artificial *"continual learning"* agent that constructs a sophisticated understanding of the world from its own experience through the autonomous incremental development of ever more complex knowledge and skills.

However, despite early speculations and few pioneering works, very little research and effort has been devoted to address this vision. Current AI systems greatly suffer from the exposure to new data or environments which even slightly differ from the ones for which they have been trained for. Moreover, the learning process is usually constrained on fixed datasets within narrow and isolated tasks which may hardly lead to the emergence of more complex and autonomous intelligent behaviors. In essence, continual *learning* and *adaptation* capabilities, while more than often thought as fundamental pillars of every intelligent agent, have been mostly left out of the main AI research focus.

In this dissertation, we study the application of these ideas in light of the more recent advances in machine learning research and in the context of *deep architectures* for AI. We propose a comprehensive and unifying framework for continual learning, new metrics, benchmarks and algorithms, as well as providing substantial experimental evaluations in different supervised, unsupervised and reinforcement learning tasks.

Acknowledgements

My interest in *Artificial Intelligence* bloomed around 2012, when I was only a curious bachelor student at the *University of Modena and Reggio-Emilia*. During that year I read, almost by chance, the best-selling book *"The Singlularity is Near"* by Raymond Kurzweil which resonated very much in me. At the time, I was specializing on data management and information retrieval systems as they appeared to be ubiquitously present and fundamental in every computer science application. I soon realized that, in the future of automation and computer science, AI was the next big paradigm shift. From data to information, from information to knowledge, *intelligence* was the next logical step.

During my master degree at the *University of Bologna* (Unibo), I started to develop my background in AI with a particular interest and focus on bio-inspired computing, machine learning and computational neuroscience. It is not after long that I discovered in *Davide Maltoni* the only professor within my university betting on this approach and sharing this perspective. Davide was working on this line of research since 2011. In particular he introduced me to *Jeff Hawkins*'s research and approach masterfully outlined in his book *"On Intelligence"* which I found fascinating.

I will be forever thankful to Davide, who later become my supervisor and mentor for my master dissertation on *hierarchical temporal memories*, and later on of my doctoral degree on *continual learning*, answering thousands of e-mails, helping me with incredibly valuable insights and providing me of a huge amount of time to discuss and develop my research ideas. His expertise and almost paternal advice, his passion and dedication have thought me how beautiful and motivating research can be.

I would not lie saying that the PhD was an easy ride. It has been tough, even exhausting at times, but highly rewarding, especially from a human perspective. It would be unfair, indeed, to present this dissertation as the result of my only doing, being it rather the culmination of extensive collaborations and discussions of many talented people who helped me at many different levels throughout my doctoral path.

In primis, I would like to thank my colleagues and friends at the department of computer science and engineering at Unibo: *Francesco Gavazzo*, *Luca Bedogni*, *Daniel Russo*, *Saverio Giallorenzo*, *Abel García Celestrín*, *Yiyi Linares Zaila*, *Micheal Lodi*, *Vincenzo Mastroandrea*, *Federico Montori*, *Stefano Giovanni Rizzo*, *Luca Sciullo*, *Liu Tong*, *Angelo Trotta*, *Stefano Pio Zingaro*, *Zio Felice*, *Adele Veschetti*, *Wilson Sakpere*, *Giacomo Bergami* who helped me with their experience and pleasant company almost on a daily basis. A special thanks goes also to my fellow *Biolab* and *SmartCity* members: *Dario Maio*, *Antonio Magnani*, *Luca Calderoni*, *Lorenzo Pellegrino*, *Federico Fucci*, *Annalisa Franco*, *Raffaele Cappelli*, *Matteo Ferrara*, *Alessandra Lumini*, *Serena Papi*, *Matteo Golfarelli*, to my other collaborators *Luciano Bononi* and *Paolo Ciancarini* and to all the fellow PhD students at Unibo *Luigi Asprino*, *Angelo Croatti*, *Andrea Pagliarani*, *Alessio Tonioni* among others I had the honor to represent in the department council.

After the first 18 months, having laid the foundations of this dissertation, I had the chance to spend six month in the *United States* as a visiting scholar at the *Purdue University* under the

supervision of *Eugenio Culurciello*. Eugenio, with his amazing background and "*full-stack*" experience in developing cutting-hedge AI systems, hosted me in his laboratory where we explored the application of continual learning to reinforcement learning scenarios. I cannot express my deepest gratitude to Eugenio and the other *e-lab* members: *Abhishek Chaurasia*, *Andre Chang*, *Aliasger Zaidy*, *Lukasz Burzawa* and *Dawood Sheik* who provided me all the help I could ask for and greatly enriched my view on this topic.

At the end of 2017, experiencing first hand the lack of resources and connections within the continual learning community, I started working on *ContinualAI*, a non-profit research organization and open community on the same topic. To my own amazement, the community has grown to date to more than 300 members worldwide with whom I had the pleasure to share countless discussions and insights other than starting meaningful collaborations. I thank them all and in particular *Keiland Cooper*, *Martin Cimmino*, *Natalia Díaz Rodríguez*, *Timothée Lesort*, *Giacomo Bartoli*, *Jidin Dinesh*, *Michele Cipriano*, *Manish Agnihotri*, *German I. Parisi* and *Lorenzo Pellegrini* without whom this would have never been possible.

During the writing of this dissertation I also had the chance to visit the *Autonomous Systems and Robotics* laboratory at *EnstaParis Tech* in Paris, where under the supervision of *David Filliat* I could develop my first ideas for applying continual learning in the context of Robotics. David and his group hosted me in their laboratory where I immediately felt at home. Special thanks are due to *Natalia Diaz Rodriguez*, *Timothee Lesort*, *Daniela Pamplona*, *Antonin Raffin*, *Hugo Caselles-Dupré*, *Florence Carton* and the many other lab members with whom I had the pleasure to share many happy moments and insightful discussions about AI.

At the end of this path, countless are the people who influenced it. However, it is only right at this point to thank the pioneers and fellow researchers in the field with whom I could share my ideas and discuss about my research: *Mark Ring*, *Marc'Aurelio Ranzato*, *Marc Pickett*, *Raia Hadsell*, *Massimiliano Versace*, *Subutai Ahmad*, *German I. Parisi* and *Gianni De Fabritiis* in particular, among many others.

Finally, words cannot describe how grateful I am to all the people who assisted me during my PhD years also outside the university scope. My family, *Elisabetta Giubilato*, *Francesco Lomonaco*, *Giovanni Lomonaco*, *Sara Lomonaco* and *Pietro Lomonaco* who have never constrained me and have always been present in times of need; my historical friends *Domenico Ivan Maggiore*, *Pasquale Grosso*, *Pierpaolo Del Coco*, *Ivan Heibi*, *Stefano Beccaletto*, *Andrea Motisi*, *Filippo Maria Di Nardo*, *Marina Foti* and roommates *Alessio Di Luca* and *Dario Sucameli* for giving me so much strength and comfort also outside the university spaces. My friends *Marta Ziosi* and *Giovanni Gaspare Righi* from *AI for People* and *Simone Scardapane* from the *Italian Association of Machine Learning*. My love, *Viviana Palumbo*, who changed my perspective on life as no one could.

Thanks to the thinkers and all the other friends I could not mention in this brief acknowledgment section, who, during my PhD, significantly shaped my professional and personal life and to the many people I crossed paths with, who, simply touching it, made it an unique and wonderful journey.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
BP	**B**ack **P**ropagation
CL	**C**ontinual **L**earning
CNN	**C**onvolutional **N**eural **N**etwork
DL	**D**eep **L**earing
GIF	**G**raphics **I**nterchange **F**ormat
HSR	**H**TM **S**upervised **R**efinement
HTM	**H**ierarchical **T**emporal **M**emory
MHWA	**M**ulti-stage **H**ubel-**W**iesel **A**rchitectures
LLL	**L**ife**L**ong **L**earning
MIT	**M**ultiple **I**ncremental **T**ask
ML	**M**achine **L**earing
MLP	**M**ulti **L**ayer **P**erceptron
MT	**M**ulti **T**ask
NC	**N**ew **I**stances
NI	**N**ew **I**stances
NIC	**N**ew **I**stances and **C**lasses
NN	**N**eural **N**etwork
PAC	**P**robably **A**pproximately **C**orrect
RGB	**R**ed **G**reen and **B**lue
RL	**R**einforcement **L**earning
SIT	**S**ingle **I**ncremental **T**ask
STDP	**S**pike-**T**iming-**D**ependent **P**lasticity

Poi che spiegat'ho l'ali al bel desio,
quanto più sott'il piè l'aria mi scorgo,
più le veloci penne al vento porgo:
e spreggio il mondo, e vers'il ciel m'invio.

Né del figliuol di Dedalo il fin rio
fa che giù pieghi, anzi via più risorgo;
ch'i' cadrò morto a terra ben m'accorgo:
ma qual vita pareggia al morir mio?

La voce del mio cor per l'aria sento:
"Ove mi porti, temerario? china,
che raro è senza duol tropp'ardimento";

"Non temer," respond'io, "l'alta ruina.
Fendi sicur le nubi, e muor contento:
s'il ciel sì illustre morte ne destina".

Sonetto II: "Poi che spiegat'ho" (ca. 1550)
in "Poesie liriche edite ed inedite di Luigi Tansillo",
pp. 214-17 (F. Fiorentino ed. 1888)
Luigi Tansillo

Introduction

The notion of learning continually from experience has alway been present in AI and *Machine Learning* (*ML*) since their births, as justified by an evident observation of our biological counterparts [Chen and Liu, 2018; Jantke, 1993; Michalski and Larson, 1978]. Nevertheless, given the (relatively) limited amount of digital data available up to the last two decade and the complexity of the early tackled problems, essentially every machine learning problem could have been framed as a *"static"* one (i.e. without the need to update the AI system over time).

With the 21st century and the *Big Data* revolution, this appears to be less and less realistic for an increasingly amount of applications, due to data *volume*, *variability* and *velocity* [Laney, 2001]. However, in the first decade of the century, the AI and Machine Learning research community have been focusing on more immediate problems like numerical optimization and ad-hoc feature engineering which were fundamental for scaling up the complexity of AI learning systems to high-dimensional domains like computer vision and speech recognition, among many others [Goodfellow et al., 2016].

After rise of *Deep Learning* (*DL*) [LeCun et al., 2015], especially after 2012 and the groundbraking work by Krizhevsky et al. [2012], the increasingly general and robust ability of learning representations from raw data has open the path to a broader range of applications, whose complexity was even unthinkable to tackle a few decades ago. Learning problems are now difficult to encapsulate and isolate into single domains or tasks. In fact, novel learning algorithms, like their biological counterpart, would ideally need access to large volumes of high-dimensional, multi-domain, *streaming* data from complex and ever-changing environments in order to *scale* in terms of intelligence [Kaiser et al., 2017] and being able to *adapt* to new circumstances over time.

This has motivated a renewed and rapidly growing interest in Continual Learning (CL), especially after 2016 [Parisi et al., 2018a]. Unfortunately, current deep learning techniques face today a number of concrete issues in learning over a continuous stream of data [French, 1999; Goodfellow et al., 2013; McCloskey and Cohen, 1989], with prediction models generally trained only on fixed and representative datasets collected a-priori and whose capabilities are very difficult to efficiently generalize or adapt over time.

In this dissertation, we propose a comprehensive overview on recent advancements in continual learning for deep architectures [Bengio, 2009] and a number of original contributions at different levels which can be summarized as follow:

- **A Comprehensive framework for continual learning**: We formalize what does it mean to learn continually in a machine learning setting, taking into account different formulations that have been proposed in the past in order to find a more shared understanding and common ground for the development of novel continual learning algorithms.
- **Four regularization strategies**: We propose a number of regularization strategies (namely SST, CWR, CWR+ and AR1) especially targeting *"Single-Incremental-Task"* scenarios, which have had little attention in the recent deep continual learning literature.
- **Five novel benchmark for CL**: We propose a number of re-designed computer vision benchmarks as well as completely original datasets for assessing different continual learning strategies in various contexts and applications. In particular the *Seq-NORB*, *Seq-COIL100*, *Seq-IcubWorld28*, *CORe50* and *3D VizDOOM Maze* benchmarks are proposed.
- **New and comprehensive evaluation metrics**: The lack of consensus in evaluating continual learning algorithms and the almost exclusive focus on *catastrophic forgetting* motivate us to propose a more comprehensive set of implementation independent metrics accounting for different factors we believe have practical implications worth considering in the deployment of real AI systems that learn continually: accuracy or performance over time, backward and forward knowledge transfer, memory overhead as well as computational efficiency. We further draw inspiration from standard Multi-Attribute Value Theory (MAVT) [Ishizaka and Nemery, 2013] to fuse these metrics into a single CL score for ranking purposes.
- **Extensive CL strategies evaluation**: We evaluate the aforementioned strategies on the classic and proposed benchmarks for continual learning and we show limits and potentials of state-of-the-art approaches in various supervised, semi-supervised and reinforcement learning settings.

The dissertation is organized as follows. Chapter 1 introduces the relative background on biological and artificial learning systems, focusing on continual learning and the main motivation behind it. Chapter 2 presents the general (and more formal) framework for continual learning, while Chapter 3, one of the most important chapter of the dissertation, describes the most popular continual learning strategies targeting deep architectures. Originally designed strategies are also introduced and explained. In Chapter 4, commonly used and newly proposed benchmarks along with their evaluation protocols and metrics are described. Finally, in Chapter 5, an extensive experimental evaluation is conducted, while in Chapter 6, conclusions of the dissertation and future challenges of continual learning are discussed.

1

Background and Motivation

> *"We are not looking for incremental improvements in state-of-the-art AI and neural networks, but rather paradigm-changing approaches to machine learning that will enable systems to continuously improve based on experience."*
>
> – Hava Siegelmann, 2018

Humans have the extraordinary ability to learn continually from experience. Not only can we apply previously learned knowledge and skills to new situations, we can also use these as the foundation for later learning. One of the grand goals of AI is building an artificial *continual learning* agent that constructs a sophisticated understanding of the world from its own experience through the autonomous incremental development of ever more complex skills and knowledge [Ring, 1994].

However, artificial learning systems today seems very far from that goal. While during the last few years we have witnessed formidable progress in the context of semi-supervised and reinforcement learning (i.e. being able to operate with less and less direct supervision) with the ability to learn more *autonomously* [Goodfellow et al., 2016; LeCun et al., 2015; Mnih et al., 2013], very little has been done in deep learning with the idea of learning *continuously*.

State-of-the-art AI systems still show very limited capabilities in terms of *adaptation*, *scalability*, *autonomy*, *common sense* and *reasoning* [Marcus, 2018; Pearl, 2018]. However, as we will see later on in this chapter, continual learning could be of great value not only for the more intuitive adaptability, but also for the others.

Neural networks and their latest reincarnation in deep learning models already took a loosely inspiration from biological learning systems, especially in their architectures (e.g. the visual hierarchies in the human cortex) [Hubel and Wiesel, 1962; LeCun et al., 2015; Poggio and Riesenhuber, 1999]. However, state-of-the-art training procedures (often gradient based) differ significantly from the little we know about what happens in the brain. Should we take more insights from biology, and if yes, to what extent? In this chapter we will tackle these among many other important questions.

1.1 Natural and Artificial Intelligence

Intelligence is a word of multiple facets. Many are the theories developed in the attempt to define what intelligence is and what are its basic principles or core objectives [Russell and Norvig, 2016]. Humphreys [1979] in his *"The construct of general intelligence"* define it as: *"The resultant of the process of acquiring, storing in memory, retrieving, combining, comparing, and using in new contexts information and conceptual skills."*

In the book *"Dynamic assessments of cognitive modifiability"*, Feuerstein [2003] prefers to define intelligence as: *"The unique propensity of human beings to change or modify the structure of their cognitive functioning to adapt to the changing demands of a life situation."*

A more focused and concise definition by Sternberg [1982] in his book *"Handbook of Human Intelligence"* cite as follows: *"Goal-directed adaptive behavior."*

While different in terms of style and perspective, a common and central idea can be noticed: the idea of **adaptation**, the ability to mold our cognitive system to deal with the always changing demanding circumstances [Schulz, 2012].

Scalability is one of the most important concept in computer science and, we argue, among the computational principles of intelligence. As we'll see in the next sections, in CL this idea force us to think at intelligence and develop algorithms which can already deal with real-world computational and memory constraints. If the long-term goal of research in AI is developing machines which are endowed with versatility and common sense, we better make sure they are scalable in terms of intelligence while being sustainable in terms of computational and memory resources [Lake et al., 2016].

Another import characteristic of intelligent agents is **autonomy**, which can be seen as the ability to learn about the external world without any direct supervision, from an external entity or oracle. While parents teaching in humans and other animals is evidently essential for fast and robust learning, especially in the early stages of life, being able to learn new knowledge and skills through autonomous trials/errors, exploration and reasoning is generally acknowledged as a core property of intelligence [Lake et al., 2016].

Common sense is one of the most debated (and important) concept in AI, consisting of *"knowledge, judgment, and taste which is more or less universal and which is held more or less without reflection or argument"* [van Holthoon and Olson, 1987]. It turns out that endowing machines with commons sense, while generally assumed aprioristically for humans, is particularly challenging, implying a number of deeply interconnected abilities that can be developed after years of learning and functioning as the basis for later learning.

Last but not least, **reasoning**, one of the fist problems tackled in AI and arguably one of the highest demonstration of intelligence, which can be generally thought as the ability to infer new knowledge based on previous knowledge through deduction, induction and abduction rules [Lake et al., 2016].

These, non exhaustive set of abilities are, we believe, of fundamental value for every Intelligent agent, regardless of whether it is natural or artificial.

1.1.1 Limits and Potentials of Deep Learning

Artificial Intelligence is conventionally recognized as a field in its own right starting from a workshop at Dartmouth College in 1956. Allen Newell, Herbert Simon, John McCarthy, Marvin Minsky and Arthur Samuel, who became the founders and scientific fathers of AI research were among the organizers. They and their students produced programs that the press described as *"astonishing"* at the time [Russell and Norvig, 2016] and were related essentially to reasoning skills like solving the game of checkers, proving basic algebra theorems or speaking English [Samuel, 1959].

By the middle of the 1960s (especially in the U.S.), research was heavily funded by the Department of Defense and laboratories had been established around the world. However, despite the initial excitement and overoptimistic predictions, progress rapidly slowed down and in 1974, in response to the criticism of Sir James Lighthill and ongoing pressure from the US Congress to fund more productive projects, both the U.S. and British governments (two of the leading countries in Computer Science and AI technologies) cut off funding for exploratory research in this area. The next few years would be later recognized as an *"AI winter"*, a period when obtaining funding for AI projects was difficult and research progresses were moderate.

In the early 1980s, AI research was revived by the commercial success of *Expert Systems* [Liao, 2005], a form of AI programs that simulated the knowledge and analytical skills of human experts based on *symbolic approaches* [Russell and Norvig, 2016]. It was only in the last decade of the 20th century and early years of the next that the more analytical approaches of statistical machine learning started to gain real traction due to its increasing applicability to real-word, industrial problems.

However, much of these techniques were conceived *ad-hoc* for each separate task they were aimed at solving. More general learning methods, like the successful *Support Vector Machines* (*SVM*) algorithms and related *kernel methods* [Steinwart and Christmann, 2008], were used only in conjunction with specific *"features extraction"* techniques, especially for more complex, high-dimensional problems such as object recognition in computer vision.

This also lead to the development of a rich set of techniques in machine learning to directly *learn* the best representations for solving specific tasks eliminating the need for manual feature engineering (which also required substantial domain expertise). This sub-field of machine learning is often called *"Feature Learning"* or *"Representation learning"* [Goodfellow et al., 2016].

Deep Learning can be seen as part of this broader family, but it takes its specific characterization in the idea of learning *deep hierarchies* (many subsequent layers) of learned representation, which turned out to be fundamental, especially for high dimensional raw data like images in computer vision [Bengio, 2009; Goodfellow et al., 2016]. But maybe more importantly of the notion of *depth* for learning representation, deep architectures have opened the door to the concept of *end-to-end learning*, i.e. the idea of assembling parameterized functional blocks and optimizing them homogeneously (often with gradient-based methods) from input raw data to output. On one hand, this means a great comeback of the *connectionists* theory of intelligence (and much nearer to biology as we will see in the next section). On the other hand, it has enormous practical

implications. It means that with a single, elegant and homogeneous algorithm we can now tackle (in principle) every learning problem and without any human expert intervention.

Multi-modal and *multi-task* algorithmic integration has never been easier and numerous are the attempt combining more than one domain at tackling complex tasks such as image captioning [Karpathy and Li, 2013], bimodal speech recognition [Hannun et al., 2014; Hinton et al., 2012] and syntheses [Simonyan et al., 2016], and many other more specific applications. However, despite this great advancement and great practical success of deep learning, much of its early progress was still in the *supervised* domain. This is why much of the recent efforts of the AI community have been devoted for enhancing deep learning in the unsupervised and reinforcement learning context with many impressive progresses [Mnih et al., 2013; Wang and Gupta, 2015].

But now we could certainly ask ourselves: *can current deep learning approach be the answer for improving our intelligent agents with respect to the five biggest manifestation of intelligence we outlined in the previous section?*

Recent advancements in reinforcement and unsupervised learning can be surely seen as major steps ahead in the *autonomy* of AI agents, and progresses in this areas seems to have not slowed down yet. The other properties, instead, seems to be far ahead for current deep learning techniques to grasp. For *adaptation*, even though in the past it has been tackled repeatedly over time and with different degree of success, very little has been done after the deep learning revolution [Long et al., 2015; Yosinski et al., 2014]. Only in recent years, after 2016, the focus of the community has started to value this attribute but with very limited success and underlining an evident need for novel approaches in this area [Parisi et al., 2018a]. The same could be said for *scalability*. Current deep learning algorithms are rarely considered as efficient and scalable with respect to biological systems, especially considering multi-domain, high-dimensional data streams, context in which, for example, the human brain excels [Marcus, 2018].

Since a big part of *Common sense* can be simply considered as the ability to *"contextualize"* information, we can safely affirm that deep learning has contributed significantly with this respect. In fact, being able to train a single model in a multi-modal and multi-task fashion and taking into account temporal dependences is surely something possible to tackle only thanks to deep learning [Kaiser et al., 2017]. However, long-term temporal dependences and efficient memory management more in general constitute a difficult challenge for current learners abilities. Finally *reasoning*, understandably constitutes one of the more difficult challenge for analytics approaches such as deep learning [Pearl, 2018]. Despite early attempts in recent years [Andreas et al., 2015; Santoro et al., 2017], we can still consider complex reasoning at a whole different level of complexity from what can be tackled today by deep learning algorithms.

Hence, current deep learning approaches seem to be insufficient for tackling many of the biggest questions in AI that still remain open. In which direction should we move to improve our ability to tackle them? Can we find (again) some inspiration from our biological counterpart? In the following sections we will try to answer to these questions.

1.1.2 Biological Learning Systems

Reverse engineering and uncovering the principles of intelligence from complex biological systems is a hard task. Our current understanding of animal brain circuits is, in fact, still very limited. This is due not only to the narrowness of our technologies for observing and recording the information signals flowing into this systems at the synapses level [Seymour et al., 2017], but also for the immense complexity of it and from which intelligent behaviors seem to emerge almost magically.

Many discussion and debates about the right level of abstraction and how much inspiration to take from biology have always been central to the world of AI [Lake et al., 2016; Russell and Norvig, 2016], but, after the deep learning revolution, this discussion has been particularly revitalized. This because, as already discussed in the previous section, deep learning can be already seen as loosely inspired by the human brain, where hierarchies of more and more abstract concepts have been demonstrated repeatedly, especially in the visual brain areas [Poggio and Riesenhuber, 1999].

On the other hand, taking inspiration from biology may not always be beneficial. An classic example often cited for appreciating this counter argument is the history of the early flying machines of the 19th century. In the *Musée des Arts et Métiers*, in Paris, is still possible to see the majestic primitive steam-powered aircraft *"Avion III"* built between 1892 and 1897 by Clément Ader (see Fig. 1.1). This aircraft prototype had a bat-like configuration, with biologically inspired wings and bone structures. Yet, it was unable to fly due to its weights. It was only in 1903 that the Wright brothers, after figuring out the principles of aerodynamics built the first successful powered airplane [Anderson Jr. et al., 1999]. Now we can fly over continents in a matter of ours, something which has never been achieved by any biological system before.

FIGURE 1.1: *"Avion III"* built between 1892 and 1897 by Clément Ader, exposed at the *Musée des Arts et Métiers* in Paris, France.

Nevertheless, many scientists are starting to raise concerns about the future of of back-propagation and gradient-based optimization techniques, the most successful and go-to learning algorithms

for current deep learning research, which suffer from severe limitations and open questions, especially in terms of efficiency and scalability. It is ironic that G. Hinton, one of the fathers of deep learning and among the authors of back-propagation [David E. Rumelhart et al., 1986], is also one of its harshest critic today, suggesting to *"throw it all away and start again"* [LeVine, 2017].

Spike-Timing-Dependent Plasticity (STDP), which is believed to be the main form of synaptic change in neurons [Gerstner et al., 1996], relates the expected change in synaptic weights to the timing difference between post-synaptic spikes and pre-synaptic spikes and it's considered to be extremely efficient (since local) and effective. Although it is the result of experimental observations in biological neurons, its interpretation, as part of a learning procedure that could explain learning in deep networks, remains unclear. In particular: (1) the back-propagation computation (coming down from the output layer to lower hidden layers) is purely linear, whereas biological neurons interleave linear and non-linear operations, (2) if the feedback paths known to exist in the brain (with their own synapses and maybe their own neurons) were used to propagate credit assignment by back-propagation, they would need precise knowledge of the derivatives of the non-linearities at the operating point used in the corresponding feed-forward computation on the feed-forward path, (3) similarly, these feedback paths would have to use exact symmetric weights (with the same connectivity, transposed) of the feed-forward connections, (4) real neurons communicate by (possibly stochastic) binary values (spikes), not by clean continuous values, (5) the computation would have to be precisely clocked to alternate between feed-forward and back-propagation phases (since the latter needs the former results), and (6) it is not clear where the output targets would come from [Bengio et al., 2015].

Many other solutions have been tried over the years [Cui et al., 2015; Ghosh-Dastidar and Adeli, 2009] but the question stays open: *is this a feature worth integrating and pursuing in our future learning algorithms?*

Nevertheless, synaptic plasticity is not the only brilliant feature we envy biological learning systems to posses. Efficient *memory management* is another issue for deep learning and many attempts has been made over the recent past for tackling the problem of factual learning and long-term memory [Graves et al., 2014; Weston et al., 2015].

The recent complementary learning systems (CLS) theory [McClelland et al., 1995], explains how the complex interplay of hippocampal and neocortical functionality is crucial to concurrently learn regularities (statistics of the environment) and specifics (episodic memories). Both brain areas are known to learn via *hebbian* and error-driven mechanisms [O'reilly and Rudy, 2001]. In the neocortex, feedback signals will yield task-relevant representations while, in the case of the hippocampus, error-driven modulation can switch its functionality between pattern discrimination and completion for recalling information [O'reilly, 2004]. Memory consolidation is also known to happen at various levels (other than at the synapses level) and at different timescales [Benna and Fusi, 2016; Clopath, 2012].

All these complex features of our brain we do not fully understand, are there for a reason or are just a product of a random evolutionary process? No one knows the answer to these questions

but a deep understanding of our biological learning system will sure be of great help moving towards strong artificial intelligence systems.

1.2 Continual Learning

Continual Learning is based on the simple, yet fundamental idea of learning continually over time [Chen and Liu, 2018; Ring, 1994; Thrun, 1996]. The basic intuition is that data are not aprioristically available, like generally assumed in machine learning research, but only in a time-delayed fashion. Acknowledging the dynamical nature of data with its volume, variety and velocity is, indeed, at the core of continual learning. This methodologically means focusing on learning techniques that can efficiently handle a possible unlimited stream of high-dimensional ever-changing data within *bounded computational and memory resources*, in order to maximize a pre-defined measure of performance at each point in time.

These, often ignored, real-world constraints put an hard limit to theoretically learnable problems. Indeed, intuitively, if A is a *Probably Approximately Correct* (PAC) learnable problem [Haussler, 1990] (i.e. we can reach an adequate level of performances in polynomial time), this may still not be learnable in practice if data are only available in small (under representative) portions over time (with an unknown idle time between them).

Moreover, being able to learn efficiently without access to future data (and possibly past data) pose a number of new challenging questions like the *stability-plasticity dilemma*, well-know to both artificial and biological learning systems [Mermillod et al., 2013]. Being able to integrate new knowledge, preserving the old one with the final objective of a greater generalization over time is not a straightforward process.

Nevertheless, continual learning may bring us surprisingly near to biological learning systems, which are known to cope with similar constraints and share the same goals. In the following sections, we will take a deeper look at continual learning, better motivating its necessity both from a theoretical and practical point of view, looking at what has been done over the past and discussing major challenges and opportunities ahead.

1.2.1 Motivation

Let us now contextualize the impact of continual learning with respect to the five characteristic traits of intelligence we discussed above.

The ability to *adapt* and *generalize* to new circumstances and environments is strictly related to the ability of learning *continuously*. In fact, adaptation may be framed as the ability to contextualize and specify already learned behaviors but also facing very different situations and data we have never seen before and from which we need to learn new behaviors, improving our ability to face them if encountered again in the future [Chen and Liu, 2018].

Adaptation capabilities may be useful for many circumstances. The simplest scenario would be the one in which input and output data distributions are invariant but new data become

available over time. A considerable amount of real-world applications (e.g. recommendation or anomaly detection systems), which are supposed to process a constant flow of new data over time, may benefit from learning continually to adapt and refine the prediction model and, in the end, improve the global performance of the model. On the same line of reasoning, only a relative modest amount of problems (also very constrained and well defined a priori) would not benefit from new data which becomes available later in time.

However, nowadays, for most of the practical or commercial deep learning application, re-training the model from scratch with the data accumulated over time is still a viable option. The challenge arises in scenarios which keep changing over time. This is where continual learning may have a profound impact and other techniques are facing strong issues. Most of the time, it is very hard to collect a large and representative dataset a priori, but it can be even impossible when the *"semantics"* of these data keeps changing over time (i.e. we are actually solving a different task, known as *"concept drift"* [Gepperth and Hammer, 2016]). For example, we can think to a reinforcement learning system in a complex environment in which the reward function keeps changing based on a hidden variable we can not observe nor model.

Adapting with respect to a continuous stream of data is also deeply connected with learning from non i.i.d. training data [Gepperth and Hammer, 2016; Hayes et al., 2018b; Pentina and Lampert, 2015]. Even tough this is fairly common if not the rule in nature, this assumption has always been present in machine learning and generally overcome by randomly shuffling data (after they have been collected) so that they are independent and identically distributed (i.i.d.). However, this assumes that possibility of collecting a big and representative dataset a-priori, which may be not realistic.

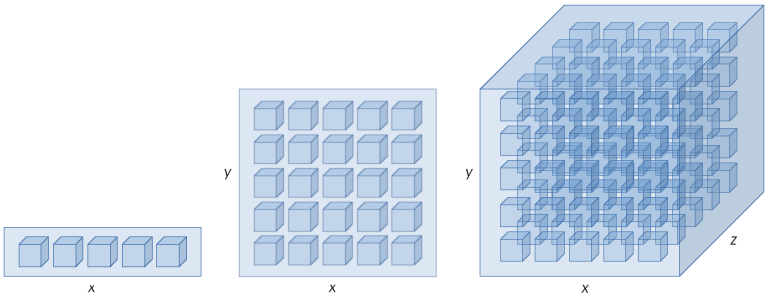


FIGURE 1.2: As the dimensionality increases, the space of all the possible input examples increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance and is often referred to as the *"curse of dimensionality"*. In the abstract representation the number of examples needed to cover the entire space starts from 5 and ends up with 125 different examples when we increase the number of dimension from 1 to 3.

Indeed, as we would like to tackle more and more complex problems, we are subject to what is known in literature as the *"curse of dimensionality"*. The expression, coined by Richard E. Bellman in the context of dynamic optimization, refers to the fast increase in the data space volume when the dimensionality of the problem increases, hence making exponentially harder to collect a representative dataset covering the entire space of possibilities (see Fig.1.2). Continual learning, is not a solution to this issue but rather a methodological approach to tame its

catastrophic implications. Like natural intelligence we do not expect to learn, once and for all, a definitive and perfect model of the universe, but rather to learn useful sub-spaces of it, only when we need them, trying to better consolidate and generalize our knowledge and skills over time.

FIGURE 1.3: Two examples of knowledge *bias* on state-of-the-art deep neural networks train on large training sets. On the left four automatically generated images that maximize the activation of the *barbell* class for a standard Imagenet classification task are reported. As it is possible to see in the pictures, the notion of *barbell* is biased to include visual features of human arms. On the right, a captioning neural network mistakenly description is reported for an uncommon picture of a flooding which is misunderstood as a beach given the biased over-representation of seaside landscapes in the training set (image originally contained in [Lake et al., 2016]).

"All models are wrong but some are useful" is a well-known aphorism in the statistics and machine learning community, and firstly introduced by Box [1979], essentially emphasizing the fact that every model is a biased simplification or approximation of reality. In Fig. 1.3 two examples of knowledge *bias* on state-of-the-art deep neural networks are reported. Even though trained on millions of images, the networks show a knowledge bias that is directly related to the bias present in the data. In the first example (on the left) of a standard classification task with 1000 classes, four automatically generated images that maximize the activation of the *barbell* class are reported. As it is possible to see in the pictures, the notion of *barbell* is biased to include visual features of human arms as most of the images present in the training set related to this class [Russakovsky et al., 2015]. On the right, instead, a captioning neural network mistakenly description is reported, where an uncommon picture of a flooding is misunderstood as a beach given the biased over-representation of seaside landscapes in the training set. These are just two examples of specific sub-volumes which are not well-represented despite the breadth of the training datasets.

Hence, how can we ensure that our artificial systems *scale* in terms of intelligence (while processing more and more data) and are *efficient* (i.e. maintaining computational/memory bounded) like our brain? This is critical in nature since the more efficient is the process of learning, the faster we can adapt to new circumstances and memorize important information which may be useful for survival. The same can be said about machine intelligence.

One of the key hallmark of continual learning is to process data only once without the need of storing them for later re-processing [Parisi et al., 2018a]. Like biological systems storing

perception data (given their high-dimensionality and noise rate) would be impossible to maintain and process cumulatively on a long time scale. The continual learning approach may be more appropriately interpreted as a learning system which *filters* perception data and retain only the most important information.

While is not clear if in the future the growth in computational power will exceed the growth in data production rate, it is a possibility worth considering. A study conducted by Reinsel et al. [2017] pointed out that by 2025, data generation rate will grow from the 16 ZB per year (zettabytes or a trillion gigabytes) we register today, to 160 ZB. The projections estimate that only between the 3% and 12% of the total amount of data produced we will be storable. This would mean that continual learning may not only constitute a more scalable and efficient solution but the only possible way of learning. On the other hand, even without considering futuristic scenarios, data may be not storable due to legal, security or privacy obligations. For these data, which we can call *ephemeral*, the idea of not storing data is not only motivated by reasonable efficiency assumptions but actually demanded by the application itself.

However, continual learning is not only about data that cannot be stored. The same problem can arise with very big dataset for which the cost of recovering and re-processing the same data multiple times may be too high. An example may be the difficulty on training an object classifier on datasets like ImageNet [Russakovsky et al., 2015] for which memory on common GPU hardware is generally insufficient to contain the whole training set and a moving data from the hard disk to the GPU is a strict bottleneck. In this case, being able to process data once (or less times as possible) would greatly improve learning efficiency and reduce computation time.

With high-dimensional and high-velocity streaming data (around 25% of the global datasphere in 2025 [Reinsel et al., 2017]) the problem appears even clearer since it would become impractical or even impossible to keep the data in memory and re-training the entire prediction model from scratch as soon as a new piece of data becomes available. CL is ideal for streaming perceptual data since it embeds the idea of continually updating the model with the new available data. Of course, in a supervised setting it would be very hard to couple real-time perception data with a supervised signal (i.e. labels), however, in an unsupervised, semi-supervised or reinforcement setting learning continually becomes quickly more appealing.

If we do not have a single stream of perception data but many of them coming from different sensors (with different input modalities) and at the same time we would like to tackle multiple tasks, continual learning may become even more interesing solution. Kaiser et al. [2017], from Google Brain, recently shown for the first time that is possible to learn, with a single deep learning model, very different tasks in very different domains (live vision and language) and with many input modalities. The model has been trained on a composition of huge training sets for each of the task considered. Updating such a model may be practically impossible in a reasonable amount of time, especially in real-world applications and with current DL techniques since requiring to re-train the entire model from scratch as soon as a new piece of data is available from one of the many input streaming sources.

Yet, we regard at multi-modal and multi-task Learning as essential ingredients towards the creation of strong AI systems, endowing machines with *common sense* and basic, implicit, *"reasoning"* skills. In Fig. 1.4 a very famous and somehow hilarious mistake made by an *Automatic Image Captioning* system [Karpathy and Li, 2013] based on state-of-the-art deep learning techniques is reported.

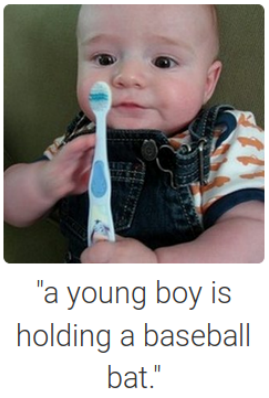


FIGURE 1.4: One of the mistakes made by the Multi-modal Captioning Network firstly proposed by Karpathy and Li [2013].

In this case, the Multi-Modal *Recurrent Neural Network* (RNN), based on the training set composed of multiple <*image, caption*> pairs, had wrongly identified the *toothbrush* as baseball bat. However, why this mistake arouse hilarity and amazement from a human perspective? Possibly because, as humans, we regard at the concept of a child holding the weight of a baseball bat be highly unlikely. As we regard as unlikely that an object of that relative dimension could be identified as a baseball bat.

We argue that all these basic inferences (which can be intended as a simple version of *reasoning*) could be also a identified as a good portion of what we have defined as *common sense*. In fact, if in the same system, other than just captioning images, we would have also trained the model to evaluate more precisely the age of a person in the picture and trying to infer the weight/size of each particular object in a scene, disambiguating the toothbrush from the baseball bat would have become much easier, since the co-occurrences of a very young boy holding that weight and an object of that relative size being classified as a baseball would be much less frequent. Of course, for even more complex tasks, multiple input modalities are also needed (e.g. disambiguating type of birds based on visual but also auditory cues).

Hence, we believe, multi-modal/multi-task learning could really have a strong impact on the creation of smarter AI systems but only if enabled through continual learning algorithms, which essentially make *asynchronous*, *alternate training* of such tasks possible, updating the model on the real-time data available from one or more streaming sources in a particular moment.

Finally, while not at its core definition we could also consider continual learning as a matter of interest for *reasoning*. In fact, even if reasoning is generally focused on the idea of inferring new knowledge from data, this process can rarely considered as static [Pham and Dimov, 1970; Reinke and Michalski, 1985]. The availability of novel pieces of evidence may trigger not only

a continual reasoning process but also the consolidation and integration of knowledge which is one of the core aspect of state-of-the-art research in continual learning today.

Even considering the young and unripe state of continual learning research today, we believe that early results shown by the research community in recent years and in this dissertation, point it out as a direction worth pursuing in the integration of the most characteristic traits of intelligence in our future AI systems.

1.2.2 An Example

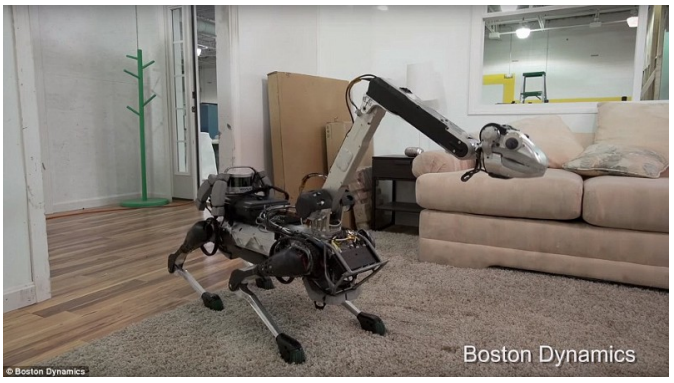


FIGURE 1.5: *SpotMini*, the new smaller version of the Spot robot from *Boston Dynamics* weighing around 14 kg [BostonDynamics, 2018].

An example of the real need of a continual learning system can once again be made in the context of robotics [Thrun and Mitchell, 1995]. As shown by Thrun [1996] in is doctoral dissertation real-world embedded robotic settings, are in strong needs of learning over time, specializing and adapting their behaviors locally (operating off-line) and efficiently, depending on the specific tasks and environment in which they are supposed to operate.

Indeed, is it clear that in this scenario, collecting data beforehand which can be representative for all the possible situations and tasks she may encounter is rather difficult if not impossible. Giving the high-dimensional, multi-modal and streaming nature of the perceptual data which will flow through the robot sensors and cameras (valued around 50 GB/s for humans [Dispenza, 2008]), with state-of-the-art hardware capabilities, would be impossible other then incredibly inefficient to think of collecting all the data during the day (around 4,320 TB) and re-training the entire robotic brain from scratch each night with the accumulated data acquired until then (especially if we want her to stay in its environment indefinitely). Indeed, just after a week we would already have encountered around 30,240 TB of data from which to train our model the following night. This is why learning continually and adaptively about the external world in this setting may be not only practical but essential: what are needed are scalable and efficient techniques which (as for biological learning systems) can learn online for the autonomous incremental development of ever more complex skills and knowledge.

1.2.3 Brief History

The concept of learning continually from experience has always been present in artificial intelligence and robotics since their birth [Turing, 1950; Weng, 2001]. However, it is only at the end of the 20th century that it has began to be explored more systematically. Within the machine learning community, the lifelong learning paradigm has been popularized around 1995 by Thrun [1996]; Thrun and Mitchell [1995] and Ring [1994]. Since then it has been researched in four main areas. Here we give a brief history of the CL research in each of these areas.

Continual Supervised Learning. Thrun [1996] was one of the first to study continual learning within a supervised context, where each previous or new task aims at recognizing a particular concept using binary classification. Several CL techniques were then proposed in the contexts of memory-based learning and artificial neural networks. The neural networks approach was improved by Silver and Mercer [2002]; Silver and Poirier [2004]. Ruvolo and Eaton [2013b] proposed an *"Efficient lifelong learning algorithm"* (ELLA) to improve the multi-task learning method proposed by Kumar and Daume III [2012]. Here the learning tasks are independent from each other and a regularization strategy based on the *Fisher Information* was firstly introduced. Ruvolo and Eaton [2013a], however, were among the first who considered ELLA also in an active task selection setting. Cheng, Hao and Fang, Hao and Ostendorf [2015] further proposed a continual learning technique in the context of *Naïve Bayesian* classification. A more theoretical study of continual learning was firstly accomplished by Pentina and Lampert [2015] within the PAC-learning framework.

Continual Unsupervised Learning. While intuitively better suited for unsupervised learning, continual learning research in this area have not been extensive and mainly focused on topic modeling and information extraction. Chen and Liu [2014a,b] and Wang et al. [2016] proposed several continual topic modeling techniques that extract knowledge from topics produced within many previously encountered tasks and use it to help generate better topics in the new tasks. Liu et al. [1999] proposed a continual learning approach based on recommendation for information extraction in the context of opinion mining. Shu et al. [2016], instead, proposed a continual relaxation labeling method to solve a unsupervised classification problem.

Continual Semi-Supervised Learning. The work in this area is well represented by the *Never-Ending Language Learner* (NELL) system by Carlson et al. [2010]; Mitchell et al. [1998], which has been reading the Web continuously for information extraction and learning since January 2010, and it has accumulated millions of entities and relations.

Continual Reinforcement Learning. Mitchell and Thrun [1993] first proposed some CL algorithms for robot learning which tried to capture the invariant knowledge about each individual task. Tanaka and Yamamura [1997] treated each environment as a task for continual learning. Ring [1994] proposed a continual learning agent that aims to gradually solve complicated tasks by learning easy tasks first withing an extensive and general approach to continual reinforcement learning. Wilson et al. [2007] proposed a hierarchical Bayesian continual reinforcement learning method in the framework of *Markov Decision Process* (MDP). Fernández and Veloso [2013], instead, specifically worked on policy reuse in a multi-task setting. A nonlinear feedback policy that generalizes across multiple tasks was proposed in [Deisenroth et al., 2014]. Ammar et al.

[2014] proposed, instead, a policy gradient efficient continual learning algorithm following the idea presented with ELLA [Ruvolo and Eaton, 2013b]. This work was further enhanced with cross-domain continual reinforcement learning by Ammar et al. [2015a] and with constraints for safe continual reinforcement learning [Ammar et al., 2015b].

Continual Learning techniques working in other areas also exist. For example, Kapoor and Horvitz [2009] studied predictive user modeling under continual learning, and worked on managing and using user feedback with the help of CL. Silver et al. [2013] wrote a survey of continual learning trying to encourage more researchers to work in this area.

As we can see, although continual or incremental learning has been proposed for more than 20 years, research in the area has not been extensive. At this point a question normally arise: *why continual learning despite its intuitiveness and naturalness is becoming a solid interest of the machine learning and AI community only now?* As we already hinted in the introduction, there were more complex and fundamental problems to solve before the deep learning revolution and a number of additional constraints:

- *Lack of a systemic approach*: machine learning research for the past 20 years has focused on statistical and algorithmic approaches on simple tasks. CL typically needs a systems approach that combines multiple components and learning algorithms. Moreover, continual learning greatly complicate training and evaluation procedures. Disentangling *"static"* learning performance from continual learning side effects was important for the very incremental nature of the research in this area.
- *Limited amount of data and computational power*: digital data are a luxury of the 21th century. Before the big data revolution collecting, processing data was a daunting task. Moreover, the limited amount of compute power available at the time, did not allow complex and expensive algorithmic solution to run effectively, especially in a continual learning setting which undoubtedly makes learning more complex dealing with multiple task at the same time and incorporating the concept of time into the learning process.
- *Manual engineered features and had hoc solution*: Before early 2000 and early works on representation learning creating a machine learning system would mean to handcraft features and finding had-hoc solution which may differ significantly depending on the task or domain [Russell and Norvig, 2016]. Having a general algorithm for a more systemic approach seemed for a long time a very distant goal.
- *Focus on supervised learning*: creating labeled data is probably the slowest and the most expensive step in most machine learning 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 and the rapid technological progress witnessed in the last 20 years, have opened the door for starting tackling more complex problems like learning continually.

In the following chapters we will focus on recent continual learning developments in the context of deep learning, as generally known from 2012. For a more detailed description of many other

classic approaches to continual learning with shallow architectures please refer to [Chen and Liu, 2018].

1.2.4 Related Learning Paradigms

Continual learning key characteristics like explicit knowledge retention and accumulation or the use of the previously learned knowledge to help new future learning are not exclusive to this paradigm. To many different extents, there are several machine learning paradigms that have related characteristics and goals. This section discusses the most related ones, i.e., *transfer learning*, *multi-task learning*, *online learning*, *reinforcement learning*, *meta-learning*, *curriculum learning*, *sequence learning* and their difference from continual learning. The first two paradigms are more closely related to CL since they both involve some form of knowledge transfer across domains or tasks, but they do not aim at learning continually and do not retain or accumulate learned knowledge in any explicit way. Online learning and reinforcement learning involves continual learning processes but, most of the times, they focus on the same learning task and posses some peculiarities. These differences will become clearer after the review of some representative techniques for each of these related learning paradigms.

Transfer Learning. *Transfer learning* is a popular research topic in machine learning and especially deep learning. It is also commonly known as domain adaptation in computer vision, natural language processing and many other domains. It usually involves two domains: a source domain and a target domain. Although there can be more than one source domain, in almost all existing research only one source domain is used. In typical transfer learning settings the source domain has a large amount of labeled training data while the target domain has little or no labeled training data. The aim of transfer learning is to use the labeled data of the source domain (and possibly a mode pre-trained on it) to help learning in the target domain [Long et al., 2015; Pan et al., 2010; Taylor and Stone, 2009].

Transfer learning is different from continual learning in the following aspects. Firstly, transfer learning is not concerned with continual learning or knowledge accumulation. Its transfer of information or knowledge from the source domain to the target domain is accomplished in a single step in time. Moreover, It *does not* retain the transferred knowledge or information for future use, meaning that the ability to solve the task in the source domain is generally lost or ignored. Knowledge retention and accumulation are essential for continual learning as they not only enable the system to become more and more knowledgeable over time, but also allow it to learn additional knowledge and skills more accurately and easily in the future. Another distinction is that generally transfer learning is unidirectional. It transfers knowledge only from the source domain to the target domain, but not on the opposite direction since the target domain has little or no training data. However, in CL, learning results from the new domain or task can be used to improve learning in previous domains or tasks if needed. Moreover, in transfer learning it is generally assumed that that the source domain is very similar to the target domain (otherwise the results may be detrimental) and two similar domains are usually selected by human experts. CL, on the other hand, normally considers a large (possibly unlimited)

sequence of unknown tasks/data. When solving a new problem, the learner is able to exploit past knowledge and skills in current learning.

Multi-Task Learning. *Multi-task learning* learns multiple (often related) tasks simultaneously, aiming at achieving a better performance by using the relevant information shared by multiple tasks [Caruana, 1997]. The rationale is to introduce inductive bias in the joint hypothesis space of all tasks by exploiting the task relatedness structure. This helps also to prevent overfitting of the individual task and thus gaining better generalization capabilities. Unlike transfer learning, we mostly use the term *multiple tasks* rather than *multiple domains* as much of the existing research (especially before deep learning) in the area is based on multiple similar tasks from the same domain of application.

The similarity of multi-task learning and continual learning is that they both aim to use some shared information across tasks to help learning. The difference is that multi-task learning is still working in the traditional paradigm. Instead of optimizing a single task, it optimizes several tasks *simultaneously*. If we regard the several tasks as one bigger task, it reduces to the traditional optimization which is actually the case in most optimization formulations of multi-task learning. Although we could argue that multi-task learning can jointly optimize all tasks whenever a new task is added, as we have described in the previous section, optimizing all tasks simultaneously rather than asynchronously would mean retaining all the training data encountered so far, making it impractical and inefficient for many applications and long-term scalability goals.

Meta-Learning. *Meta learning* is a learning process that uses meta-data about the past experiences in order to improve its capacity of learning on new experiences. It is also called *"learning to learn"*, and can be very much related to continual learning [Andrychowicz et al., 2016; Thrun and Pratt, 2012]. Indeed, it entertains the same central idea that learning is not a static process but rather a continuous process. However, while it seems to share an important part of the continual learning objectives, operatively, much of the research done in this area has been devoted to speed-up learning on fixed datasets or on specific target domain without considering anymore the source domain like in transfer learning. One constant of this approach is the presence of a dual system, one for actually learning and the second to guide the learning process.

Online Learning. *Online learning* is a learning paradigm where the training data are processed one example at a time. When a new data point arrives, the existing model is quickly updated to produce the best model so far. Its goal is thus the same as classic learning, i.e., to optimize the performance on the given learning task. However, it is normally used when it is computationally infeasible to train over the entire dataset or learning with mini-batches or it is impractical for hardware constraints or data availability. Online learning methods are typically memory and runtime efficient due to the latency requirement in a real-world scenario. However, online learning per se, does not imply that data are not processable twice as generally assumed in continual learning even though the term does not exclude either this option.

There are a large number of existing online learning algorithms. For example, Kivinen et al. [2004] proposed some online learning algorithms for kernel-based learning like SVM. By extending the classic stochastic gradient descent, they developed computationally efficient online learning

algorithms for classification, regression, and novelty detection. Mairal et al. [2009] proposed some online dictionary learning approaches for sparse coding, which model data vectors as sparse linear combinations of some basic elements. Hoffman et al. [2010] also proposed an online variational Bayes algorithm for topic modeling. Much of the online learning research focuses on one domain/task. Dredze et al. [2008] developed a multi-domain online learning method, which is based on parameter combination of multiple classifiers. In their setting, the model receives a new instance/example as well as its domain.

Although online learning deals with streaming of data, its objective is very different from continual learning. Online learning still performs the same learning over time but its objective is rather to learn more efficiently when a new piece of data becomes available. Continual learning, on the other hand, aims at learning from a sequence of different batches/tasks, retaining the knowledge and skills learned so far, and using the knowledge to help future task learning.

Reinforcement Learning. *Reinforcement Learning* [Sutton et al., 1998] is the problem where an agent learns actions through trial and error interacting with a dynamic environment. In each interaction step, the agent receives as input the current state of the environment and a possible reward. The agent then has to choose an action from a set of possible actions. The action changes the state of the environment. This process keeps repeating as the agent learns a trajectory of actions which optimize its objective. The goal of reinforcement learning is to learn an optimal policy that maps states to actions and maximizes the *future expected reward* [Sutton et al., 1998; Wiering and Van Otterlo, 2012].

Transfer learning and multi-task learning have also been applied to reinforcement learning. Banerjee and Stone [2007], for example, demonstrated that feature-based value function transfer learning learns optimal policies faster than without knowledge transfer. Taylor et al. [2008] proposed a method to transfer data instances from the source to the target in a model-based reinforcement learning setting. An excellent survey of transfer learning applied to reinforcement learning can be found in [Taylor et al., 2008].

A reinforcement learning agent learns by trial and error in its interactions with the environment which would appear, by definition, a continual learning process. However, most of the times, learning is limited to one task and one environment without the explicit accumulation of knowledge to help future learning tasks. Moreover, environments are often stationary, loosing the need for more specific and explicit incremental learning and adaptation algorithmic capabilities given that pretty much all the correlation patterns are automatically refreshed by the environment. Transfer and multi-task reinforcement learning paradigms have similar differences from continual learning as supervised transfer and multi-task learning discussed above.

Curriculum Learning. Curriculum learning [Bengio et al., 2009] is a training process that proposes a sequence of data/tasks to a learning algorithm in order to make it able to learn, at last, a generally harder task. Both CL and curriculum learning learn on a sequence of tasks (or partial experience). However, in curriculum learning, task are chosen and structured in a way that make possible to learn the last more efficiently, by taking into account the different difficulty and functional dependences among them, while in CL, tasks are not voluntarily chosen

nor ordered. Furthermore, in CL the algorithm is interested on being able to solve all tasks at the end of the training process, and not only the last one.

Sequential Learning. *Sequential Learning*, often also called *"time-series forecasting"* or *"predictive learning"* is different from other types of supervised learning problems [Sutskever et al., 2014]. The sequence imposes an order on the observations that must be preserved when training models and making predictions. Generally, prediction problems that involve sequence data are referred to as sequence prediction problems, although there are a suite of problems that differ based on the input and output sequences.

What we have described under the name of continual learning is now a fast emerging topic in AI which have been often branded as *Lifelong Learning* or *Continuous Learning* and the terminology seems not well consolidated yet in the machine learning research community. The term *"Lifelong Learning"* has been around for years in the AI community, but prevalently used in areas somehow distant from the field of Deep Learning [Chen and Liu, 2018]. This is why a more recent research trend refers to this setting as *"Continuous"* or *"Continual Learning"* targeting specifically Deep Learning algorithms [Parisi et al., 2018a]. *"Continuous Learning"* makes explicit the idea of a smooth and continuous adaptation process that never stops. The distinction with continual is subtle but important as beautifully put in the *Oxford Dictionaries*[1]:

Both can mean roughly "without interruption" [...] however, Continuous is much more prominent in this sense and, unlike Continual, can be used to refer to space as well as time [...]. Continual, on the other hand, typically means "happening frequently, with intervals between" [...].

Even though current research focuses on rigid task sequences problems where we actually stop learning at the end of each task, we argue that *"Continuous Learning"* would be more appropriate in the long term with the developments of algorithms which can deal with a continuous stream of perception data like the real world. On the other hand, the term *"Continuous"* may result too confusing in many contexts (especially in Reinforcement Learning) as often used as the opposite of *"Discrete"*. This is why the DL community seems to start converging to the use of the term *"Continual"* instead.

The term *"Online"*, as we have seen in the online learning related paradigm can be considered as opposed to *"Batch Learning"* with the technical acceptation of processing data in an algorithm rather than a paradigm of learning [Cui et al., 2015]. The term *"Incremental Learning"*, instead, while still focuses on the idea of building knowledge incrementally, doesn't really express the idea of adaptation which sometimes means also to temper or erase what has been previously learned.

1.2.5 State-of-the-art and Current Challenges

While not already at its explosion, Continual Learning has been getting more and more attention in the deep learning community over the last few years with key contributions over a short period of time ([Kirkpatrick et al., 2017; Li and Hoiem, 2016; Rebuffi et al., 2017; Zenke et al.,

[1] https://en.oxforddictionaries.com/usage/continual-or-continuous

2017]). The biggest problem faced today by continual learning algorithms is known in literature as *Catastrophic Forgetting* or *Catastrophic Inference* [French, 1999; McCloskey and Cohen, 1989]. Neural networks almost often trained with gradient-based optimization methods suffer dramatically from this problem, experiencing a rapid overriding of the model parameters when learning from different data distributions over time. This is why almost every recent work in the context of deep continual learning has been focusing on such a problem, one of the biggest obstacle to the adoption of AI systems that learn continually.

Contrasting catastrophic forgetting is possible in many ways, and not only through careful hyper-parametrizations or basic regularization techniques. As we will later discuss in Chapter 3, many different strategies have been proposed, showing, with different degree of success, that CL can be used in complex domains like computer vision and natural language modeling [Parisi et al., 2018a]. Early results in this area are promising, even though still to be proven over a long sequence of batches or tasks.

Much of the attention over the last few years in deep continual learning research has been devoted to the *multi-task* scenario where there is a clear distinction between the tasks encountered over time. However, as discussed in the previous sections, many are the scenarios where learning continually does not necessary imply learning a sequence of tasks in rigid separation. A good portion of this dissertation has been devoted instead in the development of novel continual learning algorithms that can work essentially without the notion of task, in what have been called in Chapter 2 *Single-Incremental-Task* scenarios.

Given the relative novelty of the subject into the deep learning community, another important issue in continual learning research is the difficulty on finding a consensus in defining common constraints and desiderata for developing and evaluating continual learning algorithms. In Chapter 2 and Chapter 4, we propose a comprehensive continual learning framework, novel benchmarks, protocols and metrics to help addressing this issue.

1.2.6 Applications

While not the focus of this dissertation, for further motivating the practical interest in continual learning, a short review of the large number of domains in which continual learning could have an impact is here summarized. Applications accounting for streams of data (e.g. applications running on smart-phones devices) or any other kind of real-time ephemeral signal that results impractical to store and re-process are the ones which would benefit the most from the integration of CL features. A non-comprehensive and unordered list of applications in which continual learning may be beneficial or has been already applied con be found below:

- **Computer Vision**: given the high-dimensionality and high-velocity of visual information, computer vision tasks are one of more suitable domains to prove the importance of continual learning and to actually benefit from it also from a practical point of view. *Object detection*, *recognition* and *segmentation* [Shmelkov et al., 2017] are simple examples of horizontal applications which are in high need of more efficiency and scalability, often dealing with limited hardware resources (e.g. smart cameras) and with the necessity to customize and

adapt (possibly offline) their behaviors over time (e.g. for surveillance purposes or for providing better, customer-centered specialized services).

- **Natural Language Processing and Speech Recognition**: after a period of early enthusiasm and subsequent disappointment during the second AI winter, *conversational agents* (or *chatbots*) [Lee, 2017] and *virtual assistants* are slowing regaining ground in the AI applications landscape. Their latest incarnations in Siri, Alexa, Google Now, Cortana, etc. are showing today rapidly growing application scope and success [Omale, 2019] mostly due to the recent improvements in speech recognition and natural language processing. Continual learning may substantially improve the human-to-machine interaction through efficient on-device personalization/adaptation. This may not only reduce the computational burden on the server side (and improve the adaptation speed), but given the highly personal nature of the information being processed by the virtual assistants, it may also force the raw data to never leave the device.

- **Robotics**: the robotics community has always been intrigued by endowing embodied machines with lifelong and open-ended learning of new skills and new knowledge and many are the scenarios which would highly benefit by recent CL advances. Robotics applications in unconstrained environments, indeed, have always posed questions out of reach for previous machine learning techniques often dealing with unpredictable situations. Classic continual learning setting include room navigation, e.g., using a HERO-2000 mobile robot with a radar sensor [Thrun, 1996] to perform several room mapping and navigation tasks. Action models in *Explanation-Based Neural Network* (EBNN) learning *explain* (in terms of previous experiences) and analyze observations to transfer task-independent (navigation) knowledge via predicting collisions and the prediction certainty. In the most recent literature, estimation and tracking in [Wong, 2016], odometry estimation, mask or pixel-wise segmentation in [Pinto and Gupta, 2016] have been also tacked, especially through self-supervision. However, most of these works were not conceived within the motivating principles of CL. *RL Intelligent Adaptive Curiosity* (RL-IAC) constitutes one of the few examples of the direct application of CL in a robotics setting for visual saliency learning Craye et al. [2018]. However, the proposed algorithm does not employ deep architectures.

- **Internet-Of-Things and Edge Computing**: embedded devices with highly constrained hardware resources and operating off-line (due to privacy or operational reasons) may highly benefit the introduction of more efficient learning algorithm operating on real-time data and without the need of storing them. The domestic robot example introduced in the previous section, already gave some pragmatic motivations of the need of continual learning in this area. However, many are the vertical applications we could mention, like *transportation-mode detection* [Carpineti et al., 2018] and *activity recognition* [Ravi et al., 2005] on smart-phone devices using strong (and highly private) sensor signals.

- **Machine Learning Production Systems**: machine learning production systems are becoming more and more common in every organization. Being able to fast train and deploy new prediction models over time becomes essential to provide up-to-date and always improving services. *Tensorflow Extended* [Baylor et al., 2017] constitutes an example of such systems supporting the Google machine learning infrastructure. Recommendation

and anomaly prediction systems are just two examples of application which are currently benefiting from a sophisticated prediction models management system. Continual Learning, in this scenario, may substantially reduce the computational burden incurred by such systems in re-training models from scratch every time (and possibly for every user) at a massive scale with a direct impact on resources occupation, energy consumption and ultimately financial resources.

2

A Comprehensive Framework for Continual Learning

> *"Without the Lifelong Learning capability, AI systems will probably never be truly intelligent: learning machine or agent to continually learn and accumulate knowledge, and to become more and more knowledgeable and better and better at learning."*
>
> – Bing Liu, 2014

In this chapter, we will try to define continual learning a little more formally in a comprehensive framework and with additional constraints and desiderata which will lay the formal foundations for the original proposals of the following chapters. Let us start with a simple question: *what is continual learning?* Drawing inspiration from the famous definition of *Machine Learning* by Michalski et al. [2013], we could try to summarize continual learning, operatively, in a single sentence as in the following definition.

Definition 1. Continual Learning. *A computer program is said to learn continually from experience if, given a sequence of ephemeral partial experience E_i, a target function h^* and performance measure P, its performance in approximating h^* as measured by P improves with the number of processed partial experience E_i.*

The focus is on the *ephemeral* nature of the data, which cannot be processed multiple times and the basic notion that, taken in isolation, they constitute only a *partial* amount of the information needed to approximate the target function h^*, the objective of the learning process. These natural but key constraints, as we have argued in the previous chapter, leads to profound theoretical and practical implications worth considering in the development of truly intelligent artificial systems.

2.1 Formal Definition

Early theoretical attempts to formalize the continual learning paradigm can be found in Ring [2005]. More general framework proposals include [Pentina and Lampert, 2015]. As in [Pentina and Lampert, 2015], we assume CL is tackling a PAC learnable problem in the approximation of a target hypothesis h^* but learning from a sequence of non i.i.d. training batches. Our framework could also be seen as a generalization of the setting proposed in [Lopez-paz and Ranzato, 2017], where a *"task supervised signal"* t is provided along with each training example.

In both settings, if we were capable to observe all data streamed throughout a lifetime, the distribution we would like to model would be just one and we could consider all the example being drawn from it. However, the actual reality of CL settings is that the total amount of training examples are never observed at once, but can be rather seen as drawn from a *sequence* of distributions D_i. In this section we expand and refine previous CL frameworks improving flexibility and generalization but also trying to not end up with a too abstract setting. Morever, we make sure to accommodate previously proposed algorithms and more recent ones with a number of constraints and relative relaxations and desiderata.

Definition 2. Continual Learning Algorithm. *Given X and Y as input and output random variable respectively, let us consider $\mathcal{D}$ a potentially infinite sequence of unknown distributions $\mathcal{D} = \{D_1, \ldots, D_n\}$ over $X \times Y$, we encounter over time (hence with $n \in [2, \ldots, \infty[$). A continual learning algorithm A^{CL} is an algorithm with the following signature:*

$$\forall D_i \in \mathcal{D}, \qquad A_i^{CL} : \ < h_{i-1}, B_i, M_{i-1}, t_i > \rightarrow < h_i, M_i > \tag{2.1}$$

Where:

- 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, void if not provided. It can be used to disentangle tasks and specialize the hypothesis parameters, as it is done in [Lopez-paz and Ranzato, 2017].
- B_i is the training batch of examples. For simplicity, these examples can be assumed to be drawn i.i.d. from D_i [Lopez-paz and Ranzato, 2017; Pentina and Lampert, 2013] but it is not necessary. Indeed, 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 [Gepperth and Hammer, 2016; Hayes et al., 2018b]. Each D_i can be considered as a stationary distribution.
- Each B_i is composed of a number of examples e_j^i with $j \in [1, \ldots, |B_i|]$. Each example $e_j^i =< x_j^i, f_j^i >$, where f^i is the feedback signal and can used to infer the optimal hypothesis $h^*(x, t)$ (i.e., exact label y_j^i in supervised learning or any real tensor from which we can estimate $h^*(x, t)$, such as a reward r_j^i).

2.2 Task Notion

Definition 3. Task. *A task T is defined by a unique task label $\hat{t}$ and its target function $g^*_{\hat{t}}(x) \equiv h^*(x, t = \hat{t})$, the objective of its learning.*

Please note that if t is not given as input to the CL algorithm A^{CL}, hence being all $t_i = \emptyset$, it is like having a single incremental task T where $g^*_t \equiv h^*$ (see Section 2.4).

Disentangling the notion of *task* from training batch is important in CL since data are not available all at once, but may be as well related to the same learning objective g^*_t as defined by the external supervised signal t. Hence, in our definition, even if D_i represents a different distribution from D_j for $i \neq j$, this does not necessary define a different task.

Removing the bijective correspondence between distributions and tasks it is important and convenient in many applications for improving the autonomy of the learning system with a more abstract (and potentially more natural) task supervised signal. For instance, in the robotic application *ball-in-cup*[1], tasks are defined by different lengths of the rope to which the ball is attached (defining different data distributions) [Stulp et al., 2014]. However, if we do not plan to specify to the robot the length of the rope every time (through different t labels) we may as well regard it as the single task of solving the *ball-in-cup* problem.

Another example is the *rotation MNIST* benchmark [Lopez-paz and Ranzato, 2017] often used for assessing CL strategies as we will see in Chapter 4: in this case the objective is to classify the 10 MNIST digits learning over a sequence of distributions determined by a fixed degree rotation of each image in the dataset. This can be seen as learning over a sequence of different tasks as well as the single task of classifying the 10 digits with a reasonable amount of invariance and generalization capabilities.

It really depends on the availability of the task supervised signal during training and inference. The t signal is particularly useful for correlating very different distributions (related to the same task) and, on the contrary, disentangle similar distributions related to different tasks for which a specialization rather than generalization of behaviors may be important for performance gaining.

2.3 Constraints, Relaxations and Desiderata

Having formalized a general notation for a continual learning algorithm, let us define some informative constraints that can characterize this paradigm more precisely.

Constraint 1. External Memory. *For every step in time, the number of training examples contained into the external memory is substantially lower than the total number of previously encountered training examples:* $\forall i \in [1, ..., n], |M_i| \ll \left| \bigcup_{i=1}^{i-1} B_i \right|$

In fact, if we could fit all previous examples in M, it would not be an interesting CL setting. Having defined an upper bound with respect to the number of examples being storable in the

[1] In this task the robot must make the ball go inside a cup without touching it, by holding the cup attached through a rope to it.

external memory M, we propose to constrain also the hypothesis h memory size as well as the number of operation for scalability.

Constraint 2. Memory and Computation. *Memory and computation for each iteration step i are bounded. Given two functions $ops()$ and $mem()$ computing the number of operations and memory occupation required by A_i^{CL}, it should exist two reasonably small values max_ops and max_mem, 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 bound for each iteration i is an important constraint for a continual learning algorithm. The reason is that the number of training sets B_i can be potentially unlimited and computation and memory should not be proportional to the number of the hypothesis h_i updates over time. However, in this case, we do not put a rigid upper bound a priori but just consider that an apper bound *should exist* and should be considering, especially with $n \to \infty$.

Given the difficult setting and the additional constraints imposed by continual learning with respect to the classic *"static"* learning setting, many researchers in the recent literature have proposed new CL strategies in slightly relaxed but still reasonable settings:

Relaxation 1. Memory relaxation. *Removes the fixed memory bound over $mem(h_{i-1}, M_{i-1})$.*

Relaxation 2. Computation relaxation. *Removes the fixed computational bound over $ops(A_i^{CL})$.*

In both cases we assume that for practical applications, a finite (and reasonable) number of tasks n are encountered, hence, for many settings with a generous memory and computational bound, many continual learning strategies that grows somehow proportional to the number of batches B_i in term of complexity and memory usage may still be a viable option, especially if they can guarantee better performance. On the other hand, having defined a formal framework with two important constraints we can also point out a number of possible desiderata.

Constraint 3. Storage-Free Continual Learning. *Avoids the use of the external memory M.*

Constraint 4. Online Continual Learning. *Limits the size of a training batch so that $|B_i| = 1$.*

Being able to learn without storing any example from the past is one on the holy grail of continual learning. In our biological counterparts, namely the brain, there are many evidence supporting this argument, while the idea of storing high-dimensional perceptual data appear 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.

2.4 Scenarios

Depending on the task-awareness or task-agnosticism of the problem to learn, now we can define, on a more abstract level and based on the specific t signal availability, three different and common scenarios for CL based on the proposed framework:

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

In the following sections we discuss each scenario more in detail.

2.4.1 Multi-Task

The Multi-Task (MT) scenario constitutes a typical setting for recent literature in CL where it is assumed to encounter a number of subsequent tasks over time, each corresponding to a different training batch with very different data distributions [Parisi et al., 2018a]. While this setting is useful for assessing continual learning strategy, it may revel itself less appropriate for modeling real-word problems where we can encounter many different batches of data over time, related to the same task or encounter the same task many times over our lifetime.

2.4.2 Single-Incremental-Task

The Single-Incremental-Task (SIT) is a very general scenario where we don't have a different task supervised signal for every training batch. It can be considered as solving a single task, which is incremental in nature or just to be in a *"task agnostic"* setting where data can be treated to similar or very different data distributions over time. However, it may be useful, also in this case, to detect and recognize very different data distributions to specialize the behavior of the agent even without the external supervised notion of task.

2.4.3 Multiple-Incremental-Task

The Multiple-Incremental-Task (MIT) scenario constitutes the more realistic scenario in which we consider natural to be able to exploit some supervision (like parents teaching in humans) or feedbacks about the tasks we are tackling over time. This allows the agent to learn task-related specialized behaviors as well the autonomous development of its generalization capabilities.

TABLE 2.1: Examples of the t signal for the three different scenarios: Multi-Task (MT), Single-Incremental-Task (SIT) and Multiple-Incremental-Task (MIT). Notice that a MIT setting requires breaking the constraint definition of SIT but also breaking the constraint definition of MT, i.e., not all the tasks are considered having the same id and not all the task are considered distinct.

Task/Session	**Task ID**		
CL setting	**MT**	**SIT**	**MIT**
t_1	1	0	0
t_2	2	0	1
t_3	3	0	0
...	...	...	...
t_n	n	0	0

2.4.4 Update Content Types

Orthogonal to the type of task supervised signal we could exploit, it is worth considering three different *Update Content Type* (UCT) which may greatly impact on the complexity of the continual learning scenario. They refer to the possible kind of data contained in each training batch B_i:

- **New Instances (NI)**: in this case the content of the batch is characterized by new instances (i.e. examples) of the same classes encountered in the previous batches.
- **New Classes (NC)**: the content of each batch B_i is characterize by the presence of examples belonging to always different classes never encountered before in previous batches $B_1, \ldots, B_{i-1}$.
- **New Instances and Classes (NIC)**: this update content type constitutes the the most realistic setting where new examples of previous encountered classes but also new classes are encountered over time.

For regression, the same organization can be maintained considering each class as a different regressor.

3

Continual Learning Strategies

> *"The transfer of knowledge within the lifetime of an individual has been found to be one of the dominating factors of natural learning and intelligence. If computers ever are to exhibit rapid learning capabilities similar to that of humans, they will most likely have to follow the same principles."*
>
> – Sebastian Thrun, 1996

The sudden interest in CL and its applications, especially in the context of deep architectures, has recently led to significant progress and original research directions, yet leaving the research community without a common terminology and clear objectives. Here we propose, in line with Kemker et al. [2018] and Zenke et al. [2017], a three-way fuzzy categorization of the most common CL strategies:

- **Architectural strategies**: specific architectures, layers, activation functions, and/or weight-freezing strategies are used to mitigate forgetting. Includes dual-memories-models attempting to imitate hippocampus-cortex duality.
- **Regularization strategies**: the loss function is extended with loss terms promoting selective consolidation of the weights which are important to retain past memories. Include basic regularization techniques such as weight sparsification, dropout, early stopping.
- **Rehearsal strategies**: past information is periodically replayed to the model to strengthen connections for memories it has already learned. A simple approach is storing part of the previous training data and interleaving them with new patterns for future training. A more challenging approach is pseudo-rehearsal with generative models.

In the Venn diagram of Figure 3.1, we show a non-comprehensive set of the most popular CL strategies. While each category is being populated with an increasing number of novel strategies, there is a large room for yet-to-be-explored techniques especially at the intersection of the three categories.

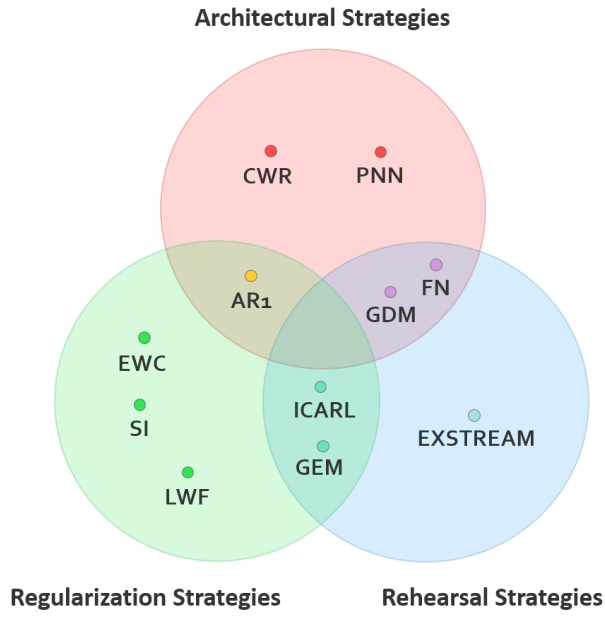


FIGURE 3.1: Venn diagram of some of the most popular CL strategies: CWR [Lomonaco and Maltoni, 2017], PNN [Rusu et al., 2016b], EWC [Kirkpatrick et al., 2017], SI [Zenke et al., 2017], LWF [Li and Hoiem, 2016], ICARL [Rebuffi et al., 2017], GEM [Lopez-paz and Ranzato, 2017], FN [Kemker and Kanan, 2018], GDM [Parisi et al., 2018b], EXSTREAM [Hayes et al., 2018a] and AR1, hereby proposed. Better viewed in color.

Progressive Neural Networks (PNN) [Rusu et al., 2016b] is one of the first architectural strategy proposed and is based on a clever combination of parameter freezing and network expansion. While PNN was shown to be effective on short series of simple tasks, the number of the model parameters keeps increasing at least linearly with the number of tasks, making it difficult to use for long sequences. The proposed *CopyWeights with Re-init* (CWR) and its evolution CWR+, constitute a simpler and lighter counterpart to PNN (at the cost of a lower flexibility), with a fixed number of shared parameters and already proven to be useful on longer sequences of tasks.

Learning Without Forgetting (LWF) [Li and Hoiem, 2016] is a regularization strategy attempting to preserve the model accuracy on old tasks by imposing output stability through knowledge distillation [Hinton et al., 2015]. Other well-known regularization strategies are *Elastic Weights Consolidation* (EWC) and *Synaptic Intelligence* (SI), both articulated around a weighted quadratic regularization loss which penalizes moving weights which are important for old tasks. In the Rehearsal category, *Gradient Episodic Memory* (GEM) [Lopez-paz and Ranzato, 2017] is an interesting approach using a fixed memory to store a subset of old patterns: it is aimed not only at controlling forgetting but also at improving accuracy on previous tasks while learning the subsequent ones (a phenomenon known as *"positive backward transfer"* see Chapter 4.3.1). *Incremental Classifier and Representation Learning* (ICARL) [Rebuffi et al., 2017] includes an external fixed memory to store a subset of old task data based on an elaborated sample selection procedure, but also employs a distillation step which makes it overlapping with the regularization category. A recent study on memory efficient implementation of pure rehearsal strategies is provided in [Hayes et al., 2018a] where a new partitioning-based method for stream clustering

named EXSTREAM is shown to be very competitive with a *Full Rehearsal* approach (storing all the past data) and with other memory management techniques.

Very recently, a growing number of techniques have been proposed on CL based on both variations of the previously introduced strategies or completely novel approaches with different degrees of success (see [Parisi et al., 2018a] for a review). In particular, *FearNet* (FN) [Kemker and Kanan, 2018] and *Growing Dual-Memory* (GDM) [Parisi et al., 2018b] are interesting approaches leveraging ideas from architectural and (pseudo) rehearsal categories: a double-memory system is exploited to learn new concepts in a short-term memory and progressively consolidate them in a long-term one.

In the following section we will better detail some of the most representative strategies for each group and at their intersection. Then the four newly proposed strategy will be detailed in depth.

3.1 Baseline Strategies

Before moving to more elaborated continual learning strategies, let us consider two basic approaches: *Naive* and *Cuumulative*, we will later use as standard baselines during the experimental evaluation counducted in Chapter 5.

3.1.1 Naive

The *Naive* strategy simply finetunes the model across the training batches without any specific mechanism to control forgetting, except early stopping and other basic regularization techniques like L1, L2 and Dropout [Goodfellow et al., 2013], which have been already found to avoid overfitting and improve generalization. The *Naive* approach has been shown to be particularly prone to catastrophic forgetting if the data distribution faced by the model are substantially different among each other. Nevertheless, in more specific settings where data distributions are often and implicitly refreshed through time, it may prove to be a reasonable strategy, being withing the constraints and desiderata detailed in Chapter 2.

3.1.2 Cumulative

The *Cumulative* strategy, also called *Full Rehearsal* [Hayes et al., 2018a], limits catastrophic forgetting by mixing *all* older examples with the new examples to be learned. When a new batch of data becomes available, there are two viable options: *i)* Finetuning h_{i-1} with all the cumulated patterns or *ii)* start from scratch (i.e. from random weights initialization). While the former is generally faster if examples in different batches share very similar features, the second has more guarantees to reach the best global performances. In the evaluation chapter we will refer to the cumulative strategy employing the second option with the idea of using it as a sort of *"upper bound"* in terms of accuracy performance for the other CL strategies.

The cumulative learning strategy is indeed very similar to the classical multi-task training setting [Caruana, 1997], which is known to yield even better performance than learning every single task in isolation. That said, it cannot really considered a formal *"upper bound"* for the accuracy metric since, depending on the specificity of the scenario, other CL strategies may prevail. For example, a scenario which constitute a natural *"curriculum"* [Bengio et al., 2009] for improving the performance of the model over time or in which more recent data are generally more relevant than the past one.

Moreover, it cannot be properly be considered as a continual learning strategy since it violates the constraint 1, which impose to not store all the examples in the external memory M.

3.2 Rehearsal Strategies

Rehearsal strategies are based on the idea of rehearsing past knowledge with a replay mechanism. Most of these strategies employ a fixed-sized external memory in which to store representative examples to reuse in conjunction with the new coming data in order to improve generalization without forgetting. More recent proposals employ generative models to generate these examples *on-the-fly*.

3.2.1 Exemplar Stream (ExStream)

Exemplar Stream (*ExStream*) was firstly introduced by [Hayes et al., 2018a] as a partitioning-based method for stream clustering and the efficient management of the external fixed-size memory for rehearsal. In addition to storing clusters, indeed, ExStream also stores counts that tally the total number of points in each cluster. Once a class-specific buffer is full and a new example $(x_t\ ,\ y_t)$ streams in, the two closest clusters in the buffer for class y_t are found using the Euclidean distance metric and merged together using:

$$w_i \leftarrow \frac{c_i \cdot w_i + c_j \cdot w_j}{c_i + c_j} \tag{3.1}$$

where w_i and w_j are the two closest clusters and c_i and c_j are their associated counts. Subsequently, the counter at c_i is updated as the sum of the counts at locations i and j and the new point is inserted into the buffer at location j. That is, $c_i \leftarrow c_i + c_j$ and $w_j \leftarrow x_t$ with $c_j = 1$.

ExStream can be also considered as an effective rehearsal strategies per-se, which, depending on the external memory size and the task at hand, can be considered competitive with the *Cumulative* approach [Hayes et al., 2018a].

3.3 Architectural Strategies

Architectural Strategies are based on the central idea of modifying the model architecture and parameters value in order to preserve old information and make space to the incoming one.

Modifying connections, activation functions, freezing parameters to mitigate forgetting are very common possibilities. This group also includes dual-memories-models attempting to imitate the hippocampus-cortex duality.

3.3.1 Progressive Neural Networks (PNNs)

Progressive Neural Networks (*PNNs*) were originally proposed by Rusu et al. [2016b] for explicitly tackling catastrophic forgetting and are one of the best examples of the architectural category. The idea is to keep a pool of pre-trained models (or *"columns"*) as knowledge base, and use lateral connections between them for fast adaptation to the new batch/task. It was originally proposed to tackle reinforcement learning in multi-task settings but the model architecture is general enough to be adapted also to other scenarios. For each new task encountered a new neural network (or a new column) is created, and its lateral connections with all previous ones are learned. The mathematical formulation is presented below. Notation is here maintained to follow the one proposed in the original paper.

In PNNs, each batch/task B_n is associated with a neural network, which is assumed to have L layers with hidden activations $h_i^{(n)}$ for the units at layer $i \leq L$. The set of parameters in the neural network for B_n is denoted by $\Theta^{(n)}$. When a new batch B_{n+1} arrives, the parameters $\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(n)}$ are frozen while each layer $h_i^{(N+1)}$, in the network related to task B_{N+1}, takes inputs from $(i-1)$th layers of all the networks related to the previously encounterd batches, i.e.,

$$h_i^{N+1} = max\left(0, W_i^{(N+1)} \cdot h_{i-1}^{(N+1)} + \sum_{n<N+1} U_i^{(n:N+1)} \cdot h_{i-1}^{(n)}\right) \tag{3.2}$$

where W_i^{N+1} denotes the weights matrix of layer i in neural network $N+1$. The lateral connections are learned via $U_i^{(n:N+1)}$ to indicate how strong the $(i-1)$th layer from task n influences the ith layer from task $N+1$. h_0 is the network input.

Unlike pre-training and fine-tuning, progressive neural networks is agnostic with respect to the type of batches/tasks encountered, which makes it more practical for real-world applications. The lateral connections can be learned for related, orthogonal, or even adversarial tasks. Non-linear lateral connections are learned through a single hidden perceptron layer, which reduces the number of parameters from the lateral connections to the same order as $|\Theta^{(1)}|$. However, this flexibility and the very nteresting *"zero-forgetting"* property, progressive neural networks come at a price: the total number of parameters tends to explode with an increasing number of batches/tasks, since it needs to learn a new neural network for every new batch and its lateral connections with all the existing ones. Rusu et al. [2016b] suggested pruning [Hassibi and Stork, 1993] or online compression [Rusu et al., 2016a] as potential solutions. More details can be found in the original paper [Rusu et al., 2016b].

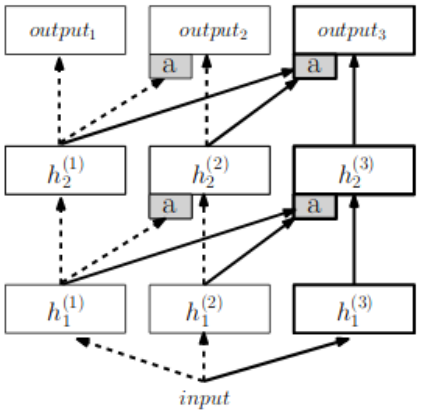


FIGURE 3.2: Depiction of a three column progressive network. The first two columns on the left (dashed arrows) were trained on task 1 and 2 respectively. The grey box labelled *a* represent the *adapter layers* (more details cab be found in [Rusu et al., 2016a]). A third column is added for the final task having access to all previously learned features.

3.4 Regularization Strategies

Regularization strategies are based on the central idea of *regularizing* the learning process on the new data for preserving past learned knowledge and skills. This is generally accomplished with an additional regularization loss for preserving the state of the weights that are important for the previously encountered data distributions.

3.4.1 Learning without Forgetting (LWF)

Learning Without Forgetting (*LWF*) [Li and Hoiem, 2016] is a regularization approach which tries to control forgetting by imposing output (i.e. prediction) stability via distillation. It has been originally conceived for a Multi-Task (MT) setting but it can be also easily adapted to other scenario.

Let us consider an output level with s classes (i.e. s neurons) and assume that some classes were already learned in previous batches. The current batch B_i includes n_i examples drawn from s_i (still unseen) classes, then LWF:

- At the beginning of batch B_i, before the training start, computes the prediction of the network for each new pattern in B_i. To this purpose it performs a forward pass and stores the s-dimensional network prediction $\hat{y}_{lwf}$ for each of the examples in B_i.
- Starts training the network with Stochastic Gradient Descent (SGD) by using a two component loss:

$$(1 - \lambda) \cdot L_{cross}(\hat{y}, t = \hat{y}_{1h}) + \lambda \cdot L_{kdl}(\hat{y}, t = \hat{y}_{lwf}) \tag{3.3}$$

  where:

  - $\hat{y}$ are the network predictions (evolving while the model is trained).

- The first part is the usual cross-entropy loss whose target vectors t take the form of one-hot vectors $\hat{y}_{1h}$ corresponding to the true pattern labels. This component adjusts the model weights to learn the new classes in B_i.
- The second part is a Knowledge Distillation Loss [Hinton et al., 2015] which tries to keep the network predictions close to $\hat{y}_{lwf}$ (here used as soft target vectors). This component tries to preserve (for the old classes which are not in the current batch) a stable response. The second term can be replaced with one term for each old task/batch; the two formulations are equivalent but the compact form here proposed is simpler to deal with in practice.
- The parameter $\lambda \in [0, 1]$ defines the relative weights of the two loss components, thus controlling the trade-off between stability and plasticity.

In the MT scenario, L_{cross} is computed only for the n_i new classes in B_i, while L_{kdl} is computed for all the $(\sum_{j<i} n_j)$ classes previously learned. In SIT there is no such distinction and both the loss components are computed for all the s classes encountered so far. When moving to a SIT setting we need to:

- replace L_{kdl} with L_{cross} in the second terms. LWF authors argued in Li and Hoiem [2016] that the Knowledge Distillation Loss can be replaced with Cross-Entropy with no significant accuracy change. In our initial experiments we obtained similar results, so for simplicity we adopted cross-entropy.

$$L_1 = (1 - \lambda) \cdot L_{cross}(\hat{y}, t = \hat{y}_{1h}) + \lambda \cdot L_{cross}(\hat{h}, t = \hat{y}_{lwf}) \tag{3.4}$$

- fuse the two loss components into a single loss with a weighted soft target vector:

$$L_2 = L_{cross}(\hat{y}, t = (1 - \lambda) \cdot \hat{y}_{1h} + \lambda \cdot \hat{y}_{lwf}) \tag{3.5}$$

  It can be simply proved that L_1 and L_2 are equivalent and lead to the same gradient flow. In fact, for cross-entropy the gradient of the loss function with respect to the logit layer o (i.e., the layer before softmax) is $\partial L_{cross}/\partial o = (\hat{y} - t)$ and therefore:

$$\begin{aligned} \frac{\partial L_1}{\partial o} = (1 - \lambda) \cdot (\hat{y} - \hat{y}_{1h}) + \lambda \cdot (\hat{y} - \hat{y}_{lwf}) = \\ (\hat{y} - \hat{y}_{1h}) + \lambda \cdot (\hat{y}_{1h} - \hat{y}_{lwf}) = \\ \hat{y} - ((1 - \lambda) \cdot \hat{y}_{1h} + \lambda \cdot \hat{y}_{lwf}) = \frac{\partial L_2}{\partial o}. \end{aligned} \tag{3.6}$$

Using a single value of λ across the sequential training batches can be suboptimal, since the importance of the past should increase with the number of classes learned. A reasonable solution is increasing λ according to the proportion of the number of examples in the current batch w.r.t. the number of examples encountered so far. A batch specific value λ_i can be obtained as:

$$\lambda_i = \begin{cases} 0, & i = 1 \\ map(1 - \frac{n_i}{\sum_{j \leq i} n_j}), & i > 1 \end{cases} \tag{3.7}$$

where *map* is a linear mapping function that can shift and stretch/compress its input. For example, considering the number of classes in the *CORe50* benchmark detailed in Section 5.1.4 (i.e. 10 in the first batch and 5 in the successive batches) and assuming that *map* is the identity function, we obtain: $\lambda_1 = 0, \lambda_2 = \frac{2}{3}, \lambda_1 = \frac{3}{4}, \ldots, \lambda_9 = \frac{9}{10}$.

Another important facet is the learning strength to adopt in the initial batch B_1 and successive batches B_i. It is worth noting that in LWF (as for EWC and SI) training on incremental batches $B_i, i > 1$ should not be forced to convergence. In fact, as the regularization part of the loss becomes dominant the training accuracy tend to decrease and trying to leverage it with aggressive learning rates and high number of epochs can lead to divergence. In our experiments (detailed in Chapter 5), we trained the model on each batch for a fixed small number of epochs without forcing convergence. Using a simple early stopping criteria is crucial for continual learning because of efficiency and lack of realistic validation sets.

Summarizing, LWF implementation with weighted soft target vectors is very simple and, for each batch $B_i, i > 1$, its overhead consists of:

- computation: one extra forward pass for each of the n_i pattern.
- storage: temporary storing (for the batch lifespan) the $\hat{y}_{lwf}$ predictions, consisting of $n_i \cdot s$ values.

3.4.2 Elastic Weights Consolidation (EWC)

Elastic Weights Consolidation (*EWC*) [Kirkpatrick et al., 2017] is a regularization approach which tries to control forgetting by selectively constraining (i.e., freezing to some extent) the model weights which are important for the previous tasks.

Intuitively, once a model has been trained on a task, thus reaching a minimum in the loss surface, the sensitivity of the model w.r.t. each of its weight θ_k can be estimated by looking at the curvature of the loss surface along the direction determined by θ_k changes. In fact, high curvature means that a slight θ_k change results in a sharp increase of the loss. The diagonal of the Fisher information matrix F, which can be computed from first-order derivatives alone, is equivalent to the second derivative (i.e. curvature) of the loss near a minimum. Therefore, the k^{th} diagonal element in F (hereafter denoted as F_k) denotes the importance of weight θ_k. Important weights must be moved as little as possible when the model is fine-tuned on new tasks. In a two tasks scenario this can be achieved by adding a regularization term to the loss function when training on the second task:

$$L = L_{cross}(\hat{y}, t = \hat{y}_{1h}) + \frac{\lambda}{2} \cdot \sum_k F_k(\theta_k - \theta_k^*)^2 \tag{3.8}$$

where:

- θ_k^* are the optimal weight values resulting from the first task.
- λ is the regularization strength.

Let us now consider a sequence of tasks or batches B_i. After training the model on batch B_i we need to compute the Fisher information matrix F^i and store the set of optimal weights Θ^i. F_i and Θ^i will be then used to regularize the training on B_{i+1}. Each diagonal element F_k^i can be computed as the variance of $\partial L_{cross}(\hat{y}, t)/\partial\theta_k$ over the n_i patterns of B_i.

Two different EWC implementations can be setup in practice:

1. A distinct regulation term is added to the loss function for each old task. This require maintaining a Fisher matrix F^i and a set of optimal weights Θ^i for each of the previous task/batch;

2. A single Fisher matrix F is initialized to 0 and consolidated at the end of a batch B_i by (element wise) summing the Fisher information: $F = F + F^i$. A single set of optimal weights Θ is also maintained by using the most recent ones ($\Theta = \Theta^i$) since Θ^i already incorporates constraints from all previous batches (refer to the discussion in [Huszár, 2018; Kirkpatrick et al., 2018]).

Option 1 can be advantageous to precisely control EWC training dynamic in the MT scenario with few tasks, but is not practical (because of storage and computation issues) in SIT scenario with several batches. It is worth noting that in option 2. the F_k values can only increase as new batches are processed, potentially leading to divergence for large λ. To better understand this issue, let us consider how the regularization term is dealt with by gradient descent: this is quite similar to L2 regularization and can be implemented as a special weight decay where weights θ_k are not decayed toward 0, but toward θ_k^*. The weight update determined by the loss function (eq. 3.8) is:

$$\theta_k' = \theta_k - \eta \cdot \frac{\partial L_{cross}(\hat{y}, t)}{\partial \theta_k} - \eta \cdot F_k(\theta_k - \theta_k^*) \tag{3.9}$$

where η is the learning rate. In the above equation if, for some k, the product $\eta \cdot \lambda \cdot F_k$ is greater than 1, the weight correction toward θ_k^* is excessive and we overshoot the desired value. Tuning λ according to the maximum theoretical value of F_k is problematic because: `i)` we do not know such value; `ii)` using a too high value might lead to unsatisfactory performance since does not allow to constrain the weights associated to mid-range F_k enough. We empirically found that a feasible solution is normalizing F after each batch B_i as:

$$\begin{aligned} F &= F + F^i \\ \hat{F} &= clip(\frac{F}{i}, max_F) \end{aligned} \tag{3.10}$$

where *clip* set to the constant max_F the matrix values exceeding max_F. Note that F/i replaces the Fisher matrix sum with an average, and this could be counterintuitive. Let us suppose that weight θ_5 is very important for batch B_1 and this is reflected by an high value of F_5^1, then if θ_5 is not important for B_2 as well (i.e., F_5^2 is small) computing the average $1/2 \cdot (F_5^1 + F_5^2)$ pulls down the combined importance. However, this can be compensated by a proper selection of a max_F in order to saturate $\hat{F}$ values even for those weights which are important for a single task. Given max_F and η we can easily determine the maximum value for λ as $1/(\eta \cdot max_F)$.

An example is shown in Figure 3.3, where the distribution of Fisher information values is reported after B_1, B_2 and B_3. In the first row F values denotes a long tail on the right. In the second row, F_k values are averaged and clipped to 0.001 thus allowing to work with higher λ and better control forgetting.

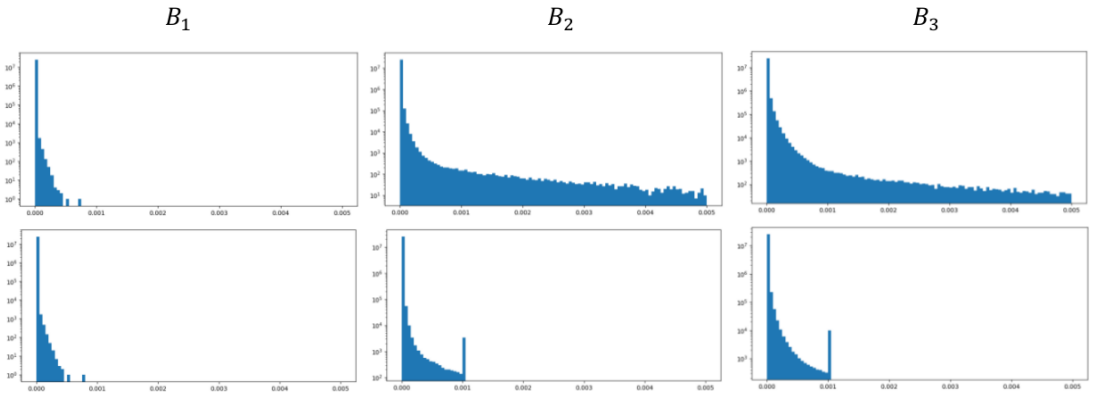


FIGURE 3.3: CaffeNet trained by EWC on CORe50 SIT (details on the experiments can be found in Section 5.1.4). The first row shows F values distribution denoting a long tail on the right: considering the logarithmic scale the number of weights associated to high F values taking high values is quite limited. The second row shows the normalized matrix $\hat{F}$ obtained with averaging F values and max clipping to 0.001. Saturation to 0.001 is well evident, but after B_3 the fraction of saturated weights is small (about 1/1000).

Summarizing, EWC implementation is moderately simple and, for each batch B_i, its overhead consists of:

- computation of Fisher information F^i, requiring one forward and one backward propagation for each of the n_i patterns.
- storage of F and Θ, totaling $2 \cdot m$ values, where m is the number of model weights (including biases).

3.4.3 Synaptic Intelligence (SI)

Synaptic Intelligence (*SI*) was introduced in [Zenke et al., 2017] as a variant of EWC. The authors argued that computation of Fisher information is expensive for continual learning and proposed to calculate weight importance on-line during SGD.

The loss change given by a single weight update step during SGD is given by:

$$\Delta L_k = \Delta\theta_k \cdot \frac{\partial L}{\partial \theta_k} \tag{3.11}$$

where $\Delta\theta_k = \theta'_k - \theta_k$ is the weight update amount and $\partial L / \partial \theta_k$ the gradient. The total loss change associated to a single parameter θ_k can be obtained as running sum $\sum \Delta L_k$ over the weight trajectory (i.e., the sequence of weight update steps during the training on a batch). The weight importance (here denoted as F_k to keep notation uniform with previous section) is then

computed as:

$$F_k = \frac{\sum \Delta L_k}{T_k^2 + \xi} \tag{3.12}$$

where T_k is the total movement of weight θ_k during the training on a batch (i.e., the difference between its final and the initial value) and ξ is a small constant to avoid division by 0 (see [Zenke et al., 2017] for more details). Note that the whole data needed to calculate F_k is available during SGD and no extra computation is needed.

In the SIT scenario we empirically found that an effective normalization after each batch B_i is:

$$\begin{aligned} F &= F + w_i \cdot F^i \\ \hat{F} &= clip(F, max_F) \end{aligned} \tag{3.13}$$

where F is set to 0 before first batch and then consolidated as a weighted sum with batch specific weights w_i. Actually in our experiments, as reported in Section 4.4, we used a small value w_1 for the first batch and a constant higher value for all successive batches: $w_2 = w_3 = \cdots = w_9$.

Considering CORe50 experiments, since in the first batch we tune a model from ImageNet weight initialization, the trajectories that most of the weights have to cover to adapt to CORe50 are longer than for successive batches whose tuning is intra dataset. This is not the case for EWC, because EWC looks at the loss surface at convergence, independently of the length of weight trajectories.

Given $\hat{F}$ values, SI regularization can be implemented as EWC. The magnitude of $\hat{F}$ values can also be made comparable to EWC by proper setting of w_i, so max_F and λ can take the similar values. Figure 3.4 compares the distributions of $\hat{F}_k$ values between EWC and SI: at first glance the distributions appear to be similar; of course more precise correlation studies could be performed, but this is out of the scope of this work.

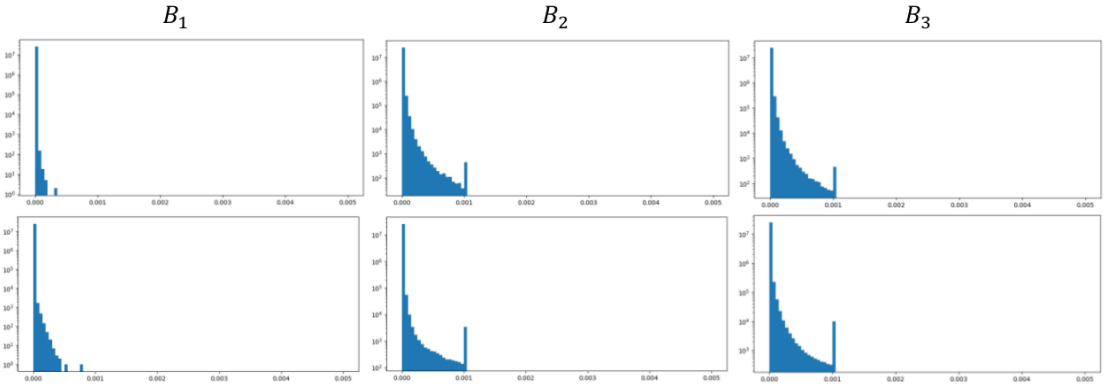


FIGURE 3.4: CaffeNet trained on CORe50, SIT setting (more details on the experiments can be found in Section 5.1.4). The first row shows $\hat{F}$ values distribution obtained by SI on batches B_1, B_2 and B_3. $\hat{F}$ values distribution from EWC is reported in the second row for comparison. The shape of the distribution is quite similar, even if in this experiments, the number of SI saturated values is about 10 times lower.

Summarizing, SI implementation is quite simple and, for each batch B_i, its overhead consists of:

- computation of weight importance F^i, based on information already available during SGD.

- storage of F and Θ, totaling $2 \cdot m$ values, where m is the number of model weights.

3.5 Hybrid Strategies

In the previous paragraph we have presented some of the most representative strategies for each of the three central approaches for learning continually. However, it is worth pointing out that, most of the times, this is just an ideal separation between strategies. This is why a fuzzy categorization is more appropriate and many strategies may fall in the Hybrid section, in the middle of two or even three approaches. In the following section, we present one of the most representative methods at the intersection of two categories of algorithmic strategies for CL.

3.5.1 Incremental Classifier and Representation Learning (ICARL)

Rebuffi et al. [2017] proposed a new model for *class-incremental learning*. Class-incremental learning (i.e. SIT with NC as *updated content type* in our terminology) requires the classification system to incrementally learn and classify new classes that it has never seen before. It assumes that examples of different classes can occur at different times, with which the system should maintain a satisfactory classification performance on each observed class. Rebuffi et al. [2017] also emphasized that computational resources should be bounded or slowly increased as more and more classed are encountered over time.

To meet these criteria, a new model called *Incremental Classifier and Representation Learning* (*ICARL*) was designed to simultaneously learn classifiers and feature representations in the class-incremental setting. Intuitively, *ICARL* maintains a set of exemplar patterns for each observed class aiming to carry the most representative information of the class and rehearse the model via distillation. The classification of a new example is performed by *nearest-mean-of-exemplars*, i.e. by choosing the class with the nearest average of prototypes in the embedded. When a new class shows up, *ICARL* creates an exemplar set for this new class while trimming the exemplar sets of the existing/previous classes, hence maintaining the external memory size within a specified threshold.

More formally, at any time, *ICARL* learns a stream of classes in the class-incremental learning setting with their training example sets, $X^s, X^{s+1}, \ldots, X^t$, where X^y is a set of examples of class y. y can either be an observed/past class or a new class. To avoid memory overflow, *ICARL* holds a fixed number (K) of exemplars in total. With C classes, the exemplar sets are represented by $P = \{P_1; \ldots, P_C\}$ where each class's exemplar set P_i maintains K/C exemplars. In [Rebuffi et al., 2017], both original examples and exemplars are images, but the proposed method is general enough for non-image datasets. A more in depth analysis and evaluation of this strategy can be found in the original paper.

3.6 Proposed Strategies

Most of the aforementioned strategies were designed and have been shown working with different degree of success in the MT scenario. In recent literature very little attention has been devoted to the SIT scenario we regard as essential for many real-world application and, arguably, more difficult. In this section, we will propose four different strategies for tackling this complex scenario especially with NI, NC and NIC update content types. All these strategies have been designed with a lower computational overhead (independent by the number of batches encountered) and to tackle or even exploit the *non i.i.d.* nature of high-dimensional steaming data we naturally encounter over time.

3.6.1 Semi-Supervised Tuning (SST)

Semi-Supervised Tuning (SST) is very light and simple continual learning strategy specifically designed to work with a temporal coherent stream of data, reducing the amount of supervision needed to learn continually from it. Semi-supervised learning [Chapelle et al., 2006; Zhu, 2006]) exploits both labeled and unlabeled data to build robust models. In particular, in self-training [Rosenberg et al., 2007], a classifier is first trained with a small amount of labeled data and then used to classify the unlabeled data. Typically the most confident unlabeled points, together with their predicted labels, are added to the training set. The classifier is re-trained and the procedure repeated. Our approach can be framed in the semi-supervised learning family since we use labeled data for initial training and unlabeled data (form the same classes) for subsequent tuning. However, our approach is continual and the labeled/unlabeled data are used at different stages to mimic human learning. Therefore, particular care must be taken to control the catastrophic forgetting.

In specific application domains semi-supervised learning approaches have been proposed to self-update initial models (or templates): see for example [Rattani et al., 2009] for biometric recognition and [Matthews et al., 2004] for tracking. Several researchers pointed out, that although the use of unlabeled data can substantially increase the system accuracy and robustness, the risk of drifts is always present. For example, in the context of face recognition, Marcialis et al. [2008] reported that even with operations of update procedures at high confidence, the introduction of impostors cannot be avoided. Analogously to many domain specific solutions our approach is continual and can exploit classification confidence. Temporal coherence has already been exploited for face recognition from video [Franco et al., 2010], but the proposed update solution is domain specific and not easily generalizable as the one here introduced.

The most related research to this strategy are the works by Mobahi et al. [2009] and Weston et al. [2012] where temporal coherence has been embedded in the semi-supervised training of deep architectures. However, in those works unlabeled data are used together with labeled one to regularize the supervised training while, in *Semi-Supervised Tuning*, we first train a system with labeled data and later we tune it with unlabeled data. The biological plausibility of the continual learning approach here proposed is discussed in [Li and DiCarlo, 2008] whose authors introduce the term UTL (*Unsupervised Temporal slowness Learning*) to describe the hypothesis

under which invariance is learned from temporal contiguity of object features during natural visual experience without external supervision.

Let S_w be a temporally coherent sequence of video frames $v^{(t)}, t = 1 \dots len(S_w)$ taken from the same object (of class w): while the total object variation (in term of pose, lighting, distortion, etc.) in the whole sequence can be very high, only a limited amount of variation is expected to characterize pairs of successive frames $v^{(t)}$ and $v^{(t-1)}, t = 2 \dots len(S_w)$. Let N be a classifier able to map an input pattern $v^{(t)}$ (i.e., a single video frame) into an output vector $N(v^{(t)})$ denoting the posterior class probabilities $P(w|v^{(t)}), w = 1 \dots n_w$. While in this work N will be instantiated with a deep architecture trained with gradient descent, in general N can be any trainable classifier returning class probabilities and whose optimization procedure minimize a cost (or loss) including the desired output $d(v^{(t)})$ for the input $v^{(t)}$. If the squared error is taken as loss function, for each pattern $v^{(t)}$ (of class w) the optimization procedure attempts to minimize:

$$\frac{1}{2} \left\| N(v^{(t)}) - d(v^{(t)}) \right\|^2 \tag{3.14}$$

Assuming that N has already been trained (with supervision) by using a first batch of data, each subsequence training can be considered as a tuning (i.e., learning continually). Given a sequence S_w, we define four ways to instantiate the desired vector $d(v^{(t)})$ during the system tuning:

- *Supervised Tuning* (***SupT***): this is the classical supervised approach where the desired output vector has the Δ form (all terms are zero except that corresponding to the pattern class w)

$$d(v^{(t)}) = \Delta_w = [0, \dots, \underset{w}{\underset{\longrightarrow}{1}}, \dots, 0] \tag{3.15}$$

- *Supervised Tuning with Regularization* (***SupTR***):

$$d(v^{(t)}) = \lambda \cdot \Delta_w + (1 - \lambda) \cdot N(v^{(t-1)}) \tag{3.16}$$

  where $\lambda \in [0, 1]$ controls the influence of the temporal coherence regularizing term. This is close to the approach proposed by Mobahi et al. [2009], but we embed the regularizing term into the desired output and then perform a single optimization step, while Mobahi et al. [2009] make disjoint optimization steps.

- *Semi-Supervised Tuning - Basic* (***SST-B***):

$$d(v^{(t)}) = N(v^{(t-1)}) \tag{3.17}$$

  This simply takes as desired output at time t the output vector at time $t-1$. The class label w is not used, but since we assume that the input pattern belongs to one of the know-classes, the update is semi-supervised.

- *Semi-Supervised Tuning - Advanced* (***SST-A***):

$$f(v^{(t)}) = \begin{cases} N(v^{(t-1)}) & t = 2 \\ \dfrac{f(v^{(t-1)}) + N(v^{(t-1)})}{2} & t > 2 \end{cases} \tag{3.18}$$

$$d(v^{(t)}) = \begin{cases} N(f^{(t)}) & \text{if } \max_i f_i(v^{(t)}) > sc \\ N(v^{(t)}) & \text{otherwise} \end{cases} \tag{3.19}$$

  At each step, we fuse the posterior probabilities $N(v^{t-1})$ with the posterior probabilities $f(v^{(t-1)})$ accumulated before; this is a sort of sum rule fusion where the weight of far (in time) patterns progressively vanishes. Then, if at least one of the fused class posteriors (in $f(v^{(t)})$) is higher than a given threshold sc, denoting high self-confidence, the desired output is set to $f(v^{(t)})$ to enforce temporal coherence. Otherwise (high uncertainty cases) no semi-supervised update have to be done, and formally, this can be achieved by passing back $N(v^{(t)})$ to equation (3.14). Here too, the class label w is not used.

With a minimal computational overhead, as we will see through the empirical evaluation conducted in Chapter 5, SST is able to increase the global performances of the model, over time, and without exploiting additional supervised signals, but just exposed to a temporal coherent video frames of the same objects encountered before (NI *update content type*).

3.6.2 Copy-weights with Re-init (CWR)

Copy-weights with Re-init (*CWR*) is a simple yet effective architectural techniques for continually learning from sequential batches. While it can work both for NC (new classes) and NIC (new instances and classes) update content type, here we focus on NC under SIT scenario.

Referring to Figure 4.1 (bottom) the most obvious approach to implement a strategy working in SIT seems to be:

1. Freeze shared weights $\bar{\Theta}$ after the first batch.
2. For each batch B_i, extend the output layers with new neurons/weights for the new classes, randomly initialize the new weights but retain the optimal values for the old class weights. The old weights could then be frozen (denoted as FW) or continued to be tuned (denoted as CW).

Implementing step 2 as above proved to be suboptimal with respect to CWR approach (see experiments in Section 5.1.4 for a comparison) where old class weights are re-initialized at each batch.

To learn class-specific weights without interference among batches, CWR maintains two sets of weights for the output classification layer: cw are the consolidated weights used for inference and tw the temporary weights used for training: cw are initialized to 0 before the first batch, while

Algorithm 1 CWR

$cw = 0$
init $\bar{\Theta}$ random or from a pre-trained model (e.g. ImageNet)
for each training batch B_i:
    expand output layer with s_i neurons for the new classes in
    random re-init tw (for all neurons in the output layer)
    Train the model with SGD on the s_i classes of B_i:
        if $B_i = B_1$ learn both $\bar{\Theta}$ and tw
        else learn tw while keeping $\bar{\Theta}$ fixed
    for each class j among the s_i classes in B_i:
        $cw[j] = w_i \cdot tw[j]$
    Test the model by using $\bar{\Theta}$ and cw

tw are randomly re-initialized (e.g., Gaussian initialization with std = 0.01, mean = 0) before each training batch. At the end of each batch training, the weights in tw corresponding to the classes in the current batch are scaled and copied in cw: this is trivial in NC case because of the class segregation in different batches but is possible also for more complex cases (see Section 5.1.4). To avoid forgetting in the lower levels, after the first batch B_1, all the lower level weights $\bar{\Theta}$ are frozen. Weight scaling (with batch specific weights w_i) is necessary in case of unbalanced batches with respect to the number of classes or number of example per class.

More formally, let $cw[j]$ and $tw[j]$ be the subset[1] of weights related to class j, then CWR learning sequence can be implemented as described in Algorithm 1.

Finally, CWR implementation is very simple and, the extra computation is negligible and for each batch B_i, its overhead consists of:

- storage of temporary weights tw, totaling $s \cdot pn$ values, where s is the class number and pn the number of penultimate layer neurons.

3.6.3 Copy-weights with Re-init Plus (CWR+)

Here we propose two simple modifications of CWR: the resulting approach is denoted as CWR+. The first modification, *mean-shift* is an automatic compensation of batch weights w_i. In fact, tuning such parameters is annoying and a wrong parametrization can lead the model to underperform. We empirically found that, if the weights tw learnt during batch B_i, are normalized by subtracting their global average, then rescaling by w_i is no longer necessary (i.e., all $w_i = 1$). Other reasonable forms or normalization, such as setting standard deviation to 1, led to worse results in our experiments.

The second modification, denoted as *zero init*, consists in setting initial weights tw to 0 instead of typical Gaussian or Xavier random initialization. It is well known that neural network weights cannot be initialized to 0, because this would cause intermediate neuron activations to be 0, thus nullifying back-propagation effects. While this is certainly true for intermediate level weights, it is not the case for the output level (see Appendix D.7 for a simple derivation). Actually, what

[1] the number of weights in each subset typically corresponds to the number of neurons in the penultimate layer.

is important here is not using the value 0, but the same value for all the weights: 0 is used for simplicity.

Even if this could appear a minor detail, we discovered that it has a significant impact on the training dynamic and the forgetting. If output level weights are initialized with Gaussian or Xavier random initialization they typically take small values around zero, but even with small values in the first training iterations the softmax normalization could produce strong predictions for wrong classes. This would trigger unnecessary errors back-propagation changing weights more than necessary. While this initial adjustment is uninfluential for normal batch training we empirically found that is detrimental for continual learning and that even a simple approach such as *Naive* can greatly benefit from zero init.

In Algorithm 2 we report the pseudocode for CWR+: the modifications w.r.t. CWR are highlighted in bold.

Algorithm 2 CWR+

```
cw = 0
init Θ̄ random or from a pre-trained model (e.g. trained on ImageNet)
for each training batch B_i:
    expand output layer with s_i neurons for the new classes in B_i
    tw = 0 (for all neurons in the output layer)
    Train the model with SGD on the s_i classes of B_i:
        if B_i = B_1 learn both Θ̄ and tw
        else learn tw while keeping Θ̄ fixed
    for each class j among the s_i classes in B_i:
        cw[j] = tw[j] − avg(tw)
    Test the model by using Θ̄ and cw
```

CWR+ overhead is basically the same of CWR since taking the average of is computationally negligible w.r.t. the SGD complexity.

3.6.4 Architect and Regularize (AR1)

A drawback of CWR and CWR+ is that weights $\bar{\Theta}$ are tuned during the first batch and then frozen. *Architect* and *Regularize* (AR1), is the combination of an architectural and regularization approach. In particular, we extend CWR+ by allowing $\bar{\Theta}$ to be tuned across batches subject to a regularization constraint (as per LWF, ECW or SI). We did several combination experiments on *CORe50* to select a regularization approach; each approach required a new hyperparameter tuning w.r.t. the case when it was used in isolation. At the end, our choice for AR1 was in favor of SI because of the following reasons:

- LWF performs nicely in isolation, but, as we will see in our experiments, it does not bring relevant contributions to CWR+. We guess that being the LWF regularization driven by an output stability criterion, most of the regularization effects go to the output level that CWR+ manages apart.

- Both EWC and SI provide positive contributions to CWR+ and their difference is minor. While SI can be sometime unstable when operating in isolation we found it much more stable and easy to tune when combined with CWR+.
- SI overhead is small, since the computation of trajectories can be easily implemented from data already computed by SGD.

In Algorithm 3 we report the pseudocode for AR1.

Algorithm 3 AR1

$cw = 0$
init $\bar{\Theta}$ random or from a pre-trained model (e.g. trained on ImageNet)
$\Theta = 0$ (Θ are the optimal shared weights resulting from the last training, see Section 3.4.3)
$\hat{F} = 0$ ($\hat{F}$ is the weight importance matrix, see Section 3.4.3).
for each training batch B_i:
    expand output layer with s_i neurons for the new classes in B_i
    $tw = 0$ (for all neurons in the output layer)
    Train the model with SGD on the s_i classes of B_i by simultaneously:
        learn tw with no regularization
        learn $\bar{\Theta}$ subject to SI regularization according to $\hat{F}$ and Θ
    for each class j among the s_i classes in B_i:
        $cw[j] = tw[j] - avg(tw)$
    $\Theta = \bar{\Theta}$
    Update $\hat{F}$ according to trajectories computed on B_i (see eq. 3.12 and 3.13)
    Test the model by using $\bar{\theta}$ and cw

AR1 overhead is the sum of CWR+ and SI overhead:

- storage:
  - Temporary weights tw, totaling $s \cdot pn$ values, where s is the class number and pn the number of penultimate layer neurons.
  - F and Θ, totaling $2 \cdot (m - s \cdot pn)$, where m is the total number of model weights.
- computation:
  - Weights importance $\hat{F}$, based on information already available during SGD.
  - Learning sw subject to SI regularization can be easily implemented as weight decay (see eq. 3.9) and is computationally light.

Considering the low computational overhead and the fact that typically SGD is typically early stopped after 2 epochs, AR1 is suitable for online implementations.

4

Continual Learning Benchmarks and Protocols

> *"Without the capability of retaining and accumulating knowledge learned in the past, making inferences about it, and using the knowledge to help future learning and problem solving, achieving artificial general intelligence (AGI) is unlikely."*
>
> – Zhiyuan Chen and Bing Liu, *Lifelong Machine Learning*, 2018

In this chapter we will look at the most common datasets and benchmarks available for assessing continual learning strategies and propose a number of original ones. We also summarize the common evaluation protocols and metrics currently adopted in CL research as well as proposing a rich set of novel metrics we regard as extremely important for the future of this line of research. Finally, we well discuss a number of practices which may be useful for a deeper understanding of the learning dynamics of a prediction model trained over time.

4.1 Benchmarks

Benchmarking CL strategies today is still highly non-standard and, even if we focus on supervised classification (e.g. leaving reinforcement learning out), researches often reports their results on different datasets by following different training and evaluation protocols. In Table 4.1, the most commonly used benchmarks for continual learning are reported.

The *Permuted MNIST* is one of the first benchmarks used for continual learning [Goodfellow et al., 2013; Srivastava et al., 2013]. Every batch/task is based on a different permutation of the pixels of each image. Despite its simplicity, the benchmark constitute an optimal choice for fast prototyping new algorithms in reasonable time and it is particularly appealing for generating an unlimited number of tasks of equilibrated complexity. The *Rotated MNIST* [Lopez-paz and Ranzato, 2017] follows the same line of reasoning, however, for this case the transformation operated is a rotation of each image instead of a random permutation of its pixels. Another

variation of the classic *MNIST* dataset is the *MNIST Split* benchmark [Zenke et al., 2017]. In this case a MT-NC scenario is addressed splitting the original dataset in 5 different batches with two digits each. Zenke et al. [2017], in the same paper introduced the *CIFAR-10/100 Split*. In this case, the two *CIFAR* datasets are sequentialized giving birth to a 6 batches continual learning scenario of an increased complexity w.r.t. the *MNIST*-based benchmarks. Rebuffi et al. [2017] were the first to address problems of even greater complexity introducing the benchmarks *iCIFAR-100* and *ILSVRC2012 Split* with 10 different batches each containing 10 and 100 classes respectively. Finally, the *Atari games* suit [Kirkpatrick et al., 2017], remains one of the few benchmark used in deep *reinforcement learning*.

TABLE 4.1: Categorizations of CL experiments from the recent literature. Most of the benchmarks are based on reshaped versions of well-known vision datasets such as *MNIST*, *CIFAR-10*, *CIFAR-100*, *ILSVR2012*, *CUB-200* and the *Atari Games* suit for reinforcement learning.

Dataset	**#Batches**	**UCT**
Permuted MNIST [Kirkpatrick et al., 2017]	10	NI
Rotated MNIST [Lopez-paz and Ranzato, 2017]	20	NI
MNIST Split [Zenke et al., 2017]	5	NC
CIFAR-10/100 Split [Zenke et al., 2017]	6	NC
iCIFAR-100 [Rebuffi et al., 2017]	10	NC
ILSVRC2012 Split [Rebuffi et al., 2017]	10	NC
Atari Games [Kirkpatrick et al., 2017]	25	NI

As, already mentioned in Chapter 1, nowadays, much of the CL studies consider a multi-task scenario, where the same model is required to learn incrementally a number of isolated tasks without forgetting how to solve the previous ones. The aforementioned *MINIST Split* [Zenke et al., 2017] is composed of 5 *isolated* tasks, where each of them consists in learning two classes (i.e. two digits). There is no class overlapping among different tasks, and accuracy is computed separately for each task. Such a model cannot be used to classify an unknown digit among the 10 classes, unless an oracle is available at inference time to associate the unknown pattern to the right sub-classification problem in order to setup the last classification layer(s) accordingly (hence providing the t signal). In other words, these experiments are well suited for studying the feasibility of training a single model on a sequence of disjoint tasks without forgetting how to solve the previous ones, but are not appropriate for addressing tasks which are incremental in their nature.

A still largely unexplored scenario, denoted in the previous chapter as Single-Incremental-Task (SIT), is addressed in [Rebuffi et al., 2017]. This particular setting assessed in the paper, referred as *class-incremental*, considers a single task which is incremental in nature and where we still add new classes sequentially but the classification problem is unique and, when using the model or computing the accuracy, we need to distinguish among all the classes encountered so far. This is quite common in natural learning, for example in object recognition, as a child learns to recognize new objects, they need to be discriminated w.r.t. the whole set of already known objects (i.e., visual recognition tasks are rarely isolated in nature).

Usually, SIT scenario is more difficult than MT one: in fact, `i)` we still have to deal with catastrophic forgetting; `ii)` we need to learn to discriminate classes that typically we never see

together (e.g. in the same batch), except when a memory buffer is used to store/replay a fraction of past data.

Figure 4.1 graphically highlights the difference between common implementation for addressing an MT and SIT scenario. While for MT the output neurons can be grouped into separate classification layers (one for each task), SIT uses a single output layer including the neurons of all the classes encountered so far. In the MT training phase, the output layer of the batch can be trained apart while sharing the rest of the model weights (denoted as $\bar{\Theta}$ in the figure). This is not the case in SIT where weights learned for the old classes could be exploited when learning the current batch classes. In the MT evaluation phase, assuming to know the task membership of each test sample, each task can be assessed separately with the corresponding classification layer. Instead, in the SIT scenario, the evaluation is performed agnostically with respect to the membership of a sample to a specific incremental batch (i.e. not exploiting the t label) and the final probabilities are computed through a unique *softmax* layer; this requires to compare objects that were never seen together during training and can have a strong impact on final accuracy. Some researchers, in the continual learning context, use the term "head" to denote the output classification layer: using this terminology, the MT scenario can be implemented with multiple disjoint heads, while SIT is characterized by a single expanding head.

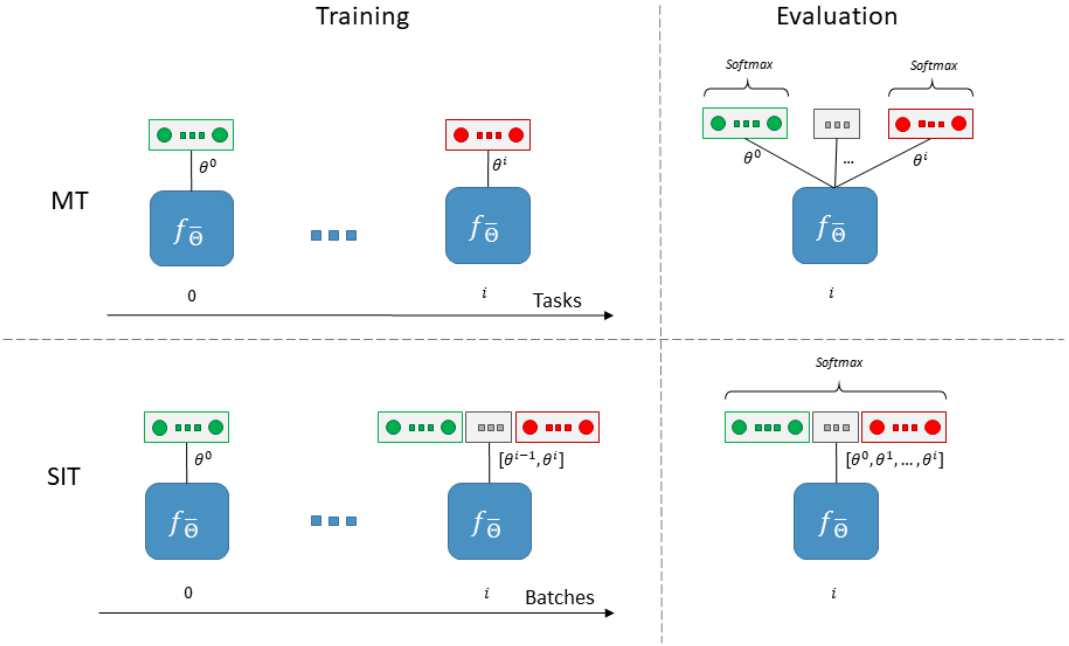


FIGURE 4.1: Key architectural differences between MT and SIT scenarios: a disjoint output layer (also denoted as "head") is used in MT for each independent task, while a single (dynamically expanded) output layer is used in SIT to include all the classes encountered so far. Better viewed in color.

Figure 4.2 provides an example that quantifies how much more complex SIT is than MIT on the *CIFAR-10/100 Split*. For direct comparison with Zenke et al. [2017] here we report the accuracy only at the end of training (i.e., after the 6th batch). It is evident that SIT represents a much more difficult challenge for state-of-the-art CL strategies. Looking at the average accuracy we can notice a gap between MT and SIT of more than 30% regardless the CL technique. Actually,

the unbalanced nature of the *CIFAR-10/100 Split* benchmark (50% of all the train and test set examples belong to the first batch) makes SIT strategies quite harder to parametrize. However, as argued by other researches [Kemker and Kanan, 2018; Kemker et al., 2018], most of the existing CL approaches perform well on MT (with a moderate number of tasks) but fail on complex SIT scenario with several (limited-size) batches. The MIT scenario, as a combination of the two, constitutes still a largely unexplored setting.

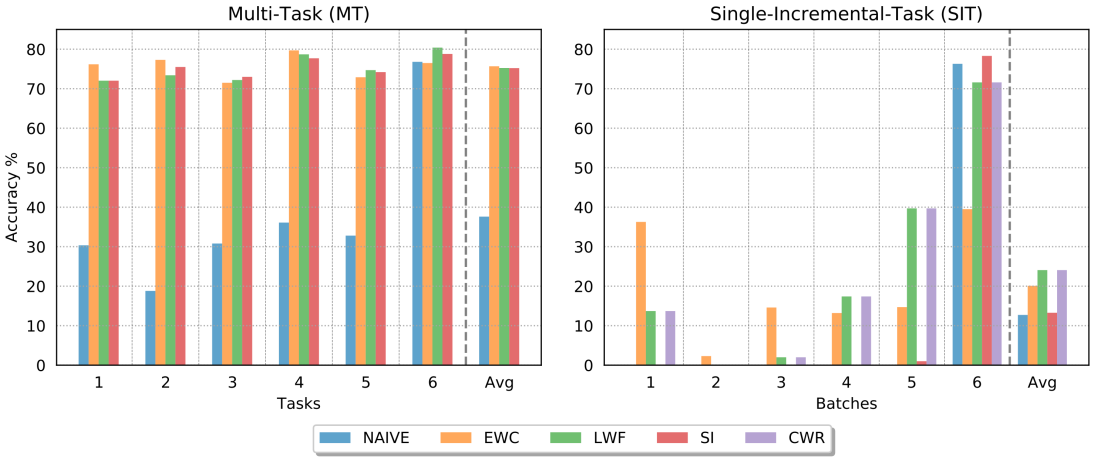


FIGURE 4.2: Accuracy results in the MT and SIT scenarios for 5 common CL strategies (NAIVE, EWC, LWF, SI, CWR) after the last training batch. Analogously to [Zenke et al., 2017], this experiment was performed on the first 6 tasks of CIFAR-10/100 split. For both MT and SIT we report the accuracies on the classes of each batch $(1, 2, \ldots, 6)$ and their average (Avg). CWR is specifically designed for SIT and was not tested under MT. Better viewed in color.

In Table 4.2 we compare existing datasets/benchmarks which, in our opinion, may be also very useful for the development and assessment on novel continual learning algorithms in more realistic settings. Indeed, all these datasets consist of temporal coherent sequences of data (or static frames from which sequences can be easily generated). In principle, datasets without this feature may be used for continual learning as well (splitting them in several batches). However, we think that temporally coherent sequences allow a larger number of real-world applications to be addressed (e.g., robotic vision scenario), since this additional (but natural) structure in data may be exploited from other unsupervised learning techniques [Li and DiCarlo, 2008]. This would also reduce the gap between continual and sequence learning, that, we believe, have a natural interplay worth considering in the near future.

YouTube-8M [Abu-El-Haija et al., 2016] provides a huge number of videos acquired in difficult natural settings. However, the classes are quite heterogeneous and acquisition conditions are completely uncontrolled in terms of object distance, pose, lighting, occlusions, etc. In other words, we believe it is too challenging for current continual learning approaches (still in their infancy).

In the first group of datasets in Table 4.2 (*NORB*, *COIL-100*, *iLAB20M*, *Washington RGB-D*, *BigBIRD*, *ALOI*), objects are positioned on turntables and acquisition is systematically controlled in term of pose/lighting. Neither complex backgrounds nor occlusions are present in these datasets. Exploration sequences can be generated for the other datasets in this group as

TABLE 4.2: Comparison of datasets (with temporal coherent sessions) for continual learning. Temporal coherence, often not considered in current continual learning research, constitute a natural property often encountered in real-world settings when learning continually from a stream of data.

Dataset	Cat.	Obj.	Sess.	Frames per sess.	Format	Acquisition setting	Outdoor sessions
NORB [LeCun et al., 2004]	5	25	20	20	grayscale	turntable	no
COIL-100 [Nene et al., 1996]	-	100	20	54	RGB	turntable	no
iLab-20M [Borji et al., 2016]	15	704	-	-	RGB	turntable	no
RGB-D [Schwarz et al., 2015]	51	300	-	-	RGB-D	turntable	no
BigBIRD [Singh et al., 2014]	-	100	-	-	RGB-D	turntable	no
ALOI [Geusebroek et al., 2005]	-	1000	-	-	RGB	turntable	no
BigBrother [Franco et al., 2009]	-	7	54	∼20	RGB	wall cam.	no
iCubWorld28 [Pasquale et al., 2015b]	7	28	4	∼150	RGB	hand hold	no
iCubWorld-Transf [Pasquale et al., 2016]	15	150	6	∼150	RGB	hand hold	no

well by randomly walking through adjacent static frames in the multivariate parameter space; however, the obtained sequences would remain quite unnatural.

The BigBrother dataset [Franco et al., 2009] has been created starting from 2 DVDs made commercially available at the end of the 2006 edition of the "Big Brother" reality show produced for the Italian TV and documenting the 99 days of permanence of 20 participants in a closed environment. It consists of 23,842 70×70 gray-scale images of faces belonging to 19 subjects (one participant was immediately eliminated at the beginning of the reality show). In addition to the typical training and test sets, an additional large set of images (called "updating set") is provided for incremental learning/tuning purposes. Details about the composition of each set can be found in [Franco et al., 2009], together with the number of days the person lived in the house. However, some subjects lived in the house for a short period and too few images are thus available for an in-depth evaluation. For this reason, a subset of the whole database, referred to as Set_B, has been defined by the authors of [Franco et al., 2009]. It includes the images of the 7 subjects who lived in the house for a longer period (such number of users seems realistic for a home environment application). In Chapter 5, we will compare some continual learning strategies on the Set_B of the Big-Brother dataset consisting of a total of 54 incremental batches. In Fig. 4.3, an example image for each of the different seven subjects of the Set_B is shown. It is worth noting that images have been automatically extracted from the video frames by Viola and Jones detector [Viola and Jones, 2001] and are often characterized by bad lighting, poor focus, occlusions, and non-frontal pose.

The *iCubWorld* datasets [Pasquale et al., 2015b, 2016], instead, have been acquired directly in a robotic vision context and are the closest ones to *CORe50*, which will be presented in the

FIGURE 4.3: Example images of the seven subjects contained in the Set_B of the BigBrother Dataset.

following section. In fact, objects are hand hold at nearly constant distance from the camera and are randomly moved.

4.2 Proposed Benchmarks

TABLE 4.3: Original video benchmarks proposed for continual learning with difficult degree of complexity.

Dataset	Cat.	Obj.	Sess.	Frames per sess.	Format	Acquisition setting	Outdoor sessions
Seq-NORB	5	25	20	20	grayscale	turntable	no
Seq-COIL-100	-	100	20	54	RGB	turntable	no
Seq-iCubWorld28	7	28	9	∼60	RGB	hand hold	no
CORe50	10	50	11	∼300	RGB-D	hand hold	yes (3)
VizDoom 3D Maze	2	6	12	unlimited	RGB	generated	no

Given the limited number of benchmarks for continual learning and especially in more realistic settings were where there is not a clear distinction between continual and sequence learning (i.e. we have stream of temporal coherent data), we propose five different benchmarks which respond to this new need with different degrees of complexity, as summarized in Table 4.3.

With respect to the existing datasets, *CORe50* consists of a higher number of longer sessions (including outdoor ones), more complex backgrounds and also provide depth information (that can be used as extra-feature for classification and/or to simplify object detection). In our opinion, the most important feature of *CORe50*, is the presence of 11 distinct acquisition sessions per object; this allows to define incremental strategies that are long enough to appreciate the learning trends. While preparing this dissertation we noted that *iCubWorld-Transf* is being expanded (see `https://robotology.github.io/iCubWorld/` for latest updates), and we think that cross-evaluating continual learning approaches on both *CORe50* and *iCubWorld-Transf* could be very interesting.

4.2.1 Seq-NORB

Instead of collecting another dataset we focused on the well know and largely used NORB dataset [LeCun et al., 2004]. Despite its simplicity, is still one of the best dataset to study invariant object recognition and well-fit our purposes because it contains 50 objects and 972 variations for each objects. The 50 objects belong to 5 classes (10 objects per class) and the 972 variations

are produced by systematically varying the camera elevation (9 steps), the object azimuth with respect to the camera (18 steps) and the lighting condition (6 steps).

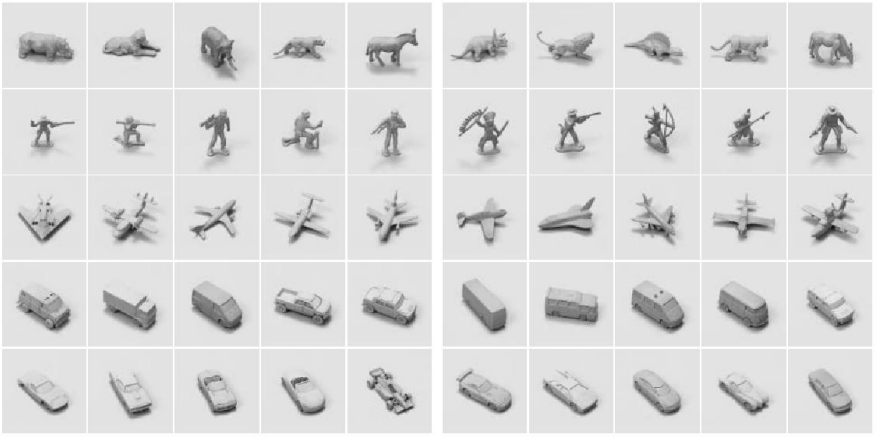


FIGURE 4.4: One image from each of the 50 objects in *NORB* dataset. The five rows denotes the object classes: four-legged animals, human figures, airplanes, trucks, and cars. Objects belonging to the first five columns of the original benchmark are included in the training set, while the others in the test set. Objects are untextured and size normalized so that only shape features can be used for recognition.

Temporally coherent video sequences can be generated from NORB by randomly walking the 3D (elevations, azimuth, lighting) variation space, where consecutive frames are characterized by a single step along one dimension. In our generation approach the random walking is controlled by some parameters like the number of frames, the probability of taking a step along each of the 3 dimensions, the probability of inverting the direction of movement (flip back), etc. Fig. 4.5 (top) shows an example of training sequence. When generating test sequences we must avoid to include frames already used in the training sequences. In particular, when generating test sequences (with a given *mindist*), we ensure that each test frame has a city-block distance of at least mindist steps ($mindist \geq 1$) from any of the training set frames. Fig. 4.5 shows a test sequence with $mindist = 4$ (bottom) with respect to the respective training sequence (top).

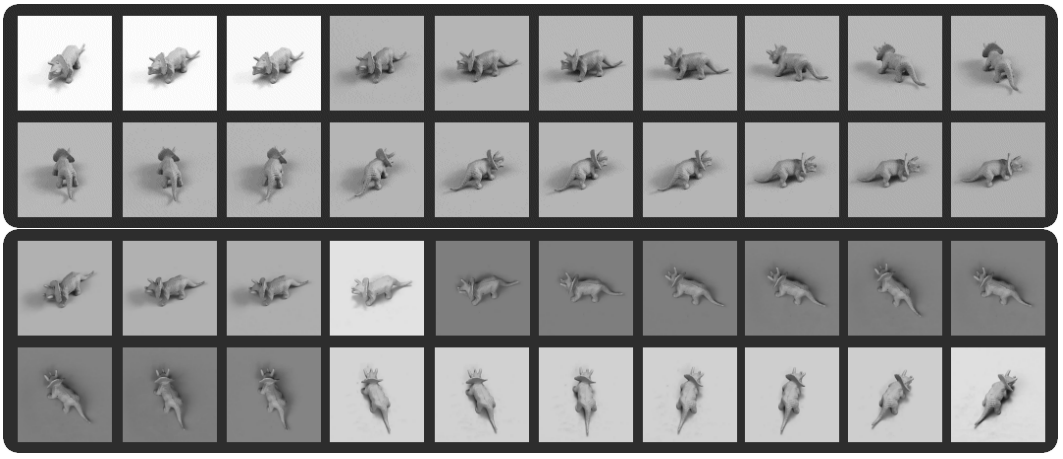


FIGURE 4.5: An example of training sequence of 20 frames (above) and a test sequence (below) with $mindist = 4$ from the previous training sequence.

In the standard NORB benchmark for each of the 5 classes, 5 objects are included in the training set and 5 objects in the test set. In the proposed benchmark we prefer to focus on pose and lighting invariance hence our training and test set are not differentiated by the object identity but by the object pose and lighting (for an amount modulated by *mindist*). However, for completeness, in Appendix B.2 we also report results on an equivalent benchmark where the native object segregation is maintained. In our benchmark we also focus on monocular representation since the availability of stereo information makes the problem unnecessarily simpler for the task at hand. The benchmark dataset used in our experimentation consists of:

- 10 training batches B_i. Each B_i is 1,000 frames wide and is composed by 50 temporally coherent sequences (20 frames wide), each representing one of the 50 objects. B_1 is used for initial training and $B_2, \ldots, B_{10}$ for successive incremental tuning. When training the system on B_i we have no longer access to the previous $B_j, j < i$. We do not enforce any *mindist* among training set sequences, so the same frame can be present in different batches.
- 10 test batches TB_i for each *mindist* = 1, 2, 3 and 4. Test batches are structured as the train batches, but here *mindist* is enforced, so each frame included in the test batches has a distance of at least *mindist* from the 10×1,000 frames[1] of the training batches. Higher *mindist* values make the classification problem more difficult, because patterns are less than similar with respect to the training set ones. The temporal coherent organization of the test batches allows two type of evaluations to be performed:
  - *Frame based classification*: here temporal organization is not considered and each frame has to be classified independently of its sequence/positions in the batch. For simplicity, for each *mindist* we can treat the batches $TB_i, i = 1, \ldots, 10$, as as a single plain test set of 10×1,000 patterns.
  - *Sequence based classification*: this evaluation (not included in the experiments carried out in this dissertation) is aimed at classifying sequences and not single frames, so one can exploit multiple frames per object and their temporal coherence. Of course this is a simpler classification problem due to the possibility of fusing information. As side effect the number of pattern to classify reduces to $\frac{10{,}000}{20} = 500$.

With the purpose of evaluating our approach on a harder problem we can consider another benchmark (denoted as the *"50-class benchmark"*) where each object is considered as a separate class. It is worth noting that this is a quite complex problem due to the sometime small variability among objects originally belonging to the same class. To setup this benchmark we can still use the above sequences, with the only caution of ignoring original class labels and taking object labels as class labels.

Original NORB images are 96×96 pixels. We noted that working on reduced resolution images (up to 32×32) does not reduce classification accuracy (on the 5-class problem). So in order to speed-up the experiments we down-sampled the NORB images to 32×32 pixels[2] The full training and test sequences used in this dissertation (provided as sequences of filenames referring to the original

[1] Actually due to the presence of duplicates in our training random walks, the number of different frames is 8,531 (smaller than 10,000).

[2] The same down-sampling was done in other works [Le et al., 2010; Saxe et al., 2011; Wagner et al., 2013].

NORB images) can be downloaded from [https://github.com/vlomonaco/norb-creator]. In the same repository we make available the tool (and the code) used to generated the sequences.

4.2.2 Seq-COIL100

COIL-100 [Nene et al., 1996] contains a larger number of classes than NORB (100 vs 5), but the available variations for each class are much more limited (72 images per class in COIL-100 vs 9720 images per class in NORB). The 72 poses of each class are spanned by a single mode of variation (i.e., camera azimuth) which is uniformly sampled with 5 degree steps. The single mode of variation and the limited number of poses make the generation of (disjoint) temporally coherent sequences for continual learning quite critical. However, we tried to setup a test-bed close to the Seq-NORB one:

- 6 poses per class (one pose every 60°) are included in the test set; for each test set pose the two adjacent ones (5° before and after) are excluded from the training batches to enforce a *mindist = 2*.
- Temporally coherent sequences are obtained for each class by randomly walking the remaining 54 = 726 -12 frames. Training batches B_i (1000 patterns wide) are then generated and used for initial supervised training (B_1) and successive incremental tuning ($B_2, \ldots, B_{10}$). It is worth noting that with respect to the NORB experiments, in this case the forgetting effect induced by incremental tuning is mitigated by an higher overlapping among the tuning batches due to the small number of frames.
- Also in this case, the images are sub-sampled (from 128×128) to 32×32 and converted from RGB to grayscale for reducing the benchmark complexity.

4.2.3 Seq-iCubWorld28

FIGURE 4.6: Example images of the 28 objects (7 categories) from one of the 4 subsets constituting icubWorld28.

The iCubWorld28 dataset [Pasquale et al., 2015a] consists of 28 distinct domestic objects evenly organized into 7 categories (see Figure 4.6). Images are 128×128 pixels in RGB format. The acquisition session of a single object consists in a video recording of about 20 s where the object is slowly moved/rotated in front of the camera. Each acquisition session results in about 200 train and 200 test images for each of the 28 objects. Being designed to assess the continual learning performance of the iCub robot visual recognition subsystem, the same acquisition approach has been repeated for 4 consecutive days, ending up with four subsets (Day 1, to 4) of around 8 K images each (39,693 in total). To better assess the capabilities of our continual learning strategies we split each training set of Day 1, 2 and 3 in three parts of equal size. On the contrary, Day 4 was left unchanged and entirely used as test set (as in [Pasquale et al., 2015a]). In Table 4.4, we report the full details about the size of the training and test set used for our experiments.

TABLE 4.4: iCubWorld28 batches size and membership to the original Day.

Partition name	Images count	Original Day
$Batch_1$	1341	Day_1
$Batch_2$	1341	Day_1
$Batch_3$	1341	Day_1
$Batch_4$	1789	Day_2
$Batch_5$	1788	Day_2
$Batch_6$	1788	Day_2
$Batch_7$	1836	Day_3
$Batch_8$	1836	Day_3
$Batch_9$	1836	Day_3
$Test$	5550	Day_4

4.2.4 CORe50

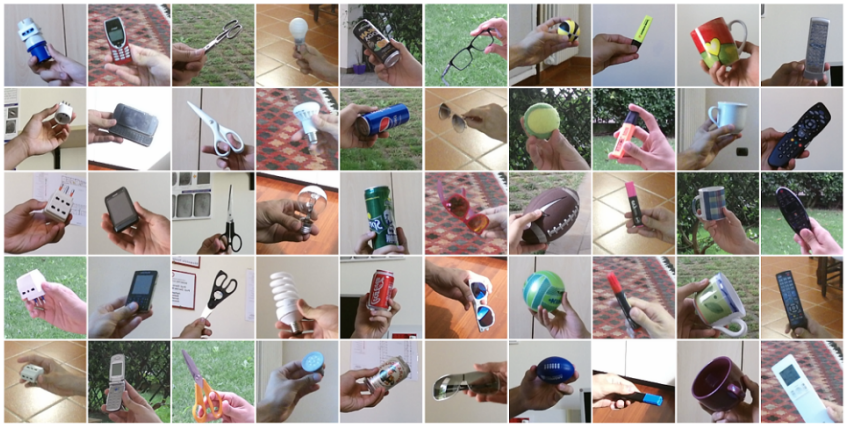


FIGURE 4.7: Example images of the 50 objects in *CORe50*. Each column denotes one of the 10 categories.

CORe50, specifically designed for (C)ontinual (O)bject (Re)cognition, is a collection of 50 domestic objects belonging to 10 categories: plug adapters, mobile phones, scissors, light bulbs, cans, glasses, balls, markers, cups and remote controls (see Figure 4.7). Classification can be

performed at object level (50 classes) or at category level (10 classes). The first task (the default one) is much more challenging because objects of the same category are very difficult to be distinguished under certain poses. The dataset has been collected in 11 distinct sessions (8 indoor and 3 outdoor) characterized by different backgrounds and lighting. For each session and for each object, a 15 seconds video (at 20 fps) has been recorded with a Kinect 2.0 sensor [Steward et al., 2015] delivering 300 RGB-D frames. Objects are hand hold by the operator and the camera point-of-view is that of the operator eyes. The operator is required to extend his arm and smoothly move/rotate the object in front of the camera. A subjective point-of-view with objects at grab-distance is well-suited for a number of robotic applications. The grabbing hand (left or right) changes throughout the sessions and relevant object occlusions are often produced by the hand itself.

Row data consists of 1024×575 RGB + 512×424 Depth frames. Depth information can be mapped to RGB coordinates upon calibration. The acquisition interface identifies a central region where the object should be kept (see red box in Figure 4.8). This allows to performs a first (fixed) cropping, thus reducing the frame size to 350×350.

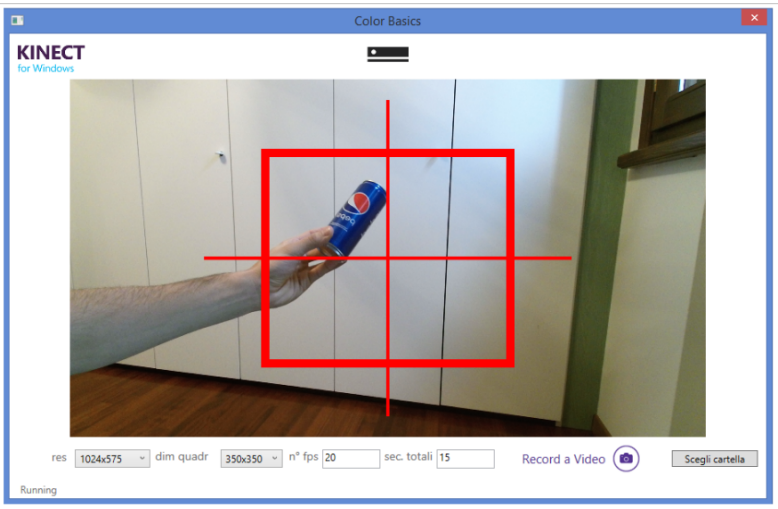


FIGURE 4.8: Acquisition interface: the red box identifies the central region where the operator is required to keep the objects while moving and rotating them.

Since our domestic objects (kept at arm distance) typically extend for less than 100×100 pixels, only a small fraction of the frame contains the object of interest. Therefore, we exploited temporal information to crop from each 350×350 frame a 128×128 box around the object. To this purpose we implemented a simple but effective motion-based tracker working only on RGB data, so that a similar approach could be used even if depth information is not available (see Figure 4.9 for an example). While in most of the cases the objects are fully contained in the crop window, sometimes they can extend beyond borders (e.g., this can happen if the object distance from the camera is reduced too much, or the tracker partially loses the object because of a too fast movement). No manual correction has been applied, because we believe that tracking imperfections are unavoidable and should be properly dealt with at later processing stages.

The final dataset consists of 164,866 128×128 RGB-D images: 11 sessions × 50 objects × (~300[3])

[3]Some sequences are slightly shorter than 300 frames because a few initial frames are necessary to initialize the automatic motion-based tracker.

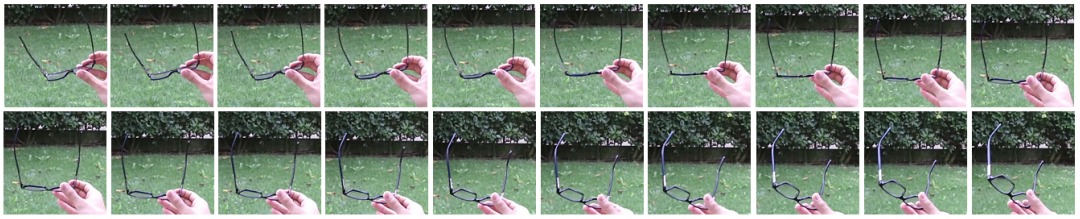


FIGURE 4.9: Example of 1 second recording (at 20 fps) of object #26 in session #4 (outdoor). Note the smooth movement, pose change and partial occlusion. The 128×128 frames here shown have been automatically cropped from 350×350 images based on a fully automated tracker.

frames per session. Figure 4.9 shows one frame of the same object throughout the eleven sessions. Three of the eleven sessions (#3, #7 and #10) have been selected for test and the remaining 8 sessions are used for training. We tried to balance as much as possible the difficulty of training and test sessions with respect to: indoor/outdoor, holding hand (left or right) and complexity of the background.

FIGURE 4.10: One frame of the same object (#41) throughout the 11 acquisition sessions. Note the variability in terms of background, illumination, blurring, occlusion, pose and scale.

The full dataset, along with further information can be downloaded from `vlomonaco.github.io/core50`. In the same repository we make available the code for the reproducibility of the benchmarks described in the following sections.

Static Object Recognition Benchmark While designed for continual learning, *CORe50* dataset can still be used as a medium size benchmark for object recognition with a static evaluation protocol. The high object pose variability and complex acquisition setting make the problem sufficiently hard to solve even when learning is performed on the whole training data.

In Table 4.5, we show the accuracy of two well-known CNN models (CaffeNet and VGG[4]) adapted to medium size and trained in three different modalities by using RGB data only (depth information will be used in future studies):

1. Mid-CNN from scratch: training a model from scratch.
2. Mid-CNN + SVM: using a model pre-trained on *ILSVRC-2012* as a fixed feature extractor in conjunction with a linear SVM classifiers. Features are extracted at *pool5* level.

[4]We refer to the VGG-CNN-M model introduced by Chatfield et al. [2014].

3. Mid-CNN + FT: fine-tuning an *ILSVRC-2012* pre-trained model on *CORe50*.

As already shown by many authors, fine-tuning a pre-trained model on the new dataset is often the most effective strategy, especially if the new dataset is large enough to avoid overfitting, but not so large to learn representative features from scratch.

TABLE 4.5: Accuracy of CaffeNet and VGG models (both adapted to size 128×128) on *CORe50* for different learning strategies. The test set consists of sessions: #3, #7 and #10; the training set of the remaining 8 sessions.

	Accuracy % (object level: 50 classes)		**Accuracy % (category level: 10 classes)**	
Strategy	**CaffeNet**	**VGG**	**CaffeNet**	**VGG**
Mid-CNN from scratch	37,82%	38,09%	48,93%	53,74%
Mid-CNN + SVM	51,35%	59,03%	61,81%	68,94%
Mid-CNN + FT	65,98%	69,08%	77,76%	80,23%

The term Mid-CNN is here used to highlight that we are not using the original 227×227 CaffeNet and 224×224 VGG models but their adaption to a mid-size of 128×128 pixels. Many researchers use available pre-trained CNN models as they are, and simply stretch their images to fit the model input size, even if the image size is much smaller than the CNN input. Stretching our input pattern (from 128×128 to 227×227) would require much more computation at inference time (about four times), so we decided to adapt the pre-trained CNN models to work with 128×128 input images. However, in case of pre-trained models, this step is not neutral and obvious as one could expect: more details are provided in Appendix C.

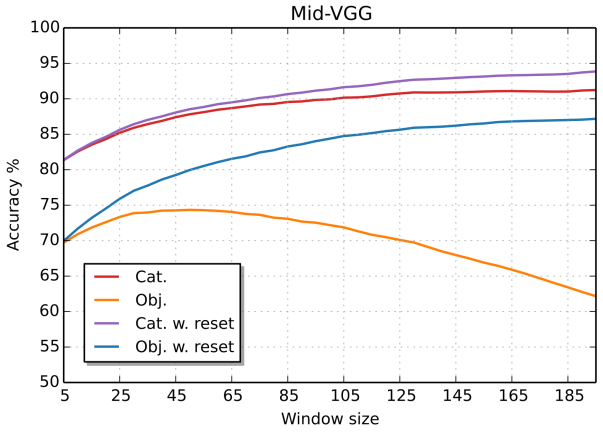


FIGURE 4.11: Mid-VGG classification accuracy (at object level and category level) when classification confidence over more adjacent frames is fused. On the horizontal axis the number of frames fused (temporal window). When end-of-sequence reset is not available using long temporal windows can lead to dangerous drifts (see the orange curve).

To improve classification accuracy, instead of classifying single frames, a set of temporally adjacent frames can be fused. To this purpose we implemented a simple sum-rule fusion at confidence level. The graph in Figure 4.11 shows the result for the Mid-VGG model. For each classification experiment (object level and category level) we tested two cases: `i`) we concatenate frames from all test sequences without considering end-of-sequence events (reset); `ii`) we assume that a reset signal is available. In the former, as the window size increases the risk of fusing frames from

different classes increases as well. In general, fusing 40-50 frames (about 2 seconds of video) seems to be a good compromise even when sequences cannot be reliably segmented.

4.2.5 3D VizDOOM Maze

Continual Learning in reinforcement learning environments is still in its infancy. Despite the the obvious interest in applying CL in less supervised setting and the early, promising results in this context [Kirkpatrick et al., 2017; Ring, 1994], reinforcement learning environments constitute a more complex setting for easily disentangle the ability to learn continuously from the lack of supervision.

It is also worth noting that state-of-the-art reinforcement learning algorithms and current hardware computational capabilities does not make prototyping and experimentations easily accomplished on complex environment where physical simulation constitute an hard problem per-se. This is even harder in a continual learning context where an exposition of the same model to a sequential stream of data is needed (and cannot be parallelized by definition). This is why, recent reinforcement learning algorithms for continual learning have been tested only on arguably simple benchmarks of low/medium input space dimension and complexity [Caselles-Dupré et al., 2018; Kaplanis et al., 1987; Kirkpatrick et al., 2017].

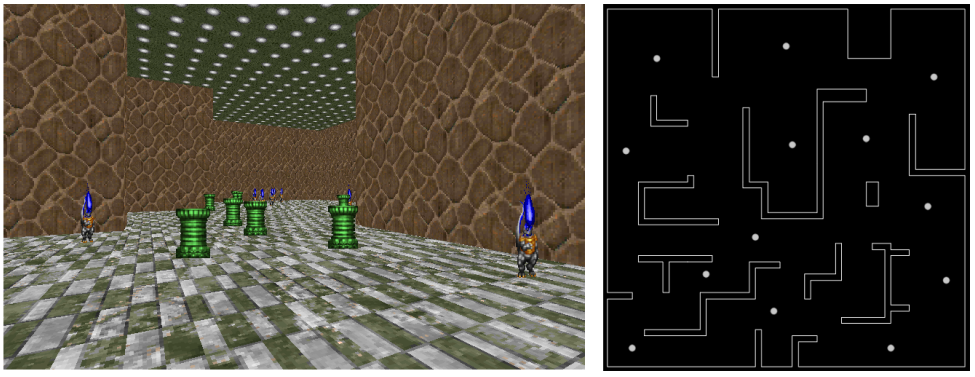


FIGURE 4.12: The 3D maze environment developed with ZDOOM and Slade. On the right an example image from the point of view of the agent is reported. On the right the plenary view of the maze structure is shown. White point on the map represent random spawning points used by the agent during both training and test episodes. Better viewed in colors.

Nevertheless, in parallel, classic reinforcement learning algorithms have started to tackle more complex problems in 3D environments. VizDOOM [Kempka et al., 2016], followed soon after by other research platforms like DeepMind Labs [Beattie et al., 2016] and Malmo [Johnson et al., 2016], allowed researchers to start exploring new, interesting research directions with the aim to scale reinforcement learning, limiting the apparent enormous amount of trials generally needed. VizDoom is essentially a reinforcement learning API build around the famous *"Doom II: Hell on Earth"* a first-person shooter game originally released for MS-DOS computers in 1994 by *id Software* and providing all the necessary utilities to train your RL agent inside it.

VizDOOM is particularly interesting since it has been released open-source by *id Software* for non-profit use and later ported by the open-source community also to UNIX systems like Linux

and Mac OS. Moreover, it was already built on the idea of flexibility and customization allowing users to create custom levels and otherwise modify the game using WAD files turned out to be a popular aspect of Doom leading to the first large mod-making community. Slade is the most common and flexible open-source map editor created for DOOM and it has been extensively used by the research community to create a number of DOOM environments for many reinforcement learning tasks of various level of difficulty.

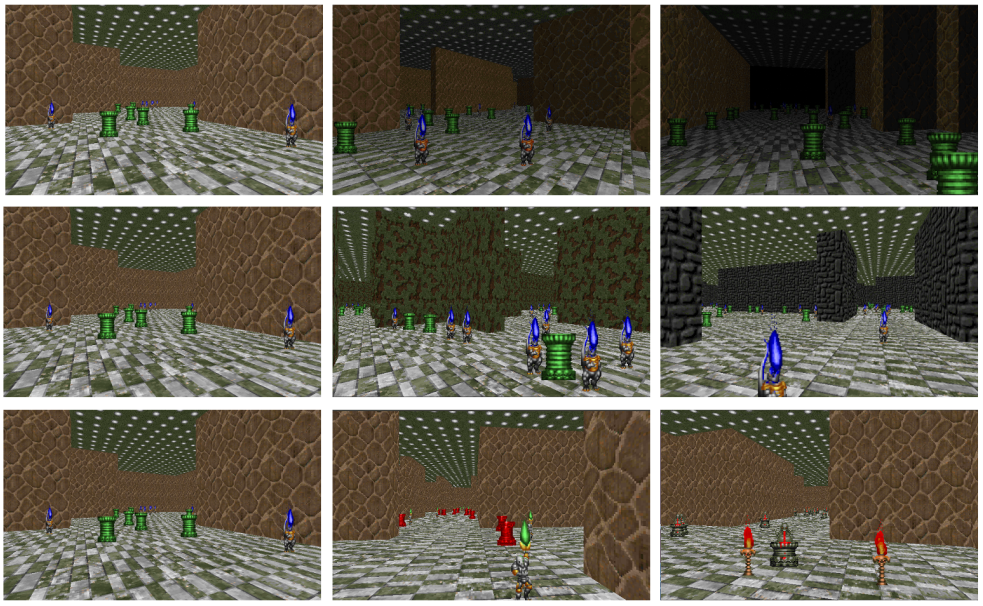


FIGURE 4.13: The environmental changes for each of scenario (Light, Texture, Object) in the 3D VizDOOM Maze. For all the environments changes are not gradual but happening at three specific points equidistant in time corresponding to the columns in figure. Better viewed in colors.

In this dissertation we propose an original 3D VizDOOM environment for continual learning and an object picking task (see Fig. 4.12). In particular, the task consist of learning how to navigate in a complex maze and pick *"column bricks"* while avoiding *"flaming lanterns"*. However, the environment in this case is *"non-stationary"* meaning that is subject to change leading to major difficulties for standard reinforcement learning algorithms.

For properly assessing novel continual reinforcement learning strategy in 3D complex environments we split the benchmarks in four different tasks of incremental difficulty with respect to different environmental changes (see Fig. 4.13):

1. **Light**: In this environment the light and visibility of the agent is changed.
2. **Texture**: In this environment walls textures are changed.
3. **Object**: In this environment the objects to pick or avoid are changed in shape and color.
4. **All together**: In this environment both light, textures as well as objects are subject to change.

For all the environments changes are not gradual but happening at three specific points equidistant in time and practically impremented as differen ZDOOM *MAPs*.

4.3 Training and Evaluation Protocols

The continual learning training protocol is the straightforward extension of what is normally done in classic machine learning on fixed training set to a sequence of multiple training batches. However, in particular cases, shuffling the order of the training batches over multiple runs may be needed for assessing stability of the proposed algorithms. More complex cross-validation techniques may be also conceived but are not very common in current CL research.

Defining an optimal testing protocol is a bit more delicate and not obvious. For example, focusing on a classification problem in the SIT-NC scenario, we could initially considered three alternatives:

1. **Partial Test Set**: at each evaluation step (i.e., after each training batch) the test set includes only patterns of the classes already presented to the network.
2. **Full Test Set**: the test set is fixed and includes patterns of all the classes. Except for the last evaluation step, the model is (also) required to classify patterns of never seen classes.
3. **Full Test Set with Rejection Option**: the test set is fixed and includes patterns of all classes, however the model has the possibility to reject a pattern if it believes the pattern does not belong to any of the known classes. Since the training set does not include "negative" examples we cannot add an extra neuron for the "unknown" class, and the rejection mechanism has to compare the max class probability with a given threshold.

Option 1. has the drawback that as we increase the number of classes in the test set the task becomes more complex and it is difficult to appreciate the learning trend and benefits of subsequent batches. Option 3. is the most realistic one for real applications but evaluation and comparison of different techniques is more difficult because at each step instead of a single point we have a ROC curve (accuracy also depends on the threshold). Considering that our aim is comparative evaluation among continual learning approaches we believe that option 2. is a good trade-off between simplicity and usefulness for the task. This option also maintains the test set coherent across all scenarios or update-content-type and can be very useful for accounting several aspects related to the impact of current learning on future data/tasks.

4.3.1 Metrics

For a deep evaluation, we can assume to have access to a series of test sets TB_i over time. 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 correspondent training batch B_i.

However, as discussed in [Lopez-paz and Ranzato, 2017], a different granularity of the evaluation at the task level can as well be achieved by having the same test batch for many B_i. For simplicity, in the description metrics described below we assume to have access to each TB_i, and define the *cumulative training set* and *cumulative test set* respectively as:

$$B_i^C = \bigcup_{i=1}^{i-1} B_i, \quad TB_i^C = \bigcup_{i=1}^{i-1} TB_i. \tag{4.1}$$

One of the first metrics for CL was proposed in [Hayes et al., 2018b] as an overall performance Ω in a supervised classification setting. It is based on the relative performance of an incrementally trained algorithm with respect to an offline (which has access to all the data at once) trained algorithm, which in our notation would be:

$$\Omega = \frac{1}{n} \sum_{i=1}^{n} \frac{A(h_i, TB_i^C)}{A(h_i^C, TB_i^C)}. \tag{4.2}$$

Where A is the accuracy measure (taking a model and a test set as input), h_i is the hypothesis trained on the sequence of training batches up to the B_i (our CL strategy) and h_i^C is the best hypothesis we can train off-line having access to all the data in B_i^C at once, and hence, our upper bound.

Serra et al. [2018] tried to directly model forgetting with the proposed *forgetting ratio* metric ρ, defined as:

$$\rho = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{A(h_{i-1}, TB_i^C) - \bar{b}_i}{A(h_i^C, TB_i^C) - \bar{b}_i} - 1 \right) \tag{4.3}$$

Where, $\bar{b}$ is the vector of test accuracies for each TB_i at random initialization.

Always in the same setting, in [Lopez-paz and Ranzato, 2017] other three important metrics are proposed: *Average Accuracy* (ACC), *Backward Transfer* (BWT), *Forward Transfer* (FWT). In this case, after the model finishes learning about the training batch B_i, its performance is evaluated on all (even future) test batches:

$$A = A(h_n, TB_n^C) \tag{4.4}$$

$$BWT = \frac{1}{n-1} \sum_{i=1}^{n-1} A(h_n, TB_i) - A(h_i, TB_i) \tag{4.5}$$

$$FWT = \frac{1}{n-1} \sum_{i=2}^{n} A(h_n, TB_i) R_{n,i} - \bar{b}_i. \tag{4.6}$$

The larger these metrics, the better the model. If two models have similar ACC, the most preferable one is the one with larger BWT and FWT. Note that it is meaningless to discuss backward transfer for the first batch, or forward transfer for the last batch.

While forgetting and knowledge transfer could be quantified and evaluated in various way, as argued in [Farquhar and Gal, 2018; Hayes et al., 2018b], 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. Stability is another important property that should be evaluated since in

many robotics tasks and safety-critical conditions, potential abrupt performance drifts would be a major concern when learning continuously.

The metrics presented above in a supervised classification context can also be generalized with different performance measure P, instead of A, and used in different setting like reinforcement and unsupervised learning. For example, in an adversarial and generative CL setting, P could be a distance based function, such as the Euclidean distance between real and generated images [Seff et al., 2017].

4.3.2 Proposed metrics

The lack of consensus in evaluating continual learning algorithms and the almost exclusive focus on forgetting motivate us to propose a more comprehensive set of implementation independent metrics accounting for several factors we believe have practical implications worth considering in the deployment of real AI systems that learn continually: accuracy or performance over time, backward and forward knowledge transfer, memory overhead as well as computational efficiency.

Drawing inspiration from the standard *Multi-Attribute Value Theory* (*MAVT*) [Ishizaka and Nemery, 2013] we further propose to fuse these metrics into a single score for ranking purposes and we evaluate our proposal with five continual learning strategies on the iCIFAR-100 continual learning benchmark.

In order to provide bounds to each metric (originally lying in $f : [0, \infty[$), we map it to a $[0, 1]$ range (as it is commonly done, e.g., in MAVT and formulate it so that its optimal value is given by its maximization. This is to preserve interpretability of the proposed aggregating CL_{score} metric, and allow to evaluate CL algorithms with respect to multiple criteria, rank them from best to worst, and accommodate weighting schemes according to constraints and desiderata.

Accuracy (A) Given the train-test accuracy matrix $R \in \mathbb{R}^{n \times n}$, which 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 [Lopez-paz and Ranzato, 2017], Accuracy (A) considers the average accuracy for training set B_i and test set TB_j by considering the diagonal elements of R, as well as all elements below it (see Table 4.7):

$$ACC = \frac{\sum_{i \geq j}^{n} R_{i,j}}{\frac{n(n+1)}{2}} \tag{4.7}$$

While the ACC criteria was originally defined to asses the performance of the model at the end of the last task [Lopez-paz and Ranzato, 2017], we believe that an accuracy metric that takes into account the performance of the model at *every timestep i in time* better characterizes the dynamic aspects of CL. The same idea is applied to the modified BWT and FWT metrics introduced below.

Backward Transfer (BWT) *Backward Transfer* measures the influence that learning a task has on the performance on previous tasks [Lopez-paz and Ranzato, 2017]. The motivation arises when an agent needs to learn in a multi-task or data stream setting. The lifelong abilities to both improve and not degrade performance are important and should be evaluated throughout its lifetime. It is defined as the accuracy computed on TB_i right after learning B_i as well as at the end of the last task on the same test set. (see Table 4.7). Here, as in the accuracy metric, we expand it to consider the average of the backward transfer *after each task*:

$$BWT = \frac{\sum_{i=2}^{N} \sum_{j=1}^{i-1} (R_{i,j} - R_{j,j})}{\frac{n(n-1)}{2}} \tag{4.8}$$

Because the original meaning of BWT assumed positive values for backward transfer and negative values to define (catastrophic) *forgetting*, in order to map BWT to also lie on $[0, 1]$ and give more importance to two semantically different concepts, BWT is broken into two different clipped terms: the originally negative (forgetting) BWT (now positive), i.e., **Remembering**, as $REM = 1 - |min(BWT, 0)|$ and (the originally positive) BWT, i.e., improvement over time **Positive Backward Transfer** $BWT^+ = max(BWT, 0)$.

Forward Transfer (FWT) It measures the influence that learning a task has on the performance of future tasks [Lopez-paz and Ranzato, 2017]. Following the spirit of the previous metrics we modify it as the average accuracy for the train-test accuracy entries $R_{i,j}$ above the principal diagonal of R, excluding it (see elements accounted in Table 4.7). Forward transfer can occur when the model is able to perform *zero-shot* learning. We therefore redefine FWT as:

$$FWT = \frac{\sum_{i<j}^{n} R_{i,j}}{\frac{n(n-1)}{2}} \tag{4.9}$$

Model size efficiency (MS) The memory size of model h_i quantified in terms of parameters θ at each task i, $Mem(\Theta_i)$, should not grow too rapidly with respect to the size of the model that learned the first task, $Mem(\Theta_1)$. Model size (MS) is thus:

$$MS = min(1, \frac{\sum_{i=1}^{N} \frac{Mem(\theta_1)}{Mem(\theta_i)}}{N}) \tag{4.10}$$

Samples storage size efficiency (SSS) Many CL approaches save training samples as a replay strategy to not forget. The memory occupation in bits by the samples storage memory M, $Mem(M)$, should be bounded by the memory occupation of the total number of examples encountered at the end of the last task, i.e. the cumulative sum of Tr_i here defined as the lifetime dataset D (associated to the set of all distributions $\mathcal{D}$). Thus, we define Samples Storage Size (SSS) efficiency as:

$$SSS = 1 - min(1, \frac{\sum_{i=1}^{N} \frac{Mem(M_i)}{Mem(D)}}{N}) \tag{4.11}$$

Computational efficiency (CE) Since the computational efficiency is bounded by the number of multiplication and addition operations for the training set Tr_i, we can define the average CE across tasks as:

$$CE = min(1, \frac{\sum_{i=1}^{n} \frac{Ops\uparrow\downarrow(B_i)\cdot\varepsilon}{Ops(B_i)}}{n}) \tag{4.12}$$

where $Ops(B_i)$ is the number of (mul-adds) operations needed to learn B_i, and $Ops\uparrow\downarrow(B_i)$ is the number of operations required to do one forward and one backward (back-propagation) pass on B_i. When the value of $Ops\uparrow\downarrow(B_i)$ is negligible w.r.t. $Ops(B_i)$, a scaling factor associated to the number of epochs needed to learn B_i, ε larger than a default value of 1, can be used to make CE more meaningful (i.e. avoiding compression of the values very near to zero). Since we are essentially moving the lower bound of the computation, which depends on the benchmark complexity, this adjustment also translates on better interpretability of CE (Fig. 4.14)

In order to assess a CL algorithm A^{CL}, following [Ishizaka and Nemery, 2013], each criterion $c_i \in \mathcal{C}$ (where $c_i \in [0,1]$) is assigned a weight $w_i \in [0,1]$ where $\sum_i^{\mathcal{C}} w_i = 1$. Each c_i should be the average of r runs. Therefore, the final CL_{score} to maximize is computed as:

$$CL_{score} = \sum_{i=1}^{\#\mathcal{C}} w_i c_i \tag{4.13}$$

where each criterion c_i that needs to be minimized is transformed to $c_i = 1 - c_i$ to preserve increasing monotonicity of the metric (for overall maximization of all criteria in $\mathcal{C}$). $CL_{stability}$ is thus:

$$CL_{stability} = 1 - \sum_{i=1}^{\#\mathcal{C}} w_i stddev(c_i) \tag{4.14}$$

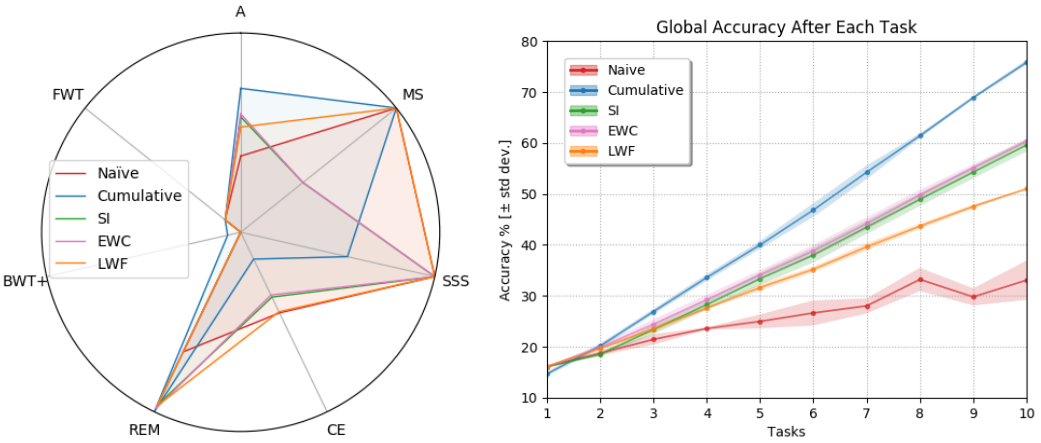


FIGURE 4.14: a) Spider chart: CL metrics per strategy (larger area is better). b) Accuracy per CL strategy computed over the fixed test set.

Experiments and Conclusions We evaluate the CL metrics on *Cumulative* and *Naive* as baseline strategies, EWC, SI, LWF on the iCIFAR-100 benchmark: each task consists of a training batch of 10 (disjoint) classes at a time.

Results for each proposed metric are illustrated in Table 4.6 (each criterion c_i reports the average over 3 runs); Fig. 4.14 illustrates the CL metrics variability for each criterion reflecting a desirable property of CL algorithms, as well as the needs of novel techniques addressing different aspects than accuracy and forgetting, that can be important depending on the application. For simplicity, we chose an homogeneous configuration of criteria weights that values each CL metric equally (i.e., each $w_i = \frac{1}{\#\mathcal{C}}$). However, Table 4.8 shows results on other possible configurations.

While the CL_{score} is optional to report, the aim of the metrics and results is to stimulate comprehensive evaluation practices. In future work we plan to refine these metrics and assess more strategies in more exhaustive evaluation settings.

TABLE 4.6: CL metrics and CL_{score} for each CL strategy evaluated (higher is better).

Strategy	**A**	**REM**	**BWT$^+$**	**FWT**	**MS**	**SSS**	**CE**	$\boldsymbol{CL_{score}}$	$\boldsymbol{CL_{stability}}$
Naive	0.3825	0.6664	0.0000	0.1000	**1.0000**	**1.0000**	**0.4492**	0.5140	**0.9986**
Cumul.	**0.7225**	**1.0000**	**0.0673**	0.1000	**1.0000**	0.5500	0.1496	0.5128	0.9979
EWC	0.5940	0.9821	0.0000	0.1000	0.4000	**1.0000**	0.3495	0.4894	0.9972
LWF	0.5278	0.9667	0.0000	0.1000	**1.0000**	**1.0000**	0.4429	**0.5768**	**0.9986**
SI	0.5795	0.9620	0.0000	0.1000	0.4000	**1.0000**	0.3613	0.4861	0.9970

TABLE 4.7: Elements in R accounted to compute the Accuracy (white and cyan elements), BWT (in cyan), and FWT (in light gray) criteria. $R^* = R_{ii}$, Tr_i = training, Te_i= test tasks.

R	Te_1	Te_2	Te_3
Tr_1	R^*	R_{ij}	R_{ij}
Tr_2	R_{ij}	R^*	R_{ij}
Tr_3	R_{ij}	R_{ij}	R^*

Matrix $R \in \mathbb{R}^{n \times n}$ contains in each entry $R_{i,j}$ the test classification accuracy of the model on task j after observing the last sample from task i [Lopez-paz and Ranzato, 2017]. Table 4.7 shows the elements in the accuracy matrix used for each metric for an example matrix of $n = 3$ tasks. $R^* = R_{ii}$ coincides with the (normally) optimal accuracy right *after* using training set B_i and testing on test set TB_i.

Note that in order to compute Accuracy, we do not only consider as [Lopez-paz and Ranzato, 2017] the last row of the accuracy matrix R, but also steps in between each new training set learned, to acknowledge the degradation and improvement through every time step in time.

In FWT, the subtraction term (vector b_i of test accuracies for each task at random initialization) in the original FWT formula in [Lopez-paz and Ranzato, 2017] was removed in our definition of FWT in order to guarantee non negative values (i.e. in case of negative FWT) and allow for potential positive transfer, as they demonstrate it is possible to happen with a shared output space. The idea is supporting the fact that algorithms can do worse than random accuracy for some strategies (we refer the reader to [Lopez-paz and Ranzato, 2017] for cases of positive FWT).

The original BWT [Lopez-paz and Ranzato, 2017] would return domains for $BWT^- \in [0, 0.5)$, and for $BWT^+ \in [0.5, 1]$, respectively which, through the clipping, are transformed, as the rest of criteria in the CL metric, to stay in [0,1].

Despite the experiments showing the CL_{score} to be optional and context dependent; the aggregation score is most meaningful when a community agrees on a particular evaluation criteria (similarly to the mAP metric), or in specific settings where the weights for the different criteria are clearly definable. Our experiments use three weight configurations $W = [w_A, w_{MS}, w_{SSS}, w_{CE}, w_{BWT}, w_{Rem}, w_{FWT}]$. The first one used homogeneous weights (each $w_i = \frac{1}{\#\mathcal{C}}$) and the second and third use W_2 = [0.4, 0.1, 0.1, 0.1, 0.2, 0.05, 0.05] and W_3 = [0.4, 0.05, 0.2, 0.2, 0.05, 0.05, 0.05], as particular examples aiming at reflecting what the recent CL literature has roughly been valuing the most; however, any configuration could be used.

The CNN model used in this experiment is the same used in [Zenke et al., 2017] and consists of 4 convolutional + 2 fully connected layers (details available in Appendix A of [Zenke et al., 2017]). Hyper-parameters are chosen to maximize the accuracy metric A for each strategy.

Spider chart in Fig 4.14 shows all objective criteria, where the larger the area occupied under the CL algorithm curve, the highest CL_{score} (more optimal) it is. Fig. 4.15 shows each of the main CL strategies put in context compared with the considered lower and upper bounds respectively, i.e., naive, and cumulative strategies. The farther away the evaluated strategy is from the cumulative (blue) surface, the larger room for improvement for the CL strategy.

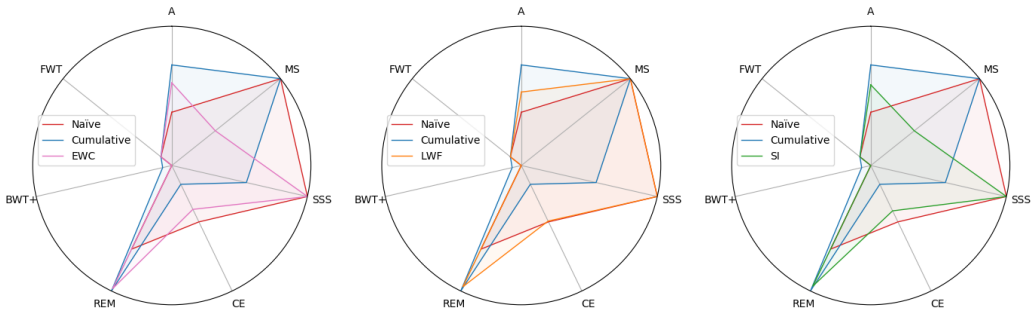


FIGURE 4.15: Spider chart with CL metrics showing CL strategies EWC, LWF and SI with their respective lower and upper bound (Naive and Cumulative resp.) as reference baselines (to properly visualize Fig. 4.14). The weight configuration for each criterion used is W_1 where $w_i = \frac{1}{7}$ for each $w_i \in W$.

TABLE 4.8: CL_{score} and $CL_{stability}$ for all CL strategies according to different weighting configurations $W_i = [w_A, w_{MS}, w_{SSS}, w_{CE}, w_{REM^+}, w_{BWT}, w_{FWT}]$, where W_1 sets $w_i = \frac{1}{7}$ for each $w_i \in W$. The second setting of a concrete metric weights is W_2 = [0.4, 0.05, 0.2, 0.1, 0.15, 0.05, 0.05]. A third arbitrary configuration is W_3 = [0.4, 0.05, 0.2, 0.2, 0.05, 0.05, 0.05].

Strategy/CL Metric	CL_{score}			$CL_{stability}$		
	W_1	W_2	W_3	W_1	W_2	W_3
Naive	0.5140	0.5529	0.5312	**0.9986**	0.9969	**0.9973**
Cumulative	0.5128	0.6223	0.5373	0.9979	0.9976	0.9964
EWC	0.4894	0.6449	0.5816	0.9972	0.9976	0.9940
LWF	**0.5768**	**0.6554**	**0.6030**	**0.9986**	**0.9990**	0.9972
SI	0.4861	0.6372	0.5772	0.9970	0.9945	0.9927

4.4 Learning Dynamics Interpretation

Hyper-parameters tuning is not easy for complex deep architectures and is still more complex for continual learning over sequential batches. Simply looking at the accuracy trend along the batches is not enough to understand if an approach is properly parametrized. For example, a poor accuracy can be due to insufficient learning of the new classes or by the forgetting of the old ones.

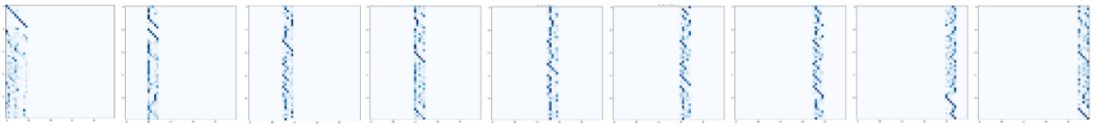


FIGURE 4.16: Sequence of confusion matrices computed after each training batch for the Naive approach on CaffeNet. On the vertical axis the true class and on the abscissa the predicted class.

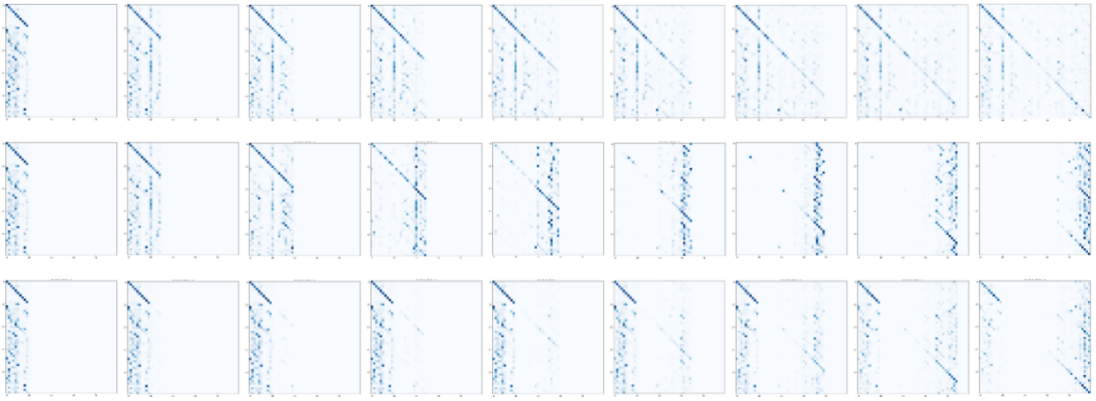


FIGURE 4.17: Sequence of confusion matrices computed after each training batch $(1, \ldots, 9)$ for the LWF approach and CaffeNet. In the first row the approach is properly parametrized (variable λ and *map* function, see Table D.3) and the model is able to continuously learn new classes without forgetting the old ones. In the second we used the same $\lambda_i = 0.5$ for all the batches: for $B_2, \ldots, B_3$ the regularization is appropriate but for successive batches is too light leading to excessive forgetting. In the third row we used the same $\lambda_i = 0.8$ for all the batches: for $B_2, \ldots, B_6$ the regularization is too strong and learning of corresponding classes is poor.

In our experience visualizing the confusion matrices (CM) is very important to understand what is happening behind the scenes. Looking at the last CM (that is after the last batch) is often not sufficient and the entire CM sequence must be considered. Figure 4.16 shows the CM sequence (one after each batch) for the naive approach: forgetting is clearly highlighted by a vertical band moving from the left to the right to cover the classes of the most recent batch. Figure 4.17 show three CM sequences for LWF approach on CaffeNet: `i)` in the first row parametrization is good; `ii)` in the second row the model forget to much old classes, regularization should be increases; `iii)` in the third row initial regularization is too strong and the model cannot learn classes in the corresponding batches.

Unfortunately, the trade-off stability/plasticity does not depends only of the regularization strength (e.g. λ_i for LWF) because the learning rate and the number of training epochs are also indirectly related. However, looking at the CM sequence allows to understand what is the

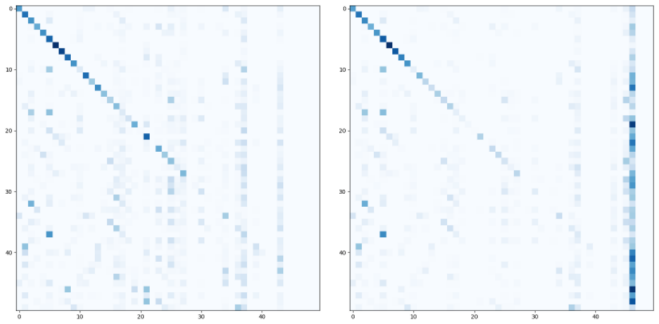


FIGURE 4.18: Confusion matrices after computed after B_8 and B_9 for the EWC approach and CaffeNet. As discussed in Section 5.1.4 EWC tends to saturate CaffeNet capacity after the first 5-6 batches. In this specific run the amount of training on the last batch is too high w.r.t. the residual model capacity and the learning result are poor: the sharp vertical band on the right is an alarm signal.

direction of change of one or more parameters. It can also happen that the amount of regularization is good for the first batches, but unsatisfactory for the successive ones; the CM sequence easily reveal this and allow to take countermeasures (e.g. the *map* function for LWF). Finally, when a model is strongly regularized its learning capacity tend to saturate (e.g. in EWC there are no "free" parameters to move in order to learn new classes); this is usually reflected by anomalies such as sharp vertical bars in correspondence of one or few classes (see Figure 4.18).

Another useful diagnostic technique is visualizing (in the form of a 3D histogram) the average amount of change of the weights in each layer at the end of each batch. To avoid cancellation due to different sign, we compute the average of absolute values. Figure 4.19 shows the histograms for CaffeNet. We can observe that in the naive approach weights (which are not constrained) are significantly changed throughout all the layers for all batches. On the other hand, in the regularization approaches, changes tend to progressively reduce batch by batch, and most of the chances occur in the top layers. While in CWR the $\bar{\Theta}$ freeze is well evident, in AR1 intermediate layers weights are moved (more than in EWC and SI) without negative impact in term of forgetting.

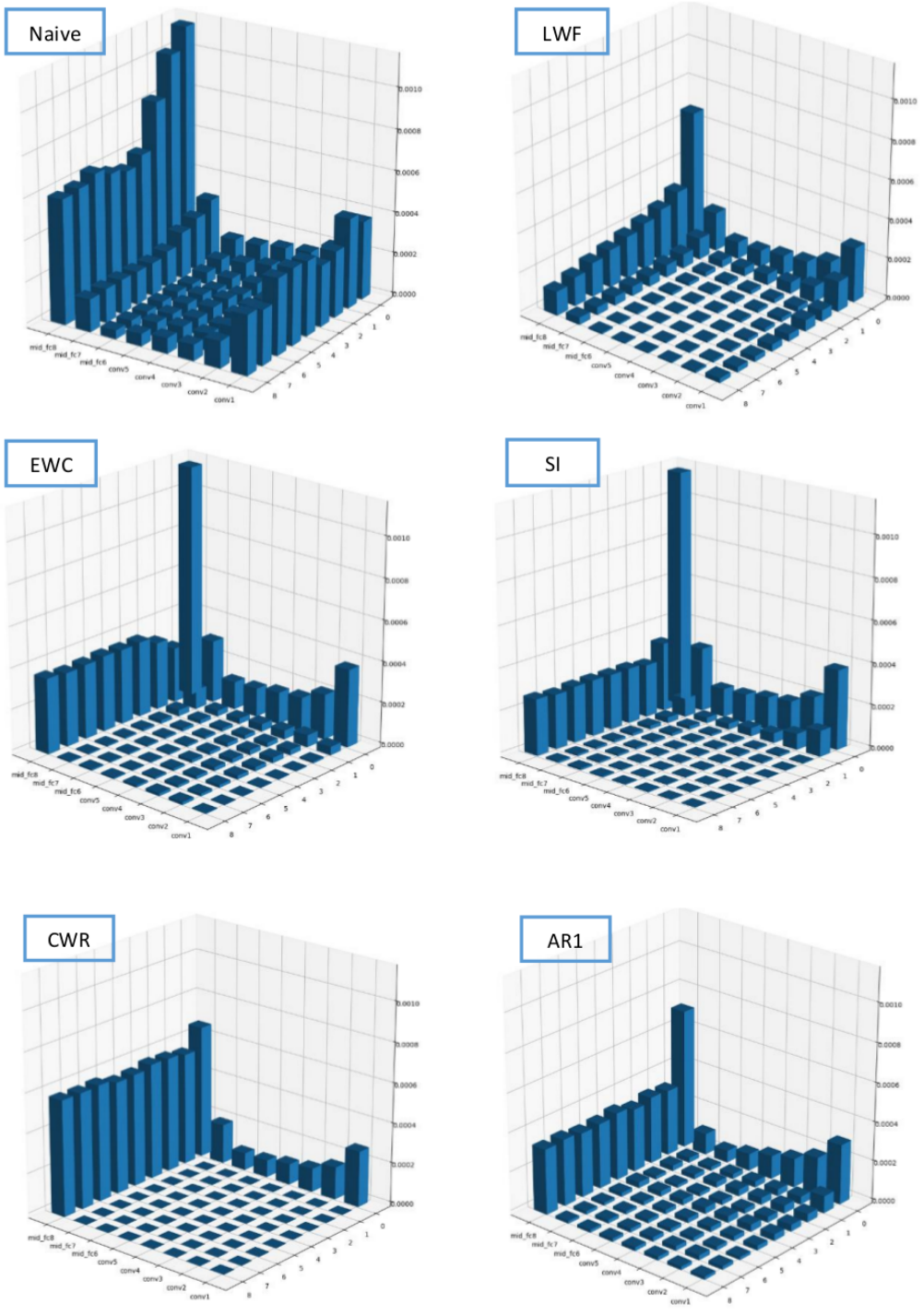


FIGURE 4.19: Amount of weight changes by layer and training batch in CaffeNet for different approaches.

5

Experimental Evaluation

> *"The lifelong learning paradigm [...] is an exciting and promising attempt to confront the issue of scaling up machine learning algorithms to more complex problems. Given the need for more accurate learning methods, it is difficult to imagine a future for machine learning that does not include this paradigm."*
>
> – Tom M. Mitchell, 1995

In this chapter we empirically assess the quality of the proposed strategies in various supervised, semi-supervised and reinforcement contexts. More specifically we run our evaluation considering different deep architectures on several continual learning benchmarks: *Seq-NORB*, *Seq-COIL-100*, *Seq-iCubWorld28*, *iCIFAR-100*, *CORe50* and *3D VizDoom Maze*. Comparing our results with other recently proposed regularization strategies for continual learning ensure the feasibility of our approaches, which, in same cases, exceed state-of-the-art performance by a good margin. In this chapter, for simplicity, we will focus only on the *Accuracy* metric for comparing performance among methods and the standard *Naive* and *Cumulative* baselines.

5.1 Continual Supervised Learning

While continual learning is better suit to work in the unsupervised and reinforcement domains in terms of practical applications, supervised learning constitute a straightforward starting point for evaluating CL approaches and help disentangle different complexity dimensions. Computer Vision, while not the only domain of application of current CL techniques, constitutes the easiest setting given the recent advances of deep learning in this area. Moreover, it is one of the best example of high-dimensional space problems which highly benefited from deep learning and hence where the impact of deep continual learning may be better appreciated.

The experiments reported in this section have been carried out with several deep architectures: LeNet7 [LeCun et al., 2004], CaffeNet [Jia et al., 2014], VGG-CNN-M [Chatfield et al., 2014], VGG-Face [Parkhi et al., 2015] and GoogLeNet [Szegedy et al., 2015]. Minor benchmark-specific

modifications that have been eventually carried out for each architecture will be defined within each specific evaluation section.

We will start our empirical investigation taking a look at more classic way of learning continually (in the simplest SIT-NI scenario) in order to highlight major limitations and motivate the need of novel CL approaches. The basic approaches we consider on the *BigBrother* and *iCubWorld28* are the following:

1. Training/tuning an ad hoc CNN architecture suitable for the problem from scratch.
2. Using an already trained CNN as a fixed feature extractor in conjunction with an incremental classifier.
3. Fine-tuning an already trained CNN.

5.1.1 BigBrother

The basic approaches illustrated above are instantiated as follows:

- *LeNet7*: consists of the classical "LeNet7" proposed by LeCun et al. [2004]. Its architecture is based on seven layers (much less than current state-of-the-art CNNs designed for large-scale datasets). However, it has been successfully applied to many object recognition datasets (NORB, COIL, CIFAR, etc.) with colorful or gray-scale images of size varying from 32×32 to 96×96, and is still competitive on low/medium scale problems.
- *CaffeNet + SVM*: In this strategy we employ a pre-trained CNN provided in the Caffe library [Jia et al., 2014] *"Model Zoo"*, BVLC Reference CaffeNet, which is based on the well-known *"AlexNet"* architecture proposed in [Krizhevsky et al., 2012] and trained on ImageNet [Russakovsky et al., 2015]. This model is used off-the-shelf to extract high-level features from the second-last hidden layer following the strategy proposed in Razavian et al. [2014]. Then a linear and incremental SVM[1] is used (instead of the native softmax output layer) for the final classification.
- *CaffeNet + FT*: Even in this case the BVLC Reference CaffeNet is employed. However, instead of using it as a fixed feature extractor the network is fine-tuned over the training batches. Even if for fine-tuning it is generally recommended to diversify the learning rate of the last layer (which is re-initialized to suit the novel number of output neurons) from the others, we found no significant difference during our exploratory analysis and therefore we kept the hyper-parametrization as homogeneous as possible.
- *VGG-Face + SVM*: identical to CaffeNet + SVM with exception of the pre-trained model used. The VGG-Face is a very deep architecture (16-levels) that has been trained directly on a very large dataset of faces (2,622 Subjects and 2.6 M images) [Parkhi et al., 2015].
- VGG-Face + FT. identical to CaffeNet + SVM but using the VGG-Face pre-trained model.

[1] We used incremental SVM from LibLinear implementation [Tsai et al., 2014].

For all these strategies, we trained the models until full convergence on the first batch of data and tuned them on the successive incremental batches, trying to balance the trade-off between accuracy gain and forgetting. This protocol fits the requirements of many real-world applications where a reasonable initial accuracy is demanded and the first batch is large enough to reach that accuracy. To control forgetting during the incremental learning phase we relied on *early stopping* and for each batch a fixed number of iterations were performed depending on the specific strategy. For example, for the LeNet7, trained with stochastic gradient descent (SGD), we chose a learning rate of 0.01, a mini-batch size of 100 and a number of iterations of 50 for all the eight incremental batches.

For the set_B of the BigBrother dataset, in order to make reproducible and comparable results, we decided to keep fixed (i.e., no shuffling) the order of the 54 updating batches as contained in the original dataset. In Figure 5.2, accuracy results are reported for each of the 5 aforementioned strategy instantiations.

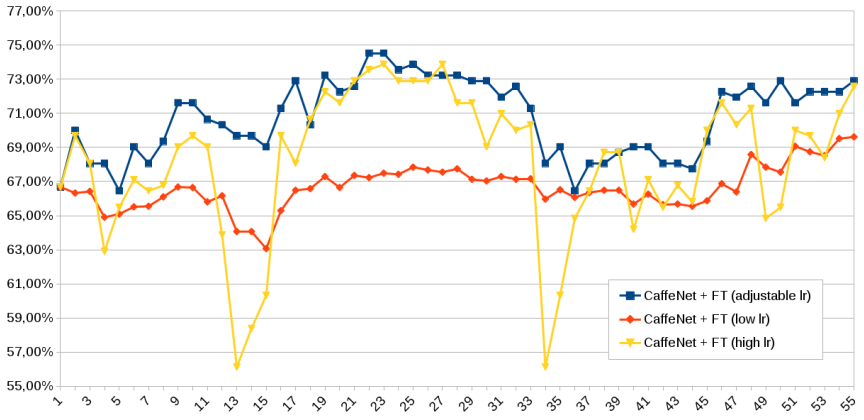


FIGURE 5.1: Accuracy results of different parameterizations for the CaffeNet + FT strategy: an example of the impact of the learning rate on the BigBrother dataset, in our continual learning scenario (54 batches). Better viewed in color.

It is worth pointing out that in this case the 54 incremental batches used for updating the model have a very high variance in terms of number of patterns they contain: in particular, it can vary from few dozens to many hundreds. This is typical in many real-world systems, where the hypothesis of collecting uniform and equally sized batches is often unrealistic. Controlling forgetting is here more complex than for the Seq-iCubWorld28 dataset (see following section). In fact, in this case, due to the aforementioned high variation in the number of patterns in the different incremental batches, we found that adapting the learning strength[2] to the batch size can lead to relevant improvements.

In Figure 5.1, an exemplifying parameterization for the CaffeNet + FT strategy is reported where we compare the learning trend by using `i)` a low learning rate, `ii)` a high learning rate, `iii)` an adjustable learning rate depending on the size of the batch. Results show that using an adjustable learning rate leads to better results. Therefore, in the rest of the experiments on the BigBrother dataset, an adjustable learning rate[3] is used.

[2] In terms of number of iterations and/or learning rate.

[3] we used a simple thresholding approach where the learning rate was varied among three fixed values, since in these experiments we did not found any significant difference using a continuous approach.

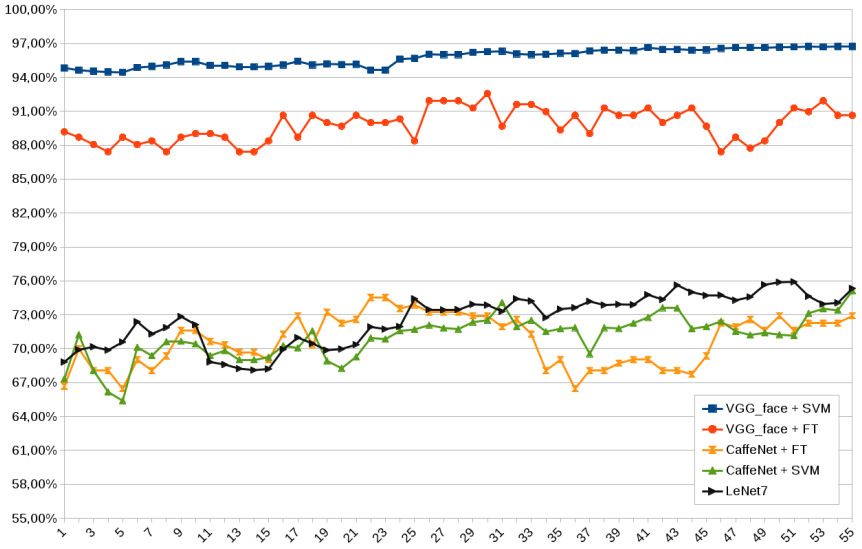


FIGURE 5.2: BigBrother dataset (Set_B): accuracy of the different strategies during incremental training (54 batches). Better viewed in color.

In Figure 5.2 the accuracy on the BigBrother dataset for all the strategies is reported. For this benchmark we note that:

- *LeNet7* model performs slightly better than *CaffeNet + SVM* or *CaffeNet + FT*. This is probably because of the high peculiar features (and invariance) requested for face recognition. Hence, learning the features from scratch for this dataset seems more appropriate than adapting general features by fine-tuning.
- The previous observation is corroborated by the really good performance of the VGG-Face + SVM and VGG-Face + FT strategies. In fact, since VGG-Face features have been learned in a face recognition task by using a dataset containing millions of faces, they are pretty effective for a transfer learning in the same domain.
- Since the features are already optimal, VGG-Face + SVM seems to be the best choice both for the accuracy and the stability. It reaches an accuracy of 96,73% that is 24,1% better than accuracy reported in [Franco et al., 2010] for the same dataset (in the supervised learning scenario).

5.1.2 Seq-iCubWorld28

The experiments here reported follows the same protocol used in the previous section with the idea of understanding and generalizing the main issues related to continual learning over different application contexts. In this case, we exclude the two strategies with the *Face-VGG* model since not inherent to the object recognition domain and we validate the remaining ones on 10 runs where we randomly shuffle the position of the training batches $B_2, \ldots, B_9$.

To better understand the efficacy of the *Naive* continual learning strategy variations and to quantify the impact of forgetting, we also tested each model on the corresponding *cumulative*

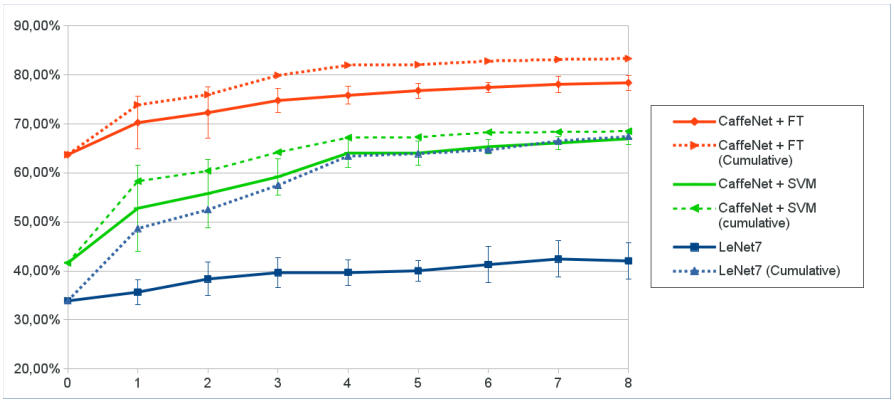


FIGURE 5.3: Average accuracy during incremental training on Seq-IcubWorld28 (9 batches). The bars indicate the standard deviation of the ten runs performed for each strategy. The dotted lines denote the cumulative strategies.

strategy. In Figure 5.3, the average accuracy over the ten runs is reported for each strategy. We note that:

- The CaffeNet + SVM has a quite good recognition rate increment along the 9 batches, moving from an accuracy of 41,63% to 66,97%. The standard deviation is initially higher with respect to the other strategies, but it rapidly decreases as new batches of data are available and the SVM model is updated. Furthermore, the small gap with respect to its cumulative counterpart proves that a fixed features extractor favors stability and reduces forgetting.
- The CaffeNet + FT is the most effective strategy in this case. This is probably because the features originally learned on the ImageNet dataset are very general and the iCubWorld28 dataset can be thought as a specific sub-domain where feature fine-tuning can help pattern classification. Moreover, even if splitting the dataset in 9 batches makes the task harder, we managed to achieve an averaged accuracy of 78.40% that outperforms previously proposed methods on the same datasets [Franco et al., 2010]. Even if in this case the gap with respect to the cumulative approach is slightly higher, the proper adjustment of early stopping and learning rate during the incremental phase allows to effectively control forgetting.
- The LeNet7 on this dataset is probably not able to learn (being the number of patterns too limited) complex invariant features that are necessary to deal with the multi-axes rotations, partial occlusions and the complex backgrounds which characterize this problem. The gap with respect to the cumulative approach is here high. This is in line with previous studies [Goodfellow et al., 2013; Mermillod et al., 2013] showing that smaller models without pre-training are much more susceptible to forgetting.

5.1.3 iCIFAR100

CIFAR-100 [Krizhevsky, 2009] is a well-known and largely used dataset for small (32×32) natural image classification. It includes 100 classes containing 600 images each (500 training + 100 test). The default classification benchmark can be translated into a SIT-NC scenario (denoted

as iCIFAR-100 by Rebuffi et al. [2017]) by splitting the 100 classes in groups. In this section we consider groups of 10 classes thus obtaining 10 incremental batches.

The CNN model used for this experiment is the same used by Zenke et al. [2017] for experiments on CIFAR-10/100 Split and whose results have been reported in Figure 4.2 for the MT scenario. It consist of 4 convolutional + 2 fully connected layers; details are available in Appendix A of [Zenke et al., 2017]. The model was pre-trained on CIFAR-10 [Krizhevsky, 2009].

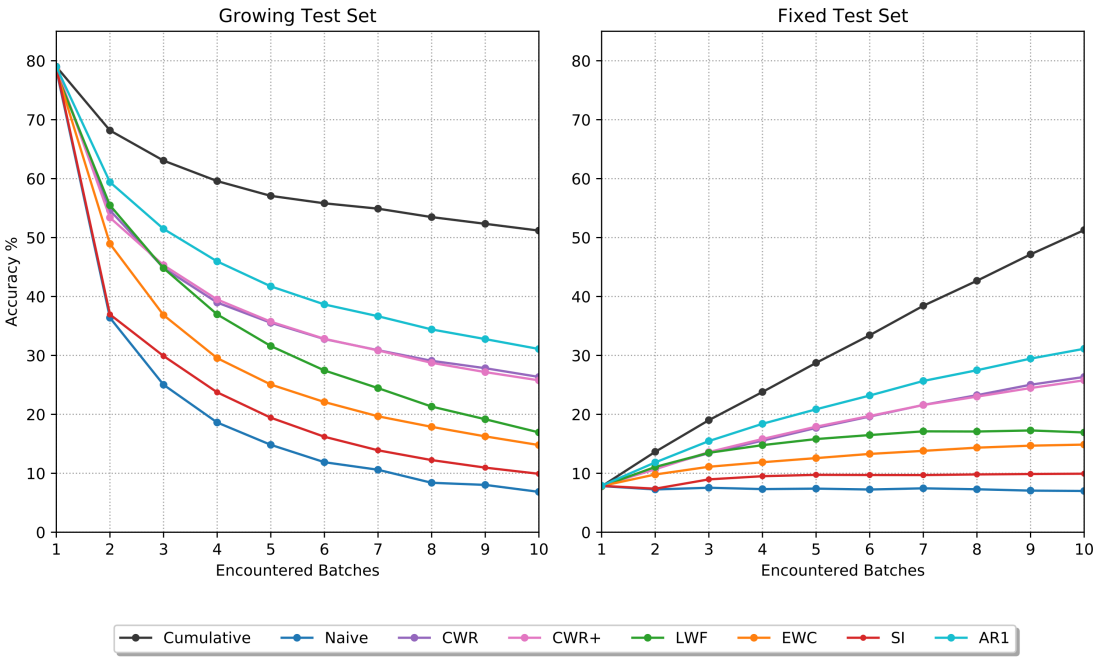


FIGURE 5.4: Accuracy on iCIFAR-100 with 10 batches (10 classes per batch). Results are averaged on 10 runs: for all the strategies hyperparameters have been tuned on run 1 and kept fixed in the other runs. The experiment on the right, consistently with CORe50 test protocol, considers a fixed test set including all the 100 classes, while on the left we include in the test set only the classes encountered so far (analogously to results reported by Rebuffi et al. [2017]). Better viewed in color.

Figure 5.4 compares the accuracy of the different approaches on iCIFAR-100. in particular:

- Unlike the **Naïve** approach, **LWF** and **EWC** provide some robustness against forgetting, even if in this incremental scenario their performance is not satisfactory. **SI**, when used in isolation, is quite unstable and performs worse than LWF and EWC.
- The accuracy improvement of **CWR+** over **CWR** is here very small, because the batches are balanced (so weight normalization is not required) and the CNN initialization for the last level weights was already very close to 0 (we used the authors' default setting of a Gaussian with std = 0.005).
- **AR1** consistently outperform all the other approaches.

It is worth noting that both the experiments reported in Figure 5.4 (i.e., fixed and expanding test set) lead to the same conclusions in terms of relative ranking among approaches, however,

we believe (as motivated in Chapter 4) that a fixed test set allows to better appreciate the continual learning trend and its peculiarities (saturation, forgetting, etc.) because the classification complexity (which is proportional to the number of classes) remains constant across the batches. For example, in the right graph it can be noted that SI, EWC and LWF learning capacity tend to saturate after 6-7 batches while CWR, CWR+ and AR1 continue to grow; the same information is not evident on the lest because of the underlying negative trend due to the increasing problem complexity.

Finally note that absolute accuracy on iCIFAR-100 cannot be directly compared with [Rebuffi et al., 2017] because the CNN model used in [Rebuffi et al., 2017] is a ResNet-32, which is much more accurate than the model here used: on the full training set the model here used achieves about 51% accuracy while ResNet-32 about 68,1% [Pan, 2018].

5.1.4 CORe50

CORe50 constitutes a major leap in complexity with respect to the commonly adopted benchmarks for continual learning in a supervised setting. Indeed, 128×128 RGB images with substantial variation in terms of environmental condition, 50 different objects and three settings with NI, NC and NIC updated content types may pose some difficulties to many current continual learning algorithms.

In the first set of experiments we reported and which can be considered as basic baselines for the following set of experiments (Only the *Naive*, *Cumulative* and *CWR* strategies are considered), we will use the same Mid-CaffeNet and Mid-VGG used for the *"static"* experiments reported in Chapter 4 for evaluating the global complexity of the CORe50 benchmark.

For all the continual learning UCT (NI, NC, NIC) we use the same test set composed of sessions #3, #7 and #10. The remaining 8 sessions are split in batches and provided sequentially during training. Since the batch order can affect the final result, we compute the average over 10 runs where the batches are randomly shuffled. Moreover, for each UCT, we provide the accuracy of the cumulative strategy (i.e., the current batch and the entire previous ones are used for training) as a target [4]. We do not use the term upper bound because in principle a smart sequential training approach could outperform a baseline cumulative training. In the following sections we report results only for *object level* classification task (the most difficult one), since both experiments lead to the analogous conclusions. Furthermore, in this study the models were trained on RGB data only (no depth information).

NI: New Instances In this setting the training batches coincides with the 8 sessions available in the training set. In fact, since each session includes a sequence (about 300 frames) for each of the 50 objects, training a model on the first session ad tuning it 7 times (on the remaining 7 sessions) is in line with NI scenario: all the classes are known since from the first batch and successive batches provide new instances of these classes to refine and consolidate knowledge. In Figure 5.5, we compare the baselines standard *Naïve* and *Cumulative* approaches in this setting.

[4] To reduce computations, the number of runs for the NC and NIC cumulative strategy is reduced to 5 and 3 respectively.

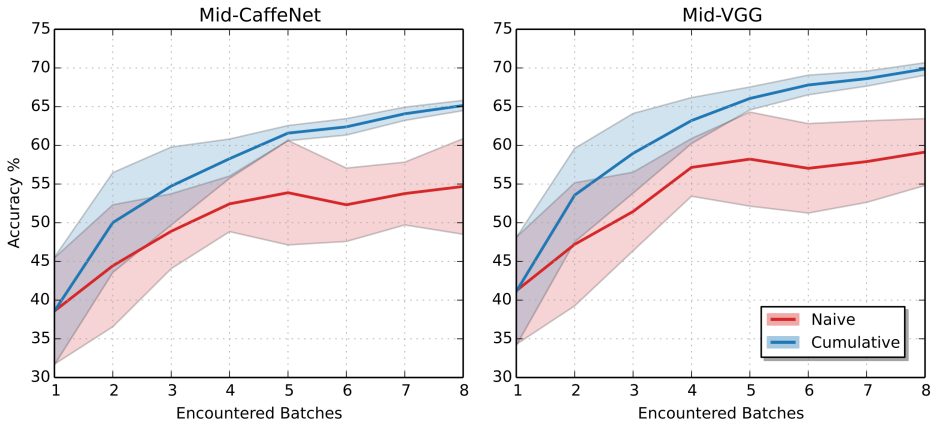


FIGURE 5.5: Accuracy results (averaged on 10 runs) for the naïve and cumulative strategies for Mid-CaffeNet and Mid-VGG. Colored areas represent the standard deviation of each curve. Tabular data available at http://vlomonaco.github.io/core50.

The accuracy gap between the naïve and the cumulative approach here is quite modest. In fact, with a careful tuning of the learning rate and number of iterations (early stopping), forgetting can be tamed in this scenario where the model memory is regularly refreshed with new poses (scale, view angle, occlusion, lighting, etc.).

NC: New Classes In this setting, for each sequential batch, new objects (i.e. classes) to recognize are presented. Each batch contains the whole training sequences (8) of a small group of classes, and therefore no memory refresh is possible across batches. In the first batch we include 10 classes, while the remaining 8 batches contain 5 classes each. To slightly simplify the task, in each of the ten runs we randomly chose the classes with a biased policy which privileges maximal categorical representation (i.e., spreading of objects of the same category in different batches). A fixed and global test set is used for the evaluation.

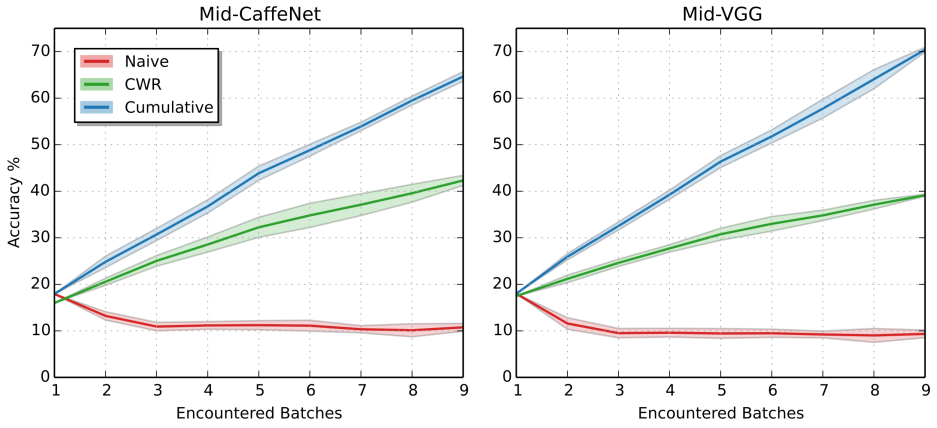


FIGURE 5.6: Mid-Caffe and Mid-VGG accuracy on NC scenario (average over 10 runs). Cumulative and Naïve approach are full depth models, while *CWR* lacks *fc6* and *fc7*. Colored areas represent the standard deviation of each curve. Tabular data available at http://vlomonaco.github.io/core50.

The naive approach in this scenario is not working at all. Graphs in Figure 5.6 show that the models completely forget the old tasks while learning the new classes: the initial accuracy drop

is due to the larger size of the first batch w.r.t. the following ones. So we investigated other approaches and find out one (denoted as *CWR*: *CopyWeights with Re-init*) that, in spite of its simplicity, performs fairly well and can be used as baseline for further studies (see Figure 5.6).

Other simple variations of CWR have been implemented and tested such as: `i`) *FW* where **cw** is not used and we just freeze in **tw** the class weights of the already encountered classes; `ii`) *CW* which is the same of *CWR* but without the re-init step; however as shown in Figure 5.7 the results obtained are significantly worse than *CWR*.

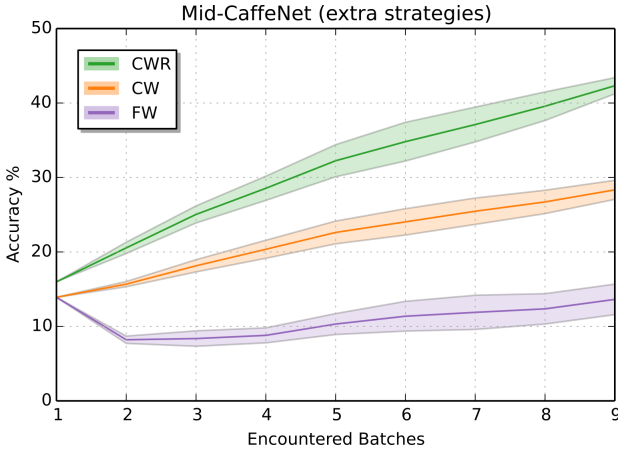


FIGURE 5.7: CWR compared with some variants: *FW* and *CW*. Colored areas represent the standard deviation of each curve.

NIC: New Instances and Classes In the third and last scenario, both new classes and instances are presented in each training batch. This scenario is the closest to many real-world applications where an agent continuously learns new objects and refines the knowledge about previously discovered ones. As for NC scenario the first batch includes 10 classes, and the subsequent batches 5 classes each. However, only one training sequence per class is here included in a batch, thus resulting in a double partitioning scheme (i.e., classes and sequences). The total number of batches is 79. We maximized the categorical representation in the first batch but we left the composition and order of the 78 subsequent batches completely random. The test set is the same as in NI and NC scenarios.

CWR approach for this scenario needs to be slightly adjusted. The first time a new class is encountered its **tw** weights are copied on **cw**, while at successive steps **cw** is updated as a weighted average. More precisely, for each class i in the current batch:

$$cw[i] = \begin{cases} tw[i] & \text{if } updated[i] = 0 \\ \dfrac{cw[i] \cdot udpdates[i] + tw[i]}{updates[i] + 1} & \text{otherwise} \end{cases}$$

where $update[i]$ is the number of times class i has been encountered so far. Figure 5.8 reports the accuracy of the naïve, cumulative and *CWR* approaches. The graph clearly shows that this scenario is very difficult (*CWR* accuracy is about half of the Cumulative approach accuracy) and there is a big room for improvements.

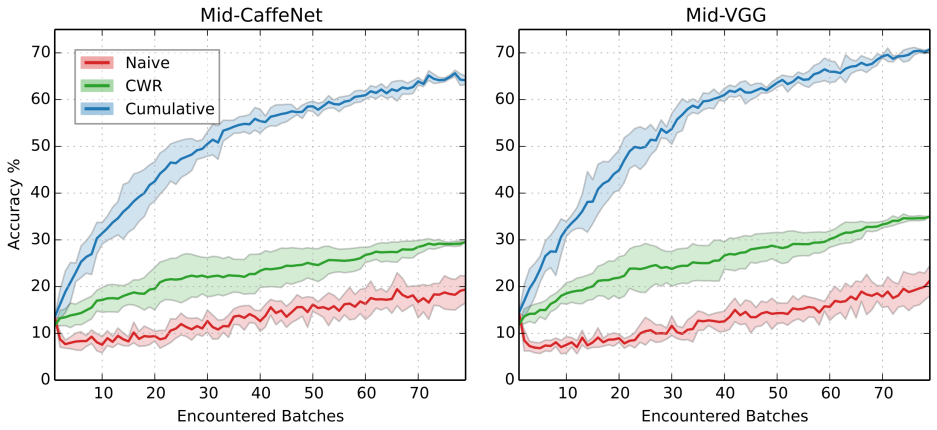


FIGURE 5.8: Mid-Caffe and Mid-VGG accuracy on NIC scenario (average over 10 runs). Colored areas represent the standard deviation of each curve. Tabular data available at `http://vlomonaco.github.io/core50`.

Improving results with CWR+ and AR1 In CWR experiments reported above, in order to better disentangle class-specific weights we used models without class-shared fully connected layers (e.g., we removed FC6 and FC7 in CaffeNet). In fact, since $\bar{\Theta}$ weights are frozen after the first batch, fully connected layer weights tend to specialize on the first batch classes only. However, since skipping fully connected layers is not mandatory for CWR, in this section to better compare different approaches we prefer to change the native architectures as little as possible and keep fully connected layers whether they exist. In particular, we use the following models:

- CaffeNet and GoogLeNet original models work on input images of size 227×227 and 224×224 respectively. CORe50 images are 128×128 and stretching them to 227×227 or 224×224 is something that should be avoided (would increase storage and amount of computation). On the other hand, as discussed in appendix C, simply reshaping the network input size leads to a relevant accuracy drop. This is mainly due to the reduced resolution of feature maps whose size along the hierarchy is about half the original. We noted that the accuracy can be restored by halving the stride in the first convolutional layer and by adjusting padding if necessary: unfortunately, this also restore much of the original computational complexity, but `i)` does not require unnecessary image stretching, `ii)` allows to save memory, and `iii)` reduces CPU→GPU data transfer when loading mini-batches.

- For CaffeNet, the number of neurons in *fc6* and *fc7* fully connected layers was halved. This substantially reduce the number of parameters without any relevant accuracy drop.

- GoogLeNet has three output layers. The deepest one is typically used for prediction, while the two intermediate ones are useful to boost gradient flow during training thank to the injection of a fresh supervised signals at intermediate depth. While the deepest output level is proceeded by a global average pooling, each intermediate output layer is preceded by a fully connected layer; in our experiment where GoogLeNet was always initialized from ImageNet such fully connected layers did not provide any advantage, hence to simplify the architecture and reduce the number of parameters we removed them. Finally, note that

when GoogLeNet is used with CWR, CWR+ and AR1 we need to maintain in memory a copy of the weights of all the three output levels.

Table D.1 in Appendix D.2 lists all the changes between original and modified models. Model weights (in convolutional layers) have been always initialized from ImageNet pre-training. We also tried other popular CNN architectures on CORe50, including NiN [Lin et al., 2014] and ResNet-50 [He et al., 2016]. Table 5.1 compares the accuracy of these models when trained on the whole training set (i.e., all training batches combined). The gap between GoogLeNet and ResNet50 is quite small, but the latter is much slower to train, so we decided to use GoogLeNet as a near state-of-the-art model.

TABLE 5.1: Model training on CORe50 by using the whole training set (fusion of all training batches). Models are adapted to work with 128x128 inputs and weights in convolutional layers are initialized from ImageNet pre-training. Time refers to a single Titan X Pascal GPU and Caffe framework.

Model (128x128)	**Test accuracy**	**Mini-batch size**	**# Epochs**	**Time (m)**
CaffeNet	74.1%	256	4	7
NiN	82.3%	256	4	14
GoogLeNet	91.3%	64	4	30
ResNet-50	92.9%	12	4	120

CORe50 dataset has been specifically designed as a benchmark for continual learning and object recognition. Here we consider NC content update type (a.k.a. incremental-class learning) where the 50 classes are partitioned in 9 batches provided sequentially: B_1 includes 10 classes while $B_2, \ldots, B_9$ 5 classes each. For each class 2400 patterns (300 frames $\times$ 9 sessions) are included in the training batches and 900 pattern (300 frames $\times$ 3 sessions) are segregated in the test set. Here again, the test set is fixed and the accuracy is evaluated after each batch on the whole test set, including still unseen classes.

LWF, EWC and SI were run with the modifications (variable lambda, normalization, clipping, etc.) introduced in Chapter 3. In fact, when the approaches were tested in their native form either we were not able to make them consistently learn across the batches or to contrast forgetting enough.

Figure 5.9 shows CaffeNet and GoogLeNet accuracy in CORe50, SIT - NC scenario. Accuracy is the average over 10 runs with different batch ordering. For all the approaches hyperparameter tuning was performed on run 1, and then hyperparameters were fixed across runs $2, \ldots, 10$. Table D.3 in Appendix D.3 shows hyperparameter values for all the methods. In the Appendix D.5, the standard deviation bars are also reported to show methods stability. From these results we can observe that:

- The effect of catastrophic forgetting is well evident for the **Naïve** approach: accuracy start from 17-20% at the end of B_1 (in line with the 20% of classes in B_1), but then suddenly drops to about 9-10% (that is the proportion of classes in each subsequent batch); in other words, as expected, after each batch, the model tends to completely forget previous classes while learning the new ones.

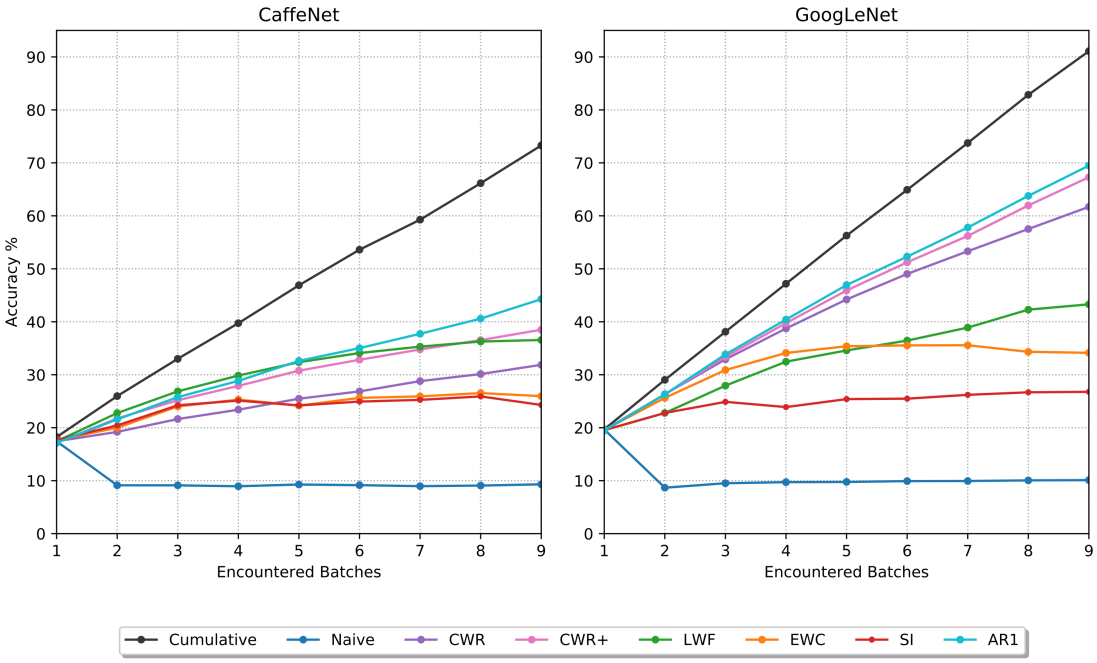


FIGURE 5.9: The graphs show CaffeNet and GoogLeNet accuracy on CORe50 after each training batch (average on 10 runs, with different class ordering). Better viewed in color.

- **LWF** behaves well for CaffeNet and moderately well for GoogLeNet: continuous learning is evident and accuracy is much better than Naive approach. A few percentage point drop can be noted in the last batches for GoogLeNet; to avoid this we tried to boost stability by increasing the lambda value: this yielded to an increasing learning trend across all the batches but the top accuracy was never higher than 32% (neither in the central batches nor in the last ones), so we decided to rollback to previous parametrization.

- Both the models are able to learn incrementally with **EWC**, but while for GoogLeNet the learning trend is quite good, CaffeNet after a few batches tends to stabilize and the final accuracy is much lower. Since GoogLeNet is a deeper and more powerful network we can expect a certain accuracy gap (see the corresponding cumulative approach curves); however here the gap is much higher than for LWF and in our experiment we noted that EWC (and SI) tend to suffer more than LWF the presence of fully connected layers such as FC6 and FC7 layer in CaffeNet[5]. Fully connected layers are usually close to the final classification layer, and any change in their weights is likely to have a major impact on classification. Even if EWC can operate on all layers, precisely constraining the weights of fully connected layers appears to be challenging due to their high importance for all tasks. LWF, whose regularization signal "comes from the top", seems to better deal with fully connected layers. To further investigate this issue some experiments have been performed by removing FC6 and FC7 from CaffeNet and, in spite of the shallower and less powerful network, for EWC we got a few percentage improvements while LWF accuracy slightly decreased.

[5]GoogleNet, as many modern network architectures, does not include fully connected layers before the last classification layer.

- While CaffeNet accuracy with **SI** is very close to EWC, GoogLeNet accuracy with SI is markedly lower than EWC. In general, we noted that SI is less stable than EWC and we believe that SI weights importance estimation can be sometimes less accurate than the EWC one because of the following reasons:
  - A weight which is not moved by SGD is not considered important for a task by SI, but this is not always true. Let us assume that Batch B_1 trains the model on classes $c_1, c_2, \ldots, c_{10}$; the output layer weights which are not connected to the above class output neurons, are probably left unchanged to their (near 0) initial value; however, this does not mean that they are not important for B_1 because if we raise their values the classification might dramatically change. More in general, if a weight already has the right value for a task and is not moved by SGD, concluding that it is not important for the task can be sometime incorrect.
  - A weight could cover a long trajectory and, at the end of the training on a batch, assume a value similar to the initial one (closed trajectory). Such situation can happen because the loss surface is highly non convex and gradient descent could increase a weight while entering a local minimum and successively restoring its value once escaped.
- **CWR** and **CWR+** accuracy is quite good, always better than LWF, EWC, SI on GoogLeNet, and better than EWC and SI on CaffeNet. Continuous learning trend is here nearly linear across batches, with no evident saturation effect. The relevant improvement of CWR+ over CWR is mainly due to zero init.
- **AR1** exhibits the best performance. SI regularization is here quite stable and pushes up CWR+ accuracy. AR1 is also stable w.r.t. parameter tuning: in Table D.3 you can observe that we used almost the same hyperparameters for CaffeNet and GoogLeNet. For GoggleNet AR1 reaches a remarkable accuracy of about 70% with very small standard deviation among runs (see Appendix D.6), and the gap w.r.t. the Cumulative approach is not large, thus proving that continuous learning (without storing/reusing old patterns) is feasible in a complex incremental class scenario.

5.1.5 Conclusions

In this chapter, we evaluated different continual learning strategies in a single-incremental-task and fully supervised scenario. The first set of experiments on *BigBrother* and *iCubWorld* in the simpler NI setting already shown the difficulty encountered by deep architecture trained continually. In particular we noted that:

- Forgetting can be a very detrimental issue: hence, when possible (i.e., transfer learning from the same domain), it is preferable to use CNN as a fixed feature extractor to feed an incremental classifier. In general, this results in better stability and often in improved efficiency (i.e., tuning all CNN layers can be computationally expensive).

- If the features are not optimized (transfer learning from a different domain), the tuning of low level layers may be preferable and the learning strength (i.e., learning rate, number of iteration, etc.) can be used to tame forgetting.
- Training a CNN from scratch can be advantageous if the problem patterns (and feature invariances) are highly specific and a sufficient number of samples are available.

Unfortunately, these basic observation do not hold in more complex scenario where an automatic "knowledge refresh" is not implicitly imposed by the nature of the task (e.g. the NC setting). This is why, for better performance and general applicability a number of more advanced CL strategies are needed.

In this section, the CWR, CWR+ and AR1 strategies have been evaluated on the *iCIFAR-100* and the *CORe50* benchmark in the SIT scenario. Early results on these benchmarks prove that AR1 allows to train sequentially complex models such as CaffeNet and GoogLeNet by limiting the detrimental effects of catastrophic forgetting. AR1 accuracy was higher than existing regularization approaches such as LWF, EWC and SI. While we did not explicitly consider rehearsal techniques in our comparison sessions, from preliminary results AR1 was also competitive with iCARL on CORe50 [Lomonaco and Maltoni, 2018]. AR1 overhead in terms of storage is very limited and most of the extra computation is based on information made available by stochastic gradient descent. We showed that early stopping SGD after very few epochs (e.g., 2) is sufficient to incrementally learn new data on CORe50. Further ideas could be investigated in the future to quantify weight importance for old tasks such as exploiting the moving average of squared gradients already considered by methods like RMSprop [Hinton, 2012] or Adam [Kingma and Ba, 2014] or the Hebbian-like reinforcements between active neurons recently proposed by Aljundi et al. [2017]. Class-incremental learning (NC update content type) is only one of the cases of interest in SIT. New instances and classes (NIC) update content type, available under CORe50, is a more realistic scenario for real applications, and would constitute the main target of our future research. AR1 extension to unsupervised (or semi-supervised) implementations, such as those described in Section 3.6.1 and [Parisi et al., 2018b], is another scenario of interest for future studies. In particular, Parisi et al. [2018b] propose an interesting 2-level self-organizing model built on the top of a convolutional feature extractor that is capable of exploiting temporal coherence in CORe50 videos and provides good results also with weak supervision. Although much more validations in complex setting and new better approaches will be necessary, based on these preliminary results we can optimistically envision a new generation of systems and applications that once deployed can continue to acquire new skills and knowledge without the need of being retrained from scratch.

5.2 Continual Reinforcement Learning

One of the greatest barrier for autonomous learning systems is their inability to learn without explicit supervision. While much of the success of Deep Learning has been due to supervised training over big and representative labeled datasets, Reinforcement Learning constitute a more

biologically sound learning paradigm for approximating the supervision from sparse rewards via trial and errors.

Reinforcement Learning, is actually quite interconnected with the idea of learning continually, since its interaction with the external world is unfolding through time. Learning instability in Reinforcement Learning, indeed, is not only due to a weak and approximate supervised signal but also from the online nature of its learning which is prone to forgetting. The problem has been tackled implicitly by the reinforcement learning community with *memory replay* [Mnih et al., 2013] or *multi-agent* [Mnih et al., 2016] training procedures which allow the agent to *"remember"* past knowledge and skills. However, as we have seen in the previous chapter, the idea of simply rehearsing past knowledge through and external buffer or multiplying the number of agent may not be the best scalable solution, especially considering very high-dimensional problems in ever-changing environments.

Very recently the *Proximal Policy Optimization* (PPO) [Schulman et al., 2017], which perform comparably or better than state-of-the-art RL approaches while being much simpler to implement and tune on many tasks, is another example on how CL techniques may be beneficial for reinforcement learning. Indeed, it employs a regularization term based on the Fisher information which may be as well seen as a regularization strategy like EWC.

Nevertheless, very little has been done in the context of pure continual reinforcement learning settings, with multiple tasks or in complex ever-changing environments with recent deep learning architectures. [Kirkpatrick et al., 2017] assessed EWC on a sequence of Atari games, and its latest evolution on simple 3D navigation environment (and only tested for *forward transfer*) [Schwarz et al., 2018].

In the following paragraph we will summarize some preliminary results on the *VizDOOM Maze*, one of the first end-to-end continual reinforcement learning experiments on complex 3D ever-changing environments and without exploiting any kind of task supervised signal t.

5.2.1 3D VizDOOM Maze

While the general focus of continual reinforcement learning research has been devoted to Multi-Task scenarios, as presented in chapter 3, *3D VizDOOM Maze* constitutes a step forward the evaluation of new continual learning strategies that have to deal with unpredictable changes in the environment within the same task. Since the notion of task may be ambiguous in general, in the experiments below we refer to a single task following the formal framework presented in Chapter 2 and meaning that the label t is always the same, hence, not providing additional information to the model for customizing its behaviors.

Intuitively, we can hence consider the experiments as designed for assessing the performance of the model in achieving a single task (i.e. object picking/avoiding) in an non-stationary complex 3D environment. It is worth noting that, even the *"end-of-task"* supervised signal is not provided to the model, meaning that, the agent is neither made aware explicitly during training that the environment distribution has changed. We regard at this situation as the most realistic setting every agent should be able to deal with in real-world conditions.

In recent literature, this problem has been tackled by using an external unsupervised generative model in order to detect big changes in input space (hence the environment). In the following experiments, instead, we propose to detect the environmental changes just by looking at the ability of the agent to actually perform the task, namely the difference between the expected reward and the actual reward. This not only signals a possible change in the environment effecting the performance of the agent but also possible changes in the reward function or instability of the learning process which may be tamed through consolidation (like in PPO). Moreover, we do not use any task-specific parametrization nor any kind of memory replay.

We use a simple batched *A2C* with synchronous updates [Wu et al., 2017], but only *within* the same *MAP* (the actual maze with fixed settings), so that when the MAP changes the model cannot access in any way previous environmental conditions. The architecture of the agent used for these experiments is a plain 4-layers ConvNet with 3 output neurons (encoding the *turn-left*, *turn-right* and *move-forward* actions). On each of the considered environmental changes (i.e. *light*, *texture*, *object*, *all*) four different general approaches are assessed:

1. **Naive**: This approach, like the homonymous strategy in the supervised context, consist in just continuing back-propagation on the changes environment without any variation. This will be considered as a lower bound for the other strategies.

2. **Supervised**: The *supervised* approach, for now on, can be considered as a second baseline in which the "end-of-task" supervised signal is actually used for memory consolidation purposes.

3. **Static**: In this strategy, we consolidate memory at fixed step in time. As we will see in the experiments results, this may be very difficult to tune and rather inefficient, depending on the memory consolidation technique used. In fact, when learning from scratch an early and "blind" consolidation of memory may also hurt performance and actually hamper the ability of learning in the future.

4. **Unsupervised**: Recent evidences in behavioural experiments on rats [Clopath, 2012] suggests behavioral correlates of synaptic consolidation especially when the subject is exposed to novel or strong external stimulus (e.g. a foot shock). Following this inspiration, the central idea of this strategy is to consolidate memory only when a substantial difference between the expected reward and the actual one is detected, i.e. when the agent encounter an unexpected situation. This is practically implemented with the difference between a long-term and short-term moving average that, when exceeds a particular threshold, triggers the memory consolidation procedure. In Figure 5.10, an example in the *light* scenario or the short and long-term moving average (computed over 6 and 50 episodes, respectively) of the average cumulative reward during training is reported.

For simplicity, in these experiments, for each approach (except *Naive*) EWC basic implementation with one Fisher information for each consolidation step is used.

In Table 5.2 the average cumulative reward and the A metric results for the *Light* scenario and the 4 different approaches is reported. Performance are evaluated at the end of the training on

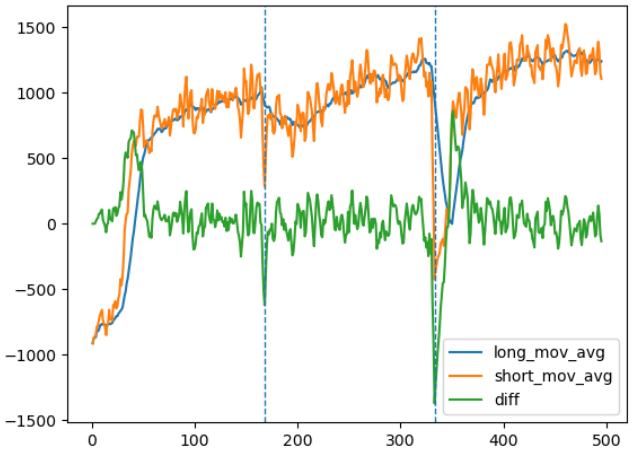


FIGURE 5.10: Short and long-term moving average (computed over 6 and 50 episodes, respectively) of the average cumulative reward during training in the *light* scenario. Dotted lines indicate when the environment is changed. In this example, the difference between the short-term and long-term moving average exceed 500 when the environment changes.

TABLE 5.2: Average cumulative reward and A metric result for the *Light* scenario and the 4 different approaches.

	Naive			**Supervised**			**Static**			**Unsupervised**		
	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
M_1	382	-785	-440	527	-712	-198	318	-845	-561	496	-664	-307
M_2	-969	677	-528	-188	704	621	-912	776	-5	-311	933	280
M_3	-915	1219	978	367	287	821	-683	134	817	-131	498	942
A		228,66			**419,66**			91,66			404,5	

each map M_i on 300 testing episodes, 100 for each different environmental condition M_i, even the ones not already encountered. The supervised approach of consolidating memory exactly when the task variation happen obtains the best A score of 419,66. The *static* approach, with a consolidation step every 100 training episodes, is not allowing the model to "remember" and reaching an average cumulative reward A of just 91,66. The *unsupervised* approach while not reaching, on average, the same performance as the supervised approach exploiting the *"end-of-task"* signal, is less than 15 reward points behind.

TABLE 5.3: Average cumulative reward and A metric result for the *Texture* scenario and the 4 different approaches.

	Naive			**Supervised**			**Static**			**Unsupervised**		
	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
M_1	829	459	-153	1240	977	69	1007	682	7	1019	767	39
M_2	-35	1142	235	136	1337	690	-35	1280	571	466	1195	710
M_3	-293	-288	1469	-171	-121	1325	-261	-188	1099	-221	-174	1092
A		470,66			**624,33**			483,66			562,83	

Table 5.3 reports the same evaluation protocol but on the *Texture* environment. Also in the

case the ranking among the strategies is the same, even though, it is worth noting that memory retention seems much easier than in the light scenario. This may be due to the ability of the model to consider the wall texture as irrelevant to the accomplishment of the task (as we would expect). However, there are situation in which this is not always true. For example, being able to recognize dead ends may help the model to avoid them ultimately improving its ability to collect more objects of interest (hence improving its cumulative reward).

TABLE 5.4: Average cumulative reward and A metric result for the *Object* scenario and the 4 different approaches.

	Naive			**Supervised**			**Static**			**Unsupervised**		
	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
M_1	754	-1009	-1005	1271	-980	-968	1149	-1031	-1003	1370	-1067	-990
M_2	-1032	193	-1013	552	326	-1075	-987	356	-933	615	358	-1023
M_3	-975	-230	819	-230	-469	890	-766	-225	821	-105	-511	925
A		-78,5			390			58			**442**	

Being able to generalize over time the concept of a *"column"* or a *"lantern"* in the *Object* scenario, instead, constitute an hard challenge for the agent. The visual features of the objects are the first elements that can help the agent learn where to go (i.e. right combination of turn left, turn-right and move forward actions) in order to maximize the cumulative reward. As the features of the objects of interest are changed the model is suddenly unable to solve the task as also inducible from the always negative *forward transfer* on the future environmental variations. In this case, as shown in Table 5.4, the model has also quite an hard time remembering past environmental condition, however, the *unsupervised* approach in this case is even better than the *supervised* approach. We argue that this may be related to an increased stability due to additional memory consolidation steps happening *within* the same map.

TABLE 5.5: Average cumulative reward and A metric result for the *All* scenario and the 4 different approaches.

	Naive			**Supervised**			**Static**			**Unsupervised**		
	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3	M_1	M_2	M_3
M_1	1066	-998	-1000	1419	-995	-1000	1337	-999	-1000	1601	-997	-1000
M_2	-855	295	-998	-253	155	-981	-1010	229	-1005	-452	236	-999
M_3	-597	-264	593	-569	-463	682	-334	-218	742	-450	-516	681
A		39,66			161,83			124,33			**183,33**	

The *All* scenario, as intuitively conceivable, constitutes the hardest challenge for an agent that learn continually. Table 5.5 summarize the results for the related experiments. Also in this case, the *unsupervised* approach perform significantly better than *Naive* and *Static* and slightly better than the *Supervised* approach.

5.2.2 Conclusions

The preliminary experiments above on four different scenarios in non-stationary environments show that an unsupervised approach without "end-of-task" signal, external models or task-specific parametrization is not only possible but may be competitive with respect to a standard

supervised approach.

However, a difficulty that may arise from such an approach, it that, in case of *positive forward transfer* followed by a possible *negative backward transfer* (i.e. being able to perform well on the new conditions but then possibly forget how to solve the task in the previous one) memory consolidation does not take place and the agent actually forget how to solve the task in previous conditions without noticing it by just looking at its cumulative reward curve. It is worth noting that this particular forgetting situation is not uncommon also for biological systems: for example, at the end of a math curriculum we may easily find ourselves to have forgotten even the most basic notions without ever noticing it.

This problem may be tackled by looking at an additional regularization loss for reconstructing the input image (or predicting the next one) since changes in the input space are more evident. In this way, while still using a single model and construct more robust features [Canziani and Culurciello, 2017; Hermann et al., 2017], we could integrate the benefit of both approaches when learning continuously.

5.3 Continual Unsupervised Learning

As in reinforcement learning, continual learning may be particularly interesting if used in conjunction with *unsupervised learning*. The idea of continually providing feedback to the model indeed is one of the main obstacle to the use and deployment of AI systems that learn continuously. A great example of application may be sequence learning problems like time series forecasting, anomaly detection or recommendation systems.

In the following sections we evaluate the simple *SST* technique on a object recognition task when a model is exposed to a continuous, temporal coherent stream of unlabeled data using the *Seq-Norb* and *Seq-COIL-100* benchmarks. Since this scenario is surely more difficult to achieve without any supervision of the ones mentioned before, in our experiments we relax the completely unsupervised assumptions starting from a pre-trained model and expecting to encounter images belonging to the same classes encountered during this initial training phase, making it essentially a *semi-supervised* scenario. A 5-layers *Hierarchical Temporal Memory* and a standard *LeNet7* will be used for the experiments presented in this section.

HTM *Hierarchical temporal memory* (*HTM*) [George and Hawkins, 2009] is a biologically inspired framework that can be framed into multistage Hubel-Wiesel architectures [Ranzato et al., 2007], a specific family of deep architectures. A brief overview of HTM is provided in Appendix A. A more comprehensive introduction can be found in [Maltoni, 2011a] and [Maltoni and Rehn, 2012].

Analogously to CNN, HTM hierarchical structure is composed of alternating feature extraction and feature pooling layers. However, in HTM feature pooling is more complex than typical sum or max pooling used in CNN, and the time is used since the first training steps, when HTM self-develops its internal memories, to form groups of feature detectors responding to temporally-close

inputs. This is conceptually similar to the unsupervised feature learning proposed in [Goroshin et al., 2015; Zou et al., 2011, 2012].

In the classical HTM approach [Maltoni, 2011a] once a network is trained, its structure is frozen, thus making further training (i.e., continual learning) quite critical. Maltoni and Rehn [2012] introduced a technique (called HSR) for HTM (incremental) supervised training based on gradient descent error minimization, where error back-propagation is implemented through native HTM message passing based on belief propagation. In this section HSR will be used for semi-supervised tuning.

The HTM architecture here adopted includes some modifications with respect to [Maltoni, 2011a; Maltoni and Rehn, 2012]. We experimentally verified that these modifications yield some accuracy improvements when working with natural images, and, at the same time, reduce the network complexity. Presenting these architectural changes in detail is not in the scope of this section, but some hints are given in the following:

- *Dilobe ordinal filters*: the feature extraction at the first level is not based on a variable number of self-learned templates as described in Maltoni [2011a], but is carried out with a bank of 50 dilobe ordinal filters Sun and Tan [2009]. Each filter, of size 8×8, is the sum of two 2d Gaussians (one positive and one negative) whose center, size and orientation are randomly generated (see Figure 5.11). Each filter computes a simple intensity relationship between two adjacent regions (i.e. the two filter lobes) which is quite robust with respect to light changes, and (in our experience) discriminant enough for low level feature extraction. Although one could setup an unsupervised approach to learn optimal filters from natural images, for simplicity in this work we generated them randomly and kept them fixed.
- *Partial receptive field.* in the classical HTM implementation the receptive field of a node is the union of the child node receptive fields, and is not possible for a node to focus only on a specific portion of its receptive field. In general, this does not allows to isolate objects from the background or to deal with partial occlusions. A simple but effective technique has been implemented to deal with partial receptive fields.

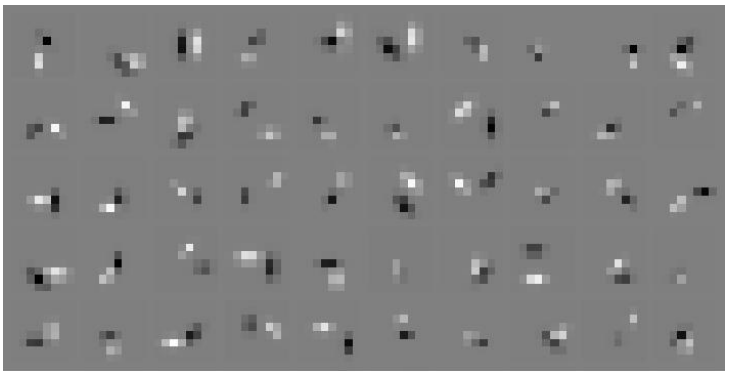


FIGURE 5.11: A graphical representation of the 50 "random" dilobe filters at level 1.

The HTM architecture used in our experiments has 5 levels:

- Input: 1024 nodes (32×32) connected to image pixels.
- Intermediate 1: 169 (13×13 nodes), each node has 8×8 child nodes.
- Intermediate 2: 169 (13×13 nodes), each node has 1 child node.
- Intermediate 3: 9 (3×3 nodes), each node has 5×5 child nodes.
- Output: 1 node with 3×3 child nodes.

Since intermediate level 2 and 3 include both feature extraction and feature pooling, and the input level in this case is not performing any particular processing, the above 5 levels correspond to 6 levels in a CNN (accidentally the same of LeNet7). Note that an HTM node is much more complex than a single artificial neuron in conventional NN, since it was conceived to emulate a cortical micro-circuit [George and Hawkins, 2009]. HTM accuracy on baseline NORB is reported as additional material in Appendix B.1.

The CNN architecture used in our experiments is an minor modification of "LeNet7" that was specifically designed by LeCun et al. [2004] to classify NORB images. This is still one of the best performing architecture on NORB benchmarks. We empirically proved that working on 32×32 images does not reduce accuracy with respect to the 96×96 original images. So our main modification concerns the reduced feature map size and filter size to deal with 32×32 monocular inputs (see Figure 5.12).

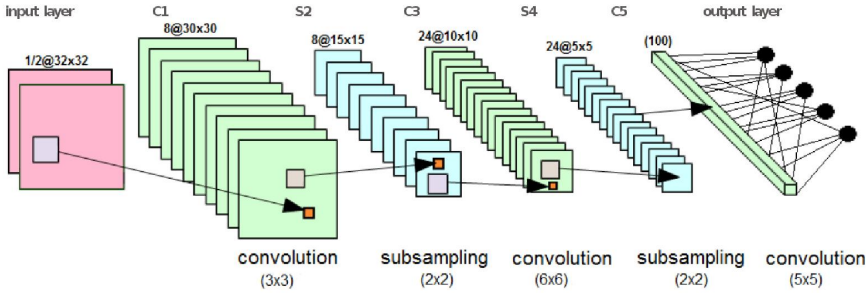


FIGURE 5.12: The CNN used in this work (original LeNet7 adapted to 32×32 inputs). X@Y×Y stands for X feature maps of size Y×Y; (Z×Z) stands for the filters of size Z×Z.

LeCun et al. [2004] suggested to train the LeNet7 with the squared error loss function, which naturally fits our semi-supervised tuning formulation. In our experiment on the standard NORB benchmark we evaluated some modifications to the architecture or the training procedure: *i*) Max pooling instead of the original sum pooling; *ii*) Soft-max + log-likelihood instead of squared error; *iii*) Dropout; but none of these changes (nowadays commonly used to train CNN on large datasets) led to consistently better accuracy, so we came back to the original version. Since we are not using any output normalization[6], the network output vector is not exactly a probability vector: however looking at the output values we noted that after a few training iterations they approximate quite well a probability vector: all elements in $[0, 1]$ and summing to 1. To be sure of the soundness of our CNN implementation and training procedure we tried to reproduce the results reported in [LeCun et al., 2004] for a LeNet7 trained on the full "normalized-uniform" dataset of 24300 patterns (4860 for each of the 5 classes). Since results in [LeCun et al., 2004] are reported only for the binocular case, for this control experiment we also used binocular patterns

[6] Any attempt to introduce a normalization (e.g. softmax) resulted in some accuracy loss.

(even if in the format 32×32). After some tuning of the training procedure, we achieved a classification error of 5.6% which is slightly better than the 6.6% reported in [LeCun et al., 2004], aligned with 5.6% of Ranzato et al. [2007] and not far from the state-of-the-art 3.9% reported in [Le et al., 2010]. More details on the CNN accuracy on baseline NORB (including a comparison with HTM) can be found in Appendix B.1.

5.3.1 Seq-NORB

In this section we evaluate the 4 different variation of the SST continual learning strategy variations (*SupT*, *SupTR*, *SST-B*, *SST-A*) introduced in Chapter 3 on the *Seq-NORB* and *Seq-COIL-100* benchmarks. In all the experiments:

- We used 32×32 monocular patterns (left eye only).
- We report classification accuracy as *frame based classification accuracy* (the *sequence based classification* scenario will be addressed in future studies) as defined in section 4.2.1.
- For semi-supervised learning, each training batch of 1,000 frames is treated as a single frame flow, without exploiting the regular sequence order and size within the batch to isolate the 50 temporally-coherent sequences. In fact, even if in natural vision abrupt gaze shifts could be detected to segment sequences, we prefer to avoid simplifying assumptions on this.
- To limit bias induced by the batch order presentation, we averaged experiments over 10 runs and at each run we randomly shuffled the batches $B_i, i = 2, \ldots, 10$ (B_1 is always used for initial supervised training). By measuring the standard deviation across the 10 runs, we can also study the learning process stability.
- To avoid overfitting we did not performed a fine adjustments of parameters characterizing the (parametric) update strategies. We set them according to some exploratory tests and then kept the same values for all the experiments:
  - For *SupTR*, the weight λ of the supervised component is set to $\frac{2}{3}$.
  - For *SST-A* the self-confidence threshold *sc* is set to 0.65.

When performing continual learning, care must be taken to avoid catastrophic forgetting. In fact, since patterns belonging to previous batches are no longer available, training the system with new patterns could lead to forget old ones. Even if in our tuning scenario the new patterns come from the same objects (pose and lighting variations) and there is some overlapping[7] in the training sequences, catastrophic forgetting is still an issue.

For HTM we experimentally found that a good trade-off between stability and plasticity can be achieved by running only 4 HSR iterations for each batch of 1,000 patterns, while for CNN we found that the optimal number of iterations is much higher (about 100 iterations).

[7] Since we are not enforcing any *mindist* between training sequences, the random walk can lead to the inclusion of the same frame in different sequences/batches. In our opinion, this better emulates unsupervised human experience with objects, where the same object view can be refreshed over time.

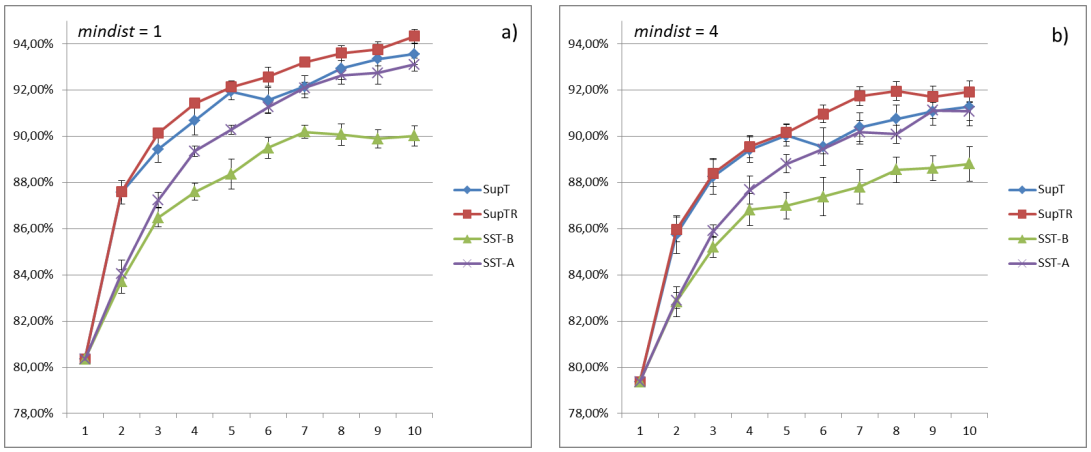


FIGURE 5.13: HTM accuracy on the test set for *mindist* = 1 (a) and *mindist* = 4 (b). Positions $2, \ldots, 10$ (x-coordinate) denote the test set accuracy after incremental tuning with batches $B_i, i = 2, \ldots, 10$. Position 1 exhibits the same accuracy for all the update strategies because it denotes the accuracy after supervised training on B_1. The bars denote 95% mean confidence intervals (over 10 runs).

Figure 5.13 shows HTM accuracy at the end of pre-training[8] on B_1 and after each training batch (points $B_2, \ldots, B_{10}$). We note that:

- Supervised tuning *SupT* works well and each new batch of data contributes to increase the overall accuracy.
- Regularized supervised tuning *SupTR* performs slightly better than *SupT*, and more important, makes the learning process more stable; this can be appreciated by the smoother trend in the graphs and by the average standard deviation over the 10 runs, that (for *mindist* = 1) is 0.7% for SupT and 0.4% for *SupTR*. This is in line with results of [Mobahi et al., 2009], where a relevant accuracy improvement was reported on COIL-100 when regularizing the supervised learning with temporal coherence. Here the gap between *SupT* and *SupTR* is smaller than in [Mobahi et al., 2009], probably due to the fact that our tuning batches are quite small (1,000 patterns) and regularization plays a minor role.
- *SST-B* and *SST-A* accuracy is surprisingly good when compared with supervised accuracy, proving that temporal continuity is a very effective surrogate of supervision for HTM. Initial trends of *SST-B* and *SST-A* are similar, then *SST-B* tends to stabilize while *SST-A* accuracy continues to increase approaching supervised update *SupT*. The self-confidence computation that *SST-A* uses to decide whether updating the gradient or not, seems to be a valid instrument to skip cases where temporal continuity is not effective (e.g. change of sequence, very ambiguous patterns, etc.).

Figure 5.14 shows the results of the same experiment performed with the CNN. Here we observe that:

- Accuracy at the end of initial supervised training (on B_1) is similar to HTM.

[8] No additional data (e.g., jittered patterns) is used for HTM pre-training.

- *SupT* and *SupTR* lead to a remarkable accuracy improvement during incremental tuning with $B_i, i = 2, \ldots, 10$ even if accuracy is about 2% lower than HTM and for *mindist* = 4 the learning process appears to be less stable.
- Unexpectedly, the semi-supervised tuning *SST-B* and *SST-A* did not work with our CNN implementation. We tried some modifications (architecture, learning procedure) but without success. The only way we found to increase accuracy in the semi-supervised scenario is with the variant of *SST-A* (denoted as *SST-A-Δ*) introduced and discussed in section 5.3.1.2, However, also for *SST-A-Δ* the accuracy gain is quite limited if compared with semi-supervised tuning on HTM.

A similar trend can be observed in the experimental results reported as additional materials (Appendix B.2), where the native object segregation is maintained.

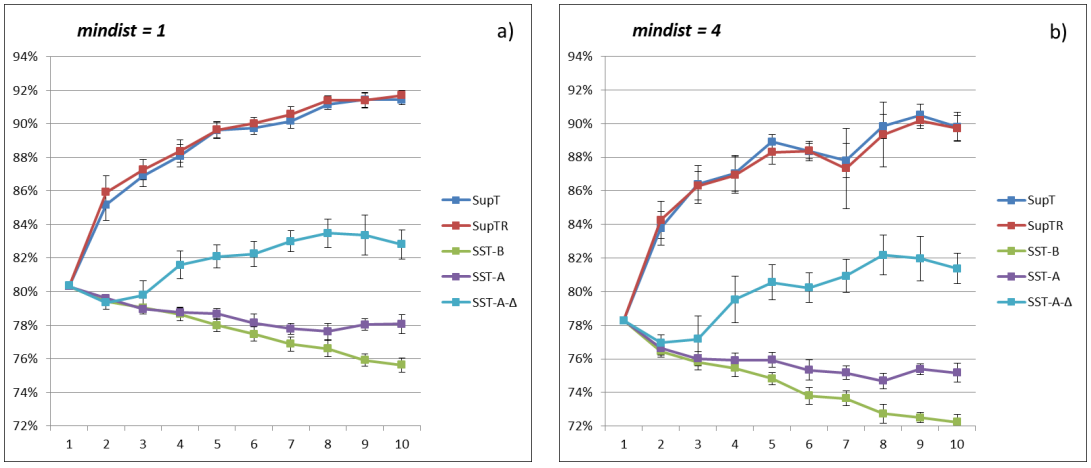


FIGURE 5.14: CNN accuracy on the test set for *mindist* = 1 (a) and *mindist* = 4 (b).

5.3.1.1 Making the problem harder

The good performance of HTM in semi-supervised tuning reported in the previous section could be attributed to the initial high-chance of self-discovering the pattern class. In fact, if the initial classification accuracy is high enough, the missing class label can be replaced by a good guess. To study SST effectiveness for harder problems, where the initial classification accuracy is lower, we set-up two experiments:

- the former consists in deliberately (and progressively) deteriorating the initial classification accuracy by providing a certain amount of wrong labels during the supervised training on B_1.
- the latter uses the same training and test batches but turns the problem into a 50-class classification. As discussed in section 4.2.1, this is much more difficult (expecially for 32×32 patterns) because different NORB objects (e.g. two cars) are visually very difficult to distinguish at certain angles (even for humans).

Figure 5.15 shows results of these experiments. We note that:

- As the initial classification accuracy degrades, *SST-A* accuracy degrades gently and the gap between initial and final accuracy remains high. Even a limited initial accuracy of about 35% does not prevent *SST-A* to benefit from semi-supervised tuning.

- Of course here the gap between *SST-A* and supervised tuning *SupT*/*SupTR* (not reported in the graph) is higher because supervised tuning is able to overcome the introduced initial degradation since the second batch, always leading to a final accuracy close to Figure 5.13.a.

- The 50-class experiment can be considered an extreme case, because the initial classification accuracy is about 25% and even supervised tuning approaches (*SupT* and *SupTR*) are not able to increase final performance over 44%. In this scenario *SST-B* after an initial stability (batches $B_2, \ldots, B_5$) starts drifting away (batches $B_6, \ldots, B_{10}$). On the contrary, *SST-A* denotes a stable (even if limited) accuracy gain, proving to be able to operate also in high uncertainty conditions.

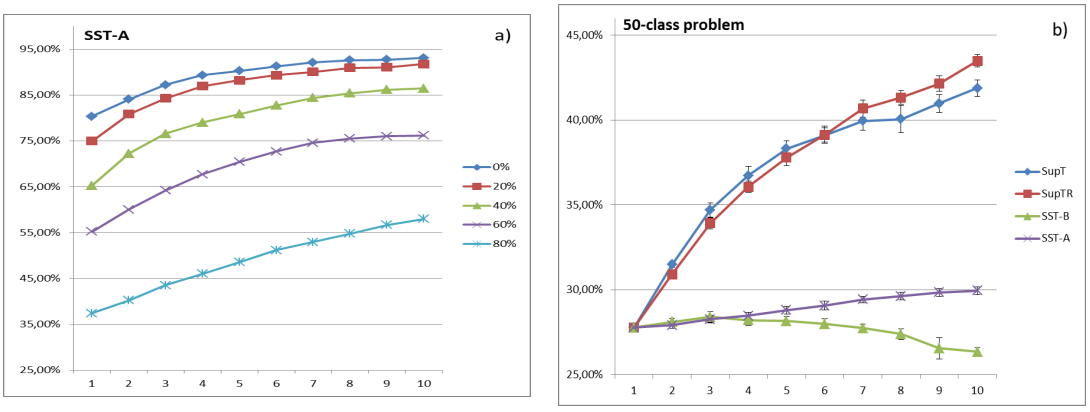


FIGURE 5.15: a) HTM + *SST-A* accuracy on the test set (*mindist* = 1) for different amounts of wrong labels provided during initial supervised training on B_1. b) HTM accuracy on the test set (*mindist* = 1) for different update strategies on the 50-class problem.

5.3.1.2 Control experiments

In this section we introduce further experiments with the aim to better understand the factors contributing to the success of semi-supervised tuning. In particular we modified *SST-A* as:

- ***SST-A*-Δ**:

$$d(v^{(t)}) = \begin{cases} \Delta_{\underset{i}{\text{argmax}}\, f(v^{(t)})} & \text{if } \max_i f_i(v^{(t)}) > sc \\ N(v^{(t)}) & \text{otherwise} \end{cases} \tag{5.1}$$

  This is very similar to *SST-A*, in fact $f(v^{(t)})$ is computed in the same way by exploiting temporal coherence, but here when the self-confidence is higher than the threshold, instead of enforcing the temporal coherent pattern $f(v^{(t)})$, we pass back the delta distribution corresponding to the self-guessed class.

- ***SST-A-Δ-noTC***:

$$d(v^{(t)}) = \begin{cases} \Delta_{\underset{i}{\operatorname{argmax}}\, N_i(v^{(t)})} & \text{if } \max_i N_i(v^{(t)}) > sc \\ N(v^{(t)}) & \text{otherwise} \end{cases} \tag{5.2}$$

Here no temporal coherence is used neither for estimating self-confidence nor for enforcing output continuity. This correspond to the basic self-training approach used in several applications.

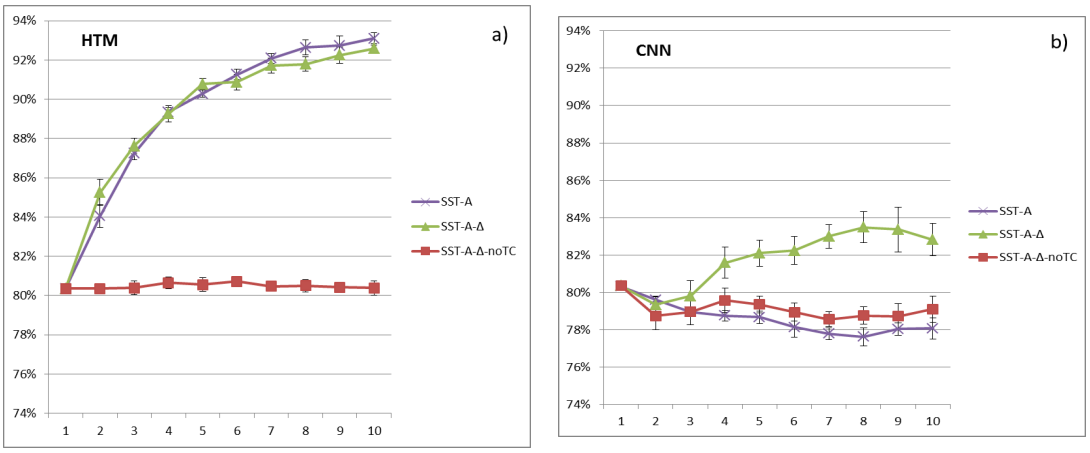


FIGURE 5.16: a) HTM accuracy (5-class problem, $maxdist = 1$) on SST-A and its two variants. b) CNN accuracy (5-class problem, $maxdist = 1$) on SST-A and its two variants.

Figure 5.16.a compares HTM accuracy on *SST-A* and the two above variants. The small gap between *SST-A* and *SST-A-Δ* (in favor of SST-A) can be attributed to the regularizing effect of passing back a temporally coherent output vector instead of a sharp delta vector. A totally unsatisfactory behavior can be observed for the second variant (*SST-A-Δ-noTC*) where the network cannot look back in time but can only exploits the current pattern: the flat accuracy in the graph testifies that in this case self-training does not allow the HTM to improve. This is a classical pitfall of basic self-training approaches where the patterns whose label can be correctly guessed do not bring much value to the improve the current representation while really useful patterns (in term of diversity) are not added because of the low self-confidence.

Figure 5.16.b shows CNN accuracy for the same experiments. While *SST-A-Δ-noTC* here too remains ineffective, in this case *SST-A-Δ* is much better than *SST-A*, even if far from semi-supervised accuracy achieved by HTM. But why our CNN implementation does not tolerate a desired output vector made of (combinations) of past output vectors, and prefer a more radical delta vector computed by self-estimation of the pattern class? By comparing the output vectors produced by HTM and CNN when making inference on new patterns, we noted that HTM posterior probabilities are quite peaked around one class (similarly to delta form) while for CNN they are more softly spread among different classes. Numerically this can be made explicit by computing the average entropy over the network outputs of 1,000 previously unseen patterns: for CNN we measured and entropy of 1.44 bit, while for HTM the entropy is 0.50 bit, which is much closer to the 0 entropy of delta vector. Therefore, it seems that HTM output vectors are

already in the right form for the loss function, while CNN output vectors need to be sharpened to make learning more effective.

5.3.2 Seq-COIL-100

To better generalize the results shown on *Seq-NORB* we extend the evaluation of *SST* also on *COIL-100* with the same deep architectures. Figure 5.17 shows HTM and CNN accuracy for different SST strategies. We observe that:

- The trend for supervised strategies is similar to NORB; both HTM and CNN constantly improve initial accuracy as new batches are presented, with the CNN slightly overperforming HTM. For HTM regularization seems not providing any advantage, probably due to the shorter sequence length (10 frames here instead of 20 frames in NORB) and the presence of gaps in the sequences (patterns segregated/excluded because of their inclusion in the test set).
- Here too, semi-supervised strategies performs better for HTM than for CNN. It is worth noting that in this case the base strategy *SST-B* outperforms *SST-A* thus indicating that the self-confidence threshold *sc* (kept fixed at 0.65) is probably too conservative for this dataset.

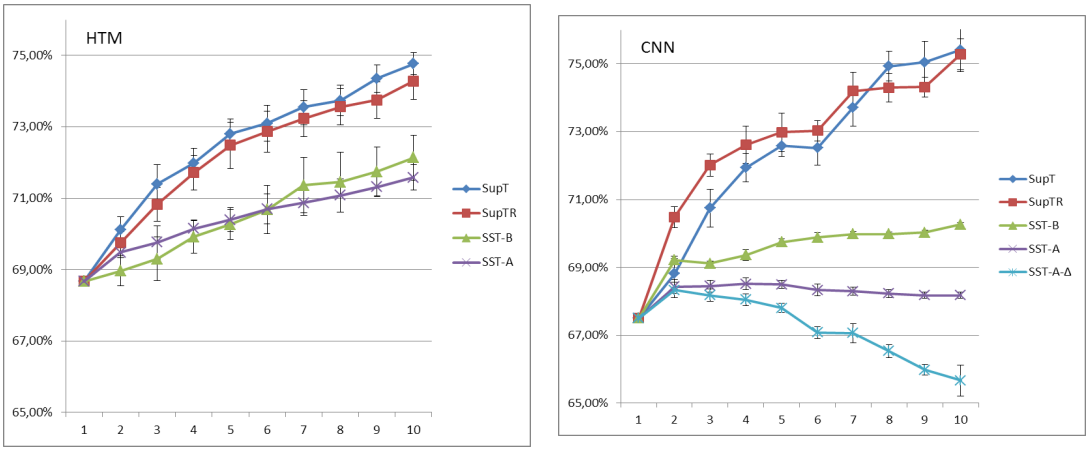


FIGURE 5.17: HTM and CNN incremental tuning accuracy on COIL-100.

5.3.3 Conclusions

In this section we studied semi-supervised tuning based on temporal coherence. The proposed tuning approaches have been evaluated on two deep architectures (HTM and CNN) obtaining partially discordant results. As to HTM our experiments proved that in some conditions even a trivial approach enforcing the output slow change (*SST-B*) can significantly improve classification accuracy. A slightly more complex approach (*SST-A*), exploiting temporal coherence twice: *i*) to enforce the output slow-change; *ii*) to compute a self-confidence value to trigger semi-supervised update, proved to be very effective, sometimes approaching the supervised tuning accuracy.

Our CNN implementation worked well with supervised tuning strategies, but (unexpectedly) demonstrated a lower capacity to deal with incremental semi-supervised tuning. Of course the encountered limitations could be due to the specific CNN architecture and training, and the outcomes of other recent studies [Goodfellow et al., 2013] can be very useful to check alternative setups (e.g., better investigating the effect of dropout). We recognize that the empirical evaluations carried out in this study are still limited, and to validate/generalize our semi-supervised tuning results, we need to test the proposed approaches on other larger datasets, including natural videos of real objects smoothly moving in front of the camera like the ones contained in *CORe50*.

However, based on the results obtained so far a question emerges: what made HTM more effective than CNN for incremental learning and semi-supervised tuning from temporal coherence? At this stage we do not have an answer to this question, and we can only formulate some hypotheses, by pointing out architectural/training differences that could have a direct impact on forgetting and capability to work with unlabeled data:

- **Pre-training**: McRae and Hetherington [1993] argued that network pre-training can mitigate catastrophic forgetting effects. During initial training HTM self-develop internal memories from patterns of the domain instead of randomly initializing weighs. This could make it more stable and resistant to pattern forgetting and lack of labels. Of course CNN can be pre-trained as well (see [Wagner et al., 2013] or a comparative evaluation of different pre-training approaches), and this is one of directions we intend to follow in our future studies.
- **Type of parameters tuned**: CNN training is mostly directed to feature extraction layers (i.e. filter parameters), while HTM + HSR main target are parameters of feature pooling layers. Maltoni and Rehn [2012] argued that the most important contribution of HSR is tuning the probabilities denoting how much each coincidence (i.e., a feature extractor) belongs to each group (i.e., a set of feature extractors). Our HTM incremental tuning by HSR is not altering feature extractors, but attempts to optimally arrange existing feature extractors in groups to maximize invariance. Referring to the stability-plasticity dilemma we speculate that keeping feature extractors stable (especially at low levels) promotes stability while moving pooling parameters is enough to get the required plasticity.

In conclusion, we believe that semi-supervised and unsupervised tuning, still scarcely studied with deep learning architectures, is a powerful approach to mimic biological learning where continual learning is a key factor. The lack of supervision, here surrogated by temporal coherence only, can be complemented by other contextual information coming from different modalities (*Multiview learning*), or from different processing paths (e.g., *Co-training*). Of course when supervisor signals are available, both supervised and unsupervised learning can be fused into an hybrid scheme (as here demonstrated for *SupTR*). Moreover, SST can be used in conjunction with other continual learning strategies presented in this chapter for explicitly addressing the issue of forgetting while learning from the new coming data.

The availability of powerful computing platforms makes the development of continual learning system feasible for a number of practical applications. For example in our non-optimized HTM

implementation, 4 HSR iterations on 1,000 patterns takes about 35 seconds (On a CPU Xeon W3550, 4 cores) we are confident that, upon proper optimization, SST can run on-line once a pre-trained system is switched in working mode.

6

Conclusions and Future Challenges

"Does anyone ever finish learning to read music? Do we finish learning how to write or do research? Do we ever learn anything completely? Or do we just keep getting better than we were before?"

– Mark Ring, 1994

6.1 Discussion and Conclusions

The intent of this dissertation was to provide a number of original contributions to the early development of continual learning research in the context of deep architectures for AI. The objective was to propose such contributions within a general approach to continual learning taking into account several practical factors as well as long-term goals.

The comprehensive framework proposed in Chapter 2, is an important step in this direction. The framework proposed, while not too abstract, is general enough to consider all the possible continual learning interpretation proposed so far and avoid possible and misleading misunderstandings that may arise when different point of view cannot find a more formal common ground. One of the most important steps in disambiguating state-of-the-art research is the disentanglement of the notion of task from the training batch. In fact, while not the principal focus of current continual learning research, many training batches may be related to the same task, or, the notion of task during training may not be available to the model at all. This is modeled in the framework with the availability of the t label, making explicit for each experiment or strategy the use of this additional supervised signal. The definition of this framework has allowed us to define three different scenarios with an intuitive interpretation: *multi-task*, *single-incremental-task* and *multiple-incremental-task* based on the nature and availability of the the t label.

Machine learning research is often driven by a practical results on complex benchmarks acknowledged by the community. However, in continual learning research, and especially considering deep learning architectures, there were no specific datasets or benchmarks available to assess new strategies and advance our understanding of the problem. This is why in this dissertation we proposed several benchmarks based on the re-designed of classic datasets such as *Seq-NORB*, *Seq-COIL100*, and *Seq-iCubWord28* but also completely new benchmarks such as *CORe50* and *3D-VizDOOM Maze*, specifically designed for continual learning research.

Having defined a rich set of benchmarks on which we could start to assess novel continual learning approaches, we proposed several CL strategies especially targeting *Single-Incremental-Task* scenarios, which, especially considering their additional complexity, was not very much explored until now, designing computationally lighter and memory efficient techniques such as *SST*, *CWR*, *CWR+* and *AR1*.

The evaluation conducted in Chapter 5, have shown that the continual learning strategies proposed may improve AI systems capabilities at many different levels. They may not always make our prediction models more adaptive and autonomous over time, but also solve many practical issues related to sustainability in terms of hardware resources with the ultimate goal of making AI more ubiquitous and scalable. The experimental evaluation carried out in different machine learning paradigms like supervised, reinforcement or semi-supervised learning, have further shown the impact CL may have not only per se, but especially if used in conjunction with many other techniques developed so far in the context of deep learning. While surely not completely exhaustive and improvable from many points of view, we think it has the sufficient expressive power to show how the pursuit of the continual learning paradigm may be beneficial for general AI research.

6.2 Open Challenges and Future Potential

All the benchmarks, metrics, strategies, experiments and the continual learning framework itself assume in the dissertation that a number of data batches and/or tasks is available to the model over time. However, as pointed out in [Chen and Liu, 2018], continual learning seen this way is just a passive process, i.e., the system has no control over the order in which the learning tasks are presented, which may be not only unrealistic for some applications but also limiting in terms of learning speed-up and scalability. Ruvolo and Eaton [2013b] considered ELLA in an active task selection setting. Assuming that there is a pool of candidate tasks, rather than choosing a task randomly as in ELLA, [Ruvolo and Eaton, 2013a] chose tasks in a certain order with the purpose of maximizing future learning performance using as few tasks as possible. However, very little has been done concerning this problem with state-of-the-art deep learning models. The problem has practical implications since each learning task may need a significant amount of time of manual labeling or each learning task may take a long time for the system to run. In such cases, learning in a task-efficient manner by choosing some tasks in certain order is more scalable to real-life continual learning problems.

Another important issue current continual learning techniques are not actually considering, is the ability to actually forget what is unimportant (i.e. removing biased knowledge for improving generalization). As brilliantly put by Kirkpatrick et al. [2017] in the development of *EWC*:

"Cascade models of synaptic plasticity construct dynamical models of synaptic states to understand the trade-off between plasticity and memory retention. Cascade models have important differences from our approach. In particular, they aim to extend memory lifetimes for systems at steady state (i.e., the limit of observing an infinite number of stimuli). As such, they allow for synapses to become more or less plastic and model the process of both retaining and forgetting memories. In contrast, we tackle the simpler problem of protecting the network from interference when starting from an empty net- work. In fact in EWC weights can only become more constrained (i.e., less plastic) with time and thus we can model only memory retention rather than forgetting."

This may be also particularly useful in circumstances where there is a semantic interference, that is when the target function h^* changes over time, and the need of *forgetting* becomes an imperative.

Learning continually over a long sequence of task is another open problem. Nowadays, state-of-the-art continual learning techniques are often assessed on a sequence of a dozen of tasks/batches or less. This does not allow us to infer much on the feasibility of learning continually in real-world applications often considering hundreds or even an unlimited amount of (often unbalanced) data batches. Up to date, being able to assess the performances of the AI system after deployment is also a related and challenging task. In fact, considering that storing data is often a not viable option, sophisticated techniques for assessing possible drifts in terms of model performance over past data distributions have not been proposed yet. Given the aforementioned problematics, *"online continual learning"*, not very much explored until now, seems to become a distant goal.

Novel local rules for enhancing synaptic plasticity, as early described in [Aljundi et al., 2017], is another interesting line of research for improving the efficiency of the continual learning process and loosen the tight dependency from a strongly supervised, end-to-end feedback signal.

Finally, the integration of continual learning with *Distributed* or *Federated Learning* is an open and particularly interesting research direction still to be explored that we regard as extremely important for the future of this field. Indeed, since the amount of information already processed and compressed within a prediction model by someone else will be much more likely to encounter and convenient to learn from than a long stream of raw data, we argue that the continual distillation and integration of already compressed knowledge will become an important topic for the future of CL.

Despite the great amount of open problems and relative youth of continual learning research with deep architectures, we regard the recent developments in this field as another important step towards the computational neuroscience community and the common objective of uncovering the computational principles of intelligence.

6.3 Closing Remarks

While most approaches in the field of artificial intelligence have successfully been demonstrated to operate in rather vertical and self-contained contexts, their application in more natural, ever-changing, multi-modal and multi-task settings has been relatively modest.

The goal of continual learning research, from a practical point of view, is to improve current machine learning methods for efficiently tackling such complex problems. Continual learning may even impact artificial software systems used today, for example in domains such as *computer vision*, *robotics* and *internet-of-things*, with applications running on the edge and dealing with complex non-stationary environments other than subject to substantial hardware limitations or privacy constraints.

Nevertheless, continual learning is not just a matter of learning efficiency. In the introduction of this dissertation, we argued that current artificial learning systems show particular weaknesses in five of the most important characterization of intelligence: *adaptation*, *scalability*, *autonomy*, *common sense* and *reasoning*. The central thesis of this dissertation is that *continual learning*, more or less explicitly, may bring us a step closer in overcoming these shortcomings.

The incremental development of cognitive abilities within the lifetime of an individual has been found to be one of the dominating factors of natural learning and intelligence. If computers ever are to exhibit rapid learning and adaptation capabilities similar to that of humans, they will most likely have to follow the same principles.

Appendix A

Hierarchical Temporal Memory Overview

This Appendix provides a brief overview of the HTM algorithm. A more detailed introduction to HTMs structure, forward and backward messaging (including equations) is given in Sections 1 and 2 of [Maltoni, 2011b]. HTMs pre-training algorithms are presented in detail in [Maltoni, 2011a] while the HTM Supervised Refinement (HSR) is introduced in [Maltoni and Rehn, 2012].

Structure An HTM has a hierarchical tree structure. The tree is built up by a number of levels, each composed of one or more nodes. A node in one level is bidirectionally connected to one or more nodes in the level below and the number of nodes in each level decreases as we ascend the hierarchy. Conversely, the node receptive fields increase as we move up in the tree structure. By allowing nodes to have multiple parents we can create networks with overlapping receptive fields. The lowest level is the *input* level, and the highest level (with typically only one node) is the *output* level. Levels and nodes in between input and output are called *intermediate* levels and nodes.

- Input nodes constitute a sort of interface: in fact, they just forward up the signals coming from the input pattern.
- Every intermediate node includes a set, C, of so-called coincidence-patterns (or just *coincidences*) and a set, G, of coincidence *groups*. A coincidence, c_i, is a vector representing a prototypical activation pattern of the node's children. Coincidence groups are clusters of coincidences likely to originate from simple variations of the same input pattern. Coincidences belonging to the same group can be spatially dissimilar but likely to be activated close in time when a pattern smoothly moves through the node's receptive field (i.e., temporal pooling). The assignment of coincidences to groups within each node is encoded in a probability matrix PCG, where each element, $PCG_{ji} = P(c_j|g_i)$, represents the probability of a coincidence c_j, given a group g_i.

- The structure of the output node differs from that of the intermediate nodes. In particular the output node has coincidences but not groups. Instead of memorizing groups and group likelihoods, it stores a probability matrix PCW, whose elements $PCW_{ji} = P(c_j|w_i)$ represent the probability of coincidence c_j given the class w_i.

Inference. HTM inference (feedforward flow) proceeds from input to output level. Each intermediate node: *i*) computes its coincidence activations by combining the messages coming from its child-nodes according to the activation patterns encoded by the coincidences themselves; *ii*) calculates its group activations by mixing coincidence activations through PCG values; finally, *iii*) passes up information to parent node(s). The output node computes its coincidence activations and turn them to class posterior probabilities according to PCW.

Pre-training. HTM pre-training is unsupervised for intermediate levels and partially supervised for the output level. Coincidences are learnt by sampling the space of activation patterns while smoothly moving training patterns across the node(s) receptive fields. Once coincidences are created they are clustered in groups by maximizing a temporal proximity criterion. The output node coincidences are learnt in the same (unsupervised) way but coincidence-class relationships are learnt in a supervised fashion by counting how many times every coincidence is the most active one (i.e., the winner) in the context of each class.

HTM Supervised Tuning (HSR). The probabilities in PCG's (remember there is one PCG matrix for each intermediate node) and PCW are the main elements manipulated by HSR. Similarly to error backpropagation, HSR incrementally updates parameter values by taking steps in direction opposite to the gradient of a loss-function. The whole process is implemented in a simple (and computationally light) way based on native HTM (backward) message passing.

Appendix B

Further Experiments on NORB

B.1 Baseline Accuracy on NORB

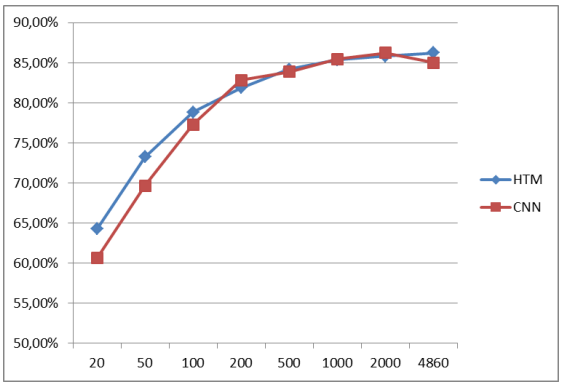
Here we report accuracy of HTM and CNN on the *"standard"* normalized-uniform NORB benchmark [LeCun et al., 2004]. We consider monocular 32×32 patterns and study the classification accuracy on the full test set of 24,300 patterns, for training sets of increasing size. Results reported below are obtained through a 5-fold cross validation, where for each round, 1/5 of the test set was taken as validation set to stop the gradient descent at an optimal point and the remaining 4/5 used to measure accuracy.

HTM training was performed as described in [Maltoni, 2011b]: a subset of the available patterns is used for pre-training and the rest of the patterns for supervised tuning through HSR. This allows to better control the network complexity when scaling to large training sets. Since the HTM pre-training algorithm [Maltoni, 2011b] internally generates a number of jittered versions[1] of the input patterns to emulate temporally coherent exploration sequences, for a fair comparison we exported these patterns and added them to the training set used for CNN training[2]. The number of HSR iterations (for optimal convergence on the validation set) is almost always less than 50. CNN training is performed with mini-batches of 100-200 patterns. The number of error backpropagation iterations (for optimal convergence on the validation set) is almost always less than 150 iterations.

Figure B.1 shows the accuracy of HTM and CNN. When the number of training patterns per class is small (i.e., 20, 50 and 100) HTM accuracy is slightly better than CNN; for larger training sets the accuracy of the two approaches is very similar. Concerning the training time, a direct comparison is not possible because of different implementation languages and hardware platforms. In particular, the CNN Theano implementation run on a GPU Tesla C2075 Fermi, while the HTM run on a CPU Xeon W3550 - 4 cores. However, to give a coarse indication, both

[1] Consisting of small translations, rotations and scale changes.

[2] This is not the case for experiment with temporally coherent sequences (reported in Chapter 5.) because when input comes from slowly moving patterns HTM does not need to internally generate jittered patterns.



Training patterns	Jittered versions	HTM	CNN
20×5	800	64.21%	60.58%
50×5	2,000	73.22%	69.64%
100×5	4,000	78.82%	77.27%
200×5	4,000	81.86%	82.80%
500×5	4,000	84.16%	83.87%
1,000×5	4,000	85.37%	85.47%
2,000×5	4,000	85.83%	86.20%
4,860×5	4,000	86.24%	85.01%

FIGURE B.1: HTM and CNN accuracy on standard normalized-uniform NORB benchmark. The labels (number of training patterns per class) in the x-coordinate are equispaced for better readability.

HTM and CNN training took about 3 hours[3] for the largest training set case: 4860×5 + 4000 patterns.

B.2 Continual Learning on Seq-NORB (Native Object Segregation)

The NORB benchmark introduced in Section 4.2.1 focuses on pose and lighting continual learning and, unlike the original NORB protocol, it does not split the objects in two disjoint groups: for each class, 5 objects in the training set and 5 objects in the test set.

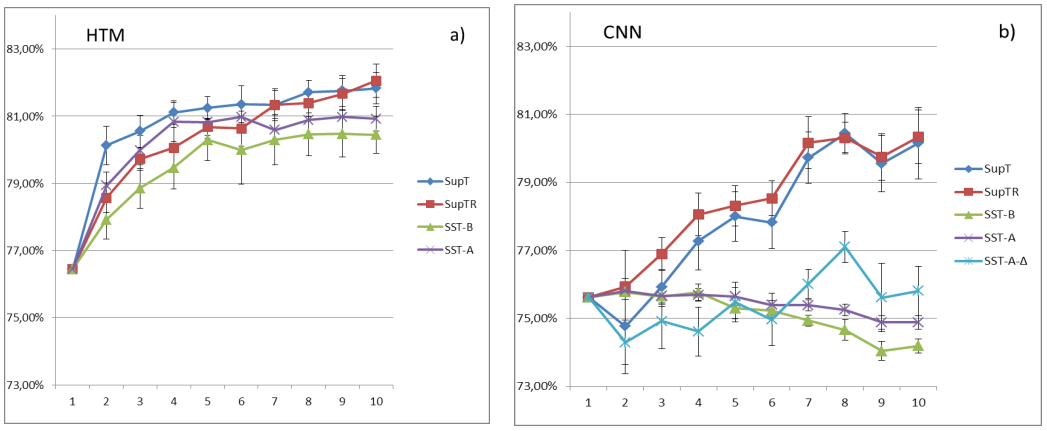


FIGURE B.2: HTM and CNN incremental tuning accuracy, when splitting class objects as in the original NORB protocol (for each class: 5 objects in the training set and 5 in the test set). No *mindist* is here necessary between test and training batches because of the object segregation.

Our choice was aimed at isolating the capability of learning pose and invariance from the capability of recognizing different objects of the same class (which is critical in NORB because of the small number of objects per class). However, to further validate the efficacy of the proposed

[3] For a single round of cross-validation.

continual tuning, here we came back to the native object segregation and report results corresponding to Section 5.3.1 results under this scenario. Figure B.2 shows HTM and CNN accuracy for different tuning strategies. We observe that:

- The trend is very similar to the Section 5.3.1 experiments: even in this case, supervised strategies work well for both strategies while semi-supervised tuning is effective for HTM but not for our CNN implementation.
- The accuracy achieved is markedly lower with respect to Section 5.3.1, but is in line with results reported above if we consider the number of training samples and the forgetting effect due to continual learning.

Appendix C

Adapting Pre-trained CNN to Different Input Size

In the recent years, the pervasiveness of deep neural networks and the complexity of training such architectures on datasets of remarkable size has led to the proliferation of pre-trained models which represent a very good starting point for many customized solutions. However, this approach requires adapting problem-specific data to a fixed size architecture which was designed and optimized to solve another task. In the context of computer vision and object recognition, for example, it is very common to stretch images of arbitrary sizes to 227×227 pixels which is the typical input of well-known CNN models pre-trained on *ImageNet*: this often leads to highly distort the original patterns and significantly increases inference time. A more elegant (and efficient) approach is adapting a pre-trained model to work with input patterns of different size. This is straightforward for convolution and pooling layers thanks to local (shared) connections, but is much more problematic for fully connected layers, whose number of weights depends on the input image size. In this case, two main strategies can be used:

1. Applying fixed size pooling (global or pyramidal) over the last convolution/pooling layer as proposed in [He et al., 2015; Lin et al., 2014; Ren et al., 2017]. However, finetuning of upper levels might be necessary if the input scale changes dramatically or the original model was not designed with a fixed-size pooling layer at all.

2. Reusing the pre-trained network up to the last convolution layer and retraining the fully connected layers from scratch on the new task and input size. A typical approach is also to train an external classifier (e.g., SVM) from pooled features just after the last convolutional layer.

Independently of the network adaption to a different input size, when the problem classes change, the final softmax layer needs to be replaced and re-trained from scratch.

Since in our experiments we used the classic CaffeNet and VGG models (which have not been trained in a multi-scale fashion) and we aimed at fast processing, we opted for the second strategy.

TABLE C.1: Accuracy differences between reduced size CNNs (Mid) and the corresponding full-size models on the 50 classes task. All the models have been pre-trained on *ILSVRC-2012*. SVM training and CNN fine-tuning were performed on *CORe50*.

		Accuracy (object level: 50 classes)	
	CNN + SVM (on top of ...)	**fc6**	**pool5**
1	CaffeNet	63,46%	63,14%
2	Mid-CaffeNet		52,84%
3	VGG	69,03%	70,91%
4	Mid-VGG		59,25%
	CNN + Finetuning	**Accuracy (object level: 50 classes)**	
5	CaffeNet	75,97%	
6	Mid-CaffeNet	65,98%	
7	VGG	77,39%	
8	Mid-VGG	69,08%	

Hence, we reshaped the input volume to 3×128×128, halved[1] the number of units in the fully connected layers *fc6* and *fc7* (from 4096 to 2048) and re-trained them from scratch. This results in a relevant speedup at inference time (3.4× for CaffeNet and 4.67× for VGG). The resulting mid-size models are now suitable to be tuned on *CORe50* native 128×128 frames.

Table C.1 summaries our findings. For the full-size models, extracting features from *fc6* or *pool5* is nearly equivalent in terms of accuracy (compare columns *fc6* and *pool5* for raw 1 and 3 in the table). So the lack of a fully pre-trained *fc6* in the mid-size models is not critical. However, in the experiments with SVM (rows 1:4), the mid-size networks loose about 10% accuracy with respect their original version. A similar gap (just slightly smaller for VGG) can be observed when the networks are finetuned (rows 5:8). The reason of such accuracy drop is not totally clear to us. On the one hand, if we consider finetuning experiments (rows 5:8), *fc6* and *fc7* have been pre-trained on a higher number of patterns in the full-size networks, and therefore it is reasonable to expect higher accuracy; on the other hand, if we consider *pool5* + SVM experiments, both the network exploits the same pre-training and stretching our input patterns from 128×128 to 227×227 (in principle) does not add new information.

We did similar experiments on other datasets (e.g., *NORB*, *COIL-100*, *BigBrother*, *iCubWorld32*) and obtained close results: it seems that the zoomed image, even if a blurred, allow a more detailed feature extraction to be performed by the network. This can be due to the spatial scale of the filters learned on *ILSVRC-2012* or by a richer hierarchical representation (more neurons and link between neurons cover the object region). We believe that more investigations are necessary to fully understand the reasons and to make available pre-trained mid-size networks (for patterns whose native size is close to 128×128) which are competitive with full-size ones.

[1] As other authors we noted that such reduction has no significant impact on accuracy.

Appendix D

Single-Incremental-Tasks Experiments Details

D.1 Implementation Details (Caffe framework)

Since implementing dynamic output layer expansion was tricky in Caffe framework, we initially implemented the different strategies by using a single *maximal head* (i.e., including all the problem classes since from the beginning) instead of an *expanding head*. In principle, the two approaches are quite similar, since if a particular batch does not contain patterns of a given class, no relevant error signals are sent back along the corresponding connections during SGD. Hoverer, looking at the details of the training process, the two approaches are not exactly the same.

For example, for CWR+ we verified, with some surprise, that the maximal head simplifying approach constantly leads to better accuracy (up to 6-7% on CORe50) w.r.t. to the expanding head approach. We empirically found that the reason is related to the gradient dynamics during the initial learning iterations: working with a higher number of classes makes initial predictions smaller (because of softmax normalization) and the gradient correction for the true class stronger; in a second stage, predictions start to converge and the gradient magnitude is equivalent in the two approaches. It seems that for SGD learning (with fixed learning rate) boosting the gradient in the first iterations favors accuracy and reduces forgetting. We checked this by experimentally verifying that the expanding head approach combined with a variable learning rate performs similarly to maximal head with fixed learning rate. Therefore, to maximize accuracy and reduce complexity, CWR and its evolutions (CWR+ and AR1) have been implemented with the maximal output layer approach. Referring to the pseudocode in Algorithms 1, 2 and 3, it is sufficient to keep to constant maximum size (e.g., 50 for CORe50) and remove the line *"expand output layer with..."*.

For the other approaches we verified that: LWF performs slightly better with expanding head approach while EWC and SI work better (and are easy to tune) with maximal head. To produce

the results presented in Section 5.1.4 we used for each strategy the approach that proved to be the most effective. Strategy specific notes are reported in the following for Caffe implementation.

LWF It is worth noting that in Caffe a cross-entropy loss layer accepting soft target vectors is not available in the standard layer catalogue and a custom loss layer need to be created.

EWC To implement EWC in Caffe we:

- compute, average and clip F^i values in pyCaffe (for maximum flexibility). To calculate F^i_k the variance of the gradient should be computed by taking the gradient of each of the n_i patterns in isolation. To speed-up implementation and improve efficiency we computed the variance at mini-batch level, that is using the average gradients over mini-batches. In our experiment we did not note any performance drop even when using mini-batches of 256 patterns.
- pass F and Θ^* to the solver via a further input layer.
- modified SGD solver, by adding a custom regularization that perform EWC regularization in weight decay style.

SI Starting from EWC implementation, SI can be easily setup in Caffe, in fact the regularization stage is the same and we only need to compute F^i values during SGD. To this purpose, in current implementation for maximum flexibility, we used pyCaffe.

D.2 Architectural Changes in the Models Used on CORe50

TABLE D.1: Summary of changes w.r.t. the original CaffeNet and GoogLeNet models used in this dissertation.

CaffeNet		
Layer	*Original*	*Modified*
data (Input)	size: 227×227	size: 128×128
conv1 (convolutional)	stride: 4	stride: 2
conv2 (convolutional)	pad: 2	pad: 1
fc6 (fully connected)	neurons: 4096	neurons: 2048
fc7 (fully connected)	neurons: 4096	neurons: 2048
fc8 (output)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)
GoogLeNet		
Layer	*Original*	*Modified*
data (Input)	size: 224×224	size: 128×128
conv1/7x7_s2 (convolutional)	stride: 2, pad: 3	stride: 1, pad: 0
loss1/ave_pool (pooling)	kernel: 5	kernel: 6
loss1/fc (fully connected)	neurons: 1024	layer removed
loss1/classifier (output int. 1)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)
loss2/ave_pool (pooling)	kernel: 5	kernel: 6
loss2/fc (fully connected)	neurons: 1024)	layer removed
loss2/classifier (output int. 2)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)
loss3/classifier (output)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)

D.3 Hyperparameter Values for CORe50

TABLE D.3: the hyperparameter values used for CaffeNet and GoogLeNet on CORe50. The selection was performed on run 1, and hyperparameters were then fixed for runs $2, \ldots, 10$.

Cumulative		
Parameters	*CaffeNet*	*GoogLeNet*
epochs, η (learn. rate)	4, 0.0025	4, 0.0025
Naive		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.005
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003

LWF		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Expanding	Expanding
map	[0.66...0.9] → [0.45...0.85]	[0.66...0.9] → [0.45...0.85]
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0002	2, 0.0002
EWC		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
max_F	0.001	0.001
λ	5.0e7	3.4e7
B_1: epochs, η (learn. rate)	2, 0.001	4, 0.002
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.000025	2, 0.000035
SI		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
ξ	1e-7	1e-7
$w_1, w_i (i > 1)$	0.00001, 0.005	0.00001, 0.005
max_F	0.001	0.001
λ	5.0e7	3.4e7
B_1: epochs, η (learn. rate)	2, 0.001	4, 0.002
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.00002	2, 0.000035
CWR		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
$w_1, w_i (i > 1)$	1.25, 1	1, 1
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003
CWR+		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003

AR1		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
ξ	1e-7	1e-7
$w_1, w_i (i > 1)$	0.0015, 0.0015	0.0015, 0.0015
max_F	0.001	0.001
λ	8.0e5	8.0e5
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003

D.4 Hyperparameter Values for iCIFAR-100

TABLE D.4: the hyperparameter values used for CifarNet [Zenke et al., 2017] on iCIFAR-100. The selection was performed on run 1, and hyperparameters were then fixed for runs $2, \ldots, 10$.

Cumulative	
Parameters	*CifarNet*
epochs, η (learn. rate)	180, 0.005
Naive	
Parameters	*CifarNet*
Head (see App. A)	Maximal
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001
LWF	
Parameters	*CifarNet*
Head (see App. A)	Expanding
map	[0.5...0.9] $\rightarrow$ [0.45...0.85]
B_1: epochs, η (learn. rate)	20, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	20, 0.001
EWC	
Parameters	*CifarNet*
Head (see App. A)	Maximal
max_F	0.001
λ	8.0e7
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	25, 0.00002

SI	
Parameters	*CifarNet*
Head (see App. A)	Maximal
ξ	1e-7
$w_1, w_i (i > 1)$	0.00001, 0.00175
max_F	0.001
λ	6.0e7
B_1: epochs, η (learn. rate)	60, 0.0005
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.00002
CWR	
Parameters	*CifarNet*
Head (see App. A)	Maximal
$w_1, w_i (i > 1)$	1, 1
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001
CWR+	
Parameters	*CifarNet*
Head (see App. A)	Maximal
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001
AR1	
Parameters	*CifarNet*
Head (see App. A)	Maximal
ξ	1e-7
$w_1, w_i (i > 1)$	0.00015, 0.000005
max_F	0.001
λ	4.0e5
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001

D.5 Standard Deviation for CORe50

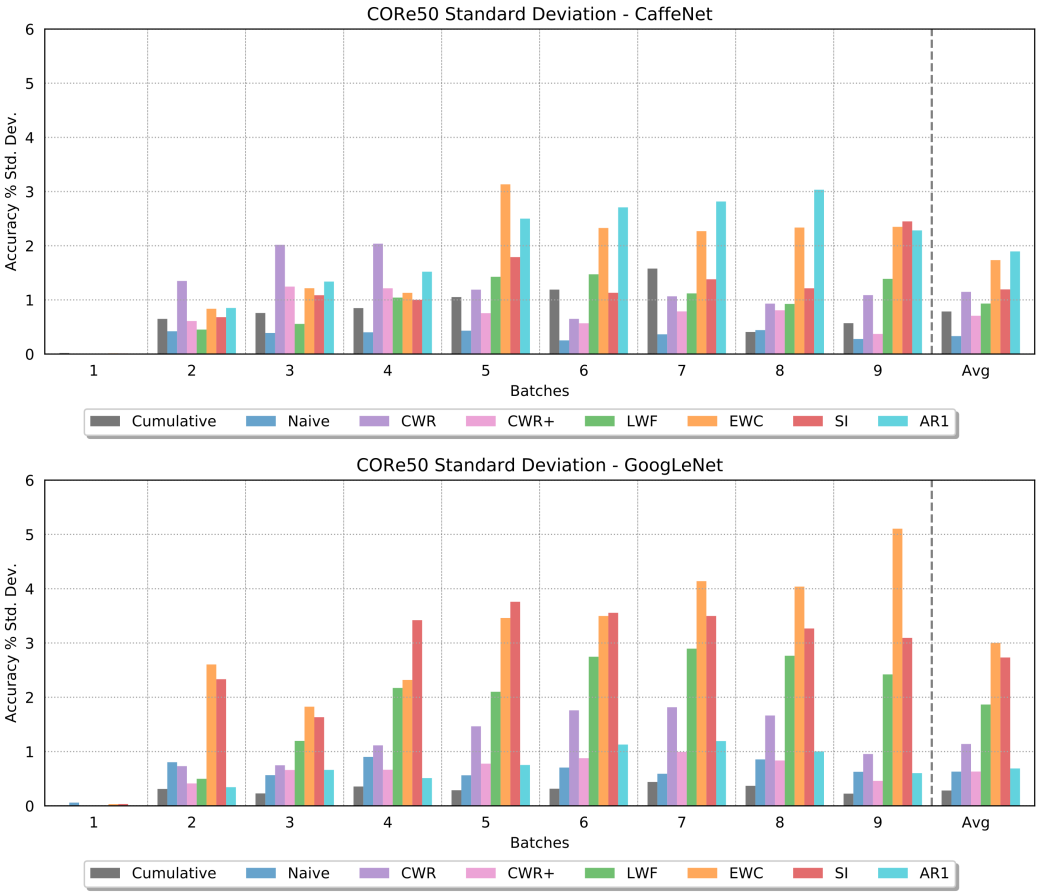


FIGURE D.1: Accuracy standard deviation visualization for each strategy and the two models (CaffeNet and GoogLeNet) over 10 distinct runs where the batches order has been randomly shuffled. Averaged values over the batches are also reported. Better viewed in color.

D.6 Standard Deviation for iCIFAR-100

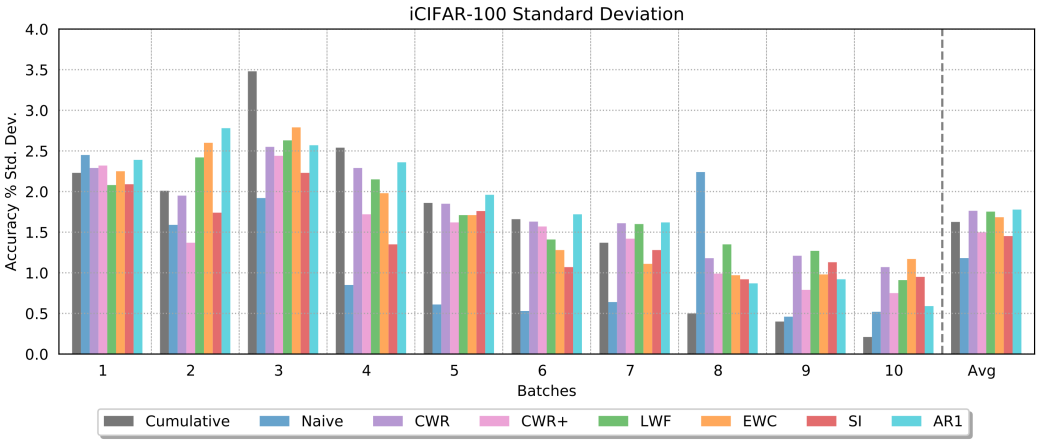


FIGURE D.2: Accuracy standard deviation visualization for each strategy over 10 distinct runs where the batches order has been randomly shuffled. Averaged values over the batches are also reported. Better viewed in color.

D.7 On Initializing Output Weight to Zero

It is well known that neural network weights cannot be initialized to 0, because this would cause intermediate neuron activations to be 0, thus nullifying backpropagation effects. While this is certainly true for intermediate level weights, it is not the case for the output level.

More formally, let θ_{ab} be a weight of level l, connecting neuron a at level $l-1$ with neuron b at level l and let net_x and out_x be the activation of neuron x before and after the application of the activation function, respectively; then, the gradient descent weight update is proportional to:

$$\frac{\partial L_{cross}(\hat{y},t)}{\partial \theta_{ab}} = \frac{\partial L_{cross}(\hat{y},t)}{\partial net_b} \cdot \frac{\partial net_b}{\partial \theta ab} = \frac{\partial L_{cross}(\hat{y},t)}{\partial net_b} \cdot out_a \tag{D.1}$$

It is well evident that if out_a is 0, weight update cannot take place; therefore weights of levels up to $l-1$ cannot be all initialized to 0. For the last level (i.e., l coincides with output level), in case of softmax activation and cross-entropy loss, eq. D.1 becomes Sadowski [2016]:

$$\frac{\partial L_{cross}(\hat{y},t)}{\partial \theta_{ab}} = (\hat{y}_b - t_b) \cdot out_a = (out_a - t_b) \cdot out_a \tag{D.2}$$

and initializing θ_{ab} to 0 does not prevent the weight update to take place.

Bibliography

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., and Vijayanarasimhan, S. (2016). YouTube-8M: A Large-Scale Video Classification Benchmark. *arXiv preprint arXiv:1609.08675*.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2017). Memory Aware Synapses: Learning what (not) to forget. *ArXiv preprint arXiv:1711.09601v2*, page 16.

Ammar, H. B., Eaton, E., Luna, J. M., and Ruvolo, P. (2015a). Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. *IJCAI International Joint Conference on Artificial Intelligence*, 2015-Janua(Ijcai):3345–3351.

Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. E. (2014). Online Multi-Task Learning for Policy Gradient Methods. *Icml*, pages 1206–1214.

Ammar, H. B., Tutunov, R., and Eaton, E. (2015b). Safe Policy Search for Lifelong Reinforcement Learning with Sublinear Regret. 37.

Anderson Jr., J. D., Rycroft, M. J., and Shyy, W. (1999). *A History of Aerodynamics: And Its Impact on Flying Machines*. Cambridge University Press.

Andreas, J., Rohrbach, M., Darrell, T., Klein, D., and Sciences, C. (2015). Deep Compositional Question Answering with Neural Module Networks. *arXiv preprint arXiv:1511.02799*, (Figure 1).

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. pages 1–16.

Banerjee, B. and Stone, P. (2007). General game learning using knowledge transfer. *IJCAI International Joint Conference on Artificial Intelligence*, pages 672–677.

Baylor, D., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E., Wicke, M., Breck, E., Wilkiewicz, J., Zhang, X., Zinkevich, M., Cheng, H.-t., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., and Jain, V. (2017). TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2017)*, pages 1387–1395.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Lefrancq, A., Green, S., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. (2016). DeepMind Lab. pages 1–11.

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127.

Bengio, Y., Lee, D.-H., Bornschein, J., and Lin, Z. (2015). Towards Biologically Plausible Deep Learning. *arXiv preprint arxiv:1502.0415*, page 18.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, New York, New York, USA. ACM Press.

Benna, M. K. and Fusi, S. (2016). Computational principles of synaptic memory consolidation. 19(12).

Borji, A., Izadi, S., and Itti, L. (2016). iLab-20M: A Large-Scale Controlled Object Dataset to Investigate Deep Learning. In *International Conference of Computer Vision and Pattern Recogniton (CVPR)*, pages 2221–2230.

BostonDynamics (2018). SpotMini: A nimble robot that handles objects, climbs stairs, and will operate in offices, homes and outdoors.

Box, G. E. (1979). Robustness in the strategy of scientific model building. *Robustness in statistics*, pages 201—-236.

Canziani, A. and Culurciello, E. (2017). CortexNet: a Generic Network Family for Robust Visual Temporal Representations. (1):1–8.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr, E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. *AAAI*, 5:3.

Carpineti, C., Lomonaco, V., Bedogni, L., Felice, M. D., and Bononi, L. (2018). Custom Dual Transportation Mode Detection by Smartphone Devices Exploiting Sensor Diversity. *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 367–372.

Caruana, R. (1997). Multitask Learning. *Machine learning*, 28(1):41—-75.

Caselles-Dupré, H., Annabi, L., Hagen, O., Garcia-Ortiz, M., and Filliat, D. (2018). Flatland: a Lightweight First-Person 2-D Environment for Reinforcement Learning.

Chapelle, O., Scholkopf, B., and Zien, A. (2006). *Semi-supervised learning*.

Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the Devil in the Details: Delving Deep into Convolutional Nets. *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1–11.

Chen, Z. and Liu, B. (2014a). Mining Topics in Documents : Standing on the Shoulders of Big Data. *Proceedings of the 20th ACM conference on Knowledge discovery and data mining – KDD' 14*, pages 1116–1125.

Chen, Z. and Liu, B. (2014b). Topic Modeling using Topics from Many Domains, Lifelong Learning and Big Data. *Proceedings of the 31st International Conference on Machine Learning - ICML '14*, 32:703–711.

Chen, Z. and Liu, B. (2018). *Lifelong Machine Learning*. Morgan & Claypool Publishers.

Cheng, Hao and Fang, Hao and Ostendorf, M. (2015). Open-Domain Name Error Detection using a Multi-Task RNN. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 737—-746.

Clopath, C. (2012). Synaptic consolidation: An approach to long-term learning. *Cognitive Neurodynamics*, 6(3):251–257.

Craye, C., Lesort, T., Filliat, D., and Goudou, J.-F. (2018). Exploring to learn visual saliency: The RL-IAC approach. pages 1–19.

Cui, Y., Surpur, C., Ahmad, S., and Hawkins, J. (2015). Continuous online sequence learning with an unsupervised neural network model.

David E. Rumelhart, Hinton, G. E., and Williams, R. J. (1986). Learning Representation by Back-propagating Errors. *nature*, 323:533.

Deisenroth, M. P., Englert, P., Peters, J., and Fox, D. (2014). Multi-Task Policy Search for Robotics. pages 3876–3881.

Dispenza, J. (2008). *Evolve Your Brain: The Science of Changing Your Mind*. Health Communications, Inc.

Dredze, M., Dredze, M., Crammer, K., and Crammer, K. (2008). Online methods for multi-domain learning and adaptation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (October):8.

Farquhar, S. and Gal, Y. (2018). Towards Robust Evaluations of Continual Learning. (Vi).

Fernández, F. and Veloso, M. (2013). Learning domain structure through probabilistic policy reuse in reinforcement learning. *Progress in Artificial Intelligence*, 2(1):13–27.

Feuerstein, R. (2003). *Dynamic assessment of cognitive modifiability*. ICELP (The International Center for the Enhancement of Learning Potential).

Franco, A., Maio, D., and Maltoni, D. (2009). The Big Brother Database : Evaluating Face Recognition in Smart Home Environments. In *Advances in Biometrics: Third International Conference (ICB)*, pages 142–150.

Franco, A., Maio, D., and Maltoni, D. (2010). Incremental template updating for face recognition in home environments. *Pattern Recognition*, 43(8):2891–2903.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135.

George, D. and Hawkins, J. (2009). Towards a Mathematical Theory of Cortical Micro-circuits. *PLoS computational biology*, 5(10).

Gepperth, A. and Hammer, B. (2016). Incremental learning algorithms and applications. (April):27–29.

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76.

Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. (2005). The Amsterdam Library of Object Images. *International Journal of Computer Vision*, 61(1):103–112.

Ghosh-Dastidar, S. and Adeli, H. (2009). Spiking Neural Networks. *International journal of neural systems*, 19(4):295–308.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*, volume 1.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2013). An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks. *arXiv preprint arXiv:1312.6211*.

Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and Lecun, Y. (2015). Unsupervised feature learning from temporal data. *arXiv preprint arXiv:1504.02518*.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing Machines. pages 1–26.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end speech recognition. pages 1–12.

Hassibi, and Stork, D. (1993). Second order derivaties for network prunning: Optimal brain surgeon. *NIPS*, pages 164–171.

Haussler, D. (1990). *Probably approximately correct learning*. University of California, Santa Cruz, Computer Research Laboratory.

Hayes, T. L., Cahill, N. D., and Kanan, C. (2018a). Memory Efficient Experience Replay for Streaming Learning. *arXiv preprint arXiv:1809.05922*.

Hayes, T. L., Cahill, N. D., and Kanan, C. (2018b). New Metrics and Experimental Paradigms for Continual Learning. pages 1–4.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 4, pages 770–778. IEEE.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., AparXiv preprint arXiv:1706.06551ps, C., and Hassabis, D. (2017). Grounded Language Learning in a Simulated 3D World. *arXiv preprint arXiv:1706.06551*.

Hinton, G. (2012). Lecture 6d: a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning. Technical report, Slides of Lecture Neural Networks for Machine Learning.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Ieee Signal Processing Magazine*, (November):82–97.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop (2015)*, pages 1–9.

Hoffman, M., Blei, D., and Bach, F. (2010). Online learning for latent dirichlet allocation. *Nips*, pages 1–9.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Neuroscience*, 160(1):106–154.

Humphreys, L. G. (1979). *The construct of general intelligence*. Elsevier.

Huszár, F. (2018). Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences of the United States of America*, 115(11):E2496–E2497.

Ishizaka, A. and Nemery, P. (2013). *Multi-criteria decision analysis: methods and software*. John Wiley & Sons.

Jantke, K. P. (1993). Types of Incremental. (413).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. *Proceedings of the ACM International Conference on Multimedia*, pages 675–678.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2016-Janua, pages 4246–4247.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One Model To Learn Them All.

Kaplanis, C., Shanahan, M., and Clopath, C. (1987). Continual Reinforcement Learning with Complex Synapses.

Kapoor, A. and Horvitz, E. (2009). Principles of lifelong learning for predictive user modeling. *Proceedings of the 11th international conference on User Modeling*, pages 37–46.

Karpathy, A. and Li, F.-f. (2013). Automated Image Captioning with ConvNets and Recurrent Nets.

Kemker, R. and Kanan, C. (2018). FearNet: Brain-Inspired Model For Incremental Learning. In *International Conference on Learning Representations (ICLR2018)*, pages 1–16, Vancouver, Canada.

Kemker, R., McClure, M., Abitino, A., Hayes, T., and Kanan, C. (2018). Measuring Catastrophic Forgetting in Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI-18)*, New Orleans, USA.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, pages 1–15.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. 114(13):3521–3526.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2018). Reply to Huszár: The elastic weight consolidation penalty is empirically valid. *Proceedings of the National Academy of Sciences of the United States of America*, 115(11):E2498.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). Online learning with kernels. *IEEE transactions on signal processing*, 8(12):2165—-2176.

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. PhD thesis.

Krizhevsky, A., Sulskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information and Processing Systems (NIPS)*, pages 1–9.

Kumar, A. and Daume III, H. (2012). Learning Task Grouping and Overlap in Multi-Task Learning. *arXiv preprint arXiv:1206.6417*.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., Gershman, S. J., and Sciences, C. (2016). Building Machines That Learn and Think Like People. pages 1–55.

Laney, D. (2001). 3D data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1.

Le, Q. V., Ngiam, J., Chen, Z., Chia, D., Koh, P. W., and Ng, A. Y. (2010). Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 1279–1287.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 97–104.

Lee, S. (2017). Toward Continual Learning for Conversational Agents. *arXiv preprint arXiv:1712.09943*.

LeVine, S. (2017). Artificial intelligence pioneer says we need to start over.

Li, N. and DiCarlo, J. J. (2008). Unsupervised Natural Experience Rapidly Alters Invariant Object Representation in Visual Cortex. *Science*, 321(5895):1502–1507.

Li, Z. and Hoiem, D. (2016). Learning without forgetting. In *14th European Conference on Computer Vision (ECCV 2016)*, volume 9908 LNCS, pages 614–629, Amsterdam, Netherlands.

Liao, S. H. (2005). Expert system methodologies and applications-a decade review from 1995 to 2004. *Expert Systems with Applications*, 28(1):93–103.

Lin, M., Chen, Q., and Yan, S. (2014). Network In Network. In *International Conference on Learning Representations (ICLR 2014)*.

Liu, B., Hsu, W., and Ma, Y. (1999). Mining association rules with multiple minimum supports. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 23:337—-341.

Lomonaco, V. and Maltoni, D. (2017). CORe50: a New Dataset and Benchmark for Continuous Object Recognition. In Levine, S., Vanhoucke, V., and Goldberg, K., editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 17–26. PMLR.

Lomonaco, V. and Maltoni, D. (2018). CORe50 Leaderboard.

Long, M., Cao, Y., Wang, J., and Jordan, M. I. (2015). Learning Transferable Features with Deep Adaptation Networks. *Icml*, 37.

Lopez-paz, D. and Ranzato, M. (2017). Gradient Episodic Memory for Continuum Learning. In *Advances in neural information processing systems (NIPS 2017)*.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8.

Maltoni, D. (2011a). Pattern Recognition by Hierarchical Temporal Memory. *DEIA Technical Report*.

Maltoni, D. (2011b). Pattern Recognition by Hierarchical Temporal Memory. *DEIA Technical Report*.

Maltoni, D. and Rehn, E. M. (2012). Incremental learning by message passing in hierarchical temporal memory. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7477 LNAI:24–35.

Marcialis, G. L., Rattani, A., and Roli, F. (2008). Biometric Template Update : An Experimental Investigation on the Relationship between Update Errors and Performance Degradation in Face Verification. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 684–693.

Marcus, G. (2018). Deep Learning: A Critical Appraisal. pages 1–27.

Matthews, L., Ishikawa, T., and Baker, S. (2004). The Template Update Problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):810—-815.

McClelland, J. L., McNaughton, B. L., and O'reilly, R. C. (1995). 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.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165.

McRae, K. and Hetherington, P. A. (1993). Catastrophic interference is eliminated in pretrained networks. *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, pages 723—-728.

Mermillod, M., Bugaiska, A., and Bonin, P. (2013). The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4(August):504.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science \& Business Media.

Michalski, R. S. and Larson, J. (1978). Selection of most Representative Training Examples and Incremental Generation of VL1 Hypoteses.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. (1998). Never-Ending Learning.

Mitchell, T. M. and Thrun, S. B. (1993). Explanation-based neural network learning for robot control. In *NIPS6*, volume 5.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *American Journal of Health Behavior*, pages 1928—-1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv: ...*, pages 1–9.

Mobahi, H., Collobert, R., and Weston, J. (2009). Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8.

Nene, S. A., Nayar, S. K., and Murase, H. (1996). Columbia Object Image Library (COIL-100). *Technical Report*.

Omale, G. (2019). Gartner Predicts 25 Percent of Digital Workers Will Use Virtual Employee Assistants Daily by 2021.

O'reilly, R. C. (2004). The division of labor between the neocortex and hippocampus. *Connectionist models in cognitive psychology*, page 143.

O'reilly, R. C. and Rudy, J. W. (2001). Conjunctive representations in learning and memory: principles of cortical and hippocampal function. *Psychological review*, 108(2):311.

Pan, S. J., , and others Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 10:1345—-1359.

Pan, X. (2018). ResNet Tensoflow Reimplementation.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2018a). Continual Lifelong Learning with Neural Networks: A Review. *arXiv preprint arXiv:1802.07569*.

Parisi, G. I., Tani, J., Weber, C., and Wermter, S. (2018b). Lifelong Learning of Spatiotemporal Representations with Dual-Memory Recurrent Self-Organization. *arXiv preprint arXiv:1805.10966*, pages 1–20.

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep Face Recognition. *Procedings of the British Machine Vision Conference 2015*, (Section 3):41.1–41.12.

Pasquale, G., Ciliberto, C., Odone, F., Rosasco, L., and Natale, L. (2015a). Real-world Object Recognition with Off-the-shelf Deep Conv Nets: How Many Objects can iCub Learn? *arXiv:1504.03154 [cs]*.

Pasquale, G., Ciliberto, C., Odone, F., Rosasco, L., and Natale, L. (2015b). Teaching iCub to recognize objects using deep Convolutional Neural Networks. In *Proceedings of Workshop on Machine Learning for Interactive Systems*, pages 21–25.

Pasquale, G., Ciliberto, C., Rosasco, L., and Natale, L. (2016). 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.

Pearl, J. (2018). Theoretical Impediments to Machine Learning With Seven Sparks from the Causal Revolution. pages 1–8.

Pentina, A. and Lampert, C. H. (2013). A PAC-Bayesian bound for Lifelong Learning. 32.

Pentina, A. and Lampert, C. H. (2015). Lifelong Learning with Non-i.i.d. Tasks. *Nips*, pages 1–9.

Pham, D. and Dimov, S. (1970). The RULES-4 incremental inductive learning algorithm. *WIT Transactions on Information and Communication Technologies*, 19.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June:3406–3413.

Poggio, T. and Riesenhuber, M. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025.

Ranzato, M., Huang, F. J., Boureau, Y.-L., and LeCun, Y. (2007). Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, volume 8, pages 1–8. IEEE.

Rattani, A., Freni, B., Marcialis, G. L., and Roli, F. (2009). Template update methods in adaptive biometric systems: A critical review. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5558 LNCS, pages 847–856.

Ravi, N., Dandekar, N., Mysore, P., and Littman, M. M. L. (2005). Activity Recognition from Accelerometer Data. In *Proc. The 17th Conference on Innovative Applications of Artificial Intelligence (IAAI'05)*, volume 3, pages 1541–1546.

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519.

Rebuffi, S.-a., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). iCaRL: Incremental Classifier and Representation Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii.

Reinke, R. and Michalski, R. (1985). Incremental Learning of Concepts Descriptions.

Reinsel, D., Gantz, J., and Rydning, J. (2017). Data age 2025: The evolution of data to life-critical. Technical report.

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149.

Ring, M. (1994). *Continual Learning in Reinforcement Environments*. PhD thesis.

Ring, M. B. (2005). Toward a Formal Framework for Continual Learning. *Inductive Transfer: 10 Years Later, NIPS 2005 Workshop*, pages 1–4.

Rosenberg, C., Hebert, M., and Schneiderman, H. (2007). Semi-supervised self-training of object detection models. In *Proceedings - Seventh IEEE Workshop on Applications of Computer Vision, WACV 2005*, pages 29–36.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: a Modern Approach*. Malaysia; Pearson Education Limited.

Rusu, A. A., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., Hadsell, R., and Deepmind, G. (2016a). Policy Distillation. pages 1–13.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016b). Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*.

Ruvolo, P. and Eaton, E. (2013a). Active Task Selection for Lifelong Machine Learning. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 862–868.

Ruvolo, P. and Eaton, E. (2013b). ELLA: An efficient lifelong learning algorithm. *Proceedings of the 30th International Conference on Machine Learning*, 28(1):507–515.

Sadowski, P. (2016). Notes on Backpropagation.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of research and development*, 3:210–229.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. pages 1–16.

Saxe, A. M., Koh, P. W., Chen, Z., Bh, M., Suresh, B., and Ng, A. Y. (2011). On Random Weights and Unsupervised Feature Learning. *ICML*, pages 1089—-1096.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schulz, L. (2012). The origins of inquiry: Inductive inference and exploration in early childhood. *Trends in Cognitive Sciences*, 16(7):382–389.

Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress & Compress: A scalable framework for continual learning. *Icml*.

Schwarz, M., Schulz, H., and Behnke, S. (2015). RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features. *IEEE International Conference on Robotics and Automation (ICRA'15)*, (May):1329–1335.

Seff, A., Beatson, A., Suo, D., Liu, H., and Engineering, F. (2017). Continual Learning in Generative Adversarial Nets. pages 1–9.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. (2018). Overcoming Catastrophic Forgetting with Hard Attention to the Task. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4548—-4557.

Seymour, J. P., Wu, F., Wise, K. D., and Yoon, E. (2017). State-of-the-art MEMS and microsystem tools for brain research. *Microsystems & Nanoengineering*, 3(July 2016):16066.

Shmelkov, K., Schmid, C., and Alahari, K. (2017). Incremental Learning of Object Detectors without Catastrophic Forgetting. *arXiv preprint arXiv:1708.06977*.

Shu, L., Liu, B., Xu, H., and Kim, A. (2016). Lifelong-RL: Lifelong Relaxation Labeling for Separating Entities and Aspects in Opinion Targets. *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, 2016(1):225–235.

Silver, D. L. and Mercer, R. E. (2002). The Task Rehearsal Method of Life-Long Learning: Overcoming Impoverished Data. *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 90–101.

Silver, D. L. and Poirier, R. (2004). Sequential Consolidation of Learned Task Knowledge. *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 217–232.

Silver, D. L., Yang, Q., and Li, L. (2013). Lifelong Machine Learning Systems: Beyond Learning Algorithms. *AAAI Spring Symposium Series*, (Solomonoff 1989):49–55.

Simonyan, K., Dieleman, S., Senior, A., and Graves, A. (2016). WaveNet. *arXiv preprint arXiv:1609.03499v2*, pages 1–15.

Singh, A., Sha, J., Narayan, K. S., Achim, T., and Abbeel, P. (2014). BigBIRD: A Large-Scale 3D Database of Object Instances. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 509–516.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhuber, J. (2013). Compete to Compute. *Advances in neural information processing systems (NIPS)*, pages 2310–2318.

Steinwart, I. and Christmann, A. (2008). *Support vector machines*. Springer Science & Business Media.

Sternberg, R. J. (1982). *Handbook of human intelligence*. CUP Archive.

Steward, J., Lichti, D., Chow, J., Ferber, R., Osis, S., and Key, C. (2015). Performance Assessment and Calibration of the Kinect 2.0 Time-of-Flight Range Camera for Use in Motion Capture Applications. In *FIG Working Week 2015*, pages 1–14.

Stulp, F., Herlant, L., Hoarau, A., and Raiola, G. (2014). Simultaneous on-line Discovery and Improvement of Robotic Skill options. In *IEEE International Conference on Intelligent Robots and Systems*, number Iros, pages 1408–1413.

Sun, Z. and Tan, T. (2009). Ordinal Measures for Iris Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2211–2226.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Advances in neural information processing systems*, pages 3104—-3112.

Sutton, R. S., Barto, A. G., and Bach, F. (1998). *Reinforcement learning: An introduction*. MIT press.

Szegedy, C., Liu, W., Jia, Y., and Sermanet, P. (2015). Going deeper with convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tanaka, F. and Yamamura, M. (1997). An approach to lifelong reinforcement learning through multiple environments. *6th European Workshop on Learning Robots*, pages 93–99.

Taylor, M. E., Jong, N. K., and Stone, P. (2008). Transferring Instances for Model-Based Reinforcement Learning. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488—-505.

Taylor, M. E. and Stone, P. (2009). *Transfer learning for reinforcement learning domains: A survey*, volume 10 of *Adaptation, Learning, and Optimization*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Thrun, S. (1996). *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Springer.

Thrun, S. and Mitchell, T. M. (1995). Lifelong Robot Learning. *The biology and technology of intelligent autonomous agents*, pages 165—-196.

Thrun, S. and Pratt, L. (2012). *Learning to learn*. Springer Science \& Business Media.

Tsai, C.-H., Lin, C.-Y., and Lin, C.-J. (2014). Incremental and Decremental Training for Linear Classification. *20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 2014*, pages 343–352.

Turing, A. M. (1950). Computing Machinery and Intelligence.

van Holthoon, F. and Olson, D. R. (1987). *Common sense: the foundations for social science*. University Press of America.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition (CVPR)*, 1:I—-511—-I—-518.

Wagner, R., Thom, M., Schweiger, R., Palm, G., and Rothermel, A. (2013). Learning convolutional neural networks from few samples. In *Proceedings of the International Joint Conference on Neural Networks*.

Wang, S., Chen, Z., and Liu, B. (2016). Mining Aspect-Specific Opinion using a Holistic Lifelong Topic Model. *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, pages 167–176.

Wang, X. and Gupta, A. (2015). Unsupervised Learning of Visual Representations Using Videos. *Iccv*, pages 2794–2802.

Weng, J. (2001). ARTIFICIAL INTELLIGENCE: Autonomous Mental Development by Robots and Animals. *Science*, 291(5504):599–600.

Weston, J., Chopra, S., and Bordes, A. (2015). Memory Networks. *International Conference on Learning Representations*, pages 1–14.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R. (2012). Deep learning via semi-supervised embedding. *Neural Networks: Tricks of the Trade*, pages 639—-655.

Wiering, M. and Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12:51.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-task reinforcement learning. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, volume 43, pages 1015–1022, New York, New York, USA. ACM Press.

Wong, J. M. (2016). Towards Lifelong Self-Supervision: A Deep Learning Direction for Robotics.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *Advances in neural information processing systems*, pages 5279—-5288.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems 27 (Proceedings of NIPS)*, 27:1–9.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual Learning Through Synaptic Intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3987–3995, Sydney, Australia.

Zhu, X. (2006). Semi-Supervised Learning Literature Survey. *Computer Science, University of Wisconsin-Madison*, 2(3):4.

Zou, W. Y., Ng, A. Y., and Yu, K. (2011). Unsupervised learning of visual invariance with temporal coherence. *NIPS 2011 workshop on deep learning and unsupervised feature learning*, 3.

Zou, W. Y., Zhu, S., Ng, A. Y., and Yu, K. (2012). Deep Learning of Invariant Features via Simulated Fixations in Video. *Advances in neural information processing systems*, pages 3203—-3211.